EventStoreDB Java Environment Installation

Overview:

The following are step-by-step instructions for setting up and using the EventStoreDB Java client using Maven.

Topics covered:

1. Creating a standalone Maven-managed Java project
2. Add eventstoredb client and associated dependencies to the project
3. Add configuration to make an executable jar file
4. Configuring the project for staging to GitHub
5. Starting the eventstoredb docker container
6. Creating an EventStoreDB client and appending and reading events to/from a stream
7. Summary of classes created in this project

This document provides a simple overview of installing and utilizing EventStoreDB's Java client. Your Java projects may be significantly more complex, but this provides a baseline working example to help you get started.

Definition of terms:

- **Maven:** Commonly used Java project management tool. Configuration is managed by a pom.xml file in the main directory and, optionally, pom.xml files in subdirectories.
- **Github:** Collaborative code repository management tool that provides version control.
- **EventstoreDB:** Open-source, event-native database for applications using Event Sourcing, Event-Driven Architecture, and Microservices.
- **Maven Archetype:** Maven project templating toolkit.
- **Pom.xml:** Configuration file for Maven-based projects.

Note: *The GitHub repo content is compiled by running the steps below. You do not need to run these steps to execute the sample code in either GitHub Codespaces or locally on your computer. These instructions allow you to reproduce this project independently and modify, extend, or use it as a starting point for using EventStoreDB.*

Create a Maven-managed Java Project from scratch.

Step 1a: Install maven and a jdk

1. On a Mac, open a terminal window and type the following code to **install Maven and a version of OpenJDK**.

```
brew install maven
```

2. If not on a Mac, or if you opt for a different method, follow the instructions here:
   https://maven.apache.org/download.cgi
   https://www.oracle.com/java/technologies/downloads/

3. Regardless of how you install Maven and OpenJDK, you can **test for successful installation** by running the following commands in your terminal window.

```
mvn --version
```

```
javac -version
```

```
java -version
```

Note: If you are unable or uninterested in installing Java, please refer to the CodeSpaceInstruction.pdf in the top level folder of this repository.

Step 1b: Use a Maven archetype to build a directory structure for your project.

Maven provides archetypes for preconfiguring a project. Archetypes create a management file (pom.xml) and a directory structure for you. This example uses a basic archetype.

1. Create an empty directory. Title it **JavaFromScratch**.

```
mkdir JavaFromScratch
```

2. Navigate to the JavaFromScratch directory.

```
cd JavaFromScratch
```

3. Run a maven archetype:generate command.

```
$ mvn archetype:generate
-DarchetypeArtifactId=maven-archetype-quickstart
-DgroupId=com.eventstoredb_demo -DartifactId=eventstoredb-demo
-DinteractiveMode=false
```

This command does the following

A. Creates a folder named eventstoredb-demo
B. Creates a pom.xml file in that directory
C. Creates a folder structure as pictured below

```
eventstoredb-demo
├── pom.xml
└── src
    ├── main
    │   └── java
    │       └── com
    │           └── eventstoredb_demo
    │               └── App.java
    └── test
        └── java
            └── com
                └── eventstoredb_demo
                    └── AppTest.java
```

Note: *Typically, you would use a groupId specific to your project. But for this demonstration,* **`com.eventstoredb_demo`** *will work. More on groupID From the Maven documentation:*

> "groupID: This element indicates the unique identifier of the organization or group that created the project. The groupId is one of the key identifiers of a project and is typically based on the fully qualified domain name of your organization. For example org.apache. maven."

Additional details for the pom.xml file

Pom.xml is an XML file that contains the project and configuration details used by Maven to build the project. Pom.xml files regularly become significantly more complex, but at this point, your pom.xml file would look like this:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.eventstoredb_demo</groupId>
    <artifactId>eventstoredb-demo</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>eventstoredb-demo</name>
    <url>http://maven.apache.org</url>
    <dependencies>
      <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>3.8.1</version>
        <scope>test</scope>
      </dependency>
    </dependencies>
  </project>
```

Step 1c: Compiling your project and running the sample application.

Often while developing code in an IDE such as vscode, or intellij, you test your code by running/debugging within the IDE.

You can however compile the code into a jar and execute classes from that jar file.

Here are instructions.

The Maven archetype has a sample Java class “App” that prints “hello world” to the console.

1. Compile your project and run the “hello world” App to verify everything works before connecting to EventStoreDB. To start, navigate to the eventstoredb-demo directory.

```
cd eventstoredb-demo
```

2. Run the following command to compile the Java classes and place them in a jar file in a newly created directory named target

```
mvn package
```

3. View the contents of the jar. Once the command runs, you should see, among other artifacts, the presence of our compiled code (com/eventstoredb_demo/App.class)

```
jar -tvf target/eventstoredb-demo-1.0-SNAPSHOT.jar
```

4. Run the class. Once the command runs, you should see **`hello world`** returned to the console.

```
java -cp target/eventstoredb-demo-1.0-SNAPSHOT.jar
com.eventstoredb_demo.App
```

Congratulations! You have successfully built a Maven-managed Java project, compiled some Java code into a jar, and executed a class from that jar!

Step 2: Adding EventstoreDB client dependencies to the project

The pom.xml file includes the Maven project dependencies. This example uses an EventStoreDB client to connect to EventStoreDB to write and read events. The two Jackson dependencies are included to write JSON-formatted events. *Jackson is a popular Java-based library for serializing or mapping Java objects to JSON.*

Working with eventstoredb will require the following dependencies. If you are building this "From Scratch" use the text editor of your choice and add these lines to your pom.xml file.

If you are using a downloaded copy of our repo as a starting point, confirm that these entries are present by viewing your "pom.xml" file.

```
<dependency>
      <groupId>com.eventstore</groupId>
      <artifactId>db-client-java</artifactId>
      <version>5.3.2</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
    <version>2.17.0</version>
</dependency>
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
```

```
    <artifactId>jackson-databind</artifactId>
    <version>2.17.0</version>
</dependency>
```

Step 3: Add configuration details to build an executable jar file

Your project now has dependencies. Adding the following configuration to the pom.xml file will cause the " **`mvn package`** " to create a jar that will contain the needed dependencies inside the jar file.

The following entries from the pom.xml are needed to build a jar with dependencies.

If you are creating a project locally "From Scratch" and want to build jars with bundled dependencies, add the following to your pom.xml file.

If you are starting with a downloaded copy of our repo, confirm the entries are present.

```
<build>
 <plugins>
 <plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-assembly-plugin</artifactId>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>single</goal>
      </goals>
      <configuration>
        <archive>
        <manifest>
         <mainClass>
          com.eventstoredb_demo.App
          </mainClass>
        </manifest>
        </archive>
        <descriptorRefs>
          <descriptorRef>jar-with-dependencies</descriptorRef>
        </descriptorRefs>
      </configuration>
    </execution>
  </executions>
 </plugin>
</plugins>
```

```
</build>
```

Step 4: Configuring the project for staging to GitHub

The project contains source code and artifacts built upon the execution of the " **`mvn package`**" command.

Best practice is to store source code and configuration files in GitHub while excluding compiled artifacts.

You can accomplish this with a ".gitignore" file. There are standard .gitignore examples for Maven and Java. The .gitignore file for this project looks like the following.

```
# maven gitignore copied from
# https://github.com/github/gitignore/blob/main/Maven.gitignore
# also java gitignore copied from
# https://github.com/github/gitignore/blob/main/Java.gitignore
# maven

target/
pom.xml.tag
pom.xml.releaseBackup
pom.xml.versionsBackup
pom.xml.next
release.properties
dependency-reduced-pom.xml
buildNumber.properties
.mvn/timing.properties
.mvn/wrapper/maven-wrapper.jar

# Eclipse m2e generated files
# Eclipse Core
.project
# JDT-specific (Eclipse Java Development Tools)
.classpath

# java
# Compiled class file
*.class

# Log file
*.log
```

```
# BlueJ files
*.ctxt

# Mobile Tools for Java (J2ME)
.mtj.tmp/

# Package Files #
*.jar
*.war
*.nar
*.ear
*.zip
*.tar.gz
*.rar

# virtual machine crash logs, see
http://www.java.com/en/download/help/error_hotsp
ot.xml
hs_err_pid*
replay_pid*
```

Congratulations! You have successfully configured a Java project ready to be managed by GitHub!

Step 5: Starting the eventstoredb docker container

To run this code in GitHub Codespaces, run the following command to start the docker container in Codespaces.

```
./start_cluster.sh
```

If you run the code locally, install Docker and run this command.

```
docker run -d --name esdb-node -it -p 2113:2113 -p 1113:1113 \
    eventstore/eventstore:lts --insecure --run-projections=All \
    --enable-external-tcp --enable-atom-pub-over-http
```

Step 6: Create an EventStoreDB client, append events to a stream, and read the events

Writing events:

"SampleWrite.java" has code to write an event to eventstoredb.

You can execute it in one of two ways.

1. To run the code in Codespaces in VSCode, click "run" above the main class (probably need a screenshot)
2. Compile the jar with dependencies by running **`mvn package`** and then execute by running the following command

```
java -cp
target/eventstoredb-demo-1.0-SNAPSHOT-jar-with-dependencies.jar
com.eventstoredb_demo.SampleWrite
```

Reading events:

"SampleRead.java" has code to read the events from the stream.

You can run "SampleRead" in one of two ways.

1. Launch GitHub Codespaces and click run above the main class
2. Run "mvn package" to build a jar file, and execute with the following command

```
java -cp
target/eventstoredb-demo-1.0-SNAPSHOT-jar-with-dependencies.jar
com.eventstoredb_demo.SampleRead
```

Step 7: Summary of classes created in this project.

App.java is a class created when "mvn archetype:generate..."
App.java simply writes "hello world" to the console to verify a basic working Java environment.

TestEvent.java is required by the Jackson libraries, which transform a Java object into a JSON object. SampleWrite uses the TestEvent class to format the JSON written to EventStoreDB.

SampleWrite.java writes an event to EventStoreDB. It assumes EventStoreDB is running locally in the EventStoreDB Docker container. If it were run against Event Store Cloud or an alternate instance, you must modify the connection string to connect to that instance.

SampleRead.java reads the events back from EventStoreDB. Similar to the write, it assumes the EventStoreDB container runs locally in unsecured mode. The connection

string must be modified to connect to another EventStoreDB instance, such as Event Store Cloud.