

**ACIT5900**  
**MASTER THESIS**

in

**Applied Computer and Information  
Technology (ACIT)**

**May 2021**

**Cloud-based Services and Operations**

**Plastic Fantastic: Harnessing architectural  
plasticity for low cost, highly distributed  
and completely green clouds**

Tara Jørgensen

**Department of Computer Science**  
**Faculty of Technology, Art and Design**

**OSLOMET**

# Abstract

The current paradigm of cloud computing dictates massive data centers that use excessive amounts of energy in order to operate and maintain their services. It is a side of the industry that has not been revamped as sustainability has become more and more important, but rather one that has followed in its own footsteps and continued to grow alongside the ever-increasing demand.

Through modeling and presenting our own architectures, with an underlying focus on green energy, we are able to offer new options to how the cloud could be structured. While these architectures are deemed different from the classical cloud architecture, they are not different enough from already existing technologies. Merging them together into a large-scale adaptive architecture creates something that is completely different from what is available today, and results in an architecture that challenges the status quo.

# Acknowledgements

First and foremost, I would like to thank my supervisor, Kyrre Begnum, who has been a vital resource while writing this thesis. Without his input, consisting of both suggestions and constructive criticism, this thesis would not have been able to reach its full potential. Our weekly meetings and the feedback he has given me along the way is something that has helped me immensely.

I would also like to thank the rest of the teachers I have had the pleasure of interacting with during my master's program here at OsloMet for the knowledge and skills they have taught me, which has helped me write this thesis.

# Table of Contents

1	Introduction .....	8
1.1	Problem Statement .....	10
1.2	Outline.....	11
2	Background.....	13
2.1	The Cloud.....	13
2.1.1	Public Cloud .....	13
2.1.2	Private Cloud .....	14
2.1.3	Hybrid Cloud.....	14
2.1.4	Historic Perspective .....	14
2.1.5	A Typical Cloud Architecture .....	18
2.2	Distributed Computing and Existing Technologies .....	18
2.3	The Green Pledge.....	20
2.4	Summary.....	22
3	Approach.....	23
3.1	Research Approaches.....	23
3.2	Research Objectives .....	25
3.3	Results .....	25
3.4	Analysis.....	26
3.5	Discussion.....	26
3.6	Conclusion and Future Work.....	26
4	Results .....	27
4.1	Common Features .....	27
4.1.1	Roles and General Role Assignment.....	27
4.1.2	Energy Harvesting .....	30
4.1.3	Small Nodes .....	32

4.1.4	Availability.....	33
4.1.5	Containers .....	34
4.1.6	Resilience .....	35
4.1.7	Distribution and Wireless Communication .....	36
4.2	Aligning Our Common Features to The Architectural Scale.....	38
4.3	Architectures .....	41
4.3.1	Architecture 1: Hierarchical Architecture.....	41
4.3.2	Architecture 2: Meshed Architecture.....	49
4.3.3	Architecture 3: Graphed Architecture .....	55
4.3.4	Handling Services.....	61
4.4	Summary.....	67
5	Analysis and the Introduction of Plasticity .....	69
5.1	Placement on the Architectural Scale .....	69
5.2	Comparison to Today’s Cloud.....	72
5.3	Most Groundbreaking Approach(es) .....	73
5.4	Cheapest Approach .....	73
5.5	Most Realistic Approach .....	74
5.6	Large-Scale Plasticity.....	75
6	Discussion.....	77
6.1	A New Discovery.....	77
6.2	Individual Qualities and Overarching Similarities .....	78
6.3	Fundamental Assumptions.....	79
6.4	Did We Drift Away from The Original Plan? .....	81
7	Conclusion and Future Work.....	83

# List of Figures

<b>Figure 2.1:</b> “Hype Cycle”, 2008, Gartner Group. (Retrieved from <a href="https://techcrunch.com/2008/08/18/where-are-we-in-the-hype-cycle/">https://techcrunch.com/2008/08/18/where-are-we-in-the-hype-cycle/</a> ) .....	16
<b>Figure 4.1:</b> The role assignment scale .....	29
<b>Figure 4.2:</b> Landscape.....	36
<b>Figure 4.3:</b> Nodes in landscape.....	37
<b>Figure 4.4:</b> The architectural scale .....	39
<b>Figure 4.5:</b> The approximate area our architectures will reside on.....	40
<b>Figure 4.6:</b> A section of an uneven hierarchical architecture.....	42
<b>Figure 4.7:</b> Hierarchical role assignment based on geographical location.....	44
<b>Figure 4.8:</b> Hierarchical role assignment based on signal strength .....	44
<b>Figure 4.9:</b> Large-scaled hierarchical role assignment based on both geography and signal strength.....	45
<b>Figure 4.10:</b> A section of the hierarchical architecture where two controllers control the same nodes.....	47
<b>Figure 4.11:</b> A controller at a higher level takes over the responsibility.....	48
<b>Figure 4.12:</b> A controller at the same level takes over the responsibility .....	48
<b>Figure 4.13:</b> Meshed architecture with simple role assignment.....	50
<b>Figure 4.14:</b> Meshed architecture with more realistic role assignment.....	50
<b>Figure 4.15:</b> A mesh restructuring itself into a hierarchical mesh .....	52
<b>Figure 4.16:</b> A node moves out of a sub-mesh after being promoted to a master-controller. While this happens, the sub-meshes are disconnected from the rest of the architecture, but can operate within itself. ....	53
<b>Figure 4.17:</b> Hierarchical mesh reverting back into mesh.....	54
<b>Figure 4.18:</b> A non-weighted graph architecture .....	55
<b>Figure 4.19:</b> A weighted graph architecture.....	56
<b>Figure 4.20:</b> Different weighted lines as a result of different types of nodes .....	56
<b>Figure 4.21:</b> Inefficient vs. efficient role assignment.....	58

**Figure 4.22:** Example of restructuring in the non-weighted graph architecture when a controller disconnects, but all the nodes are already controlled by a second controller. No new lines are drawn..... 60

**Figure 4.23:** Example of restructuring in the non-weighted graph architecture when a controller disconnects, and the nodes have to be switched to a different controller..... 60

**Figure 4.24:** A meshed hierarchy where the bottom compute node is part of two overlapping pod-like structures, represented by the greyed-out areas containing one of each role. .... 63

**Figure 5.1:** The architectural scale, with the addition of the hierarchical architecture... 70

**Figure 5.2:** The architectural scale with the addition of the meshed architecture ..... 71

**Figure 5.3:** The architectural scale with the addition of the graphed architectures ..... 72

**Figure 5.4:** The final rendition of the architectural scale ..... 72

## List of Tables

<b>Table 4.1:</b> Directional impact on the architectural scale from the common features .....	40
<b>Table 4.2:</b> Color coding for nodes in upcoming figures.....	42
<b>Table 4.3:</b> Summary of important concepts .....	68



# 1 Introduction

Cloud computing, which represents the tech industry's biggest environmental footprint, is entering a new paradigm where resource use challenges established values, such as economic and practical considerations. This shift is brought forth from both external pressures, as well as from an internal desire to help mitigate climate change. With its enormous size, any change within cloud computing would have a profound impact. The onus falls on system administrators and engineers to find ways to change the cloud from within, while keeping its functionality in place for the rest of the world.

Over the past two decades, ever since Amazon came out with their cloud solution, Amazon Web Services (AWS), and released products like their Simple Storage Service (S3) and Elastic Compute Cloud (EC2), the use of cloud services has grown rapidly. As of 2021, the global cloud computing market was worth 445.3 billion dollars, a number that is expected to grow to 947.3 billion dollars by 2026 (MarketsandMarkets, 2021). Today, the biggest cloud providers include Amazon with AWS, Microsoft with Microsoft Azure (Azure), and Google with the Google Cloud Platform (GCP). These three are responsible for providing cloud services for millions of users across the world.

For a cloud user, outsourcing things like computing power and storage to a cloud provider is a no-brainer. Not only do they not have to worry about things like maintenance and uptime, as this is all handled for them, but their personal resource use is also significantly lowered. A cloud user does not have to think about things like physical storage locations or manpower to maintain the hardware; They can lean back and let someone else handle it, while they reap the benefits of the cloud. Out of sight, out of mind. However, somewhere, someone still needs to think about these things, and resources still need to be consumed.

The cloud is, contrary to what some might think, not a hovering, omnipotent, magical entity, but just someone else's computer. Because of the cloud's immense popularity, "someone else's computer" needs to be able to accommodate millions of users at the

same time, and thus, constantly needs to consume large amounts of resources. The processing units, cooling systems and other hardware that are needed to process the user requests are located in large data centers across the world. These data centers take up lots of physical space and need a lot of energy to be able to run successfully.

Take Amazon for example, the largest cloud provider on the market today. Amazon has 84 data centers in 26 geographic regions, with additional expansions planned (Amazon, n.d.). Considering that just four data centers are estimated to take up approximately 95'000 square meters (Swinhoe, 2021), over 200 times the size of Frogner Park, it is safe to say that data centers in general take up huge portions of land across the world. If data centers didn't need as much space as they do today, those areas could've been used for food production, recreational areas or housing, or even as national parks with the intent of conserving wildlife.

Large data centers also come with a large need for energy. A couple of years ago, it was reported that data centers use more than 2% of all electricity in the world on an annual basis (Pearce, 2018). How much of this energy comes from renewable sources? Additionally, according to the same source, the data centers also contribute to 2% of the world's CO<sub>2</sub> emissions, which is equivalent to the world's entire airline industry. If data centers were much more distributed across the world and placed closer to their energy sources, their carbon footprint might be much lower.

It's time to question whether today's way of handling cloud computing is done in the best way, or if it can be improved upon. As of right now, there are no real, well-established alternatives to turn to, so we continue to do things like we have always done them. There are models and concepts that could be much better at handling emissions and resource use, like the Plan 9 operating system, grid-, and edge computing, but none of them have been translated to today's cloud architecture. There are also cloud technologies that focus on having a low environmental footprint, like Azure's underwater datacenter, IncludeOS and the GCP Carbon Footprint report. However, these

technologies have never been explored in a context of the alternative models and concepts.

By selecting certain qualities from models and concepts like these, it is possible to imagine an entirely new way of approaching cloud computing. Instead of having huge data centers that take up a lot of space, we can take inspiration from grid- and edge computing, where the processing units are smaller and more widely distributed. Wireless networks are also a source of inspiration, as units can easily enter and exit the network, and they can get power through energy harvesting. These qualities could help create a highly distributed cloud platform, which is much more minimalistic in nature compared to the infrastructure of today's colossal data centers.

Looking into alternate ways of handling cloud computing and data storage might give us groundbreaking knowledge that breaks with the current status quo of how these things are handled today. This knowledge can potentially impact future endeavors, and might help guide us towards a greener, more sustainable future. However, changing the cloud's architecture to fit the alternate models and concepts is huge and potentially disruptive, and might fundamentally change how the cloud works. Research is therefore necessary in order to determine how extensive the changes would be, and what impact they would have on the cloud.

## 1.1 Problem Statement

The intersection between technology, cloud architecture and green computing is becoming more and more relevant. The focal point of this thesis is therefore to explore that intersection, through focusing on the following problem statement:

**P1:** Challenge the status quo by investigating the effects of a highly distributed and minimalistic cloud platform driven by a green energy focus.

The current *status quo* of cloud computing is, as described, large data centers that cover massive areas and use a lot of electricity. By investigating alternate ways of handling cloud computing and its possible advantages, we challenge the status quo.

The *investigation* involves looking into how cloud computing is done today, how other technologies handle themselves, and seeing how their qualities could potentially influence cloud computing. As part of the investigation process, we will present different options for how a cloud architecture could potentially be structured and operate, compare those options to today's cloud architecture, and define whether or not they are valid options. The goal of the investigation process is simply to see if there is another way of doing cloud computing than the way that has naturally emerged, and not to change the existing cloud itself.

A *highly distributed* cloud platform indicates a cloud platform with hardware that is geographically spread out from each other, on a much higher level than today's cloud. A *minimalistic* cloud platform indicates data centers and processing units of a much smaller capacity and size than the ones we have today. The *cloud platform* itself is the digital environment that serves as a Platform as a Service, where the cloud users can make use of hardware and software tools. The cloud platform runs on the different processing units around the world and are accessible through a network connection.

Every technology, concept, location and so on that is presented in this thesis will have a *green energy focus*, meaning an all-encompassing sustainable viewpoint with focus on things like renewable energy, responsible consumption etc. Having this viewpoint and building our knowledge upon it will ensure that our proposed changes might meet the modern expectations of sustainable computing.

## 1.2 Outline

This thesis consists of 7 chapters: introduction, background, approach, results, analysis, discussion, and conclusion and future work. This section presents an outline of those chapters with a short summary of their contents.

- **Chapter 1, Introduction:** Introduces the problem domain, the impact of addressing the problem, and the problem statement and its operatization.
- **Chapter 2, Background:** Presents how today's cloud works and how it came to be, in addition to mentioning other distributed computing technologies. Also Introduces the concept of "the green pledge", which is based on sustainability, and how it impacts many of our day-to-day choices.
- **Chapter 3, Approach:** Outlines a detailed plan for the remaining parts of the thesis through presenting the type of research that will be done, additional questions that must be discussed in order to answer the problem statement, and summaries of the upcoming chapters.
- **Chapter 4, Results:** Presents the architectures that intend to challenge the status quo by listing their common features, architectural structure, role assignment and how they handle running different services. Also introduces a comparative tool that will be used during the analysis: *The architectural scale*.
- **Chapter 5, Analysis, and the Introduction of Plasticity:** Looks at the presented architectures through a critical lens in order to determine their individual qualities, and places them on the architectural scale where they are compared to other architectures. It also further improves the solution by presenting a large-scale dynamic architecture based on the phenomenon of plasticity.
- **Chapter 6, Discussion:** Discusses the previous chapters through a reversed overlook of the entire thesis and highlights important findings that have been made throughout.
- **Chapter 7, Conclusion and Future Work:** Summarizes the thesis and presents future research ideas which follow from the contributions of this project.

## 2 Background

### 2.1 The Cloud

Although we often speak of “The Cloud” as one thing, it actually consists of millions of different pieces coming together to create a variety of different and seamless functions. The cloud is a common name used to describe software and services that are available through a network connection, instead of on your local computer. Using the cloud means outsourcing things like computing and storage to other machines and being able to access that data through a digital platform.

No one person or company owns The Cloud as a whole, but everyone can own and operate *their own* clouds. There are many different cloud providers. In addition to Amazon, Microsoft and Google, other cloud providers on the market include companies like Alibaba, IBM, and Oracle. Some cloud providers also supply their users with tools to set up their own clouds. These custom clouds are often created for specific situations, like for example the need for cloud computing with high levels of security, or for educational purposes.

The cloud is available in three main forms: Public Cloud, Private Cloud and Hybrid Cloud. These categories define the way in which the cloud is accessed and run, and not their architectural patterns.

#### 2.1.1 Public Cloud

A public cloud is a multi-tenant environment, meaning that different cloud users share a pool of resources that are automatically allocated to the individual tenants (individual users or groups of users). Each tenant’s data is logically isolated from the other tenants and can not be accessed by another party, but two different tenants’ workloads could realistically be running on the same servers at the same time (IBM Cloud Education, 2020a). The public cloud is the most common type of cloud computing. A public cloud user will save money on maintenance and hardware costs as the resources are owned

and operated by a third-party service provider, and the cloud users can access the services over the internet. The public cloud is also highly reliable thanks to multiple servers and data centers across the world that ensures against failures, and it also supports the appearance of near-unlimited scalability from on-demand resources (Microsoft Azure, n.d.).

### 2.1.2 Private Cloud

A private cloud is a single-tenant environment, meaning that it's used exclusively by one business or organization. This is also referred to as *isolated access* (IBM Cloud Education, 2020b). The cloud can be physically located in an on-premises data center or hosted by a third-party service provider. Unlike in the public cloud, the services and infrastructure in a private cloud are always maintained on a private network, and any hardware and software are dedicated solely to the party that uses the private cloud (Microsoft Azure, n.d.). A private cloud allows for more customization and control of the available hardware and software, and as a result there is greater visibility into security and access control.

### 2.1.3 Hybrid Cloud

A hybrid cloud is a combination between a public and private cloud, which combines infrastructure elements from them both in a highly flexible environment. The hybrid cloud allows data and applications to effortlessly move between a party's public and private cloud, which allows cloud users to take advantage of extra computing power only when needed (Microsoft Azure, n.d.).

### 2.1.4 Historic Perspective

The two terms “Cloud” and “Cloud Computing” were introduced in 2006 by the then CEO of Google, Eric Schmidt, in a conversation hosted by journalist Danny Sullivan at the Search Engine Strategies Conference (Schmidt & Sullivan, 2006). According to Schmidt, cloud computing was built on the premise that the data services and architecture should be on remote servers “in a cloud somewhere”, hinting at its obfuscated nature. As long as users had the right kind of browser or the right kind of

access, they would be able to access the cloud. As predicted, this has all come to fruition. While “Cloud” and “Cloud Computing” were new words at the time, used to describe the new emerging model that was making headway, the underlying concept of cloud computing is actually much older.

At the centennial celebration of MIT in 1961, the American computer scientist John McCarthy gave a speech in which he presented the idea of computing as a public utility. According to him, computing could someday be organized as a public utility, just like the telephone system was. Should this happen, the computing utility could become “the basis for a new and important industry” (Greenberger, 1962). McCarthy specifies in his speech that his ideas are built on the concept of time-sharing computer systems, which goes even further back; One of the first papers on the subjects was written and presented by Christopher Strachey at the Paris International Conference of Information Processing in 1959, and previously unrealizable systems, which McCarthy argues could be solved with time-sharing technologies, were discussed already in 1945 (Greenberger, 1962).

A time-sharing computer system is made up of a large computer that is accessible through remote terminals, resulting in better utilization of the equipment by having more than one user share the same machine (Bell & Gold, 1972). McCarthy talked about how such a system will look to each user like a large private computer (Greenberger, 1962), a statement that draws parallels to how we view multi-tenancy today.

Over the next decades, the ideas of computing utility and time-sharing computer systems stayed relevant, but they ultimately faded away by the 1990’s. It wasn’t before the mid 2000’s when the idea had resurfaced in new forms, largely thanks to Eric Schmidt, that it finally took a foothold.

After being reintroduced as cloud computing, the idea of outsourcing computational power grew rapidly in popularity and feasibility. Data from the 2008 Gartner Group Hype Cycle, a visual representation of how a technology or application will likely evolve over



time (Gartner Group, n.d.), put Cloud Computing in the fast-growing phase, specifying that it would reach mainstream adoption in two to five years, as seen in Figure 2.1.

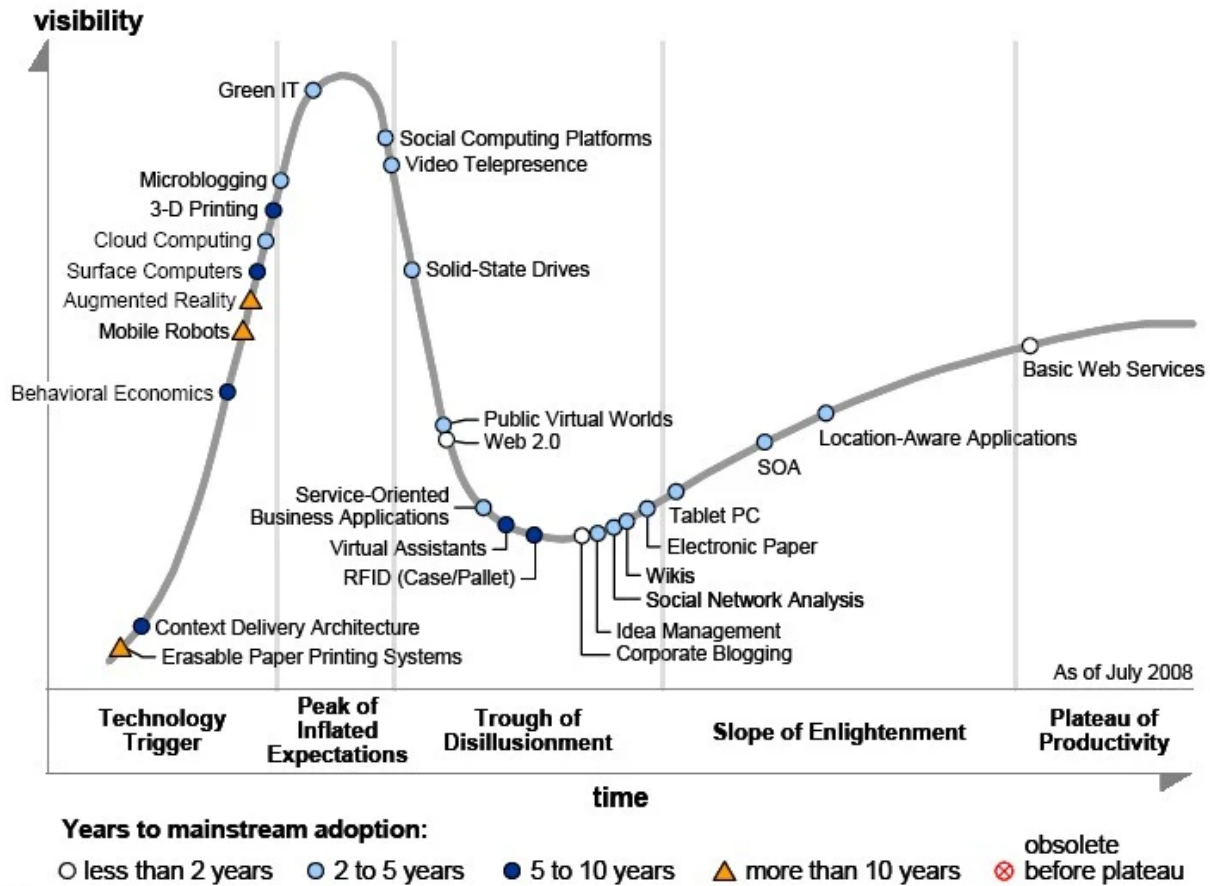


Figure 2.1: "Hype Cycle", 2008, Gartner Group. (Retrieved from <https://techcrunch.com/2008/08/18/where-are-we-in-the-hype-cycle/>)

The rise in popularity can arguably be attributed to the explosive growth in IT that has taken place over the past two decades, which has resulted in billions of internet users and infinitely better technology than what was possible just a few years ago. The theory that describes this phenomenon is known as Moore's law. Moore's law theorizes that the processing power of computers doubles every few years, resulting in an exponential growth over a short period of time. The theory is grounded in facts from the past years, where transistors have gotten smaller, processing power has increased and energy efficiency has been improved, resulting in cheap and powerful computers (Intel, n.d.).

What started out as basement dwelling server rooms in the 1970's with monolithic mainframe computing installations (Winter, 2009) has grown in line with the technological revolution, right alongside the companies themselves. Even though the physical space needed for equal amounts of computing power is much smaller, we have reached a point where the needed processing power is larger than ever. Huge companies, like Amazon, Microsoft, and Google, therefore rely on huge data centers in order to operate and provide their customers with their services. Arguably, data centers are just oversized server rooms, and not once did we stop to think if that should be the standard of the future, we just let it happen.

As part of the technical revolution, new services emerged. Cloud services like Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS) all make it possible for users to outsource things like computing power, storage and software maintenance. These kinds of services are on-demand, scalable and fast, and greatly reduce costs and increase the efficiency of the IT infrastructure (Arutyunov, 2012). Saving time and money appeals to the users, which is why millions have flocked to the cloud service providers.

In recent years, we have seen signs of a multicloud emerging. Multicloud is a word used to describe the use of cloud services from two or more cloud providers. It can be as simple as using Software-as-a-Service from a second cloud provider, but it most typically refers to running applications on a Platform-as-a-Service or Infrastructure-as-a-Service from multiple cloud providers (IBM Cloud Education, 2021). Different clouds often have tools that make it easy to send and receive data from other cloud providers, like for example GCP's Data Transfer that can get data from another provider's storage bucket by using a Shared Access Signature (SAS), or BigQuery Omni that lets users run BigQuery analytics directly on data stored in other clouds (Google Cloud, n.d.).

Collaborative tools like these might make it seem like we are moving towards *one* singular cloud where everything is possible, which we most likely are not. The cloud will still be owned and operated by different parties, and the sharing of tools and

technologies between them will in most likelihood force the cloud providers to continuously come up with technological advances that will distinguish them from the others and give them the upper hand.

### 2.1.5 A Typical Cloud Architecture

No matter how the cloud is accessed, there needs to be certain rules in place that determine which servers run what services, what responsibilities they each have, etc. These rules make up the cloud architecture and decide how the cloud operates. The rules are not likely to change, and the architecture itself will be static, even if there are dynamic changes happening within it.

To properly be able to do everything that is needed of it, a cloud needs a collection of nodes. A node is a point in a network at which pathways to other nodes intersect (Oxford Languages, 2022a), which enables them to communicate. “Node” is a common name and can be used for many different roles. At its most basic level, a cloud needs a managerial controller node that can control the other nodes, worker nodes that actually do the work, and one or more types of support nodes (IBM, 2021). These nodes come together to create the components of a cloud architecture: a front-end and back-end platform, a network, and a delivery model to get the content from the platform to the end users (VMWare, n.d. a). If the nodes do not reside in the same location, different types of software and virtualizations can be used to trick them into believing that they are.

## 2.2 Distributed Computing and Existing Technologies

Distributed computing refers to a collection of multiple individual, autonomous, asynchronous nodes that are geographically separate, yet communicate with each other over a communication network that they are all connected to (Kshemkalyani & Singhal, 2011). When these nodes come together, they are able to offer one seamless system that does not bear the marks of being split into pieces. A cloud is one example of such a system. There are also many other existing distributed systems. In this thesis, we quickly have a look at a selection of them to see how they work. These systems are the

Plan 9 OS, CDN's, grid-, fog- and edge computing, wireless sensor networks, and wireless ad hoc networks.

Plan 9 from Bell Labs, also just known as Plan 9, is a distributed operating system from the mid 1980's (Pike et al., 1995). It is based on the principles that everything is either a file or file system, all communication is done over a network, and the use of privately named namespaces which can only be accessed by their owners (Hancock, 2003). Like a cloud, Plan 9 facilitates multiple users that have access to only predefined parts of the structure.

Content Delivery Networks, or CDN's, are made up of distributed servers who work together to provide fast delivery of web content (Cloudflare, n.d. a). Arguably, a CDN is not a service the end user directly interacts with, but it is a vital part of the modern Internet as it helps with load times, bandwidth cost and latency. It is also classified as its own system, even though other architectures can incorporate CDN's to improve their network traffic.

Grid computing uses the computing powers from multiple, separate machines to accomplish a joint task (Jacob et al., 2005, p. 8). A grid allows for parallel processing, and each node might be set to perform a different task to save time. Grid computing can in crude terms be thought of as the Avengers of distributed computing, as multiple different nodes work towards the same goal, while playing to their own strengths.

Fog- and edge computing are both largely used in data collection. They both make use of distributed sensor nodes that are placed in close contact to the data source, at the very edge of the network. One of the main differences between the two architectures is where the processing is done: Edge computing processes data at the very edge of the network, close to the data source itself, while fog computing is hierarchical and can perform computing anywhere between the network core and the sensor nodes (Yousefpour et al., 2019). One of the common denominators is that fog- and edge

computing both contribute to lowering latency and bandwidth costs. Like the CDN, it is also classified as its own system, but can also be incorporated into other architectures.

Lastly, wireless sensor networks and wireless ad hoc networks also share a lot of similarities with each other, just like fog- and edge computing do, and are also largely used in data collection. The architectures often consist of smaller, much more distributed sensor nodes that tend to be unsystematically spread over larger geographic areas. These nodes communicate with each other by forwarding data over wireless communication protocols. In a wireless sensor network, the nodes communicate with other nodes within its communication range, while nodes in an ad hoc network communicate directly with each other without the need for access points. They therefore have no fixed infrastructure (Sandhiya & Bhuvaneshwari, 2018). Nodes from both architectures are able to derive energy from renewable energy sources through energy harvesting (Huang et al., 2018) (Basagni et al., 2013).

## 2.3 The Green Pledge

Pollution intensified with the emergence of the Industrial Revolution in the 18th century (History, 2020), but it wasn't until the 19th century that people started questioning the effect it could have on our planet. The phenomenon now known as the greenhouse effect (Ekholm, 1901) was discovered by the French scientist Joseph Fourier and presented in his 1827 article on the matter (Fleming, 1999). After Fourier, multiple scientists further elaborated on the idea, until the Swedish scientist Svante Arrhenius published his first estimate of a man-made global temperature change caused by CO<sub>2</sub> in 1896 (Rodhe et al., 1997). Even after these discoveries were made, it took many years before the alarm bells began to ring.

In 1965, the US President's Science Advisory Committee published their report "Restoring the Quality of Our Environment" where they warned about melting ice caps, rising sea levels and acidification of water sources (President's Science Advisory Committee, 1965), but it still took years before the mainstream acknowledged the problem. The 1960's "hippie" counterculture that rejected the cushy, comfortable

middle-class lifestyle their parents had built after the second World War, paved the way for radical social and political movements, such as the environmental movement of the 1970's (History, 2021). The first Earth Day was held in 1970, and its success led to the creation of multiple environmental laws and acts in the US (Earth Day, n.d.).

The last 50 years has brought with it an increasing focus on a greener future, and we have entered what some call the fifth industrial revolution which is focused on smart green growth for the benefit of our common future (Ellen MacArthur Foundation, 2018). In 1987, the Brundtland Report, "Our Common Future", presented the concept of sustainability (Brundtland, 1987), in 2006, former Vice President of the United States, Al Gore, raised awareness of the dangers of global warming and called for immediate action in his documentary "An Inconvenient Truth" (IMDb, n.d.), and celebrities like Leonardo DiCaprio continue to use their platform to fight against climate change (Matthews, n.d.). But the younger generations still stand at the center, just like they did in the 1970's; Climate activist Greta Thunberg organized school strikes for climate change (Crouch, 2018), people are criticizing blockchain technologies and NFT's because of the impact they have in the environment, and even in a world full of fast fashion (Nguyen, 2020), the secondhand market is rapidly growing (Khusainova, 2021).

The green pledge represents the collective vow we as a global population have taken, even if it's done unconsciously on an individual level. The zeitgeist of today is characterized by secondhand shops, handmade objects, small businesses, public transport, co-working spaces, multifunctioning establishments, smart cities, renewable energy, clean eating, and sustainable living. There are of course cultural and economic differences that impact people's individual lifestyles, but the idea of constantly striving for a better life for ourselves and our descendants seems to be pretty much ingrained into us. People are looking to revert the changes we've made on the environment, and many of our daily tasks and routines reflect that by default, because the focus on a greener future can be found all around us. Not only that, but actively going *against* the green pledge and making choices that contribute to unsustainable solutions and polluting will get you shunned as you are tampering with someone else's future.

## 2.4 Summary

The cloud is only one of many examples of distributed computing technologies. A common denominator for all of them is that they are based on one architecture with predefined roles that each have their own areas of responsibility and tasks they can perform. Although some of the architectures are built on technologies that make use of green technology, none of them have been designed specifically with a green focus. Seeing that sustainability and a green future form the basis for many other areas in our lives, perhaps it's time to introduce that aspect to cloud architectures as well.

## 3 Approach

To fully prepare the reader for the rest of the thesis, this chapter will outline the actions that will be performed in order to be able to extract specific answers from the research. The research will be based on the problem statement, which was specified in the introductory chapter as follows: *Challenge the status quo by investigating the effects of a highly distributed and minimalistic cloud platform driven by a green energy focus.*

The research will be divided into two main phases: designing and presenting our cloud architectures and analyzing and discussing their planned functionalities and impact. A detailed plan for these phases will be described in this chapter, while the actual phases themselves will be further outlined in their own, upcoming chapters later in the thesis.

### 3.1 Research Approaches

There are different ways of approaching a research project, which can be divided into three main methods: comparative research, exploratory research, and review studies.

Comparative research is a research method that relies on comparing two or more things to draw conclusions related to a hypothesis. This research method is very traditional in the sense that any experiments, expected outcomes, test subjects, control groups, etc. are well defined and documented, and that there is a certain process that needs to be followed. This all ensures that articulating falsifiable problem statements can be done straightforwardly. The purpose of comparative research is to find any similarities or differences between the things that are being compared.

Exploratory research is more flexible compared to comparative research and does not require the researchers to follow a specific recipe to get their answers. It is a type of research that aims to get insight into a problem that has not yet been fully defined, but which will become clearer as the exploratory research itself happens. Exploratory research won't necessarily result in any concrete answers, and it can be difficult to determine when the research is "done". However, the requirements regarding openness



and documentation of one's choices is even bigger compared to other research approaches. This research method is ideal when a problem domain needs to be explored and understood in depth, and the answers that can be derived from this type of research might inspire other researchers to dig deeper into the problem.

Review studies, or survey studies, is a research method that bases itself upon previous research, statistical analysis, or literature. It is often used in medical research and other evidence-based domains. A specific research paper might only look at a small part of a given problem domain, but a review study will gather up all the separate puzzle pieces and create an overview of a research field's current status, based on the individual conclusions. By sampling and analyzing the available data, it's therefore possible to reach a generalized conclusion of a problem domain that can help move research forward within its field.

This thesis can best be described as exploratory research, as the problem is not yet clearly defined. This approach brings with it the potential risk of having too much creative freedom. The cloud is built on a well-established architecture, but any potential issues within the architecture have not yet been mapped out, so there are no predefined, specific problems to latch onto, meaning that it can be difficult to gauge the workload. Documenting the approach is therefore important, as it forces the researcher to really think about their plan and to determine whether they are doing enough, or if there are clear deficiencies in their planned approach. The thesis also contains aspects of comparative research, as certain aspects of alternate cloud architectures are set up against today's infrastructure.

To address the risks outlined above, we have set clear limits for ourselves when it comes to what we will focus on during this thesis. These focus areas have been partially described in the "Problem Statement"-section, and will also be presented in more detail in the upcoming sections of this chapter.

## 3.2 Research Objectives

The objective of this thesis is to design and present different variants of a highly distributed and minimalistic cloud platform architecture that challenges today's status quo, and to discuss their functionality and potential impact on their end users and physical environment. The upcoming presented architectures will be used as the basis for the discussion of how the status quo is being challenged, and the exploratory investigation process encapsulates the whole thesis as this is all speculative and not grounded in actual implementations. During the analysis and discussion sections, the presented features and functionalities will be discussed in order to answer the exploratory questions that have been derived from the problem statement itself, if possible. These questions are as follows:

- In what way can we most drastically challenge the status quo?
- What effect does a green energy perspective have on a highly distributed and minimalistic cloud platform?
- Is a highly distributed and minimalistic cloud platform realistically feasible?

## 3.3 Results

The “Results”-chapter will contain the presentation of three highly distributed and minimalistic cloud platform architectures. For each of the proposed architectures, we will go into details about role assignment and how the architectures handle unreliability and running certain services. In addition to this, we will also go into details about some common features that apply to all the proposed architectures, regardless of how they are structured. These common features concerns are all mainly about how the nodes themselves function, both based on their software and hardware capabilities.

Taking time, cost, and workload into consideration, it's not feasible to create the setup needed for an actual distributed cloud platform, which is why we have to make do with exploratory research and discussions in order to be able to draw conclusions that are actually worthwhile. The end goal is to have expanded the horizon and gained insight

into how an imagined, distributed, minimalistic cloud platform could potentially work, and not to actually build such a system ourselves.

### 3.4 Analysis

The “Analysis”-chapter will look at the modeling and presentation of the proposed architectures through an expert lens. In it, we will place the architectures on our architectural scale, compare them to each other and discuss which of them is the most groundbreaking approach, i.e., which one is the most likely to challenge the current status quo. We will also determine which one of the approaches is the cheapest, and the most realistic. The most groundbreaking, the cheapest and the most realistic approach will in all likelihood not be the same.

Lastly, we will present a large-scale adaptive architecture which is a combination of the other architectures, and has a better chance of challenging the status quo on its own.

### 3.5 Discussion

The “Discussion”-chapter will include a backwards look at the entire thesis, starting at the end of the analysis chapter and ending up at the introduction. Throughout the chapter, we will highlight and discuss important points that have been made throughout the thesis. These points include, but are not limited to wasting resources, green energy as a fundamental assumption and the impact of wireless communication.

### 3.6 Conclusion and Future Work

The “Conclusion and Future Work”-chapter will be based on what we discussed and arrived at in the “Analysis”- and “Discussion”-chapters. Here, we present the conclusions for the exploratory questions we presented under the “Research Objectives”-section. As this is the last chapter of the thesis, it will also include a summary of what has been done, as well as a summary of future work that is potentially needed in this problem area.

## 4 Results

In this chapter, we will present different architectures that are intended to challenge the way the cloud is set up to work today. For each architecture, we will discuss how it handles role assignment, unreliability, running services, and how it compares to today's cloud. There will also be an analysis that will determine if the architecture is *radical* enough to challenge the status quo. However, before we do that, we will have a look at some common features we assume are being shared by all the different architectures that will be presented.

### 4.1 Common Features

We envision that all the proposed architectures share some common features, no matter how the nodes relate to each other or how they communicate. These common features are related to how the proposed architectures handle what roles each node can take on, it's capability towards energy harvesting from renewable energy sources, how the nodes themselves are smaller and more minimalistic than traditional cloud computing processing units, downtime, and resilience, how the nodes are geographically distributed and how they communicate with each other, as well as how the services are run on containers. These common features will all be described in further detail in the upcoming sections.

#### 4.1.1 Roles and General Role Assignment

In our architectures, there are three different types of roles a node can take on: controller, compute, and support (IBM, 2021). As mentioned, these are the types of nodes a cloud architecture needs at its most basic level. Having a certain role means having certain responsibilities, and those responsibilities might entail running actual, cloud-specific services.

A controller node can be thought of as a manager in the architecture. It is responsible for keeping track of compute nodes and their status, automating device operations

through configurations and container image updates, and identifying potential issues, and suggesting fixes to the problems, among other things (Cisco, n.d.).

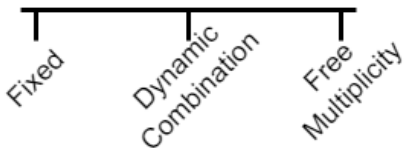
A controller node can also be responsible for directing the traffic to the compute nodes, but having this additional responsibility increases the chance of a total collapse of the cloud if it were to disconnect. This is because we not only lose the ability to keep track of the compute nodes, but they also do not receive any data and will essentially be useless. By only keeping track of the compute nodes, and not also being responsible for the network flow, the compute nodes will continue to work to some extent even if the controller nodes were to disconnect; We just wouldn't be able to see its status or update any configurations until everything is back to normal. Multiple nodes in an architecture can be controllers, this means that there can be more nodes that are responsible for organizing the cloud architecture itself.

A compute node is what we typically think about as a worker node. It is responsible for running services deployed by users, like websites or data processing, and it serves as the endpoint where the cloud users can access these services. If a compute node is inaccessible, the services it's running are also inaccessible to its users, unless there are systems in place where another compute node shares the workload. This is typically done by implementing capabilities like load balancers, where multiple compute nodes run the same service, and users are directed to different nodes in order to distribute the incoming network traffic (NGINX, n.d.).

A support node supports the other two types of nodes, but it's mainly responsible for supporting the compute nodes. It can provide things like different storage solutions, key-value stores, and caching for the cloud itself. Support nodes can also serve as intermediate waypoints and help deliver data packages to the correct places, asynchronously.

The assignment of the controller-, compute- and support roles can be done in three main ways, with additional variations for each architecture which will be described in

more detail under each respective section, further down in the thesis. The three main ways to do role assignment are *fixed*, *dynamic combination* and *free multiplicity*. These three possibilities are in reality points on a scale, as displayed in Figure 4.1. Fixed- and free multiplicity role assignment each constitutes an end point, while dynamic combination role assignment makes up the midpoint. More detailed examples of how role assignment can be done will be given further down, when we present how roles are assigned for each of the different architectures.



**Figure 4.1:** The role assignment scale

Fixed role assignment indicates a predetermined assignment as either a controller-, compute- or support role. If a node were to disconnect, it will be assigned the same role once it re-connects to the network. The predetermination can for example be grounded in hardware capabilities, physical location etc. One of the benefits of fixed role assignment is that since everything is preplanned, there is little to no need for a restructuring of the architecture when a node becomes inaccessible, which is something that can be a costly endeavor because certain nodes (storage- and computing nodes) would have to get all the data they need to run their respective services. This won't be needed if a node is re-assigned the same role time and time again. In a sense, fixed role assignment can therefore be cheaper than the other alternatives, as the data doesn't have to continuously be reassigned to new nodes. However, while fixed role assignment is a completely valid way to do role assignment in certain situations, and constitutes the norm in any conventional cloud today, it breaks with one of the fundamental qualities we have envisioned for our architectures, which is the dynamic aspect. As fixed role assignment facilitates an architecture where the structure is preplanned, and a controller node will forever be assigned as a controller node, there will be no algorithms or processes in place to analyze the architecture and dynamically assign roles, which is not what we envision.

Free multiplicity role assignment indicates, as the name suggests, a free role assignment to many roles simultaneously. With free multiplicity role assignment, we assume that each node can be assigned up to three roles at the same time, which would mean it is responsible for performing all the different tasks that belong to each one of the roles it's currently assigned. As this comes with a high need for processing power, there needs to be some sort of system in place in order to limit how much each node is actually doing in order to keep the nodes small in size, which is a feature we get more into further down. However, just because each node *can* be assigned up to three roles at the same time, it doesn't mean they always will be. The actual role assignment is based on processes that analyze the different capabilities of the nodes and designates them with the different roles. This means that each node can gain and lose a role at any given time based on its current qualities and the state of the cloud.

Dynamic combination role assignment lies right between fixed and free multiplicity role assignment. It somewhat incorporates features from them both; A given node can be reassigned the same role again and again, but it will always be based on an analysis of the node's current capabilities. Unlike in the free multiplicity role assignment, dynamic combination role assignment does not allow a node to have multiple roles at the same time, but rather facilitates an architecture where the nodes dynamically switch between the roles and their responsibilities.

As both free multiplicity and dynamic combination role assignment take the node's current capabilities into account when assigning them as a controller-, compute- or support node, they are both at risk of becoming much more expensive than, for example, the fixed role assignment. However, they facilitate a much more dynamic environment, which is what we want for our proposed architectures.

#### 4.1.2 Energy Harvesting

Energy harvesting, or power harvesting as it's also known as, is the process of deriving energy from external sources and storing it in small, wireless devices which then uses

that very energy storage in order to function. Being directly connected to an energy source is what makes the devices able to be truly wireless, as they don't need to be connected to an external power grid to get energy.

The energy is harvested from renewable energy sources, such as solar power, thermal energy, wind energy or hydropower. This means that the devices can be directly connected to objects like wind turbines, watermills, and solar panels, and that they store and run on a percentage of the total harvested energy those objects collect. The harvesting hardware could also be attached to the node, making it a fully integrated capability.

We assume that the different nodes in our proposed architectures rely on energy harvesting. This is what makes the high level of distribution, which will be discussed more further down, possible. This way, controller-, compute- and support nodes can be placed right on top of buildings, inside or next to rivers and waterfalls etc., ensuring close proximity to the cloud's end users. The problem with energy harvesting dependent nodes, however, is that they are completely reliant on the natural environment around them to ensure uptime. If the sun doesn't shine on a solar panel, if the water levels are low, or if the wind is at a standstill, there will not be any energy produced, and the nodes will consequently be unavailable. This will in turn limit the capacity of the cloud, and potentially make services temporarily inaccessible.

Being placed outside will also put the nodes at the mercy of the "weather gods" as the risk of wear and tear is considerably higher than if they were to be sheltered by a roof, four walls and stable temperatures. A resilient architecture, as described in the previous section, is therefore vital to keep our cloud up to par with already existing solutions.

A more unconventional, though still highly functional, way of looking at energy harvesting is to connect devices to objects that produce energy as a byproduct, such as exercising equipment. If a cloud computing node were to be connected to a treadmill or a stationary bike, its availability would be limited to whenever people use that equipment



to exercise, which would be the only time energy to charge the batteries would be produced. Even though it's possible to build an architecture that makes use of nodes like these whenever they are available, we have not based our architectures on this exact approach; Energy harvesting, in the context of this thesis, only relates to a direct connection to a natural, renewable energy source. This means that none of the energy will go to waste during transport or through performing tasks like cooling etc. Excessive energy also won't be produced, all of which results in a lowered carbon- and general environmental footprint, consequently resulting in a truly green cloud.

### 4.1.3 Small Nodes

Thanks to the technological innovations over the past years, as we described in relation to Moore's law earlier on in the thesis, processing units today are much smaller and more powerful than they were just a few years ago. They are also much more affordable. If we look at the processing power of a phone or a Raspberry Pi compared to the first computers, we can easily see just how far we've come.

A Raspberry Pi 4 Model B, which was released in 2019, has up to 8GB RAM, a Quad core Cortex-A72 64-bit CPU, support for both ethernet and wireless connection, and a Micro-SD card slot for data storage (Raspberry Pi, n.d.). Micro-SD cards come in sizes up to 1TB (Athow & Hanson, 2022). A Raspberry Pi is also built so it can be upgraded with different add-ons, for example Bluetooth connection and 5G modems. An Apple iPhone 13 Pro Max, which was released late in 2021, has 6GB RAM, a Hexa-core CPU, support for wireless and Bluetooth communication, and up to 1TB storage space (GSMArena, n.d.). Both these devices are considerably small; Approximately 5.1 x 8.6 cm and 16 x 7.8 cm, respectively. The Kenbak-1 from the 1970's on the other hand, which is recognized as the first personal computer (Computer History Museum, n.d.), had 0.000000256GB RAM (256 bytes), did not have a CPU as it used transistor-transistor logic, and had an average execution speed of below 1000 instructions per second. The device was approximately 10.8 x 48.9 x 29.2 cm in size, which is not huge like the first computers which took up entire rooms (Department of Computer Science and Statistics, n.d.). Still, its processing power and speed compared to the Raspberry Pi

and iPhone shows the extent of the technical advances that have happened over the past 50 years.

Modern phones and microcomputers like Raspberry Pi's are exactly the kinds of devices we assume make up the nodes in our proposed architectures; Small, powerful, and relatively affordable. Arguably, not all the nodes will necessarily be of the same type and size, as we will get more into under the sections for each respective architecture, but they will commonly be smaller and less technically complicated than today's cloud standard. This, in combination with high distribution of the nodes (which will be described in further detail in the next section) and energy harvesting, means that the nodes can be hidden in plain sight, so as to not serve as visual disturbances and use energy which no one else uses that does not come from a power plant.

It's also not unthinkable that two or more nodes can be connected to the same energy source and reside in the same area. There can for example be a box with slots for multiple Raspberry Pi's, which allows the hardware to be easily put in place and connected to the same energy source; Sort of like how it's done in a server rack or a computer case. Thanks to the small size, even the boxes that house many of them can be quite small and not take up much space.

#### 4.1.4 Availability

If a node were to disconnect from the network, there needs to be a system in place to keep the cloud architecture from collapsing, which would result in faulty services that the users won't be able to use sufficiently, let alone access.

We assume that when a node disconnects and becomes unavailable, it's either because something unpredictable happened, like some sort of a physical event affecting it, resulting in a hard disconnect, or it was scheduled to happen due to certain criteria being met, like running low on battery power, resulting in a more graceful disconnect. These criteria could be anything from recognizing that the energy is

currently expensive, to being done with performing a task, to having close to no users accessing the services, for example at nighttime.

A hard disconnect means disconnecting from the network without any prior warnings. The cloud would suddenly lose a node and its services and processing power. At times like these, it's vital to have some sort of system in place that eliminates the risk of a complete crash. A cloud architecture that allows its nodes to do a graceful disconnect, as we call it, will have these kinds of systems in place. This way, the nodes can foresee the circumstances that will cause them to disconnect and send some kind of signal to their controllers in order to warn them of the upcoming change. This requires sensors and some sort of monitoring service so that the nodes can look at trends and historical data to determine when it's most beneficial to disconnect from the network. This is the kind of data each node wants to communicate to its controllers for them to be aware of which nodes are most favorable to use. Each architecture could deliberately choose to disconnect a node if its energy source is not efficient or cheap enough at the current time, and there are other nodes that would be more beneficial to use. In totality, graceful disconnects are nothing new and most clouds have the feature in place to manually put a node into "maintenance mode" But here, we envision a level of automation to allow nodes to signal their impending disconnect, and the controllers take action.

Because of everything that needs to be in place to successfully disconnect from the network without leaving it in shambles, the disconnect itself will absolutely be an intended action, unlike the hard disconnect. We assume that all the nodes in our proposed architectures have the kind of system in place that is needed to support a graceful disconnect, so that we at all costs can avoid a potential hard disconnect from the rest of the network.

#### 4.1.5 Containers

Containers are small, independent units of software that packages up code and its dependencies, so applications can run equally as quickly and reliably from one

computing environment to another (Docker, n.d.). There are different types of container tools that will both containerize and run the applications, like Docker and Kubernetes.

We assume that all the nodes in our proposed architectures support containers. This is in no way a radical proposition. Container technologies are used worldwide by millions of users and different operating systems each day, so it's a well-ingrained phenomenon in the industry today. This also means that in terms of software platforms, our proposed architectures are unable to challenge the status quo of cloud computing, since we are basing them on existing technologies.

Having containerized applications will help when re-downloading code to a node once it has been assigned a role that requires it (support- and compute nodes). Containers are highly portable and moving them to a new place won't really affect how well the application runs. Containers also increase the resilience of the architecture, as they can quickly and automatically be turned on again if they for whatever reason stop running.

#### 4.1.6 Resilience

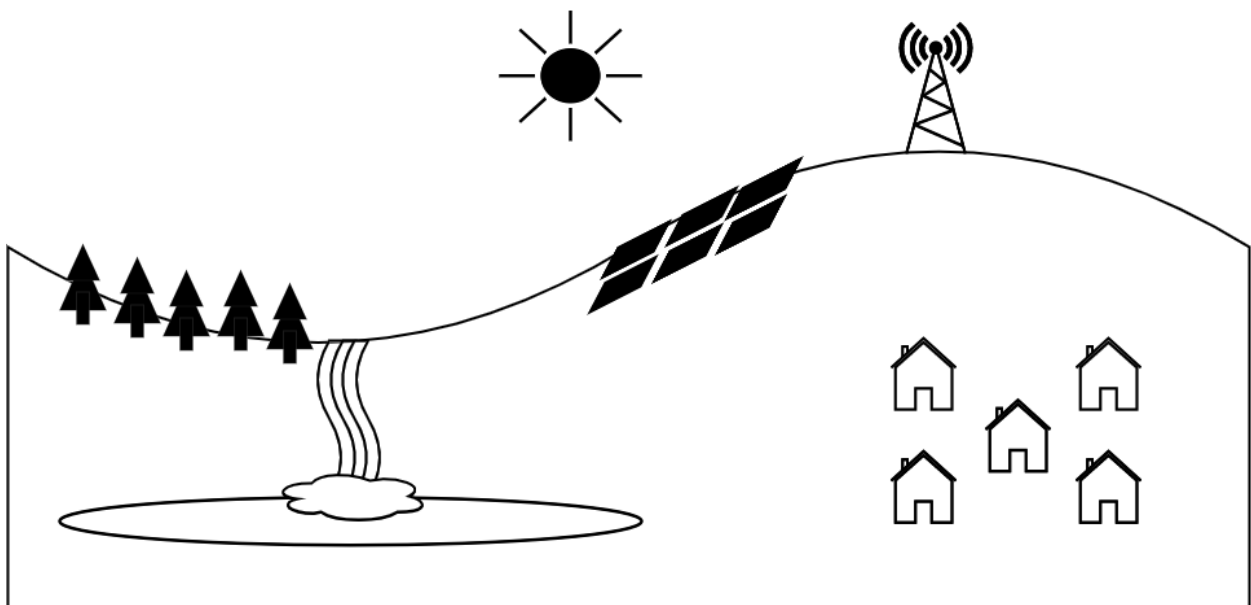
Resilience is defined as “the capacity to recover quickly from difficulties” (Oxford Languages, 2022b), which is an important quality to have for nodes in a cloud architecture. If a node can quickly get back online, it will limit the amount of time the cloud has low processing power, no database connections or lack of controller nodes.

Processes that increase the resilience of a node and ensure that they get back up quickly after a disconnect are especially important in the case of a hard disconnect, as the node hasn't had time to prepare itself and the rest of the cloud architecture for its departure. If there are external factors that cause a node to disconnect without warning and turn off, the processes behind the resilience would somehow have to be built into the hardware. If the node has lost power, it needs to be turned on again as soon as it has access to power again, or if it somehow got turned off and disconnected for another reason, there needs to be some sort of timer built in so it can be regularly tried to be turned on again.

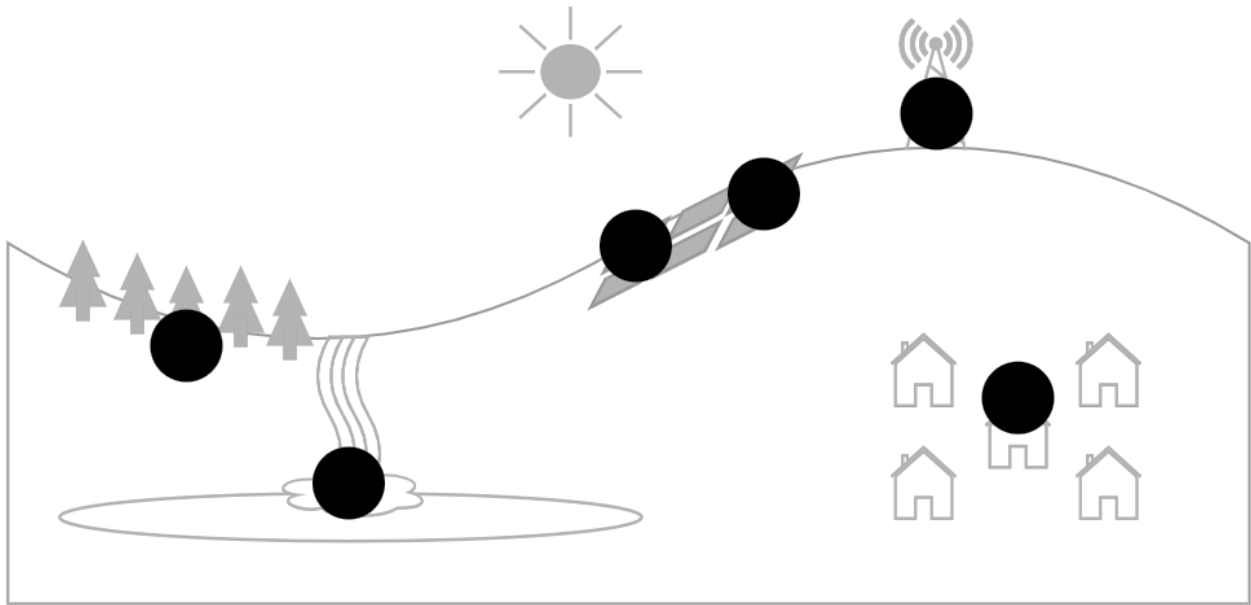
Resilience also applies to the containers that run the actual services cloud users will make use of and is a phenomenon that is already built into container software, such as Docker (Docker Docks, n.d.). Having resilient containers means that whenever a service goes down, the container responsible for that service will automatically regenerate and bring the service back up. Because this happens so quickly and seamlessly, the end user will most likely not even realize the service they wanted to access was unavailable for a fraction of time.

#### 4.1.7 Distribution and Wireless Communication

In all our proposed architectures, we assume that the nodes are spread out over relatively large geographical areas, stretching from cities into forests, hills, and deserts, resulting in a highly distributed architecture. The figure below depicts a landscape with a forest, a waterfall, solar cells, a radio mast, and a small city. The following figure depicts where nodes might be placed in such a landscape. These nodes are just examples of geographic distribution, and they do not follow the upcoming color coding, nor are they yet connected to each other to follow an architectural pattern.



**Figure 4.2:** Landscape



**Figure 4.3:** *Nodes in landscape*

Data on the Internet is moving at the speed of light, and some users might not think too much about loading times and delays. But as latency, that is the time it takes for data to pass from one point on a network to another, is impossible to completely eradicate due to distance and Internet infrastructure equipment, long distances between nodes due to high distribution could negatively impact the user experience. If there are instabilities in some areas, latency will also increase. From an economic perspective, this could result in less development in those areas, because of the inevitable problems that will come along anyways. A high amount of latency results in poor performance and can prompt a user to stop using a service due to frustration. It also negatively affects things like search engine optimization (Cloudflare, n.d. b). Therefore, it should be minimized, if possible.

Many physical devices in many different places also means that there are many points of potential failure. Any physical maintenance or complete replacement of the devices will need a lot of manpower, which is a costly and time-consuming endeavor.

An important point we make is that regardless of any problems brought forth, a truly highly distributed architecture is what makes our proposed architectures so radically

different from everything that exists today and is truly what makes trying to challenge the current status quo even possible. Distribution and energy harvesting might therefore be the two most important features of the ones we present, in addition to wireless communication which forms the basis for the network.

We expect our nodes to communicate through 5G and/or 4G signals. This, in combination with energy harvesting, allows the nodes to be completely wireless, which again will impact their ability to be highly distributed. Devices like radio masts need to be part of the communication network to help forward the traffic, as the nodes can't transfer all the data over large distances on their own. The radio masts themselves need to be distributed across the whole geographical area that makes up the cloud architecture to make sure every node has the same base level of communicability.

The "G" in 4G and 5G stands for "generation". 4G and 5G are both two generations of a broadband cellular network technology, where 5G is the newest addition (Verizon, 2022). Each new generation builds on the preceding generations, so 5G can be considered the "best one" so far. 5G has the capacity to handle high speed communication and high data rates with up to 100Mb per second downloaded and 50Mb per second uploaded for wide area coverage and supports up to 1 million connected devices per square kilometer (Australian Government, 2017). Even though 4G isn't as technically advanced with its 10Mb per second data downloaded, it still has incredibly fast download speeds (iSelect, n.d.). In addition, they can both be supported simultaneously. This means that large-scale architectures with many wireless nodes won't be a problem.

## 4.2 Aligning Our Common Features to The Architectural Scale

Earlier, we presented examples of different distributed computing architectures. These architectures, in addition to the cloud architecture, can be placed along a horizontal scale where they go from more fixed to more distributed. The two extremes are made up of the cloud (more fixed, and what we want to get furthest away from) and wireless

ad hoc networks (more distributed, and furthest away from the cloud architecture). Figure 4.4 depicts this scale, which we have decided to name *the architectural scale*.



**Figure 4.4:** *The architectural scale*

We wanted to investigate how the different features that have been presented up until this point will affect the architectures, and where they can be placed on the architectural scale as a result. As we haven't moved into specifics about each individual architecture quite yet, and as there are three of them that are going to be presented, we won't be able to place them on a specific point on the scale yet, but it's still possible to determine the general area of where they belong. This will be done by structurally going through each one of the features in the previous section (role types, general role assignment, energy harvesting, small nodes, availability, containers, resilience, high distribution and 4G/5G communication), and see if each one of them pushes our architectures more towards the fixed end of the scale, or the more distributed end.

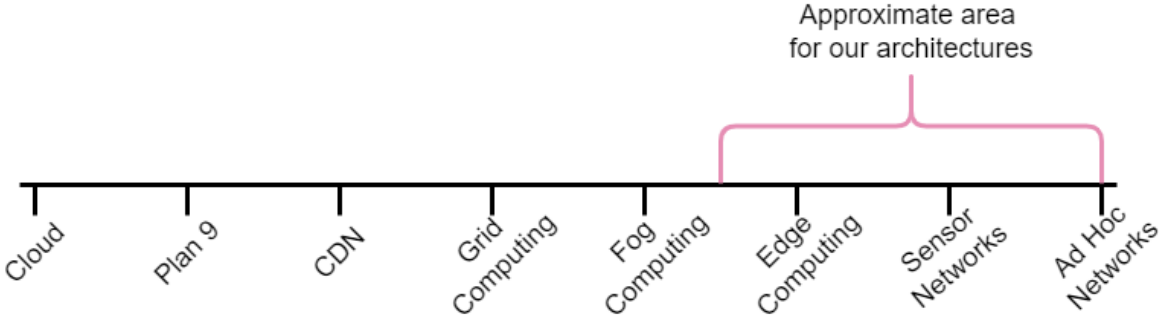
The table below displays the aforementioned features and which direction on the scale each one of them pushes our architectures, and Figure 4.5 shows the approximate area on the scale our architectures reside on, before we go into further detail for each architecture further down, where their individual placement will likely change.

Feature	Direction
Role Types	Has no implication on direction
General Role Assignment	Towards the more distributed end (Right)
Energy Harvesting	Towards the more distributed end (Right)



Small Nodes	Towards the more distributed end (Right)
Availability	Has no implication on direction
Containers	Has no implication on direction
Resilience	Has no implication on direction
High Distribution	Towards the more distributed end (Right)
4G/5G Communication	Towards the more distributed end (Right)

**Table 4.1:** Directional impact on the architectural scale from the common features



**Figure 4.5:** The approximate area our architectures will reside on

As we can understand from Figure 4.5, our proposed architectures likely share a lot of similarities with edge computing, wireless sensor networks and wireless ad hoc networks, simply based on the common features. They also share some general similarities with fog computing, since fog- and edge computing are so alike, which is why the approximate area also covers the halfway point between fog- and edge computing. Role types, availability and resilience has no real impact on the direction as these are all qualities, we expect to have in a modern cloud platform architecture. All the other qualities (general role assignment, energy harvesting, small nodes, roles, high distribution, containers and 4G/5G communication) push the architectures towards the more distributed end. This means that they challenge the status quo, as no qualities push the architectures towards the cloud architecture on the fixed end. Challenging the

status quo of the cloud involves proposing an entirely different architecture to what exists today, and thus the architectures have to be as different as possible from today's data center structure.



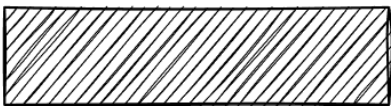
## 4.3 Architectures

We propose three different architectures that differ in the way of connectivity and role assignment; A hierarchical architecture, a meshed architecture, and finally a graphed architecture, which is split into two sub-architectures due to differences in portraying their individual qualities. The architectures share some common features related to role types, availability, container technology, resilience, energy harvesting, minimalism, distribution, and communication, as have been described in the sections above. There are also similarities related to role assignment and the potential disconnect of a node from the cloud. For each proposed architecture, we also suggest a variety of options that would change the final solution, resulting in a salad-bar-inspired ability to create one's own architecture. The different options will also be described in further detail under each section, respectively.

As a reminder to the reader, we will again reiterate that the proposed architectures are not yet complete blueprints that can be used in order to implement alternative ways of handling cloud computing, but are more so intended to make the reader aware that there is a possible variety of design patterns.

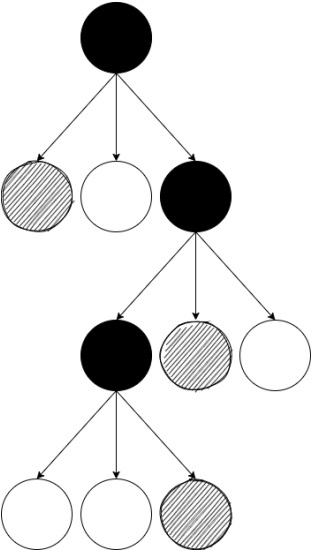
### 4.3.1 Architecture 1: Hierarchical Architecture

In the hierarchical architecture, the nodes are configured in a hierarchical pattern. Figure 4.6 shows a section of the architecture with an uneven structure, where some branches are deeper than others. The figure includes all three role types (controller-, compute- and support roles), and their individual color coding is described in the table below. This color coding will also be reused for every upcoming figure depicting both the hierarchical architecture, the meshed architecture, and the graphed architectures.

Role	Color Pattern (Explanation)	Color Pattern (Actual)
Controller	Solid black	
Compute	Solid white	
Support	White with black stripes	

**Table 4.2:** Color coding for nodes in upcoming figures

The arrows between the nodes in Figure 4.6 indicate which nodes have control over which. Normally in a figure like this, the arrows could also represent the dataflow in the architecture, but not in this case. As we described in the “Roles”-section earlier on, there are certain problems that could occur if the controller nodes are also in charge of the dataflow. As we want to limit the number of potential problems that could occur, the controllers in our architecture will only be in charge of managing the other nodes, while the communication and dataflow happens through wireless communication, as described in the “Communications”-section.



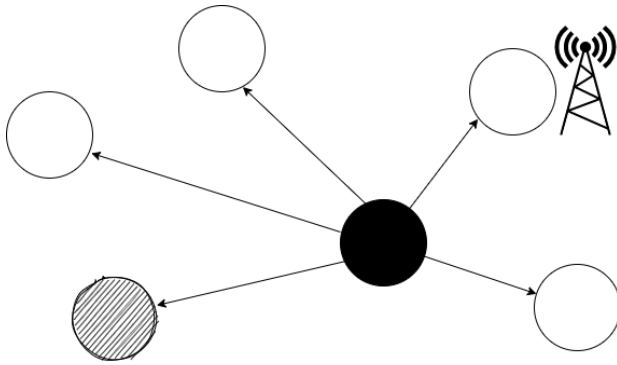
**Figure 4.6:** A section of an uneven hierarchical architecture

As the figure shows, the hierarchical architecture would potentially allow for an uneven hierarchical structure. This means that some branches can be deeper than others, and therefore also contain more nodes. In turn, this means that two controller nodes at the same level of the hierarchical architecture could have control over a different number of nodes downstream. As long as the controller nodes have the capacity to manage all the nodes, they set to have control over, there likely won't be any issues. Allowing for an uneven hierarchy also means that it's easy to add new nodes to the architecture, as they can be placed at the bottom of an existing branch without having to restructure the whole architecture in order for it to be rebalanced.

#### 4.3.1.1 Role Assignment

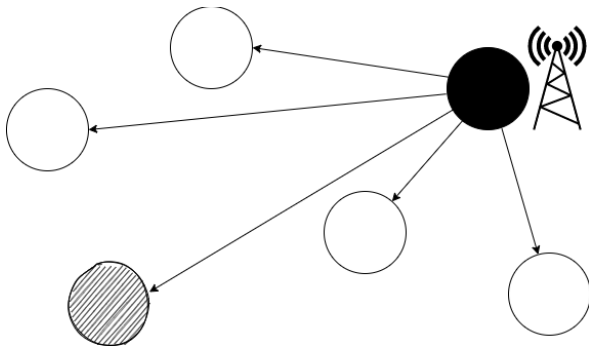
We assume that the role assignment in the hierarchical architecture will be based on dynamic combinations. In this case, it means that the different nodes have the possibility to be assigned different roles at different times but will only ever have one role at a time. It also means that the entire architecture could change based on certain criteria (while still retaining a hierarchical structure), which will be discussed in the upcoming "Handling unreliability"-section.

We assume that the role assignment will be based upon factors such as geographical distance, or specified capabilities, like signal strength for communicating with other nodes. In the following figures, Figure 4.7, and Figure 4.8, we see the same six nodes which are placed in the same physical configuration. In Figure 4.7, the nodes are hierarchically configured based on geographical location, where the controller node is given its role based on its position relative to the other nodes. Here, the controller is placed approximately the same length from each one of the leaf nodes.



**Figure 4.7:** Hierarchical role assignment based on geographical location

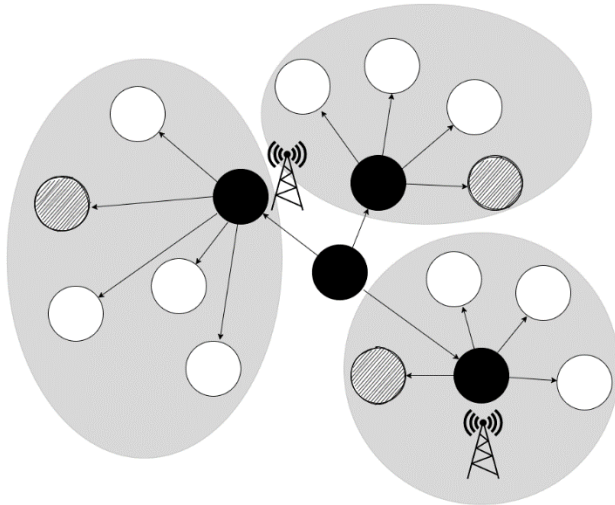
In Figure 4.8, the nodes are hierarchically configured based on a given arbitrary capability, in this case signal strength caused by close proximity to the radio mast. Even though the nodes are geographically located in the same way as in Figure 4.7, here the top right node fulfills the criteria which makes it more eligible to serve as a controller.



**Figure 4.8:** Hierarchical role assignment based on signal strength

As the cloud architecture grows, more nodes would be added. To keep control over all the nodes, we assume that the final stage of the hierarchical architecture will implement both a geographical- and a capability-based role assignment, resulting in a large-scale layered hierarchical architecture, as depicted in Figure 4.9. Here, the architecture is first hierarchically divided into different geographical zones based on geographical, to limit the nodes that are talking to each other. Without such a zoning approach, two nodes with a large geographical distance could end up being connected, which could result in

different forms of delay and lack of optimization. Within the zones, any node could theoretically be assigned any role based on its capabilities.



**Figure 4.9:** Large-scaled hierarchical role assignment based on both geography and signal strength

We've talked about assigning roles based on capabilities throughout this chapter, but what are the actual processes behind this type of role assignment? It would require an architecture-wide algorithm that analyzes each node's status as it's being communicated. This algorithm would take the different capabilities into consideration when deciding what roles to assign.

Each node will have some level of resilience, signal strength, connection to affordable green energy, etc. For the hierarchical architecture, we propose that the *least* stable nodes, with the *highest* chance to become disconnected, should be assigned the controller role. This could potentially be achieved by implementing a variable that communicates each node's probability of failure, and those with a high probability are more likely to be assigned as controllers. This is a radical approach, and something that is not often seen in architectural design, as it might not be entirely intuitive to base the architectural design around unreliability.

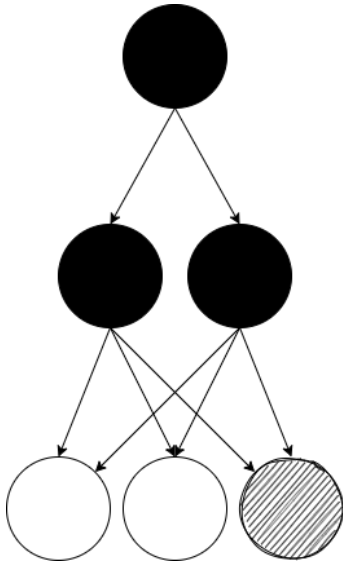
By having the most unstable nodes serve as the controllers, we ensure that the more stable nodes will be assigned as support- and compute nodes. This way, databases and

the actual services which users deploy are less likely to go down, as the relevant nodes themselves are most likely to be up. It is of course not ideal to have the controller nodes be unavailable from time to time, but that is in no way the intent. However, if the architecture is in place, the roles have been assigned and the nodes have been initiated, we consider the controllers to be the least important role type out of the three. If a controller were to disconnect, it would not immediately affect the running services as they all run on different nodes, and the only thing we would lose is the current status and the ability to do updates and restructuring of the architecture. The other kinds of nodes, as we have hinted at earlier in this paragraph, are more important to keep up as they are the ones responsible for handling the actual dataflow, and therefore it's important to have processes in place in order to handle unreliability.

#### 4.3.1.2 Handling Unreliability

One of the most important things when it comes to handling unreliability is to have a system in place that limits the amount of restructuring that must be done. Restructuring of the architecture involves re-assigning roles and changing the structure of which nodes that communicate with each other. If the architecture is frequently restructured, it could be costly as the workload each node has to do is increased (as previously described in the "Roles and General Role Assignment"-section).

Temporarily losing a controller node is not the end of the world, but it's still something we want to avoid to be able to have a realistic overview and be able to do real time monitoring of the architecture. The simplest way to increase the chance that a node is always connected to a controller node, is by assigning at least two controllers to each node. This way, there will be a backup node in place if one of the two controllers were to temporarily go down. Figure 4.10 displays how a section of the full architecture would look connected to two controllers, instead of just one.

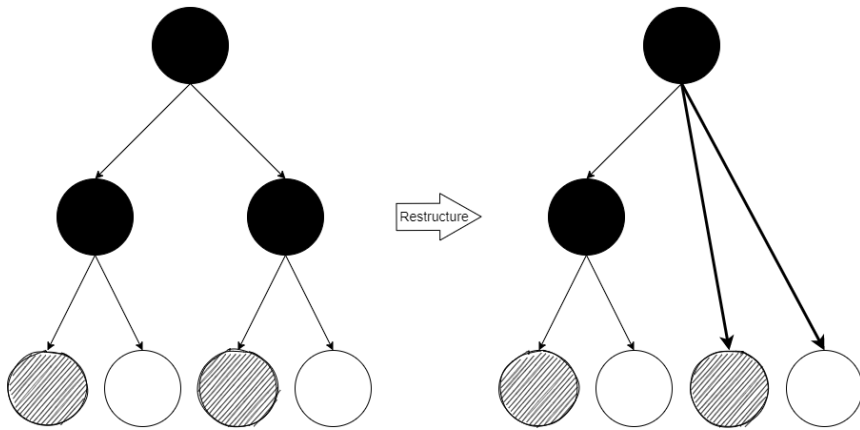


**Figure 4.10:** A section of the hierarchical architecture where two controllers control the same nodes

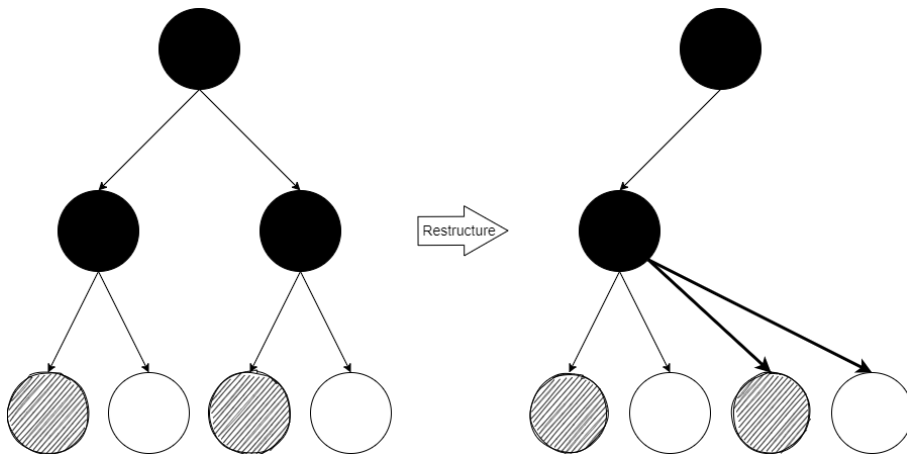
If one of the two controllers were to disconnect, the other nodes are already being managed by the second controller. Strictly speaking, this approach requires that there are more controller nodes than are needed in order to manage the whole cloud architecture. This results in us “wasting” resources, as the second controller could theoretically be assigned as a controller and provide services. However, this approach also dramatically limits the chance of architectural collapse with limiting the time a node is completely without a managerial controller node.

Other approaches need to be backed by more sophisticated algorithms that will partially restructure sections of the architecture until the whole thing is too unstable. If too many nodes disconnect, larger parts of the architecture will of course have to be restructured, but as this will be expensive, it's something we want to avoid. Figure 4.11 and Figure 4.12 below depicts two additional approaches to handling the loss of controller nodes.





**Figure 4.11:** A controller at a higher level takes over the responsibility



**Figure 4.12:** A controller at the same level takes over the responsibility

In these approaches, a completely different controller node will take responsibility for the nodes when a controller disconnects. These approaches require that each controller node has sufficient capacity in order to take responsibility for additional nodes, so that every leaf node will still be monitored and connected to the network in order to communicate its status and get updates.

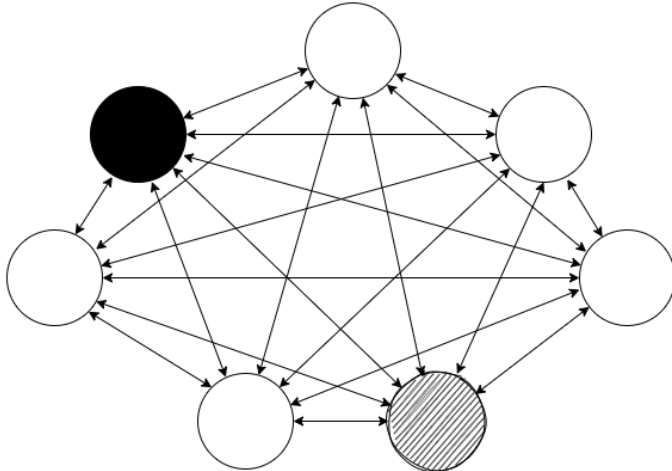
The three recently described approaches were initially imagined to be working separately, and that only one of them would be incorporated into the architecture at a time. However, by introducing all of them to the architecture at the same time and by tailoring the algorithms to take effect in different situations, we are enabling dynamic

unreliability-handling. First, we are limiting the amount of time a restructuring has to happen with two controllers, and if the second controller disconnects, the nodes will be taken under the wing by another controller based on that node's current capabilities. Here, as everywhere else when talking about capabilities, are we talking about things like level of resilience, signal strength, green energy production and connectivity to other nodes. These kinds of capabilities are dynamic and can change over time, which is why role assignment is based on the *current* capabilities.

We have talked at length about controller nodes, and little about what to do when losing a support- or compute node. If one of these node types were to disconnect, actual services would be implicated, and it will also result in more work for the controllers as they have to get the architecture successfully back up and running. One of the easiest ways of avoiding that is by having backup nodes. These backup nodes will either run the same services at the same time, like a pod in Kubernetes, at a layover when one node goes down, or they will serve as a classic backup solution for stored data, all depending on what type of node it is. What this means in practice is that there must be at least two nodes of each type, no matter how large or small the architecture is. In larger cloud architectures, this will of course be easier than in a smaller one.

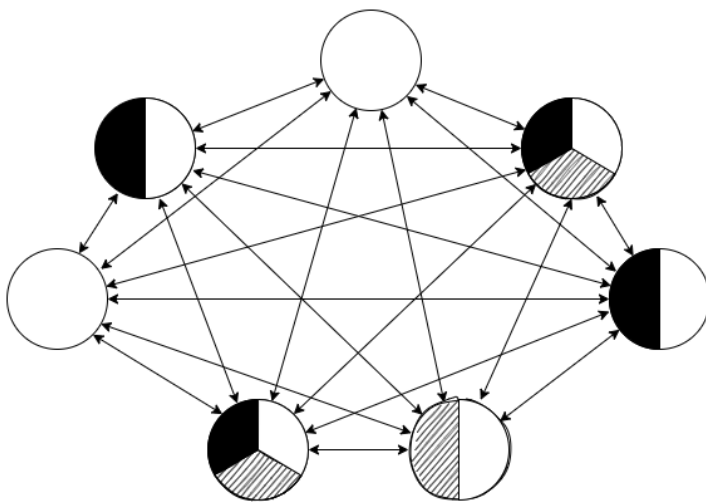
### 4.3.2 Architecture 2: Meshed Architecture

The nodes in the meshed architecture are, as the name suggests, configured in a mesh. Figure 4.13 displays an example of how the meshed architecture would look if there were 7 nodes in total. In a mesh, every node is connected to each other. This means that for a total of  $n$  nodes, each node has  $n-1$  links that connect it to the other nodes in the same network. The arrows in this figure, like with the one we presented for the hierarchical architecture, represent which nodes that have control over the other nodes and communicate with each other and. The arrows have nothing to do with actual data flow as all the communication and flow of data is done through wireless communication.



**Figure 4.13:** Meshed architecture with simple role assignment

Every node in Figure 4.13 is assigned *one* role in order to simplify the figure and be able to present an architectural overview more clearly and how the nodes control each other. The nodes in Figure 4.14, however, are assigned up to three roles simultaneously, which is a more complicated and realistic view of how the meshed architecture will actually work. In both Figure 4.13 and Figure 4.14, the lines that connect the white compute nodes are a representation of communication and software networking so that certain services think they are on the same physical network (VMWare, n.d. b). This is also reflected by the double arrows that go back to the controller nodes.



**Figure 4.14:** Meshed architecture with more realistic role assignment

#### 4.3.2.1 Role Assignment

In order to actually assign the different roles, there needs to be processes in place, so the architecture knows what role to assign each node. In contrast to how it's done in the hierarchical architecture, the nodes in the meshed architecture can have more than one role assigned to them at any given time, and they will simultaneously be able to perform the tasks required for each of those roles. This means that the meshed architecture is based on free multiplicity role assignment.

Like with the hierarchical architecture, the role assignment in the meshed architecture can be based on either geography or dynamic capabilities specific to each node. If a given node is geographically closer to a collection of other nodes, for example by having approximately the same physical distance to all of them, that node is more likely to be assigned the role of controller. Any updates of container images or communication of status will be less affected by latency issues etc. due to the close proximity. The closer distance also affects the required signal strength, and less required signal strength means less energy used.

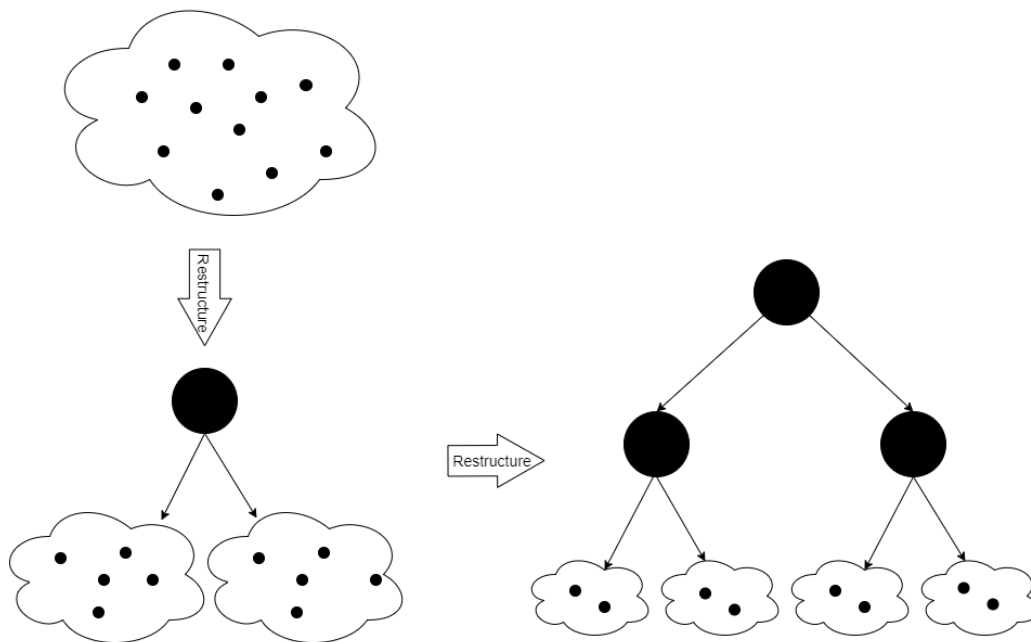
Role assignment based on dynamic capabilities can be caused by many different things, as previously mentioned. As the nodes communicate their status, the algorithm takes the current capabilities into account and determines if a restructure needs to happen to make the architecture cheaper or greener in any way. In the case of a disconnect, the algorithm also bases its decisions on these capabilities. Since a node can be assigned multiple roles, the free multiplicity role assignment algorithm must base the role assignment on multiple different capabilities. It is therefore less likely for a node to serve as both a controller and support node, as these two roles are based on completely different capabilities (most unstable and least unstable, respectively).

#### 4.3.2.2 Handling Unreliability

Overloading nodes could cause a possible disconnect from the architecture and should therefore be avoided. One way of lessening the workload of a node could normally be to add additional nodes that could take over some of the workload, but in this

circumstance, more nodes is a problem in itself. As the architecture becomes bigger and bigger with additional nodes being added, it will require more work for each node, and they might get closer and closer to their maximum capacity. This is because each node is connected to all the other nodes and has to use a certain percentage of their available capacity to send their status and keep in contact with all the other nodes.

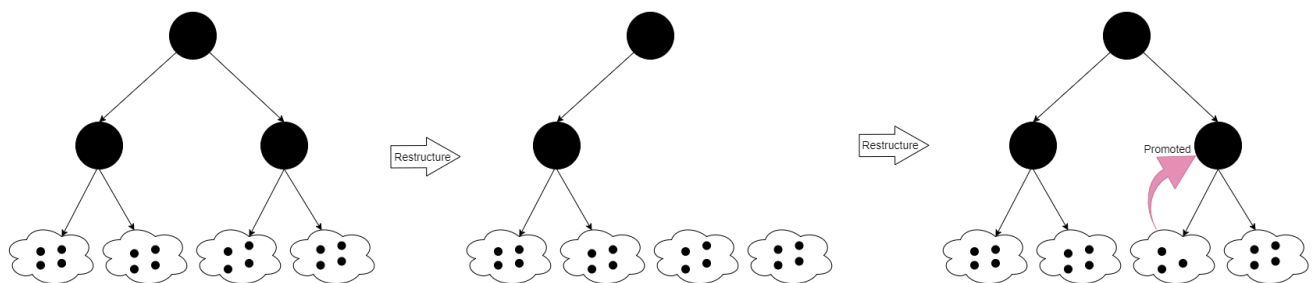
We propose the following solution to this potential problem: In order to avoid nodes being disconnected due to being overloaded because of a large mesh with many nodes, the meshed architecture is able to split into two meshes that are connected by a controller if the total capacity of the nodes reaches a certain level. Figure 4.15 displays how this would look, where the mesh is displayed as an undistinguishable cloud of nodes. Here, the original mesh is split into four sub-meshes through a process similar to cell division and ends up forming its own hierarchical structure. The meshes themselves make up the bottom layers of the hierarchy, while the other nodes are made up of high-capacity controllers known as master-nodes. It is also not unthinkable that some meshes further down in the hierarchical meshed architecture split, while others do not, which results in an uneven hierarchical mesh.



**Figure 4.15:** A mesh restructuring itself into a hierarchical mesh

The master-controllers are more stable than the internal controllers, as they are responsible for communicating status and configuring updates across multiple meshes. When restructuring the architecture from a single mesh to a hierarchical mesh, the capability-based role assignment algorithm will take the new context into consideration, and assign highly reliable nodes as master-controllers, instead of the unstable ones.

Even though they're highly reliable, the master-controllers might still disconnect from the network. If this happens, there will either be a partial restructuring similar to the ones we presented for the hierarchical architecture (Figure 4.11 and Figure 4.12) where the meshes would simply be switched over to an available master-controller at the same or higher hierarchical layer, or a new node from one of the available meshes will have to be appointed to master-controller, and consequently be promoted out of the mesh it currently resides within. Figure 4.16 depicts how this promotion looks.

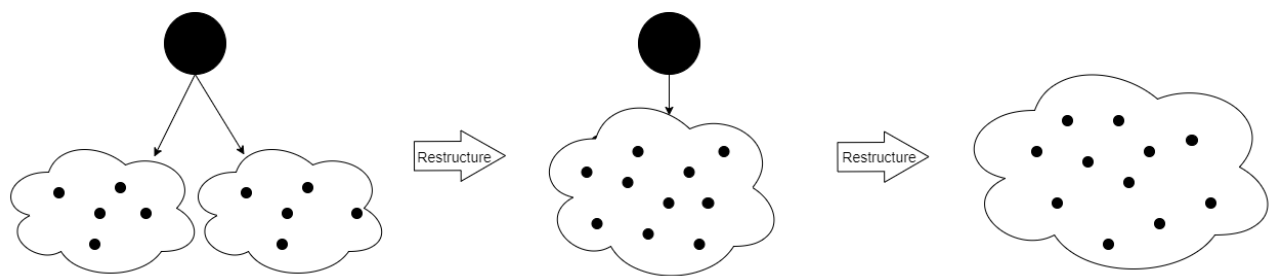


**Figure 4.16:** A node moves out of a sub-mesh after being promoted to a master-controller. While this happens, the sub-meshes are disconnected from the rest of the architecture, but can operate within itself.

If there is only one controller node inside the mesh, and this node disconnects, a new node will be assigned as controller. This assignment will be based on a line of succession, where the next in line based on its current capabilities will be assigned the role. Unlike in the hierarchical architecture, this requires no restructuring of the architecture because all the nodes are already connected, and a newly appointed controller can make use of these existing connections. There will still be a short time period before a new node is assigned the controller role where there are no controllers in the mesh, and just like in the hierarchical architecture, the best solution for this is to

always have two or more controllers in the architecture in order to limit the amount of time the cloud architecture will be un-monitorable and un-updatable.

Additionally, if there are few nodes within the sub-meshes in a hierarchical mesh and the mesh is in a generally unstable state, the architecture might benefit from reverting back into a standard mesh structure. This is depicted in Figure 4.17, and ensures that the nodes that are left can again be in strong communication with each other, instead of being thinned out in a less-than-ideal, not as strong architectural pattern.

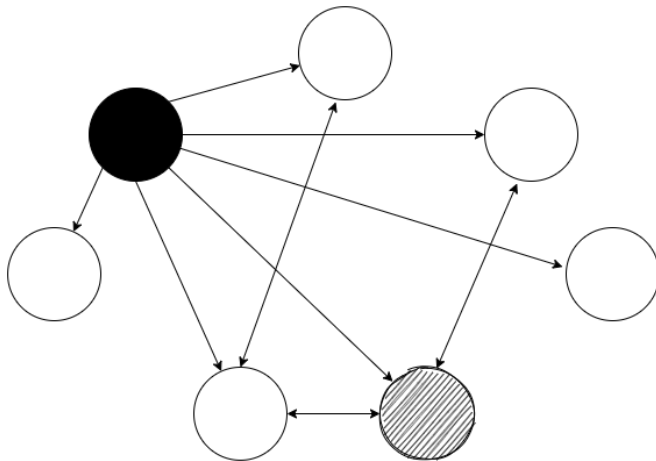


**Figure 4.17:** Hierarchical mesh reverting back into mesh

In the situation of losing a compute- or support node, the same principles as we discussed under the “Hierarchical Architecture”-section apply here as well. As actual services will most likely be impacted if such a node disconnects, there needs to be at least two nodes that are running the same services or storing the same data, depending on what kind of node it is, so that there is always an available backup node in place. If more than one node has the necessary data available, and especially if these nodes are in relatively close range to each other, it doesn’t really matter which one is available and which one isn’t. This is the same idea that is behind clustered file systems, where two or more physical devices simultaneously mount the file system so it can be accessed and managed as one single system (Weka, 2020). Like with the hierarchical architecture, always ensuring enough computing power in order to perform tasks, while also implementing permanent backup solutions will be easier in larger architectures, since there are more nodes to choose from.

### 4.3.3 Architecture 3: Graphed Architecture

The nodes in the graphed architecture are configured in a graphed structure. A graph has a lot of similarities to a mesh, but while a node *can* be connected to all the other nodes, there are no requirements that say they *have to be*. In a graph, some nodes might be connected to a double-digit number of nodes or more, while others might only be connected to one. Figure 4.18 displays an example of how the graphed architecture might look if there were 7 nodes in total. Like with the two previously presented architectures, the arrows in Figure 4.18 represent which nodes that have control over the others and which nodes that communicate with each other and says nothing about actual dataflow. The arrows between the white compute nodes, a type of node that can not be in control of another node on its own, visualize phenomena like software networking and status communication.

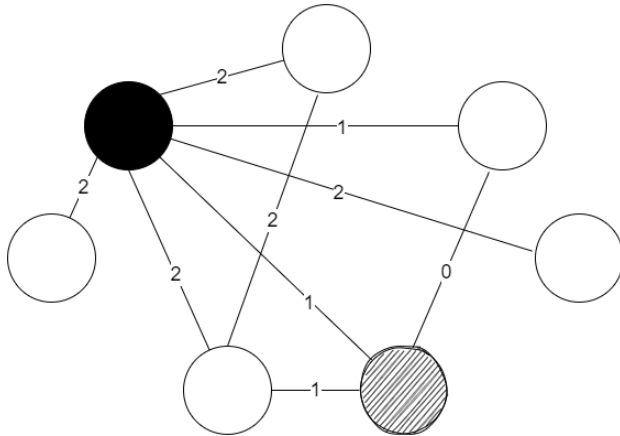


**Figure 4.18:** A non-weighted graph architecture

Something that is special about the graphed architecture, is that there is a completely different way of visualizing the structure of nodes and their connective links, that looks dissimilar to previous figures altogether. Figure 4.19 displays a structure where each node is seen in relation to the radio mast it uses to communicate to other nodes. A link between two nodes means that they have access to the same radio mast. Here, the links between the nodes are weighted and non-directional, unlike what they were in Figure 4.18. Also unlike previous figures, Figure 4.19 actually displays the architecture

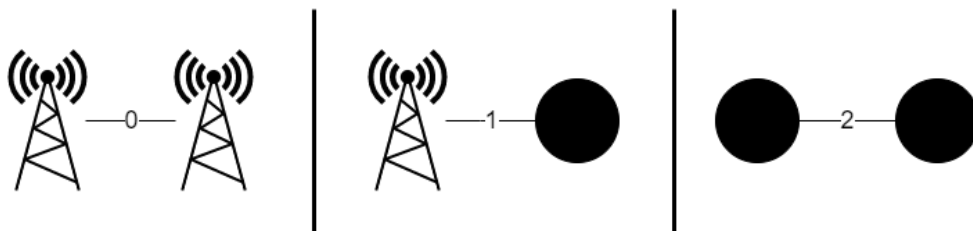


through the lens of actual dataflow. Because this approach is so different from the other approach, we chose to call this a weighted graph architecture, while the structure that was presented in Figure 4.18 is a non-weighted graph architecture.



**Figure 4.19:** A weighted graph architecture

The weight of the links between two nodes is determined by how many jumps the data package must perform in order to get from node A to node B. Two distributed nodes that are in communication with a radio mast would have a weighted line of 2, as the package has to travel from node A, to the radio mast, to node B, for a total number of two jumps. If one of the nodes is directly connected to the radio mast (and the radio mast itself has a solar panel or wind turbine connected to it to generate power) the total number of jumps would be one, resulting in a weighted line of 1. Having two such nodes communicating with each other resulting in a weighted line of 0 might seem counter-intuitive, but we argue this because the radio mast is not a data storage solution, but simply something that helps transport the data packages to their final destination. Figure 4.20 depicts the weighted lines between the type of nodes we just described.



**Figure 4.20:** Different weighted lines as a result of different types of nodes

The weight of the links is arguably based on the type of technology that helps forward the packages, and not on signal strength or distance. A lower weight number means closer connection to the radio mast, and as the nodes in our architecture communicate through wireless communication, their distance to the radio mast would impact how the algorithm assigns roles.

#### 4.3.3.1 Role Assignment

We assume that both graph architectures will be based on free multiplicity role assignment, but the weighted graph architecture also shares similarities with dynamic combinations or even fixed role assignment. The free multiplicity role assignment algorithm can assign additional roles or completely reassign the structure whenever, based on the node's current capabilities that are being signaled through the wireless communication. The role assignment algorithm in the weighted graph architecture, however, is less likely to reassign roles, causing a more fixed architecture.

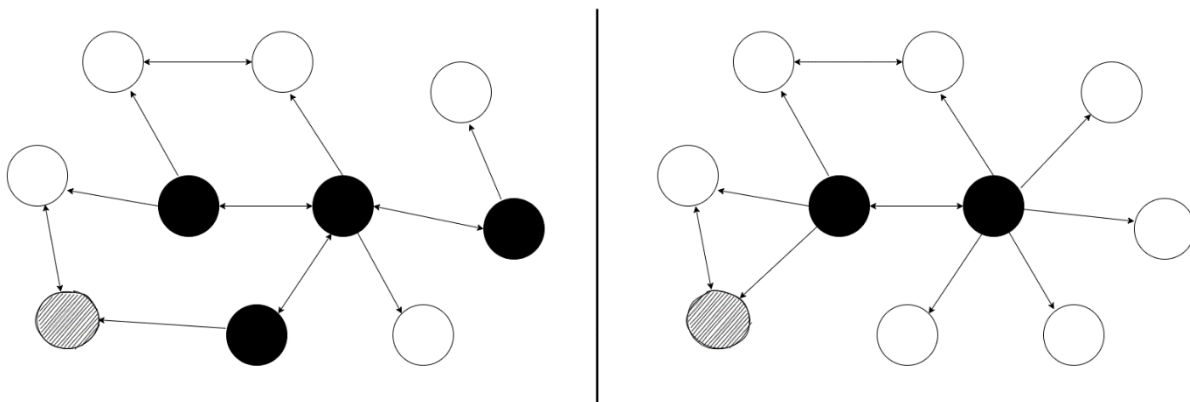
The weighted graph architecture has a higher focus on reliable communication, and nodes that are near the radio mast are more likely to be assigned the support role. This will for example help with quickly sending data from the databases to the other nodes. However, just because a node is either directly connected to a radio mast, or just one jump away, does not mean that the node is not at risk of disconnecting. This is where the weighted graph architecture breaks with previously stated role assignment processes and allows the role assignment algorithm to take other qualities into consideration, like the type of technology available and future desired behavior.

The role assignment algorithm for the non-weighted graph architecture is, like with the previously presented architectures, more likely to assign the controller role to more unstable nodes, and the support role to the most stable nodes. The remaining nodes must be assigned the compute role, and the algorithm has to take the needed amount of processing power into consideration when assigning roles. There can not be too few compute nodes, or the cloud will be unable to sufficiently offer services. This is less

likely to be a problem thanks to the free multiplicity role assignment, as all other node types can also be assigned the compute role and be part of the workforce.

In the non-weighted graph architecture, the role assignment shares similarities with both the hierarchical and meshed architecture. It is partially based on geographical location, in order for nodes to be in close proximity to the nodes they are connected to and to lower the potential latency, and partially based on the node's current capabilities. The base configuration and physical placement of nodes could look the same as in a mesh, but the end goal is to have a different constellation of communicative lines between the nodes. The nodes in the graphed architectures are likely to follow the same justification for controller role assignment as the other architectures regarding close proximity, to lower latency and increase the efficiency of the cloud.

No matter how many nodes are assigned as controllers in the meshed architecture, they have to be assigned in a pattern where every node is reached in the most efficient way without potentially wasting resources. Figure 4.21 below depicts two different configurations of role assignment for the same physical node structure. The figure depicts a non-weighted graph architecture, but the same principle applies for the weighted graph architecture as well.



**Figure 4.21:** *Inefficient vs. efficient role assignment*

On the left side, four nodes have been assigned the controller role due to their capabilities. As we can see, the two rightmost controller nodes are connected, but the

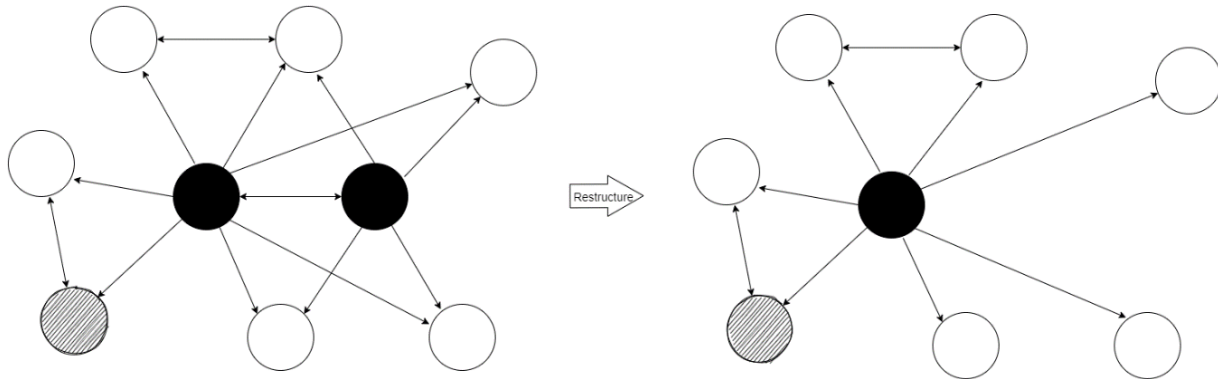
rightmost controller is only responsible for managing one node. On the right side of the figure, we see the same physical structure of nodes, but a different configuration of roles. Here, the rightmost controller node is assigned the compute role instead, and the middle controller is responsible for managing all the nodes on the right-hand side, which are all assigned as compute nodes. This figure illustrates that just because the role assignment algorithm is set to follow certain rules, there needs to be additional rulesets in place that enforces the most efficient resource use. Like with all the previously presented architectures, there needs to be processes and rules in place to handle unreliable nodes that are at a risk of disconnecting.

#### 4.3.3.2 Handling Unreliability

Completely eliminating the chance of a node disconnecting is impossible, due to both external and internal factors like natural catastrophes, hardware failure or code errors. Both the non-weighted and weighted graph architecture share similarities with the other architectures, and some of the ways of handling unreliability are the same as well.

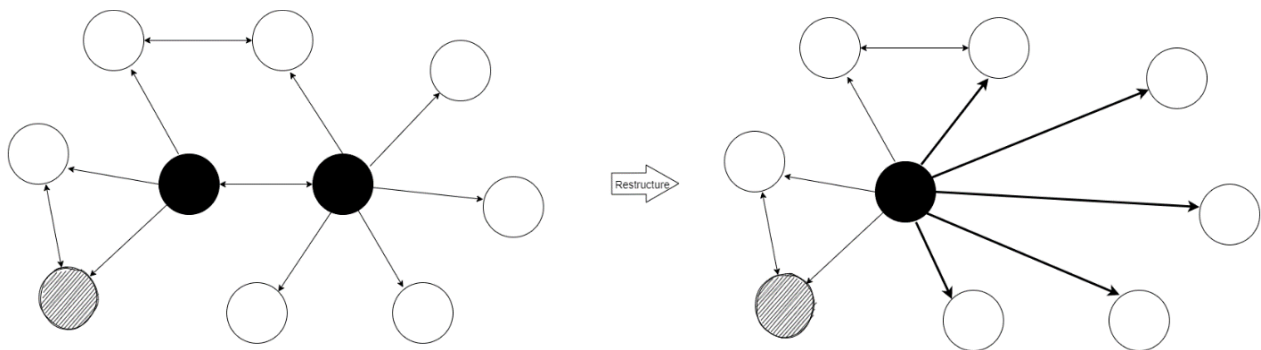
If more than one node has the controller role, and one of them disconnects, any potential restructuring and reassignment of roles will depend on whether the remaining nodes are still being managed by a controller or not. Figure 4.22 depicts a situation where this is the case. Here, the rightmost controller disconnects from the architecture, but since all the nodes it was managing were already also being managed by the middle controller, no restructuring needs to happen. By implementing a constant backup controller node for each node, there is less chance for the cloud architecture to temporarily lose contact with a given node, if one of the controllers disconnects. As we also talked about when we presented the hierarchical architecture, a constant backup controller results in a lowered chance of losing contact with other nodes, and thus results in a more realistic overview with the ability to do real time monitoring. On the other hand, as we also previously mentioned, assigning more nodes as controllers than those that are strictly needed in order to do the job results in a potential “wasting” of resources. Here we will again stress that this is a smaller problem in an architecture that

uses free multiplicity role assignment, as controllers can also serve as compute nodes and therefore participate in the cloud workforce.



**Figure 4.22:** Example of restructuring in the non-weighted graph architecture when a controller disconnects, but all the nodes are already controlled by a second controller. No new lines are drawn.

If a given node is just connected to one controller, however, and that controller disconnects, the architecture needs to be restructured in order to regain control over the lost nodes. One option is of course to just wait until the original controller node gets back on, but this is not ideal as there is no way of knowing how long it will take. Therefore, the best solution is to have certain processes in place that will ensure that other controllers can take over the managerial role. Figure 4.23 displays how this might look. To the left, we see the original structure, and to the right we see the restructure that happens after the rightmost controller node has disconnected; The left controller has taken over the responsibility for the remaining nodes.



**Figure 4.23:** Example of restructuring in the non-weighted graph architecture when a controller disconnects, and the nodes have to be switched to a different controller

The disconnect of a support or compute node has, as already described, much more dire consequences, as actual cloud services will be implicated. As mentioned, the easiest way to avoid unavailable services is to implement backup nodes that will ensure that the content is available even if the original nodes go down. To avoid services such as websites, game services etc. becoming unavailable, multiple nodes need to be able to run the same services, so whenever a node potentially goes down, there is always another node ready to take over. The same applies for support nodes that run databases or control software networking etc. If nodes like these were to go down, and there are no nodes ready to take over the workload, we either have to sit and wait until the node comes back online while the cloud users become increasingly irritated, or the controllers need to be able to quickly send new instructions to different nodes in the architecture so that *they* can download the data that is needed in order to provide the services. This will of course take longer time than what an already configured backup solution would, but thanks to the controller, compute and support nodes working together, the cloud architecture is able to get the services back up and running again.

#### 4.3.4 Handling Services

As there are similarities within the different architectures when it comes to configurations, communication and node control, there are bound to be similarities in how they handle the different services. Therefore, we chose to collect the information in this section to avoid unnecessary repetition for each respective architecture.

It's not enough for the cloud architectures to be stable and to be able to keep its nodes online; The nodes need to be able to run services that can be accessed by the end users, as well as services that will be used by other nodes (including services for communication, monitoring, software networking etc., all of which we have briefly touched upon before).

Let us look at how the different architectures would handle running websites, game servers, streaming services and data caching. Three out of four services are services end users will typically make use of, while data caching is an important service that

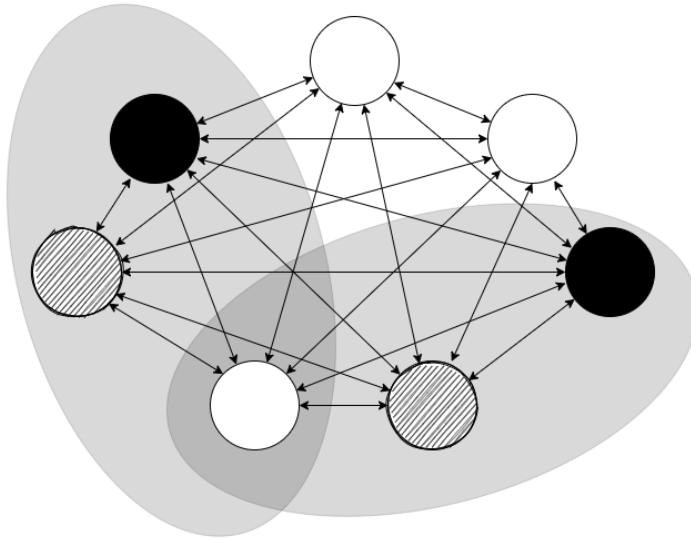
helps with making data available faster and allow for a potential increase of bandwidth speed as there is less data making its way across the network. By looking into how each of the different architectures handle running these services, we get an insight into whether or not they can be a viable architecture for a new type of cloud.

#### 4.3.4.1 Websites

A website needs content, hosting, and an IP address. When it comes to accessing websites, a lot of the traffic is dominated by sending and receiving files that are located on the web servers, or in some cases on dedicated databases. These files include code and CSS, images, videos, and sometimes API data. A lot of modern websites use JavaScript (Elliott, 2019), which can be executed in the user's browser (JavaScript, 2021). Not only can JavaScript interpret content, but it can also access images, design, and API data. This technology helps with eliminating the amount of data that has to be transferred from the web server, as transferring code files that can be interpreted on the user's end is a lot more efficient than transferring actual data-files, like HTML. On top of that, a lot of the content that is viewable on a webpage can be cached, and therefore more quickly accessed, which we will get more into further down.

A website is vulnerable both if every part of the technology stack is located on the same node in the architecture, *and* if it's spread out over multiple nodes. This is because if one of the nodes goes down, the website will become inaccessible. However, if we treat the connection between a controller node and its subjects the hierarchy the same way we treat a pod in Kubernetes, i.e., a group of one or more containers with shared storage and network resources and a specification for how to run the containers (Kubernetes, 2022), we are able to build a stable structure that can accommodate hosting a website efficiently. In these pod-like structures, the different node types work together and are responsible for running services. We can introduce load balancers that are able to deliver websites from multiple different web servers (F5, n.d.) in case one or more of the compute nodes goes down. Additionally, close proximity to their preferred support node allows for quicker and more seamless data transfer.

For the hierarchical architecture, a pod-like structure will be made up of the lower levels in the hierarchy, while the meshed- and graphed architectures have pod-like structures that can cross into each other. Since nodes can be connected to multiple controller nodes, each support- and compute node that is needed to successfully run a website can therefore be part of multiple pod-like structures, as displayed in Figure 4.24.



**Figure 4.24:** A meshed hierarchy where the bottom compute node is part of two overlapping pod-like structures, represented by the greyed-out areas containing one of each role.

One problem with the pod-like structure for the hierarchical architecture is that certain websites might only be accessible from certain nodes that are located on certain branches. If someone who is geographically further away from those nodes wants to access the content, it will impact latency. A solution to this is to make the same content available from multiple branches in the hierarchy. As the websites run on containers, and as the infrastructure needed to host a website will probably be available on multiple branches as a default, the only thing needed is to transfer the container images. The same applies to the meshed- and graphed architecture, where different nodes in the architecture will be able to serve the same content. If one node goes down, a Domain Name System (DNS) will be able to redirect the user's traffic to the one still available.



#### 4.3.4.2 Game Servers

A dedicated game server is needed for online multiplayer gaming, or cloud gaming. For most online multiplayer games, the end user needs to use a game client that connects them to the main game server. The communication between a game server and a game client is a two-way street; The server transmits data about the game's current state to the client to allow the users to view an accurate, real-time version of the game, while the client sends the users input so that the server can process that data. Cloud gaming, on the other hand, happens entirely on a dedicated server, and the only thing the user needs is a screened device they can use to access the games.

A game server does not need to always be connected to a controller, as the service lives and operates relatively independently. However, thanks to containers, it's possible to dynamically scale the service if more people are trying to access it through deploying additional containers, which *would* require a connection to the controller and storage.

Game servers which are actively engaged in a game are extremely vulnerable to restructuring, as large amounts of data would need to be moved, which will impact how the game is running. Even with the addition of a transition phase, where a different node gets ready to take over the hosting while the original node is getting ready to disconnect, the potential problem of constantly having to move things is still relevant, and the experience for the end user would without a doubt be implicated. Therefore, game servers are mainly dependent on strong and stable compute nodes with low chances of disconnecting thanks to factors such as efficient green energy, availability, and resilience. All of these have been described as common features and are therefore not related to only one proposed architecture. This means that as long as any of the architectures have access to reliable compute nodes with a low chance of disconnecting, they should be able to offer game servers.

The weighted graph architecture brings additional assessments into consideration; To limit lag, a game server should be able to quickly communicate its state to the users.

Therefore, compute nodes with a lower weighted line, meaning that they are closer to a radio tower, are more likely to be chosen to operate as game servers.

#### 4.3.4.3 Streaming Services

Streaming is a way of viewing video or listening to audio content without downloading the files onto your device (Cloudflare, n.d. c). In most cases, the actual media files will be located on a server and transmitted to the client when they themselves want to access it, but through live streaming, the content will be produced and transmitted in real time. Streaming media players, the software used to consume streamed content, ensures that a few seconds of the media is loaded ahead of time, so that it can play smoothly and continuously without interruptions. This is known as buffering. However, over slow connections or on networks with large amounts of latency, streamed media can take a long time to buffer, which results in a stuttering and choppy experience. For live streaming, this can result in the client receiving the content at a considerable delay. However, streaming performance can be improved, and buffering time can be reduced if the transmission can go through a Content Delivery Network (CDN) service, which has caching at the heart of it (Imperva, n.d.). We will get more into details about caching further down.

In our proposed architectures, the content that will be streamed will be stored on the support nodes. Then, whenever someone wants to access the content, the compute nodes send a request to the support nodes and receives the streaming media content. Live streaming will be done peer to peer by two or more compute nodes. To select the nodes that are best suited for streaming services, we look at their geographical placement. Nodes near radio masts should ideally be support nodes so that data from the databases can be accessed and transmitted fast and efficiently to the streaming clients, but compute nodes would in many instances (such as live streaming) need a good connection as well.

#### 4.3.4.4 Caching

Caching is a technique that stores a copy of a given resource and serves it when it's requested. It can be thought of as a temporary storage solution that will most likely be in closer proximity to the user than the actual origin where the data is stored long-term. If a network is using caching services and a user is requesting some sort of data, the cache will intercept the request and return a copy of the stored resource instead of redownloading the resource from the originating server (MDN Web Docks, 2022). This helps improve latency and will greatly improve the time it takes for the user to access the data. It also contributes to a potential increase of bandwidth speed as large amounts of data does not have to make their way across the whole network since they are now located in a shorter distance to the radio mast which results in fast and efficient transmitting.

Caching services are responsible for storing web content such as images and database information, as well as video and audio content. If data has not been accessed in some time, it can be deleted from the cache to make space for more frequently used content.

A caching service eliminates the need for streaming services and game servers to be located on nodes in close contact with the radio masts, as the caching services can take over some of the workload. This does however require that the caching services are located on those nodes instead. The same argument we made earlier where we specified it's hard to pinpoint a specific node that would do a sufficient job in the hierarchical, meshed, or non-weighted graph architecture applies here as well, while the weighted graph architecture is visualized in such a way that it's easy to pinpoint the best fitting nodes because of their weighted lines.

## 4.4 Summary

In this section, we will present a summary of all the important concepts we have presented in this chapter. The table below will include the different concepts and their meaning. It will also distribute the concepts into different sub-categories, like for example “roles” and “architectures”.

Category	Concept	Explanation
Scales	Role Assignment Scale	The scale containing the three role assignment methods.
	Architectural Scale	The scale containing the different distributed technologies on which our architectures will be placed and compared to the others.
Role Assignment Methods	Fixed	Predetermined role assignment, and a node will only ever be assigned the same role.
	Dynamic Combinations	A node can only have one role at a time, and can be assigned the same role again after disconnect, but it can dynamically change based on each node’s status.
	Free Multiplicity	A node can be assigned up to all three role types simultaneously, but will have the responsibility for all their included tasks. Can also change dynamically.

Roles types	Controller	A manager in the architecture, responsible for keeping track of other nodes and their status, updating container images etc.
	Compute	A worker, responsible for running services deploys by users.
	Support	Supports the other nodes through offering storage, key-value stores, caching etc.
Architectures	Hierarchical	An architecture with hierarchical structure where a controller is at the root of every new branch.
	Meshed	An architecture where each node is connected to all the other nodes.
	Non-weighted Graph	An architecture where the nodes <i>can</i> be connected to all the other nodes, but they don't have to. Less strict than mesh.
	Weighted Graph	An architecture based on the dataflow and the technology that helps forward that data. An alternate way of visualizing the graphed architecture.

**Table 4.3:** Summary of important concepts

## 5 Analysis, and the Introduction of Plasticity

In this section, we will further analyze the three proposed architectures and get more into details about how they relate to each other, other technologies, and today's cloud architecture. We will also investigate which idea is the most groundbreaking in comparison to today's cloud architecture, which one of the proposed architectures is the "cheapest" approach in terms of actual cost, and the most realistic approach, i.e., which approach is the most likely to be functionally implemented. The approaches won't necessarily consist of the same architectures.

Lastly, we present a new type of architecture which can be seen as a union of the previous ones. It is dynamic and ever-changing and takes each of the individual architecture's strengths and quirks into consideration when dealing with role assignment, communication and dataflow. As we will see in more detail further down, this architecture introduces the phenomenon of plasticity, which is a quality that enables it to take different contexts into consideration and change its architectural pattern.

### 5.1 Placement on the Architectural Scale

The architectural scale, which was originally presented in the "Results"-chapter, is made up of a horizontal scale with eight points that represents different architectures that go from more fixed on the left, to more distributed on the right.

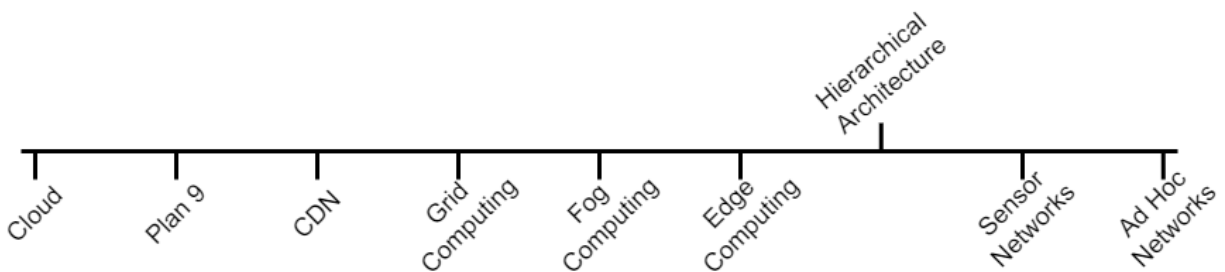
In the "The Architectural Scale"-section, before any of the architectures were even presented, we scoped out a general area that the architectures would potentially fit into, based on the common features we had presented. This area (displayed in Figure 4.5) goes from the midway point between fog- and edge computing, all the way to wireless sensor networks, which is the most distributed architecture on the scale.

Now, let us use the details we have presented about our three proposed architectures, as well as details about the different architectures already on the scale, to further identify our architectures exact position on the architectural scale. Because we split the

graphed architecture into two sub-architectures (non-weighted and weighted graph architecture), both of them will get their own individual placement on the architectural scale. For each architecture, we will create a new rendition of the architectural scale, until we finally have a new, complete scale where every architecture is included. This final architectural scale is displayed in Figure 5.4.

The hierarchical architecture fits somewhere between edge computing and wireless sensor networks. The edge computing architecture makes use of distributed nodes and processes to lower latency and save bandwidth, just like the hierarchical architecture. However, our architecture does not fit the traditional data center structure, which edge computing still largely makes use of, and it's also not quite as independent as the wireless sensor network, as it relies more on spontaneous formations of networks (Sandhiya & Bhuvanewari, 2018). While the network formation in the hierarchical architecture can also change frequently, it's not up to the nodes themselves, and the architecture-wide role assignment algorithm is responsible for defining a temporary infrastructure that defines which nodes have control over which other nodes, until the next time the architectural pattern changes.

Figure 5.1 displays the architectural scale with the addition of the hierarchical architecture.

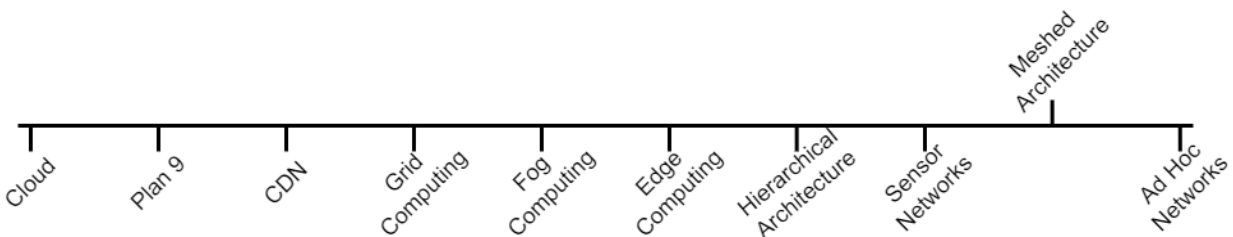


**Figure 5.1:** The architectural scale, with the addition of the hierarchical architecture

The meshed architecture fits between wireless sensor networks and wireless ad hoc networks. All three architectures are based on distributed nodes and wireless communication, but the wireless ad hoc network is much more independent. It does not

rely on a pre-existing infrastructure. Here, the nodes are free to move independently and can change communicative links frequently, but while the meshed architecture also does not have a strictly pre-existing infrastructure and is prone to frequent changes, it still requires some sort of defined frame to operate within. The wireless sensor network also requires this infrastructure to be in place and relies on the spontaneously created networks to forward their data, just like the meshed architecture. The wireless ad hoc network, on the other hand, can run on a much more scattered and chaotic structure.

Figure 5.2 displays the architectural scale with the addition of the meshed architecture.

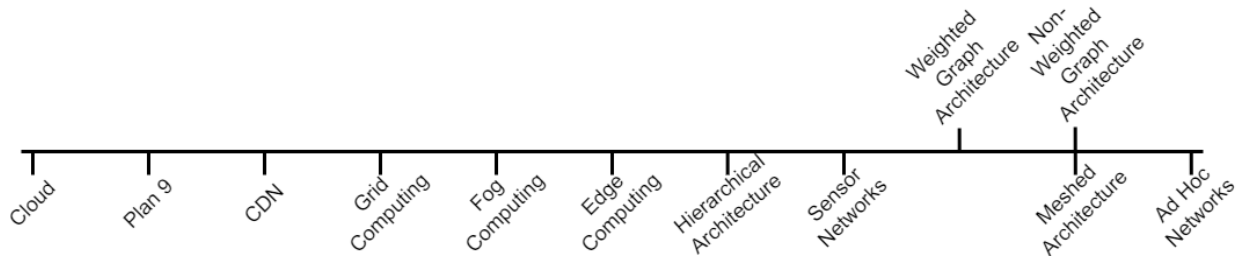


**Figure 5.2:** The architectural scale with the addition of the meshed architecture

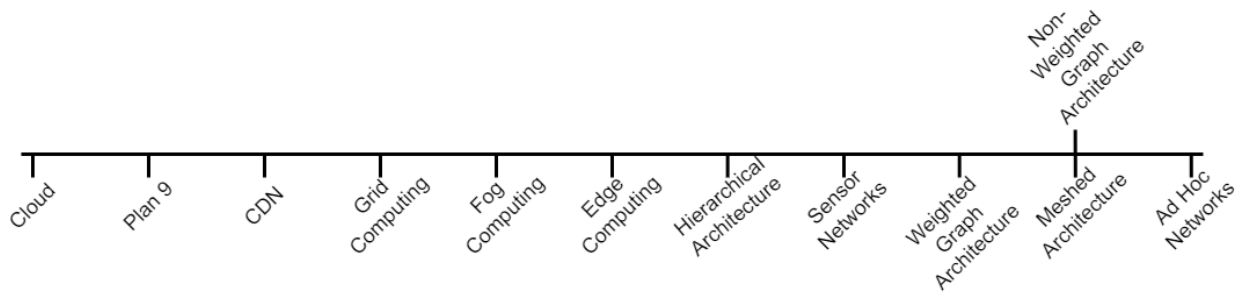
Both the non-weighted and the weighted graph architecture also fits between wireless sensor networks and wireless ad hoc networks. The graphed architectures have a lot of similarities with the meshed architecture, and therefore a lot of the reasoning behind its placement on the architectural scale is the same as it was for the meshed architecture. The graphed architectures are both built on distributed nodes and wireless communication, just like the wireless sensor networks and wireless ad hoc network, and the meshed architecture. The non-weighted graph architecture is more well-adapted to frequent changes in its infrastructure, while the weighted graph architecture relies on strong wireless connections to radio masts which results in fewer changes when a reorganization happens. This places the weighted architecture more towards the wireless sensor networks who rely on a certain infrastructure to be in place, while the non-weighted graph architecture can be placed closer to the wireless ad hoc networks, at the same place as the meshed architecture.



Figure 5.3 displays the architectural scale, with the addition of both the non-weighted and the weighted graph architecture. Figure 5.4 is the final rendition of the scale, with all of our proposed architectures uniformly placed on it.



**Figure 5.3:** The architectural scale with the addition of the graphed architectures



**Figure 5.4:** The final rendition of the architectural scale

## 5.2 Comparison to Today's Cloud

Even though distributed computing nodes are not a new concept, and today's cloud architecture already actively makes use of them in addition to their large data centers, our proposed architectures offer something new in that they are *only* based on distributed computing nodes.

Our architectures completely eliminate the need for large data centers. This will in turn save space, as large areas won't need to be allocated to hold a multitude of nodes. Some of the encompassed features that come with data centers are also eliminated because of our hardware and node location. This includes cooling, as we envision that a node that for example is placed within a waterfall and might even have internal channels that will allow the water to flow through it, will manage to cool itself.

The nodes in our architectures are also generally smaller than the ones used in a traditional cloud architecture, largely because of the absence of data centers and the hardware they use. Servers in a server rack tend to be bigger than microcomputers and mobile devices, but they might also be a lot stronger and have more processing power. However, they also use a lot more energy. Our architectures are built on the assumption that a lot of smaller, more distributed nodes will still have enough processing power to cover the needs of the cloud users, and through their distribution they would be able to seek out cheap and efficient renewable energy sources to save energy.

Perhaps what differentiates our architectures the most from the cloud architecture, is their ability to change structure. This can be seen in the meshed architecture when it morphs into a hierarchy, or in the graphed architecture which can turn into a mesh.

Despite these differences, the proposed architectures also share some similarities with the cloud architecture. These similarities mainly relate to available role types and responsibilities, as well as what is needed to be able to provide a secure, resilient cloud and wanted services.

### 5.3 Most Groundbreaking Approach(es)

Out of the proposed architectures, the one who are placed furthest away from the cloud architecture on the architectural scale is both the meshed- and non-weighted graph architecture. Being placed furthest away means that those architectures have the least in common with the cloud architecture and can therefore be determined to be the most groundbreaking approaches.

### 5.4 Cheapest Approach

Being able to determine exactly which one of the proposed architectures will result in the cheapest approach will be hard, because we haven't spent time looking into cost related to hardware, software, operation and maintenance, and that we haven't built any actual prototypes that we can gauge the price from.

Additionally, as we have mentioned multiple times throughout this thesis, in order to achieve the most robust architecture with the lowest chance of collapsing, there has to be a certain number of resources wasted in order to ensure backup of nodes, services and data. These wasted resources are contributing to increasing the price of the architectures. While a cheaper architecture is possible if we were to skip the inclusion of backup nodes, this will without a doubt result in a worse product that can not be trusted.

Taking these things into consideration, we are still able to come up with an educated guess of which architecture will be the cheapest approach. Assuming the same sized architecture, we have determined the weighted graph architecture to be the cheapest option. This is because this architecture is based on fast and efficient data transmission based on the nodes' connection to the closest radio mast. This means that a node can quickly be done with their tasks. Additionally, the architecture is based on a more fixed role assignment so it uses less resources on calculating which nodes should be assigned which roles, and more resources on actual work.

## 5.5 Most Realistic Approach

Does “realistic” refer to cheapest, easiest, fastest, most robust, or something else entirely? In this situation, we chose to look at the word to mean “which approach is most likely to be chosen by an actual company that wants to try out an alternate cloud architecture, based on the qualities we have presented so far?” Here, we have also determined that the weighted graph architecture is the most fitting approach.

We imagine that an actual company would want to take cost into consideration, which makes the weighted graph architecture a viable option. Furthermore, the weighted graph architecture says something about where nodes should ideally be located (close to radio masts, and subsequently spread out from there) for the architecture to work in the most efficient way, and this can indirectly be used as a kind of map for the company. By knowing where nodes should ideally go, they have to spend less time and money figuring that out for themselves.

Lastly, the weighted graph architecture has a more straightforward role assignment algorithm, that is largely based on physical location and distance, and not as much as the other architectures on a node's individual capabilities. This means, again, that there is time and money to be saved by choosing the weighted graph architecture, even though this architecture does not challenge the status quo in the same way as some of the other options.

## 5.6 Large-Scale Plasticity

While the proposed architectures are all placed much further to the right on the architectural scale than the cloud architecture, which takes up the extreme left point, there is something about them that feels off. Almost like they are unfinished. Yes, the architectures all accommodate different role types and services, they are built on hardware specifications and the presence of resilience and a green energy perspective, and they all offer different architectural patterns and structures. But still, none of the architectures offer anything *new* to the table that isn't already covered by an existing technology, nor do they seem like a worthy cloud architecture on their own. Would it have been better to further explore and improve something that already existed, instead of starting from scratch? Not directly basing our architectures on any existing technology allowed us to incorporate whatever features we wanted from whatever inspirational source. It is also what now allows us to put all of them together, and to form one large-scale dynamic architecture which has plasticity at its core.

We have already presented situations where the architectures are able to morph and take on qualities from the other architectures, like when the mesh becomes too big and unstable, and it transforms into a hierarchical mesh. The same situation applies to the non-weighted graph architecture; It is built on the same free multiplicity role assignment as the meshed architecture, which theoretically allows the nodes to enter into a mesh if the role assignment algorithm so sees fit. The pieces of the puzzle are already there, now we just need to put them together.

A large-scale dynamic architecture involves a more complex role assignment algorithm than the ones we have talked about up until now. The algorithm itself should be able to change along with internal and external circumstances and be able to assign roles based on different requirements at different times. Thanks to the algorithm, the large-scale architecture takes on qualities reminiscent of a living organism that is able to adapt and change within its environment. These qualities are thanks to its plasticity.

Here, our previously presented architectures become patterns to be influenced by, and not absolute truths to strive for. The entire architectural pattern might change from one thing to another in a restructure where enough nodes have been disconnected, or we might end up with simultaneous variants of the different architectures, like with the hierarchical mesh.

## 6 Discussion

In order to be able to reach a conclusion that is able to mark the problem statement as successfully achieved or not, as well as to be able to answer the three research objectives presented earlier on in the thesis, we have to take a look at what we have presented so far. Through a reversed overlook at the entire thesis, we will in this chapter draw attention to important findings that have been made throughout.

### 6.1 A New Discovery

Through the exploratory research we have conducted, we found that none of our proposed architectures offers anything new to the table on their own, nor are they different enough from existing technologies. Portraying few differences from already established cloud functionalities can be seen as a natural evolution of the cloud, and not as something that intends to challenge it. Focusing on just one architecture, instead of three, could've forced us to further develop it into something unique, with more detailed technical descriptions. However, coming up with multiple architectures showed us that there are indeed multiple options available regarding how cloud computing is done. Additionally, realizing our architectures were indeed too similar does not mean that we failed in our quest to challenge the status quo, but rather the opposite, as it helped us realize we needed something more in order to really pack a punch.

Combining the proposed architectures into one large-scale dynamic architecture created something new and represents our unexpected findings for this thesis. A living architecture that adapts and changes based on its given context; An ever-changing cloud architecture that will restructure the cloud architecture in different patterns based on internal and external circumstances. There are no rules that specify that an architecture *must* be restructured in the same pattern, this is just something we originally assumed when presenting our architectures. However, allowing the rules around restructuring to be more lenient is what consequently allows the role assignment algorithms to mix and match from the different patterns, based on the current situation of the cloud. Sort of like using a kaleidoscope; Twisting and turning until the chaos

makes sense. The cloud architecture needs some kind of structure to function, but we have found that it doesn't necessarily have to look like any predefined structure we are familiar with. This architecture is radically different from today's cloud architecture, and thus, it is an actual viable contender that can challenge the status quo.

## 6.2 Individual Qualities and Overarching Similarities

If the architectures were to be left to their own devices, determining which approach is the most realistic, cheapest and most groundbreaking is a good way to determine their individual impact. These are also some of the qualities we imagine that potential developers or stakeholders would be interested in if the architectures were to be implemented. However, determining which architecture best fit which category was harder than anticipated as we didn't have any actual data to base our choices on. As we didn't have exact numbers to determine things like uptime, computing power, latency etc., we had to stick to speculation. Therefore, if these architectures are implemented in a future prototype, we might end up seeing different results.

Comparing the architectures to today's cloud is necessary to see how far we've moved away from the original cloud architecture, but it was again difficult to gauge exactly how different they were. Again, this comes down to lack of quantifiable data. Qualities like uptime, computing power, latency are arguably more important in an architecture than things like the configuration of nodes, role assignment and level of distribution. On the other hand, the latter allowed us to find similarities with the previously existing architectures and technologies that are already different from the cloud architecture, which by proxy made ours different as well. This allowed us to place our architectures far away from the cloud architecture on the architectural scale

As a tool to visualize the general level of differences between different technologies and architecture, the architectural scale does a good job. However, while the distance between two items on the scale determine their likeness, the scale does not disclose any specifics about the technologies. If we were to create a tool like the architectural

scale again, implementing a way to differentiate between technological details more clearly might result in a different order of the items on the scale.

A cloud structure needs to be able to handle and offer many different services, both to the end users, and to other internal actors. In this thesis, we only focused on four different services (websites, game servers, streaming and caching), when we perhaps should've focused on more in order to get a deeper understanding of how the architectures would operate, and not just how they were constructed. Of the services investigated, we saw that they would all benefit from running on nodes in close proximity to a radio mast. However, not all of them are able to do that at the same time because of limited capacity, which is why the caching service is likely the best option for this as it helps with improving load times, bandwidth cost and latency

Between the architectures there were many similarities of how they handled the services, and we saw the same thing regarding how they handled unreliability; While the architectures were different on paper regarding their role assignment algorithm and architectural structure, they were also really similar to each other, and to existing technologies. These similarities are what originally prompted us to look into further evolving our solution.

## 6.3 Fundamental Assumptions

Perhaps we should've realized our architectures were too similar when we began presenting the features that were common between them; A list that continued to grow as we wrote it, and that perhaps is even bigger than we've given it credit for when we take the similarities of role assignment and handling of services and unreliability into account.

The common features are all individual, but they also somewhat impact each other. For example, energy harvesting makes it possible to place nodes in geographically distributed, remote places where they can get energy directly from sunlight or hydropower. Being smaller than traditional cloud computing processing units enables



the nodes to be placed in waterfalls, on cliff sides or on rooftops without disturbing the environment around it. They also won't need as much energy as more traditionally sized ones. Containers will help with resilience because services get back up quickly if they fail, and the resilient characteristics of the nodes will in turn benefit the fact that the nodes are distributed, as it will be hard to rely on physical manpower in order to ensure uptime. The general conclusion that can be drawn is that none of our proposed architecture can rely on just one quality, but rather a combination of many.

Through the common features of our proposed architectures, we presented areas we found to be important in order to offer a viable cloud architecture, but which we wouldn't necessarily focus on in depth. Things like availability, resilience and energy harvesting quickly became fundamental assumptions, along with details regarding how the nodes communicate and transfer data across the network. Excluding technical details was never because we ignored these functionalities, but rather because we chose to focus our attention elsewhere.

Green energy itself, which was a significant motivator and inspiration source for the thesis, also became an underlying, fundamental assumption. It continues to be an important factor throughout, but because we assumed green energy was a part of the architectures, we didn't spend too much time on going into details. To some, this might convey that green energy wasn't as important in the end, which is in no way our intention. Our focus on green energy and sustainability might've been put on the backburner compared to other features, but this is simply a result of where we put our attention. Because of the thesis's exploratory nature and lack of implementations throughout, we didn't look into how we would measure the architecture's "green-ness", nor did we have any tools to do so. Yet we continued to assume the presence of green energy and built our architectures around that.

When we assumed the underlying qualities of a reliable, green cloud, we weaved ourselves into a paradox; We wanted to have a green cloud that is more conscious about energy consumption and physical space, yet when we presented the

architectures we found having more resources on hand than necessary and “wasting” them to be one of the best ways to ensure robustness. Thankfully, because of the type of node our architectures are assumed to be built upon regarding their size, computing power and energy harvesting abilities, their green aspect still stands strong. The question becomes whether they are indeed strong enough to replace a data center structure. Our fundamental assumptions facilitated this, but as we have no numbers to refer to, it’s difficult to say for certain.

## 6.4 Did We Drift Away from The Original Plan?

Although we ended up in a different place than we had anticipated, we arguably did not drift from what we had planned. Instead, we instead went beyond it, thanks to the idea of combining the architectures. The plan to explore alternative architectures that could challenge the status quo has guided us along the way and has prompted us to be critical and to seek out more unique solutions.

To be able to challenge the status quo we had planned to present different alternatives. Exactly how many became a result of the exploratory research, as new doors continued to open as we continued to work. Through planning the approach and operationalization of the problem statement, we were able to prematurely limit our workload to help us avoid getting overwhelmed by new ideas and tasks that would undoubtedly appear along the way. Even if the merger of the architectures into the large-scale, plasticity-based architecture was such an idea, it came as a logical next step when we saw that our solution was lacking. Of course, had we spent more time on the original architectures, dived into their technical abilities, and deemed them as good enough on their own, we might not have come up with the idea of merging them together at all. We can therefore argue that the final architecture came as a surprising, yet welcoming discovery.

The concept of large data centers, both in size and in terms of energy consumption has been central the whole way through. In the background, though presenting the history of the cloud, we highlighted that the concept has grown over the years without any real

reconsideration as to how it's done. On the contrary, the introduction of the concept "the green pledge" proves that a focus on green and sustainable things has been on the forward march for years now, and if you're not with it, you're against it. The current cloud architecture is but one area where what is being done currently does not reflect good on the environmental threat landscape. Throughout the thesis, and manifested by the problem statement and potential impact, it becomes clear that our purpose has been to challenge the status quo all along. A greener cloud takes up less space and energy and reclaiming those resources for things like sustainable food production, affordable housing, and wildlife conservation, could positively impact a betterment of our environment and future.

## 7 Conclusion and Future Work

The goal of this thesis was to challenge the status quo by investigating the effects of a highly distributed, minimalistic cloud platform driven by a green energy focus. To address this, we modeled and presented our own distributed architectures: the *hierarchical, meshed, non-weighted, and weighted graph architectures*. The architectures are based on resilient, energy harvesting nodes that communicate through wireless mobile networks, can run a variety of services corresponding to the way clouds are used today, and have a plan for handling unreliability stemming from variations in energy availability and network access. A scale of distribution was developed, comprising today's most common distributed architectures. Our novel architectures were placed towards the distributed end, on the opposite side of the cloud.

Realizing our architectures had many similarities prompted us to put them together to form the large-scale dynamic architecture. This is also where we introduced the term plasticity, and found that the rules regarding restructuring didn't have to be as strict as we originally planned for. Adaptability is what lead this architecture to be the most drastic challenge to the status quo, as it's different from any existing solutions.

Concepts like the green energy perspective, which would have contributed to more detailed descriptions, became fundamental assumptions, and their effect on the highly distributed and minimalistic cloud platform were not able to be measured. The project has made important contributions towards a blueprint for a different kind of cloud, which is as elusive and ethereal as real clouds themselves, yet provide a stable whole which can still be understood as a single system when viewed from afar.

Implement such a system requires more research. As of today, the architecture represents an options for an entirely new cloud architecture, as changing the existing cloud into our proposed solution would be expensive and potentially disruptive. Future work can be divided into software and hardware focused implementations. Through prototyping and modeling on an existing cloud, it's possible to simulate how role

assignment and restructuring would work in reality, and implementing nodes in geographically distributed locations enables us to look closer at unreliability caused by external factors and communication through 4g/5g. Both are viable options in order to continuously develop the large-scale dynamic architecture, and further challenge the status quo.

# References

- MarketsandMarkets. (2021, October). *Cloud Computing Market by Service, Deployment Model, Organization Size, Vertical And Region - Global Forecast to 2026*. ReportLinker. Retrieved 26.01.2022 from [https://www.reportlinker.com/p05749258/Cloud-Computing-Market-by-Service-Deployment-Model-Organization-Size-Workload-Vertical-And-Region-Global-Forecast-to.html?utm\\_source=GNW](https://www.reportlinker.com/p05749258/Cloud-Computing-Market-by-Service-Deployment-Model-Organization-Size-Workload-Vertical-And-Region-Global-Forecast-to.html?utm_source=GNW)
- Amazon. (n.d.). *Global Infrastructure*. AWS. Retrieved 27.01.2022 from <https://aws.amazon.com/about-aws/global-infrastructure/>
- Swinhoe, D. (2021, July 5). Amazon plans more than 1 million sq ft of data centers in Manassas, Virginia. *Datacenter Dynamics*. Retrieved 27.01.2022 from <https://www.datacenterdynamics.com/en/news/amazon-plans-more-than-1-million-sq-ft-of-data-centers-in-manassas-virginia/>
- Pearce, F. (2018, April 3). Energy Hogs: Can World's Huge Data Centers Be Made More Efficient? *Yale Environment 360*. Retrieved 08.02.2022 from <https://e360.yale.edu/features/energy-hogs-can-huge-data-centers-be-made-more-efficient>
- IBM Cloud Education. (2020a, March 3). *Public Cloud*. IBM. Retrieved 13.02.2022 from <https://www.ibm.com/cloud/learn/public-cloud>
- Microsoft Azure. (n.d.). *What are public, private, and hybrid clouds?* Azure. Retrieved 13.02.2022 from <https://azure.microsoft.com/en-us/overview/what-are-private-public-hybrid-clouds/#overview>
- IBM Cloud Education. (2020b, April 10). *Private Cloud*. IBM. Retrieved 13.02.2022 from <https://www.ibm.com/cloud/learn/introduction-to-private-cloud>
- Schmidt, E. (Speaker), & Sullivan, D. (Host). (2006, August 9). *Conversation with Eric Schmidt hosted by Danny Sullivan*. [Panel transcript]. Search Engine Strategies Conference. Retrieved 21.02.2022 from <https://www.google.com/press/podium/ses2006.html>
- Greenberger, M. (Ed.) (1962). *Management and the computer of the future*, p. 220-248. [Cambridge, Mass.]: Published jointly by M.I.T. Press and Wiley, New

York. Retrieved 22.02.2022 from

<https://archive.org/details/managementcomput00gree>

- Bell, C. G., & Gold, M. M. (1972). An Introduction to the Structure of Time-Shared Computers. In J. T. Tou (Ed.), *Advances in Information Systems Science: Volume 4* (pp. 161–272). Springer US. Retrieved 22.02.2022 from [https://doi.org/10.1007/978-1-4615-9053-8\\_4](https://doi.org/10.1007/978-1-4615-9053-8_4)
- Gartner Group. (n.d.). *Gartner Hype Cycle*. Gartner. Retrieved 22.02.2022 from <https://www.gartner.com/en/research/methodologies/gartner-hype-cycle>
- Intel. (n.d.). *Over 50 Years of Moore's Law*. Intel. Retrieved 22.02.2022 from <https://www.intel.com/content/www/us/en/silicon-innovations/moores-law-technology.html>
- Winter, M. (2009). Data Center Consolidation: A Step towards Infrastructure Clouds. In M. G. Jaatun, G. Zhao, & C. Rong (Eds.), *Cloud Computing* (pp. 190–199). Springer. Retrieved 22.02.2022 from [https://doi.org/10.1007/978-3-642-10665-1\\_17](https://doi.org/10.1007/978-3-642-10665-1_17)
- Arutyunov, V. V. (2012). Cloud Computing: Its History of Development, Modern State, and Future Considerations. *Sci. Tech. Inf. Proc.* 39, 173–178 (2012). Retrieved 22.02.2022 from <https://doi.org/10.3103/S0147688212030082>
- IBM Cloud Education. (2021, August 9). *Multicloud*. IBM. Retrieved 13.02.2022 from <https://www.ibm.com/cloud/learn/multicloud>
- Google Cloud. (n.d.). *What is BigQuery Omni?* Google. Retrieved 22.02.2022 from <https://cloud.google.com/bigquery-omni/docs/introduction>
- Oxford Languages. (2022a). Node. In Google Dictionary Box. Retrieved 06.05.2022 from [https://www.google.com/search?q=node+definition&rlz=1C1GCEA\\_enNO836NO836&oq=node+definition&aqs=chrome.0.69i59j0i512l5j69i60l2.3030j1j9&sourceid=chrome&ie=UTF-8](https://www.google.com/search?q=node+definition&rlz=1C1GCEA_enNO836NO836&oq=node+definition&aqs=chrome.0.69i59j0i512l5j69i60l2.3030j1j9&sourceid=chrome&ie=UTF-8)
- IBM. (2021, March 3). *Architecture*. IBM Cloud Private. Retrieved 25.04.2022 from <https://www.ibm.com/docs/en/cloud-private/3.2.0?topic=started-architecture>
- VMWare. (n.d. a). *What is Cloud Architecture?* VMWare. Retrieved 25.04.2022 from <https://www.vmware.com/topics/glossary/content/cloud-architecture.html>

- Kshemkalyani, A. D., & Singhal, M. (2011). *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press. Retrieved 27.04.2022 from <https://eclass.uoa.gr/modules/document/file.php/D245/2015/DistrComp.pdf>
- Pike, R., Presotto, D., Dorward, S., Flandrena, B., Thompson, K., Trickey, H., Winterbottom, P. (1995). Plan 9 from Bell Labs. *Computing Systems*, 8(3), pp. 221-254. Retrieved 27.04.2022 from <https://pdos.csail.mit.edu/archive/6.824-2012/papers/plan9.pdf>
- Hancock, B. (2003). Reinventing Unix: An introduction to the Plan 9 operating system. *Library Hi Tech*, 21(4), 471–476. Retrieved 27.04.2022 from <https://doi.org/10.1108/07378830310509772>
- Cloudflare. (n.d.a). *What is a CDN? | How do CDNs work?* Cloudflare. Retrieved 27.04.2022 from <https://www.cloudflare.com/en-gb/learning/cdn/what-is-a-cdn/>
- Jacob, B., Brown, M., Fukui, K., Trivedi, N. (2005). *Introduction to Grid Computing*. IBM Redbooks. Retrieved 27.04.2022 from <https://www.redbooks.ibm.com/redbooks/pdfs/sg246778.pdf>
- Yousefpour, A., Fung, C., Nguyen, T., Kadiyala, K., Jalali, F., Niakanlahiji, A., Kong, J., & Jue, J. P. (2019). All one needs to know about fog computing and related edge computing paradigms: A complete survey. *Journal of Systems Architecture*, 98, 289–330. Retrieved 27.04.2022 from <https://doi.org/10.1016/j.sysarc.2019.02.009>
- Sandhiya, C., & Bhuvaneswari, E. (2018). An Overview on Wireless Sensor Networks (WSN) and Mobile ADHOC Networks (MANET). *International Journal of Engineering Science Invention (IJESI)*, ISSN(e) : 2319-6734, pp. 10-14. Retrieved 27.04.2022 from <http://www.ijesi.org/papers/NCIOT-2018/Volume-5/3.%2010-14.pdf>
- Huang, C., Zhou, S., Xu, J., Niu, Z., Zhang, R., & Cui, S. (2018). *Energy Harvesting Ad Hoc Networks*. (pp. 167–202). Retrieved 27.04.2022 from <https://doi.org/10.1002/9781119295952.ch5>
- Basagni, S., Naderi, M. Y., Petrioli, C., & Spenza, D. (2013). Wireless Sensor Networks with Energy Harvesting. In S. Basagni, M. Conti, S. Giordano, & I. Stojmenovic (Eds.), *Mobile Ad Hoc Networking* (pp. 701–736). John Wiley &



Sons, Inc. Retrieved 27.04.2022 from

<https://doi.org/10.1002/9781118511305.ch20>

- History. (2020, March 3). *Water and Air Pollution*. History. Retrieved 22.04.2022 from <https://www.history.com/topics/natural-disasters-and-environment/water-and-air-pollution>
- Ekholm, N. (1901). On the Variations of the Climate of the Geological and Historical Past and Their Causes. *Quarterly Journal of the Royal Meteorological Society*, 27(117), 1–62. Retrieved 22.04.2022 from <https://doi.org/10.1002/qj.49702711702>
- Fleming, J. R. (1999). Joseph Fourier, the ‘greenhouse effect’, and the quest for a universal theory of terrestrial temperatures. *Endeavour*, 23(2), 72–75. Retrieved 22.04.2022 from [https://doi.org/10.1016/S0160-9327\(99\)01210-7](https://doi.org/10.1016/S0160-9327(99)01210-7)
- Rodhe, H., Charlson, R., & Crawford, E. (1997). Svante Arrhenius and the Greenhouse Effect. *Ambio*, 26(1), 2–5. Retrieved 22.04.2022 from <https://www.jstor.org/stable/4314542>
- President’s Science Advisory Committee. (1965, November 2). Restoring the Quality of Our Environment. *The White House*. Retrieved 22.04.2022 from <https://www.climatefiles.com/climate-change-evidence/presidents-report-atmospher-carbon-dioxide/>
- History. (2021, April 21). *How the First Earth Day Was Born From 1960s Counterculture*. History. Retrieved 22.04.2022 from <https://www.history.com/topics/natural-disasters-and-environment/water-and-air-pollution>
- Earth Day. (n.d.). *The History of Earth Day*. Earth Day. Retrieved 22.04.2022 from <https://www.earthday.org/history/>
- Ellen MacArthur Foundation. (2018, July 9). *The Fifth Industrial Revolution is Happening - Is it Time to Reshape Our Future? | Summit 2018* [Video]. Youtube. <https://www.youtube.com/watch?v=dhNd3tVR1hI>
- Brundtland, G.H. (1987) Our Common Future: Report of the World Commission on Environment and Development. Geneva, UN-Document A/42/427. Retrieved 25.04.2022 from <http://www.un-documents.net/ocf-ov.htm>

- IMDb. (n.d.). *An Inconvenient Truth*. IMDb. Retrieved 25.04.2022 from <https://www.imdb.com/title/tt0497116/>
- Matthews, L. (n.d.). *10 Celebrities Leading the Fight Against Climate Change*. LeafScore. Retrieved 25.04.2022 from <https://www.leafscore.com/blog/eco-friendly-celebrities-battling-climate-change/>
- Crouch, D. (2018, September 1). *The Swedish 15-year-old who's cutting class to fight the climate crisis*. The Guardian. Retrieved 25.04.2022 from <https://www.theguardian.com/science/2018/sep/01/swedish-15-year-old-cutting-class-to-fight-the-climate-crisis>
- Nguyen, T. (2020, February 3). *Fast fashion, explained*. Vox. Retrieved 25.04.2022 from <https://www.vox.com/the-goods/2020/2/3/21080364/fast-fashion-h-and-m-zara>
- Khusainova, G. (2021, January 28). *The Secondhand Market Is Growing Rapidly, Can Challengers Like Vinokilo Thrive And Scale?* Forbes. Retrieved 25.04.2022 from <https://www.forbes.com/sites/gulnazkhusainova/2021/01/28/the-secondhand-market-is-growing-rapidly-can-challengers-like-vinokilo-thrive-and-scale/?sh=f32de2ccb642>
- Cisco. (n.d.). *What is a Network Controller?* Cisco. Retrieved 16.03.2022 from <https://www.cisco.com/c/en/us/solutions/enterprise-networks/what-is-a-network-controller.html>
- NGINX. (n.d.). *What Is Load Balancing?* NGINX. Retrieved 16.03.2022 from <https://www.nginx.com/resources/glossary/load-balancing/>
- Raspberry Pi. (n.d.). *Raspberry Pi 4 Tech Specs*. Raspberry Pi. Retrieved 29.03.2022 from <https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>
- Athow, D., Hanson, M. (2022, February 1). *Best microSD cards in 2022: flash memory for drones, cameras and more*. Techradar. Retrieved 29.03.2022 from <https://www.techradar.com/news/best-sd-and-microsd-memory-cards>
- GSMArena. (n.d.) *Apple iPhone 13 Pro Max*. GSMArena. Retrieved 29.03.2022 from [https://www.gsmarena.com/apple\\_iphone\\_13\\_pro\\_max-11089.php](https://www.gsmarena.com/apple_iphone_13_pro_max-11089.php)

- Computer History Museum. (n.d.). *What Was The First PC?* Computer History. Retrieved 29.03.2022 from <https://www.computerhistory.org/revolution/personal-computers/17/297>
- Department of Computer Science and Statistics. (n.d.). *History of Computers*. Department of Computer Science and Statistics. Retrieved 29.03.2022 from <https://homepage.cs.uri.edu/faculty/wolfe/book/Readings/Reading03.htm>
- Docker. (n.d.). *Use containers to Build, Share and Run your applications*. Docker. Retrieved 07.04.2022 from <https://www.docker.com/resources/what-container/>
- Oxford Languages. (2022b). Resilience. In Google Dictionary Box. Retrieved 20.04.2022 from [https://www.google.com/search?q=resilience+definition&rlz=1C1GCEA\\_enNO836NO836&oq=resilience+de&aqs=chrome.1.69i57j0i512j0i20i263i512j0i512j0i20i263i512j0i512i5.1371j1j1&sourceid=chrome&ie=UTF-8](https://www.google.com/search?q=resilience+definition&rlz=1C1GCEA_enNO836NO836&oq=resilience+de&aqs=chrome.1.69i57j0i512j0i20i263i512j0i512j0i20i263i512j0i512i5.1371j1j1&sourceid=chrome&ie=UTF-8)
- Docker Docks. (n.d.). *Start containers automatically*. Docker Docks. Retrieved 09.05.2022 from <https://docs.docker.com/config/containers/start-containers-automatically/>
- Cloudflare. (n.d.b). *What is latency? | How to fix latency*. Cloudflare. Retrieved 10.03.2022 from <https://www.cloudflare.com/en-gb/learning/performance/glossary/what-is-latency/>
- Verizon. (2022, April 26). *What's the difference between 4G LTE and 5G? Understanding network speeds*. Verizon. Retrieved 08.05.2022 from <https://www.verizon.com/articles/network-speed-comparison>
- Australian Government. (2017, October). *5G—Enabling the future economy*. Department of Communications and the Arts. Retrieved 18.04.2022 from <https://www.infrastructure.gov.au/sites/default/files/5g-enabling-the-future-economy.pdf>
- iSelect. (n.d.). *5G Vs. 4G*. iSelect Australia. Retrieved 08.05.2022 from <https://www.iselect.com.au/internet/5g-australia/5g-vs-4g/>
- VMWare. (n.d. b). *What is Software-Defined Networking (SDN)?* VMWare. Retrieved 16.03.2022 from

<https://www.vmware.com/topics/glossary/content/software-defined-networking.html>

- Weka. (2020, July 15). *Clustered File System*. WekaIO Inc. Retrieved 05.05.2022 from <https://www.weka.io/learn/clustered-file-system/>
- Elliott, E. (2019, May 11). How Popular is JavaScript in 2019? *Medium*. Retrieved 05.05.2022 from <https://medium.com/javascript-scene/how-popular-is-javascript-in-2019-823712f7c4b1>
- JavaScript. (2021, December 12). *An Introduction to JavaScript*. JavaScript. Retrieved 05.05.2022 from <https://javascript.info/intro>
- Kubernetes. (2022, January 10). *Pods*. Kubernetes. Retrieved 05.05.2022 from <https://kubernetes.io/docs/concepts/workloads/pods/>
- F5. (n.d.). *Load Balancer*. F5 Glossary. Retrieved 05.05.2022 from <https://www.f5.com/services/resources/glossary/load-balancer>
- Cloudflare. (n.d. c). *What is streaming? | How video streaming works*. Clourflare. Retrieved 08.05.2022 from <https://www.cloudflare.com/en-gb/learning/video/what-is-streaming/>
- Imperva. (n.d.). *CDN Caching*. Imperva, The Essential CDN Guide, Chapter 3. Retrieved 08.05.2022 from <https://www.imperva.com/learn/performance/cdn-caching/>
- MDN Web Docks. (2022, April 30). *HTTP caching*. MDN Web Docks. Retrieved 08.05.2022 from <https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching>