Bank Client Management Documentation

Tartalom

[How to Build and Deploy the App 3](#_Toc164061294)

[Requirements 3](#_Toc164061295)

[Required runtime environments and builds tools: 3](#_Toc164061296)

[Build and Deploy (Manual) 4](#_Toc164061297)

[Launch a client-db container 4](#_Toc164061298)

[Build the client-backend microservice 4](#_Toc164061299)

[Stop the client-db container 4](#_Toc164061300)

[Use docker compose to deploy the client-backend microservice 4](#_Toc164061301)

[Successful deployment 4](#_Toc164061302)

[Build and Deploy (Automated) 5](#_Toc164061303)

[start-clean-pull-build-deploy-start-project.bat 5](#_Toc164061304)

[stop-clean-docker-build.bat 5](#_Toc164061305)

[end-clean-docker-project.bat 5](#_Toc164061306)

[Note 5](#_Toc164061307)

[How to test the App 6](#_Toc164061308)

[Switching Microservice Profiles 6](#_Toc164061309)

[Postman 6](#_Toc164061310)

[Swagger and OpenApi 7](#_Toc164061311)

# How to Build and Deploy the App

## Requirements

There are 2 options for building and deploying the app: **manual** and **automated**. Both options have the same requirements.

### Required runtime environments and builds tools:

* Java 17 (developed using OpenJDK 17.0.2)
  + Must be included in the path environment variable. For example:
    - %JAVA\_HOME%\bin
* Maven 3.8.x+ (developed using Apache Maven 3.9.6)
  + Must be included in the path environment variable. For example:
    - %MAVEN\_HOME%\bin
* Docker 25.0.x+ (developed using Docker version 25.0.3, build 4debf41)
  + Must be included in the path environment variable. For example:
    - C:\Program Files\Docker\Docker\resources\bin

## Build and Deploy (Manual)

### Launch a client-db container

The microservice (client-backend) needs a database for testing during the build. Run the **start-test-client-db.bat** file (folder: **source-root/client-db**) to launch a containerized test database for building the microservice.

### Build the client-backend microservice

Go to the **source-root/client-backend** folder and execute the command **mvn clean package**.

### Stop the client-db container

Go back to the **source-root/client-db** folder and run the **stop-clean-test-client-db.bat** file in order to stop and reset the test-client-db after a (successfull) build.

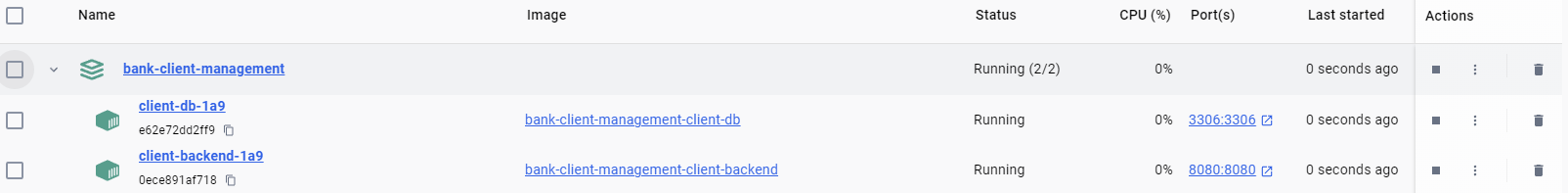
### Use docker compose to deploy the client-backend microservice

Go to the **source root** folder and execute **docker-compose up -d**. Using this command, Docker will take care of deploying the architecture contained in docker-compose.yml, including new microservice image builds and container deployments, and also the build and deployment of the database required for the microservice to run.

### Successful deployment

If all goes well, you should see something similar to the following in Docker and container logs.

**Docker Dashboard:**



**client-backend-1a9 logs ending with:**



**client-db-1a9 logs ending with:**

A képen szöveg, Betűtípus, képernyőkép látható

Automatikusan generált leírás

## Build and Deploy (Automated)

During development, I created 3 .bat scripts to automate the development lifecycle. These scripts are independent of each other. They can be executed in any order, but I prefixed them in the recommended order (**start 🡪 stop 🡪 end**) so they are easy to use.

### start-clean-pull-build-deploy-start-project.bat

This script first deletes all docker components associated with the actual project build, which means stopping the containers, deleting them and their images. But it leaves the base images (openjdk:17, mariadb) untouched, the images that serve as the base of the project images. This makes it easy to rebuild new custom images and containers during the next build.

Use this script to get a running build from the source code with just exeecuting one command.

Important:

* You will find the exact instructions for using the script at the beginning of the file.
* Using the script has some prerequisites (Java 17, Maven 3.8+, Docker 25+) without which it will not work or will work incorrectly. You will also find a detailed description of the prerequisites in comments at the beginning of the .bat file.

If the script was executed successfully you should see the following:

* Also the successful paragraph of the manual deployment section applies. Because the end result is the same.

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás

Note: Also the successful paragraph of the manual deployment section applies

### stop-clean-docker-build.bat

Use this script to stop and clean up the currently running build. This is sort of a sub-script (clean part) of the start-clean-pull-build-deploy-start-project.bat script.

### end-clean-docker-project.bat

Use this script to stop and clean up not only the currently running build (containers), but all of the Docker images (even base images) connected to this project. If you executed this script accidentally, no problem, you can use the start-clean-pull-build-deploy-start-project.bat script to recreate a running build. It will just take more time to build, because Docker will have to pull the base images (OpenJDK:17, MariaDB).

### Note

When running my .bat files, sometimes docker "errors" (not found) are seen with test-client-db-1a9. This is not an actual error, but is expected functionality. During development, I experienced that test-client-db-1a9 sometimes gets "stuck" in Docker between build processes. So I included more than the required amount of "clean up" commands in my script for test-client-db. So these commands are simply issued in a phase when, according to the optimal execution flow, the test-client-db should not exist. This causes the „not found errors”, but this has no effect on the normal flow.

The mentioned harmless error from CMD:



# How to test the App

## Switching Microservice Profiles

The **application.yml** in the resource folder (**source-root/client-backend/src/main/resources**) of the client-backend microservice contains several spring (authentication) profiles. By turning these **profiles** on and off, the microservice can operate in **different authentication modes**.

Important notes about spring profiles:

* To enable the mTLS profile, you also need to enable the TLS profile. Otherwise it will not work. This is because mTLS is an "extension" on top of TLS.
* Please **always select exactly 1** from the [**auth-disabled**, **basic-auth** and **api-key**] profiles to avoid potential conflicts caused by having multiple security configs.
* If you **do not want any authentication** then the **auth-disabled profile should be active** and tls, mtls, basic-auth and api-key profiles should be disabled.
* You can **combine (m)TLS** with **basic-auth and api-key authentication modes**, because they operate in different layers of the network.
  + Although it doesn't make much sense from an authentication point of view, because mTLS is quite enough on its own . However, it is definitely useful from an encryption point of view, because it provides a secure connection. Although it may be enough to just use tls for secure connection if another authentication method is active.

**Tip:** How to test different spring profiles quickly, easily and with your own builds?

* Enable the desired spring profile(s) in the **application.yml file**
* Execute the **start-clean-pull-build-deploy-start-project.bat** (as described above)
* Wait 20 seconds for the project to build.
* Use calls from the **Postman collection** prepared for the newly actived authentication mode.

## Postman

You can use Postman to test the microservice’s API and its different authentication methods.

I have created a postman collection, with the requests sorted into folders by authentication method. Use these precreated calls to thouroughly test the API.

Before using the calls from the collection, there are two necessary configuration steps:

* Import the collection:
  + Location: **source-root/ test-the-app-in-action-with-an-http-client/ bank\_client\_management.postman\_collection.json**
  + See details: [https://learning.postman.com/docs/getting-started/importing-and-exporting/importing-data/#import-postman-data](https://learning.postman.com/docs/getting-started/importing-and-exporting/importing-data/%23import-postman-data)
* Configure keystore used for calling the microservice:
  + Location of the preconfigured keystore: **source-root/ test-the-app-in-action-with-an-http-client/** **test.http.client.p12**
  + The keystore was generated by [Keystore Explorer](https://keystore-explorer.org/) and contains the client key, client certificate and the (trusted) certificate of the client-backend microservice.
  + During the certificate configuration, fill only the localhost, PFX and passphrase fields (as you can see below on the picture), leave the rest (CRT file, KEY file) empty.
  + More information: <https://learning.postman.com/docs/sending-requests/authorization/certificates/>

A képen szöveg, képernyőkép, Betűtípus látható

Automatikusan generált leírás A képen szöveg, képernyőkép, Betűtípus, szám látható

Automatikusan generált leírás

## Swagger and OpenApi

I integrated OpenAPI and Swagger using the „**springdoc-openapi-starter-webmvc-ui**” artifact from maven.

If you disable (watch out: negation!), the **- openapi-and-swagger-disabled** profile (source-root/client-backend/src/main/resources/application.yml) the standard Swagger page is automatically available after deployment, which provides an interactive interface for testing API endpoints using the OpenAPI standard.

I **recommend** using it when **m(tls)** and **any authentication mode is not enabled**, because its authentication capabilities are limited.

Swagger is available at the following url after a successful deployment: <http://localhost:8080/swagger-ui/index.html>