CAMUNDA

Camunda Platform Reference Architecture

Executive Summary

Camunda Platform offers significant flexibility with regards to architecture, deployment options, programming languages and supported infrastructure. This document covers Camunda process engine implementation options, supported infrastructure specifications, hardware sizing and recommended database management systems.

Process Engine Implementation Options

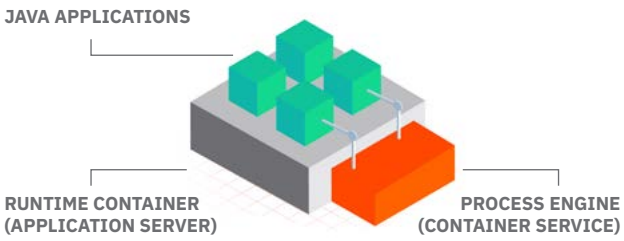
The flexibility of Camunda Platform is demonstrated with this sampling of implementation options. Typically, initial forays with Camunda use Spring Boot or a shared container though Docker is becoming a more popular option. All options work equally well and as a result there is no one recommended implementation option. And you don't have to stick to one approach for all use cases. Given our licensing flexibility you can create as many environments as needed in any topology required. Only execution metrics in your production environments count toward your license. No need to count CPUs or servers. Development and QA environments are unlimited.

Embedded Process Engine (Microservice)

The process engine is added as an application library to a custom application. This way, the process engine can easily be turned on or off with the application lifecycle. It is possible to run multiple embedded process engines on top of the same shared database.

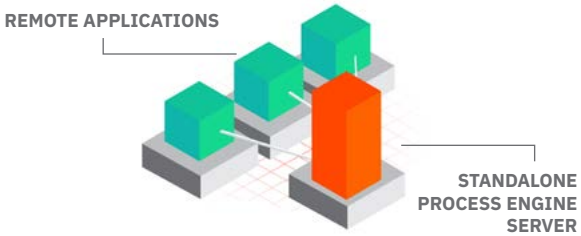
Shared, Container-Managed Process Engine

The process engine is started inside the runtime container (servlet container, application server), provided as a container service and can be shared by all applications deployed inside the container.



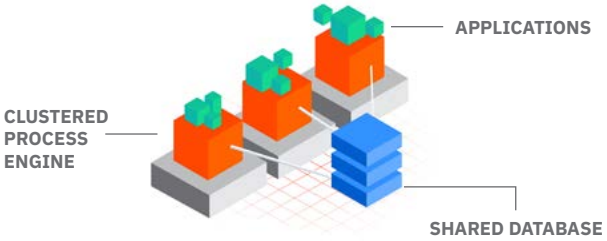
Standalone (Remote) Process Engine Server

The process engine is provided as a network service. Different applications can interact with it through remote communication, usually via the built-in REST API. Other channels such as SOAP or JMS are possible but need to be implemented by users.



Cluster Model

To provide scale-up and fail-over capabilities, the process engine can be distributed to different nodes in a cluster. Each process engine instance then connects to a shared database. The individual process engine instances do not maintain session state across transactions. Whenever the process engine runs a transaction, the complete state is flushed out to the shared database. This makes it possible to route subsequent requests which do work in the same process instance to different cluster nodes. This model is very simple and easy to manage.

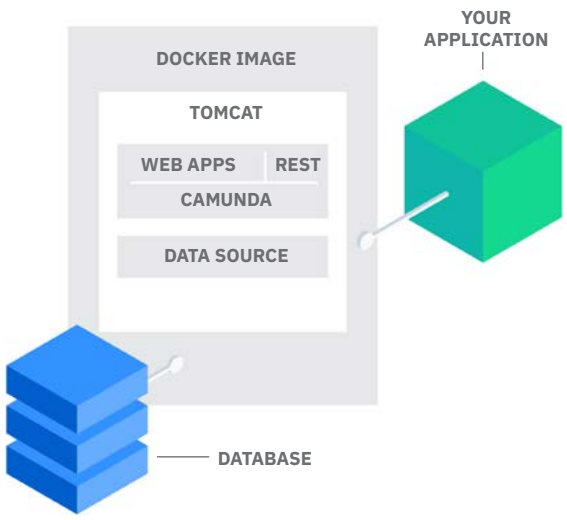


Multi-Tenancy Models

To serve multiple, independent parties with one Camunda installation, the process engine supports the following multi-tenancy models:

- Table-level data separation by using different database schemas or databases
- Row-level data separation by using a tenant marker

Users should choose the model which fits their data separation needs. Camunda's APIs provide access to processes and related data specific to each tenant.



Supported Infrastructure Options

Camunda Platform can run in any Java-runnable environment. As of version 7.16, Camunda Platform is supported with our QA infrastructure in the following environments.

Containers for Runtime Components

Application-Embedded Process Engine:

- All Java application servers
- Camunda Spring Boot Starter: embedded Tomcat
- Camunda Engine Quarkus Extension

Container-Managed Process Engine and Web Applications:

- Apache Tomcat 9.0
- JBoss EAP 7.0 / 7.1 / 7.2 / 7.3 / 7.4
- Wildfly Application Server 13.0 / 14.0 / 15.0 / 16.0 / 17.0 / 18.0 / 19.0 / 20.0 / 21.0 / 22.0 / 23.0
- IBM WebSphere Application Server 8.5 / 9.0 (**Enterprise Edition** only)
- Oracle WebLogic 12c (12R2) (**Enterprise Edition** only)

Docker

Pre-built Docker images of Camunda Platform — Enterprise Edition are available via **registry.camunda.cloud**. Packaging the components shown below, the Camunda Docker images are suitable for the remote process engine architecture.

Hardware and Sizing

Process Engine

High Availability: It is recommended to run the process engine on at least two nodes to ensure high availability. The nodes do not have to form a proper cluster in terms of an application server cluster. It is sufficient to connect two identical nodes to the same database schema.

Virtualization: Camunda can be run on virtualized systems. This does not impact licensing because licenses are not bound to CPU cores.

Resource requirements are based on expected workloads. Listed below are Camunda's recommendations:

Small	Supports most use cases, typical server configuration 1-2 CPU cores, 1-8 GB RAM
Medium	Higher volume environments averaging more than 100 instances per second, typical server configuration 2-4 CPU cores, 4-16 GB RAM
Large	Extreme volume environment or one where CPU intensive code has been deployed, typical server configuration 4-64 CPU, 16-128 GB RAM

A cluster of two small servers should suffice most common projects. Larger configurations should be considered when:

- The system needs to handle more than 100 process instances/second
- The system needs to support CPU intense delegation code or locally running services like data aggregation or transformation
- The code or deployment call for unique requirements

Load testing of deployed applications is the best approach for determining hardware sizing.

In addition, depending on the container the system requires approximately 500 MB to 1 GB of disk space. Camunda recommends at least 2 GB of storage in order to store enough logs for troubleshooting purposes.

Database Management Systems

To ensure availability, databases should be clustered and running on at least two nodes at any given time.

Recommended Database types

A large variety of database management systems (DBMS) are supported. Camunda recommends Oracle or PostgreSQL for production and H2 for development.

- MySQL 5.7 / 8.0
- MariaDB 10.2 / 10.3
- Oracle 12c / 19c
- IBM DB2 11.1 (excluding IBM z/OS for all versions)
- PostgreSQL 9.6/ 10 / 11 / 12 / 13
- Amazon Aurora PostgreSQL compatible with PostgreSQL 9.6 / 10.4 / 10.7 / 10.13 / 12.4
- Microsoft SQL Server 2014 / 2016 / 2017 / 2019 (**more information**)
- Microsoft Azure SQL with Camunda-supported SQL Server compatibility levels (**more information**)
- H2 1.4 (not recommended for production or **cluster mode**; **more information**)
- CockroachDB v20.1.3 (**more information**)

Database Clustering and Replication

Clustered or replicated databases are supported when:

- The communication between Camunda Platform and the database cluster matches the corresponding non-clustered, non-replicated configuration
- The cluster configuration guarantees the behavior of READ-COMMITTED isolation level

Galera Cluster for MariaDB is supported with specific configuration settings and some **known limitations**.

Java

Java runtimes are supported as long as they are supported by the application server or container.

Database Sizing

The amount of space required on the database depends on

1. **History Level:** Turning off history saves a huge amount of tablespace as you only keep current runtime data in the database. However, it is advised to keep it to “FULL” to get the maximum audit logging from the process engine.

2. **Process Variables** must be written to the database (in a serialized form such as JSON). With the history level “FULL,” an entry is inserted into history tables every time a variable is changed, remembering the old value. With big data objects stored and changed often, this requires a lot of space.

When calculating database size, you should also clarify if and how often you will be cleaning up historical data. The real space occupied within your database depends very much on your database product and configuration and there is no simple formula to calculate this space.