Neo4j Driver Documentation 1.0

Table of Contents

The Uniform Language Drivers are Drivers for the Neo4j Database that use the `Bolt` protocol.

Getting Started

To help you get up and running quickly.

Get the driver

You can download the driver source or acquire it with one of the dependency managers of your language.

Versions

Each version of a Driver supports up to four different versions of the Bolt protocol. This means, that for immediate future each driver version will work with all available version of the protocol. Once we start seeing Drivers that no longer support older versions of Bolt, we will make this information available here as an accessible overview.

Download

To dowload the driver do

Example 1. Download

```
COMING SOON
```

```
<dependencies>
    <dependency>
        <groupId>org.neo4j.driver</groupId>
        <artifactId>neo4j-java-driver</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>
</dependencies>

<repositories>
    <repository>
        <id>neo4j-snapshot-repository</id>
        <name>Neo4j Maven 2 snapshot repository</name>
        <url>http://m2.neo4j.org/content/repositories/snapshots</url>
        <snapshots>
            <enabled>true</enabled>
        </snapshots>
        <releases>
            <enabled>false</enabled>
        </releases>
    </repository>
</repositories>
```

```
var neo4j = require('lib/neo4j');
```

```
pip install neo4j-driver
```

Use the driver

Each driver has a database object for creating a driver. To use a driver, follow the simple pattern:

1. Ask the database object for a new driver.
2. Ask the driver object for a new session.
3. Use the session object to run statements. It returns an object representing the results.
4. Process the results.

Remember to exhaust the returned results so that the session is ready to run the next statement!

Example 2. Usage example

```
COMING SOON
```

```
String text = "MATCH (n) RETURN n";

// when
Statement statement = new Statement( text, NO_PARAMETERS );
```

```
var neo4j = require('neo4j');

var statement = ['MERGE (alice:Person {name:{name_a},age:{age_a}})',
  'MERGE (bob:Person {name:{name_b},age:{age_b}})',
  'CREATE UNIQUE (alice)-[alice_knows_bob:KNOWS]->(bob)',
  'RETURN alice, bob, alice_knows_bob'
];

var params = {
  name_a: 'Alice',
  age_a: 33,
  name_b: 'Bob',
  age_b: 44
};

var driver = neo4j.driver("bolt://localhost");

var streamSession = driver.session();
var streamResult = streamSession.run(statement.join(' '), params);
streamResult.subscribe({
  onNext: function(record) {
    // On receipt of RECORD
    for(var i in record) {
      console.log(i);
      console.log(record[i]);
    }
  }, onCompleted: function() {
    var summary = streamResult.summarize();
    //Print number of nodes created
    console.log('');
    console.log(summary.updateStatistics.nodesCreated());
    streamSession.close();
  }, onError: function(error) {
    console.log(error);
  }
});

var promiseSession = driver.session();
var promiseResult = promiseSession.run(statement.join(' '), params);
promiseResult.then(function(records) {
  records.forEach(function(record) {
    for(var i in record) {
      console.log(i);
      console.log(record[i]);
    }
  });
  var summary = promiseResult.summarize();
  //Print number of nodes created
  console.log('');
  console.log(summary.updateStatistics.nodesCreated());
})
.catch(function(error) {
  console.log(error);
})
.then(function(){
  promiseSession.close();
});
```

```
def test_can_run_simple_statement_from_bytes_string(self):
    session = GraphDatabase.driver("bolt://localhost").session()
    count = 0
    for record in session.run(b"RETURN 1 AS n"):
        assert record[0] == 1
        assert record["n"] == 1
        assert record.n == 1
        assert repr(record)
```

As seen above, it is possible to run statements directly from the session object. The session will take care of opening and closing the transaction. The session also provides the opportunity for a user to manage the transaction. When explicitly told to begin a new transaction, the session will return the transaction object, allowing fine-grained transaction control. See the transaction section for details.

Driver

- remove 'your'
- don’t talk about code here, this is conceptual

The `Driver` is your primary object of interest. It usually lives for the duration of your application, and from it you create `Session` to use for executing statements on the database.

Dependencies

Do we have anything more to say about dependencies that we haven’t said in the getting started section?

Example 3. Importing dependencies

```
COMING SOON
```

```
String text = "MATCH (n) RETURN n";

// when
Statement statement = new Statement( text, NO_PARAMETERS );
```

```
var neo4j = require('neo4j');

var statement = ['MERGE (alice:Person {name:{name_a},age:{age_a}})',
  'MERGE (bob:Person {name:{name_b},age:{age_b}})',
  'CREATE UNIQUE (alice)-[alice_knows_bob:KNOWS]->(bob)',
  'RETURN alice, bob, alice_knows_bob'
];

var params = {
  name_a: 'Alice',
  age_a: 33,
  name_b: 'Bob',
  age_b: 44
};

var driver = neo4j.driver("bolt://localhost");

var streamSession = driver.session();
var streamResult = streamSession.run(statement.join(' '), params);
streamResult.subscribe({
  onNext: function(record) {
    // On receipt of RECORD
    for(var i in record) {
      console.log(i);
      console.log(record[i]);
    }
  }, onCompleted: function() {
    var summary = streamResult.summarize();
    //Print number of nodes created
    console.log('');
    console.log(summary.updateStatistics.nodesCreated());
    streamSession.close();
  }, onError: function(error) {
    console.log(error);
  }
});

var promiseSession = driver.session();
var promiseResult = promiseSession.run(statement.join(' '), params);
promiseResult.then(function(records) {
  records.forEach(function(record) {
    for(var i in record) {
      console.log(i);
      console.log(record[i]);
    }
  });
  var summary = promiseResult.summarize();
  //Print number of nodes created
  console.log('');
  console.log(summary.updateStatistics.nodesCreated());
})
.catch(function(error) {
  console.log(error);
})
.then(function(){
  promiseSession.close();
});
```

```
def test_can_run_simple_statement_from_bytes_string(self):
    session = GraphDatabase.driver("bolt://localhost").session()
    count = 0
    for record in session.run(b"RETURN 1 AS n"):
        assert record[0] == 1
        assert record["n"] == 1
        assert record.n == 1
        assert repr(record)
```

Construction

A Driver object is created from a graph database singleton.

Do we have anything more to say about dependencies that we haven't said in the getting started section?

Example 4. Creating a Driver object

```
COMING SOON
```

```
String text = "MATCH (n) RETURN n";

// when
Statement statement = new Statement( text, NO_PARAMETERS );
```

```
var neo4j = require('neo4j');

var statement = ['MERGE (alice:Person {name:{name_a},age:{age_a}})',
  'MERGE (bob:Person {name:{name_b},age:{age_b}})',
  'CREATE UNIQUE (alice)-[alice_knows_bob:KNOWS]->(bob)',
  'RETURN alice, bob, alice_knows_bob'
];

var params = {
  name_a: 'Alice',
  age_a: 33,
  name_b: 'Bob',
  age_b: 44
};

var driver = neo4j.driver("bolt://localhost");

var streamSession = driver.session();
var streamResult = streamSession.run(statement.join(' '), params);
streamResult.subscribe({
  onNext: function(record) {
    // On receipt of RECORD
    for(var i in record) {
      console.log(i);
      console.log(record[i]);
    }
  }, onCompleted: function() {
    var summary = streamResult.summarize();
    //Print number of nodes created
    console.log('');
    console.log(summary.updateStatistics.nodesCreated());
    streamSession.close();
  }, onError: function(error) {
    console.log(error);
  }
});

var promiseSession = driver.session();
var promiseResult = promiseSession.run(statement.join(' '), params);
promiseResult.then(function(records) {
  records.forEach(function(record) {
    for(var i in record) {
      console.log(i);
      console.log(record[i]);
    }
  });
  var summary = promiseResult.summarize();
  //Print number of nodes created
  console.log('');
  console.log(summary.updateStatistics.nodesCreated());
})
.catch(function(error) {
  console.log(error);
})
.then(function(){
  promiseSession.close();
});
```

```
def test_can_run_simple_statement_from_bytes_string(self):
    session = GraphDatabase.driver("bolt://localhost").session()
    count = 0
    for record in session.run(b"RETURN 1 AS n"):
        assert record[0] == 1
        assert record["n"] == 1
        assert record.n == 1
        assert repr(record)
```

Lifecycle

A `Driver` object can be created from your database object. It typically lives for the duration of your application. Use a `Driver` to create `Session`s.

The only reason to create more than one driver would be if you have multiple destinations/URLs to create sessions for. (For HA purposes, or pulling data out of one database and pushing into another.)

URL Format

The URL for a Bolt Driver follows normal URL patterns, with `scheme://hostname`. The scheme is `bolt`, so a valid URL could be `bolt://host.domain`.

Configuration

(N/A, how to configure your Driver based on your server configuration.)

SSL / TLS

Traffic using the Uniform Language Drivers is encrypted by default, using SSL/TLS. If you want unencrpyted communication you can configure the Neo4j server to listen to a different port. Please see {some-handy-attribute-for-the-correct-page-in-the-manual} for information on how to do that.

Websockets

REMOVE

Accessed on the same port, since we're multiplexing. Functionally speaking we have four different ways of connecting

Standard TCP plain encrypted Websockets plain encrypted

Same functionality. Tradeoff: plain sockets are faster (SSL ~20% slower) so useful if you are in a *safe environment*.

Standard TCP sockets vs Websockts default: Standard

Websockets exist for enfironment where there is no standard socket library, (i.e. the browser).

All drivers support TLS or Plain All drivers support one or the other of Standard/Websockets, not necessarily both.

Session

All interacting with the server happens in a session.

Monitoring

Run Statement

TODO: This code example should be the first thing on the Session page.   Also, the heading is probably a waste.

Example 5. Run statement

```
COMING SOON
```

```
// when
Statement statement = new Statement( text, null );

// then
assertThat( statement.text(), equalTo( text ) );
assertThat( statement.parameters(), equalTo( NO_PARAMETERS ) );
```

```
    }
  });
  var summary = promiseResult.summarize();
  //Print number of nodes created
  console.log('');
  console.log(summary.updateStatistics.nodesCreated());
})
.catch(function(error) {
  console.log(error);
})
.then(function(){
  promiseSession.close();
});
```

```
def test_can_run_simple_statement_with_params(self):
    session = GraphDatabase.driver("bolt://localhost").session()
    count = 0
    for record in session.run("RETURN {x} AS n", {"x": {"abc": ["d", "e", "f"]}}):
        assert record[0] == {"abc": ["d", "e", "f"]}
        assert record["n"] == {"abc": ["d", "e", "f"]}
        assert repr(record)
        assert len(record) == 1
        count += 1
    session.close()
    assert count == 1
```

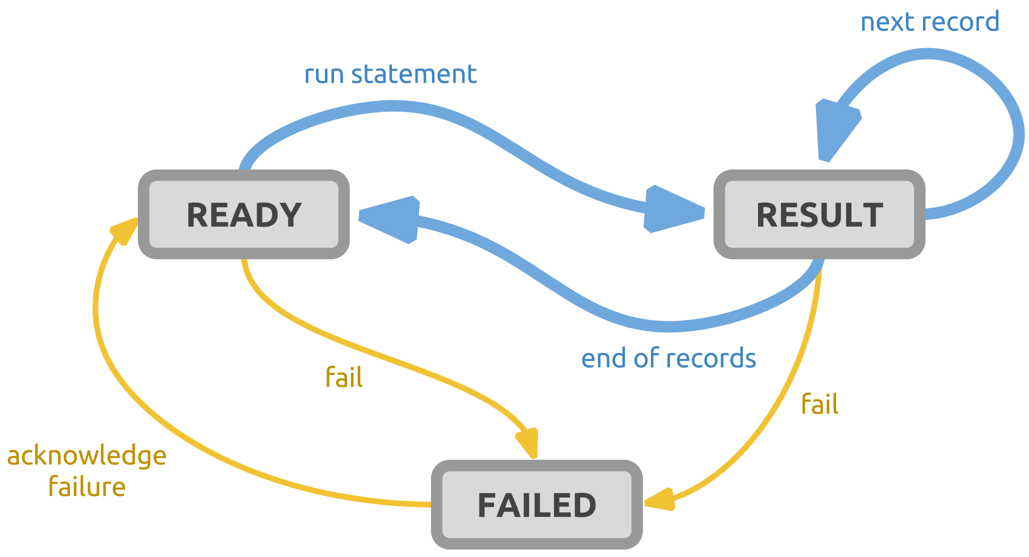
Session State

A session can be in three states: *READY*, *RESULT*, and *FAILED*.

A session begins in *READY* mode. When a query is executed, the session changes states.

If execution is successful, the session enters *RESULT* state. The results are available for consumption. When the entire result stream is consumed, the session returns to *READY* state.

If execution is not successful, the session enters *FAILED* state.



Syncing

N/A Pipelining is related to syncing.

Transactions

Read about how transactions in general work in the Neo4j Manual.

State

Implicit Transcation

On `Session.run()` an implicit transaction is created.

BEGIN, COMMIT, ROLLBACK

Nested Transactions

COMING SOON

Result

Executing a statement returns a result.

Concepts

Every result is a stream of records. A record is an ordered map of keys and values. These ordered fields (key-value pairs) can be accessed by position (integer) by key (string).

Record View

View the records.

Record Cursor

Curse the records.

TODO: Value types?

strings, numbers, nodes, relationships, paths

https://github.com/neo-technology/driver-compliance-kit/blob/08-results/08-results-synchronous.adoc

Processing

How to process results.

Example 6. Processing results

```
COMING SOON
```

```
String text = "MATCH (n) RETURN n";

// when
Statement statement = new Statement( text, NO_PARAMETERS );
```

```
var neo4j = require('neo4j');

var statement = ['MERGE (alice:Person {name:{name_a},age:{age_a}})',
  'MERGE (bob:Person {name:{name_b},age:{age_b}})',
  'CREATE UNIQUE (alice)-[alice_knows_bob:KNOWS]->(bob)',
  'RETURN alice, bob, alice_knows_bob'
];

var params = {
  name_a: 'Alice',
  age_a: 33,
  name_b: 'Bob',
  age_b: 44
};

var driver = neo4j.driver("bolt://localhost");

var streamSession = driver.session();
var streamResult = streamSession.run(statement.join(' '), params);
streamResult.subscribe({
  onNext: function(record) {
    // On receipt of RECORD
    for(var i in record) {
      console.log(i);
      console.log(record[i]);
    }
  }, onCompleted: function() {
    var summary = streamResult.summarize();
    //Print number of nodes created
    console.log('');
    console.log(summary.updateStatistics.nodesCreated());
    streamSession.close();
  }, onError: function(error) {
    console.log(error);
  }
});
```

```
var promiseSession = driver.session();
var promiseResult = promiseSession.run(statement.join(' '), params);
promiseResult.then(function(records) {
  records.forEach(function(record) {
    for(var i in record) {
      console.log(i);
      console.log(record[i]);
    }
  });
  var summary = promiseResult.summarize();
  //Print number of nodes created
  console.log('');
  console.log(summary.updateStatistics.nodesCreated());
})
.catch(function(error) {
  console.log(error);
})
.then(function(){
  promiseSession.close();
});
```

```
def test_can_run_simple_statement_from_bytes_string(self):
    session = GraphDatabase.driver("bolt://localhost").session()
    count = 0
    for record in session.run(b"RETURN 1 AS n"):
        assert record[0] == 1
        assert record["n"] == 1
        assert record.n == 1
        assert repr(record)
```

Plan + Profile

Rename to "Metadata"?

The result contains metadata. Describe it here.

Types

The Neo4j Public Type System:

- *string*
- *number*
- *boolean*

* *Node*
* *Relationship*
* *Path*

Example 7. Language mapping example

```
COMING SOON
```

```
COMING SOON
```

```
COMING SOON
```

```
COMING SOON
```

Errors

COMING SOON

Terminology

The terminology used in the Neo4j Driver manual. This section provides cross-linked summaries of common terms.

In some cases, multiple terms (e.g.,) may be used for the same or similar concept. An asterisk (*) to the right of a term indicates that the term is commonly used for Neo4j and Cypher.

bolt
: The protocol used by Neo4j drivers to communicate with the server.

driver
: An interface for operating the Neo4j database. The Neo4j drivers implement the Bolt protocol.

record
: *a result:* An ordered map of keys and values. A Neo4j driver returns results as a strem of records.