**Beginner’s Guide to Singularity (and Singularity with subMIT/condor)**

We can use images to basically transport an environment (linux kernel + state/compiled code) from one machine to another without worrying too much about the differences between the machines. To use an image takes effectively two main steps:

1. Build an image
2. Deploy the image

I will focus here on Singularity, particularly using it for DarkQuest on subMIT.

There are some more detailed tutorials here, if you want to learn more general info:

<https://awesome-workshop.github.io/intro-to-docker/>

<https://awesome-workshop.github.io/docker-cms/>

A singularity quick start guide is here: <https://singularity-userdoc.readthedocs.io/en/latest/quick_start.html>

**Building**

* In any case, first let’s focus on building an image on subMIT
* General build documentation: <https://singularity-userdoc.readthedocs.io/en/latest/build_a_container.html>
* There are a couple ways to do this:
  + Build from a Recipe file - here you have to write up a file that specifies base your base image and what commands should be run to set up the image
  + Build from a sandbox - here a directory gets created that you can build on and update dynamically
* For DQ on subMIT purposes, I think the sandbox start is easiest, so that’s what I’ll focus on

How to build an image

* One interesting feature of singularity is that your home directory (~) is always mounted, and that’s your starting location when you enter the image interactively or execute commands using the image, so it might be easiest to do this in your home directory
* First, run
  + singularity build --sandbox singularity\_test2/ docker://centos:7
    - “--sandbox singularity\_test2/” specifies that this is a sandbox build. It will create a directory called singularity\_test2 (you can change the name of course)
    - “docker://centos:7” is the base image. This image is based on centOS 7 for example
* Then you can run:
  + singularity shell -B /cvmfs/ -B /usr -B /etc singularity\_test2
    - “shell” starts up the image as a container and drops you in an interactive shell in the container (<https://singularity-userdoc.readthedocs.io/en/latest/appendix.html?highlight=shell#shell>). You can do developments here that change the image itself and the code that’s in singularity\_test2. I.e. if you compile something in the shell, now it’s compiled code in the image
    - The “-B /cvmfs/ -B /usr -B /etc” options bind mount /cvmfs, /usr, and /etc to the container. I needed these to compile the DQ code. Mounting /cvmfs, for example, is needed if your code refers to anything there
  + Within the shell, you can run the commands in the dq\_install scripts. You can git pull the code and compile, and this becomes part of the image
  + You type in “exit” to exit the container
* To send the image to other machines, you’ll want a singularity image file (.sif)
  + First, I had to change the permissions of the singularity\_test2 folder:
  + chmod -R 777 singularity\_test2/
    - I’m not sure if this is a subMIT-specific thing, or if it’s even best practice
  + Then you run something like:
  + singularity build new-sif.sif singularity\_test2/
    - Singularity build can be used to convert between images and sandboxes and vise versa (there’s also a “writable” format for images: <https://singularity-userdoc.readthedocs.io/en/latest/quick_start.html#build-images-from-scratch>)
* Now you’ve got an image file with pre-compiled code
* One thing that’s important to note is that *within the image*, any code that you had in the singularity\_test2 directory is in the “/” directory of the image. E.g., if you git pulled e1039-core into singularity\_test2, then within the image e1039-core is located at /e1039-core

**Deploying**

* There are a couple ways to deploy your new image on another machine
  + As a sandbox - here you could start up a shell in the container and make changes to the image on the new machine (then you could make a *new* image file if you wanted)
  + Or you can just run commands directly through the container. This is fast but static
* On condor, I think we need to run commands directly through the container, **but** if you wanted to do something on LPC for whatever reason, you could do things as a sandbox:
  + Import your .sif file
  + singularity build --sandbox singularity\_test2/ new-sif.sif
    - In the DQ-specific case, you’ll want to be a little careful about naming conventions, since the setup sh files might have hard-coded paths in them
  + singularity shell -B /cvmfs -B /usr -B /etc singularity\_test2/
    - The stuff that you had in the build machine should be here
* To run commands directly through the container, you just do:
  + singularity exec -B /cvmfs -B /usr -B /etc singularity\_test2/ bash runner.sh
    - Here, runner.sh is a script that contains the commands that you want to run in the container

**Running on Condor through subMIT**

* I was running in /work/submit/wmccorma/condor\_tester/
* To run on condor, you can use a submission script like:

universe = vanilla

executable = run\_condor.sh

output = logs/mu\_test.out

log = logs/mu\_test.log

error = logs/mu\_test.error

transfer\_input\_files = runscript.sh,setup\_mye1039.sh,/usr/lib64/libgsl.so.0,/usr/lib64/atlas/libsatlas.so.3,RecoE1039Sim.C

should\_transfer\_files = IF\_NEEDED

request\_memory = 4000

use\_x509userproxy = True

x509userproxy = /home/submit/wmccorma/x509up\_u211941 #Change this for individual use.

#Submit to OSG using this line

+ProjectName = DarkQuest\_Sample\_Generation\_wmccorma

#Restrict to sl7 machines

+REQUIRED\_OS = "rhel7"

requirements = \

(OSGVO\_OS\_STRING == "RHEL 7") || \

(HAS\_SINGULARITY == true || GLIDEIN\_REQUIRED\_OS == "rhel7") || \

(GLIDEIN\_Site == "MIT\_CampusFactory" && (BOSCOGroup == "paus" || BOSCOGroup == "bosco\_cms"))

queue

* There are a few things you need:
  + An executable like run\_condor.sh
  + A runscript.sh file that contains commands to run in the container
  + And for DQ specifically, you need:
    - A version of setup\_mye1039.sh
    - A version of RecoE1039Sim.C
    - Some library files: /usr/lib64/libgsl.so.0 and /usr/lib64/atlas/libsatlas.so.3
* The run\_condor script that I use is shown below:
  + It copies in an image file that’s in hadoop, transfers a few of the input files into /tmp, runs the commands in the container, then copies the output root file back to hadoop
  + The use of /tmp is because singularity automatically mounts the ~ directory, which is *not* where you are dropped in condor jobs. So the files that you want to use in the job should be somewhere that you specify. This could probably be rethought, but for the purpose of getting things up and running, it worked and was the easiest thing to do
  + I think you have to make sure that you haven’t previously written the output files to /tmp, which is why I have those rm commands in there (I was getting some hanging jobs without that)

#!/bin/bash

hostname

pwd

ls -lt

echo "Condor Job ClassAds"

cat $PWD/.job.ad

gfal-copy gsiftp://se01.cmsaf.mit.edu:2811//mitgroups/DarkQuest/wmccorma/new-sif.sif .

echo "Using singularity image"

#Want to copy the config /tmp/job\_$jobid

gfal-copy

#Might want to make a /tmp/job\_jobid

sed /tmp/ with /tmp/job\_jobid #or you could replace /tmp/ with like /XXX/ #check out the sed command in dq\_install.sh

cp setup\_mye1039.sh /tmp/job\_$jobid

cp RecoE1039Sim.C /tmp/job\_jobid

mkdir /tmp/lib64

cp libgsl.so.0 /tmp/lib64/

cp libsatlas.so.3 /tmp/lib64/

# Don’t have to do this if use job id

rm /tmp/output.root

rm /tmp/output\_DST.root

rm /tmp/eval.root

singularity exec -B /cvmfs -B /usr -B /etc new-sif.sif bash runscript.sh

echo 'I ran, now ls tmp'

ls -lth /tmp

gfal-copy -f file:///tmp/job\_jobid/output.root gsiftp://se01.cmsaf.mit.edu:2811//mitgroups/DarkQuest/wmccorma/outputFromCondor.root #Might want to use other directories

rm -rf /tmp/job\_jobid

* The runscript.sh file is below:
  + You’ll note that I source /tmp/setup\_mye1039.sh rather than the setup\_mye1039.sh that’s in the usual location, and I specify /tmp/ as the output directory for the ntuples. You don’t have write privileges within the container (hence /tmp/ for output), and the directory locations have changed relative to when you compiled (hence the new setup script)

#!/bin/bash

cd /e1039-core/DarkQuest/e1039-analysis/SimHits/

source /tmp/setup\_mye1039.sh

cd macro/

which root

ls -lth /tmp/

#cat /tmp/RecoE1039Sim.C

root -b -q '/tmp/job\_jobid/RecoE1039Sim.C(10, 3, 1, 520, true, true, false, "", "", "output.root", "/tmp/job\_jobid/")'

ls

* The setup\_mye1039.sh script is:
  + Here, the files are all relative to the “/” directory, which is due to the container structure
  + I also had to add those missing library files, which is where the added LD\_LIBRARY\_PATH line comes from

export MY\_E1039=/core-inst/this-e1039.sh

export DIR\_TOP=$(dirname $(readlink -f $BASH\_SOURCE))

echo $MY\_E1039

echo $DIR\_TOP

source $MY\_E1039

export LD\_LIBRARY\_PATH=/e1039-core/DarkQuest/e1039-analysis/SimHits/install/lib/:$LD\_LIBRARY\_PATH

export LD\_LIBRARY\_PATH=/tmp/job\_jobid/lib64/:$LD\_LIBRARY\_PATH

export DIR\_CMANTILL=/e1039-core/DarkQuest/e1039-analysis/SimHits

* Lastly, in the RecoE1039Sim.C that’s sent to the job, you need to change “eval.root” to “/tmp/eval.root”
  + The job tries to write out ./eval.root inside a non-writable directory in the container
  + eco->set\_eval\_file\_name(out\_path + "eval.root");