World Journal of Advanced Engineering Technology and Sciences

eISSN: 2582-8266

Cross Ref DOI: 10.30574/wjaets

Journal homepage: https://wjaets.com/

(REVIEW ARTICLE)

Check for updates

Best Practices for Monitoring Kubernetes Clusters: Reliability and Minimise Operational Overhead

Amar Gurajapu *

AT&T, New Jersey, United States.

World Journal of Advanced Engineering Technology and Sciences, 2026, 18(01), 007-015

Publication history: Received on 24 November 2025; revised on 31 December 2025; accepted on 01 January 2026

Article DOI: https://doi.org/10.30574/wjaets.2026.18.1.0002

Abstract

Kubernetes has emerged as the leading platform for container orchestration, delivering unprecedented flexibility and scalability for cloud-native applications. However, its dynamic, distributed nature introduces significant complexity in maintaining operational visibility and reliability. Effective monitoring is no longer luxury, rather it is essential for detecting failures, optimizing performance, and securing workloads. This paper presents a comprehensive review of best practices for monitoring Kubernetes clusters, synthesizing recommendations from industry leaders, open-source communities, and recent academic research. By adopting these practices, organizations can build resilient, observable, and manageable Kubernetes environments.

Keywords: Kubernetes; Observability-framework; Fluentbit; Filebeat; Metricsbeat; Logstash; Elasticsearch; Opensearch; Prometheus; Grafana; Kibana

1. Introduction

The rise of Kubernetes as the backbone of modern infrastructure has transformed the way applications are developed, deployed, and managed. By abstracting away server management and automating deployment processes, Kubernetes empowers teams to focus on delivering value. Yet, this abstraction also conceals vital operational details, making it challenging to detect, diagnose, and resolve issues in real-time. Traditional monitoring approaches, designed for static and monolithic environments, fall short in the face of Kubernetes' ephemeral workloads, auto-scaling, and microservices communication patterns.

Monitoring in Kubernetes is not merely about collecting logs and metrics. It is about constructing an observability framework, which is a system that enables teams to understand the internal state of their applications and infrastructure based on external outputs. This involves integrating logging, metrics, tracing, and alerting, and ensuring these tools work seamlessly together. This paper delves deep into the best practices that define effective monitoring strategies in Kubernetes, drawing from CNCF recommendations, and research from major cloud providers and open-source initiatives.

This paper illustrates different tools across monitoring scope and emphasizes the best practices with practical application. We have tried to highlight the research insights and challenges for each of those areas.

* Corresponding author: Amar Gurajapu.

2. Centralized logging

Not just for applications, the centralized logging is foundational to any Kubernetes monitoring strategy. In a Kubernetes cluster, containers are ephemeral: they can start, stop, and restart frequently as part of normal operation or during failures and scaling events. Without a robust logging pipeline, valuable diagnostic information may be lost forever when pods are terminated.

2.1. Best Practices

- Deploy Log Forwarders as DaemonSets - The most common approach is to run log shippers such as Fluent Bit, Filebeat, Metricsbeat or Logstash as DaemonSets, ensuring every node in the cluster collects logs and metrics from all running containers. This method captures stdout and stderr streams from containers, as well as node-level system logs. In some cases where it is necessary to map container logs to the volume mount on the host, we can implement a admission controller with a sidecar.
- Standardize Log Format - Use structured logging (e.g., JSON) to ensure logs are machine-parsable. Include key fields such as timestamp, log level, container name, namespace, and request IDs. Often organizations utilize common opensource packages like log4j to meet the need. This consistency enables efficient querying, correlation, and automated processing.
- Centralized Storage - Forward logs to a centralized storage solution, such as Elasticsearch, Opensearch, Azure Log Analytics, or any other cloud-native services offered by third party or internally managed. Centralization supports long-term retention, compliance, and advanced analytics.
- Label and Annotate - Enrich logs with Kubernetes metadata: labels, annotations, pod names, and node names. This enables filtering logs by workload, environment, or deployment, which is invaluable during incident response or audits. This metadata must be enforced using policies and procedures by platform teams managing the cluster.

2.2. Practical Application

A production platform for a Telecommunications company might use Fluent Bit to collect logs from all pods and ship them to Elasticsearch, with dashboards in Kibana for real-time troubleshooting and historical analysis

2.3. Research Insights

A study (IAEME, 2024) found that structured, centralized and advanced logging strategies reduced mean time to resolution (MTTR) by up to 67%.

2.4. Challenges

- Log volume can be overwhelming in large clusters. Implement log retention policies and sampling where appropriate.
- Clusters which are shared by multiple application teams might need to customize the policies based on the application needs
- Sensitive data may be present in logs; always apply encryption in transit and at rest and consider log modification for compliance.

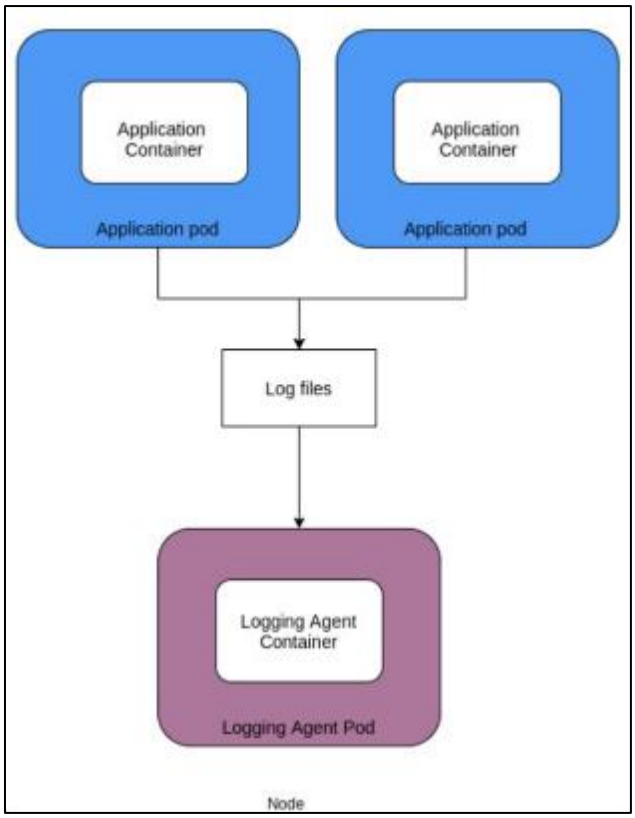
3. Metrics collection and visualization

Metrics are numerical representations of system and application performance over time. In Kubernetes, metrics enable teams to detect anomalies, trigger auto-scaling, and ensure workloads are operating within expected parameters.

3.1. Best Practices

- Prometheus as the Metrics Backbone - Prometheus has become the de facto standard for Kubernetes metrics, offering powerful querying, flexible data retention, and seamless integration with Kubernetes' API. Deploy Prometheus using the Prometheus Operator for ease of management and automatic discovery of targets.
- Expose Application Metrics - Encourage developers to instrument applications to expose metrics using libraries compatible with Prometheus. Standard endpoints (e.g., /metrics) should provide metrics such as request error rates, and business KPIs.
- Node and Cluster Metrics - Deploy node-exporter for system-level metrics and kube-state-metrics for Kubernetes resource metrics (e.g., pod status, replica counts).

- Visualization - Use Grafana to build custom dashboards, enabling real-time and historical analysis of cluster performance, resource utilization, and custom application metrics.



Source: (Sharif, 2025)

Figure 1 Node-level logging agent

3.2. Practical Application

Few of the top 500 companies use Prometheus to collect metrics from all microservices, node-exporter for system metrics, and kube-state-metrics for Kubernetes object health. Grafana dashboards display error rates, and resource trends, with alerts configured for threshold breaches.

3.3. Research Insights

According to (CNCF Survey 2022), over 86% of organizations using Kubernetes rely on Prometheus for monitoring, citing its extensibility and cloud-native alignment.

3.4. Challenges

- Metric cardinality (number of unique metric labels) can explode, impacting storage and performance. Regularly audit metric definitions and enforce label limits.
- High-frequency metrics can be costly; consider down sampling and retention policies.

The initial analysis revealed several issues:

- Missing resource limits - All Deployments lacked resource limits, increasing the risk of resource contention.
- Containers running as root - Default configurations allowed containers to run with root privileges.
- No health checks - Liveness and readiness probes were missing, making failures harder to detect.

4. Resource monitoring and alerting

Resource monitoring and alerting ensures that your cluster and workloads have sufficient resources to operate efficiently, and that you are notified of potential or actual outages before they impact users.

4.1. Best Practices

- Monitor at All Levels - Collect and analyze metrics at the node, pod, container, and namespace levels. Track CPU, memory, disk, and network utilization.
- Set Requests and Limits - Configure resource requests and limits for all workloads to prevent resource contention and ensure fair scheduling. Monitor these settings and their actual usage to identify over-provisioned or under-provisioned workloads.
- Alerting - Define alerting rules in Prometheus Alertmanager or similar tools. Set up alerts for High CPU or memory usage on nodes or pods, Disk pressure or inode exhaustion, Unscheduled or evicted pods and CrashLoopBackoff or high restart counts
- SLOs/SLAs - Define and monitor service level objectives (SLOs) and agreements (SLAs), such as "99.9% of HTTP requests must complete in under 200ms." Use alerting to detect and report SLO violations.
- Notification Integration - Integrate alerting with communication tools like Slack, PagerDuty, or email to ensure rapid response.

4.2. Practical Application

A NYSE listed company sets up Prometheus alerts for node memory > 90%, pod restarts > 5 in 10 minutes, and SLO breaches, integrating Alertmanager with PagerDuty for 24/7 incident response.

4.3. Research Insights

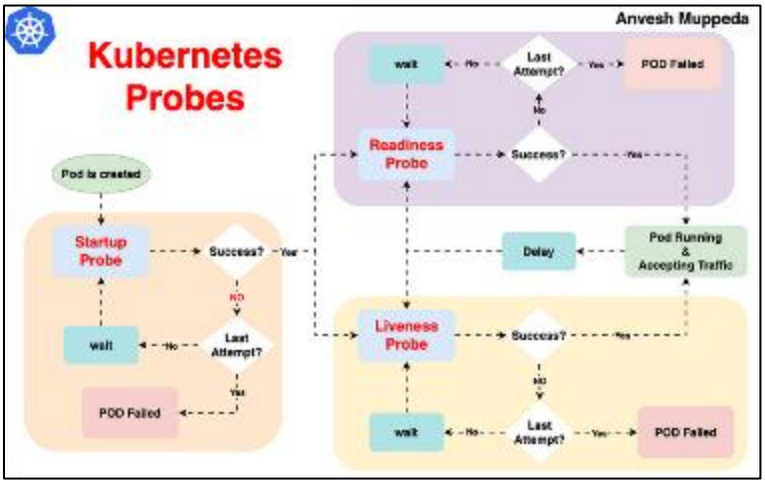
The article (Johnson at el., 2024) emphasizes that Real-time alerting and notification systems are essential for ensuring that response team acts swiftly. They must be actionable, well-defined as they are critical to reliable operations, recommending regular alert reviews to eliminate noise and refine thresholds.

4.4. Challenges

- Alert fatigue is a real risk; prioritize actionable alerts and avoid over-alerting.
- Scaling alerting infrastructure in very large clusters may require sharding or federation.

5. Health checks and probes

Kubernetes uses probes to determine if containers are healthy and ready to serve traffic. Proper configuration of liveness and readiness probes ensures high availability and fast recovery from failures.



Source: (Muppeda, 2024)

Figure 2 Kubernetes probes

5.1. Best Practices

- Liveness Probes - Detect if an application is stuck or deadlocked. If the liveness probe fails, Kubernetes will automatically restart the container. Configure liveness probes using HTTP, TCP, or command execution checks.
- Readiness Probes - Indicate when a container is ready to accept requests. If the readiness probe fails, the container is temporarily removed from service discovery. This is crucial during startup, configuration changes, or dependency initialization.
- Monitor Probe Status - Track probe failures and restart counts using metrics and logs. Frequent failures may indicate application bugs, misconfiguration, or resource constraints.
- Startup Probes - For applications with long initialization times, use startup probes to avoid premature restarts.

5.2. Practical Application

A content provider company configures HTTP readiness probes to check a /healthz endpoint, and liveness probes that execute a lightweight command. Alerts are set up for frequent probe failures, enabling rapid investigation.

5.3. Research Insights

Liveness and Readiness probes are Kubernetes capabilities that enable teams to make their containerized applications more reliable and robust (Redhat, 2019). As a result, it will reduce user-facing incidents and reduced average downtime.

5.4. Challenges

- Incorrectly configured probes can cause cascading failures (e.g., restarts during slow database startups).
- Probes should be lightweight and non-blocking to avoid impacting application performance.
- Incorrect implementation of readiness probes may result in an ever-growing number of processes in the container, and resource starvation if this is left unchecked (Redhat, 2019)

6. Distributed tracing

Distributed tracing provides visibility into request flows across microservices, helping teams identify bottlenecks, latency, and failure points in complex application architectures.

6.1. Best Practices

- Implement Tracing Libraries - Use language-specific libraries compatible with OpenTelemetry or Jaeger to instrument code. Add trace context propagation across services via HTTP headers.
- Sampling Strategies - Use head-based or tail-based sampling to control trace volume and storage requirements.
- Integration with Metrics and Logs - Correlate traces with logs and metrics for comprehensive root cause analysis. Many tools, like Grafana Tempo, support linking traces with logs and dashboards.
- Visualization - Use tracing UIs to visualize request paths, latency distributions, and error rates across services.

6.2. Practical Application

A travel booking platform uses OpenTelemetry to instrument all API endpoints. Traces are collected in Jaeger and visualized in Grafana, enabling engineers to analyze slow requests and dependency issues.

6.3. Research Insights

The CNCF survey (Das, 2025) reports that 82% of organizations now employ distributed tracing to monitor requests across microservices boundaries, compared to just 57% in 2021. This resulted in faster incident resolution and improved cross-team collaboration.

6.4. Challenges

- Instrumentation can be complex, especially in legacy or third-party code.
- Trace data can be voluminous; manage retention and sampling carefully.

7. Workload visibility to avoid operational blindspots

Kubernetes namespaces are often used to separate environments, teams, or applications. Ensuring monitoring coverage across all namespaces and workloads is essential to avoid operational blind spots.

7.1. Best Practices

- Monitor All Namespaces - By default, extend monitoring to all namespaces, including system namespaces (kube-system, monitoring) and those used by third-party operators or services.
- Labeling and Organization - Use consistent labeling and annotation strategies for namespaces, deployments, and services. This supports filtering and aggregation in dashboards and queries.
- Multi-tenancy - For multi-tenant clusters, implement per-namespace dashboards, access controls, and resource quotas. This enables teams to monitor their own workload independently, while platform teams retain cluster-wide visibility.
- Dynamic Discovery - Use tools that support dynamic discovery of new namespaces and workloads, such as Prometheus Operator, which can automatically monitor newly created resources.

7.2. Practical Application

A large enterprise with multiple product teams organizes namespaces by business unit. Each team receives custom Grafana dashboards scoped to their namespace, while central SRE teams monitor the cluster as-a-whole.

7.3. Research Insights

A development team discovered 12 unused namespaces which consume 50% of node capacity (Devops, 2025). This article also highlights several challenges pertaining to 70% of Kubernetes clusters.

7.4. Challenges

- Keeping up with dynamic namespace and workload creation.
- Avoiding duplication or omission of monitoring coverage as teams iterate rapidly.

8. Security compliance

Monitoring systems themselves are a critical part of the security and compliance posture of a Kubernetes cluster. They often contain sensitive information about applications, infrastructure, and users.

Source: (Tremblay, 2024)

Figure 3 Security best practices

8.1. Best Practices

- Encryption - Encrypt monitoring data both in transit (TLS for all endpoints and communication channels) and at rest (disk encryption for log/metric stores).
- Access Control - Use Kubernetes RBAC (Role-Based Access Control) to restrict access to monitoring data, dashboards, and alerting systems. Only authorized users should be able to view or modify sensitive information.
- Audit and Compliance - Enable audit logging for monitoring access and configuration changes. Store audit logs securely for compliance and forensics.
- Data Retention and Redaction - Define clear data retention policies to comply with GDPR, HIPAA, or other regulations. Redact or anonymize sensitive information from logs and metrics where possible.
- Security Monitoring - Monitor the monitoring stack itself for suspicious activity or configuration drift. Integrate with SIEM (Security Information and Event Management) tools as needed.

8.2. Practical Application

A healthcare provider encrypts all Prometheus and Elasticsearch traffic with TLS, uses RBAC to restrict dashboard access, and retains logs for 90 days per HIPAA requirements, with automated log redaction pipelines.

8.3. Research Insights

An article (Ehrman, 2025) highlighted that 61% of secrets are exposed in public repositories. As per (Redhat, 2021), 60% of respondents have experienced a misconfiguration incident in their environments over the last 12 months. Nearly a third have discovered a major vulnerability, and another third said they have suffered a runtime security incident. Lastly, 20% said they had failed an audit.

8.4. Challenges

- Balancing visibility with least privilege access.
- Ensuring compliance requirements are met as regulations evolve.

9. Implementation guidelines

Source: (zegl, 2022)

Figure 4 Devsecops

9.1. Toolchain Recommendations

- Metrics
  - Prometheus - Core metrics collection and querying
  - kube-state-metrics - Exposes Kubernetes object metrics
  - node-exporter - Collects node-level system metrics
- Logs
  - Fluent Bit/Filebeat - Log shipping agents
  - Elasticsearch, Loki - Centralized log storage and search
- Tracing
  - Jaeger, OpenTelemetry - Distributed tracing frameworks
- Dashboards & Visualization
  - Grafana - Unified dashboarding for metrics, logs, and traces
  - Kibana - Log analytics (for Elasticsearch)
- Alerting
  - Alertmanager - Prometheus alert routing
  - PagerDuty, Slack - Notification endpoints.

9.2. CI/CD Integration

- Validate monitoring annotations and resource limits/requests in pull requests using static analysis tools (e.g., kube-score, custom scripts).
- Automate deployment and configuration of monitoring agents as part of cluster bootstrap processes.

9.3. Continuous Improvement

- Regularly review and update monitoring coverage, alert definitions, and retention policies.
- Conduct gamedays and incident response simulations to test monitoring effectiveness and team readiness.

10. Challenges and future

10.1. Scalability

As clusters grow, monitoring pipelines must scale horizontally. Use sharding, federation, and retention strategies to manage load.

10.2. Cost Management

High cardinality metrics and large log volumes can drive up storage and processing costs. Apply sampling, aggregation, and retention policies to control spending.

10.3. AI/ML for Anomaly Detection

Emerging techniques use machine learning to detect anomalies, forecast resource needs, and reduce false positives in alerting. Projects like Prometheus Adaptive Alerting and OpenAI's time-series analysis are pushing the boundaries in this space.

10.4. Evolving Standards

The observability landscape evolves rapidly. Stay abreast of new developments in OpenTelemetry, eBPF-based monitoring, and cloud-native security integrations.

11. Conclusion

Effective monitoring is the cornerstone of reliable, secure, and performant Kubernetes operations. By implementing centralized and consistent logging, comprehensive metrics collection, proactive alerting, robust health checks, distributed tracing, full namespace visibility, and secure monitoring practices, organizations can unlock the full potential of Kubernetes while minimizing operational risk. As both the threat landscape and application complexity grow, continuous investment in observability, automation, and intelligent monitoring will be key to sustainable success in cloud-native environments.

Compliance with ethical standards

Acknowledgments

Thanks to Tony Hansen, Swapna Anne and Vasavi Yeka for reviewing the paper and providing valuable inputs.

Disclosure of conflict of interest

No conflict of interest to be disclosed.

References

[1] IAEME Publication, "OBSERVABILITY AND MONITORING STRATEGIES FOR SCALABLE BACKEND SYSTEMS," *IAEME PUBLICATION*, 2024.

[2] https://www.academia.edu/120999339/OBSERVABILITY_AND_MONITORING_STRATEGIES_FOR_SCALABLE_BACKEND_SYSTEMS (accessed Dec. 28, 2025).

[3] CNCF, "Cloud Native Observability Microsurvey: Prometheus leads the way, but hurdles remain to understanding the health of systems," *CNCF*, Mar. 08, 2022. https://www.cncf.io/blog/2022/03/08/cloud-native-observability-microsurvey-prometheus-leads-the-way-but-hurdles-remain-to-understanding-the-health-of-systems/ (accessed Dec. 28, 2025).

[4] O. B. Johnson, J. Olamijuwon, E. Cadet, Olajide Soji Osundare, and Yodit Wondaferew Weldegeorgise, "Developing Real-Time Monitoring Models to Enhance Operational Support and Improve Incident Response Times," *International Journal of Engineering Research and*, pp. 1296–1300, Nov. 2024, Available: https://www.researchgate.net/publication/387032825_Developing_Real-Time_Monitoring_Models_to_Enhance_Operational_Support_and_Improve_Incident_Response_Times

[5] "Liveness and Readiness Probes," *Redhat.com*, 2019. https://www.redhat.com/en/blog/liveness-and-readiness-probes (accessed Dec. 28, 2025).

[6] "Configure Liveness, Readiness and Startup Probes," *Kubernetes*. https://kubernetes.io/docs/tasks/configure-pod-container/configure-liveness-readiness-startup-probes/

[7] S. Das, "CLOUD-NATIVE OBSERVABILITY," *Research Journal of Modernization in Engineering*, vol. 07, no. 03, p. 7, Mar. 2025, doi: https://doi.org/10.56726/IRJMETS69547.

[8] Devops, "In 2026, 70% of Kubernetes Clusters Are Just Waiting to Be Forgotten," *Medium*, Jul. 20, 2025. https://aws.plainenglish.io/in-2026-70-of-kubernetes-clusters-are-just-waiting-to-be-forgotten-a671b5aaf902 (accessed Dec. 28, 2025).

[9] N. Ehrman, "The most common Kubernetes security issues and challenges," *wiz.io*, Sep. 30, 2025. https://www.wiz.io/academy/container-security/common-kubernetes-security-issues (accessed Dec. 28, 2025).

[10] Redhat, "Most Common Kubernetes Security Issues and Concerns to Address," *Redhat.com*, 2021. https://www.redhat.com/en/blog/most-common-kubernetes-security-issues-and-concerns-to-address

[11] A. Sharif, "Kubernetes Logging Guide: The Basics," *Crowdstrike.com*, 2025. https://www.crowdstrike.com/en-us/guides/kubernetes-logging/ (accessed Dec. 29, 2025).

[12] Anvesh Muppeda, "⎈ A Hands-On Guide to Kubernetes Probes: Liveness, Readiness, and Startup Probes ⎈," *Medium*, Nov. 11, 2024. https://medium.com/@muppedaanvesh/a-hands-on-guide-to-kubernetes-probes-liveness-readiness-and-startup-probes-ee047ebab504

[13] T. Tremblay, "ERP Security: Best Practices to Keep Data Safe," *Kohezion*, Jul. 19, 2024. https://www.kohezion.com/blog/erp-security

[14] K. Tiwari, "DevSecOps: What is It & Why is It Important," *BigStep Technologies*, May 20, 2022. https://bigsteptech.com/blog/devsecops-what-is-it-why-is-it-important

Authors short biography

The author works for AT&T and has extensive work experience playing techno-functional role leading multi-cloud complex transformation program which involves most of the technologies referred in this paper.

Amar Gurajapu is Principal Member of Technical Staff at AT&T. Amar has 25 years of experience in Telecom Software Engineering. He is leading Cyber Security, and digital initiatives for key Network systems portfolio aligned with AT&T organization goals