iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 12 **Issue:** X **Month of publication:** October 2024

DOI: https://doi.org/10.22214/ijraset.2024.64556

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue X Oct 2024- Available at www.ijraset.com

Enhancing Kubernetes Observability: A Synthetic Testing Approach for Improved Impact Analysis

Naresh Kumar Amrutham
State University of New York at Buffalo

Abstract: ***Kubernetes has become the cornerstone for deploying and managing containerized applications. Its dynamic and distributed nature, while offering scalability and resilience, introduces significant complexities in observability, monitoring, and impact analysis. Traditional monitoring solutions often fall short in providing visibility into the functionalities affected by interruptions in the Kubernetes platform, leading to prolonged downtime and impact analysis.***

This paper investigates the challenges associated with achieving effective observability for Kubernetes environments due to their distributed architecture. We highlight the limitations of relying solely on conventional metrics, logs, and events to understand the impact of system interruptions. To address these challenges, we propose an integrated approach that combines synthetic testing with existing observability tools. By executing synthetic transactions and simulating user interactions with the Kubernetes platform across various use cases, this method proactively evaluates the system's performance and detects issues that passive monitoring might overlook. This comprehensive view enables faster identification of impacted functionalities. Through practical experiments, we demonstrate how incorporating synthetic testing improves the detection of service degradations and supports more effective troubleshooting strategies.

Keywords: Kubernetes, Containerized applications, Observability, Monitoring, Synthetic tests

I. INTRODUCTION

Kubernetes, an open-source container orchestration platform initially developed by Google and now maintained by the Cloud Native Computing Foundation (CNCF) [1], designed for automating deployment, scaling and management of containerized applications. While Kubernetes offers significant benefits in managing complex, distributed systems, its dynamic nature introduces new challenges, particularly in the areas of observability, monitoring, and impact analysis. These complexities arise from the distributed architecture and the rapid, automated changes inherent in Kubernetes environments, necessitating advanced strategies for maintaining visibility and control over system behavior. One of the primary challenges in Kubernetes observability is identifying and understanding functionality degradation or failure, whether caused by internal system issues or external factors. Interruptions can stem from various sources, such as node failures, network problems, misconfigurations, system upgrades, core component failures, etc. Given the distributed nature of Kubernetes, these interruptions may not manifest immediately or may impact services in non-obvious ways. Relying solely on traditional monitoring tools that focus on individual components or services can lead to gaps in understanding how these issues affect the end-to-end user experience. Consequently, there's a critical need for more sophisticated observability solutions that can provide a holistic view of the Kubernetes ecosystem and accurately pinpoint the root causes and impacts of various interruptions. While existing research has focused on traditional monitoring methods and log analysis in Kubernetes environments, there is a notable gap in leveraging proactive, synthetic testing approaches to enhance observability and impact analysis in production Kubernetes clusters. To address these challenges, there is a need for enhanced observability strategies that provide deeper insights into system behavior and impact analysis. This paper proposes an integrated approach that combines synthetic testing with existing observability tools that are based on metrics, logs, and events. Synthetic testing involves creating and running scripted transactions that simulate typical user interactions with the application [2]. In this case, we use synthetic testing to ensure various functionalities of the Kubernetes platform are functioning as expected. By executing these tests continuously and systematically across various use cases, it is possible to proactively monitor the platform’s functionality and at the same time provide better impact analysis.

II. BACKGROUND

A. Core Kubernetes Components

Kubernetes, as a complex distributed system, consists of several core components that work together to manage containerized applications. Understanding these components and their interactions is crucial for effective observability. This section provides an overview of the key Kubernetes components and their roles in the cluster architecture.

1) *API Server*: The API server is the central management entity for the entire Kubernetes cluster [3]. It exposes the Kubernetes API, processes REST operations, validates them, and updates the corresponding objects in etcd. As the primary point of interaction for cluster management, the API server's performance and availability are critical to the overall health of the cluster.
2) *etcd*: etcd is a distributed key-value store that serves as Kubernetes' backing store for all cluster data [4]. It ensures consistency and provides reliable data storage for the cluster state. The health and performance of etcd directly impact the stability of the entire Kubernetes cluster.
3) *Scheduler:* The Kubernetes scheduler is responsible for assigning newly created Pods to nodes [5]. It watches for Pods with no assigned node and selects an appropriate node for them to run on, considering factors such as resource requirements, hardware/software/policy constraints, and affinity/anti-affinity specifications.
4) *Controller Manager:* The Controller Manager runs controller processes that regulate the state of the system, attempting to move the current state towards the desired state [6]. It includes controllers for various Kubernetes resources such as ReplicaSets, Deployments, and Services.
5) *Kubelet*: The Kubelet is an agent that runs on each node in the cluster. It ensures that containers are running in a Pod and reports node and Pod status to the API server. The Kubelet plays a crucial role in maintaining the health of individual nodes and the Pods running on them.

B. Current Monitoring Practices and Limitations

All the Kubernetes components provide metrics at various levels providing valuable insights into the health and performance of individual Kubernetes components, they often fall short in providing a comprehensive view of the cluster's overall health and its impact on the workloads running on the cluster. Traditional observability in Kubernetes involves monitoring based on metrics generated by the Kubernetes control plane, complemented by logs and events that offer additional perspectives into the behavior of the Kubernetes platform [7], as shown in Fig: 1.

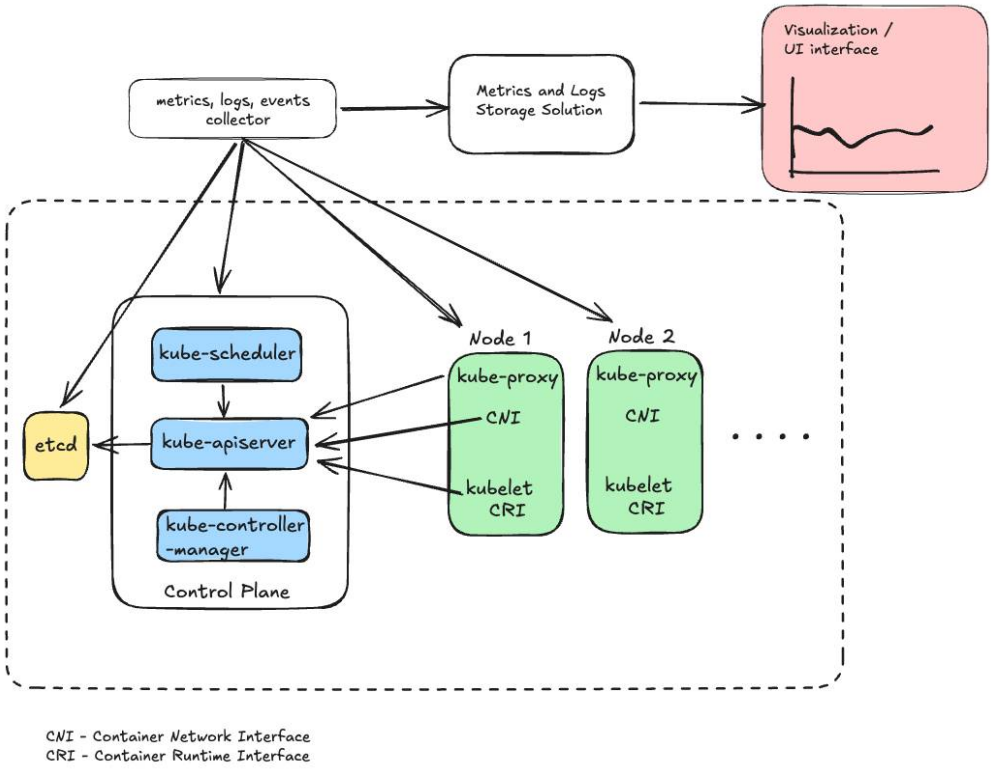


Fig. 1 Shows architecture of key Kubernetes components and observability pipeline

Kubernetes components are designed to provide metrics and logging in a unified manner, facilitating easier collection and analysis. Typically, all metrics forwarded to monitoring stacks through various pipelines by leveraging collectors such as Prometheus [8], Fluent Bit [9], and Open Telemetry (OTel) Collector [10]. These data are then visualized and analyzed using popular tools such as Grafana [11], and Prometheus. However, these approaches have several limitations:

1) *Lack of End-to-End Visibility*: While component-level metrics are useful, they often fail to capture the complex interactions between different parts of the system and their impact on overall application performance.
2) *Reactive Nature:* Traditional monitoring is often reactive, alerting operators only after issues have occurred and potentially impacted users.
3) *Difficulty in Impact Analysis:* It can be challenging to correlate issues across distributed components and understand their impact on specific application functionalities.
4) *Limited Proactive Testing:* Current practices typically lack mechanisms for proactively testing system functionalities in production environments.
5) *Complexity in Large-Scale Deployments:* As Kubernetes clusters grow, the volume of metrics can become overwhelming, making it difficult to identify and prioritize issues effectively.
6) *Dependency on Active Failures:* Additionally, metrics indicating a failure has occurred are good if there are actions that fail when there is a system interruption. However, if there is no activity in certain functionality, issues may only be discovered later. This limitation can lead to prolonged periods where critical system components or functionalities are non-operational without detection.

These limitations underscore the necessity for advanced observability solutions that complement traditional monitoring approaches. By integrating these advanced solutions with existing practices, organizations can achieve a more holistic view of the Kubernetes ecosystem. This comprehensive approach enables more accurate identification of root causes, facilitates better assessment of impacts on application functionalities, and enhances overall system observability.

III. INTRODUCTION TO SYNTHETIC TESTING IN OBSERVABILITY

Synthetic testing involves the simulation of key transactions and functionalities on a regular basis to ensure they are in optimal working condition. This proactive approach to monitoring is designed to identify potential issues before they impact end-users, providing a consistent and reliable method for verifying system performance and availability. Traditionally, synthetic testing has been widely used for frontend applications, where tests mimic user actions to ensure that user interfaces and workflows function correctly. These tests simulate interactions such as clicking buttons, filling out forms, and navigating through pages, providing valuable insights into the user experience and helping to identify issues that might not be apparent through backend monitoring alone. In this paper, we explore how synthetic testing can be effectively leveraged for testing the Kubernetes platform. Given the complexity and dynamic nature of Kubernetes environments, synthetic testing offers a powerful tool for ensuring that key use cases and functionalities of the platform are functioning as expected. By simulating operations such as API requests, pod scheduling, and service discovery, synthetic tests can provide a comprehensive view of the cluster's health and performance.

IV. INTRODUCTION TO KUBERNETES CONTROLLERS

A. Introduction to Kubernetes Controllers

Kubernetes controllers are a fundamental component of the Kubernetes architecture, responsible for managing the state of the cluster [12]. They are part of the control plane and work continuously to ensure that the actual state of the cluster matches the desired state specified by the user. Controllers operate as control loops, which are non-terminating loops that regulate the state of a system. They observe the current state of the cluster, compare it to the desired state, and take corrective actions to reconcile any differences. Users define the desired state of their applications and resources using Kubernetes manifests (YAML or JSON files). Controllers ensure that the resources in the cluster conform to these specifications [13].

B. Introduction to Custom Controllers in Kubernetes

Custom controllers extend the functionality of Kubernetes by allowing developers to define and manage custom resources. They enable the automation of complex operational tasks and the implementation of custom logic tailored to specific application needs. Custom controllers often work in conjunction with Custom Resource Definitions (CRDs), which allow users to define new resource types in Kubernetes. CRDs provide a way to extend the Kubernetes API with custom schemas [14].

Creating a custom controller involves several key steps:

1) *Define a Custom Resource (CR):* Use a CRD to define the schema for the custom resource you want to manage. This includes specifying the fields and data types that the resource will contain [15].
2) *Implement the Controller Logic:* Write the controller logic that will watch for changes to the custom resource and take appropriate actions. This typically involves using a client library, such as the Kubernetes client-go library, to interact with the Kubernetes API [16].

3) *Deploy the Controller:* Package the controller as a containerized application and deploy it to the Kubernetes cluster. The controller will run as a pod and continuously monitor the custom resources [17].

C. Use Cases of Custom Controllers:

Custom controllers are used in a variety of scenarios, such as:

1) *Automating Operational Tasks:* Automate complex workflows, such as backup and restore operations, scaling, and configuration management [18].
2) *Implementing Custom Business Logic:* Enforce business-specific rules and policies, such as compliance checks or custom scheduling algorithms.
3) *Enhancing Observability and Monitoring:* Integrate with monitoring tools to provide advanced observability features, such as custom alerts and metrics collection [19].

V. CASE STUDY: KEIKOPROJ ACTIVE-MONITOR CONTROLLER

A. Overview

Active-Monitor is a Kubernetes custom resource controller developed by Keikoproj, designed to enable deep cluster monitoring and self-healing capabilities [20]. It leverages Argo workflows to execute complex monitoring and remediation tasks, providing a powerful tool for enhancing Kubernetes cluster observability and reliability [21].

B. Core Functionality

1) *Custom Resource Definition (CRD)*: Active-Monitor introduces a custom resource called HealthCheck, which allows users to define various health checks for Kubernetes components. This CRD extends the Kubernetes API, enabling users to create, update, and delete health check definitions using standard Kubernetes tools.
2) *Integration with Argo Workflows:* The controller utilizes Argo workflows to execute health checks and remediation actions. This integration allows for complex, multi-step monitoring processes and automated healing procedures.

C. Test Management

Active-Monitor manages tests through the following mechanisms:

1) *Test Definition:* Users define tests using the HealthCheck CRD, specifying the target component, test parameters, and expected outcomes.
2) *Scheduling:* The controller schedules tests based on user-defined intervals or triggers, ensuring regular health checks of the cluster.
3) *Execution:* Tests are executed as Argo workflows, allowing for parallel execution and complex test scenarios.
4) *Result Processing:* The controller processes test results, updating the status of the HealthCheck resource and triggering alerts or remediation actions as necessary.

D. Exposing Metrics

The Active-Monitor Controller exposes metrics that provide insights into the health and performance of the monitored components. These metrics are typically integrated with observability tools like Prometheus and Grafana, allowing for real-time monitoring and visualization. Key metrics include:

1) *Health Check Status:* Tracks the successful and errored health checks.
- healthcheck_success_count: The total number of successful healthcheck resources
- healthcheck_error_count: The total number of errored healthcheck resources
- healthcheck_runtime_seconds: Time taken for the workflow to complete.

2) *Health Check Runtime*: Captures the time taken for the workflow to complete.

E. Use Cases

A wide range of tests can be run as part of this controller, such as:

- Functionality testing of core Kubernetes components
- Smoke testing for basic functionality verification
- Post incident verification tests
- New cluster readiness validation
- Other functionalities such as image pull check, DNS resolution, network ACLs validation, etc.

The following is an example of a HealthCheck resource designed to verify DNS resolution within the cluster. This test runs every 60 seconds and confirms whether DNS resolutions are successful for both internal cluster domains and external domains. The results are exported as metrics in Prometheus format. These metrics can help derive health of the DNS functionality inside the clusters.

```
apiVersion: activemonitor.keikoproj.io/v1alpha1
kind: HealthCheck
metadata:
  generateName: dns-healthcheck-
  namespace: health
spec:
  repeatAfterSec: 60
  level: cluster
  description: "check pod dns resolutions"
  workflow:
    generateName: dns-workflow-
    resource:
      namespace: health
      serviceAccount: activemonitor-controller-sa
      source:
        inline: |
          apiVersion: argoproj.io/v1alpha1
          kind: Workflow
          metadata:
            labels:
              workflows.argoproj.io/controller-instanceid: activemonitor-workflows
            generateName: dns-workflow-
          spec:
            activeDeadlineSeconds: 65
            ttlSecondsAfterFinished: 60
            entrypoint: start
            templates:
            - name: start
              retryStrategy:
                limit: 3
              container:
                image: tutum/dnsutils
                command: [sh, -c]
            args: ["nslookup some-service.svc.cluster.local && nslookup www.google.com"]
```

F. High Level Architecture Design

Figure 2 illustrates the integration of the Active-Monitor controller within the Kubernetes cluster and its metrics pipelines. Operating inside the cluster, the Active-Monitor controller utilizes existing HealthCheck resource configurations to initiate tests as Argo Workflows at predetermined intervals. Upon completion of each workflow instance, the controller performs two key actions:

1) It updates the status of the corresponding HealthCheck resource.
2) It generates health check metrics in Prometheus format.

The controller is designed to be compatible with metric collection systems, allowing for seamless scraping and export of these metrics to appropriate storage solutions. These collected metrics serve a dual purpose:

a) They provide insights into specific functionalities of the monitored components.
b) When combined with standard metrics from the respective components, they enable more comprehensive analysis.

To illustrate this synergy, consider the case of DNS resolution monitoring. By merging Active-Monitor's HealthCheck status metrics (such as `healthcheck_success_count` and `healthcheck_error_count`) with conventional DNS component metrics (like latencies and error rates), administrators can derive more nuanced and actionable insights into the overall health and performance of the DNS infrastructure.

This integrated approach to metrics collection and analysis enhances the observability of the Kubernetes environment, allowing for more informed decision-making and proactive issue resolution.

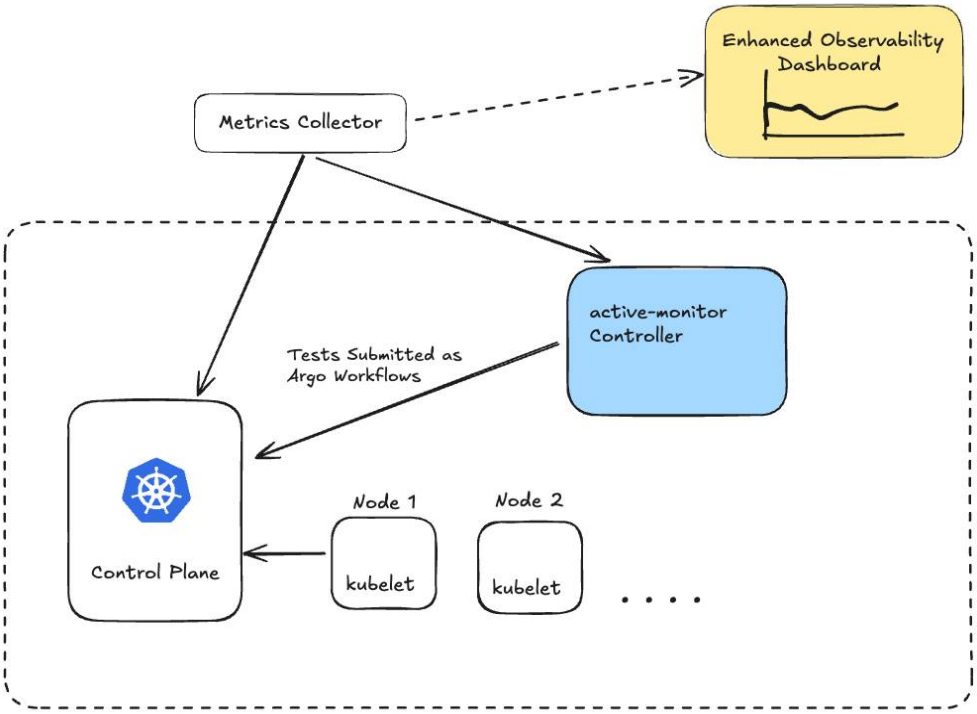


Fig. 2 Shows proposed architecture comprising of Synthetic Testing Controller along with key Kubernetes components and observability pipeline

VI. Advantages and Disadvantages of Synthetic Testing

A. Advantages

1) *Clear Identification of Degraded Functionalities:* The synthetic testing approach provides a mechanism to clearly identify which specific functionalities are degraded or have failed due to issues in any Kubernetes component. This granular insight is crucial for rapid troubleshooting and targeted remediation efforts.
2) *User-Centric Impact Assessment:* By focusing on testing functionalities that directly affect user experience, this approach enables a more accurate assessment of the overall user impact during incidents or degradations. This user-centric perspective aids in prioritizing issues based on their real-world impact.
3) *Holistic Control Plane Health Evaluation:* The data generated from synthetic tests can be aggregated to provide a comprehensive view of the Kubernetes Control Plane's health. This holistic evaluation offers a clear picture of the cluster's operational status, focusing on the functionalities that are most critical to users and applications.
4) *Proactive Issue Detection:* Regular execution of synthetic tests allows for the detection of issues before they impact real user traffic, enabling a proactive approach to cluster management and maintenance.
5) *Customizable and Extensible:* The framework allows for the creation of custom tests tailored to specific organizational needs and unique cluster configurations, providing flexibility in what aspects of the system are monitored.
6) *Objective Performance Metrics:* Synthetic tests provide consistent, reproducible results, offering objective metrics for system performance and reliability over time.
7) *Enhanced Observability:* By complementing traditional monitoring methods, synthetic testing adds another layer of observability, particularly useful for understanding the interplay between different Kubernetes components and their impact on overall functionality.

B. Disadvantages

1) *Setup and Maintenance Overhead:* Implementing and maintaining a suite of synthetic tests requires significant effort. As cluster functionalities evolve or change, tests need to be updated accordingly, adding to the ongoing maintenance burden for the operations team.
2) *Resource Cleanup Challenges:* Synthetic tests that create resources (e.g., pods, services) in the cluster need careful management to ensure proper cleanup after test execution. Failure to do so could lead to resource leaks or interference with production workloads.
3) *Resource Consumption:* Running frequent synthetic tests consumes cluster resources, which could potentially impact production workloads if not carefully managed, especially in resource-constrained environments.
4) *Potential Security Implications:* Synthetic tests that simulate user actions or interact with various cluster components may require elevated permissions, potentially introducing security risks if not properly managed.
5) *Complexity in Test Design:* Designing effective synthetic tests that accurately reflect real-world usage patterns and cover all critical functionalities can be challenging and may require specialized expertise.

C. Balancing Considerations

While the advantages of synthetic testing in Kubernetes environments are significant, particularly in terms of enhanced observability and user-centric impact assessment, the disadvantages highlight the need for careful implementation and ongoing management. Organizations adopting this approach should weigh these factors against their specific needs and resources, potentially implementing the system incrementally to manage complexity and resource demands effectively.

The benefits of clear functionality degradation identification and holistic health evaluation often outweigh the setup and maintenance challenges for many organizations, especially those operating large-scale or critical Kubernetes deployments. However, it's crucial to develop strategies to mitigate the disadvantages, such as implementing robust resource cleanup mechanisms, carefully designing test suites to minimize false results, and establishing clear processes for test maintenance and updates.

VII. CONCLUSIONS

The exploration of synthetic testing in Kubernetes environments reveals its significant potential in enhancing observability and maintaining the reliability of complex, distributed systems. Through our analysis of the Keikoproj Active-Monitor Controller, we have gained valuable insights into the practical application and benefits of synthetic testing in Kubernetes platforms.

Synthetic testing emerges as a powerful complement to traditional monitoring approaches, offering proactive and user-centric insights into system health and performance.

By simulating key transactions and functionalities, it provides a consistent and reliable method for verifying that critical Kubernetes components and services are functioning as expected. This approach is particularly valuable in the dynamic and complex landscape of containerized applications, where traditional monitoring methods may fall short.

The Keikoproj Active-Monitor Controller demonstrates how synthetic testing can be effectively implemented in Kubernetes environments. Its ability to continuously validate core functionalities, provide early detection of issues, and offer insights into user-facing operations addresses many of the observability challenges inherent in Kubernetes platforms.

However, the limitations identified, such as resource overhead and complexity in test maintenance, highlight areas for future development and research.

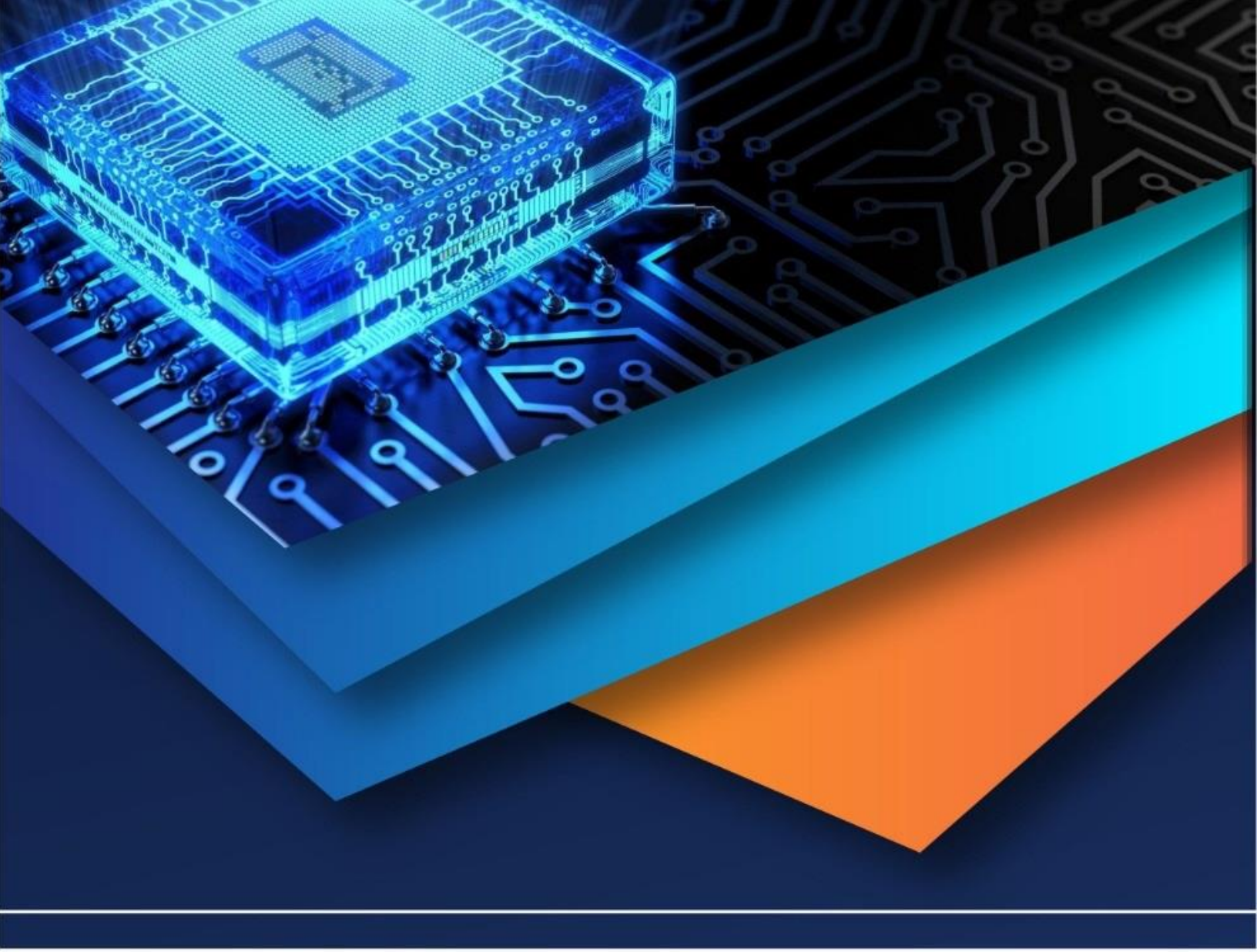
As Kubernetes continues to evolve and grow in adoption, the role of synthetic testing in ensuring platform reliability and performance becomes increasingly crucial. The insights gained from this study suggest that integrating synthetic testing into Kubernetes observability strategies can significantly enhance an organization's ability to maintain high-quality, resilient services.

Looking forward, the field of synthetic testing in Kubernetes environments presents exciting opportunities for further innovation. Areas for future research include integration with AI for predictive analysis, optimization of resource usage in large-scale deployments, and the development of standardized test suites for common Kubernetes configurations.

In conclusion, synthetic testing, as exemplified by the Keikoproj Active-Monitor Controller, offers a valuable approach to addressing the observability challenges in Kubernetes environments. By providing a proactive, user-centric perspective on system health and performance, it enables organizations to maintain more reliable, efficient, and responsive Kubernetes deployments. As the complexity of cloud-native applications continues to grow, the role of synthetic testing in ensuring robust and dependable Kubernetes environments is likely to become even more significant.

REFERENCES

[1] Kubernetes Authors, "Kubernetes | Production-Grade Container Orchestration," Kubernetes, 2018. [Online]. Available: https://kubernetes.io/
[2] Datadog, "Synthetic Testing," Datadog Knowledge Center. [Online]. Available: https://www.datadoghq.com/knowledge-center/synthetic-testing/
[3] Kubernetes Authors, "kube-apiserver," Kubernetes Documentation. [Online]. Available: https://kubernetes.io/docs/reference/command-line-tools-reference/kube-apiserver/
[4] etcd Authors, "etcd: A distributed, reliable key-value store," etcd.io. [Online]. Available: https://etcd.io/
[5] Kubernetes Authors, "kube-scheduler," Kubernetes Documentation. [Online]. Available: https://kubernetes.io/docs/reference/command-line-tools-reference/kube-scheduler/
[6] Kubernetes Authors, "kube-controller-manager," Kubernetes Documentation. [Online]. Available: https://kubernetes.io/docs/reference/command-line-tools-reference/kube-controller-manager/
[7] Datadog, "Monitoring Kubernetes performance metrics," Datadog Blog. [Online]. Available: https://www.datadoghq.com/blog/monitoring-kubernetes-performance-metrics/
[8] Prometheus Authors, "Prometheus - Monitoring system & time series database," Prometheus.io, 2018. [Online]. Available: https://prometheus.io/
[9] Fluent Bit Authors, "Fluent Bit: Cloud Native Log Processor," FluentBit.io. [Online]. Available: https://fluentbit.io/
[10] OpenTelemetry Authors, "OpenTelemetry Collector," OpenTelemetry Documentation. [Online]. Available: https://opentelemetry.io/docs/collector/
[11] Grafana Labs, "Grafana: The open observability platform," Grafana.com, 2020. [Online]. Available: https://grafana.com/
[12] Kubernetes Authors, "Kubernetes Controllers," Kubernetes Documentation. [Online]. Available: https://kubernetes.io/docs/concepts/architectire/controller/
[13] Kubernetes Authors, "Kubernetes Objects," Kubernetes Documentation. [Online]. Available: https://kubernetes.io/docs/concepts/overview/working-with-objects/kubernetes-objects/
[14] Kubernetes Authors, "Custom Resources," Kubernetes Documentation. [Online]. Available: https://kubernetes.io/docs/concepts/extend-kubernetes/api-extension/custom-resources/
[15] Kubernetes Authors, "Custom Resource Definitions," Kubernetes Documentation. [Online]. Available: https://kubernetes.io/docs/tasks/extend-kubernetes/custom-resources/custom-resource-definitions/
[16] Kubernetes Authors, "Client Libraries," Kubernetes Documentation. [Online]. Available: https://kubernetes.io/docs/reference/using-api/client-libraries/
[17] Kubernetes Authors, "Operator pattern," Kubernetes Documentation. [Online]. Available: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/
[18] M. Hightower, "Automating Tasks with Kubernetes Custom Controllers," The New Stack. [Online]. Available: https://thenewstack.io/automating-tasks-with-kubernetes-custom-controllers/
[19] CNCF, "Enhancing Observability with Kubernetes Custom Controllers," CNCF Blog, May 12, 2021. [Online]. Available: https://www.cncf.io/blog/2021/05/12/enhancing-observability-with-kubernetes-custom-controllers/
[20] Keikoproj, "active-monitor," GitHub. [Online]. Available: https://github.com/keikoproj/active-monitor/
[21] Argo Project, "Argo Workflows," Argo Project Documentation. [Online]. Available: https://argoproj.github.io/argo-workflows/



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY