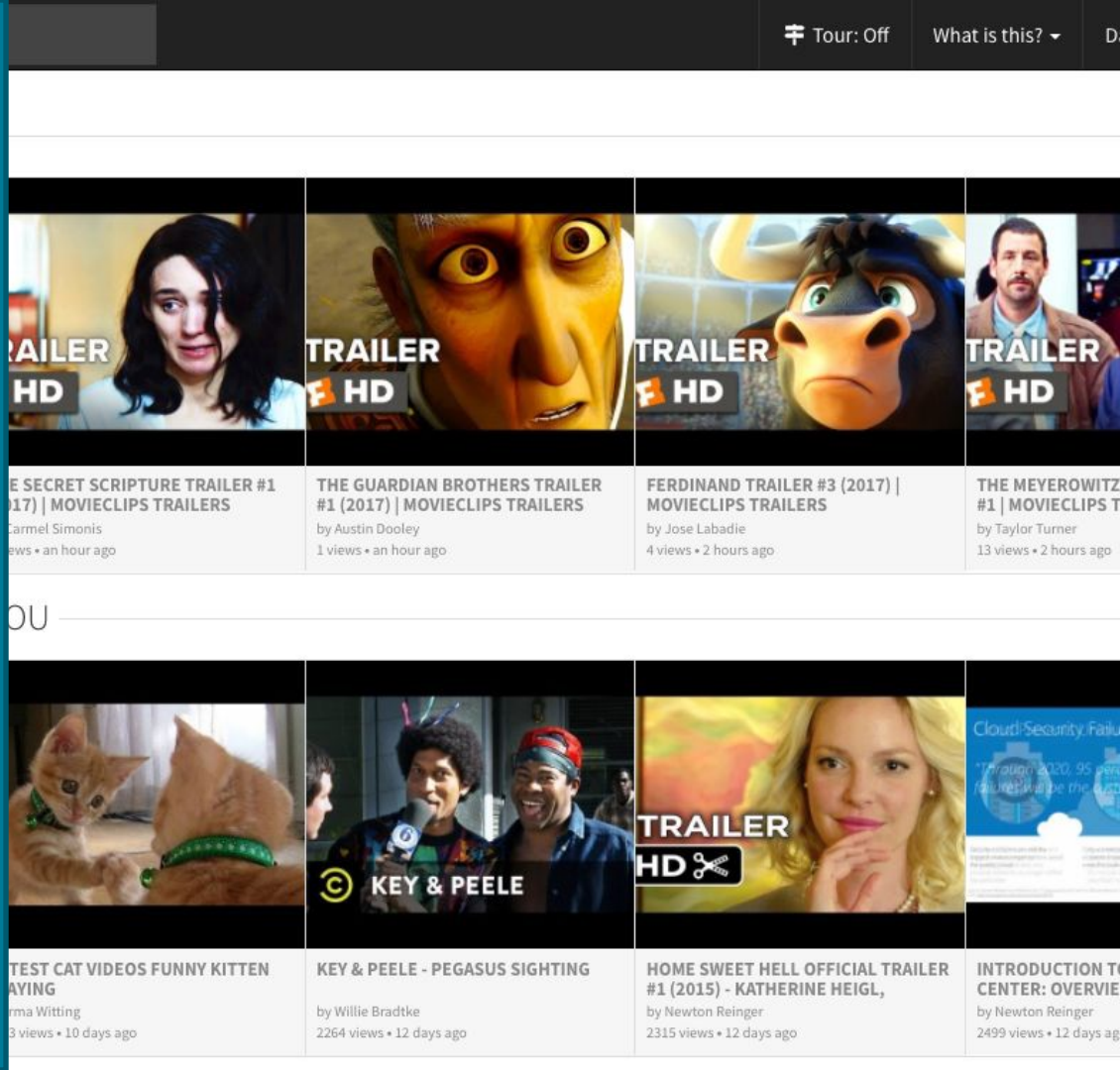
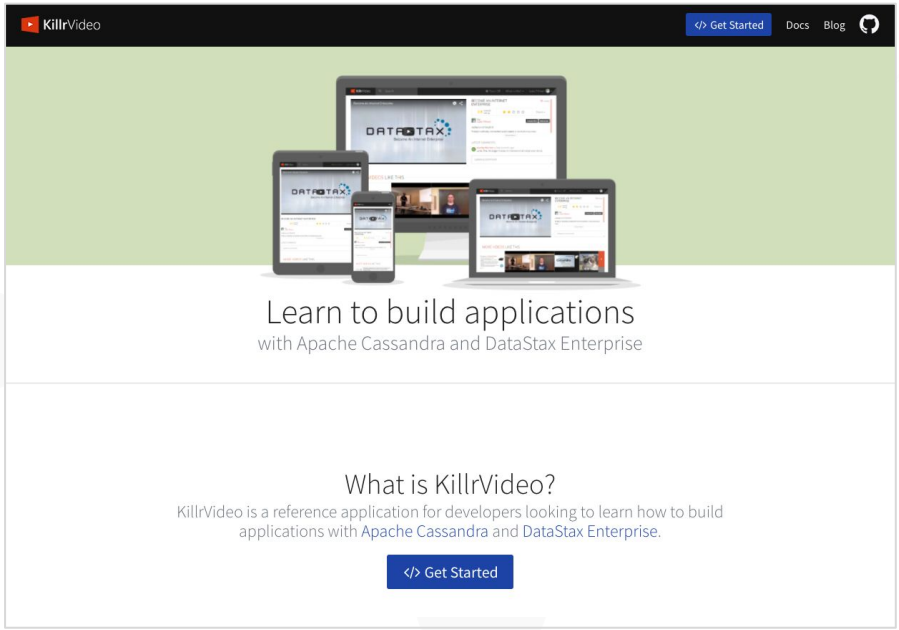
Cassandra Day Application Development

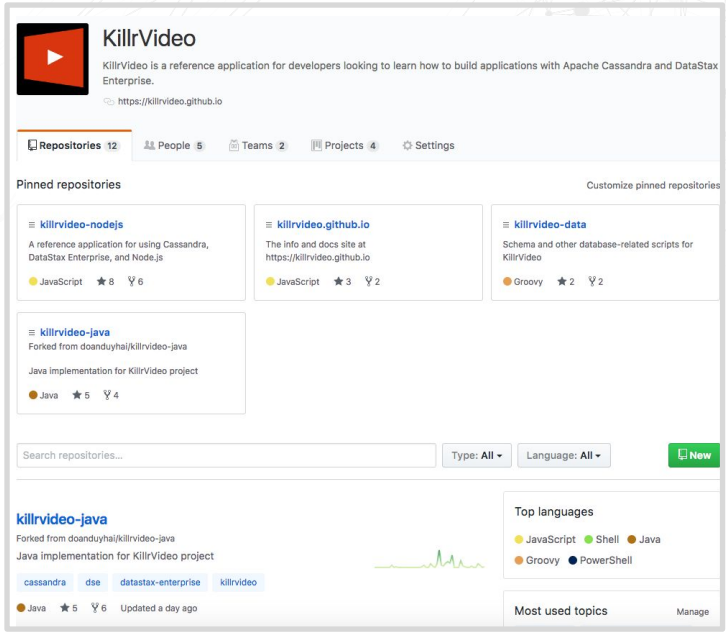
Source Code Inside

KillrVideo - a video sharing application



KillrVideo https://github.com/KillrVideo/killrvideo-java

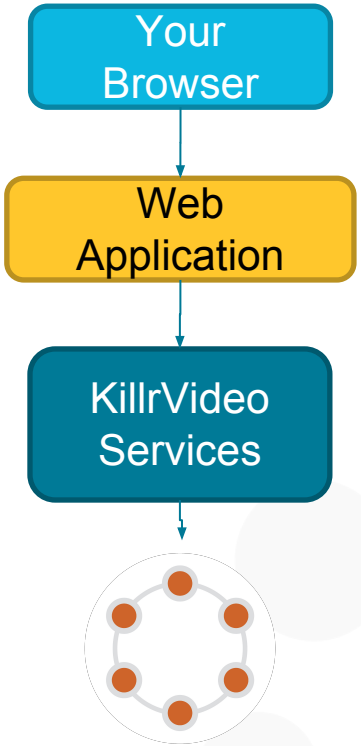




https://killrvideo.github.io

https://github.com/KillrVideo

KillrVideo Architecture



Technology Choices
• Node.js • Falcor
• Java / C# / Node.js / Python • GRPC • Etcd • 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 Drivers

- OSS Cassandra Drivers
  - CQL Support
  - Sync / Async API
  - Load Balancing
  - Auto Node Discovery
  - Object Mapper
- DataStax Enterprise Drivers
  - OSS Driver features plus...
  - Unified Authentication
  - Graph Fluent API
  - Geometric Types




- ODBC
- JDBC

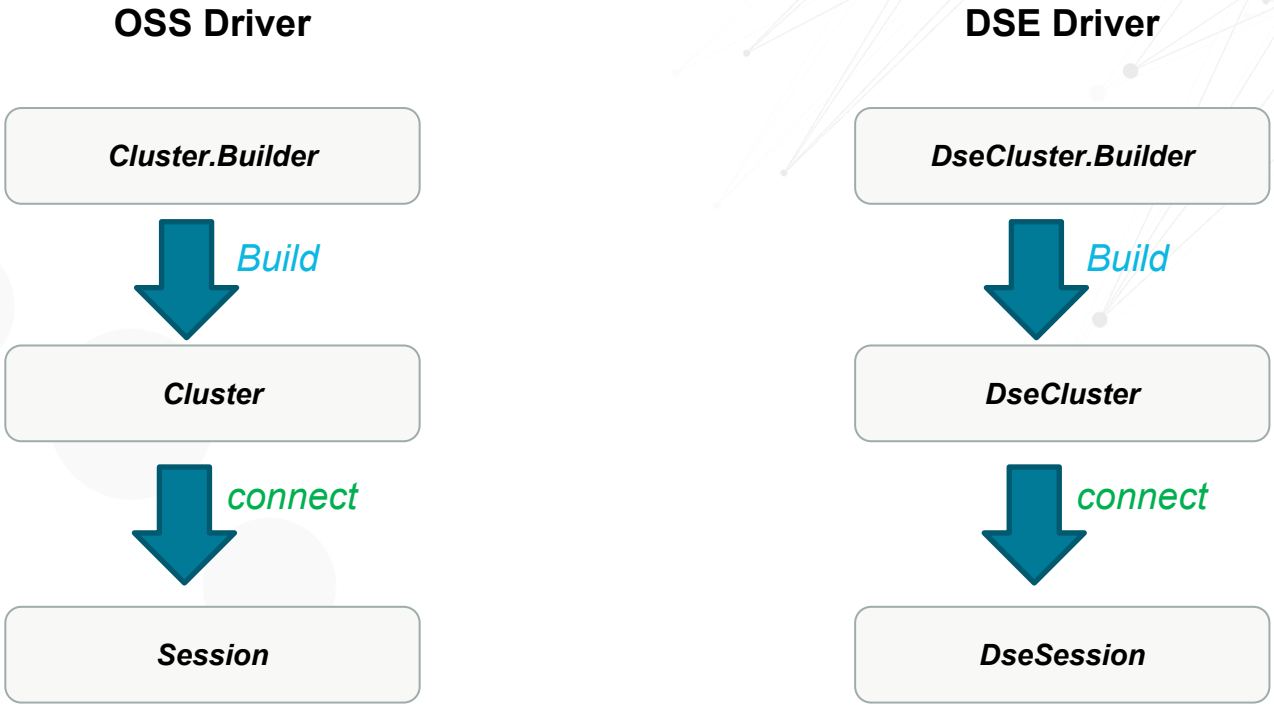
Maven

- Pom file (?)

```
<dependency>
  <groupId>com.datastax.dse</groupId>
  <artifactId>dse-java-driver-core</artifactId>
  <version>1.7.0</version>
</dependency>
```

LINK

Connectivity

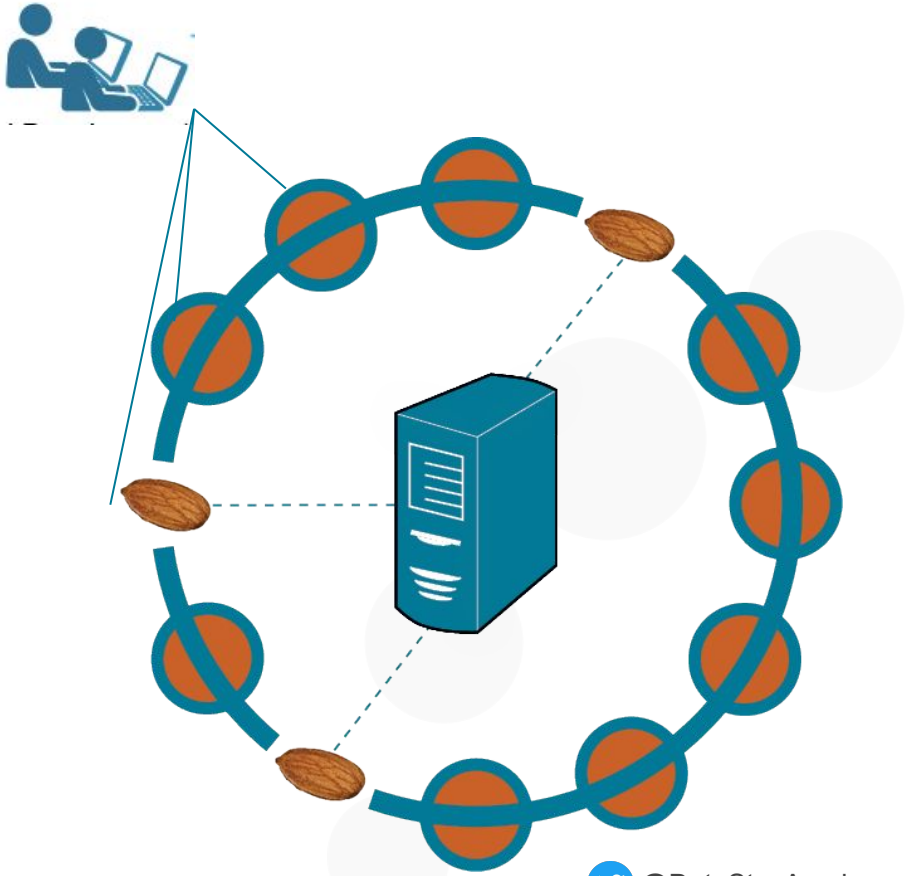


Builder ?

Builder

```
DseCluster myCluster =
  DseCluster
    .builder()
    .addContactPoint("192.168.0.1")
    .build()
```

Contact Points



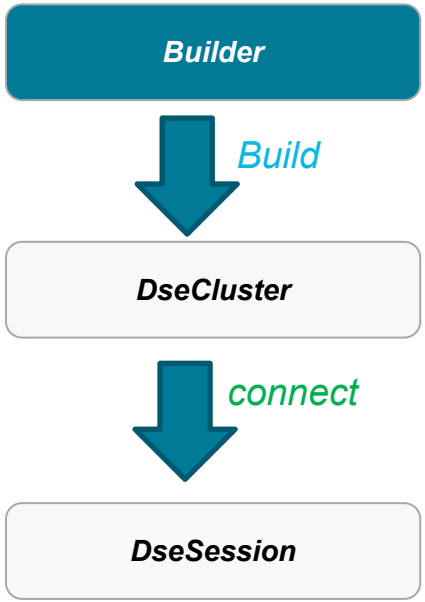
Only one necessary

Unless that node is down 

More are good

Seed nodes are good; more “in the know”

Builder !

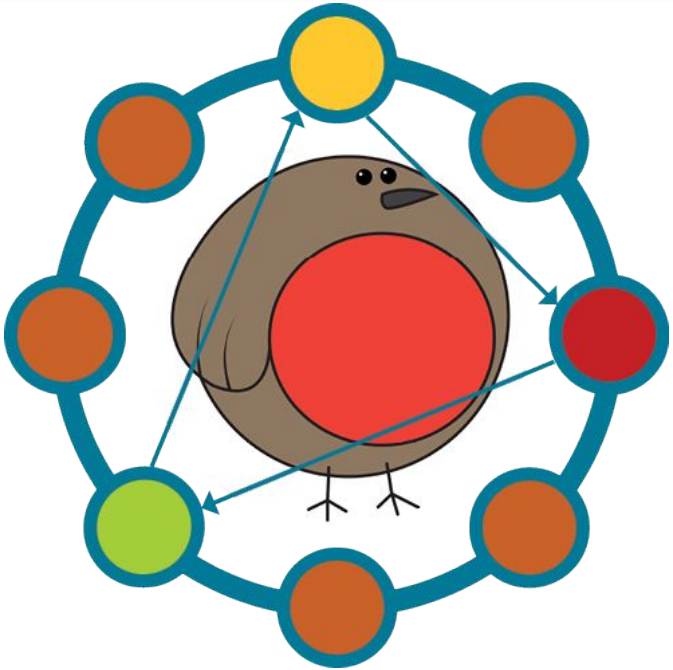



```
Builder clusterbuilder = new Builder();
clusterbuilder.addContactPoint("192.168.0.1");
clusterbuilder.addContactPoint("192.168.0.2");
clusterbuilder.addContactPoint("192.168.0.3");
clusterbuilder.withPort(9042);
```

LINK

Load balancing

```
new TokenAwarePolicy(
  new DCAwareRoundRobinPolicy("local-dc-name"))
```

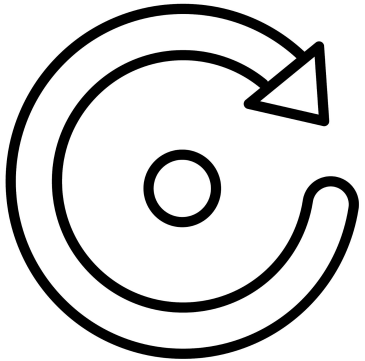


Retry Policies

- Most are deprecated (long story; don't ask)

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

- `FallthroughRetryPolicy`
  - Doesn't retry
  - Sends exception to your client application



retry

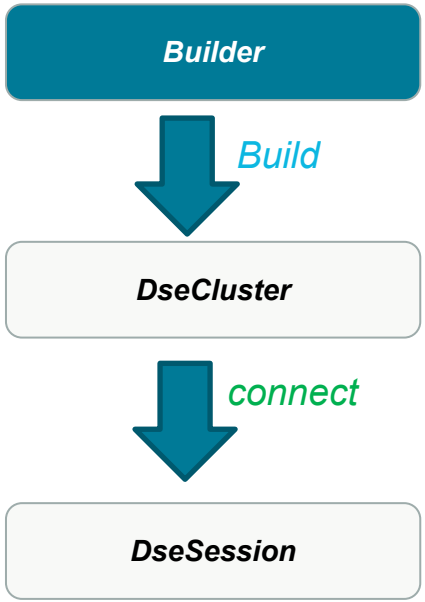
Reconnection Policies

Reconnects driver to a downed node

Two options:

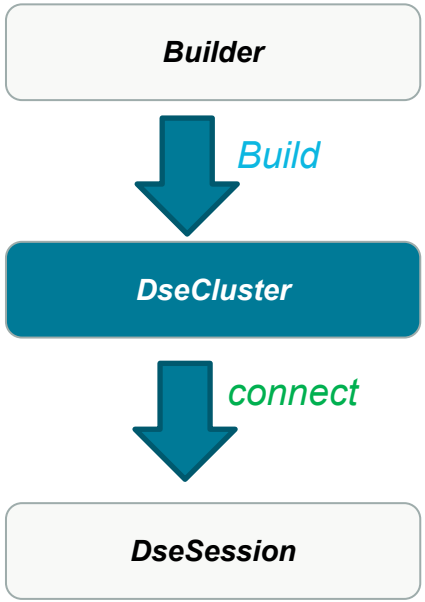
- `ConstantReconnectionPolicy`
  - Check every N milliseconds
- `ExponentialReconnectionPolicy` (default)
  - Increases every interval
  - Caps out at a max

Builder !!!!!!

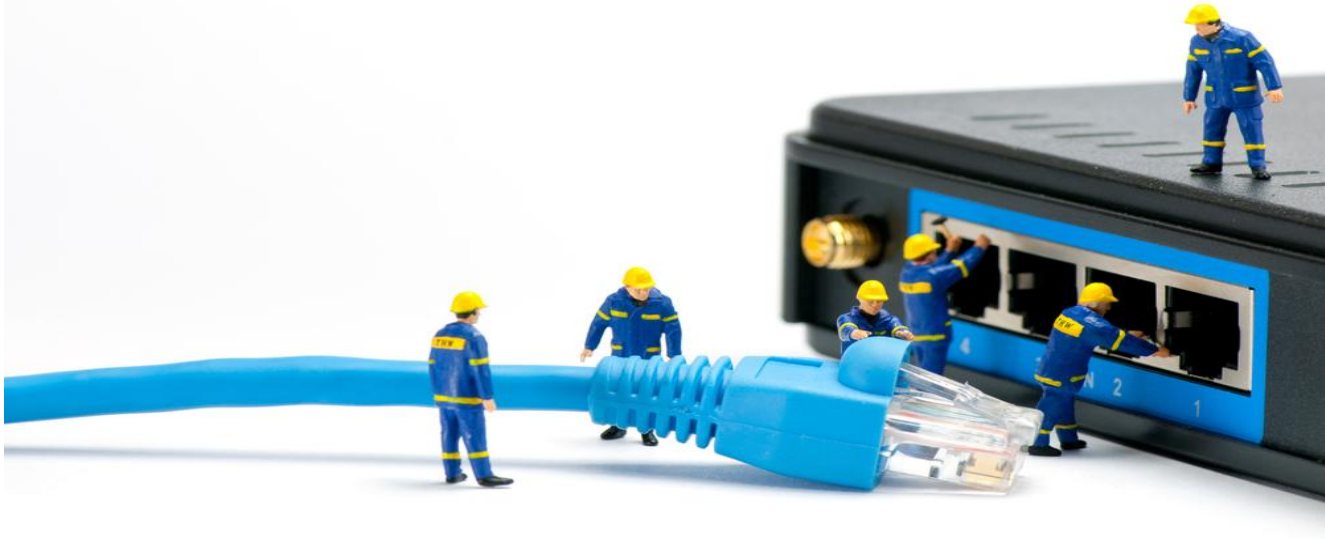



```
Builder clusterbuilder = new Builder();
clusterbuilder.addContactPoint("192.168.0.1");
clusterbuilder.addContactPoint("192.168.0.2");
clusterbuilder.addContactPoint("192.168.0.3");
clusterbuilder.withPort(9042);

clusterbuilder.withReconnectionPolicy(
  new ExponentialReconnectionPolicy(1000, 1600));

clusterbuilder.withLoadBalancingPolicy(
   new TokenAwarePolicy(
    DCAwareRoundRobinPolicy.builder()
    .withLocalDc("DC WEST").build()));

clusterbuilder.withRetryPolicy(
    DefaultRetryPolicy.INSTANCE);

clusterConfig.withAuthProvider(
  new DsePlainTextAuthProvider("Black", "Knight"));
```

connect

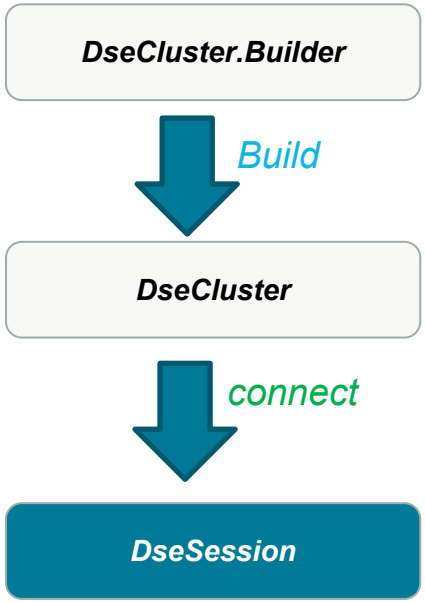



```
DseCluster myCluster = clusterbuilder.build();

DseSession mySession = myCluster.connect("killrvideo");

// DseSession mySession = myCluster.connect();
```



LINK

DseSession is all you need !



For when you need to do something quick and dirty

```
mySession.execute("SELECT * FROM users");
```

Statement

LINK

You know…sometimes you just got to get something done

SimpleStatement

- Driver interprets SimpleStatements at runtime and may include parameter placeholders
- You can set options on a `SimpleStatement` instance

```
Statement statement =
  new SimpleStatement("select * from t1 where c1 = 5" );

Statement statement2 =
  new SimpleStatement("select * from t1 where c1 = ?", 5 );

mySession.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 =
 session.prepare("SELECT * from t1 where c1 = ?");

BoundStatement bound = ps.bind(5);
```

LINK


Query
Builder

QueryBuilder

- Alternative to building CQL string queries manually
- Contains methods to build SELECT, UPDATE, INSERT and DELETE statements
- Generates a `Statement` as per the earlier techniques

Method	Return Type
`QueryBuilder.select()`	`Selection`
`QueryBuilder.insertInto()`	`Insert`
`QueryBuilder.update()`	`Update`
`QueryBuilder.delete()`	`Selection`

LINK

QueryBuilder

```
QueryBuilder
   .update("killrvideo", "video_ratings")
   .with(QueryBuilder.incr("rating_counter"))
   .and(QueryBuilder.incr("rating_total", QueryBuilder.bindMarker()))
   .where(QueryBuilder.eq("videoid", QueryBuilder.bindMarker()))
```

execute() → ResultSet

```
ResultSet rs = mySession.execute("SELECT * FROM users");

// Gets all Rows (pages) at once
List<Row> myRows = rs.all();
// Better - iterate!
rs.forEach(row -> { System.out.println(row.getDouble("ColX")); });

// Working with individual Rows
Row singleRow = rs.one();
int col1 = singleRow.getInt("c1");
String col2 = singleRow.getString("c2");
```

Exercise

DSE Application Development: Prepared Statements

Developer Day Cluster

7 days ago

```
Cluster 1
-------------------------------------------------------------
Node 1 - 18.144.54.163
Web terminal - https://18.144.54.163
Spark Jobs - http://18.144.54.163:4040/jobs
Spark Master - http://18.144.54.163:7080
DataStax Studio - http://18.144.54.163:9091
-------------------------------------------------------------
Node 2 - 13.57.198.36
Web terminal - https://13.57.198.36
Eclipse Che - http://13.57.198.36:8080/che/killrvideo
KillrVideo - http://13.57.198.36:3000
-------------------------------------------------------------
Node 3 - 54.215.253.80
Web terminal - https://54.215.253.80
OpsCenter - http://54.215.253.80:8888/opscenter
```

Object Mapper

- Map Java entity beans to CQL tables

```
CREATE TABLE IF NOT EXISTS users (
  userid uuid,
  firstname text,
  lastname text,
  email text,
  created_date timestamp,
  PRIMARY KEY (userid)
 );
```

```
public class User {

   private UUID userid;
   private String firstname;
   private String lastname;
   private String email;
   private Date createdAt;

   public User() {}

   public User(UUID userid, String firstname,
     String lastname, String email, Date createdAt) {
       this.userid = userid;
       this.firstname = firstname;
       this.lastname = lastname;
       this.email = email;
       this.createdAt = createdAt;
   }

   // Getters and Setters
   public UUID getUserid() { return userid; }
   public void setUserid(UUID userid) { this.userid = userid; }
}
```

```
@Table(keyspace = "killrvideo", name = "users")
public class User {

  @PartitionKey
  private UUID userid;

  @Column @Length(min = 1, message = "firstName must not be empty")
  private String firstname;

  @Column @Length(min = 1, message = "lastName must not be empty")
  private String lastname;

  @Column @Length(min = 1, message = "email must not be empty")
  private String email;

 @NotNull @Column(name = "created_date")
  private Date createdAt;

  // Constructors
  // Getters and Setters
}
```

Mapper

```
// Execute Once
MappingManager manager = new MappingManager(meSession);
Mapper<User> userMapper = manager.mapper(User. class);

// Usage
Result<User> users = userMapper.map(meSession.execute(meQuery));
```

Paging

- Pulling large result sets all at once may not make sense
- Retrieve sections or (pages)
- Retrieve more…if necessary
- Default of 5,000

Paging

Page Size ? setFetchSize()

```
// Init
Cluster cluster = Cluster.builder()
      .addContactPoint("127.0.0.1")
      .withQueryOptions(new QueryOptions().setFetchSize(2000))
      .build();

// At runtime
cluster.getConfiguration().getQueryOptions().setFetchSize(2000);


// On each
Statement statement = new SimpleStatement("your query");
statement.setFetchSize(2000);
```

LINK

Paging State

- Tracks location in a `ResultSet`
- “Continue where you left off…”
- Query + parameters must match

```
ResultSet resultSet = session.execute("your query");

// iterate the result set...
PagingState pagingState =
            resultSet.getExecutionInfo().getPagingState();

// Save the state
String meState = pagingState.toString()

// Later
PagingState pagingState = PagingState.fromString(meState);
Statement st = new SimpleStatement("your query");
st.setPagingState(pagingState);
ResultSet rs = session.execute(st);
```

LINK

Batches and the Driver

- All commands will *eventually* succeed
- *That's it*

```
BatchStatement insertVideoBatch = new BatchStatement();

insertVideoBatch.add("statement_1");
insertVideoBatch.add("statement_2");
insertVideoBatch.add("statement_3");
insertVideoBatch.add("statement_4");

session.execute(insertVideoBatch);
```

LINK

Batch Details

- One message to coordinator node
- Coordinator writes a batch log and replicates it to other nodes
- All affected nodes must ack the batch before coordinator removes batch log

Batches != Data Loading

- Batches about data integrity between tables
- All three inserts must succeed for batch to succeed

```
batchStatement.add(insertVideo);
batchStatement.add(insertUserVideo);
batchStatement.add(insertLatestVideo);
```


Best
Practice
Ahead

Instantiate Less

- Single **Cluster** object per physical cluster
- Single **Session** object per application
- .. And cleanup things

Run Async

```
mapper.map() vs mapper.mapAsync()
execute() vs executeAsync()
get() vs getAsync()
save() vs saveAsync()
```

Returns a ListenableFuture

Maximizes Throughput

Run Async

```
public void insertComment(final Comment comment) {
    BatchStatement batchStatement = buildBatchStatementInsertComment(comment);
    dseSession.execute(batchStatement);
}

public CompletableFuture<Void> insertCommentAsync(final Comment comment) {
    BatchStatement batchStatement = buildBatchStatementInsertComment(comment);

    // Create a callback for processing result (hey but here it is VOID so no mapping in Success())
    CompletableFuture<Void>   cfv        = new CompletableFuture<>();
    FutureCallback<ResultSet> myCallback = new FutureCallback<ResultSet>() {
        public void onFailure(Throwable ex) { cfv.completeExceptionally(ex); }
        public void onSuccess(ResultSet rs) { cfv.complete(null); }
    };

    // Bind execution and callback
    Futures.addCallback(dseSession.executeAsync(batchStatement), myCallback);
    return cfv;
}
```

Prepare Once – Bind Many

- Server parses/compiles query once
- Cached for lifetime of the application
- Reduces network traffic

Avoid Deletes and Writing Nulls

```
INSERT INTO meTable (primary_key, clustering_key)
VALUES ('pk1', 'ck1');
```

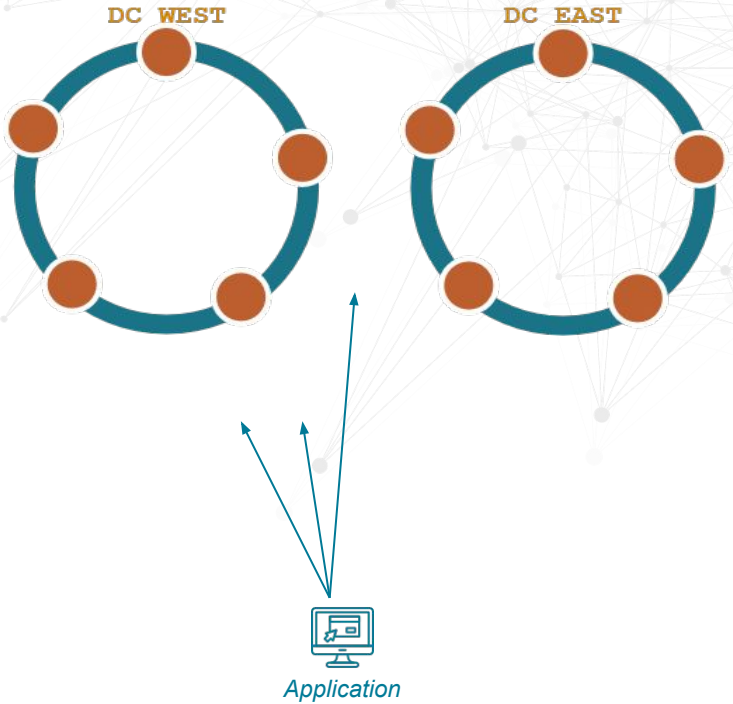
VS

```
INSERT INTO meTable (primary_key, clustering_key, regular_col)
VALUES ('pk1', 'ck1', null);
```

⇒ **Second version writes a tombstone**

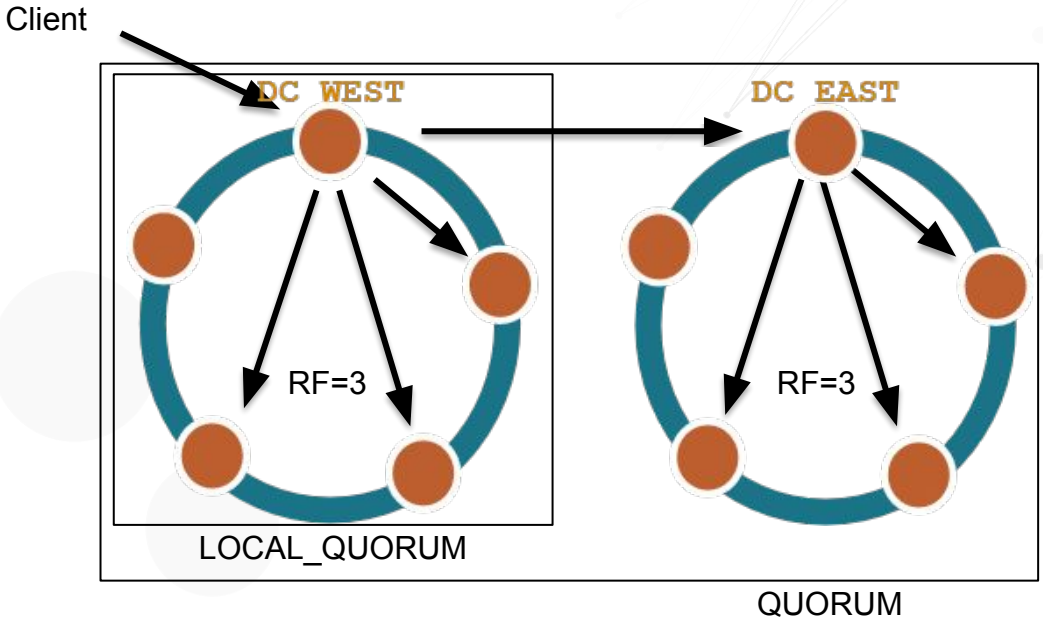
Be Datacenter Aware

```
DCAwareRoundRobinPolicy
            .builder()
            .withLocalDc("DC WEST")
```



Java Driver 4+ Note: You don't have to configure this

Consistency Across Data Centers



Consistency Level

```
// Global
QueryOptions options = new QueryOptions();
options.setConsistencyLevel(ConsistencyLevel.QUORUM);
Cluster cluster = Cluster
      .builder().addContactPoint("127.0.0.1")
      .withQueryOptions(options)
      .build();

// Or at runtime
cluster.getConfiguration().getQueryOptions().
setConsistencyLevel(ConsistencyLevel.QUORUM);
```

```
// On Each Query
Statement statement = new SimpleStatement("your query");
statement.setConsistencyLevel(ConsistencyLevel.QUORUM)
```

Spring Data Cassandra

Spring Data Cassandra is generally OK to use

But if you're going to use it

Just be aware of potentially creating bad habits that could potentially crush your production database down the road

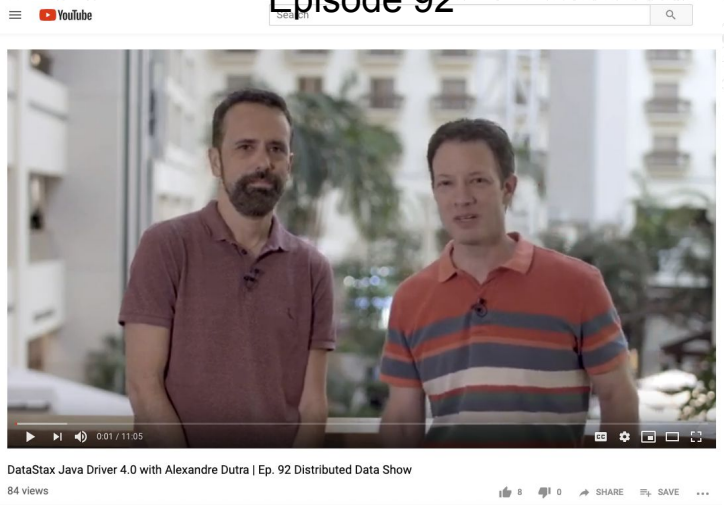
Time to Live (TTL)

- Time-series data
- Delete entire partitions when possible
- Better type of tombstones
- TimeWindowCompactionStrategy VS DateTieredCompactionStrategy

New! DataStax Java Driver v4.0

- Targeted for Java 8 and later
- Cleans up tech debt
  - Google Guava dependency removed
- File-based configuration
  - Hot reload
  - Execution profiles
- Global timeouts
  - spanning retries, speculative executions
- Reactive APIs
- More opinionated policies
  - Single default LoadBalancingPolicy
- Object mapper in progress

Distributed Data Show
Episode 92



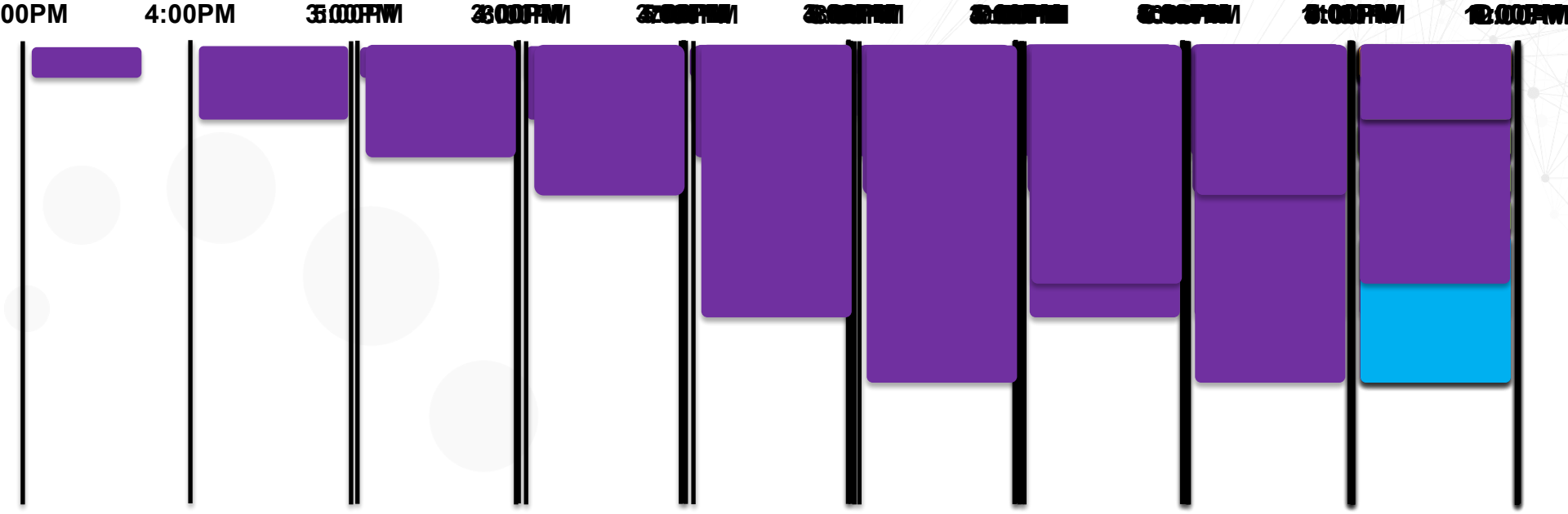
https://academy.datastax.com/content/introducing-java-driver-4

4.0 Driver API Changes - what you need to know

- Some package names changed to make a cleaner API
- Cluster combined into the Session object
  - ClusterBuilder replaced by SessionBuilder
  - Simplifies the API
- Statement objects are now immutable

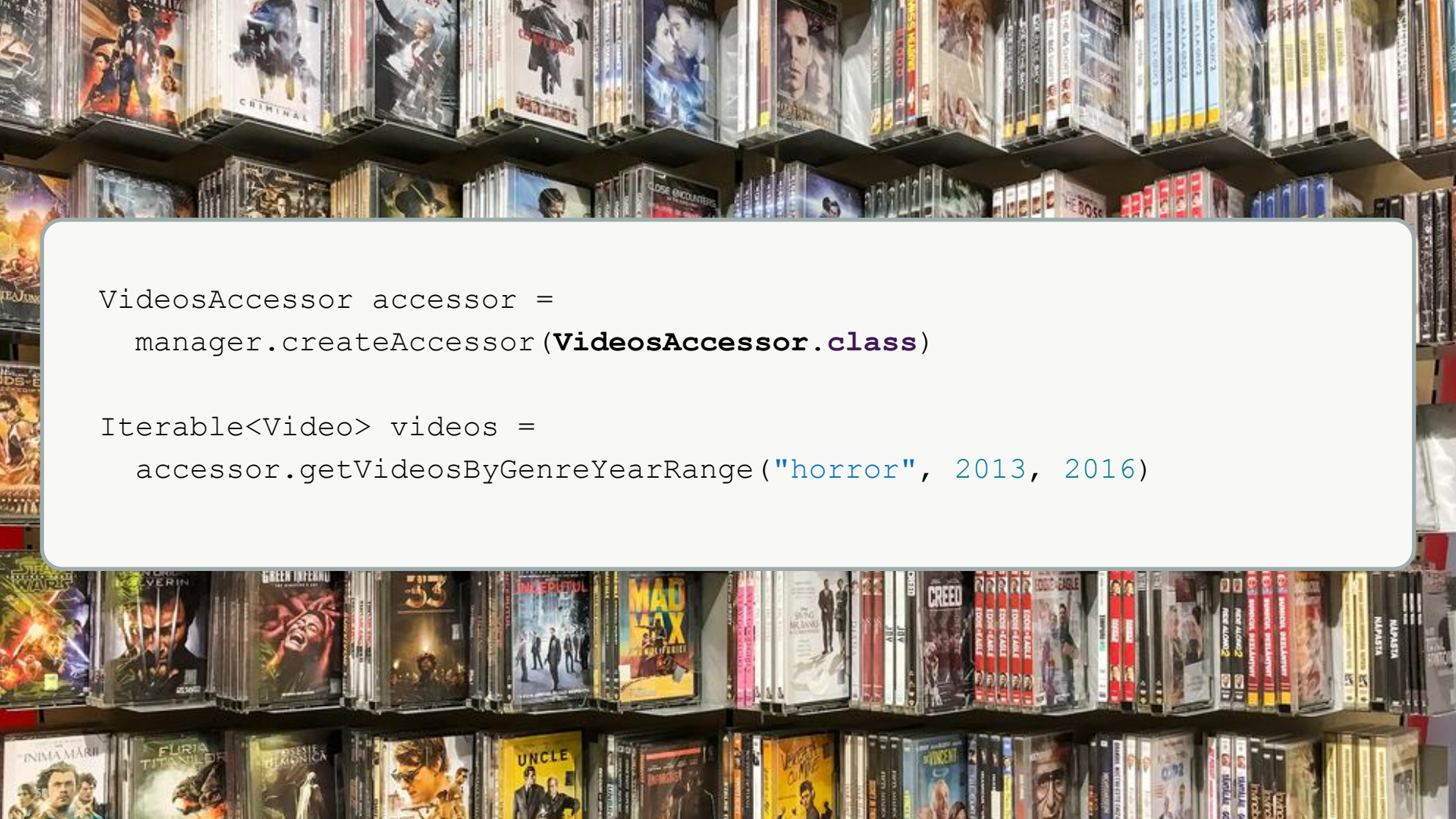
https://github.com/datastax/java-driver/tree/4.x/upgrade_guide

Time Windowed Compaction



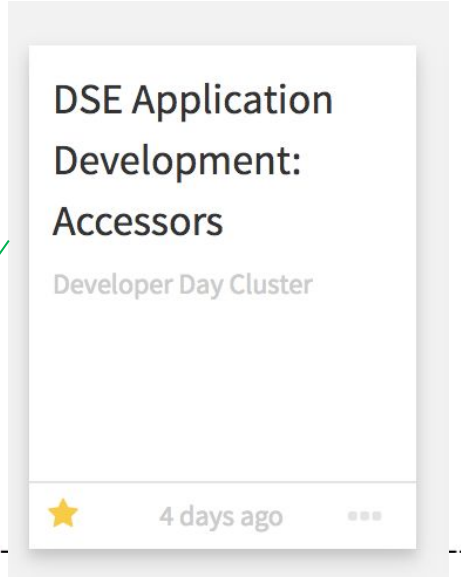
Accessors

```
@Accessor
public interface VideosAccessor {

  @Query("SELECT * FROM videos_by_genre"  +
         "WHERE genre = :genre AND release_year >= :start"  +
         " AND release_year <= :end ORDER BY release_year DESC ")
  Iterable<Video> getVideosByGenreYearRange(
                                    @Param("genre") String genre,
                                    @Param("start") int yearRangeStart,
                                    @Param("end")   int yearRangeEnd);


  //...

}
```



```
VideosAccessor accessor =
  manager.createAccessor(VideosAccessor.class)

Iterable<Video> videos =
  accessor.getVideosByGenreYearRange("horror", 2013, 2016)
```

Exercise #2




```
Cluster 1
------------------------------------------------------------
Node 1 - 18.144.54.163
Web terminal - https://18.144.54.163
Spark Jobs - http://18.144.54.163:4040/jobs
Spark Master - http://18.144.54.163:7080
DataStax Studio - http://18.144.54.163:9091
------------------------------------------------------------
Node 2 - 13.57.198.36
Web terminal - https://13.57.198.36
Eclipse Che - http://13.57.198.36:8080/che/killrvideo
KillrVideo - http://13.57.198.36:3000
------------------------------------------------------------
Node 3 - 54.215.253.80
Web terminal - https://54.215.253.80
OpsCenter - http://54.215.253.80:8888/opscenter
```