FAIR Assessment of Cloud-based Experiments

Krishna Kamath*, Nicole Brewer †, Tanu Malik*
*School of Computing, DePaul University Chicago, IL USA
†Center for Biology and Society, Arizona State University, Tempe, AZ USA
kkamath@depaul.edu, nbrewer6@asu.edu, tanu.malik@depaul.edu

***Abstract*—Several computer science experiments require cloud infrastructure to produce results. Federally-funded cloud testbeds such as Chameleon and CloudLab aim to meet this need. A direct benefit of large-scale experimentation on these federally-funded cloud testbeds is the ease of reproducing experiments on the same hardware configuration originally used by an author. In this paper, we analyze over 100 shareable computer science experiments available on Chameleon, classifying them into different types: tutorials, research experiments, bug reproduction, and course assignments. We determine the packaging requirements for these various types of experiments and assess whether the resulting packages are repeatable on Chameleon and reusable on other public cloud infrastructures like AWS. Our findings reveal that several available experiments are contingent on obtaining leases, which result in significant lag time, thus affecting their 'push-button' reproducibility. Additionally, we find that packaging systems often overlook experimental files and include hardware configuration APIs that complicate reproducing these experiments on other public cloud infrastructures. Based on these findings, we offer recommendations for creating reusable packages.**

I. Introduction

Computer science experiments often need cloud infrastructure for development and execution to ensure scalability, flexibility, and accessibility. The experiments range from developing new operating systems, virtualization methods, performance variability studies, and power management research to projects in software defined networking, artificial intelligence, and resource management. Experiments of this type, often cannot be supported by HPC resources or submitting jobs to batch schedulers [14]. Requirements of such experiments often necessitates a reconfigurable bare-metal system giving users full control of the software stack including root privileges, kernel customization, console access, as well as the ability to experiment with software defined networking using innovative features [13].

Several federally-funded bare-metal infrastructures exist that serve as specialized cloud testbeds and provide researchers with direct access to physical hardware (bare metal) instead of virtualized environments. This access allows for more precise control over hardware configurations, making these infrastructures ideal for experiments that require specific hardware setups, performance testing, and systems research. For example, the NSF-funded Chameleon [14] is a large-scale, reconfigurable testbed that provides bare metal access to a variety of hardware including high-performance CPUs, GPUs, FPGAs, and storage resources. Researchers can reconfigure and experiment with different hardware setups and software stacks. Similarly, CloudLab [8] is an NSF-funded testbed that provides researchers with control over both the hardware and software of their experimental environments, and supports a wide range of research including cloud computing, big data, and the Internet of Things (IoT).

Experimentation on shared testbeds directly results in the creation of shareable digital artifacts, such as images, orchestration templates, datasets, tools, and notebooks. These artifacts typically represent either a complete experiment or a crucial part of one, and they can be used to replicate the experiment on the testbed where it was originally conducted. As the number of available artifacts grows, effective data management and stewardship of artifacts on a cloud infrastructure becomes increasingly important. The FAIR principles—Findable, Accessible, Interoperable, and Reusable—have recently been established to characterize and guide these practices [23]. In the context of experiments available on shared cloud testbeds, we believe that it is not enough for experiments to simply adhere to the FAIR principles; they must maintain continuous FAIRness [7]. Continuous FAIRness means that all data remains Findable, Accessible, Interoperable, and Reusable at all times.

To assess continuous FAIRness, we analyze 100 computer science experiments that were originally created and shared on Chameleon. We first classify experiments into different types: tutorials, research experiments, bug reproduction, and course assignments to better assess their continuous FAIRness. We also apply FAIRness criteria at the granularity of experiments. An experiment is considered findable if it is associated with a URI that provides access to all files related to the experiment. It is deemed accessible if there are no restrictions on repeating the experiment, particularly on Chameleon. An experiment is interoperable if metadata about its execution can be generated using lightweight application virtualization tools such as Sciunit or Reprozip. Lastly, an experiment is reusable if it can be repeated on similar hardware within commercial cloud infrastructures. To measure interoperability, we generate an application virtualization package, and to assess reusability, we determine whether the resulting package can be repeated on a public cloud infrastructure like AWS.

Our findings indicate that meeting all four properties of Findable, Accessible, Interoperable, and Reusable (FAIR) remains challenging for experiments shared publicly. While most experiments are findable, their accessibility is often limited, as many available experiments require obtaining leases, which can involve significant lag time and thus hinder 'push-button'

reproducibility. Additionally, packaging systems frequently overlook experimental files and include hardware configuration APIs, complicating the reproduction of these experiments on other public cloud infrastructures. Based on these observations, we provide recommendations for creating reusable packages.

The rest of the paper is organized as follows: We describe how experiments are organized and shared within Chameleon in Section II. We analyze available experiments into different types and determine their Findable and Accessible metrics in Section III-A. We determine Interoperable and Reusable metrics via lightweight packaging systems in Section III-A. Finally, we discuss how experiments can be packaged into reusable artifacts that can be replayed on other cloud infrastructures in Section V.

II. Chameleon Overview

Chameleon [14] is an experimental cloud testbed that enables computer science systems research. It allows bare-metal reconfigurability, and provides users with full control of software stack, which includes root privileges and kernel customization. Chameleon supports a wide variety of hardware configurations including FPGAs and a range of GPU technologies like Nvidia A100 tensor core and Nvidia Tesla K80, with over 550 nodes and 5 PB of storage [4].

Trovi [21] is a portal on Chameleon where users can share and replay experiments (also called artifacts). Experiments are bundled as Jupyter Notebooks, along with requisite data, files, and software dependencies. The Jupyter Notebook interface is used by both authors and reviewers to share, publish, and reproduce experiments. Currently, there are over 100 different public experiments published on Trovi, covering a wide array of topics from machine learning to database management systems, and class assignments to bug reproduction in open source codebases.

When a Chameleon user launches an experiment with Trovi, the underlying JupyterHub infrastructure spawns a copy of the experiment on a fresh Jupyter Notebook server. This server, intended to serve as a "head node," guides users through coordinating experiments run on other dedicated hardware. As such, the Jupyter server environment is intentionally limited to 1 CPU core and 1GB of memory.

A. Chameleon Artifacts

We classify shared experiments/artifacts available on Chameleon into four categories: tutorials, research experiments, bug reproduction, and course assignments. Tutorials either cover topics relevant to running experiments in the cloud or are specifically tailored to running experiments on the Chameleon platform. Research experiments range from reproducing a machine learning model and its results to network emulations and database research. Bug reproduction artifacts involve the user reproducing a specific bug, implementing the fix, and analyzing the improvements in runtime or memory management. Chameleon is also widely used in educational settings for instruction.

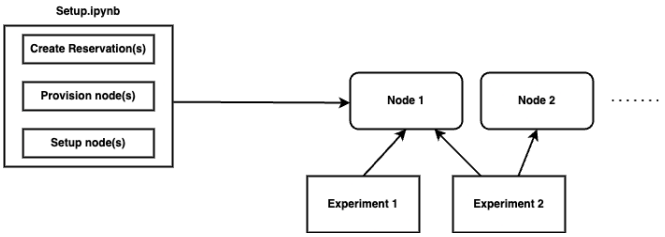

Fig. 1: Generic Chameleon Artifact Template

We analyzed 113 artifacts, added to Chameleon from April, 2022 to December, 2023. (As of August 2024, there are 173 artifacts on Chameleon.) Table I shows the number and percentage of artifacts that belong to each of the four categories.

Most Chameleon artifacts consist of one or more of the following steps (Figure 1): (i) system setup (ii) performing the experiment (iii) conducting analysis. The system setup involves the reservation of one or more specialized hardware that are available via the Chameleon cloud testbed and then provisioning the node(s) with the appropriate image(s) and connecting to the reserved node(s). The setup step also involves downloading the libraries and software required to conduct the experiment (see `Setup.ipnyb` in Figure 1). Once the setup step is complete, the user can connect to the specialized node (via ssh) and perform the experiment. The last step of the artifact template is analyzing the data obtained from the experiment. The availability of Jupyter Notebooks makes it easy to bring back analysis results and visualize via the notebook.

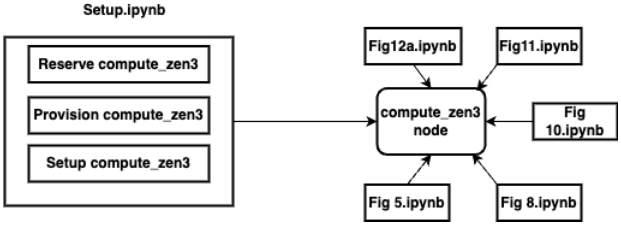

Fig. 2: SC MLEC23 Artifact

For example, the "SC23 MLEC Artifact" [17] reproduces the results of a paper presented in SC23, "Design Considerations and Analysis of Multi-Level Erasure Coding in Large-Scale Data Centers" [22]. This artifact consists of several notebooks, data, and scripts as shown in Figure 2. The first step is to set up the hardware and software via the setup.ipynb notebook, which leases and reserves a zen3 compute node for five days, and launches the hardware with an Ubuntu20.04 image (Figure 2). The setup further clones the simulator repository from GitHub, as well as necessary packages like *numpy* and *matplotlib* using Conda on the reserved node. The second step is to then run the experiment on the leased node, which executes for 8-12 hours. After the simulation experiment finishes, the user can run further analysis experiments and generate figures. One of the experiments computes burst tolerance for different MLEC schemes and repair methods using dynamic programming. In total, the experiment performs

five different kinds of analysis, each of which are described in a separate notebook. Given the significant execution time of the experiments, the authors have configured both the experiment and the analysis to run in the background, thus retaining notebook interactivity and allowing the reviewer to run any subsequent notebook cells.

While the majority of Chameleon artifacts adhere to this prototypical template, many educational artifacts do not adhere to this template. Such artifacts have a range of purposes: Some guide the user through creating and coordinating a particular kind of resource. Some contain examples of how to use packages developed by the artifact author. Others provide examples of post-experiment data analysis and visualization. Most educational notebooks highlight one or more steps in the process of running a complete experiment on Chameleon: (i) system setup (ii) performing the experiment (iii) conducting analysis. Next, we will discuss two specific examples of notebooks falling outside the standard experiment template to demonstrate the breadth of purposes of Chameleon artifacts.

In the "Machine Learning Process" artifact [15] - an introductory level courselet on Machine Learning Processing - goes through all the steps of a data life-cycle, including acquiring, exploring, processing, and analyzing data. The courselet concludes with a notebook that distinguishes between the model-development phase, as exemplified by the courselet, but does not include the production phase, which would require scaling with dedicated hardware or cloud infrastructure. In "Initializing a VM at KVM@TACC with multiple storage volumes" [12] an instructional Jupyter Notebook details how to reserve hardware, assign a floating IP address, and set up *a second, independent* Jupyter server on that dedicated hardware but does not contain any experiment or analysis.

Many Chameleon artifacts demonstrate that common practice: using the Jupyter workspace in Trovi to reserve a more powerful independent node running its own Jupyter server for conducting computationally-intensive experiments. In fact, the recursive nature of using Jupyter to manage other Jupyter servers is colloquially addressed in the Chameleon documentation as "Jupyter all the way down".

III. Artifact FAIRness

We found the definitions of FAIRness, as given by the authors in [23] limiting for reproducible analysis of artifacts. According to the original definition, artifacts are findable as long as they have persistent URI attached and the experiment metadata is indexable. However, findable artifacts may be incomplete, *i.e.,* some of the files are missing. Similarly other definitions are related to metadata and data and not to artifact execution. We think the FAIR terms are relevant to artifact reproducibility, but each term must be redefined to better suit the reproducibility requirements of artifacts as follows:

1) **Findable:** An artifact is findable if it is associated with a persistent URI that results in all the necessary files and data for a successful experiment run.

TABLE I: Chameleon Artifact Types

Type	Number of artifacts	Percentage
Tutorials	39	35%
Research Experiments	46	40%
Bug Reproduction	19	17%
Course Assignments	9	8%
Total	113	100%

2) **Accessible:** An artifact is accessible if the necessary found files result in a successful execution and produces some result.
3) **Interoperable:** An artifact is interoperable if its bundled package can be successfully executed to generate standardized metadata about its execution, such as W3C PROV [9]. In other words, both artifact file data and metadata are interoperable.
4) **Reusable:** An artifact is reusable if it can be successfully executed in an alternate environment, *i.e.,* an environment in which it was not natively found.

We would like to emphasize the subtle difference between the definitions of accessibility and reusability, especially in the context of cloud-related experiments. Accessible refers to whether the experiments can successfully run on the cloud infrastructure they are hosted on (Chameleon in our case). Reusable refers to whether the experiment can be run on different cloud infrastructures. Thus, in general, experiments may be reusable without being accessible. Finally, since the redefined terms are related to artifact execution and environments, they adhere to continuous FAIRness rather than snapshot FAIRness.

Chameleon artifacts are not badged. Thus, there is no prior confirmation or verification of an independent reviewer having run the artifact successfully. To analyze the reproducibility of artifacts, we applied the redefined FAIR criteria to the artifacts to determine and measure their findability, accessibility, interoperability, and reusability. We describe the findability and accessibility metrics in the next subsection and describe the interoperability and reusability metrics in the subsequent subsection.

A. Findability and Accessibility of Artifacts

Table II shows the results of FAIR metrics on Chameleon artifacts. As the Table shows about 92% of artifacts are findable, *i.e.,* all the files relevant to the stated experiment are present. In this case, the relevancy of files is determined via packaging tools as described further in Section IV. Amongst the 8% non-findable experiments, 7 had no files at all, and 2 had one or more missing files.

In terms of accessibility, we experimented with repeating each artifact on Chameleon. For repeatability, we applied the most relaxed definition—that all cells of the interactive notebook must run successfully, *i.e.,* without compilation or execution errors. In addition, we followed any instructions provided by the author. If there were multiple notebooks, we rant them in the order documented.

As shown in Table II, only 28.8%, or 30 out of 104 findable artifacts, were accessible. We further analyzed the reasons for the limited accessibility of the remaining 71.2%, or 74

TABLE II: Artifacts FAIRness

	Findable	Accessible	Interoperable	Reusable
Ratio of Artifacts	104/113	30/104	18/30	5/18
Percentage	92%	28.8%	60%	27.7%

TABLE III: Artifact Accessibility

Type	Accessible	Lease Issues	Code Issues	JupyterHub Issues
Tutorials	10	12	14	0
Research Experiments	10	13	14	5
Bug Reproduction	10	4	3	1
Course Assignments	0	7	1	0
Total	30	36	32	6

artifacts, as presented in Table III. The primary reason for non-accessibility was code-related issues that resulted in run-time errors. For example, "ATC/OSDI 23 Simple Filesystem Benchmark" [2] fails because setup_filebench.sh does not run. Similarly, "IGNITE-14003 reproduce" [11] stalls in the build process and doesn't complete. We attribute these issues to the author's lack of maintenance over time.

However, not all issues were author-related. About 41% of the issues stemmed from the inability to acquire timely leases for specialized and popular hardware such as GPUs, indicating heavy usage and contention for resources, and due to JupyterHub not spawning the notebook. We would like to emphasize that our results are based on several attempts to acquire leases over a five-week period. Often, for machine learning-based artifacts, the requested leases are for in-demand GPU resources and span several days. The long duration of these leases makes them non-accessible and non-repeatable. We did not attempt to reduce the lease time as we believe it is author-determined and would alter the experiment's duration.

Finally, it is worth noting that among the 28.8% of accessible artifacts, there were minor dependency issues in about 5 of them pertaining to discrepancies between the versions that were installed at runtime and the stated versions in the documentation. Such discrepancies were manually resolved. Thus, to ensure artifact reproducibility, reviewers should make minimal assumptions about packages installed on systems.

IV. Interoperability and Reusability of Artifacts

Application virtualization tools [5], [20] use system call interposition to determine files used during application execution and containerize them within a package or a container. Typically packages resulting from application virtualization are much smaller in size than the original resources provisioned. While the packages contain the necessary and sufficient files, they do not include the complete OS environment and need an instance to be executed. Most tools run in two modes: an audit mode to create a package/container, and a repeat mode to re-run a package/container [5], [20]. In AV audit mode, a container of a user application is created as the user executes the application (in the context of auditing, such an execution is termed a reference execution). In repeat mode, the application is executed from the container itself by monitoring its processes with *ptrace*, interrupting application system calls and extracting their path, and redirecting the calls to the files in the package/container. After the package is created, the contents are deduplicated to optimize space.

Artifact	Type	Sciunit	Reprozip
Data-Integration [6]	Tutorial	164MB	52 MB; 166MB
Selection-Brushing [18]	Tutorial	283MB	86.5MB;285MB
SPARK-25947 reproduce [19]	Bug Reproduction	272 MB	80MB;265MB
CA-15902 reproduce [3]	Bug Reproduction	271MB	85MB; 278MB
Machine Learning process [15]	Experiment	502MB	152MB;504MB
Alexnet(Updated) [1]	Experiment	89MB	28MB; 86MB

TABLE IV: Package sizes for Sciunit and Reprozip

To measure interoperability and reusability of artifacts, we packaged Chameleon artifacts. Since packaging requires the successful execution of artifacts, we attempted to package only 30 of the accessible artifacts. Chameleon's JupyteHub interface provides users with a bash terminal whenever a notebook of the corresponding experiment is launched. We installed the application virtualization tools using package managers such as *pip* using this terminal. The first step of packaging is to create the package by auditing the experiment execution. We determined how many of the accessible artifacts could be successfully packaged. Table II shows that only 18 of the 30 accessible artifacts could be packaged. We could not package all the 30 artifacts since we faced either leasing issues or packaging errors. For about 8 out of the 12 unpackaged artifacts, the resource was not available, so the experiment would fail. For the remaining artifacts, the application virtualization tool itself failed to successfully package, resulting in an overall 60% success rate. For example, the MLEC experiment highlighted in Section II is accessible but is not interoperable because the authors use a terminal multiplexer to run the long-running experiment in background. Thus it cannot be packaged by both Sciunit and ReproZip.

Further, we also measured the size of the package sizes created by the tools and compared them in the Table IV. The sizes of the packages created by both Sciunit and Reprozip are comparable, with a 2-3% difference on average. In general, this size is much smaller than the compute resources, which would inevitably have to be shared.

Finally, we measured the reusability of Chameleon artifacts. For this, we determined if the resulting packages were runnable on AWS cloud infrastructure. Both Sciunit and Reprozip packages can be shared on public cloud infrastructures by downloading the packaged container and repeating the experiment using `sciunit repeat` or `reprozip`. A primary motivation for determining reusability of packages was due to the delay in acquiring leases on the Chameleon cloud infrastructure.

Table II shows that only 5 of the successfully packaged 18 artifacts were reusable. The rest of the 13 packages include Chameleon-specific resource allocation commands, which make them non-reusable across commercial cloud infrastructures. For example, in the snippet provided below (taken from AlexNet(updated) [1]), node.run() is a Chameleon specific API. Using such commands prevents the reusability of the artifact on other cloud providers and requires code remapping. Since experiment details are intricately woven with Chameleon API, it is difficult to disentangle the experiment from the API and requires significant refactoring.

V. Best practices for ensuring continuous FAIRness of artifacts

Our analysis reveals that leasing high-demand resources on shared cloud infrastructure can cause significant delays. In extreme cases, where the required resource is always in high demand, the delay can extend to weeks or even months. Packaging experiments using application virtualization tools can be beneficial in such cases, especially if the user has access to similar hardware through public cloud infrastructure such as AWS. However, to repeat the experiment using Sciunit in "modified repeat" mode or with Reprozip, the user must currently edit the configuration and code files and replace cloud-specific API calls with code compliant with different cloud infrastructure.

To create more reusable packages, we have observed several ways authors can organize their experiments to achieve more FAIR (Findable, Accessible, Interoperable, and Reusable) experiments. Here are our recommendations:

1) **Organize Notebooks by Purpose.** Experiments should be broken down into multiple notebooks, with each notebook serving a single purpose. Notebooks that acquire and set up the resource should be clearly distinguished from those responsible for experiment execution. The SC23 MLEC artifact is an ideal example, as it contains multiple Jupyter Notebooks, each dedicated to a different sub-experiment and resource setup.
2) **Ensure Long-term Preservation.** Most open-source software decays without maintenance. This phenomena is called software collapse and it is especially prevalent in high-level languages with large numbers of contributors such as Python [10]. Authors should consider frequently repeating their published artifacts on Chameleon to ensure long-term reproduction. Such maintenance work is rarely incentivized [16]. However, we believe it is especially important for artifacts most likely to be reused for training or serve as templates for other experiments.
3) **Install Dependencies During Setup.** All dependencies required by an experiment must be installed during the setup step. The experiment must not make any assumptions about installing the same dependency when the head node is spawned as dependency packages, and their versions vary between different repeated runs.
4) **Automate Instructions.** Notebooks should **automate resource reservation and coordination steps at every stage of the experiment whenever feasible.** For example, some artifacts ask the user to create a reservation manually on the Chameleon platform and then run the notebooks. However, this isn't easily reproducible and should be avoided. Even notebooks that provide automated reservation steps commonly do not include automated instructions for tearing down and releasing reserved resources upon completion of the experiment. **We recommend including teardown commands**. Encouraging users to release resources upon completing their computational task may improve the overall availability of limited resources which, as we have found, is a significant contribution to non-repeatability on Chameleon. Finally, **we recommend using environment variables present in Chameleon workspaces to automate variable setting across users**. For example, many notebooks contain a line sets the project name using a hard-coded string (e.g. "CHI-231217"). As often addressed in a comment, the user must manually replace the project code with their own. Artifact authors can avoid that by using the "OS_PROJECT_NAME" environment variable available in Chameleon workspaces (See example in Figure 3). Similarly, authors should use the "USER" environment variable to prepend lease and resource names with the workspace owner's username. Such naming conventions, though not commonly used in Chameleon artifacts, avoid collisions with other users who are members of the same project, and it makes it easier to identify and clean up extraneous unused resources without accidentally deleting another project member's resources.
5) **Reduce Calls to Cloud-Specific APIs.** Cloud-specific APIs such as the Chameleon Python client, python-chi, make resource reservation and coordination of Chameleon artifacts easier read, execute, and reproduce. However, authors should keep cloud-related commands separate from experiment-related files as much as possible. For instance, an experiment may require executing a series of commands on remote hardware. We observe that many Chameleon artifacts implement such experiments by invoking the Chameleon-specific API for each individual bash command (See example in Figure 4). This approach requires anyone wishing to reproduce the experiment on another cloud platform to manually modify every line that invokes a command using a Chameleon-specific API. A better approach is to have a bash script containing all the installation commands and call the bash script using a single Chameleon-specific API invocation. This ensures minimal changes are required before the user can package the experiment and repeat it on a different cloud resource. Reviewers can then replace Chameleon-specific APIs and commands with analogous commands for commercial cloud providers if they exist.
6) **Indicate estimated wait times.** We recommend including

```
import os

# Automaticaly set region_name to user's default region
region_name = os.getenv("OS_REGION_NAME", "CHI@UC")

# Automatically set project_name to user's project
# Or user can update "CH-XXXXXX" manually as a backup
os_project_name = os.getenv("OS_PROJECT_NAME")
if not os_project_name:
    os_project_name = "CH-XXXXXX"

# Prepend lease name and other resources with user's
# name to avoid collisions
lease_name = f"{os.getenv('USER')}-power-management"
)
```

Fig. 3: Use environment variables present in Trovi's Jupyter interface to automate variable setting across user workspaces

```
from chi import ssh
with ssh.Remote(floating_ip) as node:
    node.run('pip install --upgrade pip')
    node.run('pip3 install tensorflow')
    node.run('pip install matplotlib')
    node.run('pip install tensorflow-datasets')
    node.run('pip install python-csv')
    node.run('pip install opencv-python')
    node.run('pip install Pillow')
    node.run('sudo apt-get install -y
    libsm6 libxext6 libxrender-dev')
    node.run('pip install --upgrade opencv-python')
    node.run('pip install --upgrade Pillow')
    node.run('pip install scikit-learn')
```

Fig. 4: Resource configuration using cloud-specific APIs in AlexNet(updated) Artifact [1]

estimated runtimes for tasks that take a long time to reduce the total time spent on behalf of the individual repeating the experiment. While the original experimenter may have a tacit understanding of how long each step takes, those repeating the experiment may be uncertain whether a process is progressing as expected or if an issue has arisen. Without this knowledge, they must wait for the experiment to time out or fail. We recommend including such indications at any stage in the experiment, but especially when running computationally-intensive steps.

VI. Conclusion

In this paper, we evaluated the reproducibility of artifacts available through the Chameleon platform across four key dimensions: findability, accessibility, interoperability, and reusability. We found that several artifacts were non-repeatable either due to lack of maintained artifacts or required long-term leases for in-demand hardware. We presented a detailed analysis of the repeatability of these artifacts and their ability to be packaged for reuse on other cloud infrastructures. Lastly, we provided several recommendations for creating reusable packages for the cloud.

VII. Acknowledgment

The authors would like to thank Kate Keahey and the Chameleon team for allowing us to run the artifacts on the platform, and discussing with us the issues related to artifact accessibility.

References

[1] AlexNet(updated) chameleon artifact https://chameleoncloud.org/experiment/share/8ddcf919-79d5-4e48-b095-a778ead90933.
[2] ATC/OSDI 23 Simple Filesystem Benchmark Chameleon Artifact https://chameleoncloud.org/experiment/share/b2328ffc-7208-42f8-9aad-21482c582b66.
[3] CA-15902 Reproduce https://chameleoncloud.org/experiment/share/097cf325-8835-4181-a4cd-36b36d5b745a.
[4] "Chameleon" https://chameleoncloud.org/about/chameleon/.
[5] Fernando Chirigati, Rémi Rampin, Dennis Shasha, and Juliana Freire. Reprozip: Computational reproducibility with ease. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 2085–2088. ACM, 2016.
[6] Data-Integration https://chameleoncloud.org/experiment/share/b1d3e4f1-bf88-4312-b841-dd51b237e5fd.
[7] William Dempsey, Ian Foster, Scott Fraser, and Carl Kesselman. Sharing begins at home: how continuous and ubiquitous fairness can enhance research productivity and data reuse. *Harvard data science review*, 4(3), 2022.
[8] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, et al. The design and operation of {CloudLab}. In *2019 USENIX annual technical conference (USENIX ATC 19)*, pages 1–14, 2019.
[9] Paul Groth, Luc Moreau, et al. Prov-dm: The prov data model. World Wide Web Consortium (W3C) Recommendation, April 2013. Accessed: 2024-08-25.
[10] Konrad Hinsen. Dealing with software collapse. *Computing in Science Engineering*, 21(3):104–108, 2019.
[11] IGNITE-14003 reproduce https://chameleoncloud.org/experiment/share/751f1dd6-f398-43ab-a6b1-1f4bb14616d2.
[12] Initializing a VM at KVM@TACC with multiple storage volumes https://chameleoncloud.org/experiment/share/2ea29f95-17f5-47a7-8681-71abbfeefc38.
[13] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S Gunawi, Cody Hammock, et al. Lessons learned from the chameleon testbed. In *2020 USENIX annual technical conference (USENIX ATC 20)*, pages 219–233, 2020.
[14] Kate Keahey, Joe Mambretti, Paul Ruth, and Dan Stanzione. Chameleon: a large-scale, deeply reconfigurable testbed for computer science research. In *2019 IEEE 27th International Conference on Network Protocols (ICNP)*, pages 1–2. IEEE, 2019.
[15] Machine Learning process https://chameleoncloud.org/experiment/share/27289213-f69f-43a3-97d9-8123822a47f1.
[16] Limor Peer, Lilla V. Orr, and Alexander Coppock. Active maintenance: A proposal for the long-term computational reproducibility of scientific results. *PS: Political Science 38; Politics*, 54(3):462–466, 2021.
[17] SC23 MLEC Artifact https://chameleoncloud.org/experiment/share/7a309c34-482b-4eac-b234-bbe4334830f2.
[18] Selection-Brushing https://chameleoncloud.org/experiment/share/548fd57f-62b2-4470-a25b-6384767ee191.
[19] SPARK-25947 reproduce https://chameleoncloud.org/experiment/share/61eb84b2-7d6e-45bb-82a2-1d77e2bcd6cd.
[20] Dai Hai Ton That, Gabriel Fils, Zhihao Yuan, and Tanu Malik. Sciunits: Reusable research objects. *arXiv preprint arXiv:1707.05731*, 2017.
[21] "Trovi" https://chameleoncloud.gitbook.io/trovi.
[22] Meng Wang, Jiajun Mao, Rajdeep Rana, John Bent, Serkay Olmez, Anjus George, Garrett Wilson Ransom, Jun Li, and Haryadi S Gunawi. Design considerations and analysis of multi-level erasure coding in large-scale data centers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13, 2023.
[23] Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. The fair guiding principles for scientific data management and stewardship. *Scientific data*, 3(1):1–9, 2016.