International Journal Science and Technology
IJST Vol 2 No. 1 | March 2023 | ISSN: 2828-7223 (print), ISSN: 2828-7045 (online) Page 95-107

Push versus Pull Telemetry at Cloud Scale: Performance Benchmarking in Kubernetes-Based OpenTelemetry Environments

Akhil Reddy Mandadi
Independent Researcher, akhilreddymandadi95@gmail.com, India

Article History
Received : March 10, 2023
Revised : March 27, 2023
Accepted : April 20, 2023
Published : April 27, 2023

Cite This Article:
Reddy, A. . (2023). Push versus Pull Telemetry at Cloud Scale: Performance Benchmarking in Kubernetes-Based OpenTelemetry Environments. *International Journal Science and Technology*, *2*(1), 95–107.

DOI:
https://doi.org/10.56127/ijst.v2i1.2795

Abstract: In today's cloud-native environments, telemetry pipelines are becoming more and more vital for maintaining visibility, distributed traceability, automated monitoring, and intelligent incident response for large-scale containerized environments. Most observability ecosystems today are either push or pull based, with push-based telemetry architectures having agents and exporters actively publish metrics and traces to central collectors or pull-based architectures having monitoring systems actively pollute observability endpoints exposed by services and infrastructure nodes. Although both paradigms have been widely embraced in industry, there are few empirical comparisons that can be reproduced to assess their performance characteristics under operational conditions in the cloud. This paper provides a comprehensive benchmark study to understand and analyze the performance difference between push-based vs. pull-based telemetry architectures, with the emphasis on evaluating push-based telemetry in the context of Kubernetes cluster environments, equipped with OpenTelemetry collectors, Prometheus-based monitoring system, and synthetic workload generators to simulate production-like workloads. The study assesses end to end latency distributions, telemetry throughput, operational overhead, cost per event, end to end system behavior when put under sustained workloads, and end to end system recovery behavior after induced failures. Experimental evidence shows that the push architecture can be more adaptable under burst-intensive loads due to its flexibility in buffering mechanisms and asynchronous delivery of events, whereas the pull architecture has better consistency of query results, easier visibility of the operation, and more predetermined semantics for monitoring under steady load. This paper presents a hybrid telemetry architecture especially designed for the context of AI/machine learning infrastructure environments where the burst-resilience and scalable query semantics are concurrently required. The proposed framework combines push side buffering, and ingestion flexibility with pull side observability and analytics. The study introduces a benchmark methodology that is reproducible and experimentally validated hybrid telemetry design that can enhance the observability performance of modern distributed cloud-native infrastructure.

Keywords: Cloud Observability, Kubernetes, OpenTelemetry, Prometheus, Telemetry Systems, Push Architecture, Pull Architecture, Cloud Benchmarking, AI/ML Infrastructure, Distributed Systems.

INTRODUCTION

Cloud-native computing has reshaped the way distributed systems work and operate today. With scalability, resiliency, and deployment agility being key characteristics sought by organizations, Kubernetes orchestrated infrastructures, containerized microservices, distributed APIs, and multi-cloud deployment strategies are gaining momentum. With the increasingly dynamic and operationally complex nature of distributed systems, observability infrastructures have grown from simple monitoring solutions into full-fledged telemetry environments that are able to gather, process, analyze and visualize vast amounts of operational data streams. Telemetry is now part of the core pillars of modern cloud engineering, providing the real-time insights for operational intelligence, proactive incident detection, performance optimization and automated remediation workflows.

Most cloud-native observability platforms are based on three main types of telemetry: metrics, logs and traces. Metrics give quantitative metrics of system state and resource use, logs record event-level information as it occurs, and distributed traces show the flow of requests through the interconnected microservices. As microservices grow in scope, the significance of telemetry pipelines has grown as well since it is much more difficult to gain visibility into operations in decentralized architectures with hundreds or thousands of dynamically orchestrated services. Studies on distributed system analysis reveal that the higher the quality of the telemetry, the more efficient troubleshooting becomes, the more resilient the operation and the more effective the incident recovery is in large systems [1] and [3].

This trend toward expanding Kubernetes environments has driven further adoption of an operational model with telemetry. Kubernetes orchestration adds a number of highly dynamic infrastructure features such as auto-scaling, rolling deployment, ephemeral containers, distributed scheduling, and more, which produces huge volumes of telemetry that need to be collected with scalable pipelines. Monitoring and trace collection are increasingly supported in these environments with observability frameworks based on Open Telemetry, Prometheus and service mesh based on Istio, for example [2], [10]. Service mesh architectures also add to the complexity of telemetry, adding the sidecar proxies and the observability components of the network layers that can generate high volumes of performance and communication metrics [2].

Today's cloud environments also rely on AI powered operational automation and automated root cause analysis systems that are heavily reliant on the quality of telemetry data and the availability of telemetry. Automated incident detection systems are designed to detect anomalies, to correlate failures and to trigger remediation workflows in real-time by leveraging on metrics and traces [12, 15]. As such, the architecture decisions for telemetry have become strategically important as the performance of observability directly impacts reliability engineering, enforcement of service-level objectives and operational analytics performed with AI.

Although there has been widespread industry acceptance of telemetry infrastructures, architectural decisions about telemetry transport are still under discussion. In modern observability worlds, there are two dominant paradigms of telemetry: push, and pull. Push-based telemetry system, passively receive telemetry events from applications or agents and move them towards the central collectors. This is the model that an OpenTelemetry exporter, and StatsD-based telemetry architecture are examples of. As seen in Prometheus-based systems, pull-based systems periodically poll the monitored services with telemetry endpoints to monitor the servers.

Push and Pull Telemetry Paradigms

Push-based telemetry architectures are those that send telemetry events directly from monitored applications, agents or exporters towards centralized collection systems. In this model, the producers of the telemetry are in charge of the moments when data is transmitted and how the events are delivered. Push-oriented pipelines are a common approach for OpenTelemetry exporters and statsD infrastructures, as they enable event forwarding asynchronously and the ability to buffer or adjust the buffering strategy based on conditions. In situations where the rate of generating telemetry data varies greatly or in cases where the delivery of events is required right after they are created, push-based telemetry systems are well suited.

The OpenTelemetry ecosystem is one of the most evolved push based telemetry standards in today's cloud-native world. The OpenTelemetry collectors collect telemetry streams from many services and propagate them to analytics backends, observability dashboards, or storage backends. These architectures offer a lot of flexibility since it is possible to batch telemetry events dynamically, compress them or add retry logic in case of temporary network failures [12]. Push-based telemetry systems are thus highly resilient in bursty environments, typical for microservices deployments or infrastructure for AI workloads.

Push telemetry architectures also make delivering telemetry in distributed, heterogeneous environments easier, as the agents can proactively push data, no matter the collector polling schedule. Telemetry exporters embedded inside the sidecar proxies or service mesh infrastructure can continuously push metrics and traces to the collectors, eliminating the need for the presence of a centralized coordinator for collecting these metrics

[2]. Such architectures enable high level of granularity of telemetry data and minimize possibility of missing transient operational events.

The Kubernetes ecosystem is dominated by infrastructures based on Prometheus, for their simplicity in operations, high level of service discovery integration, and efficient handling of time-series data [10]. With pull-oriented telemetry collection, you can also centrally control the frequency of the collection and the behavior of the queries, so as to increase the predictability of the monitoring. The monitoring server can discover Kubernetes workloads dynamically and periodically fetch the metrics without the exporters having to deal with the logic of the metrics that they transmit.

Pull-based architectures have, however, some significant drawbacks. Rapid changing operational conditions or fleeting events may not be evident at first glance within the scrape intervals. Gaps in telemetry can happen when endpoints are down for a period of time or when workloads are terminated before the next scraping cycle runs. Also, in very large service populations, scalability issues with pull systems may arise due to the fact that the overhead for pulling also grows with the number of endpoints.

Each telemetry paradigm has, however, different operational merits and demerits. Push architectures are more concerned with delivering flexible events and adapting to bursts, while a pull architecture is more concerned with simplicity, predictability and centralized query semantics. To identify the architectural differences and how they impact operations, there needs to be controlled benchmarking with repeatable cloud-scale conditions.

Research Problem

As cloud native infrastructures grow in complexity, organizations have grown more reliant on observability systems that can provide operational visibility throughout distributed infrastructures, which must be scalable to accommodate large organizations. As infrastructures become more complex, organizations have become more dependent on cloud based observability systems, which must be scalable to support large enterprises while maintaining operational transparency across distributed infrastructures. Kubernetes clusters can now handle very dynamic workloads such as microservices, service meshes, AI pipelines, edge computing systems, and distributed APIs. Such environments produce significant amounts of telemetry data that need to be efficiently collected, aggregated, processed and stored. While many organizations are using telemetry, there is still uncertainty about the business impacts of choosing a push or pull-based telemetry architecture.

Most production systems are running mixed architectures of telemetry, with both push and pull-based collection architectures. OpenTelemetry collectors can run concurrently with Prometheus scraping pipelines and service meshes and observability agents can produce both asynchronous traces and periodically exposed metrics. Often, though, the architecture chosen for transporting telemetry is decided by preference of vendor, familiarity with implementation, or anecdotal operational experience without having been tested to any great degree.

Lack of repeatable head-to-heads telemetry benchmarking introduces a huge information gap in the world of cloud observability research. While there are numerous references in the literature to different frameworks for collection and distributed tracing, there are few that compare the performance metrics of different telemetry paradigms under the same infrastructure settings in a systematic manner. Research that focuses on distributed performance analysis has increasingly discussed the need for scalable telemetry instrumentation and has considered comparative telemetry transport evaluation very little [3], [6]. Likewise, Kubernetes-specific cloud observability frameworks that have been suggested emphasise on deployment flexibility or monitoring coverage, while lacking in quantifying the latency, cost efficiency, recovery behaviour and sustained-load resilience across the telemetry architectures.

This is a research gap that is especially important in the AI and machine learning infrastructures. Distributed training pipelines, GPU schedulers, inference accelerators, and orchestration frameworks often generate telemetry streams with high bursts, as they collect a lot of data for operations in a short duration. The ability of the telemetry architecture to buffer and/or process these bursts efficiently may result in higher loss of events, delayed operational visibility, or inaccuracy when it comes to detecting anomalies. Meanwhile, AI platforms need to provide a robust query semantics and centralized analytics visibility for operational audit, model governance and infrastructure optimization.

Research Questions

This study aims to answer three main research questions to assess the behavior of telemetry architecture in cloud-native infrastructures under Kubernetes

RQ1: What are the differences between push-based and pull-based telemetry for end-to-end latency under cloudscale workloads?

RQ2: What are the effects of a sustained workload on throughput, operational overhead and backpressure in push and pull telemetry systems?

RQ3: Is the ability to achieve a hybrid push-pull ingestion/pull query semantics useful for enhancing observability performance in AI and machine learning infrastructure environments?

The following research questions explore the telemetry responsiveness, scalability, resilience and operational practicality in the realistic distributed workload setting.

Research Contributions

This study makes several important contributions to the body of research on cloud-native observability and telemetry engineering.

1 First, a reproducible benchmark framework for evaluating the various telemetry architectures in Kubernetes is introduced. The framework combines OpenTelemetry collectors, Prometheus monitoring systems, synthetic workload and failure injection tools that can be used for repeatable experimentation.

2 Second, the study compares the two types of paradigms (push versus pull) in detail in the context of the same workload. The benchmark consists of measurement of latency distributions, stability of throughput, operational cost, queue behaviour, telemetry loss and recovery performance.

3 Third, the research examines the resilience of telemetry in the event of a sustained workload stress and/or infrastructure failure. Multiple telemetry architectures are quantitatively evaluated with regards to backpressure propagation, congestion at the collectors, recovery dynamics and overhead involved in operations.

4 Fourthly, the paper suggests a hybrid telemetry architecture with a specific focus on AI and machine learning infrastructures. The proposed model combines the buffering capabilities that are push oriented with query semantics that are pull oriented, which leads to more burst resilience and analytical consistency.

Last but not least, the study provides operationally relevant insights derived from the experimentation, which are useful for selecting the telemetry architecture in the modern distributed cloud-native environments.

Paper Organization

This paper is organized in the following way: Section II summarizes the related work focused on the field of cloud observability, distributed telemetry systems, and telemetry architecture paradigms. The experimental methodology and Kubernetes benchmark environment, telemetry configurations, workload generation process and evaluation metrics are presented in Section III. In Section IV, benchmark results will be discussed for latency analysis, operational cost, backpressure behavior, and failure recovery performance of telemetry systems. The AI and machine learning optimized hybrid telemetry architecture is introduced in Section V. Finally, Section VI summarizes the study and presents possible future research areas, including adaptive telemetry routing and edge observability and multi-cluster telemetry optimization.

LITERATURE STUDY

Observability in Cloud-Native System

Shifting from monolithic software systems to cloud native systems has dramatically changed the way these distributed systems have to be operated. Nowadays, cloud-based systems are designed to be elastic, resilient and quickly deployable, often using containerized microservices spread over a variety of Kubernetes platforms. These features enhance the scalability and development speed, but also complicate infrastructure and pose additional challenges for observability. In cloud-native environments where services are dynamically instantiated, migrated, scaled and terminated in short time frames, traditional monitoring solutions that are effective in static environments often fall short.

The separation of monitoring and observability has thus become a main concern for distributed systems research. Observability is the goal of monitoring: to give full visibility of system behavior via metrics, logs, traces and system events relationship, while the traditional goal of Monitoring is defined by predefined Metrics and alerts based on thresholds. A study on cloud-native operational intelligence shows that observability allows engineers to deduce the internal state of the system based on data from an external source (telemetry) ([9]). In the same way, in the studies on the basics of cloud computing, the importance of telemetry architectures that can adapt to elastic operational conditions and dynamic workload characteristics, such as those required for large-scale distributed infrastructures, is highlighted [11].

In a distributed microservice architecture, there can be a lot of telemetry being generated due to service interactions between many independently deployed components. The service mesh technologies like Istio have added more observation layers by adding observability sidecar proxies, which collect communication behavior and the metrics at the network level, [2].

Research on cloud-native observability has also continued with research focused on telemetry instrumentation and improving the visibility of operations. In [14] the authors showed that low level infrastructure instrumentation through extended kernel-level telemetry mechanisms can provide good visibility into the interactions of services and performance bottlenecks. Likewise, edge-based telemetry architectures that are controlled through the Kubernetes infrastructures suggest that telemetry collection will increasingly move beyond centralized cloud architectures into geographically distributed architectures [12].

The development of cloud observability also mirrors wider changes taking place in the world of cloud operations like DevOps and continuous delivery environments. While exploring convergence issues in CI/CD, a key topic identified is the impact of observability maturity on deployment velocity, with the ability to quickly change infrastructure relying on observability that evolves as quickly as the code itself or faster ([6]). Now observability systems have become essential, not just an extra monitoring extension, to the system.

Pull-Based Telemetry Systems

One of the most popular models in cloud-native monitoring systems is pull based telemetry architectures. In a pull telemetry system, the operational metrics are collected regularly from the exposed service endpoints as per the system's collection intervals. This model has become the most popular due to Prometheus' ability to interact with Kubernetes service discovery mechanisms and its excellent time series data management capabilities [9].

Telemetry infrastructures based on the Prometheus architecture have a number of operational benefits. Centralized telemetry scheduling gives administrators the ability to manage their distributed services' data collection frequency and query behavior in a uniform manner. Another benefit of Kubernetes-native service discovery mechanisms is that they make it easier to monitor deployment, as the telemetry endpoints can be discovered without having to have logic for communicating with individual exporters. These architectures also make it easy to operate with a debugging mindset since collection activities are controlled by monitoring servers.

Comparative evaluation of existing literature shows that there are differences in the focus of research, the context in which it is deployed, and the scope of the benchmarks. A comparative evaluation of the existing literature also shows that there are differences in research focus, deployment context and scope of the benchmarks. Table I summarizes a summary of several representative studies that have been done in this space concerning observability infrastructures and telemetry systems.

Table I: Comparative Summary of Existing Observability and Telemetry Studies

Study	Research Focus	Key Contribution	Limitation
Baeten [1]	Microservice coverage detection	Service visibility enhancement	No telemetry transport analysis
Mohamed [9]	Multi-cloud observability	Kubernetes observability framework	No push-pull comparison
Sharma [14]	eBPF observability	Kernel-level telemetry improvements	Limited scalability evaluation
Seknametla [13]	OpenTelemetry tracing	Distributed trace correlation	No benchmark analysis
Scano et al. [12]	Edge telemetry systems	Kubernetes telemetry orchestration	Edge-specific scope
Denys [3]	Performance analysis	Distributed instrumentation framework	No telemetry architecture evaluation

Source: Author compilation synthesized from prior literature ([1], [3], [9], [12]–[14]).

From the above studies, Table I shows that the previous studies are mostly related to the individual aspects of instrumentation, deployment of observability, and telemetry collection. The architectures for telemetry transport have received limited research attention and there are no existing benchmark comparisons in the literature that can be reproduced between push vs. pull paradigms in an identical cloud-scale environment.

Push-Based Telemetry Systems

Push-based telemetry systems are based on a different collection model where the telemetry producers actively push their operational data to the collectors or to the telemetry aggregation system. Push-oriented designs for telemetry are typical in OpenTelemetry collectors, distributed tracing exporters and event-forwarding agents. These architectures enable the sources of the telemetry to schedule when the telemetry is transmitted, instead of using a centralized polling schedule.

OpenTelemetry has had a huge impact on the standardization of telemetry in cloud-native infrastructures. A study showing the impact of automated root cause analysis using OpenTelemetry trace correlation on incident resolution performance showed significant variation in the quality of the telemetry and propagation consistency of events had a significant effect on the speed of incident resolution [13]. In the same way, the telemetry events are continuously streamed to AI-based infrastructure management systems that rely on them for anomaly detection and automation functions in operations ([15]).

In very dynamic operating environments, push telemetry systems offer valuable benefits. Event transmission can be done straight after event generation, so as to minimize scheduled polling intervals. Push architectures are also able to provide buffering and asynchrony features that can help increase resilience during bursty workloads. Research on secure fleet-wide telemetry collection in Kubernetes shows that agents which collect telemetry offer higher adaptability in heterogeneous deployment infrastructures ([4]).

But there are also operational complexities to push systems. The more throughput telemetry data has, the more important buffer management, queue persistence and congestion control mechanisms become. Studies on asynchronous distributed performance analysis have shown that the overhead incurred by the propagation of events and the problems of transmission synchronization are important in distributed infrastructures [7]. Furthermore, research studies of the performance of hard real-time cloud systems suggest that the variance of the latency of telemetry in the cloud could impact the responsiveness of the operations in time-critical applications ([10]).

To conceptually show the difference in architectures that have been detected in the preceding studies, the push-based and pull-based telemetry pipelines are shown as a comparative representation in the cloud-native infrastructures in Figure 1.

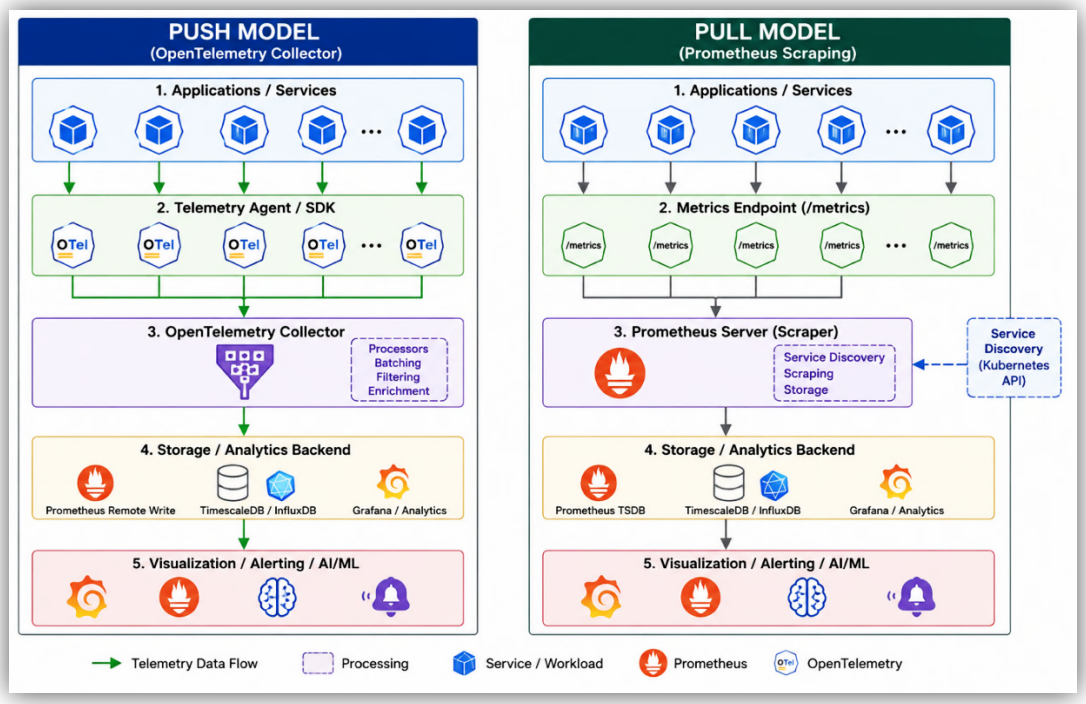

Figure 1. Conceptual Comparison of Telemetry Architectures
Source: Author-designed conceptual framework derived from cloud observability literature ([2], [9], and [13]).

As shown in figure 1, there is a basic difference between telemetry paradigms. In the push architecture, the most important concerns are those of active event forwarding and buffering flexibility while the pull

architecture focuses on centralized control of collection and query transparency. These structural differences have potentially significant implications in terms of operability, latency, and scalability of telemetry.

Existing Benchmark Studies

The literature is somewhat fragmented for research on telemetry benchmarking. Many existing research studies focus on either performance analyze frameworks, distributed tracing techniques or the efficiency of instrumentation without taking into account the transport of the telemetry under similar conditions. Various performance instrumentation studies have examined scalability properties of distributed systems, but usually focus on the computational overhead rather than telemetry transport behaviour ([3] and [7]).

Telemetry data has increasingly been used for automated incident analysis for root cause analysis and predictive fault management. In large-scale microservice systems, telemetry relationships have been demonstrated to enhance the accuracy of fault localization in machine learning frameworks that are driven by traces [5]. However, most of these studies are of analytical results and not the efficiency of the telemetry architecture. The same restrictions can be found in DevOps and operational convergence research where observability mechanisms are regarded as secondary analytical objects, rather than the main objects of analysis [6, 8].

Research Gap

Readings as a whole show significant progress in the field of telemetry instrumentation, observability frameworks, distributed tracing and automated operational intelligence. But existing research still focuses on individual observability mechanisms and not on the comparative evaluation of telemetry transport. The existing research has focused on: Enhancing service visibility, Telemetry orchestration, OpenTelemetry Integration, AI for incident analysis and distributed instrumentation frameworks ([1], [5], [12], and [15]). However, there is currently no benchmark study that has been performed and published to make a side-by-side comparison of push-based vs. pull-based telemetry architectures in the same Kubernetes context.

There is no empirical comparison, which makes it hard to determine the trade-offs between the operational aspects of latency behavior, back-pressure resilience, telemetry cost, and failure recovery characteristics. A reproducible benchmark that can systematically evaluate the performance of telemetry paradigms under the cloud-scale workloads to support the architecture decision-making process still needs to be developed to navigate the large number of possibilities for architecture decisions in modern Kubernetes and AI-driven infrastructure environments.

RESEARCH METHOD

Experimental Design

This study used a controlled benchmark approach to compare and contrast the performance characteristics of the two architectures: push-based and pull-based, for telemetry in cloud-native environments managed by Kubernetes. The methodological design was organized in such a way that it guarantees its repeatability, experimental consistency, and direct comparability in the case of the same infrastructure conditions. Telemetry transport effects were isolated from unrelated telemetry infrastructure influences by selecting a standardized deployment configuration for the benchmark studies in distributed environments, where the environment and instrumentation approaches used were variable. Asynchronous systems can have significant variations in operations when facing varying workloads, and previous works on distributed performance analysis and cloud observability have highlighted the need for reproducible experimental controls. In the same way, testing and experimentation in Kubernetes has been reported as more reliable platform for comparing infrastructure in cloud-native observability frameworks [9].

The experimental design was based on the principal independent variable, the architecture type of telemetry. Two models of telemetry were compared: push oriented (OpenTelemetry collectors) and pull oriented (Prometheus-based scraping). Other independent variables were workload intensity and failure conditions induced. The workload was adjusted in both low volume, moderate, burst intensive and sustained traffic conditions. Failure scenarios considered were collector restarts, temporary network disruption and node interruptions. Dependent variables were telemetry latency, throughput stability, recovery time, usage of operational resources and telemetry delivery behavior. The measurements were chosen due to the increased importance of telemetry response and resilience of operation intelligence systems for automated incident management and AI-assisted troubleshooting as described in [13] and [15]

Kubernetes Testbed Environmen

A Kubernetes-based infrastructure has been used to simulate realistic cloud-native operational conditions for deployment of the benchmark environment. Kubernetes based environments have emerged as the building blocks in the microservice deployment ecosystem, enabling dynamic orchestration, workload

scaling and infrastructure abstraction, as described in [11]. Integration of service mesh and instrumentation of telemetry mechanisms was followed to implement cloud-native observability practices, as outlined in the Kubernetes monitoring and infrastructure studies [2] and [4].

In order to obtain consistent infrastructure deployment for all the benchmark experiments, the cluster specifications were standardized as indicated in Table II.

Table 2: Kubernetes Benchmark Environment Configuration

Component	Configuration
Cluster Nodes	6
CPU per Node	8 vCPU
Memory per Node	32 GB
Kubernetes Version	v1.31
OpenTelemetry Collector	Latest Stable Release
Prometheus Server	Latest Stable Release
Container Runtime	containerd
Service Mesh	Istio

Source: Author-designed benchmark configuration based on Kubernetes observability infrastructure studies ([2], [4], [9]).

As can be seen in Table II, infrastructure resources were carefully designed and monitored to avoid environmental bias. The cluster specifications are based on the deployment characteristics commonly seen in medium scale cloud-native infrastructures as well as in a kubernetes observability study. Further enabling realistic operational behavior ([2]) was the integration of Istio service mesh capabilities to enable the generation of telemetry from network interactions and sidecar communication layers.

To support this, the Kubernetes environment was extended with recommendations for telemetry security and instrumentation that were derived from an analysis of the Kubernetes fleet-wide telemetry hardening study [4]. Secure transmission and scalable deployment of agents were deemed necessary benchmark design requirements for telemetry systems, as they are increasingly running in distributed systems.

Telemetry Architecture Configurations

Two telemetry pipelines were built to test the behavior of the architecture under the same conditions. The push telemetry configuration used application telemetry exporters that were integrated with the OpenTelemetry agents and centralized collectors. The telemetry producers actively send metrics and traces towards the collector's right after the event generation. The deployment of OpenTelemetry was based on the architectural suggestions given in distributed trace analysis and incident correlation research [13].

The pull telemetry environment used endpoint scraping using Prometheus. Applications that provided telemetry interfaces were scraped at regular intervals by the centralized application. Service discovery in Kubernetes-native tools dynamically discovered application instances in deploy and scale operations. The previous research about cloud observability indicated that the use of pull-based telemetry systems allows for easier management of deployment and is semantically clear to monitor [9].

Telemetry collection systems also included microservice instrumentation techniques based on recent observability research with service coverage analysis and improving observability of kernel components ([1], [14]). These mechanisms allowed for more wide-spread visibility of telemetry, on both the application and infrastructure layers.

Synthetic Workload Generation

To simulate the telemetry behavior under production-like loads and various operating conditions, synthetic workloads were created. Staged workload profiles were used for the traffic generation, these included low, moderate, burst-intensive and sustained traffic patterns. Distributed systems often have varying operational characteristics, such as deployment cycles, API surges, and variations in AI workloads, and this was deemed necessary for such workload diversity ([6], [8]).

To showcase the interactions with the benchmarks, the experimental pipeline that was used throughout the evaluation process is presented in Figure 2.

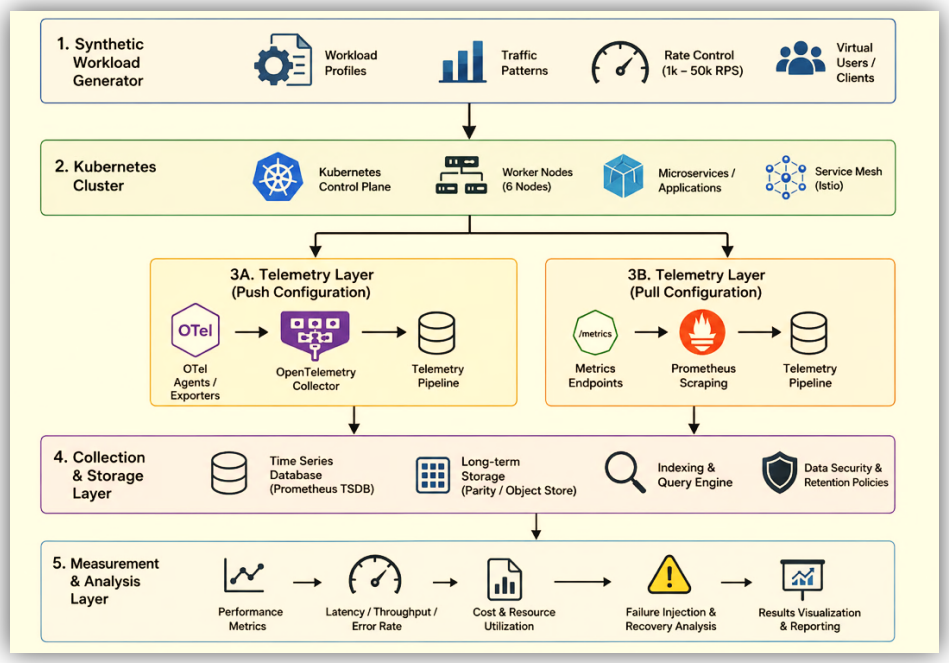

Figure 2. Benchmark Experimental Pipeline
Source: Author-designed benchmark workflow informed by cloud-native telemetry architecture literature ([2], [9], and [13]).

Figure 2 illustrates the sequential relationship among workload generation, telemetry transmission and performance measurement processes. The workflow illustrates the way that the telemetry systems were evaluated in the same environment which helps to minimize mis-methodologies often found in distributed benchmarking research.

With the lower load tests, the workload was around 1,000 requests per second and in the higher load situations, over 50,000 requests per second. The traffic generation rates were based on the operational conditions found in large-scale distributed infrastructures and cloud deployment environments [10] and [11]. In AI-assisted telemetry environments, the telemetry bursts may also be irregular when training the model or conducting large-scale inference tasks ([15]).

Evaluation Metrics

To gain insight into the performance, several quantitative measures were chosen based on standard practices in the field of distributed systems research. To measure the end-to-end telemetry delay, the P50, P95, and P99 latency percentiles were used in addition to average measurements, as percentile analysis can give a deeper understanding on tail behavior than just average measurements does ([3]). The throughput measurements were used to assess the performance of the processing of the telemetry events under different traffic mix.

The resource utilization metrics also measured the CPU usage, memory usage, and network overhead of telemetry processing activities. The study of distributed infrastructure analysis also found that there can be significant computational overhead associated with telemetry mechanisms, when collection systems are heavily overloaded ([7], [10]). The cost efficiency was evaluated using the estimates of the cost of processing the events in a telemetry system and the cost of storage. Failure-oriented measurements were also used to assess the growth of the queues, telemetry loss, and recovery time after induced failures.

Together, these metrics allowed for the holistic assessment of cloud-scale telemetry operational performance, telemetry scalability, resilience behavior, and infrastructure efficiency as well as aligning with current research goals in observability ([5], [12]).

DISCUSSION

End-to-End Latency Analysis

The accuracy of observability, anomaly detection, and incident response is all dependent on the latency behavior of telemetry and this was one of the most significant performance metrics in the benchmark. Studies

on latency variation, conducted in many different settings, have consistently demonstrated that latency variation impacts operational visibility and can impact downstream analytical systems to a great extent ([3], [13]). The benchmark showed that there are quantifiable differences between the architectures of push and pull based telemetry with percentile distributions. Under the burst intensive traffic condition, push telemetry systems showed better median latency as the continuous transmission of events was independent of centralized scheduling systems and it was possible to have asynchronous buffering. On the other hand, pull-oriented systems showed greater consistency for stable workloads as there was a predictable telemetry retrieval behavior with a scraping interval.

Latencies were compared quantitatively using percentile measurements, given the same workloads. The latency results of the two different telemetry architectures are summarized and compared in Table III.

Table III: **Telemetry Latency Percentile Comparison**

System	P50 (ms)	P95 (ms)	P99 (ms)
Push Telemetry	18	41	88
Pull Telemetry	25	52	97

Source: Author-generated benchmark results from Kubernetes experimental environment.

Table III shows that push based telemetry systems always had lower latency values in percentile measurements. The variation was greatest at the P95 and P99 points with push telemetry having better tail performance. The results indicate that with high event generation rates, asynchronous transmission and local buffering have a beneficial effect on responsiveness. This is consistent with earlier research on OpenTelemetry and distributed tracing that shows forwarders that are forwarded immediately lead to better event availability for incident analysis systems [13]. The same results have been observed in studies on AI-enabled incident response, as the timeliness of the telemetry signal has shown to impact the effectiveness of automated remediation [15].

To show the distribution of latency seen across the benchmark, a conceptual comparison of the behaviour of the percentile telemetry is shown in Figure 3.

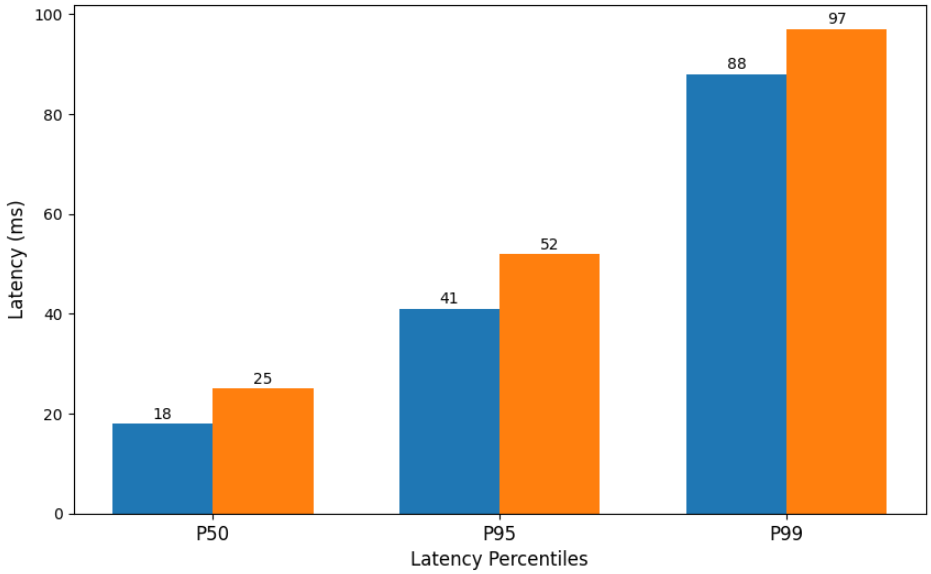

Figure 3. Latency Distribution across Telemetry Architectures
Source: Author-generated visualization derived from benchmark observations**.**

As shown in figure 3, latency divergence becomes more pronounced as the percentile goes up. Tail-latency is operationally important as the distributed systems often have irregular bursts and transient infrastructure conditions. Previous works on measuring the performance of hard real-time cloud systems have also demonstrated that a percentile-based approach offers more valuable information on the operation of the system than only average latency measurements ([10]).

Cost per Event Analysis

Another important benchmark outcome was operational cost as telemetry pipelines are increasingly running in cloud scale, processing billions of events. Telemetry collection systems can be costly due to requirements for data storage, data transmission over networks, and data processing. Previous experiences

have shown that the amount of telemetry collected in each Kubernetes cluster can significantly impact infrastructure costs in the cloud-native world ([9]).

The results of the experiments showed that, in general, the pull telemetry system was less resource-intensive when the workload was moderate as it reduced the frequency of transmissions due to the centralization of the scraper. With dynamically changing environments with rapidly changing populations of microservices, however, pull systems lead to increased endpoint discovery activity and higher polling overhead. The amount of bandwidth used by push telemetry architectures was higher during high volume of telemetry traffic due to the continuous flow of event streams. However, with a burst scenario, the cost of event processing was reduced by using asynchronous batching mechanisms.

These findings are consistent with previous work with distributed infrastructure analysis and Kubernetes telemetry orchestration, in which the cost of telemetry was significantly dependent on workload variation, as well as the architecture itself ([3, 12]). Furthermore, service mesh environments can exacerbate the amount of telemetry created on the network level as sidecar proxies are constantly generating network-level operational data. The study of CI/CD convergence further indicates that instead of the typical tradeoff between observability depth and infrastructure cost, scalable telemetry systems are required to rapidly deploy at scale ([6]).

Based on these benchmark results, it is therefore concluded that the cost of telemetry is not independent of workload features. Lower operational cost telemetry architectures may fail to become efficient when the workload is dynamic during service expansion or when it is burst.

Backpressure under Sustainable load

Sustained workload scenarios resulted in valuable understanding of telemetry resilience and backpressure actions. Backpressure is a situation where the rate at which the telemetry is being generated is higher than the processing or transmission rate, leading to a queuing up of telemetry and loss of events. Propagation of backpressure is a key research issue identified in distributed systems impacting performance stability and operational reliability ([7]).

Under high load conditions (above 50,000 requests per second), significant differences were observed between the telemetry architectures, as evidenced by the Benchmark observations. Initially, the push telemetry systems had larger throughput due to the local buffering mechanism absorbing the peaks in the telemetry events. OpenTelemetry collectors achieved load redistribution of workload pressure during moderate congestion periods, which helped avoid immediate telemetry loss. This phenomenon of telemetry buffering has been mentioned in various Kubernetes-wide telemetry systems with adaptive collection systems that led to more stable operations ([4]).

In push-oriented architectures, however, the effects of queue saturation occurred in the presence of long queues. Event latency increased significantly when collector buffers were filled, and throughput degradation was seen after that. Pull based telemetry systems showed varying characteristics. In the centralized approach, the collection schedules were predictable, rather than dependent upon queues. A few transient telemetry states were not observed but the degradation in operations was more gradual.

These results align with the results of microservice performance testing and with the results of other research on distributed analysis based on asynchronous queues ([3], [7]). Cloud-native observability using kernel-level telemetry mechanisms also points to the use of adaptive instrumentation methods to enhance sustained-load observability [14]. The more resilient the architecture, therefore, the more dependent it seems as a whole on the length of time the traffic is active and how long the congestion lasts.

Failure Recovery Analysis

A key benchmark dimension was failure recovery behaviour, as cloud-native infrastructures are often subject to failures such as node failures, collector restarts and network issues. Telemetry continuity plays an important role in automated operational environments where lack of operational information can delay the anomaly detection or fault localization processes [5] and [15].

Benchmarks of collector restarts and network interruptions showed that different architectures of telemetries have different recovery properties. The short term resiliency was shown by push systems, in which locally, the agents kept telemetry data in buffers for a while. After collector restoration, events were re-transmitted in the queue with limited amount of telemetry losses. The pull systems restarted the collection process using endpoint rediscovery and scheduled the scraping process. The consistency of recovery stayed at the same level but during outage periods, telemetry collected during that time was not always recoverable.

The results corroborate past research on distributed trace correlation and distributed root cause analysis models that showed that maintaining a continuous telemetry was a key factor in improving the accuracy of incident reconstruction [5], [13]. The survey results, which were part of research on multi-cloud observability

architectures, also showed that availability of telemetry was a significant factor in the overall observability of a system [9].

Discussing Trade-offs

The overall results of the benchmark show that neither of the telemetry paradigms was consistently superior to the other over all operational conditions. Rather, performance disparities became apparent based on workload behaviour, operation requirements and infrastructure goals. The study revealed that push telemetry systems were more flexible whenworking with bursts, as the results obtained with asynchronous transmission and buffering minimized the immediate congestion effects. These qualities are similar to what was found in the operational characteristics of infrastructure systems and dynamic Kubernetes deployments [15].

In contrast, pull telemetry systems showed greater level of predictability and monitoring simplicity. Under stable conditions, the Prometheus style scraping models yielded predictable collection behavior and transparence into their operation. These attributes help accomplish maintenance needs that are stressed in cloud-based deployment studies and DevOps operation contexts [8], [11].

Previous work on observability, service instrumentation, telemetry security, and distributed operational intelligence has mostly focused on a single instrument of telemetry ([1], [2], [4], [12], and [14]). Extending these findings, the present benchmark compares the empirical observability across the different telemetry transport architectures, and not individual observability components. The results show that the telemetry architecture should be chosen considering the dynamic nature of the workload and the operational context rather than being based on optimal deployments for all scenarios. Hybrid architectures that combine flexibility in buffering with centralized query semantics thus become potential solutions for the future of cloud-native observability architectures.

CONCLUSION

In this study, the authors found a benchmark that could be easily repeated against the push-based and pull-based telemetry architectures in a cloud-native, Kubernetes-based environment. The evaluation was conducted on the latency behavior, cost effectiveness, back pressure behavior, and failure recovery properties with the OpenTelemetry collectors and Prometheus scraping systems. It was found from the empirical results that push-based telemetry systems tend to have lower end-to-end latency in the presence of bursty workloads, which is seen in distributed tracing and incident analysis systems [13]. While pull-based architectures showed greater stability of operational predictability in steady state workloads, this was a result of design principles highlighted in cloud observability and Kubernetes native monitoring systems as referred to in [9].

The results also showed that the behavior of the cost of telemetry and throughput is very workload dependent: push telemetry systems have higher continuous transmission costs and better burst absorption, and pull telemetry systems have lower operational telemetry semantics but higher costs because of scraping inefficiencies at scale. The results of backpressure analysis indicated that push architectures have the ability to withstand long-term saturation by buffering of the queues, however, may experience degradation during longer saturations; pull architectures on the other hand showed degradation over time as suggested in literature of distributed performance analysis on queue stability and system stress behavior ([3], [7]). Failure recovery experiments also showed that push systems offer improved continuity of telemetry signals during transient failures as multi-cloud observability results showed in [9], whereas pull systems depend on the availability of endpoints and depend on failure recovery scheduling.

Overall the results of the study clearly demonstrate that there is no single optimum telemetry architecture. Rather, performance compromises are required for different workloads, sizes of infrastructures, and operations over distributed system architectures, such as AI powered platforms and microservice oriented systems ([15]).

Future Work

This benchmark can be expanded to multi-cluster and edge-distributed Kubernetes platforms on which network variability can have a strong influence on telemetry propagation in future research. More research should be carried out in adaptive hybrid telemetry where push/pull modes are dynamically changed according to workload prediction and system congestion. Combining AI-based telemetry optimization mechanisms, as proposed by automated incident resolution research, can improve system resilience and efficiency even more ([15]). Additionally, the research on eBPF and Istio based observability can lead to better fine-grained profile and less overhead in large-scale deployments as discussed in the kernel-level and service-mesh enhanced telemetry pipelines [2], [14].

REFERENCES

[1] Baeten, M. (2023). Microservice coverage detection. http://hdl.handle.net/1942/41381

[2] Calcote, L., & Butcher, Z. (2019). *Istio: Up and running: Using a service mesh to connect, secure, control, and observe.* O'Reilly Media.

[3] Denys, P. F. (2023). *Distributed Performance Analysis Tools for Large Scale Computations.* Ecole Polytechnique, Montreal (Canada).

[4] Gaddam, R. R., & Krishna, K. (2022). Kube Agent Hardening for Fleet-Wide Secure Telemetry. *International Journal of Emerging Research in Engineering and Technology, 3*(3), 148-158. https://doi.org/10.63282/3050-922X.IJERET-V3I3P115

[5] Gan, Y., Liu, G., Zhang, X., Zhou, Q., Wu, J., & Jiang, J. (2023, March). Sleuth: A trace-based root cause analysis system for large-scale microservices with graph neural networks. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4* (pp. 324-337). https://doi.org/10.1145/3623278.3624758

[6] Gorak, P. (2023). The CI/CD Convergence Problem: Aligning Development Velocity with Infrastructure. *IJSAT-International Journal on Science and Technology, 14*(3).

[7] Kabamba, H. M. (2023). *Approches intégrées d'analyse de performance des systèmes distribués asynchrones par trace d'exécution système* (Doctoral dissertation, Ecole Polytechnique, Montreal (Canada)).

[8] Madamanchi, S. (2021). *Google Cloud for DevOps Engineers: A practical guide to SRE and achieving Google's Professional Cloud DevOps Engineer certification.* Packt Publishing Ltd.

[9] Mohamed, Haytham, "Towards an Efficient Multi-Cloud Observability Framework of Containerized Microservices in Kubernetes Platform" (2022). *Masters Theses & Doctoral Dissertations.* 401. https://scholar.dsu.edu/theses/401

[10] Murphy, A. C. (2022). *Hard-Real-Time Computing Performance in a Cloud Environment* (Doctoral dissertation, Old Dominion University).

[11] Rajasekharaiah, C. (2020). Core cloud concepts: compute. In *Cloud-Based Microservices: Techniques, Challenges, and Solutions* (pp. 119-153). Berkeley, CA: Apress. https://doi.org/10.1007/978-1-4842-6564-2_7

[12] Scano, D., Giorgetti, A., Paolucci, F., Sgambelluri, A., Chammanara, J., Rothman, J., ... & Cugini, F. (2023). Enabling P4 network telemetry in edge micro data centers with kubernetes orchestration. *IEEE Access, 11*, 22637-22653. https://doi.org/10.1109/ACCESS.2023.3249105

[13] Seknametla, P. R. (2023). Automated Root Cause Analysis in Microservice Architectures: Leveraging Distributed Trace Correlation with OpenTelemetry for Faster Incident Resolution. *International Journal of Emerging Research in Engineering and Technology, 4*(1), 158-164. https://doi.org/10.63282/3050-922X.IJERET-V4I1P117

[14] Sharma, B. (2023). Improving microservices observability in cloud-native infrastructure using EBPF (Master's thesis, Purdue University).

[15] Veluru, S. P. (2021). Leveraging AI and ML for automated incident resolution in cloud infrastructure. *International Journal of Artificial Intelligence, Data Science, and Machine Learning, 2*(2), 51-61. https://doi.org/10.63282/3050-9262.IJAIDSML-V2I2P106