DOI: https://doi.org/10.15276/hait.07.2024.19
UDC 004.582

CIPerf: a benchmark for continuous integration services performance and cost analysis

Volodymyr I. Obrizan
ORCID: https://orcid.org/0000-0002-1835-4056; Volodymyr.obrizan@gmail.com
Kharkiv National University of Radio Electronics, 14, Nauki Ave. Kharkiv, 61166, Ukraine

ABSTRACT

Continuous Integration is a crucial practice in modern software development, enabling teams to automate the process of building, testing, and merging code increments to ensure continuous delivery of high-quality software. Despite its growing adoption, the cost and performance of Continuous Integration services often go unexamined in sufficient detail. This paper presents CIPerf, a comprehensive benchmark designed to analyze both the performance and cost of cloud-based and self-hosted Continuous Integration services. The study centers on a comparison between two specific services: Bitbucket Pipelines, a cloud-based offering by Atlassian, and Hetzner, a self-hosted solution. By focusing on these platforms, the research aims to provide practical insights into the real-world costs and execution performance of Continuous Integration services. To achieve this, CIPerf conducted automated tests on an hourly basis over a two-month period, measuring critical timeframes such as resource provisioning, environment setup, and the actual test execution times. The results showed significant differences in both the cost efficiency and the consistency of performance between the two services. For instance, Bitbucket Pipelines, while convenient in its cloud-based offering, demonstrated higher variability in provisioning times compared to the stable, predictable performance of Hetzner's self-hosted environment. The analysis also explored how these performance metrics influence key software development metrics, including deployment frequency and developer productivity. CIPerf provides a clear methodology for developers and organizations to objectively assess their Continuous Integration service options, ultimately helping them optimize their workflows. Moreover, this benchmark can serve as an ongoing tool to monitor service performance over time, identifying potential degradations or improvements in service quality, thus offering long-term value for teams that rely on Continuous Integration for their development processes.

Keywords: Continuous integration; performance Benchmark; Bitbucket Pipelines; service performance; DevOps metrics; developer experience; Lead Time; automated testing; test setup; networkX Benchmark

For citation: **Obrizan V. I. "CIPerf: a benchmark for continuous integration services performance and cost analysis".** ***Herald of Advanced Information Technology*****. 2024; Vol. 7 No.3: 275–283. DOI: https://doi.org/10.15276/hait.07.2024.19**

1. INTRODUCTION, FORMULATION OF THE PROBLEM

Continuous integration (CI) is a software development practice which requires running builds of separate software system components, integrating them into final deliverables and subsequently running static code analysis checks and automatic tests. It is a widely adopted practice: in 2016 70 % of the most popular projects on GitHub use CI [1].

There are several places to perform such continuous integration:

a) a local developer's computer;

b) computers in a corporate's data center (self-hosted, or on-premises continuous servers);

c) cloud-based continuous integration services (SaaS). Publicly available continuous integration SaaS are popular nowadays, notable: Atlassian Bitbucket Pipelines, GitHub Actions, GitLab, Amazon Web Services, CodePipeline, Microsoft Azure, Circle CI, Travis CI, and lots more.

There are several ways to reduce costs related to continuous integration. Algorithms reduce number of test runs:

a) batch testing algorithms [2, 18] group builds in batches, run one test suite per a group and bisects the group in case of a failure to find out a code commit which introduces the error;

b) test outcome prediction algorithms use machine-learning classifiers or similar algorithms to predict build failures and skip builds which will pass with high probability [3, 4], [5, 20].

Some authors consider the cost factor of CI as developers' efforts and time to set-up and maintain it [6, 19]. However, mentioned papers don't consider CI costs as a cost paid to cloud based CI SaaS for computing resources. Some authors report that CI is used to test performance of systems under test but not the performance of CI system itself [7, 8].

Place of CI in the DevOps cycle. DevOps Research and Assessment (DORA) group at Google has identified four key metrics that indicate the performance of a software development team [9]:

© Obrizan V., 2024

This is an open access article under the CC BY license (https://creativecommons.org/licenses/by/4.0/deed.uk)

Table 1. **Pros and Cons of different placement of CI jobs**

Deployemnt	Pros	Cons
Local computer	Quick turnaround, high performance, low cost	Computer is busy while running the CI job
Self-hosted	The local computer is free while running the CI job; Ability to configure hardware; Better security	Additional costs maintaining a server
Cloud-based provider	The local computer is free while running the CI job	Additional costs of renting a server; Low performance; Slow turnaround; Security issues

***Source:* compiled by the author**

a) Deployment Frequency – How often an organization successfully releases to production;

b) Lead Time for Changes – The amount of time it takes a commit to get into production;

c) Change Failure Rate – The percentage of deployments causing a failure in production;

d) Time to Restore Service How long it takes an organization to recover from a failure in production. They outline four grades of DevOps performers: Elite, High, Medium, and Low. For example, the Lead Time for Changes metric for Elite performers must be less than one hour. Thus, a CI job can’t take more than one hour to satisfy an Elite performers grade.

Role of CI service in developer experience (DevEx). DevEx defines how software engineers feel about, think about, and value their work. The report shows that such metrics as “Satisfaction with automated test speed and output”, “Satisfaction with time it takes to deploy a change to production”, “Time it takes to generate CI results”, “Deployment lead time (time it takes to get a change released to production)”.

Object of the research – cloud-based and self-hosted CI services. **Subject of the research** – CI service performance and costs to run tests. **The goal of the research** is to develop and apply a benchmark to analyze and compare the performance and cost efficiency of cloud-based and self-hosted continuous integration (CI) services, providing insights for developers and organizations to make data-driven decisions in selecting CI solutions.

2. OPERATION OF CONTINUOUS INTEGRATION SERVICES

The following section provides a detailed breakdown of how a typical CI service operates, from the initial code push to the final deployment of build artifacts.

1. A software developer submits a code increment by committing and pushing to a git repository.

2. A continuous integration service listens for code push events.

3. When the CI service gets a CODE_PUSH event it evaluates whether to initiate a build process based on predefined rules.

These rules may include:

a) Branch filters (e.g., only build for specific branches);

b) File path filters (e.g., ignore changes to documentation files);

c) Commit message filters (e.g., skip builds for minor changes);

d) Author filters (e.g., ignore commits from certain users); e) Time-based rules (e.g., limit build frequency).

4. Build initiation: If the commit passes the evaluation, the CI system initiates the build process. Otherwise, the commit is ignored for CI purposes.

5. Environment setup: A clean, isolated environment is prepared for the build and tests.

6. Code retrieval: The latest code is fetched from the repository.

7. Dependency resolution: Required libraries and dependencies are installed.

8. Compilation: The code is compiled (if necessary for the language used).

9. Unit testing: Automated unit tests are run to check individual components.

10. Integration testing: Broader tests are executed to ensure components work together.

11. Code quality checks: Static code analysis and linting tools may be run.

12. Artifact generation: Deployable artifacts (e.g., executables, containers) are created.

13. Reporting: Results of the build and tests are compiled and reported.

14. Notification: Developers are notified of the build status, especially if issues arise.

15. Artifact storage: Successful builds are stored for potential deployment.

3. COST AND PERFORMANCE OF CLOUD- AND SELF-HOSTED CI SERVICES

Comparison of a CI job execution cost using major CI service vendors.

Table 2. **Comparison of fees associated with different CI services providers**

Service	Job execution cost*
Bitbucket Pipelines (cloud) [10]	$0.010/min
GitLab (cloud) [11]	$0.010/min
GitHub Actions (cloud) [12]	$0.005/min
Hetzner (self-hosted) [13]	$0.001/min

* the lowest proposed hardware configuration pricing is presented.
Source: **compiled by the author**

To compare job execution cost with a self-hosted CI service we choose Hetzner datacenter and their EX44 dedicated server proposal (Intel® Core™ i5-13500, 64 GB RAM). It is priced as € 39.00/month (roughly $43.26 / month). We assumed 31 days × 24 hours = 744 hours in a single month, which gives us $43.26 / 744 / 60 ≈ 0.001 $/min.

Performance and exact specification of underlying hardware of cloud-based CI services is not clear and not advertised by the vendors. In most cases they advertise computer nodes by number of available cores (1, 2,…), by architecture (X86, ARM, AMD64), by available RAM. And it is sometimes unclear if the provided computing resources are dedicated (fully available to the client only) or shared (the same hardware resources are used by several CI service tenants).

4. THE CIPERF BENCHMARK

Main principles:

1. Independence. It is neither controller nor sponsored by any major CI service provider. CI service providers can't cherry-pick benchmark tests to highlight their service in good light and hide the worst sides of their service.

2. Open. It is open-sourced and easy to reproduce the benchmark results.

3. Actuality. It is run once per hour to observe CI service degradation or improvements over time.

4.1. The Benchmark organization

The benchmark project is stored as a public Bitbucket git repository [14]. Bitbucket Pipelines – Atalssian's CI/CD service is configured for this repository to be triggered on each code push. There is a clone of this repository on a standalone computer hosted at Hetzner [15]. Every hour an automatic script creates a small change into the local repository, commits the change and pushes the change to the Bitbucket-hosted repository. Bitbucket Pipelines listens for the changes and triggers a build automatically.

The configuration of a Bitbucket Pipeline is defined in a YAML file, typically named bitbucket-pipelines.yml, which resides in the root directory of a project's repository. This file specifies the steps to be executed during the CI process, including environment setup, dependency installation, and testing. In the example configuration for CIPerf, the pipeline is designed to run on a Python 3.10 environment and includes steps for installing necessary dependencies, such as PostgreSQL client tools and Python libraries. The script first updates the system's package manager, installs required software, and sets up a Python virtual environment. Following this, automated tests for the NetworkX library are run using the pytest framework, while performance metrics are recorded. The pipeline also connects to a PostgreSQL database to log benchmark results, such as the total execution time and performance test duration.

This setup allows Bitbucket Pipelines to automatically trigger tests upon code changes, ensuring that performance data is consistently captured and analyzed. Bitbucket-pipelines.yml is a configuration file for Bitbucket CI service [16]:

```
image: python:3.10

pipelines:
  default:
    - step:
        name: Test
        caches:
          - pip
        script:
          - started_at=$(date -uIseconds)
          - start_time=$(date +%s)
          - apt-get update
          - apt-get install -y time postgresql-
common
          - YES=yes
          - yes "" | /usr/share/postgresql-
common/pgdg/apt.postgresql.org.sh
          - apt install -y postgresql-client-16
          - python -m venv venv
          - source venv/bin/activate
          - pip install --upgrade pip
          - pip install -r requirements/default.txt
-r requirements/test.txt
          - pip install -e .
          - /usr/bin/time -ap -o benchmark.txt
pytest --pyargs networkx
          - user_time=$(grep 'user' benchmark.txt |
awk '{print int($2)}')
          - end_time=$(date +%s)
          - total_time=$(($end_time - $start_time))
        - psql -h $DBHOST -U $DBUSER -p $DBPORT -
d statistics -c "INSERT INTO public.runs
(total_sec, performance_test_sec, vendor,
benchmark_id, commit, started_at) VALUES
($total_time, $user_time, 'Bitbucket Pipelines',
'networkx1', '$BITBUCKET_COMMIT', '$started_at');"
        - "echo Total time, sec: $total_time.
Benchmark time, sec: $user_time."
```

This configuration includes two main parts:

1. Installation of needed dependencies (postgresql client, Python packages).
2. Automatic tests for the NetworkX library.

The benchmark records the following timestamps: *code_pushed, ci_job_started, dependencies_installed, test_completed* (Table 3).

Table 3. **Main events of the CI process**

Timestamp	Description
code_pushed	Recorded right after successful code push: the code repository accepted the source codes.
ci_task_ started	Recorded on execution the very first line of bitbucket-pipelines.yml script. It means the computer is provisioned for a task by continuous integration service.
dependencies_ installed	Recorded after all dependencies are installed (command line tools, Python libraries).
test_completed	Recorded after automatic tests are completed.

***Source:* compiled by the author**

The chain of events *code_pushed* → *ci_task_started* → *test_completed* is on the critical path for the “Lead Time for Changes” DORA metric and the “Time it takes to generate CI results” developer experience metric.

Meaning of selected time frames for the benchmark (Table 4):

The *provisioning* timeframe refers to the period between the moment the CI system detects a code push (or a trigger event) and when the actual job starts executing on the allocated computing resources. In other words, it measures the time required for the CI service to prepare the infrastructure needed to run the build and tests, which includes allocating or spinning up virtual machines, containers, or any other required resources.

During the provisioning phase, the CI system ensures that a clean and isolated environment is ready for the upcoming tasks. The length of the provisioning timeframe can vary depending on the CI provider, the underlying infrastructure, and the current load on shared cloud resources. For instance, cloud-based CI services often experience variability in provisioning times due to resource availability, while self-hosted CI services may have consistently shorter provisioning times since dedicated hardware is already available. The *provisioning* timeframe directly impacts the overall efficiency of the CI pipeline, especially for teams that rely on rapid feedback from their builds and tests.

Table 4. **Main timeframes of the CI process**

Timeframe	Formula	Description
provisioning	*ci_task_ started* − *code_pushed*	Time in seconds, needed for a CI provider to provision computing resources (a computer) to execute a CI job.
test_setup	*dependencies_ installed* − *ci_task_ started*	Time in seconds, needed for a computer provided by a CI service to install dependencies.
computing	*test_completed* − *dependencies_ installed*	Time in seconds, needed for a CI-service-provided computer to complete automatic tests for the NetworkX library.

***Source:* compiled by the author**

The *test_setup* timeframe represents the duration required to set up the testing environment in a Continuous Integration (CI) process. Specifically, it measures the time taken from the start of the CI job (after the computing resources are provisioned) to the point when all necessary dependencies are installed, and the environment is fully configured for testing. This stage includes actions like downloading necessary libraries, installing required software packages, setting up configurations, and preparing any other resources required to run the tests. In CI workflows, the *test_setup* phase is critical because any delays in setting up the environment can prolong the overall CI job, thereby increasing lead times and reducing efficiency.

The *computing* benchmark is a unit-test suite for NetworkX Python library [17]. It consists of functional tests for different graph algorithms. We consider this as a good choice for computing benchmark, because of the pure computational nature of graph algorithms, thus no I/O (disk, network) resources are exercised during the benchmark.

4.2. Benchmark results

Analysis of CIPerf benchmark running hourly on Bitbucket Pipelines starting 29.06.2024 to 08.09.2024, 1710 runs in total.

Benchmark statistics for Bitbucket Pipelines:

Table 5. **Benchmark statistics for Bitbucket Pipelines**

Timeframe	Min, sec	Mean, sec	p95, sec	p99, sec	Max, sec	Average, sec	Stddev
provisioning	12	21	35	54	1990	26	63
test_setup	23	38	64	92	163	41	13
computing	127	167	205	223	246	169	19

Source: **compiled by the author**

Statistics for a local personal computer:

Table 6. **Benchmark statistics for a local personal computer**

Timeframe	Min, sec	Mean, sec	p95, sec	p99, sec	Max, sec	Average, sec	Stddev
provisioning	0	0	0	0	0	0	0
test_setup	57	64	70	74	75	65	4
computing	111	120	134	134	134	121	8

Source: **compiled by the author**

Statistics for a self-hosted server (Hetzner):

Table 7. **Benchmark statistics for a self-hosted server**

Timeframe	Min, sec	Mean, sec	p95, sec	p99, sec	Max, sec	Average, sec	Stddev
provisioning	0	0	0	0	0	0	0
test_setup	27	29	31	31	31	29	1
computing	83	85	86	86	86	85	1

Source: **compiled by the author**

5. DISCUSSION AND TAKEAWAYS

The CIPerf benchmark highlights several important findings regarding the cost and performance of continuous integration (CI) services. The comparison between cloud-based (Bitbucket Pipelines) and self-hosted (Hetzner) solutions reveals clear distinctions in both pricing models and performance metrics, providing valuable insights for organizations looking to optimize their CI workflows.

1. Cost Efficiency: The cost analysis shows that self-hosted solutions like Hetzner are significantly more cost-effective than cloud-based services such as Bitbucket Pipelines, especially for long-running or frequent CI jobs. While Bitbucket Pipelines charges approximately $0.010 per minute of job execution, the self-hosted Hetzner service costs only around $0.001 per minute. This tenfold difference highlights the potential savings for organizations that are willing to manage their own infrastructure, particularly for projects with extensive CI usage.

2. Performance Variability: The performance of Bitbucket Pipelines exhibits greater variability compared to the self-hosted Hetzner solution. While Hetzner consistently delivers fast provisioning and computing times, Bitbucket shows a higher standard deviation in key timeframes, particularly in provisioning, where the p99 value reaches up to 1,990 seconds. This variability can lead to unpredictable delays in the CI process, which could hinder developer productivity, especially for teams that rely on rapid feedback from CI pipelines.

3. Provisioning Time: One of the most striking differences is in the provisioning time, where Hetzner performs significantly better with no delay in resource allocation, as it is a dedicated server. In contrast, Bitbucket Pipelines shows variability, with provisioning times ranging from 12 to 1.990 seconds. This indicates that shared cloud resources can introduce significant delays, particularly during peak usage periods, making it challenging to maintain a high level of CI performance.

4. Test Setup and Computing Time: Hetzner also outperforms Bitbucket Pipelines in test setup and computing times. While the difference in computing time (the actual test execution) is notable, the most significant gap is in test setup, where Bitbucket takes nearly 30% longer on average to install dependencies and prepare the environment. This overhead can be detrimental for CI pipelines that require frequent setup of complex environments.

5. Developer Experience and Lead Time for Changes: The unpredictability of provisioning and setup times in cloud-based solutions like Bitbucket can negatively impact developer experience (DevEx) and key DevOps metrics such as Lead Time for Changes. In contrast, the consistent performance of the Hetzner self-hosted server offers a more reliable and predictable CI experience, which can enhance

overall developer satisfaction and operational efficiency.

6. Scalability vs. Control: While cloud-based CI services offer ease of setup, scalability, and reduced infrastructure management overhead, they come at the cost of performance consistency and higher job execution fees. Self-hosted solutions, like Hetzner, provide better control, cost savings, and performance stability but require additional effort in managing hardware and software environments. Organizations must weigh these trade-offs based on their CI needs and operational constraints.

5.1. Takeaways

1. Cost Savings for Heavy CI Usage: Organizations with frequent or long-running CI jobs can achieve significant cost savings by opting for self-hosted solutions like Hetzner. However, the cost-benefit analysis should include the potential overhead of maintaining a self-hosted infrastructure.

2. Performance Stability: For teams that prioritize consistency and rapid feedback in their CI processes, self-hosted solutions may offer better reliability and predictability compared to cloud-based services, which can exhibit high variability due to shared resources.

3. Cloud-Based Services for Simplicity and Scalability: While cloud-based CI services like Bitbucket Pipelines introduce variability in performance, they are still appealing for smaller teams or projects that need fast scalability and minimal infrastructure management.

4. CI Monitoring: CIPerf can serve as a valuable tool for ongoing monitoring of CI service performance, helping teams detect potential service degradation over time. This makes it useful for both cloud and self-hosted environments, ensuring that CI processes remain optimized and responsive.

In conclusion, the CIPerf benchmark provides concrete data to guide organizations in selecting the most appropriate CI service based on their unique cost, performance, and management requirements. Future work could explore additional CI services to further expand the analysis, offering a more comprehensive view of the CI ecosystem.

6. CONCLUSIONS AND FUTURE WORK

This paper presents CIPerf, a benchmark designed to analyze and compare the cost and performance of cloud-based and self-hosted Continuous Integration (CI) services. Through an extensive study involving Bitbucket Pipelines (cloud) and Hetzner (self-hosted) over two months, the results demonstrate substantial differences in both cost efficiency and performance stability between these two options.

The scientific novelty of this research lies in the development of CIPerf, a unique, independent benchmark specifically designed to measure and compare both the performance and cost of cloud-based and self-hosted continuous integration (CI) services. Unlike previous studies, CIPerf provides a reproducible, open-source framework that evaluates the CI infrastructure itself, offering detailed insights into provisioning times, test setup durations, and computational efficiency, which have not been systematically analyzed in the context of CI service cost-performance trade-offs before.

The practical significance of this research is that it provides developers, teams, and organizations with a reliable tool (CIPerf) to objectively assess the performance and cost efficiency of continuous integration (CI) services. By offering concrete data on provisioning times, test setup, and execution costs, CIPerf helps decision-makers optimize their CI workflows, reduce operational expenses, and enhance developer productivity through informed selection of CI services, whether cloud-based or self-hosted. Additionally, it can be used to monitor performance degradation or improvements over time, ensuring consistent and efficient software development practices.

Key conclusions from this study include:

1. Cost-Performance Trade-offs: Self-hosted CI solutions, such as Hetzner, offer significantly lower job execution costs compared to cloud-based services like Bitbucket Pipelines. However, they require more operational oversight and infrastructure management, which may not be ideal for smaller teams or organizations prioritizing ease of use.

2. Performance Variability: Cloud-based services exhibit higher variability in provisioning and job execution times, potentially causing delays in the CI pipeline. In contrast, self-hosted solutions provide more consistent performance, especially in terms of provisioning and test setup times.

3. Developer Experience: For teams that prioritize rapid feedback in their CI processes, the performance stability of self-hosted solutions like Hetzner can enhance developer experience and reduce overall lead time for changes. On the other hand, cloud-based services offer convenience and scalability but may introduce unpredictable delays.

4. Benchmarking Utility: CIPerf proves to be a valuable tool for objectively measuring the performance of CI services. It provides a

reproducible framework that can be used to monitor CI service degradation or improvements over time, ensuring that organizations can optimize their CI workflows based on real data.

There are several avenues for future research that can build on the findings of this study:

1. Inclusion of More CI Services: Future work could expand the scope of CIPerf to include additional CI services such as GitHub Actions, Travis CI, and CircleCI. This would provide a broader comparative analysis across a wider range of cloud-based and self-hosted solutions, offering more comprehensive insights for organizations choosing CI tools.

2. Exploration of Different Workloads: The current benchmark focuses on a specific test suite (NetworkX). Future research could explore different types of workloads, including more I/O-intensive tests, larger codebases, or multi-language projects, to assess how CI services perform under varying conditions.

3. Real-World Application: While the current study runs automated tests in a controlled environment, future research could integrate CIPerf into real-world software development pipelines, analyzing how CI performance affects development cycles, release times, and developer productivity in diverse organizational contexts.

4. Cost-Benefit Analysis of Hybrid CI Models: Another potential area of exploration is the cost-benefit analysis of hybrid CI models, where organizations use a combination of cloud-based and self-hosted CI systems. This could provide insights into how teams can balance scalability, cost, and performance based on their specific needs.

REFERENCES

1. Hilton, M., Tunnell, T., Huang, K., Marinov, D. & Dig, D. “Usage, costs, and benefits of continuous integration in open-source projects”. *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. 2016. p. 426–437, https://www.scopus.com/inward/record.uri?eid=2-s2.0-84989159511&doi=10.1145%2f2970276.2970358&partnerID=40&md5=7e96113dc1efb43fb7b51e2d6ed14763. DOI: https://doi.org/10.1145/2970276.2970358.

2. Fallahzadeh, E., Bavand, A. H. & Rigby, P. C. “Accelerating continuous integration with parallel batch testing”. *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2023. p. 55–67. https://www.scopus.com/inward/record.uri?eid=2-s2.0-85180557197&doi=10.1145%2f3611643.3616255&partnerID=40&md5=bc128a0694c9acce1bb7d45ba94d4fce. DOI: https://doi.org/10.1145/3611643.3616255.

3. Jin, X. & Servant, F. “A cost-efficient approach to building in continuous integration”. *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering.* 2020. p. 13–25, https://www.scopus.com/inward/record.uri?eid=2-s2.0-85094315421&doi=10.1145%2f3377811.3380437&partnerID=40&md5=bb745a103bef9f097daf338a82b09882. DOI: https://doi.org/10.1145/3377811.3380437.

4. Jin, X. “Reducing cost in continuous integration with a collection of build selection approaches”. *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* 2021. p. 1650–1654, https://www.scopus.com/inward/record.uri?eid=2-s2.0-85116277591&doi=10.1145%2f3468264.3473103&partnerID=40&md5=d990ab64f1533ee41c7cd50853d356d1. DOI: https://doi.org/10.1145/3468264.3473103.

5. Hong, Y., Tantithamthavorn, C., Pasuksmit, J., Thongtanunam, P., Friedman, A., Zhao, X. & Krasikov, A. “Practitioners’ challenges and perceptions of CI Build Failure Predictions at Atlassian”. *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering.* 2024. p. 370–381, https://www.scopus.com/inward/record.uri?eid=2-s2.0-85197077100&doi=10.1145%2f3663529.3663856&partnerID=40&md5=04601519ed25be40a18966c3ce10e2ff. DOI: https://doi.org/10.1145/3663529.3663856.

6. Hilton, M., Tunnell, T., Huang, K., Marinov, D. & Dig, D. “Usage, costs, and benefits of continuous integration in open-source projects”. *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering.* 2016. p. 426–437, https://www.scopus.com/inward/record.uri?eid=2-s2.0-84989159511&doi=10.1145%2f2970276.2970358&partnerID=40&md5=7e96113dc1efb43fb7b51e2d6ed14763. DOI: https://doi.org/10.1145/2970276.2970358.

7. Yu, L., Alégroth, E., Chatzipetrou, P. & Gorschek, T. "A Roadmap for Using Continuous Integration Environments". *Communications of the ACM.* 2024. p. 82–90, https://www.scopus.com/inward/record.uri?eid=2-s2.0-85194381501&doi=10.1145%2f3631519&partnerID=40&md5=7476310c4de33d8a5d431fa2b166b79d. DOI: https://doi.org/10.1145/3631519.

8. Melone, C. & Jones, S. "Verifying functionality and performance of HPC applications with continuous integration". *Practice and Experience in Advanced Research Computing 2023: Computing for the Common Good.* 2023. p. 460–462, https://www.scopus.com/inward/record.uri?eid=2-s2.0-85176236613&doi=10.1145%2f3569951.3597557&partnerID=40&md5=8ef1de2e309c9b29543c562fcf756b10. DOI: https://doi.org/10.1145/3569951.3597557.

9. "Google Cloud. 2022. Using the Four Keys to Measure Your DevOps Performance". *Google Cloud Blog*. 2024. – Available from: https://cloud.google.com/blog/products/devops-sre/using-the-four-keys-to-measure-your-devops-performance.

10. "Atlassian. 2024. Bitbucket Pricing". *Atlassian*. 2024. – Available from: https://www.atlassian.com/software/bitbucket/pricing.

11. "GitLab. 2024. GitLab Pricing". *GitLab*. 2024. – Available from: https://about.gitlab.com/pricing.

12. "GitHub. 2024. About Billing for GitHub Actions". *GitHub Docs*. 2024. – Available from: https://docs.github.com/en/billing/managing-billing-for-github-actions/about-billing-for-github-actions.

13. "Hetzner. 2024. Dedicated Root Server EX Line". *Hetzner*. 2024. – Available from: https://www.hetzner.com/dedicated-rootserver/matrix-ex.

14. "1irs. 2024. NetworkX Benchmark". *Bitbucket*. 2024. – Available from: https://bitbucket.org/1irs/networkx_benchmark.

15. "Hetzner. 2024. Hetzner Homepage". *Hetzner*. 2024. – Available from: https://www.hetzner.com.

16. "Atlassian. 2024. Bitbucket Pipelines Configuration Reference". *Atlassian Support*. 2024. – Available from: https://support.atlassian.com/bitbucket-cloud/docs/bitbucket-pipelines-configuration-reference/

17. Hagberg, A. A., Schult, D. A. & Swart, P. J. "Exploring network structure, dynamics, and function using NetworkX". *Proceedings of the 7th Python in Science Conference (SciPy2008)*. 2008. p. 11–15. DOI: https://doi.org/10.25080/TCWV9851

18. Kamath, D. M., Fernandes, E., Adams, B. & Hassan, A. E. "On combining commit grouping and build skip prediction to reduce redundant continuous integration activity". *Empirical Software Engineering,* 2024; 29: 6, https://www.scopus.com/inward/record.uri?eid=2-s2.0-85202786343&doi=10.1007%2fs10664-024-10477-1&partnerID=40&md5=d51af9fe4e85c1d918c2bf05d04a3512.
DOI: https://doi.org/10.1007/s10664-024-10477-1.

19. Zheng, S., Adams, B. & Hassan, A. E. "Does using Bazel help speed up continuous integration builds?". *Empirical Software Engineering,* 2024; 29: 5, https://www.scopus.com/inward/record.uri?eid=2-s2.0-85199025885&doi=10.1007%2fs10664-024-10497-x&partnerID=40&md5=0a8c90067f577d393ce45d2273c28a79.
DOI: https://doi.org/10.1007/s10664-024-10497-x.

20. Lan, W., Zhang, J., Yang, H. & Cui, Z. "A directed greybox fuzzing tool for continuous integration". *SoftwareX*. 2024; 27. Scopus: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85199041782&doi=10.1016%2fj.softx.2024.101824&partnerID=40&md5=95f2c7d23fc81fe20e6260e4a6712e56. DOI: https://doi.org/10.1016/j.softx.2024.101824.

Conflicts of Interest: the authors declare no conflict of interest

Received 26.06.2024
Received after revision 30.08.2024
Accepted 16.09.2024

DOI: https://doi.org/10.15276/hait.07.2024.19
УДК 004.582

CIPerf: Бенчмарк для аналізу продуктивності та вартості сервісів безперервної інтеграції

Обрізан Володимир Ігорович[1)]
ORCID: https://orcid.org/0000-0002-1835-4056; Volodymyr.obrizan@gmail.com
[1)] Харківський національний університет радіоелектроніки, проспект Науки, 14. Харків, Україна

АНОТАЦІЯ

Безперервна інтеграція є важливою практикою в сучасній розробці програмного забезпечення, що дозволяє командам автоматизувати процес збирання, тестування та злиття кодових змін, забезпечуючи безперервну доставку високоякісного програмного забезпечення. Незважаючи на її зростаюче впровадження, вартість та продуктивність сервісів безперервної інтеграції часто залишаються недостатньо вивченими. У цій статті представлено CIPerf — комплексний бенчмарк, розроблений для аналізу як продуктивності, так і вартості хмарних та локальних сервісів безперервної інтеграції. Дослідження зосереджене на порівнянні двох конкретних сервісів: Bitbucket Pipelines, хмарного сервісу від Atlassian, та Hetzner, локального рішення. Зосереджуючись на цих платформах, дослідження має на меті надати практичні висновки щодо реальних витрат і продуктивності виконання завдань у сервісах безперервної інтеграції. Для досягнення цієї мети CIPerf проводив автоматизовані тести щогодини протягом двомісячного періоду, вимірюючи ключові часові інтервали, такі як виділення ресурсів, налаштування середовища та фактичний час виконання тестів. Результати показали суттєві відмінності як у вартості, так і в стабільності продуктивності між двома сервісами. Наприклад, Bitbucket Pipelines, незважаючи на зручність хмарного сервісу, демонстрував більшу варіативність часу виділення ресурсів порівняно зі стабільною, передбачуваною продуктивністю локального середовища Hetzner. Аналіз також досліджував, як ці показники продуктивності впливають на ключові метрики розробки програмного забезпечення, включаючи частоту розгортання та продуктивність розробників. CIPerf пропонує чітку методологію для розробників та організацій, яка дозволяє об'єктивно оцінювати варіанти сервісів безперервної інтеграції, що в кінцевому підсумку допомагає оптимізувати їхні робочі процеси. Крім того, цей бенчмарк може служити постійним інструментом для моніторингу продуктивності сервісів з часом, виявляючи потенційне погіршення або покращення якості сервісу, надаючи таким чином довгострокову цінність для команд, що залежать від безперервної інтеграції у своїх процесах розробки.

Ключові слова: безперервна інтеграція; Бенчмарк продуктивності; Bitbucket Pipelines; продуктивність сервісу; метрики DevOps; досвід розробників; час виконання змін; автоматизоване тестування; час налаштування тестів; Бенчмарк NetworkX

ABOUT THE AUTHOR

Volodymyr I. Obrizan - Doctoral student at Design Automation Department, Kharkiv National University of Radio Electronics, 14 Nauki Avenue, Kharkiv, 61166, Ukraine
ORCID: https://orcid.org/0000-0002-1835-4056; Volodymyr.obrizan@gmail.com;
Research field: Computer systems and networks

Обрізан Володимир Ігорович - докторант каф. Автоматизації проектування обчислювальної техніки. Харківський національний університет радіоелектроніки, проспект Науки, 14, Харків, Україна.