Multi-Stage Builds

> Reference to the documentation

One might want to install software that requires external libraries that are not available with the distribution or to recompile existing with different options. Usually, this will require installing common building tools and compilers that are not needed for running the executables...

Similar to Docker multi-stage builds, Singularity also offers a multi-stage builds that allows for copying files between stages (Singularity can copy only from previous to current stage). Below is an example definition file that compiles the ARPIP (Ancestral sequence Reconstruction under the Poisson Indel Process) tool. The recipe is following the local installation for the static binary build.

```
Bootstrap: docker
From: ubuntu:20.04
Stage: devel

%post
  export LC_ALL=C
  export DEBIAN_FRONTEND=noninteractive

  # Package cache in /tmp
  mkdir -p /tmp/apt20 &&  echo "Dir::Cache "/tmp/apt20";" >
/etc/apt/apt.conf.d/singularity-cache.conf

  apt-get update && apt-get -y dist-upgrade && \
  apt-get install -y wget git cmake build-essential zlib1g-dev

  # Download
  export TMPD=/tmp/downloads &&   mkdir -p $TMPD
  mkdir -p /installs

  # bpp-core http://biopp.univ-montp2.fr/
  cd /installs
  git clone https://github.com/BioPP/bpp-core
  cd bpp-core
  git checkout tags/v2.4.1 -b v241
  mkdir build
  cd build
  cmake ..
  make -j 16 install

  # bpp-seq http://biopp.univ-montp2.fr/
```

```
    cd /installs
    git clone https://github.com/BioPP/bpp-seq
    cd bpp-seq
    git checkout tags/v2.4.1 -b v241
    mkdir build
    cd build
    cmake ..
    make -j 16 install

    # bpp-phyl http://biopp.univ-montp2.fr/
    cd /installs
    git clone https://github.com/BioPP/bpp-phyl
    cd bpp-phyl
    git checkout tags/v2.4.1 -b v241
    mkdir build
    cd build
    cmake  ..
    make -j 16 install

    # boost - C++ Libraries http://www.boost.org/
    cd /installs
    wget -P $TMPD -c
  https://boostorg.jfrog.io/artifactory/main/release/1.79.0/source/bo
  ost_1_79_0.tar.gz
    tar xvf $TMPD/boost_1_79_0.tar.gz
    cd boost_1_79_0
    ./bootstrap.sh --prefix=/usr/
    ./b2
    ./b2 install

    # glog - Google Logging Library https://github.com/google/glog/
    cd /installs
    git clone -b v0.5.0 https://github.com/google/glog
    cd glog
    cmake -H. -Bbuild -G "Unix Makefiles"
    cmake --build build --target install

    # gtest - Google Test Library
  https://github.com/google/googletest/
    cd /installs
    git clone https://github.com/google/googletest.git -b release-
  1.11.0
    cd googletest
    mkdir build
    cd build
    cmake ..
    make -j 4 install

    # ARPIP
    cd /opt
    git clone https://github.com/acg-team/bpp-arpip/
    cd bpp-arpip
    cmake --target ARPIP -- -DCMAKE_BUILD_TYPE=Release-static
  CMakeLists.txt
```

```
 85     make -j 8
 86
 87     clean
 88     cd / && rm -rf /installs
 89
 90   ##############################################################
 91
 92   Bootstrap: docker
 93   From: ubuntu:20.04
 94   Stage: final
 95
 96   %files from devel
 97     /opt/bpp-arpip                          /opt/
 98     /usr/local/lib/libbpp-core.so.4         /usr/local/lib/libbpp-
 99   core.so.4
100     /usr/local/lib/libbpp-seq.so.12         /usr/local/lib/libbpp-
101   seq.so.12
102     /usr/local/lib/libbpp-phyl.so.12        /usr/local/lib/libbpp-
103   phyl.so.12
104     /usr/local/lib/libglog.so.0             /usr/local/lib/libglog.so.0
105
106   %environment
107     export LC_ALL=C
108     export PYTHONNOUSERSITE=True
109
110   %post
111     export LC_ALL=C
112     export PYTHONNOUSERSITE=True
113     export DEBIAN_FRONTEND=noninteractive

        # Package cache in /tmp
        mkdir -p /tmp/apt20 &&  echo "Dir::Cache "/tmp/apt20";" >
      /etc/apt/apt.conf.d/singularity-cache.conf

        apt-get update && apt-get -y dist-upgrade && \
        apt-get install -y wget git libc6 libstdc++6 libgcc-s1

      %runscript
        /opt/bpp-arpip/ARPIP "$@"
```

`Stage: devel` lines 1-82 are compiling all the required libraries and tools to compile the ARPIP code. This stage can be used in a container that will run perfectly fine and with a bit more luck, if the final executable was fully static, one can even try to copy the file outside the container and run it as it is. Unfortunately, extracting the executable on Rackham shows these disappointing results.

Under Ubuntu 20.04 `GLIBC...` problems are resolved but `libbpp-core.so.4`, `libbpp-seq.so.12`, `libbpp-seq.so.12`, and `libglog.so.0` we just compiled remain missing.

```
ldd ARPIP
./ARPIP: /lib64/libm.so.6: version `GLIBC_2.29' not found (required by
```

```
./ARPIP)
./ARPIP: /lib64/libstdc++.so.6: version `GLIBCXX_3.4.26' not found
(required by ./ARPIP)
./ARPIP: /lib64/libstdc++.so.6: version `CXXABI_1.3.9' not found
(required by ./ARPIP)
./ARPIP: /lib64/libstdc++.so.6: version `GLIBCXX_3.4.20' not found
(required by ./ARPIP)
./ARPIP: /lib64/libstdc++.so.6: version `GLIBCXX_3.4.21' not found
(required by ./ARPIP)
        linux-vdso.so.1 =>  (0x00007ffe9257f000)
        libbpp-core.so.4 => not found
        libbpp-seq.so.12 => not found
        libbpp-phyl.so.12 => not found
        libglog.so.0 => not found
        libpthread.so.0 => /lib64/libpthread.so.0 (0x00002b529980d000)
        libstdc++.so.6 => /lib64/libstdc++.so.6 (0x00002b5299a29000)
        libm.so.6 => /lib64/libm.so.6 (0x00002b5299d31000)
        libgcc_s.so.1 => /lib64/libgcc_s.so.1 (0x00002b529a033000)
        libc.so.6 => /lib64/libc.so.6 (0x00002b529a249000)
        /lib64/ld-linux-x86-64.so.2 (0x00002b52995e9000)
```

And that is what we are doing in `Stage: final` - we copy the compiled libraries from `Stage: devel` in to a minimum Ubuntu 20.04 (could be other flavor as well) (lines: 90-95). In this case we avoid "stuffing" the container with unnecessary packages needed for compiling - `cmake build-essential zlib1g-dev`. There are no shortcuts - one needs to check you have everything you need in the new container - in this case `apt-get install -y libc6 libstdc++6 libgcc-s1` which will make sure we have the remaining libraries in `/lib64/...`.