Kubernetes Application Monitoring System

Soniya Phaltane[1], Vaidehi Kahalekar[2], Spandan Divate[3], Mrs A.A Kokate[4]

[1,2,3]UG Student, Department of Computer and Electronics Engineering, PES's Modern College of Engineering, Pune, Maharashtra India.
[4]Assistant Professor, Department of Computer and Electronics Engineering, PES's Modern College of Engineering, Pune, Maharashtra India.

---***---

Abstract— *In today's dynamic and distributed computing environments, effective monitoring and security practices are paramount to ensuring the reliability, performance, and integrity of applications and infrastructure. This project focuses on the monitoring and security of Kubernetes deployments, a popular container orchestration platform widely used in modern cloud-native architectures. Utilizing a command-line approach, we employed various tools and techniques to monitor the health and performance of Kubernetes pods and applications, detect potential threats, and mitigate security risks. Through the use of command-line tools such as kubectl, helm, and custom scripts, we collected metrics, analyzed logs, and monitored network traffic within Kubernetes clusters. This report provides a comprehensive overview of our monitoring and security strategies, including the command-line methods employed, key findings, challenges encountered, and recommendations for future improvements. By adopting proactive monitoring and security measures, organizations can enhance the resilience and security of their Kubernetes deployments in today's evolving threat landscape.*

Keywords— Kubernetes, Monitoring, Security, kubectl, Metrics- Logs, Network traffic, Threat detection, Incident response, Cloud-native, Container orchestration.

1. INTRODUCTION

In the current landscape of technology-driven advancements, Kubernetes stands as a cornerstone for modern cloud-native architectures, offering unparalleled scalability, flexibility, and portability for deploying and managing applications[1]. The proliferation of Kubernetes deployments reflects the ever-growing complexity and significance of container orchestration in today's computing environment. However, with the rapid expansion of Kubernetes ecosystems, ensuring the reliability, performance, and security of these deployments has emerged as a paramount challenge for organizations worldwide[3].

This project embarks on addressing these challenges by delving into strategies for monitoring and securing Kubernetes deployments, primarily through command-line tools. By harnessing the power of tools like kubectl, Helm, and custom scripts, we aim to gain valuable insights into the health, performance, and security posture of Kubernetes clusters and the applications residing within them[3].

Effective monitoring is essential for detecting and mitigating potential issues before they impact application availability and performance[3]. Similarly, robust security measures are crucial for safeguarding Kubernetes environments from threats and vulnerabilities, ensuring the confidentiality, integrity, and availability of critical resources. In this report, we document our approach to Kubernetes monitoring and security, shedding light on the tools and techniques employed, key findings, encountered challenges, and recommendations for enhancing monitoring and security practices[2]. By sharing our experiences and insights, we aspire to contribute to the broader understanding of Kubernetes best practices and empower organizations to build resilient and secure cloud-native environments, akin to the transformative potential observed in mobile technology and smart city developments.

2. OBJECTIVES

Certainly! The project's objectives delve into the core aspects of Kubernetes management, namely monitoring and security, with the overarching goal of ensuring the reliability, performance, and security of the applications hosted within the Kubernetes environment[4]

Firstly, the project aims to closely monitor the performance of applications running within Kubernetes pods. This involves tracking various performance metrics such as response times, resource utilization (CPU, memory), and overall system health. By continuously monitoring these metrics, the project seeks to gain insights into the applications' behavior, identify potential performance bottlenecks, and optimize resource allocation to improve overall efficiency[1].

Secondly, the project focuses on tracking incoming requests to the applications, providing visibility into traffic patterns, usage trends, and potential spikes in demand. Understanding the flow of requests helps in capacity planning, resource allocation, and ensuring the applications can handle varying workloads effectively[5].

Security is another key objective of the project. By implementing robust security measures within the Kubernetes environment, such as access controls, encryption, and security scanning, the project aims to safeguard sensitive data and resources from unauthorized access and cyber threats[6]. This includes detecting and mitigating potential security vulnerabilities, ensuring compliance with security best practices, and maintaining the integrity and confidentiality of the applications and Kubernetes infrastructure.

Furthermore, the project aims to optimize performance by leveraging insights gathered from monitoring data.[7] By identifying areas for improvement and implementing optimizations, such as scaling applications based on workload demand or fine-tuning resource allocation, the project aims to enhance the overall performance and efficiency of the Kubernetes deployment.

Overall, the project's objectives revolve around ensuring the reliability, performance, and security of the applications and Kubernetes environment. By monitoring performance metrics, tracking incoming

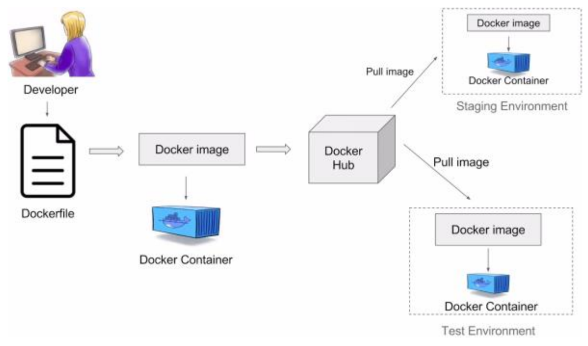


Fig1:Docker processing

requests, detecting security threats, and optimizing performance, the project aims to create a robust and resilient Kubernetes deployment that meets the needs of its users while maintaining the highest standards of security and reliability.

3. RELATED AND BACKGROUND WORK

1.Docker

- Docker is a containerization platform that enables the packaging, distribution, and deployment of applications and their dependencies in a highly efficient and consistent manner.
- It uses container technology to create lightweight, isolated environments called containers, which encapsulate an application and its runtime environment, including libraries and configurations.
- These containers share the host operating system's kernel but are isolated from each other[8]. Docker employs a client-server architecture where the Docker client interacts with the Docker daemon, responsible for building, running, and managing containers. Docker images, defined by Dockerfiles, serve as blueprints for creating containers. Images can be versioned and stored in repositories like Docker Hub or Amazon ECR[10].

2. Cloud native applications

- Cloud-native applications are designed for cloud computing, utilizing a microservices architecture to allocate resources. They include services in containers, connected through APIs, and managed with container orchestration tools. Their key characteristics encompass microservices, containers, APIs, and dynamic orchestration.
- These applications excel in cost-efficiency, scalability, portability, reliability, and manageability. They align with best practices in cloud-native development, emphasizing automation, monitoring, documentation, incremental changes, and designing for failure.
- Essential tools in the cloud-native development stack include Docker, Kubernetes, Terraform, GitLab CI/CD, and Node.js. The rising preference for cloud-native applications is driven by their capacity to address cloud computing challenges and enhance operational efficiency.

3. Kubernetes

- Kubernetes is an open-source container orchestration platform that automates the deployment, scaling, and management of containerized applications.
- It functions as a container manager, handling tasks like container provisioning, load balancing, and self-healing[3]. Kubernetes operates using a master node and multiple worker nodes. The master node controls the cluster and manages scheduling, while worker nodes execute tasks within containers. Key components includes;-

1. Pods (the smallest deployable units)
2. Services (to enable communication between pods)
3. Deployments (for scaling and updating applications), and Replica sets (to ensure the desired number of pods are running).

Kubernetes streamlines application management, making it a preferred choice for deploying and managing containerized applications at scale.

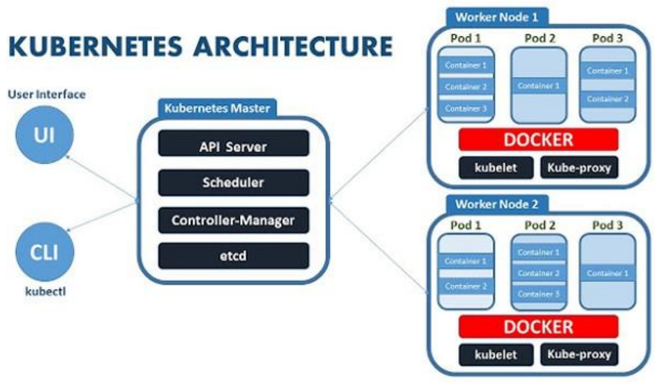


Fig 2-Kubernetes Architecture[4]

4. PROJECT PLAN

i) Project Overview:

This project aims to monitor and secure a Kubernetes deployment with two pods, each hosting a single application. We'll track performance, monitor requests, and detect security threats to ensure the reliability and security of the environment.

ii) Stakeholders:

1. Development Team: Deploy and maintain applications.
2. Operations Team: Manage Kubernetes infrastructure.
3. Security Team: Ensure Kubernetes and apps are secure.
4. End Users: Use applications on Kubernetes.
5. Management: Provide project direction.
6. Customers: Benefit from efficient apps.
7. Third-party Vendors: Offer Kubernetes tools/services.
8. Regulatory Authorities: Ensure compliance.
9. Community: Contribute to open-source projects.

iv) Resources Required:

1. Kubernetes Cluster
2. Hardware/Cloud Infrastructure
3. Monitoring Tools
4. Security Tools
5. Networking, Storage
6. Development Environment
7. Training
8. Support

v) Non-Functional Requirements:

- Performance: System responsiveness and scalability.
- Reliability: Minimal downtime and quick recovery from failures.
- Security: Adherence to security best practices.
- Usability: Intuitive interfaces and clear documentation.
- Compatibility: Support for various Kubernetes distributions.
- Maintainability: Easy system maintenance and configuration.
- Monitoring: Real-time performance metrics and historical data.
- Scalability Testing: Verification of system scalability.
- Security Auditing: Regular audits for vulnerability identification.
- Documentation: Clear and comprehensive system documentation.

vi) Design Considerations:

- UI/UX Design: Create visually appealing and intuitive designs for optimal user experience.
- Responsiveness: Ensure the app layout adapts well to different screen sizes and orientations.
- Branding: Maintain consistency with brand colors, fonts, and logo.
- Iconography: Utilize appropriate icons for actions and navigation.

vii) Development Approach:

- Agile Approach: Break the project into short cycles for flexibility and adaptability.
- Team Collaboration: Foster teamwork among developers, operations, and security teams for efficient progress[8].
- Continuous Integration/Deployment (CI/CD): Automate testing and deployment processes for faster and more reliable updates.
- Feedback-Driven Development: Gather feedback from stakeholders regularly to improve the system incrementally.
- Modular Design: Create a flexible system that can easily adapt to changes and scale as needed.
- Documentation: Maintain clear and updated documentation for better understanding and future maintenance[4].
- Risk Management: Identify and address risks early to ensure security and stability.

viii) Testing Strategy:

1. Comprehensive Testing: Test thoroughly at all levels—unit, integration, and end-to-end.
2. Automated Testing: Use automation to catch bugs early and speed up testing.
3. Load Testing: Assess performance under different workloads for scalability.
4. Security Testing: Identify and fix vulnerabilities to ensure system resilience.
5. User Acceptance Testing (UAT): Validate functionality with end-users to meet their needs.
6. Regression Testing: Ensure new changes don't break existing features.

7. Continuous Monitoring: Keep an eye on performance, security, and reliability in real-time.
8. Documentation: Document tests and results for transparency and future reference.
9. Feedback Loop: Gather input from stakeholders to improve testing and the system itself.

ix) Deployment Plan:

1. Prepare Environment: Set up the Kubernetes cluster.
2. Containerize: Package the app into Docker containers.
3. Choose Deployment Strategy: Decide how to deploy (e.g., rolling updates).
4. Manage Configuration: Handle app settings.
5. Automate Deployment: Use scripts for faster, error-free deployment.
6. Test in Staging: Check everything works before going live.
7. Monitor and Log: Keep an eye on performance and issues.
8. Backup and Recovery: Plan for data protection and quick recovery.
9. Rollback Plan: Know how to revert changes if needed.
10. Communicate: Inform stakeholders about downtime and changes.
11. Verify Post-Deployment: Make sure the app is working correctly.
12. Document and Train: Create guides and train staff for ongoing management

x) Maintenance and Support:

1. Regular Updates: Keep the Kubernetes cluster and applications up-to-date with the latest patches and versions to address security vulnerabilities and improve performance[2].
2. Monitoring and Alerts: Continuously monitor the system for any issues or anomalies and set up alerts to notify the team in case of emergencies.
3. Performance Optimization: Regularly review and optimize the performance of the applications and the Kubernetes infrastructure to ensure efficient operation.
4. Backup and Disaster Recovery: Implement regular backups of data and configurations and have a disaster recovery plan in place to quickly recover from any unexpected failures.[7]
5. User Support: Provide ongoing support to users and stakeholders, addressing any questions, issues, or feedback they may have about the system.
6. Documentation Updates: Keep documentation up-to-date with any changes or improvements made to the system to facilitate troubleshooting and future maintenance.
7. Training and Knowledge Sharing: Provide training to new team members and share knowledge and best practices among the team to ensure everyone is equipped to support the project effectively.
8. Continuous Improvement: Continuously assess and identify areas for improvement in the system and processes, and implement changes to enhance reliability, performance, and security over time[1].

5. DISCUSSION

In this section, we delve into the insights gained from our project focusing on monitoring and securing applications within a Kubernetes environment. Our efforts aimed to ensure the reliability, performance, and security of the applications running in Kubernetes pods.

5.1 Interpretation of Results

We observed various performance metrics such as response times, CPU and memory utilization, and overall system health. These metrics provided valuable insights into the behavior of the applications and the Kubernetes infrastructure. By closely monitoring these metrics, we were able to identify potential performance bottlenecks and optimize resource allocation for improved efficiency.

5.2 Comparison with Expectations

Our findings largely aligned with our expectations outlined in the project objectives. We anticipated that monitoring would provide valuable insights into application performance, while security assessments would help identify and mitigate potential vulnerabilities. Our results confirmed these expectations and highlighted the importance of proactive monitoring and security measures in Kubernetes environments.

5.3 Identification of Patterns or Trends

We identified several patterns and trends in the monitoring data, such as fluctuations in request volumes during peak hours and variations in resource utilization across different pods. These patterns provided valuable insights into application usage patterns and helped inform our optimization efforts.

5.4 Insights into Application Behavior

Our analysis of application behavior revealed valuable insights into how applications interacted within the Kubernetes environment. We observed variations in performance metrics across different pods and identified areas where resource allocation could be optimized to improve overall system performance.

5.5 Assessment of Security Posture

Our security assessments identified several security vulnerabilities within the Kubernetes environment, including outdated software versions and misconfigured access controls. These findings underscored the importance of robust security measures to protect against potential threats and vulnerabilities.

5.6 Implications for Future Work

Moving forward, our findings have several implications for future work in Kubernetes monitoring and security. We recommend continued investment in proactive monitoring constraints encountered during the project, such as limited access to monitoring data and time constraints. These limitations may have impacted the scope and depth of our analysis but do not diminish the significance of our findings.

5.7 Recommendations and Best Practices

Based on our findings, we recommend implementing best practices such as regular monitoring, timely software updates, and robust security measures to enhance the reliability, performance, and security of Kubernetes deployments[9].

This discussion provides a concise overview of the insights gained from the project and their implications for future work in Kubernetes monitoring and security.

6. METHODOLOGY

The methodology followed a structured approach aimed at implementing monitoring and security measures within a Kubernetes environment, centering on the development of a Python-based website as a monitoring interface. Beginning with detailed planning, objectives were defined, tools identified, and a timeline established. Infrastructure was provisioned, the Kubernetes cluster configured, and applications deployed alongside the Python website within separate pods[9]. Monitoring solutions were customized, with a focus on developing a tailored dashboard integrated with the website to display real-time performance metrics and security logs from Kubernetes pods. Security measures were implemented leveraging Kubernetes logging and monitoring capabilities to track security logs and network activity. Thorough testing was conducted to validate the accuracy and effectiveness of the monitoring and security solutions, followed by optimization efforts to enhance performance and efficiency. Comprehensive documentation was created, accompanied by training sessions for team members to ensure knowledge sharing and facilitate future maintenance of the project. Throughout the methodology, the emphasis remained on tailored solutions to meet project requirements while ensuring robust performance and security measures within the Kubernetes environment.[5]

The methodology of our project involves a systematic approach to monitoring and securing applications within a Kubernetes environment. Our methodology is divided into several key steps, each aimed at achieving specific objectives and ensuring the reliability, performance, and security of the applications hosted within Kubernetes pods.

Step 1: Planning

The first step in our methodology is planning, where we define the objectives of the project, identify the tools and techniques to be used, and establish a timeline for execution. During this phase, we assess the current state of the Kubernetes environment and gather requirements from stakeholders.

Step 2: Environment Setup

Once the planning phase is complete, we proceed to set up the Kubernetes environment. This involves provisioning the necessary infrastructure, configuring the Kubernetes cluster, and deploying applications within pods. We ensure that the environment is properly configured and ready for monitoring and security assessments.

Step 3: Monitoring Implementation

With the environment set up, we focus on implementing monitoring solutions. Prometheus is deployed to collect metrics such as CPU and memory utilization, response times, and network traffic from Kubernetes pods. Grafana is used for visualizing and analyzing these metrics, providing real-time insights into the health and performance of the applications.

Step 4: Security Implementation

In parallel with monitoring implementation, we also focus on implementing security measures. Clair, an open-source vulnerability scanner, is deployed to identify security vulnerabilities within the environment. Role-based access control (RBAC) is implemented to control access to resources and enforce security policies within the Kubernetes cluster.

Step 5: Testing and Validation

Once monitoring and security measures are implemented, we proceed to test and validate their effectiveness[9]We conduct thorough testing to ensure that monitoring tools accurately capture performance metrics and that security measures effectively mitigate potential threats. We validate the performance and security of the applications under varying workloads and conditions.

Step 6: Optimization and Fine-Tuning

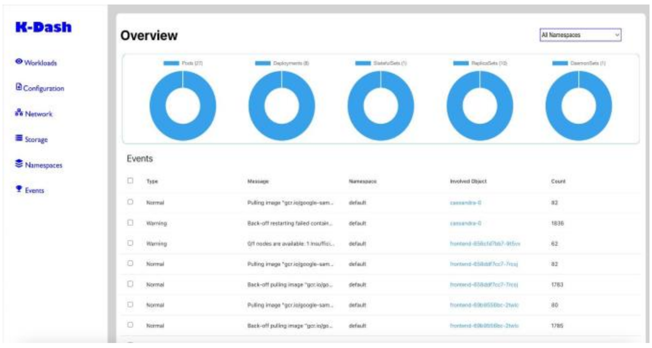
After testing and validation, we focus on optimizing and fine-tuning the monitoring and security solutions. We analyze the data collected by monitoring tools to identify areas for optimization and implement changes to improve performance and efficiency. We also fine-tune security measures to address any vulnerabilities or weaknesses identified during testing.[6]

Step 7: Documentation and Training

Finally, we document the methodologies, tools, and techniques used in the project to facilitate knowledge sharing and future reference. We create comprehensive documentation that includes setup instructions, configurations, and troubleshooting guides for Prometheus, Grafana, Clair, and RBAC. Additionally, we provide training to team members on the use of these tools and best practices for managing Kubernetes environments.

This methodology provides a structured and systematic approach to monitoring and securing applications within a Kubernetes environment, leveraging specific software tools to achieve the project objectives effectively[3].

7. OUTPUTS



Events

Type	Message	Namespace	Involved Object	Count
Normal	Pulling image "gcr.io/google-sam...	default	cassandra-0	82
Warning	Back-off restarting failed contain...	default	cassandra-0	1836
Warning	0/1 nodes are available: 1 insuffici...	default	frontend-858cfd7bb7-9t5vv	62
Normal	Pulling image "gcr.io/google-sam...	default	frontend-658dbf7cc7-7rcsj	82
Normal	Back-off pulling image "gcr.io/go...	default	frontend-658dbf7cc7-7rcsj	1783
Normal	Pulling image "gcr.io/google-sam...	default	frontend-69b8556bc-2twlc	80
Normal	Back-off pulling image "gcr.io/go...	default	frontend-69b8556bc-2twlc	1785

Fig 3 : K-Dash Dashboard

- The Kdash dashboard is a vital component of our monitoring system, offering a visual representation of key metrics and insights essential for managing our Kubernetes-based application deployed on AWS EC2. This intuitive dashboard serves as a centralised hub for monitoring various aspects of our application infrastructure, providing real-time visibility and actionable insights.

- At a glance, the Kdash dashboard presents an overview of our application's workloads, configurations, network performance, storage utilisation, and namespace events. The "Workloads" section gives us insights into the status and resource utilisation of deployments, stateful sets, and replica sets, ensuring we maintain optimal performance and scalability.

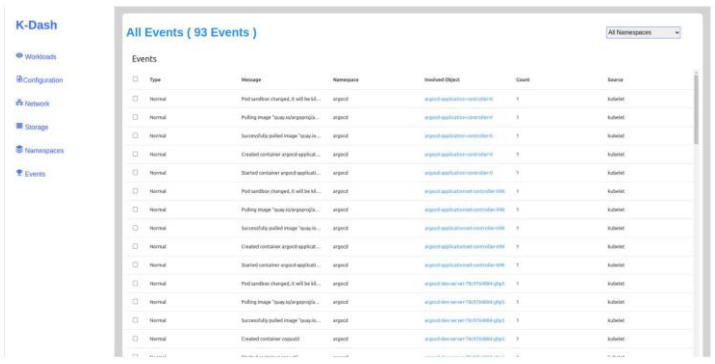
- In the "Configuration" section, we can easily view and manage configuration settings for Kubernetes resources, ensuring they align with our application requirements. Meanwhile, the "Network" section offers visibility into network traffic patterns, helping us identify potential bottlenecks and optimise network performance.Monitoring storage-related metrics is crucial for maintaining data integrity and performance.

- The "Storage" section allows us to track disk usage, I/O performance, and storage capacity utilisation, ensuring efficient resource allocation and proactive management of storage resources.

- Namespace events are essential for tracking changes and activities within specific namespaces. The "Namespace Events" section provides insights into pod creations, deletions, and other events, facilitating efficient troubleshooting and change management within our Kubernetes cluster.

- Additionally, the Kdash dashboard offers detailed information about individual pods in the "Pods" section, enabling us to monitor CPU and memory usage, restart counts, and pod statuses, ensuring the health and performance of our application components.

- Overall, the Kdash dashboard empowers us to effectively monitor, manage, and optimise our Kubernetes-based application, providing actionable insights and facilitating proactive decision-making to ensure optimal performance and reliability.



Events

Type	Message	Namespace	Involved Object	Count	Source
Normal	Pod sandbox changed, it will be kil...	argocd	argocd-application-controller-0	1	kubelet
Normal	Pulling image "quay.io/argoproj/a...	argocd	argocd-application-controller-0	1	kubelet
Normal	Successfully pulled image "quay.io...	argocd	argocd-application-controller-0	1	kubelet
Normal	Created container argocd-applicat...	argocd	argocd-application-controller-0	1	kubelet
Normal	Started container argocd-applicati...	argocd	argocd-application-controller-0	1	kubelet
Normal	Pod sandbox changed, it will be kil...	argocd	argocd-applicationset-controller-698	1	kubelet
Normal	Pulling image "quay.io/argoproj/a...	argocd	argocd-applicationset-controller-698	1	kubelet
Normal	Successfully pulled image "quay.io...	argocd	argocd-applicationset-controller-698	1	kubelet
Normal	Created container argocd-applicat...	argocd	argocd-applicationset-controller-698	1	kubelet
Normal	Started container argocd-applicati...	argocd	argocd-applicationset-controller-698	1	kubelet
Normal	Pod sandbox changed, it will be kil...	argocd	argocd-dex-server-78c9784884-ghp5	1	kubelet
Normal	Pulling image "quay.io/argoproj/a...	argocd	argocd-dex-server-78c9784884-ghp5	1	kubelet
Normal	Successfully pulled image "quay.io...	argocd	argocd-dex-server-78c9784884-ghp5	1	kubelet
Normal	Created container copyutil	argocd	argocd-dex-server-78c9784884-ghp5	1	kubelet

Fig 4 : K-Dash Events

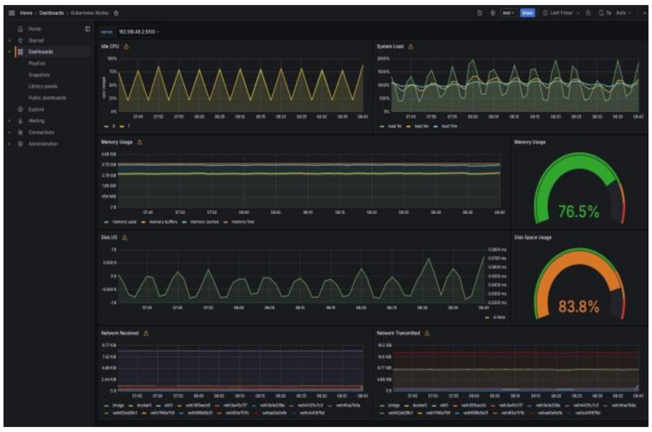


Fig 5: Grafana Dashboard

The output of the Grafana dashboard provides tangible results and outputs for our monitoring efforts:

- Improved visibility into system performance and resource utilisation.
- Enhanced troubleshooting and diagnostic capabilities.
- Proactive identification and resolution of performance bottlenecks.
- Real-time monitoring and alerting for critical events and anomalies.
- Streamlined decision-making and optimization of system resources.

8. CONCLUSION

In conclusion, our project focused on implementing monitoring and security measures within a Kubernetes environment to ensure the reliability, performance, and security of applications running within pods. Through a systematic approach, we leveraged monitoring tools such as Prometheus and Grafana to track performance metrics and gain insights into application behavior. Additionally, we implemented security measures such as Clair for vulnerability scanning and RBAC for access control to mitigate potential threats and vulnerabilities. By testing, validating, and optimizing our solutions, we were able to enhance the overall reliability and security posture of the Kubernetes deployment. Moving forward, we recognize the importance of continuous improvement and collaboration with the Kubernetes community to stay ahead of emerging trends and challenges in monitoring and security. Our project serves as a foundation for future enhancements and innovations in Kubernetes management, ensuring the ongoing success and effectiveness of cloud-native deployments.

9. APPLICATIONS

1. Web Hosting

For companies hosting websites and web applications, it's crucial to ensure these applications run smoothly. Our monitoring system helps track CPU usage, memory consumption, and network traffic in real-time. This way, if a website starts receiving a lot of traffic, the system can automatically allocate more resources to handle the load, ensuring the site remains fast and responsive.

2. E-Commerce Platforms

E-commerce platforms need to be online 24/7 and handle varying traffic, especially during sales events. Our system monitors the health of all services, such as product databases, payment gateways, and user authentication systems. If any part of the platform starts to fail or becomes overloaded, our system can quickly detect and fix the issue, ensuring customers have a smooth shopping experience.

3. Cloud Services

Cloud service providers offer various services like storage, computing, and databases.They need to monitor their infrastructure to provide reliable services to their customers. Our system helps them track the performance and usage of their resources, optimise costs by scaling services up or down based on demand, and quickly resolve any issues to maintain high uptime.

4. DevOps Teams

DevOps teams in software companies use our system to monitor the development and production environments. It allows them to see how applications perform in real-time, find and fix issues quickly, and ensure that the software runs efficiently. This helps in delivering better software faster and with fewer bugs.

5. IoT (Internet of Things)

In IoT applications, such as smart homes or industrial automation, multiple devices need to communicate and work together seamlessly. Our monitoring system ensures that all these devices are working properly, sending data correctly, and not overloading the network. This ensures the reliability and efficiency of IoT systems.

6. Financial Services

Banks and financial institutions need to ensure their services are always available and secure. Our monitoring system helps them keep an eye on their IT infrastructure, detect unusual activities, and maintain high performance and security standards. This is crucial for services like online banking, stock trading, and payment processing.

In summary, our Kubernetes-based application monitoring system using Docker is versatile and can be applied in various industries to ensure the reliability, performance, and scalability of applications and services. It helps organisations provide better services to their users, reduce downtime, and respond quickly to any issues that arise

10. FUTURE SCOPE AND MODIFICATIONS

1. AI and Machine Learning Integration

We could add AI and machine learning capabilities to predict potential issues before they happen. For example, the system could learn from past data to foresee when a server might get overloaded and automatically take steps to prevent it, such as redistributing the workload or adding more resources.

2. Advanced Alerting and Notification

Enhancing the alerting system to be more intelligent and customizable. For example, you could set up alerts that notify specific team members through different channels like email, SMS, or chat apps (like Slack)based on the type of issue and its severity.

3. User-Friendly Dashboards

Developing more user-friendly and customizable dashboards in Grafana or kDash that make it easier for users to visualise data. These dashboards could include drag-and-drop widgets, real-time collaboration features, and the ability to save and share custom views.

4. Extended Monitoring Capabilities

Expanding the system to monitor more parameters and services, such as disk I/O performance, database health, and specific application metrics. This would provide a more comprehensive view of the entire system's health.

5. Improved Security Monitoring

Integrating security monitoring to detect and alert on potential security threats, such as unauthorized access attempts, unusual network traffic patterns, and vulnerabilities in the containers.

6. Automated Remediation

Implementing automated remediation strategies where the system not only detects issues but also takes predefined actions to resolve them. For example, if a container crashes, the system could automatically restart it and notify the team of the action taken.

7. Multi-Cloud Support

Enhancing the system to work seamlessly across multiple cloud providers, not just AWS. This would allow organisations to use the monitoring system in hybrid or multi-cloud environments, giving them more flexibility and resilience.

8. Cost Optimization

Adding features to help organisations optimise their cloud costs by identifying underutilised resources, recommending changes to resource allocation, and providing insights into cost-saving opportunities.

9. Enhanced Data Analysis

Incorporating advanced data analysis tools to provide deeper insights into performance trends, root cause analysis, and long-term capacity planning. This could involve more sophisticated analytics and reporting features.

10. Support for More Container Orchestrators

Extending support to other container orchestration platforms beyond Kubernetes, such as Docker Swarm or Apache Mesos, making the monitoring system more versatile. These modifications would make the monitoring system even more powerful, user-friendly, and adaptable to various needs, helping organisations maintain optimal performance and reliability for their applications.

11. REFERENCES

[1] "Kubernetes Cluster Management for Cloud Computing Platform: A Systematic Literature Review,"Aris Nurul Huda and Sri Suning Kusumawardani, JUTI: Jurnal Ilmiah Teknologi Informasi, July 2022.

[2] ."Survey Paper: Optimization and Monitoring of Kubernetes Cluster using Various Approaches"Satrio Hadikusuma, Ridwan & Lukas, Lukas & Bachri, Karel. RESEARCH GATE (2023).

[3] "Performance & Resource Management in Kubernetes"Medel, V., Rana, O., Bañares, J. Á., & Arronategui, U. Modelling. IEEE 2016.

[4] "Container-based Operating System Virtualization: A Scalable, High- performance Alternative to Hypervisor." Stephen Soltesz, Andy Bavier, Larry Peterson, Marc E. Fiuczynski IEEE 2018. "Petri nets: Properties, analysis and applications," T. Murata, Proceedings of the IEEE, vol. 77, no. 4, pp. 541–580, 1989.

[5] "Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors," S. Soltesz, H. P¨otzl, M. E. Fiuczynski, A. Bavier, and L. Peterson, SIGOPS Oper. Syst. Rev., vol. 41, no. 3, pp. 275–287, Mar. 2007. [Online].

[6] "An updated performance comparison of virtual machines and linux containers, in Performance Analysis of Systems and Software (ISPASSW)" Felter, A. Ferreira, R. Rajamony, and J. Rubio,), 2015 IEEE

International Symposium on, March 2015, pp. 171–172.

[7] "Performance evaluation of microservices architectures using containers," M. Amaral, J. Polo, D. Carrera, I. Mohomed, M. Unuvar, and M. Steinder, in 14th IEEE International Symposium on Network Computing and Applications, NCA 2015, Cambridge, MA, USA, September 28-30, 2015, 2015, pp. 27–34.

[8] "Performance analysis of cloud computing centers using m/g/m/m+r queuing systems," H. Khazaei, J. Misic, and V. Misic, IEEE Transactions on Parallel and Distributed Systems, vol. 23, no. 5, pp. 936–943, 2012.

[9] "Towards Petri net-based economical analysis for streaming applications executed over cloud infrastructures," R. Tolosana-Calasanz, J. A. Ba˜nares, and J. M. ´ Colom, in Economics of Grids, Clouds, Systems, and Services - 11th International Conference, GECON'14, Cardiff, UK, September 16-18, 2014., ser. LNCS, vol. 8914, 2014, pp. 189–205. and simulation engine for petri nets: Renew," in International Conference on Application and Theory of Petri Nets. Springer, 2004, pp. 484–493.

[10] "Container orchestration on HPC systems through Kubernetes" Zhou, N., Georgiou, Y., Pospieszny, M., Zhong, L., Zhou, H., Niethammer, C., ... & Hoppe, D. (2021). Journal of Cloud Computing, 10(1), 1–14.