

# The REST API

Roland Guijt  
[www.rmgsolutions.nl](http://www.rmgsolutions.nl)  
[@rolandguijt](https://twitter.com/rolandguijt)



**pluralsight**   
hardcore dev and IT training

# Agenda

REST

Service Root

Indexes and  
Unique  
Constraint

Cypher via  
REST

Node and  
Relationship  
Operations

Client Access

# What You Should Know About REST

Web  
Service

HTTP

Data at  
URL

HTTP  
Methods

Hyper  
media  
Controls

# A Typical Request and Response

## Request:

**POST** http://someurl

**Accept:** application/json; charset=UTF-8

**Content-Type:** application/json

```
{  
    name: "Peter Capaldi"  
}
```

## Response:

**201:** Created

**Content-Length:** 1239

**Content-Type:** application/json; charset=UTF-8

**Location:** <http://localhost:7474/db/data/node/107>

```
{  
    <Some Data>  
}
```

## Service Root

- Provides a REST starting point
- Returns list of hypermedia links

**GET** <http://localhost:7474/db/data/>

**Accept:** application/json; charset=UTF-8

## Node Operations: Get by Id

- GET HTTP Method
- On service root node URL
- Returns data object with properties
- And hypermedia links to get the rest

**GET** <http://localhost:7474/db/data/node/1>

**Accept:** application/json; charset=UTF-8

## Node Operations: Create

- POST HTTP Method
- On service root node URL
- Returns created node

**POST** `http://localhost:7474/db/data/node`

**Accept:** `application/json; charset=UTF-8`

# Node Operations: Create with Properties

- Attach content to the POST request

**POST** http://localhost:7474/db/data/node

**Accept:** application/json; charset=UTF-8

**Content-Type:** application/json

```
{  
    name: "Peter Capaldi"  
}
```



## Node Operations: Delete

- DELETE HTTP Method

**DELETE** http://localhost:7474/db/data/node/100

**Accept:** application/json; charset=UTF-8

## Node Operations: Properties

- Use same base URL to GET all properties for a node
- PUT HTTP method: SET property on node
- Name in URL, value attached
- PUT without property name replaces all
- DELETE HTTP method: remove property from node

### PUT

<http://localhost:7474/db/data/node/1/properties/salary>

**Accept:** application/json; charset=UTF-8

**Content-Type:** application/json

100000

## Node Operations: Labels

- Like properties
- GET lists, POST adds, PUT replaces

### **POST**

**http://localhost:7474/db/data/node/1/labels**

**Accept:** application/json; charset=UTF-8

**Content-Type:** application/json

**["Person", "Actor"]**

# Relationship Operations: General

- Like nodes
- Use relationship URL
- Notable exceptions follow

## Relationship Operations: Get by node

### GET

<http://localhost:7474/db/data/node/1/relationships/all>

**Accept:** application/json; charset=UTF-8

### GET

[http://localhost:7474/db/data/node/1/relationships/all/PLAYED&REGENERATED\\_TO](http://localhost:7474/db/data/node/1/relationships/all/PLAYED&REGENERATED_TO)

**Accept:** application/json; charset=UTF-8

## Relationship Operations: Create

- POST
- Include JSON with details

### POST

`http://localhost:7474/db/data/node/1/relationships`

**Accept:** application/json; charset=UTF-8

**Content-Type:** application/json

```
{  
  "to" : "http://localhost:7474/db/data/node/19",  
  "type" : "LOVES",  
  "data" : {  
    "intensity" : "medium"  
  }  
}
```

## Node Operations: Traversals

- Traverse the graph
- One node as starting point
- Paged traversals are stored for later retrieval

### Ingredients

- URL of starting node
- What to return as URL extension  
path, fullpath, node, relationship
- Further details in attachment

# Node Operations: Traversals

**POST** http://localhost:7474/db/data/node/1/traverse/node

**Accept:** application/json; charset=UTF-8

**Content-Type:** application/json

```
{
  "order" : "breadth_first",
  "return_filter" : {
    "body" : "position.endNode().getProperty('name').toLowerCase().contains('p')",
    "language" : "javascript"
  },
  "prune_evaluator" : {
    "body" : "position.length() > 10",
    "language" : "javascript"
  },
  "uniqueness" : "node_global",
  "relationships" : [ {
    "direction" : "out",
    "type" : "REGENERATED_TO"
  }, {
    "direction" : "all",
    "type" : "PLAYED"
  } ],
  "max_depth" : 3
}
```




# Batch Operations

**POST** http://localhost:7474/db/data/batch

**Accept:** application/json; charset=UTF-8

**Content-Type:** application/json

```
[ {
  "method" : "POST",
  "to" : "/node",
  "id" : 0,
  "body" : {
    "name" : "bob"
  }
}, {
  "method" : "POST",
  "to" : "/node",
  "id" : 1,
  "body" : {
    "age" : 12
  }
}, {
  "method" : "POST",
  "to" : "{0}/relationships",
  "id" : 3,
  "body" : {
    "to" : "{1}",
    "data" : {
      "since" : "2010"
    },
    "type" : "KNOWS"
  }
}
```



# Indexes

- List all indexes for a label

**GET** <http://localhost:7474/db/data/schema/index/Actor>

**Accept:** application/json; charset=UTF-8

- Create an index on a label

**POST** <http://localhost:7474/db/data/schema/index/Actor>

**Accept:** application/json; charset=UTF-8

```
{  
  "property_keys": [ "name" ]  
}
```

- Drop index

**DELETE**

<http://localhost:7474/db/data/schema/index/Actor>

**Accept:** application/json; charset=UTF-8

# Constraints

- Like indexes
- Base URL example:

<http://localhost:7474/db/data/schema/constraint/Actor>

# Transactional Cypher Endpoint

- Execute Cypher via the REST API
- Support different output styles, all in JSON
- Transaction can remain open between requests
- Transaction can timeout

# Begin a Transaction and Commit in One Request

**POST** http://localhost:7474/db/data/transaction/commit

**Accept:** application/json; charset=UTF-8

**Content-Type:** application/json

```
{  
  "statements" : [ {  
    "statement" : "CREATE (n {props}) RETURN n",  
    "parameters" : {  
      "props" : {  
        "name" : "Peter Capaldi",  
        "salary" : 100000  
      }  
    }  
  } ]  
}
```

# Output Styles

- Specify style after statement

```
{  
  "statements": [ {  
    "statement": "CREATE (n) RETURN n",  
    "resultDataContents": [ "REST" ]  
  } ]  
}
```

- Default: Columns and contents
- REST: Same output as REST operations
- Graph: To reconstruct a graph

# Begin a Transaction

- POST to transaction base URL

POST <http://localhost:7474/db/data/transaction>

- Returns info about the transaction

201: Created

Content-Type: application/json

Location: <http://localhost:7474/db/data/transaction/7>

```
{  
  "commit" : "http://localhost:7474/db/data/transaction/7/commit",  
  "results" : .....,  
  "transaction" : {  
    "expires" : "Mon, 2 Feb 2015 20:53:51 +0000"  
  }  
}
```

# Execute Subsequent Request in Transaction

- POST to transaction returned earlier

POST <http://localhost:7474/db/data/transaction/7>

- POST to commit url for final statements in transaction

POST <http://localhost:7474/db/data/transaction/7/Commit>

- To rollback: DELETE to transaction URL

DELETE <http://localhost:7474/db/data/transaction/7>

- or let timeout expire



## Client Access

Create HTTP  
requests and  
parse JSON

- Do it yourself
- More work
- No dependency
- Total freedom

Use client  
library

- Someone else does the work
- Ready to go
- Dependency
- Maybe not entirely what you want

# Client Access Demo

Create HTTP  
requests and  
parse JSON

- C# .Net console app
- Microsoft HttpClient
- Class models for request/response

Use client  
library

- C# .Net console app
- Readify neo4jclient
- Class models for actor node

# Summary

- The REST API provides access from various platforms.
- REST accomplishes this by leveraging HTTP.
- Call the service root to get a list of URLs, called hypermedia controls, that provide a starting point.
- There are two ways to do operations on the REST API: Use pure REST operations or execute Cypher.
- To access Neo4j from your app, a client library is the easiest way, but low level HTTP calls are also a possibility.

**Thank You**

Contact me:

[roland.guijt@gmail.com](mailto:roland.guijt@gmail.com)  
**@rolandguijt**