OULUN AMMATTIKORKEAKOULU

GENERATIVE AI-ENHANCED DIAGNOSIS OF KUBERNETES RESOURCES USING K8SGPT AND OLLAMA

Muna Mussa
Bachelor's Thesis
Spring 2025
Degree Programme in Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology

Author(s): Muna Mussa
Title of thesis: Generative AI-Enhanced Diagnosis of Kubernetes Resources Using K8sGPT and Ollama
Supervisor(s): Raija Westerlund
Spring 2025
Number of pages: 52

Cloud computing and containerization have transformed IT infrastructure by providing scalable, flexible, and cost-effective application deployment methods. Cloud computing provides on-demand access to computing resources, accelerating innovation and deployment operational efficiency. Containerization complements this by providing lightweight and portable application environments, ensuring consistency across platforms and driving the adoption of microservices architectures, which enhance scalability and resilience.

However, managing large-scale distributed systems with containers brings complexity, which demands orchestration tools. Kubernetes is an opensource platform created by Google which simplifies deployment, scaling, and management of containerized applications. Despite its advantages, diagnosing and troubleshooting Kubernetes environments is challenging due to the large volume of logs, configuration complexities, and the rabid evolution of the technology.

Generative AI, using natural language processing (NLP) models such as GPT, offers new solutions to Kubernetes diagnostics by simplifying data analysis and providing actionable insights. Tools such as K8sGPT and Ollama showcase how AI enhances the efficiency and ease of Kubernetes management.

This thesis evaluates the impact of generative AI on Kubernetes diagnostics, concentrating on improving operational efficiency and user satisfaction. The study includes an evaluation of K8sGPT and Ollama in practical scenarios to assess their effectiveness in identifying and resolving Kubernetes-related issues.

CONTENTS

GLOSSARY

concept	definition
AI	Artificial Intelligence: The simulation of human intelligence processes by machines, especially computer systems.
CNCF	Cloud Native Computing Foundation: An organization that fosters the development and adoption of cloud-native technologies.
Containerization	The process of packaging software and its dependencies into a container, ensuring consistency across environments.
Generative AI	A branch of AI that generates text, images, or other outputs based on patterns learned from input data.
GPT	Generative Pre-trained Transformer: A type of Deep Neural Network AI model designed for natural language processing tasks.
K8s	Kubernetes: An open-source platform for managing containerized workloads and services.
K8sGPT	A generative AI tool designed for Kubernetes diagnostics and management, offering insights into cluster issues.
Kubelet	An agent running on Kubernetes worker nodes to manage container operations as specified by the cluster's control plane.
Master Node	The control plane in Kubernetes, responsible for cluster management and orchestration.

NLP	Natural Language Processing: A field of AI focused on the interaction with computers through natural human language interface.
Node	A worker machine in Kubernetes that runs applications and is managed by the Kubernetes control plane.
Ollama	A conversational AI tool for Kubernetes, enabling natural language interactions for cluster management.
Pods	The smallest deployable units in Kubernetes, containing one or more containers.
PV	Persistent Volume: A storage resource in Kubernetes that retains data beyond the lifecycle of a pod.
Service	A Kubernetes resource that provides stable networking and access to a set of pods.

1 INTRODUCTION

Cloud computing continues to evolve rapidly. Kubernetes has become the primary technology and the recognized standard for containerized applications. Kubernetes was introduced by Google as an open-source project in 2014. Kubernetes has significantly reshaped organizations' deployment, scaling, and management processes. Its architecture facilitates the automation of essential processes, including deployment, scaling, and management. This provides a dependable platform tailored to the demands of modern software development. Kubernetes' widespread adoption across industries highlights its central role in today’s cloud infrastructure. (1; 2.)

With the growing use of containerization and microservices architecture, managing these distributed systems has become increasingly challenging. Kubernetes alleviates this complexity by offering a unified interface for managing containerized applications. However, this simplification also introduces new challenges, particularly in troubleshooting issues that arise within its multiple layers. Administrators face a range of challenges—from pod failures to resource conflicts and network delays. The sheer volume of logs, metrics, and events generated by Kubernetes clusters can be overwhelming, making diagnostics a difficult and time-consuming process even for experienced professionals. (1; 2.)

The latest progress in AI, particularly in natural language processing and generative technologies, presents new and effective ways to tackle these previously mentioned issues. Generative AI, using models such as GPT, has demonstrated impressive capabilities in analyzing unstructured data and generating actionable insights. Such advancements can simplify the complex Kubernetes environments by providing administrators with insights and recommendations based on log data. Tools such as K8sGPT and Ollama illustrate how AI integration in Kubernetes can make diagnostics more intuitive, offering a natural language interface that even less experienced users can understand. (3; 4.)

The integration of generative AI into Kubernetes diagnostics enhances both efficiency and ease. AI-powered tools provide intelligent analyses and user-friendly interfaces, enabling a broader range of users to manage complex systems effectively. As Kubernetes deployments continue to grow in complexity, the need for advanced diagnostic tools becomes more critical. Generative AI not only simplifies these processes but also reduces the cognitive burden on system administrators, thereby accelerating issue resolution. (4; 27.)

This thesis aims to evaluate the usefulness of generative AI tools in Kubernetes diagnostics. It analyzes the performance of tools such as K8sGPT and Ollama in practical scenarios. The study focuses on how well these tools identify issues, reduce resolution times, and enhance user satisfaction. Specifically, it compares these AI-driven diagnostic tools.

The remainder of this thesis is organized as follows: Chapter 2 provides a literature review covering key technologies such as containers, Kubernetes, generative AI, K8sGPT, and Ollama. Chapter 3 describes the research methodology, experimental setup, analysis of results, and discussion of key findings. Chapter 4 analyzes the results. The thesis concludes with recommendations for future enhancements to Kubernetes diagnostics using AI technologies.

2 BACKGROUND

Software development and deployment have undergone significant transformations with the advent of modern technologies such as cloud computing and containerization. These advancements have redefined how businesses leverage IT resources to ensure flexibility, scalability, and cost-efficiency. Containerization has come up as a pivotal technology in modern application ecosystems, enabling organizations to overcome traditional deployment challenges while unlocking new capabilities for performance and maintenance. (9; 10.)

Cloud computing has significantly changed how businesses access and use IT resources. It provides on-demand availability of shared computing services, allowing for greater flexibility, scalability, and cost efficiency. Smaller businesses and startups have benefited as cloud computing reduces the requirement for large initial investments in IT infrastructure, enabling them to compete more effectively with bigger companies. (11.)

Containerization complements cloud computing by enhancing application portability and resource optimization. It enables the "build once, run anywhere" paradigm, where applications built on local environments can seamlessly transition across testing, staging, and production without modification. This principle reduces errors, accelerates deployments, and supports microservices architectures, further emphasizing containerization’s role in cloud-native application development. (12.)

Containers enhance portability, ensuring applications run reliably across different environments, including local computers, on-premises infrastructure, and cloud services. Such versatility streamlines migrations between these environments and supports multi-cloud approaches, enabling organizations to take advantage of different cloud providers' strengths without needing to modify their applications (13). Unlike traditional virtual machines (VMs), containers are lighter in weight because they share the host's operating system rather than requiring a separate OS for each application. This approach minimizes resource usage, resulting in

quicker startup times and decreased memory requirements. The result is improved system performance, making containerized applications ideal for resource-sensitive and performance-critical scenarios. (14.)

The lightweight nature of containers facilitates effortless scalability, making them an essential tool for modern, distributed systems. Containers can be scaled horizontally, allowing applications to handle increased workloads by deploying multiple instances without downtime. Tools such as Kubernetes automate this process, dynamically scaling containerized applications based on demand. This elasticity guarantees optimal resource utilization and makes containers well-suited for cloud-native applications that require resilience and adaptability to fluctuating workloads. (15; 26.)

Kubernetes has achieved widespread adoption, extensive ecosystem, and robust features such as scalability, self-healing, and declarative configuration, underscoring the importance of understanding containerization's inner workings (1; 6; 15). Containers simplify the deployment and scaling of Kubernetes resources across diverse cloud environments, improving management, diagnostics, and operational efficiency. (2; 8; 16; 26.)

Generative AI tools such as K8sGPT and Ollama enhance this process by providing intelligent diagnostics and insights, enabling faster resolution of issues and optimization of Kubernetes resources. Without these tools, identifying misconfigurations or inefficiencies in Kubernetes clusters would require significant manual effort and expertise, potentially leading to delays in resolving critical issues and suboptimal resource usage. The advanced capabilities of K8sGPT and Ollama ensure that operational teams can proactively manage workloads, streamline diagnostics, and implement predictive scaling strategies that would be challenging to achieve through conventional methods. (3; 4; 20; 27.)

For instance, K8sGPT has been used to analyze complex Kubernetes cluster configurations, detecting misconfigurations and offering actionable recommendations, while Ollama has facilitated predictive scaling by analyzing workload patterns and suggesting resource adjustments in real-time (4; 20).

Kubernetes' ability to manage containerized workloads dynamically makes it a cornerstone of scalable, resilient cloud-native architectures. (1; 16.)

Containerization, cloud computing, and generative AI tools play a key role in creating scalable and efficient application systems. Technologies such as Kubernetes, K8sGPT, and Ollama work together to improve diagnostics and operations, making IT environments more optimized and paving the way for innovation in containerized systems. (1; 3; 4; 20.)

2.1 Containers and virtual machines

Using containers to package applications provides significant advantages compared to conventional virtual machines (VMs). Containers operate by utilizing the underlying host operating system, which minimizes resource consumption and enhances efficiency, resulting in quicker and more streamlined performance. Unlike VMs, which each need their own separate operating systems, containers only package the application along with necessary dependencies. Additionally, this encapsulation eliminates common deployment challenges such as conflicting dependencies and inconsistent environments, greatly simplifying application deployment and management. (5.)

The portability of containers has transformed the way developers approach deployment. Applications built locally can run identically in testing, staging, and production environments, minimizing deployment errors and improving efficiency (5). Additionally, containers form the backbone of microservices architectures by breaking applications into smaller, independent services. Organizations can build, launch, and expand these microservices separately, allowing them to respond swiftly to evolving demands and enhance both flexibility and robustness. Containers' lightweight nature and ability to operate uniformly across environments make them an indispensable technology for modern, distributed application ecosystems. (6.)

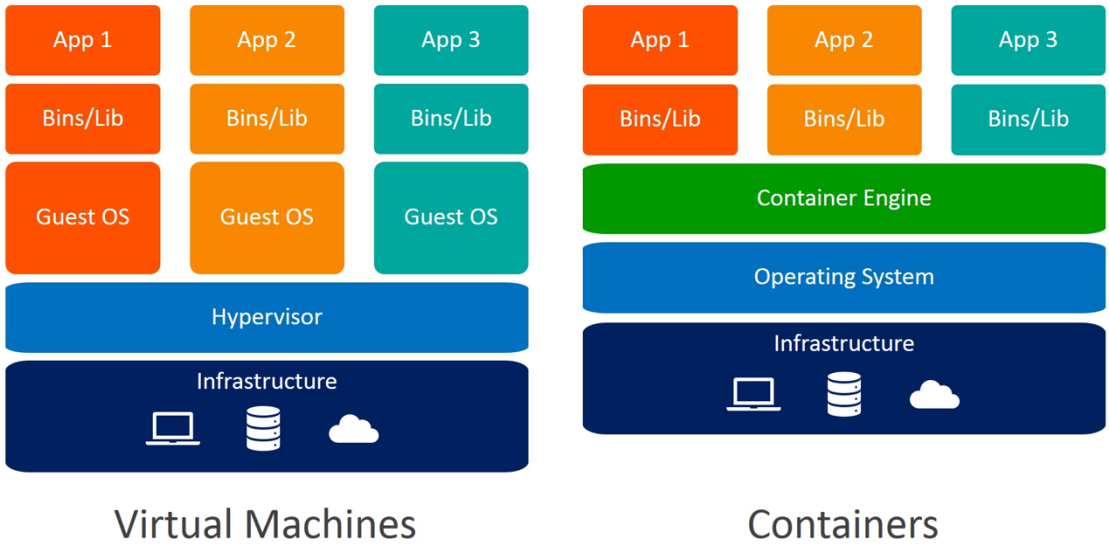

FIGURE 1. Comparison between container and virtual machine (VM). (11.)

As shown in FIGURE 1, containers and virtual machines (VMs) differ fundamentally in their architecture and resource utilization. Virtual machines operate on a hypervisor and include their own guest operating systems, which provide strong isolation but result in higher resource consumption and slower startup times. In comparison, containers share the host operating system through a container engine, making them lightweight, faster to start, and highly portable across environments. While VMs are ideal for running diverse operating systems and ensuring complete isolation, containers excel in supporting microservices architectures, dynamic scaling, and efficient use of resources. This distinction highlights the growing preference for containers in cloud-native applications, where agility and scalability are critical. (11.)

2.2 Need for Orchestration Tools

While containers have simplified application deployment, managing large numbers of containers presents new challenges. Issues such as load balancing, scaling, and fault tolerance require orchestration to maintain efficiency. These

Kubernetes tools help maintain reliability and efficiency when managing complex containerized environments. (6.)

Orchestration tools offer centralized control, enabling automated failover, service discovery, and rolling updates that help ensure high availability. Beyond basic management, they provide visibility into application performance and resource utilization, which allows teams to optimize infrastructure and address bottlenecks proactively. Orchestration tools such as Kubernetes are now integral to any cloud-native environment. (6; 9.)

2.3 Evolution and Features of Kubernetes

Kubernetes has fundamentally transformed how organizations deploy, scale, and manage containerized applications. Kubernetes was donated to the Cloud Native Computing Foundation (CNCF) by Google. This platform is a result of lessons learned from Google's previous orchestration tools, Borg and Omega, which were pivotal in shaping the evolution of container orchestration. (6; 25.)

The primary purpose of Kubernetes is to automate the operational tasks involved in managing distributed applications by simplifying the underlying infrastructure. This abstraction allows Kubernetes to automatically maintain the "desired state" of an application, if a failure occurs, Kubernetes will work to restore the system to its intended state. (6.)

Kubernetes follows a declarative model: users define what they want to achieve, and Kubernetes uses controllers and monitoring to ensure that this desired state is continuously maintained. This significantly reduces the need for manual intervention, thereby making applications more robust and resilient even when failures or infrastructure changes occur. Its integration with cloud-native tools and extensibility have made Kubernetes the leading solution for managing modern software applications. (1; 25.)

2.3.1 Key Features of Kubernetes

Key features of Kubernetes include automated deployment, self-healing, horizontal scaling, and resource efficiency. Kubernetes automates the entire deployment process, managing the application lifecycle, updates, and scaling seamlessly. In cases where a containerized application or pod fails, Kubernetes employs self-healing capabilities to automatically restart the affected components, ensuring the system remains functional without manual intervention (1; 2). Horizontal scaling is another critical feature, enabling Kubernetes to support both manual and automatic scaling of applications, dynamically adapting resources based on demand (2). Additionally, Kubernetes ensures efficient resource utilization through intelligent scheduling, optimizing infrastructure use to its fullest potential. (1; 6; 25.)

These features, including automated rollouts, rollbacks, and horizontal scaling, have firmly established Kubernetes as the go-to platform for handling fluctuating workloads and ensuring reliability. Kubernetes' ability to maintain a stable application state without manual adjustments, combined with its seamless integration with other cloud-native tools, makes it an ideal choice for modern development environments. (2; 6.)

2.3.2 Kubernetes resources

Kubernetes manages applications using a set of logical units called resources, which help organize and orchestrate workloads. These primary resources include pods, which are the smallest deployable units in Kubernetes. Pods manage one or more containers that share storage and networking and are ephemeral, meaning they are automatically replaced when failures occur. Deployments offer a more abstract way to manage pods, maintaining a defined number of replicas and supporting seamless rollouts and rollbacks. Services provide reliable access points to pods, enabling stable communication both inside and outside the Kubernetes cluster. *ConfigMaps* and *Secrets* handle configuration settings and sensitive data, helping applications stay independent of their environment. *Persistent Volumes (PV)* and *Persistent Volume Claims (PVC)* allow stateful

applications to use persistent storage that is managed separately from the application's lifecycle. (9; 24; 25.)

2.4 Kubernetes Architecture

The architecture of Kubernetes is designed to support scalability, reliability, and efficient handling of containerized applications. It is based on a master-worker node model, which separates the control and execution functions, ensuring that the system remains scalable and fault tolerant. (9; 10; 24.)

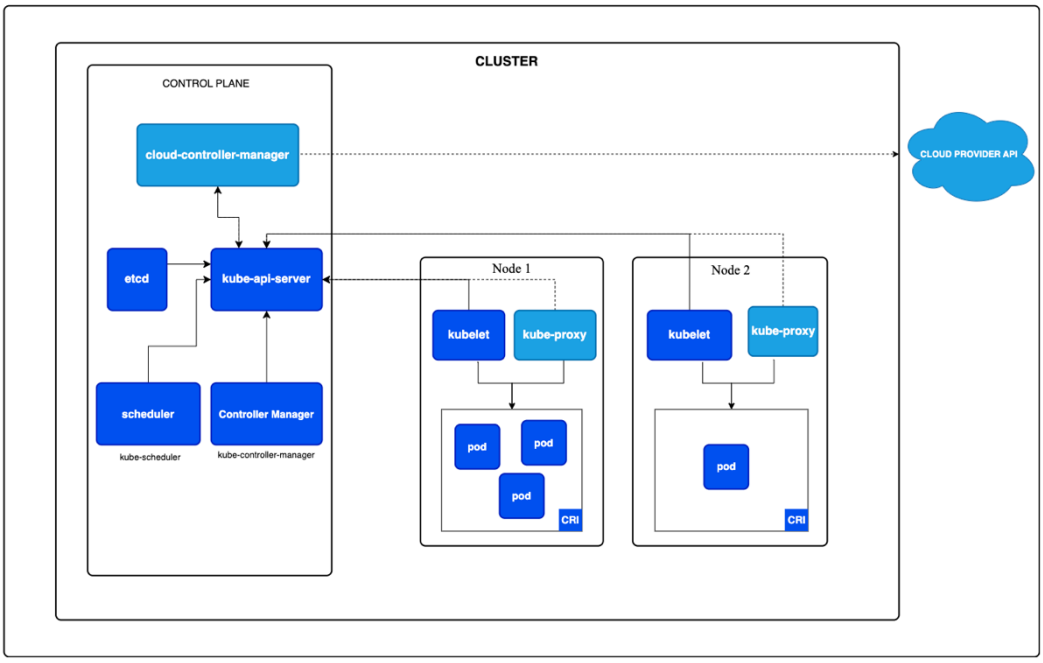

FIGURE 2. Kubernetes cluster components. (13.)

As shown in FIGURE 2, the architecture consists of a Control Plane and worker nodes. The Control Plane is central to cluster management and includes several critical components. The Kube API Server acts as the primary interface for cluster management, exposing the Kubernetes API and serving as the communication bridge between users, components, and the cluster. The *etcd* component functions as the cluster's key-value database, maintaining the desired and actual state of the system, while the *Scheduler* assigns workloads to nodes based on

available resources and constraints. Additionally, the Controller Manager runs various controllers that ensure the cluster's state aligns with its configuration, and the Cloud Controller Manager facilitates interaction with underlying cloud provider APIs to manage resources such as load balancers and storage. (13.)

Worker nodes, depicted as Node 1 and Node 2 in FIGURE 2, execute workloads and host the application pods. Each node includes a *Kubelet*, which ensures the containers within pods are running as intended, and a *Kube-proxy*, which manages network communication between pods and services. The smallest deployable units within Kubernetes, Pods, run on these nodes and encapsulate one or more containers that share resources. (13; 24.)

The Control Plane and worker nodes interact seamlessly, with the Kube API Server orchestrating workload scheduling and communication across the cluster. Each of these components will be discussed in detail in the following sections to provide a comprehensive understanding of their roles and functionalities within a Kubernetes cluster. (13; 24.)

2.4.1 Master Node

The master node oversees managing the state of the Kubernetes cluster. It functions as the brain of the cluster, orchestrating and managing containers effectively. Several key components reside within the master node, each playing a vital role in cluster management.

The API Server serves as the primary starting point for managing Kubernetes, exposing Kubernetes APIs for interaction via command line tools, HTTP requests, or third-party integrations. The Controller Manager maintains the desired state of the cluster by overseeing various controllers, such as those managing nodes, endpoints, and replication. The Scheduler assigns workloads to worker nodes based on available resources, ensuring that containerized applications are efficiently distributed across the cluster. The cluster data, including configuration information and metadata, are stored in *etcd*, a distributed

key-value store. It ensures consistency across nodes and maintains cluster state. (6; 10.)

2.4.2 Worker Nodes

Worker nodes carry out the tasks and processes required to run applications within a Kubernetes cluster. These nodes contain key components necessary for managing containers. One such component is the *Kubelet*, an agent that operates on every worker node. It communicates with the container runtime to make sure containers are functioning as expected, aligning the actual state of the system with the desired configuration defined by the control plane (6; 10.). Additionally, the Kube Proxy handles the networking aspect by setting up rules that manage traffic flow both within the cluster and externally. This includes functions like routing and load balancing to maintain efficient communication (10.). The container runtime is the software layer that launches and manages containers, and Kubernetes is compatible with various runtimes, including Docker and others. (9; 10.)

The separation between master and worker nodes enables Kubernetes to scale seamlessly while maintaining fault tolerance. When additional processing power is required, more worker nodes can be added without modifying the master node, making Kubernetes a highly adaptable platform for growing workloads (10.).

2.4.3 Key Architectural Features

Kubernetes' architecture is built on a master-worker separation, ensuring that control processes are isolated from execution processes. This design promotes high availability and fault tolerance by distributing tasks effectively between the master and worker nodes. The platform's scalability is enhanced by its ability to add worker nodes dynamically, allowing for horizontal scaling to accommodate increased workloads without disrupting existing processes. Kubernetes also excels in network management, utilizing Kube Proxy to manage communication between pods and ensure seamless integration across the cluster (10.).

Kubernetes uses these architectural components to manage containerized applications in an efficient and powerful manner. By balancing automation, scalability, and reliability, Kubernetes meets modern cloud-native environments' demands. (6; 9; 10.)

2.5 Challenges in Diagnosing Kubernetes Environments

Despite Kubernetes' powerful architecture and features, diagnosing issues in Kubernetes environments remains a complex and demanding task. Administrators face several key challenges that make troubleshooting both time-consuming and challenging as listed below. (3.)

2.5.1 Resource Conflicts

In Kubernetes environments, resources such as CPU and memory are shared across multiple containers. Misconfigurations or insufficient resource allocations can lead to application failures or degraded performance. For example, if multiple containers are trying to consume more resources than the node can provide, some applications may become unresponsive or crash. (9.)

2.5.2 Networking Issues

The networking layer in Kubernetes is a complex interplay between services, ingress controllers, and external network traffic. Misconfigurations or bottlenecks in networking components can lead to connectivity issues, latency, or even complete failure in service communication. These issues can be challenging to diagnose because they may involve multiple components of the network stack. (10.)

2.5.3 Application Misconfigurations

Kubernetes relies heavily on configuration files (YAML) to define the state of applications and resources. A simple mistake in a YAML file such as incorrect indentation, missing fields, or incompatible settings can prevent an application from deploying correctly or functioning as intended. Such misconfigurations are often difficult to identify without specialized tools. (9.)

2.5.4 Log and Metric Overload

Kubernetes clusters generate vast amounts of logs, metrics, and event data. Although this data is essential for monitoring and diagnostics, the sheer volume can overwhelm administrators, making it difficult to identify and isolate the root cause of issues. Without the right tools, sifting through these logs to detect patterns or anomalies becomes a cumbersome task. (3.)

2.5.5 Distributed Nature of Kubernetes

Kubernetes is inherently distributed, which adds an additional layer of complexity to diagnosing problems. An issue that originates in one component, such as a failing pod, can cascade through the system, impacting other services or even the entire application. Identifying the original source of these cascading failures can be very challenging. (6.)

2.5.6 Rapid Evolution and Complexity of Kubernetes

Kubernetes is an evolving technology, with frequent updates and new features being introduced regularly. Keeping up with these updates, understanding new capabilities, and adjusting configurations accordingly require ongoing learning and adaptation. This rapid evolution adds another level of complexity to diagnostics, as administrators need to stay updated on best practices and new tools. (6.)

Addressing these challenges effectively demands the use of advanced diagnostic tools. Tools that can aggregate logs, analyze metrics, detect anomalies, and provide actionable insights are critical for reducing troubleshooting times and improving operational efficiency. (3.)

Kubernetes environments need diagnostic solutions that help manage the distributed and dynamic nature of the system, effectively turning data overload into valuable information for root cause analysis and rapid problem resolution (10).

2.6 Generative AI in Kubernetes Diagnostics

Generative AI constitutes a crucial advancement in artificial intelligence, particularly through deep learning architectures like the Generative Pre-trained Transformer (GPT). These models excel at handling unstructured information and producing responses that are both logically consistent and contextually relevant. They excel in natural language processing (NLP) tasks, making them perfect for interpreting complex and varied types of information. Generative AI models are trained on massive datasets, enabling them to respond to nuanced queries with detailed and contextually appropriate answers. This makes them especially valuable for handling unstructured data in IT settings, such as logs, configuration files, and error messages. (5; 27.)

In IT operations, generative AI is transforming how diagnostics are approached. Traditional diagnostic tasks such as log analysis, anomaly detection, and root cause identification are now being automated by these AI models. By efficiently parsing through large volumes of textual data and identifying patterns or anomalies, generative AI significantly lightens the cognitive load on IT teams. This frees them to focus on resolving issues rather than spending time deciphering data. (6; 27.)

2.6.1 Application of GPT Architecture in IT Operations

The GPT architecture has fundamentally changed how unstructured data is analyzed within IT operations. It can quickly identify patterns in logs, events, or error messages, allowing real-time detection of issues and providing suggested resolutions. This speed and precision make it highly effective for troubleshooting Kubernetes environments, where the sheer volume of data can overwhelm traditional methods. (6.)

For example, GPT models can analyze error logs to spot recurring issues, identify misconfigurations in YAML files, or recommend improvements in resource allocation. They can integrate with existing monitoring systems, providing a seamless diagnostic experience across various IT ecosystems. Additionally, their natural language capabilities allow administrators to ask questions in simple, conversational language, making diagnostics accessible even to those who may not have deep technical expertise. (9; 27.)

2.6.2 K8sGPT

K8sGPT is an AI-powered tool designed to simplify Kubernetes diagnostics. By analyzing logs and configurations, it can detect issues and provide actionable solutions. Using machine learning models, K8sGPT identifies anomalies and suggests resolutions, which significantly reduces troubleshooting time and effort. Integrating K8sGPT into Kubernetes workflows enhances cluster management reliability and efficiency. (4.)

One of the notable strengths of K8sGPT is its ability to present diagnostic data through user-friendly dashboards or summaries. These visualizations help administrators quickly understand the state of their clusters. (4.)

2.6.3 Ollama

Ollama brings a conversational AI interface to Kubernetes diagnostics, making it possible for administrators to interact with clusters using simple natural language.

This feature significantly lowers the barriers for users with less technical expertise, enabling them to query cluster health, pinpoint issues, and receive solutions without requiring a deep understanding of Kubernetes architecture. Ollama's capability to generate human-such as responses enhances both accessibility and collaboration in cluster management. (12.)

What distinguishes Ollama is its focus on user experience. Instead of navigating complex diagnostic tools, administrators can ask straightforward questions such as, "Why is my pod failing?" or "How do I fix this configuration error?" and receive understandable, actionable guidance. This kind of conversational interface not only saves time but also boosts the confidence of users who might be less familiar with Kubernetes' intricate details. (4; 12.)

2.7 Comparison of AI vs. Traditional Methods

The use of AI-powered tools in Kubernetes diagnostics offers distinct advantages over traditional methods, such as manual log analysis and static monitoring systems. Traditional approaches rely heavily on human expertise, requiring administrators to manually sift through extensive logs and metrics to identify issues. This process is time-consuming, prone to errors, and often inadequate in addressing the scale and complexity of modern Kubernetes environments. (4.)

In contrast, AI-powered tools such as K8sGPTanalyze logs, configurations, and performance metrics in real-time, detecting anomalies and identifying root causes with high precision. AI accelerates the diagnostic process, reducing time-to-resolution significantly. Furthermore, AI tools offer predictive analytics capabilities, identifying potential issues before they escalate, which is something traditional methods cannot achieve. For example, while manual log analysis might miss subtle patterns indicative of resource bottlenecks, AI models can correlate historical data to predict such scenarios. (3; 4.)

TABLE 1. Comparative analysis highlights the following differences:

Aspect	Traditional Diagnosis	K8sGPTand Ollama Diagnosis
Efficiency	Time-consuming due to manual inspection of logs and metrics. Requires experienced administrators to correlate data from multiple sources.	Automates the diagnostic process, analyzing logs, metrics, and events in real time. Reduces diagnostic time significantly. (1; 4; 20.)
Accuracy	Prone to human error and oversight, especially in large or complex clusters. Missed misconfigurations or patterns are common.	Uses AI models trained on Kubernetes-specific patterns, providing higher accuracy and consistency in issue detection. (4; 20.)
Scalability	Struggles with large-scale clusters due to the volume of data and complexity. Human limitations hinder effective diagnostics in dynamic environments.	Scales seamlessly to analyze large clusters. Can handle dynamic environments with rapid changes and a high volume of data. (1; 6.)
Insights	Insights depend on the expertise and experience of the administrator. May lack structured, actionable recommendations.	Generates structured, human-readable insights and recommendations, making issue resolution easier for administrators. (4; 20.)
Customization	Limited customization predefined rules and tools need to be adjusted manually for each cluster's specifics.	Models such as K8sGPTcan be fine-tuned for specific Kubernetes environments. YAML configurations allow precise customization. (3; 4.)
Proactive Diagnosis	Relies on reactive analysis; issues are addressed only after symptoms appear.	Supports proactive monitoring and predictions. Detects issues before they escalate, such as resource

		misallocations or scaling inefficiencies. (4; 20.)
Integration	Often fragmented, requiring multiple tools for logs, metrics, and events. Integrations may be inconsistent across platforms.	Provides a unified framework for diagnostics, integrating seamlessly with Kubernetes clusters through APIs and YAML configurations. (1; 20.)
Cost	High operational costs due to the need for skilled personnel and extended diagnostic times.	Reduces operational costs by automating diagnostics and minimizing reliance on manual intervention. (1; 6.)
User Interface (UI)	Logs and metrics are typically analyzed in raw formats, requiring expertise to interpret.	Provides summaries and actionable insights in natural language, enhancing usability for administrators at all levels. (4; 20.)
Examples of Issues Diagnosed	Pod failures, resource contention, scaling issues—diagnosed through log inspection and manual debugging.	Misconfigurations, pod failures, node issues, scaling inefficiencies—diagnosed in real time with recommendations. (1; 4; 20.)

These differences underscore why AI is increasingly being adopted for Kubernetes diagnostics. Tools such as Ollama's conversational interface make it easier for less experienced users to interact with Kubernetes, further broadening the usability of these systems. (4; 6; 12.)

2.8 Impact of Generative AI on Kubernetes Management

The integration of generative AI into Kubernetes diagnostics has transformative implications for Kubernetes management. Beyond merely identifying and resolving issues, AI-powered tools enhance the overall operational efficiency, reliability, and accessibility of Kubernetes environments. By automating repetitive and complex tasks, AI reduces the workload on administrators, allowing them to focus on strategic initiatives. (4; 6; 12; 27.)

One of the most significant impacts is making Kubernetes management easier and more accessible. Tools such as Ollama's conversational AI interface allow even less-experienced users to effectively manage Kubernetes clusters. This reduces the need for highly specialized expertise, enabling more organizations to use Kubernetes effectively. (12.)

Generative AI also enables proactive management by leveraging predictive analytics to anticipate issues before they occur. For example, AI can monitor historical performance metrics to predict resource bottlenecks or identify misconfigurations that could lead to future failures. This proactive approach minimizes downtime and enhances the reliability of Kubernetes-managed applications. (4; 6; 12.)

Furthermore, AI’s ability to provide actionable insights streamlines resource allocation, optimizes system performance, and ensures seamless scalability. The cumulative impact is a more resilient, efficient, and user-friendly Kubernetes ecosystem, which is critical as cloud-native applications continue to grow in complexity and scale. (6; 12.)

3 DESIGN

This chapter discusses the requirements for implementing the proposed solution to troubleshoot Kubernetes resource issues using Generative AI-enhanced diagnostics. The focus is on leveraging K8sGPT and Ollama to efficiently identify and resolve issues related to incorrectly tagged container images and other resource-related misconfigurations within a Kubernetes cluster.

The requirements section explores the role of generative AI models, particularly K8sGPT and Ollama, in diagnosing Kubernetes resource problems. These AI-driven models enhance cluster monitoring by analyzing complex Kubernetes environments, enabling faster issue detection for troubleshooting.

To support this implementation, components such as YAML configurations, Docker, and KIND (Kubernetes IN Docker) are utilized. These tools facilitate the creation of a Kubernetes cluster, enabling controlled testing of deployment failures and resource misconfigurations. The integration of K8sGPT with Ollama enhances automated troubleshooting, streamlining Kubernetes operations and optimizing system performance.

3.1 Requirements

The thesis work involves deploying a Kubernetes cluster using Docker, YAML configurations, and a KIND (Kubernetes in Docker) to simulate real-world diagnostic challenges. The design replicates common issues in Kubernetes environments, such as incorrectly tagged container images among others for testing purposes.

3.2 Proposed Solution

Docker

Docker is widely used to containerize applications and their dependencies, creating isolated, consistent, and reproducible environments for development and testing. While Kubernetes now relies on container runtimes like *containerd* under the hood, Docker remains a key tool for building and packaging workloads that run in Kubernetes clusters. Its broad adoption, mature ecosystem, and developer-friendly tooling make it an asset in the containerization workflow, ensuring reliability and portability across environments. (3; 6.)

KIND (Kubernetes in Docker)

Facilitates local Kubernetes cluster deployment directly in Docker containers. KIND simplifies testing scenarios by enabling easy setup, teardown, and modification of clusters. (3.)

YAML

YAML a human-readable data serialization format, is vital for defining the desired state of Kubernetes components and workloads. They allow precise and flexible cluster management. For instance, workload definitions specify tasks such as CPU-intensive, memory intensive, and I/O-heavy workloads, enabling comprehensive stress-testing of the cluster. YAML manifests also help simulate fault scenarios, such as resource contention, pod disruptions, node failures, and misconfigured services, providing a controlled environment to evaluate the system's resilience. Additionally, YAML configurations facilitate cluster adjustments by modifying resource limits, affinity rules, and scaling policies to mimic real-world operational challenges. This flexibility makes YAML a key tool for optimizing and managing Kubernetes environments. (3; 5.)

K8sGPT

Configured to analyze Kubernetes logs, events, and metrics. It generates human-readable diagnostics by identifying patterns and anomalies, such as pod failures or resource contention.

Ollama

Processes K8sGPT's output to create summaries and actionable recommendations, assisting administrators in resolving issues efficiently.

Together, these tools provide a comprehensive approach to Kubernetes diagnostics, bridging the gap between raw data analysis and actionable insights. (3; 4; 20.)

OpenAI (Cloud AI)

OpenAI's cloud-based AI solutions, such as K8sGPT, leverage cloud computing to provide scalable, resource-efficient, and easily accessible tools for Kubernetes diagnostics. These solutions require minimal local resources and offer dynamic computational power, making them ideal for organizations without robust infrastructure. OpenAI ensures regular updates and improvements, keeping the tools cutting-edge. However, cloud-based AI may introduce latency, depend on stable internet connectivity, and raise concerns about data privacy, especially in regulated industries. Its usage-based pricing model can be cost-effective for smaller operations but may scale up costs for continuous, high-volume usage. (3; 20; 22.)

Local AI

Local AI solutions, such as Ollama, process data entirely on-premises, ensuring data privacy and compliance with regulations such as GDPR. These tools reduce latency by avoiding internet dependencies and offer greater control and customizability for organizations with unique requirements. Local AI is particularly suitable for sensitive workloads or environments where data security is paramount. However, it demands significant local computational resources and storage, leading to higher hardware and maintenance costs. While it offers independence from external providers, scaling Local AI for large workloads can be challenging without substantial infrastructure investments. (15; 20; 22.)

Kubernetes Architecture

As shown in FIGURE 3, the system architecture follows a multi-node Kubernetes deployment to simulate real-world cluster environments. The architecture consists of a control plane and three worker nodes, each playing a crucial role in managing and executing workloads.

The control plane is responsible for overseeing cluster operations and maintaining its desired state. It includes the API Server, which handles all Kubernetes requests and manages communication between components; the Scheduler, which assigns workloads to worker nodes based on resource availability; the Controller Manager, which ensures system components function as intended; and etcd, a distributed key-value store that holds cluster configuration and metadata.

The worker nodes are responsible for executing workloads and managing containerized applications. Each worker node runs a Kubelet, which ensures that pods and containers operate correctly; a Container Runtime (Docker), which manages and executes containers within the node; and a Kube Proxy, which maintains network communication between pods and services.

This distributed setup allows for realistic testing of resource allocation failures, scheduling errors, and deployment misconfigurations. By integrating K8sGPT and Ollama, the system can automatically diagnose and resolve different issues. For e.g. issues related to incorrectly tagged container images. In this thesis for such types of problem resolution, a Kubernetes-based environment is used for testing purposes.

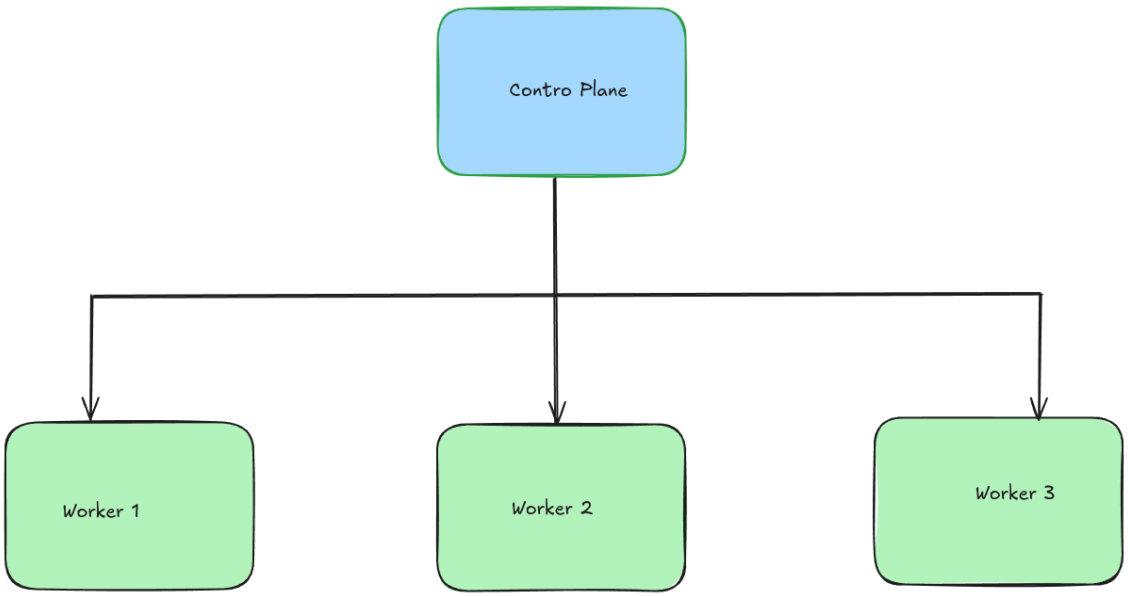

FIGURE 3. Proposed Kubernetes architecture.

The following section presents the implementation details of the proposed solution. It describes the practical steps undertaken to deploy the designed architecture, including the creation of the local Kubernetes cluster using KIND, containerization with Docker, YAML-based deployment configurations, and the integration of k8GPT and Ollama for diagnosing Kubernetes deployment issues, such as incorrectly tagged images.

4 IMPLEMENTATION

This chapter presents the detailed implementation of the proposed solution. We will be looking at the experimentation environment for resolving Kubernetes issues, as well as the details on how to build, deploy, and troubleshoot the failure in Kubernetes cluster.

4.1 Environment

The experimental setup for this thesis involved a Kubernetes cluster deployed on a single virtual machine (VM) in the CSC cPouta OpenStack cloud environment. As shown in FIGURE 4, the VM was configured with the Ubuntu 24.04 LTS operating system, using the `standard.xxlarge` flavor, which provides 8 vCPUs, 32 GB of RAM, and 80 GB of root disk storage. Within this VM, a KIND (Kubernetes IN Docker) cluster was deployed, comprising a single control plane and three worker nodes. The Kubernetes command-line tool (`kubectl`) was used to communicate with the Kubernetes API server, enabling efficient management and interaction with the cluster.

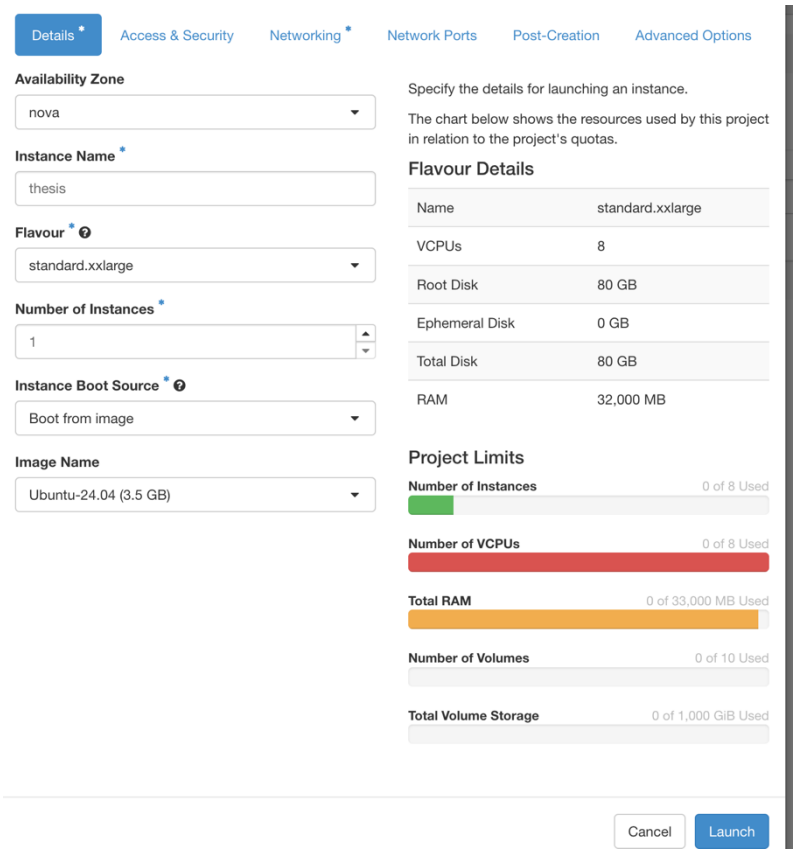

FIGURE 4. OpenStack virtual machine specification.

Upon initializing the virtual machine, essential updates and upgrades are applied using the following commands to ensure system security and reliability:

```
sudo apt update
```

```
sudo apt upgrade -y
```

Subsequently, Docker, a prerequisite for running Kubernetes IN Docker (KIND), was installed following the official Docker repository as shown in FIGURE 5.

```
ubuntu@thesis:~$ docker version
Client: Docker Engine - Community
 Version:           27.4.1
 API version:       1.47
 Go version:        go1.22.10
 Git commit:        b9d17ea
 Built:             Tue Dec 17 15:45:46 2024
 OS/Arch:           linux/amd64
 Context:           default

Server: Docker Engine - Community
 Engine:
  Version:          27.4.1
  API version:      1.47 (minimum version 1.24)
  Go version:       go1.22.10
  Git commit:       c710b88
  Built:            Tue Dec 17 15:45:46 2024
  OS/Arch:          linux/amd64
  Experimental:     false
 containerd:
  Version:          1.7.24
  GitCommit:        88bf19b2105c8b17560993bee28a01ddc2f97182
 runc:
  Version:          1.2.2
  GitCommit:        v1.2.2-0-g7cb3632
 docker-init:
  Version:          0.19.0
  GitCommit:        de40ad0
ubuntu@thesis:~$
```

FIGURE 5. Docker Installation details.

FIGURE 5 demonstrates the verification of Docker installation on the configured VM. The figure displays the execution of the docker version command, clearly detailing the Docker client and server versions, confirming a successful installation. The Docker Engine (Community Edition version 27.4.1), additional components such as `containerd (1.7.24)`, `runc (version 1.2.2)`, and `docker-init (version 0.19.0)` ensure compatibility and stability for Kubernetes container orchestration.

4.2 KIND Cluster installation and configuration

As shown in FIGURE 6 and FIGURE 7the installation and configuration of KIND cluster, displaying the installed version (v0.26.0) and the cluster configuration file (`kind-config.yaml`). This YAML file explicitly defines the Kubernetes cluster setup with one control plane and three worker nodes, required for the intended architecture. This detailed configuration enables the cluster to effectively simulate real-world Kubernetes environments for the intended diagnostic tests.

```
ubuntu@thesis:~$ kind version
kind v0.26.0 go1.23.4 linux/amd64
ubuntu@thesis:~$ cat kind-config.yaml
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
nodes:
- role: control-plane
- role: worker
- role: worker
- role: worker

ubuntu@thesis:~$
```

FIGURE 6. KIND version and the sample config file.

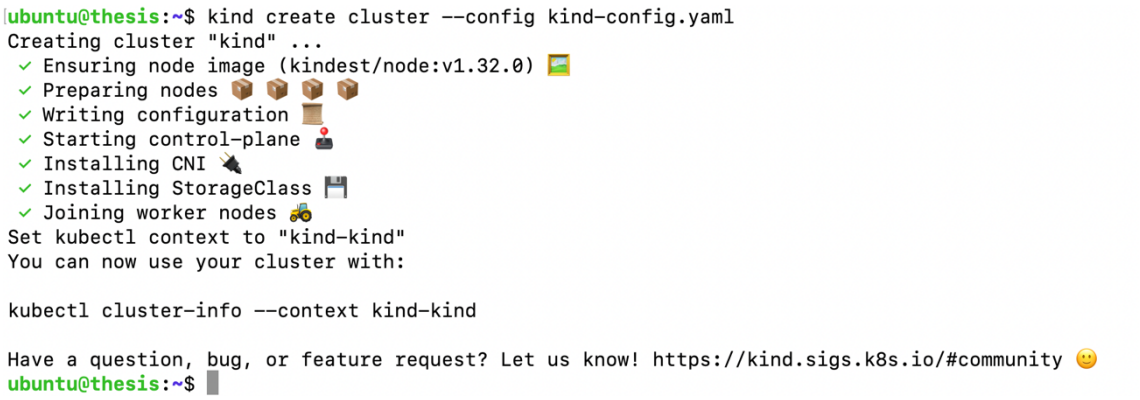

FIGURE 7. KIND create a Kubernetes cluster.

As depicted in FIGURE 3, the Kubernetes architecture implemented in this research comprises one Control Plane and three Worker Nodes. The Control Plane manages cluster operations through the API Server, Scheduler, Controller Manager and *etcd* (a distributed key-value store). On the other hand, worker nodes handle workload execution, running containerized applications using Docker, managed by components like *Kubelet* and *Kube Proxy*.

4.3 Kubectl version

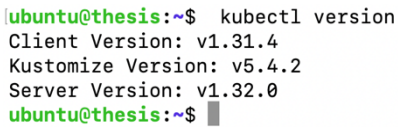

FIGURE 8. Details from kubectl version command.

The client version as shown in FIGURE 8 is `v1.31.4`, which is used to communicate with the Kubernetes cluster. Additionally, the *kustomize* version is `v5.4.2`, providing support for customizing Kubernetes objects. The server version of the Kubernetes cluster is `v1.32.0`, ensuring compatibility between the client and server for cluster management operations. This confirms the successful installation and version compatibility required for the research environment.

4.4 Setting Kubernetes cluster

The Kubernetes cluster is created using KIND, leveraging the following command:

```
kind create cluster --config kind-config.yaml
```

This command sets up a Control Plane and three Worker Nodes according to configurations defined in `kind-config.yaml`. The created cluster simulates realistic Kubernetes deployment scenarios suitable for diagnostic testing.

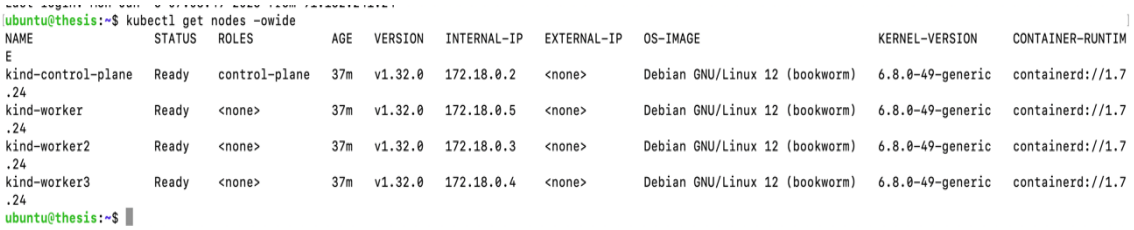

FIGURE 9.Node cluster information.

FIGURE 9 depicts the Kubernetes cluster which consists of a control plane node (kind-control-plane) and three worker nodes (kind-worker, kind-worker2, and kind-worker3), all in a Ready state. Each node is running Kubernetes version `v1.32.0` with the internal IPs `172.18.0.2, 172.18.0.5, 172.18.0.3, and 172.18.0.4`, respectively. The nodes are based on the Debian GNU/Linux 12 (Bookworm) OS image, with kernel version `6.8.0-49-`generic and contained `v1.7` as the container runtime. This configuration forms a functional multi-node Kubernetes cluster for research experiments.

4.5 Deployment of K8sGPTand Ollama

The installation and deployment of K8sGPT and Ollama in the Kubernetes environment are illustrated in FIGURE 10 and FIGURE 11, respectively.

```
ubuntu@thesis:~$ curl -LO https://github.com/k8sgpt-ai/k8sgpt/releases/download/v0.3.48/k8sgpt_amd64.deb
sudo dpkg -i k8sgpt_amd64.deb
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
100 34.5M  100 34.5M    0     0  16.3M      0  0:00:02  0:00:02 --:--:-- 27.3M
Selecting previously unselected package k8sgpt.
(Reading database ... 106596 files and directories currently installed.)
Preparing to unpack k8sgpt_amd64.deb ...
Unpacking k8sgpt (0.3.48) ...
Setting up k8sgpt (0.3.48) ...
```

FIGURE 10. Installation of the k8sgpt tool using the curl and dpkg commands.

FIGURE 10 demonstrates the installation of K8sGPT, achieved through downloading the official Debian package using the following commands:

```
curl -LO https://github.com/k8sgpt-
ai/k8sgpt/releases/download/v0.3.48/k8sgpt_amd64.deb
```

```
sudo dpkg -i k8sgpt_amd64.deb
```

The output confirms the successful setup of `k8sgpt` version `0.3.48`, making it available for diagnostic analyses of Kubernetes resource issues.

```
ubuntu@thesis:~$ curl -fsSL https://ollama.com/install.sh | sh
>>> Installing ollama to /usr/local
>>> Downloading Linux amd64 bundle
######################################################################## 100.0%
>>> Creating ollama user...
>>> Adding ollama user to render group...
>>> Adding ollama user to video group...
>>> Adding current user to ollama group...
>>> Creating ollama systemd service...
>>> Enabling and starting ollama service...
Created symlink /etc/systemd/system/default.target.wants/ollama.service → /etc/systemd/system/ollama.service
>>> The Ollama API is now available at 127.0.0.1:11434.
>>> Install complete. Run "ollama" from the command line.
WARNING: No NVIDIA/AMD GPU detected. Ollama will run in CPU-only mode.
ubuntu@thesis:~$
```

FIGURE 11. Installation of the Ollama API using the provided installation script.

Ollama, used as the local generative AI backend, is installed via an automated installation script:

```
curl -fsSL https://ollama.com/install.sh | sh
```

This script automates installation, creating necessary system users and enabling services required to integrate Ollama seamlessly into the Kubernetes diagnostic workflow.

This installation automatically configures the Ollama API, creates the necessary system groups and users, and sets up Ollama as a systemd-managed service. The output indicates the successful installation of Ollama, highlighting that it runs in CPU-only mode due to the absence of an NVIDIA/AMD GPU, which is suitable for this experimental context.

Once configured, K8sGPT leverages the Ollama backend to provide advanced, generative AI-driven diagnostics and recommendations, enabling quicker resolution of Kubernetes issues such as misconfigured image tags and resource allocation errors.

4.6 OpenAI's Access Key Creation and Authentication

FIGURE 12 illustrates the creation and management of an OpenAI access key. The access key is critical for authentication, enabling the use of OpenAI's models through the K8sGPT tool.

To authenticate and integrate OpenAI's model with K8sGPT, an API key must be generated from the OpenAI account dashboard. The provided API key, as shown in FIGURE 12 is utilized to authenticate the K8sGPT tool, enabling it to access OpenAI's generative AI services. Authentication is completed by executing the following command:

```
k8sgpt auth add --backend openai --model gpt-4o-mini
```

This command prompts the entry of the generated API key, establishing secure access and allowing the K8sGPT tool to effectively perform advanced diagnostics and provide insightful analysis within Kubernetes clusters.

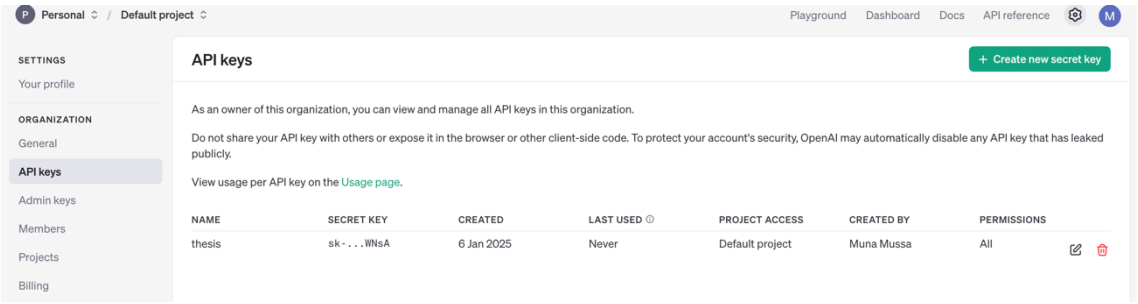

FIGURE 12. OpenAI's access key.

FIGURE 13 Output of the `k8sgpt auth list` command showing the available and active authentication options for `k8sgpt`.

The default and active authentication provider are openai, while several unused options are listed, including localai, ollama, azureopenai, cohere, amazonbedrock, amazonsagemaker, google, noopai, huggingface, googlevertexai, oci, and ibmwatsonxai. This output highlights the flexibility of `k8sgpt` in integrating with various AI and ML platforms for Kubernetes cluster analysis and operations.

```
ubuntu@thesis:~$ k8sgpt auth list
Default:
> openai
Active:
Unused:
> openai
> localai
> ollama
> azureopenai
> cohere
> amazonbedrock
> amazonsagemaker
> google
> noopai
> huggingface
> googlevertexai
> oci
> ibmwatsonxai
ubuntu@thesis:~$
```

FIGURE 13.Output of the `k8sgpt auth list` command.

4.7 Testing broken pod

This section presents the simulation of typical Kubernetes deployment errors, demonstrating the practical utility of the implemented diagnostic framework. A deliberately faulty pod configuration is introduced to evaluate the capability of the K8sGPT tool integrated with local AI services in identifying, diagnosing, and resolving deployment issues.

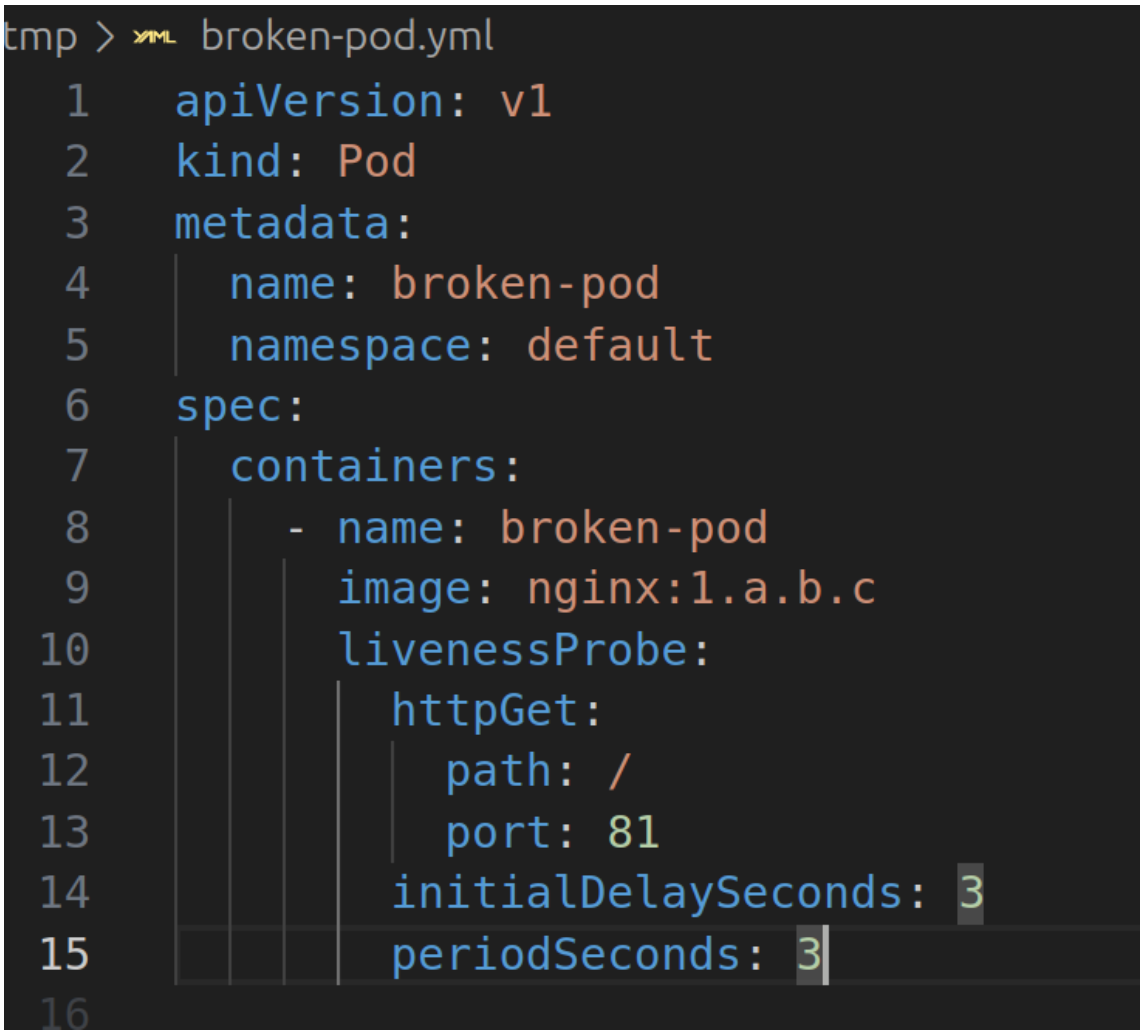

FIGURE 14. Sample YAML configuration pod file.

FIGURE 14 shows the YAML configuration file (`broken-pod.yml`) defining a Kubernetes pod named `broken-pod` in the `default` namespace.

The pod contains a single container named `broken-pod`, which attempts to use the image `nginx:1.a.b.c`. The configuration includes a `livenessProbe` using an `HTTP GET` request on port 81 at the root path (/). The probe has an initial delay of 3 seconds and checks every 3 seconds thereafter (`periodSeconds`). This file illustrates a pod with misconfigured settings, such as an invalid image tag (1.a.b.c) and an incorrect port for the liveness probe, which would result in a failure during deployment.

```
ubuntu@thesis:~$ kubectl apply -f broken-pod.yml
pod/broken-pod created
ubuntu@thesis:~$ kubectl get pods
NAME         READY   STATUS             RESTARTS   AGE
broken-pod   0/1     ImagePullBackOff   0          16s
ubuntu@thesis:~$ kubectl get pods
NAME         READY   STATUS             RESTARTS   AGE
broken-pod   0/1     ImagePullBackOff   0          24s
ubuntu@thesis:~$ kubectl get pods
NAME         READY   STATUS             RESTARTS   AGE
broken-pod   0/1     ImagePullBackOff   0          26s
ubuntu@thesis:~$ kubectl get pods
NAME         READY   STATUS         RESTARTS   AGE
broken-pod   0/1     ErrImagePull   0          28s
```

FIGURE 15. Deploying pod definition file in Kubernetes cluster.

FIGURE 15 illustrates deploying the pod configuration to the Kubernetes cluster using the command:

`kubectl apply -f broken-pod.yml`

The subsequent pod status checks via `kubectl get pods` reveal an `ErrImagePull` error, indicating failure due to the incorrect image tag.

4.7.1 Analyzing a Kubernetes pod using Openai

Analysis of a Kubernetes pod using the `k8sgpt analyze` command. FIGURE 16 demonstrates the analysis executed by the `k8sgpt` tool integrated with the `openai` AI backend.

FIGURE 16. Analysis of a Kubernetes pod using the `k8sgpt analyze` command.

The first command output, without the `--explain` option and AI integration, identifies an error where the pod default/broken-pod fails to pull the Docker image `nginx:1.a.b.c`, resulting in a `Back-off pulling image` error. The second command, with the `--explain` option and OpenAI as the AI provider, provides a detailed explanation of the error, indicating that the specified image tag is invalid and cannot be found in the Docker repository.

Additionally, the AI-generated solution suggests checking the image tag for any typos and verifying that the image exists on Docker Hub. It recommends using a valid tag, such as `nginx:latest` or `nginx:1.21`, to ensure compatibility. Furthermore, updating the Kubernetes deployment with the correct image tag is advised to resolve the issue. This output demonstrates how AI integration enhances error analysis and debugging in Kubernetes environments by providing precise, actionable solutions. The `k8sgpt analysis` processed one issue in total, achieving a processing speed of 27 iterations per minute. This metric provides insight into the efficiency and performance of the OpenAI-backed diagnostic analysis. The next step is to correct the issues using the suggested list of corrective measures. First edit the failed YAML and deploy to Kubernetes environment.

```
ubuntu@thesis:~$ nano broken-pod.yml
ubuntu@thesis:~$ kubectl apply -f broken-pod.yml
pod/broken-pod configured
ubuntu@thesis:~$ kubectl get pods
NAME         READY   STATUS    RESTARTS   AGE
broken-pod   1/1     Running   0          42m
ubuntu@thesis:~$
```

FIGURE 17. Status of the pod configuration.

FIGURE 17 depicts the corrected YAML configuration and subsequent successful deployment and verification. After updating the image tag and correcting the misconfiguration, the pod successfully transitions to a running state. This outcome confirms the effectiveness of the AI-enhanced diagnostic approach in promptly identifying and resolving Kubernetes deployment issues. The `nano broken-pod.yml` command is used to edit the pod configuration file, and the `kubectl apply -f broken-pod.yml` command deploys the configuration, resulting in the message pod/broken-pod configured. The `kubectl get pods` command confirms that the pod named `broken-pod` is successfully running with a status of Running, 1 container ready, 0 restarts, and an age of 42 minutes.

4.7.2 Analyzing a Kubernetes pod using Ollama

Now testing analyzing using Ollama that is running locally using `k8sgpt` framework. FIGURE 18 illustrates the analysis of a Kubernetes pod issue using `k8sgpt` with a local AI provider Ollama.

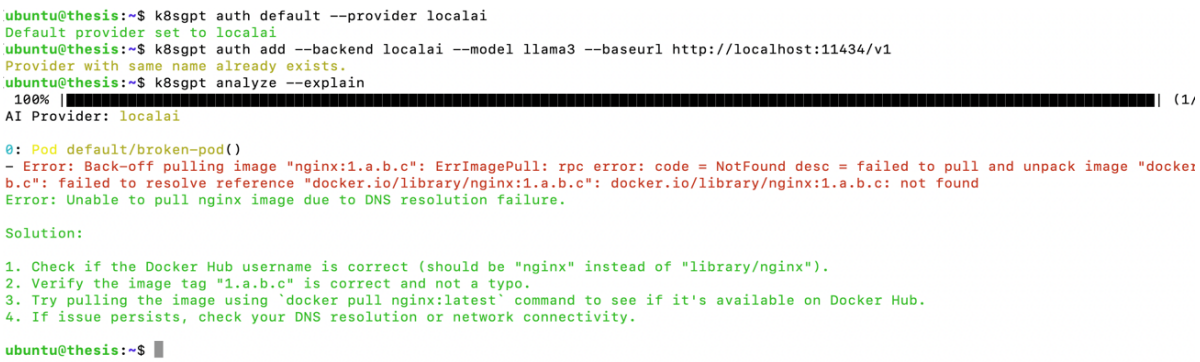

FIGURE 18. Analysis of a Kubernetes pod issue using k8sgpt with a local AI provider.

The command `k8sgpt auth default --provider localai` sets the default AI provider to `localai`, configured to use the llama3 model with the backend at http://localhost:11434/v1. The `k8sgpt analyze --explain` command is executed to identify and resolve an error in the `default/broken-pod`.

The analysis indicates that the image `nginx:1.a.b.c` cannot be pulled due to a DNS resolution failure, resulting in an `ErrImagePull` error. The AI provider suggests verifying that the Docker Hub username is correct, ensuring the image tag `1.a.b.c` is accurate and free of typos, and attempting to manually pull the image using the command `docker pull nginx:latest`. If the issue persists, DNS resolution or network connectivity should be checked. This figure illustrates the integration of a local AI backend, which provides insightful analysis and practical solutions for Kubernetes pod errors.

As shown in FIGURE 18, the notation (1/1, 42 it/min) indicates that the `k8sgpt` analysis processed one issue in total, achieving a processing speed of 42 iterations per minute. This metric provides insight into the efficiency and performance of using the Ollama-backed diagnostic analysis. The results of these tests are discussed further in the next chapter.

5 RESULTS

The results of this experiment highlight the differences between using OpenAI and Ollama as AI backends in the K8sGPT framework for diagnosing Kubernetes issues. The analysis focused on their performance, accuracy, and practical use in real-world environments.

The performance test showed that Ollama completed Kubernetes diagnostics at rate of 42 iterations per minute, while OpenAI processed them at 27 iterations per minute. The faster execution of Ollama is because it runs locally, eliminating the need for network requests. However, this advantage depends on the available system resources. If the local machine does not have enough CPU and memory, performance can drop significantly. In contrast, OpenAI, which operates in the cloud, remains stable and does not depend on local hardware.

One important factor in these results is that Ollama was tested on a CPU-based system with 8 vCPUs and 32 GB RAM. If Ollama had been run on a GPU-powered system, the performance might have been different, possibly improving its processing speed further. Additionally, if Ollama had access to a larger dataset for training, its recommendations could also improve over time, making it more comparable to OpenAI in accuracy.

One of the most important differences between the two models is the accuracy of their recommendations. OpenAI provided precise and immediately usable solutions. In many cases, it suggested direct replacement commands that a Kubernetes administrator could apply without additional troubleshooting. Ollama also produced helpful recommendations, but they were sometimes less specific and required further investigation before being used. This is likely because OpenAI has been trained on a larger dataset, including real-world Kubernetes troubleshooting cases. It also benefits from continuous updates and improvements in cloud-based machine learning. Ollama, on the other hand, runs locally and does not update frequently, meaning its recommendations might not be as refined. This requires Kubernetes administrators to verify its suggestions before applying them.

Another key observation is that both AI models use caching. If the same diagnostic command is run again, both OpenAI and Ollama retrieve the results from cache instead of performing a new analysis. This improves efficiency by reducing redundant processing and response time.

There is also a cost difference between the two AI models. OpenAI requires an API access key, and using OpenAI's service is not free. The cost depends on the number of API requests made, which could be a concern for organizations running frequent Kubernetes diagnostics. Ollama, on the other hand, runs entirely locally and is free to use after installation. This makes Ollama a more cost-effective solution for long-term Kubernetes troubleshooting, especially for companies that prefer self-hosted AI models.

From a resource perspective, OpenAI operates entirely in the cloud, requiring an API access key to function, while Ollama runs locally and consumes physical system resources. OpenAI scales easily because it does not depend on local hardware, whereas Ollama's performance is limited by the available processing power of the machine it runs on. If the hardware is not powerful enough, its performance can decrease.

The best choice depends on the use case. OpenAI is more effective for organizations that need fast, accurate, and scalable diagnostics. Ollama is a better choice for companies that require data privacy and offline AI capabilities, even if it requires additional verification. A hybrid approach, where Ollama handles local diagnostics and OpenAI is used for real-time, high-priority issues, could be the best of both worlds.

The final chapter will discuss the conclusion of this thesis and future work.

6 CONCLUSION AND FUTURE WORK

The experimental setup demonstrated that while both OpenAI and Ollama can diagnose Kubernetes issues effectively, both have their individual strengths and limitations. OpenAI provides highly accurate results and often suggests solutions that can be applied immediately, making it an excellent choice for Kubernetes administrators who need quick fixes. Ollama, while faster in execution, requires more manual validation because its recommendations are sometimes less specific. OpenAI benefits from continuous cloud-based updates and training on a large dataset, while Ollama relies on pre-trained models that do not update frequently. This makes OpenAI the better option for troubleshooting common Kubernetes issues with minimal human intervention. On the other hand, Ollama is better suited for experienced administrators who can refine the AI's suggestions before implementing them.

Both OpenAI and Ollama use caching, meaning that if a diagnostic command is run multiple times, the AI will return the previous results instead of reprocessing them. This helps speed up repeated queries and reduces computation time, but it also means users must clear the cache or re-run diagnostics when the cluster state changes.

One additional difference is cost. OpenAI requires a paid API subscription, which might be a limiting factor for organizations that need continuous AI-powered diagnostics. Ollama, in contrast, runs locally and does not require any payment, making it a cost-effective solution for Kubernetes troubleshooting, especially in security-sensitive environments. However, because Ollama does not benefit from continuous updates like OpenAI, it may not always provide the most up-to-date recommendations.

Choosing between OpenAI and Ollama depends on the needs of the organization. OpenAI is ideal for real-time, highly accurate diagnostics, especially for teams that require immediate solutions. Ollama is a better fit for environments where data privacy and offline functionality are the main priorities. If Ollama were to be run on a GPU-powered system, its speed might improve further, making it

an even stronger competitor. Similarly, if it had access to a large, continuously updated dataset, its diagnostic accuracy could reach a level closer to OpenAI's. For companies that need a balance of speed, accuracy, and security, a combination of both models may be the best approach.

A hybrid approach could also be explored as a future work, where OpenAI handles fast, real-time diagnostics while Ollama is used for offline, privacy-focused troubleshooting. This would allow organizations to balance speed, accuracy, and security.

Further research could test Ollama's performance on GPU-powered hardware. If Ollama runs on a GPU instead of a CPU, it may increase its speed and compete more closely with OpenAI.

AI-driven Kubernetes diagnostics could also expand beyond troubleshooting. Future improvements could explore predicting issues before failures happen, automating security enforcement, and optimizing workloads based on real-time performance metrics.

As AI tools improve, their role in managing and troubleshooting Kubernetes will continue to grow, making clusters more efficient and reducing downtime.

REFERENCES

1. The Kubernetes Authors s.a. Overview. Kubernetes Documentation. Available at: https://kubernetes.io/docs/concepts/overview/. Accessed: 27.3.2025.
2. Google Cloud s.a. What is Kubernetes? Google Cloud. Available at: https://cloud.google.com/learn/what-is-kubernetes. Accessed: 27.3.2025.
3. OpenAI. Generative AI and Its Applications. Available at: https://openai.com/research/. Accessed: 27.3.2025.
4. K8sGPT Documentation, https://docs.k8sgpt.ai/ . Accessed: 27.3.2025.
5. Merkel, D. "Docker: Lightweight Linux Containers for Consistent Development and Deployment." *Linux Journal*, 2014. Available at: https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment. Accessed: 27.3.2025.
6. Burns, B., Grant, B., Oppenheimer, D., Brewer, E., & Wilkes, J. "Borg, Omega, and Kubernetes." *ACM Queue*, 2016. Available at: https://queue.acm.org/detail.cfm?id=2898444 . Accessed: 27.3.2025.
7. Mell, P., & Grance, T. "The NIST Definition of Cloud Computing." National Institute of Standards and Technology, 2011.
8. Amazon Web Services. What Is Kubernetes? Available at: https://aws.amazon.com/. Accessed: 27.3.2025.
9. Docker Inc. "What is a Container?" *Docker Documentation*. Available at: https://docs.docker.com/get-started/overview/. Accessed: 27.3.2025.
10. The Kubernetes Authors. "Why Kubernetes?" *Kubernetes Documentation*. Available at: https://kubernetes.io/docs/concepts/overview/why-kubernetes/ . Accessed: 27.3.2025.
11. Mell, P., & Grance, T. "The NIST Definition of Cloud Computing." *National Institute of Standards and Technology*, 2011. Available at: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf. Accessed: 27.3.2025.
12. Fowler, M., & Lewis, J. "Microservices." *martinfowler.com*, 2014. Available at: https://martinfowler.com/articles/microservices.html. Accessed: 27.3.2025.

13. IBM Cloud Education. "What is application portability?" IBM Cloud Learn Hub. Available at: https://www.ibm.com/cloud/learn/application-portability. Accessed: 27.3.2025.
14. Red Hat. "How to scale applications in Kubernetes." *Red Hat Developer*. Available at: https://developers.redhat.com/articles/2023/01/12/how-autoscale-saas-application-infrastructure#kubernetes_native_infrastructure. Accessed: 27.3.2025.
15. Gartner. "The Role of Containers in Cloud-Native Application Development." Available at: https://gartner.com. Accessed: 27.3.2025.
16. **CNCF.** "Cloud Native Landscape." Available at: https://landscape.cncf.io/. Accessed: 27.3.2025.
17. CNCF. "Cloud Native Artificial Intelligence." Available at: https://www.cncf.io/wp-content/uploads/2024/03/cloud_native_ai24_031424a-2.pdf. Accessed: 27.3.2025.
18. The Kubernetes Project. Kubernetes Documentation. Available at: https://kubernetes.io. Accessed: 27.3.2025.
19. CNCF. "CNCF Annual Reports." Available at: https://www.cncf.io/reports/ Accessed: 27.3.2025.
20. ZippyOps, 2024. Container vs. VM Security: Which Is Better? Available at: https://www.zippyops.com/container-vs-vm-security-which-is-better . Accessed: 27.3.2025.
21. Ollama. "Ollama AI." Available at: https://ollama.ai. Accessed: 27.3.2025.
22. The Kubernetes Authors s.a. Architecture. Kubernetes Documentation. Available at: https://kubernetes.io/docs/concepts/architecture/ Accessed: 27.3.2025.
23. GDPR Compliance Resources. " General Data Protection Regulation." Available at: https://gdpr-info.eu/. Accessed: 27.3.2025.
24. Luksa, M., 2017. Kubernetes in action. Simon and Schuster.
25. Burns, B., Beda, J., Hightower, K. and Evenson, L., 2022. Kubernetes: up and running: dive into the future of infrastructure. " O'Reilly Media, Inc.".
26. Poulton, N., 2025. The kubernetes book. NIGEL POULTON LTD.
27. Roland, H., Daniele, Z., 2025. Generative AI on Kubernetes " O'Reilly Media, Inc."