The Journal of Open Source Software

udocker: a user oriented tool for unprivileged Linux containers

Jorge Gomes[*1, 2], **Mario David**[1, 2], **João Martins**[1, 2], **João Pina**[1, 2], **Samuel Bernardo**[1, 2], **Isabel Campos**[3], **Pablo Orviz**[3], and **Alvaro López-García**[3]

1 LIP, Laboratório de Instrumentação e Física Experimental de Partículas, Lisboa, Portugal **2** INCD, Infraestrutura Nacional de Computação Distribuída, Lisboa, Portugal **3** IFCA, Consejo Superior de Investigaciones Científicas-CSIC, Santander, Spain

DOI: 10.21105/joss.03295

Software

- Review
- Repository
- Archive

Editor: Daniel S. Katz

Reviewers:

- @vsoch
- @mviereck

Submitted: 12 May 2021
Published: 16 June 2021

Summary

Containers are increasingly used to package, distribute and run scientific software. udocker is a tool to enable execution of Linux containers in advanced computing environments. Distinctively from other tools, udocker is meant for easy deployment and provides multiple execution engines to cope with different host environments. udocker can execute containers with or without using Linux namespaces. udocker is being used by a wide range of projects and research communities to facilitate the execution of Linux containers across heterogeneous computing environments.

Statement of need

Researchers have at their disposal a wide range of computing resources ranging from laptops to high performance computing clusters and cloud services. Enabling execution of scientific codes across such resources often requires significant effort to adapt to the underlying system configurations. This can be particularly difficult for codes with complex software dependencies and can become a continuous effort due to system changes and software updates. Furthermore ensuring the reproducibility across heterogeneous computing resources can be challenging when the software needs to be adapted to the specificity of each resource. In this context Linux containers have gained interest as means to enable encapsulation of research software for easier execution across these environments.

udocker is designed to address the requirement of executing scientific applications easily across a wide range of computing systems and digital infrastructures where the user may not have administration privileges, and where tools and functionalities to support Linux containers may not be available. In addition, udocker also simplifies the researcher interaction with the tools required to execute containers by providing an integrated solution to execute Linux containers leveraging different approaches suitable for unprivileged users. Finally by executing containers without privileges udocker decreases the risks of privilege escalation. The udocker development started in 2016 and the original udocker paper (Gomes et al., 2018) documented the initial versions up to 1.1.1.

*corresponding author

Concept

udocker provides a self contained solution with minimal dependencies to enable execution across systems without need of source code compilation. udocker itself was initially implemented in Python 2 and later ported to Python 3.

udocker implements pulling, importing and loading of *docker* or OCI containers to a local repository in the user home directory. The layers composing a container image can then be sequentially extracted to create a flattened directory tree. Furthermore udocker also provides the logic to interface with the several execution engines that enable the execution of code extracted from the container images, thus hiding as much as possible the execution engines specificity. The execution engines are based on existing open source software that in several cases has been significantly improved, integrated and packaged to be used with udocker. The following engines are currently provided:

- **F** engine: uses the Linux shared library PRELOAD mechanism to intercept shared library calls and translate pathnames to provide an unprivileged chroot like functionality. It is implemented by an extensively enhanced *Fakechroot* shared library with versions for the *glibc* (Gomes, 2019a) and *musl* (Gomes, 2019b) *C* standard libraries. This approach requires the modification of pathnames in the ELF headers of shared libraries and executables. These changes are performed by udocker using a modified *Patchelf* (Gomes, 2020). This is the execution engine that generally provides the highest performance.
- **P** engine: uses the Linux PTRACE mechanism to implement a chroot like environment by intercepting system calls and translating pathnames. It is implemented by a modified *PRoot* (Gomes, 2021). This engine provides the highest interoperability across Linux distributions both older and newer, and constitutes the default execution engine for udocker.
- **R** engine: uses either *runc* (opencontainers.org, 2021) or *crun* (Scrivano, 2021) to execute the containers without privileges using Linux user namespaces. Both tools are provided with udocker for wider interoperability.
- **S** engine: uses *Singularity* (Kurtzer, 2017) to execute the containers using user namespaces or other *Singularity* supported execution method depending on the system configuration.

All required commands are statically compiled for execution across a wide range of systems. The shared libraries for the **F** modes are also compiled and provided for major Linux distributions. The **F** modes require the compilation of the libraries against each *libc* and therefore requires creation of different libraries for each release of a given distribution.

Support for the ARM architecture is provided for the **P** mode and is ongoing for the other modes. The binaries for the **S** engine are not provided with udocker, as this mode is provided to take advantage of local installations of *Singularity* where available.

Once the udocker Python code is transferred to the target host it can be used by an unprivileged user to download the additional executables and binaries into the user home directory. The user can then use udocker to pull images, create container directories from the images and execute them. Each extracted container can be easily setup for execution using any of the execution engines. udocker provides a command line interface with a syntax similar to docker.

Compared with other container tools that can enable unprivileged execution such as *podman*, *docker*, or *Singularity* among others, udocker is unique in offering multiple execution engines for unprivileged execution, two of these engines are based on pathname translation not requiring kernel features such as Linux user namespaces thus enabling execution across a wider range of systems and services where user namespaces are unavailable. The Linux user namespaces approach also has limitations and may create problems when accessing host files via bind mount due to the usage of subordinate uid and gid identifiers. These limitations extend to system calls that may return uid and gid or when credentials are passed across sockets. In addition user namespaces still expose code in the kernel to normal users that was previously only really accessible to root creating opportunities for new vulnerabilities to arise. If isolation between the container and the running host is important then namespaces provide the highest level of isolation at the expense of the described risks and limitations. For the users that wish to rely on Linux namespaces udocker also offers support for this approach through *runc* and *crun* or through *Singularity* if locally installed. The tools that can execute containers using `chroot` or `pivot_root` and use privileges such as *Shifter*, *Sarus* (Benedicic L., 2019) or the original mode of *Singularity* have the limitation of requiring installation and configuration by a system administrator and of having a higher risk of privilege escalation as privileges are used in some operations. Since these tools run with privileges they can use approaches such as using *squashfs* to improve file access. On de other hand udocker is focused on deployment and execution entirely by the end-user and thus cannot provide features that require privileges.

Developments since 1.1.1

udocker was initially developed in the context of the INDIGO-DataCloud (Salomoni et al., 2018) research project between 2015 and 2017 as a proof of concept aimed to show that scientific applications could be encapsulated in Linux containers to ease execution across the growing ecosystem of computing resources available to researchers including Linux batch systems and interactive clusters. In particular it aimed to show that containers could be executed by the end-users without requiring changes to the computing systems and without system administrator intervention, thus empowering users and promoting the adoption of containers in these environments.

Being a proof of concept the initial versions were not designed for production use. Later in the project it become evident that udocker had gain adoption beyond its original purpose and scope and that is was already being actively used in production environments. After the first udocker publication (Gomes et al., 2018) produced using versions 1.1.0 and 1.1.1, the development effort was directed to enhance udocker for production use by improving the design, robustness and functionality. Two code branches became supported in parallel. The *devel* branch for the production versions 1.1.x retained the original proof of concept code for Python 2, while the *devel3* branch supported the development of the new modular design with support for Python 3 that later gave origin to the 1.2.x pre-releases. The version 1.3.0 released in June of 2021 is the first production release having the new design and support for Python 2 and 3.

Since version 1.1.1 the udocker code was reorganized, largely rewritten and improved. Starting with the 1.2.0 pre-release, udocker was completely restructured moving from being a single large monolithic Python script to become a modular Python application, making maintenance and contributions easier. The new code structure has 40 Python modules and supports both Python 2.6 or higher and Python 3. Since container technologies are in constant evolution, this new code structure was essential to accommodate any future improvements such as new container formats, APIs and execution engines as they become mainstream.

The improvements added after 1.1.1 include a more robust command line interface addressing several of the problems that affected the initial versions in terms of command line parsing and validation of arguments. The parsing of the configuration files was reimplemented to simplify the design and prevent injection of code via the configuration files. Configuration is now possible at three levels, system configuration via `/etc/udocker.conf`, user configuration via `$HOME/.udocker/udocker.conf` and udocker repository level via `$UDOCKER_DIR/udocker.conf`. The most relevant configuration options can now be overridden through new environment variables.

- `UDOCKER_DEFAULT_EXECUTION_MODE`: to change the default execution engine mode, which is currently **P1** using *PRoot*.
- `UDOCKER_FAKECHROOT_SO`: to enforce the use of a specific *Fakechroot* shareable library for use in **F** execution modes.
- `UDOCKER_USE_CURL_EXECUTABLE`: to select a *curl* executable as alternative to `pycurl` for downloads and interaction with REST APIs.
- `UDOCKER_USE_PROOT_EXECUTABLE`: to enforce the use of a given *PRoot* executable for use in **P** execution modes.
- `UDOCKER_USE_RUNC_EXECUTABLE`: to enforce the use of a given *runc* or *crun* executable for use in **R** execution modes.
- `UDOCKER_USE_SINGULARITY_EXECUTABLE`: to enforce the use of a given *Singularity* executable for use in **S** execution modes.
- `UDOCKER_FAKECHROOT_EXPAND_SYMLINKS`: to control the expansion of symbolic links in paths pointing to volumes when using the **F** modes. This variable allows to disable the new path translation algorithm that is now accurate but slower.
- `PROOT_TMP_DIR`: is now supported and correctly passed to the *PRoot* execution engine.

The variables used to control the choice of images and libraries reflect the new automated selection of the engine executables and libraries based on system architecture, kernel version, and Linux distribution of both the host and container. This selection is performed automatically but can be overridden by the corresponding environment variables. Support in udocker to select the execution engine binaries for the architectures *x86_64*, *aarch64*, *arm* 32bit and *i386* was added. However the corresponding binaries must be provided and placed under `$HOME/.udocker/bin` for executables and `$HOME/.udocker/lib` for libraries. Currently the external tools and libraries compiled and provided with udocker support *x86_64*, *aarch64*, *arm* 32bit and *i386* for use with the **P** modes. The binaries for the remaining execution modes are currently only provided for *x86_64* systems, this may change in the future as these and other architectures become more widely used.

The **F** mode is particularly unique to udocker. It relies on the interception of shared library calls using a modified *Fakechroot* shared library. By default *Fakechroot* requires the same libraries and dynamic loader both in the host and in the `chroot` environment. The *Fakechroot* libraries modified for udocker in combination with udocker itself enable the execution of containers whose shared libraries and dynamic loader can be completely different from the ones used in the host system. After version 1.1.1 the *Fakechroot* implementation of udocker was much improved to enable these scenarios. A complete porting of the *Fakechroot* libraries was performed for the *musl libc*, enabling support for containers having code compiled against *musl libc* such as *Alpine* based containers. The original *Fakechroot* implementation is very limited in terms of mapping host pathnames to container pathnames. A host pathname can only be passed to the `chroot` environment if the pathname remains the same, (e.g. the host /dev can only be mapped to the container /dev). This is a strong limitation as the host pathnames may need to be mapped to different container locations. Implementing a complete mapping required extensive modifications to *Fakechroot* that were only completed for the libraries distributed with udocker version 1.1.6. Also in the **F** modes, udocker must apply changes to the ELF headers of executables and libraries. To this end a modified version of `patchelf` is used. The set of changes required included the ability to perform all header modifications in a single step, the original version had to be invoked as many times as the number of required changes. The changes required to the shared objects include the pathname to the system loader and the pathnames for shared libraries. Support for the handling of loader string tokens such as `$ORIGIN` that were previously ignored also had to be added. The complete functionality for *Fakechroot* became available with udocker 1.1.6. The shared libraries must be compiled against the *libc* of the container environment. Therefore the range of libraries provided has been growing constantly since the initial versions. Libraries to support new distributions and releases have been regularly added. This effort includes any necessary updates to *Fakechroot* such as adding support for new C standard library calls as required.

The **P** mode is based on *PRoot* and is the original execution engine supported since version 1.0.0. As shipped with udocker it offers a transparent method to execute containers across Linux distributions that conversely to the **F** mode based on *Fakechroot* does not require changes to the container binaries. Since the pathname translation is performed at the system call level, the same *PRoot* statically compiled executable can be used across a wide range of distributions and versions. Still care must be taken to dynamically adapt to the underlying kernel capabilities. Since version 1.0.1 the support for syscall interception using PTRACE and SECCOMP had to be modified to cope with kernel changes. This was a major issue the deeply affected the performance of *PRoot* and for which no upstream solution existed. A first incomplete fix was created by the udocker developers and introduced with 1.0.1. The complete implementation only became available with udocker 1.1.4, which was later extended in 1.1.7 to address the special case of distributions that backported the PTRACE kernel patches to previous versions of the kernel. In addition support for several new system calls had to be incorporated including *faccessat2()*, *newfstatat()*, *renameat()* and *statx()*. Emulation was also added enabling the execution of code invoking these calls in kernels where they are unavailable. This capability is allows applications that use newer systems calls to still work on older Linux releases where a *kernel too old* error would be issued.

The **R** execution mode was originally implemented by using *runc* in rootless mode. In this mode udocker creates the require configurations for *runc* to execute containers without requiring privileges using the Linux user namespace. In version 1.1.2 support for pseudo ttys was added to udocker for *runc* enabling execution in batch systems and other environments without a terminal. In version 1.1.4, the support for *crun* was also introduced. While *runc* is written in *go*, *crun* is written in *C* and is generally faster. Furthermore *crun* provided support for the kernel *cgroups* version 2 which became required in some distributions. Both tools are now provided statically compiled with udocker and the Python code was enhanced to support both.

udocker implements its own code to manipulate container images and interact with container repositories. The initial versions were limited to the Docker image format and were largely tied to *DockerHub*. Since then effort was put to improve the implementation of the Docker Registry API making it interoperable with other container repositories. Support for the OCI images according to the v1 specification was also added on version 1.1.4 improving interoperability. This improvement in repository interoperability led to the change of the schema used in the container image names. Since 1.1.4 the container image names can include a hostname component (e.g. hostname/repository:tag). The new name schema improved interoperability further and made easier the usage of repositories other than *DockerHub*, however it also required changes across the container image handling code and also in the command line interface.

The search functionality was reimplemented to support both the registry API v1 and v2 using `/v2/search/repositories`. In addition support to list image tags was also implemented as part of the search command. Support for the use of proxies in image searches was added enabling both `search` and `pull` of containers via socks proxies. The handling of http redirects was also implemented inside udocker to address shortcomings that affected some releases of *curl* and consequently also *pycurl*.

Also in version 1.1.4 the checksumming of container layers was improved, the *sha512* hash was added and the code was restructured to accommodate multiple hash algorithms as they may became available. The verification of container images implemented by the `verify` command was also improved to include all supported container image formats performing both the structure validation and the file checksumming where applicable.

The new udocker commands introduced since 1.1.1 include the `save` of container images to file or standard output, `rename` to change the name of a created container and `clone` to duplicate a created container including its changes and retaining udocker specific configurations. Several existing commands got new flags such as `run` where `--env-file=filename` enables reading environment variables from a file, `--device` adds additional host devices to container when using the **R** execution modes, and `--containerauth` that prevents the default udocker behavior of adding the invoking user to the container password and group files. The handling of entrypoint information provided via `run --entrypoint` or through the container metadata was changed in version 1.3.0 to match the *docker* behavior and allow bypassing the container *entrypoint*. The ps command got two new flags, `-s` to list the size of the created containers, and `-m` to list the execution engine configured for each created container, which is particularly useful since the execution mode is defined per container. The `setup` command used to configure the created containers also got new flags, namely `--purge` to remove files created within a container by such as mount points, and `--fixperms` to fix the permissions and also the ownership of files created by the **R** execution modes while using user namespaces. To this end udocker provides is own Python implementation of *unshare* to enable the removal of files owned by different subordinated uid or gid identifiers.

The `setup` command was also enhanced with the `--nvidia` flag, that provides am `nvidia-docker` like capability for udocker providing support for the execution of GPU accelerated applications across different hosts systems. For `udocker` this functionality needs to take into account the characteristics of each execution engine. While in some engines the required host pathnames can be transparently mapped into the container, in other modes this may require creation of mount points or the copy of the actual host files to the container. These requirements are now handled transparently as part of the udocker volume handling. Thanks to these improvements udocker has been increasingly used to support accelerated computing applications and in particular machine learning.

udocker has been successfully used in environments where conventional container tools cannot be used, such as when namespaces are not available or privileges are required. These include running containers within *docker* itself, and running within services and applications such as *AWS lambda*, *google colab* (Google, 2021) or *Termux* (Developers, 2021). Several enhancements were introduced since version 1.1.1 to enable the usage of udocker within this type of environments using the pathname translation approaches provided by the **P** and **F** execution engines.

The external tools and libraries used by udocker to support the execution engines are distributed in binary format in a package that was released simultaneously with udocker. The handling of the versions of both udocker and of the package was decoupled to make possible the release of new tools and libraries without requiring a new release of udocker. For each udocker version there is now a minimum release of the package containing the tools and libraries. If available new versions of the package can be installed or updated using the `install` command. The resilience of the installation process was improved and better recovery from download errors was implemented. The extraction of documentation and software licenses from the package was included as part of the installation process. The documentation is now extracted to a new directory `$HOME/.udocker/doc`. In addition the command `version` was added to display the versions and the locations from which the package containing the tools and libraries can be obtained. udocker itself can be installed from the GitHub releases and is now also available from *PyPI*(Index, 2021).

The system wide installation of udocker from a central shared filesystem has become a more frequent deployment scenario. In this situation udocker is installed in a shared location often readonly. Depending on the situation the installation may include just the executables and libraries or a combination that may also include pre-defined images or even created containers that are ready to be executed. The steps and implications of using a shared installation and in particular of using readonly locations have been addressed and are also documented in the installation manual.

The software quality assurance for udocker was improved. The *Jenkins Pipeline Library* (Samuel Bernardo, 2021) was adopted to describe the quality assurance pipelines that include stages for code style checking using *pylint*, security using *bandit* and execution of the unit and integration tests. Unit test coverage is also obtained and for version 1.3.0 is 70%. The introduction of the security checks led to several code improvements including the removal of shell context from process creation and the reimplementation of the configuration files handling to prevent the injection of undesired code.

Between versions 1.1.1 and 1.1.7 the udocker source code grew from 6663 lines to 8703 lines, the *diffstat* metrics report 3847 lines inserted and 1807 lines deleted. These metrics correspond to the changes introduced in the 1.1.x versions for Python 2, they exclude the development effort related to the execution engines, the unit tests and the development of the Python 3 version now available in production as 1.3.0.

Research with udocker

Examples of usage can be found in several domains including: physics (Emanuele Bagnaschi et al., 2018) (E. Bagnaschi et al., 2019) (P. Bezyazeekov et al., 2019) (Pavel Bezyazeekov et al., 2021), life sciences (Korhonen et al., 2019) (Ziemann et al., 2019) (Merelli et al., 2019) (Kern et al., 2020) (Chillarón et al., 2017) (Korhonen et al., 2019), coastal modeling (Anabela Oliveira et al., 2019) (A. Oliveira et al., 2020), chemistry (Nalini et al., 2020) (Schaduangrat et al., 2020), structural biology (Traynor, Daniel & Froy, Terry, 2020), fusion (Lahiff, Andrew et al., 2020), earth sciences (Kerzenmacher et al., 2021) (Aguilar Gómez et al., 2017), machine learning (Grupp et al., 2019) (López García et al., 2020) (Cavallaro et al., 2019), and computer science in general (Caballer et al., 2021) (Risco & Moltó, 2021) (Sufi et al., 2020) (Aldinucci et al., 2017) (Owsiak et al., 2017).

udocker was used in the European projects EOSC-hub (EOSC-hub, 2021) where it was further improved and DEEP-hybrid-DataCloud (López García et al., 2020) where it was ported to Python 3, enhanced to support nvidia GPUs and used to execute deep learning frameworks. Since 2021 is used in the EOSC-Synergy (Kerzenmacher et al., 2021), EGI-ACE (EGI-ACE, 2021) and BIG-HPC (Paulo et al., 2020) projects. Although is a tool meant for end-users, it is also supported by several scientific and academic computer centers and research infrastructures worldwide such as:

- EGI advanced computing infrastructure in Europe (EGI.eu, 2021)
- IBERGRID Iberian distributed computing infrastructure (IBERGRID, 2021)
- INCD Portuguese Distributed Computing Infrastructure (INCD, 2021)
- CESGA Super computing Center of Galicia (CESGA, 2021)
- HPC center of the Telaviv University (Telaviv University, 2021)
- Trinity College HPC center in Dublin (Trinity College, 2021)
- University of Utah HPC center (University of Utah, 2021)
- University of Coruña Pluton Cluster (University of Coruña, 2021)

udocker was been integrated in several research oriented frameworks such as:

- SCAR - Serverless Container-aware Architectures (Pérez et al., 2018) to enable execution of containers in Amazon Lambda exploiting function as a service (FaaS);
- common-workflow-language (Chapman et al., 2016), (Korhonen et al., 2019) to enable containers in scientific workflows;
- bioconda (Grüning et al., 2018) for the conda package manager specialized in bioinformatics software;
- openmole workflow engine (Romain Reuillon, 2013) for exploration of simulation models using high throughput computing;
- and is also referenced in the SLUM Containers Guide (SchedMD, 2021).

Acknowledgments

udocker has been developed in the framework of the H2020 projects INDIGO-DataCloud (RIA 653549), EOSC-hub (RIA 777536) and DEEP-Hybrid-DataCloud (RIA 777435). The proofs of concept have been performed at INCD-Infraestrutura Nacional de Computação Distribuída (funded by FCT, P2020, Lisboa2020, COMPETE and FEDER under the project number 22153-01/SAICT/2016), FinisTerrae II machine provided by CESGA (funded by Xunta de Galicia and MINECO) and Altamira machine (funded by the University of Cantabria and MINECO).

References

Aguilar Gómez, F., Lucas, J. M. de, Fiore, S., Monna, S., & Chen, Y. (2017). INDIGO-DataCloud solutions for earth sciences. *EGU General Assembly Conference Abstracts*, 9645.

Aldinucci, M., Bagnasco, S., Lusso, S., Pasteris, P., Rabellino, S., & Vallero, S. (2017). OCCAM: A flexible, multi-purpose and extendable HPC cluster. *Journal of Physics: Conference Series*, *898*, 082039. https://doi.org/10.1088/1742-6596/898/8/082039

Bagnaschi, Emanuele, Bechtle, P., Haller, J., Kogler, R., Peiffer, T., Stefaniak, T., & Weiglein, G. (2018). Global SM and BSM Fits using Results from LHC and other Experiments. In J. Haller & M. Grefe (Eds.), *Particles, Strings and the Early Universe: The Structure of Matter and Space-Time*. https://doi.org/10.3204/PUBDB-2018-00782/B8

Bagnaschi, E., Costa, J., Sakurai, K., Borsato, M., Buchmueller, O., De, R. A., Dolan, M., Ellis, J., Flacher, H., Hahn, K., Heinemeyer, S., Lucio, M., Martinez, S. D., Olive, K., Trifa, S., & Weiglein, G. (2019). Global analysis of dark matter simplified models with leptophobic spin-one mediators using MasterCode. *EUROPEAN PHYSICAL JOURNAL C*, *79*. https://doi.org/10.1140/epjc/s10052-019-7382-3

Benedicic L., M. A., Cruz F. A. (2019). Sarus: Highly scalable docker containers for HPC systems. *ISC High Performance 2019. Lecture Notes in Computer Science*, *11887*. https://doi.org/https://doi.org/10.1007/978-3-030-34356-9_5

Bezyazeekov, P., Budnev, N., Fedorov, O., Gress, O., Grishin, O., Haungs, A., Huege, T., Kazarina, Y., Kleifges, M., Kostunin, D., Korosteleva, E., Kuzmichev, L., Lenok, V., Lubsandorzhiev, N., Malakhov, S., Marshalkina, T., Monkhoev, R., Osipova, E., Pakhorukov, A., … Zagorodnikov, A. (2019). *Advanced signal reconstruction in tunka-rex with matched filtering and deep learning*. http://arxiv.org/abs/1906.10947

Bezyazeekov, Pavel, Budnev, N., Fedorov, O., Gress, O., Grishin, O., Haungs, A., Huege, T., Kazarina, Y., Kleifges, M., Kostunin, D., Korosteleva, E., Kuzmichev, L., Lenok, V., Lubsandorzhiev, N., Malakhov, S., Marshalkina, T., Monkhoev, R., Osipova, E., Pakhorukov, A., … Zagorodnikov, A. (2021). *Reconstruction of radio signals from air-showers with autoencoder*. http://arxiv.org/abs/2101.02943

Caballer, M., Antonacci, M., Šustr, Z., Perniola, M., & Moltó, G. (2021). Deployment of elastic virtual hybrid clusters across cloud sites. *Journal of Grid Computing*, *19*(1). https://doi.org/10.1007/s10723-021-09543-5

Cavallaro, G., Kozlov, V., Götz, M., & Riedel, M. (2019). Remote sensing data analytics with the udocker container tool using multi-GPU deep learning systems. *Conference on Big Data from Space (BiDS'19)*.

CESGA. (2021). *CESGA guía de utilización de finis terrae II*. https://portalusuarios.cesga.es/layout/download/guia_uso_FT2.pdf

Chapman, B., Chilton, J., Heuer, M., Kartashov, A., Leehr, D., Ménager, H., Nedeljkovich, M., Scales, M., Soiland-Reyes, S., & Stojanovic, L. (2016). *Common workflow language, v1.0* (P. Amstutz, M. R. Crusoe, & N. Tijanić, Eds.). figshare. https://doi.org/10.6084/m9.figshare.3115156.v2

Chillarón, M., Vidal, V., Segrelles, D., Blanquer, I., & Verdú, G. (2017). Combining grid computing and docker containers for the study and parametrization of CT image reconstruction methods. *Procedia Computer Science*, *108*, 1195–1204. https://doi.org/10.1016/j.procs.2017.05.065

Developers, T. (2021). *Termux*. https://termux.com/

EGI.eu. (2021). *EGI: Advanced computing for research*. https://www.egi.eu/

EGI-ACE. (2021). *EGI-ACE: Advanced computing for EOSC*. https://www.egi.eu/projects/egi-ace/

EOSC-hub. (2021). *EOSC in practice: WeNMR*. https://eosc-hub.eu/eosc-in-practice-wenmr

Gomes, J. (2019a). *Libfakechroot flavor libc fork for udocker*. https://github.com/jorge-lip/libfakechroot-libc-udocker/tree/udocker-1

Gomes, J. (2019b). *Libfakechroot flavor musl fork for udocker*. https://github.com/jorge-lip/libfakechroot-musl-udocker/tree/udocker-1

Gomes, J. (2020). *Patchelf fork for udocker*. https://github.com/jorge-lip/patchelf-udocker/tree/udocker-1

Gomes, J. (2021). *Proot fork for udocker*. https://github.com/jorge-lip/proot-udocker/tree/udocker-2

Gomes, J., Bagnaschi, E., Campos, I., David, M., Alves, L., Martins, J., Pina, J., López-García, A., & Orviz, P. (2018). Enabling rootless linux containers in multi-user environments: The udocker tool. *Computer Physics Communications*, *232*, 84–97. https://doi.org/10.1016/j.cpc.2018.05.021

Google. (2021). *Google colaboratory*. https://colab.research.google.com/

Grupp, A., Kozlov, V., Campos, I., David, M., Gomes, J., & López García, Á. (2019). Benchmarking deep learning infrastructures by means of TensorFlow and containers. In M. Weiland, G. Juckeland, S. Alam, & H. Jagode (Eds.), *High performance computing* (pp. 478–489). Springer International Publishing. https://doi.org/10.1007/978-3-030-34356-9_36

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

IBERGRID. (2021). *IBERGRID containers technology*. https://www.ibergrid.eu/containers/

INCD. (2021). *INCD wiki: Udocker user documentation*. https://wiki.incd.pt/books/containers-tutorials/page/udocker

Index, P. P. (2021). *PyPI*. https://pypi.org/

Kern, F., Fehlmann, T., & Keller, A. (2020). On the lifetime of bioinformatics web services. *Nucleic Acids Research*, *48*(22), 12523–12533. https://doi.org/10.1093/nar/gkaa1125

Kerzenmacher, T., Kozlov, V., Sanchis, B., Cayoglu, U., Hardt, M., & Braesicke, P. (2021). *An online service for analysing ozone trends within EOSC-synergy*. Copernicus Meetings. https://doi.org/10.5194/egusphere-egu21-8418

Korhonen, P. K., Hall, R. S., Young, N. D., & Gasser, R. B. (2019). Common workflow language (CWL)-based software pipeline for de novo genome assembly from long-and short-read data. *GigaScience*, *8*(4), giz014.

Korhonen, P. K., Hall, R. S., Young, N. D., & Gasser, R. B. (2019). Common workflow language (CWL)-based software pipeline for de novo genome assembly from long-and short-read data. *GigaScience*, *8*(4), giz014.

Kurtzer, V. A. B., Gregory M. AND Sochat. (2017). Singularity: Scientific containers for mobility of compute. *PLOS ONE*, *12*(5), 1–20. https://doi.org/10.1371/journal.pone.0177459

Lahiff, Andrew, de Witt, Shaun, Caballer, Miguel, La Rocca, Giuseppe, Pamela, Stanislas, & Coster, David. (2020). Running HTC and HPC applications opportunistically across private, academic and public clouds. *EPJ Web Conf.*, *245*, 07032. https://doi.org/10.1051/epjconf/202024507032

López García, Á., De Lucas, J. M., Antonacci, M., Zu Castell, W., David, M., Hardt, M., Lloret Iglesias, L., Moltó, G., Plociennik, M., Tran, V., Alic, A. S., Caballer, M., Plasencia, I. C., Costantini, A., Dlugolinsky, S., Duma, D. C., Donvito, G., Gomes, J., Heredia Cacha, I., … Wolniewicz, P. (2020). A cloud-based framework for machine learning workloads and applications. *IEEE Access*, *8*, 18681–18692. https://doi.org/10.1109/ACCESS.2020.2964386

Merelli, I., Fornari, F., Tordini, F., D'Agostino, D., Aldinucci, M., & Cesini, D. (2019). Exploiting docker containers over grid computing for a comprehensive study of chromatin conformation in different cell types. *Journal of Parallel and Distributed Computing*, *134*, 116–127. https://doi.org/10.1016/j.jpdc.2019.08.002

Nalini, S., Lampa, S., Saw, S., Gleeson, M. P., Ola, S., & Chanin, N. (2020). Towards reproducible computational drug discovery. *Journal of Cheminformatics*, *12*(1). https://doi.org/10.1186/s13321-020-0408-x

Oliveira, A., Fortunato, A. B., Rogeiro, J., Teixeira, J., Azevedo, A., Lavaud, L., Bertin, X., Gomes, J., David, M., Pina, J., Rodrigues, M., & Lopes, P. (2020). OPENCoastS: An open-access service for the automatic generation of coastal forecast systems. *Environmental Modelling & Software*, *124*, 104585. https://doi.org/10.1016/j.envsoft.2019.104585

Oliveira, Anabela, Rodrigues, M., Rogeiro, J., Fortunato, A. B., Teixeira, J., Azevedo, A., & Lopes, P. (2019). OPENCoastS: An open-access app for sharing coastal prediction information for management and recreation. In J. M. F. Rodrigues, P. J. S. Cardoso, J. Monteiro, R. Lam, V. V. Krzhizhanovskaya, M. H. Lees, J. J. Dongarra, & P. M. A. Sloot (Eds.), *Computational science – ICCS 2019* (pp. 794–807). Springer International Publishing. https://doi.org/10.1016/j.future.2013.05.003

opencontainers.org. (2021). *Runc*. https://github.com/opencontainers/runc

Owsiak, M., Plociennik, M., Palak, B., Zok, T., Reux, C., Di Gallo, L., Kalupin, D., Johnson, T., & Schneider, M. (2017). Running simultaneous kepler sessions for the parallelization of parametric scans and optimization studies applied to complex workflows. *Journal of Computational Science*, *20*, 103–111. https://doi.org/10.1016/j.procs.2016.05.362

Paulo, J., Vilaça, R., Ribeiro, R., & Todd, R. (2020). *A management framework for consolidated big data and HPC*.

Pérez, A., Moltó, G., Caballer, M., & Calatrava, A. (2018). Serverless computing for container-based architectures. *Future Generation Computer Systems*, *83*, 50–59. https://doi.org/10.1016/j.future.2018.01.022

Risco, S., & Moltó, G. (2021). GPU-enabled serverless workflows for efficient multimedia processing. *Applied Sciences*, *11*(4). https://doi.org/10.3390/app11041438

Romain Reuillon, S. R.-C., Mathieu Leclaire. (2013). OpenMOLE, a workflow engine specifically tailored for the distributed exploration of simulation models. *Future Generation Computer Systems*, *29*(8), 1981–1990. https://doi.org/10.1016/j.future.2013.05.003

Salomoni, D., Campos, I., Gaido, L., Lucas, J. M. de, Solagna, P., Gomes, J., Matyska, L., Fuhrman, P., Hardt, M., Donvito, G., Dutka, L., Plociennik, M., Barbera, R., Blanquer, I., Ceccanti, A., Cetinic, E., David, M., Duma, C., López-García, A., … Zok, T. (2018). INDIGO-DataCloud: A platform to facilitate seamless access to e-infrastructures. *Journal of Grid Computing*, *16*(3), 381–408. https://doi.org/10.1007/s10723-018-9453-3

Samuel Bernardo, P. O. (2021). *A library to implement SQA checks in jenkins environments*. https://github.com/indigo-dc/jenkins-pipeline-library

Schaduangrat, N., Lampa, S., Simeon, S., Gleeson, M. P., Spjuth, O., & Nantasenamat, C. (2020). Towards reproducible computational drug discovery. *J. Cheminformatics*, *12*(1), 9. https://doi.org/10.1186/s13321-020-0408-x

SchedMD. (2021). *Containers guide*. https://slurm.schedmd.com/containers.html

Scrivano, G. (2021). *Crun*. https://github.com/containers/crun

Sufi, S., Martinez Ortiz, C., Hof, C., Aerts, P., Klinkenberg, A., Lambrecht, A.-L., Sierman, B., Willigen, B. van, Olivier, B., Willing, C., & others. (2020). *Report on the workshop on sustainable software sustainability 2019 (WOSSS19)*.

Telaviv University. (2021). *Using udocker to run containers*. https://www.cs.tau.ac.il/system/udocker

Traynor, Daniel, & Froy, Terry. (2020). Provision and use of GPU resources for distributed workloads via the grid. *EPJ Web Conf.*, *245*, 03002. https://doi.org/10.1051/epjconf/202024503002

Trinity College. (2021). *Research IT support pages*. https://www.tchpc.tcd.ie/node/1312

University of Coruña. (2021). *Cluster pluton user guide*. http://pluton.des.udc.es/guide/user-guide.pdf

University of Utah. (2021). *CHPC - research computing support for the university*. https://www.chpc.utah.edu/documentation/software/udocker.php

Ziemann, M., Kaspi, A., & El-Osta, A. (2019). Digital expression explorer 2: a repository of uniformly processed RNA sequencing data. *GigaScience*, *8*(4). https://doi.org/10.1093/gigascience/giz022