

Note: You are not reading the most recent version of this documentation. **2019** is the latest version available.

A hands-on introduction to Docker

Introduction and goals

Docker is a mechanism for building and running isolated “containers” of software. Docker containers act much like virtual machines but are smaller and more flexible than VMs. The Docker culture and ecosystem also enhance Docker’s potential for aiding in reproducible computing.

Below, we’ll show you how to install Docker on an EC2 instance, use an existing Docker container, and build your own Docker container.

author: Titus Brown


date: Aug 24, 2015

Contents

- A hands-on introduction to Docker
 - Introduction and goals
 - Getting started with Docker
 - Install Docker
 - Run Docker
 - Building images
 - Build a Docker image for MEGAHIT, interactively
 - Connecting a Docker container to some external data
 - Running it all in one
 - Building an image with a Dockerfile
 - Summary points
 - Challenge exercises
 - More reading

Getting started with Docker

Install Docker

Start up an EC2 instance running blank Ubuntu 14.04 (see [Getting started with Amazon EC2](#)).  v: 2015 ▼

Then, install Docker:

```
wget -q0- https://get.docker.com/ | sudo sh
```

(This will take about 5 minutes.)

Now, configure the default user ('ubuntu') to use Docker:

```
sudo usermod -aG docker ubuntu
```

and log out and log back in.

Run Docker

The following command will start up a blank Ubuntu 14.04 docker container:

```
docker run -it ubuntu:14.04
```

(If you get the message `Post http:///var/run/docker.sock/v1.20/containers/create: dial unix /var/run/docker.sock: permission denied.` then you need to log out and log back in.)

This command will spit out a fair bit of output - what it's doing (the first time you run it) is going out to [the docker hub](#) and downloading the Ubuntu 14.04 image to your EC2 instance.

You should end up at a prompt that looks like this: ``root@77e00211fef4:/# ```. Unlike your previous prompt (which on EC2 defaults to ending in a ```$ ```), *this* prompt has placed you inside your running Docker container.

This is a blank Ubuntu machine. You can play around in here a bit, if you want, to verify this.

Now, exit by typing:

```
exit
```

This will place you back at your EC2 prompt.

At this point your docker container is shut down and you are placed back at your EC2 prompt. Importantly, everything you did to the file system in the container is basically gone at this point - container contents don't persist unless you build an image. You can verify this by re-running the `docker run -it ubuntu:14.04`, adding a file, and then exiting; if you run the same image, the file system will be missing the added file.

See `docker commit` and the [Docker image docs](#) for more info on building images. or on to the next section.

 v: 2015 ▼

Building images

Build a Docker image for MEGAHIT, interactively

Let's build a Docker image for the MEGAHIT short-read assembler. (This is not the right way to do it in general, and we'll do it the Right Way with a Dockerfile, below.) This is all based on the *Assembling E. coli tutorial* <<http://angus.readthedocs.org/en/2015/assembling-ecoli.html>>-.

Start up a new container:

```
docker run -it ubuntu:14.04
```

Now, **in this new container**, run the following commands.

First, update the base software and install g++, make, git, and zlib:

```
apt-get update && apt-get install -y g++ make git zlib1g-dev python
```

Then check out and build megahit:

```
git clone https://github.com/voutcn/megahit.git /home/megahit  
cd /home/megahit && make
```

So, now we have megahit built! On our docker container! But we face two problems:

- that took a while, and we'd probably rather not do it again; but the docker container is going to go away!
- the docker container is disconnected from the underlying machine, so we have no way of accessing any data!

Let's take these two problems on separately - we'll start by saving the docker container to an image that we can re-run.

First, take note of the container ID; it's the string between the '@' and the ':' in the command prompt, so, for a command prompt like `root@fa1bf23148a5:`, it would be `fa1bf23148a5`. Then, exit the container:

```
exit
```

Now you'll be back at the `ubuntu` prompt. To commit a copy of the container above to a docker image, type:

```
docker commit -m "built megahit" fa1bf23148a5 megahit
```

 v: 2015 ▼

but replacing `fa1bf23148a5` with your docker container ID.

This creates a new image named 'megahit' that contains all of your changes above. If you run:

```
docker images
```

you should see something like:

REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
megahit	latest	749fd74397ed	29 seconds ago	427.5 MB
ubuntu	14.04	91e54dfb1179	3 days ago	188.4 MB

Now, to run the megahit image, you can type:

```
docker run -it megahit
```

and (inside the docker container, which will have a new container ID) you can run:

```
/home/megahit/megahit
```

to verify that you still have megahit installed and running. And voila! You've created your own container! (If you want to make this available to everyone, go check out [the Docker hub](#).)

Connecting a Docker container to some external data

Now that we can run and rerun the megahit-installed container to our heart's content, we still have to figure out how to connect it to some data. How??

Well, first, let's download some data to our EC2 instance.

Make sure you're at the `ubuntu@` prompt, by typing `exit` if necessary.

Now execute:

```
cd
mkdir data
cd data
wget http://public.ged.msu.edu.s3.amazonaws.com/ecoli_ref-5m-trim.se.fq.gz
wget http://public.ged.msu.edu.s3.amazonaws.com/ecoli_ref-5m-trim.pe.fq.gz
```

This downloads those two data files into your home directory - these are E. coli short-read data from Chitsaz et al., 2011.

Now, run your `megahit` image, and connect `/home/ubuntu/data/` to `/data` on the image:

```
docker run -v /home/ubuntu/data:/data \
-it megahit
```

 v: 2015 ▼

This will “mount” your data from `/home/ubuntu/data` on the Docker container, and connect it to the `/data` directory in your container. Type:

```
ls /data
```

to verify that you see these files.

Now, let's assemble!

```
/home/megahit/megahit --12 /data/*.pe.fq.gz \  
-r /data/*.se.fq.gz \  
-o /data/ecoli -t 4
```

Now, exit your docker container with `exit` and look at your data directory:

```
ls /home/ubuntu/data
```

You should see the `/home/ubuntu/data/ecoli` directory with the assembly in it:

```
ls /home/ubuntu/data/ecoli
```

Running it all in one

You might think, “hey, wouldn't it be nice to be able to run all of this in one command, rather than starting a docker container and then running it from the command line in there?” Yep. Run this:

```
docker run -v /home/ubuntu/data:/data \  
-it megahit \  
sh -c '/home/megahit/megahit --12 /data/*.pe.fq.gz \  
-r /data/*.se.fq.gz \  
-o /data/ecoli -t 4'
```

Basically, everything after the image name gets passed directly into docker to be executed. You have to use the `'sh -c'` stuff because otherwise `/data/*.se.fq.gz` gets interpreted on your EC2 machine and not on your Docker image.

But... this is kind of long and annoying. Wouldn't it be nice to have this in a shell script? Yes, yes, it would. Let's put it in a shell script in the `'data'` directory, and then run *that*.

First, put the command in a shell script:

```
cd /home/ubuntu/data  
cat <<EOF > do-assemble.sh  
#!/bin/bash  
rm -fr /data/ecoli  
/home/megahit/megahit --12 /data/*.pe.fq.gz \  
-r /data/*.se.fq.gz \  
-o /data/ecoli -t 4
```

 v: 2015 ▼

```
chmod +x do-assemble.sh
```

and then run the shell script inside of Docker:

```
docker run -v /home/ubuntu/data:/data \
-it megahit /data/do-assemble.sh
```

and voila!

One thing to note here is that we've placed the `do-assemble.sh` script on the EC2 machine, rather than in the Docker container. You can do it either way, but in this case it was more convenient to do it this way because we'd already created the container and I didn't want to have to create a new one. The only change needed is to put the script in `/home` on the docker image, instead of `/data`.

Building an image with a Dockerfile

The image above was constructed by running a bunch of commands. Wouldn't it be nice if we could give Docker a bunch of commands and tell *it* to build an image *for us*?

You can do that with a Dockerfile, which is the Right Way to build an image.

Let's encode the commands above in a Dockerfile:

```
mkdir /home/ubuntu/make_megahit
cd /home/ubuntu/make_megahit
cat <<EOF > Dockerfile
FROM ubuntu:14.04
RUN apt-get update
RUN apt-get install -y g++ make git zlib1g-dev python
RUN git clone https://github.com/voutcn/megahit.git /home/megahit
RUN cd /home/megahit && make
CMD /data/do-assemble.sh
EOF
```

Let's look at this Dockerfile before running it:

```
cat Dockerfile
```

The 'FROM' command tells Docker what container to load; the 'RUN' commands tell Docker what to execute (and then save the results from); and the *CMD* specifies the script entry point - a command that is run if no other command is given.

Let's build a Docker image from this and see what happens!

```
docker build -t megahit .
```

 v: 2015 ▼

(This will take a few minutes.)

Once it's built, you can now run it like so:

```
docker run -v /home/ubuntu/data:/data -it megahit
```

...and voila!

If you wanted to make this broadly available, the next steps would be to log into the Docker hub and push it; I did so with these commands: `docker login`, `docker build -t titus/megahit .`, and `docker push titus/megahit`.

You can run *my* version of all of this with:

```
docker run -v /home/ubuntu/data:/data -it titus/megahit
```

and – here's the super neat thing – you don't need to repeat any of the above, other than installing Docker itself and downloading the data!

Summary points

- Docker provides a nice way to bundle multiple packages of software together, for both yourself and for others to run.
- Docker gives you a good way to isolate what you're running from the data you're running it on.
- The Dockerfile enhances reproducibility by giving explicit instructions for what to install, rather than simply bundling it all in a binary.

Challenge exercises

- Create a new image `megahit2` where the `do-assemble.sh` script created above is saved in `/home` on the image itself, rather than in `/data`.
- Create a container that has both MEGAHIT and Quast installed; see [this page](#) for Quast install instructions.
- Modify the Docker run script to also run Quast on the MEGAHIT assembly.
- Install docker on your local computer, and run the 'titus/megahit' image there.

More reading

Docker has a lot of docs.

Docker was used to make a [GigaScience paper completely reproducible](#). (I've written about this idea too.)

Binary containers can be bad for science.

 v: 2015 ▼

Dealing with data is [still complicated](#), but the landscape is changing fast.

[The impact of Docker containers on the performance of genomic pipelines](#), Di Tommaso et al., 2015 (PeerJ preprint).

LICENSE: This documentation and all textual/graphic site content is licensed under the [Creative Commons - 0 License \(CC0\)](#) -- [fork @ github](#). Presentations (PPT/PDF) and PDFs are the property of their respective owners and are under the terms indicated within the presentation.

[comments powered by Disqus](#)