Dockerizing Your Tools for the CyVerse Discovery Environment

Background

It is common to build new analysis methods utilizing multiple programs, libraries, and modules, e.g., SAMtools or R with Bioconductor. Often you need very specific versions of the operating system and underlying programs, such as Ubuntu version 14.04, Bioconductor version 3.2, R version 3.2.2, and SAMtools 1.3, to correctly achieve desired results.

This delicate balance of dependencies is often called the "Software Dependency Hell". It adversely impacts reproducibility of analyses and makes it very challenging to share your programs, pipelines, and analysis methods with collaborators and users who do not have access to identical systems, i.e., the same versions of all underlying software and operating system.

These issues have made it challenging for users to integrate new applications and analysis methods in the Discovery Environment (DE), as the underlying execution platform could only support a limited number of versions of the same software, e.g., if your program expects to find the BWA program version 0.7.12 in /usr/local/bin/bwa, and another program expects 0.7.13, it was impossible for these to coexist without modifications to your code. With advances in container technology, these issues are now very easy to resolve, and support highly customized execution environments for every analysis. CyVerse is making use of Docker one of the popular containerization platforms.

What is Docker?

Docker, a type of virtualization for software distribution, has revolutionized the way in which scientific software and all dependencies can be packaged, distributed, and deployed. Docker makes the complex and time-consuming installation procedures needed for scientific software a one-time process. Because it enables platform-independent installation, easy versioning of software and redeployment, and reproducibility across environments and versions, Docker is an ideal candidate for the deployment of software on different compute environments (XSEDE, Amazon AWS, etc.). Learn more about Docker.

Glossary

- **Docker image:** Ready snapshot of a configured software application.
- **Docker container:** Instance of a Docker image.
- **Dockerfile:** Set of instructions/commands that are used to build the Docker image.
- **Docker engine**: Docker tool that simplifies the creation of Docker hosts on your computer.
- **CyVerse tools vs. apps:** In the DE, a *tool* is an executable or binary upon which an *app* (an interface of a tool) is used to run the analyses. Before a new app can be created, the appropriate version of the underlying *tool* must be installed in the DE. Use Docker to create a tool. After that tool is installed on the DE, you use it to create an app. Your app can be private or public.

Using Docker in the CyVerse DE

CyVerse has adopted Docker for integrating software apps that run in the CyVerse DE's Compute Cluster (Condor). The user creates a Dockerfile, which is used to build the Docker image containing the tool that will be used as an app in the DE.

Integrating a Dockerized tool into the DE enables users to begin creating apps built on the tool. Because Dockerized apps use fewer resources, their analyses process more quickly. Compared to the previous method for tool integration in the DE, this method streamlines the process and makes it more likely that the final DE app will function as the user intended. It also increases the likelihood that more complicated and difficult to install software can be used in the DE.

Helpful Links

On This Page:

Related Pages:

- Learn more about Docker
- Docker: instructions, best practices, cheat sheet
- Visualizing Docker containers and images
- GitHub repo for CyVerse DE
- Biobox
- F1000 channel for containers
- List of Dockerized bioinformatics app

About Docker containers

Docker engine: The Docker engine shown in Figure 1 runs on three different containers in the DE. This spreads out the resources so that updates to the tool are easier to do, making it easy for users to move data in and out of the Docker containers.

- The **data staging-in container** delivers the data on which you want to operate to its location in the Data Store.
- The **app container**, based on your integrated Dockerized tool, runs with the data visible to it as a union file system.
- The **data staging-out container** returns data from the analysis that uses the app to the Data Store.

Video tutorial

For a step-by-step tutorial on how to create Docker images, see this video webinar and the documentation page that accompanies it.

Steps for Dockerizing a tool in the DE

Prerequisites

Before you can use a Dockerized image, you must complete a few prerequisites:

1. **Install Docker and any other dependencies:**
    - Linux: The installation procedure involves the use of package containers, such as Curl, or the use of APT (Advanced Package Tool) and Yum repositories for your installation.
    - Mac OS X and Windows: Docker Toolbox is a quick and easy way to install and set up a Docker environment.
    - Virtual Machine: Docker can be installed in a virtual machine environment through Virtual Box.
    - GUI: Docker containers can also be run through a simple and powerful graphical user interface such as Kitematic.
2. **Ensure the tool you want to Dockerize is available from a reliable source:** A reliable source is a website that hosts files/binaries necessary for tool installation and which can relied upon for future builds (for example, ubuntu apt repos, redhat/centos yum repos). Unreliable sources include a public Dropbox link, a lab computer, a personal computer, etc.
    - If the tool and all its executables are available on reliable sources, use that URL for Dockerization of the tool.
    - If installation files cannot be retrieved from a reliable source, they should be version controlled with the Dockerfile.

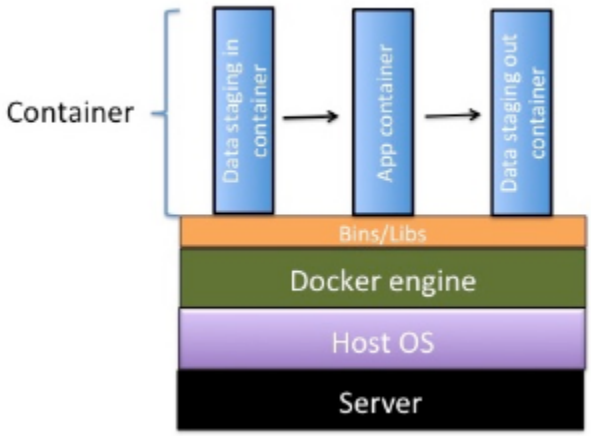

Figure 1. CyVerse container architecture

Steps

The following steps serve as a guide for Dockerizing a tool in the DE (see Figure 2 below).

Step 1: Check if the tool and correct version are already installed in the DE.

Before requesting installation of a new tool or new version of an existing tool, check the list of all tools in the DE to make sure that the tool and version you want is not already available. Because all tools now run inside a Docker containers and are installed as Docker images in the DE, if the tool is not already available, you must first create a Dockerfile for the tool before requesting its installation in the DE.

Step 2: Create the Dockerfile following best practices.

Construction of each tool's image needs to be reviewed in order to ensure that its construction is **as repeatable as possible**, as well as ensuring that execution of each tool is **as repeatable and reliable as possible**. The Dockerfile satisfies this goal of documenting how a particular tool was installed and of making the tool construction repeatable. To do so, use the following guidelines:

- Adhere to the Docker community specific set of instructions.
- Additionally, here are few CyVerse Dockerfile best practices:
    - **Include all installation steps in the Dockerfile.** For

example, a Dockerfile should not copy and run a script that performs Ubuntu APT commands; instead, the APT commands should be in the Dockerfile.

- **Write the Dockerfile to fail-fast.** This means that if anything goes wrong, construction of the image will fail.
- **Encapsulate the tool execution** to avoid external dependencies unless by design (e.g., tool provides integration with external service).
- **Derive the tools from official Docker images.**
- **Specify an ENTRYPOINT in your Dockerfile.** The entry point will allow the docker image to be run similar to an executable. Alternatively, you can also provide an entry point later when adding your tool to the Discovery Environment.

Examples of good Dockerfiles

Two simple examples of good Dockerfiles are shown below. Additionally, take a look at the list of commonly used CyVerse base images in the FAQ section below, and the troubleshooting tips in the Docker cheat sheet.

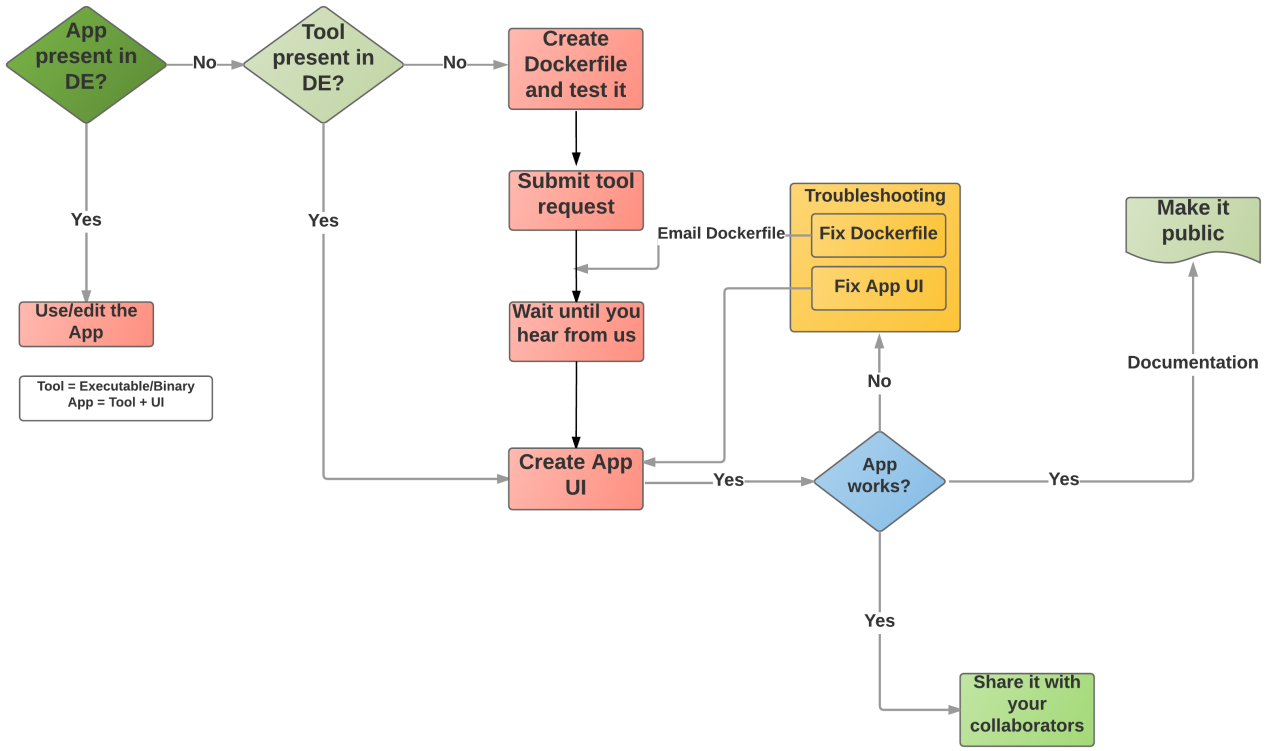

Figure 2. Flow diagram of Dockerization of tools in the DE

Sample Dockerfile 1: hisat2

Dockerfile for installing hisat2 in a Docker container based on the Ubuntu:14.04.03 image:

```
FROM ubuntu:14.04.3
MAINTAINER Eric Lyons
RUN apt-get update && apt-get install -y \
   build-essential \
   git \
   python
ENV BINPATH /usr/bin
ENV SRCPATH /usr/src
ENV HISAT2GIT https://github.com/infphilo/hisat2.git
ENV HISAT2PATH $SRCPATH/hisat2
RUN mkdir -p $SRCPATH
WORKDIR $SRCPATH
# Clone and checkout the 2.0.3-beta release version of the git repo
RUN git clone "$HISAT2GIT" \
   && cd $HISAT2PATH \
   && git checkout 3f8c81375700d4107fdfd1caeaec01b5719ae4b8
RUN  make -C $HISAT2PATH \
   && cp $HISAT2PATH/hisat2 $BINPATH \
   && cp $HISAT2PATH/hisat2-* $BINPATH
ENTRYPOINT ["/usr/bin/hisat2"]
```

Sample Dockerfile 2: NCBI SRA Submission pipeline

Dockerfile for installing NCBI SRA Submission in a Docker container based on a python:2.7 base image.

```
FROM python:2.7-slim
WORKDIR /root
COPY requirements.txt ./
RUN set -x \
    && apt-get update \
    && apt-get install -y gcc libxml2-dev libxslt1-dev lib32z1-dev
--no-install-recommends \
    && rm -rf /var/lib/apt/lists/* \
    && pip install -r requirements.txt \
    && apt-get purge -y --auto-remove gcc lib32z1-dev
# Download Aspera Connect client from http://downloads.asperasoft.com/connect2/
ADD
http://download.asperasoft.com/download/sw/connect/3.5/aspera-connect-3.5.1.92523-linu
x-64.sh aspera-connect-install.sh
# Install Aspera Connect client to ~/.aspera
RUN chmod 755 aspera-connect-install.sh
RUN ./aspera-connect-install.sh \
    && rm aspera-connect-install.sh
COPY ncbi_sra_submit.py metadata_client.py ncbi_sra_report_download.py ./
VOLUME [ "/root/config", "/root/templates", "/root/schemas" ]
ENTRYPOINT [ "python", "/root/ncbi_sra_submit.py" ]
CMD [ "--help" ]
```

Why are these good Dockerfiles? Both Dockerfiles adhered to best practices by using official Docker images and installing the tool from a reliable source.

Example of a poor Dockerfile

Here is a hypothetical example of how a poor Dockerfile looks like. Let's assume we have a tool named FooBar.

```
FROM test/Ubuntu
RUN apt-get upgrade
RUN apt-get update
RUN apt-get install -y emboss python
RUN wget https://someuniversity.edu/test_tool/test.py
```

Why is this a poor Dockerfile? It used an untested/unofficial Docker image. It contains a step that fetches binaries from some server at a university.

In addition, no fail-fast was written to the Dockerfile. At some point in time, this server was taken offline, despite assurances that it would remain online forever. This causes the image build to fail because its binaries cannot be retrieved, and no errors were written to the Dockerfile.

Step 3: Build and test the Dockerized tool.

Before you request the installation of the Dockerized tool:

- **Build a new image:** Use docker build command as shown below to test your new image:

```
docker build -t <your/docker-image> .
```

- **Test the new image:** Use docker run command as shown below to test your new image:

```
docker run --rm <your/docker-image> <entrypoint arguments>
```

- **Test the inputs and outputs.** Your tool will most likely require inputs and produce outputs. A docker run command like the following example should be used to ensure your tool will run inside the DE. If the tool's image was built from a Dockerfile with an ENTRYPOINT a nd tagged your/docker-image, then place some test input files into a scratch directory (~/my-scratch-dir in this example), and run a command like the following:

```
docker run --rm -v ~/my-scratch-dir:/working-dir -w /working-dir your/docker-image
user-input-1 user-input-2 ...
```

The -v option mounts the scratch directory on the host machine into that /working-dir directory inside the container, and the -w option sets the working directory inside the container to that same /working-dir directory.

- **Note**: The DE will run a tool's Docker image, using a combination of the docker run flags -w and -v in order to mount the Condor node's working directory to some arbitrary working directory inside the container. All inputs will be placed inside this working directory, and the DE expects the tool to generate outputs under this working directory as well.
- Additionally, all arguments entered by the user in the DE's app interface will be passed as command-line arguments to the docker run co mmand following the your/docker-image image name. From the example command above, these would be user-input-1 user-input-2 ... E xceptions are the Environment Variable fields, which will be passed to the docker run command as `-e` flags.
- Also note that Reference Genome/Sequence/Annotation input arguments are passed to the tool differently from other arguments, so if your tool requires these types of inputs, please include that requirement on the form when you request installation.

If the tool's container produced outputs in that host's scratch directory, then this tool is ready for the next step (Request installation of the Dockerized tool in the DE).

Step 4: Request installation of the Dockerized tool.

Read the main steps in the DE user manual for submitting your request for installation of the new tool (executable) in the DE. Once the tool is installed, you will receive an email notification.

As noted in the previous step, if your tool requires Reference Genome/Sequence/Annotation input arguments, note that in your request.

Step 5: Create and save the new app interface in the DE.

Once the Dockerized tool is installed, go to the CyVerse wiki to learn how to design a new interface, preview it, and save the new app within the DE.

Step 6: Test your app in the DE.

After creating the new app according to your design, test your app in the DE to make sure it works properly.

- If your app works the way you expect it to, skip to Optional steps.
- If your app still needs a bit of work and If the changes you make affect your Dockerfile (for example, it uses a newer version of the software and subsequently new dependencies are created), go back to step 2 and repeat.

Having problems?
If you've tried and tried and still can't Dockerize your tool, contact CyVerse Support for help by clicking on the chat icon in the lower right corner of the User Portal or Discovery Environment.

Optional steps

Complete the additional optional steps as needed for your tool.

Sharing (publishing) your app in the DE

Once the app is working to your satisfaction and you have published it, it is immediately available in your personal workspace in the DE and you can begin using it to run your own analyses. If you want to share it with other users, you can either keep it in your personal workspace and share it with selected users (including defining their permissions in the app), or share it with the public. For more information, see Sharing your App or Workflow and Editing the User Manual.

Editing an unshared app

If you have not yet shared the app with the public (that is, it is still listed in your Apps under development folder in your personal workspace), you can still edit the file and create a new Dockerfile. Then email CyVerse Support to replace the Dockerfile.

Deleting/editing a publicly shared app

Once you have shared an app with the public, it cannot be deleted because of CyVerse's commitment to supporting reproducible science. Because public apps cannot be edited once they have been made public, if you need to change the app you must create a new version of the app and then create a new Dockerfile. Learn more about editing apps.

Requesting a different category for your app

When you share your app with the public, you will indicate the category or categories into which you think it should be placed. To request that your app be moved or added to a different or additional category, email CyVerse Support with the app name, current category or categories, and desired target category or categories.

Examples of Dockerization of tools in the DE

Before you Dockerize a tool, it is important that you understand program dependencies (check the program documentation/manual thoroughly).

Example 1: Dockerizing a simple bioinformatics tool - Kallisto

The Kallisto Docker image was built on an Ubuntu-64 bit Virtual Machine using Virtual Box.

1. Install Docker:

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

2. Create a Dockerfile:

```
FROM ubuntu:14.04.3
MAINTAINER Kapeel Chougule
LABEL Description="This image is used for running Kallisto RNA seq quantification
tool"
# Install dependencies
RUN apt-get update && apt-get install -y build-essential cmake zlib1g-dev libhdf5-dev
# Install git and clone the kallisto tool
RUN apt-get install --yes git
RUN git clone https://github.com/pachterlab/kallisto.git \
&& cd kallisto \
&& git checkout 5c5ee8a45d6afce65adf4ab18048b40d527fcf5c \
&& mkdir build \
&& cd build \
&& cmake .. \
&& make \
&& make install
ENTRYPOINT ["kallisto"]
```

3. Build a Docker image:

```
Docker build -t"=ubuntu/kallisto" .
```

4. Test the built Kallisto image:

```
docker run --rm -v=/Users/kchougul/Downloads:/kallisto_data -w kallisto_data
ubuntu/kallisto kallisto index -i transcripts.idx transcripts.fasta.gz
```

5. Tag the built Kallisto image:

```
docker tag ubuntu/kallisto:latest kapeel/kallisto:latest
```

6. Push the Kallisto image to Dockerhub (optional):

```
docker login
docker push kapeel/kallisto:latest
```

Example 2: Dockerizing a bioinformatics tool - ParaAT

The ParaAT Docker image was built on Mac OS X using the Docker toolkit (quick start terminal).

1. Install Docker Toolbox for Mac OS X (see step 1 in Example 1).

2. Create a paraAT git repo on GitHub.

3 Create a paraAT git repo locally on your computer:

```
git init paraAT
```

4. Clone the paraAT git repo to local:

```
git clone https://github.com/jdebarry/paraat.git
```

5. Download the paraAT files, and then add and commit them to the local paraAT GitHub repo folder:

```
git add . && git commit -m "adding paraat files"
```

6. Push the local paraAT repo to the remote repo:

```
git push -u origin master
```

7. Create a Dockerfile:

```
FROM ubuntu:14.04
MAINTAINER Jeremy DeBarry   jdebarry@cyverse.org
#Get paraat.pl and epal2nal.pl code and add into $PATH to make it executable anywhere,
then make it executable
ADD https://raw.githubusercontent.com/jdebarry/paraat/master/ParaAT2.0/ParaAT.pl
/usr/bin/
ADD https://raw.githubusercontent.com/jdebarry/paraat/master/ParaAT2.0/Epal2nal.pl
/usr/bin/
RUN [ "chmod", "+x",  "/usr/bin/ParaAT.pl" ]
RUN [ "chmod", "+x",  "/usr/bin/Epal2nal.pl" ]

#Installing aligners, renaming clustalw executable so paraat.pl will use it - this is
a bandaid but it works
RUN echo "deb http://archive.ubuntu.com/ubuntu trusty multiverse" >>
/etc/apt/sources.list
RUN DEBIAN_FRONTEND=noninteractive apt-get -qq update
RUN DEBIAN_FRONTEND=noninteractive apt-get install --no-install-recommends -y clustalw
RUN DEBIAN_FRONTEND=noninteractive apt-get install --no-install-recommends -y mafft
RUN DEBIAN_FRONTEND=noninteractive apt-get install --no-install-recommends -y muscle
RUN DEBIAN_FRONTEND=noninteractive apt-get install --no-install-recommends -y t-coffee
RUN [ "mv", "/usr/bin/clustalw" , "usr/bin/clustalw2" ]
ENTRYPOINT ["ParaAT.pl"]
```

8. Build the paraAT image:

```
docker build -t paraat .
```

9. Test the paraAT image:

```
docker run --rm -v=/Users/jdebarry/Dropbox/Docker/paraat/Development/input/:/data
paraat -h /data/test.homologs -n /data/test.cds -a /data/test.pep -p proc -o
```

FAQs – Dockerizing

- **How do you troubleshoot the build process and local testing of a Dockerfile?**

Start with Google. Also check out the websites Stack Overflow or Biostars, as well as the Docker cheat sheet for help troubleshooting the build and Dockerfile-related issues. For local testing of Dockerfiles, please refer to step 3.

- ## What is the base image required to build a Docker container?

  The base image depends on the tool or script. You can pursue available images by searching Docker Hub for the domain (e.g., “biology”, “science”) and read the documentation for specific images.

  Most frequently used CyVerse base images:

  - ubuntu:12.04
  - ubuntu:14.04

  Other CyVerse base images:

  - perl:latest
  - r-base
  - ubuntu
  - ubuntu:latest
  - python:2.7
  - centos:7
  - biobakery/base
  - java:7
  - centos:6.6
  - gcc:5.1
  - debian:latest

- ## Is licensing information required for creating Docker images?

  Licensing is a definite factor to consider in Docker image creation. Many programs want to count downloads or have other restraints, and creating a Dockerfile means you are essentially automating access to the software. Due diligence is required to ensure that the onus is on the user and not on CyVerse (click below to view the example).

View sample licensing information

Copyright 2015. The Regents of the University of California (Regents). All Rights Reserved. Permission to use, copy, modify, and distribute this software and its documentation for educational and research not-for-profit purposes, without fee and without a signed licensing agreement, is hereby granted, provided that the above copyright notice, this paragraph and the following two paragraphs appear in all copies, modifications, and distributions. Contact The Office of Technology Licensing, UC Berkeley, 2150 Shattuck Avenue, Suite 510, Berkeley, CA 94720-1620, (510) 643-7201, for commercial licensing opportunities. Created by Nicolas Bray, Harold Pimentel, Pall Melsted and Lior Pachter, University of California, Berkeley IN NO EVENT SHALL REGENTS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF REGENTS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE. REGENTS SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE AND ACCOMPANYING DOCUMENTATION, IF ANY, PROVIDED HEREUNDER IS PROVIDED "AS IS". REGENTS HAS NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

- ## Where should I store the Dockerfiles and executables for an app?

  Dockerfile creators should store their Dockerfiles and dependencies (wrapper scripts, etc.) in a reliable source such as GitHub or Bitbucket. CyVerse has its own GitHub repository where sources and Dockerfiles are saved for tools installed in the DE.

- ## What are the hardware limitations for running a Docker container? How much RAM and disk space is allowed for a Docker app?

  The DE does not specifically cap the disk size for each Docker container running in Condor nodes, so we are using the default Docker cap of 10 GB per image. This limit applies only to the tool's Docker images and data containers, not to any inputs or outputs used when running the analysis (which are mounted from the node's working directory).

  Although a tool's Docker image may contain up to 10 GB of data if required, users should still strive to keep their Docker images small. This will ensure that jobs run faster for DE users when the tool's image needs to be pulled from our registry to a Condor node before running an analysis.

- ## Do I need to import reference genomes to Docker containers?

  The DE has a number of reference genomes uploaded and available for use. View the list here. If the genome you want to use is not listed, contact CyVerse Support to ask that it be added.

- ## Can HPC apps be Dockerized?

No, due to security issues, currently Docker is not available to apps that require HPC resources, i.e., apps integrated using the Agave API.

Future enhancements

Future enhancements include the ability to:

- Import Docker images from a private repository such as GitHub or Docker Hub, or from files, etc.
- Share an app or Docker container within groups.
- Bring your own compute for containers, attach the container to the CyVerse pool, and manage the list of users who can access the container.
- Run Docker containers on HPC systems.
- Perform network functions. Currently, this is not available for security reasons.