Cecilia Noecker
June 2017

Docker basics

Docker is a program that allows you to build and distribute images, which are collections of software and data that run independently of other processes on a computer. This means you can build an image with many programs and students can simply download and run the image without having to install all the programs one by one on their own operating system. A running instance of an image is called a container.
More here: https://docs.docker.com/engine/docker-overview/#the-docker-platform

The Dockerfile

The instructions to build the image we used are found in the Dockerfile. The Dockerfile requires using some specified Docker commands (in capital letters), but the "RUN" command can be used to do basically any command line operation. Each line of the Dockerfile will add a "layer" to the image, and if you change anything in it after building, the new build will only rebuild layers impacted by the change. You can rebuild the image from scratch by running "docker build -t tagname ." in the directory where the Dockerfile is located, where 'tagname' is the tag ID for that build.

The workshop Docker image

This Dockerfile starts with a base image of a minimal Ubuntu operating system, then installs a series of programs using either **apt** or by downloading with **wget** (some installations, like Java, were somewhat finicky to get working). The list of installed programs are:
Utilities: wget, man, perl, unzip, make, g++, python, Java and Java-related stuff
Genomics programs: bwa, samtools, bedtools, fastqc, sratoolkit (fastq-dump)
All of these programs run inside the container just as they would on any Ubuntu computer. Graphical displays are apparently tricky to set up, so we didn't use the container for anything beyond the command line.
The Dockerfile also species some reference data and a couple scripts to be copied in, and finally sets the starting working directory and shell language for the image.

The data

The reference data

The reference dataset we used is custom-curated by previous Borenstein lab members for a metagenomic annotation study, but it would be relatively straightforward to substitute in other reference information in a similar format. I pulled 3 pathogen genomes from a larger lab database of genomes of common gut microbiome species. The genome IDs are NCBI Taxon IDs, but the ORF IDs were generated in house. The .bed file of gene information does also include the corresponding NCBI Locus Tag for each ORF where available. The list of marker genes is from the same project, and was generated based on Cicarrelli 2006. I downloaded the list of virulence genes for two of the genomes from the Virulence Factors of Pathogenic Bacteria Database and mapped them to the custom gene IDs based on NCBI Locus Tags.

The samples

We had them analyze shotgun metagenomic data from a recent diagnostic metagenomic study: Joensen 2017

I picked this study because it included these nice tables showing for all samples the diagnosis using traditional clinical assays in comparison what was found using their metagenomic pipeline. However, most metagenomic samples are sequenced pretty heavily and that was true for many of these, so one possible future improvement would be to give them downsampled versions so that they don't take so long to download and analyze. Below is an "answer key" of the samples we had them use and what the actual diagnosis was for each. Notably, there is a lot of cross-mapping between these 3 genomes, so it's not necessarily straightforward to get to these answers from the simple in-class analysis. It probably would have been better to include something more unrelated like a *C. dificile* genome for a clearer difference.

Sample ID SRA	Sample ID paper	File size	Healthy/pathogen	Abx genes/other observations
ERR1543975	S_144	93Mb	Shigella (198214)	
ERR1543982	S_153	85Mb	Salmonella (220341) & E. coli	Salmonella really doesn't map that well even for the supposedly positive samples. Probably wrong strain.
ERR1543986	S_157	139.9 Mb	E. coli	Eae (commonly used E.coli virulence indicator, discussed in paper) pops up with high abundance
ERR1543950	S_115	131.4 Mb	Undiagnosed/ negative	
ERR1543952	S_119	230.3 Mb	Undiagnosed/ negative	
ERR1543953	S_120	279.4 Mb	Undiagnosed/ negative	
ERR1543973	S_142	103.7 Mb	E. coli (155864)	
ERR1543944	S_105	260.6 Mb	C. difficile	

The scripts

The scripts in the "scripts" directory are mainly copied and slightly modified from a Borenstein lab pipeline. The only one we ended up having students use is "get_geneCov.pl", which calculates read counts and coverage for each gene based on the output of a "bedtools intersect" command. It takes as an argument the number of base pairs of overlap required between a gene and a read for it to be included in the count (in the command we gave them this is supplied as 5bp). The output produced is a 4-column text file where the columns are gene ID, gene length, gene read count, and gene coverage.

Ideas/thoughts for next time
Mainly, simplify the activity and/or prepare them well - they needed more exposure to the general concept of command line tools and more practice with doing basic tasks like moving between directories.

Have them do some advance setup, either in quiz section or as homework. I think even the last 2 or 3 quiz sections could be spent working on this as a project, which might allow for them to be a little more independent with it as well.

They also probably needed more introduction into the concept of the study and a reminder of bacterial genetics (only 1 chromosome, no introns, etc)

IGV is not part of the container, but I think seeing the interactive visualization was important for them to better understand the goals of the alignment.

It would not be too difficult to keep the structure of the image and replace the data to run some other analysis with data from some other organism.