



UMEÅ UNIVERSITY

# An evaluation of edge deployment models for Kubernetes

*Robin Sörensen*

**Robin Sörensen**

Spring 2023

Degree Project in Computing Science Engineering, 30 ECTS

Supervisor: Ola Ringdahl

Extern Supervisor: Ahmed Ali-Eldin & Ayoub Ed-Dafali

Examiner: Henrik Björklund & Michael Minock

M.Sc.Eng in Computing Science, 300 ECTS

## **Abstract**

With organisations moving away from in-house servers but still requiring low latency, high security and stability, edge nodes make their appearance as the best alternative. However, it is unclear which deployment method to use on those systems. This study explored the benefits and disadvantages of the multi-cluster and remote worker approach. Two lightweight Kubernetes distributions were also included in determining the impact of the deployment method compared to using different Kubernetes distributions. The remote worker approach used the lightweight K3s, while Microk8s and K3s were tested on the multi-cluster. The remote worker approach showed lower power consumption and performed better in a file transfer. A CPU-heavy task showed k3s multi-cluster approach setups outperformed the remote worker approach while using Microk3s performed similarly to the remote worker with the same number of edge nodes. Determining which methods to use should be determined based on the use case, as the two methods got different benefits in regard to security and stability.

# Acknowledgements

I want to thank Elasticsys for allowing me to do this master's thesis and for providing office space with many competent employees to ask for help whenever I got stuck. A big thank you to Ahmed Ali-Eldin and Ayoub Ed-Dafali for all the help setting up the test environments and figuring out the best approach to this project. I would also like to thank Ola Ringdahl, my internal university supervisor, for his swift and precise support and feedback on the report. Thanks to my dear friend, desk mate and workout partner André Lundqvist for his ability to make me laugh and for our daily discussions. Finally, thanks to Tomas Forsman and Mattias Åsander for setting up all the hardware on short notice.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Aims &amp; Objectives</b>	<b>3</b>
<b>3</b>	<b>Background</b>	<b>4</b>
3.1	Cloud	4
3.2	Fog computing	4
3.3	Edge nodes	5
3.4	Kubernetes	5
3.4.1	Containers	5
3.5	Clusters	6
3.6	Deployment Method	6
3.6.1	Multi-cluster Approach	6
3.6.2	Remote Worker Approach	7
<b>4</b>	<b>Related Work</b>	<b>10</b>
<b>5</b>	<b>Method</b>	<b>11</b>
5.1	System Description	11
5.1.1	Hardware	11
5.1.2	Orchestration Tool	11
5.1.3	Lightweight Kubernetes	11
5.1.4	Locust	12
5.2	Tests	12
5.2.1	Throughput Test	13
5.2.2	CPU-heavy test	13
5.3	Analysis	14
5.3.1	Student t-test	14

<b>6</b>	<b>Results</b>	<b>15</b>
6.1	Power Consumption	15
6.2	Throughput	17
6.3	CPU-heavy test	19
6.4	Hardware Utilization	20
<b>7</b>	<b>Discussion</b>	<b>22</b>
7.1	Performance differences	22
7.1.1	CPU-heavy	22
7.1.2	Throughput	23
7.2	Power Consumption	24
7.3	Hardware Utilization	24
<b>8</b>	<b>Future Work</b>	<b>25</b>
<b>9</b>	<b>Conclusion</b>	<b>26</b>
	<b>References</b>	<b>27</b>
<b>A</b>	<b>Appendix</b>	<b>29</b>
A.1	Appendix: Code used in testing	29
A.2	Appendix: K8s deployment file	30

# 1 Introduction

Cloud computing has exploded in recent years as organisations move away from in-house servers to use the cloud for scalability and stability. Moving away from in-house servers adds challenges, such as security, latency and instability in bandwidth. To combat these challenges, edge computing was introduced and has, in recent years, become very popular. Edge computing systems place computing resources at the network’s edge, close to applications that require higher processing powers than those available on the device running them. These systems have been envisioned to enable the next revolution in digital services, from fully autonomous vehicles and smart farming to edge-based prosthetic devices; in this promised future, applications not viable using today’s infrastructure will be realised. Edge systems will deploy various hardware, including GPUs, TPUs, neural accelerators, and traditional CPUs. From an application point of view, common features between edge applications include the extreme amount of data they produce, the mobility of the devices producing this data, and in many cases, the need for distributed intelligence that can guide and control these devices. Hence, for these applications to reach their potential, both the resources and the applications must be managed to achieve the required Quality-of-Service (QoS) and latency.

Kubernetes, abbreviated as K8s, a well-established platform for hosting microservices, has been suggested as a possible solution for edge resource orchestration [7]. For example, during KubeCon Europe 2021, a use-case of monitoring the longest sea crossing in the world, the 55-kilometre-long Hong Kong–Zhuhai–Macao bridge was shown where KubeEdge (a K8s variant) was used for monitoring the bridge and the massive amount of IoT devices required. To create a monitoring system for the bridge, a combination of technologies, including 5G communication, positioning solutions, and KubeEdge, were deployed. The edge devices identify issues and monitor the bridge’s safety; many sensors are deployed to monitor temperature, vibrations and noise, for example. Combined with positioning data, these identify where to send help in an emergency [16].

While promising, it is unclear what deployment models should be used in these systems. For the management of nodes, a control plane (earlier called a master node, changed for clarity but mainly the cultural undertone) is deployed. The control plane provides fault tolerance, high availability and load balancing. A deployment model defines where the control plane runs w.r.t. the nodes in the system and how much autonomy each edge site should have [4].

The two possible deployment methods evaluated in this project are the multi-cluster approach, see Section 3.6.1 and the remote worker approach, see Section 3.6.2. The benefit of multi-cluster is the autonomy each edge node gets by having a control plane on each node. Even if the central control plane fails, the node can continue working. The benefit of the

remote worker approach is the elimination of the overhead needed to run a control plane on each node. This could benefit edge nodes like Raspberry Pi's and similar low-performance nodes. However, the remote worker approach needs a stable internet connection as all workloads are distributed by the central control plane, compared to the multi-cluster approach, which accepts work directly from users.

Another important aspect is the power consumption difference in the deployment methods. Edge nodes might have an unreliable or limited power supply. In a network of thousands of edge nodes, the issue of supplying power and the costs must be considered. Currently, the average resource utilization of a cloud server is around 30% making the power consumption in idle critical [2].

## 2 Aims & Objectives

In this thesis, the aim is to investigate how deployment models of Kubernetes over the edge perform for different edge applications and their respective performance overheads. This would enable organisations to decide which deployment method suits them best. This is, however, not a simple question as challenges emerge, such as bandwidth restraints, unpredictable connectivity, latency and the price of electricity.

Aims and Objectives for this project:

- O1** Identify available lightweight Kubernetes distributions and select relevant distributions to include in evaluation based on popularity, key differences and currently still supported and maintained.
- A1** Evaluate performance differences between lightweight kubernetes distributions.
  - O2** Install distribution on edge nodes and install test service.
  - O3** Create and run performance tests on edge nodes.
- A2** Evaluate the performance impact of different degrees of autonomy in edge nodes.
  - O4** Run tests with limited autonomy *remote worker approach* Section 3.6.2 and with full autonomy as a *full fledged cluster* Section 3.6.1. This will be in combination with **O2** and **O3** to cover all possible configurations.
- A3** Evaluate energy consumption differences between the deployment models.
  - O5** Measure energy consumption when edge nodes are idle and throughout the tests.

## **3 Background**

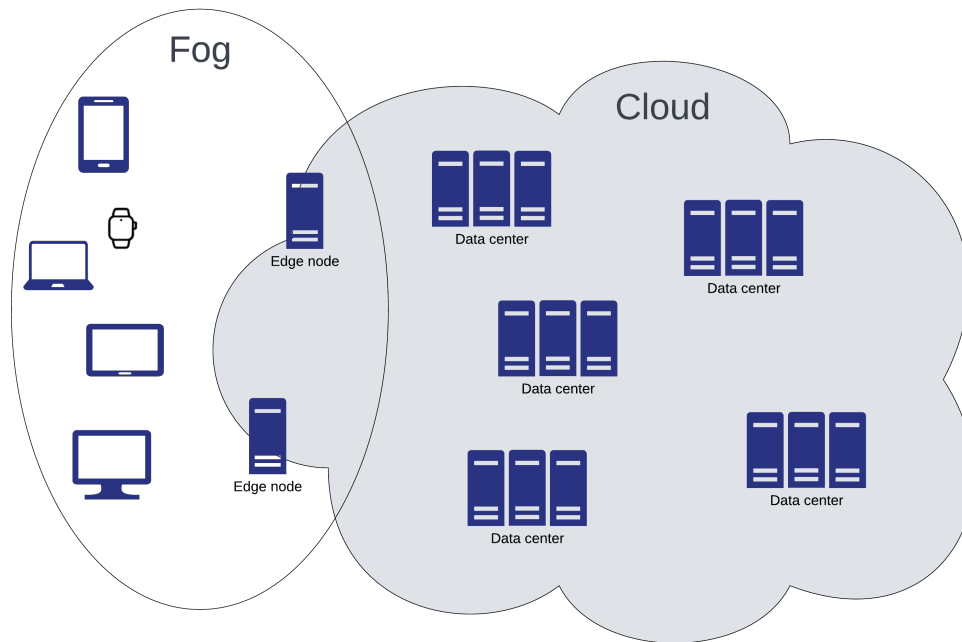
With cloud and edge come many new terms, concepts, and theories. Many terms and concepts lack consensus in their use and definitions. In this chapter, the most important of those will be explained.

### **3.1 Cloud**

The definition of cloud is unclear, but it is essentially servers accessible over the internet. This includes both the software and databases that run on the servers [8]. The servers can be designed for various reasons, a few common and widely used services are video streaming, webmail and data storage.

### **3.2 Fog computing**

Fog computing is the term used to describe the architecture of using edge nodes and edge devices for computation. Cisco coined the term in 2014, so it is pretty new to the general public [1]. Fog computing is used interchangeably with edge computing. Fog covers not only edge nodes but also edge devices that send data but do not do any computation other than for the edge device user, see Figure 1.



**Figure 1:** Overview image of difference between edge, cloud and fog. Shows edge nodes, data centres and edge devices such as smart watches, laptops, and smartphones.

### 3.3 Edge nodes

An edge node is a virtual or physical device near a user’s location or a data source, such as the edge nodes on the Hong Kong-Zhuhai-Macao bridge. An example of when an edge node is beneficial is when users in an area are trying to access the resource from a server. The resources can be buffered on an edge node close to the users to reduce the latency, and the traffic on the servers can be reduced as the users only need to connect to the edge node. The primary motive for using edge nodes is reduced latency and bandwidth and the possibility for better security [14]. Edge nodes are sometimes called fog nodes [12]. In this report, we will use the term edge nodes.

### 3.4 Kubernetes

Kubernetes is an open-source container orchestration system created by Google, whose task is automating deployment, scaling and management. Kubernetes version 1 was released in 2015 [6].

#### 3.4.1 Containers

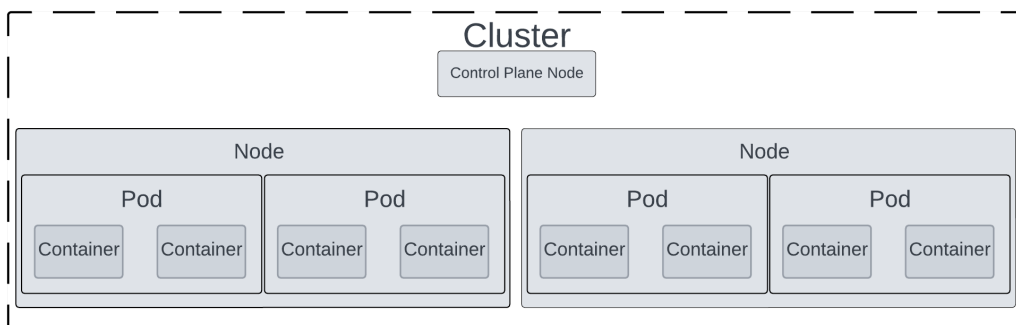
A container can be viewed as a small application with a bunch of restrictions applied to it. A container includes everything needed for the application to run and is decoupled from the operating system for easier deployment on different operating systems or clouds [5].

Containers run in pods, and a pod can run multiple containers where all the containers in a pod share resources, see Figure 2.

### 3.5 Clusters

In a traditional cloud environment, a cluster is a group of nodes usually hosted on a virtual machine <sup>1</sup>. All nodes in a cluster work together, acting like a single machine. The main advantage of clusters is to run containers over multiple devices [10].

Clusters are used because they provide load-balancing and high availability. The high availability is provided by the redundant nodes maintained in a cluster. When a node fails, another node will take over the task the failed node was performing [11].



**Figure 2:** Diagram overview of the structure of Kubernetes: Shows multiple nodes in a cluster, multiple pods in a node and finally multiple containers in a pod

### 3.6 Deployment Method

A deployment method is chosen for deploying, updating and, in other ways, managing nodes in a cluster. In the traditional cluster, the nodes are not responsible for managing themselves; instead, a control plane is responsible. This report analyzes the multi-cluster approach and remote worker approach.

#### 3.6.1 Multi-cluster Approach

A multi-cluster approach is when all edge nodes run a complete cluster with control plane nodes and worker nodes, see Figure 3. This approach can also be called the full-fledged cluster approach [4]. The overhead of running a control plane on the cluster might impact the node's performance as the control plane needs to manage the nodes on the cluster. Managing nodes include but are not limited to scaling the number of nodes in each node, scheduling work and monitoring performance metrics. When running multiple autonomous clusters, a deployment method could be used to manage the clusters.

There are many reasons why a multi-cluster approach could be advantageous. One reason is when a company have multiple development teams, as Kubernetes is not a multi-tenant

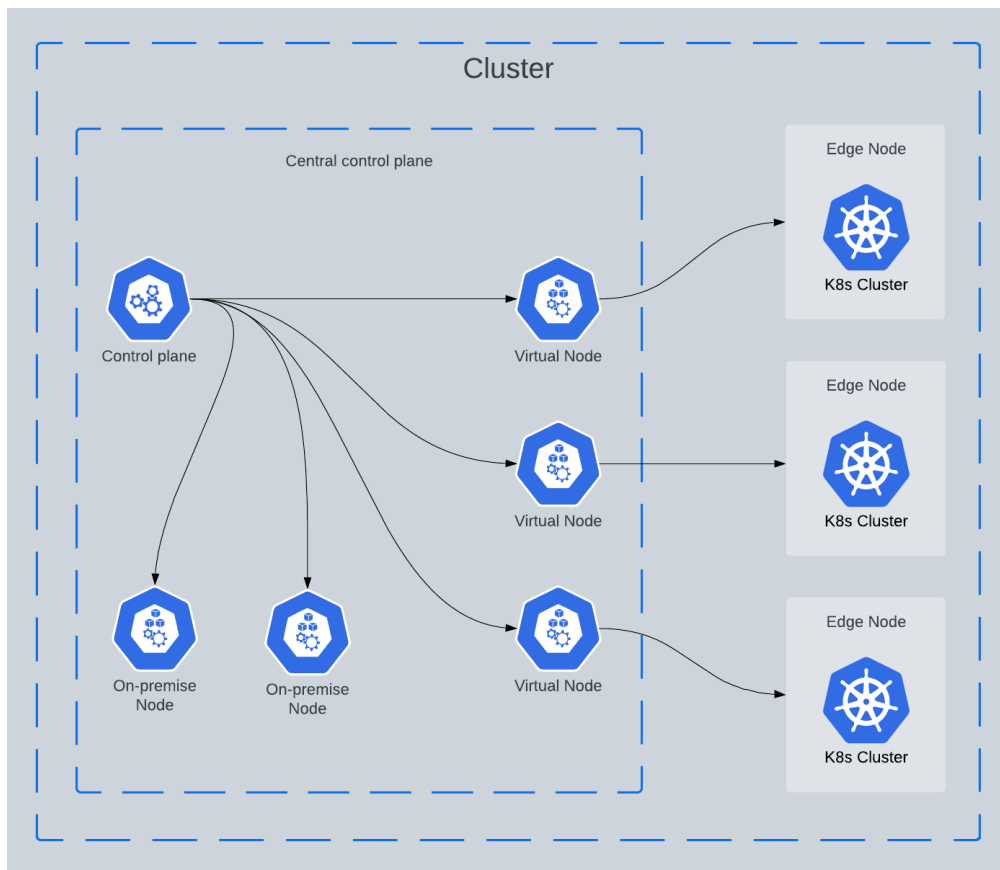
<sup>1</sup><https://kubernetes.io/docs/concepts/overview/components/>

architecture. Namespaces can be used to get isolation to some degree, but Kubernetes does not allow for complete isolation.

Having multiple clusters but no tool to manage them can be a nightmare, and this is where multi-cluster orchestration tools appear.

A multi-cluster orchestration tool is used to simplify the deployment, updating or removal of services to multiple clusters. Orchestration tools can also support features like traffic scheduling and failover. Failover in a multi-cluster environment is handled differently depending on the tool used. For example, in case of an edge node failure, some tools will re-deploy the services on another edge node, while other tools will re-route traffic to other nodes with the same service deployed.

Other reasons to use multiple clusters are, e.g. geographic or country-specific regulations, availability and performance [9]. All of the orchestration tools in this thesis support dynamic joining and removal of clusters adding a simple vertical scaling mechanism.



**Figure 3:** Diagram overview of the full-fledged cluster approach with edge nodes and on-premise nodes

### 3.6.2 Remote Worker Approach

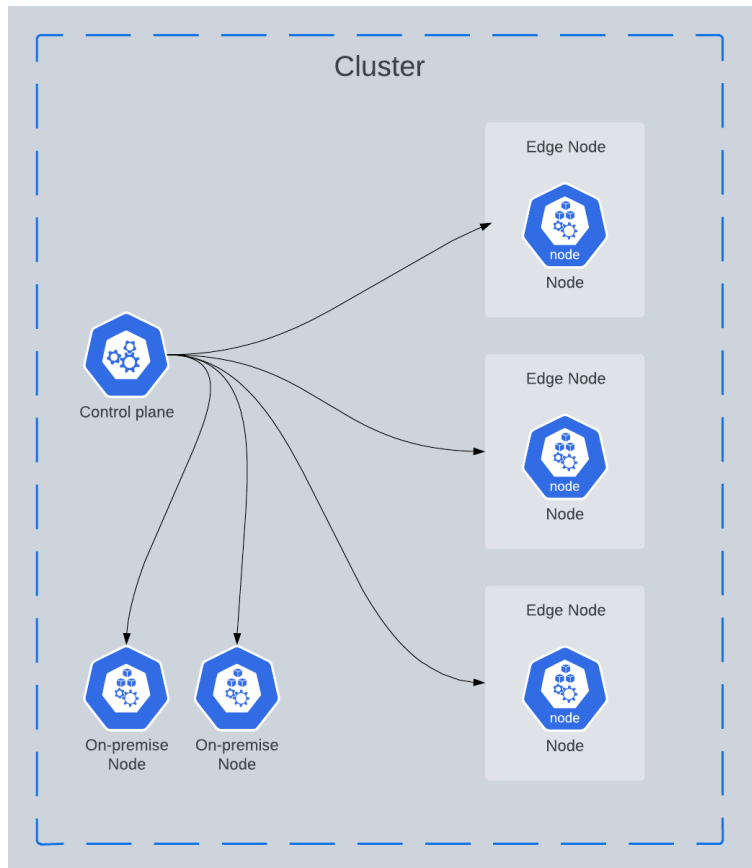
The remote worker approach runs edge nodes as remote workers but still needs a central control plane to orchestrate. The edge nodes do nothing but work the control plane dis-

tributed to them. The central control plane handles the responsibilities mentioned in Section 3.6.1.

In Figure 4, a high-level overview of the remote worker approach is shown. A worker node does not do anything in regard to management. If a pod or a container in the node fails, the central control plane is responsible for restarting the pod or container. With this approach, there is also a possibility that the node is seen as unresponsive or has failed and will not be distributed any work if the latency between the node and the control plane is high. This could result in an edge node not being utilized even though it is alive. In Kubernetes, `node-status-update-frequency` is the required frequency a node needs to update the control plane to schedule tasks to the node. The frequency can be changed if the latency is expected to be high, but with the tradeoff that functions can be scheduled for failed nodes.

This approach reduces the load on edge nodes by eliminating the overhead of running a control plane on each edge node. In a network of hundreds or thousands of edge nodes, the performance gained from not running a control plane on each edge node could be large and still uphold the QoS.

Using this approach does require work to be scheduled by the central control plane, and all traffic must go through the control plane to be distributed. Therefore, this method will add more network traffic and makes the control plane a single point of failure. In addition, the need to connect to the central control plane could add latency and makes the service more vulnerable. Finally, the service is unusable if the control plane fails or communication is delayed too much. In short, this approach might not be feasible in an environment with unpredictable connectivity, unreasonable high latency or lack of necessary bandwidth for the service.



**Figure 4:** Diagram overview of the remote worker approach with edge nodes and on-premise nodes

## 4 Related Work

There is a moderate amount of research related to performance evaluations of lightweight Kubernetes distributions. The research found compares at most two different types of lightweight Kubernetes distributions but there was no comparison between running these with a central control plane and with full autonomy. The scope of the research found is not very broad. The hardware differs between the research and is therefore hard to compare to each other.

In 2019 a research team compared KubeEdge and K3s, two lightweight Kubernetes versions on a Raspberry Pi 3 B+ [3]. The experiments ran for a week on a delivery truck with a service providing the GPS location of the truck. The results showed a similar RAM usage but KubeEdge needed 20% more CPU resources than K3s under load. In idle both used similar amounts of resources. The team also mention K3s binary file is only 40 MB.

2021 a team of researchers at the National Technical University of Ukraine and Igor Sikorsky Kyiv Polytechnic Institute compared lightweight distributions of Kubernetes [13]. Their comparison included the original Kubernetes, MicroKubernetes and K3s. The experiment ran on virtual machines with 2 vCPU and 8 GB of RAM on Google cloud. In comparison, the Raspberry Pi 3 b+ has 4 vCPU and 1 GB of RAM. The authors concluded that Kubernetes performed better than both lightweight versions except for K3s which had the best disk utilization. They end their conclusion by writing the differences might be neglected in real-world examples and that Kubernetes requires a lot of resources by default which might let the lightweight versions be applicable on limited-resource devices.

# 5 Method

The experiment in this project will cover the aims **A1**, **A2** and **A3**.

## 5.1 System Description

This section describes the test environment, the lightweight distributions, the orchestration tools used and the tests conducted.

### 5.1.1 Hardware

The edge nodes and the central control plane existed of five Raspberry Pi 4 with 8 GB RAM and a 128GB SSD for storage. The edge nodes were connected to an intelligent switch providing power via ethernet cables. A traffic control filter [15] was used to simulate latencies. An intelligent switch fetched power consumption data from the edge nodes every 15 seconds.

Umeå University provided the hardware for this project.

### 5.1.2 Orchestration Tool

#### Liqo

Liqo does not include a failover policy but recommends using K8gb. K8gb can provide load balancing among clusters using round robin or favouring closer endpoints through the GeoIP strategy<sup>1</sup> and adopt a failover policy<sup>2</sup>. In addition, Liqo supports different types of cloud providers allowing AWS, GKE, and other cloud provider clusters to be added as edge nodes to a Liqo multi-cluster deployment. Liqo offers automatic peering and cluster-to-cluster communication.

### 5.1.3 Lightweight Kubernetes

The experiment includes two different lightweight Kubernetes distributions. K3s and Microk8s were included to compare a distribution's performance impact on the various deployment methods. They were chosen by their popularity, low hardware requirements, and the fact that both of them support arm64 architecture. See Table 1 for the distributions hardware minimum and recommended requirements. K3s is packaged as a <70 MB binary file allowing for installations on devices without a lot of storage. Microk8s can run on arm64

---

<sup>1</sup>Determining the geographic location using an IP address

<sup>2</sup>See <https://www.k8gb.io/docs/> for a more detailed explanation of what K8gb offers

architecture but not arm32, making it less versatile than K3s. Microk8s allows for fast and easy installations of add-ons such as DNS, Load balancing, dashboard and community repositories with simple commands.

**Table 1 Minimum and recommended requirements for lightweight Kubernetes distributions**

Requirements	CPU Minimum	CPU Recommended	RAM Minimum	RAM Recommended	Storage Minimum	Storage Recommended
K3s	1 vCPU	-	512 MB	1 GB	200 MB	-
MicroK8s	1 vCPU	-	540 MB	4 GB	-	20 GB

K3s<sup>3</sup> uses an upstream, open-source version of Kubernetes. K3s is packaged as a tiny binary file of 50 MB.

#### 5.1.4 Locust

Locust<sup>4</sup> is a load testing tool able to run on arm32 architecture. Creating tests for Locust only requires Python code; it can run distributed over multiple machines. Locust produces resulting statistics as a .csv file.

## 5.2 Tests

To cover aim A1, Microk8s and k3s, two lightweight Kubernetes distributions were included in the tests. All tests created to cover A2 and A3 were tested using all lightweight distributions to evaluate if the distribution impacted performance. Evaluating the average and 95th percentile of the metrics collected determines the impact of k8s distribution. To cover the aim, A2 tests were run in a multi-cluster environment using a multi-cluster orchestration tool and comparing it to a traditional one-cluster environment running the same tests.

K8gb and round robin were used for load balancing in the multi-cluster environment, as seen in the deployment file in Appendix A.2. K8gb was not configured in any other way. The load balancing in the remote worker approach used the provided service load balancer, formerly called Klipper LoadBalancer<sup>5</sup> is responsible for the balancing.

The metrics were collected over time with experiments of constant, variable, long-running, and short-running tasks to simulate different real-world examples.

Some metrics were collected by installing and running the different configurations with idle edge nodes. The metrics collected from edge nodes in idle are RAM and CPU usage and energy consumption.

Appendix A.1 shows the code for both services used for testing, the deployment code and the code used by Locust.

<sup>3</sup><https://k3s.io/>

<sup>4</sup><https://locust.io/>

<sup>5</sup><https://docs.k3s.io/networking#service-load-balancer>

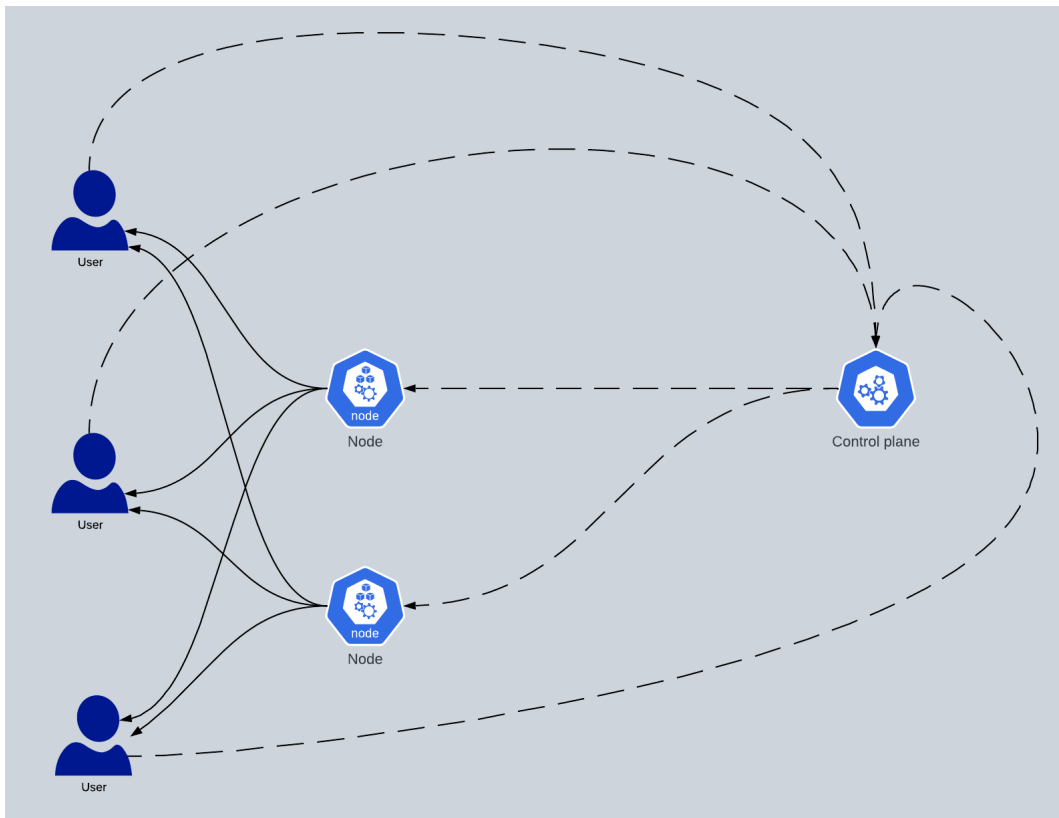
### 5.2.1 Throughput Test

The throughput test was done by simulating multiple users downloading a file from the raspberries. A video file of 17 MB was downloaded to the edge nodes; this was the file a user would download. In the test, Locust simulates the different amounts of users downloading the file simultaneously. The test ran for 2 minutes, and the number of finished downloads was multiplied by 17 and divided by 120 to get the result in MB/s. The test was repeated ten times to avoid outliers affecting the final results.

To simulate real-life environments, latency was added to the control plane and the remote worker, simulating a control far away from the user. In Figure 5, the dotted lines are the paths that could be impacted by high latency in a real-life use case, which the added latency aims to mimic. For example, in the multi-cluster case, latency was added to the first node, which received a work request from a user. Tests are conducted with multiple latencies to cover as many cases as possible.

### 5.2.2 CPU-heavy test

Due to the difference in placement in the two deployment methods, the CPU-heavy test was conducted with latency as mentioned in Section 5.2.1 with the addition of latency only put on the control plane in the remote worker approach.



**Figure 5:** Flowchart displaying dataflow. Latency was added on traffic in dotted lines.

## 5.3 Analysis

The resulting metrics gathered from the tests were compared to find performance differences. These were then analyzed and matched with the configurations and critical differences in the lightweight Kubernetes distributions.

### 5.3.1 Student t-test

To determine if there is a significant difference between the power consumption, a student t-test, also called a t-test, was used. A two-sample t-test can test whether the means of two populations are equal and were chosen for that attribute. The test was used to compare the power consumption in the idle state between the edge nodes. The student t-test requires the mean of the samples to follow a normal distribution curve. In addition to the t-test, the *Mann-Whitney U* was used to confirm the significance as it does not require an assumption of normal distribution.

## 6 Results

This section contains the results from the experiments mentioned in Section 5. In addition, the metrics collected are presented in graph form to compare the different implementations easily.

Boxplots are used to show the spread of data in power consumption. Lastly, a couple of screenshots show the hardware utilization in idle.

### 6.1 Power Consumption

The power consumption of the hardware over time is shown in Figures 6, 7, 8 and a boxplot exposing the difference in outliers between the deployment methods in Figure 9. As seen in the Boxplot and the timeline, the remote worker approach power consumption does not fluctuate as much as the multi-cluster approach. The average power consumption for the edge nodes idling is shown in Table 2. The power consumption values are min, max and the average of 60 samples. The average of the control plane compared to the multi-cluster approaches suggest a lower power consumption is needed for the remote worker. The results were the same for both the K3s multi-cluster and the microk8s multi-cluster.

A Student t-test comparing the idle power consumption of the remote worker control plane and multi-cluster edge nodes 1 and 2 showed that the difference is statistically significant ( $p < 0.05$ ). T-tests comparing the remote worker approach worker and multi-cluster edge nodes also yielded a statistically significant difference between the workers and the multi-cluster nodes.

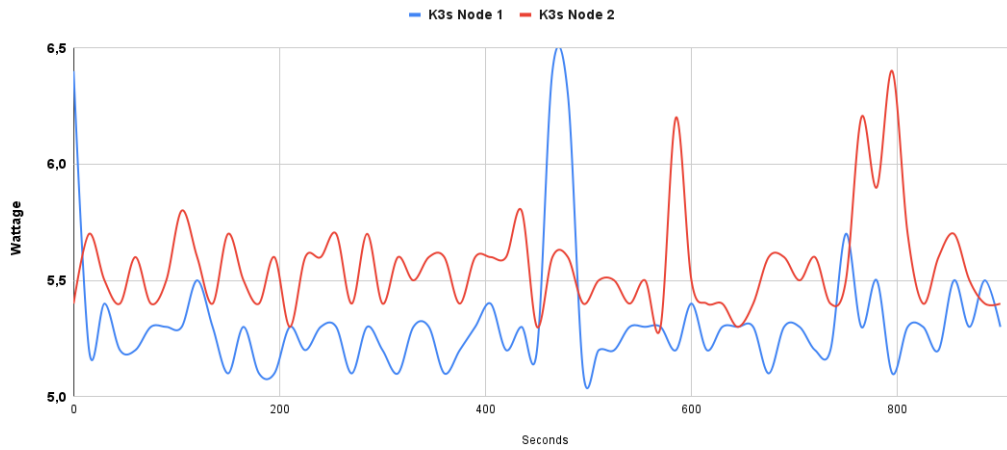
A t-test comparing the nodes in the same multi-cluster did not show a statistically significant difference with  $p = 0.37$ . However, comparing the two multi-cluster with different lightweight Kubernetes showed a statistically significant ( $p < 0.05$ ), rejecting the null hypothesis that there is no difference in power consumption. The power consumption of Microk8s was higher than K3s.

The Mann-Whitney U tests support the results generated by the t-test.

**Table 2** Min, max and average power consumption for edge nodes comprised 60 samples.

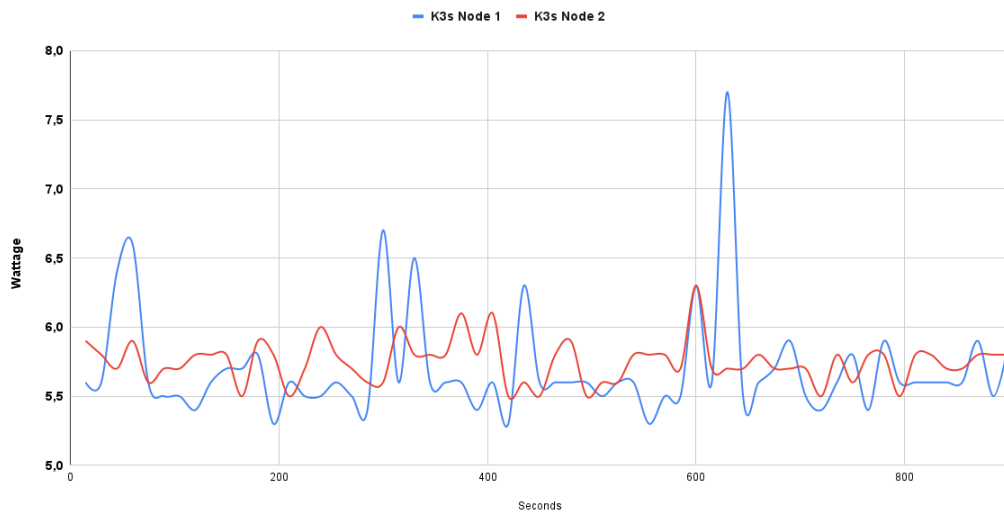
	Remote worker K3s Control plane/worker	Remote worker K3s Worker	Multi-cluster K3s Edge node 1	Multi-cluster K3s Edge node 2	Multi-cluster Microk8s Edge node 1	Multi-cluster Microk8s Edge node 2
Average	5.3	5.6	5.7	5.7	6.1	5.8
Max	6.4	6.4	7.7	6.3	8.3	7.7
Min	5.1	5.3	5.3	5.5	5.4	5.3

K3s Remote Worker

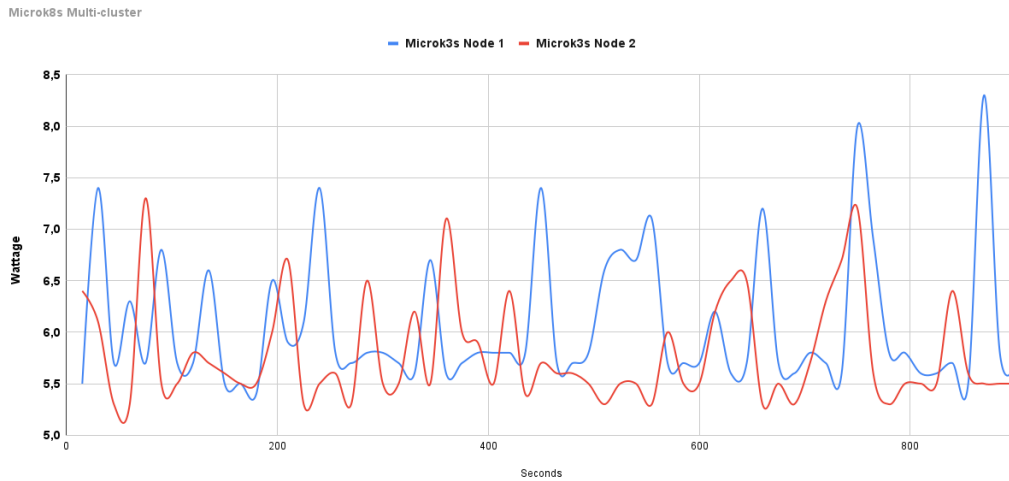


**Figure 6:** Timeline of 15 minutes of idle power consumption with 15-second sample rate for node 1 and 2 in K3s remote worker.

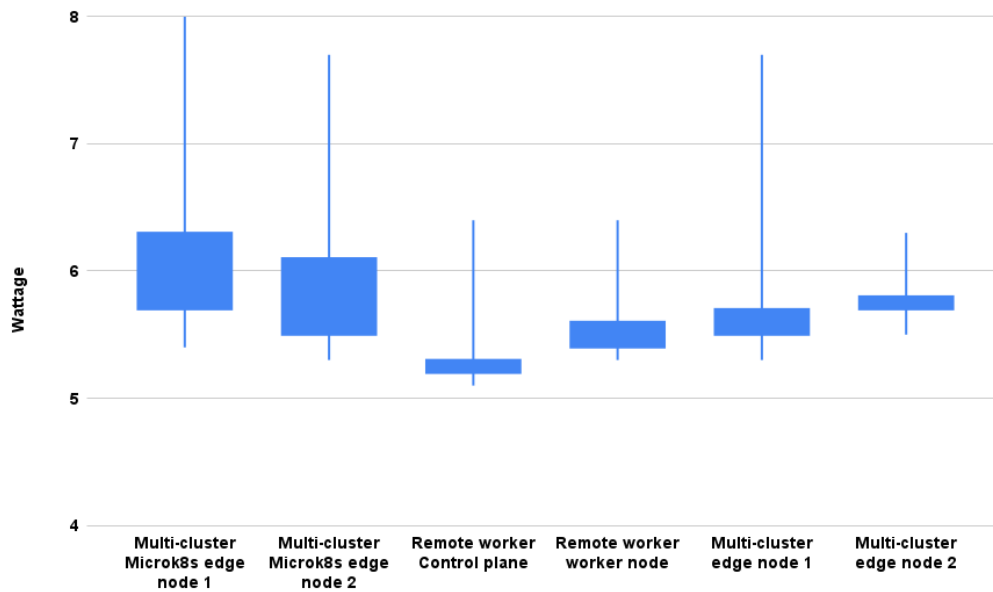
K3s Multi-cluster



**Figure 7:** Timeline of 15 minutes of idle power consumption with 15-second sample rate for node 1 and 2 in K3s multi-cluster.



**Figure 8:** Timeline of 15 minutes of idle power consumption with 15-second sample rate for node 1 and 2 in Microk8s multi-cluster.

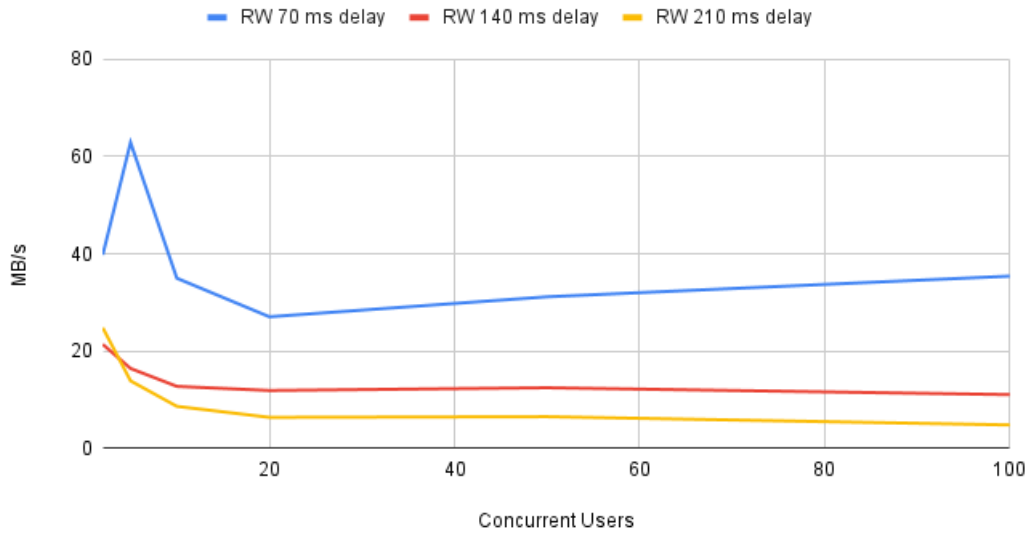


**Figure 9:** Boxplot showing power consumption over 15 minutes of idle for each node in the remote worker and multi-cluster approaches.

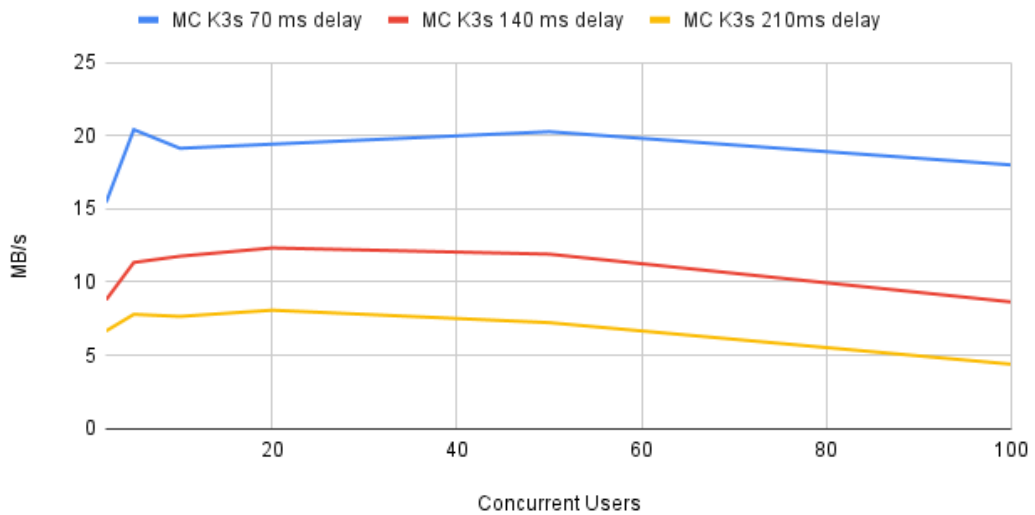
## 6.2 Throughput

The throughput is tested with network traffic delay of 70, 140, and 210 ms to simulate different environments. Figures 10, 11 and 12 indicate a decrease in throughput when delay increased. Throughput decreases when more than 50 users interact with the service concurrently, except for the remote worker approach, where it increases with 70 milliseconds

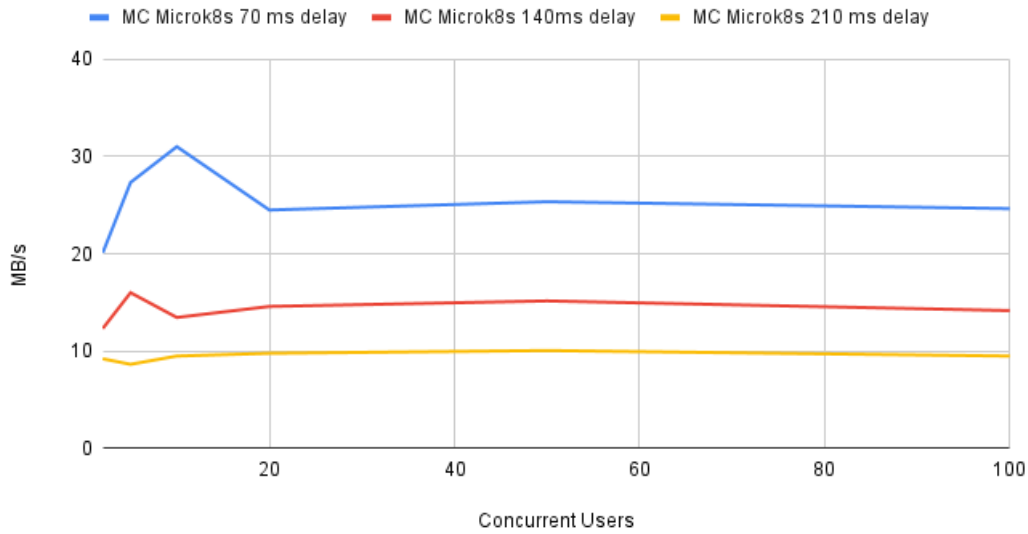
of delay and only slightly decreases with delays of 140 and 210. The throughput of the multi-cluster approach on K3s degrades more than the Microk8s. The throughput of the remote worker approach with a 70 ms delay spikes in the test with five users downloading concurrently.



**Figure 10:** Average throughput downloading a 17 MB video file using K3s on Remote Worker (RW) with 70, 140, and 210 ms delay respectively.



**Figure 11:** Average throughput downloading a 17 MB video file using K3s on Multi-cluster (MC) with 70, 140, and 210 ms delay respectively.

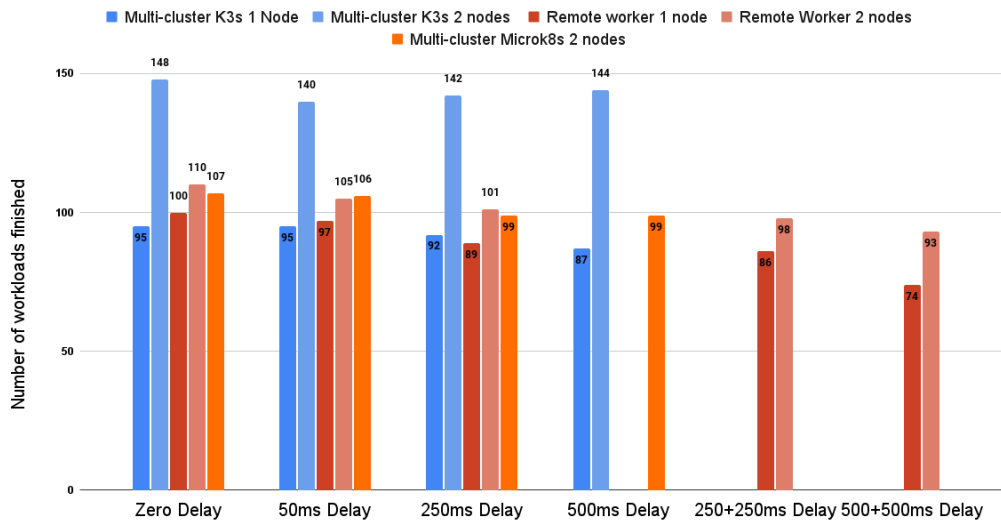


**Figure 12:** Average throughput downloading a 17 MB video file using Microk8s on Multi-cluster (MC) with 70, 140, and 210 ms delay respectively.

### 6.3 CPU-heavy test

The average number of workloads finished in 10 separate the CPU-heavy performance test running for two minutes is seen in Figure 13. The test ran with four concurrent users requesting the closest prime to 500 000 000 from below. The test was first conducted with one edge node. The test was then repeated with two nodes and load balancing in each deployment method. Due to time constraints, the Microk8s was tested with only two edge nodes. The latencies added to the network traffic was only added to one edge node except in the 250+250 and 500+500 case where latency was added to both the control plane and an edge node. The remote worker is the only one with two separate added latencies because the multi-cluster approach does not have a central control plane.

The test shows a 56% increase in finished workloads for the K3s multi-cluster approach going from one node to two nodes and 11% for the remote worker approach. With two edge nodes, the multi-cluster approach outperformed the remote worker approach in every instance, finishing around 36% more workloads.



**Figure 13:** Number of workloads finished in a 2-minute CPU-heavy performance test.

## 6.4 Hardware Utilization

Observing a fifteen-minute hardware utilization window shows a significant difference in utilization between the two approaches. The K3s multi-cluster approach was observed idling around 14-16% CPU utilization, the Microk8s utilized 24-26% CPU. RAM utilization was, on average, 1.4 GB for K3s, and 1.73 GB for Microk8s, as seen in Table 3.

The remote worker approach had a lower average CPU utilization of around 8% for the control plane/worker and 1% for the remote worker. RAM was also lower, with the control plane/worker utilization of 980 MB and the edge node worker using only 435 MB which can be seen in Table 3.

**Table 3** Average CPU and RAM utilization for both multi-cluster and remote worker approach in idle state.

	Average RAM load	Average CPU load
Remote worker approach K3s Control Plane and worker	980 MB	8%
Remote worker approach K3s Worker	435 MB	1%
Multi-cluster approach K3s Node 1	1.43 GB	14%
Multi-cluster approach K3s Node 2	1.39 GB	16%
Multi-cluster approach Microk8s Node 1	1.71 GB	26%
Multi-cluster approach Microk8s Node 2	1.75 GB	24%

## 7 Discussion

This section discusses the results presented in Section 6, answer the three research questions and examine other related topics.

### 7.1 Performance differences

Answering research question **A2**: *How do different degrees of autonomy impact performance in an edge node environment?*

#### 7.1.1 CPU-heavy

To answer this, we will first look at the number of workloads finished in two minutes shown in Figure 13. The remote worker approach performs slightly better than the K3s multi-cluster approach in a CPU-heavy service with one node until a delay of 250 milliseconds is added between the user and the control plane for the remote worker approach and the first edge node in the multi-cluster approach. However, when scaled to two nodes, the K3s multi-cluster approach scales way better than the remote worker. This could be because both deployments are load balanced, and the load balancing is performed by one of the nodes used as a worker in both cases, indicating a significant overhead for the remote worker load balancing compared to the balancing done by K8sb. The Microk8s multi-cluster performs on par with the remote worker approach using two edge nodes. The reason for Microk8s is probably the resources needed for Microk8s to run compared to the resources K3s needs. Table 3 show Microk8s requires almost double the amount of CPU in idle compared to K3s.

Comparing the deployment methods is not as simple as comparing them with the same latency. If the edge node is very close to the user, but the control plane is far away, it could be reasonable to compare the zero-delay multi-cluster with the 500+500 remote worker results. It all depends upon the placement of the control plane and the expected average latency for a user.

A central control plane could run with the sole work to load balance and keep track of all the nodes, which could improve the performance of the remote worker approach. The performance gained by having the control plane running on a separate machine might be in vain as the multi-cluster latency should, in most cases, be lower as the workload does not need to be transferred to a central control plane. But to remember, the Multi-cluster deployment method would need to be combined with a DNS to direct a user to the closest edge node adding some latency. In addition, if work is to be load balanced, it is important to configure it not to distribute work to nodes far away from the user. Otherwise, the added security benefit of a close proximity edge node only getting access to the data and the reduced latency compared to a central control plane is mainly lost when data is sent further

away.

Answering the research question **A2**. The deployment methods can impact performance for CPU-heavy services, and the K3s multi-cluster approach will probably perform better than the remote worker approach. As discussed, the gain in performance when scaling will need more extensive testing to investigate the difference if the control plane does not include a worker, adding a DNS for the multi-cluster approach, and what happens when scaling to more than only two nodes.

### 7.1.2 Throughput

The second performance test investigating the average maximum throughput of the deployment methods is shown in Figures 10, 11 and 12. Figure 10 contains an outlier. This outlier occurs when five concurrent users download a file in the remote worker method with 70 milliseconds of latency. To find an explanation for the outlier test was conducted with 3,4,6 and 7 concurrent users and 60 and 80 milliseconds of latency. Neither of the different test results aligned with the outlier. The test with five users and 70 ms latency was tested on other days to avoid the possibility of the state of the node impacting performance. However, the test results were consistent with the initial results. The remote worker approach with 70 milliseconds of latency is also one of two which increases throughput after adding more than 20 users, with the other being Microk8s multi-cluster with 140 milliseconds delay. The 70 milliseconds remote worker approach is the only configuration increasing throughput with more than 20 concurrent users. It will need more testing to give a definitive answer to the reason for the behaviour of the performance. For this project, the outlier with five users is assumed to be an anomaly.

Overall the performance degrades with more concurrent users when testing the remote worker approach. The cause of this could be the considerable overhead for load balancing. However, when observed during testing, the throughput test did not utilize much CPU or RAM. If the overhead of the load balancing were the reason for the performance difference, we would assume a lot of hardware utilization.

The remote worker approach did perform better or on par with the K3s multi-cluster approach except for 210-millisecond latency and 20-50 users, where the K3s multi-cluster approach performed slightly better.

The Microk8s multi-cluster approach performed better for the throughput, especially with more concurrent users but lost to the remote worker on low latencies and fewer than ten users. If the edge nodes are mainly used to transfer data to the end users, the importance of the security benefit from the multi-cluster approach should not be the priority. Instead, the remote worker method should be considered with the performance gained using it. It is essential to mention the need for a control plane to send live check messages to all edge node worker nodes. If the latency between one of the nodes and the control plane is too big, the control plane will stop scheduling work to that node and reschedule work already assigned to the specific node. The latency limit can be adjusted but will affect the network negatively if nodes often crash, as the control plane will schedule work for longer before reacting to a node crash. If Liqo is used, there is also an issue when peering nodes, as the peering will fail with high latencies. In the project, a network latency of one second failed the peering. However, no tests were done to determine whether the timeout limit can be configured.

## 7.2 Power Consumption

Answering research question **A3**: *Does the deployment method impact the power consumption of edge nodes?*

To determine if the deployment method impacts the power consumption, we will look at Figures 9, 7, 8, 6 and Table 2. The average power consumption is higher for all of the multi-cluster nodes. The control plane's low average power draw is interesting as the control plane node included a worker; it was assumed to require more power than the worker node. An explanation for this could be differences in power draw between the nodes because the hardware differs in the amount of power drawn in with idle the same software installed.

Table 3 shows the average utilization of CPU and RAM, also supporting the higher power draw for the multi-cluster approach. However, it does not support the reasoning of the control plane using less power than the worker node, which uses less than half of the RAM and less than half a per cent of the CPU compared to the control plane node. Answering the research question, there is a difference in power consumption; the remote worker approach requires less power than the multi-cluster approach.

## 7.3 Hardware Utilization

Answering research question **A1**: **Does lightweight Kubernetes distribution impact performance?**

With the multi-cluster nodes using more than 14% of the CPU and between 1.4-1.7 GB of RAM, the multi-cluster approach might not be the ideal setup for very low-spec edge nodes. This is, however, arguable as the multi-cluster approach performs almost as well as the remote worker approach in the CPU-heavy tests with one node, and the k3s version scales better.

If the edge nodes are idling most of the time and the performance of the nodes is not critical, the multi-cluster is not the best choice. It utilizes a lot of CPU and RAM even in idle, resulting in higher power consumption. In a network of hundreds or thousands of edge nodes, the cost of the extra power consumption should be considered when deciding which approach to use.

To answer the research question, it does impact both the power consumption and the performance. The CPU-heavy test shows Microk8s scale as bad as the remote worker approach but performed better in the file transfer test. Choosing which lightweight distribution to use is as important as choosing the deployment method.

## 8 Future Work

To give confident recommendations on which deployment method to use, comparing the deployment methods will need a more comprehensive arrangement of tests. Not only a multitude of varieties of tests but also the test environment. A DNS for the multi-cluster would result in a more realistic test environment. The scalability, one of the most important benefits of the cloud, must be tested more extensively. As mentioned before, the control plane should also run on a separate machine with the task of only running the control plane.

Researching what types of hardware are mainly used in edge nodes could be an idea, and using similar hardware to simulate what deployment methods should be used with the hardware used in the current edge nodes.

With the popularity of streaming services today, it would be a good idea to test the performance with similar software. Another widespread use case is storing and fetching data with tools like Dropbox, which also could be tested.

## 9 Conclusion

The experiments successfully demonstrated differences in power consumption between the deployment methods. The multi-cluster method needs more power than the remote worker method and performs as well or better than the remote worker in a CPU-heavy service. Furthermore, it is shown that it scales better than the remote worker method. Still, it does need more comprehensive testing to showcase the scalability of a cluster consisting of tens of hundreds of nodes. It is important to notice the lack of a DNS in the multi-cluster tests and placement of the control plane combined with a worker node.

This project does not give a definitive answer to which deployment methods to use. Still, it does showcase some differences between the deployment methods, which should be considered when planning to implement one of them. Both deployment methods have advantages and disadvantages, and it is up to the implementor to choose the best fit for the use case.

## References

- [1] S. P. Ahuja and N. Deval. From cloud computing to fog computing: Platforms for the internet of things (iot). In *Research Anthology on Architectures, Frameworks, and Integration Strategies for Distributed and Cloud Computing*, pages 999–1010, 2021.
- [2] Kashif Bilal, Saif Ur Rehman Malik, Samee U. Khan, and Albert Y. Zomaya. Trends and challenges in cloud datacenters. *IEEE Cloud Computing*, 1(1):10–20, 2014.
- [3] Halim Fathoni, Chao-Tung Yang, Chih-Hung Chang, and Chin-Yin Huang. *Performance Comparison of Lightweight Kubernetes in Edge Devices*, pages 304–309. Springer Cham, 11 2019.
- [4] Rimma Iontel and Fatih Nar. Kubernetes deployment models for edge applications. <https://www.redhat.com/architect/kubernetes-edge-applications>. Accessed: 2023-01-23.
- [5] Asif Khan. Key characteristics of a container orchestration platform to enable a modern application. *IEEE Cloud Computing*, 4(5):42–48, 2017.
- [6] Frederic Lardinois. As kubernetes hits 1.0, google donates technology to newly formed cloud native computing foundation. <http://tcn.ch/1HGruA7>. Accessed: 2023-02-21.
- [7] Charles Mahler. Kubernetes on the edge: getting started with kubeedge and kubernetes for edge computing. <https://www.cncf.io/blog/2022/08/18/kubernetes-on-the-edge-getting-started-with-kubeedge-and-kubernetes-for-edge-computing/>. Accessed: 2023-01-13.
- [8] What is the cloud? <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-the-cloud>. Accessed: 2023-02-21.
- [9] What is kubernetes multi-cluster? <https://www.mirantis.com/cloud-native-concepts/getting-started-with-kubernetes/what-is-kubernetes-multi-cluster/>. Accessed: 2023-02-15.
- [10] What is a kubernetes cluster? <https://www.redhat.com/en/topics/containers/what-is-a-kubernetes-cluster>. Accessed: 2023-02-06.
- [11] Naidila Sadashiv and S. M. Dilip Kumar. Cluster, grid and cloud computing: A detailed comparison. In *2011 6th International Conference on Computer Science & Education (ICCSE)*, pages 477–482, 2011.
- [12] Ali Sunyaev. *Fog and Edge Computing*, pages 237–264. Springer International Publishing, 2020.

- [13] Sergii Telenyk, Oleksii Sopov, Eduard Zharikov, and Grzegorz Nowakowski. A comparison of kubernetes and kubernetes-compatible platforms. In *2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, volume 1, pages 313–317, 2021.
- [14] Blesson Varghese, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick, and Dimitrios S. Nikolopoulos. Challenges and opportunities in edge computing. In *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 20–26, 2016.
- [15] J. Vila-Carbo, J. Tur-Masanet, and E. Hernandez-Orallo. An evaluation of switched ethernet and linux traffic control for real-time transmission. In *2008 IEEE International Conference on Emerging Technologies and Factory Automation*, pages 400–407, 2008.
- [16] Huan Wei. Kubeedge and kubernetes help manage all the monitoring devices on the world’s longest cross-sea bridge. [https://www.youtube.com/watch?v=iXFvT-NtElk&ab\\_channel=CNCF%5BCloudNativeComputingFoundation%5D](https://www.youtube.com/watch?v=iXFvT-NtElk&ab_channel=CNCF%5BCloudNativeComputingFoundation%5D). Accessed: 2023-02-17.

# A Appendix

## A.1 Appendix: Code used in testing

```
package main

import (
    "fmt"
    "net/http"
    "io/ioutil"
    "strconv"
    "os"

    "github.com/gin-gonic/gin"
)

func isPrime(c *gin.Context) {
    number, err := strconv.Atoi(c.Param("nr"))
    fmt.Println(number)
    fmt.Println(err)

    if err == nil {
        for i := 2; i < number; i++ {
            if number%i == 0 {
                number--
                i = 1
            }
        }

        fmt.Println(number)
        c.JSON(http.StatusOK, gin.H{
            "code": 200,
            "msg":  number,
            "data": number,
        })
    }
}

func getvideo(c *gin.Context) {
    path, err := os.Getwd()
    fmt.Println(err)
}
```

```

home, err := os.UserHomeDir()

fmt.Println(err)

files, err := ioutil.ReadDir(home)
var dirs []string
for _, f := range files {
    dirs = append(dirs, f.Name())
    fmt.Println(f.Name())
}
fmt.Println(err)
c.JSON(http.StatusOK, gin.H{
    "code": 200,
    "msg": path,
    "data": dirs,
})
//c.File("/home/robin/toolbox.tar.gz")
c.File("/home/c18rsn/vid.mp4")
}

func main() {
    r := gin.Default()
    r.GET("/video", getvideo)
    r.GET("/prime/:nr", isPrime)
    r.Run() // listen and serve on 0.0.0.0:8080 (for windows "
        localhost:8080")
}

```

## A.2 Appendix: K8s deployment file

```

# Source: podinfo/templates/service.yaml
apiVersion: v1
kind: Service
metadata:
  namespace: liqo-test
  name: liqo-test
spec:
  type: ClusterIP
  selector:
    app: liqo-test
  ports:
    - port: 9898
      targetPort: 8080
      protocol: TCP
---
```

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: liqo-test
  namespace: liqo-test
spec:
  replicas: 6
  selector:
    matchLabels:
      bb: web
  template:
    metadata:
      labels:
        app: liqo-test
        bb: web
    spec:
      containers:
      - name: liqo-test
        image: ztolpotato/stresstest:v9
        imagePullPolicy: Always
        ports:
        - containerPort: 8080
          protocol: TCP
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            - labelSelector:
                matchExpressions:
                - key: app.kubernetes.io/name
                  operator: In
                  values:
                  - liqo-test
              topologyKey: kubernetes.io/hostname

```

----

```
# Source: podinfo/templates/ingress.yaml
```

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: liqo-test
  namespace: liqo-test
  labels:
    app: liqo-test
  annotations:
    k8gb.io/strategy: roundRobin
spec:
  ingressClassName: nginx
  rules:
  - http:

```

```

    paths:
      - path: /
        pathType: ImplementationSpecific
        backend:
          service:
            name: liqo-test
            port:
              number: 9898
---
apiVersion: v1
kind: PersistentVolume
metadata:
  name: example-pv
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadOnlyMany
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: /
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
                - liqo-test

```