cmake\_minimum\_required(VERSION 3.0.2 FATAL\_ERROR)

project(afina LANGUAGES C CXX)

set(CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS} -std=c++11")

set(CMAKE\_C\_FLAGS "${CMAKE\_C\_FLAGS} -std=c99 -Wall -Werror -fPIC")

set(CMAKE\_MODULE\_PATH ${CMAKE\_MODULE\_PATH} "${CMAKE\_SOURCE\_DIR}/cmake/Modules/")

set(CMAKE\_THREAD\_PREFER\_PTHREAD)

set(CMAKE\_THREAD\_LIBS\_INIT "-lpthread")

set(CMAKE\_HAVE\_THREADS\_LIBRARY 1)

set(CMAKE\_USE\_WIN32\_THREADS\_INIT 0)

set(CMAKE\_USE\_PTHREADS\_INIT 1)

# If system has ccache - use it

find\_program(CCACHE\_FOUND ccache)

if(CCACHE\_FOUND)

set\_property(GLOBAL PROPERTY RULE\_LAUNCH\_COMPILE ccache)

set\_property(GLOBAL PROPERTY RULE\_LAUNCH\_LINK ccache)

endif(CCACHE\_FOUND)

# Use native optimizations, for example fast crc32

include(CheckCXXCompilerFlag)

CHECK\_CXX\_COMPILER\_FLAG("-march=native" COMPILER\_OPT\_ARCH\_NATIVE\_SUPPORTED)

if (COMPILER\_OPT\_ARCH\_NATIVE\_SUPPORTED)

set(CMAKE\_CXX\_FLAGS "${CMAKE\_CXX\_FLAGS} -march=native")

set(CMAKE\_C\_FLAGS "${CMAKE\_C\_FLAGS} -march=native")

endif()

##############################################################################

# Dependencies

##############################################################################

## GTest

set(BUILD\_GTEST ON CACHE BOOL "Builds the googletest subproject")

set(BUILD\_GMOCK ON CACHE BOOL "Builds the googlemock subproject")

set(gtest\_disable\_pthreads ON CACHE BOOL "Disable uses of pthreads in gtest.")

add\_subdirectory(third-party/googletest-release-1.8.0)

## Logging

add\_subdirectory(third-party/spdlog-0.13.0)

## Stack traces

add\_subdirectory(third-party/backward-cpp)

## Command line arguments parsing

set(CXXOPTS\_BUILD\_TESTS OFF CACHE BOOL "Set to ON to build examples")

set(CXXOPTS\_BUILD\_EXAMPLES OFF CACHE BOOL "Set to ON to build examples")

add\_subdirectory(third-party/cxxopts-1.4.3)

##############################################################################

# Setup build system

##############################################################################

# Generate version information

IF (NOT AFINA\_VERSION)

include(GetGitRevisionDescription)

git\_local\_changes(AFINA\_VERSION\_DIRTY)

git\_describe(AFINA\_VERSION --long --tags --match "v[0-9]\*" --abbrev=4)

ENDIF()

# major.minor.patch part of version

MESSAGE(STATUS "Version: " ${AFINA\_VERSION} )

if ("${AFINA\_VERSION}" MATCHES "^v[0-9]+\\.[0-9]+\\.[0-9]+")

string(REGEX REPLACE "^v([0-9]+)\\..\*" "\\1" VERSION\_MAJOR "${AFINA\_VERSION}")

string(REGEX REPLACE "^v[0-9]+\\.([0-9]+)\\..\*" "\\1" VERSION\_MINOR "${AFINA\_VERSION}")

string(REGEX REPLACE "^v[0-9]+\\.[0-9]+\\.([0-9]+).\*" "\\1" VERSION\_PATCH "${AFINA\_VERSION}")

else()

message(FATAL\_ERROR "Version has wrong format, check major/minor/patch parts of version")

endif()

# additional git part of version

if ("${AFINA\_VERSION}" MATCHES "^v[0-9]+\\.[0-9]+\\.[0-9]+-[0-9]+-[a-zA-Z0-9]+")

string(REGEX REPLACE "^v[0-9]+\\.[0-9]+\\.[0-9]+-([0-9]+).\*" "\\1" VERSION\_COMMITS "${AFINA\_VERSION}")

string(REGEX REPLACE "^v[0-9]+\\.[0-9]+\\.[0-9]+-[0-9]+-(.\*)" "\\1" VERSION\_SHA1 "${AFINA\_VERSION}")

else()

message(WARNING "Version has no full git info")

set(VERSION\_COMMITS "")

set(VERSION\_SHA1 "")

endif()

MESSAGE( STATUS "VERSION\_MAJOR: " ${VERSION\_MAJOR} )

MESSAGE( STATUS "VERSION\_MINOR: " ${VERSION\_MINOR} )

MESSAGE( STATUS "VERSION\_PATCH: " ${VERSION\_PATCH} )

MESSAGE( STATUS "VERSION\_COMMITS: " ${VERSION\_COMMITS} )

MESSAGE( STATUS "VERSION\_SHA1: " ${VERSION\_SHA1} )

MESSAGE( STATUS "VERSION\_DIRTY: " ${AFINA\_VERSION\_DIRTY} )

##############################################################################

# Sources

##############################################################################

include(ECMEnableSanitizers)

## Build services

add\_subdirectory(src)

## Build tests

enable\_testing()

add\_subdirectory(test)