Develop / Deploy with Singularity

my pattern and practice

https://www.ssec.wisc.edu/~jimd/Singularity/Singularity_Pattern_Practice.pdf

apptainer.org

Apptainer

THE CONTAINER SYSTEM FOR SECURE HIGH PERFORMANCE COMPUTING

Apptainer/Singularity is the most widely used container system for HPC. It is designed to execute applications at bare-metal performance while being secure, portable, and 100% reproducible. Apptainer is an open-source project with a friendly community of developers and users. The user base continues to expand, with Apptainer/Singularity now used across industry and academia in many areas of work.

In the past year I have delivered 3 CSPP packages in Singularity containers.

CSPP_MIRS was the most recent so I am going to use that as an example. It is a command line interface to a science algorithm (MiRS) from NOAA.

I have started to use Singularity for in-house projects too. Anything that I am likely to hand over to someone else to run, or that I want to move from machine to machine, is now a candidate for the Singularity treatment.

RUN-TIME HELP

```
$ $CSPP_MIRS_HOME/scripts/run_mirs.bash --help
Detected singularity version 3.8.5-1.el8

  Options:
  --version             show program's version number and exit
  ?, -h, --help         show this help message and exit
  -n, --nothing         make the PCF and SCS files, but do not execute.

  Mandatory Arguments:
    At a minimum these arguments must be specified [i.e. no default].

    -i INPUT_FILES, --input_files=INPUT_FILES
                        Fully qualified path to input files. May be directory
                        name, or directory/filemask (use * and ?, not regexp).
                        Filemasks MUST be "quoted" so OS does NOT expand!
                        e.g. -i "/dataDir/{GATMO,SATMS,TATMS}_npp*t1823486*.h5"
                        OR -i "/dataDir/{amsua,mhs}l1b_noaa18_*_1051*l1b" OR
                        -i "/dataDir/*.{AMA,MHS}X.M2.D13329.S0346.E0529.B*.SV"
                        (BUT -i /dataDir  really is the simplest way).
    -s MISSION, --sat=MISSION
                        The mission satellite data for MiRS algorithm.
                        Possible values (following MiRS documentation) ...
                        npp, n18, n19, n20, n21, metopA, metopB, metopC.

  Extra Options:
    These options may be used to customize behaviour of this program.

    -w WORK_DIR, --work=WORK_DIR
                        The directory in which all activity will occur,
                        this is where output files will be put.
                        [default: current directory].
    -d DYNAMIC_ANC, --dynanc=DYNAMIC_ANC
                        Path to root of dynamic ancillary data tree; beneath
                        here data are organized as per SSEC remote ancillary
                        server for CSPP, i.e. %Y_%m_%d_%j where %Y is 4-digit
                        year, %m is 2-digit month, %d is 2-digit day of month
                        and %j is 3-digit day of year.
                        [default: environment var. $CSPP_DYNAMIC_ANCIL_DIR].
    -r RES, --resolution=RES
                        Executes retrieval in RES resolution mode for sensors
                        aboard n18, n19, metopA, metopB and metopC.  RES is
                        'HI' or 'LO'.  When 'LO' the higher resolution sensor
                        (MHS) is aggregated to the lower resolution sensor
                        (AMSUA) footprint. 'HI' means AMSUA interpolated to
                        MHS footprint.
                        [default: 'HI'].
    -p PROCESSORS, --processors=PROCESSORS
                        Number of cpus to use for CSPP_MIRS processing.
                        [default: --processors=1, max is 32].
    -b BLOCK, --block=BLOCK
                        Choose bias correction file for block BLOCK SDRs for
                        S-NPP and NOAA-20 ATMS only.  Represents ADL Block 1
                        or Block 2 software version.  ADL block 2 SDRs in
                        effect from about March 2017.
                        [default: 2; the only other valid option is 1].
    -f, --SFR           Generate SnowFall Rate (SFR).  The only step that
                        requires ancillary (GFS) data to be fetched, so we
```

OLD WORLD

MiRS was delivered to SSEC by having the NOAA POC untar the package on an SSEC machine with the module system, they would load the necessary modules (compiler, libraries), build the system and run some test data through it.

I'd copy the system over to my user space on the same machine, verify that I could build and run the same test data, then set about writing a wrapper script to run the system in a Direct Broadcast context. Then I'd package up as a tarball the pre-compiled system, with all required support libraries and utilities, and go test it on several other machines running the target OS. (I'd also make a test package of Direct Broadcast data to go with).

Hopefully, when I run on the test machines, I do not find that I am missing a library or utility. If all good, other people will test it, I will write an installation guide, the package goes into operations here, and a few weeks later we announce its availability to CSPP users around the world.

NEW WORLD (note-to-self: change in FS visibility)

MiRS was delivered to me the same way - maybe we will move to providing a build container to NOAA POC - but this time it was the same as OLD WORLD.

I'd verify the same way, but then I'd set about rebuilding in a containerized environment - that's what I'll run through now.

HOW TO BUILD

This Singularity version of MIRS can be built using the bash scripts in the top-level directory and the contents of resource_dir/.

The order of operations is explained below. Note that whilst I have badged this repo CSPP-S_MIRS - to indicate a Singularity version of MiRS - for the user it is still CSPP_MIRS, and distribution and test data packages are named that way.

sandbox

```
$ ./sandbox.bash
```

This step makes a Singularity sandbox containing the required support libraries (hdf4, hdf5, netcdf4) and then copies & compiles the MiRS DAP using the sandbox. The reason for this step is that a sandbox can be --writable and can be more easily explored and interactively updated if system components are found to be missing or if DAP compiliation errors occur. If I can build a sandbox and the MiRS DAP with it, then I can probably be successful in building a .sif and then using it to build the MiRS DAP for distribution with the .sif as a CSPP package.

download

```
$ ./download.bash
```

This step makes a Singularity .sif file for building and running MiRS called mirs.sif based on rocker-geospatial. It is populated with the required support libraries (hdf4, hdf5, netcdf4).

build

```
$ ./build.bash
```

This step uses the .sif made in the the download step to build the MiRS DAP. There is an additional top-level bash script called noaa.bash that runs the build against the noaa test data granules provided in the MiRS DAP. I have not made this a separate step as this part of wrapping a NOAA DAP tends to be custom dependent on how NOAA provide test data. For MiRS it is in the DAP itself, but as binary files (EDR & DEP) and so need to be run through mirs2nc before I can use them as comparisons. These data need to be excised from the final CSPP package (as does the src/ tree as per our agreement with NOAA). So the time to do these comparisons is after the build, but before the package. It could be made part of the build.bash, but I prefer to take piecemeal as it won't be quick.

package

```
$ ./package.bash
```

This step integrates into a distribution directory the CSPP glue scripts in resource_dir/scripts/, the runtime .sif download_dir/mirs.sif and the DAP build in build_dir/DAP/. A version.txt is added, and a squashfs overlay made to house the DAP. The distribution is tarred up, but not compressed because the bulk of the system is inside the squashfs file and already compressed. Work takes place in package_dir/ which is where to find the newly made distribution tarball.

test

```
$ ./test.bash
```

This step uses the version in package_dir/ and runs it with the test data copied from resource_dir/data. The goal is to generate output data that can be distributed with the inputs in a test data package, where the output data is a baseline for users to compare against their results for the same satellite inputs. Afterwards the test data package can be found tarred and gzipped in test_dir/.

verify

```
$ ./verify.bash
```

This step follows essentially the same steps that a user would follow once they have downloaded the CSPP_MIRS distribution tarball and test data. It unpacks both, runs the package on the test data, and finally runs a verification script that is part of the distribution (i.e. not part of the test data package).

The page at left is lifted from the project documentation on gitlab.

It is sometimes useful for me to make a **sandbox** to start with, especially if I am unsure just what software I need in the container and I want to tinker with it - but I don't think it is helpful to get side-tracked by that here.

The **test & verify** steps are important to the specific enterprise of making CSPP software for delivery, with test data to verify correct installation, but again not central to the theme of this talk.

I am just going to run through **download, build, & package**.

Then I'll take a look at how I invoke the package via a bash script that has a little bit of work to do upfront - mainly to do with filesystem visibility from within the container.

This is what I start with…

```
[jimd@leo CSPP-S_MIRS]$ tree --charset==ASCII -L 3
|-- build.bash
|-- clean.bash
|-- download.bash
|-- package.bash
|-- README.md
|-- resource_dir
|   |-- bin
|   |   |-- h5diff
|   |   |-- h5dump
|   |   |-- h5ls
|   |   |-- h5repack
|   |   |-- h5stat
|   |   |-- nagg
|   |   |-- ncdump
|   |   |-- ncgen
|   |   |-- rename
|   |   `-- wgrib2
|   |-- data
|   |   `-- cspp_test
|   |-- mirs_v11r8_r110821321_20211117
|   |-- mirs_v11r8_r110821321_oper_20211117.tar.gz
|   |-- rocker-geo-mirs.def
|   |-- scripts
|   |   |-- bind_wrangle.R
|   |   |-- cspp_mirs_env.sh
|   |   |-- dirlist_mirs.R
|   |   |-- mirs_srcdiff.R
|   |   |-- mirs_verify.R
|   |   |-- run_mirs.bash
|   |   |-- run_mirs.pl
|   |   `-- sing_test.bash
|   `-- tarfiles
|       |-- hdf-4.2.15.tar.gz
|       |-- hdf5-1.10.6.tar.gz
|       |-- hdf-eos2-3.0-src.tar.gz
|       |-- netcdf-c-4.7.3.tar.gz
|       `-- netcdf-fortran-4.4.5.tar.gz
|-- sandbox.bash
|-- test.bash
`-- verify.bash
```

~6GB untarred

This is what I end with…

```
[jimd@leo CSPP_MIRS_3_0]$ tree --charset==ASCII -L 2
.
|-- bin
|   |-- h5diff
|   |-- h5dump
|   |-- h5ls
|   |-- h5repack
|   |-- h5stat
|   |-- nagg
|   |-- ncdump
|   |-- ncgen
|   |-- rename
|   `-- wgrib2
|-- docs
|   |-- MIRS_Algorithm_Theoretical_Basis_Document.pdf
|   |-- MIRS_Delivery_Memo.pdf
|   |-- MIRS_Interface-Control-Document.pdf
|   |-- MIRS_Operations_Manual.pdf
|   |-- MIRS_ProcessControl_and_ProductionRules.pdf
|   |-- MIRS_System_Description_Document.pdf
|   |-- MIRS_System_Maintenance_Manual.pdf
|   |-- MIRS_Users_Manual.pdf
|   |-- NOAA_Products_MSPPS2MIRS_Transition.pdf
|   `-- Performances
|-- mirs.sif
|-- scripts
|   |-- bind_wrangle.R
|   |-- cspp_mirs_env.sh
|   |-- dirlist_mirs.R
|   |-- mirs_srcdiff.R
|   |-- mirs_verify.R
|   |-- run_mirs.bash
|   |-- run_mirs.pl
|   `-- sing_test.bash
`-- version.txt
```

~2GB squashfs

download

```
$ ./download.bash
```

This step makes a Singularity .sif file for building and running MiRS called mirs.sif based on rocker-geospatial. It is populated with the required support libraries (hdf4, hdf5, netcdf4).

```
#!/bin/bash
#
# Top-level script to download some requirements to build a version of MiRS
#   for distribution to users with a recent version of Singularity/Apptainer
#
# Execute this script in the directory where you found it:
#
# ./download.bash
#

# exit when the first non-zero exit status is encountered
set -e

# test for singularity
resource_dir/scripts/sing_test.bash

# print every command we run
set -v

# make a download directory
mkdir -p download_dir/tarfiles
cd download_dir

# copy tarfiles needed for support libraries
rsync -a ../resource_dir/tarfiles .

# need to be able to bail and carry on now
set +e

# create the .sif used for both build & runtime
sudo singularity build mirs.sif ../resource_dir/rocker-geo-mirs.def

# Done
echo Done.
```

These are HDF & NetCDF libraries

Initially I had this as part of the build step (that compiles the algorithm package).

I prefer to have this separate a) so that I don't risk breakage using a new image on dockerhub that I did not need to fetch and b) so that I can continue to work the next steps disconnected from internet.

Clearly the .def file is crucial.

This is the singularity build command and container definition file - note sudo

```
Bootstrap: docker

From: rocker/geospatial

%files

    tarfiles /opt

%post

    # Install system packages
    apt-get update
    apt-get --assume-yes install tree lftp rsync bison byacc flex puppet rename ripgrep
    apt-get --assume-yes install imagemagick >/dev/null

    # Install extra R packages
    install2.r --skipinstalled getopt sodium Cairo magick cowplot akima \
      ggnewscale kableExtra flexdashboard DT smoothr

    mkdir -p /opt/build

    # For gcc/g++/gfortran 10.0 you may need to uncomment
    # export FCFLAGS="-w -fallow-argument-mismatch -O2"
    # export FFLAGS="-w -fallow-argument-mismatch -O2"

    # Install hdf4 from source
    cd /opt/build
    tar -xzf ../tarfiles/hdf-4.2.15.tar.gz
    cd hdf-4.2.15
    ./configure --prefix=/usr/local --with-szlib --enable-netcdf=yes
    make clean && make -j 16 && make install

    # Install hdfeos from source
.
.
.

    # Clean up & exit
    cd /opt
    rm -rf build
    rm -rf tarfiles
    touch I_AM_SINGULARITY

%environment

    export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib
```

Why rocker/geospatial? I do not have a good answer for that - the scripting for this project is in Perl. I think it just had a compiler set that verified the MiRS test data to machine precision and I like having R tools around for scripting. The .sif created is pretty big and most of it I do not need - so one could argue it is a bad choice. Base distro is Ubuntu.

So I have cut out some bits to save space here

Here is the start of a Singularity .def file from a different project.

I have used “devtoolset” to choose the compiler set needed for this algorithm delivery.

```
Bootstrap: docker

From: centos/s2i-base-centos7

%files

    tarfiles /opt

%post

    # Install packages (centos-release-scl is already installed)
    yum -y install tree tcsh perl-core lftp zip bison byacc flex devtoolset-7-gcc*

    mkdir -p /opt/build

    # Install kai from source
    cd /opt/build
    tar -xzf ../tarfiles/kai-1.11.tar.gz
    cd kai-1.11
    ./configure
    make
    make install

    # Install hdf4 from source
    cd /opt/build
    tar -xzf ../tarfiles/hdf-4.2.15.tar.gz
    cd hdf-4.2.15
    echo ./configure --prefix=/usr/local --enable-netcdf=yes > build-unix.sh
    echo 'make clean && make && make install' >> build-unix.sh
    chmod 755 build-unix.sh
    scl enable devtoolset-7 ./build-unix.sh

    # Install hdf5 from source
    cd /opt/build
    tar -xzf ../tarfiles/hdf5-1.10.6.tar.gz
    cd hdf5-1.10.6
    echo ./configure --prefix=/usr/local --enable-fortran --enable-unsupported > build-unix.sh
    echo 'make clean && make && make install' >> build-unix.sh
    chmod 755 build-unix.sh
    scl enable devtoolset-7 ./build-unix.sh
```

```
Bootstrap: docker

From: centos/s2i-base-centos7

%files

    compiler_pkg /opt

%post

    # Install packages to run ACSPO build
    yum -y install tcsh perl-core lftp bison byacc flex ftp

    # Install packages that I find helpful
    yum -y install tree zip

    # make directories for compiler and workspace
    mkdir -p /opt/intel /opt/work

    #  Install Intel compiler suite
    cd /opt/work
    tar -xzf ../compiler_pkg/parallel_studio_xe_2019_update5_cluster_edition.tgz
    cd parallel_studio_xe_2019_update5_cluster_edition
    sh ./install.sh --ignore-cpu -s /opt/compiler_pkg/silent.cfg_2019.0.5
    cd ..
    rm -rf parallel_studio_xe_2019_update5_cluster_edition

    # Clean up & exit
    cd /opt
    rm -rf compiler_pkg
    touch I_AM_SINGULARITY
    history -c

%environment

    export CC=icc
    export CXX=icpc
    export F77=ifort
    export FC=ifort
    export F90=ifort
    export F9X=ifort
    export F95=ifort
    export INTEL_HOME=/opt/intel/19
    export PATH=$PATH:$INTEL_HOME/bin
    export INTEL_LIBS=$INTEL_HOME/compilers_and_libraries_2019.5.281/linux
    export CPATH=$INTEL_LIBS/ipp/include:$INTEL_LIBS/mkl/include:$INTEL_LIBS/pstl/include
    export CPATH=$CPATH:$INTEL_LIBS/tbb/include:$INTEL_LIBS/daal/include
    export MKLROOT=$INTEL_LIBS/mkl
    export IPPROOT=$INTEL_LIBS/ipp
    export PSTLROOT=$INTEL_LIBS/pstl
    export LIBRARY_PATH=$INTEL_LIBS/compiler/lib/intel64:$INTEL_LIBS/ipp/lib/intel64
    export LIBRARY_PATH=$LIBRARY_PATH:$INTEL_LIBS/mkl/lib/intel64:$INTEL_LIBS/daal/lib/intel64
    export LIBRARY_PATH=$LIBRARY_PATH:$INTEL_LIBS/tbb/lib/intel64/gcc4.7
    export LD_LIBRARY_PATH=$LIBRARY_PATH
```


Here is another one for a project that needed to be built with INTEL compilers and so I had to install the compiler suite and have access to SSEC’s license at build-time.

(I ended up making two containers for this project - one for building and a separate one for deployment).

OK, back to the main thread.

After ./download.bash I have a directory download_dir/ that is a sibling to resource_dir/

The important thing is the singularity container file mirs.sif

If I have built this right, I can use it to compile the NOAA algorithm and I can use it as part of the final deployment tarball because it will contain all the support libraries needed and will run on any users OS provided they have a recent version of Singularity installed.


```
download_dir/
|-- mirs.sif
`-- tarfiles
    |-- hdf-4.2.15.tar.gz
    |-- hdf5-1.10.6.tar.gz
    |-- hdf-eos2-3.0-src.tar.gz
    |-- netcdf-c-4.7.3.tar.gz
    `-- netcdf-fortran-4.4.5.tar.gz
```

build

```
$ ./build.bash
```

This step uses the .sif made in the the download step to build the MiRS DAP. There is an additional top-level bash script called noaa.bash that runs the build against the noaa test data granules provided in the MiRS DAP. I have not made this a separate step as this part of wrapping a NOAA DAP tends to be custom dependent on how NOAA provide test data. For MiRS it is in the DAP itself, but as binary files (EDR & DEP) and so need to be run through mirs2nc before I can use them as comparisons. These data need to be excised from the final CSPP package (as does the src/ tree as per our agreement with NOAA). So the time to do these comparisons is after the build, but before the package. It could be made part of the build.bash, but I prefer to take piecemeal as it won't be quick.

```
#!/bin/bash
#
# Top-level script to build a version of MiRS for distribution to
#   users with a recent version of Singularity.
#
# Execute this script in the directory where you found it:
#
# ./build.bash
```

snipped out a few lines of house-keeping so this would fit on page

```
< S N I P >
# name the .sif in downloads_dir that we are using to build the DAP
sif=$(realpath download_dir/mirs.sif)

# make the build directory
mkdir -p build_dir
cd build_dir
build_home=$PWD
```

make a build directory

```
# rysnc the support static binaries into place
rsync -a ../resource_dir/bin .

# rsync the scripts into place
rsync -a ../resource_dir/scripts .
```

these are my wrapper scripts

```
# rsync the .sif into place
rsync -a $sif .

# make a DAP directory in build_dir
mkdir -p DAP
rsync -a ../resource_dir/mirs_v11r8_r110821321_20211117 DAP/
```

this is the delivery that I need to compile

```
# get ready to compile the MiRS code
cd DAP/mirs_v11r8_r110821321_20211117
export MIRS_ROOT=$PWD

# change MIRS_ROOT in paths
cd setup
sed -i "s|<replace_with_mirs_root>|$MIRS_ROOT|" paths

# make a build script
cd $build_home
cat <<EOF > build_mirs.bash
```

create a little bash script to run in the container

```
#!/bin/bash
cd $MIRS_ROOT/src/crtm/REL-2.1.1/configure
source gfortran.setup
cd ..
make && make install
cd ../..
make
EOF
chmod 755 build_mirs.bash
```

execute the script so that the compilation is in the container environment

```
# build the mirs DAP
singularity --silent exec --home=$PWD mirs.sif ./build_mirs.bash

# Done
echo Done.
```

```
[jimd@leo build_dir]$ tree --charset==ASCII -L 3
|-- bin
|   |-- h5diff
|   |-- h5dump
|   |-- h5ls
|   |-- h5repack
|   |-- h5stat
|   |-- nagg
|   |-- ncdump
|   |-- ncgen
|   |-- rename
|   `-- wgrib2
|-- build_mirs.bash
|-- DAP
|   `-- mirs_v11r8_r110821321_20211117
|       |-- bin
|       |-- cmake
|       |-- CMakeLists.txt
|       |-- configure-with-cmake.bash
|       |-- data
|       |-- doc
|       |-- gui
|       |-- LICENSE
|       |-- logs
|       |-- readme
|       |-- scripts
|       |-- setup
|       |-- src
|       `-- version.txt
|-- mirs.sif
`-- scripts
    |-- bind_wrangle.R
    |-- cspp_mirs_env.sh
    |-- dirlist_mirs.R
    |-- mirs_srcdiff.R
    |-- mirs_verify.R
    |-- run_mirs.bash
    |-- run_mirs.pl
    `-- sing_test.bash
```

So now I have another sibling directory …

build_dir/

.. and an executable NOAA algorithm package.

This is where I work on my Perl wrapper script because I now have a system that runs.

It is convenient to have a short bash script to pass my run_mirs.pl wrapper script to the container.

Rather than look at run_mirs.bash now, I will push on and do the packaging and take a peek at it later.

package

```
$ ./package.bash
```

This step integrates into a distribution directory the CSPP glue scripts in resource_dir/scripts/, the runtime .sif download_dir/mirs.sif and the DAP build in build_dir/DAP/. A version.txt is added, and a squashfs overlay made to house the DAP. The distribution is tarred up, but not compressed because the bulk of the system is inside the squashfs file and already compressed. Work takes place in package_dir/ which is where to find the newly made distribution tarball.

```
< S N I P >
# define version and tarfile
VERSION=CSPP_MIRS_3_0
TARFILE=CSPP_MIRS_V3.0.tar

# make a package directory
mkdir -p package_dir
cd package_dir
package_home=$PWD

# create a directory to turn into a squashfs partition
revision=mirs_v11r8_r110821321_20211117
mkdir -p squash/MIRS/DAP/$revision
cd squash/MIRS/DAP/$revision

# rsync the DAP from build_dir, excluding the sources
rsync -a  ../../../../../build_dir/DAP/$revision/bin .
rsync -a  ../../../../../build_dir/DAP/$revision/doc .
rsync -a  ../../../../../build_dir/DAP/$revision/scripts .
rsync -a  ../../../../../build_dir/DAP/$revision/setup .
mkdir -p data
rsync -a  ../../../../../build_dir/DAP/$revision/data/SemiStaticData data
rsync -a  ../../../../../build_dir/DAP/$revision/data/StaticData data

# squash it
cd $package_home
mksquashfs squash squash.sqsh

# make a version directory that will become the tarball directory
mkdir -p $VERSION
cd $VERSION

# Make version.txt and time-stamp it
echo 3.0    $(date --utc --date="today" +\%Y\-\%m\-\%dT\%H:\%M:\%S)  > version.txt
echo CSPP_HEAP Version 3.0 release. >> version.txt

# populate with resource_dir/bin/ and resource_dir/scripts/
rsync -a ../../resource_dir/scripts .
rsync -a ../../resource_dir/bin .

# make a directory for docs from the DAP that you want to pull out
rsync -a ../squash/MIRS/DAP/$revision/doc .
mv doc docs

# bring the runtime sif across
rsync -a ../../download_dir/mirs.sif .

# add the squashfs overlay
singularity sif add --datatype 4 --partfs 1 --parttype 4 --partarch 2 --groupid 1 mirs.sif $package_home/squash.sqsh

# make the distribution tarball
cd $package_home
tar -cf $TARFILE $VERSION/

# Done
echo Done.
```

This is package.bash,
I snipped the head off it.

So I make another sibling directory
package_dir/
and copy across just that part of
the NOAA algorithm package that
I need/want to have in my delivery.

The directory I put it in I am
going to make a squashfs
copy of …

grab the wrapper scripts
and binary utilities

… and then add that
to my build container.

tar it up and we are done.

This is what I end with…


```
[jimd@leo CSPP_MIRS_3_0]$ tree --charset==ASCII -L 2
.
|-- bin
|   |-- h5diff
|   |-- h5dump
|   |-- h5ls
|   |-- h5repack
|   |-- h5stat
|   |-- nagg
|   |-- ncdump
|   |-- ncgen
|   |-- rename
|   `-- wgrib2
|-- docs
|   |-- MIRS_Algorithm_Theoretic
|   |-- MIRS_Delivery_Memo.pdf
|   |-- MIRS_Interface-Control-D
|   |-- MIRS_Operations_Manual.p
|   |-- MIRS_ProcessControl_and_
|   |-- MIRS_System_Description_Document.pdf
|   |-- MIRS_System_Maintenance_Manual.pdf
|   |-- MIRS_Users_Manual.pdf
|   |-- NOAA_Products_MSPPS2MIRS_Transition.pdf
|   `-- Performances
|-- mirs.sif
|-- scripts
|   |-- bind_wrangle.R
|   |-- cspp_mirs_env.sh
|   |-- dirlist_mirs.R
|   |-- mirs_srcdiff.R
|   |-- mirs_verify.R
|   |-- run_mirs.bash
|   |-- run_mirs.pl
|   `-- sing_test.bash
`-- version.txt
```

cspp_mirs_env.sh

```
#!/bin/bash
#
# Environment script for CSPP_MIRS
#
# EDIT THIS FOR YOUR INSTALLATION AND SOURCE BEFORE RUNNING CSPP_MIRS
#
# CSPP_MIRS_HOME points to the CSPP_MIRS installation directory.
# CSPP_DYNAMIC_ANCIL_DIR is default local dynamic ancillary directory.
#
# These directories must exist for CSPP_MIRS to run. Set CSPP_MIRS_HOME in your
# environment then source this file OR uncomment and edit the line(s) below.
#
# export CSPP_MIRS_HOME=/data/jimd/Projects/CSPP-S_MIRS/test_dir/CSPP_MIRS_3_0
export CSPP_DYNAMIC_ANCIL_DIR=$CSPP_MIRS_HOME/data/dynanc
#
# JPSS_REMOTE_ANC_DIR is the URL of ancillary data server and, unless you
# are mirroring to another location, you should not need to change these.
#
export JPSS_REMOTE_ANC_DIR=https://jpssdb.ssec.wisc.edu/cspp_v_2_0/ancillary
export PATH=$CSPP_MIRS_HOME/scripts:$PATH
```

sing_test.bash

```
#!/bin/bash
singularity_version='None'
IFS=':'
for dir in $PATH
do
  if [[ -x $dir/singularity ]] ; then
    singularity_version=`singularity --version`
  fi
done
unset IFS
if [[ $singularity_version == 'None' ]] ; then
  echo Singularity not detected
  exit 1
else
  echo Detected $singularity_version
fi
```

```
#!/usr/bin/env bash
#
# Skinny wrapper environment script for MIRS (run_mirs.pl) under CSPP with singularity
#
# The location of the singularity container is required.
# So I am going to require CSPP_MIRS_HOME to find it.
#
# exit when the first non-zero exit status is encountered
set -e

# check installation home and chdir to it
if [[ -z $CSPP_MIRS_HOME ]]; then
  echo "Must provide CSPP_MIRS_HOME in environment. See cspp_mirs_env.sh" 1>&2
  exit 1
fi

# convert $CSPP_MIRS_HOME to canonical form
export CSPP_MIRS_HOME=$(readlink -f $CSPP_MIRS_HOME)

# check for singularity
$CSPP_MIRS_HOME/scripts/sing_test.bash

# name the container
mirs=$CSPP_MIRS_HOME/mirs.sif

# use readlink -f to track hidden links needed to grok bind string for singularity container of app
readlinked=
dirlist=$(singularity --silent exec --home=$PWD --bind=$CSPP_MIRS_HOME $mirs Rscript $CSPP_MIRS_HOME/scripts/dirlist_mirs.R $@)
for mydir in $dirlist; do export readlinked=$readlinked,$(readlink -f $mydir); done
bind=$(singularity --silent exec --home=$PWD --bind=$CSPP_MIRS_HOME $mirs Rscript $CSPP_MIRS_HOME/scripts/bind_wrangle.R $readlinked)

# run run_mirs.pl in the singularity container built for mirs with grokked $bind
singularity --silent exec --home $PWD $bind $mirs perl $CSPP_MIRS_HOME/scripts/run_mirs.pl $@

# Done
echo Done.
```

This is the run script of the application, run_mirs.bash

1) System requirements I want to adhere to are: Singularity and bash
2) --bind tells Singularity what parts of the filesystem are visible to it.
3) Singularity cannot follow a symlink on part of the filesystem it can “see” to a target on part of the filesystem it cannot “see”.
4) For me - an inexpert bash programmer - the tools I want to use to resolve this problem (viz a short R script) require access to the containerized environment.
5) “readlink -f” in the host OS environment is key part of the solution.

```
$ run_mirs.bash -s n20 -i input/ -w ~/work -d dynanc/ -p 4


dirlist=$(singularity --silent exec --home=$PWD --bind=$CSPP_MIRS_HOME $mirs Rscript
$CSPP_MIRS_HOME/scripts/dirlist_mirs.R $@)

echo $dirlist

input/ /home/jimd/work dynanc/ /home/jimd/Data/jimd/MIRS/V3 /home/jimd/Data/jimd/MIRS/
CSPP_MIRS_3_0/data/dynanc /media/jimd/data/jimd/MIRS/CSPP_MIRS_3_0


for mydir in $dirlist; do export readlinked=$readlinked,$(readlink -f $mydir); done

echo $readlinked

,/media/jimd/data/jimd/MIRS/V3/input,/home/jimd/work,/media/jimd/data/jimd/MIRS/V3/dynanc,/media/
jimd/data/jimd/MIRS/V3,,/media/jimd/data/jimd/MIRS/CSPP_MIRS_3_0


bind=$(singularity --silent exec --home=$PWD --bind=$CSPP_MIRS_HOME $mirs Rscript
$CSPP_MIRS_HOME/scripts/bind_wrangle.R $readlinked)

echo $bind

--bind /media,/home
```

```
singularity --silent exec --home $PWD $bind $mirs perl $CSPP_MIRS_HOME/scripts/run_mirs.pl $@
```

New?

The Apptainer action commands (run, exec, shell, and instance start) will accept the --bind/-B command-line option to specify bind paths, and will also honor the $APPTAINER_BIND and $APPTAINER_BINDPATH environment variables (in that order).

Develop / Deploy with Singularity

my pattern and practice

PROS

Allows package integrator to choose/control the development platform.

Singularity does not limit cores/disk/memory at container creation time (does Docker, VMs? I don't know) which means all the resources of host system are available at runtime.

Performance does not seem to be adversely impacted compared to bare metal.

Portability. I have a very high confidence that my package will work on a target machine provided it has Singularity installed. I can ensure all other code dependencies are within the container.

CSPP has provided pre-compiled "install and run in a few minutes" packages from its inception. Containerization is a pathway for current source package distributors to deliver as "ready-to-run" and could simplify running on cloud infrastructure.

CONS

Makes package integrator responsible for systems-level tasks; this can be a new responsibility for traditional applications programmers.

Adds several layers of complexity that can be onerous, especially if the package to be built is very simple.

Requires sudo access for .sif creation step.

Filesystem visibility requires a good deal of attention - well that is what has caused me most trouble.

Will this give us seamless access to Windows users? Not yet, by the sounds of it:

You will need a Linux system to run Apptainer natively. Options for using Apptainer on Mac and **Windows** machines, along with alternate Linux installation options are discussed in the installation section of the admin guide.

END