The Journal of Open Source Software

Damona: a Singularity environment manager for reproducible analysis

Thomas Cokelaer[1,2]

1 Hub de Bioinformatique et Biostatistique – Département Biologie Computationnelle, Institut Pasteur, USR 3756 CNRS, Paris, France **2** Plate-forme Technologique Biomics – Centre de Ressources et Recherches Technologiques (C2RT), Institut Pasteur, Paris, France

DOI: 10.21105/joss.02561

Software

- Review
- Repository
- Archive

Editor: Pending Editor

Submitted: 07 August 2020
Published: 10 August 2020

Summary

Next Sequencing Generation (NGS) (Goodwin, McPherson, & McCombie, 2016) implies complex analysis using pipelines that depend on a plethora of software. A few years ago, setting up a local system to reproduce an analysis was a challenge even for computer scientists. Indeed, one needed to be proficient in several languages. For instance, you would need to compile C/C++ code or set up the correct JAVA or R environments just to cite a few examples. Hours were spent in retrieving the correct source code, days in compilation time, weeks in reproducing analysis. Reproducing the analysis on another operating system would have been even more challenging.

Solutions are now available to help researchers installing reproducible NGS pipelines, and in particular their third-party dependencies. One of them is **Conda** (Conda, 2017), an open-source *package* management system, and an *environment* management system. Conda provides pre-compiled releases of software; they can be installed in different local environments that do not interfere with your system. Thereupon, different communities have emerged using this framework. One of them is **Bioconda** (Grüning et al., 2018), which is dedicated to bioinformatics. A complementary tool is **Singularity** (Kurtzer, Sochat, & Bauer, 2017). It allows you to create containers that package your favorite environment and software (even data) in a way that is portable and reproducible. You can build a container locally, copy it elsewhere and run it straight away (e.g., on a High-Performance Computing (HPC) systems). It is a simple flat file that can be shared between environments and guarantee execution and reproducibility. Note that nothing prevents you from including a Conda environment inside a Singularity container to speed up, and simplify the creation of the container.

With *Conda* and *Singularity*, Bioinformaticians have now all the tools to build reproducible pipelines. In practice, Conda environment can also be used to provide a development framework. You can quickly install software in an environment and develop your software in it. Then, you can share your environment with your colleagues. It is a quite effective solution to reproduce analysis while moving forward with development. Yet, with an increasing number of tools used in NGS pipelines, practical issues may arise: impossibility to reproduce an environment, conflicts when installing software, a long time to resolve dependencies. You could have an environment per pipeline but then you need to install common packages several times in different environments. What if you realise that a tool has a wrong version and you do not want, or cannot update your Conda environment. A simple solution consists in providing a Singularity container for that specific package and put it in your environment. We could generalise this idea, and move all the packages from a Conda environment within the Singularity container or the Conda environment itself. Then we have a 100% reproducible environment. However, you cannot change it easily. This is not an environment you can deploy iteratively.

This is why we have moved little by little to a very light Conda environment where known-to-cause-problem packages have been shipped into Singularity containers. Since we still have

the Conda environment, we keep its flexibility and ability to update our Python software easily as well. This gives us the flexibility of a Conda environment(s) while having the complex packages available as Singularity containers. Yet, with an increasing number of Singularity containers, we need to have aliases and make them available for each environment. A manager for Singularity containers would be of help for end-users and developers.

Statement of Need

Damona is a light manager for Singularity containers. It should be straightforward to use for end-users with little knowledge in computer science of software deployment. For developers, it is still compatible with your Conda environment: you may have as many **Damona** environments in parallel that will host Singularity containers. An environment can be set up for a given analysis. It is then easy to export the containers and share them in another environment. Each developer will adjust the trade-off between packages installed with Conda and containers installed with **Damona** based on its needs.

The goal of **Damona** is not to replace Conda or Singularity but to use them effectively and complement them when required. In particular, we designed **Damona** so as to provide the containers required by Sequana pipelines (Cokelaer, Desvillechabrol, Legendre, & Cardon, 2017). Therefore, we provide some singularities but more as examples and proof-of-concept rather than an exhaustive set of Singularity containers.

The main purpose of **Damona** is for developers to provide an easy solution for their end-users to install third-party tools more easily thanks to Singularity containers.

In the following, we quickly describe the principle of **Damona** followed by some test cases.

Damona to manage Singularity containers and environments

This is the egg and chicken paradox. To allow reproducibility you need a tool to start with and it is singularity. To use **Damona**, one first need to install Singularity itself. Instructions from their user guide should be sufficient. On HPC systems, your administrator should have installed it already.

Then, install **Damona** itself. It is written in Python, and it is available on The Python Pypi website. The following command should install the latest version of the software:

```
pip install damona --upgrade
```

You can now test the installation by typing

```
damona --help
```

By default, **Damona** provides a few Singularity recipes, which are stored on external servers, in particular on cloud.sylabs.io and SALSA group as explained later. You can see the default list of downloadable containers using the *list* command:

```
damona list
```

You should see at least two instances of the tool called *fastqc* (Simon, 2010)(fastqc:0.11.9 and fastqc:0.11.8). Given the name and version you can now download and install one of those version (e.g., the oldest):

```
damona install fastqc:0.11.8
```

This command downloads the requested container and copies it in the **Damona** path (`~/.config/damona/images`). Then, it creates a binary in `~/.config/damona/bin` named after the executable (here, *fastqc*). All you need to do is to append the *bin* directory to your environment (PATH). You should now be able to launch *fastqc* command. If you prefer to use the latest version, you could just type:

```
damona install fastqc
```

or even more explicitly:

```
damona install fastqc:0.11.9
```

The binary alias is updated to match the latest container you have installed (not necessary the latest version).

The default Singularity containers have their recipes within **Damona** together with a registry file. This file tells us explicitly the name of the container, where it can be found, its version and the type of containers. We consider 3 types of containers:

- *executable*: like in the previous example, this is a container that ships only one main executable. The container is intended to be used as an executable and the name of the container should be the name of the executable.
- *environment*: this type of container is meant to be used by other recipes to build executable. This is to make the build of recipes quicker by being more modular. We have R and Conda environments examples within **Damona**.
- *set of executables*: one executable per application may not be optimal; instead you may wish to provide several executables within a single container; for instance all java-related tools in a container, all perl-related tools in another, etc.

When installing an *executable* container, as shown in the example here above, a binary/alias is created in the *bin* directory. When installing an *environment* container, no binary/alias are created. Finally, when installing a *set* container, you may have more than one binary to expose. The maintainer of the container recipe should fill a registry file (YAML format) with the list of binaries that are available. For instance, when you install the following *set* container:

```
damona install sequana_tools:0.9.0
```

then about 30 binaries are created. An informative message should tell you about their names.

So far we have installed containers in the `~/config/damona/image directory`, and all binaries in `~/.config/damona/bin`. The main feature of **Damona** is to manage different environments for Singularity containers. So, let us create an environment:

```
damona env --create test1
```

You would need to set it as your working environment. To do so, create the environmental variable:

```
export DAMONA_ENV=~/.config/damona/envs/test1
```

All new installation will still copy the images in the default environment (to avoid duplication) but new binaries will be stored in `~/.config/damona/envs/test1/bin`

You can have as many environments as you want. You can create and delete them as well.

As stated above, by default, when using the *list* command you will see only a few containers, which are provided with **Damona** for demonstration. Nevertheless, we provide a mechanism for each individual or research group to post their singularity containers on their website together with a very simple registry file. The file should look like:

```
[exe]
fastqc_0.11.9.img
fastqc_0.11.8.img
salmon_1.3.0.img
[env]
conda_4.7.12.img
[set]
sequana_tools_0.9.0.img dsrc pigz bwa minimap2
```

Here you can recognise the 3 type of containers discussed above. The `[exe]` and `[env]` sections give the list of images to be found on the server (same directory as the registry.txt file). In the `[set]` section, each container is followed by the binaries that are provided by the container.

You would then use the following commands to see the list of containers and install them:

```
damona list --from-url https://biomics.pasteur.fr/drylab/damona/registry.txt
damona install sequana_tools_0.9.0.img --from-url https://biomics.pasteur.fr/d
```

Aliases can be set in the configuration file on `~/.config/damona/damona.cfg`. The previous commands simplify to:

```
damona list --from-url damona
damona install sequana_tools_0.9.0.img --from-url damona
```

With this mechanism in place you can set your own registry server by placing the container images and corresponding registry.txt in an accessible place so that your users can easily download your dependencies.

You can now set different singularity environments for different NGS pipelines.

Test cases

Testing same pipeline with two different version of a third-party tool

In order to test a pipeline with two different version of a software, you may create a conda environment where your pipeline is installed. Then, you would install the software you want to test for different versions in 2 different Damona environments. Consequently, you will not interfere with your conda environment.

```
damona env --create version1
damona env --create version2
```

```
# activate the first environment to install version 0.11.8
export DAMONA_ENV="version1"
damona install fastqc:0.11.8

# activate the second environment to install version 0.11.9
export DAMONA_ENV="version1"
damona install fastqc:0.11.9
```

Before running the pipeline, change your system so if users either version 1 or 2.

Sharing your environment made easy

Sharing a **Damona** environment is as simple as copying a directory. There is no interference with your system, no databases either. So a simple recursive copy is enough.

Updating an entire environment replacing only one file

Imagine that one of the tool is actually buggy. All you need to do is to find a Singularity container or build it to fix the bug. You may post it on a server with a registry file as explained above and then install it with **Damona**. Since the environment is a a set of files, as a developer, you may also simply replace the existing file. This means you can update an environment in an instant.

Build a production environement for your users

Here is a final example with Sequana (Cokelaer et al., 2017). We used to have a unique Conda environment where sequana and its dependencies would be installed. Then, for each pipeline, we would also install their third-party requirements. The Conda environement quickly became unstable. We mean that upgrading the version of a package would start to be difficult: other packages would be downgraded or upgraded breaking the reproducibility of other analysis. We therefore moved little by little to a paradigm where the conda environment is as simple as possible (e.g. installing Python software only) and all other tools would reside in a single singularity file. With **Damona** we create one environment per pipeline keeping all singularity files independent from th conda environement. It is then easy to either create a new conda environement that would use the same **Damona** singularity containers. Updating a buggy singularity image would also benefit to all Conda environments.

Future directions

Damona is quite recent but already used in production. The version at the time of writting this document was 0.4.1. New features and roadmap are available on the github repository as well as on the on line documentation.

Acknowledgments

This work has been supported by the France Génomique Consortium (ANR 10-INBS-09-08).

References

Cokelaer, T., Desvillechabrol, D., Legendre, R., & Cardon, M. (2017). 'Sequana': A set of snakemake ngs pipelines. *Journal of Open Source Software*, *2*(16), 352. doi:10.21105/joss.00352

Conda. (2017). Conda package and environment management system. Retrieved from https:/docs.conda.io/en/latest

Goodwin, S., McPherson, J. D., & McCombie, W. R. (2016). Coming of age: ten years of next-generation sequencing technologies. *Nature Reviews Genetics*, *17*(6), 333–351. doi:10.1038/nrg.2016.49

Grüning, B., Dale, R., Sjödin, A., Chapman, B. A., Rowe, J., Tomkins-Tinch, C. H., Valieris, R., et al. (2018). Bioconda: sustainable and comprehensive software distribution for the life sciences. *Nature Methods*, *15*(7), 475–476. doi:10.1038/s41592-018-0046-7

Kurtzer, G. M., Sochat, V., & Bauer, M. W. (2017). Singularity: Scientific containers for mobility of compute. *PLoS ONE*, *12*(5), 1–20. doi:10.1371/journal.pone.0177459

Simon, A. (2010). FastQC: A quality control tool for high throughput sequence data. Retrieved from http://www.bioinformatics.babraham.ac.uk/projects/fastqc/