**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**
**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА»**
WIN-WIN EDIH



Co-funded by
the European Union



Співфінансується
Європейським Союзом




# ВІЛЬНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ В ОСВІТІ, НАУЦІ ТА БІЗНЕСІ

FREE/OPEN SOURCE SOFTWARE IN EDUCATION, SCIENCE & BUSINESS (FOSS 2026)

Матеріали II Міжнародної науково-практичної конференції
01–02 жовтня 2026 р.
м. Чернігів

**Чернігів 2026**

УДК 004.49:[37+001+334.722]
В41

*Рекомендовано до друку вченою радою Національного університету «Чернігівська політехніка» (протокол № 12 від 02.11.2026 р.)*

Вільне програмне забезпечення в освіті, науці та бізнесі : матеріали ІІ Міжнар. В41 наук.-практ. конф. (м. Чернігів, 01–02 жовтня 2026 р.) / відп. за вип. Д. Б. Мехед. Чернігів : НУ «Чернігівська політехніка», 2026. 156 с.

ISBN 978-617-8584-09-2

Збірник містить матеріали ІІ Міжнародної науково-практичної конференції «Вільне програмне забезпечення в освіті, науці та бізнесі» (FOSS 2026), що відбулася онлайн 01–02 жовтня 2026 року в Національному університеті «Чернігівська політехніка». Розглянуто питання використання вільного та відкритого програмного забезпечення в освіті, науці, бізнесі, ІТ-інфраструктурі, кібербезпеці, аналізі даних і штучному інтелекті. Матеріали подано українською та англійською мовами.

Для науковців, викладачів, здобувачів освіти, фахівців ІТ-галузі та всіх, хто цікавиться вільним програмним забезпеченням.

*Матеріали подано в авторській редакції. Відповідальність за зміст, достовірність фактів, цитат і посилань несуть автори.*

*Конференцію проведено в межах проєкту WIN-WIN EDIH (грантова угода № 101191647), співфінансованого Європейським Союзом у межах програми Digital Europe. Висловлені погляди та думки належать виключно автору(ам) і не обов'язково відображають позицію Європейського Союзу чи Європейської Комісії.*

Відповідальний за випуск: Мехед Дмитро Борисович

ISBN 978-617-8584-09-2

© НУ «Чернігівська політехніка», 2026

**Чопорова О. В., Чопоров С. В.** ВИКОРИСТАННЯ 3D-МОДЕЛЮВАННЯ ТА АЛГОРИТМІВ НЕЙРОННИХ МЕРЕЖ В НАВЧАЛЬНИХ ПРОЄКТАХ ІНЖЕНЕРНО-ТЕХНІЧНОГО СПРЯМУВАННЯ / USE OF 3D MODELING AND NEURAL NETWORK ALGORITHMS IN EDUCATIONAL PROJECTS OF ENGINEERING AND TECHNICAL DIRECTION ..... 82

**Чуб М. В.** ВИЯВЛЕННЯ ТА ПРОТИДІЯ АДАПТИВНИМ ВЕБ-АТАКАМ ЗА ДОПОМОГОЮ МОДЕЛЕЙ ГЛИБОКОГО НАВЧАННЯ ..... 86

**Шевченко Ю. О.** ПОРІВНЯЛЬНЕ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ POSTGRESQL ТА INFLUXDB ДЛЯ ЗБЕРІГАННЯ ДАНИХ ФІЗИЧНОЇ АКТИВНОСТІ / COMPARATIVE STUDY OF POSTGRESQL AND INFLUXDB PERFORMANCE FOR STORING PHYSICAL ACTIVITY DATA ..... 89

**Al-Akwaa K. H. M. A.** OPEN-SOURCE DATA SCIENCE FOR SUSTAINABLE DEVELOPMENT: A REPRODUCIBLE ANALYTICAL FRAMEWORK ..... 93

**Borysenko O. A.** AN ARCHITECTURAL APPROACH TO LLM-BASED ROOT CAUSE ANALYSIS FOR CONTAINER ORCHESTRATION INCIDENTS ..... 100

**Burmaka I.** LOCAL, OPEN, AND ENERGY-EFFICIENT: HARNESSING X86 NPUS FOR EVERYDAY AI TASKS ..... 102

**Byvoino T. P., Byvoino P. G.** SIMULATION MODELING AND STATISTICAL ANALYSIS OF THE EFFICIENCY OF AN EMERGENCY MEDICAL SERVICE STATION ..... 106

**Didusenko O. A., Bula Y. I., Shymchuk G. V.** REPRODUCIBLE FOSS FRAMEWORK FOR RESILIENCE EVALUATION OF CONTAINERIZED DISTRIBUTED SYSTEMS ..... 109

**Didusenko O. A., Bula Y. I., Shymchuk G. V.** REPRODUCIBLE FOSS TESTBED FOR FAULT-TOLERANT MULTIPATH ROUTING IN IP NETWORKS ..... 112

**Ebirim V., Jadama A. F. R.** RESILIENT DIGITAL EDUCATION IN DISRUPTED ENVIRONMENTS: THE CCDS APPROACH ..... 115

**Fraya** DIGITAL TRANSFORMATION OF BUSINESS THROUGH FREE AND OPEN SOURCE SOFTWARE: OPPORTUNITIES, BARRIERS, AND STRATEGIC PATHWAYS ..... 119

**Kala P.** RUN MONGODB YOUR WAY: A PRACTICAL OPEN-SOURCE STACK FOR SELF-MANAGED DEPLOYMENTS ..... 121

**Lukin O.** A MODERN C++ KMIP CLIENT BASED ON A REUSABLE PROTOCOL LIBRARY ..... 124

**Mekhed D. B.** DEVELOPING STUDENTS' CYBER HYGIENE SKILLS THROUGH THE USE OF FREE SPECIALIZED SOFTWARE ..... 127

**Nahornyi B. I., Lysenok D. V., Savenko A. O.** AN OPEN BROWSER-BASED CO-SIMULATOR OF STM32 FIRMWARE AND ELECTRICAL CIRCUITS AS A FREE ALTERNATIVE TO PROTEUS ..... 130

**Romanenko M. V.** FOSS APPROACH IN MOBILE DEVELOPMENT: CREATING AN OPEN-SOURCE ANDROID APPLICATION WITH THE INTEGRATION OF READY-MADE LIBRARIES FOR STYLISTIC RECOMMENDATIONS ..... 134

**Shevchenko N. A., Kibitkin D. S., Shymchuk G. V.** REPRODUCIBLE OPEN-SOURCE BENCHMARK OF CONCEPT-DRIFT DETECTORS FOR STREAMING REGRESSION ..... 137

**Shevchenko N. A., Kibitkin D. S., Shymchuk G. V.** INTERPRETABLE OPEN-SOURCE MACHINE LEARNING FOR NATURAL GAS CONSUMPTION FORECASTING WITH WEATHER AND LAG FEATURES ..... 140

**Shivashankar C.** ARCHITECTING DATABASE DEPLOYMENTS ON KUBERNETES: PATTERNS, TRADE-OFFS AND EXPERIENCES ..... 143

**Shymchuk G. V., Yatskiv P. O.** DETECTION OF ANOMALOUS STATES IN IP NETWORKS USING MACHINE LEARNING AND OPEN-SOURCE SOFTWARE TOOLS ..... 146

**Shymchuk G. V., Yatskiv P. O.** OPEN-SOURCE SOFTWARE ENVIRONMENT FOR MODELING FAULT-TOLERANT MULTIPATH ROUTING IN IP NETWORKS ..... 149

**Silva F.** HOSTING SOMEONE ELSE'S APP: IFRAME CHOREOGRAPHY IN PMM ..... 152

**Šupić A.** ZERO-DOWNTIME MIGRATION WITH PERCONA CLUSTERSYNC FOR MONGODB ..... 154

**Didusenko O. A., Bula Y. I., Shymchuk G. V.**
*Ternopil Ivan Puluj National Technical University, Department of Computer Science, Ternopil, Ukraine*
*Corresponding author: Grigorii Shymchuk, e-mail: gorych@gmail.com, ORCID: 0000-0003-2362-7386*

# REPRODUCIBLE FOSS FRAMEWORK FOR RESILIENCE EVALUATION OF CONTAINERIZED DISTRIBUTED SYSTEMS

*Abstract. A reproducible open-source framework for evaluating the resilience of containerized distributed systems is proposed. The framework combines k3s/Kubernetes, Chaos Mesh, Prometheus, and Python and focuses on failures that are common in distributed services: pod termination, node unavailability, increased network latency, and packet loss. Four deployment configurations were defined, from a single replica to three replicas with health probes, anti-affinity, and a PodDisruptionBudget. A pilot stochastic workload model with 30 repetitions per configuration was used to validate the metric pipeline before cluster experiments. Median modeled recovery time decreased from 4.54 s for a single replica to 0.48 s for the most resilient configuration; mean error rate decreased from 7.80% to 0.90%, while availability increased from 92.00% to 98.55%. The work provides a compact FOSS methodology for reproducible resilience experiments and clearly separates preliminary model-based results from future execution in a real k3s cluster.*

*Keywords: open-source software; Kubernetes; k3s; distributed systems; resilience; chaos engineering; fault injection; microservices.*

## Introduction

Container orchestration has become a standard mechanism for deploying distributed applications, but replication alone does not guarantee service resilience. Failures may occur at several layers: a process can become unresponsive, a pod can be terminated, a worker node can disappear, or network communication can experience latency and loss. Kubernetes provides readiness and liveness probes to remove unready pods from service traffic and restart failed containers, while ReplicaSets maintain a desired number of replicas [1]. These mechanisms create a practical basis for controlled resilience experiments.

Recent research treats fault injection and chaos engineering as systematic techniques for evaluating microservice robustness. Singh et al. assessed Kubernetes microservices under chaos and measured response time, errors, resource use, and request throughput [2]. Wang et al. proposed Kubernetes architecture graphs for selecting resilience experiments rather than relying only on random single-point faults [3]. A 2025 systematic review identified fault-injection configuration, execution, test oracles, and system-under-test selection as core design decisions for microservice testing [4]. Digital-twin approaches have also been used to reduce the cost of testing Kubernetes applications under computing and network faults [5].

The objective of this work is to define a small, fully open-source framework that can be repeated in university laboratories and used as a baseline for further research. The proposed implementation uses k3s as a lightweight conformant Kubernetes distribution [6], Chaos Mesh for fault injection [7], Prometheus-compatible metrics, and Python for experiment orchestration and analysis. A pilot model is used only to verify the metric definitions and expected direction of effects before running the framework on real nodes.

## Materials and Methods

The target testbed consists of one k3s server and three worker nodes. A simple HTTP microservice is deployed behind a Kubernetes Service, while a load generator sends requests at a constant target rate. Prometheus records request counters, latency, pod state, and node state. Chaos Mesh is responsible for controlled pod failures and network disturbances. The logical workflow is shown in Figure 1. The framework is designed so that manifests, failure schedules, and analysis scripts can be stored in a repository and replayed without commercial cloud services.

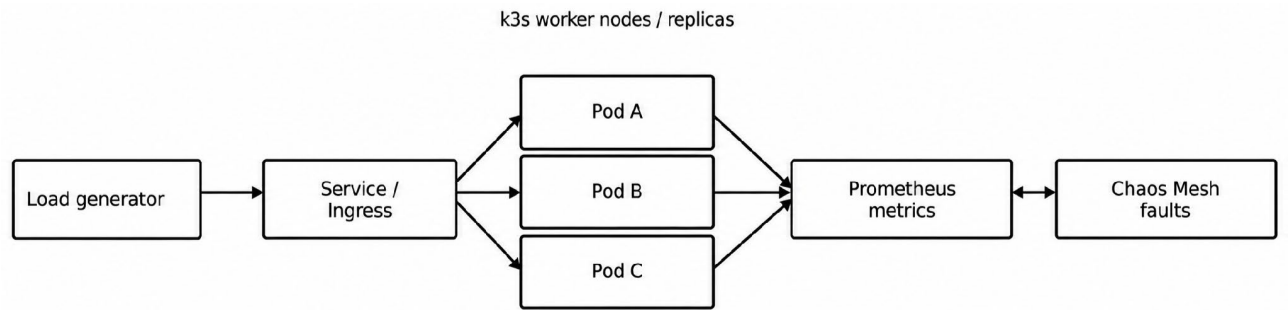


*Figure 1 – Reproducible FOSS workflow for container resilience experiments*

Four deployment configurations were considered. C1 contains one application replica without dedicated probes. C2 contains three replicas distributed by the scheduler. C3 uses three replicas with readiness and liveness probes. C4 adds pod anti-affinity and a PodDisruptionBudget (PDB) so that replicas are preferentially separated and planned disruptions cannot remove too many healthy instances simultaneously. Readiness probes are important because an unready pod is excluded from Service traffic even if its process is still running [1].

Table 1 – Fault scenarios for the resilience experiment

| Scenario | Fault | Duration / level | Observed effect |
|---|---|---|---|
| C-POD | Pod termination | one active pod | rescheduling/restart |
| C-NODE | Worker unavailable | 30 s | replica relocation |
| C-LAT | Network latency | +250 ms | tail latency |
| C-LOSS | Packet loss | 5% | errors/retries |

The pilot workload model generated 30 repetitions for each configuration with a fixed seed. Each repetition included a steady-state period followed by one fault episode. Recovery time was measured from fault activation to restoration of the target request success rate. Availability over the observation window was defined as

$$A = \frac{(T_{total} - T_{unavailable})}{T_{total}} * 100\%. \quad (1)$$

The request error rate was calculated as

$$R_{err} = \frac{N_{error}}{N_{requests}} * 100\%. \quad (2)$$

The p95 latency was used to capture tail behavior. For the future cluster experiment, the same metrics can be derived from HTTP counters and histogram buckets exported to Prometheus. The model does not emulate the internal Kubernetes controller loop; its purpose is to test the analysis workflow and produce falsifiable expectations for the real FOSS testbed.

## Results

Table 2 presents the pilot results aggregated over all four fault types. Replication alone reduced the modeled recovery interval and error rate, but health probes produced a further improvement by preventing traffic from being sent to instances that had not yet recovered. The configuration combining probes, anti-affinity, and a PDB had a median recovery time of 0.48 s, a mean error rate of 0.90%, and mean availability of 98.55%. The single-replica baseline was substantially more sensitive to the same injected conditions.

Table 2 – Pilot model-based resilience results

| Configuration | Median recovery, s | P95 recovery, s | Error rate, % | p95 latency, ms | Availability, % |
|---|---|---|---|---|---|
| C1: 1 replica | 4.54 | 6.09 | 7.80 | 422.88 | 92.00 |
| C2: 3 replicas | 1.72 | 2.12 | 2.89 | 321.48 | 95.59 |
| C3: 3 replicas + probes | 0.79 | 1.03 | 1.56 | 259.74 | 97.38 |
| C4: probes + anti-affinity + PDB | 0.48 | 0.57 | 0.90 | 220.47 | 98.55 |

## Discussion

The modeled trend is consistent with the design of Kubernetes fault-tolerance mechanisms. Multiple replicas reduce dependence on one pod, readiness probes protect the request path from

instances that cannot currently serve traffic, and liveness probes provide automatic restart for processes that enter an unrecoverable state [1]. Anti-affinity reduces the probability that all replicas share one worker, while a PDB constrains voluntary disruptions. These mechanisms address different failure modes and should therefore be tested together rather than treated as interchangeable.

Chaos engineering is useful because it evaluates the system during controlled violations of its steady state. Chaos Mesh is an open-source platform that supports orchestration of pod and network abnormalities [7], which makes it suitable for reproducing C-POD, C-LAT, and C-LOSS. The framework also aligns with current research that combines Kubernetes with controlled faults and evaluates response time, errors, and availability [2-5]. K3s reduces deployment overhead while remaining Kubernetes-conformant, making it appropriate for small laboratories, edge testbeds, and teaching environments [6].

The principal limitation is the synthetic nature of Table 2. The values should not be interpreted as measured k3s performance. Real recovery time depends on probe periods, scheduler load, image availability, node resources, CNI behavior, and application startup time. Consequently, the next stage must deploy the same four configurations on identical nodes, inject each fault repeatedly with Chaos Mesh, and calculate confidence intervals from Prometheus traces. The testbed should also separate planned disruptions from unplanned node loss, because a PDB does not prevent every failure type.

## Conclusions

A reproducible FOSS framework for evaluating resilience of containerized distributed systems was proposed. The framework combines k3s/Kubernetes, Chaos Mesh, Prometheus-compatible monitoring, and Python and defines common pod, node, latency, and packet-loss faults. In the pilot model, the transition from one replica to a configuration with probes, anti-affinity, and a PDB reduced median recovery time from 4.54 s to 0.48 s and the mean request error rate from 7.80% to 0.90%.

Future work will execute the scenarios in a physical or virtual k3s cluster, compare application-level and infrastructure-level faults, study the influence of probe parameters on false recovery decisions, and extend the experiment to stateful services and multi-service dependency chains.

## References


1. Kubernetes Authors. Configure Liveness, Readiness and Startup Probes. Kubernetes Documentation. https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/ (accessed 01.09.2026).
2. Singh S., Muntean C. H., Gupta S. Boosting Microservice Resilience: An Evaluation of Istio's Impact on Kubernetes Clusters Under Chaos. 2024 9th International Conference on Fog and Mobile Edge Computing (FMEC), 2024, pp. 245-252. https://doi.org/10.1109/FMEC62297.2024.10710237.
3. Wang H., Liu L., Yue C., Wang L. et al. Cloud-native systems resilience assessments based on Kubernetes architecture graph. Service Oriented Computing and Applications, 2024. https://doi.org/10.1007/s11761-024-00406-x.
4. Yu S., Wu H., Niu X., Nie C. A Systematic Literature Review on Fault Injection Testing of Microservice Systems. IEEE Transactions on Services Computing, 2025, pp. 1-22. https://doi.org/10.1109/TSC.2025.3621564.
5. Zaccarini M., Borsatti D., Cerroni W., Foschini L., Grabarnik G. Ya., Manca L. et al. TELKA: Twin-Enhanced Learning for Kubernetes Applications. IEEE ISCC, 2024, pp. 1-6. https://doi.org/10.1109/ISCC61673.2024.10733736.
6. K3s Project. K3s – Lightweight Kubernetes. https://docs.k3s.io/ (accessed 01.09.2026).
7. Chaos Mesh Authors. Chaos Mesh Overview, version 2.8.4. https://chaos-mesh.org/docs/ (accessed 01.09.2026).
8. Figueira J., Coutinho C. Developing self-adaptive microservices. Procedia Computer Science, 2024, 232, 264-273. https://doi.org/10.1016/j.procs.2024.01.026.