Going Native With Apache Cassandra™

QCon London, 2014
www.datastax.com
@DataStaxEMEA

About Me

Johnny Miller
Solutions Architect
www.datastax.com
@DataStaxEU

 jmiller@datastax.com

 @CyanMiller

 https://www.linkedin.com/in/johnnymiller

©2014 DataStax. Do not distribute without consent.

DataStax

- Founded in April 2010
- We drive Apache Cassandra™
- 400+ customers (24 of the Fortune 100)
- 220+ employees
- Contribute approximately 80% of the code to Cassandra
- Home to Apache Cassandra Chair & most committers
- Headquartered in San Francisco Bay area
- European headquarters established in London

We are hiring

www.datastax.com/careers

©2014 DataStax. Do not distribute without consent.

What do I mean by going native?

DATASTAX

- Traditionally, Cassandra clients (Hector, Astynax[1] etc..) were developed using Thrift
- With Cassandra 1.2 (Jan 2013) and the introduction of **CQL3** and **the CQL native protocol** a new easier way of using Cassandra was introduced.
- Why?
  - **Easier to develop and model**
  - **Best practices for building modern distributed applications**
  - **Integrated tools and experience**
  - **Enable Cassandra to evolve easier and support new features**

[1]Astynax is being updated to include the native driver: https://github.com/Netflix/astyanax/wiki/Astyanax-over-Java-Driver

©2014 DataStax. Do not distribute without consent.

CQL

DATASTAX

- **C**assandra **Q**uery **L**anguage
- CQL is intended to provide a common, simpler and easier to use interface into Cassandra - and you probably already know it!

e.g. SELECT * FROM users

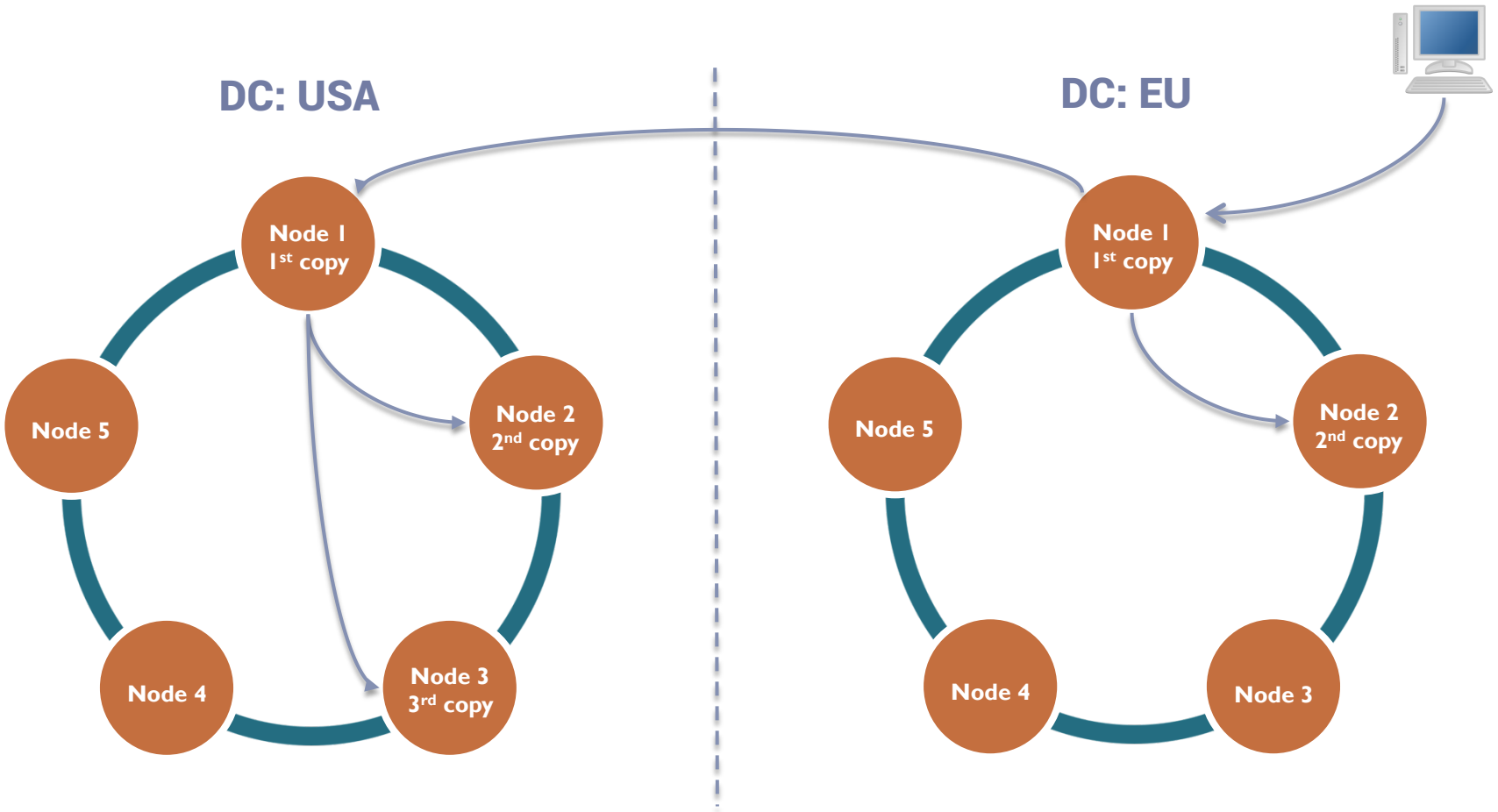
- Usual statements
  - CREATE / DROP / ALTER TABLE / SELECT

©2014 DataStax. Do not distribute without consent.

Creating A Keyspace

DATASTAX

```
CREATE KEYSPACE johnny WITH REPLICATION =
{'class':'NetworkTopologyStrategy', 'USA':3, 'EU': 2};
```



©2014 DataStax. Do not distribute without consent.

CQL Basics

DATASTAX

```
CREATE TABLE sporty_league (
   team_name   varchar,
   player_name varchar,
   jersey      int,
   PRIMARY KEY (team_name, player_name)
);
```

```
SELECT * FROM sporty_league WHERE team_name = ‘Mighty Mutts’ and player_name = ‘Lucky’;
```

```
INSERT INTO sporty_league (team_name, player_name, jersey) VALUES ('Mighty Mutts','Felix’,
90);
```

Predicates

- On the **partition key:** = and IN
- On the **cluster columns:** <, <=, =, >=, >, IN

©2014 DataStax. Do not distribute without consent.

Collections Data Type

DATASTAX

- CQL supports having columns that contain collections of data.
- The collection types include:
  - **Set, List and Map.**

```
CREATE TABLE collections_example (
    id int PRIMARY KEY,
    set_example set<text>,
    list_example list<text>,
    map_example map<int, text>
);
```

- Some performance considerations around collections.
  - Sometimes more efficient to denormalise further rather than use collections if intending to store lots of data.
  - **Favour sets over list – more performant**

- **Watch out for collection indexing in Cassandra 2.1!**

©2014 DataStax. Do not distribute without consent.

Query Tracing

- You can turn tracing on or off for queries with the TRACING ON | OFF command.
- This can help you understand what Cassandra is doing and identify any performance problems.

```
estamp FROM order_by_vendor WHERE vendor='VooDoo BBQ & Grill Franchising' AND bucket = 1;

e                | order_timestamp
-----------------+-------------------------
berry Grapefruit | 2013-08-08 23:02:58+0000


                                                                                                | timestamp    | source          | source_elapsed
------------------------------------------------------------------------------------------------+--------------+-----------------+----------------
                                                                         execute_cql3_query | 00:29:07,691 | 192.168.184.176 |              0
tamp FROM order_by_vendor WHERE vendor='VooDoo BBQ & Grill Franchising' AND bucket = 1 LIMIT 10000; | 00:29:07,691 | 192.168.184.176 |            140
                                                                         Preparing statement | 00:29:07,691 | 192.168.184.176 |            350
                                                      Executing single-partition query on order_by_vendor | 00:29:07,692 | 192.168.184.176 |           1085
                                                                      Acquiring sstable references | 00:29:07,692 | 192.168.184.176 |           1120
                                                                      Merging memtable tombstones | 00:29:07,692 | 192.168.184.176 |           1155
                                                    Merging data from memtables and 0 sstables | 00:29:07,692 | 192.168.184.176 |           1271
                                                    Read 1 live and 0 tombstoned cells | 00:29:07,692 | 192.168.184.176 |           1354
                                                                         Request complete | 00:29:07,692 | 192.168.184.176 |           1608
```

Worth reading: http://www.datastax.com/dev/blog/tracing-in-cassandra-1-2

©2014 DataStax. Do not distribute without consent.

Plus much, much more…

- **Light Weight Transactions**

```
INSERT INTO customer_account (customerID, customer_email) VALUES (‘LauraS’, ‘lauras@gmail.com’) IF NOT EXISTS;

UPDATE customer_account SET customer_email=’laurass@gmail.com’
IF customer_email=’lauras@gmail.com’;
```

- **Counters**

```
UPDATE UserActions SET total = total + 2
WHERE user = 123 AND action = 'xyz';
```

- **Time to live (TTL)**

```
INSERT INTO users (id, first, last) VALUES (‘abc123’, ‘abe’, ‘lincoln’) USING TTL 3600;
```

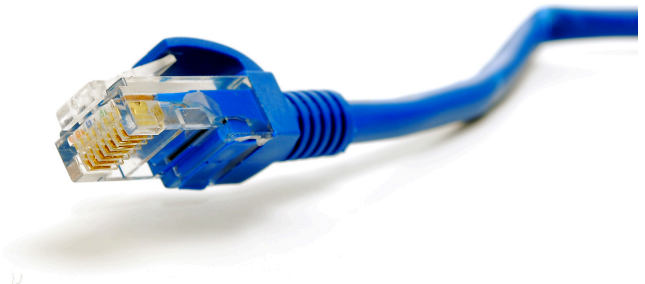
- **Batch Statements**

```
BEGIN BATCH
  INSERT INTO users (userID, password, name) VALUES ('user2', 'ch@ngem3b', 'second user')
  UPDATE users SET password = 'ps22dhds' WHERE userID = 'user2'
  INSERT INTO users (userID, password) VALUES ('user3', 'ch@ngem3c')
  DELETE name FROM users WHERE userID = 'user2'
APPLY BATCH;
```

- **New CQL features coming in Cassandra 2.0.6**
  - http://www.datastax.com/dev/blog/cql-in-2-0-6

©2014 DataStax. Do not distribute without consent.

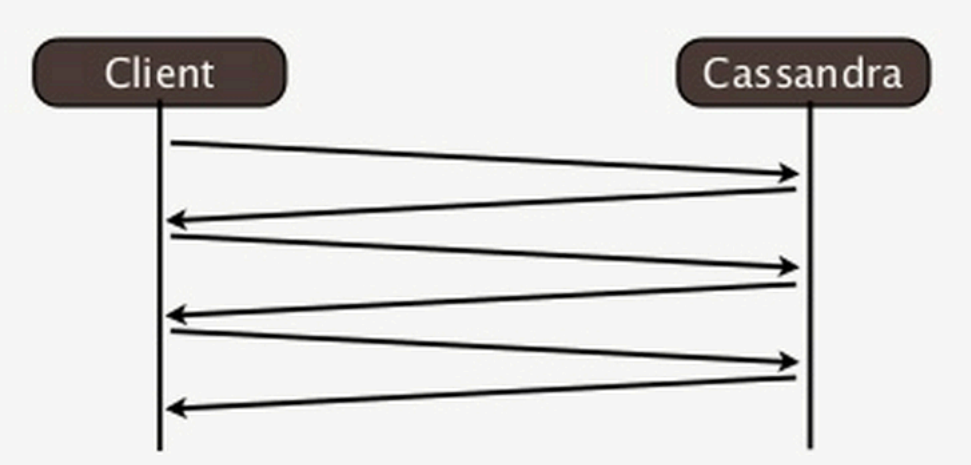
CQL Native Protocol



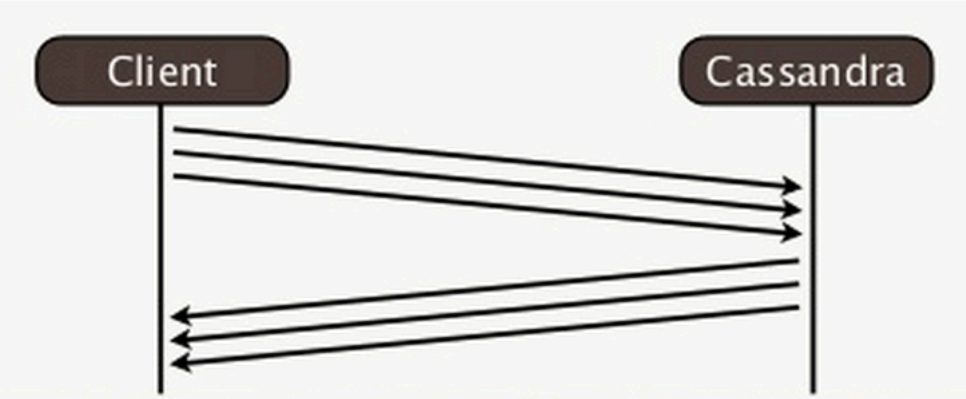
Request Pipelining

DATASTAX

With it:

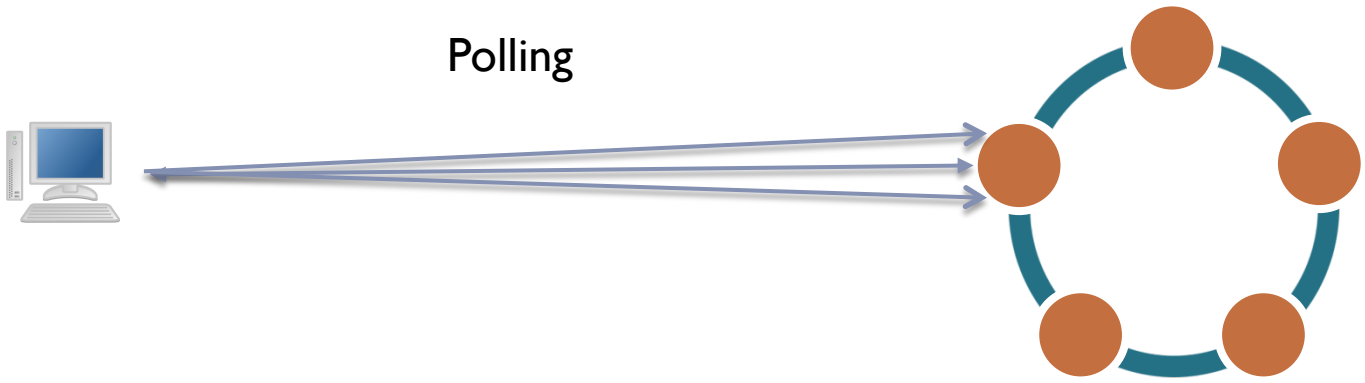


Without it:



©2014 DataStax. Do not distribute without consent.

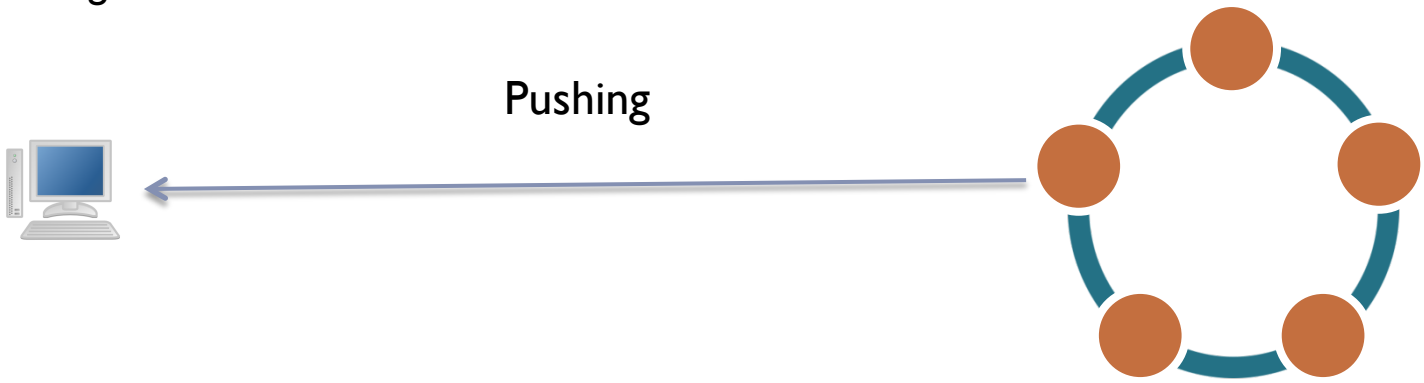
Notifications



Without Notifications

Notifications are for technical events only:

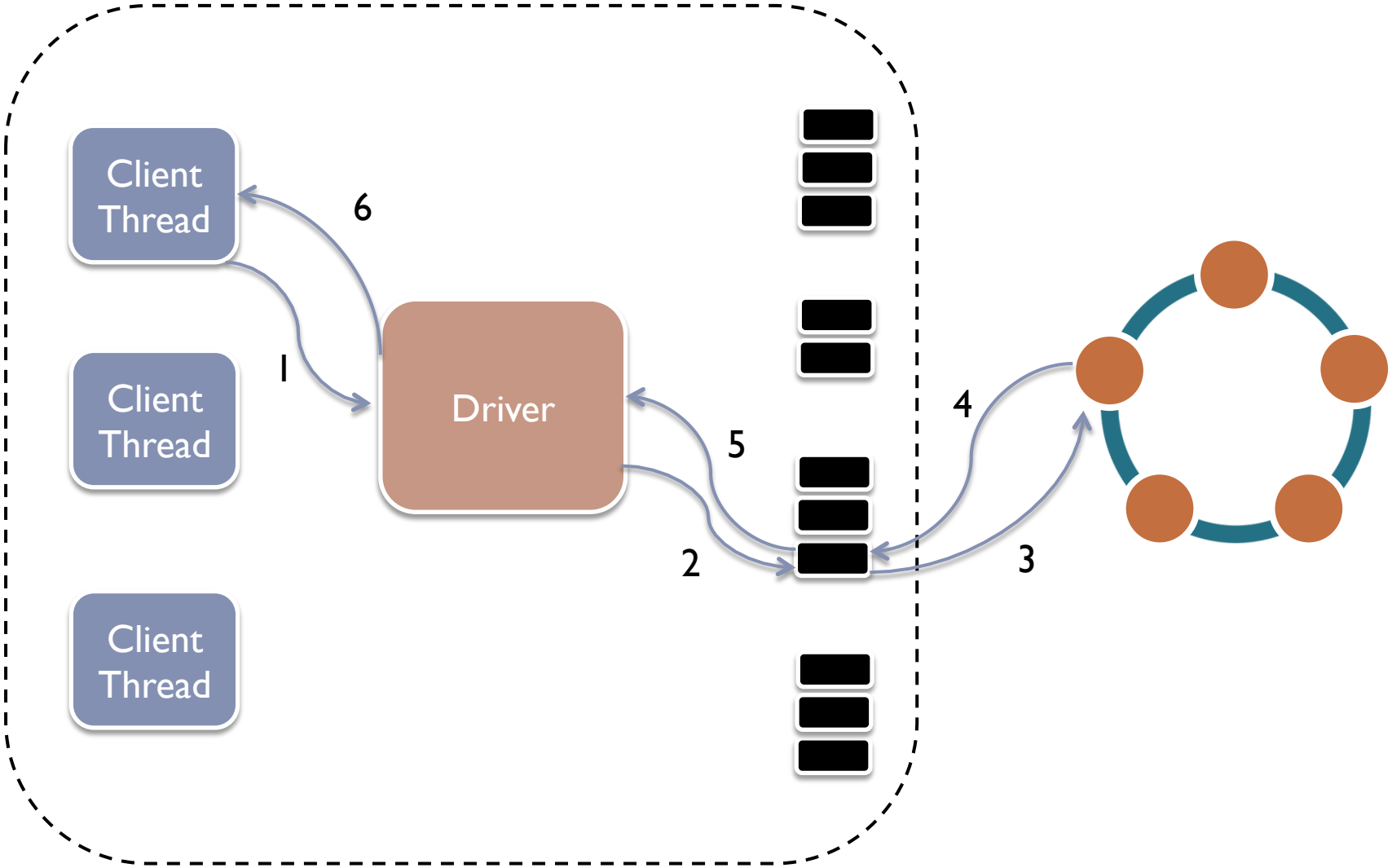
- Topology changes
- Node status changes,
- Schema changes



With Notifications

©2014 DataStax. Do not distribute without consent.

Asynchronous Architecture



[1]http://netty.io/

©2014 DataStax. Do not distribute without consent.

Native Drivers

Native Drivers

Get them here: http://www.datastax.com/download

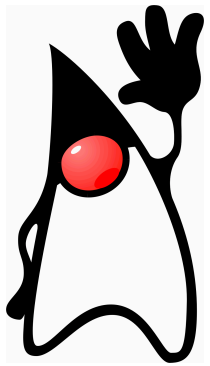
- **Java**
- **C#**
- **Python**
- **C++ (beta)**
- **ODBC (beta)**
- Clojure
- Erlang
- Node.js
- Ruby
- Plus many, many more….

©2014 DataStax. Do not distribute without consent.

Connect and Write

DATASTAX

```
Cluster cluster = Cluster.builder()
                  .addContactPoints("10.158.02.40", "10.158.02.44")
                  .build();

Session session = cluster.connect("akeyspace");

session.execute(
        "INSERT INTO user (username, password) ”
        + "VALUES(‘johnny’, ‘password1234’)"
);
```



Note: Clusters and Sessions should be long-lived and re-used.

©2014 DataStax. Do not distribute without consent.

Read from a table

DATASTAX

```
ResultSet rs = session.execute("SELECT * FROM user");

List<Row> rows = rs.all();

for (Row row : rows) {
        String userName = row.getString("username");
        String password = row.getString("password");
}
```

©2014 DataStax. Do not distribute without consent.

Asynchronous Read

```
ResultSetFuture future = session.executeAsync(
                                        "SELECT * FROM user");

for (Row row : future.get()) {
        String userName = row.getString("username");
        String password = row.getString("password");
}
```

Note: The future returned implements Guava's ListenableFuture interface. This means you can use all Guava's Futures[1] methods!

[1]http://docs.guava-libraries.googlecode.com/git/javadoc/com/google/common/util/concurrent/Futures.html

©2014 DataStax. Do not distribute without consent.

Read with Callbacks

DATASTAX

```
final ResultSetFuture future =
            session.executeAsync("SELECT * FROM user");

future.addListener(new Runnable() {

      public void run() {
            for (Row row : future.get()) {
                  String userName = row.getString("username");
                  String password = row.getString("password");
            }
      }

}, executor);
```

©2014 DataStax. Do not distribute without consent.

Parallelize Calls

```
int queryCount = 99;

List<ResultSetFuture> futures = new ArrayList<ResultSetFuture>();

for (int i=0; i<queryCount; i++) {
        futures.add(
                session.executeAsync("SELECT * FROM user "
                                +"WHERE username = '"+i+"'"));
}

for(ResultSetFuture future : futures) {
        for (Row row : future.getUninterruptibly()) {
                //do something
        }
}
```

©2014 DataStax. Do not distribute without consent.

Tip

DATASTAX

- If you need to do a lot of work, it's often better to make many small queries concurrently than to make one big query.
  - executeAsync and Futures – makes this really easy!
  - Big queries can put a high load on one coordinator
  - Big queries can skew your 99th percentile latencies for other queries
  - If one small query fails you can easily retry, if a big query than you have to retry the whole thing

©2014 DataStax. Do not distribute without consent.

Prepared Statements

```
PreparedStatement statement = session.prepare(
            "INSERT INTO user (username, password) "
            + "VALUES (?, ?)");

BoundStatement bs = statement.bind();

bs.setString("username", "johnny");
bs.setString("password", "password1234");

session.execute(bs);
```

©2014 DataStax. Do not distribute without consent.

Query Builder

DATASTAX

```
Query query = QueryBuilder
              .select()
              .all()
              .from("akeyspace", "user")
              .where(eq("username", "johnny"));


query.setConsistencyLevel(ConsistencyLevel.ONE);

ResultSet rs = session.execute(query);
```

©2014 DataStax. Do not distribute without consent.

Multi Data Center Load Balancing

- Local nodes are queried first, if none are available the request will be sent to a remote data center

```
Cluster cluster = Cluster.builder()
                .addContactPoints("10.158.02.40", "10.158.02.44")
                .withLoadBalancingPolicy(
                        new DCAwareRoundRobinPolicy("DC1"))
                .build();
```

Name of the local DC

©2014 DataStax. Do not distribute without consent.

Token Aware Load Balancing

- Nodes that own a replica of the data being read or written by the query will be contacted first

```
Cluster cluster = Cluster.builder()
                .addContactPoints("10.158.02.40", "10.158.02.44")
                .withLoadBalancingPolicy(
                        new TokenAwarePolicy(
                                new DCAwareRoundRobinPolicy("DC1")))
                .build();
```

http://www.datastax.com/drivers/java/2.0/com/datastax/driver/core/policies/TokenAwarePolicy.html

©2014 DataStax. Do not distribute without consent.

Retry Policies

DATASTAX

- This defined the behavior to adopt when a request returns a timeout or is unavailable.

```
Cluster cluster = Cluster.builder()
        .addContactPoints("10.158.02.40", "10.158.02.44")
        .withRetryPolicy(DowngradingConsistencyRetryPolicy.INSTANCE)
        .withLoadBalancingPolicy(new TokenAwarePolicy(new
DCAwareRoundRobinPolicy("DC1")))
        .build();
```

- DefaultRetryPolicy
- DowngradingConsistencyRetryPolicy
- FallthroughRetryPolicy
- LoggingRetryPolicy

http://www.datastax.com/drivers/java/2.0/com/datastax/driver/core/policies/RetryPolicy.html

©2014 DataStax. Do not distribute without consent.

Reconnection Policies

- Policy that decides how often the reconnection to a dead node is attempted.

```
Cluster cluster = Cluster.builder()
        .addContactPoints("10.158.02.40", "10.158.02.44”)
        .withRetryPolicy(DowngradingConsistencyRetryPolicy.INSTANCE)
        .withReconnectionPolicy(new ConstantReconnectionPolicy(1000))
        .withLoadBalancingPolicy(new TokenAwarePolicy(new
DCAwareRoundRobinPolicy("DC1")))
        .build();
```

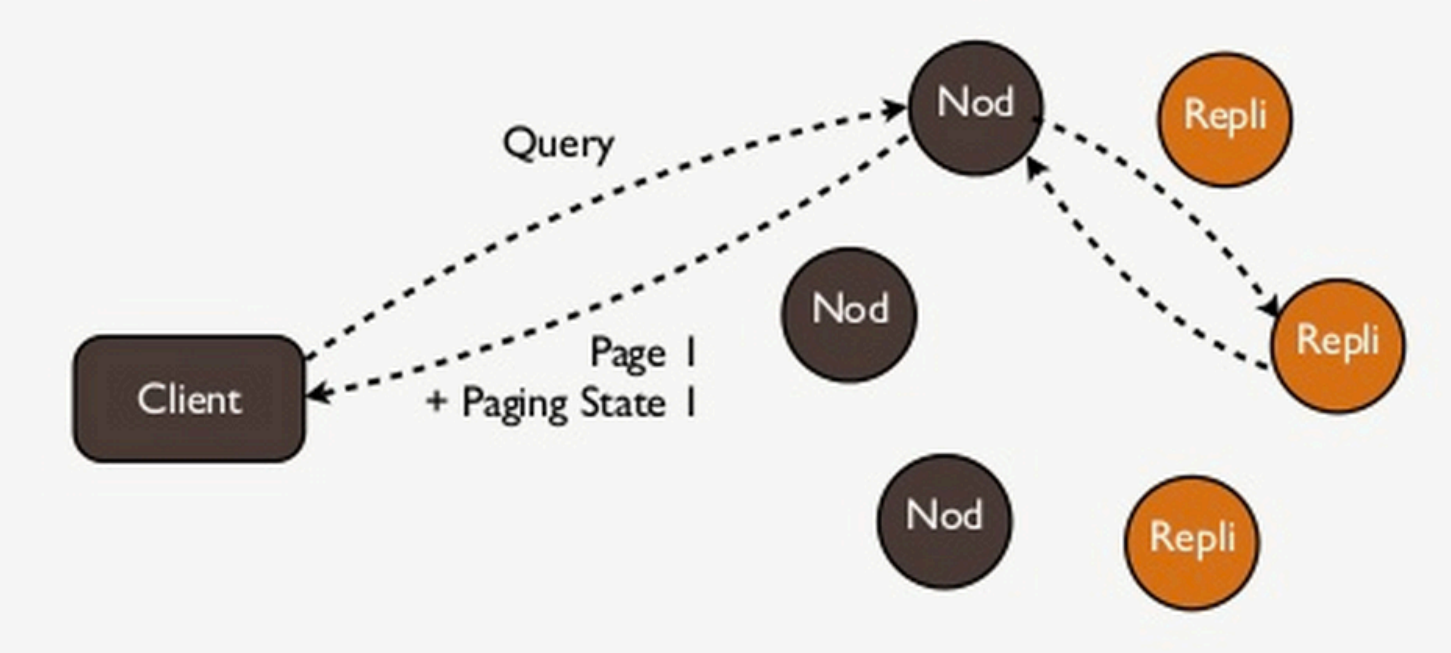
- ConstantReconnectionPolicy
- ExponentialReconnectionPolicy

©2014 DataStax. Do not distribute without consent.

Automatic Paging

DATASTAX

- This was new in Cassandra 2.0
- Previously – you would select data in batches



©2014 DataStax. Do not distribute without consent.

Query Tracing

DATASTAX

- Tracing is enabled on a per-query basis.

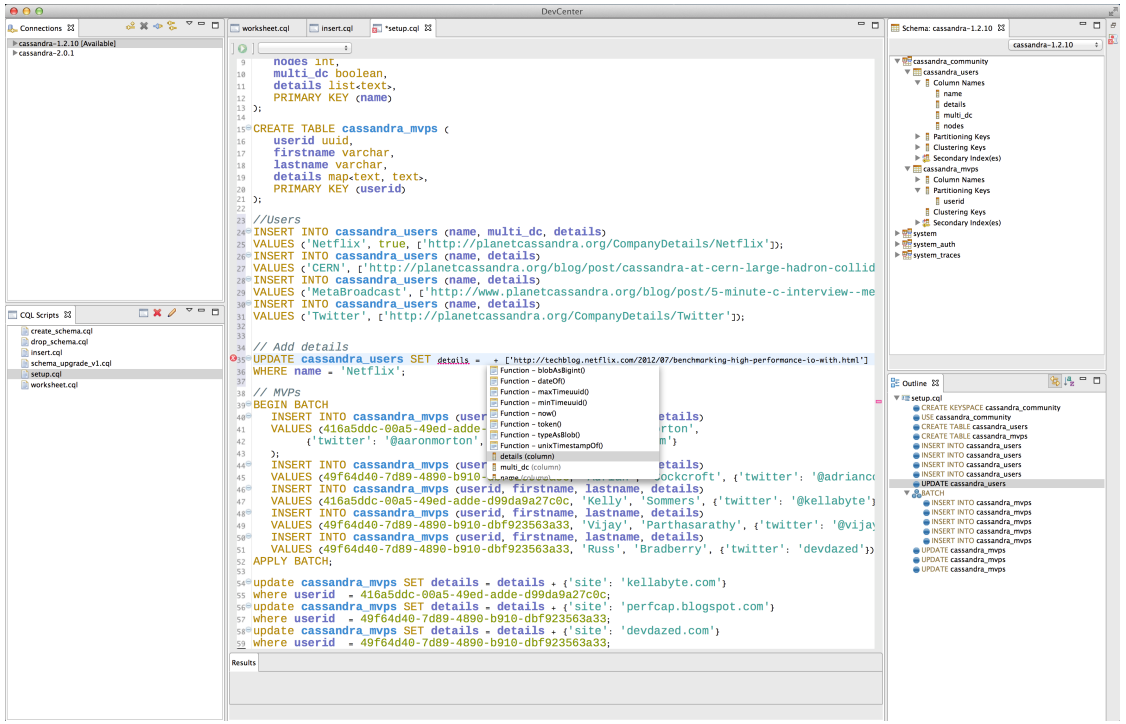
```
Query query = QueryBuilder
                .select()
                .all()
                .from("akeyspace", "user")
                .where(eq("username", "johnny"))
                .enableTracing();

ResultSet rs = session.execute(query);
ExecutionInfo executionInfo = rs.getExecutionInfo();
QueryTrace queryTrace = executionInfo.getQueryTrace();
```

©2014 DataStax. Do not distribute without consent.

DevCenter

- Desktop app
  - friendly, familiar, productive
  - Free

http://www.datastax.com/devcenter



©2014 DataStax. Do not distribute without consent.

Find Out More

DataStax:

- http://www.datastax.com

Getting Started:

- http://www.datastax.com/documentation/gettingstarted/index.html

Training:

- http://www.datatstax.com/training

Downloads:

- http://www.datastax.com/download

Documentation:

- http://www.datastax.com/docs

Developer Blog:

- http://www.datastax.com/dev/blog

Community Site:

- http://planetcassandra.org

Webinars:

- http://planetcassandra.org/Learn/CassandraCommunityWebinars

©2014 DataStax. Do not distribute without consent.

©2014 DataStax. Do not distribute without consent.