International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 06 | June - 2024 SJIF Rating: 8.448 ISSN: 2582-3930

Enhanced Visibility for Real-time Monitoring andAlerting in Kubernetes by Integrating Prometheus,Grafana, Loki, and Alerta

Karthik Pai
Department of ISE
R. V. College of Engineering®
Bengaluru, India

Prof. B.K Srinivas
Department of ISE
R. V. College of Engineering®
Bengaluru, India

***Abstract*—With the increasing popularity of Kubernetes as the go-to platform for containerized applications, numerous companies are adopting Kubernetes as their preferred platform for containerized applications. As organizations embrace this container orchestration technology for its scalability, flexibility, and portability benefits, the need for robust monitoring solutions becomes paramount. Monitoring Kubernetes environments is essential to ensure the health, performance, and availability of applications running within the cluster. This paper aims to provide a comprehensive approach for monitoring Kubernetes via Prometheus, Grafana, and Alerta**

Prometheus, a powerful open-source monitoring system, collects metrics from Kubernetes pods and nodes, enabling real-time monitoring of resource utilization, performance metrics, and application health. Grafana complements Prometheus by providing intuitive visualization of collected metrics through customizable dashboards, facilitating comprehensive insights into cluster performance and trends. Loki and Promtail by Grafana are used to collect and aggregate the logs associated with the cluster. Alerta enhances the monitoring setup by enabling alerting based on predefined thresholds and conditions, ensuring prompt notification of potential issues or anomalies.

Together, this stack empowers administrators to gain deep visibility into their Kubernetes infrastructure, proactively identify and mitigate issues, and maintain the high availability and reliability of their applications and services.

***Index Terms*—Kubernetes, Prometheus, Grafana, Alerta, Loki, Promtail, Monitoring, Alerting, Logging.**

I. Introduction

In today's dynamic computing landscape, Kubernetes has emerged as the de facto standard for orchestrating containerized applications. As organizations increasingly embrace Kubernetes for its scalability, resilience, and agility, the need for robust monitoring solutions to ensure the health and performance of applications running on Kubernetes clusters has become paramount. In response to this demand, monitoring tools such as Grafana, Alerta, and Prometheus have gained prominence for their ability to provide comprehensive visibility into the state of Kubernetes deployments.

Prometheus serves as a powerful monitoring and alerting system, specifically designed for cloud-native environments like Kubernetes. Prometheus excels in its ability to scrape and store time-series data, enabling the collection of metrics from various Kubernetes components, such as pods, nodes, and services. With its flexible querying language and robust alerting mechanisms, Prometheus empowers operators to define and manage alerts based on predefined thresholds and conditions, ensuring proactive detection and resolution of potential issues.

Grafana, with its intuitive dashboarding capabilities, allows operators and developers to visualize key metrics and performance indicators, facilitating effective monitoring and troubleshooting. Loki and Promtail by Grafana are used to collect and aggregate the logs associated with the cluster. These logs would further allow us to debug various issues within the cluster. By leveraging Grafana's rich set of visualization options and customizable dashboards, organizations can gain insights into the behavior of their Kubernetes applications, identify trends, and respond to emerging issues in real time.

In collaboration with Grafana and Prometheus, Alerta serves as a centralized alert management platform, streamlining the intricate process of alert deduplication, escalation, and resolution. While Prometheus offers its own alerting capabilities through Alertmanager, it's important to note some inherent limitations. The Alertmanager interface, while functional, may not always provide the most intuitive user experience, making it challenging to effectively discern various system conditions. Additionally, Alertmanager's scalability can pose challenges, particularly in environments with multiple Prometheus instances, necessitating individual Alertmanager instances for each Prometheus deployment.

This is where Alerta shines. By consolidating alerts from diverse sources, including Prometheus, Alerta delivers them to a unified dashboard, offering comprehensive visibility into the system's health. Through seamless integration with popular communication channels such as Slack and the flexibility to create custom plugins (e.g., for generating Jira tickets), Alerta enhances collaboration among teams and facilitates timely notification. This collaborative approach empowers teams to swiftly respond to incidents and work towards rapid resolution, ensuring minimal downtime and optimal system performance.

Together, Grafana, Alerta, and Prometheus form a potent monitoring stack that empowers organizations to gain actionable insights, streamline operations, and maintain the reliability and performance of their Kubernetes applications. By leveraging the capabilities of these tools, organizations can effectively monitor, manage, and optimize their Kubernetes deployments, ensuring they meet the demands of modern, cloud-native environments.

II. Literature Review

A variety of research papers have been published in this domain. The research by Carcassi et at.,[1] discussed on multi cluster monitoring using Thanos which acts as a centralized storage to all the Prometheus deployed in different clusters. Further more Grafana was used as a dashboard to visualize the data. The Grafana plots were later integrated at SLATE console.

Another paper by Ioannis Tzanettis et al.,[2] talks about about orchestration of distributed applications. It discusses using data fusion to improve the observability of these applications. Observability signals are used to monitor the health of the application. The authors propose a new method for data fusion that leverages machine learning. This method can be used to improve the decision-making process for orchestration.

The research by Ridwan Satrio Hadikusuma et al.,[3] article is about optimizing and monitoring Kubernetes clusters. It discusses different methods to achieve this goal. The authors analyze three journals that explore various approaches. They find that a Prometheus and Grafana for monitoring, and efficient cluster frameworks can all improve performance. It showcases how Prometheus and Grafana is effective for monitoring Kubernetes applications

The paper by P., Prerana et al.,[4] discusses a method for monitoring multiple clusters using Prometheus Operator and the standard prometheus.

The paper by Thanh-Tung Nguyen et al.,[5] investigates Kubernetes' HPA, comparing Kubernetes Resource Metrics (KRM) and Prometheus Custom Metrics (PCM), and provides insights on optimizing HPA's performance.Experiments show that PCM reacts more responsively to load changes, leading to quicker scaling than KRM. However, this can result in a higher number of failed requests during scaling operations.

The research by Ioannis Korontanis et al.,[6] discusses a Prometheus-based monitoring stack for applications and resources on Kubernetes clusters, used within the platform to monitor applications across various development units and host types.The monitoring stack successfully monitors both applications and resources, characterizes hosts in a K8s cluster.

The paper by Sai Vimal Kumar V et al., [7] discusses multi cluster fault detection with Prometheus on Kubernetes and Docker containers.It was found that the Prometheus on Kubernetes performed better.

The research by Octavian Mart et al., [8] discusses the traditional Kubernetes observability parameters and compared them with that which Prometheus provides. It tells about the limitations of the traditional observability parameters and tells advantage of using Prometheus.

The paper by Lea Matlekovic et al., [9] discusses converting a monolithic application to microservices-based app using Fastapi. Prometheus was used for monitoring.

The research by Lei Chen, Ming Xian et al.,[10] This paper discusses using Prometheus to discuss OpenStack cloud platform.It was found that Prometheus with Grafana is an effective monitoring system.

III. Proposed System

A. Architecture

The architecture can be seen in Figure 1. The various components involved in the application are

1) **Kubernetes Metrics:** Metrics of the cluster itself, providing data on pods, deployments, etc. The kube-state-metrics and node exporters agents by Prometheus can be used for this purpose.
2) **Prometheus:** Open-source monitoring tool collecting metrics data. It is responsible for collecting the metrics from the various agents deployed in the Kubernetes cluster. Prometheus operator will be used to manage the Prometheus deployed in the cluster.
3) **Alertmanager:** It provides alerting capabilities to Prometheus. A webhook is configured to forward its alerts to Alerta.
4) **Grafana:** Open-source analytics and visualization tool, querying from Prometheus. It queries various metrics from Prometheus using the PromQL language and can be used to build custom dashboards to get to know the condition of the cluster and the various services that are running on it.
5) **Loki:** Log aggregation system for Kubernetes, collecting logs. It acts as a centralized storage for all the logs. Promtail an agent by Grafana will perform the duty of sending all the logs from cluster to Loki.
6) **Alerta:** Alert monitoring system, collecting and managing alerts, with options for notifications. It collects all the alerts from Prometheus and sends notifications to sources such as Slack. A custom plugin can also be written to create Jira Tickets based on the type of Alert generated.

B. Methodology

The Kubernetes cluster was created using minikube. The monitoring system requires the setup of 3 main components Prometheus, Alerta, Grafana and Logging Component.

1) Prometheus: The various steps involved in setting up Prometheus are

- **Set up Node Exporters and kube-state-metrics:** The initial step involves configuring kube-state-metrics and node exporters for the Prometheus cluster. kube-state-metrics will listen to the Kubernetes API and generate metrics regarding the various Kubernetes resources present in the system. Node exporters, on the other hand, collect system-level metrics from each node, providing crucial data on resource utilization, performance, and health.

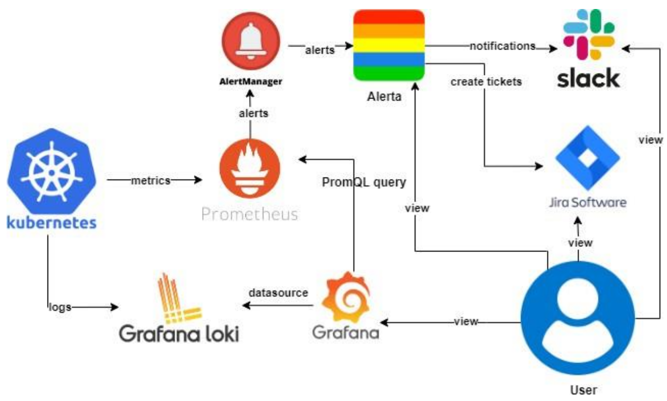

Fig. 1. Architecture of the Proposed System

- **Set up Prometheus using Prometheus Operator:** The Prometheus Operator is used to set up Prometheus. Traditionally, Prometheus setup requires a `prometheus.yaml` along with the `scrape_configs` and `rules` files specifying the targets to monitor and the different rules to evaluate for alerting purposes. But Prometheus Operator provides various CRDs to dynamically manage these configurations. `ServiceMonitor` resource was used to monitor all the resources in the cluster. The operator also provides `PodMonitor` for monitoring the pods directly used in case of deployments having no service resource assigned to it. `ScrapeConfig` resource can be used to monitor the services that are not deployed in the Kubernetes cluster. `PrometheusRule` resource is used to write up the recording and alerting rules for Prometheus to evaluate based on which alerting takes place. The metrics from the kube-state-metrics are used to write the various rules for detecting various scenarios such as a pod going down, a node going down, deployments not having enough replicas, etc. The metrics collected by `ServiceMonitor` can be used to write application-specific rules. Here the rules written only consist of making sure that the target is up. After writing up all these resources the Prometheus setup would be completed.
- **Set up AlertManager:** The final step in setting up Prometheus involves setting up Prometheus AlertManager. Prometheus Operator can be used for this purpose. All the alerts generated here will be forwarded to Alerta through a webhook which will be configured later.

2) Alerta: Once the setup of Prometheus is done next is to set the alerting component which is Alerta

- **Set up Alerta:** The first step is to set up Alerta. It requires setting up the various API keys, users, etc. Alerta offers a diverse range of authentication providers including basic auth, SAML, OpenID, among others. For this project, basic auth was employed. Different roles can be established, each with varying levels of permissions to ensure proper access control. Alerta comes equipped with its own predefined severity levels, but these can be customized to align with project requirements. Postgres database is used to store all the alerts associated with Alerta. Once Alerta configuration is done `AlertManagerConfig` resource can be utilized to facilitate the creation of a configuration for Alertmanager, enabling seamless forwarding of all alerts from Alertmanager to Alerta.
- **Enabling Notifications through Slack:** Alerta provides a plugin for sending notifications to Slack. The plugin can be directly used just modifying the to the message sent using the `SLACK_PAYLOAD` env variable. It offers routing to different Slack channels based on factors such as environment, severity, or event. The `routing` plugin was employed to set up routing based on tags.
- **Custom Plugin for Jira:** To handle more severe alerts effectively, a custom plugin was developed to create Jira tickets. The plugin automatically closed the ticket when the alert was resolved. Additionally, it was capable of adding comments in Jira whenever a note was added to Alerta.

3) Grafana and Logging Component:

- **Set up Loki:** Loki is deployed on the cluster to store all the logs. It offers both monolithic and scalable deployment options. For this project, the monolithic version of Loki was chosen. While Loki supports various storage backends such as S3 or GCS, the FileSystem option was utilized for storage.
- **Promtail Configuration for Loki:** Promtail is a agent by Grafana used for collecting logs from various sources and forwarding them to Loki. Promtail is configured to scrape logs from kubernetes and send them to Loki. Additionally, writing up pipelines for Loki involves defining how multiline and singleline logs should be processed, filtered, and indexed by Loki for efficient querying and visualization.
- **Set up Grafana:** The setup up Grafana consisted of providing the various datasources which in our case was Loki and Prometheus. 3 different dashboards were built in Grafana. These consisted the

IV. Result & Discussion

The project effectively leveraged a variety of open-source tools to establish a comprehensive monitoring system for Kubernetes. This system adeptly monitored the diverse targets deployed within Kubernetes clusters.

At the heart of the monitoring infrastructure was Prometheus, serving as the central monitoring system. Prometheus diligently observed various Kubernetes deployments, promptly detecting instances when critical services faltered or when pods experienced disruptions. The prometheus deployed can be seen by Figure 2.

International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 06 | June - 2024 SJIF Rating: 8.448 ISSN: 2582-3930

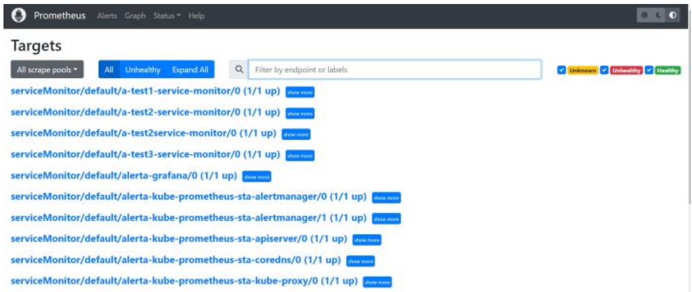

Fig. 2. Prometheus Dashboard

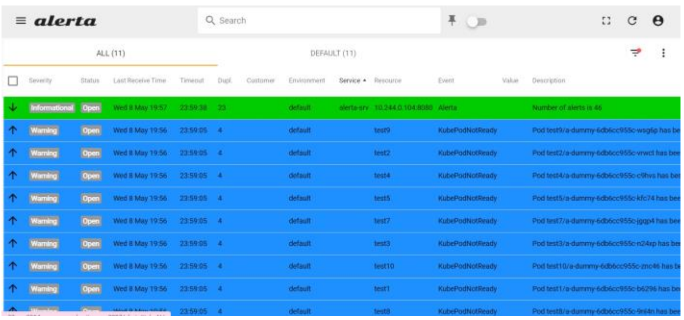

Fig. 3. Alerta Dashboard

Complementing Prometheus was Alerta, the centralized alerting system. The deployed dashboard can be seen from Figure 3. Alerta seamlessly aggregated alerts from Prometheus, facilitating swift incident response and resolution. Notifications were efficiently disseminated through Slack, ensuring real-time communication among team members. Moreover, where necessary, Alerta seamlessly interfaced with Jira, automatically creating tickets to streamline issue tracking and resolution processes. Figures 5 and 4 show the slack message received and ticket created.

Various dashboards were created to visualize the various data provided by the kube-state-metrics exporter. The dashboards built were for different levels which were cluster, node, namespace, and pod. The dashboards consisted of displaying the CPU and RAM usage on the cluster level as well as graphs showing different other levels such as namespace, pod, etc. At node and namespace levels it gave a list of all the pods in it.

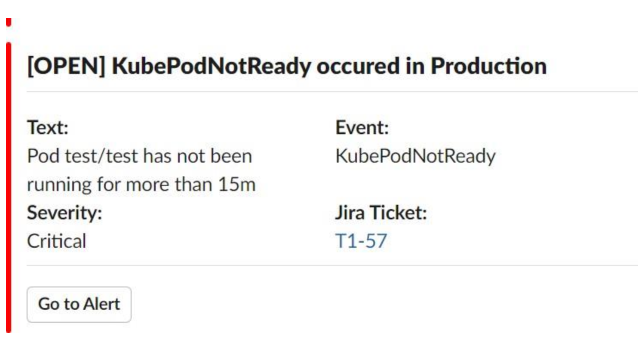

Fig. 4. Slack Message for PodDown

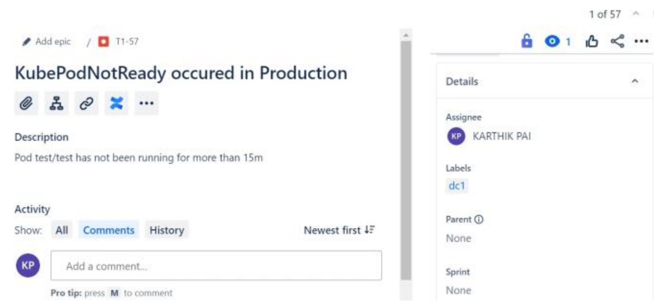

Fig. 5. Jira Ticket for PodDown

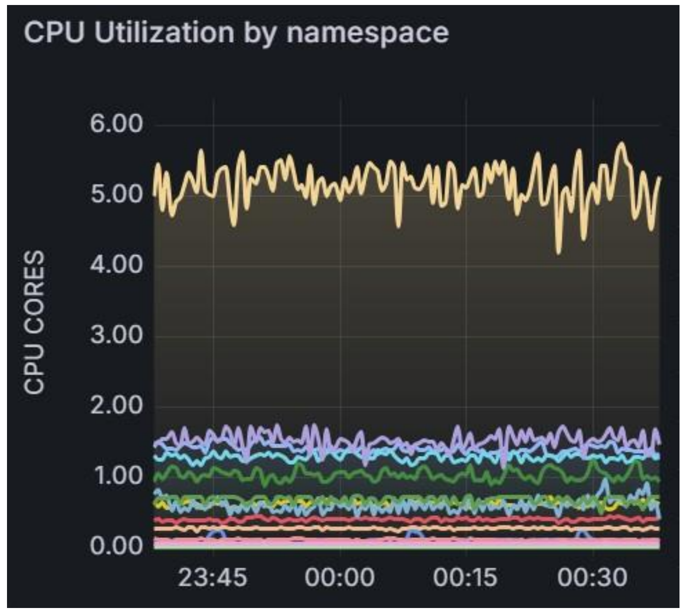

Fig. 6. CPU usage shown by namespace

At pod level it gave a list of all the containers running in the pod.It also gave insights on the state of the different pods, the ips associated with it etc. Some of the visualizations done are as follows. Figure 7 shows cluster-level CPU and RAM usage. It would provide insights on the load on the cluster. Figure 6 shows the CPU utilization of different resources by namespace, which is one such graph among many others that shows the CPU utilization.

Figure 8 shows some details associated with the pod that was displayed including the resource responsible for creating the pod as well as the QOS class associated with it. The QOS class gives details about how likely the pod is to be evicted when the cluster doesn't have enough CPU or RAM available. Other details that were displayed included the node in which the pod is running etc. Figure 9 tells the count of the total pods, namespaces, and nodes in the cluster.

Figure 10 gives the different kubernetes resources available. Figure 11 gives the replicas available by deployment. Other graphs were created like this which give insights on the status of the different Kubernetes resources.

Figure 12 is a table that gives the CPU and memory usage

International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 06 | June - 2024 SJIF Rating: 8.448 ISSN: 2582-3930

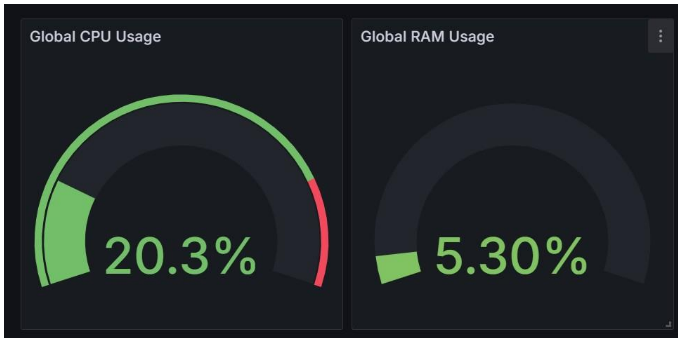

Fig. 7. Cluster level CPU usage

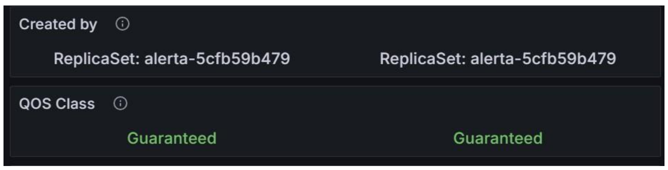

Fig. 8. Pod Details

by the different containers in the pods. This usage will enable to know the resources on the container level and if any problems are found actions can be taken accordingly. Finally Figure 13 shows the logs collected using Loki at pod level. The logs shown here are that from Alerta

The table I makes a comparison on the traditional based monitoring and the improvements provided by the Prometheus based monitoring on Kubernetes monitoring proposed by system. We get more range of metrics rather than just cpu/ram. Application-specific metrics are also possible. It provides better alerting and visualization capabilities as well.

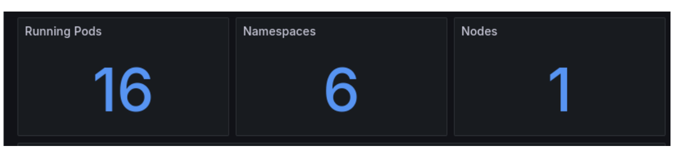

Fig. 9. Count of pods, namespaces and deployments in the cluster

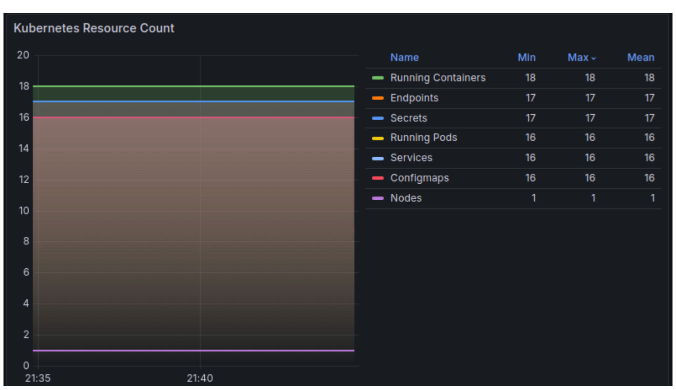

Fig. 10. Status of Pods

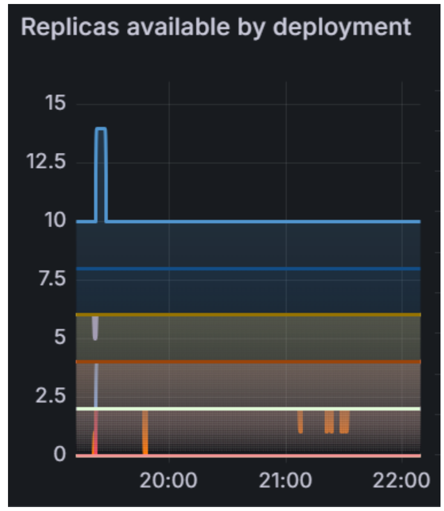

Fig. 11. Replica Availability

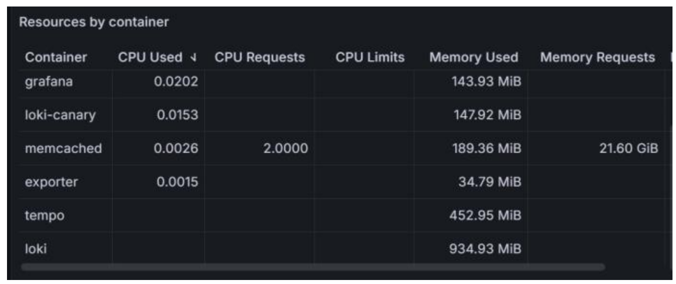

Fig. 12. Resources by containers

V. Conclusion

The project aimed to establish a robust monitoring and alerting infrastructure for the various applications deployed in a Kubernetes environment. Through the integration of Prometheus, Grafana, and Alerta, coupled with meticulous configuration and testing, the project successfully achieved its objectives.

By leveraging Prometheus for monitoring, the system was able to collect a wide range of metrics from various components within the Kubernetes cluster, including the various applications deployed in the cluster, This provided valuable insights into the application's performance, request traffic, error rates, and resource utilization. Grafana's intuitive dashboard-

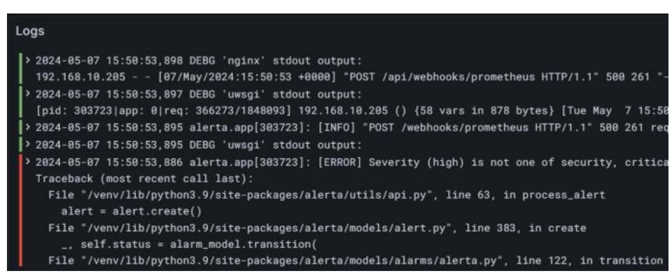

Fig. 13. Logs of the pods

International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 08 Issue: 06 | June - 2024 SJIF Rating: 8.448 ISSN: 2582-3930

TABLE I
COMPARISON BETWEEN TRADITIONAL KUBERNETES MONITORING AND PROMETHEUS KUBERNETES MONITORING

Feature/Aspect	Traditional Kubernetes Monitoring	Prometheus Kubernetes Monitoring
Data Collection and Storage	- **Metrics Server**: Collects resource usage (CPU, memory). - **Storage**: Requires third-party solutions.	- **Prometheus Operator**: Manages Prometheus instances. - **Exporters**: node_exporter, kube-state-metrics, etc. - **TSDB**: Efficient time-series database.
Architecture	- **Centralized Aggregation**: Central server or set of servers. - **Push/Agent-Based Model**: Agents deployed on nodes.	- **Decentralized and Federated**: Independent Prometheus instances. - **Pull-Based Model**: Prometheus scrapes metrics.
Metrics and Logs	- **Resource Metrics**: Basic CPU and memory. - **Application Metrics**: Limited, needs third-party tools.	- **Extensive Metrics**: Detailed metrics on resources and applications. - **Logs**: Integrated with solutions like Grafana Loki.
Querying and Alerting	- **Basic Querying**: Limited capabilities with Metrics Server. - **Basic Alerting**: Doesn't have much alerting capabilities	- **PromQL**: Powerful query language. - **Advanced Alerting**: Sophisticated rules, multi-condition and time-based alerts.
Visualization	- **Kubernetes Dashboard**: Basic web UI.	- **Grafana Integration**: Customizable dashboards. - **Built-In Web UI**: For querying and visualizing metrics.

ing capabilities allowed stakeholders to visualize and analyze these metrics effectively, enabling proactive monitoring and performance optimization.

Additionally, Alerta served as a central alert management system, facilitating timely notification and triaging of alerts generated by Prometheus. Integration with external communication channels such as Slack and Jira ensured that relevant stakeholders were promptly informed of critical incidents, streamlining incident response and resolution processes.

Overall, the monitoring infrastructure established through this project enhances the reliability, availability, and performance of the applications in the Kubernetes environment. By proactively monitoring and addressing potential issues, the system empowers stakeholders to maintain optimal application performance and deliver a seamless user experience. Moving forward, continued monitoring, periodic evaluation, and iterative improvements will be essential to sustain and enhance the effectiveness of the monitoring infrastructure in meeting evolving business needs and application requirements.

REFERENCES

[1] G. Carcassi, J. Breen, L. Bryant, R. W. Gardner, S. Mckee, and C. Weaver, "Slate: Monitoring distributed kubernetes clusters," in Practice and Experience in Advanced Research Computing, ser. PEARC '20, Portland, OR, USA: Association for Computing Machinery, 2020, 19–25, isbn: 9781450366892. doi: 10.1145/3311790.3401777.

[2] I. Tzanettis, C.-M. Androna, A. Zafeiropoulos, E. Fotopoulou, and S. Papavassiliou, "Data fusion of observability signals for assisting orchestration of distributed applications," Sensors, vol. 22, no. 5, p. 2061, 2022. doi: 10.3390/s22052061.

[3] R. Satrio Hadikusuma, L. Lukas, and K. Bachri, "Survey paper: Optimization and monitoring of kubernetes cluster using various approaches," Sinkron, vol. 8,pp. 1357–1365, Jul. 2023. doi: 10.33395/sinkron.v8i3.12424.

[4] P. P., S. Soudri, R. P., and S. Bailuguttu, "Enhancement of observability using kubernetes operator," Indonesian Journal of Electrical Engineering and Computer Science, vol. 25, p. 496, Jan. 2022. doi: 10.11591/ijeecs.v25.i1.pp496-503.

[5] T. K. D.-H. P. S. K. Thanh-Tung Nguyen Yu-Jin Yeom, "Horizontal pod autoscaling in kubernetes for elastic container orchestration," Sensors, vol. 20, no. 16, p. 4621, 2020. doi: 10.3390/s20164621.

[6] I. Korontanis, T. Theodoropoulos, A. Makris, and K. Tserpes, "Real-time monitoring and analysis of edge and cloud resources," Proceedings of the 3rd Workshop on Flexible Resource and Application Management on the Edge (FRAME '23), 2023. doi: 10.1145/3589010.3594892.

[7] M. K. Sai Vimal Kumar V, "Multi cluster monitoring for fault detection using novel kubernetes with prometheus over docker container," Journal of Pharmaceutical Negative Results, 1548–1555, 2022. doi: 10.47750/pnr.2022.13.S04.185.

[8] O. Mart, C. Negru, F. Pop, and A. Castiglione, "Observability in kubernetes cluster: Automatic anomalies detection using prometheus," in 2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2020, pp. 565–570. doi: 10.1109/HPCC-SmartCity-DSS50907.2020.00071.

[9] L. Matlekovic and P. Schneider-Kamp, "From monolith to microservices: Software architecture for autonomous uav infrastructure inspection," in Embedded Systems and Applications, ser. EMSA 2022, Academy and Industry Research Collaboration Center (AIRCC), Mar. 2022. doi: 10.5121/csit.2022.120622.

[10] L. Chen, M. Xian, and J. Liu, "Monitoring system of openstack cloud platform based on prometheus," in 2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL), 2020, pp. 206–209. doi: 10.1109/CVIDL51233.2020.0100.

[11] T. Abirami, C. Vasuki, P. Jayadharshini, and R. R. Vigneshwaran, "Monitoring and alerting for horizontal auto-scaling pods in kubernetes using prome theus," in 2023 International Conference on Computer Science and Emerging Technologies (CSET), 2023, pp. 1–8. doi: 10.1109/CSET58993.2023.10346811.

[12] J. S. Nunes, S. C. Sampaio, R. S. Malaquias, and I. de Morais Barroca Filho, "Deploying the observability of the sigsaude system using service mesh," in 2020 20th International Conference on Computational Science and Its Applications (ICCSA), 2020, pp. 9–15. doi: 10.1109/ICCSA50381.2020.00014.

[13] N. Sukhija and E. Bautista, "Towards a framework for monitoring and analyzing high performance computing environments using kubernetes and prometheus," in 2019 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computing, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (Smart-

World/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), 2019, pp. 257–262. doi: 10 . 1109 / SmartWorld - UIC - ATC - SCALCOM-IOP-SCI.2019.00087.

[14] M. Song, C. Zhang, and E Haihong, "An auto scaling system for api gateway based on kubernetes," in 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS), 2018, pp. 109–112. doi: 10.1109/ICSESS.2018.8663784.

[15] C.-C. Chang, S.-R. Yang, E.-H. Yeh, P. Lin, and J.-Y. Jeng, "A kubernetes-based monitoring platform for dynamic cloud resource provisioning," in GLOBECOM2017 - 2017 IEEE Global Communications Conference, 2017, pp. 1–6. doi: 10.1109/GLOCOM.2017.8254046.

[16] H. Kitahara, K. Gajananan, and Y. Watanabe, "Highly-scalable container integrity monitoring for large-scale kubernetes cluster," in 2020 IEEE International Conference on Big Data (Big Data), 2020, pp. 449–454. doi: 10.1109/BigData50022.2020.9377815.

[17] H. Kitahara, K. Gajananan, and Y. Watanabe, "Real-time container integrity monitoring for large-scale kubernetes cluster," Journal of Information Processing, vol. 29, pp. 505–514, 2021. doi: 10.2197/ipsjjip.29.505.

[18] M. Gupta, K. Sanjana, K. Akhilesh, and M. N. Chowdary, "Deployment of multitier application on cloud and continuous monitoring using kubernetes," in 2021 5th International Conference on Electrical, Electronics, Communication, Computer Technologies and Optimization Techniques (ICEECCOT), 2021, pp. 602–607. doi: 10.1109/ICEECCOT52851.2021.9707957.

[19] S. Y. An, Y. S. Cha, E. J. Jeon, G. Y. Gwon, B. C. Shin, and B. R. Cha, "A prestudy on the open source prometheus monitoring system," Journal of the Korean Institute of Smart Media, vol. 10, no. 2, pp. 110–118, 2021. doi: https://doi.org/10.14371/KISM.2021.10.2.110.

[20] N. Sukhija, E. Bautista, O. James, et al., "Event management and monitoring framework for hpc environments using servicenow and prometheus," in Proceedings of the 12th International Conference on Management of Digital EcoSystems, ser. MEDES '20, Virtual Event, United Arab Emirates: Association for Computing Machinery, 2020, 149–156, isbn: 9781450381154. doi: 10.1145/3415958.3433046.

[21] M. Brattstrom and P. Morreale, "Scalable agentless cloud network monitoring," in 2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud), 2017, pp. 171–176. doi: 10.1109/CSCloud.2017.11.

[22] C. Cao, A. Blaise, S. Verwer, and F. Rebecchi, "Learning state machines to monitor and detect anomalies on a kubernetes cluster," in Proceedings of the 17th International Conference on Availability, Reliability and Security, ser. ARES '22, Vienna, Austria: Association for Computing Machinery, 2022, isbn: 9781450396707. doi: 10 . 1145 / 3538969 . 3543810.

[23] C. Cassé, P. Berthou, P. Owezarski, and S. Josset, "A tracing based model to identify bottlenecks in physically distributed applications," in 2022 International Conference on Information Networking (ICOIN), 2022, pp. 226–231. doi: 10.1109/ ICOIN53446.2022.9687217.

[24] C. Cassé, P. Berthou, P. Owezarski, and S. Josset, "Using distributed tracing to identify inefficient resources composition in cloud applications," in 2021 IEEE 10th International Conference on Cloud Networking (CloudNet), 2021, pp. 40–47. doi: 10.1109/CloudNet53349.2021.9657140.