CTEngine Architecture

Technical overview

VERSION 1.2 (2020.11.03)

For public use

Table of Contents

1 Introduction

CTEngine (Computer Telephony Engine) is software based telecommunication platform. Its main purpose is to provide easy development of the telecom solutions and services.

CTEngine it is built as a stand-alone software, independent of the other systems. It is built as modular and service-oriented framework. Such architecture gives us possibility for further improvements and implementation of new features in the future. In the same time CTEngine acts as server software which is able to host and run telephony services. It is able to simultaneously process hundreds or even thousands of various telecommunication services related to voice and video phone calls.

CTEngine establishes communication with other telephony systems via standard telephony TDM (ISDN) and VoIP (SIP, IAX2) protocols. Currently this is achieved by interaction with Asterisk open source communication platform, but in the future other communication platforms could be integrated as well. With such architecture we were able to focus on creation of platform that will produce new value in the telecom world and let a low level telecommunication processing to reliable and trusted organizations.

Platform also provides real-time management of its components via web based user interface. Via UI user is able to upload newly created or update or replace existing services into running system without interruption of existing processes. UI also provides configuration, monitoring, statistics, debugging and other features.

2 Prerequisites

CTEngine is built with Java 8 programming language, but is also executable on newer JRE versions.

For its telecommunication abilities, CTEngine is using and controlling Asterisk open source software which is one of the leaders in the telecom world platforms and has huge community. This is achieved by connecting to Asterisk REST Interface (ARI) which is available since version 12. Asterisk versions before 12 can't be used with CTEngine. Recommended versions are the ones with a long time support (LTS) such as 13 or 16. Since version 13 has recently reached end-of-life for integration of new features and bug fixes, currently we recommend version 16. In the future there will be for sure new Asterisk releases and we are planning to keep CTEngine up to date and utilize those versions as well. To date, the last Asterisk version is 18.0.0, but it is still quite new and not recommended yet. To speed up bindings and communication with Asterisk, CTEngine platform is using a fork of ARI4Java open source library which is accessible on link https://github.com/modrljin/ari4java.

Since there are no other dependencies, CTEngine can be executed on any operating system (OS) such as Linux, Windows or Mac OS.

CTEngine and Asterisk can be installed on the same server physical machine or separately on its own because communication between them is made over TCP protocol. There is no preferably way if this should be done on one way or other and this depends on the power of the server hardware and expected call throughput. To achieve better performance, hardware with more CPU cores should be used. Both software are able to run on virtual machines as well.

3 System core

3.1 Overview

CTEngine is stand-alone java application which uses following 3rd party libraries:

- Common2 (maxcom Proprietary License)
- Ari4java (GNU GPLv3)
- Log4j (Apache License 2.0)
- Slf4j (MIT License)
- Jackson JSON (Apache License 2.0)
- JSON.org (Free)
- JavaMail API (CDDL v1)
- Netty (Apache License 2.0)

It consists of different modules which are working together to process different tasks. Several modules are exchanging data with external systems via provided application interfaces (API). System architecture is shown in Figure 1.

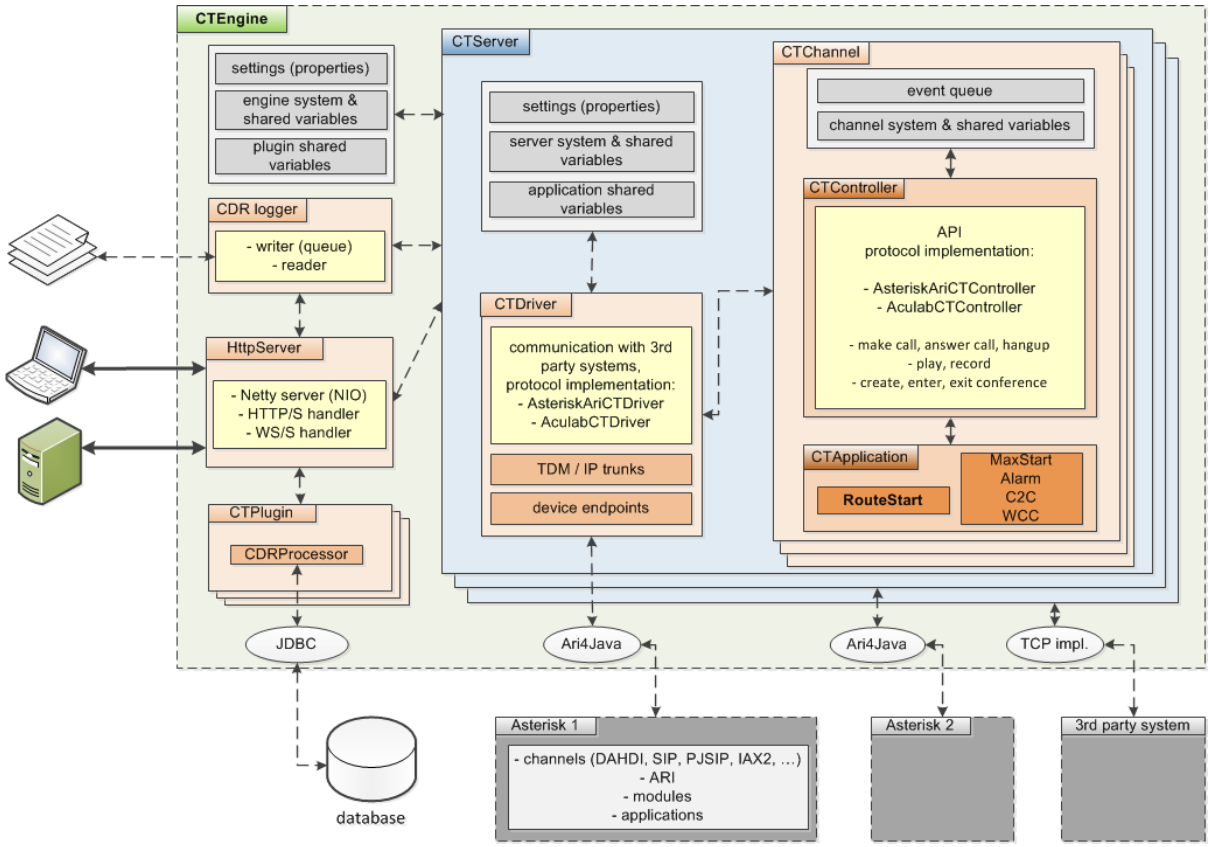

Figure 1

Figure 2. shows software stack architecture.

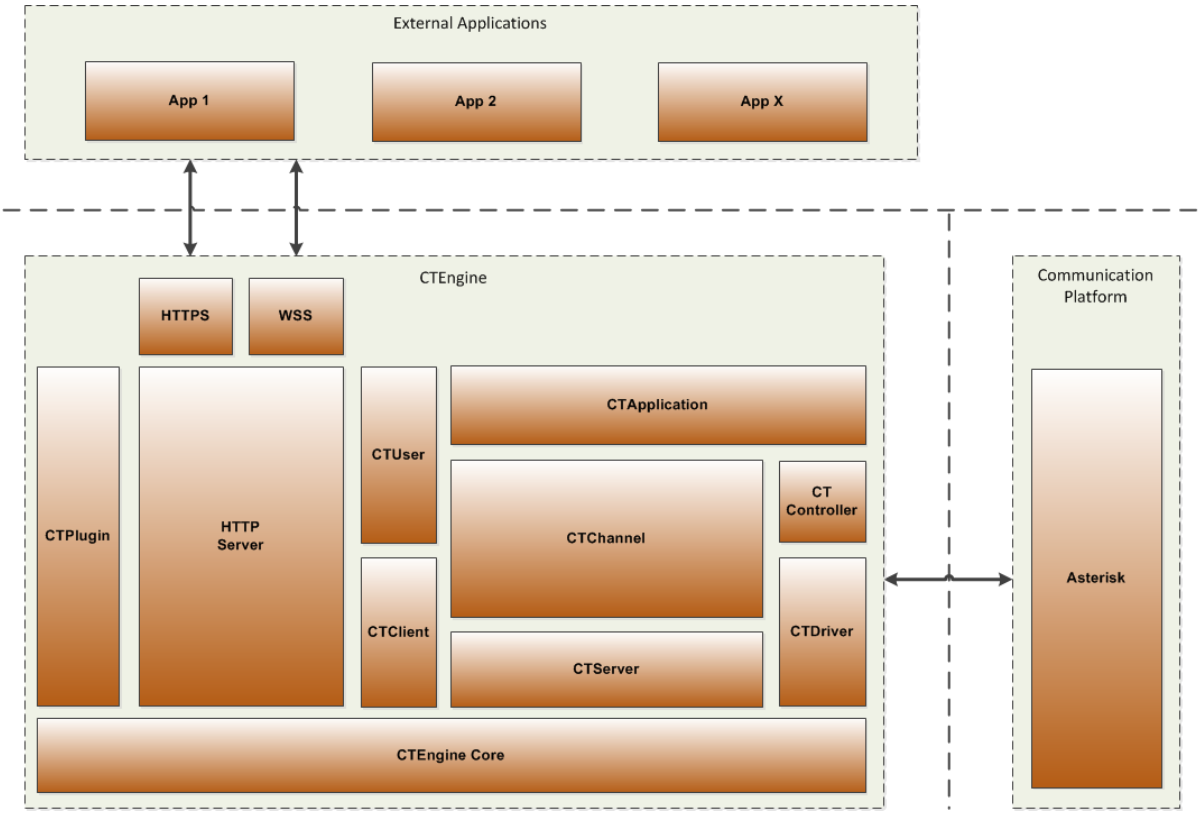

Figure 2.

3.2 CTEngine

Engine (CTEngine) is the main application container which instantiates other significant system components such as CTServer, HTTP server, CDR logger, etc. During initialization it reads main configuration file 'ct_engine.cfg'. This file contains main system settings such as:

- name
- main logger
- CDR logger
- configuration directory paths
- HTTP server settings
- mail server settings
- language settings
- native Java window settings (available only on Windows)

3.3 CTServer

Engine contains single or multiple server containers which are called CTServer. In most cases single server would be preferable especially in the environments where heavier concurrent calls processing is expected. One server contains single driver (CTDriver) instance which is used for communication with remote 3rd party telecommunication system. Server reads its configuration from file 'servers.cfg' which contains following settings:

- number (id)
- name
- auto start
- channels settings (count per type, distribution and reservation algorithms, TDM default groups, default start application)
- driver settings (driver class, auto start, specific driver settings – e.g. Asterisk ARI application name, connection username, password and version)

Server can be started or stopped at runtime via provided user interface.

3.3.1 CTTrunk

For telecommunication with other telephony systems via telecom protocols such as ISDN, SIP, IAX2, etc., server is using trunks (CTTrunk) which should be defined in both, CTServer and 3rd party systems, e.g. Asterisk.

3.3.2 CTChannel

Server contains multiple channels which are called CTChannel. They are counted starting with number 1 up to X. Channels can be started, stopped, reloaded, etc. via provided user interface at runtime. Each channel is running in its own thread. One channel is able to process single inbound or outbound call at the time. When there are no calls for processing, channel is in idle state and it is waiting for either inbound call or notification from other channel (internal) or other system (external). Internal notifications are implemented on channel level. They are used for communication between them. External notifications are HTTP or WS requests which are coming from external applications.

Channel provides a lot of methods, but the most important are:

- number
- type (TDM, IP, SW)
- id
- state (IDLE, BUSY)
- line state (UNAVAILABLE, IDLE, RINGING, DIALING, CONNECTED)
- current application (CTApplication)

Each channel has its own queue for processing following events:

- ring
- disconnect
- mode change
- conference entered
- conference exited
- conference destroyed
- route
- unroute
- notify
- http notify

Application running on a channel has possibility to enable, disable or discard certain events. If some event is enabled on a channel, application will be notified about it when it comes to the queue, but only after previous event is marked as processed.

3.4 CTDriver

CTDriver is a base module for communication with 3rd party telecom platforms (e.g. Asterisk). To date, we have implemented communication to Asterisk system via in AsteriskAriCTDriver which is an extension of CTDriver module.

Driver can be runtime started or stopped via provided user interface.

3.4.1 CTController

For telecommunication and call control, one channel is using single controller instance (CTController) which is implemented for specific driver, e.g. AsteriskAriCTController.

Generic and specifically implemented controller methods are:

- initialize
- reset
- wait call
- answer call
- make call
- hangup call
- ring/stop ring
- play/stop play
- play/stop play MOH (music on hold)
- play silence
- record
- send DTMF
- mute (unmute)
- hold (unhold)
- listen volume up/down
- talk volume up/down
- create/destroy conference
- enter/exit conference
- play on conference
- start/stop MOH on conference
- record conference
- etc…

3.5 CTApplication

CTApplication is independent basic abstract software component used for service creation on the platform. It provides various methods to its extended objects:

- initialize
- destroy
- start
- stop
- terminate
- kill
- change application
- delay
- all controller's methods
- etc.

To be able to process service logic, each application is running on a separate channel in its own thread. Applications are able to communicate with other internal system components such as engine, server, channel and other applications on other channels. Additionally, they are able to provide its own API for incoming HTTP and WS requests which are coming from other external systems, but they are also able to initiate communication with them by sending requests.

CTRouter is our first service built on the CTEngine platform. It is implemented in the application class RouteStart. More details could be found in chapter 4.

3.6 CTPlugin

Plugin is independent basic abstract software component that is able to run on the CTEngine platform outside of the channel containers. It can be added, started, stopped or updated in the platform at runtime. Plugins are usually used for processing tasks that are not strictly related to live call processing. For example, it can be used for importing CDRs into external database which provides advanced search and statistics.

3.7 HTTPServer

HTTP server is platform module used for communication with external systems or applications. It is able to process non-secure or secure connections over HTTP and WS protocol. Secure protocols are HTTPS and WSS. Module is responsible for handling and dispatching all incoming requests. Before request is distributed to other platform components, HTTP server will perform all necessary authentication and security checks and reject unauthorized ones.

Any component that wants to implement HTTP/WS APIs must register itself to HTTP server. Server will create default root URI path depending of the component type and other settings. Components that are able to register to HTTP server are following:

- CTEngine
- CTServer
- CTApplication
- CTPlugin

3.8 CTClient

Client is platform component used for defining service owners. Service owner is usually a legal person (company) which owns one or more services on the system. One client usually contains of one or more users (CTUser).

3.8.1 CTUser

User is a natural person or application interface that connects to CTEngine platform via user interface (UI). There are three user types:

- ADMIN
- AGENT
- INTERFACE

ADMIN is a human (natural person) that is able to use web based UI for CTEngine platform administration. AGENT is a human that is able to use web based UI for managing or monitoring some parts of the platform depending of assigned permissions. INTERFACE is technically non-human, application (bot) that is able to communicate with platform depending of assigned permissions. It can manage or just monitor some components (applications, plugins, statistics, etc.). For example, interface user is able to trigger any service or check for service status.

3.9 CDR logger

CDR logger module is responsible for logging (writing and saving) Call Detail Records in the platform at runtime into specific system files and serving them to different users based on their requests and access permissions. This module also summarizes CDRs into compact and grouped data on a regular basis interval, e.g. every 5 minutes. Module is saving CDRs into files on a daily basis rotation, while summarized data are saved into files with yearly rotation.

4 CTRouter

4.1 Overview

CTRouter is implemented and extended CTApplication. It provides advanced call routing mechanism by defining failover trunks and routing rules.

Application's architecture is shown in Figure 3.

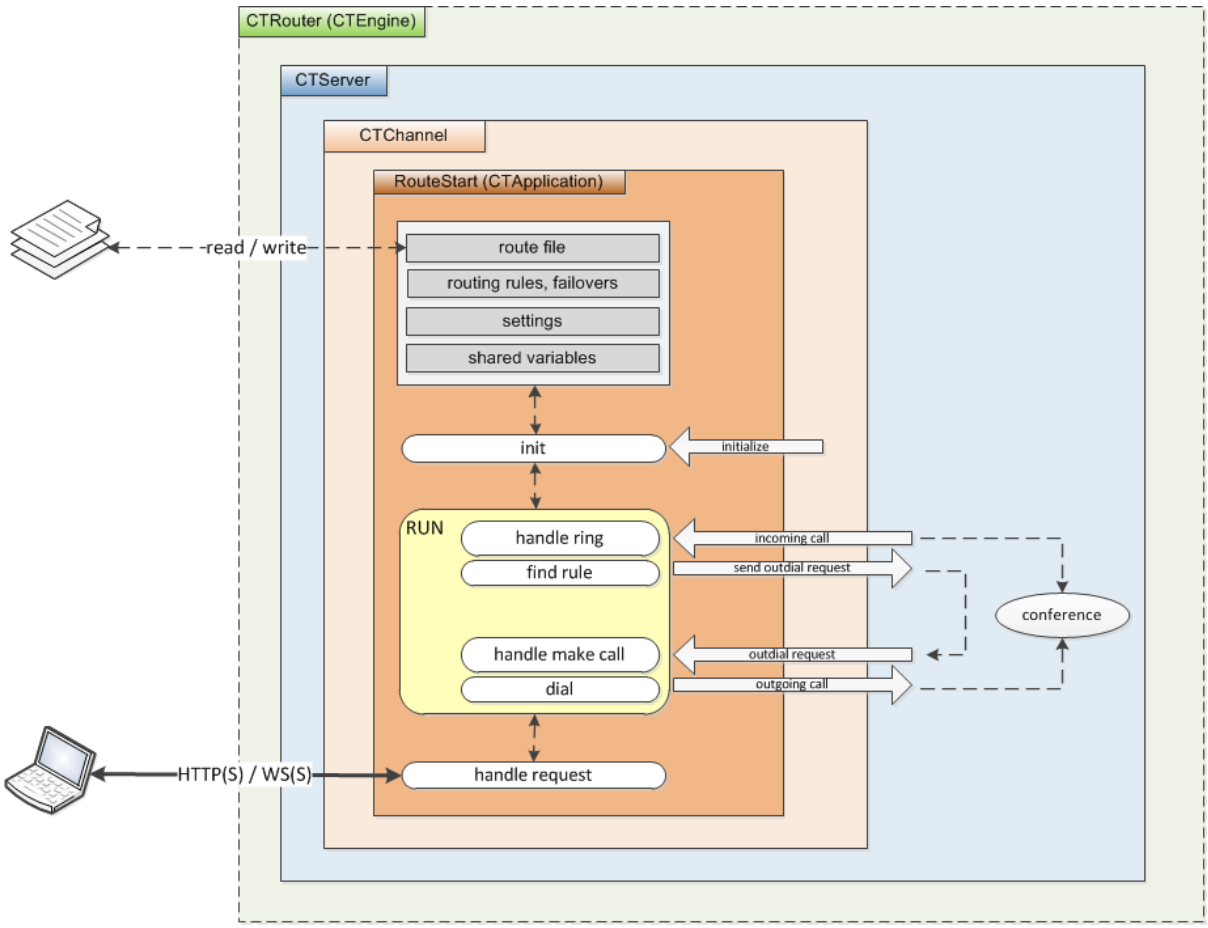

Figure 3.

CTRouter is implemented in java class RouteStart.

Application is performing service logic in the following order:

- wait for inbound call (IN)
- find routing rule
- if rule is not found, hangup IN call, reload and wait again

- if rule is found, initiate outbound call (OUT) on other free channel
- wait for OUT call progress
- if OUT call is not connected (busy, no answer, etc.), hangup IN call with the same status code
- if call is connected, create conference between IN and OUT call
- wait for any call leg to disconnect and hangup other call with the same status code

While processing OUT request application is preforming following operations:

- wait for outbound request for dial
- check if requested trunk is available, otherwise find available failover
- if there are no available trunks, send failure response to IN call
- start dialing
- wait for dial status
- send dial progress to IN channel

4.2 Configuration

Application reads and writes its configuration settings into file 'router.cfg'. The most important configuration settings are following:

- default forward caller subaddress, called subaddress and caller name
- default no answer timeout
- trunk failover settings
- routing rules

4.3 Failovers

Failovers is list of trunks and replacement trunks that will be used for routing when main trunk is not available (connected to remote system). It is used while performing dialing on OUT channel. If requested trunk and all failovers are unavailable, OUT call will fail and related IN call will hangup.

4.4 Routing rules

There could be one or more routing rules defined. When IN call comes into application, it will try to find first matching rule in the same order as they are defined in

configuration. Each rule can override default configuration settings – forward subaddress / name and no answer timeout. Routing rule settings are following:

- id
- name
- active
- IN caller number regex
- IN called number regex
- IN trunks
- OUT caller number regex substitution
- OUT called number regex substitution
- OUT trunk
- forward caller subaddress, called subaddress and caller name (not mandatory)
- no answer timeout (not mandatory)

If rule is inactive, it is considered as nonexistent and not considered while searching for matching rule. IN caller and called numbers are searched by regular expression syntax. OUT caller and called numbers could be forwarded as original IN values or modified to any other value. Examples:

1. OUT number is forwarded as original IN value:
   IN number: (.*)
   OUT number: $1

2. OUT number is modified to fix value '100':
   IN number: (.*)
   OUT number: 100

3. OUT number is modified by adding prefix '99':
   IN number: (.*)
   OUT number: 99$1

5 3rd party dependencies

5.1 Asterisk

For low level telephony processes, CTEngine is using 3rd party telecommunication platforms. Currently, only Asterisk is fully supported. How to install and configure Asterisk is out of the scope in this document and can be found in "CTEngine – installation" document. Recommended Asterisk version currently is 16.

Trunks that will be used in CTEngine have to be defined in Asterisk as well. CTEngine is just using them to achieve defined service processing logic.

5.2 Java libraries

There are several java libraries that must be included into java runtime path while starting CTEngine.

5.2.1 Common2

- common2-util.jar
- common2-net.jar

5.2.2 Ari4java

- ari4java-maxcom-0.12.1.jar

5.2.3 Log4j

- log4j-1.2.17.jar

5.2.4 Slf4j

- slf4j-api-1.7.30.jar
- slf4j-log4j12-1.7.30.jar

5.2.5 Jackson JSON

- jackson-core-2.9.6.jar
- jackson-annotations-2.9.6.jar
- jackson-databind-2.9.6.jar

5.2.6 JSON.org

- json-org-20180813.jar

5.2.7 JavaMail

- mailapi.jar
- smtp.jar
- pop3.jar
- imap.jar
- dsn.jar

5.2.8 Netty

- netty-all-4.1.50.Final.jar