Build Guide

MOJA BASE LIBRARIES

Version 1.0 ● 30 March 2020

www.moja.global

Contents

Introduction

Moja relies on a number of third-party, open source libraries to achieve its mission. Like any software project, it utilises these libraries during development, build and run time. Some of these libraries are obtained as precompiled binaries while others are obtained as source archives that need compiling.

This wouldn't be so complicated except for the fact that Moja implementations run both locally and within Docker environments. Thus there is a need, not only to prepare and serve these libraries in the correct formats but to serve them correctly within the needed environments.

This document provides a step by step guide for building the Moja Base Libraries for both the Local and Docker Based environments.

Requirements

This chapter highlights the hardware and software requirements for building Moja Base Libraries:

Hardware Requirements

a)	Recommended Hardware Specifications	
01.	Processor	Intel Core i7, Minimum, with Virtualization Support
02.	RAM	16GB, Minimum
03.	Hard Drive	1TB, Minimum

Refer to the following guides to audit how your hardware stacks against these recommendations:

- Annex A1 : Check Processor Capacity
- Annex A2 : Check RAM Capacity
- Annex A3 : Check Hard Drive Capacity
- Annex A4 : Check Support for Virtualization

Software Requirements

a)	Recommended Operating Systems	
01.	Windows 10 Pro	Latest Version
02.	Windows 10 Enterprise	Latest Version

Refer to the following guides to audit how your Operating System stacks against these recommendations:

- Annex C1 : Check Windows Version Edition
- Annex C2 : Check Windows Version Build Number

Please note that the information on the latest build of Windows 10 can be obtained from this page:

- https://support.microsoft.com/en-us/help/4464619/windows-10-update-history

b) Required Tools

01.	CMake	Latest Version
02.	Docker	Latest Version
03.	GIT	Latest Version
04.	Notepad++	Latest Version

Environment Preparation

This chapter highlights how to prepare the environment for building Moja Base Libraries.

Hardware Preparation

a)	**Firmware Virtualization**
01.	Check if the Firmware Virtualization is enabled
02.	Enable it if not

The following guides have been added as references for carrying out the above task:

- Annex A5 : Check Firmware Virtualization Enablement Status
- Annex B1 : Enable Firmware Virtualization

Operating System Preparation

a)	Windows Version
01.	Check if the Windows version is the latest
02.	Update it if not

The following guides have been added as references for carrying out the above task:

- Annex C1 : Check Windows Version Edition
- Annex C2 : Check Windows Version Build Number
- Annex D1 : Update to the latest version of Windows 10

Please note that the information on the latest build of Windows 10 can be obtained from this page:

- https://support.microsoft.com/en-us/help/4464619/windows-10-update-history

b)	Administrative Rights
01.	Check if the logged in user account has administrative rights
02.	Switch to an account that has administrative rights if not

The following guide has been added as a reference for carrying out the above task:

- Annex C4 : Check whether a user account has administrative privileges

c)	Account Password
01.	Check if the logged in user account has a password
02.	Set it if not

The following guides have been added as a references for carrying out the above task:

- Annex C5 : Check whether a user account has a password
- Annex D2: Add a password to a user account

d)	Windows Hyper-V Features
01.	Check whether Windows Hyper-V Features have been turned on
02.	Turn them on if not

The following guides have been added as a references for carrying out the above task:

- Annex C6 : Check whether Windows Hyper-V Features have been turned on
- Annex D3 : Turn on Windows Hyper-V Features

e)	**Port 445**
01.	Check whether Port 445 is open for TCP connections
02.	Open it if not

The following guides have been added as a references for carrying out the above task:

- Annex C7 : Check if port 445 is open for TCP connections
- Annex D4 : Open port 445 for TCP connections

Tools Installation

CMake

a)	**Pre Installation**
01.	Go to https://cmake.org/download/
02.	Download the latest CMake Binary Distribution for Windows

b)	**Steps**
⚠	The following installation steps were written with reference to CMake 3.17.0
01.	Right click the CMake installer and select **Install**
02.	Click **Next** to confirm that you want to proceed with the installation
03.	Acknowledge the license terms and Click **Next**
04.	Optionally select the second option to the add **CMake to the system PATH for all users**
05.	Optionally check the last option to create a **CMake Desktop Icon**
06.	Click **Next** to proceed
07.	Leave the install path unchanged to install CMake in the default location
08.	Click **Next** to proceed
09.	Click **Install** to begin the installation
10.	Click **Finish** to exit the installation

Docker

a)	**Pre Installation**
01.	Complete the hardware preparation instructions as described earlier
02.	Complete the operating system preparation instructions as described earlier
03.	Go to https://www.docker.com/products/docker-desktop
04.	Download the Docker Desktop installer for Windows [1]

b)	**Steps**
⚠	The following installation steps were written with reference to Docker Desktop 2.2.0.3
01.	Right click the Docker Desktop installer and select **Run as administrator**
02.	Wait for Docker Desktop to download the required packages
03.	Leave the first checkbox checked to add a Docker Desktop shortcut to your desktop
04.	Click **OK** to proceed
05.	Wait for Docker Desktop to unpack its files and install
06.	Click **Close** to exit the installation

[1] This step will ask you to sign-in into your **docker hub** account. If you don't have one, please sign-up up for free here: https://hub.docker.com/signup

c)	**Configuration**
⚠	The following configuration steps were written with reference to Docker Desktop 2.2.0.3
01.	Double click the **Docker Desktop Shortcut** to start it
02.	Wait for Docker Desktop to notify you that its up and running and then proceed to the next step
03.	Go to the System Tray [2] and click the Docker Desktop icon [3]
04.	Select **Settings** on the pop-up menu
05.	Select the **Resources** menu on the Settings Window
06.	Click the **Advanced** Resource subcategory if not currently selected
07.	Increase the **Memory** available to Docker to at least 4GB
08.	Click the **File Sharing** Resource subcategory
09.	Select drive **C:** as the local drive you want to be available to your containers
10.	Click **Apply & Restart** to save the changes
11.	Close the settings window after Docker Desktop successfully restarts

[2] The System Tray is another name given to the Notification Area found at the right-side of the Windows Taskbar.
[3] If you cannot see the Docker Desktop icon at first glance, try looking for it in the pop-up drawer of the System Tray

Git

a)	**Pre Installation**
01.	Go to https://git-scm.com/downloads
02.	Download the latest Git binary release for Windows

b)	**Steps**
⚠	The following installation steps were written with reference to Git 2.26.0
01.	Right click the Docker Desktop installer and select **Run as administrator**
02.	Click **Install** to acknowledge the license terms and carry out the installation

c)	**Configuration** [4]
⚠	The following configuration steps were written with reference to Git 2.26.0
01.	Open the Windows 10 search tool
02.	Search for **Windows PowerShell**, open it and use it to execute the commands that follow (i) git config --global user.name "<your_user_name>" [5] (set your global username) (ii) git config --global user.email "<your_email@someservice.com>" [6] (set your global email)

[4] These steps assume that you have a Github Account. If you don't, please sign up here: https://github.com/join
[5] Replace the content in the angular bracket, including the bracket itself, with your github username
[6] Replace the content in the angular bracket, including the bracket itself, with your github email

Notepad++

a)	**Pre Installation**
01.	Go to https://notepad-plus-plus.org/downloads/
02.	Download the latest Notepad++ release for Windows

b)	**Steps**
⚠	The following installation steps were written with reference to Notepad++ 7.8.5
01.	Right click the Notepad++ installer and select **Run as administrator**
02.	Leave **English** as the selected language and click **OK**
03.	Click **Next** to confirm that you want to proceed with the installation
04.	Acknowledge the license terms and Click **Next**
05.	Leave the install path unchanged to install Notepad++ in the default location; click **Next**
06.	Leave the selected features unchanged to install the default components; click **Next**
07.	Click **Install** to carry out the installation
08.	Click **Finish** to complete the installation

Instructions

1. Local Build Instructions

The third-party open source libraries that the Moja ecosystem depends upon are all required locally (i) for development purposes and (ii) for carrying out local builds and runs. This section provides two strategies for acquiring and building those libraries for the two listed purposes - one manual and the other based upon the vcpkg package manager.

1.1. Building Manually

The open source libraries that the Moja ecosystem depends upon can be acquired and conditionally built into the local environment one at a time. This section provides a step by step guide of how this can be done.

1.1.1. Building the Boost Library into the local environment

a)	**Get Archive**
01.	Go to https://sourceforge.net/projects/boost/files/boost/1.63.0/boost_1_63_0.zip
02.	Download the Boost archive

b)	**Extract Archive**
⚠	The following steps require that you have a folder name **Development** at the root of your **C:** drive
01.	Right click the Boost archive and select **Extract All**
02.	Set **C:\Development** as the target destination for the files extraction
03.	Leave the **Show extracted files when complete** option checked and click **Extract**
04.	Confirm that the library was extracted to **C:\Development\boost_1_63_0**

c)	**Build Library**
01.	Open the Windows 10 search tool
02.	Search for **Windows PowerShell** and open it
03.	Type cd $home and **Enter** [7]

[7] This will take you to your home directory if you are not already there

(c)	**Build Library - Continued**
04.	Type New-Item -Path 'user-config.jam' -ItemType File and **Enter** [8]
05.	Type Start-Process notepad++ user-config.jam and **Enter** [9]
06.	Paste the following contents into the **user-config.jam** file, replacing previous content if found [10] using msvc : 14.0 ; import toolset : using ; using python : 3.8 : C:\\Python38\\python.exe # cmd-or-prefix : C:\\Python38\\include : C:\\Python38\\libs : <address-model>64 : <define>BOOST_ALL_NO_LIB=1 ;
07.	Save the **user-config.jam** file and **close** Notepad++
08.	Go back to PowerShell
09.	Type cd C:\Development\boost_1_63_0 and **Enter** [11]
10.	Type .\bootstrap and **Enter** [12]

[8] This will create a new file named user-config.jam in the current directory if it does not already exist
[9] This will open the user-config.jam file in Notepad++ assuming you installed it as prescribed earlier
[10] This will configure Boost.Build for your toolsets and libraries. It assumes you installed Python 3.8.2 in C:\Python38
[11] This will change the working directory to the directory that the Boost library was extracted to
[12] This will build the Boost.Build tool (b2.exe)

(c) Build Library - Continued

11. Type and **Enter** the following command [13]

```
.\b2 install `
address-model=64 `
architecture=x86 `
link=static,shared `
threading=multi `
toolset=msvc-14.0 `
-j4 `
--prefix="C:\Development\boost" `
--libdir=C:\Development\boost\lib\boost-1_63\lib64-msvc-14.0
```

12. Close **PowerShell**

[13] This will build Boost. See https://boostorg.github.io/build/manual/develop/index.html#bbv2.overview.invocation

1.1.2. Building the Eigen Library into the local environment

a)	**Get Archive**
01.	Go to https://gitlab.com/libeigen/eigen/-/archive/3.3.3/eigen-3.3.3.zip/
02.	Download the Eigen archive

b)	**Extract Archive**
⚠	The following steps require that you have a folder name **Development** at the root of your **C:** drive
01.	Right click the Boost archive and select **Extract All**
02.	Set **C:\Development** as the target destination for the files extraction
03.	Leave the **Show extracted files when complete** option checked and click **Extract**
04.	Confirm that the library was extracted to **C:\Development\eigen-3.3.3**

1.1.3. Building the OpenSSL Library into the local environment

a)	**Get Installer**
01.	Go to https://slproweb.com/download/Win64OpenSSL-1_1_0L.exe/ [14]
02.	Download the OpenSSL installer

b)	**Install**
01.	Right click the OpenSSL installer and select **Run as administrator**
02.	Acknowledge the License Agreement and click **Next**
03.	Leave the install path unchanged to install OpenSSL in the default location; click **Next**
04.	Leave the Start Menu folder that OpenSSL's shortcut will be placed under unchanged; click **Next**
05.	Leave the Additional Tasks to do at default; click **Next**
06.	Click **Install** to continue with the installation
07.	Click **Finish** to wind-up the installation

[14] Use https://slproweb.com/download/Win32OpenSSL-1_1_0L.exe for a 32 bit installer version

1.1.4. Building the POCO Libraries into the local environment

a)	**Get Archive**
01.	Go to https://pocoproject.org/releases/poco-1.7.7/poco-1.7.7-all.zip
02.	Download the POCO archive

b)	**Extract Archive**
⚠	The following steps require that you have a folder name **Development** at the root of your **C:** drive
01.	Right click the POCO archive and select **Extract All**
02.	Set **C:\Development** as the target destination for the files extraction
03.	Leave the **Show extracted files when complete** option checked and click **Extract**
04.	Confirm that the library was extracted to **C:\Development\poco-1.7.7-all**

c)	**Build Library**
01.	Open the Windows 10 search tool
02.	Search for **Windows PowerShell** and open it
03.	Type cd C:\Development\poco-1.7.7-all and **Enter** [15]

[15] This will change the working directory to the directory that the POCO library was extracted to

c)	**Configuration - Continued**
04.	Type Start-Process notepad++ components and **Enter** [16]
05.	**Delete** the following lines from the **components** file CppUnit CppUnit/WinTestRunner Data/MySQL NetSSL_OpenSSL Crypto PageCompiler PageCompiler/File2PageA
07.	Save the **components** file and **close** Notepad++
08.	Go back to PowerShell
09.	Type .\buildwin 140 build all both x64 and **Enter** [17]

[16] This will open the components file in Notepad++ assuming you installed it as prescribed earlier
[17] This will build the 64 bit version of the POCO library; Type .\buildwin 140 build all both x32 for the 32 bit version

1.1.5. Building the SQLite Amalgamation Libraries into the local environment

a)	**Get Archive**
01.	Go to https://www.sqlite.org/2017/sqlite-amalgamation-3170000.zip/
02.	Download the SQLite Amalgamation archive

b)	**Extract Archive**
⚠	The following steps require that you have a folder name **Development** at the root of your **C:** drive
01.	Right click the SQLite Amalgamation archive and select **Extract All**
02.	Set **C:\Development** as the target destination for the files extraction
03.	Leave the **Show extracted files when complete** option checked and click **Extract**
04.	Confirm that the library was extracted to **C:\Development\sqlite-amalgamation-3170000**

1.1.6. Building the Turtle Libraries into the local environment

a)	**Get Archive**
01.	Go to http://downloads.sourceforge.net/project/turtle/turtle/1.3.0/turtle-1.3.0.zip/
02.	Download the Turtle archive

b)	**Extract Archive**
⚠	The following steps require that you have a folder name **Development** at the root of your **C:** drive
01.	Right click the Turtle archive and select **Extract All**
02.	Set **C:\Development** as the target destination for the files extraction
03.	Leave the **Show extracted files when complete** option checked and click **Extract**
04.	Confirm that the library was extracted to **C:\Development\turtle-1.3.0**

1.2. Building With Vcpkg

The open source libraries that the Moja ecosystem depends upon can be acquired and conditionally built into the local environment courtesy of **Moja's vcpkg.** Moja's vcpkg is a cross-platform open source package manager based-off Microsoft's vcpkg. To obtain it, one needs to clone and build a fork of Microsoft's vcpkg repository maintained by Moja Global. This section details how to do this followed by how to use it to install the base libraries.

a)	**Building the package manager**
01.	Open the Windows 10 search tool
02.	Search for **Windows PowerShell** and open it
03.	Type cd c\ and **Enter** [18]
04.	Type New-Item -path "C:\Development\moja-global\" -type directory and **Enter** [19]
05.	Type cd "C:\Development\moja-global\" and **Enter** [20]
06.	Type git clone https://github.com/moja-global/vcpkg and **Enter** [21]
07.	Type cd .\vcpkg\ and **Enter** [22]
08.	Type .\bootstrap-vcpkg and **Enter** [23]

[18] This will take you to the root of your c:\ drive if you are not already there
[19] This will create the folder tree "C:\Development\moja-global\" in one line
[20] This will change the working directory to "C:\Development\moja-global\"
[21] This will clone Moja's Fork of the vcpkg repository into the current directory
[22] This will change the working directory to "C:\Development\moja-global\vcpkg"
[23] This will create the vcpkg installer

b)	**Using the package manager**
01.	Open the Windows 10 search tool
02.	Search for **Windows PowerShell** and open it
03.	Type cd “C:\Development\moja-global\vcpkg" and **Enter** [24]
04.	Type the following command and **Enter** [25] .\vcpkg install ` boost-test:x64-windows ` boost-program-options:x64-windows ` boost-log:x64-windows ` turtle:x64-windows ` zipper:x64-windows ` poco:x64-windows ` libpq:x64-windows ` gdal:x64-windows ` sqlite3:x64-windows ` boost-ublas:x64-windows

[24] This will change the working directory to "C:\Development\moja-global\vcpkg"
[25] This will use vcpkg to build the required libraries

2. Docker Build Instructions

The third-party, open source libraries that the FLINT ecosystem relies on can all be lumped together and conveniently distributed as a single Docker image. Moja refers to such an image as the Moja Base Image. Subsequent images that depend upon these libraries, including the Moja FLINT Library Image, typically extend this base image obtaining and utilising the libraries in return.

This section provides a step by step guide on the preparation and building of the Moja Base Image.

2.1. Starting Off

2.1.1. Specify the Parent Image from which the image should be built:

```
FROM ubuntu:bionic
```

- Ubuntu 18.04, code-named "Bionic Beaver", is a good place to start.

2.1.2. Add a little metadata to describe the image:

```
LABEL project="FLINT Examples"\
      image="FLINT Docker Parent Image"\
      version="1.0"\
      maintainer="Moja Global <info@moja.global>"
```

- It's considered good practice to add a little description about our images so that users can learn more about them should they choose to run the "docker inspect" command. This is typically done through the use of Docker label commands.

2.1.3. Set the image's frontend to noninteractive:

```
ARG DEBIAN_FRONTEND=noninteractive
```

- Ubuntu has several interfaces that can be swapped at will. One of these interfaces: The noninteractive frontend, is considered an anti-frontend. It never interacts with its users at all. Instead, it chooses default answers for all of the questions asked. This makes it an ideal candidate for automatic installs.

2.1.4. Specify the number of CPUs that can be comfortably allocated for the build process:

```
ARG NUM_CPU=1
```

- This specification will come in handy when controlling the number of jobs that can be run concurrently via “make commands”.

2.1.5. Specify the versions of the core dependencies to be built into the image:

```
ARG BOOST_VERSION=1_69_0
ARG BOOST_VERSION_DOT=1.69.0
ARG CMAKE_VERSION=3.14.3
ARG FMT_VERSION=5.3.0
ARG GDAL_VERSION=2.4.0
ARG POCO_VERSION=1.9.0
ARG RABBITMQ_C_VERSION=0.9.0
ARG SIMPLE_AMQP_CLIENT_VERSION=2.4.0
ARG SQLITE_VERSION=3270200
ARG TURTLE_VERSION=1.3.0
```

- Exercise due diligence when using different versions of the core dependencies other than the ones specified above. At a minimum, only introduce new libraries when you are absolutely sure that they will not affect the integrity of the downstream build processes.

- **Appendix 2 : Core Dependencies** provides a brief description of these libraries.

2.1.6. Provide a link to the archives of each of the libraries above:

```
ARG BOOST_ARCHIVE=https://dl.bintray.com/boostorg/release/\
${BOOST_VERSION_DOT}/source/boost_${BOOST_VERSION}.tar.bz2

ARG CMAKE_ARCHIVE=https://github.com/Kitware/CMake/releases/download/\
v${CMAKE_VERSION}/cmake-${CMAKE_VERSION}.tar.gz

ARG FMT_ARCHIVE=https://github.com/fmtlib/fmt/archive/\
${FMT_VERSION}.tar.gz

ARG GDAL_ARCHIVE=https://download.osgeo.org/gdal/\
${GDAL_VERSION}/gdal-${GDAL_VERSION}.tar.gz

ARG POCO_ARCHIVE=https://pocoproject.org/releases/\
poco-${POCO_VERSION}/poco-${POCO_VERSION}.tar.gz

ARG RABBITMQ_C_ARCHIVE=https://github.com/alanxz/\
rabbitmq-c/archive/v${RABBITMQ_C_VERSION}.tar.gz

ARG SIMPLE_AMQP_CLIENT_ARCHIVE=https://github.com/alanxz/\
SimpleAmqpClient/archive/v${SIMPLE_AMQP_CLIENT_VERSION}.tar.gz

ARG SQLITE_ARCHIVE=https://www.sqlite.org/2019/\
sqlite-autoconf-${SQLITE_VERSION}.tar.gz

ARG TURTLE_ARCHIVE=https://sourceforge.net/projects/turtle/files/turtle/\
${TURTLE_VERSION}/turtle-${TURTLE_VERSION}.tar.gz

ARG ZIPPER_ARCHIVE=https://github.com/sebastiandev/\
zipper.git
```

- These specifications will come in handy when downloading the core resources in order to build and install them.

2.1.7. Specify the root directory that should be used for the installations:

```
ENV ROOTDIR /usr/local/
```

- The word root directory, as used here, refers to the topmost directory, within the series of directories, within which the installations should be performed. Since /usr/local is the recommended top-most directory for all system administrator installations, it's hereby used as the root directory.

2.1.8. Set the working directory to a src folder in the specified root directory:

```
WORKDIR $ROOTDIR/src
```

- This will see to it that the subsequent RUN commands are executed from this directory.

2.2. Adding Basic Dependencies

2.2.1. Update the package cache and install the basic dependencies:

```
RUN apt-get update \
 && apt-get install -y \
     bash-completion \
     build-essential \
     git \
     gdb \
     graphviz \
     libcurl4-gnutls-dev \
     libeigen3-dev \
     libgeos-dev \
     libhdf4-alt-dev \
     libhdf5-serial-dev \
     libnetcdf-dev \
     libpoppler-dev \
     libpq-dev \
     libproj-dev \
     libspatialite-dev \
     libssl-dev \
     libxml2-dev \
     nasm \
     openssl \
     postgis \
     postgresql-client-10 \
     python3-dev \
     python3-numpy \
     python3-pip \
     software-properties-common \
     sqlite3 \
     wget \
 && apt-get -y autoremove \
 && apt-get clean
```

- Please take a note of the last two commands: "apt-get autoremove" removes packages that are no longer needed while "apt-get clean" removes all packages kept in the apt cache, regardless of their age or need. Together, they prevent the image from swelling up in size, possibly by several hundred MBs, which is undesirable.

- **Appendix 1** : **Basic Dependencies** provides a brief description of all these libraries.

2.2.2. Use the installed python3-pip library to install setuptools:

```
RUN pip3 install setuptools
```

- Setuptools is a package development process library designed to facilitate packaging Python projects by enhancing the Python standard library distutils (distribution utilities)

2.2.3. Set the version of the installed Python in the user-config.jam file:

```
RUN echo "using python : 3.6 : /usr ;" > ~/user-config.jam
```

- When using the Boost library, a file called user-config.jam is typically placed in a user's home directory to tell Boost the tools and libraries available to the build system.

- Setting this piece of information guides Boost Build on how to invoke Python, #include its headers, and link with its libraries.

2.2.4. Create a symlink to link the Eigen 3 header files:

```
RUN ln -sf /usr/include/eigen3 /usr/local/include/eigen
```

- After a successful installation, Eigen 3 header files typically go to a different subdirectory: /usr/include/eigen3. Because of this, build scripts are forced to add an extra flag that points to these files (-I/usr/include/eigen3) everytime they need to compile a file that uses the library. To avoid this, a link should be made, inside of /usr/local/include to /usr/include/eigen3.

2.3. Adding Core Dependencies

2.3.1. Download the CMAKE archive, extract it, build it, install it and then clean up.

```
ADD ${CMAKE_ARCHIVE} $ROOTDIR/src/
WORKDIR $ROOTDIR/src/
RUN cd $ROOTDIR/src \
 && tar -xzf cmake-${CMAKE_VERSION}.tar.gz \
 && cd cmake-${CMAKE_VERSION} \
 && ./bootstrap \
 && make -s -j $NUM_CPU \
 && make install \
 && make clean \
 && cd $ROOTDIR \
 && rm -Rf src/cmake*
```

2.3.2. Download the POCO archive, extract it, build it, install it and then clean up.

```
ADD ${POCO_ARCHIVE} $ROOTDIR/src/
WORKDIR $ROOTDIR/src/
RUN cd $ROOTDIR/src \
 && tar -xzf poco-${POCO_VERSION}.tar.gz \
 && cd poco-${POCO_VERSION} \
 && ./configure \
 --omit=Data/ODBC,Data/MySQL,FSM,Redis \
 --no-samples \
 --no-tests \
 && make -s -j $NUM_CPU DEFAULT_TARGET=shared_release \
 && make install \
 && make clean \
 && cd $ROOTDIR \
 && rm -Rf src/poco*
```

2.3.3. Download the BOOST archive, extract it, build it, install it and then clean up.

```
ADD ${BOOST_ARCHIVE} $ROOTDIR/src/
WORKDIR $ROOTDIR/src/
RUN cd $ROOTDIR/src \
 && tar --bzip2 -xf boost_${BOOST_VERSION}.tar.bz2 \
 && cd boost_${BOOST_VERSION} \
 && ./bootstrap.sh --prefix=/usr/local \
 && ./b2 -j $NUM_CPU cxxstd=14 install \
 && ./b2 clean \
 && cd $ROOTDIR \
 && rm -Rf src/boost*
```

2.3.4. Download the FMT archive, extract it, build it, install it and then clean up.

```
ADD ${FMT_ARCHIVE} $ROOTDIR/src/
WORKDIR $ROOTDIR/src/
RUN cd $ROOTDIR/src \
 &&  mkdir libfmt-${FMT_VERSION} \
 && tar -xzf ${FMT_VERSION}.tar.gz \
 -C libfmt-${FMT_VERSION} --strip-components=1 \
 && cd libfmt-${FMT_VERSION} \
 && cmake -G"Unix Makefiles" -DCMAKE_INSTALL_PREFIX=/usr/local . \
 && make -j $NUM_CPU install \
 && make clean \
 && cd $ROOTDIR \
 && rm -Rf src/libfmt*
```

2.3.5. Download the SQLITE archive, extract it, build it, install it and then clean up.

```
ADD ${SQLITE_ARCHIVE} $ROOTDIR/src/
WORKDIR $ROOTDIR/src/
RUN cd $ROOTDIR/src \
 && tar -xzf sqlite-autoconf-${SQLITE_VERSION}.tar.gz \
 -C /usr/local/ \
 && cp /usr/local/sqlite-autoconf-${SQLITE_VERSION}/sqlite3.c \
   /usr/include/ \
 && cd $ROOTDIR \
 && rm -Rf src/sqlite*
```

2.3.6. Download the GDAL archive, extract it, build it, install it and then clean up.

```
ADD ${GDAL_ARCHIVE} $ROOTDIR/src/
WORKDIR $ROOTDIR/src/
RUN cd $ROOTDIR/src \
 && tar -xvf gdal-${GDAL_VERSION}.tar.gz \
 && cd gdal-${GDAL_VERSION} \
 && ./configure \
  --with-python --with-spatialite --with-pg --with-curl \
  --with-netcdf --with-hdf5=/usr/lib/x86_64-linux-gnu/hdf5/serial \
  --with-curl \
 && make -j $NUM_CPU \
 && make install \
 && ldconfig \
 && apt-get update -y \
 && apt-get remove -y --purge build-essential \
 && cd $ROOTDIR/src/gdal-${GDAL_VERSION}/swig/python \
 && python3 setup.py build \
 && python3 setup.py install \
 && cd $ROOTDIR \
 && rm -Rf src/gdal*
```

2.3.7. Download the ZIPPER archive, extract it, build it, install it and then clean up.

```
ADD ${ZIPPER_ARCHIVE} $ROOTDIR/src/
WORKDIR $ROOTDIR/src/zipper/build
RUN cmake .. \
 && make -s -j $NUM_CPU \
 && make install \
 && make clean \
 && cd $ROOTDIR \
 && rm -Rf src/zipper*
```

2.3.8. Download the TURTLE archive, extract it, build it, install it and then clean up.

```
WORKDIR $ROOTDIR/src/
RUN wget ${TURTLE_ARCHIVE} \
 && tar xzf turtle-1.3.1.tar.gz -C /usr/local/ \
 && cd $ROOTDIR \
 && rm -Rf src/turtle*
```

2.3.9. Download the RABBITMQ C archive, extract it, build it, install it and then clean up.

```
ADD ${RABBITMQ_C_ARCHIVE} $ROOTDIR/src/
WORKDIR $ROOTDIR/src/
RUN cd $ROOTDIR/src \
 && tar xzf v${RABBITMQ_C_VERSION}.tar.gz \
 && mkdir -p rabbitmq-c-${RABBITMQ_C_VERSION}/build \
 && cd rabbitmq-c-${RABBITMQ_C_VERSION}/build \
 && cmake -DCMAKE_BUILD_TYPE=Release \
 -DCMAKE_INSTALL_PREFIX=$ROOTDIR .. \
 && make --quiet -j $NUM_CPU \
 && make --quiet install \
 && make clean \
 && cd $ROOTDIR \
 && rm -Rf src/rabbitmq*
```

3.10. Download the SIMPLE AMQP CLIENT archive, extract it, build it, install it and then clean up.

```
ADD ${SIMPLE_AMQP_CLIENT_ARCHIVE} $ROOTDIR/src/
WORKDIR $ROOTDIR/src/
RUN cd $ROOTDIR/src \
 && tar xzf v${SIMPLE_AMQP_CLIENT_VERSION}.tar.gz \
 && mkdir -p SimpleAmqpClient-${SIMPLE_AMQP_CLIENT_VERSION}/build \
 && cd SimpleAmqpClient-${SIMPLE_AMQP_CLIENT_VERSION}/build \
 && cmake -DCMAKE_BUILD_TYPE=RELEASE \
  -DBoost_USE_STATIC_LIBS=OFF \
  -DBUILD_SHARED_LIBS=ON \
  -DCMAKE_INSTALL_PREFIX=$ROOTDIR .. \
 && make --quiet -j $NUM_CPU \
 && make --quiet install \
 && make clean \
 && cd $ROOTDIR \
 && rm -Rf src/SimpleAmqpClient*
```

2.4. Saving the Image

Save the image as "`Dockerfile.baseimage.bionic`".

> - Please don't include the quotes in the image name.

2.5. Building the image

2.5.1. Change the working directory to the directory with your Docker file.

> - If you've been following the instructions in the previous chapter: preparing image, then this is the directory with the file named "Dockerfile.baseimage.bionic".

2.5.2. Run the command below to build the image:

```
docker build -f Dockerfile.baseimage.bionic -t moja/baseimage:bionic .
```

- The -f option specifies the name of the docker file to be built - in this case, “Dockerfile.baseimage.bionic” .
- The -t option specifies the name that the built image should be tagged with - in this case “moja/baseimage:bionic”.
- The period, “.”, at the end of the command specifies the location of the Docker file to be built - in this case the current directory.

You can optionally update all the variables declared using the ARG directive at the build phase through the use of the --build-arg option. For example:

```
docker build -f Dockerfile.baseimage.bionic \
             --build-arg NUM_CPU=4 \
             -t moja/baseimage:bionic \ .
```

2.6. Conclusion

This guide illustrated how to prepare and build the Moja Base Image.

All the code associated with it is available at : https://github.com/moja-global/flint-examples.git.

This is a Docker-based project, so it should be easy to import and use as is

Annex A : Hardware Audits Reference

A1 : Check Processor Capacity

01.	Open the Windows 10 search tool
02.	Search for the **System Information** tool and open it
03.	Select **System Summary** menu on the **System Information** window
04.	Look for the **Processor** specification on the right pane

A2 : Check RAM Capacity

01.	Open the Windows 10 search tool
02.	Search for the **System Information** tool and open it
03.	Select the **System Summary** menu on the **System Information** window
04.	Look for the **Physical Memory** specifications on the right pane

A3 : Check Hard Drive Capacity

01.	Open the Windows 10 search tool
02.	Search for the **System Information** tool and open it
02.	Expand the **Components** category in the **System Information** window
03.	Expand the **Storage** subcategory under the **Components** category
04.	Click the Disks subcategory under the **Storage** subcategory
05.	Look for the **Size** specifications under the disk descriptions [26]

A4 : Check Support for Virtualization

01.	Open the Windows 10 search tool
02.	Search for the **Task Manager** tool and open it
03.	Open the **Performance** tab on the opened window [27]
04.	Look for a line that says **"Virtualization: (En/Dis)abled"** on the bottom-right side of the opened tab

[26] Watch out for multiple disk descriptions with different sizes when multiple Hard Drives are present
[27] You might have to click on **More details** to see this tab the very first time you open Task Manager

A5 : Check Firmware Virtualization Enablement Status

01.	Open the Windows 10 search tool
02.	Search for the **Task Manager** tool and open it
03.	Open the **Performance** tab on the opened window [28]
04.	Look for a line that says **“Virtualization: Enabled”** on the bottom-right side of the opened tab

[28] You might have to click on **More details** to see this tab the very first time you open Task Manager

Annex B : HW Configurations Reference

B1 : Enable Firmware Virtualization

01.	Restart the PC
02.	Press the key required to enter BIOs (See Appendix 3: Keys For Accessing BIOS settings)
03.	Navigate to either the **Advanced**, **Security** or the **Systems Configurations** tab
04.	Select **Virtualization** or **Virtualization Technology** and then press the **Enter** key [29]
05.	Select **Enabled** and then press the **Enter** key
06.	Press the **F10** key then select **Yes** and press the **Enter** key to save the changes and **Reboot** [30]

[29] On some Lenovo PCs, the Virtualization option will be found buried one level deeper under a **CPU Setup** option
[30] On some Sony PCs, you will need to navigate to a dedicated **Exit** tab to save changes and exit

Annex C : OS Audits Reference

C1 : Check the Windows Version Edition

01.	Open the Windows 10 search tool
02.	Search for the **System Information** tool and open it
03.	Select the **System Summary** menu on the **System Information** window
04.	Look for the **OS Name** specification on the right pane

C2 : Check the Windows Version Build Number

01.	Open the Windows 10 search tool
02.	Search for the **System Information** tool and open it
03.	Select the **System Summary** menu on the **System Information** window
04.	Look for the **Version** specification on the right pane

C3 : Check for the latest Windows Operating System

01.	Open https://en.wikipedia.org/wiki/List_of_Microsoft_Windows_versions
02.	Look for the latest Windows **Version, Edition** and **Build Number**

C4 : Check whether a user account has administrative privileges

01.	Open the Windows 10 search tool
02.	Search for the **Manage your account** tool and open it
03.	Look for the word "**Administrator**" underneath the **account** name

C5 : Check whether a user account has a password

01.	Open the Windows 10 search tool
02.	Search for the **Manage your account** tool and open it
03.	Click the **Sign-in options** on the left pane of the opened window
04.	Scroll down to the **Password** section on the right pane of the opened window
05.	Look for a statement that says "**Sign in with your account's password**" underneath it

C6 : Check whether Windows Hyper-V features are turned on

01.	Open the Windows 10 search tool
02.	Search for the **Turn Windows features on or off** tool and open it
03.	Locate the Hyper-V section and find out if it's checked

C7 : Check if port 445 is open for TCP connections

01.	Open the Windows 10 search tool
02.	Search for the **Windows Defender Firewall** tool and open it
03.	Click **Advanced settings** on the left pane of the **Windows Defender Firewall** window
04.	Click the **Inbound Rules** category on the left pane of the newly popped up window
05.	Locate the **Local Port** column on the newly opened **Inbound Rules** table
06.	Scroll down this **Local Port** column and see whether there's a TCP entry for port 445
07.	Click the **Outbound Rules** category on the left pane of the newly popped up window
08.	Locate the **Remote Port** column on the newly opened **Outbound Rules** table
09.	Scroll down this **Remote Port** column and see whether there's a TCP entry for port 445

Annex D : OS Configurations Reference

D1 : Update to the latest version of Windows 10

01.	Go to https://www.microsoft.com/en-us/software-download/windows10
02.	Click the **Update now** button to download the **Windows 10 Update Assistant**
03.	Right click the downloaded **Windows 10 Update Assistant** and select **Run as administrator**
04.	Click **Update Now** on the newly opened window
05.	Click **Next** after the PC is ascertained as being compatible with the update
06.	Click **Minimise** to optionally have the update run in the background
07.	Click **Restart now** to restart your PC when the update is complete

D2 : Add a password to a user account

01.	Open the Windows 10 search tool
02.	Search for the **Manage your account** tool and open it
03.	Click the **Sign-in options** on the left pane of the opened window
04.	Scroll down to the **Password** section on the right pane of the opened window
05.	Click the **Add** button underneath it
06.	Enter the password and password hint details and click **Next**
07.	Click **Finish**

D3 : Turn on Windows Hyper-V features

01.	Open the Windows 10 search tool
02.	Search for the **Turn Windows features on or off** tool and open it
03.	Locate the Hyper-V section
04.	Check it and click **OK**
05.	Click **Restart now** to finish installing the requested changes

D4 : Open port 445 for TCP connections

01.	Open the Windows 10 search tool
02.	Search for the **Windows Defender Firewall** tool and open it
03.	Click **Advanced settings** on the left pane of the **Windows Defender Firewall** window
04.	Click the **Inbound Rules** category on the leftmost pane of the newly popped up window
05.	Click the **New Rule** option on the rightmost pane of the newly popped up window
06.	Select **Port** as the type of rule to be created
07.	Click **Next**
08.	Select **TCP** as the protocol of the rule to created
09.	Select **Specific local ports** and enter **445** as the port that the rule should be apply to
10.	Click **Next**
11.	Select **Allow the connection** as the action to take when a connection matches the conditions
12.	Check **Domain, Private and Public** to have the rule apply to each of these profiles
13.	Click **Next**
14.	Enter **Docker** as the name of the rule and click **Finish**
15.	Repeat Steps 04 to 14 for **Outbound Rules**

Appendix 1 : Basic Dependencies

Dependency	About
bash-completion	Programmable completion for the bash shell. This package extends bash's standard completion behavior to achieve complex command lines with just a few keystrokes. It was conceived to produce programmable completion routines for the most common Linux/UNIX commands, reducing the amount of typing sysadmins and programmers need to do on a daily basis.
build-essential	Informational list of build-essential packages. This package contains an informational list of packages which are considered essential for building Debian packages. It also depends on the packages on that list, to make it easy to have the build-essential packages installed.
doxygen	Documentation generation tool. Doxygen is a documentation system for C, C++, Java, Objective-C, Python, IDL and to some extent PHP, C#, and D. It can generate an on-line class browser (in HTML) and/or an off-line reference manual (in LaTeX) from a set of documented source files.
doxygen-latex	Doxygen dependency package. Adds dependencies for all LaTeX packages required to build documents using the default stylesheet.
git	Fast, scalable, distributed version control system. This package provides the git main components with minimal dependencies.
gdb	GNU Debugger. GDB is a source-level debugger, capable of breaking programs at any specific line, displaying variable values, and determining where errors occurred. Currently, gdb supports C, C++, D, Objective-C, Fortran, Java, OpenCL C, Pascal, assembly, Modula-2, Go, and Ada. A must-have for any serious programmer.
graphviz	Open source graph visualization software. Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks. This package contains graph visualization command-line tools.

libcurl4-gnutls-dev	Development files and documentation for libcurl (GnuTLS flavour). libcurl is an easy-to-use client-side URL transfer library, supporting DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, POP3, POP3S, RTMP, RTSP, SCP, SFTP, SMTP, SMTPS, TELNET and TFTP. libcurl supports SSL certificates, HTTP POST, HTTP PUT, FTP uploading, HTTP form based upload, proxies, cookies, user+password authentication (Basic, Digest, NTLM, Negotiate, Kerberos), file transfer resume, http proxy tunneling and more.
libeigen3-dev	Lightweight C++ template library for linear algebra. Eigen 3 is a lightweight C++ template library for vector and matrix math, a.k.a. linear algebra. Unlike most other linear algebra libraries, Eigen 3 focuses on the simple mathematical needs of applications.
libgeos-dev	Geometry engine for GIS. GEOS provides a spatial object model and fundamental geometric functions. It implements the geometry model defined in the OpenGIS Consortium Simple Features Specification for SQL.
libhdf4-alt-dev	Hierarchical Data Format development files (without NetCDF). HDF is a multi-object file format for storing and transferring graphical and numerical data mainly used in scientific computing. HDF supports several different data models, including multidimensional arrays, raster images, and tables. Each defines a specific aggregate data type and provides an API for reading, writing, and organizing the data and metadata.
libhdf5-serial-dev	Packages providing libhdf5-serial-dev This is a virtual package.
libnetcdf-dev	Creation, access, and sharing of scientific data. NetCDF (network Common Data Form) is a set of interfaces for array-oriented data access and a freely distributed collection of data access libraries for C, Fortran, C++, Java, and other languages. The netCDF libraries support a machine-independent format for representing scientific data. Together, the interfaces, libraries, and format support the creation, access, and sharing of scientific data.
libpoppler-dev	PDF rendering library. Poppler is a PDF rendering library based on Xpdf PDF viewer.
libpq-dev	Header files for libpq5 (PostgreSQL library). Header files and static library for compiling C programs to link with the libpq library in order to communicate with a PostgreSQL database backend.
libproj-dev	Cartographic projection library.

	Proj and invproj perform respective forward and inverse transformation of cartographic data to or from Cartesian data with a wide range of selectable projection functions (over 100 projections).
libspatialite-dev	Geospatial extension for SQLite. The SpatiaLite extension enables SQLite to support spatial (geometry) data in a way conformant to OpenGis specifications, with both WKT and WKB formats.
libssl-dev	Secure Sockets Layer toolkit. This package is part of the OpenSSL project's implementation of the SSL and TLS cryptographic protocols for secure communication over the Internet. It contains development libraries, header files, and manpages for libssl and libcrypto.
libxml2-dev	Development files for the GNOME XML library. XML is a metalanguage to let you design your own markup language. A regular markup language defines a way to describe information in a certain class of documents (eg HTML). XML lets you define your own customized markup languages for many classes of documents. It can do this because it's written in SGML, the international standard metalanguage for markup languages.
nasm	General-purpose x86 assembler. Netwide Assembler: NASM will currently output flat-form binary files, a.out, COFF and ELF Unix object files, and Microsoft 16-bit DOS and Win32 object files.
openssl	Secure Sockets Layer toolkit - cryptographic utility. This package is part of the OpenSSL project's implementation of the SSL and TLS cryptographic protocols for secure communication over the Internet. It contains the general-purpose command line binary /usr/bin/openssl, useful for cryptographic operations.
postgis	Geographic objects support for PostgreSQL. PostGIS adds support for geographic objects to the PostgreSQL object-relational database. In effect, PostGIS "spatially enables" the PostgreSQL server, allowing it to be used as a backend spatial database for geographic information systems (GIS).
postgresql-client-10	Front-end programs for PostgreSQL This metapackage always depends on the currently supported database client package for PostgreSQL.
python3-dev	Header files and a static library for Python (default).

	Header files, a static library and development tools for building Python modules, extending the Python interpreter or embedding Python in applications.
python3-numpy	Fast array facility to the Python 3 language. Numpy contains a powerful N-dimensional array object, sophisticated (broadcasting) functions, tools for integrating C/C++ and Fortran code, and useful linear algebra, Fourier transform, and random number capabilities.
python3-pip	Python package installer. pip is the Python package installer. It integrates with virtualenv, doesn't do partial installs, can save package state for replaying, can install from non-egg sources, and can install from version control repositories.
software-properties-common	Manages the repositories that you install software from (common). This software provides an abstraction of the used apt repositories. It allows you to easily manage your distribution and independent software vendor software sources.
sqlite3	Command line interface for SQLite 3. SQLite is a C library that implements an SQL database engine. Programs that link with the SQLite library can have SQL database access without running a separate RDBMS process.
wget	Retrieves files from the web. Wget is a network utility to retrieve files from the web using HTTP(S) and FTP, the two most widely used internet protocols. It works non-interactively, so it will work in the background, after having logged off. The program supports recursive retrieval of web-authoring pages as well as FTP sites.

Appendix 2 : Core Dependencies

Dependency	About
boost	Free, peer-reviewed, portable C++ source libraries. boost libraries are a collection of C++ libraries that provide support for standard tasks and structures such as linear algebra, pseudorandom number generation, multithreading, image processing, regular expressions, and unit testing..
cmake	Build process manager. CMake is an open-source, cross-platform family of tools designed to build, test and package software.
fmt	A modern formatting library. {fmt} is an open-source formatting library for C++ that can be used as a safe and fast alternative to (s)printf and iostreams
gdal	Raster and Vector translation library. GDAL is a translator library for raster and vector geospatial data formats that is released under an X/MIT style Open Source License by the Open Source Geospatial Foundation.
poco	C++ libraries for building network- and internet-based applications. The POrtable COmponents (POCO) C++ Libraries are a set of cross-platform C++ libraries for developing computer network-centric, portable applications in C++.
rabbitmq-c	This is a C-language AMQP client library for use with v2.0+ of the RabbitMQ broker. RabbitMQ is an open-source message-broker software that originally implemented the Advanced Message Queuing Protocol (AMQP) and has since been extended with a plug-in architecture to support Streaming Text Oriented Messaging Protocol (STOMP), Message Queuing Telemetry Transport (MQTT), and other protocols.
SimpleAmqpClient	C++ wrapper around the rabbitmq-c C library SimpleAmqpClient is an easy-to-use C++ wrapper around the rabbitmq-c C library
sqlite	Database engine.

	SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine.
turtle	Mock object library. Turtle is a C++ mock object library based on Boost with a focus on usability, simplicity and flexibility.
zipper	C++ wrapper around minizip compression library. Zipper is a reliable, simple and flexible compression library that supports all kinds of inputs and outputs. Moreover it allows the compression of files into memory instead of being restricted to file compression only, and using data from memory instead of just files as well.

Appendix 3: Keys For Accessing BIOS settings

Manufacturer	F1	F2	F3	F6	F10	F11	F12	ESC	INS	DEL
Acer	A	**C**								**C**
Asus		**C**							A	A
DELL	A	**C**	A				A			A
HP	A	A		A	**C**	A	A	**C**		
Lenovo	**C**	**C**								
Sony	A	**C**	**C**							
Toshiba	A	C						A		

Where C = Most Common and A = Alternative

Abbreviations

Abbreviation	Meaning
CEIP	Customer Experience Improvement Program
CPU	Central Processing unit
FLINT	Full Lands Integration Tool
HW	Hardware
OS	Operating System
RAM	Random Access Memory
TCP	Transmission Control Protocol

References

Basic Dependencies:

Ubuntu Packages	https://packages.ubuntu.com/

Core Dependencies:

Boost C++ Libraries	https://www.boost.org/
CMake	https://cmake.org/
fmtlib/fmt	https://github.com/fmtlib/fmt
GDAL	https://gdal.org/
POCO	https://pocoproject.org/
RabbitMQ C	https://github.com/alanxz/rabbitmq-c
SimpleAmqpClient	https://github.com/alanxz/SimpleAmqpClient
SQLite	https://www.sqlite.org/index.html
Turtle	http://turtle.sourceforge.net/
Zipper	https://github.com/sebastiandev/zipper