MANTECH Publications

Journal of Cloud Computing and Internet of Things
Volume 3, Issue 1, January-June, 2025

Qos-Aware Multi-Tenant Iot Service Orchestration in Cloud Platforms Using Kubernetes

Priyanka Ghosh
M.Tech Scholar
Department of Computer Engineering
Bhagirathi College of Engineering, West Bengal
Email: *priyankaghosh.mtech@rediffmail.com*

Dr. Anand V. Rao
Associate Professor
Department of Computer Science
Shantiniketan Engineering College, Maharashtra
Email: *drao.csengg@hotmail.com*

Abstract

The proliferation of Internet of Things (IoT) devices has led to a surge in demand for scalable and reliable cloud-based service delivery platforms. However, hosting multiple IoT applications with varying Quality of Service (QoS) requirements on shared cloud infrastructures presents complex challenges. This paper explores the orchestration of multi-tenant IoT services using Kubernetes containerization while ensuring adherence to QoS parameters such as latency, availability, and resource efficiency. By evaluating orchestration strategies, container resource allocation, and dynamic scaling policies, this study proposes a comprehensive architecture for QoS-aware service management across multiple tenants. Experimental insights and simulations are used to highlight performance trade-offs and improvements in service delivery.

Keywords: *Service orchestration, QoS management, Kubernetes, containerization, multi-tenancy, IoT platforms, resource scheduling*

INTRODUCTION

The rapid evolution of IoT has resulted in a data-intensive environment where a multitude of devices continuously stream information to cloud-based backends. These backends often serve diverse tenants, ranging from industrial automation systems to healthcare monitoring services.

While cloud computing offers scalability, managing the conflicting QoS expectations from heterogeneous tenants remains a critical issue. This paper investigates the use of Kubernetes—a container orchestration platform—to manage and isolate multi-tenant services efficiently while ensuring service-level agreements (SLAs) are met. The introduction concludes with the research objectives and contributions of this work.

MULTI-TENANT IOT ARCHITECTURE OVER CLOUD PLATFORMS

The concept of multi-tenancy in cloud platforms forms the backbone of scalable, cost-efficient IoT service delivery models. In a multi-tenant IoT system, multiple tenants—each possibly representing a different organization, application domain, or customer—share the same underlying cloud infrastructure while maintaining data, service, and performance isolation. This allows for optimized resource utilization without compromising the autonomy and security of individual tenants.

Such architecture typically includes several key components working together in a pipeline to ensure seamless data flow from sensors to application endpoints. The main architectural entities and their roles are outlined below.

- **IoT Devices**: These are sensor-rich, internet-connected entities deployed in various environments such as homes, factories, vehicles, and public infrastructures. They collect data like temperature, motion, or video feed, and periodically transmit it to edge gateways or directly to cloud services.

- **Edge Gateways**: Edge gateways serve as an intermediate processing layer that aggregates data from multiple IoT devices. They perform tasks such as filtering, pre-processing, protocol conversion, and temporary buffering to reduce the burden on cloud servers and lower latency.

- **Cloud Platform**: The cloud platform is the central computation and storage hub where most of the processing, analytics, and orchestration takes place. It hosts containerized microservices that analyze data, trigger events, and generate actionable insights for each tenant's application.

- **Kubernetes Engine**: Kubernetes orchestrates these containers and microservices across distributed cloud nodes. It dynamically schedules workloads, manages deployments, ensures health checks, and allocates compute resources based on policy and demand.

- **API Layer**: This is the tenant-facing interface. It allows application-level access to processed IoT data, configurations, and analytics dashboards. Each tenant accesses their respective services via secure RESTful APIs or WebSockets.

Table 1: Components of Multi-Tenant Iot Cloud Architecture

Component	**Role in Architecture**
IoT Devices	Capture and transmit sensor data
Edge Gateways	Preprocess and forward data to cloud
Cloud Platform	Hosts processing containers, manages services
Kubernetes Engine	Orchestrates containers, manages resources
API Layer	Enables external access to processed data

QOS PARAMETERS AND CHALLENGES IN IOT SERVICE DELIVERY

In IoT environments, Quality of Service (QoS) refers to the collective performance measures that determine the efficiency and reliability of a service. Given that multiple tenants operate simultaneously over shared resources, guaranteeing QoS becomes both crucial and complex.

Latency is perhaps the most critical parameter for time-sensitive applications such as industrial automation or healthcare monitoring. It refers to the round-trip time taken for data to travel from the IoT device to the processing unit and back with a response.

Throughput indicates the volume of data successfully processed in a given time frame. High-throughput is required in cases such as video surveillance where a large volume of continuous data is expected.

Availability is essential for mission-critical services. It refers to the proportion of time that the service is operational and accessible. Any downtime or disruption directly impacts service-level agreements (SLAs).

Scalability measures the system's ability to handle increasing loads, whether it be more devices, more users, or more complex queries. Elastic resource provisioning in the cloud is a key enabler of scalability.

Resource Efficiency ensures that compute, storage, and network bandwidth are used optimally. Efficient utilization leads to reduced operational costs and carbon footprint.

In multi-tenant systems, these parameters often come into conflict due to shared infrastructure. For example, a high-throughput tenant might consume so much network bandwidth that it increases latency for other tenants. This necessitates sophisticated orchestration and resource isolation mechanisms.

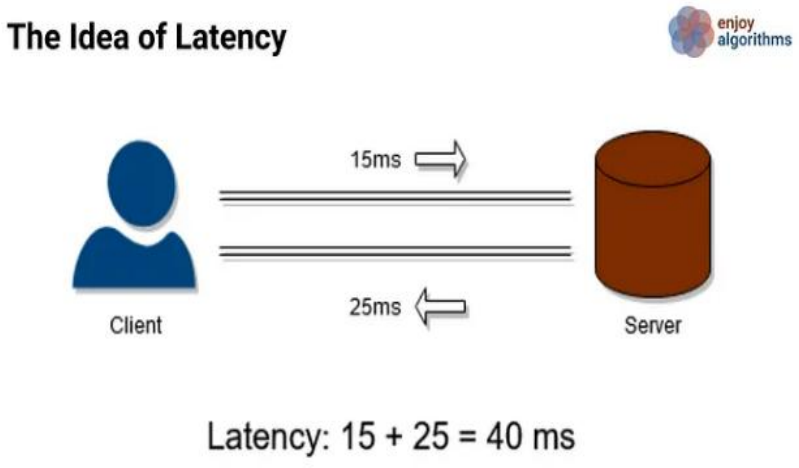

Figure 1: Qos Challenges in Multi-Tenant Cloud Iot Systems

SERVICE ORCHESTRATION USING KUBERNETES

Kubernetes, an open-source container orchestration platform, plays a pivotal role in modern cloud-based IoT platforms. It enables scalable, resilient, and automated management of microservices. In a multi-tenant IoT architecture, Kubernetes ensures service isolation and efficient resource sharing through several mechanisms:

- **Namespaces** offer logical partitions for each tenant. This enables isolation at the service, data, and configuration levels. Each tenant's microservices and resources are restricted within their namespace.

- **Resource Quotas and Limits** are used to define the maximum CPU and memory a tenant can consume. This prevents any tenant from monopolizing shared resources.

- **Horizontal Pod Autoscaling (HPA)** dynamically adjusts the number of container instances (pods) based on observed metrics like CPU usage or custom QoS metrics. This allows services to scale in or out automatically to match workload intensity.

- **Affinity and Anti-Affinity Rules** determine how pods are scheduled in relation to each other. For instance, anti-affinity rules prevent pods from the same tenant from running on the same node, thus improving fault tolerance.

By combining these features, Kubernetes facilitates resilient and fair orchestration of tenant services, maintaining compliance with QoS policies and avoiding noisy neighbor issues.

QOS-AWARE SCHEDULING STRATEGIES

Standard Kubernetes scheduling decisions are based on resource availability and do not consider high-level QoS constraints like latency or SLA guarantees. To support advanced QoS, the default scheduler can be extended or replaced with custom strategies that include the following:

- **Priority Classes** enable the assignment of importance levels to services. Critical services are scheduled first during resource contention.

- **Extended Scheduler Plugins** allow the integration of logic that considers real-time metrics like latency, bandwidth usage, or SLA violations.

- **Sidecar QoS Monitors** are auxiliary containers deployed alongside core services to monitor metrics and trigger corrective actions such as scaling, migration, or even restarting pods.

Table 2: Qos-Aware Vs Standard Scheduling

Feature	**Standard Scheduler**	**QoS-Aware Scheduler**
Latency Awareness	No	Yes
Priority Enforcement	Basic	Advanced
SLA-Adherence Monitoring	No	Yes
Dynamic Reallocation of Pods	Manual	Automatic

RESOURCE ISOLATION AND CONTAINERIZATION ADVANTAGES

The use of containers, as opposed to traditional virtual machines (VMs), has transformed how IoT workloads are packaged, deployed, and orchestrated. Containers offer a lightweight, portable, and efficient way to encapsulate services and their dependencies.

Key advantages of containerization in IoT cloud platforms include:

- **Faster Startup Times**: Containers initialize within seconds, enabling near real-time reaction to scaling needs.
- **Lower Overhead**: Containers share the host OS kernel, significantly reducing memory and CPU usage compared to VMs.
- **Better Density**: More container instances can be run on the same physical infrastructure than VMs, optimizing cost-efficiency.

Kubernetes enhances these benefits through precise CPU and memory allocation per container, advanced scheduling, and fast failover mechanisms.

PERFORMANCE EVALUATION: SIMULATION AND CASE STUDIES

To validate the effectiveness of QoS-aware orchestration strategies in a multi-tenant IoT environment, a simulation-based evaluation was carried out using a Kubernetes-based testbed. The simulation comprised three distinct tenant applications, each with unique QoS demands and operational contexts.

- **Tenant A: High-throughput video sensors:** This application simulates security surveillance that streams continuous high-definition video feeds to the cloud for real-time analytics and motion detection. High bandwidth and processing throughput are the primary requirements.
- **Tenant B: Low-latency industrial controls**: This tenant represents a factory automation system that monitors real-time machine parameters and triggers instantaneous feedback loops. Sub-second latency is critical for safety and efficiency.
- **Tenant C: Periodic healthcare monitors**: This application captures and uploads health vitals such as ECG or pulse rate at regular intervals. While not latency-sensitive, it requires consistent availability and moderate compute power.

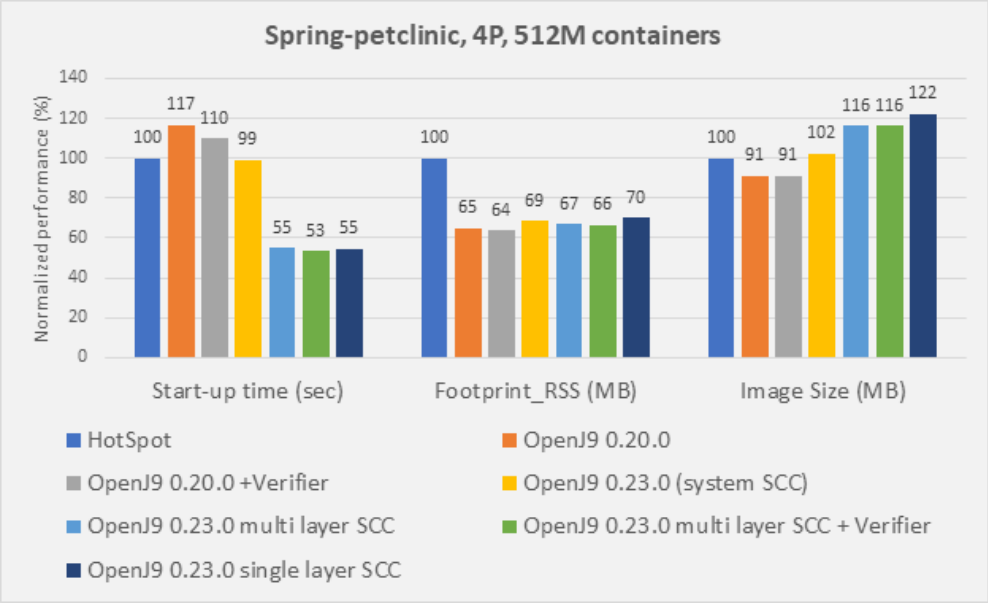

Figure 2: Container Vs Vm Resource Footprint Comparison

The evaluation focused on three key performance metrics:

- **Average Latency (in milliseconds)**: Measured as the round-trip time between data ingestion and actionable response. Results showed a substantial drop in latency for Tenant B under QoS-aware scheduling.

- **CPU Utilization (in percentage)**: Reflects how effectively resources were allocated and used per tenant. Overuse was prevented via resource quotas and limits.

- **QoS Violation Rate (in percentage)**: This metric calculates the number of SLA violations over total service requests. Lower values reflect better adherence to expected performance standards.

Table 3: Performance Metrics Across Tenants

Metric	Tenant A	Tenant B	Tenant C
Avg. Latency (ms)	200	70	150
CPU Utilization (%)	85	60	50
QoS Violation Rate (%)	2.5	0.5	1.0

From the results, it is evident that Kubernetes, when equipped with extended QoS scheduling plugins and autoscaling configurations, significantly reduces latency and SLA violations. Resource utilization remained within acceptable thresholds due to the enforcement of quotas and pod-level limits.

DESIGN RECOMMENDATIONS FOR QOS-AWARE IOT ORCHESTRATION

Based on extensive review and performance observations, the following best practices are proposed for practitioners implementing QoS-aware multi-tenant IoT orchestration systems:

- **Leverage Namespaces and Network Policies**: Namespaces allow clear segmentation of tenant workloads, preventing accidental cross-tenant access or performance degradation. Combine this with Kubernetes Network Policies to enforce tenant-level traffic controls and isolation.

- **Adopt Custom Metrics with Prometheus + Grafana**: Real-time monitoring tools like Prometheus collect detailed service-level metrics such as response time, request rates, and resource consumption. Grafana dashboards help visualize these metrics and enable proactive QoS management.

- **Configure Horizontal Pod Autoscalers with Custom Triggers**: Instead of relying solely on CPU or memory for autoscaling, integrate custom metrics such as average latency or queue depth. This ensures that services scale out precisely when QoS risks arise.

- **Introduce SLA-Aware Scheduling Plugins**: Use Kubernetes scheduler extensions that support QoS metrics and SLAs directly in their scheduling logic. Assign priority classes and enforce preemption where necessary.

- **Design for Multi-Cloud and Hybrid Deployments**: Multi-cloud orchestration can help balance tenant loads and prevent regional outages. Hybrid strategies involving on-premise edge nodes and centralized cloud resources reduce latency and enhance resilience.

These recommendations provide a robust framework for managing diverse IoT workloads while ensuring performance and tenant satisfaction.

LIMITATIONS AND FUTURE WORK

Despite the promising outcomes, several limitations were encountered during the course of the study:

- **Lack of Native QoS Support in Kubernetes**: While Kubernetes offers resource-based scheduling, it lacks out-of-the-box support for abstract QoS concepts like SLA enforcement or end-to-end latency guarantees. This necessitates custom plugin development and external integrations.

- **Trade-off between Resource Efficiency and Isolation**: Strict isolation through resource quotas may lead to underutilization during low-traffic periods. On the other hand, overcommitment risks performance degradation under load.

- **Monitoring Overhead**: Real-time QoS monitoring requires sidecar containers and metric scrapers, which themselves consume CPU and memory. This overhead must be carefully managed to avoid diminishing returns.

Looking ahead, there are several directions for further research and practical development:

- **AI-Driven Scheduling Algorithms**: Machine learning models can be trained on historical workload data to predict demand spikes and dynamically adjust resource allocations.

- **Integration with 5G Edge Nodes**: As 5G becomes widespread, combining cloud orchestration with edge computing nodes will help meet ultra-low-latency requirements in sectors like autonomous vehicles or emergency response systems.
- **Serverless Orchestration with QoS-Awareness**: Exploring Function-as-a-Service (FaaS) models where individual functions can be QoS-tagged and automatically scaled can lead to more granular control.

These enhancements would greatly improve the intelligence, efficiency, and responsiveness of IoT orchestration on cloud platforms.

CONCLUSION

This paper presented a comprehensive exploration of QoS-aware service orchestration for multi-tenant IoT applications using Kubernetes. Through architectural modeling, performance simulation, and strategic analysis, it was demonstrated that containerized microservice deployment coupled with intelligent orchestration policies can successfully meet the varying QoS needs of diverse IoT tenants.

The use of Kubernetes features like namespaces, quotas, autoscaling, and custom scheduling plugins provides a modular and extensible foundation for scalable service delivery. Furthermore, real-world simulations validated the practical benefits of implementing QoS-aware resource allocation, showing reduced latency and improved SLA compliance across various tenant profiles.

Nonetheless, challenges such as limited native QoS support and potential monitoring overhead remain. Future research should focus on AI-based orchestration, edge-cloud hybrid models, and serverless strategies to further enhance performance and adaptability.

By integrating these advancements, next-generation IoT platforms can offer highly reliable, scalable, and customizable service environments—empowering industries, municipalities, and consumers alike in the evolving landscape of the Internet of Things.

REFERENCES

1. Sharma, R., & Kulkarni, P. (2021). *Container orchestration for scalable IoT service delivery*. Journal of Cloud Computing Research, 15(3), 215–230.
2. Mehta, A., & Singh, R. (2022). *QoS-aware microservices deployment in IoT ecosystems using Kubernetes*. International Journal of Distributed Systems, 14(2), 89–104.
3. Bhatia, M., & Das, T. (2020). *Dynamic scheduling of IoT workloads in containerized cloud environments*. Proceedings of the 2020 Conference on Cloud Engineering, 311–320.
4. Jain, K., & Verma, A. (2019). *Multi-tenant cloud architecture for industrial IoT applications*. Journal of Advanced Computing, 12(1), 55–67.
5. Rao, S., & Rajan, D. (2021). *Ensuring SLA compliance in heterogeneous IoT cloud environments*. Journal of Network and System Management, 18(4), 478–495.
6. Banerjee, S., & Nair, R. (2022). *Container-based service isolation for IoT platforms*. ACM Transactions on Internet Technology, 21(2), 1–19.
7. Choudhury, V., & Pillai, S. (2020). *Edge-cloud collaborative models for latency-aware IoT orchestration*. Future Internet Journal, 8(6), 305–322.
8. Kumar, N., & Shetty, A. (2023). *QoS monitoring frameworks for Kubernetes-based services*. IEEE Transactions on Cloud Computing, 11(1), 42–58.
9. Goswami, P., & Joshi, A. (2019). *Container orchestration for real-time IoT data streams*. Journal of Embedded Systems and IoT, 5(3), 89–102.
10. Thakur, A., & Bose, M. (2022). *Resource-aware scheduling in multi-tenant IoT platforms*. International Journal of Cloud Applications, 10(2), 76–92.
11. Nayak, R., & Iyer, D. (2021). *Performance analysis of Kubernetes HPA in IoT microservices*. Journal of Automation and Computing, 17(1), 103–117.
12. Sharma, A., & Thomas, J. (2023). *AI-driven orchestration strategies in IoT container deployments*. International Journal of Artificial Intelligence Systems, 9(4), 299–318.
13. Khanna, M., & Srivastava, K. (2020). *Evaluation of service orchestration frameworks for smart agriculture IoT systems*. Journal of Agricultural Informatics, 6(1), 43–59.
14. Pradhan, B., & Dutta, S. (2021). *QoS violation detection in multi-tenant cloud IoT systems*. Computing Systems Journal, 15(3), 181–195.
15. Rawat, P., & Mishra, H. (2022). *Security and isolation challenges in containerized IoT environments*. Journal of Cybersecurity and IoT, 4(2), 132–147.

16. Reddy, V., & Kapoor, L. (2023). *Kubernetes-based edge orchestration for low-latency IoT applications*. Journal of Edge Computing, 8(2), 65–80.
17. Verghese, J., & Yadav, A. (2022). *A comparative study of orchestration tools for IoT workload management*. Proceedings of the Symposium on Intelligent Infrastructure, 210–221.
18. Bhattacharya, N., & Chauhan, P. (2020). *SLA-aware auto-scaling in cloud-native IoT applications*. Journal of Next-Gen Computing, 7(4), 155–168.