Java

Managing, querying, and extending Stardog with Java, Java Frameworks, and Popular RDF Java Libraries

Taught by:

Rafael Campo
Software Engineer

Learning Objectives

- Manage and query Stardog using its native Java APIs
- Manage and query Stardog using its Spring Framework support
- Set up an application to connect to Stardog via a RDF4J or Jena bridge
- Understand Stardog's extension points that allow a Java developer to modify certain behaviors to the platform at runtime

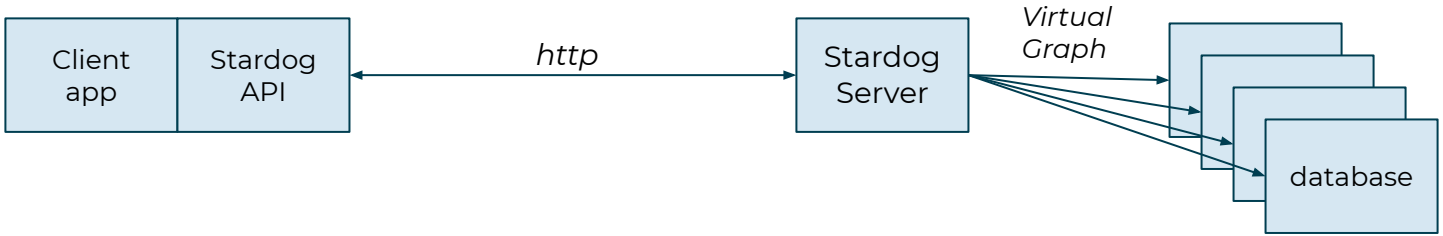
API Overview

Java API: Introduction

- SNARL (**S**tardog **N**ative **A**PI for the **R**DF **L**anguage) is the programming interface that allows developers to connect and manage their knowledge graph from Java for:
  - Performing administrative tasks
  - Working with (parameterized) SPARQL queries
  - Inference Engine
  - Integrity Constraints
- STARK provides utilities and interfaces for writing parameterized queries, managing constraints, RDF graph statements, and Axioms.

How to Use the Stardog API

- Client applications communicate with Stardog over HTTP
- Java API provides a developer friendly API around performing those HTTP transactions to interact with the server
- Typical Java development best practices apply
- Download the API and build an application that communicates with Stardog
- Stardog client libraries become client application dependencies



Where to Get the API

- Stardog client libraries published with the server
- Client bindings available from a Maven compatible repository
- Popular build tools like Maven and Gradle are easily used with Stardog artifacts
- Single top level client-http artifact used for dependency management

Access pom.xml Example

- Java APIs are available on Maven Central and can be added using your build tool of choice

Maven™

```
<repositories>
        <repository>
                <id>stardog-public</id>
                <url>http://maven.stardog.com</url>
        </repository>
</repositories>
<dependencies>
        <dependency>
                <groupId>com.complexible.stardog.core</groupId>
                <artifactId>stardog</artifactId>
                <version>7.4.0</version>
        </dependency>
        <dependency>
                <groupId>com.complexible.stardog</groupId>
                <artifactId>client-http</artifactId>
                <version>7.4.0</version>
                <type>pom</type>
        </dependency>
</dependencies>
```

Access: build.gradle Example

```
apply plugin: 'java'
apply plugin: 'application'

group 'com.stardog.examples'
version '1.0-SNAPSHOT'

repositories {
   maven { url "http://maven.stardog.com" }
   mavenLocal()
   mavenCentral()
}

dependencies {
   // Core Dependencies
   compile ('com.complexible.stardog.core:stardog:7.4.0')
   compile ('com.complexible.stardog:client-http:7.4.0')
}

mainClassName  = "com.stardog.examples.StardogClient"
```

Development

- SNARL connects to Stardog via a **Connection** or **AdminConnection** Interface
  - **AdminConnection** lets you perform administrative tasks such as creating and dropping databases
    - `ClusterAdminConnection`, `VirtualGraphAdminConnection`
  - **Connection** lets you perform tasks like executing queries and adding data
    - `BitesConnection`, `GraphQLConnection`, `ICVConnection`, `ReasoningConnection`, `SearchConnection`

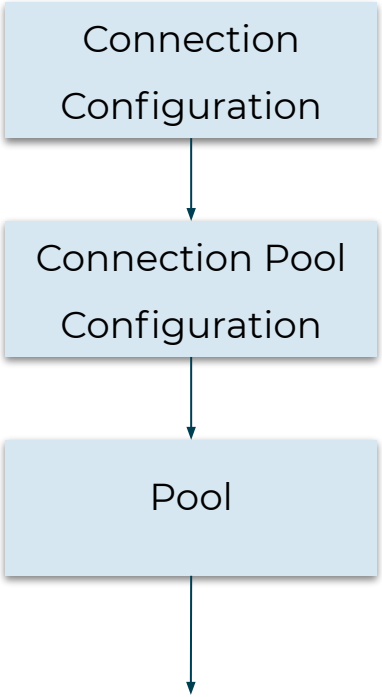
Working with the API

STARDOG ACADEMY

API Overview

- Stardog SNARL API provides a Connection object
- Conventions for creating pools of connections are followed
- Standard Database Connection Pool (DBCP) library is used
  - Apache Commons DBCP
- Connection API design provides a similar interface to Java's JDBC API
  - Connection open and close semantics
  - ResultSet
  - Exception handling

Connection Pool

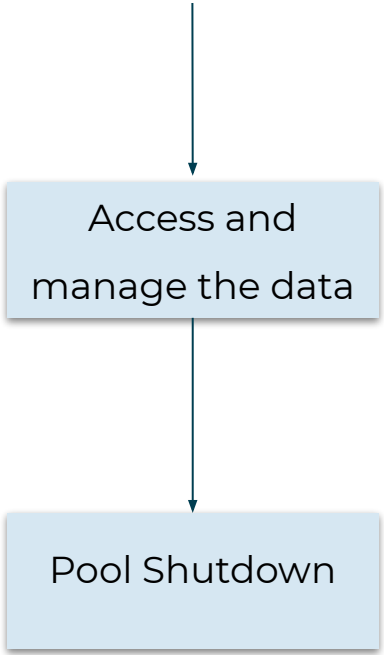



```
ConnectionConfiguration aConnConfig = ConnectionConfiguration
                    .to(dbName)
                    .server(url)
                    .credentials(username, password);

ConnectionPoolConfig aConfig = ConnectionPoolConfig
                    .using(aConnConfig)
                    .minPool(minPool)
                    .maxPool(maxPool)
                    .expiration(expirationTime, expirationTimeUnit)
                    .blockAtCapacity(blockCapacityTime, blockCapacityTimeUnit);

ConnectionPool pool = aConfig.create();
```

Connection Pool




```
Connection aConn = pool.obtain();

/*
      Do something with the connection like query,
      add, or remove data.
*/

// release the connection back to the pool
// to return that resource
pool.release(aConn);
```

```
pool.shutdown();
```

Admin Connections

- Open an AdminConnection
- List available databases contained in the Stardog Server
- If the myDb database exists delete it and create a new one with Search enabled
- At the end of this process close the admin connection

Admin Connections

```
try (final AdminConnection aConn = AdminConnectionConfiguration.toServer(url)
                .credentials(username, password)
                .connect()) {

            aConn.list().forEach(database -> System.out.println(database));

            if (aConn.list().contains(myDb)) {
                aConn.drop(myDb);
            }

            aConn.newDatabase(myDb).set(SearchOptions.SEARCHABLE, true).create();
            aConn.close();
}
```

VirtualGraphAdminConnection

- VirtualGraphAdminConnection class provides an interface for managing virtual graphs

```
try (final AdminConnection aConn = AdminConnectionConfiguration.toServer(url)
                                        .credentials(username, password)
                                        .connect()) {

      VirtualGraphAdminConnection vgConnection = aConn.as(VirtualGraphAdminConnection.class);
      vgConnection.addGraph(vgName, props, mappings);
}
```

BitesConnection

- This allows us to add files to Stadog's Document Store using the putDocument method
- Other methods available through the API:
  - getDocument - returns a document's contents
  - reindex - reindexs all documents in the document store
  - deleteDocument - deletes a specific document from the Document Store
  - clear - deletes all documents
  - documentCount - returns the number of documents in the current database

```
try (final AdminConnection aConn = AdminConnectionConfiguration.toServer(url)
                                        .credentials(username, password)
                                        .connect()) {

      BitesConnection bitesConnection = aConn.as(BitesConnection.class);
      bitesConnection.putDocument("/Users/Shared/filename.txt");

}
```

Connections for Typical Application Development

- Add data to the database
- Query data
- Handle parameter mapping
- Marshal Java object types to RDF types
- Looking at Feature connections
  - ICV
  - Reasoning
  - Search

Adding Data from a File

- Here we use a connection object to add some data from a turtle file to a database
  - The transaction starts with the begin() method to start a transaction
  - The next line uses the add() method to add data
    - The io helper class is used since we are reading from a stream and will automatically close the stream once the data has been read
  - When we are done adding data we must commit our transaction

```
connection.begin();

connection.add().io()
      .format(RDFFormats.TURTLE)
      .stream(new FileInputStream("data/sp2b.ttl"));

connection.commit();
```

Adding Data with Statements

- Here the transaction begins in the same way we did before
- A statement object is created on line two, called boldStatement and added on the third line
- Finally, we commit the transaction just as we did in the previous slide

```
connection.begin();

Statement boldStatement = Values.statement(
     Values.iri("http://example.com/subjects#MiamiDolphins"),
     Values.iri("http://example.com/predicates#areA"),
     Values.iri("http://example.com/objects#GreatTeam"));

connection.add().statement(boldStatement);

connection.commit();
```

Adding Data: Statements with Context

```
connection.begin();

Statement boldStatement = Values.statement(
     Values.iri("http://example.com/subjects#MiamiDolphins"),
     Values.iri("http://example.com/predicates#areA"),
     Values.iri("http://example.com/objects#GreatTeam"),
     Values.iri("http://example.com/context#MiamiDolphinsFans"));

connection.add().statement(boldStatement);

connection.commit();
```

Querying

- The first line creates and sets the parameters to our query
- The the second line sets the maximum number of results to be returned to ten
- The last two statement will print out the Results
    - Try-with-resources is used so that we can ensure the result set gets close

```
SelectQuery aQuery = connection.select("select * where { ?s ?p ?o }");
aQuery.limit(10);

try (SelectQueryResult aResult = aQuery.execute()) {
      System.out.println("The first ten results...");
      QueryResultWriters.write(aResult, System.out, TextTableQueryResultWriter.FORMAT);
}
```

Parameterized Querying

- Parameterized queries
  - Prevents against SPARQL injection attacks
  - Are managed via methods, string concatenation is not recommended

Parameterized Querying

- Here we bind the "**s**" variable from our previous query with the article object

```
SelectQuery aQuery = connection.select("select * where { ?s ?p ?o }");
IRI article = Values.iri("http://localhost/publications/articles/Journal1/1940/Article1");

aQuery.parameter("s", article);

try (SelectQueryResult aResult = aQuery.execute()) {
      System.out.println("\n" + "Showing resources that belong to the selected article.");
      while (aResult.hasNext()) {
            BindingSet aBindingSet = aResult.next();
            String aStrValue = aBindingSet.value("?o").toString();  // get the ?o value in the result set
       }
       result.close(); // make sure to close result sets
}
```

User Connection Types

- ReasoningConnection - Exposes specific reasoning functionality
  - Explain reasoning results
  - Consistency Checking
- ICVConnection - Integrity Constraint Validation (ICV)
  - add/remove constraints
  - Perform data validation
- SearchConnection - Provides the full-text search support

ICVConnection

- Integrity Constraint Validation (ICV) connections will perform user-defined validation against connection objects

```
Constraint aConstraint = ConstraintFactory.constraint(subClassOf(Product, some(manufacturedBy, Manufacturer)));

ICVConnection aValidator = aConn.as(ICVConnection.class);

aValidator.addConstraint(aConstraint);

// So we can check whether or not our data is valid,
// which it isn't; we're lacking the assertion that m1 is a Manufacturer.
System.out.println("The data " + (aValidator.isValid(ContextSets.DEFAULT_ONLY)
                                                ? "is"
                                                : "is NOT") + " valid!");
```

ReasoningConnection

- Here a generic query is created so that we can later bind the PERSON type to the ?type variable in the query
- We can execute the query just as before when we were not using reasoning
- With Reasoning ON, any subject that is an instance of any subclass of Person, like an Artist is a type of Person, will be counted toward our results

```
ReasoningConnection aReasoningConn = aConn.as(ReasoningConnection.class);

SelectQuery aQuery = aReasoningConn.select("SELECT (count(?s) AS ?numberOf) WHERE {" +
                "?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?type }");

aQuery.parameter("type", PERSON);

try (SelectQueryResult aResult = aQuery.execute()) {
        QueryResultWriters.write(aResult, System.out, TextTableQueryResultWriter.FORMAT);
}
```

SearchConnection

- Creates a Searcher which will a conduct full-text search against Stadog via the Waldo Semantic Search engine
- Search Parameters are set in line 2, results are obtained in line 3, and results are printed within the try with resources block

```
SearchConnection SearchCon = aConn.as(SearchConnection.class);
Searcher aSearch = aSearchConn.search()
                        .limit(50)
                        .query("mac")
                        .threshold(0.5);

SearchResults aSearchResults = aSearch.search();

try (CloseableIterator<SearchResult> result = aSearchResults.iterator()) {
                        System.out.println("\nAPI results: ");
                        while (result.hasNext()) {
                                SearchResult aHit = result.next();
                                System.out.println(aHit.getHit() + " with a score of: " + aHit.getScore());
                        }
}
```

Stardog Spring Framework Bindings

STARDOG ACADEMY

Spring Framework

- The Spring Framework integration for Stardog's RDF Database provides Spring aware beans to provide an analogous feature set to Spring's jdbcTemplate
  - DataSourceFactoryBean - for managing stored connections
  - SnarlTemplate - for transaction and connection pool safe Stardog programming
  - DataImporter - for easy bootstrapping the import of data into Stardog
- To support enterprise applications, integration is done with different Spring projects
  - stardog-spring
  - stardog-spring-batch

Spring Application Context Snippet

DataSourceFactoryBean

```
<bean name="dataSource" class="com.stardog.ext.spring.DataSourceFactoryBean">
        <property name="to" value="testdb"/>
        <property name="username" ref="admin"/>
        <property name="password" ref="admin" />
        <property name="reasoningType" value="true"/>
        <property name="url" value="http://localhost:5820"/>
</bean>
```

SnarlTemplate

```
<bean name="template" class="com.stardog.ext.spring.SnarlTemplate">
       <property name="dataSource" ref="dataSource"/>
</bean>
```

Spring Application Context Snippet

DataImporter

```
<bean name="importer" class="com.complexible.stardog.ext.spring.DataImporter">
       <property name="snarlTemplate" ref="template"/>
       <property name="format">
              <util:constant static-field="com.stardog.stark.io.RDFFormats.N3"/>
       </property>
       <property name="inputFiles">
              <list>
                     <value>classpath:foaf.rdf</value>
                     <value>classpath:marvel.rdf</value>
                     <value>classpath:marvel_v2.rdf</value>
              </list>
       </property>
</bean>
```

Spring: Setup

- The following two lines are all we need to have in our spring code. No need to think about connections that job is done by SnarlTemplate.

```
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

SnarlTemplate snarlTemplate = (SnarlTemplate) context.getBean("template");
```

Spring: Query

- The created SnarlTemplate object is all we need now to interact with the data

```
if (snarlTemplate != null) {
      String sparql = "PREFIX foaf:<http://xmlns.com/foaf/0.1/> " +
                  "select * { ?s rdf:type foaf:Person }";

      // Queries the database using the SnarlTemplate and gets back a list of mapped objects
      List<Map<String, String>> results = snarlTemplate.query(sparql, new SimpleRowMapper());
}
```

Spring: Reasoning

Reasoning set by the applicationContext

```
<bean name="dataSource" class="com.stardog.ext.spring.DataSourceFactoryBean">
        <property name="to" value="testdb"/>
        <property name="username" ref="admin"/>
        <property name="password" ref="admin" />
        <property name="reasoningType" value="true"/>
        <property name="url" value="http://localhost:5820"/>
</bean>
```

Reasoning set in the code

```
snarlTemplate.setReasoning(false);
```

Spring: Example

- Full example using a RowMapper to create a list of key value pairs in a List<Map<String,String>> results

```
String sparql = "SELECT ?a ?b WHERE { ?a ?c ?b } LIMIT 5";

Map<String, Object> params = new HashMap<String, Object>() {{
    put("c", Values.iri("http://purl.org/dc/elements/1.1/title"));
}};

List<Map<String,String>> results = tmp.query(sparql, params, new RowMapper<Map<String,String>>() {
    @Override
    public Map<String,String> mapRow(BindingSet bindingSet) {
        Map<String,String> map = new HashMap<String,String>();
        map.put("a", bindingSet.value("a").toString());
        map.put("b", bindingSet.value("b").toString());
        return map;
    }
});
```

Testing

Test Utility

- Automated Testing can be accomplished with Stardog's embedded server
  a. EmbeddedServer is only for testing
  b. Requires STARDOG_HOME to be set
  c. Requires a license to run
- In stardog-spring the embedded server can be integrated with the EmbbededProvider bean, where customization of the embedded server is possible

```
Stardog aStardog = Stardog.builder().create();

// Open an `AdminConnection` to Stardog so that we can setup the database for the example
try (AdminConnection dbms = AdminConnectionConfiguration.toEmbeddedServer()
                                                        .credentials("admin", "admin")
                                                        .connect()) {
        //...
}
```

Test Harness: Application Context

- This example sets up the embedded Server to be made available in our Tests

```
<bean name="embeddedProvider" class="com.stardog.ext.spring.EmbeddedProvider" />

<bean id="supplierFunc" class="com.stardog.ext.spring.CCSupplier" factory-method="getSupplier" />

<bean name="dataSource" class="com.stardog.ext.spring.DataSourceFactoryBean">
       <property name="to" value="testdb"/>
       <property name="provider" ref="embeddedProvider"/>
       <property name="supplier" ref="supplierFunc" />
       <property name="username" value="admin"/>
       <property name="password" value="admin"/>
</bean>

<bean name="template" class="com.stardog.ext.spring.SnarlTemplate">
       <property name="dataSource" ref="dataSource"/>
</bean>
```

Test Harness: Example

- JUnit test harness done in @Before method to handle resource creation

```
@Before
public void setUp() throws Exception {
        System.setOut(new PrintStream(outContent));
        System.setErr(new PrintStream(errContent));
        assertNotNull(dataSource);

        SnarlTemplate tmp = new SnarlTemplate();
        tmp.setDataSource(dataSource);

        DataImporter importer = new DataImporter();
        importer.setSnarlTemplate(tmp);
        importer.inputFile(RDFFormats.N3, applicationContext.getResource("classpath:sp2b_10k.n3"));
}
```

3rd Party Interoperability

3rd Party Interoperability: RDF4J

- RDF4J
  - StardogRepository: The Stardog-specific implementation of the RDF4J Repository can take a ConnectionConfiguration, as shown, or a connection string

```
// Create a RDF4J `Repository` from a Stardog `ConnectionConfiguration`. The configuration will be used
// when creating new `RepositoryConnection` objects
Repository aRepo = new StardogRepository(ConnectionConfiguration
                                          .to("testRDF4J")
                                          .credentials("admin", "admin"));

// init the repo
aRepo.initialize();

// now you can use the repository like a normal RDF4J Repository
RepositoryConnection aRepoConn = aRepo.getConnection();
```

3rd Party Interoperability: Jena

- Apache Jena
  - Stardog supports Jena via an internal bridge available as SDJenaFactory.
  - Stardog implements the Jena interfaces such that Jena APIs can be used once reference is done

```
//We obtain a Jena `Model` for a specified stardog database which is backed by a connectionObject
Model aModel = SDJenaFactory.createModel(connectionObject);
```

Extension Points

Extension Points

- Stardog's extension points utilize the JDK ServiceLoader to make them available at runtime
- Provide direct customization to the Stardog Server
- Typically used in advanced use-cases

Extension Points by Use Case

- Monitoring, Alerting, and Admission Control
  - **HTTP-Handler:** Customized HTTP endpoints
  - **Transaction Listener**
- Custom Text Search or Additional Language Support
  - **Search Analyzer**
    - Lucene's Standard Analyzer set by default
    - Lucene supports over 30 languages
    - Customizations implement the Analyzer Factory

Extension Points by Use Case

- Customized Query Behaviour
  - **Query Functions**
    - Extensible Value Testing SPARQL specification: users can define their own own SPARQL functions
  - **Describe**
    - Allows users to change the output of the SPARQL Describe query
    - Out of the box support:
      - Bi-directional
      - Concise Bounded Description (CBD)

Learning Objectives

- Manage and Query Stardog using its native Java APIs
- Manage and Query Stardog using its Spring Framework support
- Setup an application to connect to Stardog via a RDF4J or Jena bridge
- Understand Stardog's extension points that allow a Java developer to modify certain behaviors to the platform at runtime

Thank you

References

- Stardog Documentation:
  - Stardog Java Programming: https://www.stardog.com/docs/#_java_programming
  - Stardog Java API: https://www.stardog.com/docs/java/snarl/
  - Spring Programming: https://www.stardog.com/docs/#_spring_programming

- Tutorials:
  - SNARL Tutorial - Build a Java App in 5 steps https://www.stardog.com/tutorials/how-to-build-java-app-in-stardog/
  - Stardog Spring Tutorial - Springtime for Stardog https://www.stardog.com/blog/springtime-for-stardog/

- Github Repositories
  - SNARL and Stark API Examples: https://github.com/stardog-union/stardog-examples/tree/develop/examples/api/main/src/com/complexible/stardog/examples/api
  - Stardog Spring Bindings: https://github.com/stardog-union/stardog-spring

References: Java Interoperability

- Jena
  - Example
    https://github.com/stardog-union/stardog-examples/blob/develop/examples/api/main/src/com/complexible/stardog/examples/jena/JenaExample.java
  - Documentation
    https://github.com/stardog-union/stardog-examples/blob/develop/examples/api/main/src/com/complexible/stardog/examples/rdf4j/RDF4JExample.java

- RDF4J
  - Example
    https://github.com/stardog-union/stardog-examples/blob/develop/examples/api/main/src/com/complexible/stardog/examples/rdf4j/RDF4JExample.java
  - Documentation
    https://github.com/stardog-union/stardog-examples/blob/develop/examples/api/main/src/com/complexible/stardog/examples/rdf4j/RDF4JExample.java

References: Stardog Extensions

- Search Analyzer
  - Example https://github.com/stardog-union/stardog-examples/tree/develop/examples/analyzer
  - Documentation https://docs.stardog.com/developing/extending-stardog/search-analyzers

- Transaction Listener
  - Example https://github.com/stardog-union/stardog-examples/tree/develop/examples/connectable
  - Documentation https://docs.stardog.com/operating-stardog/database-administration/#transactions

References: Stardog Extensions

- Describe Customization
  - Example https://github.com/stardog-union/stardog-examples/tree/develop/examples/describe
  - Documentation https://docs.stardog.com/query-stardog/#describe-queries
- HTTP Handler
  - Example https://github.com/stardog-union/stardog-examples/tree/develop/examples/http_handler
  - Documentation https://docs.stardog.com/developing/extending-stardog/http-server
- Customized Functions
  - Example https://github.com/stardog-union/stardog-examples/tree/develop/examples/function
  - Documentation https://docs.stardog.com/developing/extending-stardog/query-functions