Grid computing, containers, and big data

Joe Parker

16th November 2017

Running containers

0. For this practical we're lucky enough to use Apocrita, the main QMUL HPC resource with 1000s of cores available. We can do a ***lot*** of damage here, so we need to be responsible! Before going any further make sure you've read the QMUL Usage Policy.

1. First we'll login to the grid. Use your QMUL username (`xyz123`) to log in with ssh (QMUL login help):

```
ssh xyz123@login.hpc.qmul.ac.uk
```

2. Singularity is an academic counterpart to Docker, used to run containers on HPC grids safely (you can do a lot of damage with all that power…!). First we'll use it to run a Docker Whalesay image (sort-of a HelloWorld for Docker images); then you'll run your image from this morning; finally we'll look at grid submissions. First we need to load the singularity module, pull the Whalesay image from Docker Hub, and then we can run it locally:

```
# load the Singularity module
module load singularity

# pull the Whalesay container image - by default to the current directory.
# (the -size argument determines the default image size)
singularity pull --size 1000 docker://docker/whalesay

# now we can run the container locally, using Singularity
singularity exec whalesay.img foo
```

Output:

```
[xyz123@frontend11 ~]$ singularity exec whalesay.img cowsay foo
 _____
< foo >
 -----
    \
     \
      \
                    ##        .
              ## ## ##       ==
           ## ## ## ##      ===
       /""""""""""""""""___/ ===
  ~~~ {~~ ~~~~ ~~~ ~~~~ ~~ ~ /  ===- ~~~
       \______ o          __/
        \    \        __/
          \____\______/
```

Using the grid to submit jobs

To make full use of the cluster, we need to use the grid engine to submit our job. Look at the file `helloworld_singularity.sh`

This can be submitted to the queue with `qsub helloworld_singularity.sh`.

Writing a Docker container

We're going to write a container that can (could) run on the cluster.

Steps:

1. Install docker To install Docker on Linux see https://docs.docker.com/get-started/part2

2. Start an instance of the Docker machine on your host computer: `docker-machine start`
3. Check docker is working by pulling whalesay to this host:

```
docker pull docker/whalesay
docker image list
# you should see the whalesay image listed
docker run whalesay cowsay foo
# as above
```

2. Clone this repository with the dockerfile in it and other bits Clone this repository with `git clone https://github.com/lonelyjoeparker/C`
   If you've previously cloned it, you need to make sure your copy is up to date (it almost certainly isn't). `cd` into the directory and use `git fetch && git pull` to update it.
3. Build it (bit of editing) We're going to build a really simple container image. Navigate to the `helloworld_container` directory. The `Dockerfile` contains the instructions for building a Docker image, while `requirements.txt`, `really_simple.py` and `simpler.sh` are other required files.

To build this image into a Docker container, just run `docker build -t <some_container_name> .` Note that images can take up more and more room if you aren't careful, so use `docker container list` to see which you have, and how big they are; and `docker container rm <container>` to remove them. Note that you need to refer to containers with their hash e.g. `2f46202d0519` in:

```
REPOSITORY                TAG                 IMAGE ID            CREATED             SIZE
cyverseuk-raxml           latest              2f46202d0519        3 hours ago         369MB
lonelyjoeparker/phylodev  helloworld          2f46202d0519        3 hours ago         369MB
```

4. Check it works: Try `docker run some_container_name`
5. Get a docker hub account. Now this container works we can run it anywhere that supports Docker or Singularity - but we'll use Apocrita! First we have to upload our image to https://hub.docker.com and you'll need to get an account to do this. You will also need to create a repository. This will be public.
6. Tag it `docker tag <your container name> <your_dockerhub_username>/<your_repository>:<some_random_tag>`
7. Push (upload) it to your repository `docker push <your_dockerhub_username>/<your_repository>:<some_random_tag>`
   You should be able to see it (after a few minutes) if you open your account e.g. `https://hub.docker.com/r/<your_dockerhub_username>`

Pulling our own Docker container

We will pull our container from Docker Hub.

1. Go to the correct dir on apocrita
2. Pull your image: `singularity pull <your_dockerhub_username>/<your_repository>:<some_random_tag>` It will be created as a .img image file.
3. Check it works: `singularity run <your_image.img>`
4. Run through the queue properly - write a minimal qsub file.

Putting it together - running our container on the grid to fit a model.

I have put a RAxML docker image on the hub. It just runs raxml with arguments we give it on the qsub file, and exits.

- We run it with singularity
- We call it with qsub
- We need to mount ('bind' in singularity language) it to the machine e.g. `singularity run -B <full_path>:/class_data`.
  We can now run this with a qsub script.

We will (fairly exhastively run a script to collect the results)

Now we can run a class problem! The directory