IJDC | *Conference Pre-print*

Klaus Rechert
University of Freiburg

Oleg Stobbe
University of Freiburg

Oleg Zharkov
University of Freiburg

Rafael Gieschke
University of Freiburg

Dennis Wehrle
University of Freiburg

Abstract

In contrast to books or published articles, pure digital output of research projects is more fragile and, thus, more difficult to preserve and more difficult to be made available and to be reused by a wider research community. Not only does a fast-growing format diversity in research data sets require additional software preservation but also today's computer assisted research disciplines increasingly devote significant resources into creating new digital resources and software-based methods.

In order to adapt FAIR data principles, especially to ensure re-usability of a wide variety of research outputs, novel ways for preservation of software and additional digital resources are required as well as their integration into existing research data management strategies.

This article addresses preservation challenges and preservation options of containers and virtual machines to encapsulate software-based research methods as portable and preservable software-based research resources, provides a preservation plan as well as an implementation.

Submitted 18 December 2019 ~ *Accepted* 19 February 2020

Correspondence should be addressed to Klaus Rechert, Hermann-Herder Str. 10 79102 Freiburg, Germany. Email: Klaus.rechert@rz.uni-freiburg.de

This paper was presented at International Digital Curation Conference IDCC20, Dublin, 17-19 February 2020

The *International Journal of Digital Curation* is an international journal committed to scholarly excellence and dedicated to the advancement of digital curation across a wide range of sectors. The IJDC is published by the University of Edinburgh on behalf of the Digital Curation Centre. ISSN: 1746-8256. URL: http://www.ijdc.net/

Introduction

In contrast to books or published articles, pure digital output of research projects is more fragile and, thus, more difficult to preserve as well as more difficult to be made available and to be reused by a wider research community. Additionally, specialization of research disciplines leads to a fast-growing diversity of formats used in research data sets. For instance, a study of data sets in public repositories showed a wide variation of file formats used in general, and in particular multiple different data formats were used within individual data sets (cf. Wehrle and Rechert, 2019). Appropriate software – probably together with operational knowledge – will be required to ensure meaningful long-term access to these data sets. Furthermore, today's computer assisted research does not only rely on existing software and digital resources, but increasingly devotes significant resources into creating new digital resources and tailored software-based methods, i.e., to process data or to create novel (software-based) models. In order to adapt FAIR data principles (Wilkinson et al., 2016) to software-based research methods (Lamprecht et al., 2019) and to ensure re-usability of a wide variety of digital research outputs, novel ways for preservation are to be developed and to be integrated in research data management strategies.

Operational knowledge, in particular of highly specialized software, vanishes and the technological environment changes quickly, such that a retrospective reconstruction of complex scientific workflows will become increasingly difficult (Macleod et al., 2014; Baker, 2014). Therefore, software-setups should be captured in a (re-)usable manner (Grüning et al., 2018). Since there exist already concepts and tools to capture and encapsulate software-based experiments and workflows (e.g., Munafò et al., 2017; Chard et al., 2019), preservation workflows should build on-top of these existing tools and practices. To ensure long-term access, however, archived experiments need to be maintained in an efficient manner, since not only the original technical environment is changing, but also technical progress will change curation and access methodology over time. To tackle these challenges, a suitable generic representation of captured (self-contained) software-based experiments and workflows is required. Within the CiTAR project[1] we have explored options to publish and preserve software-based research methodology as container or virtual machines while ensuring scalable long-term access and usability.

Preservation Challenges

Previous efforts of preserving research objects focused mostly on static, singular artifacts. Recently, concepts and practice of software citation have been developed[2] as well as guidelines and infrastructure for the management of software dependencies.[3][4] For productive re-use scenarios, software needs to be made available and accessible in a functional way, i.e., to be used with research data sets or within a complex scientific workflow. With technical progress and especially the advance of virtualization, container, Cloud and related technologies, research environments became interconnected and interactive, most importantly, however, research data and software became more interdependent (e.g. Pimentel et al., 2019), such that access and re-use will only be possible if all components are available, configured and executable (Ivie and Thain, 2018).

Popular manifestations of this trend are virtual machines (VM) and containers (e.g., Docker or Singularity). These technologies have been adopted quickly by researchers (e.g., Boettiger,

[1] CiTAR - Citing and archiving research methods, a three-year Baden-Württemberg state project: http://citar.eaas.uni-freiburg.de/

[2] cf. Software Citation Principles: https://www.force11.org/software-citation-principles

[3] Checklist for a Software Management Plan: https://zenodo.org/record/2159713

[4] Software Deposit Guidance: https://softwaresaved.github.io/software-deposit-guidance/

2005) because they enable researchers to encapsulate complex software-based research environments – e.g., multiple software components, including application-specific settings – into a single portable entity and ease reproduction of scientific results (Meng et al., 2015). Furthermore, these technologies allow researchers to develop, prepare and test a complex software setup locally and deploy it with little additional effort in the Cloud or high-performance computing (HPC) facilities. This way, a pre-configured software-based research environment can be shared and (re-)used independently of its original creation context (Howe, 2012). Commercial[5] and community driven solutions[6] [7] built on-top of aforementioned technologies and provide tools as well as public portals to publish, share and reproduce experiments. However, these "public – in production" solutions rely – sometimes implicitly – on a complex, contemporary technical landscape (e.g., a specific Docker version on a specific Linux kernel version on the Intel/x86 architecture), publicly available software repositories (registries), such as Docker Hub, GitHub and similar, to assemble and re-run a published experiment. However, due to the implicit encapsulation and simplicity of publishing and sharing, VMs and containers therefore seems to be promising preservation targets for complex software setups. Still, preserving VMs and containers pose new technical and new conceptual challenges.

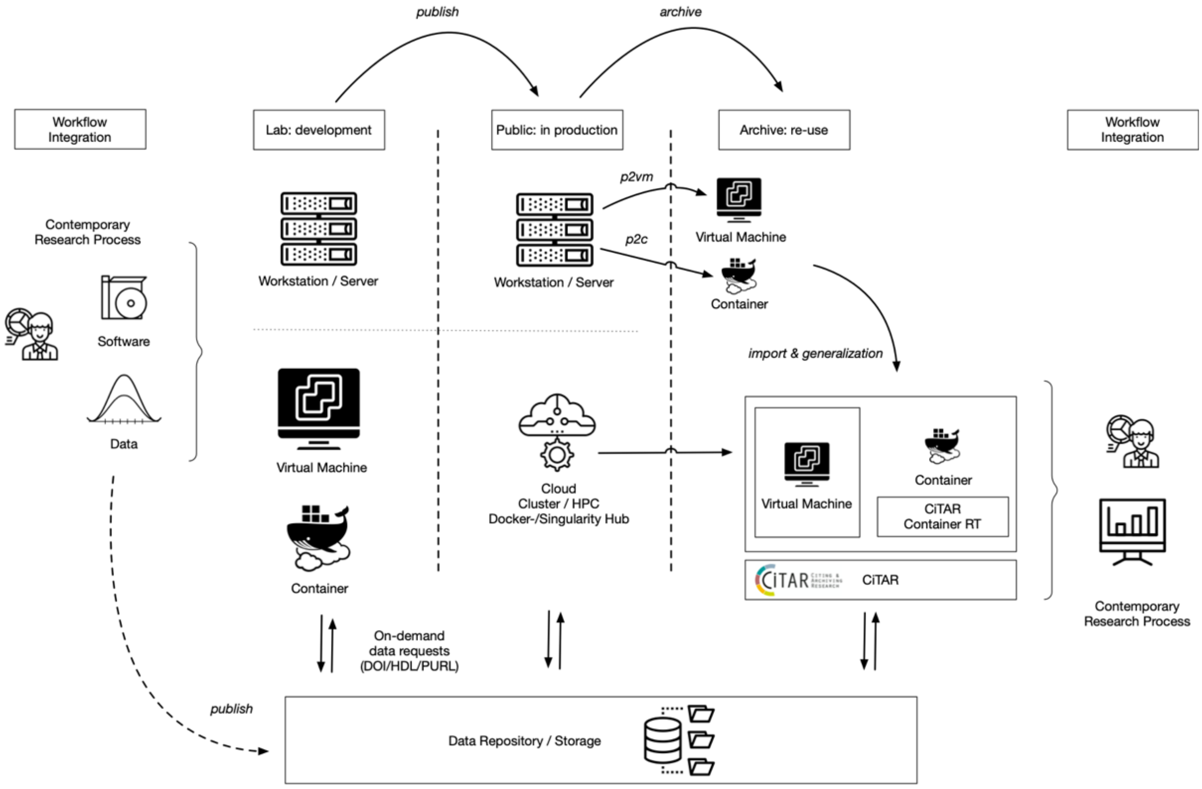


Figure 1. Life-cycle of software-based research methods.

The CiTAR project aims to provide infrastructure and workflows for the "archive and reuse" stage (cf. Figure 1) of the research data management life-cycle (cf. Ball, 2012). Fig. 1 depicts a simplified life-cycle of software-based research methods with three main stages. Our focus is on improving the archive workflow of "in use" software setups and maintaining future integration (re-use) possibilities. The CiTAR framework focuses on workflows and builds on-top

[5] For instance, Code Ocean: https://codeocean.com/

[6] The Galaxy Project: https://usegalaxy.org/

[7] Reprozip: https://www.reprozip.org/

of existing technologies, such as (public) compute Clouds, (public) research data repositories and Emulation-as-a-Service[8] (EaaS) to substitute hardware dependencies.

Defining a Preservation Target

The number of already created artifacts (VMs and containers) as well as their technical diversity is overwhelming. Even though, the resulting artifacts are built on common technical foundations – VMs are based on hardware virtualization technology (prominently Intel/x86), containers build on virtualization of operating system interfaces technology – different vendors and implementations created a variety of "flavors", all similar but slightly different. In order to ensure future reuse of these objects in an evolving technological landscape, appropriate technical runtime environments are necessary, e.g., emulation providing an abstract hardware equivalent for execution, possibly in combination with additional software.

While for any technical flavor an individual solution eventually can be found (e.g., manual adaption, preservation of original software, etc.), the scale of this task makes such an approach not feasible for a generic data management strategy and following, a preservation plan. To develop and implement an efficient and scalable preservation plan for VMs and containers, in both cases a common structure, format and/or technical description is necessary to reduce the technical variety with a minimal and manageable set of technical dependencies to be monitored and eventually substituted.

Integration and Securing Access

A further preservation challenge is to provide access to preserved VM and containers. Traditional emulation workflows usually focus on providing interactive access to the user. This approach has been proven useful for multimedia objects like CD-ROMs or games (Espenschied et al., 2013), however, in the case of software-based research methods, interactive access might not be as useful, since these need to be integrated into a researcher's environment or workflow, e.g., via a network connection to automate its operation, processing input date and produced output.

Making old software setups available via network can become a security risk. Maintaining security problems within of preserved software-setups is difficult, if not impossible. After a research project ended, developers as well financial support is usually unavailable, especially if the created setups, e.g., specialized databases, tools or similar have small research communities. Furthermore, any modification, including security fixes, to the preserved software might change its behavior and make the reproduction of the original results impossible.

Artifact Generalization – Defining Common Object Structures

With the goal to implement a framework for preserving containers and VMs, in a first step a preservation plan for VMs and containers has to be developed and implemented.

Virtual Machines

Old software encapsulated as a VM expects technical interfaces to be present to run, e.g., an appropriate instruction set architecture (ISA) as well as the presence of several hardware components. Emulation allows to bridge the gap between old and contemporary hardware by utilizing software to "emulate" an old hardware environment on modern hardware and thus is

[8] Emulation-as-a-Service: https://eaas.uni-freiburg.de

able to ensure functionality of VMs in the long run. From today's perspective, we can assume that there will be a future demand for emulators, e.g., to emulate today's popular Intel/x86-based hardware platform, therefore we assume the availability of appropriate emulation software for future computer systems. However, it is impossible to predict their actual technical characteristics: while all emulators of a given technical architecture support a common ISA (e.g. x86), they may differ significantly in their technical configuration, resulting in VMs that perform well on one emulator and are non-operational on others. Hence, it is necessary to prepare VMs conceptually and technically for the event of replacing obsolete emulators with a future generation of emulators.

To simulate these changes in emulated hardware, we conducted an experiment migrating VMs from one virtual hardware to another, to determine an effective procedure for this task. We chose today's popular emulation and virtualization software VMware, VirtualBox and QEMU and installed several Windows and Linux operating system versions on all three platforms using default choices wherever possible, to create typical disk images. Then, we tried to run all of these images on the alternative two platforms. The first observation was that differing hardware setups prevented even a basic system start-up (e.g. to safe boot /repair) failed for every possible combination (Windows VMs). In a further step, we have searched for relevant information in online resources for hints or hacks to get a rudimentary boot-setup, from which self-repair mechanisms of the operating system could be accessed. We have succeeded in only two out of six attempts (Windows XP). For a more scalable and predictable solution, we evaluated potential system and hardware settings to find the most generic configuration, compatible with any emulated or virtual hardware setup. While such a configuration won't yield the most preformat setup (e.g., due to the selection of generic hardware drivers), this approach worked with every virtualization / emulation setup and operating system. Furthermore, we could implement automated procedures to be applied during VMs ingest for hardware driver adaption.

These experimental results highlight the difficulties of managing software environments manifested as VM disk images from different or unknown hardware configurations, but more importantly they help managing future changes in emulator setups. Additionally, the knowledge created about required technical adaptations, and preserved as meta-data and tools, will reduce the complexity and cost of future adaptations, especially if all archived disk images are generalized to a small number of technical configurations.

Containers

As a consequence of virtualizing operating system interfaces, containers are isolated from the host system and thus need to be self-contained, i.e., all software dependencies (libraries, applications, etc.) have to be included within the container. The main content of a container is just a self-contained filesystem with installed and configured software components (except the operating system kernel, which is provided by the host system) (c.f. Figure 2). The remaining external or unresolved software dependencies are the container runtime and the underlying operating system kernel, i.e., the operating system kernel's application binary interface (ABI).

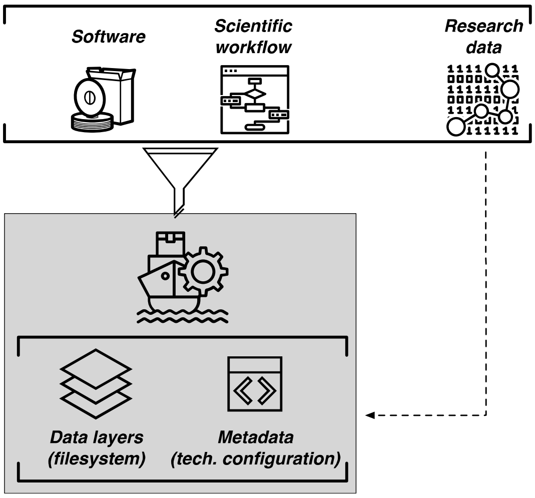


Figure 2. Container structure.

A container's technical runtime is composed of two components: a hardware component (the computer) and a software component. In general, the software component represents typically a basic Linux installation with an installed and configured container runtime (cf. Fig. 3 left). As a first preservation step, the hardware component is replaced by an emulated hardware equivalent which allows containers (together with the vendor specific software dependencies) to be archived and such that the containers can unchanged over time (cf. Fig 3 centre).

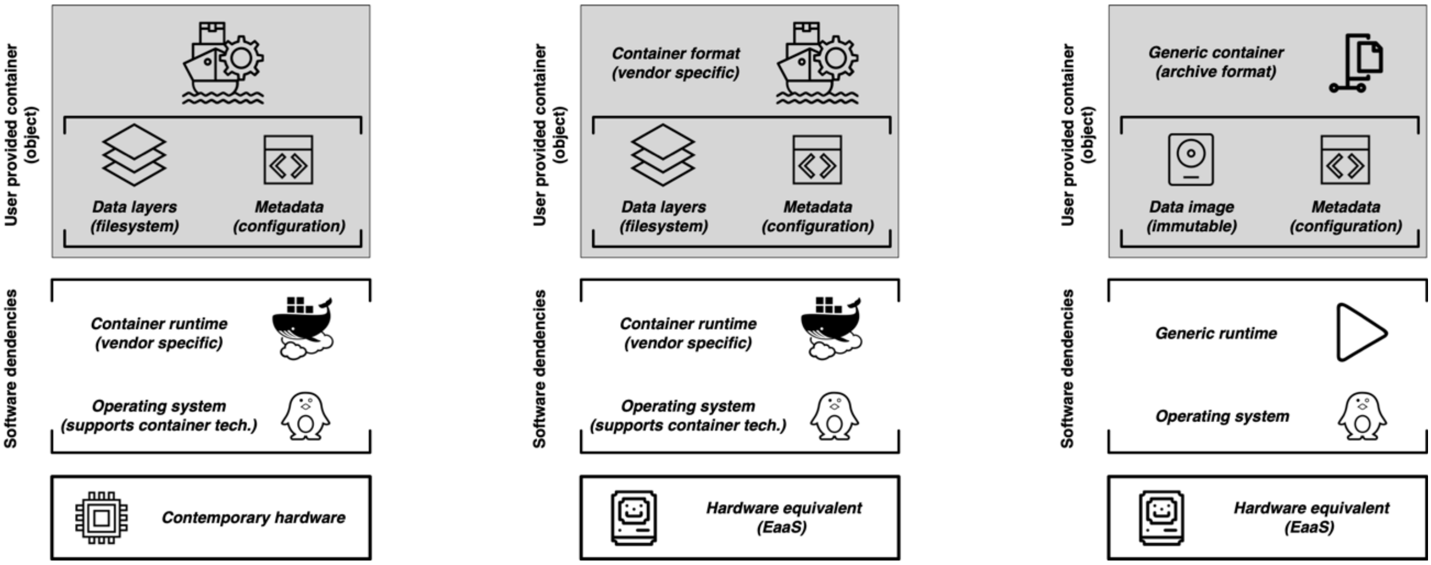
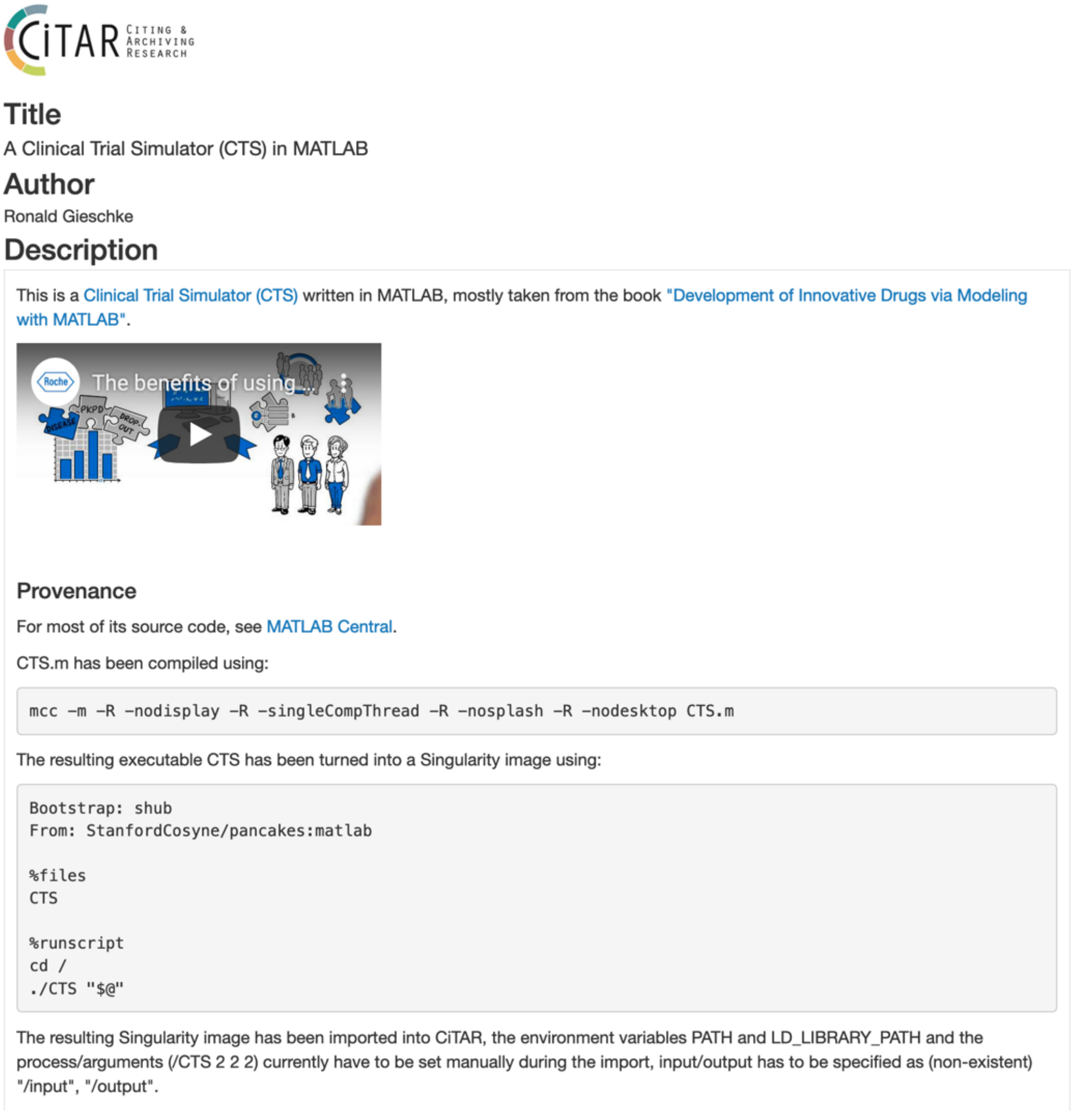

User provided container (object)
Data layers (filesystem)
Metadata (configuration)
Software dendencies
Container runtime (vendor specific)
Operating system (supports container tech.)
Contemporary hardware
User provided container (object)
Container format (vendor specific)
Data layers (filesystem)
Metadata (configuration)
Software dendencies
Container runtime (vendor specific)
Operating system (supports container tech.)
Hardware equivalent (EaaS)
User provided container (object)
Generic container (archive format)
Data image (immutable)
Metadata (configuration)
Software dendencies
Generic runtime
Operating system
Hardware equivalent (EaaS)

Figure 3. Left: full contemporary software and hardware setup. Centre: hardware is replaced by a hardware equivalent (emulator), software stack remains unchanged. Right: Vendor specific container format is generalized to a generic container representation.

Even though all containers are built on top of the same technical foundations, today's popular container implementations such as Docker, Singularity, or Shifter use different internal representations and configuration formats. Thus, these representations require a (vendor) specific technical runtime with the appropriate software version. With respect to long-term maintenance of such user-provided objects, these different runtimes need to be maintained over time. But also, the access framework requires adaptations specific to the runtime used. These adaptations concern in particular reusing containers with user-provided data, providing interactive access to the contained software process, for instance to tweak variables, as well as retrieving a process's output for inspection or further processing. Hence, a small number of software runtimes being able to run a large number of preserved containers is required, as a common, generic container representation and configurations reduces the complexity and simplifies developing access services. By leveraging the fact that all containers consist of a filesystem representation and configuration meta-data, we have implemented a common archive representation of containers together with a workflow to ingest and convert container "flavors" into this common format. Through this workflow the vendor-specific filesystem representation is migrated to an immutable image format, which is then managed and archived like any other disk image. Configuration meta-data is converted to the Open Container Initiative (OCI) standard runtime *runC*. The generic runtime, a quite minimal Linux system (about 60 MB) is built as a VM. Building a new container runtime VM is only necessary if the ABI of the underlying operating system kernel has changed, which for the Linux kernel is very rarely the case. Figure 3 (right) shows the final CiTAR container technology stack. Figure 4 shows an example of a CiTAR landing-page of a preserved container, which contains usage information and the option to execute the experiment and retrieve the result.



Title

A Clinical Trial Simulator (CTS) in MATLAB

Author

Ronald Gieschke

Description

This is a Clinical Trial Simulator (CTS) written in MATLAB, mostly taken from the book "Development of Innovative Drugs via Modeling with MATLAB".

Provenance

For most of its source code, see MATLAB Central.

CTS.m has been compiled using:

```
mcc -m -R -nodisplay -R -singleCompThread -R -nosplash -R -nodesktop CTS.m
```

The resulting executable CTS has been turned into a Singularity image using:

```
Bootstrap: shub
From: StanfordCosyne/pancakes:matlab

%files
CTS

%runscript
cd /
./CTS "$@"
```

The resulting Singularity image has been imported into CiTAR, the environment variables PATH and LD_LIBRARY_PATH and the process/arguments (/CTS 2 2 2) currently have to be set manually during the import, input/output has to be specified as (non-existent) "/input", "/output".

Figure 4. CiTAR landing page example.

Access and Integration

An archived software environment requires versatile access options to be useful, in particular in the research domain. For instance, the user might want to interact with an archived Web server or re-run an archived data analysis software with new or modified data.

As an example for perpetual access to software-based research resources, we have used the outcome of the SlaVaComp project (2013-2015), which created an electronic meta glossary of regional and diachronic varieties of Church Slavonic – a language that was used in the Orthodox Slavia between the 10th and 16th centuries. Until the creation of a digital database, researchers had to consult printed dictionaries, which meant that even simple lookups could take a tremendous amount of time. Fifteen printed Church Slavonic and Greek glossaries with various regions of origin were combined into an easy-to-use online web-based application. As the support for the underlying server operating system will expire in the near future, the SlaVaComp service's future is uncertain. Even if the operating system is upgraded to the next long-term support version, there is no guarantee that any other software dependency e.g. the database remains compatible or has long-term security support, crucial for a public online service. Furthermore, it is highly unlikely that former employees could adapt the service to a modern software stack, mainly because they have left after the project ended and with them most of the specific knowledge about the software they created. This fate is shared by numerous software developments that emerge from scientific projects. The costs for maintaining a server and the software after the end of a project are usually not covered, especially if these projects are only of interest for a small and specialized research community. Leaving an unmaintained, outdated machine connected to the internet poses a latent and increasing security risk.

The main challenges for accessing out of production software setups are security, simplicity of access and workflow integration. These environments have been archived to remain in its original state, with all the potential security threats. Worldwide network-based access (i.e., public Internet access) would promote re-use and would simplify integration into current tools and workflows, but would pose unmanageable security threats. In order to provide (secure) network access, all archived machines are deployed (on-demand) in their own private network and we provide several options (port-forwarding / NAT, SOCKS and local gateway) to access preserved software. Non-interactive containers or VMs can be accessed via a landing page and can be connected to a data-source or data-provider for input data.

Emulating Networks

We identified that Ethernet can serve as a universal common denominator for network types. While higher-level network protocols like IP are not necessarily implemented on all archived systems, for layer-2 protocols Ethernet is predominant; other layer-2 protocols like Token Ring were neither widely deployed in archived systems nor are they in use in contemporary systems. Ethernet, thus, can serve as a basis for network access. Therefore, we spawn a virtual Ethernet network for every emulated environment, consisting of a software-based Ethernet switch with an arbitrary number connected to the single nodes of the environment. To exchange network traffic between nodes in this this virtual network, Ethernet frames are encapsulated in WebSocket over TLS connections and transported over the public Internet. This allows for a strict separation of virtual network traffic from traffic on the host environment. This approach eliminates any danger from archived environments attacking the host system or using the host system's network resources to attack third-party entities on the Internet (or the host system's private network environment). It also shields the archived (and unmaintained) environments from the attacks from the public Internet. At the same time, this approach also helps with reproducibility of the archived software's output. Archived software might interact with other network components or download resources from the Internet, which has to be contained and controlled to guarantee reproducibility at a later date. Otherwise, external resources might either respond differently at a later date or not be available any more at all.

Within the virtual Ethernet network, we have implemented different auxiliary services, which provided infrastructural services normally present in a network and can be seen as building blocks for an emulated network environment. They can be configured and added to the environment by a curator via the user interface. One such service is a DHCP and DNS server, which can be turned on by the user to provide IP addresses to emulated environments. Additionally, the user can assign domain names to single environment under which the environment will be reachable in the emulated network environment. While, technically, this functionality is provided by the described DNS server, the assignment of domain names to environments happens and is saved in a more descriptive manner not coupled to the specific implementation of the domain name resolution mechanism. This allows to change the implementation or reuse this metadata in other contexts. For instance, one might imagine a search over network metadata which can provide an archived/emulated version of a specific domain.

A second building block is the possibility to allow emulated environments to access the (live) public Internet. This functionality is implemented by a software component which can act as a gateway in the virtual network and forward any (TCP/UDP) connections to a physical network on the host system. Traffic between the environments and the gateway is still encapsulated in the described way in a WebSocket connect over TLS. This makes it possible to place the gateway on a special machine separated from the rest of the rest of the machines (it might even be placed on an externally rented server reachable via the Internet), decreasing the potential security impact on the host's network. Figure 5 depicts the CiTAR virtual network schema.

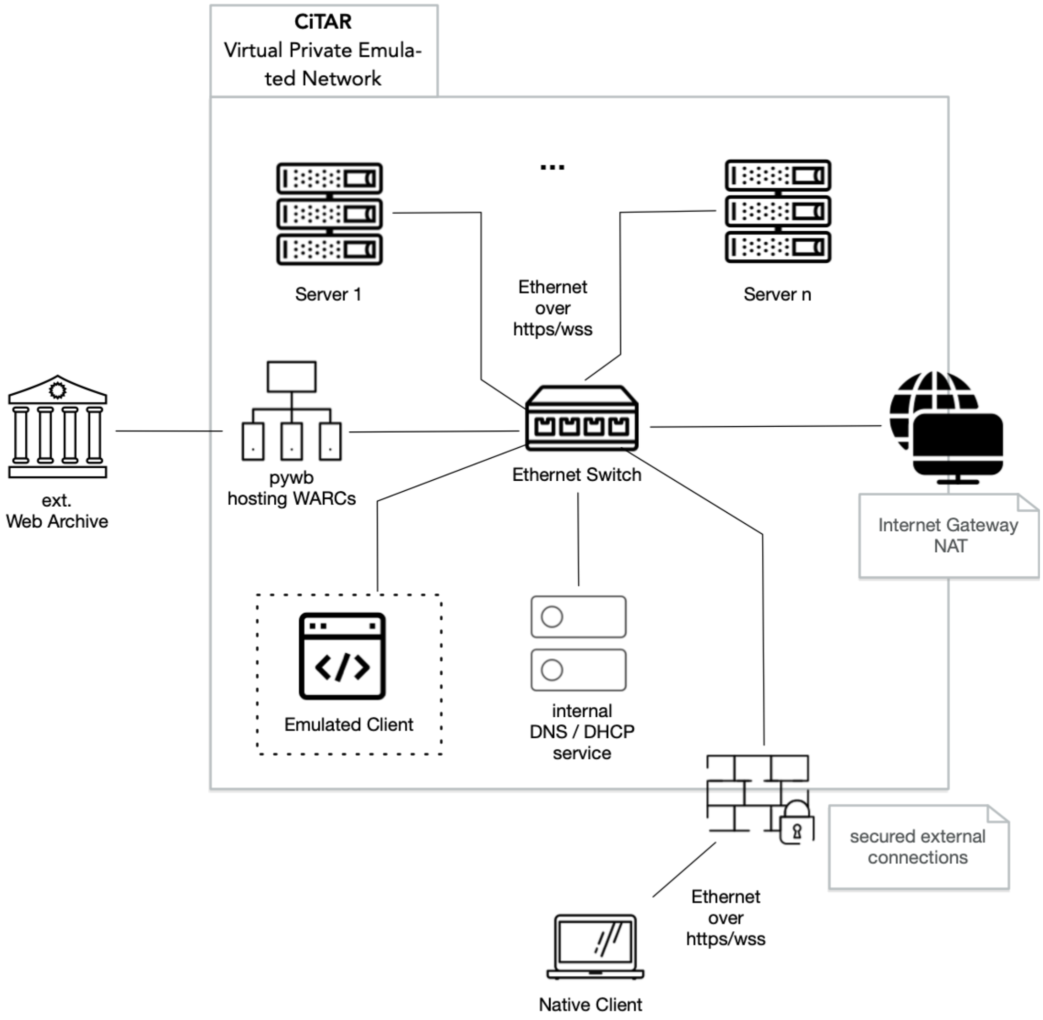


Figure 5. CiTAR emulated network components.

A third and final building block provides the ability to forward traffic from external sources like the public Internet to the emulated network environment. To convert TCP connections to Ethernet frames transported in the virtual network, this block, technically, has to act as a TCP/IP stack. Different access scenarios can be offered using this approach. Firstly, the user can download a local application (eaas-proxy, currently available for Windows, macOS, and GNU/Linux) and allow their Web browser to start it in the context of an emulation session. The local eaas-proxy will open a local server socket and forward any connection (again via an encrypted WebSocket connection over HTTPS) to the emulated network environment. The target emulation environment and port inside the virtual network can be configured by the curator during ingestion. Alternatively, eaas-proxy can be configured to act as a SOCKS5 proxy server.

Finally, eaas-proxy can be run on a temporary server and expose a port of an emulated environment or a SOCKS5 proxy server directly to the user. While, in this case, the user will not have to install any local software on their computer (an individual public IP/port will be shown for each emulation session instead), the connection will be unencrypted. It also might not work in (enterprise) networks which block external connections to non-standard ports, while a local eaas-proxy only has to be able to open a WebSocket connection to a HTTPS server on its default port.

As a last option, as eaas-proxy was implemented in JavaScript, it can be run directly in the user's Web browser. This is especially helpful if the archived environment consists of one or several Web servers and was implemented as a prototype. Using W3C's Service Workers specification, any HTTP requests of the browser to the archived Web server can here be intercepted, serialized to Ethernet frames on the client and sent via a WebSocket connection directly to the emulated network environment. As the archived Web server is never exposed directly to the Internet in this case, this helps to decrease risks with both attacks from the Internet on the (unmaintained) Web server as well as attacks from the (potentially malicious) archived Web server to the user's Web browser.

Example

Figure 6 shows the output of SlaVaComp as a virtual machine and a Docker container containing the Chromium browser were imported to CiTAR. They can now be run together in the same virtual network environment. The user can then access the archived SlaVaComp Web server via the Chromium browser. This does not directly expose the emulated environments to the public Internet and allows continued access to SlaVaComp using the archived Web browser even if future Web browsers were to drop features used by the archived Web application (e.g., Adobe Flash).

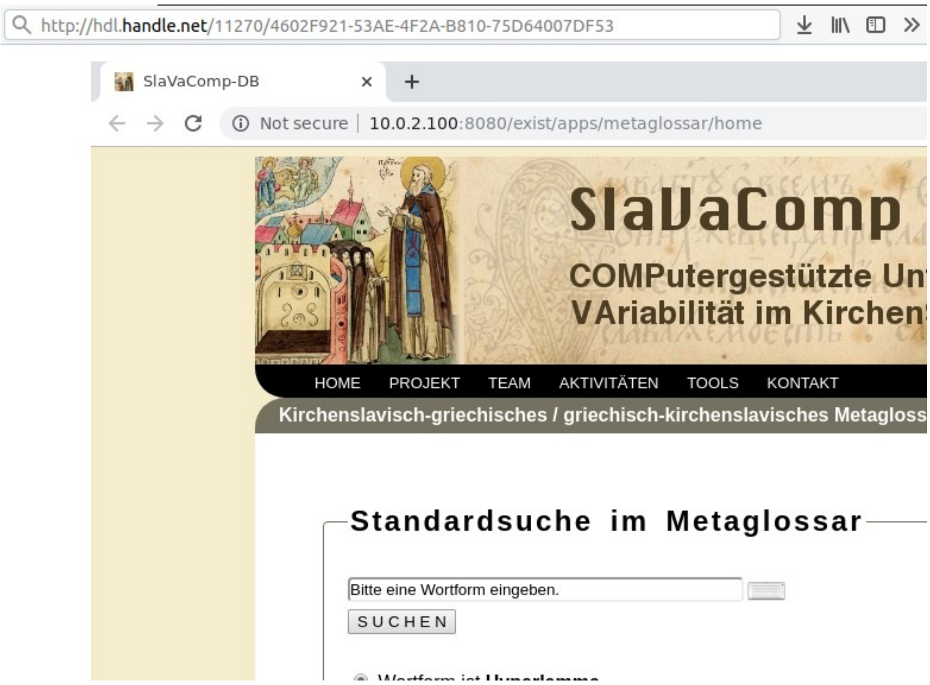
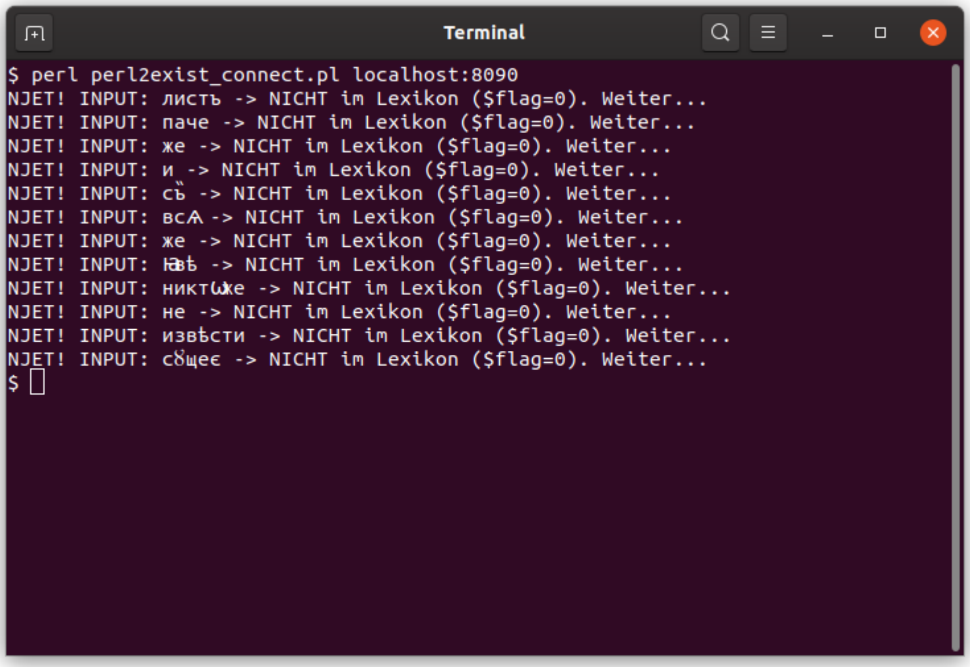

http://hdl.handle.net/11270/4602F921-53AE-4F2A-B810-75D64007DF53
SlaVaComp-DB
Not secure | 10.0.2.100:8080/exist/apps/metaglossar/home
SlaVaComp
COMPutergestützte Un
VAriabilität im Kirchen
HOME
PROJEKT
TEAM
AKTIVITÄTEN
TOOLS
KONTAKT
Kirchenslavisch-griechisches / griechisch-kirchenslavisches Metagloss
Standardsuche im Metaglossar
Bitte eine Wortform eingeben.
SUCHEN
Wortform ist Hyperlemma

Figure 6. Accessing a preserved database with its original web-based frontend.

In Figure 7 the same emulation session, eaas-proxy was run on the user's local computer to connect their computer to the virtual network. A Perl script developed in the SlaVaComp project can then access the archived SlaVaComp environment from the local user's computer in a secure way.



```
$ perl perl2exist_connect.pl localhost:8090
NJET! INPUT: листъ -> NICHT im Lexikon ($flag=0). Weiter...
NJET! INPUT: паче -> NICHT im Lexikon ($flag=0). Weiter...
NJET! INPUT: же -> NICHT im Lexikon ($flag=0). Weiter...
NJET! INPUT: и -> NICHT im Lexikon ($flag=0). Weiter...
NJET! INPUT: съ̏ -> NICHT im Lexikon ($flag=0). Weiter...
NJET! INPUT: всѧ -> NICHT im Lexikon ($flag=0). Weiter...
NJET! INPUT: же -> NICHT im Lexikon ($flag=0). Weiter...
NJET! INPUT: ꙗвѣ -> NICHT im Lexikon ($flag=0). Weiter...
NJET! INPUT: никтѡже -> NICHT im Lexikon ($flag=0). Weiter...
NJET! INPUT: не -> NICHT im Lexikon ($flag=0). Weiter...
NJET! INPUT: извѣсти -> NICHT im Lexikon ($flag=0). Weiter...
NJET! INPUT: сꙋщеє -> NICHT im Lexikon ($flag=0). Weiter...
$ 
```

Figure 7. Perl script running batch job connecting to the preserved database.

Conclusion

With CiTAR we have explored options to publish and preserve software-based research methodology as well as ensuring long-term access. Since both, operational knowledge vanishes and the technological environment changes quickly, a retrospective reconstruction of complex scientific workflows will become increasingly difficult. Preserving pre-configured and executable software-setups will simplify reproducing, re-using and validating software-based scientific workflows. Since there exist already concepts and tools to capture and encapsulate software-based experiments and workflows, CiTAR builds its preservation workflow on-top of these existing tools and practices as well as on top of existing compute, storage and preservation infrastructure. To reduce maintenance of a diverse and technically complex technology and fast-growing number of user-provided objects, we propose a generalization process for VMs and containers, such that a small number of hardware-independent and portable software-runtimes are able to render a large number of archived objects. Finally, we addressed challenges of accessing preserved scientific software setups, by providing secure networked access to preserved software services, both for user-machine and machine-machine interaction.

References

Ball, A. (2012). Review of data management lifecycle models. University of Bath, IDMRC.

Baker, M. (2016). 1,500 scientists lift the lid on reproducibility. *Nature News, 533*(7604), 452.

Boettiger, C. (2015). An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review, 49*(1), 71-79.

Chard, K., Gaffney, N., Jones, M. B., Kowalik, K., Ludäscher, B., Nabrzyski, J., ... & Willis, C. (2019, June). Implementing computational reproducibility in the Whole Tale environment. In Proceedings of the 2nd International Workshop on Practical Reproducible Evaluation of Computer Systems (pp. 17-22). ACM.

Grüning, B., Dale, R., Sjödin, A., Chapman, B. A., Rowe, J., Tomkins-Tinch, C. H., ... & Köster, J. (2018). Bioconda: Sustainable and comprehensive software distribution for the life sciences. *Nature methods, 15*(7), 475.

Espenschied, D., Rechert, K., von Suchodoletz, D., Valizada, I., & Russler, N. (2013, September). Large-scale curation and presentation of cd-rom art. In Proceedings of the 10th International Conference on Digital Preservation (IPRES), Lisbon, Portugal, September 3–5 (pp. 45-52).

Howe, B. (2012). Virtual appliances, cloud computing, and reproducible research. *Computing in Science and Engg. 14*, 4.

Ivie, P., & Thain, D. (2018). Reproducibility in scientific computing. *ACM Comput. Surv. 51*, 3.

Lamprecht, A. L., Garcia, L., Kuzak, M., Martinez, C., Arcila, R., Martin Del Pico, E., ... & McQuilton, P. (2019). Towards FAIR principles for research software. *Data Science,* (Preprint), 1-23.

Macleod M. R., Michie S., Roberts I., et al. (2014). Biomedical research: Increasing value, reducing waste. *Lancet, 383*(9912).

Meng, Haiyan, Rupa Kommineni, Quan Pham, Robert Gardner, Tanu Malik, & Douglas Thain. (2015). An invariant framework for conducting reproducible computational science. *Journal of Computational Science 9*: 137-142.

Munafò, M. R., Nosek, B. A., Bishop, D. V., Button, K. S., Chambers, C. D., Du Sert, N. P., ... & Ioannidis, J. P. (2017). A manifesto for reproducible science. *Nature human behaviour, 1*(1), 0021.

Pimentel, J. F., Murta, L., Braganholo, V., & Freire, J. (2019, May). A large-scale study about quality and reproducibility of jupyter notebooks. In Proceedings of the 16th International Conference on Mining Software Repositories (pp. 507-517). IEEE Press.

Smith, A.M., Katz, D.S., & Niemeyer, K.E. (2016). Software citation principles. P*eerJ Computer Science 2*: e86.

Wilkinson M.D., Dumontier M., Aalbersberg I.J., Appleton G., Axton M., Baak A., & Blomberg, N. (2016). The FAIR guiding principles for scientific data management and stewardship. *Scientific Data, 3*.

Wehrle, D. & Rechert K. (2019). Are Research Datasets FAIR in the Long Run? I*nternational Journal of Digital Curation,* 13(1).