Testing Container Orchestration Systems: A Literature Review

He Wen
Vrije Universiteit Amsterdam

Matthijs Jansen
Vrije Universiteit Amsterdam

Daniele Bonetta
Vrije Universiteit Amsterdam

Abstract

With the rise of microservices and cloud-native applications, container orchestration systems have become essential for managing the complexity of containerized service infrastructure. They enable organizations to automate deployment, scale services dynamically, and ensure high availability in distributed environments. Therefore, it is crucial to validate the correctness of container orchestration systems. This paper systematically identifies key testing objectives for COSs, encompassing functionality, resiliency, performance, security, and observability. We conduct a comprehensive literature review of state-of-the-art testing techniques, organizing them into a structured taxonomy. Additionally, we highlight research gaps and challenges, providing insights into future research directions for improving COS validation.

1 Introduction

Containers have gained significant attention in recent years for their ability to encapsulate software and its dependencies within standalone, self-contained units. By isolating processes at the operating system level, containers allow multiple users and applications to share the resources of a single compute node. While similar to virtual machines (VM), containers are lightweight [26] and offer benefits like easy deployment, testing, and application composition. These characteristics have made containers a foundational technology for modern software architectures, particularly in the context of microservices [37] and cloud-native applications [57].

As microservices and cloud-native architectures grow, managing containerized services has become more complex, leading to the widespread adoption of container orchestration systems (COS). By simplifying both initial deployments and ongoing management of multiple containers as a single unit, COSs provide a framework for managing containers at scale. They are essential for automating deployments, scaling services as needed, and ensuring high availability in distributed environments.

Therefore, testing COSs is a vital process for validating service functionality as well as maintaining system reliability, performance, and security. According to Cloud Native Computing Foundation (CNCF) Annual Survey 2023 [22], 84% of surveyed organizations are using or evaluating Kubernetes [10], one of the most popular COSs. As organizations increasingly rely on container orchestration to manage their infrastructure, the need for rigorous testing practices has grown.

However, traditional testing methodologies often fall short in capturing the complexity of COSs due to their dynamic and distributed nature. Containers can be created, terminated, or rescheduled across nodes, making consistent testing conditions difficult to maintain [66, 78]. Testing for scalability and resource management requires specialized techniques that account for dynamic resource allocation [54]. Security testing is similarly challenging due to container and network isolation risks [72, 90]. Additionally, failure modes are varied, with failures and partial outages that traditional tests might overlook [6, 88].

There has been an active and continuing research in container orchestration testing, including chaos engineering [7, 56, 79], bug analysis [6, 48, 88], and security validation [47, 93]. However, *there has not been a recent and thorough survey of testing objectives and techniques of COSs*. This literature review aims to provide a comprehensive overview of existing research on testing strategies for COSs. It explores the main testing objectives and the current state of testing practices. By synthesizing existing work, this review seeks to identify research gaps and suggest future directions for advancing testing methodologies in container orchestration environments. The main contributions of this literature review are as follows:

1. **Identification of key testing objectives:** This review analyzes the core components and execution pipeline of COS to extract their primary operational tasks. Based on this analysis, this review identifies the primary objectives of testing in container orchestration environments, which

Table 1: Evaluation of related publications

Publication	Year	Container	Orchestration	Testing	Objectives
Nachiyappan et al. [53]	2015			✔	Cloud testing dimensions & tools & challenges
Weerasiri et al. [86]	2017		✔		Taxonomy of cloud resource management techniques
Pahl et al. [57]	2017	✔	✔		Container-based technologies classification framework
Bachiega et al. [5]	2018	✔		✔	Survey on container performance evaluation
Bertolino et al. [8]	2019			✔	Cloud testing classification framework
Casalicchio et al. [14]	2019	✔	✔		Taxonomy of container and container orchestrator
Sultan et al. [72]	2019	✔		✔	Survey on container security and solutions
Rodriguez et al. [64]	2019	✔	✔		Taxonomy of container orchestration systems
Watada et al. [85]	2019	✔	✔		Trends & challenges in containerization
Casalicchio et al. [15]	2020	✔	✔		Study of SotA container technologies
Goodarzy et al. [29]	2020		✔		Cloud resource management using ML
Zhong et al. [94]	2022	✔	✔		Taxonomy of ML-based container orchestration solutions
Queiroz et al. [61]	2023	✔	✔		Container virtualization in real-time industrial systems
Solayman et al. [69]	2023	✔			Container virtualization in IoT
Malhotra et al. [49]	2024	✔		✔	Container maintenance
This literature review	2025	✔	✔	✔	Testing container orchestration systems

serve as the foundation for understanding the purpose and direction of existing testing practices. (**Section 4**)

2. **Design of a taxonomy of testing approaches:** The review introduces a structured taxonomy of testing methods and techniques, organized by the identified testing objectives. This taxonomy offers a framework for analyzing the breadth and depth of testing practices in COSs. (**Section 5**)

3. **Investigation of research gaps and future directions:** The review uncovers unresolved research challenges and proposes future research directions to advance testing methodologies, providing insights to container orchestration testing practices. (**Section 6**)

2 Related Work

Table 1 illustrates the increasing body of literature reviews as containerization has gained popularity. Existing studies have explored various dimensions of container-related technologies, including cloud testing methodologies, resource management techniques, and container orchestration framework taxonomies. While related work provides valuable insights into the components and operations of COSs, systematic research on testing COSs remains limited. This study aims to fill that gap by contributing a focused literature review on testing COSs, bridging existing research domains, identifying existing contributions to COS testing, and outlining the need for future work in validating COS behavior.

Cloud Testing. Cloud testing has been extensively studied, with researchers exploring various dimensions of testing methodologies and tools. Nachiyappan et al. [53] present a study that identifies key testing dimensions, compares cloud testing tools, and highlights the associated challenges. Bertolino et al. [8] extend this research by introducing a classification framework that segments cloud testing into Testing in the Cloud (TiC), Testing of the Cloud (ToC), and Testing of the Cloud in the Cloud (ToiC). These studies provide a foundational understanding of cloud testing practices, which can be extended to the unique complexities of COS environments. However, while cloud testing frameworks can offer useful insights, they do not directly address the orchestration-specific challenges of distributed, containerized systems. This highlights the need for a literature review focusing specifically on testing COSs.

Cloud Resource Management. Efficient resource management is critical to optimizing cloud and containerized environments. Weerasiri et al. [86] propose a multidimensional taxonomy that facilitates the systematic analysis, comparison, and classification of cloud resource orchestration technologies. Furthermore, Goodarzy et al. [29] explore machine learning (ML) techniques for resource management, illustrating how ML algorithms can enhance cloud resource allocation and optimization. These studies collectively inform the development of more adaptive and efficient resource management strategies for COSs. While resource management is tightly coupled with COSs, the impact of resource orchestration, especially container orchestration, on system correctness and

reliability has not been systematically explored from a testing perspective.

Container-related Technologies. The evolution of container technologies has motivated considerable research into their performance [5], security [72], and orchestration capabilities. Pahl et al. [57] systematically classify and compare research on container-based technologies, focusing on their cloud applications. Watada et al. [85] perform a detailed study of research trends and challenges in various aspects of containerization such as container networking, security, orchestration, and performance analysis. Similarly, Casalicchio et al. [15] present a taxonomy of state-of-the-art container technique solutions from aspects of containerized application performance, orchestration, and cyber-security. While existing studies provide an understanding of container-based technologies and evaluation approaches, they have not fully addressed the challenges in container orchestration.

Container Orchestration. The orchestration of cloud and containerized environments has been the subject of extensive research, with studies focusing on classification and analysis. Casalicchio et al. [14] provide a comprehensive survey of container orchestration technologies, introducing a taxonomy that classifies orchestration solutions based on resource limit control, scheduling mechanisms, and other key features. This taxonomy serves as a structured framework for understanding the capabilities and distinctions among COSs. Rodriguez et al. [64] further contribute by proposing a taxonomy that classifies COSs from the perspectives of scheduling, application models, and resource management. While these studies provide valuable insights into how COSs function, they do not deeply explore how these orchestration processes should be tested to ensure the correctness and robustness of COSs.

3 Research Methodology

This literature review follows the methodology by Kitchenham et al. [36], including three main phases: planning the review (**Section 3.1**), conducting the review (**Section 3.2**), and reporting the results (**Section 5, 6**).

3.1 Planning the Review

The main goal of this literature review is to understand the main objectives and state-of-the-art approaches in testing COS. In particular, this literature review aims to answer the following research questions:

- **RQ1:** What are the main objectives for testing container orchestration systems?
- **RQ2:** What are the current test methods, techniques, and tools mainly used in container orchestration systems testing?
- **RQ3:** What are the research gaps and future research directions of container orchestration systems testing?

3.2 Conducting the Review

This section outlines the approach used to identify, select, and analyze relevant studies. The process of conducting the review includes defining publication selection criteria, applying a snowballing [87] strategy to enhance coverage, and systematically extracting and evaluating key insights from the reviewed works.

Publication Selection. To obtain relevant publications for this review, we conducted a comprehensive search across digital libraries, including Google Scholar[1], dblp[2], and Semantic Scholar[3].

We formulated a structured query string to target key concepts related to container orchestration and testing methodologies. The search included keywords such as `"container orchestration"` and `"testing"` and was performed by matching terms within the title, abstract, and keyword fields of publications. Given that Kubernetes and Docker Swarm [24] are the two most widely used COSs, their testing methodologies were explicitly included in the query. The search was restricted to English-language papers published between 2019 and 2025.

After gathering an initial set of related publications, we identified the primary testing objectives by analyzing their core topics and referencing the architecture outlined in Section 4. To ensure comprehensive coverage, we conducted a secondary query using keywords corresponding to various testing objectives, including `"functional testing"`, `"fault injection"`, `"performance"`, and `"security"`.

The final query string is as shown in Listing 1.

Listing 1: Query String

```
('container'
 or 'container orchestration'
 or 'container management'
 or 'container orchestration system'
 or 'container orchestration framework'
 or 'Kubernetes' or 'Docker Swarm' )
and
('functional' or 'fault injection'
 or 'chaos engineering'
 or 'performance' or 'scalability'
 or 'security' or 'observability')
and
('test' or 'testing'
 or 'validation' or 'verification')
```

[1]Google Scholar: https://scholar.google.com/
[2]dblp: https://dblp.org/
[3]Semantic Scholar: https://www.semanticscholar.org/

Table 2: Inclusion and exclusion criteria

Inclusion Criteria
- Studies presenting or benchmarking container orchestration architecture, platform, or framework - Studies related to testing containerized services - Studies presenting container testing issues or goals - Studies presenting container orchestration testing issues or goals - Studies presenting container orchestration testing strategies or tools
Exclusion Criteria
- Studies discussing general cloud computing, virtualization, or microservices without linking to COS - Blog posts, theses, and unpublished preprints (*e.g.* arXiv)

The obtained publications were then evaluated according to the *Inclusion and Exclusion Criteria* defined in **Table 2**. The *inclusion criteria* focuses on studies that directly contribute to understanding COS and their testing methodologies. Studies addressing testing challenges, goals, and tools for COS are prioritized. To ensure a comprehensive review, studies that examine general container testing challenges are also included, as these often reveal relevant orchestration-related insights. The *exclusion criteria* filters out studies that lack direct relevance to COS. Furthermore, non-peer-reviewed sources are omitted to maintain the reliability and academic rigor of the review.

Snowballing. To enhance the comprehensiveness of the literature review, snowballing is employed as a complementary search strategy. This includes *backward snowballing*, which examines reference lists of selected papers to identify foundational works on COS architectures and testing strategies, and *forward snowballing*, which explores citations of selected papers to uncover recent research that builds upon or extends their findings.

Data Extraction and Analysis. The collected publication data is systematically processed in order to obtain relevant information to address the research objectives. The publications are categorized by labels from multiple dimensions, including testing objective, testing strategy, and context.

4 Testing Objectives of COS

To address RQ1, we analyze several reference architectures [3, 64] to establish a systematic mapping between the COS's primary components and their corresponding testing objectives. Our analysis follows a two-step approach. First, we review the key responsibilities of COSs as identified in prior work. Second, we examine each component's primary duty according to the previous step.

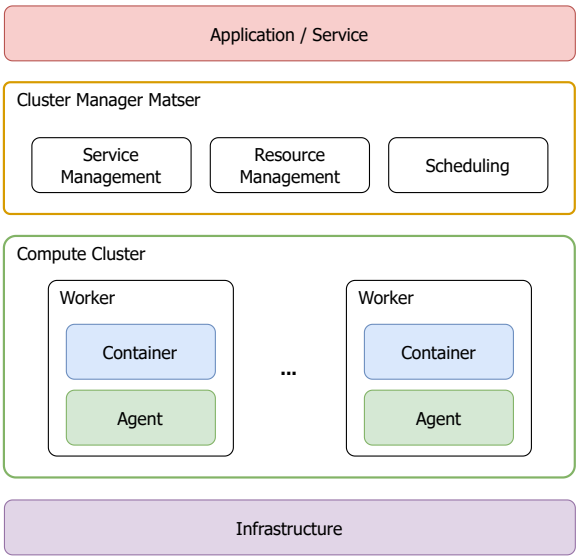

Figure 1: Architecture for Container Orchestration Systems (COS)

4.1 Reference Architecture

Khan [35] highlights responsibilities such as cluster state management, scheduling, high availability, fault tolerance, security, networking, service discovery, continuous deployment, monitoring, and governance. In addition, Andreadis et al. [3] propose a detailed model that defines four fundamental responsibilities: job processing, task processing, scheduler management, and resource management. Rodriguez et al. [64] further emphasize the inter-component interactions that shape the overall COS workflow.

As shown in **Figure 1**, four main layers are identified in the reference architecture: the *Application/Service* layer, the *Cluster Manager Master*, the *Compute Cluster*, and the physical *Infrastructure*.

From a high-level perspective, the end-user application or service is built, packaged into containers, and deployed to the *Compute Cluster* via the orchestration system (*Cluster Manager Master*). The underlying hardware or virtualized *Infrastructure* provides the foundational resources needed to host the clusters, which can be either VMs, cloud infrastructures, physical machines, or edge devices.

Cluster Manager Master. The cluster manager master serves as the core component of a COS. The **service management** module provides high-level functional capabilities for building, deploying, and managing applications. It is respon-

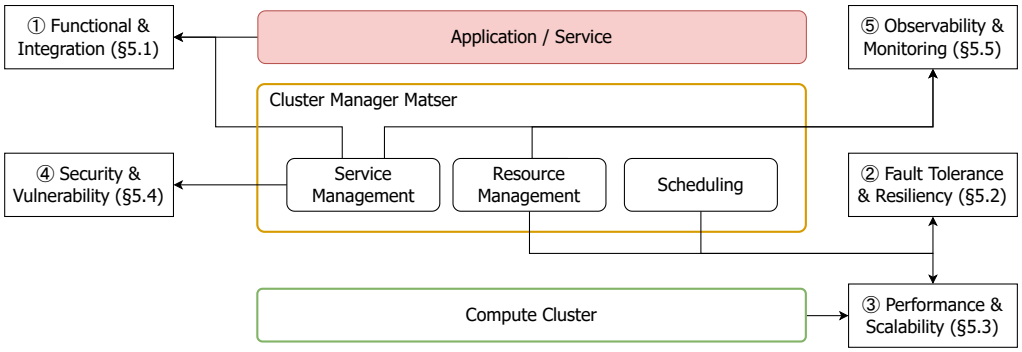

Figure 2: Mapping from components to testing objectives for Container Orchestration Systems (COS)

sible for launching containers on cluster nodes, supporting service discovery, ensuring service readiness, and continuously monitoring task life cycles. The **resource management** module is responsible for the efficient allocation and provisioning of resources, including memory, CPU/GPU, disk space, local and persistent volumes, and virtual networking components. This module tracks resource consumption in real time, provisions new nodes when necessary, and balances resource usage. The **scheduling** module manages task placement across cluster nodes, orchestrates scheduling hierarchies, and resolves competing task priorities. It also manages various stages of the task life cycle, such as preemption, rescheduling, and replication, to ensure the efficient and reliable execution of tasks across the cluster.

Compute Cluster. The compute cluster forms the execution layer of a COS, consisting of multiple worker nodes that run containerized workloads. Each worker node includes two primary components: the container and the agent. The **container** encapsulates the application and its dependencies, providing an isolated runtime environment. The **agent** acts as a local control point, communicating with the cluster manager master to receive and execute scheduling decisions, monitor resource usage, and report the node's status. The agent is also responsible for managing container life-cycle events, including starting, stopping, and restarting containers based on the master's directives.

4.2 Testing Objective Mapping

Each of the COS components plays a crucial role in ensuring system correctness, reliability, and efficiency. However, their complexity introduces potential points of failure that require rigorous testing. As the central control plane of the COS, the *Cluster Manager Master* manages service deployment, resource allocation, and task scheduling. Given its critical role, failures at this level can cause widespread service disruptions or performance bottlenecks. To systematically validate COS behavior, we map the architectural components of *Cluster Manager Master* to corresponding testing objectives. Figure 2 illustrates this mapping, providing a visual summary of the relationships.

The **Service Management Module** is responsible for deploying and maintaining applications by launching containers and managing service discovery. Failures in this functionality, such as unsuccessful container launches or broken service discovery, can lead to application downtime or incorrect routing. To address this, *Functional and Integration Testing* is required to validate container lifecycle management and inter-service communication. Additionally, this module ensures service availability through readiness checks and continuous monitoring. Ineffective readiness probes or monitoring failures may cause delayed failure detection or improper failovers. Therefore, *Observability and Monitoring Validation* is necessary to verify the accuracy of collected metrics and logs. Furthermore, this module governs service isolation and access control, ensuring security in multi-tenant environments. Misconfigurations could result in privilege escalation or unauthorized access, making *Security and Vulnerability Testing* critical for validating authentication, authorization, and network policies.

The **Resource Management Module** ensures efficient allocation and provisioning of computing resources, dynamically balancing workloads across nodes. Failures in this function, such as resource exhaustion or inefficient scaling, can degrade performance and cause service starvation. *Performance and Scalability Testing* is essential to evaluate how well the module handles dynamic workload changes. Additionally, the module is responsible for real-time resource tracking to optimize cluster efficiency. Inaccurate tracking may lead to over-provisioning or under-utilization, necessitating *Observability and Monitoring Validation* to ensure accurate telemetry and reporting. Finally, workload reallocation during failures is a key responsibility, preventing service disruptions by redistributing workloads to healthy nodes. Failures in this function could lead to prolonged outages or increased response times, requiring *Fault Tolerance and Resiliency Testing* to validate resource reallocation under stress conditions.

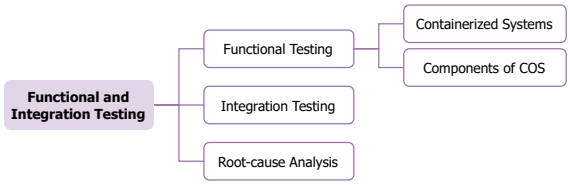

Figure 3: Taxonomy of functional and integration testing for Container Orchestration Systems (COS)

The **Scheduling Module** is tasked with placing tasks across the cluster based on scheduling policies and available resources. Inefficient scheduling can result in high scheduling latency, poor resource utilization, and degraded performance. *Performance and Scalability Testing* is necessary to measure task placement efficiency and scheduling overhead. Furthermore, the module handles task preemption, rescheduling, and replication to ensure continuous workload execution under node failures or workload fluctuations. Ineffective rescheduling strategies can lead to system instability or failed recovery, making *Fault Tolerance and Resiliency Testing* essential for validating task recovery mechanisms.

5 Taxonomy of COS Testing Objectives

To answer RQ2, we present a taxonomy of COS testing based on testing objectives as mentioned in Figure 2. For each objective, the state-of-the-art testing techniques based on the obtained publications are introduced.

5.1 Functional and Integration Testing

Observations:
O-1: Most studies emphasize automated testing frameworks to reduce manual effort and enhances test coverage.
O-2: Most functional testing approaches focus on predeployment validation rather than dynamically adapting to runtime changes in COSs.
O-3: Most functional testing approaches operate in a single-cluster environment.
O-4: The verifications of core components of COS mostly focus on controllers.

Ensuring the reliability of COSs requires rigorous functional and integration testing to validate the correctness of individual components and the seamless interaction of containerized services. As shown in Figure 3, this section explores key approaches to testing COSs, structured into three main areas. First, functional testing ensures that both containerized applications and core COS components operate as expected, covering automated frameworks, stress testing, and model-based verification. Second, integration testing focuses on validating interactions between microservices and orchestration layers, incorporating smoke testing and large-scale system integration. Finally, root-cause analysis addresses debugging challenges in COS environments, particularly in diagnosing failures across complex, distributed infrastructures.

The primary objective of **functional testing** in COSs is to validate the correctness and expected behavior of both containerized applications and the orchestration infrastructure itself. This encompasses two key aspects: (1) verifying that containerized applications function as intended within the orchestrated environment and (2) ensuring that the individual components of the COS operate according to specified requirements.

To automate the process of **testing containerized microservices**, various approaches have been proposed to assess application functionality. Astyrakakis et al. [4] introduce a stress-testing framework that validates Kubernetes applications submitted to a public marketplace, ensuring they align with their YAML descriptors and comply with cloud-native principles such as scalability and redundancy. Expanding on automated testing for cloud-native applications, Nikolaidis et al. [55] propose *Frisbee*, a Kubernetes-native platform for systematically testing containerized applications. *Frisbee* provides an integrated architecture for fault injection, system interaction monitoring, and service structure modeling for cloud-native systems within Kubernetes. By leveraging Kubernetes' abstraction, *Frisbee* overcomes the limitations of traditional cloud testing approaches [8, 16], allowing engineers to design tests in local environments and seamlessly execute them across diverse infrastructures. Further refining the validation of microservice behavior, Turin et al. [80] propose a formal model that simulates resource consumption and scaling in Kubernetes-managed services, comparing simulation results against real-world observations to ensure system correctness.

Beyond application testing, COSs consist of numerous critical components, including schedulers, controller managers, and network proxies, all of which must function reliably to maintain system stability. Ensuring **the correctness of core orchestration components** requires systematic verification techniques. A model-based approach is often employed for formal verification. Xu et al. [88] investigate Kubernetes Operator bugs, categorizing common patterns and assessing their impact on system stability. Addressing the need for automated verification, Gu et al. propose *Acto* [30], an end-to-end testing framework that models Kubernetes Operators as state machines. By systematically exploring state transitions, *Acto* verifies both operator correctness and the ability of managed systems to achieve their intended states. Extending formal verification to broader COS components, Liu et al. present *Kivi* [43], a model-checking framework that encodes controllers and their interactions as processes. *Kivi* exhaustively verifies these interactions, identifying misconfigurations and

controller logic flaws that could compromise system behavior.

While functional testing validates individual components, **integration testing** is essential for ensuring that containerized services interact correctly and function cohesively under orchestration. As an initial step in this process, Cannavacciuolo et al. [12] focus on smoke testing, introducing an automatic test generator that derives test cases from predefined oracles. Their approach includes platform sanity checks, log analysis, response validation, and persistent volume verification. Expanding on integration testing, Reile et al. [63] present *Bunk8s*, a testing tool for microservices in Kubernetes. It interacts with both the CI/CD pipeline and COS, launching and coordinating test runner containers in test runner pods.

While functional and integration testing validates system and component behavior, effective **root-cause analysis** is essential for diagnosing and preventing infrastructure-related bugs that may occur post-deployment. To diagnose failures in distributed orchestration environments, Sun et al. [75] propose an approach that reconstructs service execution histories to analyze failures. Tang et al. [76] conduct an in-depth root-cause analysis of cross-system interaction failures in modern cloud system orchestrations and advocate for cross-system testing and verification. Extending failure analysis beyond COSs, Drosos et al. [25] conduct an extensive analysis of bugs in Infrastructure as Code (IaC) software. Given the architectural similarities between IaC and COSs, they suggest adapting testing techniques from COSs to enhance IaC validation.

5.2 Fault Tolerance and Resiliency Testing

O-5: Fault injection in general is not limited to containerized applications but also extends to the COS itself.
O-6: Both fault injection and chaos engineering rely on predefined failure scenarios.
O-7: There are plenty of chaos engineering tools targeting Kubernetes, however, there is a lack of systematic research on chaos engineering of COS.

To ensure the fault tolerance and resilience of COSs, various testing methodologies have been developed to assess their behavior under failure scenarios and validate their correctness. As shown in Figure 4, this section examines three most popular approaches: fault injection, chaos engineering, and formal verification, each contributing to a comprehensive reliability assessment.

Fault injection is a widely adopted approach for evaluating the robustness of cloud systems by deliberately introducing faults into protocols and implementations [36, 44, 52]. In the context of containerized services, Flora et al. [28] highlight its significance in assessing the fault tolerance of COS managing microservices. Various fault injection techniques have been developed to target different categories of bugs in containerized environments.

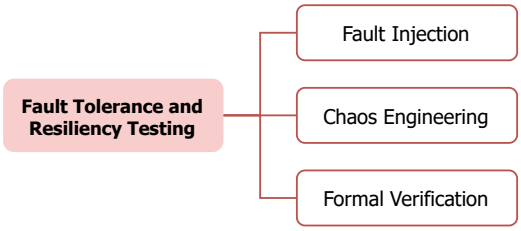

Figure 4: Taxonomy of fault tolerance and resiliency testing for Container Orchestration Systems (COS)

For instance, Lu et al. [45] introduce *CrashTuner*, a fault-injection testing approach designed to detect crash recovery bugs. *CrashTuner* leverages meta-information analysis to automatically infer high-level system state variables, whose access points serve as fault injection targets likely to reveal errors. Addressing network partition bugs, Chen et al. [18] propose *CoFI*, a consistency-guided fault injection technique that strategically injects network partitions to expose consistency violations in cloud systems. *CoFI* not only controls when the partition begins but also determines the optimal stopping point to maximize bug discovery. Beyond crash recovery and network faults, time-out bugs pose additional challenges in distributed environments. To address these, Chen et al. [19] introduce *Chronos*, an automated testing framework that employs fuzzing to dynamically generate fine-grained delay sequences at runtime. Unlike traditional fault injection techniques that rely on direct system perturbations, fuzzing injects faults through malformed inputs, enabling the detection of transient failures caused by unexpected delays in *Chronos*.

Apart from evaluating the fault tolerance ability of containerized systems, fault injection technique can also be utilized to access the resiliency of COSs themselves. Focusing on COS controllers, Sun et al. [73] introduce *Sieve*, an automated reliability testing tool that verifies whether controllers can maintain correct system operations under common perturbations. Their findings demonstrate that fault injection is an effective method for assessing orchestration resiliency. Barletta et al. [6] further reinforce this perspective by conducting a comprehensive analysis of real-world Kubernetes failures. Through fault injection experiments that modify the cluster state data, they successfully reproduce known incidents and uncover new failure patterns, highlighting the role of fault injection in identifying vulnerabilities within COS.

While fault injection introduces specific failures to test system behavior, **chaos engineering** [7] expands this approach by systematically experimenting on distributed systems to evaluate their resilience under real-world conditions. Unlike conventional fault injection, which often targets predefined failure scenarios, chaos engineering applies systematic disruptions of various system components in a controlled environ-

ment to uncover unexpected weaknesses and improve system reliability.

Camacho et al. [11] propose *Chaos as a Software Product Line*, an architecture that enables fault injection as a configurable software product line in cloud environments, targeting OpenShift and Kubernetes-based clusters. Similarly, Nikolaidis et al. [56] introduce *Frisbee*, an automated chaos testing platform for Kubernetes-based distributed applications. *Frisbee* employs an event-driven approach, integrating runtime event collection with a scenario modeling language to define and execute test cases dynamically.

Expanding the scope beyond Kubernetes, Higgins et al. [31] highlight the challenges of conducting chaos experiments in multi-cloud Docker Swarm clusters and present *Swarm Storm*, a framework designed to orchestrate chaos engineering experiments in a cloud-agnostic environment with comprehensive testing features. Meanwhile, Simonsson et al. [67] focus on system call-level chaos testing in containerized applications. Their tool, *ChaosOrca*, stands out by performing experiments under production-like workloads without instrumenting the application itself, ensuring minimal interference. Further refining this approach, they introduce realistic error model generation, which aims to replicate failures that naturally occur in production environments [92]. Finally, Ikeuchi et al. [32] bridge chaos engineering with machine learning, presenting a framework that utilizes failure injection tools to train a recovery policy using deep reinforcement learning. Their approach emphasizes automated self-healing mechanisms in containerized environments.

Similar to fault injection, while chaos engineering techniques have been extensively applied to containerized applications, their impact extends beyond individual services to the orchestration systems themselves. Several chaos engineering tools have been proposed to assess the fault tolerance and resiliency capabilities of COSs by enabling controlled disruption of network, pods, and control-plane components [17, 39, 41, 68].

Although fault injection and chaos engineering effectively expose weaknesses in COSs by simulating failures, they inherently rely on specific failure scenarios, thus they do not provide exhaustive guarantees of correctness. In contrast, **formal verification** offers a more rigorous approach by proving safety and liveness across all possible system executions. To ensure that COS components behave correctly under all conditions, studies have explored verification techniques that mathematically establish correctness of control components.

Liu et al. [42] propose a verification approach that models control components and their environment as parametric transition systems, using symbolic model checking to verify safety and liveness properties. Similarly, Sun et al. [74] introduce *Anvil*, a framework for formally verifying that COS controllers implement eventually stable reconciliation, a concise temporal logic liveness property ensuring that the system consistently converges to a correct state.

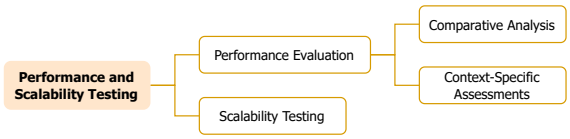

Figure 5: Taxonomy of performance and scalability testing for Container Orchestration Systems (COS)

5.3 Performance and Scalability Testing

O-8: Kubernetes generally outperforms its counterparts in resource efficiency and performance in comparative analyses.
O-9: Cluster provisioning time and failure recovery time are common metrics for performance evaluation.
O-10: Kubernetes' performance in specific context (*e.g.* edge computing) varies significantly.
O-11: There is no universally accepted methodology for evaluating COS performance across different use cases.
O-12: Many evaluations on scheduling algorithms rely on simulated workloads rather than real-world traffic patterns.

As COSs are increasingly deployed in complex, large-scale environments, evaluating their performance and scalability becomes essential. Performance testing focuses on how efficiently a COS manages resources, schedules workloads, and handles failures, while scalability testing examines how well it adapts to increasing demands and distributed workloads. Researchers have approached this evaluation through comparative benchmarking, deployment-specific assessments, and large-scale stress testing.

As shown in Figure 5, this section presents key studies that assess COS performance across different metrics, explore the impact of scheduling strategies, and introduce tools designed for large-scale benchmarking. By analyzing these works, we gain insights into the trade-offs between efficiency, fault tolerance, and resource management in modern COSs.

Various studies have **compared COS** by analyzing their efficiency across different deployment scenarios and performance metrics. Pan et al. [59] compare Kubernetes and Docker Swarm, assessing the overhead introduced at worker nodes and the impact on inter-container communication. Their findings highlight Kubernetes as generally more resource-efficient than Swarm. Expanding the comparison scope, Jawarneh et al. [33] conduct a comprehensive benchmarking analysis of multiple COSs, including Docker Swarm, Kubernetes, Apache Mesos, and Cattle. They examine both qualitative aspects—such as resource management, service management, and scheduling capabilities—and performance metrics, including cluster provisioning time, application de-

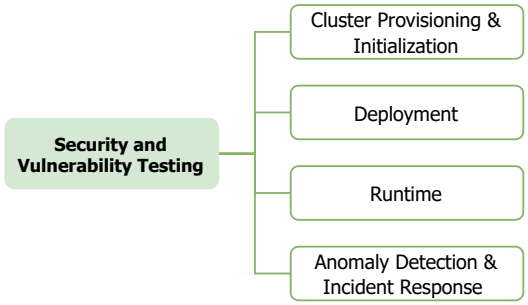

Figure 6: Taxonomy of security and vulnerability testing for Container Orchestration Systems (COS)

ployment time under varying complexities, provisioning time using local images versus Docker registry, and failover recovery time. In a more structured evaluation, Straesser et al. [71] introduce *COFFEE*, a systematic benchmarking framework that correlates performance metrics with key orchestration requirements. *COFFEE* evaluates COS performance across three dimensions: container provisioning and networking, failure recovery, and rolling updates, providing a standardized methodology for assessing orchestration efficiency.

Beyond general benchmarking, some studies dive into performance evaluations within **specific deployment contexts**. Cilic et al. [21] examine the performance of container orchestration systems in edge computing environments. Their evaluation considers the efficiency of Kubernetes, K3s, KubeEdge, and ioFog under conditions such as private network deployments, node heterogeneity, and resource-constrained edge nodes. Offering insights into real-world resource allocation challenges, Prahong et al. [60] assess COS effectiveness using a structured evaluation framework. Their study examines key performance indicators, including request throughput, response rate, and resource extension periods, across Kubernetes, Docker Swarm, and Apache Mesos.

Scalability testing, in contrast, focuses on how COS handles increasing workloads and ensures efficient resource distribution as the system scales. Voievodin et al. [82] propose a set of evaluation metrics for COS scheduling strategies, including cluster resource utilization, container deployment density, scheduling request satisfaction ratio, and fault tolerance. Expanding on this, their subsequent work [83] presents a comprehensive application design for evaluating scheduling strategies' performance.

To model and analyze scheduling algorithms, Pan et al. [58] develop a queuing theory-based framework for evaluating three scheduling algorithms: FIFO, Capacity Scheduler, and Fair Scheduler. Meanwhile, Straesser et al. [70] bridge the gap between microservice performance simulation and modern COS frameworks by enabling realistic scalability testing scenarios with minimal overhead.

Focusing on benchmarking large-scale Kubernetes deployments, Malleni et al. [50] introduce *Kube-burner*, a tool designed for performance and scalability testing under large-scale workloads. *Kube-burner* facilitates benchmarking by automating resource creation, deletion, and patching at scale, collecting detailed metrics via the Kubernetes API, and providing observability by scraping user-defined metrics under load.

5.4 Security and Vulnerability Testing

O-13: Static analysis is a popular approach for security validation.
O-14: There is a lack of research targeting COS component-related security.
O-15: Machine learning-related approaches are popular for anomaly detection.
O-16: Most security testing research focus on Kubernetes.

As shown in Figure 6, the research on security and vulnerability testing of COS covers all stages of a COS lifecycle.

The security lifecycle of a COS consists of several critical stages, each presenting unique challenges and risks. The process begins with **cluster provisioning and initialization**, which involves setting up control planes, configuring nodes, and defining network policies. At this stage, misconfigurations, insecure defaults, exposed API endpoints, and weak authentication mechanisms can create vulnerabilities that attackers may exploit. The next stage, **deployment**, introduces additional risks as applications are launched and security policies are applied. Without proper admission controls and privilege management, containers may gain excessive permissions, leading to privilege escalation and unauthorized access. Once deployed, the **runtime** phase focuses on protecting active workloads from threats such as container escapes, lateral movement, and API exploitation. To ensure the ongoing security of the system, **anomaly detection and incident response** are employed to provide continuous oversight of COS activities. This phase identifies security threats in real time, allowing for swift mitigation before they escalate. Addressing security concerns across all these phases is essential for maintaining a resilient and secure container orchestration environment.

During **cluster provisioning and initialization**, security testing plays a crucial role in mitigating vulnerabilities related to misconfigurations, insecure defaults, and weak authentication mechanisms. Rahman et al. [62] conduct an empirical study on Kubernetes security, categorizing common misconfigurations and introducing *SLI-KUBE*, a static analysis tool designed to quantify the frequency and severity of these security weaknesses. Mahajan et al. [46] highlight the security implications of Kubernetes operators, emphasizing the need

for validation mechanisms to assess their impact on cluster security. Expanding on this, Mahavaishnavi et al. [47] propose a comprehensive security framework aimed at addressing orchestrator-level security concerns. Their approach integrates real-time monitoring to detect misconfigurations, privilege escalation attempts, and unauthorized access within the orchestration infrastructure. Beyond general misconfigurations, Zheng et al. [93] target REST API vulnerabilities in Kubernetes, introducing *KubeFuzzer*, a black-box fuzzing tool that leverages Natural Language Processing (NLP) to generate sophisticated request sequences, increasing the probability of discovering security flaws.

As the system transitions into the **deployment** phase, security risks shift towards vulnerabilities in container images, policy misconfigurations, and compliance violations. Ensuring that only secure and authorized configurations are deployed is crucial in this stage. Kudo et al. [38] enhance Kubernetes application security by validating resource manifests during the admission process, preventing unauthorized configurations. Kamieniarz et al. [34] further refine security assessments by presenting a comparative analysis tool that evaluates Kubernetes security postures under different configurations, identifying weaknesses in resource policies and compliance gaps. From a hardware security perspective, Fernandez et al. [27] propose a container orchestration engine that integrates hardware-based Trusted Execution Environments (TEE) to strengthen data protection at the infrastructure level. Complementing these strategies, Zhu et al. [95] introduce an automated system for Kubernetes AppArmor profile generation. This system collects and transforms behavioral data from distributed worker nodes into security policies, enforcing them seamlessly without service disruption.

During **runtime**, when containers actively interact with networks, execute workloads, and manage sensitive data, real-time security enforcement becomes paramount. Security vulnerabilities at this stage may arise from container escapes, privilege escalation, lateral movement, and API exploitation. Addressing these threats, Dell'Immagine et al. [23] present *KubeHound*, a tool that detects security smells in microservice applications by leveraging both static and dynamic analysis techniques. Using similar approach, Verderame et al. [81] extend security automation within the DevOps pipeline, proposing a framework that not only prevents vulnerabilities but also actively mitigates runtime security risks through container hardening, compliance verification, and runtime monitoring. Moving beyond static defenses, Torkura et al. [79] adopt chaos engineering principles, introducing security fault injection to simulate real-world attack scenarios. This proactive approach assesses infrastructure resilience by testing how systems handle security failures affecting confidentiality, integrity, and availability.

Extending runtime security, **anomaly detection and incident response** are critical for identifying, analyzing, and mitigating security threats in real time. As traditional monitoring techniques struggle to keep pace with evolving attack vectors, AI-driven security analytics have emerged as a promising approach. Several studies [1, 2, 9, 13, 77] explore machine learning and neural network-based models that analyze runtime behaviors to enable real-time, multi-class threat detection in Kubernetes environments. By continuously learning from historical attack patterns and system anomalies, these models enhance detection accuracy while reducing false positives. Such adaptive security mechanisms ensure that containerized environments remain resilient against both known and emerging cyber threats.

5.5 Observability and Monitoring Validation

O-17: LLM-related approaches are popular for enhancing COS observability.

Beyond security-specific anomaly detection, observability plays a crucial role in maintaining the overall health, reliability, and performance of Kubernetes-based environments. Effective monitoring mechanisms not only detect security threats but also help identify system failures, misconfigurations, and operational inefficiencies. By leveraging advanced monitoring frameworks, automated logging, and AI-driven analytics, observability enhances system resilience and facilitates proactive incident response.

To address the challenges of change management in large-scale cloud systems, Yan et al. [89] introduce *Aegis*, an end-to-end analytical service for attributing the impact of control plane changes across computing layers and service components. *Aegis* helps mitigate the risks of misconfigurations by alerting service teams and recommending the suspension of potentially disruptive changes, thereby improving system stability.

Enhancing observability within Kubernetes clusters, Mart et al. [51] integrate automated anomaly detection and alerting mechanisms with Prometheus, an open-source monitoring system. Their approach enables real-time monitoring of cluster performance metrics, allowing operators to detect irregularities before they escalate into critical failures. Furthermore, recent advancements in anomaly detection leverage Large Language Models (LLMs) to improve automated monitoring. Yu et al. [91] propose *MonitorAssistant*, an end-to-end anomaly detection system that utilizes LLMs to automate model configuration recommendations, achieve knowledge inheritance, and generate guided anomaly reports. Similarly, Wang et al. [84] introduce *COMET*, an innovative monitoring framework that filters out non-critical logs and utilizes an LLM for intelligent keyword extraction. This approach reduces noise in log data and improves the efficiency of anomaly detection, making it easier to pinpoint critical system issues.

6 Research Challenges and Future Directions

In this section, following the literature review and taxonomy of testing COS, we answer RQ3 by identifying research gaps and challenges that could provide insights on future research directions.

6.1 Research Challenges and Gaps

Despite the growing body of research on testing COS, several critical challenges and research gaps remain unaddressed.

A key issue is the lack of a **standardized benchmarking suite** for systematically evaluating testing techniques across diverse deployment scenarios. While many studies introduce valuable frameworks (*e.g.*, *COFFEE*, *Kube-burner*, and *MonitorAssistant*) and methodologies, there is currently no universally accepted benchmarking suite that enables a comprehensive comparison of testing techniques across diverse deployment scenarios. The lack of standardized benchmarking methodologies obstructs repeatability and limits the ability to assess the effectiveness of different testing strategies in varied environments.

One of the fundamental challenges in COS testing is the **verification of large-scale, highly dynamic systems**. Current formal verification models face state-space explosion, lacking the scalability to handle dynamic scaling operations and large-scale COS deployments with frequent state changes. As large-scale COS dynamically adjust workloads, node configurations, and networking policies in real time, existing verification approaches struggle to scale efficiently, leaving a gap in ensuring the correctness of orchestration behaviors under real-world conditions.

Another significant gap lies in **real-time adaptability of testing methodologies**. Current testing approaches (*e.g.*, static analysis or chaos engineering) primarily rely on predefined scenarios rather than dynamically adjusting to evolving workloads and system behaviors (**O-6, O-12**). The development of adaptive testing frameworks, which are capable of evolving alongside the system based on real-time telemetry and observability insights, remains an open research challenge.

Additionally, existing research **mainly focuses on testing microservices and applications** deployed on COS, rather than the orchestration system itself (**O-7, O-14**). While studies like *Sieve* and *Anvil* examine aspects of COS control-plane reliability (**O-4**), the majority of testing frameworks prioritize workload resilience rather than the robustness of the core components. Given that the orchestration layer is the backbone of modern cloud-native architectures, its failure could lead to widespread system disruptions. However, there is a notable lack of dedicated research on systematically testing COS components.

Finally, existing COS testing frameworks predominantly focus on traditional cloud environments, with limited research addressing **multi-cloud, hybrid, and edge computing contexts**. These environments introduce unique challenges, including heterogeneous resource constraints, intermittent connectivity, and decentralized orchestration, which differ significantly from centralized cloud infrastructures (**O-10**), leaving a critical gap in validating COS resilience and efficiency across diverse infrastructures.

6.2 Future Directions

Addressing the research gaps requires the development of scalable, adaptive, and cross-environment testing methodologies.

A promising direction involves leveraging **AI-driven adaptive testing techniques**. AI-powered methods, such as ML-based anomaly detection and automated test generation, can dynamically adapt to evolving system states. Zhong et al. [94] discuss the potential of ML-based container orchestration technologies. Meanwhile, AI-related approaches have been an emerging trend (**O-15, O-17**) and have proven their effectiveness in various scenarios, including misconfiguration detection [40] and root-cause analysis [20, 65].

Moreover, to **expand testing subjects and scenarios**, a stronger emphasis on testing COS components is essential to enhance overall system reliability. Future research can explore systematic testing methodologies for scheduler behavior, API server interactions, controller-manager resilience, and cluster-wide networking components. Meanwhile, as cloud computing architectures continue to evolve, future work should expand testing methodologies beyond traditional cloud environments. Research on COS testing for multi-cloud, hybrid-cloud, edge computing, and serverless platforms is still in its early stages.

7 Conclusion

In this work, we have identified key testing objectives of container orchestration systems (COS) by analyzing the system architecture. Moreover, we have conducted a comprehensive literature review on how container orchestration systems and containerized services under their orchestration are tested from different testing objectives in the state-of-the-art studies. Our findings highlight important research gaps, particularly in scalability, adaptability, and testing COS components themselves. To address these challenges, we outlined future directions that could help improve COS testing approaches. Advancing these methodologies will play a vital role in enhancing security, stability, and scalability, ultimately strengthening the reliability of containerized and cloud-native ecosystems.

References

[1] ALMARAZ-RIVERA, J. G. An anomaly-based detection system for monitoring kubernetes infrastructures. *IEEE Latin America Transactions 21*, 3 (2023), 457–465.

[2] ALY, A., FAYEZ, M., AL-QUTT, M., AND HAMAD, A. M. Multi-class threat detection using neural network and machine learning approaches in kubernetes environments. In *2024 6th International Conference on Computing and Informatics (ICCI)* (2024), IEEE, pp. 103–108.

[3] ANDREADIS, G., VERSLUIS, L., MASTENBROEK, F., AND IOSUP, A. A reference architecture for datacenter scheduling: design, validation, and experiments. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018* (2018), IEEE / ACM, pp. 37:1–37:15.

[4] ASTYRAKAKIS, N., NIKOLOUDAKIS, Y., KEFALOUKOS, I., SKIANIS, C., PALLIS, E., AND MARKAKIS, E. K. Cloud-native application validation & stress testing through a framework for auto-cluster deployment. In *24th IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks, CAMAD 2019, Limassol, Cyprus, September 11-13, 2019* (2019), IEEE, pp. 1–5.

[5] BACHIEGA, N. G., SOUZA, P. S. L., BRUSCHI, S. M., AND DO RÓCIO SENGER DE SOUZA, S. Container-based performance evaluation: A survey and challenges. In *2018 IEEE International Conference on Cloud Engineering, IC2E 2018, Orlando, FL, USA, April 17-20, 2018* (2018), A. Chandra, J. Li, Y. Cai, and T. Guo, Eds., IEEE Computer Society, pp. 398–403.

[6] BARLETTA, M., CINQUE, M., MARTINO, C. D., KALBARCZYK, Z. T., AND IYER, R. K. Mutiny! how does kubernetes fail, and what can we do about it? In *54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2024, Brisbane, Australia, June 24-27, 2024* (2024), IEEE, pp. 1–14.

[7] BASIRI, A., BEHNAM, N., DE ROOIJ, R., HOCHSTEIN, L., KOSEWSKI, L., REYNOLDS, J., AND ROSENTHAL, C. Chaos engineering. *IEEE Softw. 33*, 3 (2016), 35–41.

[8] BERTOLINO, A., ANGELIS, G. D., GALLEGO, M., GARCÍA, B., GORTÁZAR, F., LONETTI, F., AND MARCHETTI, E. A systematic review on cloud testing. *ACM Comput. Surv. 52*, 5 (2019), 93:1–93:42.

[9] BHARDWAJ, A. K., DUTTA, P., AND CHINTALE, P. Ai-powered anomaly detection for kubernetes security: A systematic approach to identifying threats. *Babylonian Journal of Machine Learning 2024* (2024), 142–148.

[10] BURNS, B., GRANT, B., OPPENHEIMER, D., BREWER, E. A., AND WILKES, J. Borg, omega, and kubernetes. *Commun. ACM 59*, 5 (2016), 50–57.

[11] CAMACHO, C., CAÑIZARES, P. C., LLANA, L., AND NÚÑEZ, A. Chaos as a software product line - A platform for improving open hybrid-cloud systems resiliency. *Softw. Pract. Exp. 52*, 7 (2022), 1581–1614.

[12] CANNAVACCIUOLO, C., AND MARIANI, L. Automatic generation of smoke test suites for kubernetes. In *ISSTA '22: 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, South Korea, July 18 - 22, 2022* (2022), S. Ryu and Y. Smaragdakis, Eds., ACM, pp. 769–772.

[13] CAO, C., BLAISE, A., VERWER, S., AND REBECCHI, F. Learning state machines to monitor and detect anomalies on a kubernetes cluster. In *ARES 2022: The 17th International Conference on Availability, Reliability and Security, Vienna,Austria, August 23 - 26, 2022* (2022), ACM, pp. 117:1–117:9.

[14] CASALICCHIO, E. Container orchestration: A survey. In *Systems Modeling: Methodologies and Tools*, A. Puliafito and K. S. Trivedi, Eds. Springer, 2019, pp. 221–235.

[15] CASALICCHIO, E., AND IANNUCCI, S. The state-of-the-art in container technologies: Application, orchestration and security. *Concurr. Comput. Pract. Exp. 32*, 17 (2020).

[16] CAVALLI, A. R., HIGASHINO, T., AND NÚÑEZ, M. A survey on formal active and passive testing with applications to the cloud. *Ann. des Télécommunications 70*, 3-4 (2015), 85–93.

[17] CHAOSMESHAUTHORS. Chaosmesh.

[18] CHEN, H., DOU, W., WANG, D., AND QIN, F. Cofi: Consistency-guided fault injection for cloud systems. In *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020* (2020), IEEE, pp. 536–547.

[19] CHEN, Y., MA, F., ZHOU, Y., GU, M., LIAO, Q., AND JIANG, Y. Chronos: Finding timeout bugs in practical distributed systems by deep-priority fuzzing with transient delay. In *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024* (2024), IEEE, pp. 1939–1955.

[20] CHEN, Y., XIE, H., MA, M., KANG, Y., GAO, X., SHI, L., CAO, Y., GAO, X., FAN, H., WEN, M., ZENG, J., GHOSH, S., ZHANG, X., ZHANG, C., LIN, Q., RAJMOHAN, S., ZHANG, D., AND XU, T. Automatic root cause analysis via large language models for cloud incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems, EuroSys 2024, Athens, Greece, April 22-25, 2024* (2024), ACM, pp. 674–688.

[21] CILIC, I., KRIVIC, P., ZARKO, I. P., AND KUSEK, M. Performance evaluation of container orchestration tools in edge computing environments. *Sensors 23*, 8 (2023), 4008.

[22] CLOUDNATIVECOMPUTINGFOUNDATION. Cloud native computing foundation annual survey 2023.

[23] DELL'IMMAGINE, G., SOLDANI, J., AND BROGI, A. Kubehound: Detecting microservices' security smells in kubernetes deployments. *Future Internet 15*, 7 (2023), 228.

[24] DOCKER. Docker swarm.

[25] DROSOS, G., SOTIROPOULOS, T., ALEXOPOULOS, G., MITROPOULOS, D., AND SU, Z. When your infrastructure is a buggy program: Understanding faults in infrastructure as code ecosystems. *Proc. ACM Program. Lang. 8*, OOPSLA2 (2024), 2490–2520.

[26] FELTER, W., FERREIRA, A., RAJAMONY, R., AND RUBIO, J. An updated performance comparison of virtual machines and linux containers. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2015, Philadelphia, PA, USA, March 29-31, 2015* (2015), IEEE Computer Society, pp. 171–172.

[27] FERNANDEZ, G. P., AND BRITO, A. Secure container orchestration in the cloud: policies and implementation. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 8-12, 2019* (2019), C. Hung and G. A. Papadopoulos, Eds., ACM, pp. 138–145.

[28] FLORA, J., GONÇALVES, P., TEIXEIRA, M., AND ANTUNES, N. A study on the aging and fault tolerance of microservices in kubernetes. *IEEE Access 10* (2022), 132786–132799.

[29] GOODARZY, S., NAZARI, M., HAN, R., KELLER, E., AND ROZNER, E. Resource management in cloud computing using machine learning: A survey. In *19th IEEE International Conference on Machine Learning and Applications, ICMLA 2020, Miami, FL, USA, December 14-17, 2020* (2020), M. A. Wani, F. Luo, X. A. Li, D. Dou, and F. Bonchi, Eds., IEEE, pp. 811–816.

[30] GU, J. T., SUN, X., ZHANG, W., JIANG, Y., WANG, C., VAZIRI, M., LEGUNSEN, O., AND XU, T. Acto: Automatic end-to-end testing for operation correctness of cloud system management. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023* (2023), J. Flinn, M. I. Seltzer, P. Druschel, A. Kaufmann, and J. Mace, Eds., ACM, pp. 96–112.

[31] HIGGINS, T., JHA, D. N., AND RANJAN, R. Swarm storm: An automated chaos tool for docker swarm applications. In *Proceedings of the 33rd International Symposium on High-Performance Parallel and Distributed Computing, HPDC 2024, Pisa, Italy, June 3-7, 2024* (2024), P. Dazzi, G. Mencagli, D. K. Lowenthal, and R. M. Badia, Eds., ACM, pp. 367–369.

[32] IKEUCHI, H., GE, J., MATSUO, Y., AND WATANABE, K. A framework for automatic failure recovery in ICT systems by deep reinforcement learning. In *40th IEEE International Conference on Distributed Computing Systems, ICDCS 2020, Singapore, November 29 - December 1, 2020* (2020), IEEE, pp. 1310–1315.

[33] JAWARNEH, I. M. A., BELLAVISTA, P., BOSI, F., FOSCHINI, L., MARTUSCELLI, G., MONTANARI, R., AND PALOPOLI, A. Container orchestration engines: A thorough functional and performance comparison. In *2019 IEEE International Conference on Communications, ICC 2019, Shanghai, China, May 20-24, 2019* (2019), IEEE, pp. 1–6.

[34] KAMIENIARZ, K., AND MAZURCZYK, W. A comparative study on the security of kubernetes deployments. In *International Wireless Communications and Mobile Computing, IWCMC 2024, Ayia Napa, Cyprus, May 27-31, 2024* (2024), IEEE, pp. 718–723.

[35] KHAN, A. Key characteristics of a container orchestration platform to enable a modern application. *IEEE Cloud Comput. 4*, 5 (2017), 42–48.

[36] KITCHENHAM, B. Procedures for performing systematic reviews. *Keele, UK, Keele University 33*, 2004 (2004), 1–26.

[37] KRATZKE, N. About microservices, containers and their underestimated impact on network performance. *CoRR abs/1710.04049* (2017).

[38] KUDO, R., KITAHARA, H., GAJANANAN, K., AND WATANABE, Y. Application integrity protection on kubernetes cluster based on manifest signature verification. *J. Inf. Process. 30* (2022), 626–635.

[39] LEDENEV, A. Pumba.

[40] LIAN, X., CHEN, Y., CHENG, R., HUANG, J., THAKKAR, P., ZHANG, M., AND XU, T. Large language models as configuration validators. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)* (2024), IEEE Computer Society, pp. 204–216.

[41] LINKHORST, M. Chaoskube.

[42] LIU, B., KHERADMAND, A., CAESAR, M., AND GODFREY, P. B. Towards verified self-driving infrastructure. In *HotNets '20: The 19th ACM Workshop on Hot Topics in Networks, Virtual Event, USA, November 4-6, 2020* (2020), B. Y. Zhao, H. Zheng, H. V. Madhyastha, and V. N. Padmanabhan, Eds., ACM, pp. 96–102.

[43] LIU, B., LIM, G., BECKETT, R., AND GODFREY, P. B. Kivi: Verification for cluster management. In *Proceedings of the 2024 USENIX Annual Technical Conference, USENIX ATC 2024, Santa Clara, CA, USA, July 10-12, 2024* (2024), S. Bagchi and Y. Zhang, Eds., USENIX Association, pp. 509–527.

[44] LIU, H., WANG, X., LI, G., LU, S., YE, F., AND TIAN, C. Fcatch: Automatically detecting time-of-fault bugs in cloud systems. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2018, Williamsburg, VA, USA, March 24-28, 2018* (2018), X. Shen, J. Tuck, R. Bianchini, and V. Sarkar, Eds., ACM, pp. 419–431.

[45] LU, J., LIU, C., LI, L., FENG, X., TAN, F., YANG, J., AND YOU, L. Crashtuner: detecting crash-recovery bugs in cloud systems via meta-info analysis. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles, SOSP 2019, Huntsville, ON, Canada, October 27-30, 2019* (2019), T. Brecht and C. Williamson, Eds., ACM, pp. 114–130.

[46] MAHAJAN, A., AND BENSON, T. A. Suture: Stitching safety onto kubernetes operators. In *CoNEXT'20: Proceedings of the Student Workshop, Barcelona, Spain, December 1, 2020* (2020), ACM, pp. 19–20.

[47] MAHAVAISHNAVI, V., SAMINATHAN, R., AND PRITHVIRAJ, R. Secure container orchestration: A framework for detecting and mitigating orchestrator-level vulnerabilities. *Multimedia Tools and Applications* (2024), 1–21.

[48] MAJUMDAR, R., AND NIKSIC, F. Why is random testing effective for partition tolerance bugs? *Proc. ACM Program. Lang. 2*, POPL (2018), 46:1–46:24.

[49] MALHOTRA, R., BANSAL, A., AND KESSENTINI, M. A systematic literature review on maintenance of software containers. *ACM Comput. Surv. 56*, 8 (2024), 193:1–193:38.

[50] MALLENI, S. S., CANAVATE, R. S., AND CHALLA, V. Into the fire: Delving into kubernetes performance and scale with kube-burner. In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering, ICPE 2024, London, United Kingdom, May 7-11, 2024* (2024), S. Balsamo, W. J. Knottenbelt, C. L. Abad, and W. Shang, Eds., ACM, pp. 89–90.

[51] MART, O., NEGRU, C., POP, F., AND CASTIGLIONE, A. Observability in kubernetes cluster: Automatic anomalies detection using prometheus. In *22nd IEEE International Conference on High Performance Computing and Communications; 18th IEEE International Conference on Smart City; 6th IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2020, Yanuca Island, Cuvu, Fiji, December 14-16, 2020* (2020), IEEE, pp. 565–570.

[52] MENG, R., PÎRLEA, G., ROYCHOUDHURY, A., AND SERGEY, I. Greybox fuzzing of distributed systems. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023* (2023), W. Meng, C. D. Jensen, C. Cremers, and E. Kirda, Eds., ACM, pp. 1615–1629.

[53] NACHIYAPPAN, S., AND JUSTUS, S. Cloud testing tools and its challenges: A comparative study. *procedia computer Science 50* (2015), 482–489.

[54] NASCIMENTO, B., SANTOS, R., HENRIQUES, J., BERNARDO, M. V., AND CALDEIRA, F. Availability, scalability, and security in the migration from container-based to cloud-native applications. *Comput. 13*, 8 (2024), 192.

[55] NIKOLAIDIS, F., CHAZAPIS, A., MARAZAKIS, M., AND BILAS, A. Frisbee: automated testing of cloud-native applications in kubernetes.

[56] NIKOLAIDIS, F., CHAZAPIS, A., MARAZAKIS, M., AND BILAS, A. Event-driven chaos testing for containerized applications. In *High Performance Computing - ISC High Performance 2023 International Workshops, Hamburg, Germany, May 21-25, 2023, Revised Selected Papers* (2023), A. Bienz, M. Weiland, M. Baboulin, and C. Kruse, Eds., vol. 13999 of *Lecture Notes in Computer Science*, Springer, pp. 144–157.

[57] PAHL, C., BROGI, A., SOLDANI, J., AND JAMSHIDI, P. Cloud container technologies: A state-of-the-art review. *IEEE Trans. Cloud Comput. 7*, 3 (2019), 677–692.

[58] PAN, W., LI, S., LI, F., ZHANG, J., AND FANG, C. Performance analysis of kubernetes job scheduling model based on queuing theory. In *2024 IEEE 2nd International Conference on Sensors, Electronics and Computer Engineering (ICSECE)* (2024), IEEE, pp. 592–595.

[59] PAN, Y., CHEN, I., BRASILEIRO, F. V., JAYAPUTERA, G. T., AND SINNOTT, R. O. A performance comparison of cloud-based container orchestration tools. In *2019 IEEE International Conference on Big Knowledge, ICBK 2019, Beijing, China, November 10-11, 2019* (2019), Y. Gao, R. Möller, X. Wu, and R. Kotagiri, Eds., IEEE, pp. 191–198.

[60] PURAHONG, B., SITHIYOPASAKUL, J., SITHIYOPASAKUL, P., LASAKUL, A., AND BENJANGKAPRASERT, C. Automated resource management system based upon container orchestration tools comparison. *Journal of Advances in Information Technology 14*, 3 (2023).

[61] QUEIROZ, R., CRUZ, T., MENDES, J., SOUSA, P., AND SIMÕES, P. Container-based virtualization for real-time industrial systems - A systematic review. *ACM Comput. Surv. 56*, 3 (2024), 59:1–59:38.

[62] RAHMAN, A., SHAMIM, S. I., BOSE, D. B., AND PANDITA, R. Security misconfigurations in open source kubernetes manifests: An empirical study. *ACM Trans. Softw. Eng. Methodol. 32*, 4 (2023), 99:1–99:36.

[63] REILE, C., CHADHA, M., HAUNER, V., JINDAL, A., HOFMANN, B., AND GERNDT, M. Bunk8s: Enabling easy integration testing of microservices in kubernetes. In *IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2022, Honolulu, HI, USA, March 15-18, 2022* (2022), IEEE, pp. 459–463.

[64] RODRIGUEZ, M. A., AND BUYYA, R. Container-based cluster orchestration systems: A taxonomy and future directions. *Softw. Pract. Exp. 49*, 5 (2019), 698–719.

[65] SARIKA, P. K., BADAMPUDI, D., JOSYULA, S. P., AND USMAN, M. Automating microservices test failure analysis using kubernetes cluster logs. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering, EASE 2023, Oulu, Finland, June 14-16, 2023* (2023), ACM, pp. 192–195.

[66] SHUKLA, S. Streamlining integration testing with test containers: Addressing limitations and best practices for implementation. *Inter. J. Latest Engg. Manag. Res.(IJLEMR) 9* (2023), 19–26.

[67] SIMONSSON, J., ZHANG, L., MORIN, B., BAUDRY, B., AND MONPERRUS, M. Observability and chaos engineering on system calls for containerized applications in docker. *Future Gener. Comput. Syst. 122* (2021), 117–129.

[68] SOBTI, A. Kubemonkey.

[69] SOLAYMAN, H. E., AND QASHA, R. P. On the use of container-based virtualisation for iot provisioning and orchestration: a survey. *Int. J. Comput. Sci. Math. 18*, 4 (2023), 299–311.

[70] STRAESSER, M., HAAS, P., FRANK, S., HAKAMIAN, M. A., VAN HOORN, A., AND KOUNEV, S. Kubernetes-in-the-loop: Enriching microservice simulation through authentic container orchestration. In *Performance Evaluation Methodologies and Tools - 16th EAI International Conference, VALUETOOLS 2023, Crete, Greece, September 6-7, 2023, Proceedings* (2023), E. Kalyvianaki and M. Paolieri, Eds., vol. 539 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Springer, pp. 82–98.

[71] STRAESSER, M., MATHIASCH, J., BAUER, A., AND KOUNEV, S. A systematic approach for benchmarking of container orchestration frameworks. In *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering, ICPE 2023, Coimbra, Portugal, April 15-19, 2023* (2023), M. Vieira, V. Cardellini, A. D. Marco, and P. Tuma, Eds., ACM, pp. 187–198.

[72] SULTAN, S., AHMAD, I., AND DIMITRIOU, T. Container security: Issues, challenges, and the road ahead. *IEEE Access 7* (2019), 52976–52996.

[73] SUN, X., LUO, W., GU, J. T., GANESAN, A., ALAGAPPAN, R., GASCH, M., SURESH, L., AND XU, T. Automatic reliability testing for cluster management controllers. In *16th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2022, Carlsbad, CA, USA, July 11-13, 2022* (2022), M. K. Aguilera and H. Weatherspoon, Eds., USENIX Association, pp. 143–159.

[74] SUN, X., MA, W., GU, J. T., MA, Z., CHAJED, T., HOWELL, J., LATTUADA, A., PADON, O., SURESH, L., SZEKERES, A., AND XU, T. Anvil: Verifying liveness of cluster management controllers. In *18th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2024, Santa Clara, CA, USA, July 10-12, 2024* (2024), A. Gavrilovska and D. B. Terry, Eds., USENIX Association, pp. 649–666.

[75] SUN, X., SURESH, L., GANESAN, A., ALAGAPPAN, R., GASCH, M., TANG, L., AND XU, T. Reasoning about modern datacenter infrastructures using partial histories. In *HotOS '21: Workshop on Hot Topics in Operating Systems, Ann Arbor, Michigan, USA, June, 1-3, 2021* (2021), S. Angel, B. Kasikci, and E. Kohler, Eds., ACM, pp. 213–220.

[76] TANG, L., BHANDARI, C., ZHANG, Y., KARANIKA, A., JI, S., GUPTA, I., AND XU, T. Fail through the cracks: Cross-system interaction failures in modern cloud systems. In *Proceedings of the Eighteenth European Conference on Computer Systems, EuroSys 2023, Rome, Italy, May 8-12, 2023* (2023), G. A. D. Luna, L. Querzoni, A. Fedorova, and D. Narayanan, Eds., ACM, pp. 433–451.

[77] TIEN, C.-W., HUANG, T.-Y., TIEN, C.-W., HUANG, T.-C., AND KUO, S.-Y. Kubanomaly: Anomaly detection for the docker orchestration platform with neural network approaches. *Engineering reports 1*, 5 (2019), e12080.

[78] TIMONEN, S., SROOR, M., MOHANANI, R., AND MIKKONEN, T. Anomaly detection through container testing: A survey of company practices. In *Product-Focused Software Process Improvement - 24th International Conference, PROFES 2023, Dornbirn, Austria, December 10-13, 2023, Proceedings, Part I* (2023), R. Kadgien, A. Jedlitschka, A. Janes, V. Lenarduzzi, and X. Li, Eds., vol. 14483 of *Lecture Notes in Computer Science*, Springer, pp. 363–378.

[79] TORKURA, K. A., SUKMANA, M. I. H., CHENG, F., AND MEINEL, C. Cloudstrike: Chaos engineering for security and resiliency in cloud infrastructure. *IEEE Access 8* (2020), 123044–123060.

[80] TURIN, G., BORGARELLI, A., DONETTI, S., JOHNSEN, E. B., TARIFA, S. L. T., AND DAMIANI, F. A formal model of the kubernetes container framework. In *Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part I* (2020), T. Margaria and B. Steffen, Eds., vol. 12476 of *Lecture Notes in Computer Science*, Springer, pp. 558–577.

[81] VERDERAME, L., CAVIGLIONE, L., CARBONE, R., AND MERLO, A. Secco: Automated services to secure containers in the devops paradigm. In *Proceedings of the 2023 International Conference on Research in Adaptive and Convergent Systems, RACS 2023, Gdansk, Poland, August 6-10, 2023* (2023), ACM, pp. 10:1–10:6.

[82] VOIEVODIN, Y., ROZLOMII, I., AND YARMILKO, A. Approach to evaluate scheduling strategies in container orchestration systems. In *Modeling, Control and Information Technologies: Proceedings of International scientific and practical conference* (2023), no. 6, pp. 292–295.

[83] VOIEVODIN, Y. V., AND ROZLOMII, I. O. Advanced software framework for comparing balancing strategies in container orchestration systems. In *Proceedings of the 4th Edge Computing Workshop (doors 2024), Zhytomyr, Ukraine, April 5, 2024* (2024), T. A. Vakaliuk and S. O. Semerikov, Eds., vol. 3666 of *CEUR Workshop Proceedings*, CEUR-WS.org, pp. 60–69.

[84] WANG, Z., LI, J., MA, M., LI, Z., KANG, Y., ZHANG, C., BANSAL, C., CHINTALAPATI, M., RAJMOHAN, S., LIN, Q., ZHANG, D., PEI, C., AND XIE, G. Large language models can provide accurate and interpretable incident triage. In *35th IEEE International Symposium on Software Reliability Engineering, ISSRE 2024, Tsukuba, Japan, October 28-31, 2024* (2024), IEEE, pp. 523–534.

[85] WATADA, J., ROY, A., KADIKAR, R., PHAM, H., AND XU, B. Emerging trends, techniques and open issues of containerization: A review. *IEEE Access 7* (2019), 152443–152472.

[86] WEERASIRI, D., BARUKH, M. C., BENATALLAH, B., SHENG, Q. Z., AND RANJAN, R. A taxonomy and survey of cloud resource orchestration techniques. *ACM Comput. Surv. 50*, 2 (2017), 26:1–26:41.

[87] WOHLIN, C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14, London, England, United Kingdom, May 13-14, 2014* (2014), M. J. Shepperd, T. Hall, and I. Myrtveit, Eds., ACM, pp. 38:1–38:10.

[88] XU, Q., GAO, Y., AND WEI, J. An empirical study on kubernetes operator bugs. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2024, Vienna, Austria, September 16-20, 2024* (2024), M. Christakis and M. Pradel, Eds., ACM, pp. 1746–1758.

[89] YAN, X., HSIEH, K., LIYANAGE, Y., MA, M., CHINTALAPATI, M., LIN, Q., DANG, Y., AND ZHANG, D. Aegis: Attribution of control plane change impact across layers and components for cloud systems. In *45th IEEE/ACM International Conference on Software Engineering: Software Engineering in Practice, SEIP@ICSE 2023, Melbourne, Australia, May 14-20, 2023* (2023), IEEE, pp. 222–233.

[90] YANG, Y., SHEN, W., RUAN, B., LIU, W., AND REN, K. Security challenges in the container cloud. In *3rd IEEE International Conference on Trust, Privacy and Security in Intelligent Systems and Applications, TPS-ISA 2021, Atlanta, GA, USA, December 13-15, 2021* (2021), IEEE, pp. 137–145.

[91] YU, Z., MA, M., ZHANG, C., QIN, S., KANG, Y., BANSAL, C., RAJMOHAN, S., DANG, Y., PEI, C., PEI, D., LIN, Q., AND ZHANG, D. Monitorassistant: Simplifying cloud service monitoring via large language models. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, FSE 2024, Porto de Galinhas, Brazil, July 15-19, 2024* (2024), M. d'Amorim, Ed., ACM, pp. 38–49.

[92] ZHANG, L., MORIN, B., BAUDRY, B., AND MONPERRUS, M. Maximizing error injection realism for chaos engineering with system calls. *IEEE Trans. Dependable Secur. Comput. 19*, 4 (2022), 2695–2708.

[93] ZHENG, T., TANG, R., CHEN, X., AND SHEN, C. Kubefuzzer: Automating restful api vulnerability detection in kubernetes. *Computers, Materials & Continua 81*, 1 (2024).

[94] ZHONG, Z., XU, M., RODRIGUEZ, M. A., XU, C., AND BUYYA, R. Machine learning-based orchestration of containers: A taxonomy and future directions. *ACM Comput. Surv. 54*, 10s (2022), 217:1–217:35.

[95] ZHU, H., AND GEHRMANN, C. Kub-sec, an automatic kubernetes cluster apparmor profile generation engine. In *14th International Conference on COMmunication Systems & NETworkS, COMSNETS 2022, Bangalore, India, January 4-8, 2022* (2022), IEEE, pp. 129–137.