The Journal of Open Source Software

containerit: Generating Dockerfiles for reproducible research with R

Daniel Nüst[1] **and Matthias Hinz**[2]

1 Institute for Geoinformatics, University of Münster, Germany **2** Professorship for Geoinformatics and Geodesy, Faculty of Agricultural and Environmental Sciences, University of Rostock, Germany

DOI: 10.21105/joss.01603

Software

- Review
- Repository
- Archive

Submitted: 25 July 2019
Published: 21 August 2019

Statement of Need

Linux containers have become a promising tool to increase transparency, portability, and reproducibility of research in several domains and use cases: data science (Boettiger, 2015), software engineering research (Cito & Gall, 2016), multi-step bioinformatics pipelines (Kim, Ali, Lijeron, Afgan, & Krampis, 2017), standardised environments for exchangeable software (Belmann et al., 2015), computational archaeology (Marwick, 2017), packaging algorithms (Hosny, Vera-Licona, Laubenbacher, & Favre, 2016), or geographic object-based image analysis (Knoth & Nüst, 2017). Running an analysis in a container increases reliability of a workflow, as it can execute packaged code independently of the author's computer and its available configurations and dependencies. However, capturing a computational environment in containers can be complex, making container use difficult for domain scientists with limited programming experience. `containerit` opens up the advantages of containerisation to a much larger user base by assisting researchers, who are unfamiliar with Linux, command lines or containerisation, in packaging workflows based on R (R Core Team, 2018) in container images by using only user-friendly R commands.

Recently containerisation took off as a technology for packaging applications and their dependencies for fast, scalable, and secure sandboxed deployments in cloud-based infrastructures (cf. Osnat, 2018). The most widely used containerisation software is Docker with the following core building blocks (cf. Docker: Get Started): The *image* is built from the instructions in a recipe called `Dockerfile`. The image is executed as a *container* using a *container runtime*. An image can be moved between systems as a file (image tarball) or based on an *image registry*. A `Dockerfile` may use the image created by another `Dockerfile` as the starting point, a so-called *base image*. While containers can be manually altered, the common practice is to conduct all configurations with the scripts and instructions originating in the `Dockerfile`.

An important advantage of containers over virtual machines is that their duality between recipe and image provides and additional layer of transparency and safeguarding. The `Dockerfile` and image can be published alongside a scientific paper to support peer review and, to some extent, preserve the original results (Nüst et al., 2017). Even if an image cannot be executed or a `Dockerfile` can no longer be built, the instructions in the `Dockerfile` are human-readable, and files in the image can be extracted to recreate an environment that closely resembles the original. Further useful features are (a) portability, thanks to a single runtime dependency, which allows readers to explore an author's virtual laboratory, including complex dependencies or custom-made code, either on their machines or in cloud-based infrastructures (e.g., by using Binder, see Project Jupyter et al., 2018), and (b) transparency, because an image's filesystem can be easily inspected. This way, containers can enable verification of reproducibility and auditing without requiring reviewers to manually download, install, and re-run analyses (Beaulieu-Jones & Greene, 2017).

Container preservation is an active field of research (Emsley & De Roure, 2018; Rechert et al., 2017). It is reasonable to assume that key stakeholders interested in workflow preservation, such as universities or scientific publishers, should be able to operate container runtimes on a time scale comparable to data storage requirements by funding agencies, e.g., 10 years in case of the German DFG or British EPSRC. To enable and leverage the stakeholders' infrastructure, container creation must become easier and more widespread.

Summary

The package `containerit` automates the generation of `Dockerfiles` for workflows in R, based on images by the Rocker project (Boettiger & Eddelbuettel, 2017). The core feature of `containerit` is that it transforms the local session information into a set of instructions which can be serialised as a `Dockerfile`, as shown in the code snippet below:

```
> suppressPackageStartupMessages(library("containerit"))
> my_dockerfile <- containerit::dockerfile(from = utils::sessionInfo())
> print(my_dockerfile)
FROM rocker/r-ver:3.5.2
LABEL maintainer="daniel"
RUN export DEBIAN_FRONTEND=noninteractive; apt-get -y update \
  && apt-get install -y git-core \
    libcurl4-openssl-dev \
    libssl-dev \
    pandoc \
    pandoc-citeproc
RUN ["install2.r", "curl", "digest", "evaluate", "formatR", \
  "futile.logger", "futile.options", "htmltools", "jsonlite", \
  "knitr", "lambda.r", "magrittr", "Rcpp", "rjson", \
  "rmarkdown", "rsconnect", "semver", "stevedore", "stringi", \
  "stringr", "xfun", "yaml"]
WORKDIR /payload/
CMD ["R"]
```

The created `Dockerfile` has installation instructions for the loaded packages and their system dependencies. It uses the `r-ver` stack of Rocker images, matching the R version to the environment encountered locally by `containerit`. These images use MRAN snapshots to control installed R package versions in a reproducible way. The system dependencies required by these packages are identified using the `sysreqs` package (Csardi, 2019) and the corresponding database and API.

`dockerfile(..)` is the package's main user function and accepts session information objects, session information saved in a file, a set of R commands, an R script file, a `DESCRIPTION` file, or an R Markdown document (Allaire et al., 2018). Static program analysis using the package `automagic` (Brokamp, 2017) is used to increase the chances that the capturing environment has all required packages available, such as when creating Dockerfiles for R Markdown documents as a service (Nüst, 2018). To capture the workflow environment, `containerit` executes the whole workflow in a new R session using the package `callr` (Csárdi & Chang, 2018), because static program analysis can be broken by using helper functions, such as `xfun::pkg_attach()` (Xie, 2018), by unintended side effects, or by seemingly clever or user-friendly yet customised ways of loeading packages (cf. first lines in R script file `tgis_a_1579333_sm7524.r` in https://doi.org/10.6084/m9.figshare.7757069.v1). Further parameters for the function comprise, for example, image metadata, base image, versioned installations, and filtering of R packages already installed in the base image.

The package containerit's main contribution is that it allows for automated capturing of runtime environments as Dockerfiles based on literate programming workflows (Gentleman & Lang, 2007) to support reproducible research. Together with stevedore (FitzJohn, 2019), containerit enables a completely R-based creation and manipulation of Docker containers. Using containerit only minimally affects researchers' workflows because it can be applied after completing a workflow, while at the same time the captured snapshots can enhance the scholarly publication process (in particular review, interaction, and preservation) and may form a basis for more reusable and transparent publications. In the future, containerit may support alternative container software such as Singularity (Kurtzer, Sochat, & Bauer, 2017), enable parametrisation of container executions and pipelines as demonstrated by Kliko (Molenaar, Makhathini, Girard, & Smirnov, 2018), or support proper accreditation of software (Jones et al., 2017; D. S. Katz & Chue Hong, 2018).

Related Work

renv is an R package for managing reproducible environments for R providing isolation, portability, and pinned versions of R packages, but it does not handle system dependencies. The Experiment Factory similarly focuses on ease of use for creating Dockerfiles for behavioural experiments, yet it uses a CLI-based interaction and generates extra shell scripts to be included in the images. ReproZip (Chirigati, Rampin, Shasha, & Freire, 2016) packages files identified by tracing in a self-contained bundle, which can be unpacked to a Docker container/Dockerfile. In the R domain, the package dockerfiler (Fay, 2018) provides an object-oriented API for manual Dockerfile creation, and liftr (Xiao, 2018) creates a Dockerfile based on fields added to the metadata header of an R Markdown document. automagic (Brokamp, 2017), Whales, dockter, and repo2docker use static program analysis to create environment descriptions from common project configuration files for multiple programming languages. Namely, automagic analyses R code and can store dependencies in a bespoke YAML format. Whales and dockter provide different formats, including Dockerfile. Finally, repo2docker primarily creates containers for interactive notebooks to run as a Binder (Project Jupyter et al., 2018) but does not actively expose a Dockerfile. None of them apply the strict code execution approach as containerit does.

Acknowledgements

This work is supported by the project Opening Reproducible Research (Offene Reproduzierbare Forschung) funded by the German Research Foundation (DFG) under project numbers PE 1632/10-1 and 1632/17-1.

References

Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., et al. (2018). *rmarkdown: Dynamic documents for R*. Retrieved from https://rmarkdown.rstudio.com

Beaulieu-Jones, B. K., & Greene, C. S. (2017). Reproducibility of computational workflows is automated using continuous analysis. *Nature Biotechnology*, *advance online publication*. doi:10.1038/nbt.3780

Belmann, P., Dröge, J., Bremges, A., McHardy, A. C., Sczyrba, A., & Barton, M. D. (2015). Bioboxes: Standardised containers for interchangeable bioinformatics software. *GigaScience*, *4*(1), 47. doi:10.1186/s13742-015-0087-0

Boettiger, C. (2015). An introduction to Docker for reproducible research, with examples from the R environment. *ACM SIGOPS Operating Systems Review*, *49*(1), 71–79. doi:10.1145/2723872.2723882

Boettiger, C., & Eddelbuettel, D. (2017). An Introduction to Rocker: Docker Containers for R. *The R Journal*, *9*(2), 527–536. doi:10.32614/RJ-2017-065

Brokamp, C. (2017). *Automagic: Automagically document and install packages necessary to run R code*. Retrieved from https://CRAN.R-project.org/package=automagic

Chirigati, F., Rampin, R., Shasha, D., & Freire, J. (2016). ReproZip: Computational reproducibility with ease. In *Proceedings of the 2016 international conference on management of data*, SIGMOD '16 (pp. 2085–2088). San Francisco, California, USA: ACM. doi:10.1145/2882903.2899401

Cito, J., & Gall, H. C. (2016). Using Docker Containers to Improve Reproducibility in Software Engineering Research. In *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16 (pp. 906–907). ACM. doi:10.1145/2889160.2891057

Csardi, G. (2019). *Sysreqs: Install systemrequirements of packages*. Retrieved from https://github.com/r-hub/sysreqs

Csárdi, G., & Chang, W. (2018). *Callr: Call R from R*. Retrieved from https://CRAN.R-project.org/package=callr

Emsley, I., & De Roure, D. (2018). A Framework for the Preservation of a Docker Container International Journal of Digital Curation. *International Journal of Digital Curation*, *12*(2). doi:10.2218/ijdc.v12i2.509

Fay, C. (2018). *Dockerfiler: Easy Dockerfile creation from R*. Retrieved from https://CRAN.R-project.org/package=dockerfiler

FitzJohn, R. (2019). *Stevedore: Docker client*. Retrieved from https://CRAN.R-project.org/package=stevedore

Gentleman, R., & Lang, D. T. (2007). Statistical Analyses and Reproducible Research. *Journal of Computational and Graphical Statistics*, *16*(1), 1–23. doi:10.1198/106186007X178663

Hosny, A., Vera-Licona, P., Laubenbacher, R., & Favre, T. (2016). AlgoRun: A Docker-based packaging system for platform-agnostic implemented algorithms. *Bioinformatics*, *32*(15), 2396–2398. doi:10.1093/bioinformatics/btw120

Jones, M. B., Boettiger, C., Mayes, A. C., Arfon Smith, Slaughter, P., Niemeyer, K., Gil, Y., et al. (2017). CodeMeta: An exchange schema for software metadata. KNB Data Repository. doi:10.5063/schema/codemeta-2.0

Katz, D. S., & Chue Hong, N. P. (2018). Software Citation in Theory and Practice. In J. H. Davenport, M. Kauers, G. Labahn, & J. Urban (Eds.), *Mathematical Software – ICMS 2018*, Lecture Notes in Computer Science (pp. 289–296). Springer International Publishing. doi:10.1007/978-3-319-96418-8_34

Kim, B., Ali, T. A., Lijeron, C., Afgan, E., & Krampis, K. (2017). Bio-Docklets: Virtualization Containers for Single-Step Execution of NGS Pipelines. *bioRxiv*, 116962. doi:10.1101/116962

Knoth, C., & Nüst, D. (2017). Reproducibility and Practical Adoption of GEOBIA with Open-Source Software in Docker Containers. *Remote Sensing*, *9*(3), 290. doi:10.3390/rs9030290

Kurtzer, G. M., Sochat, V., & Bauer, M. W. (2017). Singularity: Scientific containers for mobility of compute. *PLOS ONE*, *12*(5), e0177459. doi:10.1371/journal.pone.0177459

Marwick, B. (2017). Computational Reproducibility in Archaeological Research: Basic Principles and a Case Study of Their Implementation. *Journal of Archaeological Method and Theory*, *24*(2), 424–450. doi:10.1007/s10816-015-9272-9

Molenaar, G., Makhathini, S., Girard, J. N., & Smirnov, O. (2018). Kliko—The scientific compute container format. *Astronomy and Computing*, *25*, 1–9. doi:10.1016/j.ascom.2018.08.003

Nüst, D. (2018, December). Reproducibility Service for Executable Research Compendia: Technical Specifications and Reference Implementation. Zenodo. doi:10.5281/zenodo.2203844

Nüst, D., Konkol, M., Pebesma, E., Kray, C., Schutzeichel, M., Przibytzin, H., & Lorenz, J. (2017). Opening the Publication Process with Executable Research Compendia. *D-Lib Magazine*, *23*(1/2). doi:10.1045/january2017-nuest

Osnat, R. (2018, March). A Brief History of Containers: From the 1970s to 2017. Retrieved from https://blog.aquasec.com/a-brief-history-of-containers-from-1970s-chroot-to-docker-2016

Project Jupyter et al. (2018). Binder 2.0 - reproducible, interactive, sharable environments for science at scale. In *Proceedings of the 17th python in science conference*. doi:10.25080/Majora-4af1f417-011

R Core Team. (2018). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from https://www.R-project.org/

Rechert, K., Liebetraut, T., Kombrink, S., Wehrle, D., Mocken, S., & Rohland, M. (2017). Preserving Containers. In J. Kratzke & V. Heuveline (Eds.), *Forschungsdaten managen* (pp. 143–151). doi:10.11588/heibooks.285.377

Xiao, N. (2018). *Liftr: Containerize R Markdown documents for continuous reproducibility*. Retrieved from https://CRAN.R-project.org/package=liftr

Xie, Y. (2018). *Xfun: Miscellaneous functions by 'Yihui Xie'*. Retrieved from https://CRAN.R-project.org/package=xfun