

# Unit 3

Using the REST API

Creating and Deleting REST Instances

Loading Documents and Metadata

Document Reads and Searches

Document Deletes

DIY: Create REST Instances

Appendix: Samplestack Gradle Deployment Roadmap

## Exercise 1: Setup a MarkLogic REST API Instance with XML Configuration

In this exercise we will use an XML configuration document and the REST API to create an instance of a MarkLogic HTTP Application Server configured for REST. A content database and modules database will also be created by the REST API service.

1. Navigate to **C:\mls-developer-node\Unit03\config** and open the file **xml-rest-instance.xml** in the editor of your choice.

a. Atom, Notepad++, and Sublime Text have been provided on your Virtual Machines.

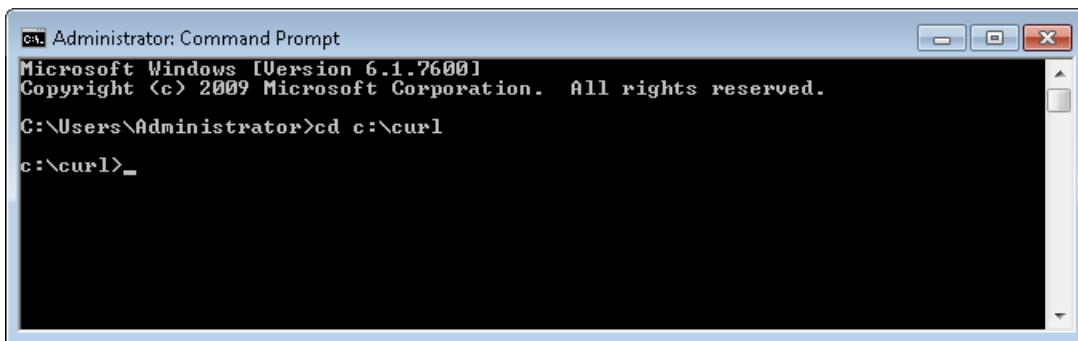
2. Study the structure of the XML configuration document:

```
<rest-api xmlns="http://marklogic.com/rest-api">
  <name>top-songs-appserver</name>
  <group>Default</group>
  <database>top-songs-content</database>
  <modules-database>top-songs-modules</modules-database>
  <port>7010</port>
  <forests-per-host>1</forests-per-host>
</rest-api>
```

3. Open an instance of the Command Prompt.

a. A shortcut to the command prompt has been placed on the Desktop.

4. At the command prompt, key **cd c:\curl** and press enter:



5. Navigate to **C:\mls-developer-node\Unit03** and open **3-1 (create REST instance-XML).txt**.

6. Study the curl statement we will execute:

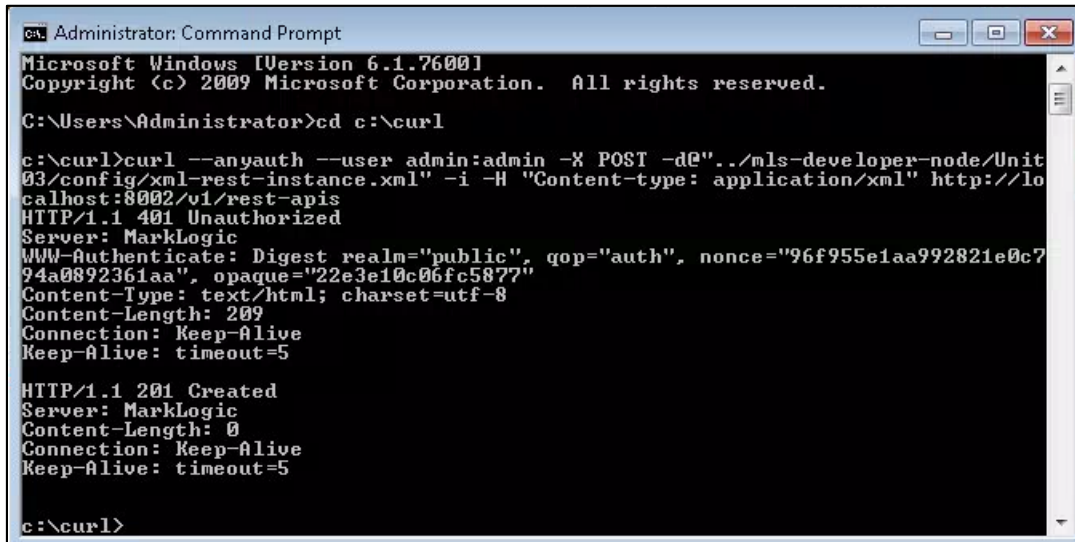
```
curl --anyauth --user admin:admin -X POST \
-d@"../mls-developer-node/Unit03/config/xml-rest-instance.xml" \
-i -H "Content-type: application/xml" \
http://localhost:8002/v1/rest-apis
```

7. Copy the curl statement from the text file and paste into the command prompt.

*Note:*

*CTRL + V does not work in the command prompt. You must Right Click → Paste.*

8. Press Enter to execute the statement, yielding the following results:



```

Administrator: Command Prompt
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>cd c:\curl

c:\curl>curl --anyauth --user admin:admin -X POST -d@'../mls-developer-node/Unit
03/config/xml-rest-instance.xml' -i -H "Content-type: application/xml" http://lo
calhost:8002/v1/rest-apis
HTTP/1.1 401 Unauthorized
Server: MarkLogic
WWW-Authenticate: Digest realm="public", qop="auth", nonce="96f955e1aa992821e0c7
94a0892361aa", opaque="22e3e10c06fc5877"
Content-Type: text/html; charset=utf-8
Content-Length: 209
Connection: Keep-Alive
Keep-Alive: timeout=5

HTTP/1.1 201 Created
Server: MarkLogic
Content-Length: 0
Connection: Keep-Alive
Keep-Alive: timeout=5

c:\curl>

```

*Note:*

*The first HTTP response that comes back is a 401 Unauthorized. This is normal due to the use of digest authentication in the request.*

*The second HTTP response is a 201 Created. This indicates that the statement was successful and the resources were created according to the XML configuration document that was referenced.*

9. Next, we will open the MarkLogic Administration tool to validate that the resources were created as outlined in our configuration XML.
10. Open an instance of a web browser.
  - a. Chrome has been provided on the VM.
11. From your browser, navigate to <http://localhost:8001>
12. When prompted to authenticate, use the following credentials:
  - a. Username = admin
  - b. Password = admin
13. You should see that the following new resources have been created:

System Summary

Summary
Status
Support
Logs
Usage
Help

**Databases (14)** — Index, query, and content processing configuration

- App-Services
- Documents
- Extensions
- Fab
- Last-Login
- Meters
- Modules
- samplestack
- samplestack-modules
- Schemas
- Security
- top-songs-content
- top-songs-modules
- Triggers

**App Servers (6)** — Enable connections from client software

- Default :: Admin : 8001 [HTTP]
- Default :: App-Services : 8000 [HTTP]
- Default :: HealthCheck : 7997 [HTTP]
- Default :: Manage : 8002 [HTTP]
- Default :: samplestack : 8006 [HTTP]
- Default :: top-songs-appserver : 7010 [HTTP]

**Groups (1)** — Allow hosts to share a common configuration

- Default

**Forests (16)** — Manage physical content storage for databases

- App-Services
- Documents
- Extensions
- Fab
- Last-Login
- Meters
- Modules
- samplestack-1
- samplestack-2
- samplestack-3
- samplestack-modules-1
- Schemas
- Security
- top-songs-content-1
- top-songs-modules-1
- Triggers

**Security** — Resources describing the role-based security model

- Users (7)
- Roles (71)
- Execute Privileges (364)
- URI Privileges (5)
- Amps (663)
- Collections (7)
- Certificate Authorities (132)
- Certificate Templates (0)
- External Security (0)
- Credentials

**Hosts (1)** — Computers belonging to this cluster

- Default :: win-po7q9it73j8

**Clusters (1)** — Cluster configuration

- win-po7q9it73j8-cluster (Local Cluster)

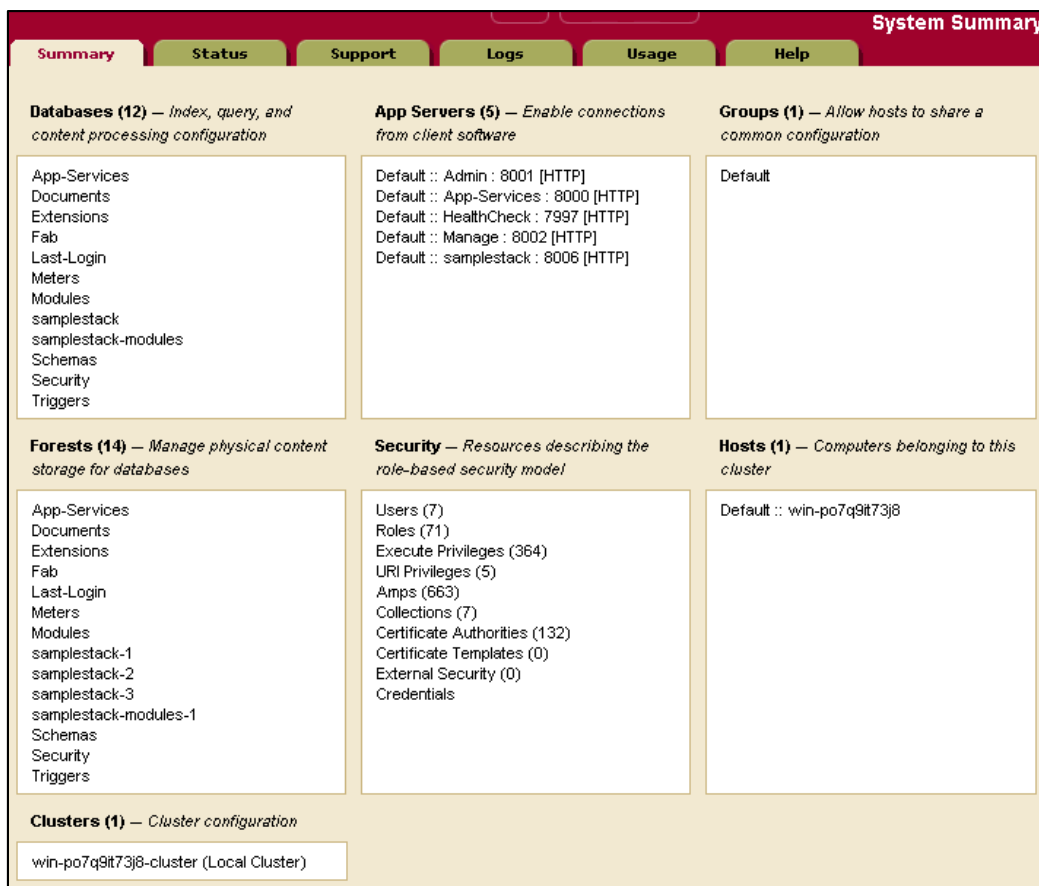
## Exercise 2: Delete a MarkLogic REST API Instance

In this exercise we will invoke a REST API service to delete the HTTP App Server, Content Database, Modules Database, and forests that we created in the prior exercise.

1. Navigate to **C:\mls-developer-node\Unit03** and open **3-2 (delete REST instance).txt**.
2. Study the curl statement we will execute:

```
curl -X DELETE --anyauth --user admin:admin \
"http://localhost:8002/v1/rest-apis/top-songs-
appserver?include=content&include=modules"
```

3. Press Enter to execute the statement.
4. In your browser, navigate to the MarkLogic Admin interface.
  - a. <http://localhost:8001>
  - b. Or if you still have the Admin interface open, simply refresh the browser (F5).
5. Validate the resources have been deleted:



The screenshot shows the MarkLogic Admin Console 'System Summary' page. The page has a top navigation bar with tabs: Summary (selected), Status, Support, Logs, Usage, and Help. The main content area is divided into several sections:

- Databases (12)** — Index, query, and content processing configuration: Lists App-Services, Documents, Extensions, Fab, Last-Login, Meters, Modules, samplestack, samplestack-modules, Schemas, Security, and Triggers.
- App Servers (5)** — Enable connections from client software: Lists Default :: Admin : 8001 [HTTP], Default :: App-Services : 8000 [HTTP], Default :: HealthCheck : 7997 [HTTP], Default :: Manage : 8002 [HTTP], and Default :: samplestack : 8006 [HTTP].
- Groups (1)** — Allow hosts to share a common configuration: Lists Default.
- Forests (14)** — Manage physical content storage for databases: Lists App-Services, Documents, Extensions, Fab, Last-Login, Meters, Modules, samplestack-1, samplestack-2, samplestack-3, samplestack-modules-1, Schemas, Security, and Triggers.
- Security** — Resources describing the role-based security model: Lists Users (7), Roles (71), Execute Privileges (364), URI Privileges (5), Aps (663), Collections (7), Certificate Authorities (132), Certificate Templates (0), External Security (0), and Credentials.
- Hosts (1)** — Computers belonging to this cluster: Lists Default :: win-po7q9it73j8.
- Clusters (1)** — Cluster configuration: Lists win-po7q9it73j8-cluster (Local Cluster).

### Exercise 3: Setup a MarkLogic REST API Instance with JSON Configuration

In this exercise we will use a JSON configuration document and the REST API to create an instance of a MarkLogic HTTP Application Server configured for REST. A content database and modules database will also be created by the REST API service.

1. Navigate to **C:\mls-developer-node\Unit03\config** and open the file **json-rest-instance.json**
2. Study the structure of the JSON configuration document:

```
{
  "rest-api":
  {
    "name": "top-songs-appserver",
    "group": "Default",
    "database": "top-songs-content",
    "modules-database": "top-songs-modules",
    "port": "7010",
    "forests-per-host": 1
  }
}
```

3. Open an instance of the Command Prompt and key **cd c:\curl** and press enter.
4. Navigate to **C:\mls-developer-node\Unit03** and open **3-3 (create REST instance-json).txt**.
5. Study the curl statement:

```
curl --anyauth --user admin:admin -X POST \
-d@"../mls-developer-node/Unit03/config/json-rest-instance.json" -i \
-H "Content-type: application/json" \
http://localhost:8002/v1/rest-apis
```

6. Copy the curl statement from the text file and paste it into the command prompt.
7. Press Enter.
8. You should see the following results:

```

Administrator: Command Prompt

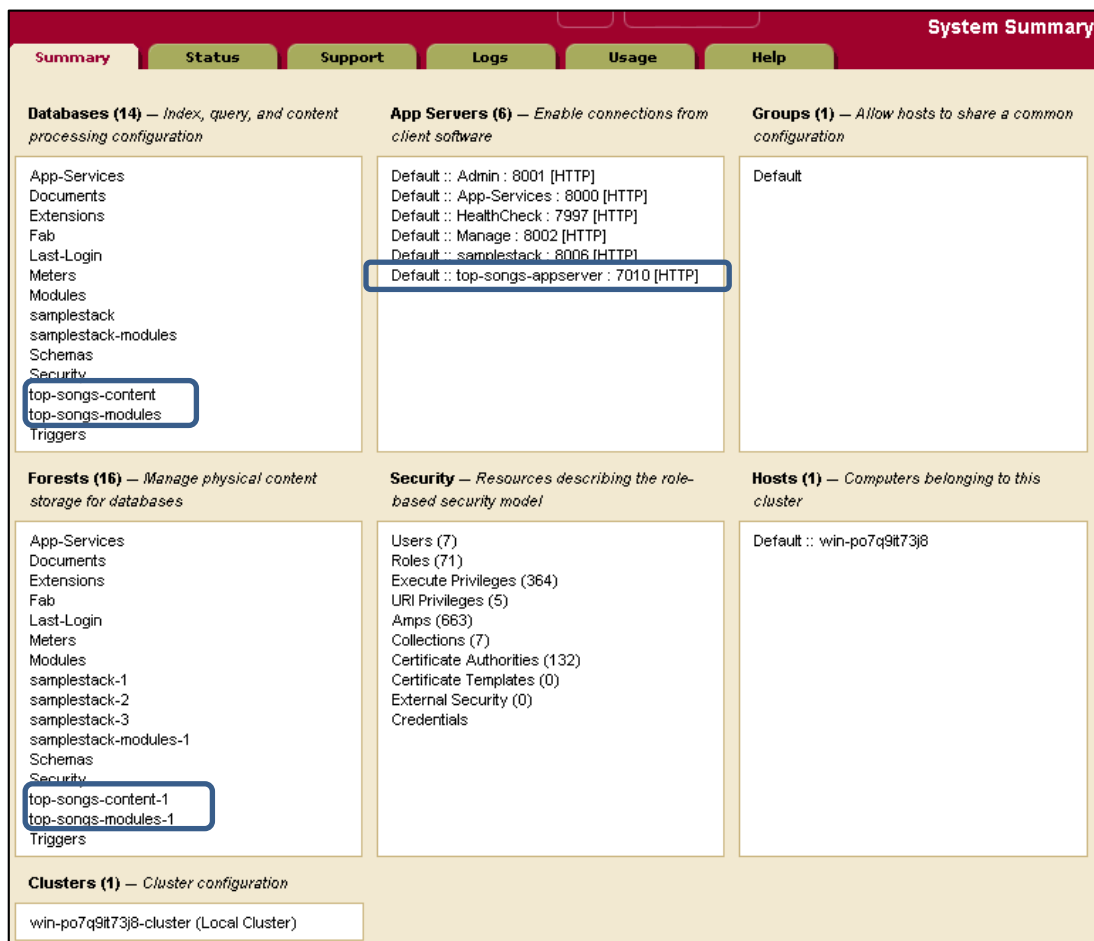
c:\curl>curl --anyauth --user admin:admin -X POST -d"../mls-developer-node/Unit
03/config/json-rest-instance.json" -i -H "Content-type: application/json" http:/
localhost:8002/v1/rest-apis
HTTP/1.1 401 Unauthorized
Server: MarkLogic
WWW-Authenticate: Digest realm="public", qop="auth", nonce="3b5d1b4979a89f78af85
a28009b31b88", opaque="cfd7aedic4ed0db7b"
Content-Type: text/html; charset=utf-8
Content-Length: 209
Connection: Keep-Alive
Keep-Alive: timeout=5

HTTP/1.1 201 Created
Server: MarkLogic
Content-Length: 0
Connection: Keep-Alive
Keep-Alive: timeout=5

c:\curl>

```

- Open your web browser and navigate to the MarkLogic Administration tool (<http://localhost:8001>) to validate that everything has been created as defined in the JSON configuration document:



## Exercise 4: Load an XML Document

In this exercise you will use the REST API to load an XML document into a database.

1. Navigate to **C:\mls-developer-node\Unit03\docs** and open the file **song1.xml** in an editor.
2. Take a minute to view the contents of the document that we will load:

```
<top-song>
  <title>Against the Wind</title>
  <artist>Bob Seger</artist>
</top-song>
```

3. Navigate to **C:\mls-developer-node\Unit03** and open **3-4 (load an XML document).txt**.
4. Study the curl statement:

```
curl --anyauth --user admin:admin -X PUT \
-T ../mls-developer-node/Unit03/docs/song1.xml \
"http://localhost:7010/v1/documents?uri=/songs/song1.xml&format=xml&collection=music&collection=classic rock"
```

5. Copy the curl statement from the text file and paste it into the command prompt.
6. Press Enter.
7. Next we will validate the document was loaded using the Query Console tool.
8. Open a browser window and navigate to **http://localhost:8000/qconsole**
9. Login with your MarkLogic Admin account (admin / admin).
10. From the **Content Source** drop down menu, select **top-songs-content(top-songs-modules)**

Content Source:
top-songs-content (top-songs-modules ▼)
Explore

11. Click the **Explore** button to view the contents of the database.
12. Notice the document we inserted now exists:

top-songs-content (top-songs-modules: /) 1 Documents			
Document	Format	Properties	Collections
/songs/song1.xml	top-song	(no properties)	music, classic rock



## Exercise 5: Load a JSON Document with Metadata

In this exercise you will use the REST API to load a JSON document into a database.

1. Navigate to **C:\mls-developer-node\Unit03\docs** and open the file **song2.json** in an editor.
2. Take a minute to view the contents of the document that we will load:

```
{
  "top-song": {
    "title": "Free Falling",
    "artist": "Tom Petty and the Heartbreakers"
  }
}
```

3. Navigate to **C:\mls-developer-node\Unit03** and open **3-5 (...).txt**.
4. Study the curl statement:

```
curl --anyauth --user admin:admin -X PUT \
-T ../mls-developer-node/Unit03/docs/song2.json \
"http://localhost:7010/v1/documents?uri=/songs/song2.json&format=json&collection=music
&collection=classic rock&prop:album=Full Moon Fever&prop:misc=some additional
metadata"
```

5. Copy the curl statement from the text file and paste it into the command prompt.
6. Press Enter.
7. Next we will validate the document was loaded using the Query Console tool.
8. Go back to your Query Console session (or if you have closed it, open a browser window and navigate to **http://localhost:8000/qconsole**
9. From the **Content Source** drop down menu, select **top-songs-content(top-songs-modules)**

Content Source:
▼

10. Click the **Explore** button to view the contents of the database.
11. Notice that both the XML and JSON documents we inserted now exist in the database:

top-songs-content (top-songs-modules: /) 2 Documents			
Document	Format	Properties	Collections
/songs/song1.xml	top-song	(no properties)	music, classic rock
/songs/song2.json	object	(properties)	music, classic rock

12. Click on the **properties** link for the **/songs/song2.json** URI to view the document metadata:

```
<?xml version="1.0" encoding="UTF-8"?>
<prop:properties xmlns:prop="http://marklogic.com/xdmp/property">
  <misc>some additional metadata</misc>
  <album>Full Moon Fever</album>
</prop:properties>
```

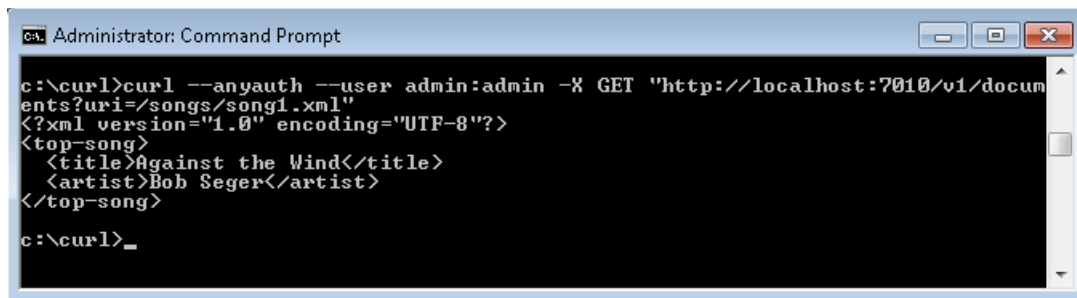
## Exercise 6: Read a Document

In this exercise we will use the REST API to read a document from a database.

1. Navigate to **C:\mls-developer-node\Unit03** and open **3-6 (...).txt**.
2. Study the **first** curl statement:

```
curl --anyauth --user admin:admin -X GET \
"http://localhost:7010/v1/documents?uri=/songs/song1.xml"
```

3. Copy / Paste into the command prompt and press Enter.
4. View the results as follows:



```
Administrator: Command Prompt

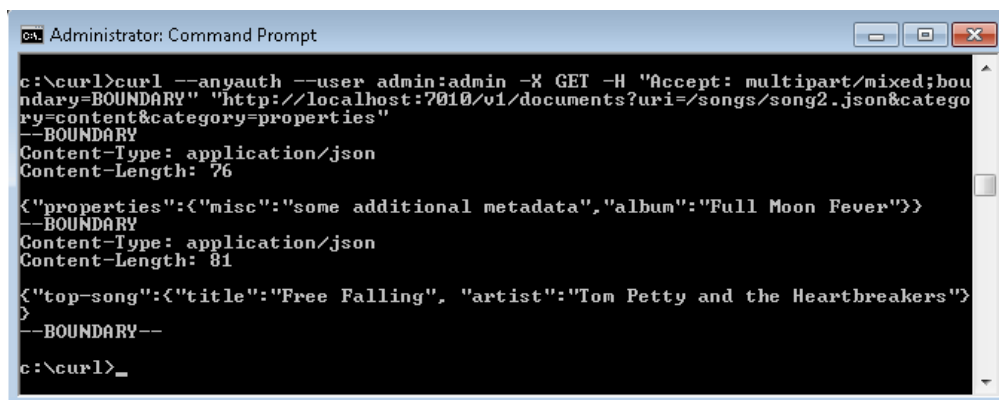
c:\curl>curl --anyauth --user admin:admin -X GET "http://localhost:7010/v1/documents?uri=/songs/song1.xml"
<?xml version="1.0" encoding="UTF-8"?>
<top-song>
  <title>Against the Wind</title>
  <artist>Bob Seger</artist>
</top-song>

c:\curl>_
```

5. Navigate to **C:\mls-developer-node\Unit03** and open **3-6 (...).txt**.
6. Study the **second** curl statement:

```
curl --anyauth --user admin:admin -X GET -H "Accept: multipart/mixed;boundary=BOUNDARY" \
"http://localhost:7010/v1/documents?uri=/songs/song2.json&category=content&category=properties"
```

7. Copy / Paste into the command prompt and press Enter. View the results as follows:



```
Administrator: Command Prompt

c:\curl>curl --anyauth --user admin:admin -X GET -H "Accept: multipart/mixed;boundary=BOUNDARY" "http://localhost:7010/v1/documents?uri=/songs/song2.json&category=content&category=properties"
--BOUNDARY
Content-Type: application/json
Content-Length: 76

{"properties":{"misc":"some additional metadata","album":"Full Moon Fever"}}
--BOUNDARY
Content-Type: application/json
Content-Length: 81

{"top-song":{"title":"Free Falling", "artist":"Tom Petty and the Heartbreakers"}}
--BOUNDARY--

c:\curl>_
```

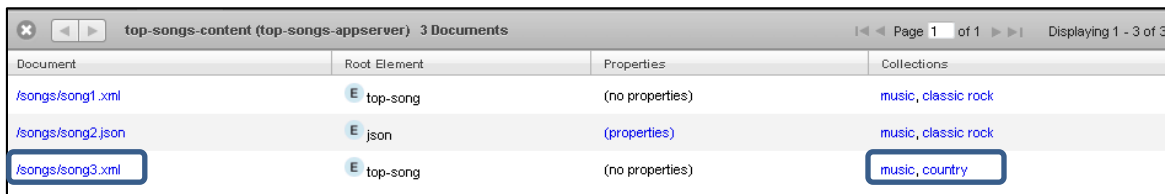
## Exercise 7: Searching for Matching Documents

In this exercise we will learn to invoke the REST API to execute queries against a dataset, as well as gain some exposure to some basic search concepts in MarkLogic such as grammar and stemming.

1. Navigate to **C:\mls-developer-node\Unit03** and open **3-7 (...).txt**.
2. Study the **first** curl statement:

```
curl --anyauth --user admin:admin -X PUT \
-T ../mls-developer-node/Unit03/docs/song3.xml \
"http://localhost:7010/v1/documents?uri=/songs/song3.xml&format=xml&collection=music&c
ollection=country"
```

3. Copy / Paste into the command prompt and press Enter.
4. Use the Query Console tool to validate that the document was loaded into the database and the collections were defined correctly:



Document	Root Element	Properties	Collections
/songs/song1.xml	top-song	(no properties)	music, classic rock
/songs/song2.json	json	(properties)	music, classic rock
/songs/song3.xml	top-song	(no properties)	music, country

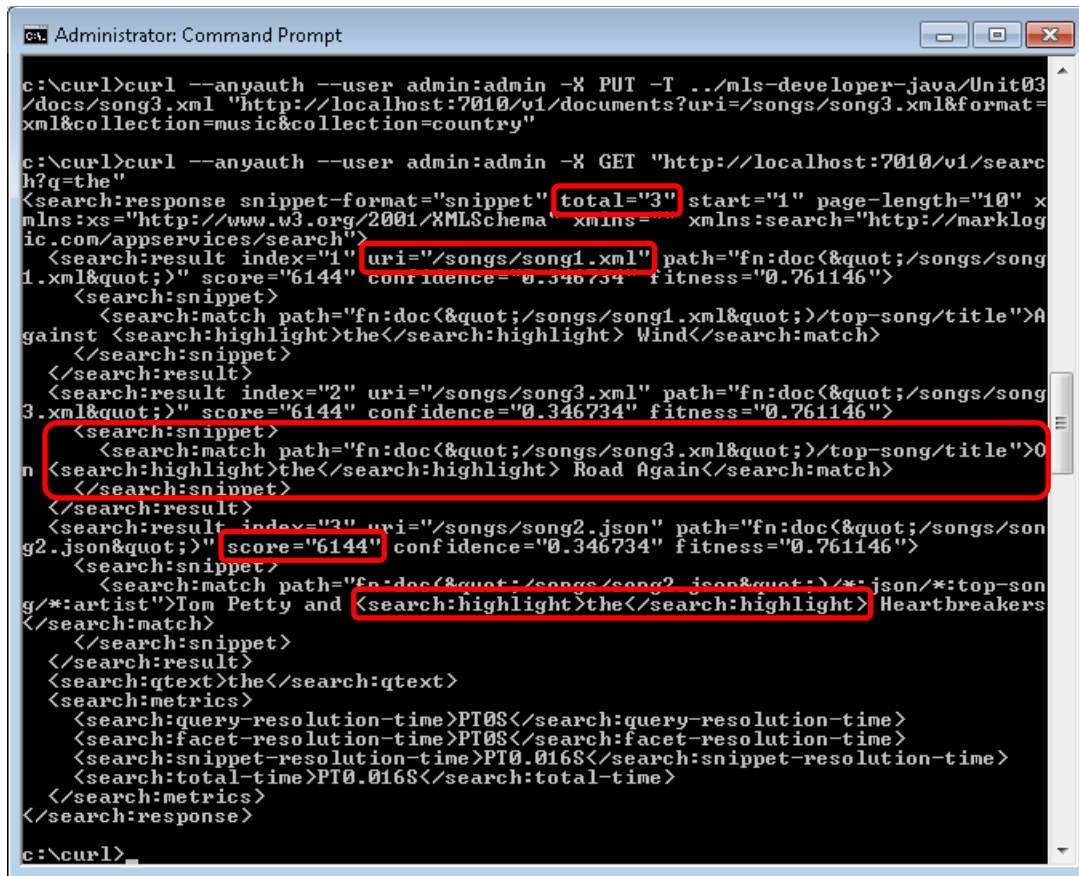
5. Review the contents of the document at URI **/songs/song3.xml**
  - a. You can accomplish this with a curl statement or by using the Query Console tool.

```
<?xml version="1.0" encoding="UTF-8"?>
<top-song>
  <title>On the Road Again</title>
  <artist>Willie Nelson</artist>
</top-song>
```

6. Next we will write some queries against our database using the REST **/v1/search/** service.
7. Navigate to **C:\mls-developer-node\Unit03** and open **3-7 (...).txt**.
8. Study the second curl statement:

```
curl --anyauth --user admin:admin -X GET "http://localhost:7010/v1/search?q=the"
```

9. Copy / Paste into the command prompt and press Enter to view the results:



```

Administrator: Command Prompt

c:\curl>curl --anyauth --user admin:admin -X PUT -T ../mls-developer-java/Unit03/docs/song3.xml "http://localhost:7010/v1/documents?uri=/songs/song3.xml&format=xml&collection=music&collection=country"

c:\curl>curl --anyauth --user admin:admin -X GET "http://localhost:7010/v1/search?q=the"
<search:response snippet-format="snippet" total="3" start="1" page-length="10" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:search="http://marklogic.com/appservices/search">
  <search:result index="1" uri="/songs/song1.xml" path="fn:doc(/songs/song1.xml)>
    <search:snippet>
      <search:match path="fn:doc(/songs/song1.xml)>the</search:match>
      <search:highlight>the</search:highlight> Wind</search:match>
    </search:snippet>
  </search:result>
  <search:result index="2" uri="/songs/song3.xml" path="fn:doc(/songs/song3.xml)>
    <search:snippet>
      <search:match path="fn:doc(/songs/song3.xml)>the</search:match>
      <search:highlight>the</search:highlight> Road Again</search:match>
    </search:snippet>
  </search:result>
  <search:result index="3" uri="/songs/song2.json" path="fn:doc(/songs/song2.json)>
    <search:snippet>
      <search:match path="fn:doc(/songs/song2.json)>the</search:match>
      <search:highlight>the</search:highlight> Heartbreakers</search:match>
    </search:snippet>
  </search:result>
  <search:qtext>the</search:qtext>
  <search:metrics>
    <search:query-resolution-time>PT0S</search:query-resolution-time>
    <search:facet-resolution-time>PT0S</search:facet-resolution-time>
    <search:snippet-resolution-time>PT0.016S</search:snippet-resolution-time>
    <search:total-time>PT0.016S</search:total-time>
  </search:metrics>
</search:response>

c:\curl>

```

10. Notice the XML format of the search results, paying attention to the following:
  - a. **<search:response>** Contains attributes about the overall search, including the **total** documents that matched our query. In this case, all 3 documents in the database matched.
  - b. **<search:result>** This markup exists for each document that matches the query, and it contains details about the match such as its **relevancy score**, the **document URI**, the context surrounding the matching term (known as a **snippet**), and the **highlight** markup around the matching term itself. This response data will be important as we build a search application and interact with these results using the Java API.
11. Navigate to **C:\mls-developer-node\Unit03** and open **3-7 (...).txt**.
12. Study the third curl statement:

```
curl --anyauth --user admin:admin -X GET \
"http://localhost:7010/v1/search?q=the&collection=classic rock"
```

13. Copy / Paste into the command prompt and press Enter to view the results:

```

Administrator: Command Prompt

c:\curl>curl --anyauth --user admin:admin -X GET "http://localhost:7010/v1/search?g=the&collection=classic rock"
<search:response snippet-format="snippet" total="2" start="1" page-length="10" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="" xmlns:search="http://marklogic.com/appservices/search">
  <search:result index="1" uri="/songs/song1.xml" path="fn:doc(/songs/song1.xml)>" score="6144" confidence="0.346734" fitness="0.761146">
    <search:snippet>
      <search:match path="fn:doc(/songs/song1.xml)>/top-song/title">Against the Wind</search:match>
    </search:snippet>
  </search:result>
  <search:result index="2" uri="/songs/song2.json" path="fn:doc(/songs/song2.json)>" score="6144" confidence="0.346734" fitness="0.761146">
    <search:snippet>
      <search:match path="fn:doc(/songs/song2.json)>/*:json/*:top-song/*:artist">Tom Petty and the Heartbreakers</search:match>
    </search:snippet>
  </search:result>
</search:response>

c:\curl>_
  
```

14. Notice that since we used a collection to limit the data that we queried, we now only have 2 matching documents returned.
15. Navigate to **C:\mls-developer-node\Unit03** and open **3-7 (...).txt**.
16. Study the fourth curl statement. Notice that we are no longer using the **/v1/search/** service, and are now using the **/v1/keyvalue/** service:

```
curl --anyauth --user admin:admin -X GET \
"http://localhost:7010/v1/keyvalue?element=title&value=Free Felling"
```

17. Copy / Paste into the command prompt and press Enter to view the results:

```

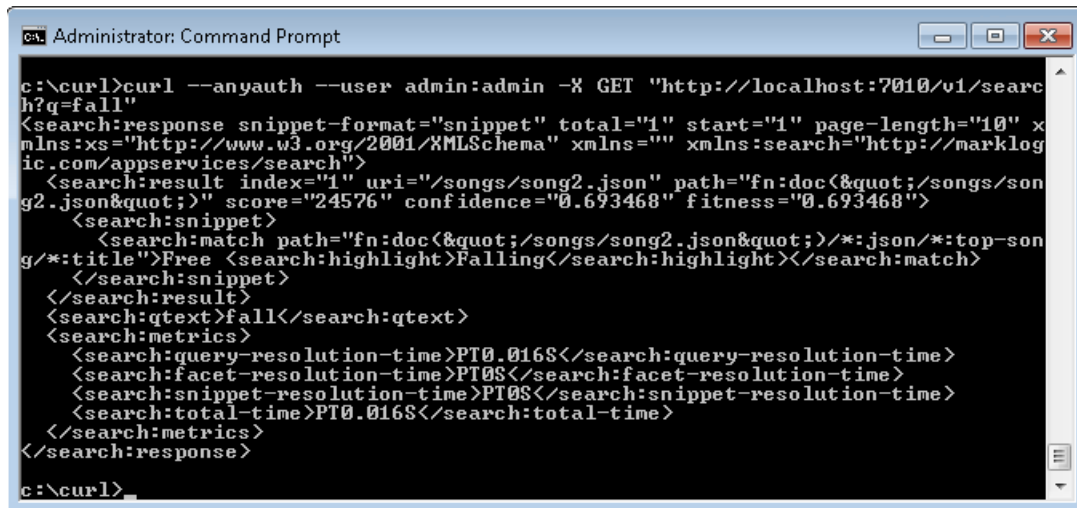
c:\curl>curl --anyauth --user admin:admin -X GET "http://localhost:7010/v1/keyvalue?element=title&value=Free Felling"
<search:response snippet-format="snippet" total="1" start="1" page-length="10" xmlns:search="http://marklogic.com/appservices/search">
  <search:result index="1" uri="/songs/song2.json" path="fn:doc(/songs/song2.json)>" score="24576" confidence="0.6934684" fitness="0.6934684" href="/v1/documents?uri=%2Fsongs%2Fsong2.json" mimetype="application/json" format="json">
    <search:snippet>
      <search:match path="fn:doc(/songs/song2.json)>/object-node()">Free Felling Tom Petty and the Heartbreakers</search:match>
    </search:snippet>
  </search:result>
  <search:metrics>
    <search:query-resolution-time>PT0.016S</search:query-resolution-time>
    <search:facet-resolution-time>PT0S</search:facet-resolution-time>
    <search:snippet-resolution-time>PT0S</search:snippet-resolution-time>
    <search:total-time>PT0.016S</search:total-time>
  </search:metrics>
</search:response>
  
```

18. Navigate to **C:\mls-developer-node\Unit03** and open **3-7 (...).txt**.

19. Study the fifth curl statement:

```
curl --anyauth --user admin:admin -X GET "http://localhost:7010/v1/search?q=fall"
```

20. Copy / Paste into the command prompt and press Enter to view the results:



```
Administrator: Command Prompt
c:\curl>curl --anyauth --user admin:admin -X GET "http://localhost:7010/v1/search?q=fall"
<search:response snippet-format="snippet" total="1" start="1" page-length="10" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns="" xmlns:search="http://marklogic.com/appservices/search">
  <search:result index="1" uri="/songs/song2.json" path="fn:doc(&quot;/songs/song2.json&quot;)" score="24576" confidence="0.693468" fitness="0.693468">
    <search:snippet>
      <search:match path="fn:doc(&quot;/songs/song2.json&quot;)/*:json/*:top-song/*:title">Free <search:highlight>Falling</search:highlight></search:match>
    </search:snippet>
  </search:result>
  <search:qtext>fall</search:qtext>
  <search:metrics>
    <search:query-resolution-time>PT0.016S</search:query-resolution-time>
    <search:facet-resolution-time>PT0S</search:facet-resolution-time>
    <search:snippet-resolution-time>PT0S</search:snippet-resolution-time>
    <search:total-time>PT0.016S</search:total-time>
  </search:metrics>
</search:response>
c:\curl>
```

21. Notice that one document is returned when we search for the term **“fall”**:

- a. Title: Free Falling
- b. Artist: Tom Petty and the Heartbreakers

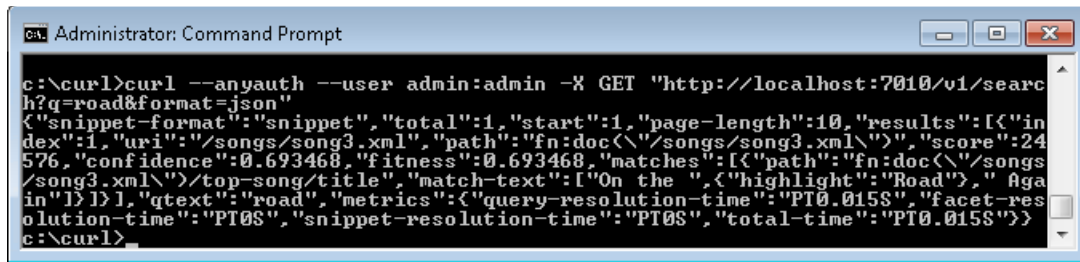
22. Under the covers, the **/v1/search/** service utilizes the MarkLogic Search API, which implements a concept called **stemming**. This is why a search for the term **fall** matches a document containing the word **falling**. MarkLogic understands that the root of the word falling is equal to fall. We’ll dig deeper into these concepts later on.

23. Navigate to **C:\mls-developer-node\Unit03** and open **3-7 (...).txt**.

24. Study the sixth curl statement:

```
curl --anyauth --user admin:admin -X GET \
"http://localhost:7010/v1/search?q=road&format=json"
```

25. Copy / Paste into the command prompt and press Enter to view the results:



```
c:\curl>curl --anyauth --user admin:admin -X GET "http://localhost:7010/v1/search?q=road&format=json"
{"snippet-format":"snippet","total":1,"start":1,"page-length":10,"results":[{"index":1,"uri":"/songs/song3.xml","path":"fn:doc<\"/songs/song3.xml\">","score":24576,"confidence":0.693468,"fitness":0.693468,"matches":[{"path":"fn:doc<\"/songs/song3.xml\">/top-song/title","match-text":["On the ","highlight":"Road"}, {"path":"fn:doc<\"/songs/song3.xml\">/top-song/title","match-text":["Aga in"]}]}],"qtext":"road","metrics":{"query-resolution-time":"PT0.015S","facet-resolution-time":"PT0S","snippet-resolution-time":"PT0S","total-time":"PT0.015S"}}
```

26. Notice that we have decided to return the results as JSON (even though the matching document is stored in the database as XML). As a developer you have control over the format of the response data. When using the Node.js client API, we will work with JSON data in the response.



## Exercise 8: Delete a Document

In this exercise we use the REST API `/documents/` service to delete a document from our database.

1. Navigate to **C:\mls-developer-node\Unit03** and open **3-8 (...).txt**.
2. Study the curl statement:

```
curl --anyauth --user admin:admin -X DELETE \
"http://localhost:7010/v1/documents?uri=/songs/song3.xml"
```

3. Copy / Paste into the command prompt and press Enter to view the results.
4. Explore the contents of the database using the Query Console tool (<http://localhost:8000>) to validate that the document was deleted:

top-songs-content (top-songs-appserver) 2 Documents			
Document	Root Element	Properties	Collections
/songs/song1.xml	top-song	(no properties)	music, classic rock
/songs/song2.json	json	(properties)	music, classic rock

## **DIY: Create REST Instances**

As you progress through this course you will be creating several different projects, specifically the Star Wars and Geophoto projects. Each of these projects will require a REST instance.

Apply what you've learned throughout the lab exercises in this unit to create a REST instance for the Star Wars and Geophoto projects. The REST instances should meet the following requirements:

### **Star Wars Configuration Requirements:**

1. REST instance name = star-wars
2. Group = Default
3. Port = 5002
4. Database = star-wars-content
5. Modules database = star-wars-modules
6. Only create 1 forest per host

### **Geophoto Configuration Requirements:**

1. REST instance name = geophoto
2. Group = Default
3. Port = 5003
4. Database = geophoto-content
5. Modules database = geophoto-modules
6. Only create 1 forest per host

## Appendix: Gradle Deployment Roadmap

The REST API plays a large role in the automated deployment of Samplestack. In this appendix we will walk through step by step what is occurring during the automated deployment of the middle tier and MarkLogic resources through the use of Gradle, Groovy, the MarkLogic Java API and the MarkLogic Management REST API. Note that the first release of the Node.js version of Samplestack uses the same Gradle deployment that the Java version of Samplestack implements, which is documented here in this appendix. The next release of the Node.js version of Samplestack will implement a full JavaScript deployment.

1

Launch the batch file from the command line passing in a parameter:

```
c:\marklogic-samplestack\appserver\java-spring>gradlew appserver
```

2

The batch file invokes Gradle, passing it the parameter **appserver**:

```
org.gradle.wrapper.GradleWrapperMain %CMD_LINE_ARGS%
```

3

The **appserver** task is identified within the **build.gradle** file:

```
36 /*
37 * appserver
38 * assembles the project, loads sample data and starts the appserver on port 8090
39 */
40 task appserver <<
41 {
42     println "Bootstrapping, seeding and starting Samplestack appserver"
43 }
```

4

The **appserver** task dependencies are identified and executed:

```
141 /**
142 * appserver dependency
143 */
144 appserver.dependsOn(clean, assemble, dbLoad, bootRun)
145 bootRun.shouldRunAfter(dbLoad)
```

5

Execute the **clean** task.

- The **clean** task is a standard Gradle task that deletes the build directory if it exists.

6

Now that **clean** has completed its job, we need to continue with the remaining dependencies of the **appserver** task. Specifically the **assemble** task:

```
141 /**
142  * appserver dependency
143  */
144 appserver.dependsOn(clean, assemble, dbLoad, bootRun)
145 bootRun.shouldRunAfter(dbLoad)
```

7

Execute the **assemble** task.

- The **assemble** task has some defined dependencies in our **build.gradle** file. These dependencies must first be executed:

```
85 /**
86  * ASSEMBLE
87  */
88 assemble.dependsOn(dbInit, dbConfigure, test)
```

8

Execute the **dbInit** task.

- The **dbInit** task has some defined dependencies in our **build.gradle** file. These dependencies must first be executed:

```
79 dbInit.dependsOn(dbConfigureClean)
```

9

Execute the **dbConfigureClean** task.

- The **dbConfigureClean** task deletes the **/build/database/** directory if it exists. It will be rebuilt fresh during deployment.

```
69 task dbConfigureClean << {
70     delete("${buildDir}/database")
71 }
```

10

Now that the dependency for **dbInit** is complete, **dbInit** will proceed.

- The **dbInit** task calls **MarkLogicInitTask.groovy** and passes it configuration information about roles and users.

```
46 /*
47 * INIT
48 * Run this task to initialize a fresh MarkLogic server
49 * and install security objects.
50 */
51 task dbInit(type: MarkLogicInitTask)
52 dbInit.roles = file("../..../database/security/roles")
53 dbInit.users = file("../..../database/security/users")
```

- For more information on role and user configuration see:
  - /marklogic-samplestack/database/security/roles
  - /marklogic-samplestack/database/security/users

11

The real action when **dbInit** gets called is defined in the groovy task.

- Within Eclipse find:
  - /marklogic-samplestack/buildSrc/src/main/groovy/
- Open **MarkLogicInitTask.groovy**
- Study the code. Notice it uses the MarkLogic Management REST API to perform the following actions:
  - Initialize the MarkLogic instance.
  - Create the MarkLogic admin account.
  - Create the MarkLogic users and roles for the application.
  - Create a MarkLogic REST instance.
- Within Eclipse find and open:
  - /marklogic-samplestack/gradle.properties
  - Study the properties. Note that this is where certain MarkLogic configuration details are defined.

12

Now that **dbInit** has completed its job, we need to continue with the remaining dependencies of the **assemble** task. Specifically the **dbConfigure** task:

```
85 /**
86 * ASSEMBLE
87 */
88 assemble.dependsOn(dbInit, dbConfigure, test)
```

13

Execute the **dbConfigure** task.

- The **dbConfigure** task references another Groovy task:

```
67 task dbConfigure(type: MarkLogicConfigureTask)
```

- Within Eclipse open **MarkLogicConfigureTask.groovy**
- Study the code. Notice it uses the MarkLogic Management REST API to perform the following actions:
  - Configure the Samplestack database.
  - Put some transforms, extensions and query options into the modules database.
  - Configure the MarkLogic REST instance

14

Now that **dbConfigure** has completed its job, we need to continue with the remaining dependencies of the **assemble** task. Specifically the **test** task:

```
85 /**
86  * ASSEMBLE
87  */
88 assemble.dependsOn(dbInit, dbConfigure, test)
```

15

Execute the **test** task.

- The **test** task is a standard Gradle task. In summary it will kick off all the unit and integration tests that are defined for the Samplestack application.

16

Now that **test** has completed its job, all the dependencies of the **assemble** task have now been completed.

```
85 /**
86  * ASSEMBLE
87  */
88 assemble.dependsOn(dbInit, dbConfigure, test)
```

17

Now that the **assemble** task is completed, our main **appserver** task can proceed with its next dependency, specifically the **dbLoad** task:

```
141 /**
142  * appserver dependency
143  */
144 appserver.dependsOn(clean, assemble, dbLoad, bootRun)
145 bootRun.shouldRunAfter(dbLoad)
```

18

Execute the **dbLoad** task.

- The **dbLoad** task references another Groovy task:

```
118 task dbLoad(type: MarkLogicSlurpTask) {
119     seedDirectory = file("${buildDir}/seed-data")
120     inputs.dir seedDirectory
121 }
```

- Within Eclipse open **MarkLogicSlurpTask.groovy**
- Study the code. Notice it uses the MarkLogic Java API to load all the Samplestack seed data.

19

Now that **dbLoad** has completed its job, we can move onto the final dependency of the **appserver** task, specifically **bootRun**.

```
141 /**
142  * appserver dependency
143  */
144 appserver.dependsOn(clean, assemble, dbLoad, bootRun)
145 bootRun.shouldRunAfter(dbLoad)
```

20

Execute the **bootRun** task.

- **bootRun** is a Spring specific Gradle task and it enables us to run the project in place on the middle tier.
- When complete you will see the following at the command prompt:

```
> Building 86% > :bootRun
```

- Now you can run Samplestack by hitting the Spring app server at:
  - <http://localhost:8090>