BioContainers Documentation

Yasset Perez-Riverol

Jan 26, 2023

Contents

CHAPTER 1

Contents

1.1 Introduction

1.1.1 Topics

- Learn basic concepts of containers, Docker and BioContainers
- Get hands to create, deploy and use a BioContainers
- Master the workflow for deploying and publishing a BioContainer in Quay.io or DockerHub

1.1.2 Getting help

Our education portal is work in progress. So if you encounter a logical inconsistency or just want to ask a question - don't hesitate to contact us.

For technical questions related to containers (broken containers, container requests) Containers Issues are preferred.

If you find a typo or want to help us to make this tutorial even better, you are invited to click on "Edit on GitHub".

1.1.3 Concepts in this tutorial

- Conda, Containers, Docker, rkt
- Docker containers creation and deployment
- Minimal rules -> conventions and gold standards

Note: The BioContainers project is truly open source, even this tutorial.

1.2 What is a Container?

Basically a software container is used to encapsulate a software component and the corresponding dependencies. You don't start from scratch or install all the software over and over. It contains **independently deployable bits of code** that can be used to build and run agile applications. It could be anything from a FASTA parser, a tree algorithm or a simple visualization module.

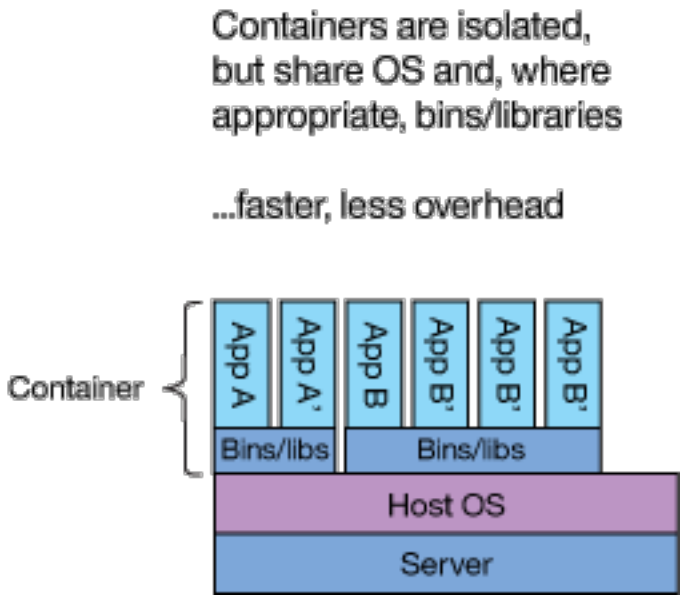
- Containers encapsulate discrete components of application logic provisioned only with the minimal resources needed to do their job.
- Containers are easily packaged, lightweight, and designed to run anywhere.

Let's install a container quickly:

```
$ docker pull biocontainers/dia-umpire:v1.4256_cv2
```

Note: This simple command installs dia-umpire and all its dependencies

Containers are built from existing operating systems. They are different from virtual machines because they don't possess an entire guest OS inside; instead, containers are built using optimized system libraries and use the host OS memory management and process controls. Containers normally are centralized around a specific software and you can make them executable by instantiating images from them.



1.2.1 Why should I use a container?

Most of the time when a bioinformatics analysis is performed, several bioinformatics tools and software should be installed and configured. This process can take several hours and demands a lot of effort including the installation of multiple dependencies and tools. BioContainers provide ready-to-use packages and tools that can be easily deployed and used on local machines, HPC and cloud architectures.

In the next video you can check out what you can achieve by using docker containers:

1.2.2 Container technologies

BioContainers has been built around three main technologies: **Conda**, **Docker** and **Singularity**. The BioContainers Community releases for every bioinformatics software containers in these three technologies or flavours.

Note: We do not provide detailed documentation about these three technologies because that can be found on their corresponding web sites, although we may explain some concepts important for understanding BioContainers as needed.

Conda

Conda is an open source package management system and environment management system that runs on Windows, macOS and Linux. Conda quickly installs, runs and updates packages and their dependencies. It was created for Python programs, but it can package and distribute software for any language.

Docker

Docker provides container software that is ideal for developers and teams looking to get started and experimenting with container-based applications. It provides an integrated container-native development experience; and access to the largest library of community and certified Linux and Windows Server content from Docker Hub.

Singularity

Singularity enables users to have full control of their environment. Singularity containers can be used to package entire scientific workflows, software and libraries, and even data. This means that you don't have to ask your cluster admin to install anything for you - you can put it in a Singularity container and run.

1.3 BioContainers

BioContainers is a community-driven project that provides the infrastructure and basic guidelines to create, manage and distribute bioinformatics packages (e.g conda) and containers (e.g docker, singularity). BioContainers is based on the popular frameworks Conda, Docker and Singularity.

1.3.1 BioContainers Goals

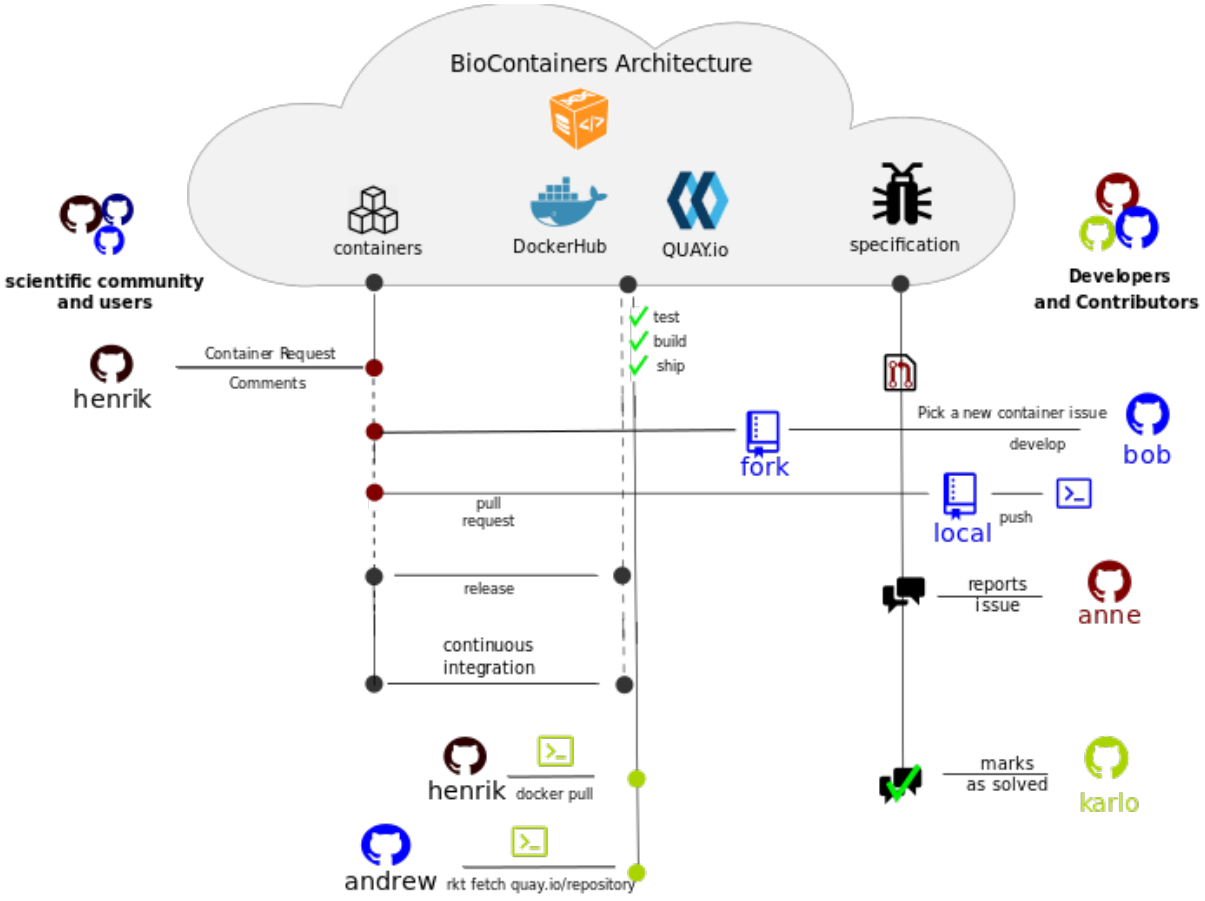
- Provide a base specification and infrastructure to develop, build and deploy new bioinformatics software including the source and examples.
- Provide a series of containers ready to be used by the bioinformatics community (https://biocontainers.pro/#/registry).
- Provide guidelines and help on how to create reproducible pipelines and workflows using bioinformatics containers.
- Coordinate and integrate developers and bioinformaticians to produce best practices of documentation and software development.

1.3.2 Main components of BioContainers:

- Docker Containers provides a list of *Dockerfile recipes* to automatically build containers in BioContainers.
- Conda based Containers provides a list of *Conda recipes* to automatically build **first a conda package** and then a docker container.
- Biocontainers Registry is a hosted registry of all BioContainers images that are ready to be used (read more here biocontainersregistry).
- Specifications defines a set of guidelines and rules to contribute with BioContainers.

1.3.3 BioContainers Community Architecture

BioContainers is a community-driven project that allows bioinformaticians/developers to request, build and deploy bioinformatics containers. The following figure present the general BioContainers Architecture:



How to Request a Container

Users can request a container by opening an issue in the containers repository , In the previous workflow this is `the first step` performed by user `henrik`. The issue should contains the name of the software, the url of the code or binary to be packaged and information about the software see BioContainers specification. When the container is deployed and fully functional, the issue will be closed by the developer or the contributor to BioContainers.

Note: Before requesting a Container you should check the BioContainers Registry to make sure your requested tool does not exist already (read more about the registry in: biocontainersregistry).

Hint: Importantly, the BioContainers community has implemented a "labeled legend" to tag each issue in the containers repository that should be used properly for on each issue. For example, for new containers the label **Container Request** should be used.

Use a Docker BioContainer.

When a container is deployed and the developer closes the issue in GitHub, the user `henrik` receives a notification that the container is ready. Then, the user can use `docker` command to pull or fetch the corresponding container.

```
$ docker run biocontainers/blast:2.2.31
```

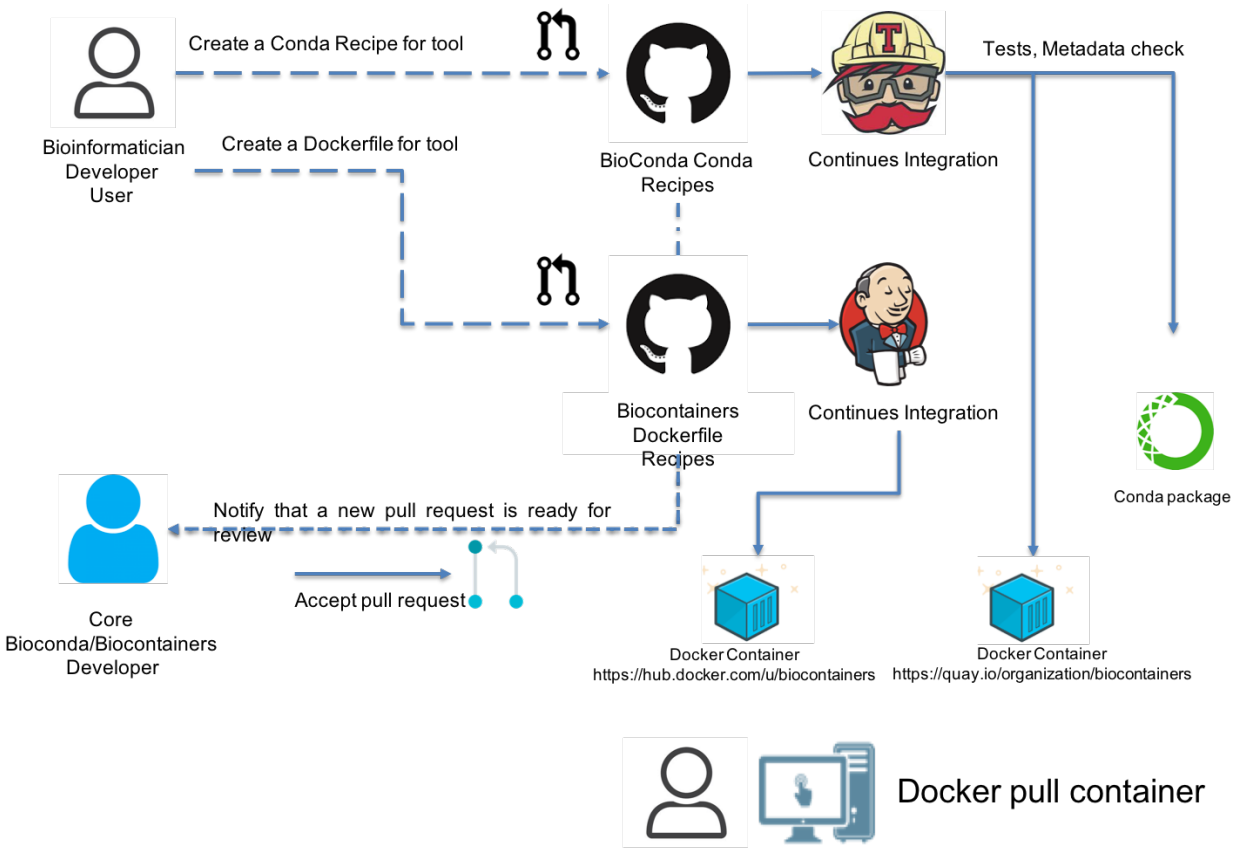
Note: You can read other sections about *Integration with BioConda* and singularitycontainers

Reporting a problem with a container

If the user finds a problem with a container an issue should be opened in the container repository, the user should use the **broken tag** (see tags). Developers of the project will pick-up the issue and deploy a new version of the container. A message will be delivered when the container has been fixed.

1.3.4 BioContainers build architecture

BioContainers build the docker containers from two different sources the Dockerfile or Conda recipes . The BioContainers team first try to create a Conda recipe (if it is possible). If not possible, then a Dockerfile is created. The system automatically builds the container after the `pull request` is merged into Conda github or BioContainer github repos.



In order to be able to contribute to BioContainers you should be able to create a BioConda recipe or a Dockerfile recipe.

Create a BioConda recipe

Note: Before you start please read the Conda documentation and how to setup conda in your machine

In summary should follow these steps:

- Fork the BioConda recipes in GitHub
- Create your conda recipe (following this tutorial)
- Create a Pull Request in BioConda

After the PR gets merged, a Conda package gets created and the corresponding docker container get pushed into Quay.io Registry and the BioContainers Registry

Create a Dockerfile recipe

This is a standard template for creating a new Dockerfile for BioContainers:

Note: Please always follow the *Best Practices* to create a Dockerfile.

Here, an example Dockerfile for a Biocontainer:

```
################## BASE IMAGE ######################
FROM biocontainers/biocontainers:v1.0.0_cv4

################## METADATA ######################
LABEL base_image="biocontainers:v1.0.0_cv4"
LABEL version="3"
LABEL software="crux"
LABEL software.version="3.2"
LABEL about.summary="a software toolkit for tandem mass spectrometry analysis"
LABEL about.home="http://cruxtoolkit.sourceforge.net/"
LABEL about.documentation="http://cruxtoolkit.sourceforge.net/"
LABEL about.license_file="http://cruxtoolkit.sourceforge.net/"
LABEL about.license="SPDX:Apache-2.0"
LABEL extra.identifiers.biotools="crux"
LABEL about.tags="Proteomics"

################## MAINTAINER ######################
MAINTAINER Yasset Perez-Riverol <ypriverol@gmail.com>

################## INSTALLATION ######################

USER biodocker

RUN ZIP=crux-3.2.Linux.x86_64.zip && \
    wget https://github.com/BioContainers/containers/releases/download/Crux/$ZIP -O /
↪tmp/$ZIP && \
    unzip /tmp/$ZIP -d /home/biodocker/bin/ && \
    rm /tmp/$ZIP && \
    ln -sv /home/biodocker/bin/*/bin/* /home/biodocker/bin/

ENV PATH /home/biodocker/bin:$PATH

WORKDIR /data/
```

Every Dockerfile must have a metadata header with the following items:

- **Base Image**: All containers are based on a specific GNU/Linux system. There is no preference for a specific OS flavor but, to reduce disk usage, we recommend to use our own biocontainers/biocontainers image.
- **Dockerfile Version**: This is a single-number version system (ex: v1, v2, v3 . . .). Every change in the file must increase in 1.
- **Software**: The name of the software installed inside the container. This can be a little tricky because some software demands libraries or dependencies. In this case the idea is to describe the "principal" software of the container, or the reason for building it.
- **Software Version**: The version of the software installed.
- **Description**: Single line description of the tool.
- **Website**: URL(s) for the program developer.
- **Documentation**: URL(s) containing information about how to use the software.
- **License**: URL(s) containing Licensing information.
- **Tags**: Program tags: Genomics, Protemomics, Transcriptomics, Metabolomics, General.

Image Setting

The next element is the base image and any configuration to the system you are installing. In the example above the Base Image is defined as biocontainers/biocontainers which is based on ubuntu latest LTS (Long Term Support)

release and kept up to date with updates.

Signature

The File Author/ Maintainer signature. By default the Dockerfile only accepts one MAINTAINER tag. This will be the place the original author name. After updates are added to the file, the contributors name should appear in commented lines.

```
# Maintainer
MAINTAINER Yasset Perez-Riverol <ypriverol@gmail.com>
```

Installation

The installation area is where the instructions to build the software will be defined. Here is the correct place to put Dockerfile syntax and system commands.

```
USER biodocker

RUN ZIP=comet_binaries_2016012.zip && \
  wget https://github.com/BioDocker/software-archive/releases/download/Comet/$ZIP -O /
↪tmp/$ZIP && \
  unzip /tmp/$ZIP -d /home/biodocker/bin/Comet/ && \
  chmod -R 755 /home/biodocker/bin/Comet/* && \
  rm /tmp/$ZIP

RUN mv /home/biodocker/bin/Comet/comet_binaries_2016012/comet.2016012.linux.exe /home/
↪biodocker/bin/Comet/comet

ENV PATH /home/biodocker/bin/Comet:$PATH

WORKDIR /data/

CMD ["comet"]
```

Tips

- Commands should be merged with '&& ' whenever possible in order to create fewer intermediate images.
- A biodocker user has been created (id 1001) so that applications are not run as root.
- If possible, add the program to /usr/bin, otherwise, add to /home/biodocker/bin
- return to the regular USER
- change the WORKDIR to the data folder
- set the VOLUME to be exported (/data and /config are exported by default by the base image)
- EXPOSE ports (if necessary)

1.3.5 Get involved

Whether you want to make your own software available to others as a container, to just use them on your pipelines and analysis or just give opinions, you are most welcome. This is a community-driven project, that means everyone has a voice.

Here are some general ideas:

- Browse our list of containers
- Propose your own ideas or software

- Interact with others if you think there is something missing.

1.4 Getting started with Docker

1.4.1 Docker Configuration

Docker is the world's leading platform for software containerization. Docker includes multiple tools and components such as: docker, docker engine, docker hub.

Hint: Open your terminal of choice and check whether you have the command `docker` installed. You should be able to run the following command without any error.

Other commands:

```
$ docker --version
Docker version 1.12.0, build 8eab29e

$docker-compose --version
docker-compose version 1.8.0, build f3628c7

$docker-machine --version
docker-machine version 0.8.0, build b85aac1
```

If your machine is already set up, continue to the *Running first container*.

The following sections provided a short summary of how to install **docker**. If these steps do not work for you, please refer to the full documentation of docker. You can also get in contact with us using Specifications Issues.

Mac OSX

On Mac installing Docker can be done by installing the complete Docker Toolbox available in Mac and Windows. The Docker Toolbox contains the Docker Engine, Compose, Machine, etc.

Note: Mac Docker ToolBox can be download from this page. The installation provides Docker Engine, Docker CLI client, Docker Compose, and Docker Machine.

Linux

Installing Docker in Linux can be done using the specific Linux distribution. Some of the supported distributions are: Arch Linux, CentOS, CRUX Linux, Debian, Fedora, Gentoo, Oracle Linux, Red Hat Enterprise Linux, openSUSE and SUSE Linux Enterprise, Ubuntu

Windows

Docker for Windows is our newest offering for PCs. It runs as a native Windows application and uses Hyper-V to virtualize the Docker Engine environment and Linux kernel-specific features for the Docker daemon.

Note: Please read through these topics on how to get started. To give us your feedback on your experience with the app and report bugs or problems, log in to Docker for Windows forum.

1.4.2 How does a Docker image work?

Docker images are read-only templates from which Docker containers are launched. Each image consists of a series of layers. Docker makes use of union file systems to combine these layers into a single image. Union file systems allow files and directories of separate file systems, known as branches, to be transparently overlaid, forming a single coherent file system.

One of the reasons Docker is so lightweight is because of these layers. When you change a Docker image—for example, update an application to a new version— a new layer gets built. Thus, rather than replacing the whole image or entirely rebuilding, as you may do with a virtual machine, only that layer is added or updated. Now you don't need to distribute a whole new image, just the update, making distributing Docker images faster and simpler.

Every image starts from a **Base image** , for example ubuntu, a base Ubuntu image, or fedora, a base Fedora image. You can also use images of your own as the basis for a new image.

Docker images are then built from these base images using a simple, descriptive set of steps we call instructions. Each instruction creates a new layer in our image. Instructions include actions like:

- Run a command
- Add a file or directory
- Create an environment variable
- What process to run when launching a container from this image

These instructions are stored in a file called a Dockerfile. A **Dockerfile** (see example of a BioContainers Dockerfile biocontainers) is a text based script that contains instructions and commands for building the image from the base image. Docker reads this Dockerfile when you request a build of an image, executes the instructions, and returns a final image.

1.4.3 Docker input and output files

In many cases the software you are using requires an input or an output file to work. This can be achieved using a container image, but it requires that you change the way how you execute the docker daemon. The first step is to have a specific folder in your computer to serve as a shared folder between your system and the container. In this case let's place a file called prot.fa inside a folder called `/home/user/docker/`. The `/home/user/docker/` folder will be mapped inside the container, that way this can server as a gateway for files, both the input and the output files will be placed there.

After setting the folder and necessary files inside we can execute the image we want. In the example bellow we are running an image built from an example container:

Our local path `/home/user/docker/` will map inside the container into a folder called `/data/` and the software will have access to the file. After the processing is done the result file will be saved on the same place and we can have it outside the container in the same folder. In this scenario, the only difference is the use of the parameter `-v` when running the image, this parameter tells the docker daemon that when are mapping something from the host system internally.

1.4.4 References

- Understand Docker Architecture
- Docker Tutorial - What is Docker & Docker Containers, Images, etc?
- Docker Ecosystem

1.5 Running first container

More Examples

It can make your life easier if you want to explore individual examples:

- *BioContainers in Examples*

We will run the first example with BLAST. The Basic Local Alignment Search Tool for comparing biological sequence information, such as the amino-acid sequences of proteins or the nucleotides of DNA sequences. We chose BLAST because it frequently used in bioinformatics.

The first step is to `pull` the software from Biocontainers registry:

```
$ docker pull biocontainers/blast:2.2.31
```

Note: This is the magic of containers; the software is distributed with all the dependencies and shared OS needed to run.

Docker allows applications to be isolated into containers with instructions for exactly what they need to survive and these instructions can be easily ported from machine to machine.

1.5.1 Running BLAST

```
$ docker run biocontainers/blast:2.2.31 blastp -help
```

This will print the help page for `blastp` tool. The first part of the command `docker run biocontainers/blast` enables docker to identify the correct container in your local registry. The second part `blastp -help` is the command that you want to use in the container.

Note: If you want to *list all the containers/images <https://docs.docker.com/engine/reference/commandline/images/>* you have pulled from public repositories and are available in your local machine, you can use the following command: `docker images`.

For this example let's try something practical, suppose that we are molecular biologists studying prion proteins, and we want to find out if the zebrafish, a model organism, has a prion protein similar to the human form.

1) Downloading the human prion sequence, We can grab the human prion FASTA sequence from UniProt:

```
$ docker run biocontainers/blast:2.2.31 curl https://www.uniprot.org/
↪uniprot/P04156.fasta >> P04156.fasta
```

Note: Some biocontainer base images contain multiple linux command that are useful for bioinformatics like `curl`, `wget`. You should note that not all the containers contains those additional tools.

2) Downloading the zebrafish database

Now, let's download and unpack our database, from NCBI

```
$ mkdir host-data
$ docker run -v `pwd`/host-data/:/data/ biocontainers/blast:2.2.31 curl -O␣
↪ftp://ftp.ncbi.nih.gov/refseq/D_rerio/mRNA_Prot/zebrafish.1.protein.faa.gz
$ docker run -v `pwd`/host-data/:/data/ biocontainers/blast:2.2.31 gunzip␣
↪zebrafish.1.protein.faa.gz
```

Note: The docker command can be run with the option `-v`; this will bind a local volume (in the example path host-data within the current working directory) into a container volume /data/. You can read more about here . In the example every operation performed in `/data/` will be stored in the bind directory.

3) Preparing the database

We need to prepare the zebrafish database with `makeblastdb` for the search, but first we need to make our files available inside the containers. The docker daemon has a parameter called volume (-v), it allows us to map a folder from our operating system inside the container, that way all files in that folder will be visible inside the container, and the BLAST results will also be available to us, outside the container. In the example below, I'm mapping the folder /Users/yperez/workplace (my computer) into /data/ (the container). When running the command on your computer, you should use the correct paths for your files.

```
$ docker run -v `pwd`/host-data/:/data/ biocontainers/blast:2.2.31␣
↪makeblastdb -in zebrafish.1.protein.faa -dbtype prot
```

The program's log will be displayed on the terminal, indicating if the program finished correctly. Also, you will see some new files in your local folder, those are part of the BLAST database.

Download a query sequence from the UniProt Knowledgebase (UniProtKB).

```
$ docker run biocontainers/blast:2.2.31 curl https://www.uniprot.org/uniprot/
↪P04156.fasta >> host-data/P04156.fasta
```

Now, that you know how to run a container with all the tricks, let's go for the final alignments:

```
$ docker run -v `pwd`/host-data/:/data/ biocontainers/blast:2.2.31 blastp -
↪query P04156.fasta -db zebrafish.1.protein.faa -out results.txt
```

The results will be saved in the results.txt file, then you can proceed to analyze the matches. By looking at the list of the best hits we can observe that zebrafish have a few predicted proteins matching the human prion with better scores than the predicted prion protein (score:33.9, e-value: 0.22). That's interesting isn't ?

Now that you have enough information to start comparing sequences using BLAST, you can move your analysis even further.

We hope that this short example can shed some light on how important and easy it is to run containerized software.

Run everything in one go

```
$ cd /Users/yperez/workplace   # Replace by your path
$ mkdir host-data
$ docker run biocontainers/blast:2.2.31 blastp -help
$ docker run -v `pwd`/host-data/ biocontainers/blast:2.2.31 curl -O ftp://
↪ftp.ncbi.nih.gov/refseq/D_rerio/mRNA_Prot/zebrafish.1.protein.faa.gz
$ docker run -v `pwd`/host-data/:/data/ biocontainers/blast:2.2.31 gunzip␣
↪zebrafish.1.protein.faa.gz
$ docker run -v `pwd`/host-data/:/data/ biocontainers/blast:2.2.31␣
↪makeblastdb -in zebrafish.1.protein.faa -dbtype prot
$ docker run biocontainers/blast:2.2.31 curl https://www.uniprot.org/uniprot/
↪P04156.fasta >> host-data/P04156.fasta
$ docker run -v `pwd`/host-data/:/data/ biocontainers/blast:2.2.31 blastp -
↪query P04156.fasta -db zebrafish.1.protein.faa -out results.txt
```

1.6 Integration with BioConda

Biocontainers build automatically docker containers for all BioConda package. For this reason a BioContainer can be created not only using Dockerfile biocontainers. This automatic containers has the benefit that the user can switch between `docker` and `conda environments` knowing that the tools are available in both environments.

Bioconda is a channel for the conda package manager specializing in bioinformatics software:

- a repository of recipes hosted on GitHub
- a build system that turns these recipes into conda packages
- a repository of more than 6000 bioinformatics packages ready to use with conda install

The conda package manager makes installing software a vastly more streamlined process. Conda is a combination of other package managers you may have encountered, such as `pip`, `CPAN`, `CRAN`, `Bioconductor`, `apt-get`, and `homebrew`. Conda is both language- and OS-agnostic, and can be used to install C/C++, Fortran, Go, R, Python, Java etc programs on Linux, Mac OSX, and Windows.

Note: Installing Conda: Bioconda requires the conda package manager to be installed. If you have an Anaconda Python installation, you already have it. Otherwise, the best way to install it is with the Miniconda package. The Python 3 version is recommended.

1.6.1 Defining a Conda package

The preferred way to build a conda package is to write a conda recipe and submit this it the BioConda community. As soon as your PR is merged and continues integration testing was successful, we will build you automatically a container and publish it at quay.io/biocontainers and BioContainers Registry.

The BioConda specification Contribution Guide define how to create a recipe. In summary, a BioConda recipe should contain the following parts:

- Source URL is stable (details)
- md5 or sha256 hash included for source download (details)
- Appropriate build number (details)
- .bat file for Windows removed (details)

- Remove unnecessary comments (details)
- Adequate tests included
- Files created by the recipe follow the FSH (details)
- License allows redistribution and license is indicated in meta.yaml
- Package does not already exist in the defaults, r, or conda-forge channels with some exceptions (details)
- Package is appropriate for bioconda
- If the recipe installs custom wrapper scripts, usage notes should be added to `extra -> notes` in the meta.yaml

Example Yaml for bowtie2:

```
package:
  name: unicycler
  version: 0.3.0b

build:
  number: 0
  skip: True # [py27]

source:
  fn: unicycler_0.3.0b.tar.gz
  url: https://github.com/rrwick/Unicycler/archive/
↪906a3e7f314c7843bf0b4edf917593fc10baee4f.tar.gz
  md5: 5f06d2bd8ef5065c8047421db8c7895f

requirements:
  build:
  - python
  - setuptools
  - gcc

  run:
  - python
  - libgcc
  - spades >=3.6.2
  - pilon
  - java-jdk
  - bowtie2
  - samtools >=1.0
  - blast
  - freebayes

test:
  commands:
    - unicycler -h
    - unicycler_align -h
    - unicycler_check -h
    - unicycler_polish -h

about:
  home: https://github.com/rrwick/Unicycler
  license: GPL-3.0
  license_file: LICENSE
  summary: 'Hybrid assembly pipeline for bacterial genomes'
```

When the recipe is ready, a Pull Request should be created (https://bioconda.github.io/contributor/workflow.html). Finally, the container is automatically created for the new BioConda Package.

1.6.2 Automatic build from conda recipes

We utilize mulled with involucro in an automatic way. This is for example used to convert all packages in `bioconda` into Linux Containers (Docker and rkt at the moment). We have developed small utilities around this technology stack which is currently included in galaxy-lib.

```
pip install galaxy-lib
```

Here is a short introduction:

Search for conda-based containers

This will search for containers in the biocontainers organisation.

Build all packages from bioconda from the last 24h

The BioConda community is building a container for every package they create with a command similar to this:

```
$ mulled-build-channel --channel bioconda --namespace biocontainers \
      --involucro-path ./involucro --recipes-dir ./bioconda-recipes --diff-hours 25␣
↪build
```

Building Docker containers for local Conda packages

Conda packages can be tested with creating a busybox-based container for this particular package in the following way. This also demonstrates how you can build a container locally and on-the-fly.

Note: We modified the samtools package to version 3.0 to make clear we are using a local version.

---

1) Build your recipe

```
$ conda build recipes/samtools
```

2) Index your local builds

```
$ conda index /home/bag/miniconda2/conda-bld/linux-64/
```

3) Build a container for your local package

```
$ mulled-build build-and-test 'samtools=3.0--0' \
      --extra-channel file://home/bag/miniconda2/conda-bld/ --test 'samtools --help'
```

The `--0` indicates the build version of the conda package. It is recommended to specify this number otherwise you will override already existing images. For Python Conda packages this extension might look like this `--py35_1`.

Build, test and push a conda-forge package to biocontainers

Note: You need to have write access to the biocontainers repository

You can build packages from other Conda channels as well, not only from BioConda. `pandoc` is available from the conda-forge channel and conda-forge is also enabled by default in Galaxy. To build `pandoc` and push it to BioContainers, you could do something along these lines:

```
$ mulled-build build-and-test 'pandoc=1.17.2--0' --test 'pandoc --help' -n␣
↪biocontainers
$ mulled-build push 'pandoc=1.17.2--0' --test 'pandoc --help' -n biocontainers
```

- Galaxy Conda documentation: ./conda_faq.rst
- IUC: https://wiki.galaxyproject.org/IUC
- Container annotation: https://github.com/galaxyproject/galaxy/blob/dev/test/functional/tools/catDocker.xml#L4
- BioContainers: https://github.com/biocontainers
- bioconda: https://github.com/bioconda/bioconda-recipes
- BioContainers Quay.io account: https://quay.io/organization/biocontainers
- galaxy-lib: https://github.com/galaxyproject/galaxy-lib

1.7 BioContainers Registry

Every container is deployed and permanent deposited in a public registry Docker Hub or Quay.io. These two registries allows developers and contributors to deploy their software to a public repository without needs to implement a new registry.

Note: The BioContainers community use currently for docker containers two docker registries (quay.io and docker hub). In both cases we use the same name space `biocontainers` to group all BioContainers: Docker Hub and Quay.io

1.7.1 Docker Hub

Docker Hub provides a centralized resource for container image discovery, distribution and change management, user and team collaboration, and workflow automation throughout the development pipeline.

1.7.2 Quay.io

Quay.io is a non-profit docker registry whose major advantage is that it supports `docker` and `rkt` technologies. In Quay.io each Container is presented with the maximum amount of useful information, including a full tags list, markdown based description and repository push and pull counts.

1.7.3 BioContainers Registry

BioContainers Registry UI provides the interface to and UI to **search**, **tag**, and **document** a BioContainers across all the registries.

The users can search containers by using the search box:

The containers registry allow the users to `sort` the containers by any of these properties:

- **Container Name**: Container Name
- **Description**: Description Provided by the developer of the container.
- **Real Name**: The corresponding registry + container name
- **Last Modified**: Last date where the container has been modified.
- **Starred/Start**: If the container has been starred in any of the repos.
- **Popularity**: How many times a container has been pull from a registry.
- **Registry Link**: the registry Link.

Note: Depending on the container provider, the registry will show the original `Dockerfile` provided by the developer or it will provide the `Yaml` (see example) configuration file for the `Conda based` containers.

1.8 BioContainers in Examples

In this section we provide a set of examples about how to use BioContainers in your analysis. Please feel free to contribute to this help with your own examples.

1.8.1 Peptide MS search engine (Tide)

Let's run a proteomics search engine to identified proteins using Tide. Proteomics data analysis is dominated by database-based search engines strategies. Amount Search Engines, **Tide** is one of the most popular nowadays.

Get the docker container

Note: The BioContainers community had decided to remove the `latest` tag. Then, the following command `docker pull biocontainers/crux` will fail. Read more about this decision in *Getting started with Docker*

Getting the data

First thing to do is to arrange the necessary. For this pipeline we need some mass spectrometry data and a protein database in FASTA format. This are the files we are using for this tutorial:

- demo.ms2
- small-yeast.fasta

I'm placing all these files inside a folder in your machine, in my case I will use /Users/yperez/workspace/

Start searching

Note: To have access to files inside the container, and vice-versa, we need to map a local (`/Users/yperez/workplace`) folder inside the container (`/data/`). This is possible using the `-v` and passing the complete path from our folder that we wish to map inside the container.

After running this command you will see a new folder called yeast-index in your path (`/Users/yperez/workplace`).

If everything went well you should see two new folder (*crux-output*) containing the result files.

1.8.2 Command Resume

So far, you launched your first containers using the docker run command. You ran an interactive container that ran in the foreground. You also ran a detached container that ran in the background. In the process you learned about several Docker commands:

- **docker run** - Run a docker container
- **dcoker pull** - Download the container from Biodocker Hub
- **docker ps** - Lists containers.
- **docker logs** - Shows us the standard output of a container.
- **docker stop** - Stops running containers.

Now, you have the basis learn more about Docker Go to "Run a simple application"

1.9 Best Practices

The BioContainers community develop and build guidelines to create, build containers. Here, we describe the guidelines to create Dockerfile containers. If you are interested to read more about BioContainers guidelines, you can follow our recent manuscript in F1000.

- Gruening, B., Sallou, O., Moreno, P., da Veiga Leprevost, F., Ménager, H., Søndergaard, D., Röst, H., Sachsenberg, T., O'Connor, B., Madeira, F. and Del Angel, V.D., BioContainers Community, Perez-Riverol Y. 2018. Recommendations for the packaging and containerizing of bioinformatics software. F1000Research, 7.

1.9.1 Language Specific

Python

- Use requirements.txt instead of listing dependencies
- Define library versions

1.9.2 Optimization

- Use environment variables to avoid repeating yourself

This is a trick I picked up from reading the Dockerfile (link) of the "official" node.js docker image. As an aside, this Dockerfile is great. My only criticism is that it sits on top of a huge buildpack-deps (link) image, with all sorts of things I don't want or need.

You can define environment variables with ENV and then reference them in subsequent RUN commands. Below, I've paraphrased an excerpt from the linked Dockerfile:

```
ENV NODE_VERSION 0.10.32

RUN curl -SLO "http://nodejs.org/dist/v$NODE_VERSION/node-v$NODE_VERSION-linux-x64.
↪tar.gz" && \
    tar -xzf "node-v$NODE_VERSION-linux-x64.tar.gz" -C /usr/local --strip-
↪components=1 && \
    rm "node-v$NODE_VERSION-linux-x64.tar.gz"
```

Merge RUN commands

instead of running:

```
RUN acb
RUN cbd
RUN bde
```

run:

```
RUN acb && cbd && bde
```

or:

```
RUN acb && \
    cbd && \
    bde
```

Note:

- Whenever possible, reuse the same base image and use a LTS (Long Term Support) image preferably (Ubuntu 12.04 or 14.04)
- You can use our biocontainers/biocontainers image based on ubuntu 14.04 with frequent updates and default folders created.
- Use a .dockerignore file: In most cases, it's best to put each Dockerfile in an empty directory. Then, add to that directory only the files needed for building the Dockerfile.
- Avoid installing unnecessary packages: In order to reduce complexity, dependencies, file sizes, and build times, you should avoid installing extra or unnecessary packages just because they might be "nice to have." For example, you don't need to include a text editor in a database image.
- Run only one process per container: In almost all cases, you should only run a single process in a single container. Decoupling applications into multiple containers makes it much easier to scale horizontally and reuse containers. If that service depends on another service, make use of container linking.
- Minimize the number of layers: You need to find the balance between readability (and thus long-term maintainability) of the Dockerfile and minimizing the number of layers it uses. Be strategic and cautious about the number of layers you use.
- Sort multi-line arguments: Whenever possible, ease later changes by sorting multi-line arguments alphanumerically. This will help you avoid duplication of packages and make the list much easier to update. This also makes PRs a lot easier to read and review. Adding a space before a backslash () helps as well.

Here's an example from the buildpack-deps image:

```
RUN apt-get update && apt-get install -y \
    bzr \
    cvs \
    git \
    mercurial \
    subversion
```

Note: Note: Don't install build tools without good reason: Build tools take up a lot of space, and building from source is often slow. If you're just installing somebody else's software, it's usually not necessary to build from source and it should be avoided. For instance, it is not necessary to install python, gcc, etc. to get the latest version of node.js up and running on a Debian host. There is a binary tarball available on the node.js downloads page. Similarly, redis can be installed through the package manager.

There are at least a few good reasons to have build tools:

- you need a specific version (e.g. redis is pretty old in the Debian repositories).
- you need to compile with specific options.
- you will need to npm install (or equivalent) some modules which compile to binary.

In the second case, think really hard about whether you should be doing that. In the third case, I suggest installing the build tools in another "npm installer" image, based on the minimal node.js image.

Don't leave temporary files lying around

The following Dockerfile results in an image size of 109 MB:

```
FROM debian:wheezy
RUN apt-get update && apt-get install -y wget
RUN wget http://cachefly.cachefly.net/10mb.test
RUN rm 10mb.test
```

On the other hand, this seemingly-equivalent Dockerfile results in an image size of 99 MB:

```
FROM debian:wheezy
RUN apt-get update && apt-get install -y wget
RUN wget http://cachefly.cachefly.net/10mb.test && rm 10mb.test
```

Thus it seems that if you leave a file on disk between steps in your Dockerfile, the space will not be reclaimed when you delete the file. It is also often possible to avoid a temporary file entirely, just piping output between commands. For instance,

```
wget -O - http://nodejs.org/dist/v0.10.32/node-v0.10.32-linux-x64.tar.gz | tar zxf -
```

- Clean up after the package manager

If you run apt-get update in setting up your container, it populates /var/lib/apt/lists/ with data that's not needed once the image is finalized. You can safely clear out that directory to save a few megabytes.

This Dockerfile generates a 99 MB image:

```
FROM debian:wheezy
RUN apt-get update && apt-get install -y wget
```

while this one generates a 90 MB image:

```
FROM debian:wheezy
RUN apt-get update && apt-get install -y wget && apt-get clean && apt-get purge && rm␣
↪-rf /var/lib/apt/lists/* /tmp/* /var/tmp/*
```

- Pin package versions

While a docker image is immutable (and that's great), a Dockerfile is not guaranteed to produce the same output when run at different times. The problem, of course, is external state, and we have little control over it. It's best to minimize the impact of external state on your Dockerfile to the extent that it's possible. One simple way to do that is to pin package versions when updating through a package manager. Here's an example of how to do that:

```
# apt-get update
# apt-cache showpkg redis-server
Package: redis-server
Versions:
2:2.4.14-1
# apt-get install redis-server=2:2.4.14-1
```

- Map log files outside

1.9.3 General

- ADD vs COPY: Both ADD and COPY adds local files when building a container but ADD does some additional magic like adding remote files and ungzipping and untaring archives. Only use ADD if you understand this difference.
- WORKDIR and ENV: Each command will create a new temporary image and runs in a new shell hence if you do a cd **:raw-html-m2r:‘<directory>‘** or export **:raw-html-m2r:‘<var>‘=:raw-html-m2r:‘<value>‘** in your Dockerfile it won't work. Use WORKDIR to set your working directory across multiple commands and ENV to set environment variables.
- CMD and ENTRYPOINT: CMD is the default command to execute when an image is run. The default ENTRYPOINT is /bin/sh -c and CMD is passed into that as an argument. We can override ENTRYPOINT in our Dockerfile and make our container behave like an executable taking command line arguments (with default arguments in CMD in our Dockerfile).
- ADD your code last: ADD invalidates your cache if files have changed. Don't invalidate the cache by adding frequently changing stuff too high up in your Dockerfile. Add your code last, libraries and dependencies first. For node.js apps that means adding your package.json first, running npm install and only then adding your code.
- USER in Dockerfiles: By default docker runs everything as root but you can use USER in Dockerfiles. There's no user namespacing in docker so the container sees the users on the host but only uids hence you need the add the users in the container.
- CMD: The CMD instruction should be used to run the software contained by your image, along with any arguments. CMD should almost always be used in the form of CMD [“executable”, “param1”, “param2”…]. Thus, if the image is for a service (Apache, Rails, etc.), you would run something like CMD [“apache2”,”-DFOREGROUND”]. Indeed, this form of the instruction is recommended for any service-based image.
- ENV: In order to make new software easier to run, you can use ENV to update the PATH environment variable for the software your container installs. For example, ENV PATH /usr/local/nginx/bin:$PATH will ensure that CMD [“nginx”] just works. The ENV instruction is also useful for providing required environment variables specific to services you wish to containerize, such as Postgres's PGDATA.
- USER: If a service can run without privileges, use USER to change to a non-root user. Start by creating the user and group in the Dockerfile with something like

```
RUN groupadd -r postgres && useradd -r -g postgres postgres.
```

> Note: Users and groups in an image get a non-deterministic UID/GID in that the "next" UID/GID gets assigned regardless of image rebuilds. So, if it's critical, you should assign an explicit UID/GID. You should avoid installing or using sudo since it has unpredictable TTY and signal-forwarding behavior that can cause more problems than it solves. If you absolutely need functionality similar to sudo (e.g., initializing the daemon as root but running it as non-root), you may be able to use "gosu".

Lastly, to reduce layers and complexity, avoid switching USER back and forth frequently.

- WORKDIR: For clarity and reliability, you should always use absolute paths for your WORKDIR. Also, you should use WORKDIR instead of proliferating instructions like RUN cd … && do-something, which are hard to read, troubleshoot, and maintain.

1.9.4 Volumes

- Should always map to the same /data and /config folders
- Should be RW (read/write) unless there's a good reason not to
- Config and Log files can be mapped to the host but should preferentially be symbolically linked to the /data or /config folder
- Additional mappings can be created if necessary

1.9.5 Images

- Images should be based on the latest LTS image available (Ubuntu 12.04 and 14.04) or to one of our images

1.9.6 Using the BioContainers base image

BioContainers project is using a custom base image for most of its containers. The image is based on Ubuntu Trusty 14.04 LTS and its going to be updated frequently.

1.9.7 Image name and versions

biodckr/biodocker:latest

1.9.8 Core Software and Packages

- curl
- fuse
- git
- wget
- zip
- openjdk-7-jre
- build-essential
- python

- python-dev
- python-pip
- zlib1g-dev

1.9.9 Sources and Useful Links

- https://github.com/veggiemonk/awesome-docker#optimizing-images
- https://labs.ctl.io/optimizing-docker-images/?hvid=1OW0br
- https://docs.docker.com/articles/dockerfile_best-practices/
- http://csaba.palfi.me/random-docker-tips/
- https://docs.docker.com/articles/dockerfile_best-practices/
- http://jonathan.bergknoff.com/journal/building-good-docker-images

1.10 How to cite BioContainers

We recommend to cite BioContainers manuscript every time a container is used:

da Veiga Leprevost F, Grüning BA, Alves Aflitos S, Röst HL, Uszkoreit J, Barsnes H, Vaudel M, Moreno P, Gatto L, Weber J, Bai M, Jimenez RC, Sachsenberg T, Pfeuffer J, Vera Alvarez R, Griss J, Nesvizhskii AI, Perez-Riverol Y. Bioinformatics. 2017 Aug 15;33(16):2580-2582. doi: 10.1093/bioinformatics/btx192. BioContainers: an open-source and community-driven framework for software standardization.

Other important publications:

- Gruening, B., Sallou, O., Moreno, P., da Veiga Leprevost, F., Ménager, H., Søndergaard, D., Röst, H., Sachsenberg, T., O'Connor, B., Madeira, F. and Del Angel, V.D., BioContainers Community, Perez-Riverol Y. 2018. Recommendations for the packaging and containerizing of bioinformatics software. F1000Research, 7.

Here, BioContainers Google Scholar

1.11 Advanced topics

1.11.1 rkt

rkt is a next-generation container manager for Linux clusters. In contrast to the Docker daemon, `rkt` is a single binary that is currently available on CoreOS and Kubernetes only. It is designed for modern Linux clusters environments.

To reference a Docker image, use the `docker://` prefix when fetching or running images.

```
$ rkt --insecure-options=image run docker://biocontainers/comet
```

According to the original documentation:

This behaves similarly to the Docker client, if no specific registry is named, the Docker Hub is used by default. As with Docker, alternative registries can be used by specifying the registry as part of the image reference.

1.11.2 Docker useful tips

Bash interactive script

In this example:

Note: `-t` flag assigns a pseudo-tty or terminal inside the new container. `-i` flag allows you to make an interactive connection by grabbing the standard in (STDIN) of the container. `/bin/bash` launches a Bash shell inside our container.

Let's try running some commands inside the container:

In this example `ls` displays the directory listing of the root directory inside the container.

Note: When completed, run the `exit` command or enter Ctrl-D to exit the interactive shell.

Start a daemonized Hello world

Let's create a container that runs as a daemon.

Note: `-d` flag runs the container in the background (to daemonize it). The command `/bin/sh -c "while true; do echo hello world; sleep 1; done"` run the container in an infinite loop.

We can use this container ID to see what's happening with our hello world daemon. First, let's make sure our container is running. Run the `docker ps` command. The docker `ps` command queries the Docker daemon for information about all the containers it knows about.

1.12 Contributing

This document briefly describes how to contribute to the BioContainers project.

1.12.1 Before you Begin

If you have an idea for a feature/container to add or an approach for a bugfix, it is best to communicate with BioContainers developers early. The most common venues for this are GitHub issues for common specification issues and the Containers and Tools for container/docker related issues. Browse through existing GitHub issues and if one seems related, comment on it. If no existing issue seems appropriate, a new issue can be opened using this form. BioContainers developers are also generally available via Gitter

1.12.2 How to Contribute

- All changes to the specifications BioContainers project should be made through pull requests to this repository (with just two exceptions outlined below).
- If you are new to Git, the Try Git tutorial is a good places to start. More learning resources are listed at https://help.github.com/articles/good-resources-for-learning-git-and-github/ .
- Make sure you have a free GitHub account.

- Fork the container repository on GitHub to make your changes. (While many Containers instances track active development happens in the containers GitHub repository and this is where pull requests should be made).
- Choose the correct branch to develop your changes against.
  - Additions of new features to the code base should be pushed to the `dev` branch (`git checkout dev`).
  - Most bug fixes to previously release components (things in bidocker-dist) should be made against the recent `release_XX.XX` branch (`git checkout release_XX.XX`).
  - Serious security problems should not be fixed via pull request - please responsibly disclose these by e-mailing them (with or without patches) to biodockers@gmail.com . The BioContainers core development team will solve those issues and we will provide you credit for the discovery when publicly disclosing the issue.
- If your changes modify containers/images - please ensure the resulting files conform to BioContainers Specifications BioContainers Specifications.
- Commit and push your changes to your fork.
- Open a pull request with these changes. You pull request message ideally should include:
  - A description of why the changes should be made.
  - A description of the implementation of the changes.
  - A description of how to test the changes.
- The pull request should pass all the continuous integration tests which are automatically run by GitHub using e.g. Travis CI.

1.12.3 Ideas

BioContainers's BioContainers Specification and Design is filled with comments and ideas for enhancements and we believe would make the best entry points for new developers.

1.12.4 A Quick Note about Containers

> For the most part, BioContainers containers should be published to the BioContainers containers and not in this repository directly. If you are looking to supply new containers first check if an existing container exists in this repository BioContainers containers - please checkout the repository on GitHub.

1.12.5 Handling Pull Requests

Everyone is encouraged to express opinions and issue non-binding votes on pull requests, but only members of the *contributors* group may issue binding votes on pull requests.

Votes on pull requests should take the form of +1, 0, -1, and fractions as outlined by the Apache Foundation.

Pull requests modifying pre-existing releases should be restricted to bug fixes and require at least 2 *+1* binding votes from someone other than the author of the pull request with no *-1* binding votes.

Pull requests changing or clarifying the procedures governing this repository:

- Must be made to the `dev` branch of this repository.
- Must remain open for at least 192 hours (unless every qualified committer has voted).
- Require binding *+1* votes from at least 25% of qualified *committers* with no *-1* binding votes.

- Should be titled with the prefix *[PROCEDURES]* and tagged with the *procedures* tag in Github.
- Should not be modified once open. If changes are needed, the pull request should be closed, re-opened with modifications, and votes reset.
- Should be restricted to just modifying the procedures and generally should not contain code modifications.
- If the pull request adds or removes committers, there must be a separate pull request for each person added or removed.

Any other pull request requires at least 1 *+1* binding vote from someone other than the author of the pull request. A member of the committers group merging a pull request is considered an implicit +1.

Pull requests marked *[WIP]* (i.e. work in progress) in the title by the author(s), or tagged WIP via GitHub tags, may *not* be merged without coordinating the removal of that tag with the pull request author(s), and completing the removal of that tag from wherever it is present in the open pull request.

Timelines

Except in the case of pull requests modifying governance procedures, there are generally no objective guidelines defining how long pull requests must remain open for comment. Subjectively speaking though - larger and more potentially controversial pull requests containing enhancements should remain open for a at least a few days to give everyone the opportunity to weigh in.

Vetoes

A note on vetoes (*-1* votes) taken verbatim from the Apache Foundation:

> "A code-modification proposal may be stopped dead in its tracks by a -1 vote by a qualified voter. This constitutes a veto, and it cannot be overruled nor overridden by anyone. Vetoes stand until and unless withdrawn by their casters.
>
> To prevent vetoes from being used capriciously, they must be accompanied by a technical justification showing why the change is bad (opens a security exposure, negatively affects performance, etc.). A veto without a justification is invalid and has no weight."

For votes regarding non-coding issues such as procedure changes, the requirement that a veto is accompanied by a *technical* justification is relaxed somewhat, though a well reasoned justification must still be included.

Reversions

A *-1* vote on any recently merged pull request requires an immediate reversion of the merged pull request. The backout of such a pull request invokes a mandatory, minimum 72 hour, review period.

- Recently merged pull requests are defined as a being within the past 168 hours (7 days), so as to not prevent forward progress, while allowing for reversions of things merged without proper review and consensus.
- The person issuing the -1 vote will, upon commenting `-1` with technical justification per the vetoes section, immediately open a pull request to revert the original merge in question. If any committer other than the -1 issuer deems the justification technical - regardless of whether they agree with justification - that committer must then merge the pull request to revert.

Direct Commit Access

The BioContainers *committers* group may only commit directly to BioContainers (i.e. outside of a pull request and not following the procedures described here) the following two categories of patches:

- Patches for serious security vulnerabilities.
- Cherry-picking and/or merging of existing approved commits to other branches.