Nebula

Comparing two waves of cloud compute

Marius Nilsen Kluften

Thesis submitted for the degree of

Master in Informatics: Programming and System Architecture

60 credits

Institute of Informatics

Faculty of mathematics and natural sciences

University of Oslo

2024

Nebula

Comparing two waves of cloud compute

Marius Nilsen Kluften

Todo list

Nebula

https://duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

The ever increasing demand for cloud services has resulted in the expansion of energy-intensive data centers, the ICT industry accounts for about 1 % of global electricity use, highlighting a need for sustainable options in cloud computing architectures.

This thesis investigates WebAssembly, a technology originally intended for running in the browser, as a potential contender in the space of technologies to consider in cloud native applications. Leveraging the inherent efficiency, portability and lower startup times of WebAssembly modules, this thesis presents an approach that aligns with green energy principles, while maintaining performance and scalability, essential for cloud services.

Preliminary findings suggest that programs compiled to WebAssembly modules have reduced startup and runtimes, which hopefully leads to less energy consumption and offering a viable pathway towards a more sustainable cloud.

Acknowledgments

The idea for the topic for this thesis appeared in an episode of the podcast ”Rustacean station”. Matt Butcher, the CEO of Fermyon, told the story of his journey through the different waves of cloud computing, and why Fermyon decided to bet on WebAssembly for the next big wave of cloud compute.

The capabilities of WebAssembly running on the server, with the aid of the WebAssembly System Interface (WASI) project, caught my interest and started the snowball that ended up as the avalanche that is this thesis.

I'd like to thank Matt Butcher and the people over at Fermyon for inadvertently inspiring my topic.

Furthermore I'd like to thank my two supervisors Joachim Tilsted Kristensen and Michael Kirkedal Thomsen, whom I somehow managed to convinced to help guide me through such a cutting edge topic. Their guidance and insight have been invaluable the past semesters.

Finally I would like to thank Syrus Akbary, founder of Wasmer, whom I met at WasmIO 2024 who showed me how to reduce my startup times by a further 100 times.

Contents

List of Figures

List of Tables

Part I

Overview

Chapter 1

Introduction

> *If WASM+WASI existed in 2008, we wouldn't have needed to created Docker. That's how important it is. Webassembly on the server is the future of computing. A standardized system interface was the missing link. Let's hope WASI is up to the task!*
>
> —Solomon Hykes, *Founder of Docker*

In the digital age (Freitag et al., 2021), cloud computing has emerged as a foundational technology in the technological landscape, driving innovation and increased efficiency across various sectors. Its growth over the past decade has not only transformed how consumers store, process, and access data, but it has also raised environmental concerns as more and more data centers are built around the globe to accommodate the traffic, consuming vast amounts of power. The Information and Communication Technology (ICT) industry, with cloud computing at its core, accounts for an estimated 2.1% to 3.9% of global greenhouse gas emissions. Data centers, the backbone of cloud computing infrastructures, are responsible for about 200 TWh/yr, or about 1% of the global electricity consumption, a figure projected to escalate, potentially reaching 15% to 30% of electricity consumption in some countries by 2030 (Freitag et al., 2021).

The sustainability of cloud computing is thus under scrutiny, and while some vendors strive to achieve a net-zero carbon footprint for their cloud computing services, many data centers still rely on electricity generated by fossil fuels, a leading contributor to climate change (Mytton, 2020). This reality emphasizes an urgent need to explore alternative technologies that promise enhanced energy efficiency while meeting customers demands. In this vein, serverless computing has emerged as a compelling paradigm, offering scalability and flexibility by enabling functions to execute in response to requests, rather than having a server running all the time. However, the inherent startup latency associated with containerized serverless functions pose a challenge, particularly for on-demand applications and to mitigate this, vendors often opt for keeping the underlying servers *warm* to keep the startup latency as low as possible for serving functions. Reducing the startup time for serving a function significantly should mitigate the need for keeping servers warm and therefore reduce the standby power consumption of serverless architectures.

This thesis proposes exploring WebAssembly with WebAssembly System Interface (WASI) as an innovative choice for deploying functions to the cloud, through developing a prototype Functions-as-a-Service platform named Nebula. This platform will run functions compiled to WebAssembly, originally designed for high-performance tasks in web browsers, which coupled with WASI, allows us to give WebAssembly programs access to the underlying system. This holds potential for a more efficient way to package and deploy functions, potentially reducing the startup latency and the overhead associated with traditional serverless platforms. WebAssembly and WASI offers a pathway where the demands of today is met, while reducing the carbon footprint for cloud applications.

1.1 Motivation

The environmental footprint of cloud computing, particularly the energy demands of data centers, is a pressing issue. As the digital landscape continues to evolve, the quest for sustainable solutions has never been more critical. This thesis is motivated by the need to reconcile the growing demand for cloud services with the pressing need for environmental sustainability. Through the lens of WebAssembly and WASI, this thesis aims to investigate innovative deployment methods that promise to reduce energy consumption without sacrificing performance, thereby contributing to the development of a more sustainable cloud computing ecosystem.

1.2 Problem Statement

The goal of the thesis is to:

1. Develop a prototype cloud computing platform for the Functions-as-a-Service (FaaS) paradigm
2. Use this platform for conducting experiments that either prove or disprove the claim that WebAssembly is the more energy efficient choice

By achieving these goals this thesis seeks to shed light on the feasibility and implications of adopting WebAssembly and WASI for a greener cloud.

1.3 Outline

The thesis has five chapters; this introduction, a chapter that goes through the background for how cloud computing got to this point, a chapter dedicated to the process of building Nebula, a chapter for discussing the results from the experiments, and ending with a chapter suggesting future works.

Chapter 2

Three waves of cloud compute

9 out of 10 cloud providers
hate this one simple trick.

Joachim, my supervisor

The evolution of cloud computing represents a transformative adventure, driven by the pursuit for efficiency, scalability and reliability, yet it also poses challenges, notably it's environmental impact. This chapter steps through this adventure by introducing the concept of the "Three waves of cloud computing", coined by the WebAssembly community (Butcher & Dodds, 2024). Where the two first waves of cloud compute represent the shit from Virtual Machines to Containerization, the third wave encompasses utilizing WebAssembly and the WebAssembly System Interface (WASI) to build the next era of cloud compute with the potential to significantly reduce the carbon footprint.

2.1 Ashore: Before the waves

Before delving into the waves themselves, it's essential to understand the landscape that preceded cloud computing. Prior to the cloud era, companies were required to building and mantaining their digital services in-house. This required companies to invest heavily into both expensive hardware and expensive engineers to buy, upkeep and oversee their own physical servers and network hardware. (See figure 2.1 for an example)

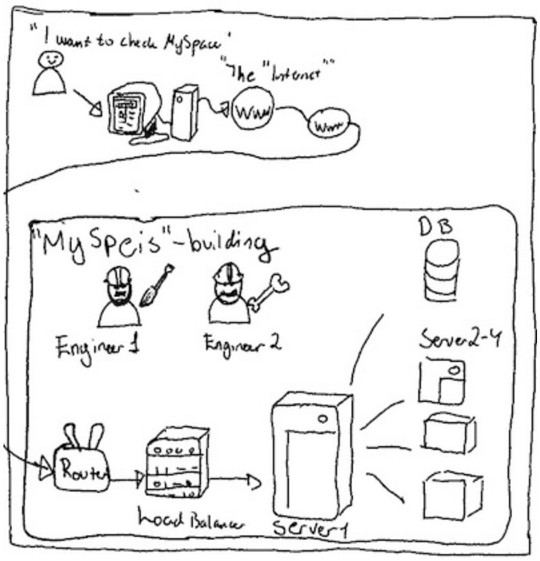

Figure 2.1: Example of a company that host their own infrastructure.

This sort of setup mandates a significant upfront costs involved in setting up and maintaining such an infrastructure, which puts a considerable financial strain on organizations, and kept smaller companies that were unable to invest in this, at an disadvantage.

As a response to this, some companies found a market for taking on the responsibility of managing infrastructure, and offer Infrastructure-as-a-Service (IaaS) services to an evolving ecosystem of companies with a digital landscape. On these managed infrastructures companies could deploy their services on top of Virtual machines that allowed more flexibility, and lowered the bar to new companies.

2.2 The First Wave: Virtual Machines

The start of cloud computing can be traced back to the emergence of virtualization, more specifically virtual machines, a response to the costly and complex nature of managing traditional, on-premise data centers. During the mid-2000s, Amazon launched its subsidiary, Amazon Web Services (AWS), who in turn launched Amazon S3 in March 2006, followed by Elastic Compute Cloud (EC2) in August the same year (Barr, 2006). With these services, AWS positioned itself as a pioneer in this space, marking a major turning point in application development and deployment, and popularized cloud computing. EC2, as an Infrastructure-as-a-Service (IaaS) platform, empowered developers to run virtual machines remotely.

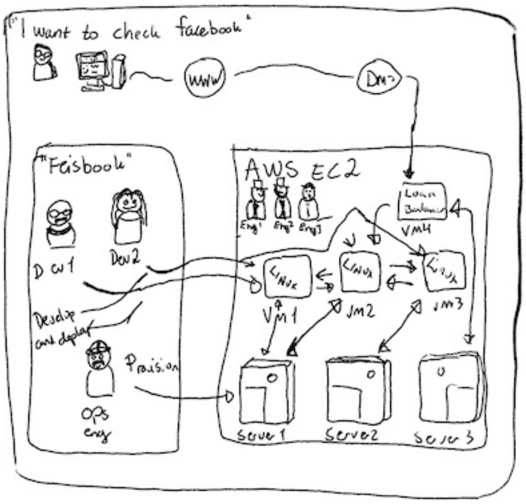

Figure 2.2: Example of "Feisbook" building their services on EC2

While similar services existed before 2006, with Amazon's existing large customer base helped them gain significant traction, and ushered in a the first era, or wave, of *cloud computing*.

2.3 The Second Wave: Containerization

As we entered the 2010s, the focus shifted from Virtual Machines to containers, largely due to the limitations of VMs in efficiency, resource utilization, and application deployment speed. Containers, being a lightweight alternative to VMs, designed to overcome these hurdles (Bao et al., 2016).

In contrast to VMs, which require installation of resource-intensive operating systems and minutes to start up, containers along with their required OS components, could start up in seconds. Typically managed by orchestration tools like Kubernetes [1], containers enabled applications to package alongside their required OS components, facilitating scalability in response to varying service loads. Consequently, an increasing number of companies have since established platform teams to build orchestrated developers platforms, thereby simplifying application development in Kubernetes clusters.

[1] https://kubernetes.io

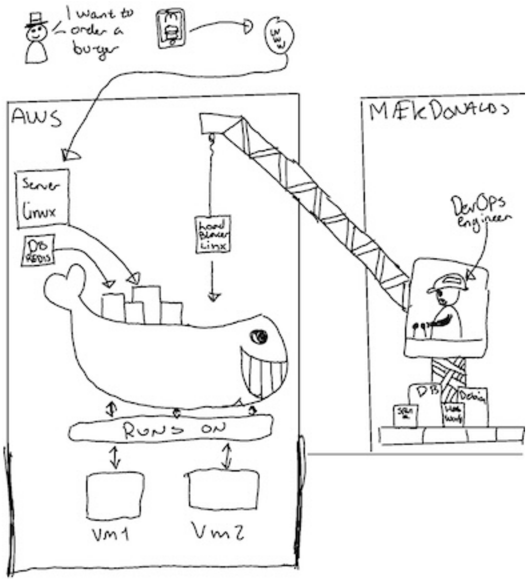

Figure 2.3: DevOps engineer deploying services as containers on AWS

Containers are not a perfect solution however, and while they simplify the means of developing and deploying applications, docker images can easily reach Gigabytes in image size (Durieux, 2024), and can take a long time to start up. These solutions are more efficient than manually installing an operating system on a machine, but they still have leave a large footprint, is there a more efficient way to package and deploy our programs? Maybe WebAssembly and WebAssembly System Interface, as mentioned in epigraph of this thesis can pose a promising alternative?

2.4 The Third Wave: WebAssembly

WebAssembly has had a surge of popularity the past three to four years when developers discovered that what it was designed for - to truly run safely inside the browser - translated well into a cloud native environment as well. Containers has had a positive shift on the cloud native landscape as a whole, but while a lot better than the previous iterations of cloud applications, some shortcomings remain. The image sizes can get quite large, starting up a docker image can be a costly affair, and the abstraction layer between the underlying computer and the code running in the container increases the resources required to run programs.

WebAssembly is a compilation target with many languages adopting support, and by itself, it is sandboxed to run in a WebAssembly VM without access to the outside world, meaning that it can't access the underlying system. This means that a "vanilla" WebAssembly module can't write to the file system, update a Redis cache or transmit a POST request to another service.

To make this possible, the WebAssembly System Interface project was created. This project allows developers to write code that compiles to WebAssembly that can access the underlying system. This is the key project that turned many developers onto the path of exploring WebAssembly as a potential contender for building cloud applications. With WebAssembly, developers can write programs in a programming language that supports it as a compilation target, and build tiny modules that can run on a WebAssembly runtime. These WebAssembly runtimes can run on pretty much any architecture with ease, the resulting binary size are quite small, and the performance is near-native. These perks combined with the potential for reduced overhead, smaller image sizes, and faster startup times make WebAssembly and WASI a promising candidate for the third wave of cloud compute with a lower impact on the environment.

In summary, the three waves of cloud computing - virtual machines, containers, and now WebAssembly with WASI - represent the industry's pursuit of more efficient, scalable and reliable solutions for building cloud applications. While each wave has attempted to tackle pressing challenges of its time, it's exciting to see how WebAssembly and WASI can be leveraged in this third wave and see if it's promise of more efficient applications can lead to reducing the environmental impact of ICT.

Chapter 3

Background

Data has gravity, and that gravity pulls hard

David Flanagan

3.1 Cloud Computing Overview

Cloud computing, commonly referred to as "*the cloud*", refers to the delivery of computing resources served over the internet, as opposed to traditional on-premise hardware setups. The National Institute of Standards and Technology (NIST) defines Cloud computing like so:

NIST definition of Cloud Computing

> Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.
>
> (Mell & Grance, 2011)

Cloud computing traces its root back to the 1960s, with the Compatible Time-Sharing System (CTSS) project at MIT, which demonstrated the potential for multiple users accessing and sharing computing resources simultaneously (Crisman, 1963). While CTSS was a localized system, it paved the way for the concept of shared computing resources, a fundamental principle of cloud computing.

Over the following decades, advancements in networking, virtualization and the ubiquity of the internet led to the development of today's sophisticated cloud services. The term "cloud computing" was first coined in the year 1996 by Compaq, (Favaloro & O'Sullivan, 1996), but it wasn't until Amazon launched its subsidiary Amazon Web Services in the 2006 that the adoption became wide spread.

Alongside the founding of AWS a suite of cloud services launched, including Elastic Compute Cloud (EC2) for virtual servers (Barr, 2006), Simple Storage Service (S3) for object storage. By providing these services over the internet on a pay-as-you-go basis, AWS drastically lowered the bar for accessing computing resources, making it easier and more cost-effective for businesses and developers to build and deploy applications without the need for considerable upfront investment in hardware and infrastructure.

With the success of AWS, other major technology companies saw their fit to enter the cloud computing market. In 2008, Google launched the Google App Engine (McDonald, 2008), a platform for building and hosting web applications in Google's data centers. Microsoft followed with the launch of Azure in 2010, its cloud computing platform that offers a range of services comparable to AWS.

The rapid growth of cloud computing also fueled the rise of DevOps practices and containerization technologies like Docker, which facilitate the development, deployment and management of applications on the cloud . Orchestration tools like Docker Swarm and Kubernetes further simplify the process of managing and scaling containerized applications across cloud enviroments (Bernstein, 2014).

Today, cloud computing has become an essential part of modern IT infrastructure, where major cloud provides, like AWS, Microsoft and Google, continue to innovate and expand their offerings. Cloud computing has also enable new paradigms like serverless computing and edge computing, allowing for even more efficient and distributed computing models. (Baldini et al., 2017)

3.2 Energy efficiency and Sustainability in Cloud Computing

One of the downsides to contrast the benefits of cloud computing, is the ever increasing demand for energy consumption required for running these servers in giant data centers. With the increased demand for energy consumption, comes an increased impact on the environment. As mentioned in chapter 1, the ICT industry accounts for an estimated 2.1% to 3.9% of global greenhouse emissions. The latest figure from 2021 calculates that data centers across the globe consumes about 200 TWh/year, accounting for 1% of the global electricity consumption. This estimate is projected to escalate to reach even 15% to 30% of electricity consumption in some countries by 2030 (Freitag et al., 2021).

In Norway, for example, Google is constructing a data center in Skien, expected to be fully operational by 2026. As of April 2024, they have been granted a capacity of 240 Megawatts, but they have applied for a capacity of 860 Megawatts in total (Rivrud, 2024). At full capacity, Google's data center is aiming to consume 7.5 TWh each year, and according to Google's most recent sustainability report, they consumed a total of 22.29 TWh globally in 2022 ("Google 2023 Environmental Report", 2023). In other words, in 2026 the data center in Skien alone is projected to consume ~33% of the energy Google consumed globally in 2022.

The other side of this coin, is that Google is commited for a net-zero carbon footprint by 2030, and the data center in Skien is built to reflect this. However, this commitment is not universally adopted, and many data centers still rely on electricity generated by fossil fuels, a leading contributor to climate change (Mytton, 2020).

Several factors make up the energy consumption required to service a data center. One of these factors is cooling down the servers while running, and a study from 2017 discovered that cooling accounted for about 38% of total energy consumption in data centers, ranging from 21% to 61% depending the effectiveness of the facility's heating, ventilation, and air conditioning (HVAC) system (Ni & Bai, 2017).

One innovative example of attempting to mitigate the environmental impact of cooling data centers can be found by data center providers like DeepGreen [1], who submerge their servers into dielectric fluid which gets warmed up by the excess heat of the computers. This heat is then transferred to a host's hot water system via a heat exchange and used for heating up swimming pools in London. Another broad strategy cloud providers opt for is the implementation of power management techniques, such as dynamic voltage and frequency scaling (DVFS), which adjusts the power consumption of servers based on workload demands (Beloglazov et al., 2012).

Virtualization and resource pooling, two key components to cloud computing, also contribute to energy efficiency. By consolidating virtual machines onto shared physical servers, cloud providers are able to improve resource utilization and reduce the energy consumption of their data centers. (Beloglazov et al., 2012)

[1] https://deepgreen.energy/faqs/

3.3 Virtualization and Container

What do I want to convey here? This chapter is going to start introducing virtualization as a concept, and move on to virtual machines, and try to explain the technical details of the first wave of cloud compute.

TODO

Cloud would be difficult to reach the scale it has without the creation of virtualization. Virtualization is a process that allows for more efficient usage of physical hardware, by using software to create an abstraction layer over computer hardware.

AWS definition of Virtualization

> Virtualization is technology that you can use to create virtual representations of servers, storage, networks, and other physical machines. Virtual software mimics the functions of physical hardware to run multiple virtual machines simultaneously on a single physical machine. Businesses use virtualization to use their hardware resources efficiently and get greater returns from their investment. It also powers cloud computing services that help organizations manage infrastructure more efficiently.
>
> (AWS, n.d.)

As mentioned in section 2.2, the emergence of virtualization and consequently the creation of Virtual Machines, laid the foundation that allowed the cloud ecosystem to evolve.

3.3.1 Virtual Machines

Write about virtual machines, the technical details of it, and how it relates to my project. I'm not using a VM directly, but it might be interesting to know how it lays the foundation for moving onto containers/docker for deploying applications. Keywords: Hypervisor, Host hardware, Host Os, internal OS, application.

TODO

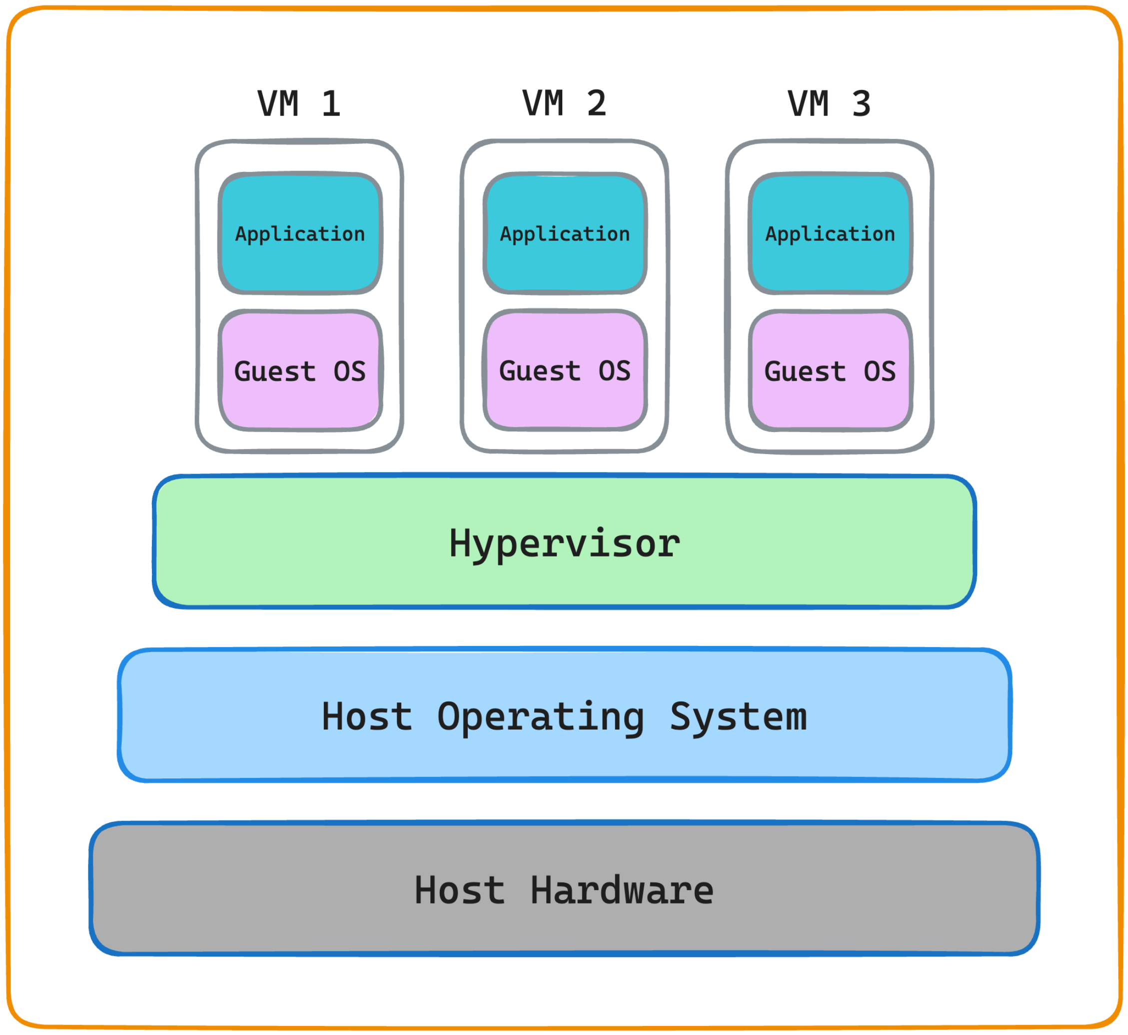

Figure 3.1: Virtual Machines running on top of Hypervisor on a computer

3.3.2 Containers and Docker

Go into containers, reference to the figure and lay out how containers has a smaller footprint and better performance for running applications compared to hoisting entire virtual machines. Keywords: Host hardware / VM, Host OS, Docker engine, bins/libs and application.

TODO

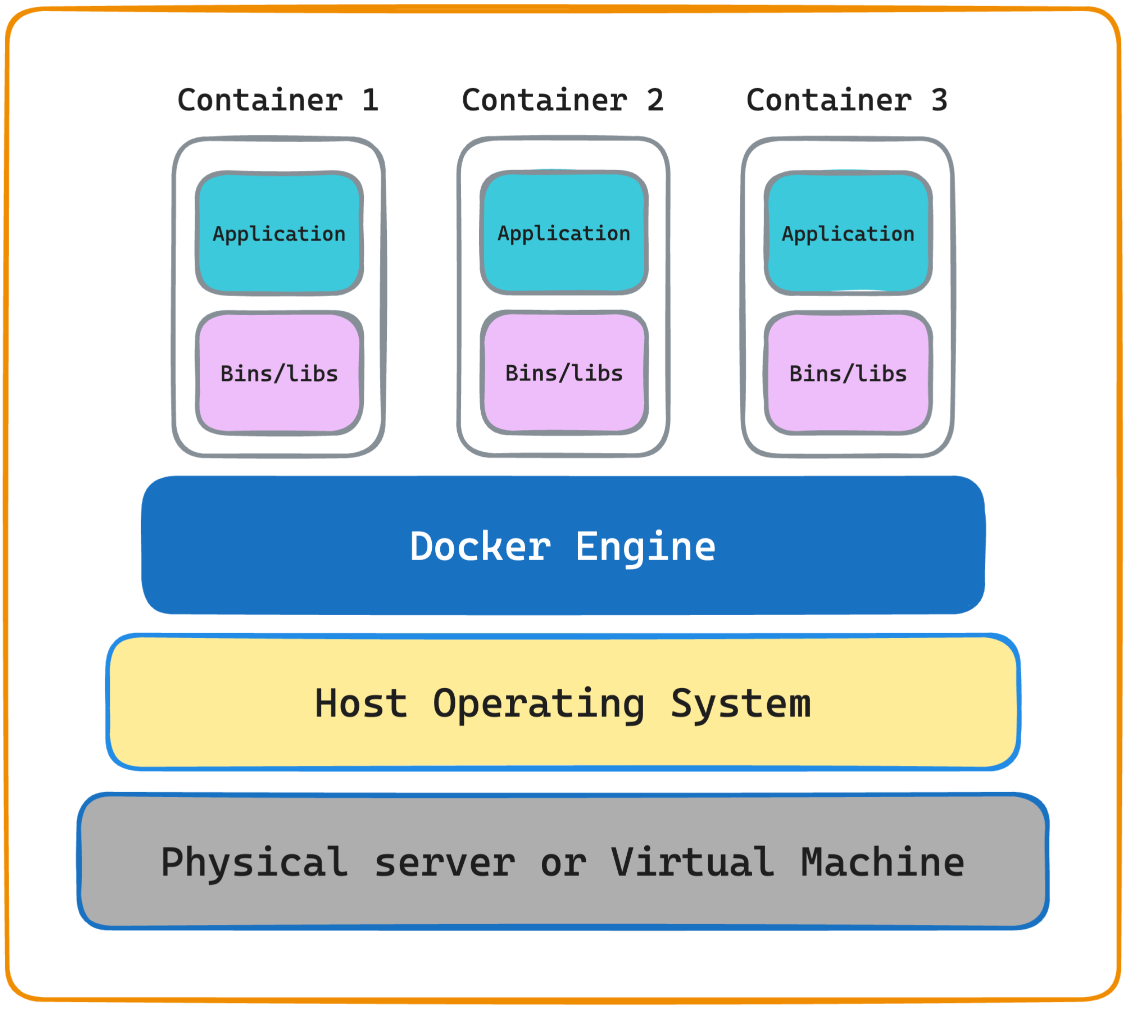

Figure 3.2: Containers running on the Docker Engine

3.3.2.1 Container orchestration

Get into Docker swarm and Kubernetes, and how applications are meant to be built, deployed and finally orchestrated on top of X amount of machines. Keywords: K8s, Dockerswarm, networking, energy consumtion, 60

TODO

3.4 Serverless Computing

While building developer platforms atop orchestration tools like Kubernetes has gained popularity due to improved efficiency over managing physical infrastructure, this approach still entails significant operational overhead. In certain situations, developers could greatly benefit from the ability to develop and deploy smaller services without incurring the operational costs associated with expanding the company's existing platform to accommodate them. This need paved the way for the emergence of serverless computing.

Despite its somewhat misleading name, serverless computing doesn't imply the absence of servers. Instead, it means that the underlying infrastructure is abstraced away, enabling developers to focus solely on writing code and dpeloying functions without worrying about provisioning or managing servers. This paradigm shift alleviates the operational burden and allows for more efficient resource utilization, as resources are dynamically allocated based on demand, and the provider handles scaling and maintenance.

The serverless model promotes a more granular approach to application development, where individual functions can be deployed independently, promoting modularity and scalability. Developers can focus on building and iterating on specific features without the complexities of infrastructure management.

By embracing serverless computing, organizations can achieve faster time-to-market, reduced operational costs, and improved developer productivity. However, it's important to note that while serverless architectures eliminate the need for server management, they introduce their own set of considerations, such as potential vendor lock-in, cold start latencies, and the need for a well-designed event-driven architecture.

From serverless computing, we can derive a subset - Functions-as-a-Service - a popular format for third-party vendors to offer serverless computiation to their customers.

3.4.1 Functions-as-a-Service (FaaS)

Functions-as-a-Service (FaaS) focuses on the execution of individual functions in response to events or triggers. Major cloud providers, including Amazon Web Services (AWS Lambda), Google (Cloud Functions), and Microsoft (Azure Functions), offer FaaS platforms, allowing developers to write and deploy functions without managing the underlying infrastructure.

FaaS platforms typically employ container technology to execute functions, with each function running in an isolated container environment. This approach provides security, scalability, and efficient resource utilization. However, the startup overhead associated with containers can introduce latency, particularly for on-demand applications with strict performance requirements.

This overhead and the lack of a *true* platform-agnostic way to run containers culminated in the quote cited in the epigraph of chapter 1, where the creator of Docker saw WebAssembly and WebAssembly System Interface as a promising way to package and run application code across all platforms.

3.5 WebAssembly and WASI

WebAssembly, commonly referred to as Wasm, is a modern binary instruction format that has risen to prominence as a versatile technology across a diverse amount of computing environments, originating in the web browser. This section introduces the project that WebAssembly evolved from - *asm.js* - and illustrate how WebAssembly lets developers write programs in a high-level language and run them across a multitude of platforms.

3.5.1 asm.js

Mozilla released the first version of asm.js in 2013 and designed it to be a subset of JavaScript, designed to allow web applications written in other languages than JavaScript, such as C or C++, to run in the browser. The intention of asm.js is to allow for web applications to run at performance closer to native code than applications written in standard JavaScript can achieve. A simplified flow for how source code written in C/C++ is compiled to bytecode that can be executed in the browser can be found in figure 3.3 below.

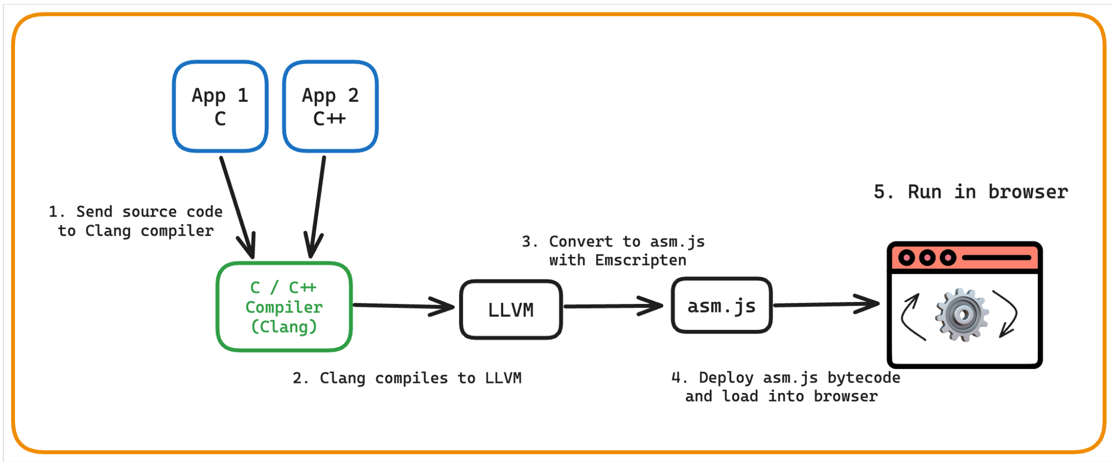

Figure 3.3: Source code in C/C++ compiled to asm.js and run in browser

While asm.js was a great leap forward, being a subset of JavaScript limited the scope of what it could become, leading to the development of a more efficient and portable format. The team at Mozilla built upon the lessons learned from asm.js and went on to develop WebAssembly and launch the first public version in 2017.

3.5.2 WebAssembly

From the ashes of asm.js we get WebAssembly, which is a low-level code format designed to serve as a compilation target for high-level programming languages. It is a binary format that gets executed by a stack-based virtual machine, comparable to how Java bytecode runs on the Java Virtual Machine (JVM).

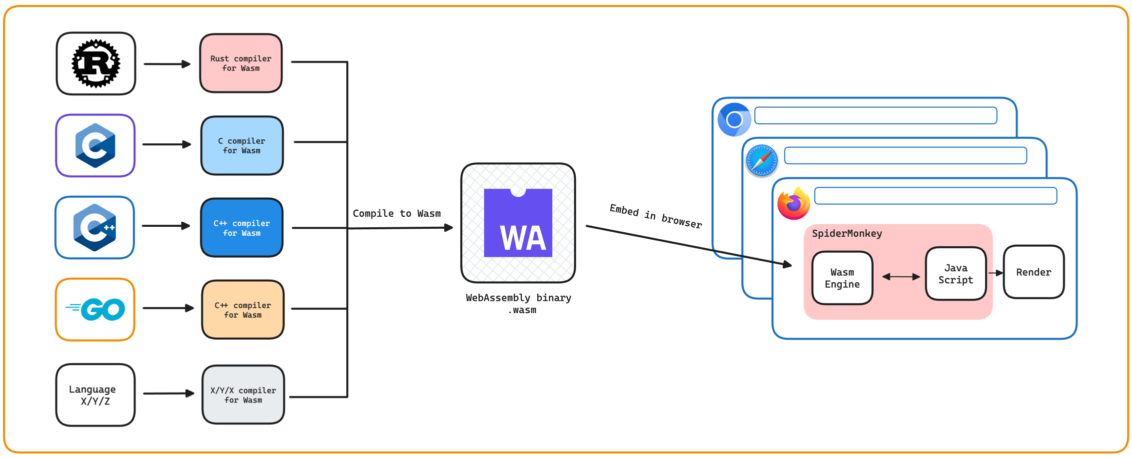

Figure 3.4: Source code compiled to WebAssembly and embedded in browser

WebAssembly, originally designed for running demanding computations in web browsers, present a promising technology that could help reduce the energy consumption of cloud services. It offers an interesting option for packaging functions with its compact binary format and fast execution time. This has the potiential to significantly reduce startup latency and resource overhead associated with traditional serverless platforms. This increased efficiency could lead to a direct decrease in energy consumption for cloud services, which in turn could motivate the industry to adopt alternative technology that enable a more sustainable cloud.

Rewrite the rest of this section, it's a bit of a mess right now

The WebAssembly team defines WebAssembly as such:

WebAssembly definition

> WebAssembly (abbreviated Wasm) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications.
>
> webassembly.org

In other words, WebAssembly is a low-level code format designed to serve as a compilation target for high-level programming languages. It's a binary format that gets executed by a stack-based virtual machine, similar to how Java bytecode runs on the Java Virtual Machine (JVM). It was originally designed for running in a browser environment, and every major browser has implemented a way for running it.

WebAssembly have promising properties that makes it interesting to investigate if it can find a home outside the browser environment it was designed for:

Efficiency and Speed: WebAssembly was designed to be fast, enabling near-native performance. Its binary format is compact and designed for quick decoding, contributing to quicker startup times, important aspects of cloud native applications.

Safety and Security: WebAssembly is designed to run safely in a secure sandbox. Each WebAssembly module executes within a confined environment without direct access to the host system's resources. This isolation of processes is inherent in WebAssembly's design, promoting secure practices.

Portability: WebAssembly's platform-agnostic design makes it highly portable. It can run across a variety of different system architectures. For cloud native applications, this means WebAssembly modules, once compiled, can run anywhere - from the edge to the server - on any environment.

Language Support: A large amount of programming languages can already target WebAssembly. This means developers are not restricted to a particular language when developing applications intended to be deployed as WebAssembly modules.

This provides greater flexibility to leverage the most suitable languages for particular tasks.

In contrast, traditional methods such as deployment with containers or VMs can be resource-intensive, slower to boot up, less secure due to a larger surface attack area, and less efficient. Given these, WebAssembly, with its efficiency, security, and portability, can potentially offer an attractive alternative deployment method for building and running cloud native applications, like the "Academemes" service we will explore in this essay.

3.5.3 WebAssembly System Interface

WebAssembly (WASM) and WebAssembly System Interface (WASI) present promising choices to traditional ways of deploying and hosting Function as a Service (FaaS) platforms, offering several notable advantages, in terms of startup times and energy efficiency.

Reduced Startup Times: One of the greatest strengths of Wasm is its compact binary format designed for quick decoding and efficient execution. It offers near-native performance, which results in significantly reduced startup times compared to container-based or VM-based solutions. In a FaaS context, where functions need to spin up rapidly in response to events, this attribute is particularly advantageous. This not only contributes to the overall performance but also improves the user experience, as the latency associated with function initialization is minimized.

Improved Energy Efficiency: Wasm's efficiency extends to energy use as well. Thanks to its optimized execution, Wasm can accomplish the same tasks as traditional cloud applications but with less computational effort. The CPU doesn't need to work as hard, which results in less energy consumed. With data centers being responsible for a significant portion of global energy consumption and carbon emissions, adopting Wasm could lead to substantial energy savings and environmental benefits.

Scalability: Wasm's small footprint and fast startup times make it an excellent fit for highly scalable cloud applications. Its efficiency means it can handle many more requests within the same hardware resources, hence reducing the need for additional servers and thus reducing the energy footprint further.

Portability and Flexibility: WASI extends the portability of Wasm outside the browser environment, making it possible to run Wasm modules securely on any WASI-compatible runtime. This means that FaaS platforms can run these modules on any hardware, operating system, or cloud provider that supports WASI. This portability ensures flexibility and mitigates the risk of vendor lock-in.

While runtime efficiency is an important aspect and typically a strength of Wasm, it might not be the primary focus of this thesis. That being said, it is worth mentioning that the efficient execution of Wasm modules does contribute to the overall operational efficiency and energy savings of Wasm-based FaaS platforms.

In summary, introducing WASM+WASI as a component for deploying and hosting FaaS platforms can offer significant benefits. Focusing on energy efficiency and reduced startup times, this approach could pave the way for more sustainable, efficient, and responsive cloud services. In the context of our "Academemes" service, this could lead to a scalable, performant, and environmentally friendly platform.

3.6 Rust programming language

3.6.1 Introduction to Rust

3.6.2 Rust and WebAssembly

3.6.3 Building Nebula with Rust

3.7 Energy monitoring

Write about MQTT and Zwave/Oh my Gude

Part II

Project

Chapter 4

Approach

To investigate the problem statements posed in section 1.2, roughly summarized to exploring if WebAssembly and WebAssembly System Interface can lead to a more efficient and energysaving way to build our cloud services, an exploratory approach will be used. Different benchmarking experiments will be run against a prototype developed for this thesis, where different functions can be invoked with different inputs and reveal startup and runtimes of invoking functions compiled to WebAssembly modules and compare these with functions packaged as Docker images.

Chapter 5

Analysis

Chapter 6

Design

Chapter 7

Implementation

This is the chapter on implementing Nebula.

7.1 Tech stack

Rust/Docker/Etc.

Part III

Results

Chapter 8

Evaulation

Chapter 9

Discussion

Chapter 10

Conclusion

References

Chapter 11

Appendices

Bibliography

AWS. (n.d.). *What is Virtualization? - Cloud Computing Virtualization Explained - AWS*. Amazon Web Services, Inc. Retrieved April 3, 2024, from https://aws.amazon.com/what-is/virtualization/

Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Slominski, A., & Suter, P. (2017, June 10). *Serverless Computing: Current Trends and Open Problems* (1). arXiv: 1706.03178 [cs]. https://doi.org/10.48550/arXiv.1706.03178

Bao, W., Hong, C., Sudheer Chunduri, Chunduri, S., Krishnamoorthy, S., Sriram Krishnamoorthy, Sriram Krishnamoorthy, Pouchet, L.-N., Rastello, F., & Sadayappan, P. (2016). Static and Dynamic Frequency Scaling on Multicore CPUs. *ACM Transactions on Architecture and Code Optimization*, *13*(4), 51. https://doi.org/10.1145/3011017
MAG ID: 2567319156 S2ID: 7ee529c7a72f7f228ba1e60011d5e1d5078730d6.

Barr, J. (2006, August 25). *Amazon EC2 Beta | AWS News Blog*. Retrieved April 1, 2024, from https://aws.amazon.com/blogs/aws/amazon_ec2_beta/

Beloglazov, A., Abawajy, J., & Buyya, R. (2012). Energy-aware resource allocation heuristics for efficient management of data centers for Cloud computing. *Future Generation Computer Systems*, *28*(5), 755–768. https://doi.org/10.1016/j.future.2011.04.017

Bernstein, D. (2014). Containers and Cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing*, *1*(3), 81–84. https://doi.org/10.1109/MCC.2014.51

Butcher, M., & Dodds, E. (2024, January 10). *How WebAssembly is Enabling the Third Wave of Cloud Compute with Matt Butcher of Fermyon Technologies* (No. 172). Retrieved April 1, 2024, from https://datastackshow.com/podcast/how-webassembly-is-enabling-the-third-wave-of-cloud-compute-with-matt-butcher-of-fermyon-technologies/

Crisman, P. A. (1963). *Computer Time-Sharing System*. MIT. Retrieved March 29, 2024, from https://people.csail.mit.edu/saltzer/Multics/CTSS-Documents/CTSS_ProgrammersGuide_1963.pdf

Durieux, T. (2024, March 12). *Empirical Study of the Docker Smells Impact on the Image Size* [Comment: Accepted at ICSE'24. arXiv admin note: text overlap with arXiv:2302.01707]. arXiv: 2312.13888 [cs]. https://doi.org/10.1145/3597503.3639143

Favaloro, G., & O'Sullivan, S. (1996, November 14). *Internet Solutions Division Strategy for Cloud Computing*. Retrieved April 1, 2024, from https://s3.amazonaws.com/files.technologyreview.com/p/pub/legacy/compaq_cst_1996_0.pdf

Freitag, C., Berners-Lee, M., Widdicks, K., Knowles, B., Blair, G. S., & Friday, A. (2021). The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations. *Patterns*, *2*(9), 100340. https://doi.org/10.1016/j.patter.2021.100340

Google 2023 Environmental Report. (2023).

McDonald, P. (2008, April 7). *Introducing Google App Engine + our new blog*. Google App Engine Blog. Retrieved April 8, 2024, from https://googleappengine.blogspot.com/2008/04/introducing-google-app-engine-our-new.html

Mell, P., & Grance, T. (2011, September 1). The NIST Definition of Cloud Computing.

Mytton, D. (2020). Hiding greenhouse gas emissions in the cloud. *Nat. Clim. Chang.*, *10*(8), 701–701. https://doi.org/10.1038/s41558-020-0837-6

Ni, J., & Bai, X. (2017). A review of air conditioning energy performance in data centers. *Renewable and Sustainable Energy Reviews*, *67*, 625–640. https://doi.org/10.1016/j.rser.2016.09.050

Rivrud, K. (2024, February 7). *Investeringen av et Google-senter i Skien er gigantisk – det blir også strømforbruket*. NRK. Retrieved April 12, 2024, from https://www.nrk.no/vestfoldogtelemark/investeringen-av-et-google-senter-i-skien-er-gigantisk-_-det-blir-ogsa-stromforbruket-1.16753588