Automated Discovery of Container Executables

VANESSA SOCHAT (iD)

MATTHIEU MUFFATO (iD)

AUDREY STOTT (iD)

MARCO DE LA PIERRE (iD)

GEORGIA STUART (iD)

*Author affiliations can be found in the back matter of this article

ABSTRACT

Linux container technologies such as Docker and Singularity offer encapsulated environments for easy execution of software. In high performance computing, this is especially important for evolving and complex software stacks with conflicting dependencies that must co-exist. Singularity Registry HPC ("shpc") was created as an effort to install containers in this environment as modules, seamlessly allowing for typically hidden executables inside containers to be presented to the user as commands, and as such significantly simplifying the user experience. A remaining challenge, however, is deriving the list of important executables in the container. In this work, we present a new modular methodology that allows for discovering new containers in large community sets, deriving container entries with relevant executables therein, and fully automating both recipe generation and updates over time. As an exemplar outcome, we have employed this methodology to add to the Registry over 8,000 containers from the BioContainers community that can be maintained and updated by the software automation. All software is publicly available on the GitHub platform, and can be beneficial to container registries and infrastructure providers for automatically generating container modules, thus lowering the usage entry barrier and improving user experience.

CORRESPONDING AUTHOR:

Vanessa Sochat

Computer Scientist, Lawrence Livermore National Lab, USA

sochat1@llnl.gov

(1) OVERVIEW

1 INTRODUCTION

Containerization technologies [1, 2] have been a game-changer for reproducible research, allowing users to not only build and use reproducible environments on high performance computing (HPC) systems, but also allowing for saving the container binaries in a registry for others in the future to do the same. As HPC has matured over the decades, tools have been created that make it easier to install software to this environment, whether that be in the form of a package manager [3, 4] or module software [5, 6], which allow a user to load a namespace of commands to use. As most package managers and modules typically install software from source, the Singularity Registry HPC (shpc) [7] software was created as a marriage between the two – allowing HPC administrators and users to install containers as modules. However, the software relied on manual curation of containers to be available to install – a bottleneck that the work in this paper aims to address.

1.1 The Design of Singularity Registry HPC

Singularity Registry HPC is a command-line tool to manage installations of containers as modules. Modules are created from recipes that must live in a registry. Once the modules are installed, the shpc software is not needed to load or use the software provided by the modules. At its initial creation, each registry entry was only allowed to be a YAML file [8] in a local filesystem folder organized by the container unique resource identifier namespace. An example with the unique resource identifier "quay.io/biocontainers/samtools" is shown below.

```
$tree quay.io/biocontainers/samtools/
quay.io/biocontainers/samtools/
> container.yaml
```

The entry provides a main "container.yaml" file with an installation source (e.g., a container registry), a maintainer and description, along with container tags and digests [9], and aliases or named paths [7]. An example alias – a mapping of a term "samtools" to a path is shown below:

```
aliases:
  samtools: /usr/local/bin/samtools
```

This named path makes it easy to find useful executables within a container. The shpc software is able to provide a set of aliases that are made available to users as shell commands. This significantly lowers the barrier to container adoption in HPC, as users need to know almost nothing about container usage and syntax. For the example above, loading the container as a module would expose multiple aliases, the main one of interest being likely the "samtools" alias for the user to interact with. An example of the underlying command that the user would need to know without the shpc software is shown below in the case of the Singularity [2] container technology.

```
$ singularity <singularity-options> exec <options>
  -B <bind> <container> /usr/local/bin/samtools
  "$@"
```

Providing a full path to the "samtools" executable inside the container also ensures that, if an equivalently named executable exists on a path mounted in the container in the user's home directory, the correct one inside the container is targeted. The shpc software provides equivalent aliases for container shell, run and various inspection commands, and also supports custom container options and binds. The original set of approximately 200 entries was derived manually by the creator and contributors, and recognizing the constant change of container tags, a GitHub workflow was quickly developed to retrieve updated tags from container registries [7]. As the design evolved, there was desire for more automation and separation of responsibility between the manager software and the registry entries. As a result, the software was updated to support local or remote registries, and the main registry, along with the monthly automation to update tags, moved into a remote version-controlled repository [10].

This decoupling of the manager software from the underlying registry presented a new opportunity to extend automation. Previously, updating the entries of a registry required pulling the updated software along with the container entries from the version-control source. With a remote endpoint that serves a static API, this is no longer required – the user's local registry can retrieve the latest registry entries without updating the software, and the entries can be updated separately with tagged releases provided monthly. However, while updating tags in registry entries was automated, creating an entirely new registry entry was not. This presented a new challenge for automation, not only for the manual addition of a new container entry requested by a user, but also for the addition of potentially thousands of containers from an external source. Complete automation would not only require deriving container tags and digests, but also the executable paths.

In this work, we walk through the steps and methodology used to first semiautomate addition of a single entry, and then to automate adding thousands of containers from the BioContainers set [11]. We present supporting software to shpc [7] including a remote registry [10], software to derive and sort container versions [12], tools to discover executables within container binaries [13, 14], and a database that stores records of BioContainers executables [15]. The newly added 8,000+ containers are available for install using

the shpc software, and all work is publicly available with active automation to keep the registry updated.

2 IMPLEMENTATION AND ARCHITECTURE

2.1 Automated Recipe Generation

A semi-automated solution to generate a container entry to the online registry was introduced [7] as a workflow. It takes a container unique resource identifier, description, and website reference, and generates a pull request to the repository to add the container. To support this workflow, we developed the "guts" software [13] that knows how to pull a container, retrieve and parse the "PATH" from the image manifest [16], and then dump the container filesystem to a temporary location to search those locations statically and clean up. This procedure makes the assumption that the developer of the container has added locations of important executables to the "PATH" variable. An additional filtering step is needed to discover important executables (and not those found in the operating system provided by the container). We do an additional "diff" [17] with known executables from the most common base containers, which are provided via another automated workflow that is updated nightly [14]. The final set of executables represents a set that is unique to the changes developers made from the base containers.

To retrieve updated tags, we make a request to the Docker API [18] to retrieve a list of image tags, and retrieve associated digests via image manifests [16]. The "pipelib" software [12] is then used to sort tags by semantic versions, allowing us to filter tags and identify the "latest," a special tag provided by the shpc software as the default version to install. With this workflow to automatically derive tags and executables given any container identifier, it became possible to request a recipe for a specific container directly on the remote registry repository, and then receive a pull request with a prepared registry entry. Without any further filtering of the executables (e.g., installed dependencies unique to the container that are not of interest), the entry typically requires additional curation to filter down the discovered executables to those that are the most important. This semi-automated workflow allows for easy addition of hand-picked containers, but would require substantial work to add tens to thousands more.

2.2 Container Executable Frequency

With a request to make available more BioContainers [11], a next logical step was to figure out how to combine the semi-automated generation with a means not only to add an individual container, but also to add potentially thousands of containers from an external source. At this scale, manual edits could not be required – it would need to be possible to identify the most important executables without human curation. Addressing this challenge would require better understanding of the

distribution of executables across containers, and then determining a strategy to identify the most unique to a container. Toward this aim, we first developed a cache – the Singularity Registry HPC cache – to store a complete list of executables on the "PATH" across all BioContainers [15]. The cache is enabled by a generalized container binary discovery workflow designed to work on GitHub [19], called an action. First, the action is provided with an updated list of container binaries provided by the Galaxy Project [20]. For each container identifier, we then again use the guts software to discover all binaries on the "PATH," and save a JSON data file of the binaries to the repository, organized again by the container identifier. As a final step, we derive a second JSON data file with counts of executable names across containers. This counts data file can next be used alongside the shpc remote registry to intelligently filter down an entire set of executables in one container to a more relevant set. This entire set of actions is enabled by a few lines added to a workflow file, as shown in the shpc remote registry workflow [21].

2.3 Automated Scaled Recipe Generation

Given the availability of summary counts for over 8,000 BioContainers (see Figure 1), we observe that many containers have commands that are rare or unique to the container, and hence can deploy the following algorithm to generate a list of meaningful executables per container. This assumes that we have a regularly updated executable count cache derived from the Galaxy Project listing.

1. Identify a new container, C, not in the registry from the executable cache
2. Create a set of global executable counts, G
3. Define a set of counts from G in C as S
4. Rank order S from least to greatest
5. Include any entries in S that have a frequency <10
6. Include any entries in S that have any portion of the name matching the container identifier
7. Above that, add the next 25 executables with the lowest frequencies, and <1,000

The algorithm above assumes that the most unique executables in a container are less likely to appear in other containers, represented by a lower frequency. Always including executables that appear fewer than 10 times across the entire dataset allows for a container to have many unique commands. We chose these thresholds based on manual testing and visualization of the final list of executables, and found that these steps produced the set of binaries that we would expect or want for manual curation. We can combine programmatically derived tags and digests with these container aliases and other automated metadata to generate a final "container.yaml." From this YAML file, the shpc software
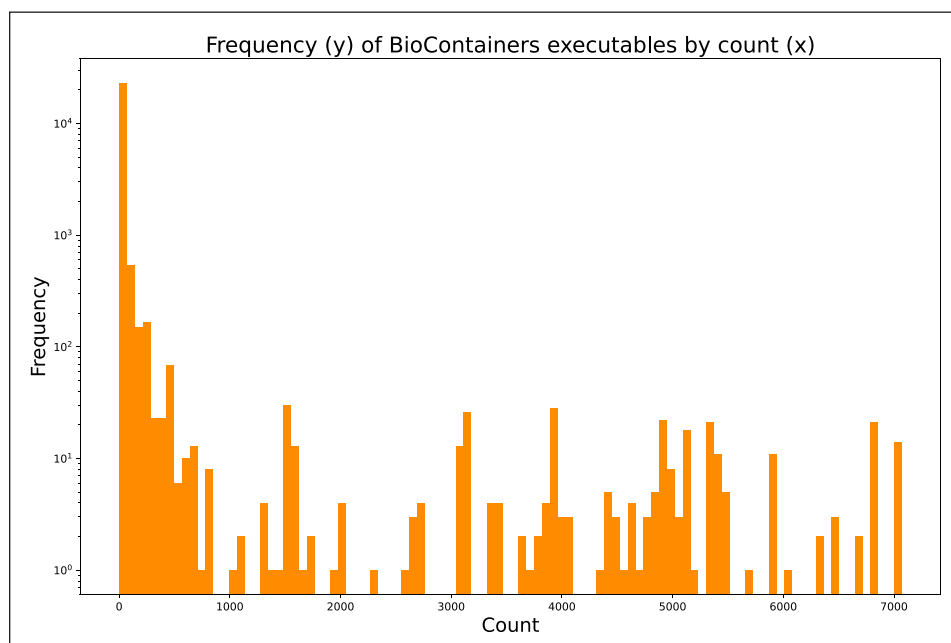
Figure 1 Frequency of BioContainers executables by count as of 04/2023. As an example, a count of "1" with a high frequency over 10,000 indicates that there are over 10,000 unique commands that appear in only one container. Manual inspection reveals that we start to see shared executables approximately after a count of 1000, and thus it serves as a good threshold for unique or "special" container commands. A Jupyter notebook was used to generate the plot [22].

can install the module to an HPC system and generate the respective executables as module commands.

2.4 Automated Recipe Updates

The original workflow to automatically update container tags and digests uses a native "update" command provided by the Singularity Registry HPC client, and this was run once a month across all containers in the current registry directly before a monthly release. However, with the addition of 8,000+ containers this monthly update would no longer be feasible within the 6 hour limit of a GitHub action runner [23]. To address this challenge, we developed a simple strategy to break up an entire list of container identifiers into equal groups, and have those groups remain consistent even given new additions to the registry. To do this, we first generate hashes for each of our container identifiers, and then generate hexdigests [24] that we convert into integer numbers. Then we take the modulus of the minimum number of days that can possibly appear in any month (N = 28) to assign each number into a specific group in the range 0–27. We add 1 to this number for a range that matches with days of the month, 1–28. On a high level, this means that we can reliably split our container identifiers into equal groups, each of which is matched to a specific day of the month. In our workflow, we can then derive the groups, take the subset for the day the workflow is running, and update that set. This algorithm is represented and provided in a GitHub action [25] for the interested reader, and the entire workflow from the addition of a new BioContainer through install of a container module on the system is included in Figure 2.

3 APPLICATIONS

We took this work to the Pawsey Supercomputing Research Centre (Pawsey) [26], a tier-1 Australian national high performance computing facility, where having these BioContainers made available as modules is perceived to vastly improve the accessibility and usage of containers in the life sciences. Through their involvement in the Australian BioCommons [27] Bring-Your-Own-Device (BYOD) Expansion Project, earlier phase discussions and surveys have highlighted repeatedly that containers are an integral part of life science research, but uptake is impeded by the lack of knowledge, confusion and time in learning about containers. Along with other Australian BioCommons partners, Pawsey role was to provide technical and compute expertise for user access to an existing Galaxy Project's repository of BioContainers images, through a read-only filesystem called CernVM-FS [28]. While this filesystem cache significantly reduces duplication of images and time for building, researchers still face the hurdle of container syntax.

Our automation of shpc recipes for BioContainers means that Pawsey, as well as other tier-1 and tier-2 partners of the Australian BioCommons, can easily install and have the same list of over 8,000 BioContainers available as modules. With a simple script to match a discovered container in the filesystem to an shpc container entry [29], these compute facilities can utilize the existing library of Singularity images through their CernVM-FS filesystem. Recipe updates provided by shpc also ensure that all new versions of BioContainers, while being added to the CernVM-FS repository, are simultaneously made available to their researchers as modules via shpc.
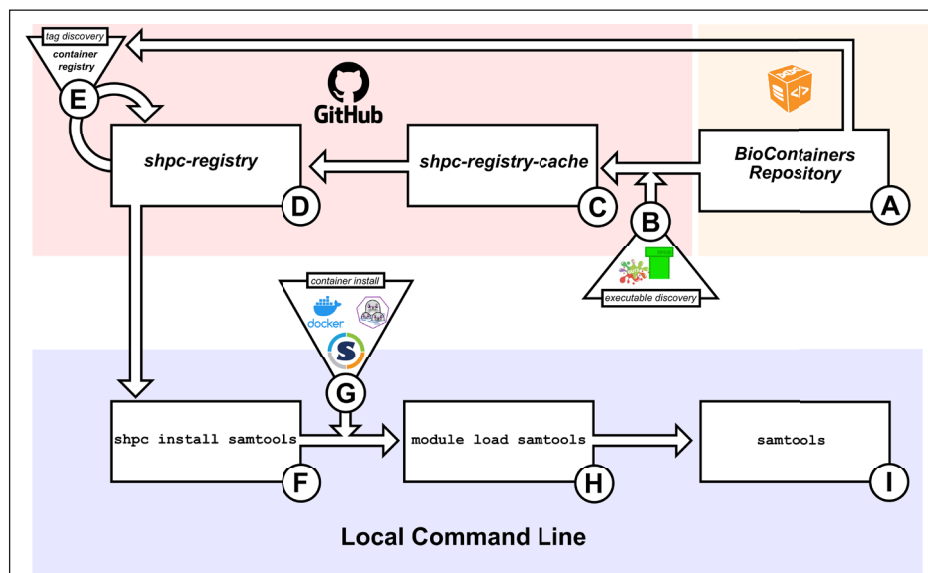
Figure 2 Movement of a new BioContainers entry from original repository through being available as a module via the shpc software. The BioContainers repository **(A)** provides an updated listing of containers from a web-accessible address. Three times a week, the container-executable-discovery action [19] **(B)** is run alongside this shpc-registry-cache [15] repository to discover new executables, derive their counts, and populate the cache **(C)**. This step uses pipelib [12] to parse and sort container tags to derive newer ones, and guts [13, 14] to extract executables on a container path. The shpc-registry [10], the remote registry with container YAML files, can then run an action provided directly by shpc to use the cache to generate new container recipes to install **(D)**. Existing recipes in the remote registry are updated in increments each day of the month to discover new tags **(E)** using an action to assign entries to days of the month [25] and the shpc software "update" command [7]. On the command line, a user that has installed shpc can then request a module to be installed from the registry. This installation pulls a container from a container registry **(G)** and installs to the system module software **(H)** where it can be loaded by a user, exposing the executables discovered in (B) for easy interaction (H).

4 QUALITY CONTROL

The algorithm was devised and tuned with a pre-existing set of 135 manual container annotations, from the Pathogen Informatics team of the Wellcome Sanger Institute. Of those, 100 matched BioContainers available in our cache. Those annotations allowed us to tweak the algorithm and the thresholds until a satisfying amount of concordance was reached. We also found that some manual annotations had been carried over from previous container versions and were missing commands added in later container versions or including commands that had since been removed, highlighting a further benefit of the proposed automation.

The container registry updates and additions are now done via an automated workflow, and manually checked by the main developer, author VS, for any changes. Lists of executables provided in the cache are spot checked by developers to ensure what is expected is there (e.g., a samtools container should minimally have the executable for samtools). Feedback comes in from the user base about executables that might be removed or added to further tweak added container recipes.

For shpc, tests are run via continuous integration for each pull request into the main branch by means of GitHub Actions. Tests span all functionality of the software across several versions of module software and container technologies.

Finally, we wanted to test that software installations made with shpc are suitable for bioinformatics analyses.

We adapted Nextflow's RNA-seq pipeline [30] to introduce a "shpc" profile that uses modules created by shpc. This profile [31] generates identical files to Nextflow's "singularity" profile, which directly calls Singularity. We found the results to be identical.

(2) AVAILABILITY

OPERATING SYSTEM

Singularity Registry HPC and associated tooling should work on most Unix and Linux flavored distributions. The software was developed on Ubuntu 22.04.

PROGRAMMING LANGUAGE

This set of tools is developed to support Python 3.7 and higher. Python 2.x is not supported.

DEPENDENCIES

The newly released cache and automation can run on GitHub Actions with the environment encapsulated by the runner. The shpc set of tools requires the requests library, jsonschema, and generally expects module software to be installed [6, 5]. See the "version.py" in each project for details. Naturally, shpc also requires a container execution runtime, such as Singularity [2], Podman [32] or Docker [1]. Note that any container technology that supports pull by a tag or digest can be integrated into the software upon request.

LIST OF CONTRIBUTORS

Authors VS, MM, AS, MDLP, and GS have all contributed directly to the software.

SOFTWARE LOCATION: ARCHIVE

- Name: shpc-registry
  - Identifier: https://zenodo.org/record/7709586
  - License: MPL-2.0
  - Publisher: Vanessa Sochat
  - Version published: 2022-11
  - Date published: 2 November 2022
- Name: shpc-registry-cache
  - Identifier: https://zenodo.org/record/7709593
  - License: MPL-2.0
  - Publisher: Vanessa Sochat
  - Version published: 2022-10
  - Date published: 21 November 2022
- Name: guts
  - Identifier: https://zenodo.org/record/7703378
  - License: MPL-2.0
  - Publisher: Vanessa Sochat
  - Version published: 0.0.1
  - Date published: 07 March 2023
- Name: shpc-guts
  - Identifier: https://zenodo.org/record/7703380
  - License: MPL-2.0
  - Publisher: Vanessa Sochat
  - Version published: 0.0.1
  - Date published: 07 March 2023
- Name: pipelib
  - Identifier: https://zenodo.org/record/7703389
  - License: MPL-2.0
  - Publisher: Vanessa Sochat
  - Version published: 0.1.15
  - Date published: 07 March 2023
- Name: singularity-hpc
  - Identifier: https://zenodo.org/record/7644421
  - License: MPL-2.0
  - Publisher: Vanessa Sochat
  - Version published: 0.1.16
  - Date published: 5 November 2022

SOFTWARE LOCATION: CODE REPOSITORY

All repositories are hosted on https://github.com.

- Name: shpc-registry
  - Identifier: singularityhub/shpc-registry
  - License: MPL-2.0
  - Date published: 31 July 2022
- Name: shpc-registry-cache
  - Identifier: singularityhub/shpc-registry-cache
  - License: MPL-2.0
  - Date published: 18 October 2022
- Name: guts
  - Identifier: singularityhub/guts
  - License: MPL-2.0
  - Date published: 07 March 2023
- Name: shpc-guts
  - Identifier: singularityhub/shpc-guts
  - License: MPL-2.0
  - Date published: 07 March 2023
- Name: pipelib
  - Identifier: vsoch/pipelib
  - License: MPL-2.0
  - Date published: 07 March 2023
- Name: singularity-hpc
  - Identifier: singularityhub/singularity-hpc
  - License: MPL-2.0
  - Date published: 3 April 2021

SOFTWARE LOCATION: LANGUAGE

English.

(3) REUSE POTENTIAL

The documentation pages [33] for Singularity Registry HPC have been updated to include a developer tutorial [34] that guides through the steps to setup and deploy a registry of container modules based on automatic recipe generation and executable lookup. Separate automated workflows exist for cache generation [35] and for using a cache to update a registry [15, 19]. In this way, container registries and infrastructure providers can reuse the software presented in this manuscript to automatically generate a registry for a custom collection of containers that is of relevance for their operations and activities. Some parameters of the protocol can be modified to tune the resulting registry; for instance, this flexibility applies to the filtering of container tags/digest, and to the selection of container executable based on executable frequencies.

Two package modules that constitute part of the presented toolkit, pipelib [12] and guts [13], can be reused independently for purposes that go beyond the generation of container registries. The former offers functionalities to filter containers tags and digests, whereas the latter allows to extract information on available executables in a container.

Finally, it is also worth mentioning the usability impact of the real use case presented as application for the presented software. The collection of 8,000 recipes to generate container modules for the whole collection of BioContainers represents a valuable, impactful resource for the community of life scientists, as well as that of infrastructure providers and support staff that offer services to the former.

All packages are open source on GitHub, and contributions and ideas are welcome. We suggest to the interested reader to open GitHub issues for help or support on any of the software components.

(4) CONCLUSION

4.1 SUMMARY

In this work, we present complete automation to support and continually update a set of over 8,000 container entries for installation to an HPC system using the Singularity Registry HPC software. Our interesting contributions that we desire to share with the community include:

- Singularity Registry HPC, with support for remote registries and automated updates [7];
- shpc-registry, a self-updating, version-controlled static container registry and API of container metadata [10];
- shpc-registry-cache, a self-updating, version-controlled database of executable frequencies [15];
- the guts software to extract container executables on the "PATH" [13];
- the pipelib software to intelligently filter and sort container tags [12];
- a library of over 8,000 containers to install to an HPC system with Singularity Registry HPC [7].

This manuscript presents as a strong example of a research software paper, as the primary focus is on the development of workflows, interfaces, and software to support installing software to complex environments. We hope that any of the automation, data, or software presented is of use or interest to the larger community.

4.2 DISCUSSION

The algorithm presented in this paper is a necessity caused by the lack of standard metadata for describing the content of a package or container. Ideally, there should be a machine-readable manifest that would list the primary content (installed by "make install" or equivalent), the dependencies, and the base image. This could be tackled first within the Conda [36] system. The Conda build system knows which binaries are installed by a given package and what its dependencies are. The listing could be exposed at the package level. Such metadata would take research software closer to the "FAIR principles for research software" [37] (FAIR stands for Findable, Accessible, Interoperable, Reusable). All BioConda packages (and many from other channels) are automatically turned into Docker images by automation at BioContainers [38]. The build could load those manifests into standard container labels that shpc could then use to derive the commands to expose.

It is expected that some recipes created by the algorithm have too few or too many aliases. Being the official shpc registry openly hosted on GitHub, contributions are welcome in the form of pull requests to modify the list of aliases, and we invite the research community to report any error they find. This curation process will happen concurrently to the regular update of tags and digests.

DATA ACCESSIBILITY STATEMENT

Data and code relevant to this publication are all available publicly through the supporting Github repositories [7, 10, 12, 13, 14, 15].

ACKNOWLEDGEMENTS

FUNDING INFORMATION

COMPETING INTERESTS

The authors have no competing interests to declare.

AUTHOR CONTRIBUTIONS

Author VS designed and created the Singularity Registry HPC software, pipelib, and guts, and the automation detailed in this work. MM, AS, MDLP, and GS contributed to the codebase and design of the Singularity Registry HPC software and the automation.

AUTHOR AFFILIATIONS

Vanessa Sochat orcid.org/0000-0002-4387-3819
Computer Scientist, Lawrence Livermore National Lab, USA

Matthieu Muffato orcid.org/0000-0002-7860-3560
Tree of Life, Wellcome Sanger Institute, Cambridge, UK

Audrey Stott orcid.org/0000-0003-0939-3173
Pawsey Supercomputing Research Institute, Kensington, WA, Australia

Marco De La Pierre orcid.org/0000-0002-7429-1846
Pawsey Supercomputing Research Institute, Kensington, WA, Australia

Georgia Stuart [ID] orcid.org/0000-0003-2787-5299
University of Massachusetts, Amherst, MA, USA

REFERENCES

1. **Ratliff J.** Docker: Accelerated, Containerized Application Development. https://www.docker.com/, 2022. Accessed: 2022-10-25.

2. **Kurtzer GM, Sochat V, Bauer MW.** Singularity: Scientific containers for mobility of compute. *PLoS One*. May 2017; 12: e0177459. DOI: https://doi.org/10.1371/journal.pone.0177459

3. **Gamblin T, LeGendre M, Collette MR, Lee GL, Moody A, De Supinski BR, Futral S.** The Spack package manager: bringing order to HPC software chaos. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 2015; 1–12. DOI: https://doi.org/10.1145/2807591.2807623

4. **Hoste K, Timmerman J, Georges A, De Weirdt S.** Easybuild: Building software with ease. In: *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*. IEEE. 2012; 572–582. DOI: https://doi.org/10.1109/SC.Companion.2012.81

5. **Furlani JL.** Modules: Providing a flexible user environment. In: *Proceedings of the fifth large installation systems administration conference* (LISA V). 1991; 141–152.

6. Lmod: A New Environment Module System — Lmod 8.7.13 documentation. 2022. https://lmod.readthedocs.io/en/latest/. Accessed: 2022-10-25.

7. **Sochat V, Scott A.** Collaborative Container Modules with Singularity Registry HPC. *Journal of Open Source Software*. 2021; 6(63): 3311. DOI: https://doi.org/10.21105/joss.03311

8. YAML Specification Index. 2022. https://yaml.org/spec/. Accessed: 2022-10-25.

9. **Savanth V.** Docker images: Name vs. tag vs. digest. Sept. 2021. https://hackernoon.com/docker-images-name-vs-tag-vs-digest. Accessed: 2022-11-4.

10. shpc-registry: Testing a remote registry for Singularity Registry HPC. 2022. https://github.com/singularityhub/shpc-registry.

11. **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.** BioContainers: an open-source and community-driven framework for software standardization. *Bioinformatics*. Aug. 2017; 33: 2580–2582. DOI: https://doi.org/10.1093/bioinformatics/btx192

12. pipelib: a library for creating pipelines for parsing, filtering, and sorting iterables. 2022. https://github.com/vsoch/pipelib.

13. guts: Actions and client to derive container guts! 2022. https://github.com/singularityhub/guts.

14. shpc-guts: Singularity Registry HPC... container guts! 2022. https://github.com/singularityhub/shpc-guts.

15. shpc-registry-cache: A cache of commands (currently for biocontainers). 2022. https://github.com/singularityhub/shpc-registry-cache.

16. opencontainers/image-spec manifest. 2022. https://github.com/opencontainers/image-spec/blob/main/manifest.md.

17. **Wikipedia contributors.** diff. Oct. 2022. https://en.wikipedia.org/w/index.php?title=Diff&oldid=1118832554. Accessed: 2022-12-8.

18. Docker hub HTTP API V2. Nov. 2020. https://docs.docker.com/registry/spec/api/. Accessed: 2020-11-23.

19. container-executable-discovery action: GitHub action to assist in creating a cache of container executables. 2022. https://github.com/singularityhub/container-executable-discovery/. Accessed: 2022-11-29.

20. **Afgan E, Baker D, Batut B, Van Den Beek M, Bouvier D, Čech M, Chilton J, Clements D, Coraor N, Grüning BA,** et al. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic acids research*. 2018; 46(W1): W537–W544. DOI: https://doi.org/10.1093/nar/gky379

21. shpc-registry workflow to update Biocontainers. 2022. https://github.com/singularityhub/shpc-registry/blob/main/.github/workflows/update-biocontainers.yaml. Accessed: 2022-11-29.

22. looking-at-counts.ipynb at shpc-registrycache repository. 2022. https://github.com/singularityhub/shpc-registry-cache/blob/main/looking-at-counts.ipynb. Accessed: 2023-03-16.

23. Usage limits, billing, and administration. 2022. https://docs.github.com/en/actions/learn-github-actions/usage-limits-billing-and-administration.

24. hashlib — secure hashes and message digests — python 3.11.0 documentation. 2022. https://docs.python.org/3/library/hashlib.html. Accessed: 2022-11-4.

25. split-list-action: Simple GitHub action to evenly split (and present a subset) of items based on random selection or the day of the month! 2022. https://github.com/vsoch/split-list-action. Accessed: 2022-11-8.

26. Pawsey Supercomputing Research Centre. 2022. https://pawsey.org.au. Accessed: 2022-11-8.

27. Australian BioCommons. 2022. https://biocommons.org.au. Accessed: 2022-11-8.

28. CernVM File System. 2022. https://cernvm.cern.ch/fs/. Accessed: 2022-11-8.

29. biocontainer-match.py at main shpc repository. 2022. https://github.com/singularityhub/singularity-hpc/blob/main/example/biocontainer-match.py. Accessed: 2022-11-16.

30. A proof of concept of RNA-seq pipeline, written in Nextflow. 2022. https://github.com/nextflow-io/rnaseq-nf.

31. Demo/toy RNAseq pipeline, to test interplay of Nextflow with SHPC container modules. 2022. https://github.com/marcodelapierre/demo-shpc-nf.

32. **Walsh D.** The Pod Manager tool (podman). 2022. https://podman.io/. Accessed: 2022-11-26.

33. Singularity registry hpc – documentation. 2022. https://singularity-hpc.readthedocs.io. Accessed: 2022-12-8.

34. Singularity registry hpc – developer tutorial. 2022. https://singularity-hpc.readthedocs.io/en/latest/getting_started/developer-guide.html#developer-tutorial. Accessed: 2022-12-14.

35. container-executable-discovery cache creation action: GitHub action to assist in creating a cache of container executables. 2022. https://github.com/singularityhub/shpc-registry-cache/blob/main/.github/workflows/update-cache.yaml. Accessed: 2022-11-29.

36. Conda: Package, dependency and environment management for any language. 2022. https://docs.conda.io/.

37. **Barker M, Chue Hong NP, Katz DS, Lamprecht A-L, Martinez-Ortiz C, Psomopoulos F, Harrow J, Castro LJ, Gruenpeter M, Martinez PA, Honeyman T.** Introducing the FAIR Principles for research software. *Scientific Data*. Oct. 2022; 9: 622. DOI: https://doi.org/10.1038/s41597-022-01710-x

38. **Bray S, Chilton J, Bernt M, Soranzo N, van den Beek M, Batut B, Rasche H, Čech M, Cock PJA, Grüning B, Nekrutenko A.** The Planemo toolkit for developing, deploying, and executing scientific data analyses in Galaxy and beyond. *Genome Research*. Jan. 2023; 33: 261–268. DOI: https://doi.org/10.1101/gr.276963.122