CiTAR - Citing & Archiving Research

Felix Bartusch, Jens Krüger*
High-Performance and Cloud Computing Group
Zentrum für Datenverarbeitung
University of Tübingen, Germany
*jens.krueger@uni-tuebingen.de

Klaus Rechert, Oleg Zharkov
University of Freiburg

Kyryll Udod
University of Ulm

***Abstract*—The importance of software and web services is changing. So it sufficed to state the software used during scientific data processing, when publishing results in journals. This also applies to web services, which can be methods used in scientific work as well as results of this work. Awareness for the evanescence of tools and services, which hinders the reproducibility of published work, increases.**

Similar to data repositories, which archive datasets and make them citable, a service for software artifacts would enhance sustainability of science. So we present the CiTAR (Citing & Archiving Research) service in this paper, which enables researches to preserve computational environments and make them citable. In contrast to pure data repositories, CiTAR guarantees the executability of archived environments by providing generic runtimes.

***Virtual Machines, Containerization, Long-term Archiving, Preservation, Reproducibility*—**

I. Introduction

While the institutional introduction of infrastructure for the collection and conservation of primary scientific data is currently under construction or partially exists [1], a parallel problem awareness arises for the associated models and methods, in particular for data evaluation [2], [3]. However, there is hardly any usable infrastructure and service offerings yet. Although the DFG (The German Research Foundation) recommendations on "good scientific practice" currently only suggest the retention of primary scientific data [4], the remainder of the recommendation refers to mandatory records of "materials and methods" that are not only necessary for comprehensible results but also for the publication process . If scientific results are to be reproducible, for example for an independent verification, a reconstruction of the experimental setup is necessary. However, in the digital age, with its extremely short life spans (and availability) of hardware and software components, replicating a data processing workflow that is identical in all components can not be achieved solely on the basis of records [5]. CiTAR (Citing and Archiving Research) 1 develops infrastructure to support computer assisted research. One major outcome of this project are means to publish, cite and provide long-term access to virtual research environments. The aim of this project is to develop a cooperative, multidisciplinary technical-organizational service in order to support teaching and research in the further development of "good scientific practice". The service should provide data and scientific methods jointly citable and reproducibly in order to meet the requirements of modern journals. CiTAR realizes reuse of research data and long-term availability in terms of a modern research data management.

The developed service provides automated import of virtual machines and popular container formats like Docker and Singularity. CiTAR assigns persistent identifiers to the imported research environments and provides resources to re-run the archived objects with external data.

II. Related Work

Computational science communities have already recognized the need for reproducible compute-based research results to improve scientific practice and to create sustainable results [6]. While publication and citation of research data has made progress recently, management of software-based research methods remains an open challenge. "Software is a critical part of modern research and yet there is little support across the scholarly ecosystem for its acknowledgment and citation." [7]. Concepts and practice of software citation are currently discussed [8], but software also needs to be available, i.e. retrievable from a dedicated software-archive. Currently, several projects are working on software preservation concepts and services [9] [1] [2] [3]

However, if software reproducibility is defined as "the ability for someone to replicate a computational experiment that was done by someone else, using the same software and data, and then to be able to change part of it (the software and/or the data) to better understand the experiment and its bounds[.]"[4] then just availability of software is usually not sufficient. A software-based research process may contain multiple individual software components. And even with detailed documentation - if available at all - manually rebuilding a complex software setup is a laborsome and error-prone process, in particular because (implicit) operational knowledge is lost over time. Additionally, all of the preserved softwares dependencies also need to remain available, e.g. operating system, libraries, build environment and a suitable hardware

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

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

[3] Source Code Archive https://archive.softwareheritage.org/

[4] https://danielskatzblog.wordpress.com/2017/02/07/is-software-reproducibility-possible-and-practical/

platform are necessary to run the whole software setup. Hence, reproducible computational science requires a portable, self-contained software setup and additionally, a defined runtime environment.

We will shortly discuss some methods used to tackle some parts of the reproducibility aspects stated above. Prominent container hubs like Docker Hub [5] and Singularity Hub [10] offer a service for storing and referencing container images. But the hubs do not provide resources and runtimes for the execution of containers. There is also no guarantee that future container runtimes will run older containers. Researchers started to containerize workflow managements systems together with explicit pipeline descriptions and make them available via the mentioned container hubs [11], [12].

Platforms like Figshare [6] and Zenodo [7] offer repositories for storing research data and share it with other researchers. Figshare provides a DOI for stored projects, and thus make it citable. Researchers currently usually use it to share their research data and scripts. In principle also containers and virtual machine images could be stored via these repositories. But as for container hubs, there is no possibility to guarantee their executability in the future.

The platform Code Ocean [8] is a cloud-based computational reproducibility platform. It allows researchers to create so-called 'capsules' with pre-defined environments and run their code on their data in these capsules. The environments are mainly based on the popular operating systems Ubuntu. There are also environments with often used software like R or Python. Although one can install additional packages and software to the pre-defined environments, CiTAR enables users to preserve custom environments created without the limitations of pre-defined operating systems or software versions.

III. Implementation

In this section we will describe in detail the implementation of the CiTAR components and how they are composed. The CiTAR service consists of the following components: the graphical user interface (frontend) providing self-service user workflows, the Emulation-as-a-Service (EaaS) backend is used for implementing a generic container and virtual machine runtime and an image archive faced, used to orchestrate storage services.

A. CiTAR components

The main design goals of the CiTAR service were building workflows and a light-weight distributed service on top of existing storage and compute infrastructure.

1) CiTAR Container: The software stack needed by the CiTAR service is provided as a self-contained Docker container to simplify setup and deployment. docker-compose is used to run the CiTAR container and starting the service. The compose file specifies mounts for the CiTAR components described below and the mapping of the host IP-address to the container network interface.

2) Graphical User Interface (Frontend): CiTAR provides a simple graphical frontend to provide self-service workflows to the user. The user interface implements the CiTAR RESTful API and can be implemented and deployed independently of the CiTAR backend. Two example implementations are currently available, an administration backend and a user-facing landing page. Figure 2 depicts the interface elements responsible for archiving virtual machine and container images. Figure **??** depicts a formatted environment's landing-pageas.

3) EaaS-Server (Backend): The EaaS-server provides functionality for to maintain virtual machine and container images, e.g. normalizing images during import as well as necessary means to run the archived archived environments. How the images are imported, normalized and run will be explained in detail in subsection III-B. Since running virtual machines and containers is compute intensive, EaaS is implemented as a scalable, distributed Cloud service. A gateway component provides a RESTful API for the user frontend as well as for machine-machine interaction. On demand, e.g. a request to instantiate a virtual machine or container, the gateway is able to allocated compute resources as needed. CiTAR can be configured to run within a fixed size compute cluster or as a scalable Cloud service. Currently, Google Compute, AWS and bwCloud (OpenStack Nova) can be used to allocate compute resources on demand.

4) Image Archive: Archival of imported virtual machine and container images is organized by the image archive component. The image archive provides a unified API for compute nodes to access images as well as their technical metadata. While the current CiTAR container ships with a simple file-backed image archive, bit-stream storage services are not a part of the CiTAR service. Especially for long-term preservation there are dedicated services available. Instead CiTARs image archive is implemented as an simple API facade, which can be implemented for different storage backends (e.g. AWS S3) as well as archival repositories (e.g. Preservica []). Currently the image archive also maintains access to the archived environments technical metadata. This metadata describes the images and their technical dependencies, such that the images (both container and VMs) can be instantiated by the EaaS compute nodes. Beside technical information the hypervisor or emulator configuration the the metadata my contain further information about input and output directories of the container, the process to run inside the environment, and environment variables which should be set when executing the image. In general, CiTAR only maintains metadata that is required to re-run containers and VMs as well as to ensure their longevity, i.e. preservation planning, for instance, ensuring migration from an obsolete hypervisor or emulator to a new one. Even though, the CiTAR example UI and its backend provide infrastructure to store and maintain also descriptive metadata, descriptive metadata should be maintained in respective repository systems or library catalogs which use CiTAR as a functional backend, to enable access to a listed research method.

[5] https://hub.docker.com/
[6] https://figshare.com/
[7] https://zenodo.org/
[8] https://codeocean.com/

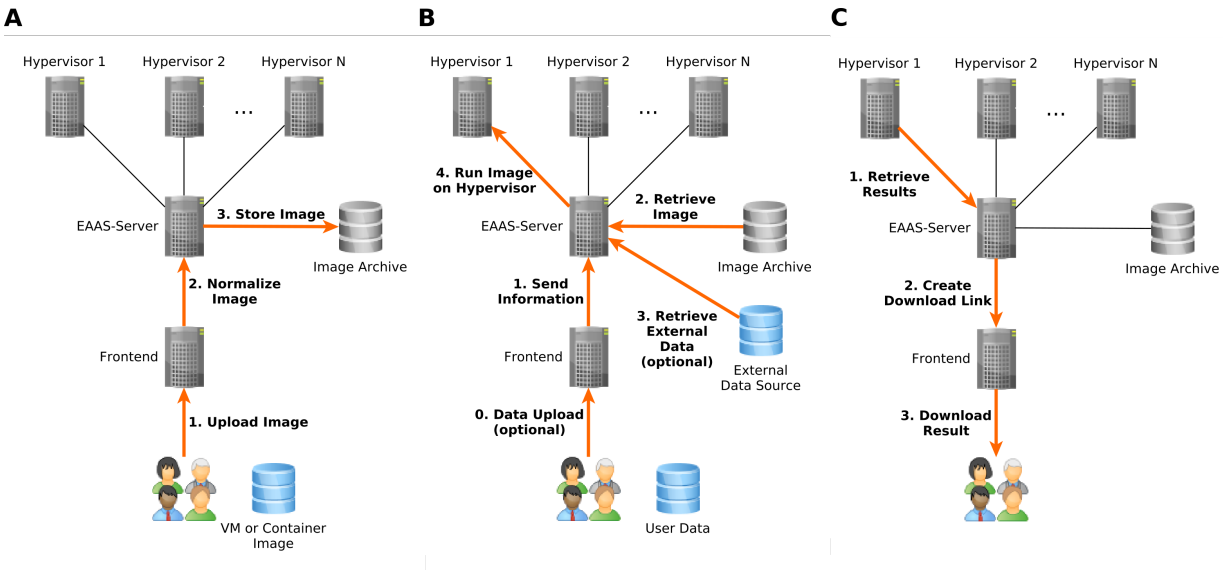

Fig. 1. Schematic depiction of CiTAR architecture and typical workflows. **A** User preserves a image. The preservation is started via dialog in the frontend. The image is then normalized by the EaaS-server and stored in the image archive. **B** A user wants to execute an archived environment by specifying the environment to run and additional data sources (from local machine or external). The image is first retrieved from the image archive and then started on a hypervisor. **C** The output of the run is retrieved by the EaaS-server and can be downloaded via a link.

B. CiTAR Workflows

This section describes the technical workflows of archiving, citing virtual machine and container images in CiTAR. The normalization of virtual machine images and subsequent virtualization in the EaaS-server component are based on the bwFLA project [13]. The methods for preserving software containers, which are implemented in CiTAR, are derived from earlier work [14].

1) Archiving virtual machine and container images: Archiving forms the first application of the CiTAR project. First, the user navigates through the user interface to the suitable input form for virtual machine or specific container software. For virtual machine images, the user specifies the underlying operating system, which results in a suitable emulator and configuration for the particular operating system. The user then specifies an URL to the disk image. Allowed disk image formats are Raw, Qemu, QCOW2, VirtualBox VDI. The following formats just have limited support: Virtual PC disks VHD, VMware VMDK. The user interface then sends a POST request to the backend. The request is handled by a Web Service Interface (SEI) implemented using JAX-WS. In the end, curl is used to retrieve the image file from the specified URL, followed by a conversion to QCOW2 if required. The QCOW2 image is then started by an emulator on a hypervisor via the EAAS-server. The screen of the virtual machine is displayed in the browser. The user is able to test run the machine and to further customize it to improve usability, e.g. change passwords, auto-start applications etc. All changes are tracked and result in versioned revisions of the machine. In the end one can specify a description used on the landing-page of the archived environment.

The archival procedure for software container images is similar (See Figure 1A). The supported container formats are the ones used by Singularity and Docker, as well as a simple container root filesystem as .tar.gz file. For Docker containers, the import from an DockerHub is possible by specifying the image name and tag. For Singularity images the user can specify the URL to an image or upload an image. The user also specifies the process, which should be executed in the archived environment later. This can be a simple command or script, which is available in the container. The user can specify two directories used as mount points for input and output data when the container is executed. The image is then retrieved and normalized. For Docker images, skopeo pulls the Docker image from Docker Hub and then extracts the root file system. For Singularity images, the create sandbox feature is used to extract the root file system. As for virtual machine images, the user has to provide a description of the archived container, which is shown on the containers landing page.

2) Citing archived images: CiTAR assigns a persistent identifier to uploaded images using the Handle system [15]. CiTAR uses an own prefix of the handle system and adds an universally unique identifier (UUID) of the image as local name to the prefix to build the handle. The handle points then to the the landing-page of the imported environment and can be used in publications to cite the preserved virtual machine or container image.

3) Execution of archived images: CiTAR ensures the executability of archived images by generalizing the images to a common format and providing a runtime for this format. An image can be executed via its landing-page. As for the import of images, the execution of virtual machine and container images differ. An example landing-page is shown in Figure 3.

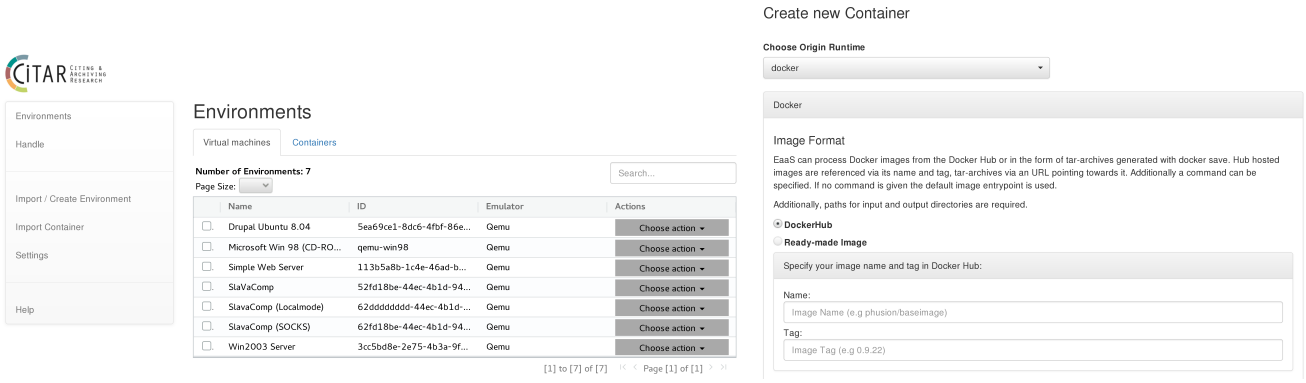

Fig. 2. Two screenshots of the CiTAR user interface. **Left** The preserved virtual machines are listed. The other tab of the 'Environments' section lists the preserved container images. The user can search preserved environments and visit the landing page of an environment via the 'Choose action' box. **Right** The dialog for importing a Docker container image is shown. A user specifies the name and tag of an image available on Docker Hub. Scopeo will then retrieve this image

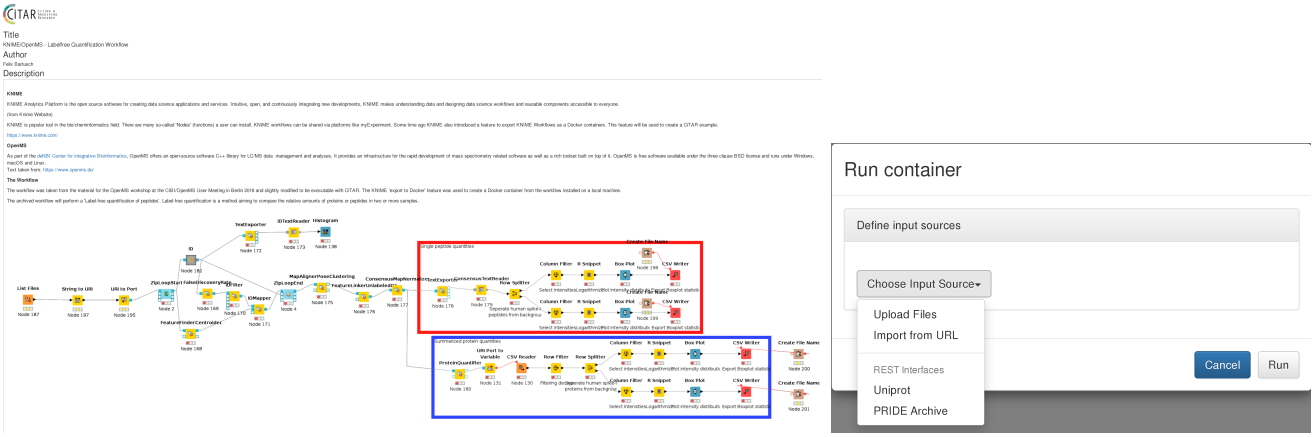

Fig. 3. Two screenshots of the CiTAR user interface. **Left** A landing-page of an preserved software container. The landing-page can be used to describe the preserved workflow in detail. A user can execute the environment from this landing-page. **Right** When running a container, the user can specify external data sources. The screenshot shows the dialog where the data source can be chosen.

Emulation based on the Emulation-as-a-Service (EaaS) framework with a corresponding preservation strategy for generalized/normalized disk images and VMs is used as preservation technology. The handle associated with the archived VM redirects the user to a landing page describing the virtual environment and possible interactions. Most of these machines target specialized communities and would not be accessed often, therefore, a visitor of the respective landing page initiates the startup of the instance. Provisioning and startup takes between takes at least 30 seconds, depending on the machines size and complexity.

The main challenge for accessing workflows is the security of archived machine. As these machines have been archived to remain in their original state, a (permanent) internet connection could be harmful. In order to provide (secure) network access, all archived machines are deployed in their own private network. The CiTAR gateway acts as a bridge between the private network and the users network. Depending on the security requirements and/or access infrastructure the we offer currently three different network access options: Port forwarding, SOCKS, and Local-mode. For local network deployments, network ports of an emulated server can be forwarded to a user-accessible network. For instance, the SlaVaComp landing page provides a Connection Information window, allowing the user to connect directly to the web application running on the internal port 8080.

If the machine provides multiple services (TCP ports), the SOCKS5 mode can be used. The Connection Information window will display necessary information for a local proxy configuration. Optional password protection is possible. In some scenarios, e.g. remote access, a fully private connection is necessary. Additionally, to use an archived machine with current research software multiple (local) network ports might be required. For this, CiTAR offers a local connection mode. The user has to install a small program (available for Linux, MacOS and Windows) which registers a CiTAR network URL-handler. Alternatively, source code and automated builds are available on GitLab. The use-case SLAVAComp presented in the results section states an example landing-page for each of the three connection methods.

Running a preserved container is slightly different. In scientific context, software containers are usually used to for the reproducible and easy deployment of tools and workflows. When using the preserved tool, one wants to inject arbitrary data into the container. Ideally the landing-page states which data and format is required. In this way, it is possible to apply archived methods to new datasets. Supported methods are uploading datasets from the workstation, specifying URLs to the datasets. In principle CiTAR can retrieve datasets from research data repositories (See Figure 1B). We implemented support for two bioinformatic data repositories: Uniprot for sequence data and PRIDE for proteomic datasets (see Figure 3). After the process has finished, the results saved in the containers output directory are packaged and the user can retrieve this data by following a download link (See Figure 1C).

IV. Results

As an example for perpetual access to software-based research resources, we have used the outcome of the SlaVaComp project (2013-2015) [16], 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 services 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 landing-pages for each connection method are accessible via the following handles:
CiTAR SlaVaComp Landingpage Example:
http://hdl.handle.net/11270/52fd18be-44ec-4b1d-94b4-887fa139142815
CiTAR SlaVaCom Landingpage (SOCKS mode)
http://hdl.handle.net/11270/62fd18be-44ec-4b1d-94b4-887fa139142815 CiTAR SlaVaComp Landigpage (local mode):
http://hdl.handle.net/11270/62ddddddd-44ec-4b1d-94b4-887fa139142815

To illustrate how CiTAR works with software containers we use a tutorial workflow provided by the developers of OpenMS [17] for the workflow engine KNIME (Konstanz Information Miner) [18]. OpenMS is a open-source software used to process mass spectrometry data. With KNIME one can create workflows by adding and connecting so-called nodes. Nodes provide configurable functions to process data in the workflow. The user can install additional nodes in order to add functionality to KNIME. The OpenMS developers created KNIME nodes that are extensively used in the example presented here.

This use-case is a good example for a complex software stack used by researchers for processing their datasets. It would be difficult, if not impossible, to recreate the exact same software stack and explicit workflow description at some later point in time just from a written description. Therefore archiving such software stacks together with workflow descriptions using containers and CiTAR seems to be a solution for long-time preservation of computational methods. The workflow can be executed through the handle to its landing-page:
http://hdl.handle.net/11270/188B6194-EA68-4E59-88C2-61D652941E29

V. Discussion

We implemented a service for preservation of virtual machines and containers and applied it to preserve results of a scientific project (SlaVaComp) and scientific methods (KNIME workflow). The preserved environments are executable and accessible although the original research project ended (SlaVaComp) or the used software stack will be outdated in the future (KNIME workflow). This is an important result, as services and databases may not be maintained forever and workflows tend to be irreproducible over time. Hence, a service for the preservation of these databases and workflows is highly desirable. The current CiTAR workflows are suitable for preserving single machines and containers. As we have already implemented a virtual network and means to orchestrate networked machines, such that the CiTAR service is able to re-enact a networking group of virtual machines or containers. A suitable workflow for archiving of complex networked environments is to be implemented.

Many popular scientific applications such as machine learning frameworks highly depend on graphical devices (GPU) in order to increase their efficiency massively. Also some compute jobs on high performance computing (HPC) systems are executed on a set of compute nodes. Both types of computational methods are currently not supported by CiTAR.

Long-time access to datasets via data repositories in combination with computational tools preserved by CiTAR would in principle ensure the reproducibility of many scientific computations. As data general purpose and field specific data repositories are widely established, researcher should now focus on a portable, self-contained software setup.

ACKNOWLEDGMENT

The authors acknowledge the use of bwCloud funded by the Ministry of Science, Research and the Arts of the State of Baden-Wrttemberg. The authors also acknowledge the use of de.NBI cloud and the support by the High Performance and Cloud Computing Group at the Zentrum fr Datenverarbeitung of the University of Tbingen and the Federal Ministry of Education and Research (BMBF) through grant no 031A535A and the state of Baden-Wrttemberg through bwHPC and the German Research Foundation (DFG) through grant no INST 37/935-1 FUGG.

REFERENCES

[1] D. J. Lee and B. Stvilia, "Practices of research data curation in institutional repositories: A qualitative view from repository staff," *PLoS ONE*, vol. 12, no. 3, pp. 1–44, 2017.

[2] "Software with impact," *Nature Methods*, vol. 11, no. 3, pp. 211–211, 2014.

[3] K. Hinsen, "Platforms for publishing and archiving computer-aided research," *F1000Research*, vol. 3, no. 289, pp. 1–18, 2014.

[4] D. Forschungsgemeinschaft, *Sicherung guter wissenschaftlicher Praxis (Safeguarding Good Scientific Practice)*, 2013.

[5] D. Garijo, S. Kinnings, L. Xie, L. Xie, Y. Zhang, P. E. Bourne, and Y. Gil, "Quantifying reproducibility in computational biology: The case of the tuberculosis drugome," *PLoS ONE*, vol. 8, no. 11, pp. 1–11, 2013.

[6] R. D. Peng, "Reproducible research in computational science," *Science*, vol. 334, no. 6060, pp. 1226–1227, 2011.

[7] D. S. Katz, K. E. Niemeyer, and A. M. Smith, "Strategies for Biomedical Software Management, Sharing, and Citation," *PeerJ Preprints*, 2016. [Online]. Available: https://peerj.com/preprints/2640.pdf

[8] D. S. Katz, "Software Citations and the ACAT Community," in *Journal of Physics: Conference Series*, vol. 1085, no. 2, 2018.

[9] M. e. Jackson, "Checklist for a Software Management Plan," 2018. [Online]. Available: https://zenodo.org/record/2159713{\#}.XDSdeiDjJEY

[10] V. V. Sochat, C. J. Prybol, and G. M. Kurtzer, "Enhancing reproducibility in scientific computing: Metrics and registry for Singularity containers," *PLoS ONE*, vol. 12, no. 11, pp. 1–24, 2017.

[11] P. Ewels, A. Peltzer, D. Moreno, rfenouil, M. Garcia, S. Panneerselvam, marchoeppner, jun wan, S. F., aanil, S. Haglund, A. Jemt, P. D. Tommaso, L. Veeravalli, J. Alneberg, C. Davenport, R. Suchecki, M. N. Adulyanukosol, Francesco, and C. Wang, "nf-core/rnaseq: nf-core/rnaseq version 1.0," Aug. 2018. [Online]. Available: https://doi.org/10.5281/zenodo.1400711

[12] F. Bartusch, M. Hanussek, and J. Krüger, "Containerization of Galaxy Workflows increases Reproducibility," *Proceedings of the bwHPC Symposium*, pp. 16–19, 2017. [Online]. Available: https://publikationen.uni-tuebingen.de/xmlui/handle/10900/83810

[13] K. Rechert, I. Valizada, D. von Suchodoletz, and J. Latocha, "bwFLA A Functional Approach to Digital Preservation," *PIK - Praxis der Informationsverarbeitung und Kommunikation*, vol. 35, no. 4, pp. 259–267, 2013.

[14] K. Rechert, T. Liebetraut, D. Wehrle, and E. Cochrane, "Preserving Containers – Requirements and a Todo-List," in *Digital Libraries: Knowledge, Information, and Data in an Open Access Society*, A. Morishima, A. Rauber, and C. L. Liew, Eds. Cham: Springer International Publishing, 2016, pp. 225–230.

[15] S. Sun, L. Lannom, and B. Boesch, "Handle system overview," Tech. Rep., 2003.

[16] I. Podtergera, S. Mocken, and J. Besters-Dilger, *SlaVaComp-COMPutergest{ü}tzte Untersuchung von VAriabilit{ä}t im KirchenSLAvischen: Forschungsergebnisse*. Albert-Ludwigs-Universit{ä}t Freiburg, 2016.

[17] J. Pfeuffer, T. Sachsenberg, O. Alka, and M. Walzer, "OpenMS A platform for reproducible analysis of mass spectrometry data ," *Journal of Biotechnology journal*, vol. 261, no. May, pp. 142–148, 2017.

[18] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, K. Thiel, and B. Wiswedel, "KNIME - the Konstanz information miner," *ACM SIGKDD Explorations Newsletter*, vol. 11, no. 1, p. 26, 2009.