

Capella Operational / Develop / Developer Tutorial /
Set Up and Connect the Couchbase Java SDK


Set Up and Connect the Couchbase Java SDK

 CAPELLA OPERATIONAL

 TUTORIAL

Learn how to configure the cluster connection and set up the Couchbase Java SDK to connect and interact with your student cluster.

Prerequisites

- You have created a Capella operational free tier cluster. For more information, see [Create and Deploy Your Free Tier Operational Cluster](#).
- You have created the required bucket, scope, and collections. For more information, see [Implement the Data Model](#).
- You have installed the Java Software Development Kit (version 8, 11, 17, or 21).
 - The recommended version is the latest Java LTS release. Make sure to install the highest available patch for the LTS version.
- You have installed [Apache Maven](#)  (version 3+).



Tip:

The easiest way to install and manage Java SDKs and Maven on your machine is through [SDKMan](#) .

After following the instructions to install SDKMan, open a terminal window and run the commands `sdk install java` and `sdk install maven` to install the latest versions of Java and Maven.

For more information about the Java SDK installation, see the [full installation page](#).

Configure Your Cluster Connection

Before you can establish a connection to your cluster, you must obtain the public connection string, add an allowed IP address, and configure cluster access credentials.

To configure your connection:

1. On the **Operational Clusters** page, click the name of your free tier operational cluster.
2. Go to **Connect > SDKs**.
3. Make a note of the **Public Connection String**. You'll need this to connect to the cluster.
4. Add your current IP address as an allowed IP on your cluster. For more information, see [Configure Allowed IP Addresses](#).
5. Create new cluster access credentials.

Make sure to save the username and password for your credentials and give the credentials **Read/Write** access to the **student-bucket** and **art-school-scope**.

For more information, see [Create Cluster Access Credentials](#).

You can now set up the Couchbase Java SDK and connect it to your cluster.

Set Up the Java SDK

To set up the Java SDK:

1. Create the following directory structure on your computer:

```
~ (your home directory)
├── student
│   ├── src
│   │   ├── main
│   │   │   └── java
```

2. In the student directory, create a new file called `pom.xml`.

```
~ (your home directory)
├── student
│   ├── pom.xml ← here!
│   ├── src
│   │   ├── main
│   │   │   └── java
```

3. Paste the following code block into your `pom.xml` file:

XML

[View](#)

[Copy](#)

```

<?xml version="1.0" encoding="UTF-8"?>
<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>org.example</groupId>
  <artifactId>couchbase-java</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>16</maven.compiler.source>
    <maven.compiler.target>16</maven.compiler.target>
    <encoding>UTF-8</encoding>
    <project.build.sourceEncoding>${encoding}</project.build.sourceEncoding>
    <project.reporting.outputEncoding>${encoding}</project.reporting.outputEncoding>
    <project.resources.sourceEncoding>${encoding}</project.resources.sourceEncoding>
    <archetype.encoding>${encoding}</archetype.encoding>
  </properties>

  <!-- The `<dependencies>` section of the code block lists all the dependencies -->
  <dependencies>
    <dependency>
      <groupId>com.couchbase.client</groupId>
      <artifactId>java-client</artifactId>
      <version>3.7.9</version>
    </dependency>
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-api</artifactId>
      <version>2.0.9</version>
    </dependency>
    <dependency>
      <groupId>org.slf4j</groupId>
      <artifactId>slf4j-simple</artifactId>
      <version>2.0.9</version>
    </dependency>
  </dependencies>

</project>

```

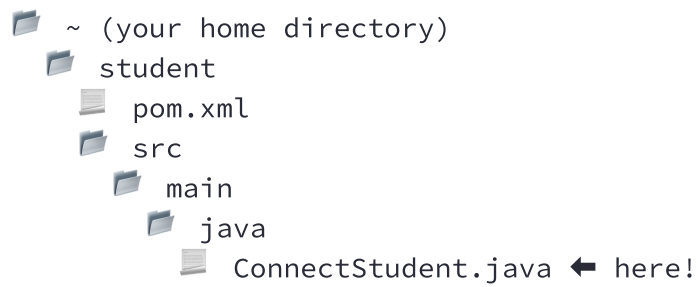
4. Open a terminal window and navigate to your student directory.
5. Run the command `mvn install` to pull in all the dependencies and finish your SDK setup.

Next, connect the Java SDK to your cluster.

Connect the SDK to Your Cluster

To connect to the cluster:

1. In your java directory, create a new file called `ConnectStudent.java`.



```
~ (your home directory)
├── student
│   ├── pom.xml
│   └── src
│       ├── main
│       └── java
│           └── ConnectStudent.java ← here!
```

2. Paste the following code block into your `ConnectStudent.java` file:

JAVA

 [View](#)

 [Copy](#)

```
import com.couchbase.client.java.Bucket;
import com.couchbase.client.java.Cluster;
import com.couchbase.client.java.Collection;
import com.couchbase.client.java.Scope;
import com.couchbase.client.java.ClusterOptions;
import java.time.Duration;

public class ConnectStudent {

    public static void main(String[] args) {

        String connectionString = "<<connection-string>>"; // Re
        String username = "<<username>>"; // Replace this with u
        String password = "<<password>>"; // Replace this with p

        //Connecting to the cluster
        Cluster cluster = Cluster.connect(connectionString, Clus
        // Use the pre-configured profile below to avoid latency
            .environment(env -> env.applyProfile("wan-de
        );

        // The `cluster.bucket` retrieves the bucket you set up
        Bucket bucket = cluster.bucket("student-bucket");

        // Most of the Couchbase APIs are non-blocking.
        // When you call one of them, your application carries o
        // When the function has finished executing, it sends a
        // While this usually works, in this code sample the app
        // This means that you now have to try to retrieve the s
        // To fix this, you can use the `waitUntilReady` call.
        // This call forces the application to wait until the bu
        bucket.waitUntilReady(Duration.ofSeconds(10));

        // The `bucket.scope` retrieves the `art-school-scope` f
        Scope scope = bucket.scope("art-school-scope");

        // The `scope.collection` retrieves the student collecti
        Collection student_records = scope.collection("student-r

        // A check to make sure the collection is connected and
        // You can see the output using maven.
        System.out.println("The name of this collection is " + s

        // Like with all database systems, it's good practice to
        cluster.disconnect();
```

```
}  
}
```

3. In the `ConnectStudent.java` file, replace the `<<connection-string>>`, `<<username>>`, and `<<password>>` placeholders with your cluster's public connection string, and the username and password from your cluster access credentials.

Important:

You must re-run `mvn install` in your student directory whenever you make a change to a Java file to rebuild your application.

4. Open a terminal window and navigate to your student directory.
5. Run the command `mvn install` to pull in all the dependencies and rebuild your application.
6. Run the following command to check that the connection works:

SH

 Copy

```
mvn exec:java -Dexec.mainClass="ConnectStudent" -Dexec.cleanupDa
```

If the connection is successful, the collection name outputs in the console log.

If you come across errors in your console, see [Troubleshooting the Developer Tutorial](#).

Next Steps

After setting up and connecting the Java SDK, you can [add student and course records to your cluster](#).

[Prev](#)[Next](#)

[< Implement the Data Model](#)

[Create Student and Course Records >](#)