Beyond Pass/Fail: Evaluating Infrastructure Agents Across Layers, Lifecycle, and Risk

Yuan Gao
ygao355@wisc.edu
University of Wisconsin–Madison
Madison, Wisconsin, USA

Zeren Yang
zyang667@wisc.edu
University of Wisconsin–Madison
Madison, Wisconsin, USA

Junnan Li
jli2786@wisc.edu
University of Wisconsin–Madison
Madison, Wisconsin, USA

Shawn (Wanxiang) Zhong
shawn.zhong@wisc.edu
University of Wisconsin–Madison
Madison, Wisconsin, USA

Ahmed Dajani
adajani@iastate.edu
Iowa State University
Ames, Iowa, USA

Mai Zheng
mai@iastate.edu
Iowa State University
Ames, Iowa, USA

Andrea Arpaci-Dusseau
dusseau@cs.wisc.edu
University of Wisconsin–Madison
Madison, Wisconsin, USA

Remzi Arpaci-Dusseau
remzi@cs.wisc.edu
University of Wisconsin–Madison
Madison, Wisconsin, USA

Abstract

Managing modern computing infrastructure has become a steadily harder problem due to the ever-increasing complexity. Recent advances in AI agents create a timely opportunity to automate infrastructure management tasks, but it remains unclear how well such agents can handle real-world infrastructure complexity. We present InfraBench, a benchmark suite for evaluating AI agents on realistic infrastructure tasks across the full system stack and full operational lifecycle with fine-grained risk assessment. Preliminary experiments with seven agent–model configurations show that even the strongest agent cannot secure a full score across all tasks. The mean scores range from 56.1% to 90.8%, and per-check scoring reveals a general failure pattern: agents may routinely satisfy short-term objectives while leaving non-durable changes, broken distributed invariants, unsafe side effects, and uncleaned state behind.

1 Introduction

As computing infrastructures continue to grow in scale and complexity, managing them has become a steadily harder problem. Even initial deployment now requires navigating diverse configurations across heterogeneous environments, from on-prem clusters to cloud interactions. Beyond deployment, continuous maintenance introduces additional burdens (e.g., upgrades and patching [18, 25], failure handling [9, 12, 15, 36, 40], migration and backup [6, 16, 17, 24]), all of which must be handled without disrupting service. This rising complexity turns infrastructure management into a persistent, long-standing challenge [13, 14, 23, 36].

Recent advances in artificial intelligence (AI) agents [1, 5, 22, 30, 37, 38] create a timely opportunity to revisit the challenge. A key question is whether AI agents can meaningfully automate these infrastructure-level tasks, and if so, to what extent they can handle the complexity and variability seen in the real world. Answering this requires rigorous system setups and measurements, yet existing benchmarks for AI agents focus on relatively simple scenarios which cannot capture the full spectrum of infrastructure management [7, 20, 21, 26, 33, 37]. As summarized in Table 1, they are largely limited in system environments (e.g., container only [33]), infrastructure lifecycle (e.g., no deployment or decommissioning phases [7, 21, 33]), scale (e.g., single-node only [21, 26]), and often lack of risk assessments.

Benchmark	Sys. Environ.	Lifecycle	Scale	Risk Access.
SREGym [7]	Cloud-native	△	✓	△
ITBench [20]	K8s/RHEL	△	△	△
DevOps-Gym [33]	Project env.	–	△	–
SWE-Bench [21]	Repo/Docker	–	–	–
Terminal-Bench [26]	Container	△	–	–
InfraBench	***Full-Stack***	✓	✓	✓

Table 1. InfraBench vs. Others. *InfraBench supports full system stack, complete lifecycle, distributed setup, and risk assessment. ✓ first-class; △ partial; – limited.*

To bridge the gaps, we introduce InfraBench, a comprehensive benchmark suite for evaluating the capabilities of AI agents in infrastructure-related tasks. Different from existing efforts, InfraBench is designed with four main goals:

- **Full-Stack**. Practical infrastructures often involve many layers (e.g., bare-metal (BM) or virtual machines (VM), operating systems (OS), distributed storage [32, 35] and compute [2, 8, 31]) that cannot be ignored.
- **Full-Lifecycle**. Infrastructures live through multiple phases (e.g., deployment, runtime usage, maintenance, decommissioning), each with a set of unique operations and requirements.

The 3rd Workshop on Hot Topics in System Infrastructure (HotInfra'26), co-located with ISCA'26, June 28, 2026, Raleigh, NC, USA

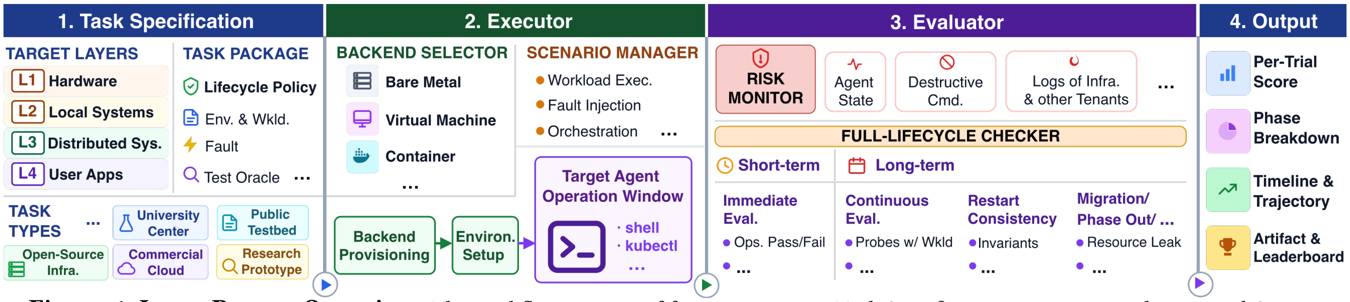

Figure 1. InfraBench Overview. *The workflow consists of four components: Task Specification, Executor, Evaluator, and Output.*

- **Risk-Aware.** Infrastructure tasks are fundamental and one simple error may cause cascading problems (i.e., blast radius issues [14, 15]), so accessing potential risks and side effects are necessary.
- **Realistic & Extensible**. Finally, we must reflect real-world scenarios (e.g., BM/VM clusters) to ensure high fidelity and practicality, and enable easy extension for the broad community.

To achieve the goals, we build InfraBench from four complementary sources: (1) semi-structured interviews with infrastructure providers and practitioners, including three university centers [27–29] and one cross-city testbed [19] at the time of writing, to elicit first-hand experiences on systems and operational constraints that are seldom captured in the literature; (2) open-source repositories and issue trackers of widely deployed infrastructure software (e.g., Slurm [39], Pelican [34], Ceph [35]); (3) documentations of commercial cloud platforms; (4) systems research prototypes that stress current designs. Triangulating across these sources lets us model a wide-spectrum of infrastructures and derive a general workflow to support systematic benchmarking across infrastructure layers and lifecycle with fine-grained risk monitoring and assessments (See §2).

We have implemented a preliminary prototype of InfraBench with twelve seed tasks, and evaluated seven agent–model configurations at the time of writing. The experimental results are promising: InfraBench shows that state-of-the-art (SOTA) agents often satisfy short-term checks while leaving operational obligations unresolved, which may cause negative impacts on the underlying infrastructures in the long term, including incomplete deployment state, non-durable changes, unsafe side effects, and missed cleanup requirements. We will release InfraBench as an open-source platform to facilitate infrastructure-level benchmarking of AI agents in the broad community.

2 InfraBench Design & Implementation

Figure 1 shows an overview of InfraBench. The general workflow consists of four components: (1) *Task Specification*, (2) *Executor*, (3) *Evaluator*, and (4) *Output*. It supports each benchmark instance as a controlled infrastructure operation trial, which may involve a variety of operations (e.g., configuration, recovery, migration, cleanup) across four layers:

- *L1 Hardware:* physical level operations (e.g., BMC/IPMI control, power cycling, RAID configuration);
- *L2 Local Systems:* host-level system software (e.g., OS, compiler, container runtime);
- *L3 Distributed Systems:* networked systems operating across nodes (e.g., Ceph [4], Slurm [39], Fabric [3]);
- *L4 User Applications:* user-facing applications or services running on local (L2) or distributed systems (L3).

By mapping tasks to layers (L1–L4), InfraBench provides first-class support in three dimensions: layer-aware backend selection, full-lifecycle evaluation, and operational risk monitoring. We elaborate on the main components below.

(1) Task Specification. Each trial begins with a task package that defines two types of information: (a) agent-visible instructions; (b) the hidden evaluation context, such as setup and bootstrap requirements of the target layer, workloads and fault conditions, oracles for verification, and lifecycle policies specifying which must be applied. Additional constraints based on infrastructure specifics (e.g., university center requirements) can also be added to improve coverage.

(2) Executor. This component instantiates the task on a faithful backend and exposes an operational interface to the target agent. There are two sub-modules: (1) *Backend Selector* maps each task to an execution layer and provisions resources (e.g., BM/VM nodes, Kubernetes clusters) to support task execution. (2) *Scenario Manager* configures the selected backend to an initial state based on policies (e.g., CPU/RAM limits, fault models and triggering conditions), and opens the agent operation window while keeping the evaluation context (e.g., validators and oracle scripts) transparent.

(3) Evaluator. This component evaluates the target agents in terms of both task completion and risks via two sub-modules:

Full-Lifecycle Checker separates short-term success from long-term operational correctness through four gates. *Immediate Evaluation* is the short-term gate: it compares the baseline and post-operation state after the agent operation window and checks whether the immediate objective was satisfied. The remaining gates provide long-term validation. E.g., *Live Evaluation* continues under sustained workload or periodic probes to detect configuration drift, delayed degradation, or loss of availability; *Restart/Durability* restarts the

relevant services or resources and checks post-restart invariants and persistent configuration; *Decommission* verifies that the the infrastructure can be restored to initial states, and/or requested resources are torn down cleanly with no leakage.

Risk Monitor. InfraBench treats operational risk and side effects as first-class evaluation signals alongside lifecycle correctness. The Monitor observes both agent actions and environment states, and records suspicious operations throughout the lifecycle (e.g., destructive commands, disabled safety checks, unnecessary privilege escalation, configuration drift, resource leaks, interference with unrelated services). These signals flag cases where an agent reaches an immediate objective by relying on unsafe shortcuts or leaves collateral damage that pass/fail checks would miss.

(4) Output. For each trial, InfraBench reports a per-trial score with phase-level breakdown, risk and side-effect records, execution timeline, trajectory artifacts, and leaderboard-ready summaries. The phase breakdown attributes failures to the responsible lifecycle stage or side-effect category, rather than collapsing them into a binary pass/fail result.

3 Experimental Results

We report preliminary experiments with InfraBench to understand where infrastructure agents fail, not only whether they complete a task. The current task set includes 12 seed tasks, spanning four infrastructure levels with characteristics summarized in Table 2. All trials run on the CloudLab Wisconsin testbed [10] on c220g1 nodes; depending on its backend, a task is provisioned as a single container, a multi-node VM cluster, or a set of bare-metal machines.

Table 3 summarizes the results on seven target agent/model configurations. Each task is scored by a task-specific verifier in $[0, 1]$. Overall, the mean scores range from 58.6% to 90.5%, suggesting that the benchmark is not saturated even by the strongest agent/model. The imperfect scores indicate that agents often satisfy visible objectives while still missing deeper operational obligations, such as durable state, distributed consistency, peer safety, and cleanup. We skip detailed analysis due to space limit, but highlight one general observation and one case study below.

General Failure Patterns Across Layers. Failures form a layer-wise gradient rather than a uniform pattern across layers. Per-check scoring exposes this pattern: agents often pass 60–90% of checks before stalling on one lifecycle obligation. More specifically, we observe the following: **L1** tasks are consistently solved, suggesting that out-of-band hardware controls are within reach; **L2** task failures usually come from non-durable changes: agents update the live runtime state but fail to persist the change across restart; **L3** accounts for most failures, as agents either stop after surface-level cluster checks pass or lose track of global ordering in long multi-step operations, leaving hidden quorum, replication, or configuration state inconsistent; **L4** failures are often functionally successful but operationally incomplete: agents make the data plane available but miss cleanup or drift obligations such as incident markers and stale configuration.

Task	Characteristics & Core Issues
ipmi-power-recovery	L1; Multi BM; power management
nic-split-brain	L2; Multi VM; local network stack misconf.
fileserver-raid10-silent-disk	L2; Multi VM; local storage stack degrad.
cassandra-dead-node	L3; Multi VM; failure handling (crash)
cassandra-hung-recovery	L3; Multi VM; failure handling (hung)
cords-propagation	L3; Multi VM; adapted from CORDS [11]
ceph-bootstrap	L3; Multi BM; dist. sys. deployment
ceph-pool-degraded	L3; Multi VM; dist. resource degradation
pelican-key-mismatch	L3; Multi VM; identify drift
slurm-puppet-config-drift	L3; Multi VM; scheduler config drift
db-wal-recovery	L4; Container; adapted from Terminal-Bench [26]
postgres-pgbouncer-drift	L4; Multi VM; long-term config drift

Table 2. Preliminary tasks in InfraBench.

Agent	Model	Mean Score
Claude Code	Claude Opus 4.7	**90.8%**
Codex	GPT-5.5	86.0%
Gemini CLI	Gemini 3.5 Flash	76.5%
OpenCode	DeepSeek V4 Flash	71.3%
Claude Code	Claude Sonnet 4.6	65.6%
OpenCode	MiMo V2.5 Pro	60.4%
OpenCode	DeepSeek V4 Pro	56.1%

Table 3. Mean score across all tasks by agent and model.

Case Study: Incident at A University Center. The Pelican [34] task is derived from a real incident at a high throughput computing (HTC) center. After an Origin host was rebuilt, its new identity drifted out of sync with the federation registry that authorizes it, blocking all client traffic. A correct trial must reconcile cross-component trust, restore the dependent cache and routing services, and close out the incident state. The task is graded by a 15-check verifier; no agent/model configuration can secure a perfect score, and the partial scores include three distinct failure patterns:

(1) Trusting a surface status signal. Some agents read a high-level "approved" indicator and stop, without checking the underlying key material that the indicator is supposed to summarize. They report success while the registry still holds the stale trust record, leaving the root cause untouched.

(2) Correct fix, destructive side effect. Other agents reconcile the trust mismatch correctly but, in doing so, render a peer node unreachable, so end-to-end retrieval still fails. The verifier credits the core repair and localizes the regression to the peer node, rather than scoring the trial as a non-fix.

(3) Functional fix, missing cleanup. The strongest configurations succeeded in 14 of 15 checks: they restore federation trust and end-to-end data retrieval, but leave behind the marker that signals an open incident to operators. The service is fully usable, yet by the standard of the infrastructure operator, the incident is not closed.

Overall, these failure patterns suggest that real infrastructure tasks often carry implicit obligations (e.g., durable state, intact peers, closed-out incident records) beyond "the service came back up". By providing first-class support in system layers, lifecycle, and operational risk accessment, InfraBench can address the benchmarking challenge effectively.

4 Discussions & Future Work

The work presented in this paper suggests many opportunities for follow up improvements. For example, we observe that the same agent may vary by more than 30 points across underlying models, which suggests the mean score metric may conflate capability with cost. We plan to add other metrics (e.g., score per token, per dollar, per wall-clock minute) to improve the fairness. Also, the current risk assessment is parallel: it records destructive commands and other side effects equally, even though a fix that takes down a peer node may be more harmful than one that performs the same fix safely. We plan to add ranking to address the problem. Finally, the task set in the current prototype is small and skewed toward L3. We plan to derive more tasks with collaborators and call for the collective efforts of the community to fully realize the potentials of InfraBench.

Acknowledgments

The authors thank the anonymous reviewers for their invaluable feedback. The authors also thank system administrators and practitioners at UW-Madison's Division of Information Technology (DoIT), Center for High Throughput Computing (CHTC), Computer Science Department IT (CIDS IT), and ISU's ARA Wireless Living Lab (arawireless.org) for sharing their real-world infrastructure management experiences. This work was supported in part by National Science Foundation (NSF) under grants #1943204, #2130889, #2402858, and #2402859. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of the sponsor.

References

[1] Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. 2023. Recommending Root-Cause and Mitigation Steps for Cloud Incidents using Large Language Models. In *Proceedings of the 45th International Conference on Software Engineering (ICSE)*. 1737–1749.

[2] Amazon Web Services. 2026. *AWS Lambda*. Amazon.com, Inc. https://aws.amazon.com/lambda/ Accessed: May 20, 2026.

[3] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. 2018. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*. 1–15.

[4] Ceph. [n. d.]. CEPHFS QUOTAS. https://docs.ceph.com/en/latest/cephfs/quota/. Accessed: June 5, 2025.

[5] Yinfang Chen, Manish Shetty, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Jonathan Mace, Chetan Bansal, Saravan Rajmohan, and Dongmei Zhang. 2025. AIOpsLab: A Holistic Framework to Evaluate AI Agents for Enabling Autonomous Clouds. In *Proceedings of Machine Learning and Systems (MLSys)*.

[6] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. 2005. Live migration of virtual machines. In *Proceedings of the 2nd Symposium on Networked Systems Design & Implementation (NSDI)*. 273–286.

[7] Jackson Clark, Yiming Su, Saad Mohammad Rafid Pial, Yifang Tian, Lily Gniedziejko, Hans-Arno Jacobsen, Yinfang Chen, and Tianyin Xu. 2026. SREGym: A Live Benchmark for AI SRE Agents with High-Fidelity Failure Scenarios. *arXiv preprint arXiv:2605.07161* (2026).

[8] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[9] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*. 1285–1298.

[10] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *Proceedings of the 2019 USENIX Annual Technical Conference (USENIX ATC '19)*. 1–14.

[11] Aishwarya Ganesan, Ramnatthan Alagappan, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau. 2017. Redundancy does not imply fault tolerance: Analysis of distributed storage reactions to file-system faults. *ACM Transactions on Storage (TOS)* 13, 3 (2017), 1–33.

[12] Om Rameshwar Gatla, Mai Zheng, Muhammad Hameed, Viacheslav Dubeyko, Adam Manzanares, Filip Blagojevic, Cyril Guyot, and Robert Mateescu. 2018. Towards robust file system checkers. *ACM Transactions on Storage (TOS)* 14, 4 (2018), 1–25.

[13] Haryadi S. Gunawi, Mingzhe Hao, Tanakorn Leesatapornwongsa, Tiratat Patana-anake, Thanh Do, Jeffry Adityatama, Kurnia J. Eliazar, Agung Laksono, Jeffrey F. Lukman, Vincentius Martin, and Anang D. Satria. 2014. What Bugs Live in the Cloud? A Study of 3000+ Issues in Cloud Systems. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*. 1–14.

[14] Haryadi S Gunawi, Mingzhe Hao, Riza O Suminto, Agung Laksono, Anang D Satria, Jeffry Adityatama, and Kurnia J Eliazar. 2016. Why does the cloud stop computing? lessons from hundreds of service outages. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*. 1–16.

[15] Runzhou Han, Om Rameshwar Gatla, Mai Zheng, Jinrui Cao, Di Zhang, Dong Dai, Yong Chen, and Jonathan Cook. 2022. A study of failure recovery and logging of high-performance parallel file systems. *ACM Transactions on Storage (TOS)* 18, 2 (2022), 1–44.

[16] Runzhou Han, Chao Shi, Tabassum Mahmud, Zeren Yang, Vladislav Esaulov, Lipeng Wan, Yong Chen, Jim Wayda, Matthew Wolf, and Mai Zheng. 2024. Revisiting erasure codes: A configuration perspective. In *Proceedings of the 16th ACM Workshop on Hot Topics in Storage and File Systems*. 93–100.

[17] Michael R. Hines and Kartik Gopalan. 2009. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE)*. 51–60.

[18] Petr Hosek and Cristian Cadar. 2013. Safe software updates via multi-version execution. In *2013 35th International Conference on Software Engineering (ICSE)*. 612–621.

[19] Taimoor Ul Islam, Joshua Ofori Boateng, Md Nadim, Guoying Zu, Mukaram Shahid, Xun Li, Tianyi Zhang, Salil Reddy, Wei Xu, Ataberk Atalar, et al. 2025. Design and implementation of ARA wireless living lab for rural broadband and applications. https://arawireless.org/. *Computer Networks* 263 (2025), 111188.

[20] Saurabh Jha, Rohan Arora, Yuji Watanabe, Takumi Yanagawa, Yinfang Chen, Jackson Clark, Bhavya Bhavya, et al. 2025. ITBench: Evaluating AI Agents across Diverse Real-World IT Automation Tasks. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*.

[21] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. SWE-bench: Can Language Models Resolve Real-World GitHub Issues?. In *The Twelfth International Conference on Learning Representations (ICLR)*.

[22] Pengxiang Jin, Shenglin Zhang, Minghua Ma, Haozhe Li, Yu Kang, Liqun Li, Yudong Liu, Bo Qiao, Chaoyun Zhang, Pu Zhao, Shilin He, Federica Sarro, Yingnong Dang, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 2023. Assess and Summarize: Improve Outage Understanding with Large Language Models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*.

[23] Chang Lou, Peng Huang, and Scott Smith. 2020. Understanding, Detecting and Localizing Partial Failures in Large System Software. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*. 559–574.

[24] Ali Mashtizadeh, Emré Celebi, Tal Garfinkel, and Min Cai. 2011. The Design and Evolution of Live Storage Migration in VMware ESX. In *2011 USENIX Annual Technical Conference (USENIX ATC)*.

[25] Ben Maurer. 2015. Fail at Scale: Reliability in the Face of Rapid Change. *Commun. ACM* 58, 11 (2015), 44–49.

[26] Mike A. Merrill, Alexander G. Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, et al. 2026. Terminal-Bench: Benchmarking Agents on Hard, Realistic Tasks in Command Line Interfaces. In *The Fourteenth International Conference on Learning Representations (ICLR)*.

[27] University of Wisconsin-Madison. [n. d.]. Center for High Throughput Computing (CHTC), UW-Madison. https://chtc.wisc.edu/. Accessed: June 15, 2026.

[28] University of Wisconsin-Madison. [n. d.]. Division of Information Technology (DoIT), UW-Madison. https://it.wisc.edu/. Accessed: June 15, 2026.

[29] University of Wisconsin-Madison. [n. d.]. IT of Computer Sciences Department at UW-Madison (CIDS-IT), UW-Madison. https://www.cs.wisc.edu/. Accessed: June 15, 2026.

[30] Manish Shetty, Yinfang Chen, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Xuchao Zhang, Jonathan Mace, Dax Vandevoorde, Pedro Las-Casas, Shachee Mishra Gupta, Suman Nath, Chetan Bansal, and Saravan Rajmohan. 2024. Building AI Agents for Autonomous Clouds: Challenges and Design Principles. *arXiv preprint arXiv:2407.12165* (2024).

[31] Chao Shi, Anthony Manschula, Tabassum Mahmud, Zeren Yang, Mai Zheng, Yong Chen, Jim Wayda, Matthew Wolf, and Byungwoo Bang. 2025. Revisiting Computational Storage for Data Integrity and Security. *arXiv preprint arXiv:2504.15293* (2025).

[32] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. Ieee, 1–10.

[33] Yuheng Tang, Kaijie Zhu, Bonan Ruan, Chuqi Zhang, Michael Yang, Hongwei Li, Suyue Guo, Tianneng Shi, Zekun Li, Christopher Kruegel, Giovanni Vigna, Dawn Song, William Yang Wang, Lun Wang, Yangruibo Ding, Zhenkai Liang, and Wenbo Guo. 2026. DevOps-Gym: Benchmarking AI Agents in Software DevOps Cycle. *arXiv preprint arXiv:2601.20882* (2026).

[34] The Pelican Platform Team. 2024. The Pelican Platform: A Data Federation Platform Powering the Open Science Data Federation. https://pelicanplatform.org/. Center for High Throughput Computing, University of Wisconsin–Madison and Morgridge Institute for Research.

[35] Sage Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. 2006. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Conference on Operating Systems Design and Implementation (OSDI'06)*. 307–320.

[36] Erci Xu, Mai Zheng, Feng Qin, Yikang Xu, and Jiesheng Wu. 2019. Lessons and actions: What we learned from 10k {SSD-Related} storage system failures. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 961–976.

[37] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[38] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations (ICLR)*.

[39] Andy B Yoo, Morris A Jette, and Mark Grondona. 2003. Slurm: Simple linux utility for resource management. In *Workshop on job scheduling strategies for parallel processing*. Springer, 44–60.

[40] Mai Zheng, Duo Zhang, and Ahmed Dajani. 2026. On fault tolerance of data storage systems: A holistic perspective. *Fault Tolerance in Modern Engineering Systems* (2026).