Artemii Medianov

Deploying FIWARE Context Broker automatically in Kubernetes environment

Bachelor's thesis

Bachelor of Engineering

Information Technology

2025

South-Eastern Finland
University of Applied Sciences

South-Eastern Finland University of Applied Sciences

Degree title	Bachelor of Engineering
Author	Artemii Medianov
Thesis title	Deploying FIWARE Context Broker automatically in Kubernetes environment
Commissioned by	DAME Project
Year	2025
Pages	52 pages
Supervisor	Ville Kauppi

ABSTRACT

The objective of this thesis was to design and implement a method for automating the deployment process of the FIWARE Context Broker using modern practices and Kubernetes-native tools. The study also aimed to address challenges related to configuration management, resource optimization and integration with other FIWARE components.

The FIWARE Context Broker is a core component of the FIWARE platform, which assists in the management of context information for smart applications. Efficient deployment of this component in containerized environments such as Kubernetes is essential for scalability, reliability and ease of use.

As a research-based development study, the approach involved designing, implementing and evaluating an automated deployment solution. The deployment process was automated using tools such as Helm and Kubernetes operators. The methodology included designing deployment templates, testing them in a controlled environment and evaluating the results based on predefined metrics.

The study demonstrates that automating the deployment of the FIWARE Context Broker significantly reduces the complexity and time required for setup in Kubernetes environments. The use of Helm charts and custom Kubernetes operators enables seamless integration and efficient resource management. These findings offer valuable insights for developers and organizations aiming to utilize FIWARE in their IoT and smart projects.

Keywords: Kubernetes, Fiware, Context Broker, automation, smart applications, Helm

CONTENTS

1 INTRODUCTION

The increasing popularity of smart applications and IoT solutions has led to a growing demand for efficient data management and interoperability between various IT systems. The Context Brokers play a crucial role in handling real-time data exchanges by enabling a smooth and coherent integration between IoT devices and applications. However, deploying and managing Context Brokers presents challenges related to scalability, automation and resource efficiency.

Kubernetes, a leading container orchestration platform, provides a robust solution for automating deployments and managing containerized applications. By utilizing Kubernetes-native tools such as Helm, this thesis explores a method for automating the deployment of the FIWARE Context Broker, reducing manual effort and enhancing system resilience.

This study was conducted as part of the DAME Project within the framework of Memory Lab.

1.1 DAME Project

The South Savo Data Economy Accelerator: Shared Data as a Joint Success Factor, known as the “DAME project”, is an initiative by the South-Eastern Finland University of Applied Sciences (Xamk) aimed at advancing the data economy in the South Savo region of Finland.

The project runs from 1 June 2024 to 31 December 2025 and aims to advance the data economy in South Savo. It involves conducting a situational analysis through surveys and research and implementing sector-specific trials in areas such as food, well-being, cultural heritage, circular economy, technology and society to develop data spaces and test data-sharing technologies. Additionally, the project focuses on designing support services to help businesses and organizations improve data management, sharing and utilization. It also seeks to establish a regional data economy value network to drive data-driven business

opportunities. The project is co-funded by the EU and was granted by the Centre for Economic Development, Transport and the Environment. (XAMK 2024).

By promoting data sharing and collaboration, the DAME Project aims to renew and diversify the regional economy, support small and medium-sized enterprises and integrate South Savo into the broader European data economy.

1.2 Memory Lab

Memory Lab is a Finnish company that explores the potential of artificial intelligence in collaboration with partners to refine data and produce commercial services for emerging industries. It is developed by Digitalia, a joint research center of Xamk, University of Helsinki and the National Archives of Finland. This collaboration focuses on advancing digital information management and preservation, contributing to the development of Memory Lab's services and research initiatives. Specialists and companies in digital information management are brought together by Memory Lab and they together aim to develop RDI activities. Memory Lab serves memory organizations, companies and other players in the field as a new RDI environment and centre of expertise. Additionally, Memory Lab is associated with the Open Memory Lab FIWARE iHub which brings together businesses, educational institutions, research facilities and the public sector. (XAMK Memory Lab 2023).

With these collaborations, Memory Lab integrates academic research, practical applications and technological advancements to support the digital preservation needs of various organizations.

Their services include:

- Digitization of analog media. Memory Lab converts various analog media formats, such as VHS tapes, audio cassettes and photographs, into digital formats to preserve accessibility.
- Restoration services. Memory Lab offers restoration services to improve the quality of aging media, including color correction, noise reduction and repair of damaged footage or images.

- Custom archiving solutions. Memory Lab provides enhanced archiving solutions for private customers and organizations to help organize digital media and store it according to clients' specific needs. (XAMK Memory Lab 2023a).

As for hardware, Memory Lab utilizes the Hippu supercomputer, which features 8 A100 graphics cards, 640 GB of GPU memory, 2 TB of regular memory and a computational power of 5 petaflops. (XAMK Memory Lab 2023b).

With such a combination of services and hardware, Memory Lab helps clients in preserving their analog media collections, ensuring longevity and accessibility in digital formats.

1.3 Objectives and scope

The primary objective of this thesis is to design and implement an automated deployment solution for the FIWARE Context Broker in a Kubernetes environment. Utilizing the scalability, flexibility and orchestration capabilities of Kubernetes, this study aims to automate the deployment process, minimize manual intervention and ensure high availability of the Context Broker.

This thesis will explore:

1. the integration of Kubernetes tools (Helm charts, operators and custom scripts) for deploying the FIWARE Context Broker,
2. automation strategies to enhance efficiency in deployment and scaling,
3. best practices for configuring and managing resources in a cloud-native environment.

The scope of this thesis is limited to the deployment and orchestration of the FIWARE Context Broker within Kubernetes, focusing on automation processes. Other FIWARE components or non-Kubernetes-based solutions are excluded, unless relevant to demonstrating compatibility or functionality.

2 FIWARE CONTEXT BROKER

FIWARE is an open-source platform that helps developers create smart applications for cities, industry, agriculture and energy. It provides tools to collect and share data from different sources, such as IoT devices and cloud services. FIWARE has the Orion Context Broker, which handles real-time data and makes it accessible to different systems. It is designed to support both small and large-scale projects and includes security features to control user access and protect data. FIWARE is used in many areas, such as traffic control and waste management in smart cities, automation and monitoring in industry, smart farming and irrigation in agriculture and renewable energy tracking in the energy sector. It is widely adopted by businesses, governments and researchers to build smarter and more efficient systems. (FIWARE n.d.).

The FIWARE Context Broker is a software component that supervises real-time data management and enables applications to receive, store and update data modifications. Effectively, it serves as an API that integrates data from diverse systems, creating a coherent view and functioning as a central node for context-aware applications while managing information related to various real-world entities, such as sensors, devices, vehicles and buildings. Each entity is characterized by attributes that represent its current state, so applications can engage with this data through standardized APIs. The broker adheres to the NGSI standard which simplifies a systematic approach to context information management via HTTP requests. (Zangelin 2013).

An important feature of the Context Broker is its ability to manage subscriptions and notifications. It helps applications to receive automated alerts when specific data changes, eliminating the need for continuous polling for updates. This functionality is especially beneficial for real-time applications in areas such as industrial automation or IoT platforms.

The Orion Context Broker, which is the most prevalent implementation, was developed within the FIWARE ecosystem. It can be seamlessly integrated with other components to facilitate historical data storage, analytics and interaction

with IoT devices. Designed with scalability in mind, the broker is well-suited for extensive deployments where multiple systems must collaboratively share and respond to dynamic information in an organized and efficient manner. With these integration needs, the deployment of the FIWARE Context Broker efficiently requires an orchestration system capable of managing containerized applications at scale, which Kubernetes provides through its comprehensive framework. (Dianese 2020).

Figure 1 illustrates an architecture of the practical implementation of the FIWARE Orion Context Broker. At the bottom there are several IoT devices: smart meters, micro-grid management systems, EV diagnostic devices, metering stations, EV chargers and fleet monitoring systems. They all collect real-time energy data which is centralized in the FIWARE Orion Context Broker. It acts as the core data exchange hub that ensures interoperability between devices and applications. The broker interacts with MongoDB data base for data storage and FIWARE Cygnus which processes and forwards data to repositories (Cassandra DB and Distributed Ledger). This setup enables real-time monitoring, decision-making and optimization of energy resources and supports decentralized energy markets and smart grid applications. (Dianese 2020).

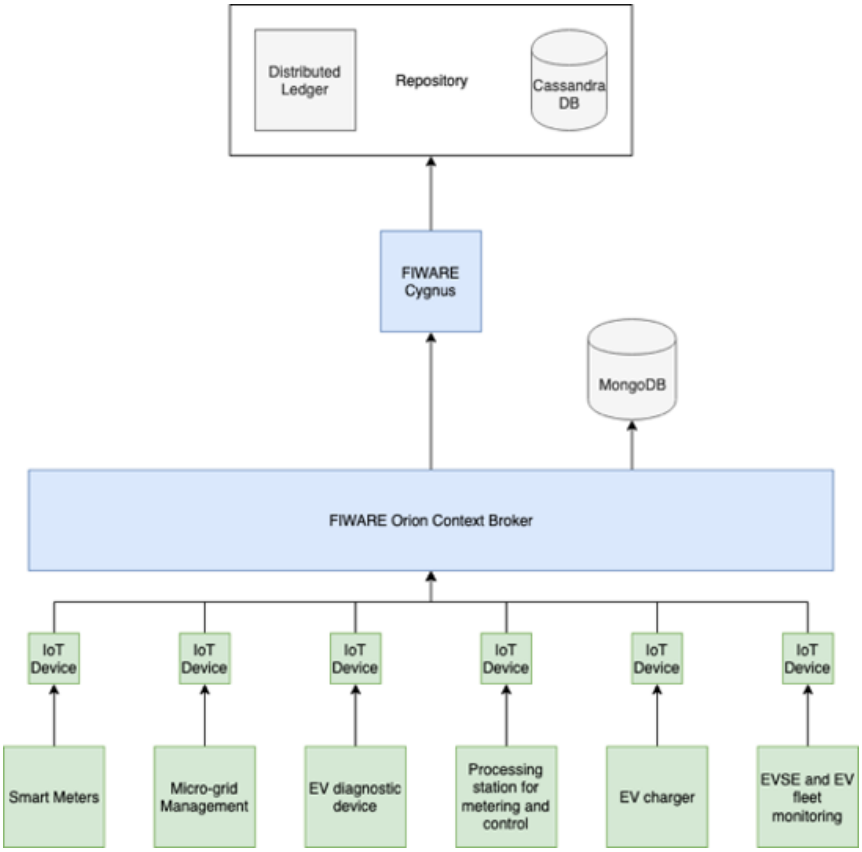

Figure 1. Architecture of the FIWARE Context Broker in an energy system (FIWARE 2020)

3 KUBERNETES

Kubernetes is an open-source portable platform for orchestrating the containers, which automates the deployment, scaling and management of containerized applications across various hosts. It was originally created by Google, but now is controlled and maintained by the Cloud Native Computing Foundation (CNCF). (Susnjara, Smalley 2024).

Fundamentally, Kubernetes provides a foundation for operating distributed systems in a highly adaptable manner. It abstracts away the challenges of managing infrastructure, allowing developers to focus primarily on crafting application logic.

With Kubernetes, a whole cluster of containerized applications can be managed, instead of maintaining a separate virtual or physical machine with deployed containers. In other words, Kubernetes ensures maximum compatibility of all

projects. The main advantage of using it is an improved model for scaling and launching containers, which greatly simplifies management and monitoring. Kubernetes is the leading container orchestration standard due to its flexibility and scalability. (Prakarsh 2019). A modern application may consist of dozens or hundreds of containerized microservices that must communicate smoothly. They are hosted on multiple host machines called “nodes”, which can be connected into a single cluster. Considering the interaction scheme between containers and nodes, it becomes obvious that a number of special tools - orchestration platforms - are needed to coordinate such a distributed system. Kubernetes does not create containers, but relies on a ready-made way to implement them, such as Docker. Kubernetes demonstrates well in operations with container clusters, thereby solving the problems of deployment automation, networking, scaling and monitoring. Due to this, Kubernetes can be considered a good choice for orchestrating large distributed applications with many connected microservices, including databases, secrets and external dependencies. (Microsoft n.d.)

Figure 2 illustrates one of the potential use cases of Kubernetes in a CI/CD pipeline. The process starts when a developer pushes code to a GitHub repository. This repository is accessible by the Jenkins server, which allows developers to build, test and deploy their software. Jenkins then triggers the deployment to the Kubernetes cluster, which later deploys a container using the Docker service. Ultimately, the containerized application is delivered to users computers.

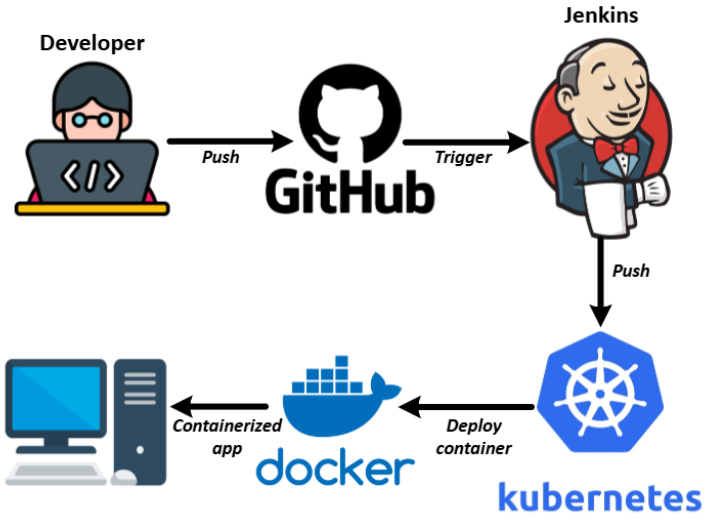

Figure 2. Kubernetes use cases (Aleksic 2022)

Kubernetes is widely used across industries for various purposes:

- It facilitates deploying, managing and scaling multiple interconnected services, commonly seen in environments like e-commerce platforms and financial systems.
- It automates deployment pipelines and integrates with tools like Jenkins and ArgoCD to speed up software delivery cycles in DevOps and CI/CD pipelines.
- It enables cloud-native applications to be deployed on platforms such as AWS, Azure, Google Cloud or on-premises private clouds, supporting hybrid and multi-cloud strategies.
- It helps manage scalable computing resources for big data analytics and machine learning pipelines, frequently used with TensorFlow and Apache Spark frameworks.
- It powers high-traffic web applications and ensures high availability and load balancing, that is commonly used by SaaS companies and web service providers.
- It extends orchestration capabilities to edge devices in IoT and telecommunications use cases for edge computing.
- In game development, it dynamically scales game servers based on player activity. (Kubernetes 2024a).

Kubernetes is utilized by many companies, including Netflix, Spotify and Airbnb. Kubernetes has emerged as the standard solution for container orchestration in industry, enabling modern application development and infrastructure management. (Tettamanti 2025).

3.1 Containers

Containerization is a method by which program code is packaged into a single executable file along with libraries and dependencies to ensure that it runs correctly. Such files are called containers. Containers can be managed and deployed in different environments. An individual container does not depend on the settings of the underlying operating system and can run on any platform or in the cloud. Unlike traditional virtual machines, which virtualize hardware, containers operate at the operating system level rather than the hardware level, making them more efficient in terms of resource utilization and performance. (The Cloud Native Experts 2024).

Containers rely on the host operating system's kernel to function, which allows them to share resources, for example CPU, memory and storage, without the overhead of running a full OS for each instance. This architecture makes containers both lightweight and fast, ideal for modern application development and deployment.

Containers have several mechanisms that help them become lightweight. As was said before, they use the host operating system's kernel, isolating processes using namespaces. Namespaces ensure that each container operates in its own private environment, separated from other containers and the host system. Additionally, control groups regulate the amount of system resources a container can use, ensuring fair allocation and preventing one container from monopolizing resources.

Containers use a union file system, which allows them to create images in layers. These layered file systems enable efficient storage and updates, as containers can share base layers while customizing only the application-specific

components. A container image serves as a blueprint, containing all functionalities needed to run the application, from the operating system libraries to the application code itself. (Sayfan 2018).

When a container starts from an image, it becomes a runtime instance of that image, running as an isolated process on the host system. This makes containers highly portable and easy to reproduce across environments.

Figure 3 illustrates a design of a common application container. It consists of an application itself, different dependencies required for running this application and an information about hardware requirements, such as CPU model, network status, RAM memory and disk space availability.

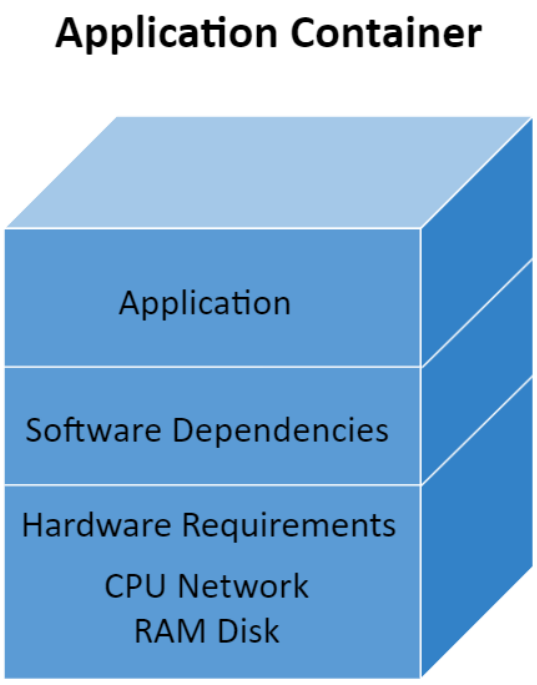

Figure 3. Application container design

Containers are widely used for application deployment, because they offer a consistent environment that eliminates issues related to differences in development and production setups. They are very important in the microservices architecture where each service of an application is packaged into its own container. This modular approach simplifies scaling, updates and maintenance.

In DevOps practices, containers enable rapid build–test–deploy cycles. They also facilitate resource optimization by allowing multiple containers to run on a single host without the overhead associated with VMs. (Hohn 2022).

Containers are an efficient solution for achieving horizontal scalability, as they can be rapidly duplicated to meet rising demand. Tools such as Kubernetes streamline the process of container deployment and scaling, providing high performance and reliability even during variable conditions such as traffic surges or version rollbacks. Within a Kubernetes environment, containers are organized into pods, which are the smallest units that can be deployed. Kubernetes oversees the entire lifecycle of these pods, handling tasks such as rolling updates, load distribution and failure recovery. This architecture makes containers particularly suitable for deploying modern applications such as the FIWARE Context Broker.

3.2 Kubernetes architecture

Kubernetes architecture is designed to manage and orchestrate containers efficiently across a cluster of machines. It is a distributed system with a client-server model, consisting of two primary components: the Control Plane and the node (or Worker node). These components cooperate to maintain the desired state of applications and manage the lifecycle of containers. (Walker 2025).

3.2.1 Control Plane

The Control Plane is the central management component of the Kubernetes cluster overseeing the entire system. It makes global decisions about the cluster, such as scheduling applications, maintaining the desired state and responding to failures. (Khisty 2024).

Figure 4 illustrates the interaction between the Control Plane, the administrator and the Worker nodes. Administrative input is handled via the API Server which acts as the central interface for managing the cluster. The Scheduler allocates pods to appropriate Worker nodes based on current resource conditions, while

the Controller Manager maintains the cluster's actual state in line with its intended configuration. Cluster state and metadata are stored in the etcd key-value database, which ensures high availability and data consistency. The Control Plane actively supervises Worker nodes, coordinates with kubelets and manages network traffic using kube-proxy. Additionally, it oversees application scaling and facilitates automated recovery in case of failures.

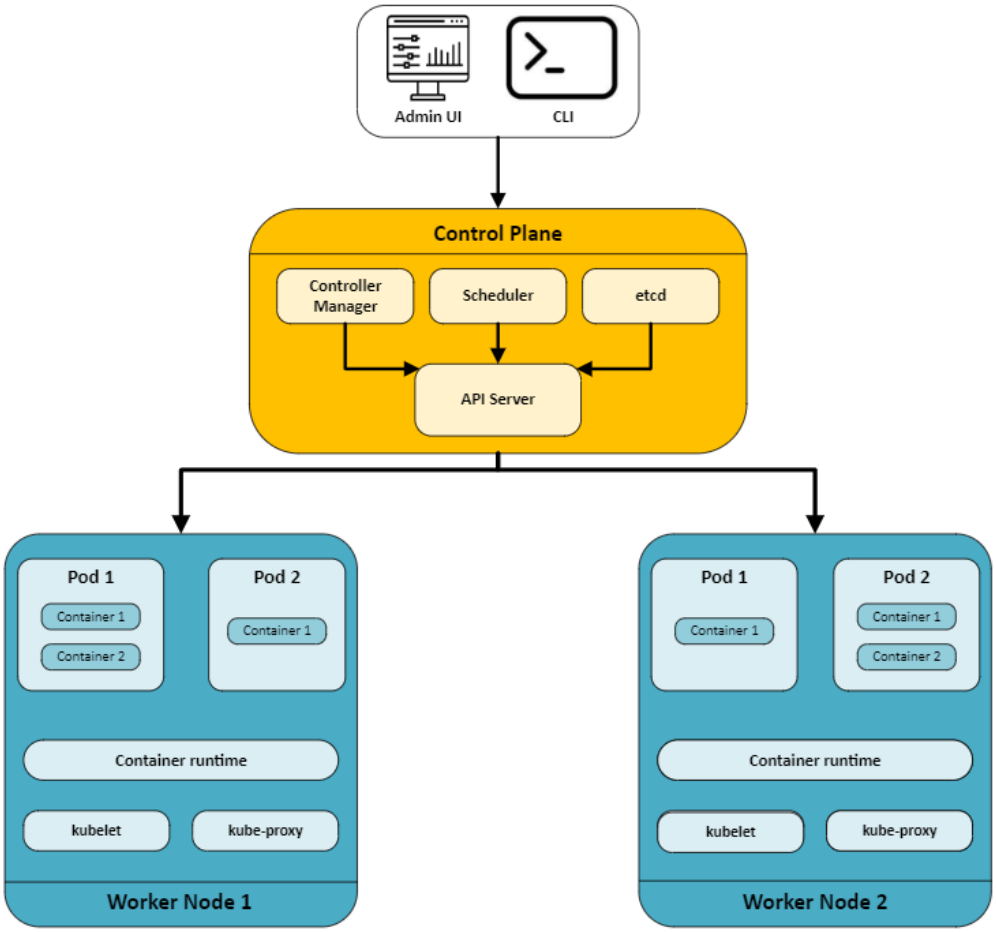

Figure 4. Kubernetes architecture scheme

The key components of the Control Plane are:

- API Server
- Controller Manager
- Scheduler
- etcd

API Server functions as the entry point to the Kubernetes Control Plane and exposes the Kubernetes API for external clients to interact with the cluster. It acts

as a central hub facilitating communication with other core components and maintaining an up-to-date view of the cluster state.

Controller Manager operates various controllers, each responsible for managing a specific part of the cluster state. These controllers observe the current conditions and make necessary changes to ensure alignment with the intended configuration, for example, the Deployment Controller maintains the desired number of active pods.

Scheduler handles the tasks of placing newly created pods onto suitable nodes within the cluster. It evaluates factors such as resource availability, constraints and scheduling policies to optimize workload distribution and maintain balance across nodes.

Etcd serves as the cluster's central key-value store which preserves configuration details and the overall system state. It stores all Kubernetes objects, such as pods, services and deployments, and is essential for maintaining accuracy and consistency. Every change to the cluster is recorded in etcd which acts as the authoritative source for the cluster's current state.

3.2.2 Nodes

A node is a machine in the Kubernetes cluster that runs application containers. (Kubernetes 2025a). Each node contains the necessary services to run and manage containers, including the following key Kubernetes components:

- Kubelet
- Kube-Proxy
- Container Runtime

Kubelet is a node-level agent which ensures that containers are running as intended. It communicates with the API Server to receive task instructions and regularly reports the node status. In addition to managing the lifecycle of pods, Kubelet monitors their health and relays updates back to the Control Plane.

Kube-Proxy is responsible for handling network traffic within the cluster. It routes requests to the appropriate pods and implements load balancing to ensure even distribution of service traffic.

Container Runtime is the underlying software which runs the containers themselves. While Docker is a commonly used runtime, Kubernetes also supports alternatives such as containerd and CRI-O.

3.2.3 Pods

A pod is the smallest deployable unit in Kubernetes. Pods can contain one or multiple containers that share the same network and storage resources. Pods are used to run applications, and containers within a pod can communicate with each other using localhost. It makes them ideal for tightly coupled application components. When deploying the FIWARE Context Broker, for example, the application could run in a pod, and additional containers could be added to the same pod if needed (for logging or monitoring purposes). (Kubernetes 2025b).

Figure 5 presents the structure of a Kubernetes cluster, showing the relationship between the Control Plane and nodes. The Control Plane orchestrates the cluster by scheduling workloads, maintaining state and managing communication. Nodes are responsible for executing workloads and hosting pods which are the fundamental units of deployment. Each pod consists of one or multiple containers that run the application processes. This structure enables efficient resource utilization and automated management of applications. (Kubernetes 2024b).

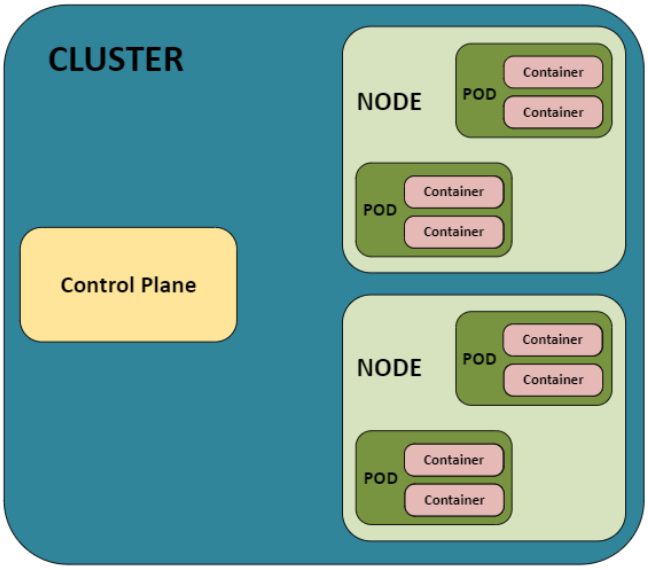

Figure 5. Kubernetes cluster scheme

3.2.4 Services

A service in Kubernetes is an abstraction that defines a set of pods and a policy by which to access them. Services enable reliable communication between pods, even if the pods are dynamically scaled or replaced. (Kubernetes 2025c). Kubernetes provides several types of services:

- ClusterIP (the default type) – exposes the services on an internal IP within the cluster
- NodePort – exposes services on a specific fixed port across the IPs of each node which allows external access
- LoadBalancer – automatically provides a load balancer for the services and exposes them externally
- ExternalName – associates services to external DNS names
- Headless Service – created when ClusterIP is not assigned, allows direct pod-to-pod communication using DNS. (Walker 2024).

Kubernetes resources, such as pods, deployments and services, are described using configuration files, typically written in YAML. These files declare the intended state of the system. Kubernetes continuously monitors the cluster and takes corrective actions to align the actual state with what was defined. For

instance, if a pod fails, Kubernetes will automatically redeploy it on a different node to preserve the specified number of running pods.

Kubernetes architecture is a highly scalable and resilient system designed to manage containerized applications. The Control Plane makes global decisions and maintains the desired state, while Worker nodes execute the containers. (Ferrer 2025). Collectively, these components create a robust platform for automating the deployment, scaling and management of application containers. For a deployment such as the FIWARE Context Broker, Kubernetes' architecture ensures reliability and ease of management.

3.3 Kubernetes deployment models

Kubernetes supports various deployment models that serve diverse application requirements. These deployment models define how workloads are managed, scaled and updated within a Kubernetes cluster. The following sub-sections introduce the key deployment models relevant to this thesis.

3.3.1 Single-cluster deployment

The single-cluster model involves deploying all components of an application within a single Kubernetes cluster. It is ideal for scenarios where resources are centralized, and inter-component communication requires low latency. Single-cluster deployments are easier to manage and are suitable for development and testing environments.

Figure 6 illustrates a scheme of the single-cluster deployment structure. At its core, there is a Master node that controls and manages the entire cluster and specifically all the Worker nodes, which perform the computational tasks, and their processes. Node processes are distributed across these Worker nodes that enables parallel execution and efficient resource utilization within the cluster.

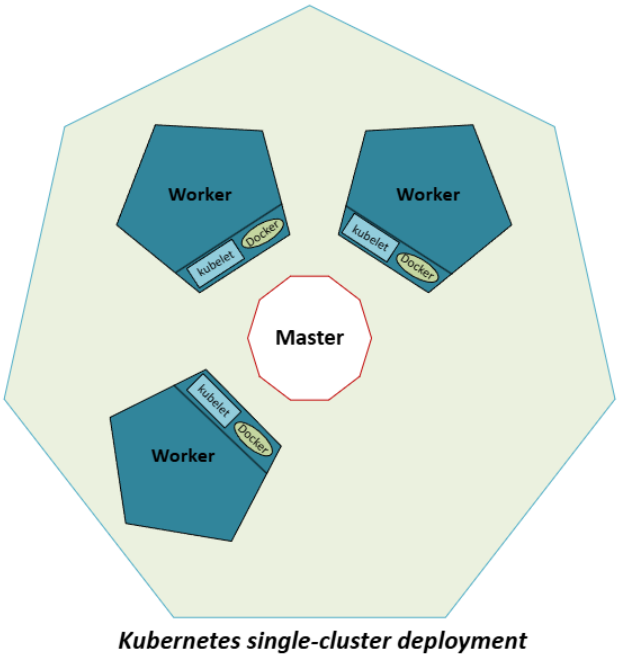

Figure 6. Kubernetes single-cluster deployment scheme

3.3.2 Multi-cluster deployment

In the multi-cluster model, multiple Kubernetes clusters are used, either for redundancy or to distribute workloads across different geographic regions. Multi-cluster deployments offer better fault tolerance, disaster recovery and meeting data localization requirements. (Tigera 2025).

Figure 7 illustrates how multiple independent clusters are managed under a single load balancer for improved scalability, high availability and fault tolerance. Each cluster consists of Master nodes for control, etcd for state management and multiple Worker nodes running workloads across different namespaces. This architecture enables better resource distribution and resilience by balancing traffic across clusters.

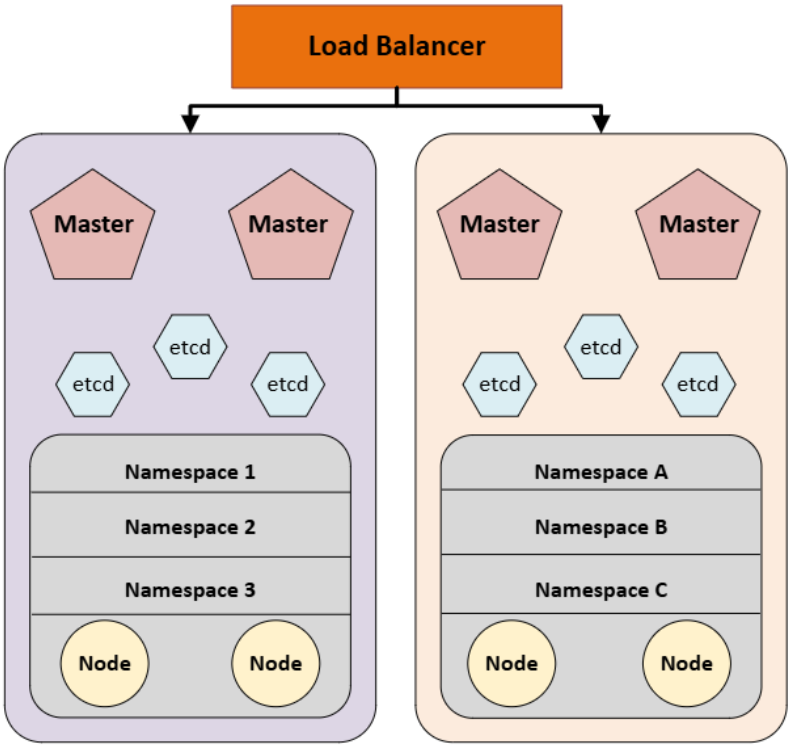

Figure 7. Kubernetes multi-cluster deployment scheme

3.3.3 Hybrid deployment

Hybrid deployments combine on-premises Kubernetes clusters with cloud hosted clusters. This model is often used when there is a need to leverage cloud scalability while retaining critical components for security or compliance reasons. (Ghosh 2023).

Figure 8 illustrates this hybrid architecture, where on-premises self-hosted Kubernetes clusters integrate with cloud-based managed Kubernetes clusters using centralized Azure services. They are managed evenly through Azure Monitor for observability and Azure Policy for governance and compliance. This setup ensures security, monitoring and policy enforcement across both environments.

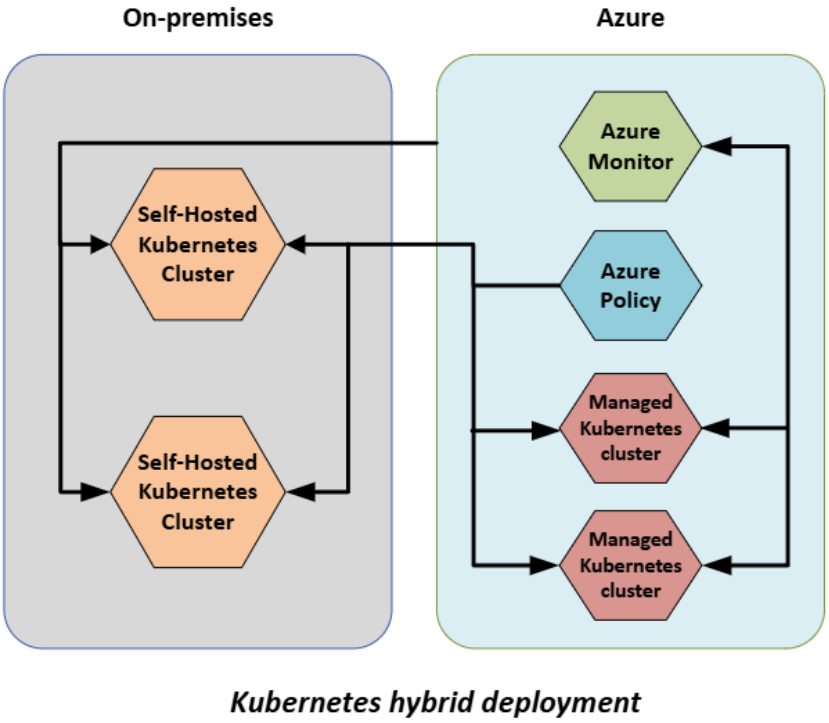

Figure 8. Kubernetes hybrid deployment scheme

3.3.4 Edge deployment

Edge deployments involve deploying Kubernetes clusters at the edge of the network, closer to users or IoT devices. This model reduces latency and enhances performance for edge-computing scenarios. Deploying, for example, the FIWARE Context Broker in edge environments is particularly relevant for smart city applications or IoT use cases. (Mahler 2022).

Figure 9 illustrates the interaction between cloud and edge components. The cloud section manages workloads using a Kubernetes API server, an Edge controller and a Cloud hub to communicate with edge nodes. The edge section contains Edge agent which manages workloads interacting with components like Meta manager, Twin manager and Edge hub to handle storage, device communication and metadata management. The architecture integrates Docker for containerized applications and an MQTT broker for communication with IoT applications using the Event bus which facilitates message passing between edge components and enables communication with external systems.

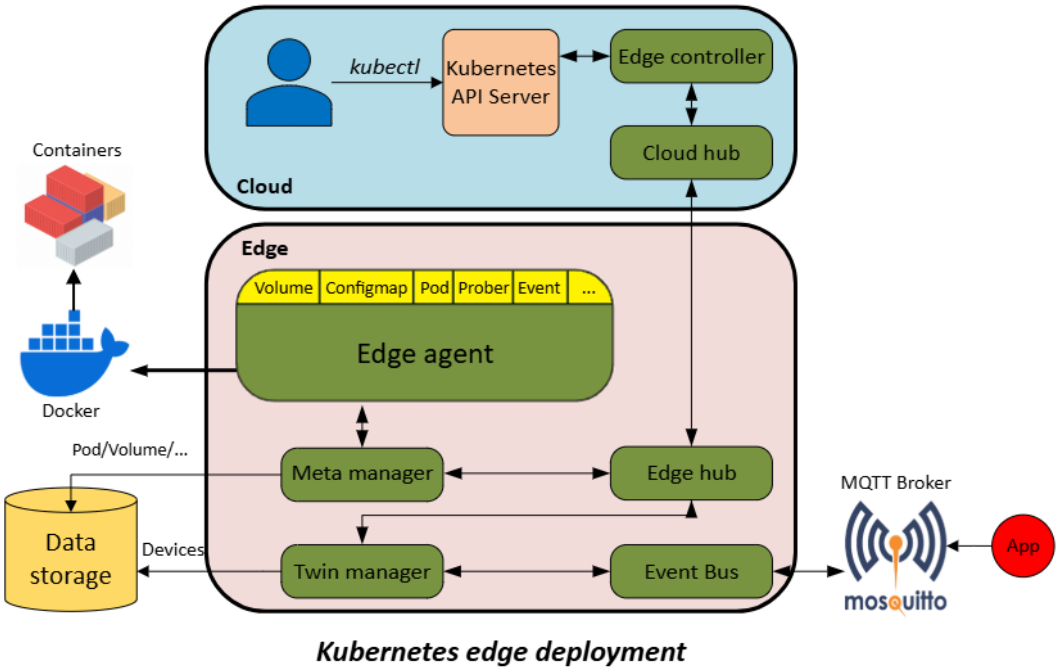

Figure 9. Kubernetes edge deployment scheme

3.3.5 Managed Kubernetes services

Managed Kubernetes services, such as Azure Kubernetes Service, Amazon Elastic Kubernetes Service or Google Kubernetes Engine, provide a platform for deploying applications without the operational costs of managing Kubernetes clusters directly. (Akamai 2025).

Figure 10 illustrates how a managed cloud service provider operates Kubernetes clusters. It demonstrates the hierarchical structure, where a managed Kubernetes service oversees Kubernetes Control Planes which in turn manage Worker nodes that run containerized workloads. Multiple Kubernetes clusters can be managed under the same service, emphasizing automation, scalability and abstraction of infrastructure complexities.

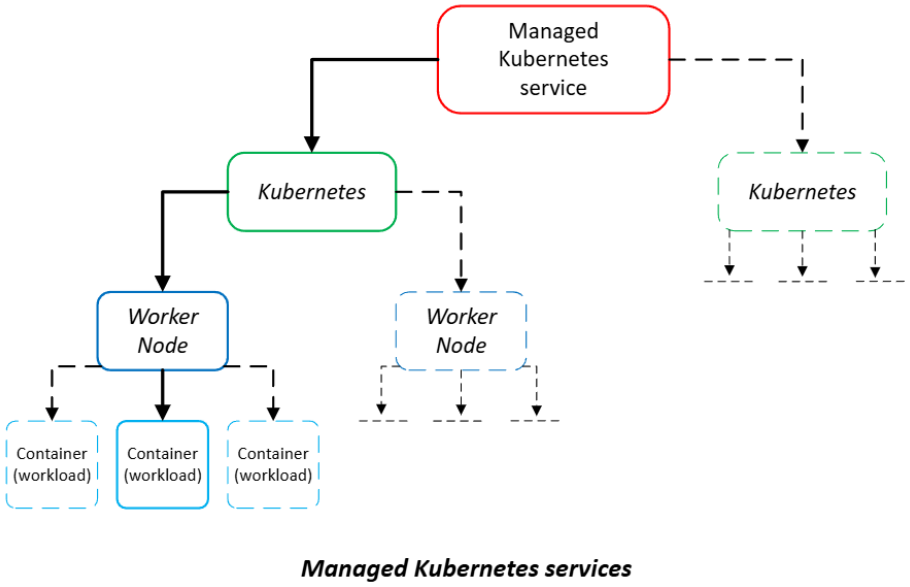

Figure 10. Kubernetes managed services scheme

3.4 Monitoring and logging

Monitoring and logging are critical components of managing applications in a Kubernetes environment. They provide insights into the health and reliability of the system and ensure that issues can be identified and resolved quickly. This section explores the tools and techniques for effective monitoring and logging in Kubernetes, with a focus on deploying and managing the FIWARE Context Broker. (Geekforgeeks 2024).

3.4.1 Monitoring in Kubernetes

Kubernetes monitoring involves tracking the performance and health of the entire cluster and its pods, deployed applications and nodes. (Tigera n.d). Key tools include:

- Prometheus – a popular open-source monitoring system that collects and stores metrics from Kubernetes components and workloads. Prometheus integrates seamlessly with Kubernetes and provides detailed metrics (such as CPU usage, memory utilization and pod status).
- Grafana – a visualization tool that enables the creation of dashboards for real-time insights into Kubernetes metrics. It is often paired with Prometheus.

- Kubernetes Metrics Server – a lightweight component that provides resource usage data (CPU or memory) for pods and nodes, supporting features such as Horizontal Pod Autoscaling.
- Prometheus Alertmanager (or similar tools) – enables the configuration of alerts and notifications based on predefined conditions.
- Kubernetes Dashboard – a web-based UI that allows clients to manage and monitor Kubernetes clusters. It provides a simple graphical interface for administrators to manage different resources, such as pods, deployments and services. Users can view the current state of the cluster, access logs, handle deployments, scale applications and diagnose issues.

Figure 11 displays an interface of the Kubernetes Dashboard utility, which allows users to manage and monitor applications running in Kubernetes clusters. It provides a visual representation of Kubernetes resources such as pods, deployments, services and namespaces. It also provides the ability to view logs, monitor health and make configuration changes in the cluster.

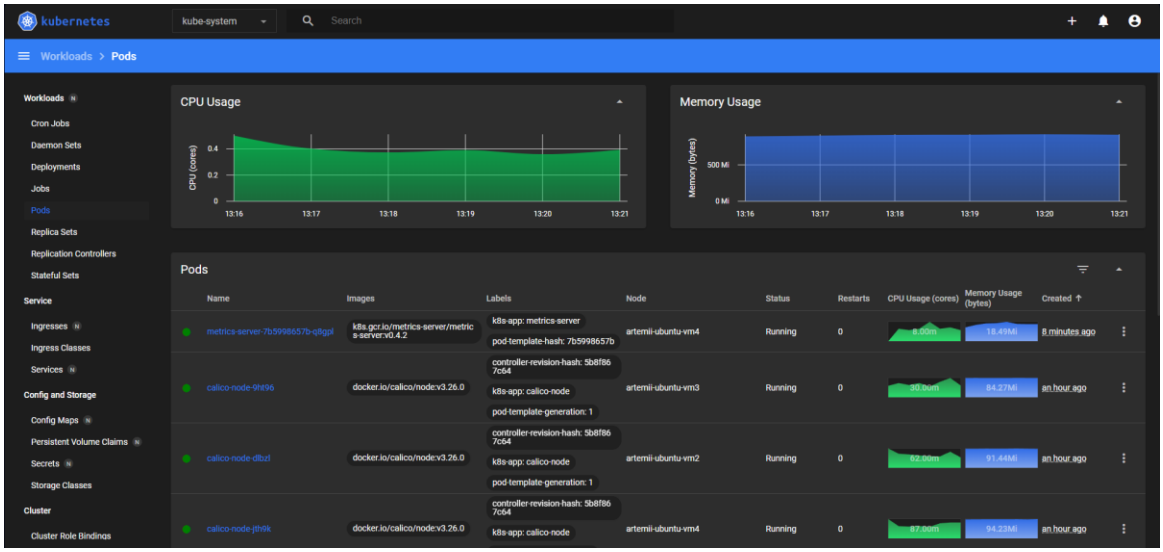

Figure 11. Kubernetes monitoring tool

3.4.2 Logging in Kubernetes

Logs are essential for understanding the behavior of applications and debugging issues. (Lumigo n.d). In Kubernetes, logging can be implemented at multiple levels:

- Container logs – logs generated by individual containers running within pods. These logs can be accessed using kubectl logs.
- Node-level logs – logs from the Kubernetes system components (such as the kubelet, controller manager and scheduler), which provide insights into cluster operations.
- Centralized logging: Fluentd, Elasticsearch, and Kibana (EFK Stack) – Fluentd collects logs from nodes and pods, Elasticsearch indexes them and Kibana provides visualization and search capabilities. Also, there is Loki, a lightweight log collection tool that integrates with Grafana.
- Log retention and analysis – centralized logging ensures that logs are saved and analyzed over time. It helps with long term trend analysis and management.

Figure 12 illustrates a Kubernetes-based node-level logging architecture, where logs from an application running in a pod ("application-pod") are written to a shared log file. A logging agent in a separate pod ("logging-agent-pod") collects these logs and forwards them to a logging backend for storage and analysis. Additionally, a log rotation mechanism ("logrotate") manages log file size and retention. This setup ensures efficient log management and scalability.

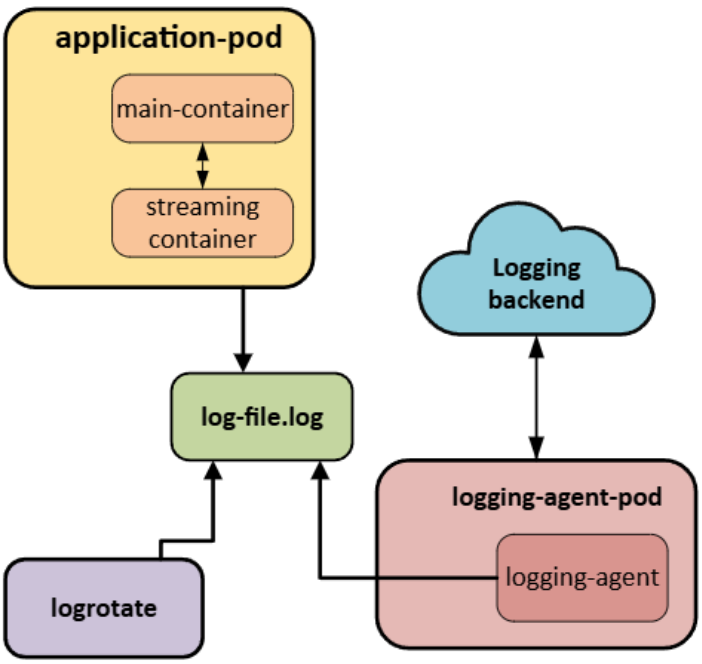

Figure 12. Kubernetes node-level logging scheme

3.5 System Requirements and Core Components

This section outlines essential system configurations and core components required for deploying and operating the Kubernetes cluster and FIWARE Context Broker effectively. It includes details on memory management (swap), application deployment tooling (Helm), data persistence (MongoDB) and key networking and container runtime settings (conntrack and containerd).

3.5.1 Swap

Swap is a disk space portion that the OS uses as virtual memory when RAM is full. It helps to keep processes running when physical memory is exhausted, but at a significant performance cost since disk speed is much slower than that of RAM. Kubernetes requires swap to be disabled because its scheduler and memory management system are designed to work with real available memory. If swap is enabled, the OS might move some Kubernetes-managed processes to disk (swap) that can lead to performance issues and unpredictable behavior in the cluster.

3.5.2 Helm

Helm is a Kubernetes package manager which streamlines the process of deploying and maintaining applications within a cluster. It uses Helm charts, which are predefined and preconfigured configuration templates, which help users to define, install and upgrade complex applications with ease. (RedSwitches 2024).

3.5.3 MongoDB

MongoDB is a NoSQL database that stores data in a flexible JSON format. It is required for the FIWARE Orion Context Broker because Orion manages context information dynamically, and MongoDB provides efficient storage and retrieval of this data, allowing Orion to persist entity states, support queries and handle subscriptions.

3.5.4 Conntrack and containerd

Conntrack enables network connection tracking and ensures proper pod-to-pod communication and networking, while containerd serves as the container runtime to manage the lifecycle of containers within the cluster. Both are crucial for the correct operation of the Kubernetes cluster.

4 PRACTICAL PART: KUBERNETES CLUSTER SETUP AND FIWARE CONTEXT BROKER DEPLOYMENT

For this study, Digitalia's server hardware was utilized. There are a total of 4 virtual machines (VM1, VM2, VM3 and VM4, with IPs 192.168.99.105, 192.168.99.108, 192.168.99.104 and 192.168.99.106 respectively) that are used as nodes of the cluster, and one separate Ubuntu OS VM for configurations. Server VM3 has Quadro M4000 and is used as the main Worker node for the cluster. VM1 is used as the Master node. An access to these VMs is made available via an OpenVPN connection, which was provided by DAME Project. Four virtual machines that are used as nodes can only be accessed using SSH connection, while Ubuntu is available with NoMachine application.

4.1 Setting up a Kubernetes cluster

Before installing Kubernetes, it is important to make sure the system is updated, and required dependencies are installed on all four VMs. The system can be updated, and dependencies can be installed by running the following commands separately on all machines:

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y curl apt-transport-https ca-certificates
```

4.1.1 Installing kubeadm, kubelet and kubectl

After all the updates and installations are successfully completed, the tools that are required for a Kubernetes cluster (kubeadm, kubelet and kubectl) must be

installed. First, a Kubernetes APT repository which enables the installation of Kubernetes-related packages must be added to the Ubuntu system:

```
curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add - echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list
```

Figure 13 shows a PGP public key block that is presented in the terminal after the Kubernetes repository has been added.

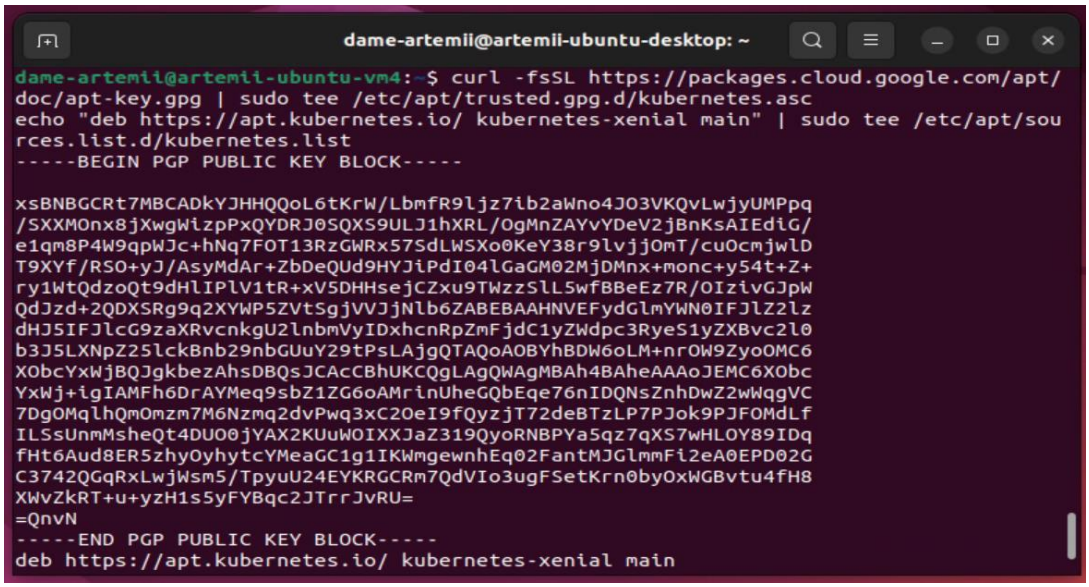

Figure 13. Kubernetes APT repository and PGP public key block

Next, each of the tools (kubelet, kubeadm and kubectl) must be installed separately on each virtual machine using the following commands:

```
sudo apt update
sudo apt install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
```

The command “`apt-mark hold`” prevents packages from being automatically upgraded when the command “`apt upgrade`” is run.

4.1.2 Disabling swap

In order to proceed with setting up a cluster, it is necessary to disable swap on all the servers. Swap can be disabled using the following command:

```
sudo sed -i '/ swap / s/^/#/' /etc/fstab
```

In some cases, the system might prevent “sed” from creating a temporary file in “/etc/” due to restrictive permissions. The alternative is to manually edit the “fstab” file, that can be done by running the following command which opens the “fstab” file in the console:

```
sudo nano /etc/fstab
```

The line that contains “swap” image should be determined and commented out using the “#” sign at the beginning of the line. After that, the file can be saved by pressing CTRL+X, typing “Y” and pressing Enter.

Figure 14 shows the structure of the “fstab” file that has been opened in a terminal. The last line should be commented out to disable swap.

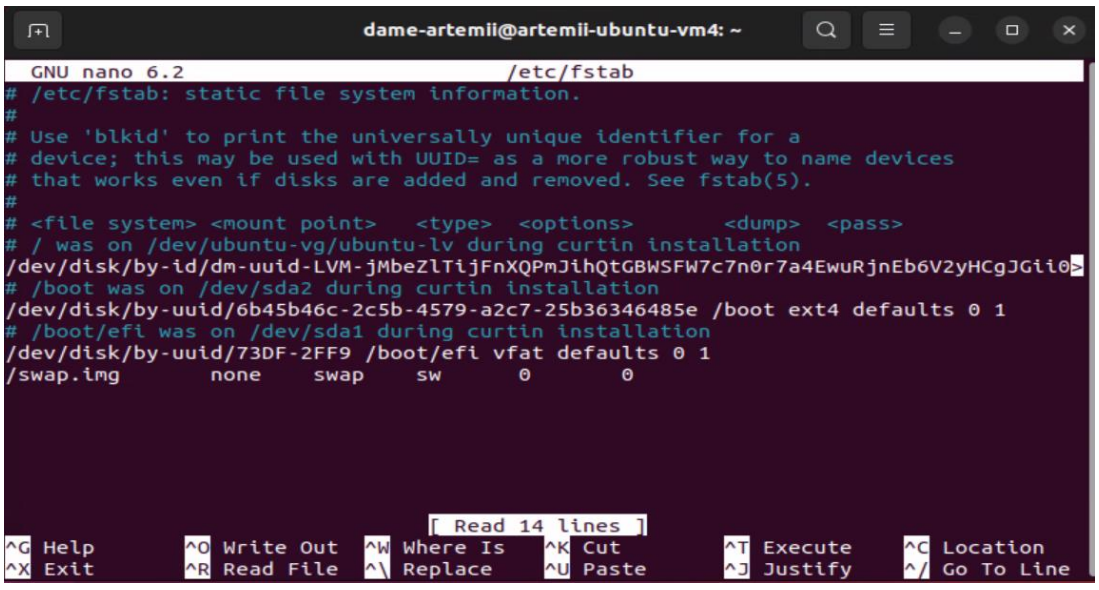

Figure 14. fstab file opened in a terminal

4.1.3 Initializing Master node

The next step is to initialize the Kubernetes Master node. It is necessary to do this only on one Master server, because Kubernetes follows a Master-Worker architecture where only one node handles cluster management. The Master node is responsible for controlling and scheduling workloads, while Worker nodes run the actual applications.

First, conntrack and containerd services must be installed. They can be installed using the following command:

```
sudo apt install -y conntrack containerd
```

IP Forwarding must be enabled for Kubelet services to function properly. This can be achieved by using two commands:

```
echo "1" | sudo tee /proc/sys/net/ipv4/ip_forward
sudo sysctl -w net.ipv4.ip_forward=1
```

New changes can be made permanent using these commands:

```
echo "net.ipv4.ip_forward = 1" | sudo tee -a
/etc/sysctl.conf
sudo sysctl -p
```

Containerd also requires manual configurations that can be done with the following set of commands:

```
sudo mkdir -p /etc/containerd
containerd config default | sudo tee
/etc/containerd/config.toml > /dev/null
sudo systemctl restart containerd
sudo systemctl enable containerd
```

Additionally, Container Runtime CLI, which is required for interacting with the container runtime to manage containers directly, must be installed with the following set of commands:

```
VERSION=$(curl -s https://api.github.com/repos/kubernetes-
```

```
sigs/cri-tools/releases/latest | grep tag_name | cut -d '"' -f 4)
wget https://github.com/kubernetes-sigs/cri-tools/releases/download/$VERSION/crictl-$VERSION-linux-amd64.tar.gz
sudo tar -C /usr/local/bin -xzf crictl-$VERSION-linux-amd64.tar.gz
rm crictl-$VERSION-linux-amd64.tar.gz
```

Kubelet service can be enabled using:

```
sudo systemctl enable --now kubelet
```

The Master node can be initialized using the following command:

```
sudo kubeadm init
```

If port-related errors occur, there is an option to add parameter "`--ignore-preflight-errors=all`" to the previous command, which will ignore all non-critical errors during the initialization.

A few minutes later, the output of the initialization process should be presented containing more details regarding the process of joining the Worker nodes to a cluster.

Figure 15 shows the output that should be received after the Kubernetes Master node has been successfully initialized.

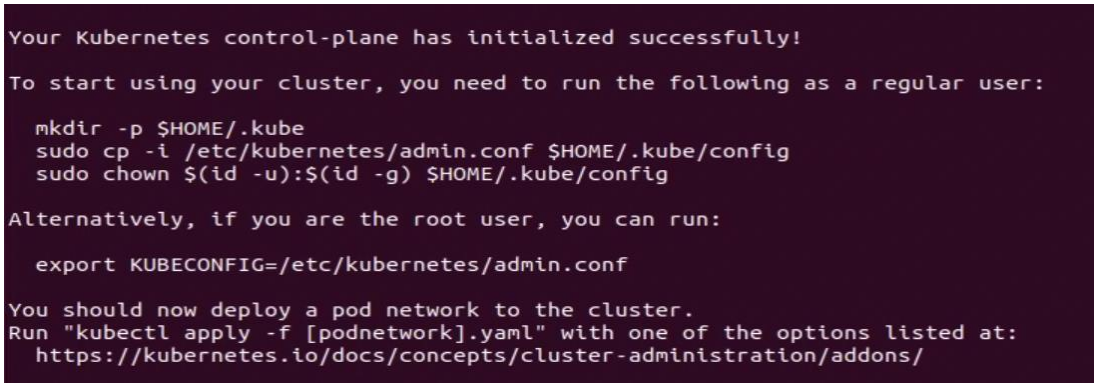

Figure 15. Master node initialization output

After entering the necessary commands provided by the output instructions (`mkdir -p $HOME/.kube`
`sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config`
`sudo chown $(id -u):$(id -g) $HOME/.kube/config`), the current state of a cluster can be verified using "`kubectl cluster-info`" and "`kubectl get nodes`".

Figure 16 shows the output of the aforementioned cluster state verification commands that contains information about the current state of the cluster and a list of connected nodes with respective statuses, roles, age and versions.

```
dame-artemii@artemii-ubuntu-vm1:~$ kubectl cluster-info
Kubernetes control plane is running at https://192.168.99.105:6443
CoreDNS is running at https://192.168.99.105:6443/api/v1/namespaces/kube-system/
services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
dame-artemii@artemii-ubuntu-vm1:~$ kubectl get nodes
NAME                STATUS     ROLES           AGE    VERSION
artemii-ubuntu-vm1  NotReady   control-plane   116s   v1.28.15
```

Figure 16. Master node cluster status info

4.1.4 Joining the Master node

In order to establish the cluster, which was created as a functional Kubernetes cluster, each of the Worker nodes must join the Master node. It can be done using the following command on each Worker node separately:
`sudo kubeadm join 192.168.99.105:6443 --token lx5rhs.l3pp2ahsc0f52rnf \--discovery-token-ca-cert-hash sha256:f980f53514edf12361c49a2f3a126044a2ebdfbc79baf0571c3424655fc32599`
"192.168.99.104" is a local IP address of the Master node, "n43uq0.1ig13wgmpv97z9ig" is a unique token that is used to authenticate the joining node with the Control Plane and "48241765803f184b3c73f6fdaa7c8d094019e7fdad0a5c0a20a7c62223cea020" is a unique hash of the certificate authority of the Kubernetes Control Plane. If the

token has expired, the following command can be used to create a new token for joining the cluster:

```
kubeadm token create --print-join-command
```

Figure 17 shows the output that should be shortly presented, confirming that the Worker node has successfully joined the cluster.

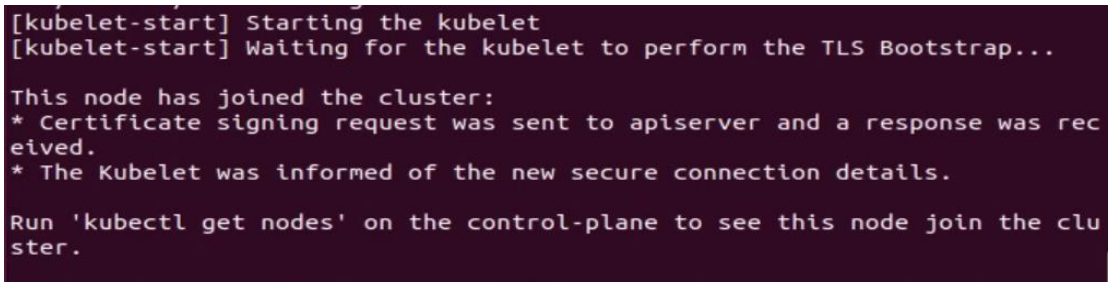

Figure 17. Worker node successfully joined the cluster

The roles of each of the Worker nodes can be assigned using a command "`kubectl label node <node-name> node-role.kubernetes.io/worker=`", where "<node-name>" should be replaced with a hostname of the Worker node.

Figure 18 shows the list of nodes after all the nodes have joined the cluster and all the roles have been assigned.

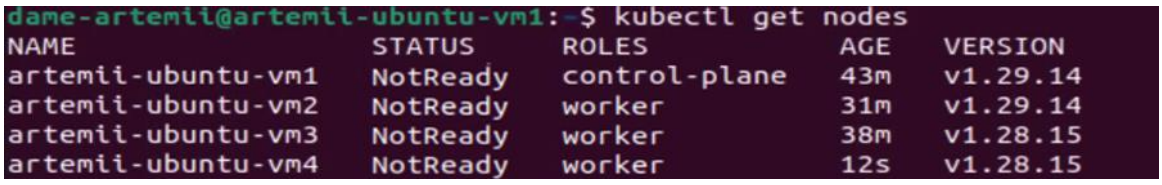

Figure 18. Kubectl nodes info with roles

4.1.5 Enabling communications between pods

In order to enable communications between pods in the cluster, a network plugin Calico is required. It can be installed on the Master node with the following command:

```
kubectl apply -f
https://raw.githubusercontent.com/projectcalico/calico/v3.26
.0/manifests/calico.yaml
```

After it has been initialized, the status of pods can be checked using "`kubectl get pods -n kube-system`".

Figure 19 shows the list of all the available pods in the kube-system namespace. The kube-system namespace is a default namespace where Kubernetes system pods are typically deployed. It contains the names of the pods and their current statuses, as well as number of restarts and age of the pods.

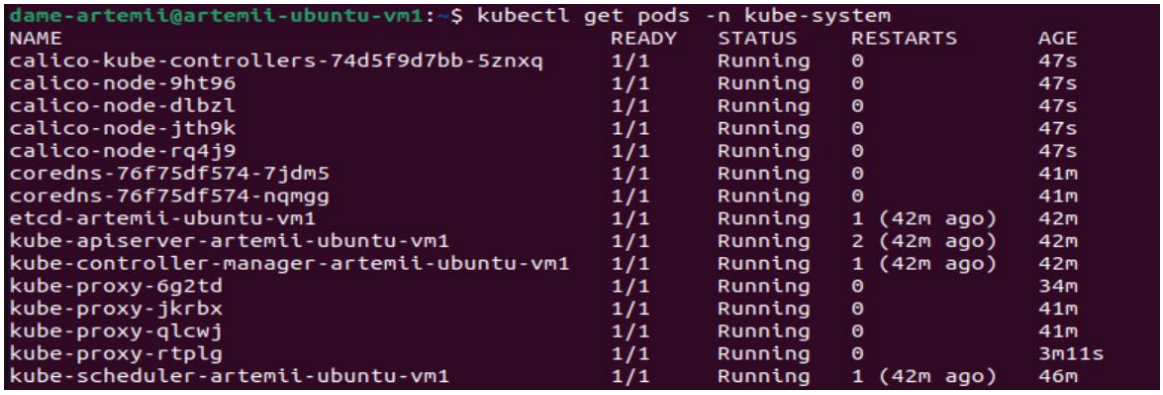

Figure 19. Kubectl list of pods in kube-system

At this stage, all the nodes, including a Master node, should be ready. This can be verified by running "`kubectl get nodes`" once again. It confirms that all the nodes are active, and that Kubernetes cluster is fully functional.

Figure 20 shows that the status of all the nodes changes to "Ready" after they have been configured and enabled.

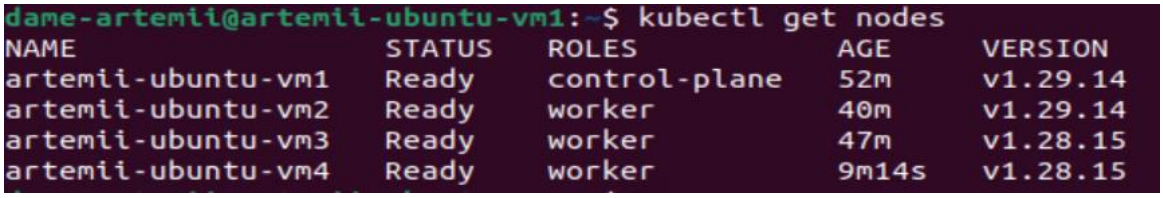

Figure 20. Kubectl nodes info with statuses "Ready"

4.2 Installing Kubernetes Dashboard in a cluster

In order to install Kubernetes Dashboard, it is first important to install Helm, which will also be used in the future for installations. Helm can be installed using “`sudo apt-get install helm`”.

After Helm has been installed, the Kubernetes-dashboard repository must be added using the following command:

```
helm repo add kubernetes-dashboard
https://kubernetes.github.io/dashboard/
```

4.2.1 Deploying Helm release

In order to deploy a Helm release named "kubernetes-dashboard" using the “kubernetes-dashboard” chart, the following command must be run:

```
helm upgrade --install kubernetes-dashboard kubernetes-
dashboard/kubernetes-dashboard --create-namespace --
namespace kubernetes-dashboard
```

A local server with the dashboard can be started using the “`kubectl -n kubernetes-dashboard port-forward --address 0.0.0.0 svc/kubernetes-dashboard-kong-proxy 8443:443`” command, which exposes the kubectl proxy to external connections.

4.2.2 Accessing the dashboard

In order to access the dashboard, it is required to obtain the “bearer” token, which can be done using the following command:

```
kubectl -n kubernetes-dashboard create token admin-user
```

After accessing the dashboard through a web browser by entering the local IP address of the server with a respective port as an URL and entering the previously obtained Bearer token, the status of all the cluster related services and

resources can be seen. CPU and memory usage can be displayed as graphs by installing the Metrics Server with the following commands:

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/download/v0.4.2/components.yaml
kubectl patch deployment metrics-server -n kube-system --type 'json' -p '[{"op": "add", "path": "/spec/template/spec/containers/0/args/-", "value": "--kubelet-insecure-tls"}]'
```

Figure 21 illustrates the display of the Kubernetes Dashboard. It shows the currently existing nodes, their statuses, CPU and memory capacities and time since they were created. This dashboard also contains information about pods (their status and CPU and memory usage), services and events with logs.

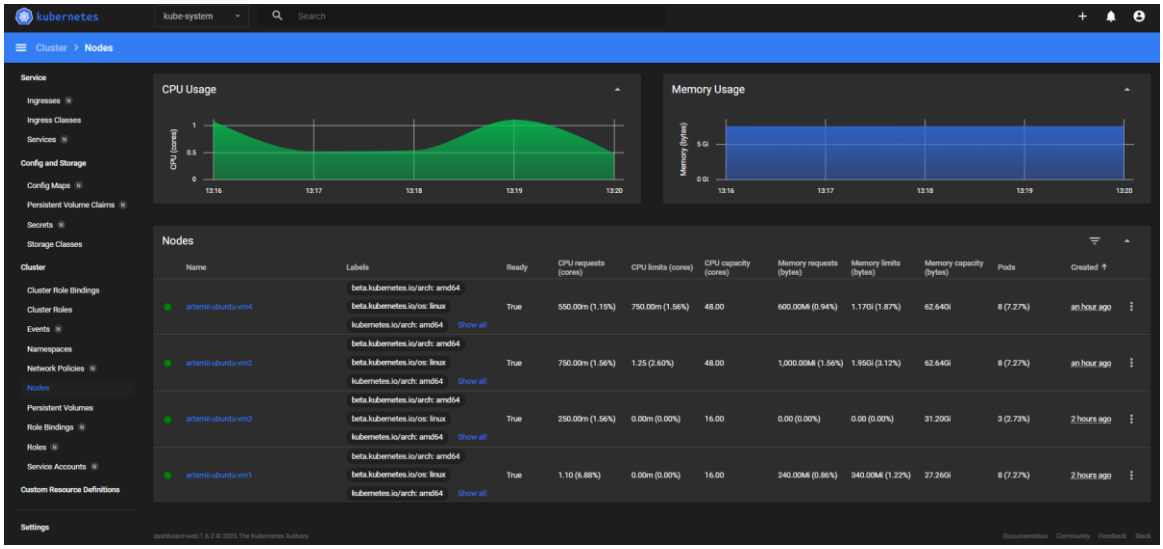

Figure 21. Kubernetes Dashboard opened in a web browser

4.3 Deploying FIWARE Context Broker in the cluster

In order to deploy a FIWARE Context Broker, it is preferable to create a separate namespace (for example “fiware”), which can be achieved using the following command:

```
kubectl create namespace fiware
```

4.3.1 Deploying MongoDB

Before deploying the Context Broker, it is necessary to deploy MongoDB first. Firstly, it is necessary to create a “values.yaml” file, which will contain the configuration for persistent storage and authentication, since MongoDB needs to store data permanently. This file can be created using the following command:

```
cat > values.yaml <<EOF
persistence:
  enabled: true
  storageClass: "standard"
  accessMode: ReadWriteOnce
  size: 8Gi

architecture: replicaset
replicaCount: 3

auth:
  enabled: true
  rootPassword: "qwe123"
EOF
```

After “values.yaml” has been created, MongoDB can be deployed via Helm using the following command:

```
helm install my-mongodb bitnami/mongodb -f values.yaml -n
fiware
```

The status of MongoDB instance can be verified by running “`kubectl get pods -l app.kubernetes.io/name=mongodb -n fiware`”.

Figure 22 displays the status of recently deployed MongoDB instance.

```
dame-artemii@artemii-ubuntu-vm1:~$ kubectl get pods -l app.kubernetes.io/name=mo
ngodb -n fiware
NAME                   READY   STATUS    RESTARTS      AGE
my-mongodb-0           1/1     Running   0             69m
my-mongodb-1           1/1     Running   0             33m
my-mongodb-2           1/1     Running   0             91s
my-mongodb-arbiter-0   1/1     Running   1 (77m ago)   77m
```

Figure 22. MongoD**B** pod status

4.3.2 Deploying the Context Broker

After MongoDB has been successfully deployed, it is possible to deploy the Context Broker itself by first adding a repository and then installing it using Helm with the following commands:

```
helm repo add fiware https://fiware.github.io/helm-charts
helm repo update
helm install fiware-context-broker fiware/orion --namespace
fiware
```

Figure 23 shows the output of successfully installing the Orion-LD Context Broker in a “fiware” namespace.

```
dame-artemii@artemii-ubuntu-vm1:~$ helm install fiware-context-broker fiware/ori
on -f values.yaml -n fiware
NAME: fiware-context-broker
LAST DEPLOYED: Mon Mar  3 17:38:21 2025
NAMESPACE: fiware
STATUS: deployed
REVISION: 1
NOTES:
Successfully deployed Orion-LD.

Connect at fiware-context-broker-orion.fiware:1026
```

Figure 23. Orion-LD successfully deployed

After the Orion Context Broker has been deployed, its service can be started using the following command where port-forwarding to 1026 is specified and made accessible from outside of the localhost:

```
kubectl port-forward svc/fiware-context-broker-orion
1026:1026 -n fiware --address 0.0.0.0
```

The connection can be tested by opening “`http://<master-node-ip>:1026/version`” where “master-node-ip” must be replaced with a local IP of the Master node (in this case 192.168.99.105) in a web browser.

Figure 24 shows the output of the FIWARE Context Broker opened in a web browser. It has no graphical user interface, and it is interactable via RESTful API requests and commonly supports JSON format only.

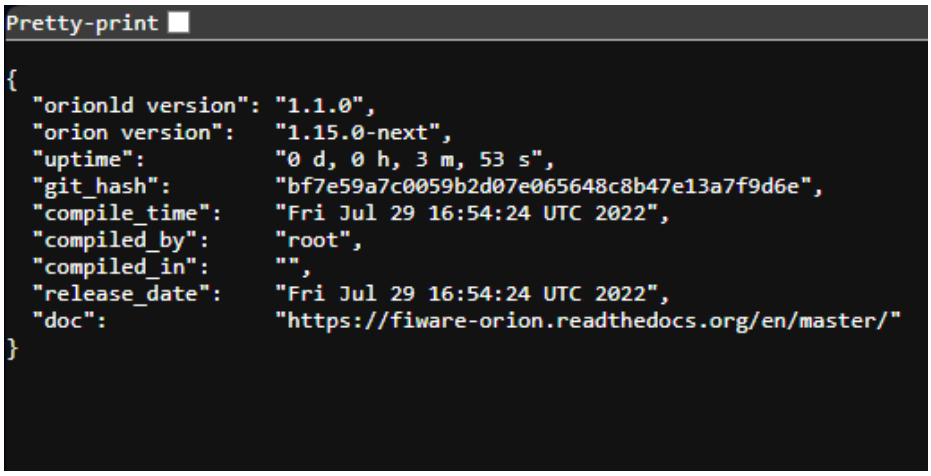

Figure 24. FIWARE Context Broker version opened in a web browser

The operability of the FIWARE Context Broker can be verified by adding some entries. For example, the following command can be run:

```
curl -X POST \
  http://192.168.99.105:1026/v2/entities/ \
  -H 'Content-Type: application/json' \
  -d '{
        "id": "Room1",
        "type": "Room",
        "temperature": {"value": 22.5, "type": "Float"}
      }'
```

It adds a new JSON entry to the broker in a “Room1” ID section with a type of “Room” and a float temperature value of “22.5”.

The accessibility of the Broker can be verified using “`curl http://<master-node-ip>:1026/v2/entities/Room1`”, where “master-node-ip” should be replaced with an actual local IP of the Master node. The output should be as follows:

{"id":"Room1","type":"Room","temperature":{"type":"Float","value":22.5,"metadata":{}}}

The complete deployment status can be verified by viewing currently running pods and services with the following command:

```
kubectl get pods,svc -n fiware
```

4.4 Automating the deployment of the FIWARE Context Broker

Since both MongoDB and FIWARE Context Broker are deployed via Helm, the most suitable way to automate the process is by creating a Helm values file and using a script to install and update the services. First, a MongoDB yaml configuration file (“mongodb-values.yaml”) is created. It should contain the following data:

```
persistence:
  enabled: true
  storageClass: "standard"
  accessMode: ReadWriteOnce
  size: 8Gi
architecture: replicaset
replicaCount: 3
auth:
  enabled: true
  rootPassword:
    secretKeyRef:
      name: mongodb-secret
      key: qwe123
```

4.4.1 Creating the deployment script

After the MongoDB configuration file has been created, also the deployment script (“deploy.sh”) must be created. It should contain the most important configuration values that were previously used in this thesis for manual deployment, such as namespace name, Helm repositories and commands to

deploy MongoDB and FIWARE Context Broker. A sample of such script is the following:

```
NAMESPACE="fiware"
kubectl get namespace $NAMESPACE >/dev/null 2>&1 || kubectl
create namespace $NAMESPACE
curl
https://raw.githubusercontent.com/helm/helm/master/scripts/g
et-helm-3 | bash
helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo add fiware https://fiware.github.io/helm-charts
helm repo update
helm upgrade --install my-mongodb bitnami/mongodb -f
mongodb-values.yaml -n $NAMESPACE
kubectl wait --for=condition=ready pod -l
app.kubernetes.io/name=mongodb -n $NAMESPACE --timeout=300s
helm upgrade --install fiware-context-broker fiware/orion -n
$NAMESPACE
kubectl wait --for=condition=ready pod -l
app.kubernetes.io/name=orion -n $NAMESPACE --timeout=300s
echo "FIWARE Context Broker has been successfully deployed"
```

After the script has been created, it must be made executable using “`chmod +x deploy.sh`” and can be run using “`./deploy.sh`”.

If the Kubernetes cluster is private and not accessible from an external network, the “deploy.sh” script can be executed directly on a host server within the cluster network and will automatically deploy the FIWARE Context Broker in a cluster.

4.4.2 Configuring Git service

In order to make the configuration of the FIWARE Context Broker automatically deployable using helm, GitHub repository must be used. Git service can be installed, and a repository can be added to the cluster, by running the following commands:

```
sudo apt update && sudo apt install git -y
git init
git remote add origin https://github.com/username/fiware-
deployment.git
git remote -v
```

The “username” should be replaced with a respective GitHub username.

In order to automate the deployment using GitHub Actions platform, a new GitHub Actions Workflow YAML file (“deploy.yml”) must be created inside the “.github/workflows/” folder using “`mkdir -p .github/workflows`” and “`nano .github/workflows/deploy.yml`” commands with the following content:

```
name: Deploy FIWARE to Kubernetes
on:
  push:
    branches:
      - main
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v3
      - name: Set up Kubernetes CLI
        uses: azure/setup-kubectl@v3
        with:
          version: 'latest'
      - name: Set up Helm
        uses: azure/setup-helm@v3
        with:
          version: 'latest'
      - name: Authenticate with Kubernetes Cluster
        run: |
```

```
          echo "${{ secrets.KUBECONFIG }}" | base64 --decode
> kubeconfig
          export KUBECONFIG=kubeconfig
          kubectl cluster-info
      - name: Deploy FIWARE Context Broker with Helm
        run: ./deploy.sh
```

For the CI/CD pipeline to deploy into the Kubernetes cluster, GitHub Actions requires authentication. This can be obtained and submitted by first opening the "kubeconfig" file with the command "`cat ~/.kube/config | base64`", copying the base64 output and entering it as a GitHub Secret. GitHub Secret can be entered by opening the GitHub repository and following this path: Settings, Secrets and variables, Actions, New secret. The new secret should be named "KUBECONFIG", and a base64 string received from kubeconfig file should be pasted. The GitHub Action will use this secret to authenticate.

4.4.3 Committing and pushing the files

After Kubernetes authentication has been set up, it is possible to automatically deploy the FIWARE Context Broker by committing and pushing the files from GitHub repository by running the following commands:

```
git add .github/workflows/deploy.yml
git commit -m "Add CI/CD deployment for FIWARE"
git push origin main
```

GitHub Actions automatically deploys the FIWARE Context Broker to the physical Kubernetes cluster whenever changes are pushed to the main branch after manually modifying files or deploying services.

5 CONCLUSION

This thesis has demonstrated an automated approach to deploying the FIWARE Context Broker in Kubernetes environment, improving scalability, reliability and ease of management. By leveraging Helm and Kubernetes operators, the

process is significantly optimized, reducing manual intervention and enhancing adoption for IoT and smart applications.

While this study focused on the FIWARE Orion Context Broker, the deployment methodology and automation strategies that were outlined can be adapted to other Context Brokers and FIWARE components with minimal modifications, offering a reusable framework for deployment. Moreover, automation in containerized environments improves system resilience, facilitates continuous deployment and simplifies large-scale application management.

Beside deployment, this study highlights the broader impact of automation in modern containerized environments, enhancing system resilience, facilitating continuous integration and delivery and simplifying the management of large-scale distributed architectures. These advantages make Kubernetes an optimal choice for deploying FIWARE components in production environments.

In the future, further optimizations, such as integrating managed cloud-native Kubernetes services for enhanced system resilience or extending automation to multi-cluster and hybrid-cloud deployments, could be explored. These enhancements would improve FIWARE’s adaptability for diverse infrastructure needs.

In summary, this thesis provides a scalable and efficient foundation for FIWARE deployments in Kubernetes, offering valuable insights for developers and organizations aiming to seamlessly integrate context-aware applications.

REFERENCES

XAMK. 2024. South Savo Data Economy Accelerator – Shared Data as a Joint Success Factor (DAME). Web page. Available at: https://www.xamk.fi/en/project/dame/ [Accessed 21 February 2025].

XAMK Memory Lab. 2023. Home. Web page. Available at: https://memorylab.fi/en/ [Accessed 21 February 2025].

XAMK Memory Lab. 2023a. Conducted pilots and cases. Web page. Available at: https://memorylab.fi/en/AIKO/ [Accessed 21 February 2025].

XAMK Memory Lab. 2023b. Equipment. Web page. Available at: https://memorylab.fi/en/laitteisto/ [Accessed 21 February 2025].

Zangelin, K. 2013. GitHub - FIWARE/context.Orion-LD. Web page. Available at: https://github.com/FIWARE/context.Orion-LD [Accessed 7 March 2025].

Dianese, G. 2020. FIWARE Context Broker: The engine for future energy systems. Web page. Available at: https://www.fiware.org/2020/12/11/fiware-context-broker-the-engine-for-future-energy-systems/ [Accessed 7 March 2025].

FIWARE. n.d. About FIWARE. Web page. Available at: https://www.fiware.org/about-us/ [Accessed 7 March 2025].

The Kubernetes Authors. 2024. Kubernetes Documentation. Web page. Available at: https://kubernetes.io/docs/home/ [Accessed 7 March 2025].

Kubernetes. 2024a. Overview. Web page. Available at: https://kubernetes.io/docs/concepts/overview/ [Accessed 7 March 2025].

Microsoft. n.d. What is Kubernetes? Web page. Available at: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-kubernetes#watch-how-kubernetes-works [Accessed 7 March 2025].

Prakarsh. 10 October 2019. Why Use Kubernetes for Container Orchestration? Web page. Available at: https://devtron.ai/blog/why-use-kubernetes-for-container-orchestration/#:~:text=Kubernetes%20is%20often%20considered%20the,application%20development%20and%20deployment%20needs [Accessed 7 March 2025].

Aleksic, M. 28 July 2022. Kubernetes Use Cases. Web page. Available at: https://phoenixnap.com/kb/kubernetes-use-cases [Accessed 7 March 2025].

Tettamanti, L. 2 January 2025. Kubernetes Explained: Benefits, Use Cases, and Why Airbnb,Spotify and CERN Rely on It. Web page. Available at: https://dev.to/lorenzo_tettamanti/kubernetes-explained-benefits-use-cases-and-why-airbnbspotify-and-cern-rely-on-it--315f [Accessed 7 March 2025].

Susnjara, S., Smalley, I. 11 March 2024. What is Kubernetes? Web page. Available at: https://www.ibm.com/think/topics/kubernetes [Accessed 7 March 2025].

The Cloud Native Experts. 7 May 2024. Managing Containers in Kubernetes. Web page. Available at: https://www.aquasec.com/cloud-native-academy/kubernetes-101/managing-containers-in-kubernetes/ [Accessed 7 March 2025].

Hausenblas, M. May 2018. Container Networking. Ebook. Sebastopol: O'Reilly Media. Available at: https://www.dbooks.org/container-networking-1492036811/read/ [Accessed 7 March 2025].

Mannambeth, M. n.d. Kubernetes for beginners. PDF document. Available at: https://kodekloud.com/wp-content/uploads/2020/11/Kubernetes-for-Beginners.pdf [Accessed 7 March 2025].

Stratoscale. n.d. EVERYTHING KUBERNETES: A PRACTICAL GUIDE. PDF document. Available at: https://iamondemand.com/wp-content/uploads/2019/11/Kubernetes-eBook.pdf [Accessed 7 March 2025].

Waleed, M. November 2024. CONTAINER ORCHESTRATION USING KUBERNETES. PDF document. Available at: https://trepo.tuni.fi/bitstream/handle/10024/162010/WaleedMuhammad.pdf [Accessed 7 March 2025].

Coullon, H., Perdereau, E. n.d. Kubernetes – Containers orchestration. PDF document. Available at: https://helene-coullon.fr/download/teachings/ue-nuage/cours-23-24/cours_k8s.pdf [Accessed 7 March 2025].

Hohn, A. 6 September 2022. The Book of Kubernetes: A Complete Guide to Container Orchestration. No Starch Press.

Chesterfield, G. 12 September 2024. The Kubernetes Handbook. Amazon Digital Services LLC.

Sayfan, G. 1 April 2018. Mastering Kubernetes. Master the art of container management by using the power of Kubernetes. 2nd edition. Packt Publishing Limited.

Block, A., Dewey, A. 2022. Managing Kubernetes Resources Using Helm. Packt Publishing Limited.

Baier, J., Sayfan, G., White, J. 1 May 2019. The Complete Kubernetes Guide. Become an expert in container management with the power of Kubernetes. Packt Publishing Limited.

Markelov, A. 2019. Vvedenie v tekhnologii konteinerov i Kubernetes. DMK Press.

Walker, J. 3 February 2025. What is Kubernetes Architecture? – Components Overview. Web page. Available at: https://spacelift.io/blog/kubernetes-architecture [Accessed 7 March 2025].

Walker, J. 26 February 2024. Kubernetes Service – What It is, Types & Examples. Web page. Available at: https://spacelift.io/blog/kubernetes-service [Accessed 7 March 2025].

Kubernetes. 2024b. Cluster Architecture. Web page. Available at: https://kubernetes.io/docs/concepts/architecture/ [Accessed 7 March 2025].

Khisty, S. 2024. Kubernetes Architecture: The Ultimate Guide. Web page. Available at: https://devtron.ai/blog/kubernetes-architecture-the-ultimate-guide/ [Accessed 7 March 2025].

Graph. n.d. Container Runtime. Web page. Available at: https://www.graphapp.ai/engineering-glossary/containerization-orchestration/container-runtime [Accessed 7 March 2025].

Warren, W. March 2020. Getting Started with Containers and Kubernetes. PDF document. Available at: https://www.cncf.io/wp-content/uploads/2020/08/Getting-Started-with-Containers-and-Kubernetes_-March-2020-CNCF-Webinar.pdf [Accessed 7 March 2025].

Nyambati, T. 6 December 2018. Kubernetes 101 part 3: Deployments and pods. Web page. Available at: https://maskedweaver.cloud/kubernetes-101-part-3-deployments-and-pods-217808afddfe [Accessed 7 March 2025].

Raina, A. February 2022. Demystifying Kubernetes with 100 slides. PDF document. Available at: https://collabnix.com/wp-content/uploads/2022/02/Understand-Kubernetes-in-100-slides.pdf [Accessed 7 March 2025].

Kubernetes. 2025a. Nodes. Web page. Available at: https://kubernetes.io/docs/concepts/architecture/nodes/ [Accessed 7 March 2025].

Kubernetes. 2025b. Pods. Web page. Available at: https://kubernetes.io/docs/concepts/workloads/pods/ [Accessed 7 March 2025].

Kubernetes. 2025c. Services. Web page. Available at: https://kubernetes.io/docs/concepts/services-networking/service/ [Accessed 7 March 2025].

Pant, A. 28 December 2024. Kubernetes 101: A Beginner’s Guide to the Container Orchestration Platform. Web page. Available at: https://medium.com/@arvindpant/kubernetes-101-a-beginners-guide-to-the-container-orchestration-platform-12a75a51e76f [Accessed 7 March 2025].

Sedor, R. n.d. Kubernetes Patterns. Reusable Elements for Designing Cloud-Native Applications. PDF document. Available at: https://events.redhat.com/accounts/register123/redhat/events/7013a000002dbroaa4/Kubernetes_Design_Patterns.pdf [Accessed 7 March 2025].

Czako, Z. n.d. Stupid Simple Kubernetes. PDF document. Available at: https://more.suse.com/rs/937-DCH-261/images/SUSE%20269%20Stupid%20Simple%20Kubernetes%20v07.pdf [Accessed 7 March 2025].

Ferrer, J. Kubernetes Architecture Explained: A Deep Dive into Cloud-Native Scalability. Web page. Available at: https://www.datacamp.com/blog/kubernetes-architecture-explained [Accessed 7 March 2025].

Platform9. September 2020. Operating Kubernetes: Everything You Need to Know. PDF document. Available at: https://platform9.com/wp-content/uploads/2020/09/Operating-Kubernetes.pdf [Accessed 7 March 2025].

Burns, B., Tracey, C. 2018. Managing Kubernetes. Operating Kubernetes Clusters in the Real World. O'Reilly Media.

Luksa, M. 29 January 2018. Kubernetes in Action. Manning Publications.

Mahler, C. 18 August 2022. Kubernetes on the edge: getting started with KubeEdge and Kubernetes for edge computing. Web page. Available at: https://www.cncf.io/blog/2022/08/18/kubernetes-on-the-edge-getting-started-with-kubeedge-and-kubernetes-for-edge-computing/ [Accessed 7 March 2025].

Tigera. 2025. Multi-Cluster Kubernetes: A Practical Guide. Web page. Available at: https://www.tigera.io/learn/guides/kubernetes-networking/kubernetes-multi-cluster/ [Accessed 7 March 2025].

Ghosh, B. 17 September 2023. Hybrid Kubernetes Model: Bridging Clouds and On-Premises. Web page. Available at: https://medium.com/@bijit211987/hybrid-kubernetes-model-bridging-clouds-and-on-premises-4206cd430d50 [Accessed 7 March 2025].

Intercept. 2024. Going hybrid with Kubernetes. Web page. Available at: https://intercept.cloud/en-gb/blogs/going-hybrid-with-kubernetes [Accessed 7 March 2025].

Akamai. 2025. What Is Managed Kubernetes?. Web page. Available at: https://www.akamai.com/glossary/what-is-managed-kubernetes [Accessed 7 March 2025].

Gcore. 24 April 2024. What is Managed Kubernetes. Web page. Available at: https://gcore.com/learning/what-is-managed-kubernetes/ [Accessed 7 March 2025].

Platform9. August 2018. The Ultimate Guide to Deploy Kubernetes. PDF document. Available at: https://platform9.com/wp-content/uploads/2018/08/the-ultimate-guide-to-deploy-kubernetes.pdf [Accessed 7 March 2025].

Sayfan, G. 25 May 2017. Mastering Kubernetes: Large scale container deployment and management. Packt Publishing Limited.

Geekforgeeks. 28 May 2024. Kubernetes Monitoring and Logging: Tools and Best Practices. Web page. Available at: https://www.geeksforgeeks.org/kubernetes-monitoring-and-logging/ [Accessed 7 March 2025].

Pratap Bhuyan, A. 11 November 2023. Logging and Monitoring in Kubernetes. Web page. Available at: https://dev.to/adityapratapbh1/logging-and-monitoring-in-kubernetes-53o2 [Accessed 7 March 2025].

Sussmann, M. 17 January 2023. Logging and monitoring Kubernetes. Web page. Available at: https://www.sumologic.com/blog/kubernetes-logs/ [Accessed 7 March 2025].

Lumigo. n.d. 3 Kubernetes Logging Methods with Examples. Web page. Available at: https://lumigo.io/kubernetes-monitoring/3-kubernetes-logging-methods-with-examples/ [Accessed 7 March 2025].

Tigera. n.d. Kubernetes Monitoring: 5 Tools and 4 Best Practices You Must Know About. Web page. Available at: https://www.tigera.io/learn/guides/kubernetes-monitoring/ [Accessed 7 March 2025].

Bhogayata, K. 25 February 2025. Kubernetes Monitoring: A Complete Guide. Web page. Available at: https://middleware.io/blog/kubernetes-monitoring/ [Accessed 7 March 2025].

Cooney, C. March 2022. Logging in Kubernetes. Ebook. Available at: https://coralogix.com/wp-content/uploads/2022/03/eBOOK_04-kubernetes_v6.pdf [Accessed 7 March 2025].

Helanova, T. 2024. Kubernetes Logging Operator. Bachelor’s thesis. PDF document. Available at: https://is.muni.cz/th/ovfjr/kubernetes_logging_operator_helanova.pdf [Accessed 7 March 2025].

Huang, K., Jumde, P. 9 July 2020. Learn Kubernetes Security. Securely orchestrate, scale, and manage your microservices in Kubernetes deployments. Packt Publishing Limited.

Goasguen, S., Hausenblas, M. February 2018. Kubernetes Cookbook. O'Reilly Media.

Burns, B., Villalba, E., Strebel, D., Evenson, L. November 2019. Kubernetes Best Practices. O'Reilly Media.

RedSwitches. August 2024. What is Helm and Helm Charts: A Comprehensive Guide. Web page. Available at: https://medium.com/@redswitches/what-is-helm-and-helm-charts-a-comprehensive-guide-ca8a3526f7fd [Accessed 8 May 2025].