EventStoreDB Java Environment Installation

Overview

Welcome to the Java example of Event Store's **From Scratch** series. This series lets you quickly overcome the common challenges of setting up and configuring a new development environment and focus on advancing your EventStoreDB skills.

The **From Scratch** series provides working code examples for basic reads and writes to EventStoreDB, a tested environment to run the code, and instructions that clearly describe the steps required to run the code successfully.

Each **From Scratch** repository provides the following:

- A working Github Codespaces environment
- Instructions on running EventStoreDB locally
- Instructions to set up a similar project on your own

We recommend you progress through the **From Scratch** projects in the following order:

1. Run the code in Codespaces
2. Clone the For Scratch GitHub repo, and follow the instructions to run it locally
3. Build your own project

This document provides detailed instructions for **setting up your own Java project in EventStoreDB.** ***This is the recommended third stage in Event Store's From Scratch Java series.***

Other clients in the **From Scratch** series include:

- Node
- .NET
- Python

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

This is intended as a baseline working example of an EventStoreDB Java project. Your Java projects may be significantly more complex.

Glossary

Maven: A 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: A code repository management tool that allows version control and collaboration.
EventstoreDB: An open-sourced event-native database for applications using Event Sourcing, Event-Driven Architecture, and Microservices.
Maven Archetype: A Maven project templating toolkit.
Pom.xml: Configuration file for Maven-based projects.
Jackson: Popular Java-based library for serializing or mapping Java objects to JSON.

Note: The contents of the GitHub repo result from running the following steps. You do not need to run these steps to execute the sample code in either GitHub Codespaces or locally on your computer. These instructions are included so you can reproduce, modify, or extend this project or use it as a starting point for using EventStoreDB.

1. Create a standalone Maven-managed Java project

Install Maven and a JDK

On a Mac, open a terminal window and run the following command to install Maven and a version of OpenJDK.

```
brew install maven
```

If you are not using a Mac or do not use Homebrew, please follow the following instructions to install Maven:

- https://maven.apache.org/download.cgi
- https://www.oracle.com/java/technologies/downloads/

Regardless of how you install Maven and OpenJDK, test for a successful install by running the following commands.

```
mvn --version
```

```
javac --version
```

```
java --version
```

Use a Maven archetype to build a directory structure

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.

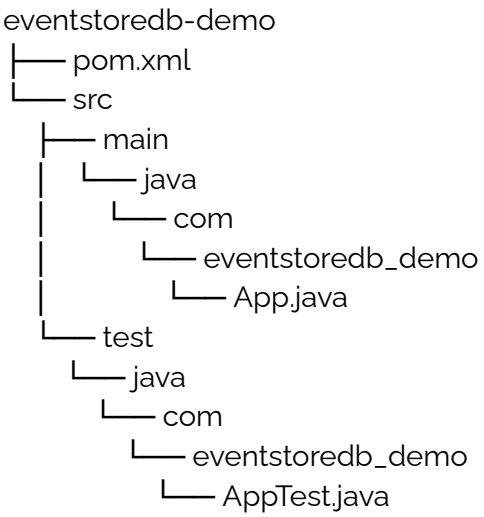
```
cd JavaFromScratch
```

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



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 should 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>
```

Compiling your project and running the sample application

While developing code in an IDE, such as VS Code or IntelliJ, code is tested by running/debugging within the IDE.

The next section provides instructions to compile the code into a jar and execute classes from that jar file.

The Maven archetype has a sample Java class "App" that prints "hello world" to the console. In later steps, you will add EventStoreDB content; the focus here is to verify that a basic "Hello World" works.

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. This command will create a directory named "target" with the compiled jar file.

```
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 Java code into a jar, and executed a class from that jar!

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.

Working with EventStoreDB requires the following dependencies. If you are building this "From Scratch" use the text editor of your choice or a vi editor to add these lines to your pom.xml file.

If you use 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>
```

Once these dependencies have been added, your pom.xml file should look similar to the following (new dependencies highlighted in green). Notice you are adding the new dependencies in the "dependencies" section of the file.

```
<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>
    <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>
  </dependencies>
</project>
```

Your project now has what it needs to connect to EventStoreDB to read and write events to streams. Refer to the SampleWrite and SampleRead files in the From Scratch repository for code samples.

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 initiate the `mvn package` to create a jar containing the needed dependencies inside the jar file.

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 experience trouble with the pom.xml file, refer to the **From Scratch** pom.xml as an example.

```
<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>
```

Once these build configurations have been added, your pom.xml file should look similar to the following (new dependencies highlighted in green). Notice you are adding the builds after the "dependencies" section of the file, but ensuring they remain in the "project."

```
<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>
    <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>
  </dependencies>
  <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>
    </project>
```

4. Configuring the project for staging to GitHub

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

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

You can accomplish this by using a ".gitignore" file. Standard .gitignore examples exist for Maven and Java. This project's .gitignore file (*located in the Java From Scratch GitHub repo*) 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!

Create a repo on GitHub and push your directory as the first commit

Navigate to your GitHub repositories, and create a new repo titled **FromScratch_Java** (or a name of your choosing) by selecting the green "New" button in the upper right of the screen.

After creating the repo, GitHub presents a page with instructions. Follow the instructions titled "**...or push an existing repository from the command line**." You will see two steps below that are not included in GitHub. Ensure you follow the steps below.

Run the following commands to link and push your local Git repo to your GitHub repo. Remember to replace `<YOUR REPO NAME>` with the name of your repository.

```
git remote add https://github.com/<YOUR REPO NAME>.git
```

```
git branch -M main
```

```
git add -A
```

```
git commit -m "first commit"
```

```
git push -u origin main
```

Congratulations! You now have a Java project that includes EventStoreDB dependencies. This can be the foundation for building more advanced and complete projects.

Next Steps

Now that you have successfully created a Java project, you have completed the **From Scratch** Java lessons. Please feel free to venture into another **From Scratch** series. Event Store offers similar content for Python, .Net, and Node.js.

Or continue your learning, you can find additional examples in the following repo:

https://github.com/EventStore/samples

In particular, we recommend the Quickstart examples here:

https://github.com/EventStore/samples/tree/main/Quickstart