Code Conversion

As Camunda 8 is a complete rewrite of Camunda 7, you must convert your models (BPMN and DMN) and some of your code to work with the Orchestration Cluster REST API.

Overview

You must especially rewrite code that does the following:

- Uses the Client API (to start process instances for example).
- Implements service tasks, which can be:
  - External tasks, where workers subscribe to the engine.
  - Java code attached to a service task and called by the engine directly (in-VM).

This guide helps you do this if your code is written in Java, and covers the following:

- Typical code conversion patterns.
- OpenRewrite recipes as a possibility to automate refactoring.
- Diagram Converter to convert your BPMN and DMN models.
- Leveraging AI to ease refactoring.

API mapping guide

The Camunda 7 and Orchestration Cluster APIs share many similarities, but several aspects have been modernized in Camunda 8. Some of these changes are structural:

- API endpoints for retrieving or searching resources are streamlined. Instead of separate endpoints (for example, `GET /resource` and `GET /resource/count`), Camunda 8 uses a single `POST /search` endpoint.
- In Camunda 8, the `tenantId` is passed in the request body rather than as a path parameter, reducing the need for multiple endpoint variants as seen in Camunda 7.
- Camunda 8 does not have separate API endpoints for historic data.

To help you understand the differences between the two APIs, the Camunda 7 to 8 API Mapping Guidemaps the complete Camunda 7 REST API to its Camunda 8

counterparts and highlights key differences. For example, why a specific endpoint or parameter is no longer available, or if it is planned for future implementation.

Find the API mapping guide here.

Code conversion patterns

Because of the flexibility of Camunda 7, users leveraged different ways to write code, resulting in many possible conversion patterns.

- Our approach to collect these is to use a collaborative GitHub repository, where our consultants, partners, and users can add their own patterns to the catalog.
- You might still adapt the patterns to your situation, for example, if you use your own data handling or glue code abstractions.

Find the pattern catalog here.

Refactoring recipes (using OpenRewrite)

OpenRewrite is an open-source framework that can automate refactorings by so-called recipes. It is provided with an Apache License, making it easy to adopt in any context. Technically, to run recipes, you need to add a Maven plugin to your build.

Those recipes might work out-of-the-box for your environment, but most often they need to be adjusted to your code patterns. In this case, use the existing patterns as a basis to make your own adjustments or extensions.

Find the existing recipes and documentation on how to use them here.

Diagram converter

Your BPMN and DMN models need to be adjusted.

The Migration Analyzer & Diagram Converter takes care of most changes. Depending on how you refactor your code and what elements of Camunda 7 you have used, you can extend or customize the diagram converter to suit your needs.

Find the diagram conversion tooling and its documentation here.

Leveraging AI for refactoring

You can use any AI you have available to assist you with refactoring tasks. In our experiments with ChatGPT and GitHub Copilot for example, we had success by relatively simple prompts, but you needed to do a couple of rounds to make sure the AI refactors correctly and according to your target architecture whishes.

In the migration example we could let ChatGPT rewrite test cases for us with this sample prompt:
Please refactor the following Camunda 7 JUnit test case to Camunda 8 using the official migration pattern described in https://github.com/camunda-community-hub/camunda-7-to-8-code-conversion/blob/main/patterns/ALL_IN_ONE.md. The refactored test must:

- Use `@SpringBootTest` and `@CamundaSpringProcessTest`
- Use `CamundaClient` to start the process
- Use `CamundaProcessTestContext.completeUserTask(...)` to complete user tasks
- Use `CamundaProcessTestContext.increaseTime(Duration)` to simulate timer events (no manual job execution)
- Use `CamundaAssert` with `byName(...)` selectors to check activity state
- Use `assertThat(processInstance).hasVariable(...)` to check process variables

Here is the Camunda 7 test case:
[... add full test case code...]

Example: Adjusting a Spring Boot application

See end-to-end migration example on GitHub.