Camunda Reference Architecture

Executive Summary

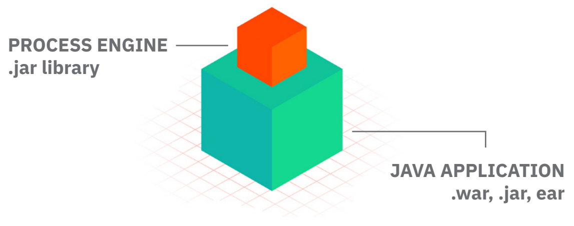
Camunda BPM offers significant flexibility with regards to architecture, deployment options, programming languages and supported infrastructure. This document covers Camunda process engine implementation options, supported infrastructure specifications, hardware sizing and recommended database management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

Process Engine Implementation Options

The flexibility of Camunda BPM is demonstrated with this sampling of implementation options. Typically, initial forays with Camunda use Spring Boot starter or Camunda Run as a standalone engine, the latter often though Docker. There are additional rarely used options like container-managed engines on Java application servers available as well. 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 to count CPUs or servers. Development and QA environments are unlimited.

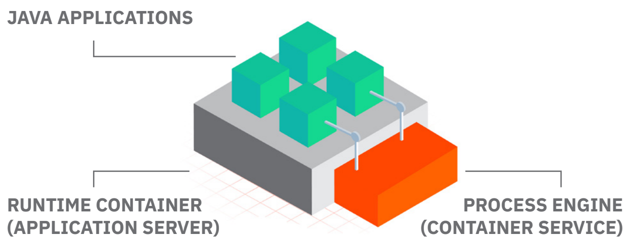
Embedded Process Engine

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. This is a typical option for microservices built in Java.



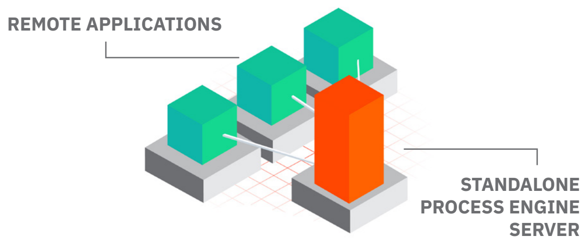
Container-Managed Process Engine

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.



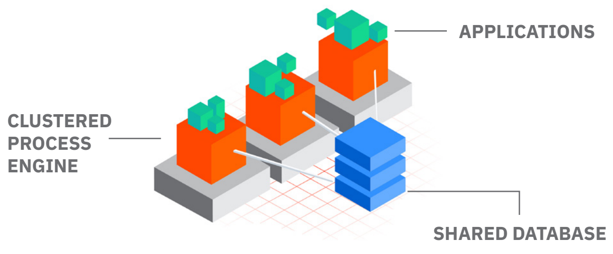
Camunda Run Standalone (Remote) Process Engine Server

The process engine is operated as a standalone remote component in the network. Different applications can interact with it through remote APIs, usually via the builtin 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

The Camunda BPM platform can be run in any Java-runnable environment. As of the current version 7.13 Camunda BPM is supported with our QA infrastructure in the following environments.

Containers for Runtime Components

Application-Embedded Process Engine:

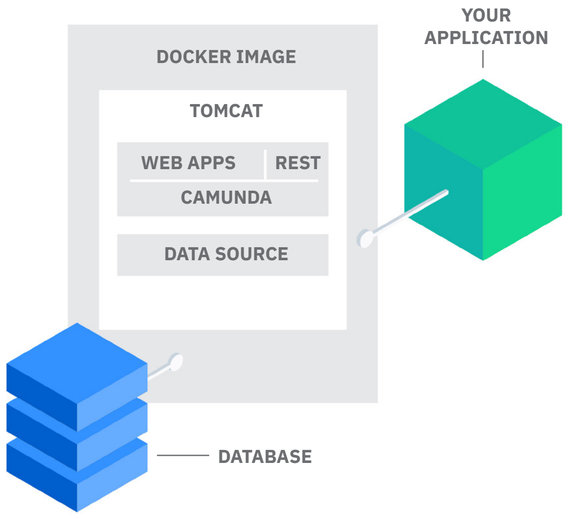
- All Java application servers
- Camunda Spring Boot Starter: embedded Tomcat

Container-Managed Process Engine and Web Applications:

- Apache Tomcat 7.0 / 8.0 / 9.0
- JBoss EAP 6.4 / 7.0 / 7.1 / 7.2
- Wildly Application Server 10.1 / 11.0 / 12.0 / 13.0 / 14.0 / 15.0 / 16.0 / 17.0 / 18.0
- IBM WebSphere Application Server 8.5 / 9.0 (Enterprise Edition only)
- Oracle WebLogic 12c (12R1,12R2) (Enterprise Edition only) and Deployment scenarios)

Docker

Pre-built Docker images of Camunda Enterprise 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 since 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 500MB - 1GB of disk space. Camunda recommends at least 2GB 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 (note: not for production use)

Supported Database types

- MySQL 5.6 / 5.7
- MariaDB 10.0 / 10.2 / 10.3
- Oracle 11g / 12c / 18c / 19c
- IBM DB2 10.5 / 11.1
  (excluding IBM z/OS for all versions)
- PostgreSQL 9.4 / 9.6 / 10.4 / 10.7 / 11.1 / 11.2
- Amazon Aurora PostgreSQL compatible with PostgreSQL 9.6 / 10.4 / 10.7
- Microsoft SQL Server 2012/2014/2016/2017
  (see Configuration Note)
- H2 1.4 (not recommended for production or Cluster Mode see Deployment Note)

Database Clustering and Replication

Clustered or replicated databases are supported while the communication between Camunda BPM and the database cluster matches the corresponding non-clustered / non-replicated configuration and 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.
See Details.

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, e.g. 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.

To learn more, please visit our **Best Practices** resource center.

Learn more