A Kubernetes-Based Architecture for Tier-Aware Multimedia Streaming with Quality-of-Service Management

Adelina-Nicoleta Pirjol
Communications Department
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
pirjol.co.adelina@student.utcluj.ro

Robert Botez
Communications Department
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
robert.botez@com.utcluj.ro

***Abstract*- This paper introduces a Kubernetes-based multimedia streaming application with microservices. It separates free and premium users through dedicated services and bandwidth control using Kube-OVN. Monitoring is done via Prometheus and Grafana. The system is cloud-native and deployable locally, demonstrating effective Quality-of-Service.**

Keywords—cloud-native, Kubernetes, microservices, Quality-of-Service, Prometheus

I. INTRODUCTION

Multimedia streaming platforms have grown in complexity, requiring scalable architectures that support user-tier differentiation. This paper proposes a Kubernetes-based, cloud-native solution built on microservices and container orchestration, enabling separation between free and premium users through network Quality-of-Service (QoS) policies. The system integrates Prometheus and Grafana for monitoring and demonstrates how Kubernetes extensions can enforce user-aware resource control in distributed environments [1].

I. RELATED WORK

Recent trends in software architecture tend to use cloud-native systems that promote modularity and scalability. As emphasized by Abou El Kalam et al., Kubernetes plays an essential role in managing independent software components (microservices), allowing each to be updated or restarted without affecting the entire application [2].

Providing differentiated service levels is especially important in multimedia platforms that offer both free and premium user experiences. Kubernetes allows this distinction through network rules and bandwidth restrictions applied individually to application components. For example, the Kube-OVN plugin supports bandwidth limits per container, and Calico, a networking solution, enables strict control based on user category [3], [4].

Monitoring tools help keep applications responsive even when usage fluctuates. Prometheus, combined with Grafana, enables real-time tracking of system performance, helping developers detect problems quickly and scale resources automatically when needed [5].

II. IMPLEMENTATION

The proposed application adopts a microservice-based architecture designed for multimedia content delivery with differentiated user experiences. The system is deployed locally using Minikube to simulate a Kubernetes cluster, and it enables streaming functionality for two distinct user tiers: Free and Premium. Modularity is made possible by the general design, which ensures that each service is logically and functionally separated.

A container named *frontend-deployment* is responsible for handling user interactions, maintaining session state, and acting as a gateway that routes requests to the appropriate backend services. It acts as the backbone of the application, forwarding user actions such as authentication, search, and playlist operations to the corresponding components. When handling user authentication, the container stores session details, including the user's tier (free or premium), and based on this information, it directs streaming requests either to the Free Service or the Premium Service. This allows the system to enforce different network quality constraints. Three microservices interact with the frontend, as in Fig. 1: the authentication service, the playlist service, and the search service. The authentication service is responsible for user registration and login, as well as tier classification. It is the component that stores data, utilizing a lightweight SQLite database for user credentials and tier information. The playlist service manages user-created playlists and their contents, while the search service handles video discovery through external Application Programming Interface (API) such as the YouTube Data API, a service provided by the Google Cloud platform [6].

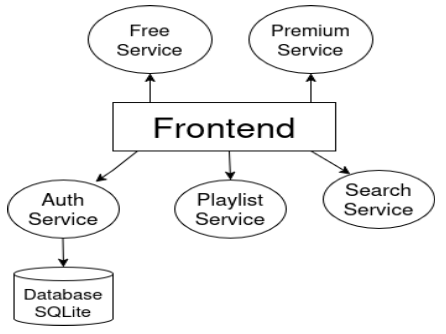


Fig. 1. Diagram of the application

To support differentiation between free and premium users, the application defines two separate streaming endpoints, referred to as the Free Service and the Premium Service. These are exposed through distinct Kubernetes service definitions, allowing different traffic rules to be applied. For instance, pods handling the Free Service have a bandwidth limit of 1 Mbps, configured using a Kube-OVN annotation (kube-ovn.io/egress-bandwidth: "1M"), while the Premium Service operates without any restriction. All components are deployed using Kubernetes Deployment objects, which ensure that a specified number of instances (pods) are always running and automatically replaced if they fail. Services that only need to be accessed within the cluster use the default communication mode (ClusterIP). The main entrypoint to the system—responsible for handling user interactions and forwarding requests—is exposed externally using the NodePort service type, which opens a specific port on the cluster node to receive traffic from outside.

Monitoring is implemented using Prometheus, which collects metrics from all running components. These metrics are then visualized in real-time through Grafana dashboards, enabling analysis of performance, user activity, and the effects of network restrictions. Docker is used as the container runtime, and all services run inside the default namespace of a local Minikube cluster [7].

This setup allows testing traffic control, user-tier separation, and monitoring within a Kubernetes-native environment.

III. Experimental Results

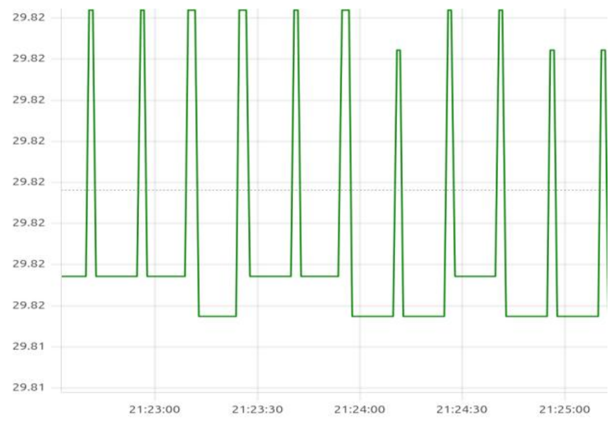
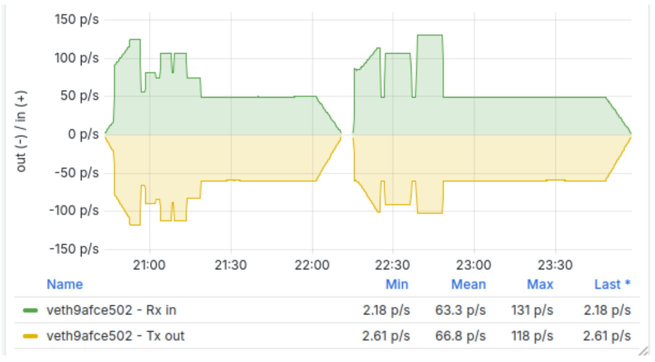


Fig. 2. Prometheus metrics

The cumulative network traffic received by the container handling user requests (interface veth9afce502) steadily increases over time, as reported by the node_network_receive_bytes_total metric in Prometheus. This trend, illustrated in Fig. 2, reflects consistent user activity such as starting streams or interacting with playlists.

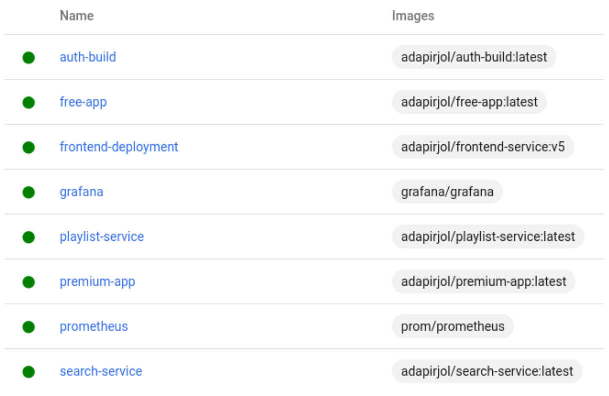


Name	Min	Mean	Max	Last *
🟩 veth9afce502 - Rx in	2.18 p/s	63.3 p/s	131 p/s	2.18 p/s
🟨 veth9afce502 - Tx out	2.61 p/s	66.8 p/s	118 p/s	2.61 p/s

Fig. 3. Grafana metrics

Real-time network traffic rates (Rx/Tx) show short bursts and symmetrical spikes that correspond to user-triggered events, indicating a balanced flow of data during streaming activity, as shown in Fig. 3.

Both graphs confirm that the frontend microservice maintains a stable reception rate over time and handles continuous traffic without drops, demonstrating the reliability of the deployed Kubernetes architecture under light-to-moderate load.



	Name	Images
●	auth-build	adapirjol/auth-build:latest
●	free-app	adapirjol/free-app:latest
●	frontend-deployment	adapirjol/frontend-service:v5
●	grafana	grafana/grafana
●	playlist-service	adapirjol/playlist-service:latest
●	premium-app	adapirjol/premium-app:latest
●	prometheus	prom/prometheus
●	search-service	adapirjol/search-service:latest

Fig. 4. Minikube dashboard

In Fig. 4 it is presented the Minikube dashboard that provides a graphical overview of the running deployments in the local Kubernetes cluster. Each microservice (e.g., frontend, auth, premium) is shown with its corresponding Docker image and current status, facilitating real-time monitoring and debugging.

V. Conclusions and Future Work

The project successfully delivered a functional multimedia streaming application using Docker, Kubernetes and microservices and it handles user tier separation including monitoring tools. The results confirm stable operation and real-time traffic visibility. The platform will be extended by deploying multiple streaming servers, adding caching, and routing users to the nearest server based on location. Further separation between free and premium users will be enforced through advanced traffic policies.

References

[1] A. G. Abou El Kalam, M. Laurent-Maknavicius, and Y. Deswarte, “Quality of service in Kubernetes-based microservices architectures: A survey and research directions,” *Future Generation Computer Systems*, vol. 127, pp. 186–202, 2022.

[2] M. Villamizar et al., “Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud,” 2015 10th Computing Colombian Conference (10CCC), IEEE.

[3] G. Lu et al., “NBWGuard: Network Bandwidth as a Resource in Kubernetes,” Proceedings of the ACM Symposium on Cloud Computing (SoCC), 2018.

[4] M. L. Girtler and M. Kosek, “Assessing Container Network Interface Plugins: Functionality, Performance and Scalability,” 2020 23rd International Symposium on Measurement and Control in Robotics (ISMCR), 2020.

[5] R. A. Fattah et al., “ML-based prediction for autoscaling Kubernetes workloads,” Frontiers in Computer Science, 2023.

[6] S. Newman, *Building Microservices*, 2nd ed. O'Reilly Media, 2021.

[7] C. Biewald, M. Weigel, and F. Hupfeld, "Monitoring Kubernetes Applications with Prometheus and Grafana," arXiv preprint arXiv:2007.00223, 2020