Container-based Dynamic Infrastructure for Education On-Demand

Sven Strickroth [1], Dirk Bußler [2] and Ulrike Lucke [3]

Abstract: Teaching requires different tools for collaboration and communication. University data centres often only provide one-size-fits-all solutions which are manageable and maintainable for them. However, these solutions are not flexible enough for teachers in many use-cases. Therefore, there is a need for a dynamic infrastructure that can provide e-learning tools with a high level of flexibility to university members on demand in a short time and that are still manageable as well as maintainable for data centres. In this paper a container-based architecture with a self-service portal named Cook.UP is presented and evaluated in a user study. The paper also discusses possible security aspects and outlines possible usage for research.

Keywords: software-as-a-service, platform-as-a-service, docker, blended learning

1 Introduction & Motivation

Different platforms for collaboration and communication are necessary for teaching. Teachers need or at least want to use a variety of (often web-based) tools for teaching and learning in blended scenarios to support (collaborative) content-creation, self-regulated learning, reflection, coordination, or communication. The (non-exhaustive) spectrum ranges from simple blogs, over wikis, polls, learning diaries, e-portfolios to chats. Not all demands can be fulfilled by the existing central one-size-fits-all Learning Management System (LMS). Also, the locally provided services compete with external products that may have better usability and more features.

Based on the GDPR and data protection concerns, teachers and learners want or are encouraged to use the university's infrastructure and therefore request certain services from the local data centres. Due to the nature of teaching and its preparation, applications are often selected just-in-time causing peaks in data centres and delays [Ze17] especially at the beginning of semesters. Moreover, setting up a service might sound like a one-time task; however, each service needs proper maintenance as most users are not competent enough. Therefore, data centres consider carefully what services (also in terms of plugins

[1] Ludwig-Maximilians-Universität München, Institute of Computer Science, Oettingenstraße 67, 80538 München, Germany, sven.strickroth@lmu.de; https://orcid.org/0000-0002-9647-300X
[2] University of Potsdam, Center for Information Technology and Media Management, Am Neuen Palais 10, 14469 Potsdam, Germany, dirk.bussler@uni-potsdam.de, https://orcid.org/0000-0001-8714-0257
[3] University of Potsdam, Institute of Computer Science, An der Bahn 2, 14476 Potsdam, Germany, ulrike.lucke@uni-potsdam.de, https://orcid.org/0000-0003-4049-8088

for LMS) they officially provide, because these need to be maintained for a longer period. This demand for innovation collides with the legitimate interest of data centres in the stable operation of its services [KLL17].

Hence, there is a need for dynamic infrastructure providing a wide range of software that on the one hand provides a high level of flexibility in education on demand. On the other hand, from the perspective of the data centre, there is also a strong need for automatisms and scalability. In order to fill this gap, we propose a container-based architecture for the automated demand-oriented provisioning of web-based services for university members.

The remainder of this paper is organized as follows: The paper begins with a presentation of the state of the art. It then describes the requirements for such an approach and details of the architecture. Then, security aspects are discussed and a user evaluation is presented. The paper concludes with a discussion, a summary, and an outlook.

2 State-of-the-Art

An easy approach for simple web applications would be to encourage the users to use a shared webspace provided by the data centres (not all provide web space any more for various reasons, e.g., University Potsdam). However, not all requested applications are simple (e.g., PHP-based) but require at least a database (relational or special ones such as MongoDB) or are Node.js-based (e.g., RocketChat) and, thus, require the installation of further software on a server. In addition, this approach does not scale for a growing number of applications or individual applications that suddenly become more heavily used.

As outlined in the introduction, LMS are often extendable using plugins and there exist a wide range of different plugins.[4] However, deciding whether to install a plugin is not an easy task, because as soon as it is installed, it needs to be maintained and users also want support for it. Moreover, all installed plugins run in the context of the LMS and, therefore, a "misbehaving" plugin can cause severe damage to the whole LMS. Instead of using a plugin, there are standards such as IMS Learning Tools Interoperability (LTI, cf. [Bo11, OT17]) that allow to integrate external services without the need to install additional plugins, however, these also need to be hosted somewhere and only "few" applications support such learning specific standards.

There are general approaches such as wiki farms [Ar09] which are provided by university data centres; however, such one-size-fits-all solutions don't provide much flexibility to the users or are mostly optimized for automation on the provider side: Teachers are often not granted admin permissions on the system and the customization and installation of

[4] e.g., Moodle: https://moodle.org/plugins/ lists more than 1,800 publicly available plugins, last accessed 2021-03-17

plugins is not possible or needs to be requested individually (as discussed above). Moreover, various types of software are requested which cannot be easily farmed.

Other approaches would be to deploy multiple services on a single server (possible threat to all installed services in case of a security incident), setting up new separate virtual machines (VM), or using container-based technologies (such as Docker). Data centres often provide VM housing for university members. However, one major problem here is that not everyone can administrate a server professionally. Therefore, centrally managed solutions are required. [Doe11] proposed setting up VMs (in the cloud) with a specific LMS automatically on request of teachers that can then be used exclusively. Still, all separate VMs usually contain many unrelated daemons (e.g., sshd, ntpd, …) and need to be kept up to date in terms of security updates – either manually or using some automatisms on each server (e.g., Puppet or Ansible; often there are only automatisms for the operating system, require manual development and do not scale for heterogeneous environments). Container-based technologies are already used for short-lived tasks such as automatic assessment of student programming solutions (e.g., [SHS15]), micro-services in data centres (e.g., [LCH14]), and can to a certain degree automatically managed in terms of deployment, scalability (e.g., using Mantl, Rancher, Kubernetes, …) and upgrading by design (disposable containers with centrally managed images). Also, containers are lightweight compared to whole VMs as they (should only) contain required binaries/ libraries for a service and, therefore, require less storage and memory. Moreover, there exists a quasi-standard for describing container images (so-called Dockerfiles) and repositories. However, the process of the initial setup for a service requires manual intervention.

In summary, all these approaches cause significant overhead for operating/maintaining, are not flexible enough, require a high number of virtual machines, or cannot be updated easily. The following approach tries to address these issues.

3 Requirements for a Flexible Self-Service Platform

As a first step to close the gap for getting flexible platforms for teaching, a qualitative interview study comprising 14 faculty members of four different universities in Germany was conducted [Ze17]. Participants were potential users (university teachers, academic staff) and system administrators. In the semi-structured interviews, participants were asked for the current application process, duration from the request to the first login, and problems with the current process as well as the usage of the provided services. The results show that, e.g., setting up a blog software is indeed still a manual process as outlined above, responsible persons in the data centre are not always clear, that an overview of existing/available IT services is missing, that services are not always deployed or modified in a timely manner for being usable for the concrete teaching scenario, and that there is a need for automatisms to cope with the demand for new services.

Based on the interviews, functional and non-functional requirements from different perspectives were deduced. The key requirements for requesting a service are:

- for *users*: minimize time-consuming reconciliations, get full access rights to interact with the service, easy start and maintenance of services, online requests with the ease of an e-commerce order, convenience and user experience comparable to commercial services.
- for *deployers*: identify the duration of operations and service responsibility at all times, introduce new services from official program sources, introduce new service versions from official program sources for constant updates.
- for *providers*: subordinate user services to the university domain scheme, utilization of the existing infrastructure to run additional services, re-using existing official service descriptions as a basis for new and for updating existing services.

More detailed requirements and their prioritization can be found in [Ze17]. These requirements were taken as an empirical foundation when designing the architecture of our proposed approach.

4 Implementing own Tool Sets as Recipes in Cook.UP

As Fig. 1 illustrates, Cook.UP consists of 5 components: the Cook.UP frontend (based on Angular), the Cook.UP backend (based on Node.js, Express.js and a database), container management software and HTTPS "router" (Rancher and Træfik), Catalog Repository, and a file storage on a network file system (NFS) server. The Cook.UP front-end can be used by university members with their central account via Shibboleth Single Sign-on.

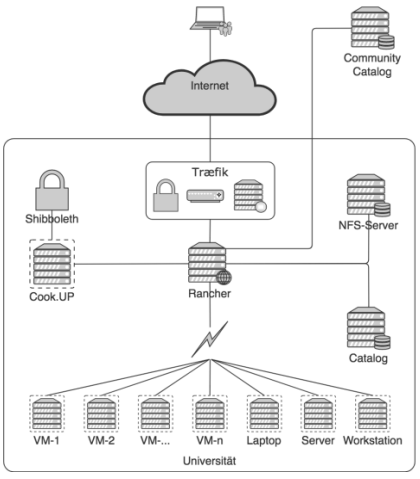

Fig 1: Architectural diagram of Cook.UP

In the main view, predefined so-called recipes for deployable services are provided, as depicted in Fig. 2. After selecting a service, the user is asked for a subdomain to be used, an optional starting date, the expiration date (at the moment a maximum of 3 months before and after the current semester, for security reasons; cf. section 5) as well as service specific data such as an initial password for the service. The view also lists all active services, allows to pause, update, or delete services. Furthermore, the user front-end provides web-based command-line shell access to the container for advanced users.

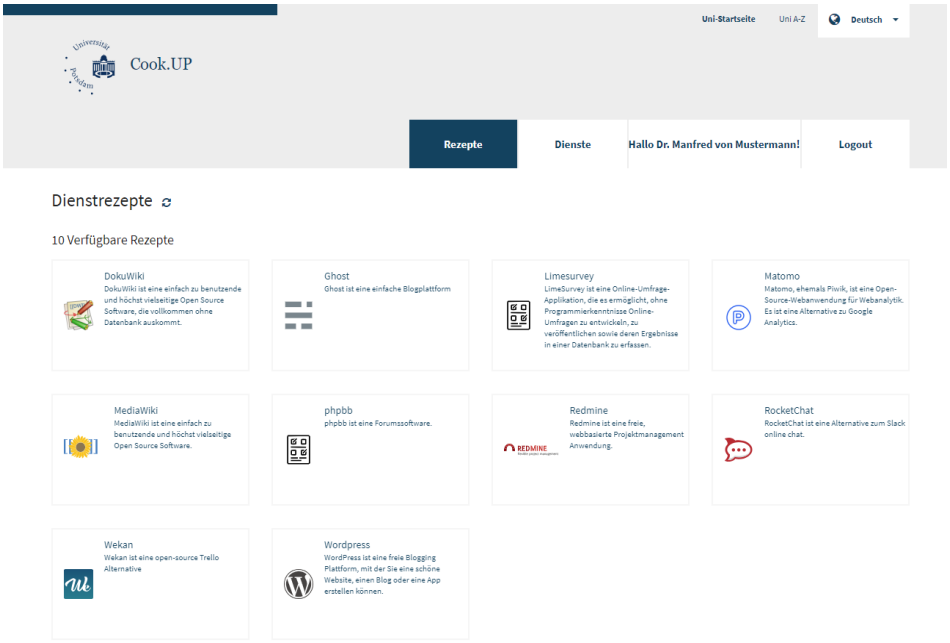

Fig. 2: Cook.UP front-end showing examples of possible recipes for web-based services to be deployed

All interactions with the front-end are forwarded to the Cook.UP backend via a RESTful API. The backend is responsible for the containers' life cycle, their association to the university members, unique service names and corresponding container volumes (stored on NFS), permission management, and communication with the container manager Rancher. When a new service is requested, it is stored in the database and in a queue for asynchronous bulk actions. The containers are automatically deployed and managed by Rancher. As soon as the service is created and available an email is sent to the requesting user. The Cook.UP backend is also responsible for periodic checks for expired services that are automatically stopped and deprovisioned as well as performing the update process of services.

Based on a comparison of existing container management platforms [Ze17], Rancher was selected because it is open source, is under active development, professional support is available, allows to combine an on-premises resources (bare-metal and VMs) with external cloud providers, supports small and growing set-ups, and supports different container management backends (such as Cattle, Kubernetes, and Docker Swarm) in a single platform with a central API. Træfik is used as an HTTP router/load balancer because this software is also open source and has built-in support for Rancher. Moreover, the front and backend are designed to run in containers on the very same infrastructure.

The available services need to be defined in a YAML format and are stored in Git-based a central catalogue repository together with docker-compose or rancher-compose files. These files describe how the containers are automatically created and what different helper-services need to be composed for a service (e.g., the blogging software WordPress consists of the WordPress web-application, a web server, and a database). Therefore, these descriptions can be seen as recipes that can be automatically executed for setting up a service including its configuration, runtime-environment, data storage etc. The proposed template format [Ze17] is based on the one used by Rancher whereas specific variables (e.g., service/subdomain name or initial password) are automatically filled in by Cook.UP before handing to Rancher. In addition to the internal repository, a community driven catalogue can be used, too. Due to security considerations (cf. later section) a private actively maintained repository should be preferred – still, the community driven catalogue can be used as a basis. Currently there are recipes for WordPress, Ghost (blog service), DokuWiki, MediaWiki, LimeSurvey, phpbb, RedMine, RocketChat, and Wekan (Trello alternative; cf. Fig. 2).

5 Security Considerations

For privacy reasons, productive services need valid server certificates and proper isolation against each other. The first version of the Cook.UP prototype used a single domain with a separate path for each service based on the service name. This, however, has security implications as all services share the same origin in the browser and, thus, could access all cookies for reading and writing. The final solution uses separate subdomains for each service, e.g. https://<servicename>.farm.university.tld. Still, a registration on the Public Suffix List[5] for *.farm.university.tld needs to be considered for proper separation of the origin. Træfik supports the automated application and deployment of Let's Encrypt certificates for sub domains, however, these certificates are discouraged by the German Research Network (DFN) and did not work reliably in the first place with Træfik. Since the DFN certificate authority doesn't provide an API for automatically issuing certificates, a good final solution without using wildcard certificates (which should be avoided and were not available until 2020) is still an issue.

[5] https://publicsuffix.org/

A major security aspect is to keep the software up to date. Regarding the included application software, some tools such as WordPress come with a built-in automated updater. However, there were cases in which these did not work accordingly or were manually disabled. Docker containers often also include a "full" lightweight Linux system with specific applications/libraries (e.g., the web server) that need to be kept up to date. In general, the longer a container is alive, the more important are such updates (likely). As an easy mitigation against possible security vulnerabilities the services have a defined maximum lifetime of one semester (plus pre and post phase). This is not an optimal solution yet. To mitigate automated attacks from the Internet, Cook.UP could be extended to restrict access to services to the internal University network that is also accessible using VPN by default. Here, other expiration rules could apply compared to public services. As containers were invented to be disposable, one could think that just rebuilding the contains and keeping the data volumes might be an easy solution. There are two issues to be considered: On the one hand, this does not work automatically if manual changes outside the persistent volume were made using the shell access. A solution could be to disable the shell access or using separate Dockerfiles or (idempotent) shell scripts provided by users which are applied on top. Here, separate expiration rules could be applied whether shell access is activated or not. However, developing such scripts is not easy for ordinary users, but only more advanced users are likely to need these freedoms. On the other hand, the images need to be regularly rebuilt (locally) with the Docker build cache mechanism disabled (can be done fully automatic). The reason is on the one hand that the caching mechanism prevents rebuilding an image when the base image did not change. As the base images are usually trimmed for size they change rarely, and therefore, the cached layer is used for additional package installations even when there are security updates available. On the other hand, there are Docker containers provided by people who only rebuild an image when there is an update of the main service of the container. This means that the latest security updates of the base image are not included in the provided image until there is a new release of the main contained service.

A lot of web applications need to send mails to function properly (e.g., to send notifications or password reset mails). For this a separate mail-server on a separate IP should be used to prevent blacklisting the main university mail server in case of SPAM-flagged mails. Also, quotas could be configured, or mail delivery restricted to the local university mail domain.

As Docker containers are isolated against the host and each other and Rancher provides secure communication between the containers, these aspects as well as best practices such as dropping capabilities and using a non-root user are not discussed here.[6]

[6] e.g., https://sysdig.com/blog/dockerfile-best-practices/, last accessed on 2021-03-17

6 Evaluation

The prototype was evaluated using a qualitative semi-structured interview study accompanying the Technology Acceptance Model [Da89]. During the study, the participants had the task to request a service, access it, and delete it again afterwards. Eight teachers and system administrators from two German universities in Germany took part in the evaluation – except for one person all persons took already part in the first interview study. All participants were able to fulfil the task.

The interviews were recorded and partly transcribed – the key results from the interviews are: From the user's side, there were statements such as "I would like to use it immediately" – also the usage of Cook.UP was seen as easy enough. It was also stressed by the teachers that usage of the services should also be possible across different universities. The necessity for updates was seen by all participants, however, the consequences for fully automatic updates varied. On both sides, a possibility of automatic updating is expected, however, with the current design the possibility for updates was evaluated positively. In general, the system administrators were satisfied but aspects regarding the usage of the university account, the concrete adaption of service versions from official program sources, and for estimated manual efforts in deployment were seen critically. However, the degree of additional efforts was rated differently under the system administrators: On the one hand, the easier availability of services is seen as a reason for higher demand for support; on the other hand, the availability of fully automatically configured services is expected to lead to a decrease in requests for setting these up. Only two features were partly rated as insufficient or missing: the stability of the deployment process of the prototype and the technology choice. Still, the proposed approach as seen as a good starting point for further developments.

The Technology Acceptance Model combines the perceived usefulness ("the degree to which a person believes that using a particular system would enhance his or her job performance" [Da89]) and perceived ease-of-use ("the degree to which a person believes that using a particular system would be free from effort" [Da89]) as key indicators of willingness to use a new technology. The model is particularly helpful for systems that are not yet in use. The core component of the model is a standardized questionnaire for which a 7-point Likert-scale was used in the study (1 very unlikely to 7 very likely). From the users' side Cook.UP was rated positively by 80 to 100 % of the participants [Ze17]: the perceived usefulness (PU) was rated as Ø=5.9 and the perceived ease-of-use (PEOU) was rated as Ø=5.8. The providers' side shows fewer positive results, the PU rating is Ø=5.0 and the PEOU rating is Ø=5.7. The results should be interpreted with caution due to the small number of participants. Particularly, from the provider's side one person one person stands out with significant worse ratings compared to the others.

In summary, the prototype is rated overall positive and acceptable from the providers' and the users' perspective. Future work includes improvements to stability and the extension

of available services. Also, usage examples shall be provided to teachers to simplify tests and configurations.

7 Discussion

Existing LMS already provide a broad range of features such as a Wiki and allow to build complex learning designs. However, from the interviews as well as our own experience, we have learnt that teachers still request specific services or also use external (commercial or freemium) tools (often with privacy doubts) because the integrated features do not seem to fit their needs or expectations regarding functionality or usability. Hence, teachers seem to see a higher benefit of using an "external" service compared to the possible better learning design integration in a LMS (as the proposed approach is web-based it can also be linked but the data transfer is be limited if the service does not have a specific interface such as LTI). The proposed approach does not reduce and might lead to further fragmentation of services. However, this must be seen in the context that teachers frequently use (different) external services anyway.

Closely linked are the statements of participants of the requirements interview study that often there is no overview of services officially provided by the central data centre. On the one hand this might be the reason why data centres requests for specific new services without knowing the reason for what it should be used and why the specific service is necessary. Figuring this out causes even more effort for a comprehensive consulting for which the system administrators in the data centre don't have the time or might not be qualified or the right contact person. On the other hand, this might be another reason why known external services are preferred. Approaches that also consider concrete teaching scenarios for recommending specific tools such as Tool.UP [KLS15] should be considered and also be integrated with Cook.UP so that teachers find suitable tools more easily, further service fragmentation can be mitigated, and can directly set them up.

In this paper only the teacher and system administrator perspectives were evaluated. On the one hand, having such a dynamic infrastructure at hand could also be very beneficial for students for building or integration of helpful services into their own personal learning environment (PLE). An integration into existing PLE systems such as Campus.UP [HKL14] should be considered. On the other hand, dealing with a larger number of separate services with a limited lifetime might not be optimal for students. In the current form (depending on the service), students would need to handle credentials for each service, data might be fragmented, and students might not be able to access course material in later semesters. These are valid issues that need to be addressed, however, those are also true for external services. Regarding the first issue, using an out-of-the-box integration of Single Sign-on into the services should be considered (e.g., by adapting the recipes). As the services are web-based, the contents can also be saved quite easily to mitigate the second issue.

For productive use, it was problematic that Rancher did not provide high availability of the container management software out of the box (at least with the used version). If the VM with the Rancher instance failed, then Cook.UP was no longer functional regarding the management of services. A possible solution is to use a different management software or a different Rancher backend which requires additional effort for development, deploying and testing. This solution could, then, also enable synergy effects and form a basis for other services that will be operated by the data centre in the future based on that container technology on the same infrastructure (cf. [Bu21]).

8 Summary & Outlook

In this paper a dynamic infrastructure approach based on container technology for supporting education was proposed and evaluated. Key points of the proposed approach include to support non-IT users in easy and automated deployment of web-based platforms, which are also operable by the data centre with manageable effort. Without this approach users had to request manually configured services or, if not possible, users often requested whole VMs for deploying their services without having the necessary competencies to properly configure and manage a server.

The prototype Cook.UP was positively evaluated with potential users and data centre staff. The proposed approach is not yet in productive use but provides a solid technical foundation. Open points are: solving the high availability issue (even if it was rated low priority by the users), integration into the central backup, measuring the performance and scalability, improving the update mechanism (in order to optimally remove the automatic expiration policy; resilience tests for recovering from failed updates), working on solutions to allow more advanced users to provide/use/exchange their own recipes, support for student containers, and sustained maintenance of the prototype. In general, introducing such an infrastructure can also be seen as an enabler for data centres to explore and use new technologies for their own services, too [Bu21].

Based on the general architecture of Cook.UP, it would be feasible to consider its use also for the research context. Creating and establishing a container-based research environment could greatly improve cooperation of researchers as well as transparency and reproducibility of research results [Co15, St19]. The even more salient conflict between reproducibility by keeping the original versions of systems, software, and libraries, while security strongly requires keeping systems up to date is not yet solved. Current initiatives in research data management[7] take up this approach for building an environment that fit better to researchers' needs.

[7] https://nfdixcs.org/

Acknowledgements

The authors would like to thank Manuel Zedel for the requirements analysis, design, implementation, and evaluation of Cook.UP in his Master thesis. Also, the authors are deeply grateful to all participants of the study for sharing their valuable perspectives.

Bibliography

[Ar09] Arazy, O., Gellatly, I., Jang, S., & Patterson, R. (2009). Wiki deployment in corporate settings. IEEE Technology and Society Magazine, 28(2), 57–64.

[Bu21] Bußler, D., Lucke, U., Strickroth, S. & Weihmann, L. (2021). Managing the Transition of Educational Technology from a Research Project to Productive Use. SE-SE 2021. Proceedings of the Software Engineering 2021 Satellite Events. CEUR-WS, Vol. 2841.

[Co15] Collberg, C., Proebsting, T. A. & Warren, A. M. (2015). Repeatability and Benefaction in Computer Systems Research: A Study and a Modest Proposal, TR 14-04, University of Arizona.

[Da89] Davis, F.D. (1989). Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. MIS Quarterly 13/3, 319–340.

[Bo11] Booth, S., Peacock, S. & Vickers, S.P. (2011). Plug and play learning application integration using IMS Learning Tools Interoperability. In G. Williams, P. Statham, N. Brown, B. Cleland (Eds.) Changing Demands, Changing Directions. Proceedings ascilite Hobart 2011 (pp.143–147).

[Doe11] Doelitzscher, F., Sulistio, A., Reich, C., Kuijs, H., & Wolf, D. (2011). Private cloud for collaboration and e-Learning services: from IaaS to SaaS. Computing, 91(1), 23–42.

[HKL14] Hafer, J., Kiy, A. & Lucke, U. (2014). Moodle & Co. Auf dem Weg zur Personal Learning Environment. e-learning & education (eleed), 10, p. 27.

[KLL17] Kiy, A, List, C. & Lucke, U. (2017). A Virtual Environment and Infrastructure to ensure future readiness of Data Centers. Eur. J. of Higher Education IT (EJHEIT), 2017-1.

[KLS15] Kiy, A., Lucke, U. & Sass, K., (2015). Gewusst was: Mit einer E-Learning-Toolbox die persönliche virtuelle Umgebung gestalten. In: Pongratz, H. & Keil, R. (Hrsg.), DeLFI 2015 – Die 13. E-Learning Fachtagung Informatik. Bonn: Köllen, p. 43–55.

[LCH14] Limoncelli, T. A., Chalup, S. R. & Hogan, C. J. (2014). The Practice of Cloud System Administration: DevOps and SRE Practices for Web Services, Volume 2, Addison-Wesley Professional.

[OT17] Ochoa X., Ternier S. (2017) Technical Learning Infrastructure, Interoperability and Standards. In: Duval E., Sharples M., Sutherland R. (eds) Technology Enhanced Learning. Springer, Cham.

[SHS15] Schlarb, M., Hundt, C., & Schmidt, B. (2015). Sauce: A web-based automated assessment tool for teaching parallel programming. In Hunold, S., Costan, A., Gimenez, D., Iosup, A., Ricci, L., Gomez Requena, M. E., Scarano, V., Varbanescu, A. L., Scott,

S. L., Lankes, S., Weidendorfer, J., and Alexander, M., editors, Euro-Par 2015: Parallel Processing Workshops, pp. 54–65, Cham. Springer International Publishing.

[St19] Stagge, J. H., Rosenberg, D. E., Abdallah, A. M., Akbar, H., Attallah, N. A. & James, R. (2019). Assessing data availability and research reproducibility in hydrology and water resources. Scientic Data, 6, 190030

[Ze17] Zedel, M. (2017). Konzeption und Implementierung einer Selfservice Container Farm. Master thesis, University of Potsdam.