EPIC Workshop

Building the SRW Application in Containers and on Bare-Metal Systems

Objective: Learn how to use Singularity containers to modify, build, and run the UFS SRW Application.

Mandy Parson - Technical Training Specialist - ?'s about EPIC, Slack, Technical Issues

Dr. Natalie Perlin - Cloud Computing Engineer - ?'s about Accessing the Cloud

Michael Kavulich, Jr. - SRW App Developer - ?'s about the SRW App

How to use Slack

Please refer to your email or the chat for the invite code to the **EPIC slack workspace**

If you have a question, pose it in the **#2023-ams** channel.

Click “reply in thread” **if answering a question.**

The **#technical-issues** space is for access/audio issues.

“**Reply in Thread**” Function should be utilized

Any Questions?

EPIC's Short Course vs. EPIC's Workshop at the AMS 2023

Short Course on UFS SRW

- Overview of *UFS SRW App*
- Set and run the workflow in AWS cloud
  - Control case (25-km grid)
  - Modified case (3-km grid)
  - Modified case-2 (3-km grid)
- Grid generation methods
- Plot and view the results

January 8, 2023, 8:00am-12noon MST

UFS SRW Workshop

- Software container basics
- Singularity containers
- Build a container from an image
- Download and build the SRW App
- Run the forecast of a control case (25-km grid)

January 8, 2023, 2:00-3:45pm MST

Agenda

2:00 - 2:10 - Introductions and logging in

2:10 - 2:25 - Container basics and Singularity containers

2:30 - 2:45 - Build a writable “sandbox” container

2:45 - 2:55 - Explore the container, download the SRW and build it (10 min BREAK)

3:05 - 3:10 - Setting up a script to run containerized executables

3:10 - 3:30 - Run a test case

3:30 - 3:45 - Change paths and re-run the test case using rebuilt ufs_model

Leveraging Singularity Containers to Build and Run the UFS SRW Application

Dr. Natalie Perlin

EPIC/EMC
RedLine Performance Solutions

Get Logged In

- Separate head-node instances for each participant are set up in the Amazon cloud by EPIC
- Every participant receives the **IP-address** of their **head node** to log in during the Workshop. A google sheet with a list of head node IP-addresses next to each participant name is in the slack channel as well.
- Everyone uses the same **PEM file** which can be downloaded from the slack channel, and needs to log as user "**ubuntu**"
- Open a Terminal (or PowerShell on Windows), set permissions 400 (read-only) to PEM file

```
$ chmod 400 /path-to/EPIC_RSA.pem
$ ssh -i /path-to/EPIC_RSA.pem ubuntu@your-ip-address
```

- Make sure to log into your own head node only (IP address). Upon login, type "`who`" to verify you are the only one logged into the instance

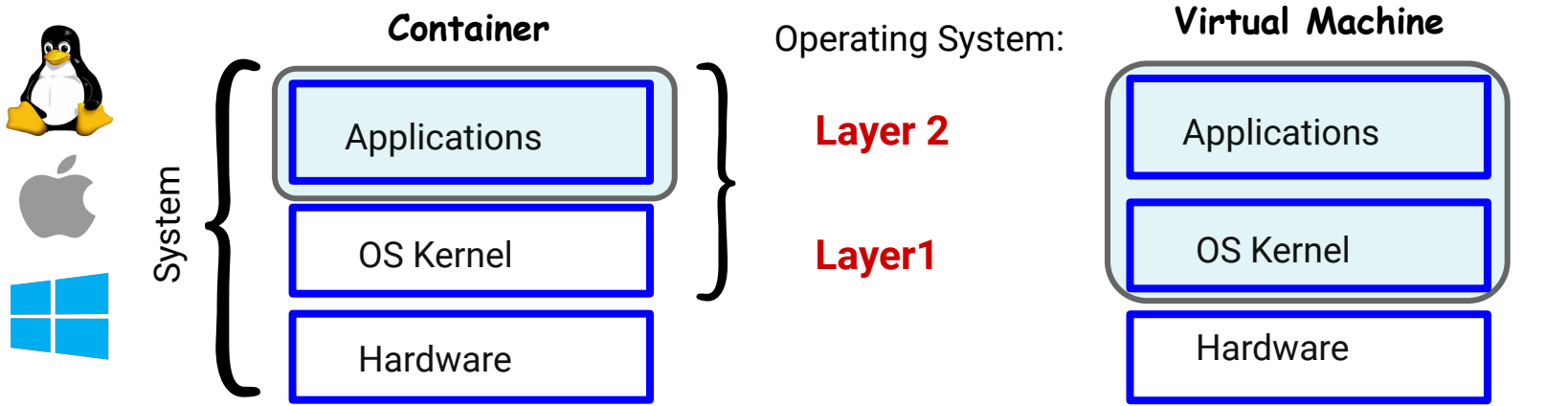
Software Containers for Using the SRW App

- Use a *software container* with prepackaged SRW App and dependencies
- Use *Singularity* software to configure and run the SRW on a new AWS cloud instance
- Rebuild the SRW App code using the prepackaged dependencies and run it

Software Container Basics: What is a Container?

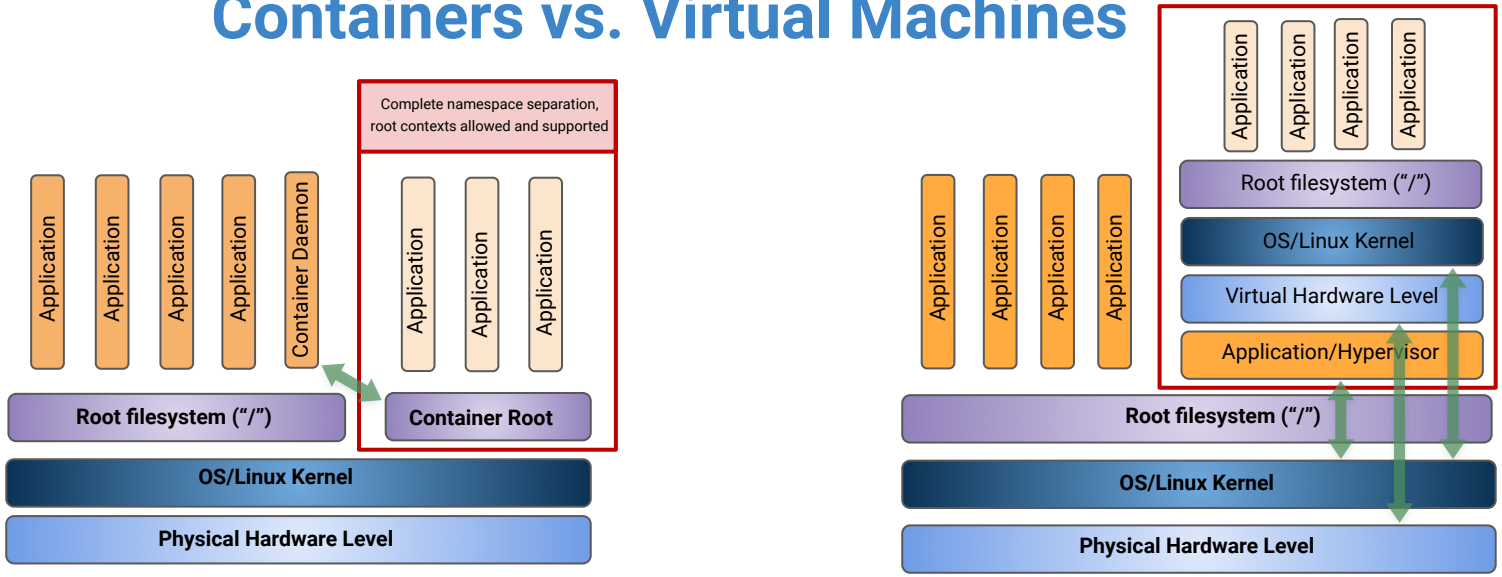
- Container is a way to package applications with all the necessary dependencies and configurations
- Container is another process on your machine, running in an isolated environment
- Container is an abstraction for set of technologies aimed to solve software portability problem
- Container is a virtual interface at the OS level to package software and deploy them to different systems, aimed at solving software portability problem
- Container is a file (or collection of files) that stores everything needed to run a target application

Container vs. Virtual Machine: what part is virtualized?



- Containers are **not** Virtual Machines, they use the same *OS kernel* as a host system

Containers vs. Virtual Machines

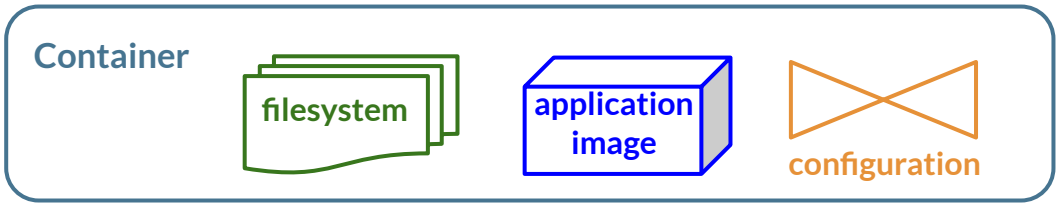


- Containers are not fully isolated, and could use host system network and directories

Container vs. Image

Terms "**container**" and "**image**" are often used interchangeably, but there are subtle differences:

- ★ Image is a static file, a "snapshot" of the files/filesystem taken at a specific time
- ★ Container has layers such as a) *Linux base image*, b) *application image*, c) *configuration*



Container is a **running environment** of an image

Container implementation software: Singularity Containers

- Container software: ***Docker*** and ***Singularity*** are most common (both used by EPIC)
- ***Singularity*** containers could be run as a **regular user** on HPC systems, no root privileges needed
- Containers can be **executed** from local image files, or pulling images from a Docker registry (hub).
- Containers can be built as an **non-writable image** file (***.sif**, ***.sqsh**, ***.img**) or as a **writable "sandbox"**.
- Containers can **bind to directories** on the host system where files can be both read and written
- **Environment variables** from host system can be passed into the container, or cleaned ("**-e**" option)
- **Environment variables** to be used inside the container can also be specified from the host system, e.g.,

```
export SINGULARITYENV_LD_LIBRARY_PATH=/opt/lib/
```

- Common *Singularity* container commands –
  - **`build`** – build an image local or remote, could read and write if bound correctly
  - **`exec`** or **`run`** – run an executable or a script that is built into the container
  - **`shell`** – similar to logging into the container; exit with an **`exit`** command

Singularity Container Commands (examples)

- Build **an image** from a *Docker* registry: build local *Ubuntu* image, latest version

```
singularity build ubuntu-latest.sif docker://ubuntu:latest
```

- Build a **writable sandbox**, "`-s`" or "`--sandbox`" options, into a local directory:

```
singularity build -s ubuntu-latest docker://ubuntu:latest
singularity build -s ubuntu-latest ubuntu-latest.sif
```

- Run a **single command** in the container (started from the image.sif), and then exit:

```
singularity run -e image.sif gcc -v
singularity run -e image.sif cp -r my_directory .
```

- Run an **interactive session**:

```
singularity shell -e -B /scratch:/scratch -B /data image.sif
     cd my_directory
     gcc p.c && ./a.out
exit
```

Explore the New Container

Command to *shell* (log) into a newly built container is as follows (**$** - a shell prompt):

```
$ singularity shell -e -B /home:/home -B /data:/data
./ubuntu20.04-intel-srwapp-AMS-release-public-v2.1.0
```

We can make this a little simpler for repeated use with an environment variable

```
$ export img=$PWD/ubuntu20.04-intel-srwapp-AMS-release-public-v2.1.0
```

Now the command becomes

```
$ singularity shell -e -B /home:/home -B /data:/data $img
```

Build the container with the public release of SRW v2.1.0

($ = terminal prompt)

- Build a writable *sandbox container* from a *Singularity image* (a static file) in **/data/**

```
$ singularity build --sandbox ubuntu20.04-intel-srwapp-AMS-release-public-v2.1.0
/data/ubuntu20.04-intel-srwapp-AMS-release-public-v2.1.0.img
```

```
ubuntu@ip-10-0-7-49:~$ pwd
/home/ubuntu
ubuntu@ip-10-0-7-49:~$ singularity build --sandbox ubuntu20.04-intel-srwapp-AMS-release-public-
v2.1.0 /data/ubuntu20.04-intel-srwapp-AMS-release-public-v2.1.0.img
INFO:    Starting build...
INFO:    Verifying bootstrap image /data/ubuntu20.04-intel-srwapp-AMS-release-public-v2.1.0.img
WARNING: integrity: signature not found for object group 1
WARNING: Bootstrap image could not be verified, but build will continue.
INFO:    Creating sandbox directory...
INFO:    Build complete: ubuntu20.04-intel-srwapp-AMS-release-public-v2.1.0
ubuntu@ip-10-0-7-49:~$
```

Explore the New Container

```
ubuntu@ip-10-0-7-49:~$ export img=$PWD/ubuntu20.04-intel-srwapp-AMS-release-public-v2.1.0
ubuntu@ip-10-0-7-49:~$ singularity shell -e -B /home:/home -B /data:/data $img
Singularity> ls /
bin      dev          glade  lfs2  lib32   media  root     scratch1     sys  var
boot     discover     home   lfs3  lib64   mnt    run      scratch2     tmp  work
contrib  environment  lfs    lfs4  libx32  opt    sbin     singularity  u
data     etc          lfs1   lib   lustre  proc   scratch  srv          usr
Singularity> ls /opt
miniconda3  spack-stack  ufs-srweather-app
Singularity> ls /opt/ufs-srweather-app/container-bin/
chgres_cube       filter_topo   global_equiv_resol  make_solo_mosaic   sfc_climo_gen  vcoord_gen
cpld_gridgen      fregrid       inland              orog               shave          weight_gen
emcsfc_ice_blend  fvcom_to_FV3  lakefrac           orog_gsl           ufs_model
emcsfc_snow2mdl   global_cycle  make_hgrid          regional_esg_grid  upp.x
Singularity>
```

Container Contents: SRW App and dependencies

The container is based on **Ubuntu 20.04** with additional software installed in **/opt**

Under the **/opt** directory, the **spack** package management system used by the UFS and JEDI communities to handle dependencies is installed.

The *Intel OneAPI* compilers and MPI implementations are installed via **spack** in **/opt** and all the dependencies required by the UFS SRW Application have been built using *Intel OneAPI* and are installed there as well.

All the libraries and dependencies can be accessed using **Lmod** modules

A source code of the SRW is located in **/opt/ufs-srweather-app/**, and a compiled version also installed in **/opt/ufs-srweather-app/container-bin .**

- **Initialize the *Lmod***
- **Load the modules for building the SRW**
- **List the modules**

```
$ source /usr/share/lmod/lmod/init/bash
$ module use
/opt/ufs-srweather-app/modulefiles
$ module load build_singularity_intel
$ module list
```

```
Singularity> source /usr/share/lmod/lmod/init/bash
Singularity> module use /opt/ufs-srweather-app/modulefiles
Singularity> module load build_singularity_intel
Singularity> module list

Currently Loaded Modules:
  1) intel-oneapi-compilers/2022.1.0       26) nemsio/2.5.4
  2) stack-intel/2021.6.0                  27) netcdf-c/4.7.4
  3) intel-oneapi-mpi/2021.6.0             28) netcdf-fortran/4.5.4
  4) stack-intel-oneapi-mpi/2021.6.0       29) openblas/0.3.19
  5) bacio/2.4.1                           30) parallel-netcdf/1.12.2
  6) bufr/11.7.0                           31) parallelio/2.5.2
  7) ca-certificates-mozilla/2022-07-19    32) patchelf/0.15.0
  8) cmake/3.22.1                          33) pkg-config/0.29.2
  9) crtm/2.3.0                            34) py-cython/0.29.30
 10) curl/7.49.1                           35) py-numpy/1.22.3
 11) ecbuild/3.6.5                         36) py-pip/22.1.2
 12) esmf/8.3.0b09                         37) py-setuptools/63.0.0
 13) fms/2022.01                           38) py-wheel/0.37.1
 14) g2/3.4.5                              39) sfcio/1.4.1
 15) g2tmpl/1.10.0                         40) sigio/2.3.2
 16) gftl-shared/1.5.0                     41) sp/2.3.3
 17) git/2.25.1                            42) stack-python/3.8.10
 18) git-lfs/2.9.2                         43) w3emc/2.9.2
 19) hdf5/1.12.1                           44) w3nco/2.4.1
 20) ip/3.3.3                              45) wgrib2/2.0.8
 21) jasper/2.0.25                         46) wrf-io/1.2.0
 22) libjpeg/2.1.0                         47) yafyaml/0.5.1
Singularity>
```

UFS
UNIFIED FORECAST SYSTEM

Download a new copy of the public release of SRW v2.1.0 on the host

- Since the sandbox container is writable, we could start changing things inside it
- Clone the SRW on the host system (one long line)
- Check out the external repositories to the SRW

```
$ git clone -b release/public-v2.1.0
https://github.com/ufs-community/ufs-srweather-app.git
$ cd ufs-srweather-app
$ ./manage_externals/checkout_externals
```

UFS
UNIFIED FORECAST SYSTEM

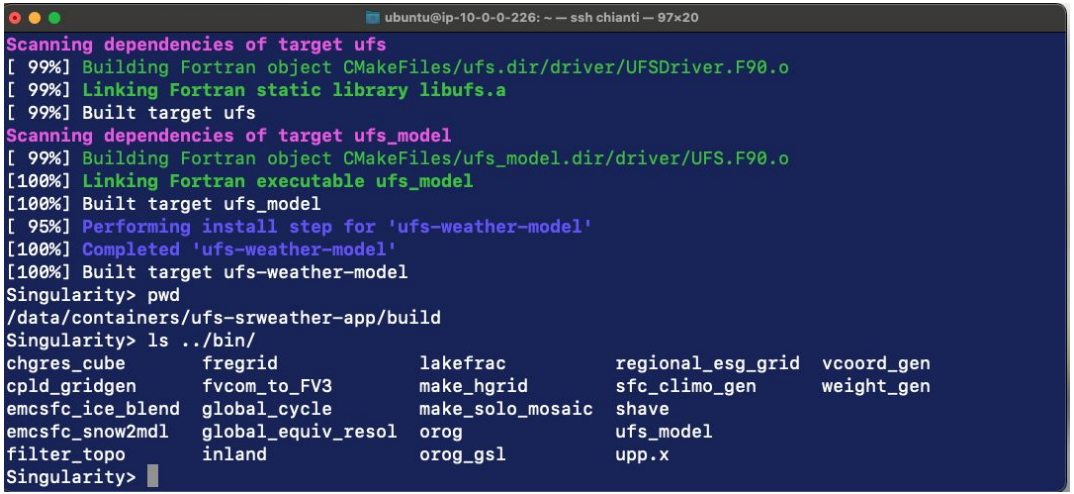
Build the SRW that you downloaded to your host machine using the dependencies (software modules) while *inside* the container

- Go to the top directory of the SRW
- Create a build directory and cd into it
- Configure the application with *cmake* and compile it

```
$ cd ~/ufs-srweather-app
$ mkdir build && cd build
$ cmake -DCMAKE_INSTALL_PREFIX=.. -DCMAKE_INSTALL_BINDIR=exec ..
$ make -j 8
```

UFS
UNIFIED FORECAST SYSTEM

Completed build will look like this:



UFS
UNIFIED FORECAST SYSTEM

In and Out of the Sandbox Container: Run a Newly Built SRW App

- While the newly created executables exist on the host file system, they require the container to run.
- They can be run inside the container directly, but will not run across multiple nodes that way
- To execute from the host system, a script is required that will enable multi-node execution via MPI
- A script look something like this (next slide):

```
#!/bin/bash
#turn on debugging
set -x
#Internal setting for MPI jobs
export SINGULARITYENV_FI_PROVIDER=tcp
#ensure singularity is using bash
export SINGULARITY_SHELL=/bin/bash
#Container name-can be a sandbox image as well
img="/data/ubuntu20.04-intel-srwapp-AMS-release-public-v2.1.0.img"
#A soft link to this script with the name of the executable desired (e.g. ufs_model,
#chgres_cube, etc.)
cmd=$(basename "$0")
arg="$@"
echo running: singularity exec "${img}" $cmd $arg
#script launches singularity with bindings
/usr/bin/singularity exec -B /home:/home -B /data:/data "${img}" $cmd $arg
```

Create a Runscript for the SRW App to Launch it using a Container

- Copy **/opt/ufs-srweather-app/container-scripts/srw.sh-template** to your host system and call it **run_container_executable.sh**

```
$ cd ~/ufs/srweather-app
$ cp /opt/ufs-srweather-app/container-scripts/srw.sh-template
run_container_executable.sh
```

Open up the run_container_executable.sh script using your favorite text editor

```
$ vi run_container_executable.sh
```

ubuntu@ip-10-0-13-186: /data/EPIC/UFS_SRW_App/v2p1/input_model_data — ssh aws — 133

```
#!/bin/bash
set -x
export SINGULARITYENV_FI_PROVIDER=tcp
export SINGULARITY_SHELL=/bin/bash
dir="$(dirname $(readlink -f ${BASH_SOURCE[0]}))"
img="IMAGE"
cmd=$(basename "$0")
arg="$@"
echo running: singularity exec "${dir}/${img}" $cmd $arg
PATH_TO_SINGULARITY exec -B /BINDDIR:/BINDDIR -B /LOCDIR:/LOCDIR "${dir}/${img}" $cmd $arg

~
~
~
```

- Change the line "**img=IMAGE**" to be "**img=full-path-to-your-container-image**". This will be the same as the variable you set outside the script.
- Inquire the singularity installation location by running a command `which singularity`, record the path, and replace **PATH_TO_SINGULARITY** variable with that path
- Replace each "**BINDDIR**" with "**home**" and each "**LOCDIR**" with "**data**" to make sure we bind to both **/home** and **/data** directories on the host system
- Save and close

Should end up looking like this–

```
#!/bin/bash
set -x
export SINGULARITYENV_FI_PROVIDER=tcp
export SINGULARITY_SHELL=/bin/bash
img="/home/ubuntu/ubuntu20.04-intel-srwapp-AMS-release-public-v2.1.0"
cmd=$(basename "$0")
arg="$@"
echo running: singularity exec "${img}" $cmd $arg
/usr/local/bin/singularity exec -B /data:/data -B /home:/home "${img}" $cmd $arg
```

- If we make a soft link from this script to any name in the Singularity path, the executable of that name will be run via the container.

As an example, make a link from run_container_executable.sh to “which”

```
$ ln -s run_container_executable.sh which
```

- Now run the ***which*** command (make sure to add “./ “) to find executables inside the container

```
$ ./which ufs_model
```

```
ubuntu@ip-10-0-7-49:~/ufs-srweather-app$ ./which ufs_model
+ export SINGULARITYENV_FI_PROVIDER=tcp
+ SINGULARITYENV_FI_PROVIDER=tcp
+ export SINGULARITY_SHELL=/bin/bash
+ SINGULARITY_SHELL=/bin/bash
+ img=/home/ubuntu/ubuntu20.04-intel-srwapp-AMS-release-public-v2.1.0
++ basename ./which
+ cmd=which
+ arg=ufs_model
+ echo running: singularity exec /home/ubuntu/ubuntu20.04-intel-srwapp-AMS-release-public-v2.1.0 which ufs_mo
del
running: singularity exec /home/ubuntu/ubuntu20.04-intel-srwapp-AMS-release-public-v2.1.0 which ufs_model
+ /usr/local/bin/singularity exec -B /data:/data -B /home:/home /home/ubuntu/ubuntu20.04-intel-srwapp-AMS-rel
ease-public-v2.1.0 which ufs_model
/opt/ufs-srweather-app/container-bin/ufs_model
ubuntu@ip-10-0-7-49:~/ufs-srweather-app$
```

- How can we get the container to run our newly built version of the **ufs_model**? We need to change the path used inside the container.
- Edit run_container_executable.sh and add the following line after other “export” commands:
  export SINGULARITYENV_PREPEND_PATH=/home/ubuntu/ufs-srweather-app/exec
- Alternatively, you could use sed to edit it in the following way:

```
$ sed -i "3 i export
SINGULARITYENV_PREPEND_PATH=/home/ubuntu/ufs-srweather-app/exec"
run_container_executable.sh
```

- Remove the “debug” line with “set -x”

```
$ sed -i "/set -x/d" run_container_executable.sh
```

Let's re-run the which command to see which ufs_model will be run now.

```
./which ufs_model
```

```
ubuntu@ip-10-0-7-49:~/ufs-srweather-app$ ./which ufs_model
running: singularity exec /home/ubuntu/ubuntu20.04-intel-srwapp-AMS-release-public-v2
.1.0 which ufs_model
/home/ubuntu/ufs-srweather-app/exec/ufs_model
ubuntu@ip-10-0-7-49:~/ufs-srweather-app$
```

The script should now is finding your newly built **ufs_model** first in its path.

- Let's test the container version of the ufs_model using a sample test case.

```
$ cd ~
$ cp -r /data/Test_fcst .
$ cd Test_fcst
```

- `# this link should be in place already, but if it is not run the following`

```
$ ln -s /home/ubuntu/ufs-srweather-app/run_container_executable.sh ./ufs_model
$ sbatch run_fcst_post.sh
```

- Upon completion, there will be post-production grib2 files in the **./postprd** directory.

Modify the SRW Code and Rebuild the SRW App

- For example, open up **~ufs-weather-app/sorc/ufs-weather-model/FV3/ccpp/physics/physics/radiation_surface.F90** and go to **line 840**, change one or two of the constants slightly (e.g. 0.97 to 0.98) before building. This will result in slightly different results that they will be able to verify after running the original version of **ufs_model** and their newly built version.
- Re-build it inside the container!

```
$ cd ~/ufs-srweather-app
$ ./devclean.sh -a
$ mkdir build && cd build
$ cmake -DCMAKE_INSTALL_PREFIX=.. -DCMAKE_INSTALL_BINDIR=exec ..
$ make -j 8
```

- Let's re-run the sample forecast using the newly built **ufs_model**

```
$ cd ~/Test_fcst
```

- Since we changed things slightly in our new build we will expect the new forecast to be slightly different than the original. Copy postprd files to postprd.orig, so we can compare and re-submit

```
$ mkdir postprd.orig
$ mv postprd/* postprd.orig
$ sbatch run_fcst_post.sh
```

- When the new forecast is finished, check to see if the output files differ–

```
$ diff postprd/rrfs.t18z.natlev.f006.subconus_ind_3km.grib2
   postprd.orig/rrfs.t18z.natlev.f006.subconus_ind_3km.grib2
```

Try it out on your own!

We are available to take questions through zoom and slack

Continued use of this application

We are so glad you would like to continue to use this application on your own!

The sandbox will close at 5pm Mountain Standard Time.

The singularity image we used today can be pulled from
https://epic-sandbox-srw.s3.amazonaws.com/ubuntu20.04-intel-srwapp-AMS-release-public-v2.1.0.img

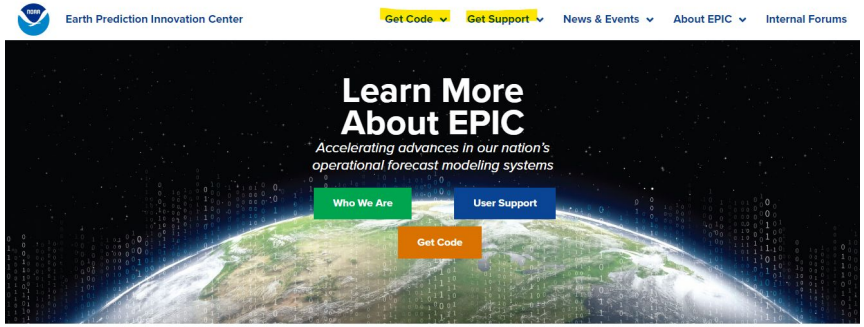
Or, you can build a container directly from docker hub with the command–
singularity build ubuntu20.04-intel-srwapp-AMS-release-public-v2.1.0.img \
docker://noaaepic/ubuntu20.04-intel-srwapp:AMS-v2.1.0

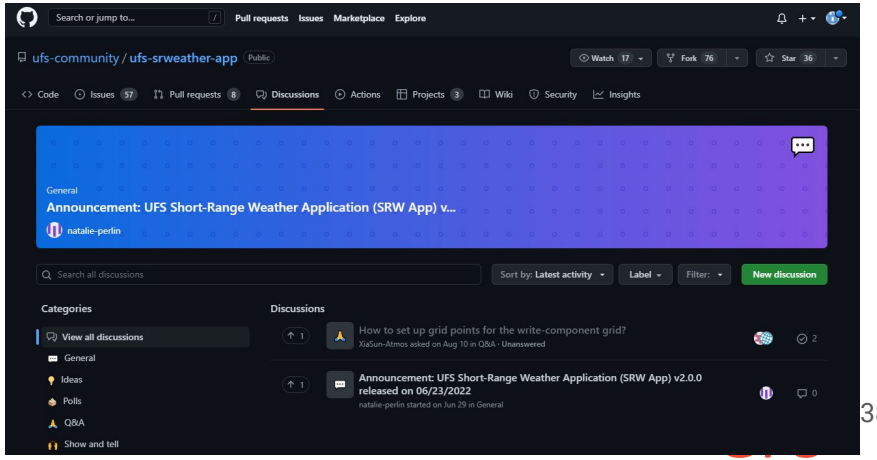
The slack channel #2023-ams will remain open and monitored for a few weeks (until end of January) if you would like to ask questions there.

Getting Help

Check out our “Get Code” and “Get Support Sections on our website (epic.noaa.gov) for more help.

Still have questions? Use Github Discussions to have your questions answered by the EPIC team and SRW community users





Wrapping Up

Please check your email for our feedback survey.

We appreciate your input!

Feedback Survey