

Seminar Report

Benchmarking KubeEdge

Vincenz Dumann

MatrNr: 22308641

Supervisor: Jonathan Decker

Georg-August-Universität Göttingen
Institute of Computer Science

March 31, 2023

Abstract

This report presents a study on benchmarking KubeEdge, a software that enables edge computing using Kubernetes. The objective of the study was to assess the resource requirements of KubeEdge and determine its suitability for edge computing. Two networks were installed, and the performance of KubeEdge was measured. The paper also provides a theoretical overview of the pros and cons of edge computing and details the installation process of KubeEdge. The benchmark results indicate that KubeEdge is well-suited for Kubernetes in edge computing, and its resource consumption is minimal. Overall, the study concludes that KubeEdge is a suitable software for edge computing and can be used for various edge computing applications.¹

¹An AI-Generated Poem made out of this abstract can be found in the appendix. Because - why not?

Contents

List of Tables	iii
List of Figures	iii
Listings	iii
List of Abbreviations	iv
1 Introduction	1
1.1 Outline	1
1.2 Contributions	1
1.3 Motivation	1
1.4 Related Work	2
2 Basics	3
2.1 EdgeComputing	3
2.1.1 Layers of EdgeComputing	3
2.2 Kubernetes	4
2.2.1 Pros of using Kubernetes	5
2.2.2 Cons of using Kubernetes	5
2.3 Kubernetes and Edge Computing	6
2.4 KubeEdge	6
2.4.1 KubeEdge Architecture:	6
3 Setting up KubeEdge	7
3.1 Project Plan	7
3.2 Approach 1: Single Knot	8
3.2.1 Preperation and Environment Setup	8
3.2.2 Benchmarking	9
3.2.3 Performance Measurements	10
3.3 Approach 2: Network	11
3.3.1 Preperation and Environment Setup	11
3.3.2 Installation	11
3.3.3 Benchmarking	12
4 Conclusion	13
4.1 Summary	13
4.2 Review of Goals	13
4.3 Outlook	14
References	15
A Log File	A1
B Misc	A1
B.1 The abstract in a lyrical form	A1

List of Tables

List of Figures

1	EdgeComputing - Architecture	4
2	VT-X/AMD-v not enabled - a problem without a real solution	9
3	Ressource need of Minikube with KubeEdge	10
4	Example Log File	A1
5	A total amount of 21 VMs were created for that project	A3

Listings

List of Abbreviations

HPC High-Performance Computing

1 Introduction

Managing large computer networks brings many challenges - the individual parts that make up the network need to be set up and managed, and the work that needs to be done needs to be divided up sensibly (ideally intelligently). In addition, the system must be able to react to unexpected incidents, such as the failure of one or even several machines. For the solution of these tasks, which can be summarised under the term "orchestration", there are several approaches, and of course corresponding offers on the market. One common solution is Kubernetes[Ros+21] - an open source system developed by Google for managing container applications. However, Kubernetes is not designed for edge computing[Xio+18] - this is a decentralised approach to network architecture in which many calculations are not made classically on the central server, but directly in the edges. This approach dramatically increases the number of computing units in the network - and thus also the need for good orchestration. Kubernetes would be very well suited for this, were it not for some weaknesses, which will be discussed in more detail later in this paper. A solution for this is promised by the application "KubeEdge" - a software, also available as open source, which, according to the developers, enables the use of Kubernetes for edge computing. The seminar work, which was completed by this thesis, focused on this software - KubeEdge: It is installed and analysed in terms of functionality, resource consumption and user-friendliness.

1.1 Outline

In this introduction, the exact background of the project and its motivation are discussed, and further literature is presented. Afterwards, a chapter on the basics takes a closer look at Kubernetes, EdgeComputing and, of course, KubeEdge, including a look at the respective architectures. The following chapter describes the approach chosen to test Kubernetes and provides detailed instructions for installing and configuring a KubeEdge-based network. Afterwards, the results of the performance analysis are presented, and a final verdict is given.

1.2 Contributions

This thesis was written as part of the seminar "Scalable Computing Systems and Applications in AI, BigData and HPC", which was taught by Professor Dr. Julian Kunkel. Supervisor of this work is M.Sc. Jonathan Decker, Author Vincenz Dumann, Master student in the fourth semester.

1.3 Motivation

Of course, this project is not just an end in itself - in addition to the educational purpose that every university project should fulfil, the knowledge gained here should also flow into the "Decice" project[DECnd]. Decice is a joint project of the universities of Göttingen, Stuttgart, Marmara, Bologna and others, as well as some companies from the researching, private economy. The GWDG Göttingen is also involved. The goal of the project is stated as follows: "DECICE aims to develop an AI-based, open and portable cloud management framework for automatic and adaptive optimization and deployment of applications in a federated infrastructure, including computing from the very large (e.g., HPC systems) to

the very small (e.g., IoT sensors connected on the edge)."[Kau+19]. Furthermore, the roadmap of the project is also described: "Working at such vastly different scales requires an intelligent management plane with advanced capabilities that allow it to proactively adjust workloads within the system based on their needs, such as latency, compute power and power consumption. Therefore, we envision an AI-model, which can use a digital twin of the resources available, to make real-time scheduling decisions based on telemetry data from the resources."[DECnd]

The DECICE framework will be able to dynamically balance different workloads, optimise the throughput and latency of the system resources (compute, storage, and network) regarding performance and energy efficiency and quickly adapt to changing conditions."[DECnd] Basically, it is about creating, configuring and controlling the most diverse networks, including those with an edge computing approach, via a central framework, using AI. KubeEdge is being discussed as a possible software to be used, and should therefore be examined in detail before a final decision is made. Accordingly, it is important to find out how much resources KubeEdge uses, and how easy it is to install and initialise - after all, these tasks are to be taken over automatically by the framework later.

1.4 Related Work

For this work, of course, we have also drawn on various academic works that deal with similar topics. The basic chapter of this thesis is largely based on the following sources:

For basic knowledge about Kubernetes, the book "Production Kubernetes - Building Successful Application Platforms" (freely available on the web) by the authors Josh Rosso, Rich Lander, Alex Brand, John Harris was very helpful; the connection to edge computing is made by the work "Kubernetes and Edge Computing" by Sergio Mendez.

An overview of edge computing and current research in this area is provided in "An Overview on Edge Computing Research" by the authors Keyan Cao, Yefan Liu, Gongjie Meng and Qimeng Sun, while the paper "Challenges and Opportunities in Edge Computing" by Blessen Varghese, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick and Dimitrios S. Nikolopoulos provides more in-depth information on advantages and disadvantages. However, in contrast to the other works presented here, this is already somewhat older (from 2016), and therefore already seems somewhat outdated in some places.

Scientific studies on KubeEdge are of course much rarer than on the more general topics, and it is also not always easy to differentiate how the studies are related to the developers of KubeEdge. "Extend Cloud to Edge with KubeEdge", for example, provides a fairly detailed overview of the functions, but also of the underlying architecture, although at least some of the authors Ying Xiong, Yulin Sun, Li Xing and Ying Huang are also mentioned, at least by name, on the homepage of KubeEdge itself - which also contains a "paper" about a test of KubeEdge in real conditions, "Test Report on KubeEdge with 100000 Nodes" by Wack Xu. A similar, but much more detailed scenario is described in "Node-Based Horizontal Pod Autoscaler in KubeEdge-Based Edge Computing Infrastructure" by Le Hoang Phuc, Majid Kundroo, Dae-Heon Park Sehan Kim and Taehong Kim, which comes to similar results, but where at least no obvious connection to the developers can be found.

2 Basics

This chapter will first introduce some basics - in detail, it will deal with the terms "Kubernetes", "EdgeComputing" and "KubeEdge". The architecture of these individual elements will also be discussed and - at least as far as the scope of this work allows - a look will be taken at any weaknesses or risks for their use. In addition, it will be worked out how these three technologies, or paradigms, are interrelated and what areas of application there may be.

2.1 EdgeComputing

Edge computing is defined slightly differently - depending on the source - but all definitions can be reduced to a common denominator: "Edge computing refers to a distributed computing paradigm that enables data processing and analysis to be carried out closer to the source of data generation, typically at or near the edge of the network. This approach leverages a network of decentralized devices and sensors to perform real-time data processing, reducing latency, improving response times, and enabling faster decision-making. Edge computing also minimizes the amount of data that needs to be transmitted to centralized cloud servers, reducing bandwidth requirements and associated costs. This technology is increasingly used in IoT (Internet of Things) applications, smart homes, autonomous vehicles, and industrial automation, among others." [Cao+20]

Edge computing typically consists of three layers: the cloud layer, the edge layer, and the device layer. Those are displayed in Figure 1 and explained more in-depths in the following subchapter.

2.1.1 Layers of EdgeComputing

- The cloud layer is the top layer of the edge computing architecture and is made up of cloud data centers that provide infrastructure and services to the edge devices. The cloud layer is responsible for managing the edge devices and coordinating data flows between the edge devices and the cloud data centers. It provides centralized management, monitoring, and orchestration of edge devices and applications. [Var+16]
- The edge layer is the middle layer of the edge computing architecture and is made up of edge servers or gateways located closer to the end-user or data source than the cloud data centers. The edge layer is responsible for processing and analyzing data from the device layer and forwarding the relevant data to the cloud layer for further processing or storage. The edge layer enables real-time data processing, reduces network traffic, and improves overall system performance.
- The device layer is the bottom layer of the edge computing architecture and is made up of the end-user devices or sensors that generate data. The device layer includes a wide range of devices such as smartphones, sensors, wearables, and IoT devices. The device layer is responsible for collecting and transmitting data to the edge layer for processing and analysis.

In summary, the cloud layer, edge layer, and device layer are the three key components of the edge computing architecture. Each layer plays a specific role in the overall system, with the cloud layer providing centralized management and orchestration, the edge layer

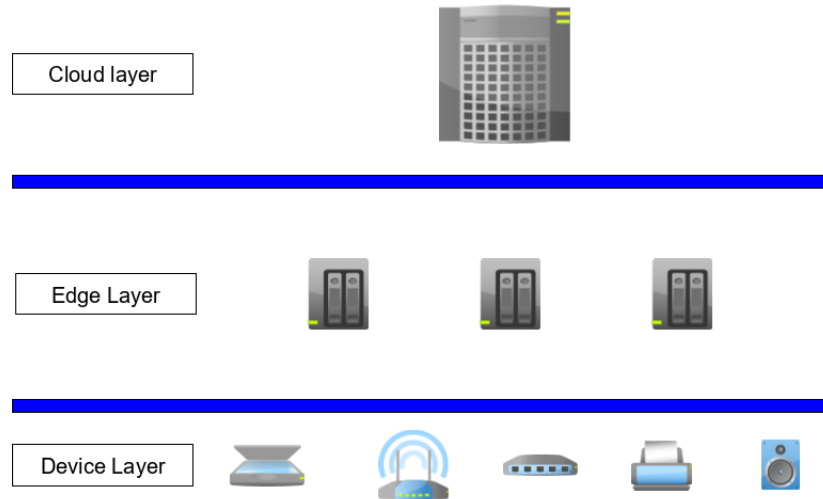


Figure 1: EdgeComputing - Architecture

enabling real-time data processing and analysis, and the device layer generating data from end-user devices and sensors.[Var+16]

2.2 Kubernetes

Kubernetes is a popular open-source platform used for managing containerized workloads and services. It was originally designed by Google and is now maintained by the Cloud Native Computing Foundation (CNCF). Kubernetes provides a robust and scalable solution for automating deployment, scaling, and management of containerized applications[Ros+21].

At a high level, Kubernetes operates by deploying and managing containerized applications using a declarative model. A user or administrator defines the desired state of the application in a Kubernetes manifest file, which specifies the desired deployment configuration, including the number of replicas, container images, networking, and storage requirements. Kubernetes then takes responsibility for ensuring that the actual state of the application matches the desired state, using a range of built-in components and mechanisms.

One of the key components of Kubernetes is the kubelet, which is responsible for managing the state of individual nodes within a Kubernetes cluster[Ros+21]. The kubelet is responsible for launching containers, monitoring their health and resource usage, and responding to failures or other events that require action. Kubernetes also includes a number of additional components, such as the Kubernetes API server, which provides a centralized control plane for managing the cluster, and the etcd database, which is used for storing cluster state and configuration data.

Kubernetes provides a number of powerful features for managing containerized applications, including:

- **Automatic scaling:** Kubernetes can automatically scale the number of replicas of a given application based on demand, ensuring that the application can handle spikes in traffic or load.

- Rolling updates: Kubernetes can perform rolling updates of containerized applications, updating them one at a time while ensuring that the application remains available and responsive.
- Self-healing: Kubernetes includes built-in mechanisms for detecting and responding to failures within the cluster, such as by automatically restarting failed containers or rescheduling workloads to other nodes.

Kubernetes also supports a wide range of networking and storage options, allowing users to configure their deployments in a highly customizable and flexible way.

Like any technology, kubernetes has advantages and disadvantages. For a better understanding, these are briefly highlighted in the next subsections.

2.2.1 Pros of using Kubernetes

- Scalability: Kubernetes enables automatic scaling of containerized applications, ensuring that they can handle increases in traffic or load without manual intervention.
- Flexibility: Kubernetes supports a wide range of containerization technologies, allowing users to run their applications on their preferred platform and easily switch between platforms as needed.[Var+16]
- Portability: Kubernetes is designed to be platform-agnostic, meaning that it can run on a variety of operating systems and cloud providers, providing flexibility in deployment options.
- Resilience: Kubernetes includes built-in mechanisms for self-healing and fault tolerance, ensuring that applications remain available and responsive even in the face of failures or disruptions.
- Ecosystem: Kubernetes has a large and growing ecosystem of tools and extensions, making it easier to integrate with other technologies and extend its functionality as needed.

2.2.2 Cons of using Kubernetes

- Complexity: Kubernetes can be complex and difficult to learn, requiring significant time and effort to become proficient in its use.
- Management overhead: Kubernetes requires ongoing management and maintenance to ensure proper operation, which can be a significant burden for smaller organizations or teams with limited resources.
- Resource-intensive: Kubernetes requires significant computational resources to operate, which can be a concern for organizations with limited computing resources or budget constraints.
- Network complexity: Kubernetes introduces additional network complexity, which can be challenging to manage and troubleshoot, particularly for organizations with limited networking expertise.

- Learning curve: The learning curve of Kubernetes can be steep and can lead to errors during implementation and deployment.
- In summary, Kubernetes is a powerful platform that provides many benefits, including scalability, flexibility, portability, resilience, and a large ecosystem. However, it also has some drawbacks, including complexity, management overhead, resource-intensiveness, network complexity, and a steep learning curve.[Men22]

2.3 Kubernetes and Edge Computing

As one can see from the former chapter, Kubernetes is a powerful tool for managing containerized workloads, but it does have some limitations when it comes to edge computing. Some of these limitations include:

- Bandwidth: Edge computing applications often require low latency and high bandwidth, which can be challenging for Kubernetes to provide in a distributed environment.
- Resource constraints: Edge devices often have limited resources, such as CPU, memory, and storage. Kubernetes may struggle to manage containers on these devices, especially if they have different architectures.
- Network connectivity: Edge devices may be deployed in remote or harsh environments with limited network connectivity, making it difficult to manage them with Kubernetes.
- Security: Edge devices may be more vulnerable to security threats than traditional data centers, and Kubernetes may not have the necessary security features to protect against these threats.
- Complexity: Kubernetes is a complex system with a steep learning curve, which can make it difficult for organizations to implement and manage in an edge computing environment.

The software "KubeEdge" promises a solution for these problems. This will be presented as the last part of the basic chapter in the following.

2.4 KubeEdge

KubeEdge is an open-source cloud-native edge computing framework that extends Kubernetes to the edge[Xio+18]. It allows developers to deploy containerized applications to edge nodes and manage them using Kubernetes tools and APIs. KubeEdge is designed to support various edge computing scenarios, including offline computing, local data processing, and real-time analysis of data streams[Xio+18].

2.4.1 KubeEdge Architecture:

KubeEdge has a two-tier architecture consisting of a cloud side and an edge side.

Cloud Side: The cloud side includes a Kubernetes cluster that manages and deploys the edge nodes. The cloud side also includes several core components:

- **CloudCore:** CloudCore is a Kubernetes controller that manages the edge nodes and the data synchronization between the cloud and edge nodes. It also provides a user interface for managing the edge nodes.
- **MQTT Broker:** KubeEdge uses the MQTT protocol to communicate between the cloud side and edge side. The MQTT broker is responsible for receiving and forwarding messages between the two sides.

Edge Side: The edge side includes lightweight runtime environments called EdgeCore, which run on the edge nodes. The EdgeCore provides the container runtime, device management, and data synchronization with the cloud side.

- **EdgeCore:** EdgeCore is responsible for managing the containerized applications running on the edge nodes. It also provides device management and data synchronization with the cloud side.
- **Device Twin:** Device Twin is a key feature of KubeEdge that allows the edge nodes to synchronize their device state with the cloud side. This ensures that the cloud side has an up-to-date view of the devices and can take appropriate actions when needed.
- **MQTT Agent:** The MQTT agent is responsible for establishing and maintaining the connection with the cloud side using the MQTT protocol.
- **Data Collection:** KubeEdge provides a data collection feature that allows the edge nodes to collect data and send it to the cloud side for analysis.

Overall, KubeEdge’s architecture provides a scalable and flexible framework for managing and deploying containerized applications to edge nodes. It extends Kubernetes to the edge, allowing developers to use familiar tools and APIs to manage edge computing resources.

3 Setting up KubeEdge

Now that the basics are clear, and the advantages and disadvantages of the various components have been discussed theoretically, they are to be tested in a practical environment. Furthermore, a manual is still missing, which should make it possible to follow the individual steps of the installation.

This chapter dedicates itself to the practical part of the project. First of all, it will be presented what exactly was planned and why this approach was chosen. Then, the respective installations, as well as the corresponding setup, will be presented in as much detail as possible, with a description of possible difficulties and further experience.

3.1 Project Plan

During the planning of the project, it quickly became apparent that Kubernetes is not a trivial topic - as already mentioned in the chapter on disadvantages, the learning curve for new users is very steep, and since this is a university project, no previous knowledge can be assumed. Therefore, the decision was made to split the project into two parts: First,

initial experience with Kubernetes should be gained in a smaller, more closed environment, before starting with the implementation of a real network. In both cases, benchmarks should be created if possible to get an idea of the resource consumption of the individual components.

3.2 Approach 1: Single Knot

The first approach is mainly for getting to know the technology itself. The following goals have been defined for this purpose:

- First, an environment needs to be set up
- Second, Minikube is to be installed
- The server part of KubeEdge is to be installed on top of this
- The latter is to be benchmarked.

These steps are described individually in more detail in the next subsections. It will also be mentioned that there were some difficulties that could not be solved due to time constraints. In particular, a lot of improvisation was necessary for benchmarking, but more on this in the corresponding section.

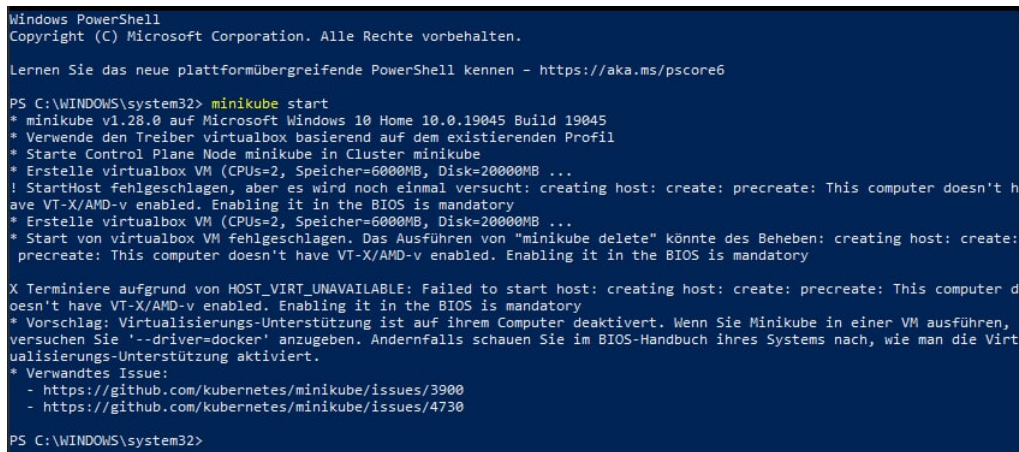
3.2.1 Preperation and Environment Setup

A Linux environment is necessary for the implementation of this project. However, such an environment was not available natively, so a virtual machine was used with the help of Oracle Virtual Box. The following settings were selected for this:

- Operating system: Ubuntu Server 64-bit
- Main memory: 50gb
- Processors: 4
- Mass storage: up to 500 gb

It is worth mentioning here that the number of processors should not be low (with one it is simply not possible to run KubeEdge, with 2-3 the manufacturers warn of performance bottlenecks), and also the main and mass storage should be set generously for the server unit - the first installation attempt failed due to lack of storage space, and changing the partitions on a virtual machine is not exactly trivial - a complete reinstallation, including a new virtual machine was the much more effective solution.

With these settings, the installation (more on this in the next subsection) could be performed relatively smoothly - however, there is also a limitation that could not be fixed, and which is a strong argument against using a virtual environment on a Windows operating system. For the full use of MiniKube, it is needed to have VT-x enabled. VT-x (for Intel processors) or AMD-v (for AMD processors) is a technology that provides hardware-level virtualization support. It allows virtual machines to run with better performance and security by allowing them to directly access the physical resources of the host system,



```

Windows PowerShell
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

Lernen Sie das neue plattformübergreifende PowerShell kennen - https://aka.ms/pscore6

PS C:\WINDOWS\system32> minikube start
* minikube v1.28.0 auf Microsoft Windows 10 Home 10.0.19045 Build 19045
* Verwende den Treiber virtualbox basierend auf dem existierenden Profil
* Starte Control Plane Node minikube in Cluster minikube
* Erstelle virtualbox VM (CPUs=2, Speicher=6000MB, Disk=20000MB ...)
! StartHost fehlgeschlagen, aber es wird noch einmal versucht: creating host: create: precreate: This computer doesn't have VT-X/AMD-v enabled. Enabling it in the BIOS is mandatory
* Erstelle virtualbox VM (CPUs=2, Speicher=6000MB, Disk=20000MB ...)
* Start von virtualbox VM fehlgeschlagen. Das Ausführen von "minikube delete" könnte das Beheben: creating host: create: precreate: This computer doesn't have VT-X/AMD-v enabled. Enabling it in the BIOS is mandatory
X Terminiere aufgrund von HOST_VIRT_UNAVAILABLE: Failed to start host: creating host: create: precreate: This computer doesn't have VT-X/AMD-v enabled. Enabling it in the BIOS is mandatory
* Vorschlag: Virtualisierungs-Unterstützung ist auf ihrem Computer deaktiviert. Wenn Sie Minikube in einer VM ausführen, versuchen Sie '--driver=docker' anzugeben. Andernfalls schauen Sie im BIOS-Handbuch ihres Systems nach, wie man die Virtualisierungs-Unterstützung aktiviert.
* Verwandtes Issue:
  - https://github.com/kubernetes/minikube/issues/3900
  - https://github.com/kubernetes/minikube/issues/4730
PS C:\WINDOWS\system32>
    
```

Figure 2: VT-X/AMD-v not enabled - a problem without a real solution

such as the CPU, memory, and I/O devices - and enabling this is only available on Windows 10 professional, which was not available. The error message can be seen in Figure 2.

A possible Solution would be the purchase of Windows 10 professional - but spending this 220 Dollars did not seem worth an investment here, especially when considering, that this first approach was mainly to get in touch with the technology. Apart from that, the basic functions seem to work without an activated VT-x, which is why it was ignored in the further course of the work - which, however, also means that any performance measurements may be slower than potentially possible.

3.2.2 Benchmarking

For the benchmarking, a TIG-Stack was the preferred approach. A TIG stack is a software stack used for monitoring and analytics of time-series data. It consists of three main components:

- **Telegraf:** Telegraf is a plugin-driven server agent that collects and reports metrics from various systems, services, and databases. It supports a wide range of input and output plugins, making it a highly flexible and customizable tool for collecting metrics.
- **InfluxDB:** InfluxDB is a time-series database that stores and queries metrics data collected by Telegraf. It provides a SQL-like query language and a rich set of APIs for managing data.
- **Grafana:** Grafana is a popular open-source platform for visualizing and analyzing time-series data. It provides a rich set of dashboards, visualizations, and alerts, and supports integration with various data sources, including InfluxDB.

Together, these three components form the TIG stack, which provides a comprehensive solution for collecting, storing, analyzing, and visualizing time-series data. The TIG stack is commonly used in DevOps, IoT, and other applications that require real-time monitoring and analytics of metrics data.

Unfortunately, it was not possible to install such a stack - although the individual components were created correctly and a lot of time was spent on troubleshooting, no benchmarks arrived in InfluxDB. Due to time constraints, another solution was chosen:

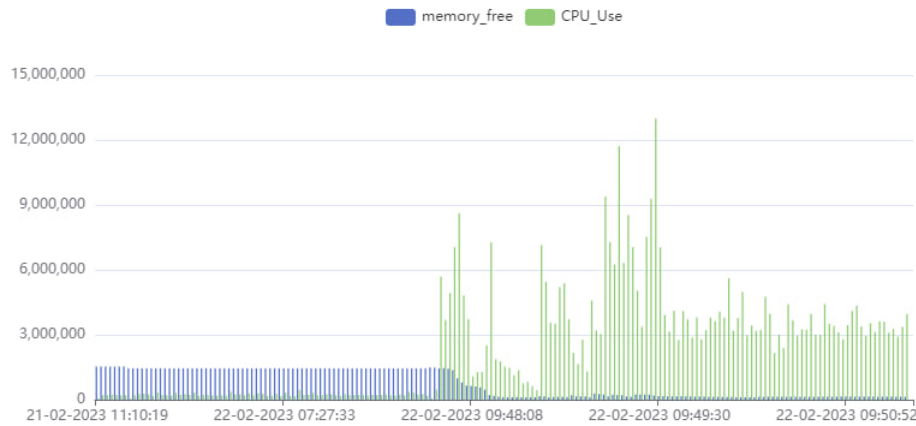


Figure 3: Ressource need of Minikube with KubeEdge

Core of the benchmark was the following command, executed on the virtual machine:

```
1 sudo nohup vmstat 0.5 604879 | (while read; do echo "$(date +%d-%m-%Y
  " "%H:%M:%S) $REPLY"; done) >> myvmstatfile.dat
```

Core of this command is the 'vmstat': This command is used to display information about the system's virtual memory, processes, CPU activity, and other performance metrics. The 0.5 parameter specifies the sampling interval in seconds (in this case, every half second), and 604879 specifies the number of samples to be taken before the command exits. Putting it all together, this command runs 'vmstat' with a sampling interval of 0.5 seconds and a total of 604879 samples, and pipes the output to a while loop that adds a timestamp to each line of output. The final output is then appended to a file named myvmstatfile.dat.

To visualize the results, a small, web-based application was developed, using the Angular Framework. The code is available on my GitHub: <https://gitlab.gwdg.de/vincenz.dumann/benchmarking-kube-edge>².

3.2.3 Performance Measurements

Now that the methods are clear, let's have a first look to the results. (Note: All result files are in the appendix and on the projects GitHub³.

In Figure 3 one can see the start free memory and the cpu-use of the KubeEdge-Main node. On the graph, you can clearly see at which point the software was started - after memory and CPU consumption initially remained constant, this increases sharply with the start of KubeEdge. The machine used here was allocated 220 MB of memory, which, as can be seen very clearly in the diagram, is also an absolute lower limit - from the start of the application it was almost completely used, and no further memory was released in the further course. This also explains, why KubeEdge won't start on a machine with lesser memory available - There is even an error message, when one is trying to start it on a machine with less, which tells exactly the minimum amount - 220mb.

Less constant than the memory consumption is the curve of the computing capacities used - these rise in waves, then level off at a relatively high level as soon as KubeEdge

²(This should be available for all members of the gwdg-gitlab-repository)

³<https://gitlab.gwdg.de/vincenz.dumann/benchmarking-kube-edge>

is running consistently. The maximum value for CPU consumption was 12986, but this was only a single value directly at the start of the application. Once the application was running, the values settled at round about 3200 - this value can be used as a base value for CPU consumption in idle state.

This first approach shows what the minimum requirements are for a core node. What is much more interesting, however, is what the side nodes need in terms of resources, as these are usually much more limited in this respect. Accordingly, the second approach will be presented in the next chapter: In this approach, not only a main node is implemented, but also corresponding edge nodes are connected, and again the performance is measured.

3.3 Approach 2: Network

This second, much more complex approach is intended to reflect a real use case of KubeEdge - there should be a MainNode where one or more EdgeNodes can register to potentially send data to this MainNode. Accordingly, it is to be expected that the setup will be significantly more complex - however, it is also to be expected that the measurement results will provide significantly more useful information about the requirements, performance and resource consumption.

In this chapter, as in the previous one, the infrastructure used is first described and then the installation is briefly discussed. Here, too, the detailed installation instructions can be found in the corresponding GitLab⁴. Finally, measurement results are presented, both for the Edge node and for the Main node.

3.3.1 Preperation and Environment Setup

Virtual machines were also used for this project - for the main node, the requirements are clearly defined here and the same settings were used as for the previous installation. For the edge nodes, on the other hand, there are no clear minimum requirements - accordingly, these virtual machines were set up with the same specifications as the main node - it seems logical that they will consume at most just as much, rather significantly less resources, and thus it should be ensured that these are sufficient - anything else would be a big surprise and would directly disqualify KubeEdge as software for edge computing. Networks where the edges are heavier than the main unit do not seem to make much sense.

In total, 4 (according to the specifications) virtual machines were created - one of them will later serve as the main node, while the others simulate the edge nodes. In addition, an installation was carried out on a private RaspberryPi 4 (8gb) which, however, only plays a subordinate role for the further course, as it was unfortunately not possible to connect to the main node - and making the computer on which this is running publicly accessible seemed somewhat risky, measured against the value of possibly obtained findings. For this RasPi, Ubuntu for RasPi was used, and the installation was identical to that on the virtual machines, which is described in the next subchapter. The specifications for this device can be looked up in more detail by the interested reader on the corresponding website.

3.3.2 Installation

The full installation guide, as said, can be found on gitlab. This subchapter summarises these in broad terms and goes into some specifics that had to be taken into account.

⁴<https://gitlab.gwdg.de/vincenz.dumann/benchmarking-kube-edge>

Basically, setting up a network with KubeEdge takes place in three phases: First, some steps are carried out on the edge nodes as well as on the main node, then the corresponding specific components are installed, before the edges are linked to the main node in the last part. The individual steps of these phases are as follows:

- Installation of docker on both machines
- Installation of Kubernetes
 - For this project, kind and kubectl were used
- Install KubeEdge
 - On both nodes, keadm needs to be installed
 - On the MainNode, cloudcore needs to be deployed
 - On the MainNode, the token can be generated now
 - On the EdgeNode, edgecore needs to be installed. Here, the generated token needs to be used

All the steps, with all needed commands, are explained in-depths on the Gitlab.

3.3.3 Benchmarking

After the installation, the existing network was tested accordingly. Here, too, another unsuccessful attempt was made to implement a TIG stack, which unfortunately was again unsuccessful but cost a lot of time. Accordingly, the other method was used again, which works, but is of course much less professional.

The following scenario was tested:

- First, KubeEdge was started on the EdgeNode to measure the idle state.
- This EdgeNode was added to the cluster by entering the corresponding ID. This step is also already mentioned in the installation description.
 - Here, measurements were taken on the edge and main nodes.

The measured results are certainly very pleasing for the developers and users of KubeEdge, but rather disappointing for the writing of this paper - there was simply no real decisive change to be found in the corresponding metrics, the resource consumption was almost linear, even repeating this step several times hardly produced any meaningful figures. However, this observation is consistent with the official test reports, and also with what other, less biased sources reveal.

Besides memory and CPU utilisation, latency is certainly an interesting metric. However, due to time and know-how constraints, it was not possible to make a really meaningful measurement. However, a study on this is presented on the official KubeEdge website, and there are other articles on this topic that are unfortunately not always publicly accessible.

In summary, one can say that KubeEdge is designed to be lightweight and resource-efficient, and its resource usage can be optimized by adjusting the configuration of various components. For example, one can adjust the number of replicas for each component, set

resource limits for each container, and adjust other configuration parameters to optimize resource usage.

In addition, the resource usage of KubeEdge can vary depending on the size of the cluster, the number of devices, the workload being run, and other factors. Therefore, it's important to monitor the resource usage of a KubeEdge deployment and adjust the configuration as needed to ensure optimal performance.

4 Conclusion

This chapter forms the conclusion of this thesis. Here, the entire thesis is first briefly summarized, and then a look back at the goals set, as well as their achievement is discussed. It concludes with an outlook on potential next steps.

4.1 Summary

The report starts with an introduction to the terms "Kubernetes", "Edge Computing", and "KubeEdge", and describes how they are related. It then examines the theoretical pros and cons of Kubernetes and provides a detailed explanation of the architecture of KubeEdge. The report also describes the setup of two practical approaches, with a basic description of the installation process. However, a related GitLab page provides a detailed description of the installation process for interested readers. Finally, the practical approaches are further investigated with a benchmarking analysis of resource consumption.

4.2 Review of Goals

This chapter takes a look at the goals and summarizes whether they were achieved.

- Theoretical analysis of Kubernetes in edge computing
 - The report includes a theoretical analysis of Kubernetes in edge computing, with relevant literature linked in the "Related Work" chapter. The motivation, the pros and cons were explained.
- Setup of basic KubeEdge on MiniKube
 - The setup was done, but the limitations of the virtual environment were an anchor during the process
- Setup of a network with multiple EdgeNodes using KubeEdge
 - The setup was done, but took way longer then expected, mainly due to smaller problems during the implementation. It was not possible to use different devices (f.e. the RasPi) due to limitations of virtual machines and security concerns.
- Benchmarking the idle state of KubeEdge on MainNode and EdgeNodes

- The benchmarking was done, with results as expected: The amount of resources is very small on the edges, and on a higher level on the main node. The original plan to use a TIG Stack for benchmarking did not work out, but a solution was found.
- Providing a how-to to install KubeEdge
 - The how-to is available on GitLab. It contains a step-by-step guide, and it also points out steps that can hide unexpected difficulties.
- Benchmarking further Tasks on the network
 - This was not completed, mainly due to time limitations, but also due to a certain limit of skill.

4.3 Outlook

With the completion of this project, the question now naturally arises as to how the insights gained from it can be used - especially, of course, in relation to the "Decice" project. The instructions created will save colleagues working on this project from some pitfalls later on, and the manual installation can be automated. In addition, the measurements generated in the course of this work confirm that KubeEdge can definitely be used, at least in terms of resource consumption.

With all the positive things that can be written about KubeEdge, one should not forget that it is not the only solution on the market. Possible alternatives include K3S ⁵, and microk8s ⁶. However, there was no more time for this in the context of this thesis; a corresponding comparison could be a question for another practical thesis, or even a bachelor's thesis, as all these solutions more or less promise the same: Easy implementation for Kubernetes in Edge Computing.

⁵K3s is a highly available, certified Kubernetes distribution designed for production workloads in unattended, resource-constrained, remote locations or inside IoT appliances, quoted from www.k3s.io)

⁶MicroK8s is a lightweight, fast and simple Kubernetes distribution developed by Canonical, the company behind Ubuntu Linux. It is designed to run on a single machine, and provides all the components needed to deploy, manage and scale containerized applications in a Kubernetes environment, quoted from www.microk8s.io

References

- [Cao+20] Keyan Cao et al. “An Overview on Edge Computing Research”. In: *IEEE Access* (May 2020). URL: <https://ieeexplore.ieee.org/abstract/document/9083958>.
- [DECnd] DECICE. *About DECICE*. Website. Retrieved March 28, 2023, from <https://decice.eu/about/>. n.d.
- [Kau+19] Kuljeet Kaur et al. “KEIDS: Kubernetes-Based Energy and Interference Driven Scheduler for Industrial IoT in Edge-Cloud Ecosystem”. In: *IEEE Access* (2019). URL: <https://ieeexplore.ieee.org/abstract/document/8825476>.
- [Men22] Sergio Mendez. *Edge Computing Systems with Kubernetes*. 2022.
- [Ros+21] Josh Rosso et al. *Production Kubernetes - Building Successful Application Platforms*. 2021. URL: <https://books.google.de/books?hl=de&lr=&id=WrlEAAAQBAJ&oi=fnd&pg=PP1&dq=kubernetes+pro+and+cons&ots=Donah3qAR4&sig=zo8hPqmonKeXJCotTILS7KfjBZI#v=onepage&q=kubernetes%20pro%20and%20cons&f=false>.
- [Var+16] Blessen Varghese et al. “Challenges and Opportunities in Edge Computing”. In: *IEEE Cloud Computing* (Nov. 2016). URL: https://ieeexplore.ieee.org/abstract/document/7796149?casa_token=po0r7rl1LJsAAAAA:DsdxhNwPL0p7zC-o-GBVrDqeIbGN98kB8qLT6DbGnips4XG5B4DG38r3yyBut5reS1in2mvy5b.
- [Xio+18] Ying Xiong et al. “Extend Cloud to Edge with KubeEdge”. In: *IEEE Cloud Computing* (Oct. 2018). URL: <https://ieeexplore.ieee.org/abstract/document/8491077>.

```

21-02-2023 11:10:19 procs -----memory----- --swap-- --io-- --system-- --cpu-----
21-02-2023 11:10:19 r b swpd free buff cache si so bi bo in cs us sy id wa st
21-02-2023 11:10:19 0 0 0 1524788 159664 2623428 0 0 9 12 72 58 0 0 100 0 0
21-02-2023 11:10:29 0 0 0 1524284 159688 2623436 0 0 0 10 258 213 0 0 100 0 0
21-02-2023 11:10:39 2 0 0 1524284 159712 2623444 0 0 0 9 252 211 0 0 100 0 0
21-02-2023 11:10:49 4 0 0 1523952 159736 2623448 0 0 0 5 305 231 0 0 100 0 0
21-02-2023 11:11:00 1 0 0 1523952 159760 2623428 0 0 0 10 262 207 0 0 100 0 0
21-02-2023 11:11:10 0 0 0 1523704 159776 2623444 0 0 0 8 278 187 0 0 100 0 0
21-02-2023 11:11:20 0 0 0 1523704 159800 2623448 0 0 0 6 280 203 0 0 100 0 0
22-02-2023 07:26:58 procs -----memory----- --swap-- --io-- --system-- --cpu-----
22-02-2023 07:26:58 r b swpd free buff cache si so bi bo in cs us sy id wa st
22-02-2023 07:26:58 1 0 0 1448300 187556 2621612 0 0 7 10 71 57 0 0 100 0 0
22-02-2023 07:26:59 0 0 0 1448048 187556 2621620 0 0 0 4 309 197 0 0 100 0 0
22-02-2023 07:27:00 1 0 0 1448048 187556 2621620 0 0 0 0 326 255 0 0 99 0 0
22-02-2023 07:27:01 0 0 0 1448048 187556 2621620 0 0 0 0 312 269 0 0 100 0 0
22-02-2023 07:27:02 0 0 0 1448048 187564 2621624 0 0 0 32 299 239 0 0 100 0 0
22-02-2023 07:27:03 1 0 0 1447796 187564 2621624 0 0 0 0 260 171 0 0 100 0 0
22-02-2023 07:27:04 1 0 0 1447796 187572 2621624 0 0 0 20 283 340 0 0 100 0 0
22-02-2023 07:27:05 0 0 0 1447796 187572 2621624 0 0 0 0 359 222 0 0 100 0 0
22-02-2023 07:27:06 0 0 0 1447796 187572 2621624 0 0 0 0 300 203 0 0 100 0 0
22-02-2023 07:27:07 0 0 0 1447796 187572 2621624 0 0 0 0 270 183 0 0 100 0 0
22-02-2023 07:27:08 0 0 0 1447796 187572 2621624 0 0 0 0 314 308 0 0 100 0 0
22-02-2023 07:27:09 0 0 0 1447796 187580 2621624 0 0 0 24 350 221 0 1 100 0 0
22-02-2023 07:27:10 0 0 0 1447796 187580 2621624 0 0 0 4 368 247 0 0 100 0 0
22-02-2023 07:27:11 0 0 0 1447796 187580 2621624 0 0 0 0 359 215 0 0 100 0 0
22-02-2023 07:27:12 0 0 0 1447796 187588 2621624 0 0 0 32 358 313 0 0 99 0 0
22-02-2023 07:27:13 0 0 0 1447796 187588 2621628 0 0 0 0 303 176 0 0 100 0 0
22-02-2023 07:27:14 0 0 0 1447796 187596 2621620 0 0 0 16 248 223 0 0 100 0 0
22-02-2023 07:27:15 0 0 0 1447796 187596 2621628 0 0 0 0 297 201 0 0 100 0 0
22-02-2023 07:27:16 0 0 0 1447796 187596 2621628 0 0 0 4 334 185 0 0 100 0 0
22-02-2023 07:27:17 0 0 0 1447796 187596 2621628 0 0 0 0 244 179 0 0 100 0 0
22-02-2023 07:27:18 1 0 0 1447796 187596 2621628 0 0 0 0 232 197 0 0 100 0 0
22-02-2023 07:27:19 procs -----memory----- --swap-- --io-- --system-- --cpu-----
22-02-2023 07:27:19 r b swpd free buff cache si so bi bo in cs us sy id wa st
22-02-2023 07:27:20 2 0 0 1447796 187596 2621628 0 0 0 0 228 179 0 0 100 0 0
22-02-2023 07:27:21 0 0 0 1447796 187604 2621628 0 0 0 16 376 362 1 0 99 0 0
22-02-2023 07:27:22 0 0 0 1447796 187604 2621628 0 0 0 0 286 237 0 0 100 0 0
22-02-2023 07:27:23 0 0 0 1447796 187612 2621632 0 0 0 36 371 240 0 0 100 0 0
22-02-2023 07:27:24 0 0 0 1447796 187612 2621636 0 0 0 0 222 204 0 0 100 0 0
22-02-2023 07:27:25 0 0 0 1447796 187612 2621636 0 0 0 16 270 289 0 1 99 0 0
22-02-2023 07:27:26 0 0 0 1447796 187620 2621640 0 0 0 16 222 195 0 0 100 0 0
22-02-2023 07:27:27 0 0 0 1447796 187620 2621640 0 0 0 0 314 292 0 0 100 0 0
22-02-2023 07:27:28 0 0 0 1447796 187620 2621640 0 0 0 0 281 265 0 0 100 0 0
22-02-2023 07:27:29 0 0 0 1447796 187620 2621640 0 0 0 8 273 181 0 0 100 0 0
22-02-2023 07:27:30 0 0 0 1447796 187620 2621640 0 0 0 0 278 183 0 0 100 0 0
22-02-2023 07:27:31 0 0 0 1447796 187628 2621632 0 0 0 24 284 275 0 0 100 0 0
22-02-2023 07:27:32 0 0 0 1447796 187628 2621640 0 0 0 0 224 171 0 0 100 0 0
22-02-2023 07:27:33 0 0 0 1447796 187636 2621636 0 0 0 32 300 317 0 0 99 0 0
22-02-2023 07:27:34 0 0 0 1447796 187636 2621644 0 0 0 0 306 174 0 0 100 0 0
22-02-2023 07:27:35 0 0 0 1447796 187636 2621644 0 0 0 4 273 161 0 0 100 0 0
22-02-2023 07:27:36 0 0 0 1447796 187636 2621644 0 0 0 12 487 446 0 1 99 0 0
22-02-2023 07:27:37 2 0 0 1447548 187644 2621644 0 0 0 16 262 215 0 0 100 0 0
22-02-2023 07:27:38 3 0 0 1447548 187644 2621644 0 0 0 0 220 224 0 0 100 0 0
22-02-2023 07:27:39 0 0 0 1447548 187644 2621644 0 0 0 0 374 312 0 0 100 0 0
    
```

Figure 4: Example Log File

A Log File

In Figure 5 one can see an example of a Log File from the Benchmarking. More examples are on the Gitlab.

B Misc

B.1 The abstract in a lyrical form

So, as ChatGPT is THE thing right now, i copied the abstract into the chat and asked for a transformation in a lyrical form, and to add in some drama. Not all informations are represented correctly, but i thought, the result is just to hilarious, to not show it here.

Benchmarking KubeEdge

A study on KubeEdge we pursue
 To assess if it's suited for edge computing too
 The stakes are high, the pressure's on
 Will KubeEdge deliver, or will it be gone?

KubeEdge is software for Kubernetes, you see
 To enable edge computing, it's designed to be

But can it handle the edge, with all its demands?

We put it to the test, with our bare hands

Two networks we install to measure its might

To determine if KubeEdge is worth the resource it might incite

The clock is ticking, the tension is high

Will KubeEdge pass, or will it die?

We start with a theoretical view

Of edge computing's pros and cons, old and new

But now it's time to see, can KubeEdge stand tall

Or will it crumble, and let us fall?

We detail the software's installation

And move on to the benchmarking operation

Our eyes are glued to the screen, our hearts beating fast

Will KubeEdge be the winner, or end up last?

We test its latency, we examine its speed

And note its reliability, a key factor indeed

The results are in, and we're glad to say

KubeEdge has passed the test, and made our day

For Kubernetes in edge computing, it's fit

And its resource consumption is really quite a bit

We breathe a sigh of relief, the drama is done

KubeEdge has won, and we've had our fun

Overall, KubeEdge is a fine choice to make

For all sorts of edge computing tasks it can take

Our study concludes, and we hope you'll see

KubeEdge's potential for edge computing, just like we.

Source: ChatGPT, Version 3, <https://chat.openai.com/chat>. Generated at the 23rd of March, 2023. As Input parameter, the abstract of this report was used, and the questions were: "Can you form the following text into a poem and add some basic information about KubeEdge to it, and spice it up with some Drama" [Abstract]

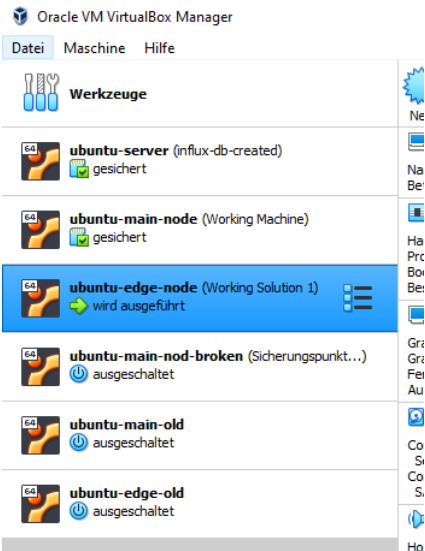


Figure 5: A total amount of 21 VMs were created for that project