Containers

Docker (https://www.docker.com/) is one of the most popular technologies for putting all your code and dependecies a “container” that can then be executed on any system that can execute Docker containers. For example, if you needed a program that only works with Python 2 and you don’t want to actually install Python2 on your system (things can get really confused with `pip` and `conda` and such), it might be better to build the app into a container.

One down side to Docker is that it requires “root” permissions to run. Most HPC systems will not allow non-privedged users to run as root, so the wonderul folks at Lawrence Berkeley National Labs created Singularity to run containers as a normal user. Singularity containers can be built directly from Docker containers. Because of Singularity’s flexibility and abilty to run on HPC platforms, I will focus only on this container.

NB: You must have root access on a Linux box (or an appropriate Linux VM) in order to build Docker and Singularity containers.

Simple Example

In the `examples/simple` directory is a Makefile that will show you the Singularity commands to build an image and a sample `image.def` definition file to instruct Singularity to build a container from a base Ubuntu image, add `git`, use `git` to clone this repository, add the necessary directory to the `$PATH`, and execute the `hello.sh` script automatically when the container is run.

Let’s look at the Makefile first:

```
$ cat -n Makefile
     1	.PHONY = img img_old shell run clean
     2	
     3	SINGULARITY = /usr/local/bin/singularity
     4	IMG = hello-0.0.1.img
     5	DEF = image.def
     6	
     7	img: clean
     8		sudo $(SINGULARITY) build $(IMG) $(DEF)
     9	
    10	img_old: clean
    11		sudo $(SINGULARITY) create --size 512 $(IMG)
    12		sudo $(SINGULARITY) bootstrap $(IMG) $(DEF)
    13	
    14	shell:
    15		sudo $(SINGULARITY) shell --writable -B $(shell pwd):/tmp $(IMG)
    16	
    17  run:
    18      sudo $(SINGULARITY) exec $(IMG) hello.sh
    19
    20  clean:
    21      rm -f $(IMG)
```

You just need to type `make` or `make img` to execute the `img` target. Notice this target has a dependency on the `clean` target which will get rid of any existing image.

Here is the `image.def` that tells Singularity how to build the image:

```
$ cat -n image.def
     1  BootStrap: docker
     2  From: ubuntu:latest
     3
     4  %environment
     5      PATH=/app/biosys-analytics/lectures/15-containers/examples/simple:$PATH
     6
     7  %runscript
     8      exec hello.sh
     9
    10  %post
    11      apt-get update
    12      apt-get install -y locales git
    13      locale-gen en_US.UTF-8
    14
    15      mkdir -p /app
    16      cd /app
    17
    18      git clone https://github.com/hurwitzlab/biosys-analytics.git
```

Notice how I set the `%environment` to include the `$PATH` to the directory containing the `hello.sh` program. After you have an image, you can execute the image itself like it were a program and Singularity with automatically use the defined `runscript`:

```
$ make img
<...output ellided...>
Finalizing Singularity container
Calculating final size for metadata...
Skipping checks
Building Singularity image...
Singularity container built: hello-0.0.1.img
Cleaning up...
$ ./hello-0.0.1.img
Hello from Singularity!
$ make run
```

```
sudo /usr/local/bin/singularity exec hello-0.0.1.img hello.sh
Hello from Singularity!
```

I've included in the `img_old` targe the older syntax to `create` and `bootstrap` a new container in case you are restricted to that version. One problem with the older syntax is that the image is statically sized at build time. You may need to allocate more disk space than you actually need for the purpose of building, but after the build you might like to shrink the image's size. This is not possible. The newer `build` command will create a base image and expand and contract the size as needed leaving you with an image that is compressed into the smallest possible size. The problem is that you cannot `shell` into the image to test and debug; therefore, it is necessary to build with the older syntax. Notice that I cannot `make shell` until I `make img_old`. With a shell, I can `cd` into my directory and execute the script in the environment that it will be deployed! (This is truly a big deal. You can finally deliver a complete executable environment for your users!!!)

```
$ make shell
sudo /usr/local/bin/singularity shell --writable -B /home/u20/kyclark/work/biosys-analytics/
ERROR  : Unable to open squashfs image in read-write mode: Read-only file system
ABORT  : Retval = 255
make: *** [shell] Error 255
$ make img_old
$ make shell
sudo /usr/local/bin/singularity shell --writable -B /home/u20/kyclark/work/biosys-analytics/
WARNING: Non existent bind point (file) in container: '/etc/localtime'
Singularity: Invoking an interactive shell within container...

Singularity hello-0.0.1.img:~> cd /app/biosys-analytics/lectures/15-containers/examples/simp
Singularity hello-0.0.1.img:/app/biosys-analytics/lectures/15-containers/examples/simple> ./
Hello from Singularity!
```