An Innovative Approach of API Automation Testing Implemented on Cloud Environments Using Container Management Services

Tanja Dimova[1], Igor Kalendar[1], Daniel Denkovski[2], Danijela Efnusheva[2] and Marija Kalendar[2]
[1]*MIR! Software Solutions, Londonska Str. 6, 1000 Skopje, N. Macedonia*
[2]*Computer Technologies and Engineering Department, Faculty of Electrical Engineering and Infomation Technologies, University "SS. Cyril and Methodius" in Skopje, Rugjer Boshkovikj Str. 18, 1000 Skopje, N. Macedonia*
{tanjadimova8, igor.kalendar}@gmail.com,{danield, danijela,marijaka}@feit.ukim.edu.mk

Keywords: Automation Testing, Cloud Platforms, Container Services, Docker, Kubernetes, Google Cloud Platform, Services Monitoring.

Abstract: This research paper focuses on developing a complete system for daily automation testing of comprehensive web applications implemented on cloud environments, encompassing the execution of automated API tests, real-time monitoring and results visualization of the testing environments. Despite the tools for developing automated API tests, the study uses containerization tools as Docker and Kubernetes, showcasing their integration into a cohesive testing framework. Furthermore, the implementation leverages the potential of the Google Cloud Platform (GCP) to demonstrate the usage of cloud computing services, emphasizing scalability and efficiency. Additionally, the paper details the integration of monitoring tools, specifically Elasticsearch, to assess and visualize the health and performance of the underlying Kubernetes cluster. Through a comprehensive approach, encompassing a wide variety of tools, the research establishes a continuous and automated testing environment essential for cutting-edge software applications. Results showcase the successful orchestration of all the technologies, highlighting their collective impact on achieving a robust and efficient system for continuous automation testing and monitoring.

1 INTRODUCTION

In today's world of technology, every successful organization needs to be connected and accessible over the Internet. This would require setting-up an organizational data center. The traditional data center is on-premises, performing all of its functions in a physical location within the enterprise's office space, and usually managed by an in-house IT team. Thus, the digitalization of business processes in a traditional way would entail: a physical data center, buying and managing hardware (servers), software and licenses, creating a physical network, building the entire infrastructure and hiring a team of experts to manage or maintain this data center.

On the other hand, cloud computing is the current state-of-the-art technology for delivery of computing resources and services such as servers, data storage, databases, networking, software, analytics, and intelligence over the Internet to offer flexible resources, faster innovation, and economies of scale. Using this paradigm, instead of owning data centers, organizations can rent access to the service provider infrastructure (storage, computer servers, databases, networks) and pay only for the resources they use.

Consequently, the transition from traditional to cloud computing data centers is a very current topic. Our research focuses on a very specific topic, automation of API testing of a modern web-based application in a cloud environment, encompassing, researching and evaluating all aspects of building a complete, comprehensive, scalable and manageable automation testing cloud solution.

The rest of the paper is organized as follows. Section 2 elaborates the related work in the area of cloud computing and automation testing. Section 3 elaborates the design of the system for automated API testing. Section 4 presents the container elements incorporated into the automated testing environment, while Section 5 considers the cloud elements comprising the system. Section 6 showcases the performance characteristics of the Kubernetes cluster system deployment in the cloud, thus comparing all system elements and technologies. Section 7 concludes the paper pointing out the main results and benefits of the system.

2 RELATED WORK

Research in the domain of automated API testing, containerization, and cloud computing has provided valuable insights that enhance the understanding of these technologies. Authors in [1] examine DevOps technologies, offering perspectives on continuous integration and deployment, laying a foundation for our research on automating testing services. The research on API testing using Postman [2] contributes insights into automated API tests execution, while [3] presents considerations for secure implementation of the automated testing processes. Authors in [4] offer insights into building modern clouds by integrating Docker, Kubernetes, and cloud platforms, aligning with our research context. Additionally, [5] presents a study on the Grafana visualization tool, using an Influx DB data source. The authors in [6] provide a foundation for understanding the role of monitoring tools in the system health and performance evaluation, while the study on observability using Kubernetes operators [7] complements our goal of improving system monitoring and visualization by automating deployment, visualization, and monitoring within the cloud system and the Kubernetes cluster.

Our research builds upon these foundations, introducing a novel approach that integrates regression end-to-end API testing, containerization, and cloud computing. The solution optimizes daily testing processes, improves test coverage, and ensures quick error detection, making it a significant advancement in the field of automated testing. The comprehensive integration of these technologies positions our system as a more scalable, reliable, and efficient solution for automated testing services in modern cloud environments.

3 DESIGNING AN INNOVATIVE SYSTEM FOR AUTOMATED API TESTING

Automation testing is crucial in the process of modern software development for its efficiency, accuracy, and speed. It ensures comprehensive test coverage, particularly for regression testing, and integrates seamlessly into *Continuous Integration*/*Continuous Deployment* pipelines. By leveraging new tools and optimizing the testing ecosystem, automatic testing facilitates early bug detection and reduces overall testing costs for the software development company.

Illustrated in Figure 1 is the proposed design of an innovative holistic system enabling automated API testing utilizing cloud computing resources and advanced containerization technologies.

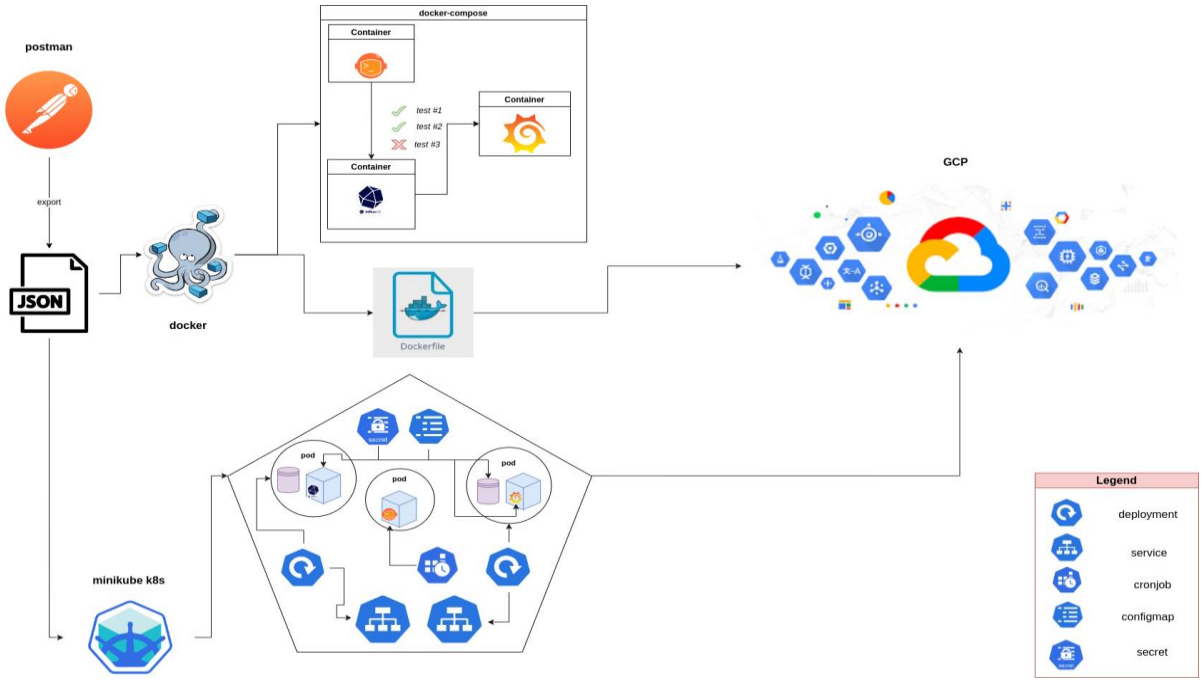

Figure 1: Components of an automated API testing system using containers and cloud computing.

The innovative system comprises many elements to achieve the goals for automation API testing in the cloud, managed by container orchestration:

1) ***Postman*** - a tool for testing API access points and creating and documenting automated tests.
2) ***Newman*** - command line tool allowing collections to be executed directly from Postman.
3) ***InfluxDB*** – a time series database used to store time-stamped sub-data (e.g. the execution time of a single query).
4) ***Grafana*** – a tool for visualization and analysis of data stored in the InfluxDB database.
5) ***Docker*** – system automation component (using Docker images and containers for Newman, Grafana and InfluxDB).
6) ***Kubernetes*** – an open source container orchestration system for automating software deployment, scaling and management.
7) ***Google Cloud Platform (GCP)*** - cloud computing services available and managed by Google.

The backend application intended to be tested can be represented as a group of API access points. Postman tool enables the environment for creating and saving simple and complex HTTP/s requests, as well as for reading their responses. Each application module is represented in Postman as a collection of requests (tests) organized as multiple folders that correspond to specific API access points.

A general scenario for testing the functionality of an API usually consists of the following steps:

1) Create Object (POST).
2) Verify previously created object (GET by ID).
3) Update object (PUT).
4) Get a list of all existing objects and check if the newly created object is in the list (GET All).
5) Delete object (DELETE).
6) Get a list of all objects again and verify that the deleted object is no longer in the list (GET All, deleted object not present).

The test script associated with a request will be executed after the request has been sent and a response has arrived. These tests verify that the APIs work as expected, determine that integrations between services work reliably, and that new changes have not broken the existing functionality.

Newman is the next (command line) tool in the toolchain, used after the Postman collections are set-up and configured. It allows collections to be executed directly using scripting. Its extensibility enables easy integration with continuous integration servers, and at the same time providing a rich suite of performance customization options.

InfluxDB, a time series database, is the next tool of choice that enables storing the reports and obtained test results using the influxDB Newman report tool. This database enables storing detailed statistical information and results from each test run.

To visualize the gathered data, the system uses Grafana, an open-source platform used for visualization, monitoring and analysis of data from the Influx DB database. Grafana enables creating dashboards with panels, each representing specific metrics over a specific time frame, thus presenting visual interpretation of the test runs, and the occurring failed tests.

The entire process presented in Figure 2 allows a well-defined and mostly automated process for API access points testing, and enables a simple further analysis and diagnostics from the saved data.

Upon detecting errors by the automated system, final check needs to be done manually, with two possibilities:

1) Either there was a change in the API not reflected into the test (the test needs to be corrected-fixed).
2) Or, the error cannot be detected into the test itself, thus the API request is broken, and an actual bug in the code has been detected.

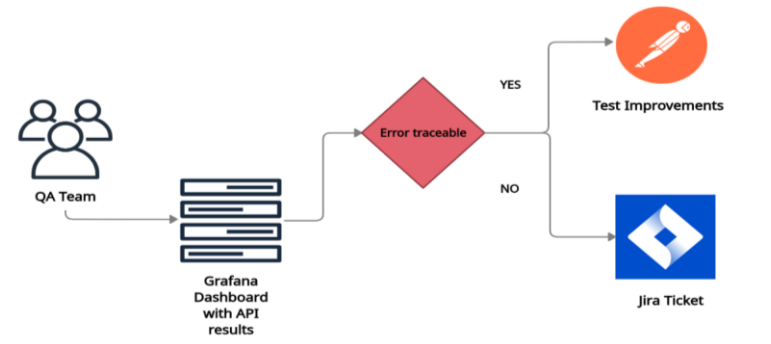

Figure 2: Analysis and diagnostics of test results.

4 CONTAINER TECHNOLOGIES FOR AUTOMATION TESTING

Today, containerization plays a crucial role in software development due to its efficiency, portability, and scalability. By using tools like Kubernetes, Docker, and GCP services, the testing workflow is streamlined, scalability is enhanced, and resource management is optimized. Figure 1 illustrates the core tools essential for automating testing processes and achieving efficient and reliable testing outcomes.

4.1 Containers for Automation Testing

Container technology presents an advantageous choice for integration within automation systems due to its inherent characteristics: lightweight design, minimal memory footprint, and efficient resource utilization, thus offering unparalleled portability and scalability. Furthermore, their isolation properties ensure secure deployment, and parallel coexistence on a single server without interference. The swift creation and destruction of containers, enhance operational speed, supporting dynamic workloads. Leveraging containerization facilitates seamless software deployment and management in cloud environments, abstracting applications from their underlying infrastructure.

Docker technology simplifies software delivery by providing a universal tool for creating, deploying, and running applications using containers, ensuring portability and consistency across different environments.

Four services are established using Docker container technology. The first service, InfluxDB's setup includes a Docker image, container name, and a port. Additionally a database and user are configured. However, the creation of the service alone does not facilitate storing of Newman results to the database, prompting the establishment of a second service to notify InfluxDB when it becomes operational. Similarly, Grafana is configured with its image, port, and container name.

The services defined with docker-compose need to be connected to enable data exchange and communication among themselves in order to provide a seamless system pipeline. Configuring Grafana to use InfluxDB data enables visual representation additionally facilitating analysis through preconfi-gured and modified dashboards and panels.

4.2 Orchestration of Container Services

In order to enable a more suitable usage of resources, and to simplify the management of the elements, the proposed systems encompasses the most widely used Kubernetes service orchestration tools. Kubernetes provides a highly elastic infrastructure with zero downtime deployment, automatic rollback, self-healing, and container scaling features, enabling seamless management of the entire lifecycle.

Before transitioning to cloud computing, the initial step involves setting up a local cluster. This is accomplished by utilizing Minikube, a lightweight implementation of Kubernetes. Compatible with all major OSs, Minikube aims to excel as a development tool for Kubernetes applications, ensuring comprehensive support for all Kubernetes features.

4.2.1 Establishing a Kubernetes Cluster

Establishing a Kubernetes cluster incorporating InfluxDB and Grafana involves several key steps. First, configuration files must be created for InfluxDB deployment, service, secret, and persistent volume claim. These files, specify essential details like the Docker image, exposed ports, and environment variables needed for database configuration. For instance, the deployment file outlines metadata and specifications, including the image source and container port, while the service file connects the service to the deployment through selectors. Additionally, a secret configuration file is essential for securely storing sensitive data, such as passwords, required for database configuration. This secret file uses Kubernetes secrets to securely store and inject the necessary data into the deployment as environment variables.

Moreover, enabling persistent storage for the InfluxDB database requires a persistent volume claim (PVC) configuration file. This file describes the type and details of the storage space required by InfluxDB, enabling Kubernetes to allocate or provision the appropriate volume to meet the database's storage needs. Setting up Grafana components follows a similar process to InfluxDB. ConfigMaps, serving a similar purpose to Secrets, are utilized to add configuration files to pod containers. Grafana requires three configuration files to be written to the running container. These files are added to a ConfigMap and mounted in different locations within the container for proper configuration.

This ample setup ensures proper functioning and configuration of both InfluxDB and Grafana, facilitating efficient data storage, management, and access within the Kubernetes environment.

4.2.2 Test Execution Schedule

The Kubernetes CronJob acts like a traditional cron utility, scheduling tasks within containers. Its purpose in the system is orchestrating API collection tests with Newman and sending results to InfluxDB. By setting a daily schedule, Kubernetes automates these tests.

As Minikube clusters lack access to local images, image construction takes place directly within the cluster using Minikube based command. Following image construction, the CronJob specification is created, including a jobTemplate that outlines the executing task. The schedule parameter dictates the recurring execution of tasks, ensuring periodic test runs.

After each run, the associated pod terminates until the next scheduled execution, allowing test results to be displayed in Grafana within a 24-hour window.

Although Minikube deployments do not inherently assign external IP addresses, accessing the Grafana service is made feasible via Minikube command, which seamlessly integrates with InfluxDB and automatically presents test results for analysis.

5 CLOUD IMPLEMENTATION

Introducing our system's move to the cloud involves tapping into the powerful advantages of cloud technology for refining our testing infrastructure. By transitioning to the cloud, scalability, reliability, and resource management are optimized. Cloud technology enables flexible scaling of testing resources, ensuring top-notch performance during peak demand. Furthermore, cloud platforms provide robust infrastructure, built-in security, and global accessibility, bolstering the resilience and efficiency of our testing environment.

5.1 Choosing Google Cloud Platform

Google Cloud Platform (GCP) is selected as the platform for several reasons. Firstly, it offers a comprehensive suite of cloud computing services integrated with Google's global network infrastructure. Additionally, GCP seamlessly integrates with the existing container technologies ecosystem, simplifying Docker and Kubernetes management. Its commitment to innovation, reliability, and security aligns with the system's requirements, making it an ideal choice for the cloud implementation for the automated testing system deployment.

5.2 Kubernetes as a Service

Kubernetes as a Service (KaaS) provides a streamlined approach to managing Kubernetes clusters in the public cloud, with Google Kubernetes Engine (GKE) being a prominent solution within GCP. The setup of a Kubernetes cluster on GCP is initiated through the user-friendly interface of the Google Cloud console's Kubernetes Engine section.

Utilizing Cloud Shell, a versatile tool provided by GCP, a seamless connection is established between the Google Cloud console and the Kubernetes cluster. This connection is pivotal for effectively managing and overseeing the cluster's operations, including deployment, monitoring, and maintenance tasks.

To deploy containerized applications on GCP, we use Google Container Registry (GCR). Storing Docker images in GCR ensures they are accessible and available across different platforms and environments.

With the foundational setup in place, the necessary components for applications are created and managed using familiar Kubernetes configuration files. These configurations are applied through standard procedures, orchestrating the deployment and scaling of applications on the Kubernetes cluster within GCP.

The seamless integration of Kubernetes with GCP empowers users to harness the scalability, reliability, and flexibility of cloud-native technologies for their applications and services. Through intuitive interfaces and robust infrastructure, GCP simplifies the complexities of managing Kubernetes clusters, enabling efficient and effective cloud operations.

6 PERFORMANCE CHARACTERISTICS OF THE KUBERNETES CLUSTER SYSTEM DEPLOYMENT

Finally, exploring the performance characteristics of the final Kubernetes cluster deployment for the system is crucial for ensuring the reliability and efficiency of the automated testing processes. Monitoring and analyzing the performance of the Kubernetes cluster within the system plays a pivotal role in optimizing resource utilization, identifying

potential bottlenecks, and enhancing overall performance. Through the examination of key metrics and the implementation of monitoring solutions, the objective is to achieve optimal performance and reliability.

6.1 Kubernetes Cluster Monitoring

Operating a Kubernetes cluster introduces complexities due to the distributed nature of its components and the vast array of available metrics. One crucial aspect of monitoring Kubernetes clusters is tracking the state of objects within the cluster, including deployments, nodes, and pods. kube-state-metrics (KSM) addresses this need by providing a simple, yet effective, mechanism for generating metrics directly from the Kubernetes server API.

KSM offers insights into the current state of the Kubernetes objects, enabling a detailed assessment of the health and performance of the cluster. Unlike traditional monitoring tools focusing on individual components, KSM collects various metrics related to the state and resource usage of objects within the cluster: the number of running pods, the available CPU and memory resources, and the overall health status. These metrics are crucial for delivering the operational efficiency and performance of the cluster.

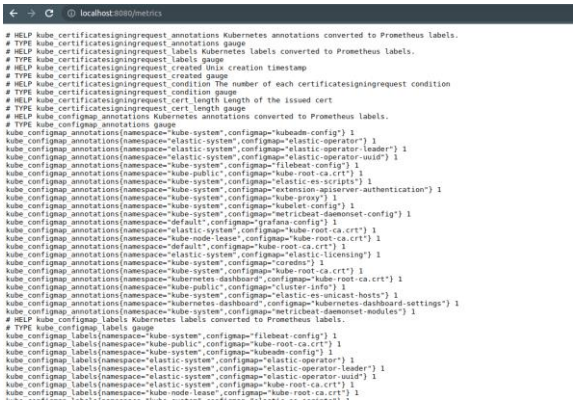

Figure 3: Kubernetes cluster health state.

Deploying KSM within the Kubernetes cluster involves creating essential Kubernetes objects, such as Service accounts, Cluster Roles, and Cluster Role Bindings, along with the kube-state-metrics deployment itself. These objects are crucial for enabling KSM to access and monitor Kubernetes API objects seamlessly. KSM exposes metrics via the HTTP /metrics endpoint, providing real-time visibility into the cluster's state. By analyzing these metrics, valuable insights into anomalies, resource utilization, and cluster performance can be gained as illustrated in Figure 3.

6.2 Cloud Environments Log Management

The ELK Stack, comprising Elasticsearch, Logstash, and Kibana tools, plays an important role in log management within cloud-based environments. It offers a centralized way for tracking and analyzing various issues across system infrastructure, including performance monitoring and node failure detection.

Elasticsearch, serving as the storage engine, efficiently stores and retrieves log data. Logstash handles log delivery, processing, and storage, ensuring seamless data handling. Kibana, the visualization tool, provides an intuitive interface for visualizing log data and conducting advanced data analysis.

Deploying ELK Stack involves creating essential Kubernetes objects and resources: Elasticsearch clusters and Logstash configurations. These components aggregate, process, and visualize log data, helping effective monitoring and analysis. Kibana completes the ELK Stack with visualization features, empowering users to gain insights into system performance and troubleshoot issues efficiently.

6.3 Results from Kubernetes Monitoring

In the deployed system, the Elastic Stack serves as a fundamental component for Kubernetes monitoring, offering essential tools such as Filebeat and Metricbeat to collect monitoring data. These lightweight agents, deployed as Daemon Sets in Kubernetes, capture both system and application-level metrics and logs. Filebeat is responsible for collecting logs from pods, containers, and applications running on Kubernetes.

Filebeat dynamically detects components within pods and applies logging modules accordingly, thus providing real-time access to log data, enabling efficient log analysis and troubleshooting. Metricbeat, on the other hand, collects and preprocesses system and service metrics, including CPU, memory, disk, and network data. Deployed on each node in the cluster, Metricbeat gathers metrics from the Kubelet API, offering insights into the state of nodes, pods, containers, and other Kubernetes resources. Additionally, Metricbeat accesses cluster-wide metrics directly from the kube-state-metrics service.

Deployment of Filebeat and Metricbeat in our Kubernetes cluster involves configuring YAML files provided by Elastic. These files define deployment settings and specify connection details, ensuring seamless integration with existing Elasticsearch deployments and Kubernetes environments. Connecting Filebeat to Elasticsearch entails configuring index patterns to define indexed data for efficient retrieval and visualization in Kibana. Filebeat utilizes predefined index patterns to seamlessly transmit data to an existing Elasticsearch deployment, establishing direct integration. Address specification and TLS certificate inclusion in the Filebeat configuration ensure secure communication with Elasticsearch. For effective container log collection, Filebeat instances require access to the local log path mounted by the host, enabling comprehensive log data collection from Kubernetes pods, containers, and applications.

Once deployed and configured, Filebeat's integration with Elasticsearch and Kibana facilitates efficient log analysis and troubleshooting, empowering users with valuable insights into system performance and health. Accessing log data collected by Filebeat is straightforward, as it automatically becomes available for exploration in Kibana's Logs application, as demonstrated in Figure 4.

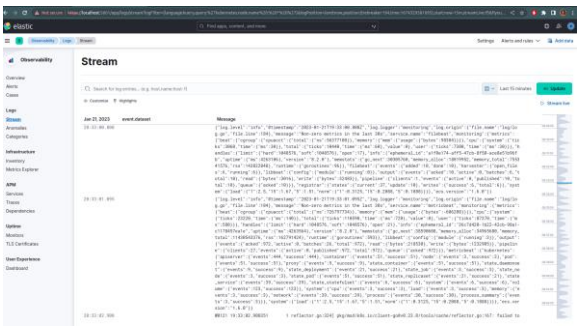

Figure 4: Collecting Kubernetes logs using filebeat.

In the Kibana Logs Stream captured from the Kubernetes cluster logs, we observe a real-time display of log events, including timestamps, log messages, and associated metadata. These may include error messages related to failed deployments, application crashes, resource constraints, network issues, security breaches, and other operational challenges. Additionally, anomalies such as sudden spikes or drops in log activity, unusual patterns in resource consumption, or unexpected behavior in application logs may indicate underlying issues requiring investigation and remediation.

Similarly, Figure 5 illustrates the performance and health metrics collected by Metricbeat, showcased in Kibana's Observability > Metrics section, providing an overview of the containers and pods within the Kubernetes environment.

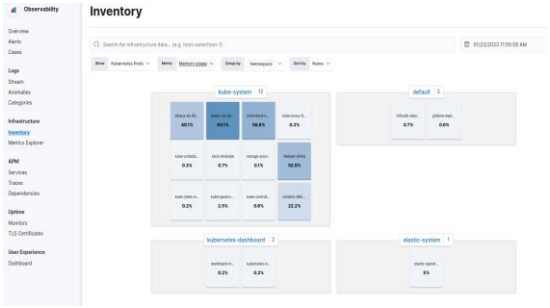

Figure 5: Overview of pods and containers.

The configuration of Filebeat and Metricbeat deployments within our Kubernetes cluster allows for the direct visualization of our Kubernetes resources. The use of ready-made panels is highlighted, which inherently pull information about our configured Kubernetes resources, as shown in Figure 6.

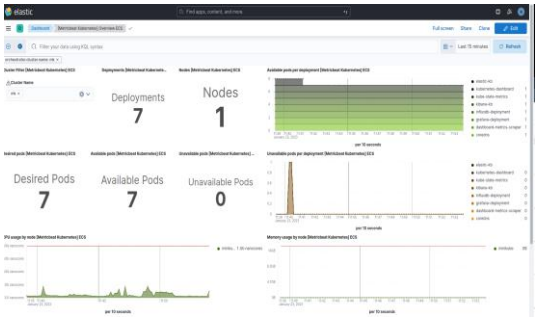

Figure 6: Kubernetes cluster overview.

Utilizing pre-built Kibana dashboards provided by Metricbeat, we gain insights into various aspects of our Kubernetes environment, including node, deployment, and pod overviews. These dashboards, shown in Figure 6, offer valuable insights into the performance and health of our Kubernetes cluster, enabling us to identify potential issues, optimize resource utilization, and ensure the reliability and efficiency of our system.

For instance, if the dashboard indicates a sudden increase in CPU usage across multiple pods, we promptly investigate potential resource bottlenecks and scale up affected pods to ensure optimal performance. Similarly, consistent patterns of pod failures within a deployment prompt analysis of root causes, adjustments to configurations, or implementtation of auto-recovery mechanisms to enhance system reliability. This approach, facilitated by comprehensive insights from Metricbeat dashboards,

allows us to effectively address issues, optimize resource allocation, and sustain overall cluster efficiency.

7 CONCLUSIONS

This research paper explores the development of an automated API testing system with a focus on real-time monitoring and visualization. By integrating containerization tools like Docker and Kubernetes, the study establishes a cohesive testing framework for efficiency and scalability. Using Google Cloud Platform (GCP) further enhances the system's scalability and performance. Additionally, the integration of monitoring tools, particularly Elasticsearch, enables the assessment and visualization of the health and performance of the Kubernetes cluster underlying the testing environment.

By adopting a continuous and automated approach, the research successfully orchestrates these technologies to create a robust and efficient system for daily automation testing. The results highlight the collective impact of these integrated technologies in achieving reliable and effective automated testing processes, ultimately contributing to the advancement of software development.

REFERENCES

[1] P. Agrawal and N. Rawat, "Devops, 'A New Approach To Cloud Development & Testing'," Proceedings of the 2019 Int. Conf. on Issues and Challenges in Intelligent Computing Techniques (ICICT), India, 27-28 Sep. 2019, pp. 1-4.

[2] P.P. Kore, M.J. Lohar, M.T. Surve, and S. Jadhav, "API Testing Using Postman Tool," Int. Journal for Research in Applied Science & Engineering Tech. (IJRASET), 2022, doi: 10.22214/ijraset.2022.48030.

[3] Dh.K. Sharma, "Security Testing of API using Postman and Swagger tools and its use in Internet of Things (IOT)," Journal of Emerging Technologies and Innovative Research, Feb. 2019, vol. 6, no. 2.

[4] J. Shah and D. Dubaria, "Building Modern Clouds: Using Docker, Kubernetes & Google Cloud Platform," IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), 2019, doi: 10.1109/ccwc.2019.8666479.

[5] S. Kumar and C. Saravanan, "A Comprehensive study on Data Visualization tool - Grafana," Journal of Emerging Technologies and Innovative Research, ISSN:2349-5162, vol.8, no. 5, page no.f908-f914, May-2021.

[6] N. Kathare, O.V. Reddy, and V. Prabhu, "A Comprehensive Study of Elasticsearch," International Journal of Science and Research (IJSR), vol. 10, no. 6, June 2021, doi: 10.21275/SR21529233126.

[7] P. Shenoy, S.V. Soudri, R. Kumar, and S. Bailuguttu, "Enhancement of observability using Kubernetes operator," Indonesian Journal of Electrical Engineering and Computer Science, 2022, doi: 10.11591/ijeecs.v25.i1, pp. 496-503.

[8] S. Kaiser, M.S. Haq, A. Tosun, and T. Korkmaz, "Container technologies for ARM architecture: a comprehensive survey of the state-of-the-art," IEEE Access, 2022, doi: 10.1109/ACCESS.2022.3197151.

[9] G. Ambrosino, G.B. Fioccola, R. Canonico, and G. Ventre, "Container mapping and its impact on performance in containerized Cloud environments," IEEE Int. Conf. on Service Oriented Systems Engineering (SOSE), 2020, doi: 10.1109/SOSE49046.2020.00014.

[10] S. Garg and S. Garg, "Automated Cloud Infrastructure, Continuous Integration and Continuous Delivery using Docker with Robust Container Security," Conference: 2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR), 2019, doi: 10.1109/MIPR.2019.00094.

[11] A.M. Potdar, D.G. Narayan, S. Kengond, and M.M. Mulla, "Performance Evaluation of Docker Container and Virtual Machine," Third International Conference on Computing and Network Communications, 2020, doi: 10.1016/j.procs.2020.04.152.