Packaging and containerizing of bioinformatics software: advances, challenges, and opportunities

This manuscript (permalink) was automatically generated from Mangul-Lab-USC/pkg-manager-review@0af2f9f on September 5, 2020.

Unsorted list of authors

- **Sharon Waymost**
  0000-0003-1176-5386 · sbpw
  Computer Science Department, Samueli School of Engineering, University of California, Los Angeles

- **Serghei Mangul**
  0000-0003-4770-3443 · smangul1
  Department of Clinical Pharmacy, School of Pharmacy, University of Southern California

- **Neha Rajkumar**

  Department of Bioengineering, Samueli School of Engineering, University of California, Los Angeles

- **Ram Ayyala**

  Department of Neuroscience, School of Life Sciences, University of California Los Angeles

- **Heng Li**

  None

- **Nathan LaPierre**
  0000-0003-2394-8868 · nlapier2 · nlapier2
  Computer Science Department, Samueli School of Engineering, University of California, Los Angeles

- **André Santos**

  None

- **Titus Brown**

  University of California, Davis

- **Casey S. Greene**
  0000-0001-8713-9213
  Department of Systems Pharmacology and Translational Therapeutics, Perelman School of Medicine, University of Pennsylvania; Childhood Cancer Data Lab

- **Brendan Lawlor**
  0000-0002-2250-0669 · blawlor · blawlor
  Department of Computer Science, Cork Institute of Technology, Cork, Ireland.

- **Qiyang Hu**

  None

- **Jaqueline J. Brito**
  0000-0002-7158-3253 · jaquejbrito · jaquejbrito
  Department of Clinical Pharmacy, School of Pharmacy, University of Southern California

- **Thiago Mosqueiro**

  None

- **Benjamin J. Heil**
  0000-0002-2811-1031 · ben-heil · autobencoder
  Genomics and Computational Biology Graduate Group, Perelman School of Medicine, University of Pennsylvania

Abstract

Introduction

The rapid advancement of omics and sequencing technologies has led to an accelerated growth of genomic data. Driven by this data, there are similar increases in the number of available bioinformatics software tools, often containing novel and diverse algorithms of particular interest to biomedical researchers.[1,2]. Biomedical researchers wanting to use this software often have access to high-performance clusters ("clusters"), but lack the operating system-level permissions and advanced skills required to install and run the software. None of the necessary skills are currently included in the traditional life science curriculum at major universities. Usability is further hindered by the absence of any standardization and the wide variety of software development tools employed. This is directly observable in many academic software tools lacking a user-friendly interface[5].

As the dependence of biomedical researchers on computational software continues to increase, software developers need to consider more user-friendly distribution and install methods. Modern software distribution, having already faced user-friendliness issues, is already becoming more reliant on platforms such as package managers[6,7] and containers[8]. Both promise to simplify software development while increasing the usability and reproducibility of biomedical research[8,9].

Package managers first appeared nearly thirty years ago as software developers sought to streamline the entire installation process. To install via package manager, the user must only specify the desired software, called a "package;" the download, installation, configuration, and dependencies are all handled by the package manager. Many operating systems have built-in package managers (e.g. APT), which may not be available to cluster users, while others must be downloaded and installed by the user. Some package managers (e.g. Conda[6]) are programming language agnostic, while others are designed for a particular language (e.g pip[10]).

A more recent software distribution solution is containerization. The end-user downloads a container "image" that includes the software, dependencies, and anything else necessary to run the software. Though the imaged software is not typically itself installed, many do require the installation of containerization runtime software. When run, the runtime software creates a consistent, isolated sandbox environment, then runs the imaged software inside the sandbox[11]. This sandbox design makes the images both highly portable (compatible with different computers) and easily shareable (transferable between different computers), which has already led to wide adaptation in bioinformatics[7,12].

In addition to the ease of installation and use offered by both package managers and containers, such platforms must also meet the biomedical community's need for compatibility with high performance clusters. However, the relative performance of these package managers and containers remains unknown. Our review summarizes developing practices across the most common package managers and containers, while discussing the challenges, advantages, and limitations of using them from both the developer and the user perspectives. By taking a survey of all the available package managers and container software, there is now a comprehensive list of the different attributes of each one, informing the community so that people can make educated decisions about which package manager or container makes the most sense for their project. We also propose principles that can make packaging and containerizing of bioinformatics software more sustainable and reproducible, ultimately increasing the usability of bioinformatics software.

Discussion

Existing problems with software distribution and installation

The installation process of bioinformatics research software is typically a multi-step process, starting with the end-user locating and downloading the software. Next is the actual installation, during which the end-user must determine what dependencies are missing and resolve them by installing the required software[5]. Even in cases where the end-user is familiar with this process, they are typically installing on high-performance clusters where they are constrained by user-level permissions that prevent them from following standard installation procedures.

In addition to being difficult, manual installation of bioinformatics software can lead to irreproducible findings. Software versions change over time, sometimes leading to different runtime behavior. If an analysis is dependent on recently introduced features of a tool, users trying to reproduce the work with older installations will fail. Likewise, even existing functionality can change over time. For example, the `input` function in Python 2 is used to run code while in Python 3 it creates a command line prompt [**???**]. To prevent such issues without using a package manager, it is necessary to manually keep track of the versions of all dependencies.

- root access limitations
- reproducibility of findings
- version conflicts
- dependency resolution

Definitions and explanations of distribution system types

Package Managers

Package managers appeared nearly thirty years ago as software developers sought to automate and standardize the software installation process. The earliest package managers (e.g. APT and RPM) were built into Linux operating systems and lacked now common features like dependency resolution. Modern package managers are available for all major operating systems, with both integrated and user-installed options available. Some are package managers are limited to software written in a specific programming language (e.g. pip distributes software written in Python), while others are limited to sofware with a particular purpose (e.g. Bioconda distributes omics software).

Every package managers uses its own package format, which follows the general structure shown in Figure 1.

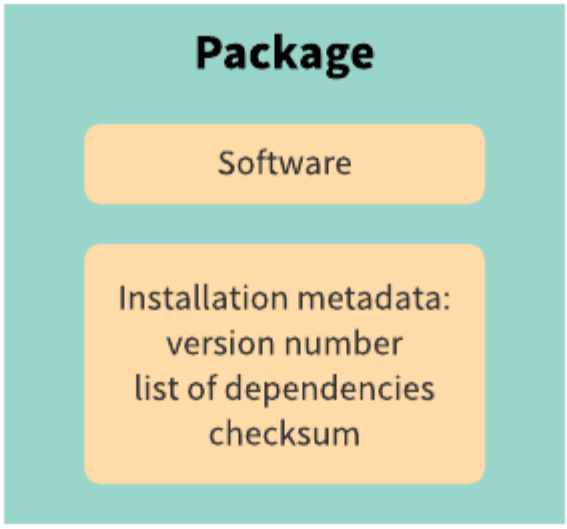

Figure 1: Schematic showing standard package contents. Though the exact contents varies between different package formats, all include the software and relevant metadata necessary to complete installation.

The metadata gives the package manager the necessary information to install the packaged software.

To use a package manager, a user instructs the package manager to install software, then the package manager performs all the necessary steps to install the software as shown in Figure 2. Assuming the software is not already installed, the package manager retrieves the appropriate package from the repository, a server that stores all of the available packages. The package manager then uses the package's metadata to determine what it needs to do to complete installation. For most packages, this includes verifying the installation of all dependencies, which are other pieces of software that must be installed prior to the initial software being installed.

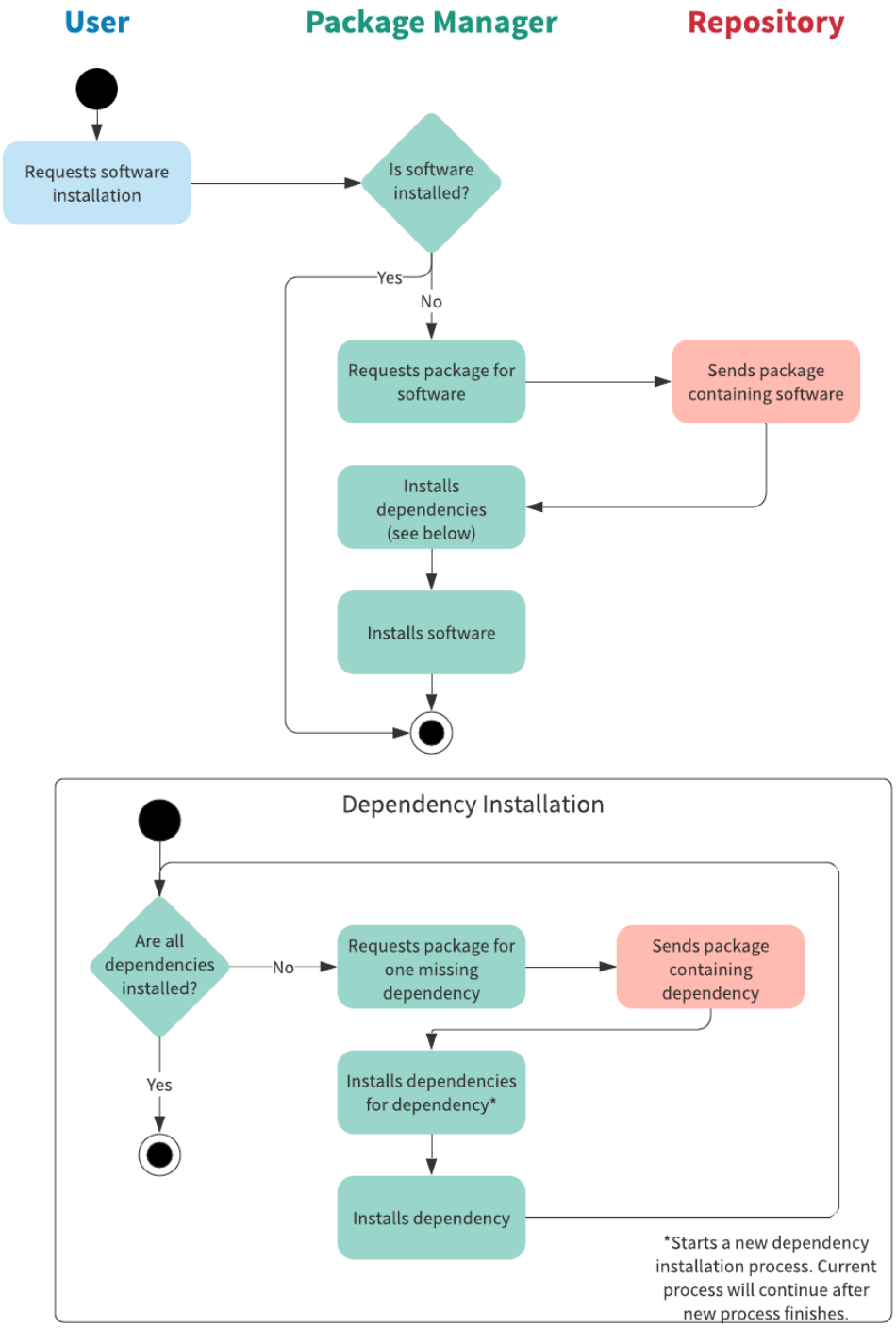

Figure 2: Standard workflow for installing software via a package manager. The user, usually an admin, asks the package manager to install a specific piece of software. If the software is not already installed, the package manager fetches the approrpriate package from the repository. If any of the dependencies are not already installed, the package manager "loops" through the missing dependencies. Once all dependencies are installed, the original software is installed.

The package manager verifies each dependency by confirming that the dependency is already installed or installing the dependency if it is missing. For missing dependencies, the package manager retrieves the dependency's package from the repository and starts the installation procedure for that package. Once all the dependencies are installed, the initially requested software is installed. However, the package manager often goes through several iterations of this process because every dependency can have it's own list of dependencies, in which case each of the dependency's dependencies must be verified through the same process.

In most cases, a package manager is able to install all of the missing dependencies, but there are two relatively common problems that a package manager cannot overcome. The first is circular dependencies, which occurs when software A has a dependency on software B and software B has a dependency on software A. A package manager cannot install A without installing B first, but B cannot be installed without installing A first. The second is conflicting versions, which occurs when software A and software B have the same dependency, but require different versions. Package managers only allow one version of software to be installed at a time, preventing both A and B from having their depedencies met simultaneously.

Administrators often block users from installing software to prevent unwanted changes to both the operating system and existing sofware. This includes blocking access to package managers built into the operating system, as well as preventing users from installing their own.

- benefits for the developer
  - mature technology - higher degree of familiarity
  - allows dependency specification (including versions) -limitations for the developer
  - can't always use to install missing dependencies for end-user
- benefits for the end-user
  - package size is minimal (dependencies aren't duplicated)
  - installs missing dependencies
- limitations for the end-user
  - not always accessible (unless admin user)
  - can't install multiple versions of same software

Containerization and Virtualization

One approach to sharing software involves virtualization. Virtualization is a technique that allows the creation of instances of a software environment, often called 'images'. These images present many benefits for software sharing: only a single image file needs to be downloaded by users, and these image files can have all necessary dependencies for the software pre-installed by the developer, thereby avoiding installation difficulties for users.

Traditional virtual machines emulate operating systems and are run directly on the host's hardware. These are sometimes known as 'type 1 hypervisors' [1]. Later, 'type 2' virtualization techniques were developed, running on top of the host operating system (OS) by translating all OS instructions within the guest image [1]. While these approaches excel in portability, they incur a significant CPU and memory overhead, prompting the development of a more recent approach called Operating System (OS)-Level Virtualization. OS-Level Virtualization runs on top of the same kernel as the host OS, allowing multiple isolated user spaces to be run in parallel and incurring lower CPU, memory, and networking overhead [1]. These user spaces, e.g. OS-level images being executed by users, are often called "containers", but also go by a variety of other names. Examples of this technique include Docker [3], LXC [4], OpenVZ [5], Linux-VServer [6], and rkt [7].

Of these, Docker is perhaps the most widely-used platform. A key aspect of the Docker approach is its modularity: images are stored in a public repository on Docker's website, and it is easy to use an existing image as a "base" to build a new image off of. Docker images are defined by a human-readable, plaintext "Dockerfile" in which the developer specifies whether to build off of a previous base docker image and what additional software libraries and dependencies should be installed. The Dockerfile also serves as an easy-to-read documentation of what dependencies the software is built on [8]. Docker then builds an image based on the developer's specifications and this image can be uploaded to the Docker website.

Several limitations exist with Docker and other OS-level virtualization platforms. OS-level images are, as the name implies, limited to a specific host operating system [1]. Security concerns have been raised about the vulnerability of containers to compromise, denial of service, and privilege escalation attacks [2]. Deployment on high-performance computing clusters is a promising possibility but presents substantial engineering challenges having to do with maximizing resource utilization and allocating resources effectively [9].

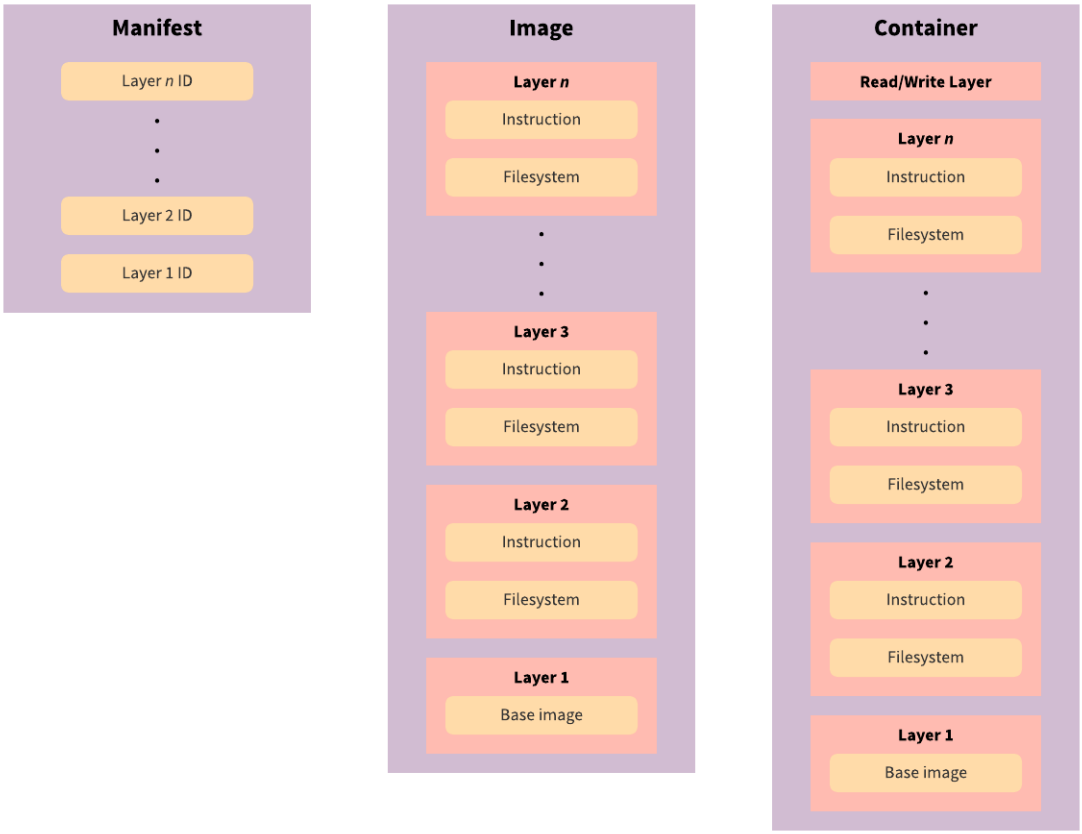

Figure 3: Schematic showing standard containerization objects.

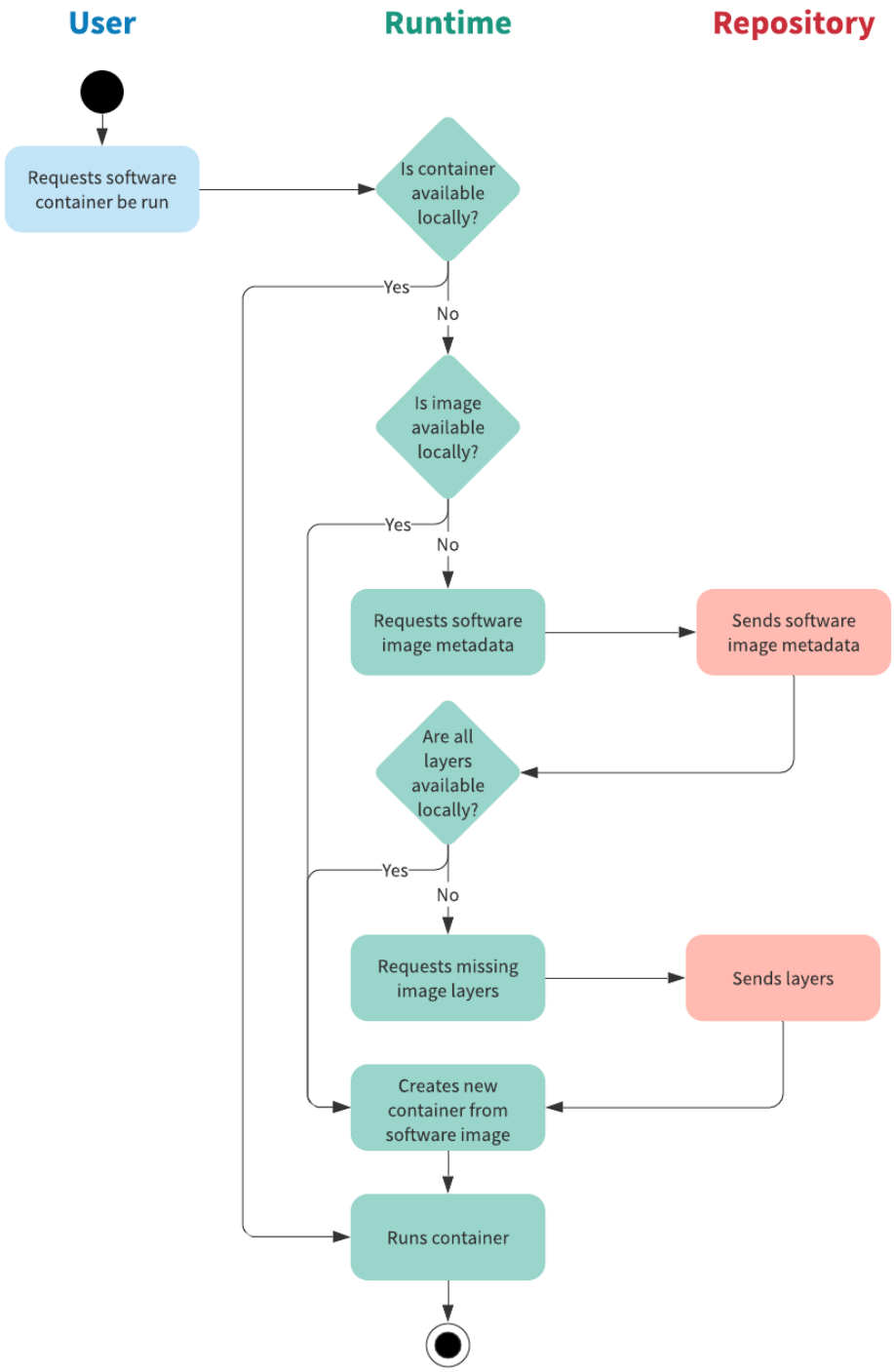

Figure 4: Running containerized software

- benefits for the developer
  - include specific versions of dependencies
  - known running environment
    - fewer test variables
    - reproducibility of results
- limitations for the developer
  - learn a new system instead of focusing on research
- benefits for the end-user
  - no installation (except possible runtime)
  - no dependency issues
  - sandbox provides computer system security
- limitations for the end-user

    - container size
    - duplication of dependencies
    - root access requirement to install runtime
    - configuration in cluster
- centralized repositories
    - definition
        - channels
    - benefits
        - known download site
        - hosting is taken of
    - limitations
        - repo specific restrictions

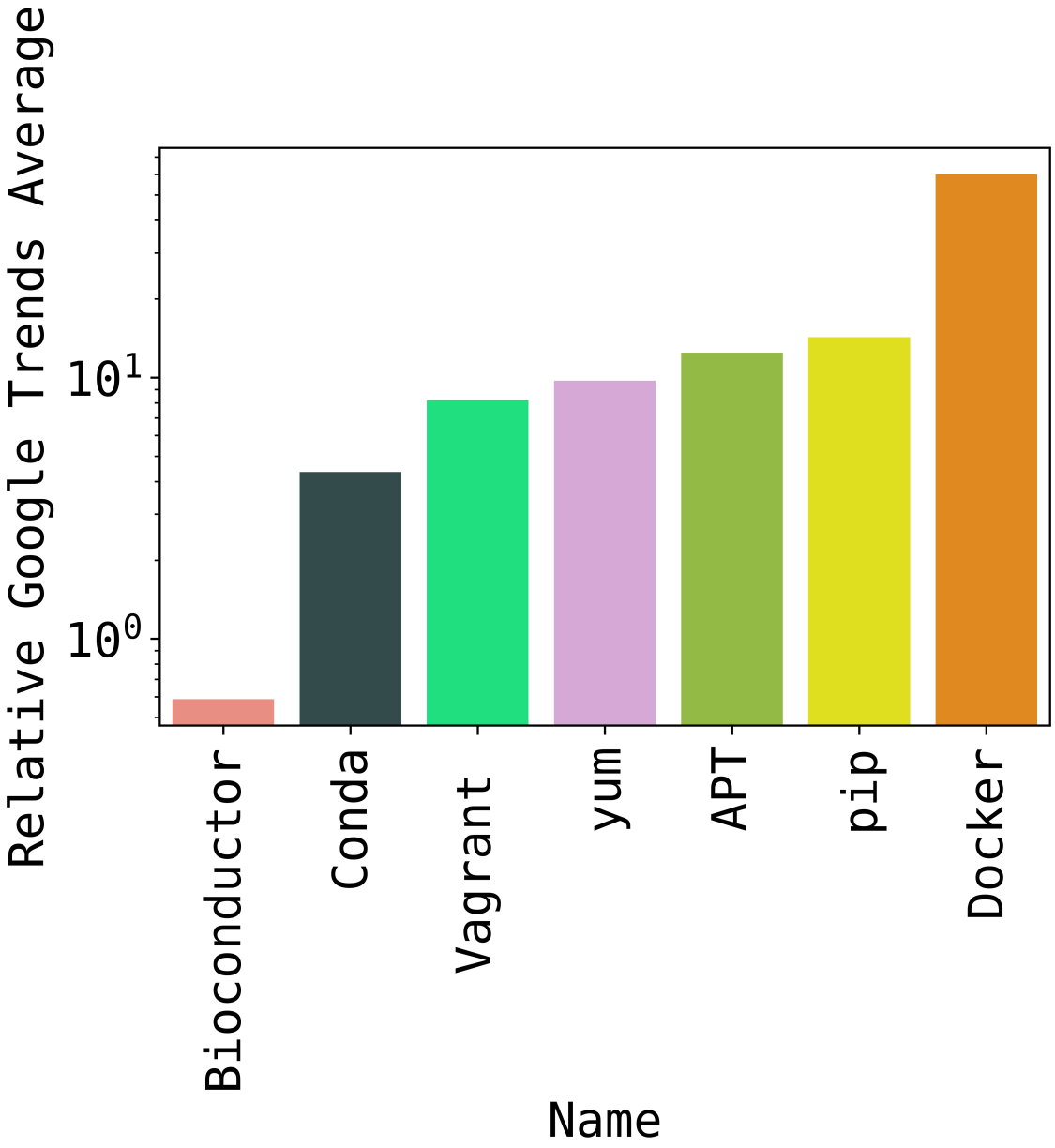

Figure 5: Relative popularity of package managers. According to Google Trends, general use package managers have at least ten times the popularity of their omics-only counterparts, which is expected when their limited-use is taken into account. Docker, the most popular containerization software is nearly five times as popular as the most popular package manager, indicating a software industry in trend towards containerization over the past .

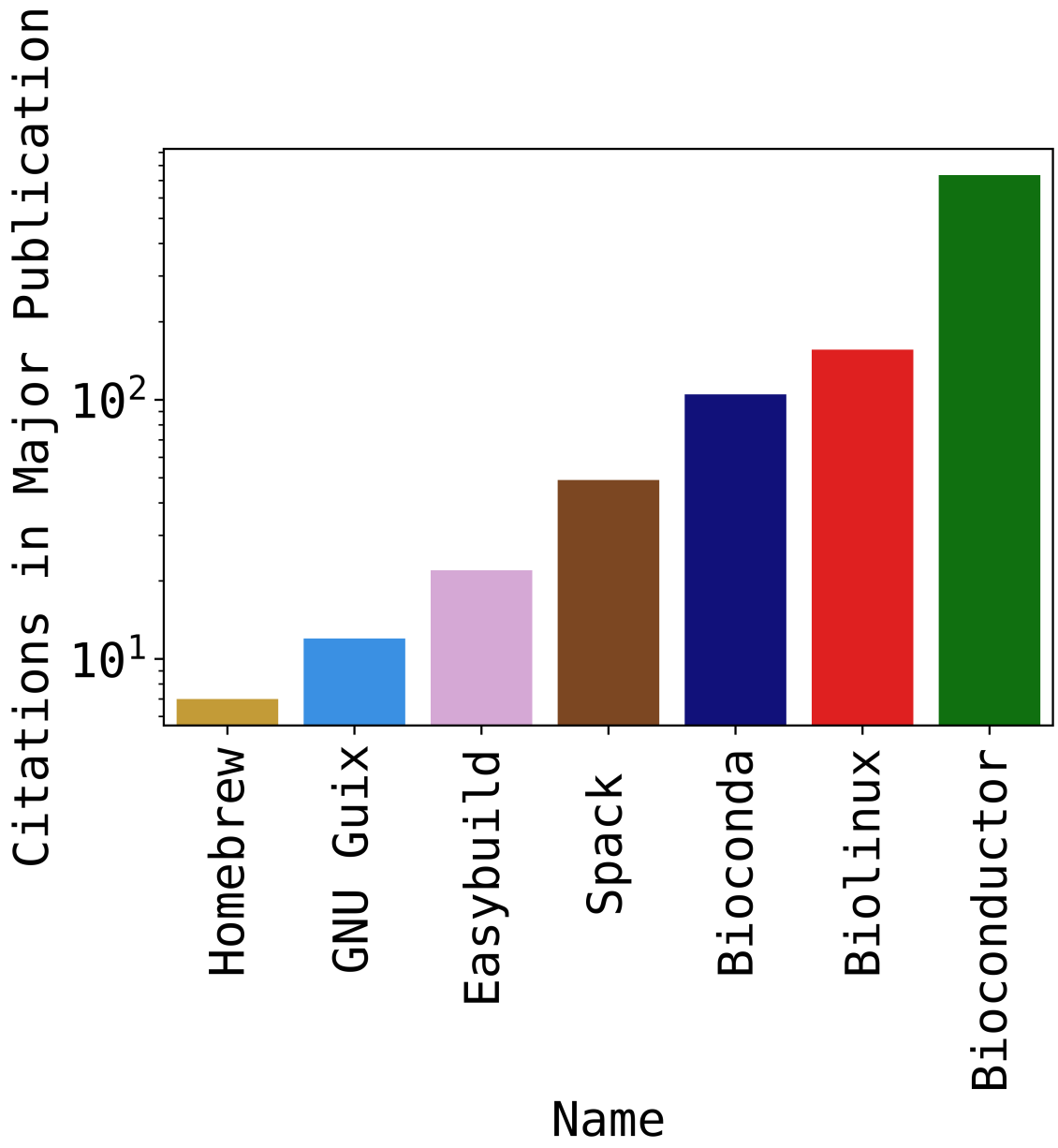

Figure 6: Usage of package managers in academia based on citations.

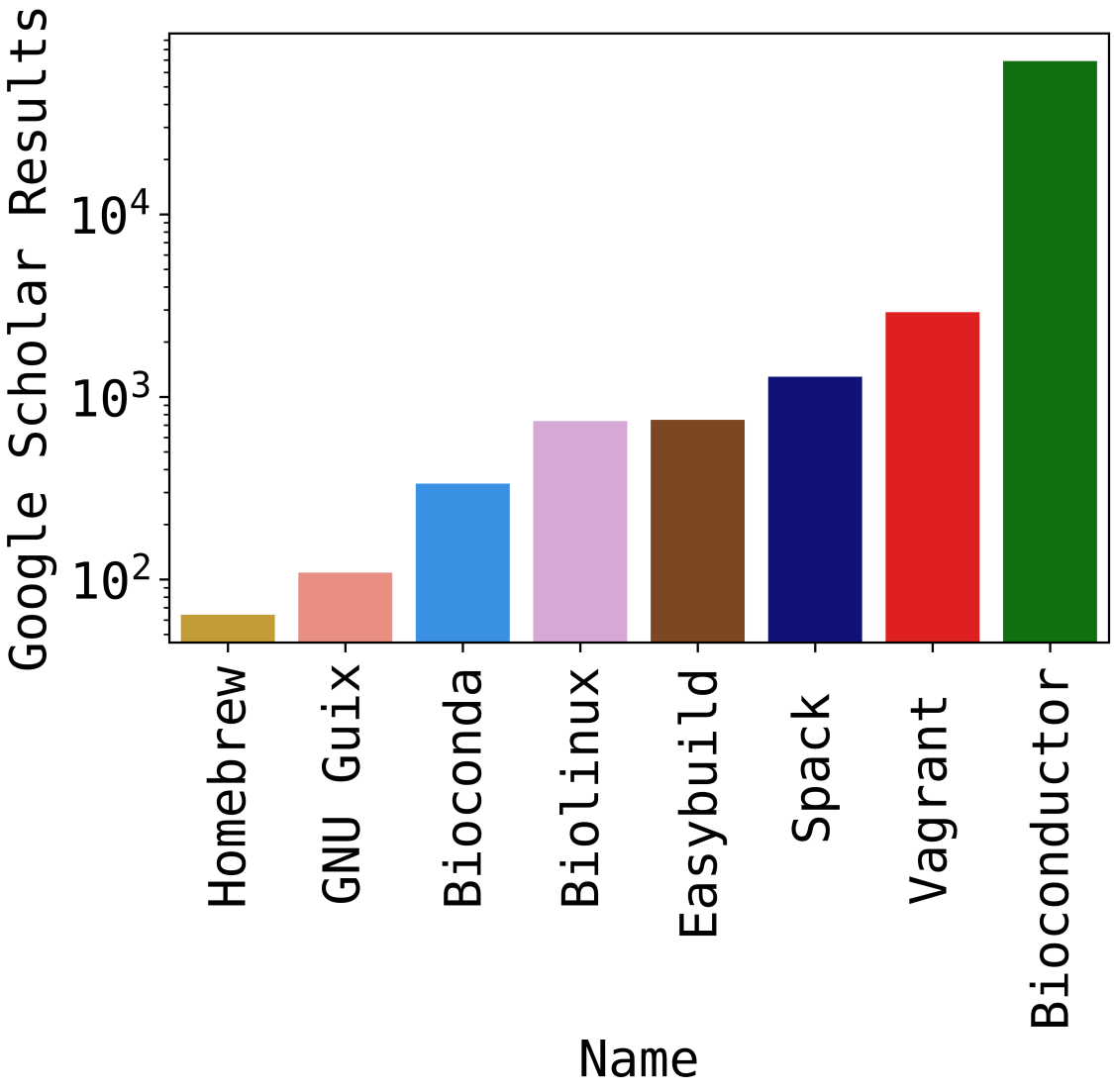

Figure 7: Usage of package managers in academia based on Google Scholar Results.

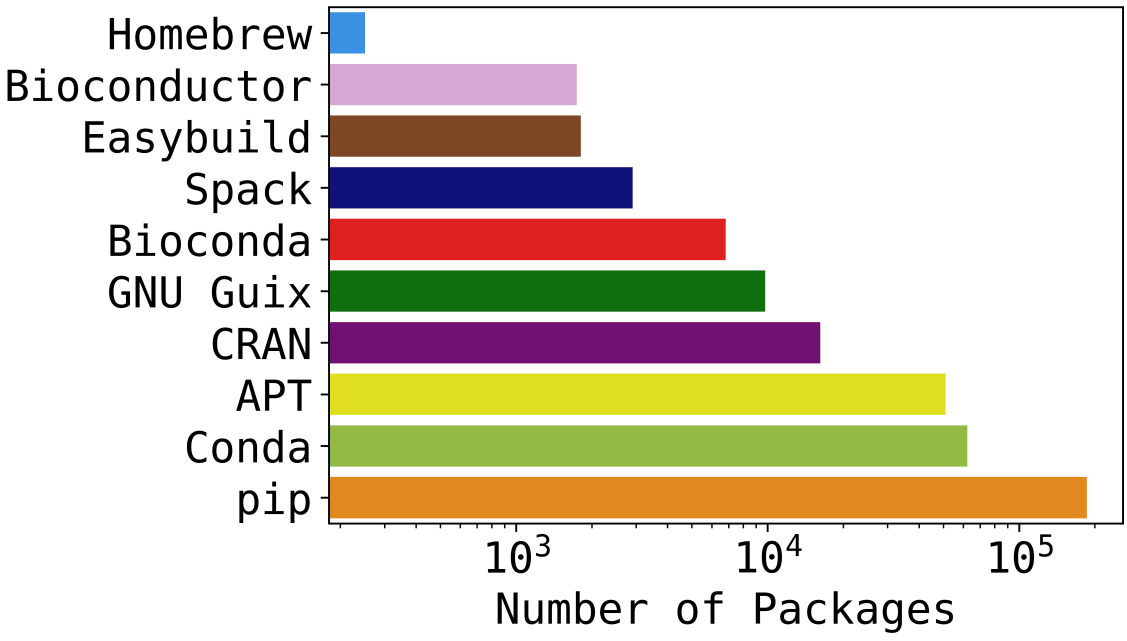

Figure 8: Total number of packages.

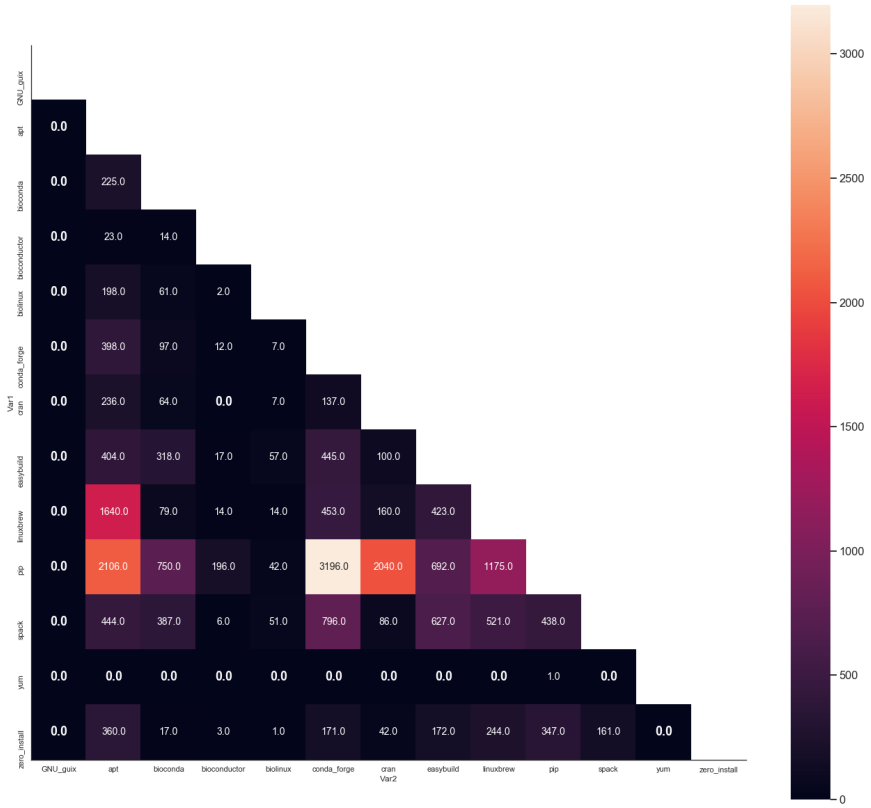

Figure 9: Packages heatmap

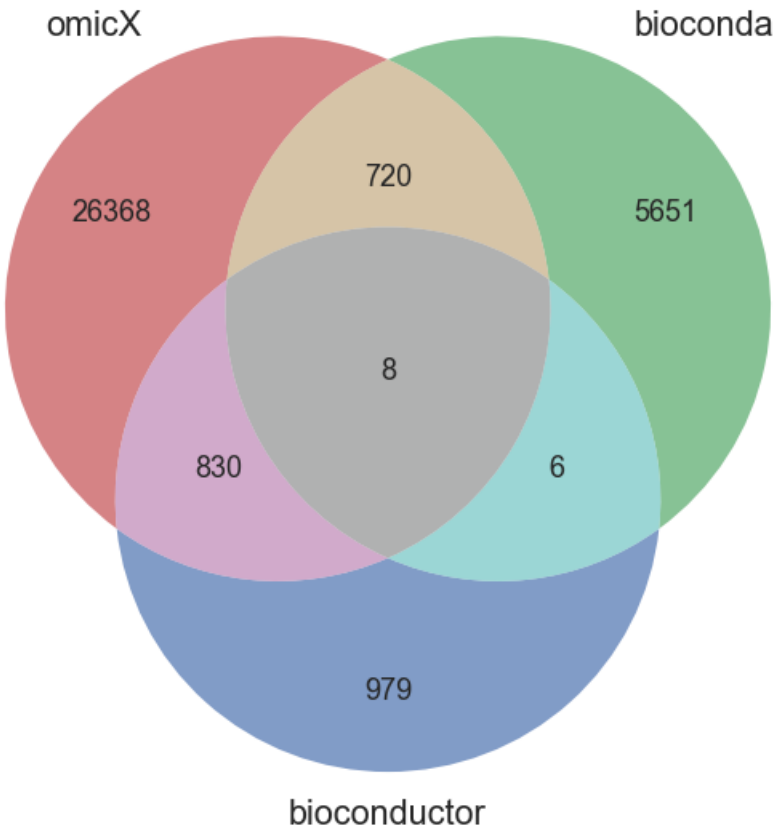

Figure 10: Overlap in available packages between omics package managers. Omics software available on omicX or Bioconda is usually only available from that source, while software available on Bioconductor is usally also available omicX. There is no omics package manager with an overwhelming majority of all sofware, so users can expect to get their software from multiple sources.

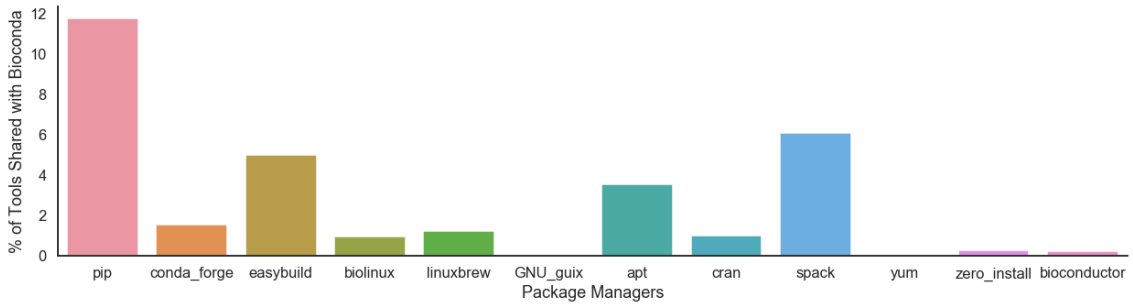

Figure 11: Availablity of Bioconda's packages in non-omics package managers.

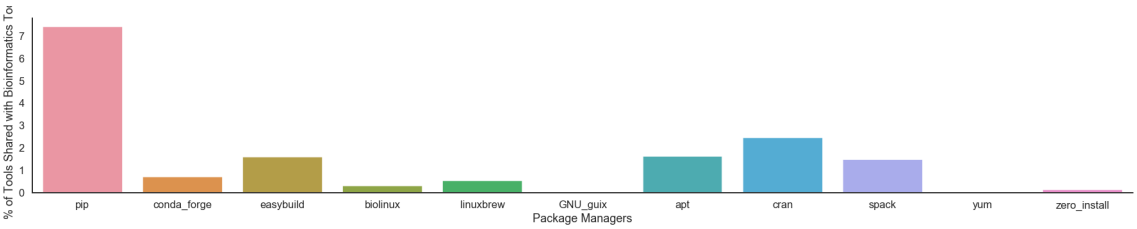

Figure 12: Availablity of Bioinformatics Tools' packages in non-omics package managers.

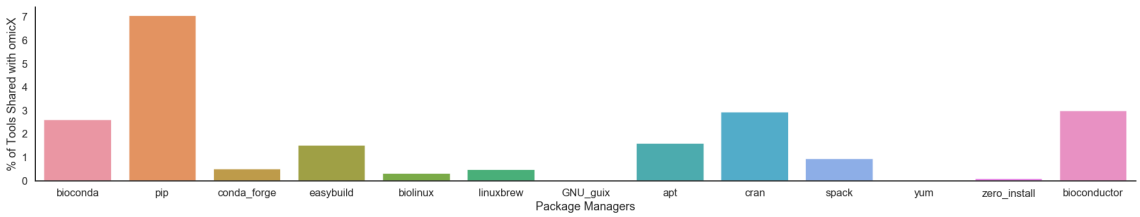

Figure 13: Availability of omicX's packages in non-omics package managers.

Methods

5 Google Trends 10/ was used to assess the relative average popularity of each package manager. Package managers were omitted when Google Trends did not have enough data to return a result. The method detailed in “How do you compare large number of items in Google Trends?” 11/ was used to overcome Google Trends’ limitation of five search terms.

6 For package managers introduced with a major publication, the number of citations for their publication was found via Google Scholar 12/ and recorded. Package managers without a major publication were excluded.

7 Each package manager’s name was used as a search term on Google Scholar 12/ and the number of results was recorded. Package manager names that returned results for things other than the package manager were omitted.

8 The initial list of all available sofware for each package manager was compiled by parsing its online repository with WebScraper 13/ (AppImage 14, Bioconda 15, Bioconductor 16, BioContainers [**???**], CRAN 17, Docker 18, EasyBuild 19, Flatpak 20, GNU Guix 21, omicX 22, pip 23, Singularity 24, Snap 25, Spack 26, Vagrant 27, and Zero Install 28) or by querying the package manager directly (APT, Biocontainers, Conda, Homebrew). Each list converted to a set with pandas 29/ and NumPy 30/, which removed all capitalization and duplicates. The number of entries in a package manager’s set was taken to be the total number of packages available for that package manager.

9 The sets obtained in 8 were compared with the set intersection function to find the number of commonalities between all pairs of package managers.

?? The sets obtained for Bioconda, Bioconductor, and omicX in 8 were compared with the set intersection function to find the number of commonalities between all combinations of the three package managers.

11 The sets obtained for Bioinformatics Tools and the non-omics package managers in 8 were compared with the set intersection function to find the number of commonalities between Bioconda and the non-omics package managers.

12 The sets obtained for Bioinformatics Tools and the non-omics package managers in 8 were compared with the set intersection function to find the number of commonalities between Bioinformatics Tools and the non-omics package managers.

13 The sets obtained for omicX and the non-omics package managers in 8 were compared with the set intersection function to find the number of commonalities between omicX and the non-omics package managers.

Glossary

Acknowledgements

Author Contributions

References

[**???**] https://docs.python.org/3.0/whatsnew/3.0.html

1. **Performance Evaluation of OS-Level Virtualization Solutions for HPC Purposes on SoC-Based Systems**
   David Beserra, Manuele Kirsch Pinheiro, Carine Souveyet, Luiz Angelo Steffenel, Edward David Moreno
   2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA) (2017-03)
   DOI: 10.1109/aina.2017.73

2. **Security of OS-level virtualization technologies: Technical report**
   Elena Reshetova, Janne Karhunen, Thomas Nyman, N. Asokan
   arXiv:1407.4245 [cs] (2014-07-16) http://arxiv.org/abs/1407.4245

3. **Empowering App Development for Developers | Docker**
   Docker
   (2020-09-03) https://www.docker.com/

4. **Linux Containers - LXC - Introduction** https://linuxcontainers.org/lxc/

5. **openvz** https://openvz.livejournal.com/

6. **Linux-VServer** http://linux-vserver.org/Welcome_to_Linux-VServer.org

7. **CoreOS** https://coreos.com/rkt/

8. **An introduction to Docker for reproducible research**
   Carl Boettiger
   ACM SIGOPS Operating Systems Review (2015-01-20) https://doi.org/10.1145/2723872.2723882
   DOI: 10.1145/2723872.2723882

9. **Container-based Cluster Orchestration Systems: A Taxonomy and Future Directions**
   Maria A. Rodriguez, Rajkumar Buyya
   arXiv:1807.06193 [cs] (2018-11-02) http://arxiv.org/abs/1807.06193

10. **Google Trends**
    Google Trends
    https://trends.google.com/trends

11. **How do you compare large numbers of items in Google Trends?**
    jonathanbriggs
    Digital Jobs To Be Done (2017-07-10) https://digitaljobstobedone.com/2017/07/10/how-do-you-compare-large-numbers-of-items-in-google-trends/

12. **Google Scholar** https://scholar.google.com/

13. **Web Scraper - The #1 web scraping extension** https://webscraper.io/

14. **Apps in AppImage format**
    AppImageHub
    AppImageHub https://appimage.github.io/apps/

15. https://bioconda.github.io/conda-recipe_index.html

16. **Bioconductor - BiocViews** http://bioconductor.org/packages/release/BiocViews.html#___Software

17. **CRAN Packages By Name** https://cran.r-project.org/web/packages/available_packages_by_name.html

18. **Docker Hub** https://hub.docker.com/search?q=&type=image&page=1

19. **List of supported software — EasyBuild v4.2.2 documentation (release 20200708.0)**
    https://easybuild.readthedocs.io/en/latest/version-specific/Supported_software.html

20. **Flathub—An app store and build service for Linux** https://flathub.org/apps/category/All

21. **Packages — GNU Guix** https://guix.gnu.org/packages/

22. https://omictools.com/software?page=1

23. **Simple index** https://pypi.org/simple

24. **singularity-hub**
    vsoch
    Singularity Hub https://singularity-hub.org/apps

25. **Install Linux apps using the Snap Store | Snapcraft**
    @snapcraftio
    Snapcraft https://snapcraft.io/store

26. **Package List — Spack 0.15.4 documentation** (2020-08-28)
    https://spack.readthedocs.io/en/latest/package_list.html

27. **Vagrant Cloud by HashiCorp**
    Vagrant Cloud by HashiCorp
    https://app.vagrantup.com

28. http://roscidus.com/0mirror/feed-list

29. **pandas - Python Data Analysis Library** https://pandas.pydata.org/

30. **NumPy** https://numpy.org/

Tables

Distribution System Name	URL	Publication	Type	License
AppImage	https://appimage.org	-	containerization	MIT
APT	https://wiki.debian.org/Apt	-	package manager	GNU GPL 2+
Bioconda	https://bioconda.github.io	Grüning et al, 2018	package manager	MIT
Bioconductor	https://www.bioconductor.org	Gentleman et al, 2004	package manager	MIT
conda	https://docs.conda.io/en/latest/	-	package manager	3-Clause BSD
CRAN	https://cran.r-project.org/index.html	-	package manager	GNU GPL
Docker	https://www.docker.com	-	containerization	Apache 2.0
Easybuild	https://easybuilders.github.io/easybuild/	Hoste et al, 2012	package manager	GNU GPL 2
Flatpak	https://flatpak.org	-	containerization	LGPL
GNU Guix	https://guix.gnu.org	Courtès, 2013	package manager	GNU AGPL
Homebrew	https://brew.sh	-	package manager	2-Clause BSD
pip	https://pypi.org/project/pip/	-	package manager	MIT
Singularity	https://sylabs.io	-	containerization	3-Clause BSD
Snap	https://snapcraft.io	-	containerization	propriertary
Spack	https://spack.io	Gamblin et al, 2015	package manager	MIT or Apache
Vagrant	https://www.vagrantup.com	-	virtual machine	MIT
yum	http://yum.baseurl.org	-	package manager	
Zero Install	https://0install.net	-	package manager	GNU LGPL 2.1+

Distribution System Name	Supported Operating Systems	Supported Languages	Root to Install	Root to Run
AppImage	Linux	any	n/a	no
APT	Debian, Ubuntu	any	yes	yes
Bioconda	Linux, macOS, Windows	any	no	no
Bioconductor	Linux, macOS, Windows	R	no	no
conda	Linux, macOS, Windows	any	no	no

Distribution System Name	Supported Operating Systems	Supported Languages	Root to Install	Root to Run
CRAN	Linux, macOS, Windows	R	no	no
Docker	Linux, macOS, Windows	any	yes	no
Easybuild	Linux	any	no	no
Flatpak	Linux	any	no	no
GNU Guix	Linux	any	no	no
Homebrew	Linux, macOS	any	no	no
pip	Linux, macOS, Windows	Python	no	no
Singularity	Linux, macOS	any	yes	no
Snap	Linux	any	yes	no
Spack	Linux, macOS	any	no	no
Vagrant	Linux, macOS, Windows	any	yes	
yum	Linux, macOS, Windows	any	no	yes
Zero Install	Linux, macOS, Windows	any	no	no

Distribution System Name	First Release	Latest Release	Age in Years	Number of Releases	Number of Tools	Number of Bio Tools
AppImage	2014-01-24	2020-06-01	7	121		
APT	1998-03-31	2020-05-08	22	362		
Bioconda	2014-01-24	2016-09-06	7	39		
Bioconductor	2002-05-01	2020-04-28	17	37		
conda	2014-01-24	2020-04-13	6	261		
CRAN	1997-04-23	2020-02-29	22	29		
Docker*	2013-03-23	2020-06-01	7	121		
Easybuild	2012-11-09	2020-04-14	7	51		
Flatpak	2015-03-23	2020-04-03	5	128		
GNU Guix	2012-07-07	2020-04-15	7	23		
Homebrew	2009-05-20	2020-05-04	10	155		
pip	2009-01-20	2020-04-28	11	81		
Singularity	2012-07-07	2020-04-15	7	23		
snapd	2014-12-09	2020-07-15	5	232		
Spack	2014-07-09	2020-04-15	5	27		
Vagrant						
yum**	2002-06-08	2011-06-28	18	221		
Zero Install	2005-02-04	2020-05-04	15	145		

*Docker Engine

**need to find someone with a redhat license who can confirm numbers

Distribution System Name	Official Repository Name	Repository URL
AppImage	AppImageHub	https://appimage.github.io/apps/
APT	-	-
Bioconda	bioconda channel	https://github.com/bioconda/bioconda-recipes
Bioconductor	-	https://www.bioconductor.org/packages/release/BiocViews.html#___Software
conda	-	https://repo.anaconda.com/pkgs/
CRAN	-	https://cran.r-project.org/web/packages/available_packages_by_name.html
Docker	Docker Hub	https://hub.docker.com
Easybuild		
Flatpak	Flathub	https://flathub.org
GNU Guix	-	https://guix.gnu.org/packages/
Homebrew	Homebrew Formulae	https://formulae.brew.sh
pip	Python Package Index (PyPI)	https://pypi.org
Singularity	Singularity Hub	https://singularity-hub.org
Snap	Snapcraft	https://snapcraft.io/store
Spack	-	-
Vagrant	Vagrant Cloud	https://app.vagrantup.com/boxes/search
yum	-	-
Zero Install	-	https://apps.0install.net