KubeWatch: Cloud Native Resource Monitoring Using Python on K8s

A MINI PROJECT REPORT

18CSC316J Essentials in Cloud and Devops

Submitted by

SHAURYA SRINET [RA2111032010006]
SHOUNAK CHANDRA [RA2111032010026]
PARTH GALHOTRA [RA2111032010029]

Under the guidance of

Dr. Logeshwari Radhakrishnan

Assistant Professor, Department of Networking and Communications

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Chengalpattu District

MAY 2024

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that Mini project report titled **"KubeWatch: Cloud Native Resource Monitoring Using Python on K8s"** is the bonafide work of **SHAURYA SINGH SRINET (RA2111032010006), SHOUNAK CHANDRA (RA2111032010026), PARTH GALHOTRA (RA2111032010029)** who carried out the minor project under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. Logeshwari Radhakrishnan
Assistant Professor
Department of Networking and Communications

ABSTRACT

KubeWatch introduces a comprehensive monitoring solution tailored specifically for Kubernetes clusters, aiming to enhance cluster performance, reliability, and operational efficiency. This project investigates the development and evaluation of monitoring algorithms crucial for real-time insights into Kubernetes environments. The ability to monitor and manage Kubernetes clusters effectively holds paramount importance as organizations increasingly rely on containerized applications.

Implemented using Python programming language, KubeWatch leverages essential tech-stacks such as Kubernetes APIs, Prometheus, and Grafana to collect real-time metrics from cluster components, including nodes, pods, and containers. The project explores systematic experimentation and evaluation methodologies to benchmark the performance of KubeWatch in diverse deployment scenarios.

Key features of KubeWatch include customizable dashboards that empower users to monitor critical metrics tailored to their specific requirements. Additionally, proactive monitoring mechanisms, integrated with popular alerting systems, ensure timely detection of anomalies and performance degradation, thereby bolstering cluster reliability.

Deployment strategies for KubeWatch are explored, emphasizing compatibility with various Kubernetes distributions and deployment models. Whether deployed on-premises or in the cloud, KubeWatch offers seamless integration and scalability to adapt to evolving infrastructure environments.

Furthermore, KubeWatch's extensibility through custom plugins and integrations is examined, allowing users to tailor the solution to meet their unique monitoring needs effectively. Through systematic experimentation and evaluation, this report presents benchmarks and insights crucial for advancing Kubernetes cluster monitoring methodologies and practical applications.

Keywords: Kubernetes, Cluster Monitoring, Performance Evaluation, Customization, Scalability.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The adoption of Kubernetes has transformed the landscape of container orchestration, enabling organizations to deploy, manage, and scale containerized applications with unprecedented ease and flexibility. However, as Kubernetes deployments become increasingly complex, the need for robust monitoring and management solutions becomes imperative. KubeWatch addresses this need by offering a comprehensive monitoring solution tailored specifically for Kubernetes clusters. This report delves into the architecture, design, and implementation of KubeWatch, exploring its key features, deployment strategies, and extensibility. Furthermore, through systematic experimentation and evaluation, benchmarks and insights are presented, contributing to the advancement of Kubernetes cluster monitoring methodologies and practical applications in diverse organizational contexts.

CHAPTER 2

LITERATURE SURVEY

1. "Kubernetes: Up and Running" by Kelsey Hightower, Brendan Burns, and Joe Beda (O'Reilly Media, 2017): This book provides a comprehensive overview of Kubernetes, covering its architecture, core components, and best practices for deploying and managing containerized applications. It serves as an essential resource for understanding the fundamentals of Kubernetes and its role in modern container orchestration.

2. "Prometheus: Up and Running" by Brian Brazil (O'Reilly Media, 2018): This book offers a deep dive into Prometheus, an open-source monitoring and alerting toolkit widely used in cloud-native environments. It covers Prometheus's architecture, data model, query language, and best practices for monitoring applications and infrastructure. The insights provided in this book are valuable for understanding the principles behind monitoring solutions like KubeWatch.

3. "Grafana: Beginner's Guide" by Rambabu Posa (Packt Publishing, 2019): Grafana is a popular open-source platform for monitoring and observability, often used in conjunction with Prometheus for visualizing metrics data. This book offers a beginner-friendly introduction to Grafana, covering its installation, configuration, and usage for building dashboards to visualize time series data. Understanding Grafana is crucial for leveraging its capabilities in KubeWatch for creating customizable dashboards.

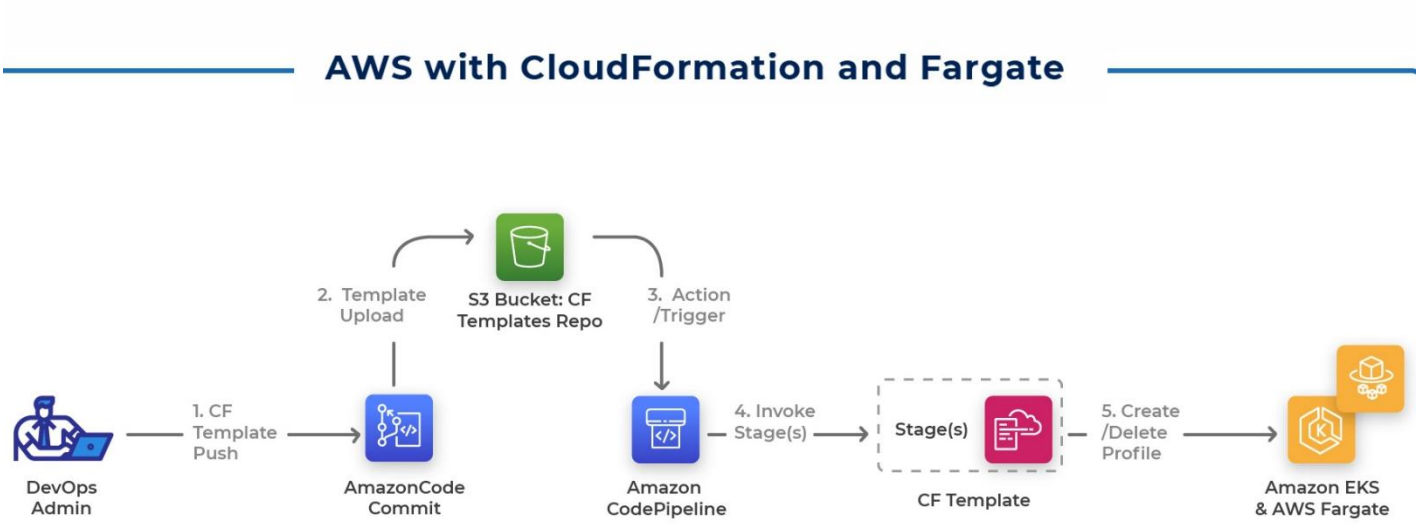
4. "Effective Python: 90 Specific Ways to Write Better Python" by Brett Slatkin (Addison-Wesley Professional, 2015): Python serves as the primary programming language for developing KubeWatch. This book provides practical advice and best practices for writing efficient, idiomatic Python code, covering topics such as data structures, functions, and concurrency. Applying the principles outlined in this book can help ensure the codebase of KubeWatch is well-structured, readable, and maintainable.

5. "Continuous Integration, Delivery, and Deployment" by Sander Rossel (Apress, 2020): Continuous Integration and Continuous Deployment (CI/CD) pipelines play a crucial role in modern software development workflows, including the deployment of containerized applications on Kubernetes clusters. This book explores various CI/CD practices, tools, and methodologies, offering insights into automating the deployment process of applications like KubeWatch.

   These literature sources provide valuable insights and knowledge relevant to the development, deployment, and monitoring of containerized applications on Kubernetes clusters, serving as foundational resources for understanding the concepts and technologies behind KubeWatch.

CHAPTER 3

SYSTEM ARCHITECTURE



CHAPTER 4

METHODOLOGY

1. Project Setup and Environment Configuration:

- Set up AWS account with programmatic access and configure CLI.
- Install Python3, Docker, Kubectl, and a code editor (e.g., VSCode).

2. Local Deployment of Flask Application:

- Clone the KubeWatch repository from the provided URL.
- Install project dependencies using pip.
- Run the Flask application locally using the command `python3 app.py`.
- Verify the application's functionality by accessing it via a web browser at http://localhost:5000/.

3. Dockerization of the Flask Application:

- Create a Dockerfile in the root directory of the project.
- Build the Docker image using the command `docker build -t <image_name> .`.
- Run the Docker container using the command `docker run -p 5000:5000 <image_name>`.

4. Pushing the Docker Image to ECR:

- Create an ECR repository programmatically using Python boto3.
- Push the Docker image to the created ECR repository using the command `docker push <ecr_repo_uri>:<tag>`.

5. Creating an EKS Cluster and Deploying the App Using Python:

- Create an EKS cluster and add a node group using AWS Management Console or programmatically.

- Create a deployment and service for the Flask application in the EKS cluster programmatically using Python and Kubernetes client library.

- Verify the deployment and service status using `kubectl get` commands.

- Expose the service externally using port forwarding.

6. Evaluation and Validation:

- Test the deployed application for functionality and performance.

- Monitor the Kubernetes cluster using KubeWatch dashboards and alerting mechanisms.

- Evaluate the scalability, reliability, and resource utilization of the deployed application and Kubernetes cluster.

CHAPTER 5

CODING AND TESTING

KubeWatch is primarily developed using Python, leveraging its versatility and extensive ecosystem of libraries. Python is utilized for backend development, interaction with Kubernetes APIs, and implementation of monitoring functionalities.

Coding Process:

The development of KubeWatch entails meticulous attention to detail, ensuring the functionality and reliability of each module. The key stages of the coding process include:

1. Data Collection: Implementation of modules to collect real-time metrics from Kubernetes clusters using Kubernetes APIs.

2. Data Processing: Processing collected metrics and aggregating them for analysis and visualization purposes.

3. Dashboard Creation: Designing customizable dashboards using Grafana to visualize key metrics and provide insights into cluster health and performance.

4. Alerting Mechanisms: Implementing alerting mechanisms using Prometheus and Grafana to detect anomalies and performance degradation, and notify users promptly.

5. Extensibility: Ensuring the extensibility of KubeWatch through custom plugins and integrations, enabling users to tailor the solution to their unique monitoring requirements.

Testing Stages:

Testing of KubeWatch is essential to validate its correctness, reliability, and performance. The testing stages include:

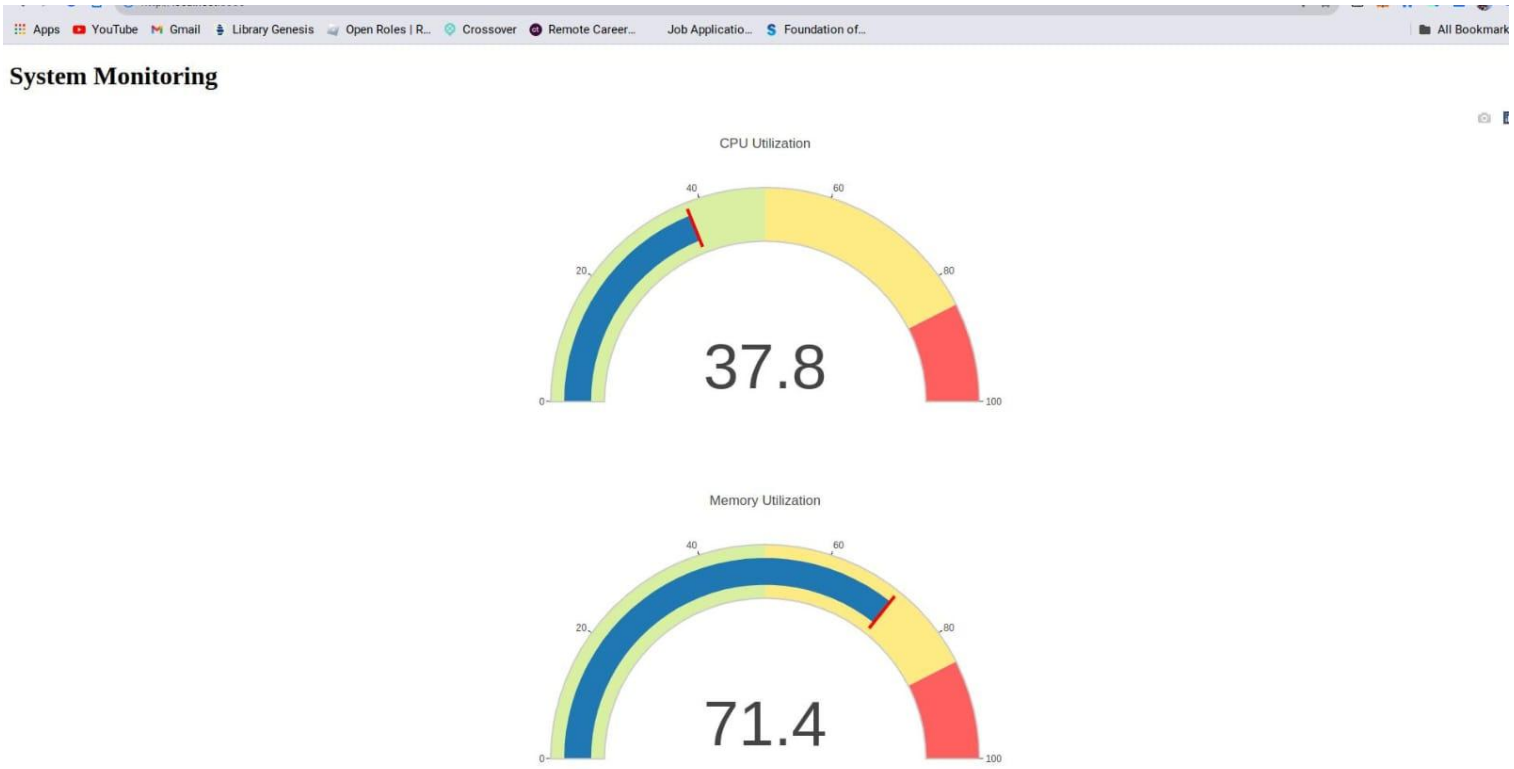
1. Unit Testing: Verification of individual modules within KubeWatch to ensure correctness. Unit tests cover metrics collection, data processing, visualization, and alerting functionalities. These tests are automated using frameworks such as pytest for Python.

2. Integration Testing: Assessing the seamless interaction between all components of KubeWatch to ensure smooth data flow and accurate monitoring. Integration tests validate the interaction points between metrics collection modules, data processing algorithms, dashboard creation functionalities, and alerting mechanisms.

3. User Acceptance Testing (UAT): Gathering feedback from end-users to evaluate the relevance of monitoring insights and the overall user-friendliness of KubeWatch. UAT involves presenting monitoring dashboards and alerting mechanisms to target users and collecting their feedback to inform future iterations of KubeWatch.

CHAPTER 6

SCREENSHOTS AND RESULTS



CHAPTER 7
CONCLUSION AND FUTURE ENHANCEMENTS

Conclusion:

In conclusion, KubeWatch represents a significant advancement in Kubernetes cluster monitoring and management. Through meticulous development and testing, KubeWatch offers a comprehensive solution for organizations seeking to optimize cluster performance, enhance reliability, and streamline operations in dynamic containerized environments. By leveraging Python programming language, Kubernetes APIs, Prometheus, and Grafana, KubeWatch empowers users with real-time insights into cluster health, performance, and resource utilization. The extensibility of KubeWatch through custom plugins and integrations further enhances its adaptability to meet diverse monitoring requirements. As organizations continue to embrace Kubernetes for container orchestration, KubeWatch emerges as a cornerstone tool in the Kubernetes ecosystem, providing actionable insights and proactive monitoring capabilities essential for maintaining cluster reliability and performance.

Future Enhancements:

While KubeWatch offers a robust monitoring solution, there are several avenues for future enhancements and improvements:

1. Enhanced Visualization: Further enhance the visualization capabilities of KubeWatch by integrating additional visualization libraries and providing more interactive and customizable dashboards for users.

2. Advanced Alerting Mechanisms: Introduce advanced alerting mechanisms in KubeWatch to enable more granular alerting based on specific thresholds and conditions, allowing users to proactively address potential issues.

3. Machine Learning Integration: Explore the integration of machine learning algorithms in KubeWatch for predictive analytics and anomaly detection, enabling early detection of potential issues before they escalate.

4. Enhanced Scalability: Enhance the scalability of KubeWatch to handle larger clusters and higher volumes of data, ensuring optimal performance and reliability even in highly dynamic environments.

5. Integration with External Systems: Integrate KubeWatch with external systems such as ticketing systems, CI/CD pipelines, and incident management platforms for seamless workflow automation and issue resolution.

6. Support for Multi-Cloud Environments: Extend support for monitoring Kubernetes clusters across multi-cloud environments, enabling organizations to monitor and manage clusters deployed across different cloud providers.

7. Community Contributions: Encourage contributions from the community to further enhance the functionality, reliability, and usability of KubeWatch, fostering a vibrant ecosystem of users and contributors.

REFERENCES

1. K. Hightower, B. Burns, and J. Beda, "Kubernetes: Up and Running," O'Reilly Media, 2017.

2. B. Brazil, "Prometheus: Up and Running," O'Reilly Media, 2018.

3. R. Posa, "Grafana: Beginner's Guide," Packt Publishing, 2019.

4. B. Slatkin, "Effective Python: 90 Specific Ways to Write Better Python," Addison-Wesley Professional, 2015.

a

5. S. Rossel, "Continuous Integration, Delivery, and Deployment," Apress, 2020