Singularity Containers

Containerization of workloads has become popular, particularly using Docker. However, Docker is not suitable for HPC applications due to needing root level access. There are a couple of alternatives for HPC containers, with Singularity being the one that covers the more general cases. Singularity has been deployed on the Argon cluster, and can import docker containers. The Singularity Documentation - Sylabs.io site is where you should go for documentation on using singularity. This wiki page will only cover some basic tenets and provide information on how to get a functioning singularity container on the Argon HPC system.

One of the primary uses of containers is to build a software stack or a workflow. With singularity, the process is to

1. create an empty container
2. bootstrap a minimal OS into the container
3. install packages, import data, etc. into the container
4. copy the container image to the target run system
5. run commands in the container via the singularity interface

Note that steps 1-3 are done as the root user. This means that you must do those on a Linux machine that you are an administrator of. You will not be able to build containers on the HPC system, but will only be able to run them. All modifications to the container must happen on a Linux machine that you have root access to. If you have a Mac or Windows machine you can use VirtualBox to set up a virtual Linux machine.

Your home directory and scratch directories, in `/nfsscratch` and `localscratch`, will be accessible from the container when run on Argon. Note that using the host home directory could present issues as the dot files in your home directory may not be suitable for the container environment. One way around this is to use an alternate home directory in the container with the `-H` flag. That will limit the scope of your home directory in the container so plan accordingly.

Note that if you are running a GPU job in a container that the code will need to interface with the GPU (Nvidia) kernel module on the host system. If you run that container on different host systems then it is possible that the kernel module versions will be different. Even in the case where the kernel module is the same version across host systems it may have been built against a different OS kernel. This is something to keep in mind, particularly if the goal of using a container is to get a reproducible environment, as the host Nvidia kernel module would be a variable in the environment.

The following is an example of a Ubuntu container definition. Note that this is an example, as the header section can change, and represents the minimum of what is needed to create the container OS.

```
BootStrap: debootstrap
OSVersion: bionic
MirrorURL: http://us.archive.ubuntu.com/ubuntu/
```

Running MPI jobs in a singularity container will require using openmpi-2.1.x. Please see the Singularity on HPC article for details and note that may change over time.

Regardless of whether a job in a container is MPI or not, it should be submitted to the SGE queue system, as any other resource intensive job would. Singularity is queue system agnostic so jobs in singularity containers will run in SGE without any changes. The only difference being, of course, that the job is run through the singularity interface. Also, note that you can specify a `%runscript` in the definition file. If your container is dedicated to a single process or workflow then you could simply execute the container to trigger the entire workflow. The runscript can also accept parameters so you could vary the run based on parameters passed to a container runscript.

```
singularity run my_container.img
```

See the singularity documentation and the Singularity Hub for examples.

Please note that we are providing singularity on the HPC system so that you can run containerized workflows and/or use custom built software stacks. We will strive to make things work as smoothly as possible as far as running the containers but building containers is up to the end user at this time.