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

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 T. Kristensen and Marcus 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. [@tapioniemiGreenBigData2018]

Contents

List of Figures

List of Tables

Part I

Overview

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*

Chapter 1

Introduction

In the digital age, 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 also raised environmental concerns. 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 underscores 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.

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

This thesis focuses on two primary objectives:

1. Assessing the potential for optimizing cloud service deployments using WebAssembly, with an eye towards the trade-offs in energy consumption associated with different deployment strategies.
2. Evaluate the potential energy savings and potential performance gains by employing WebAssembly for function execution in cloud environments.

By addressing these objectives, this thesis seeks to shed light on the feasibility and implications of adopting WebAssembly and WASI for a more energy-efficient cloud computing world.

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 **?**. 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)

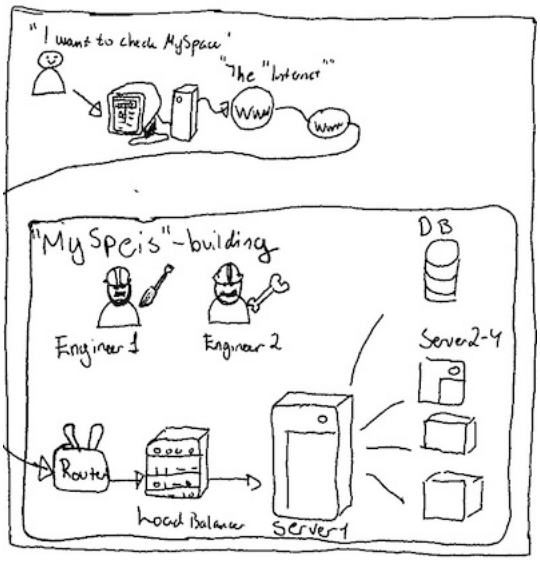

Figure 2.1: Example of a company that host their own infrastructure.

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 **?**. 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.

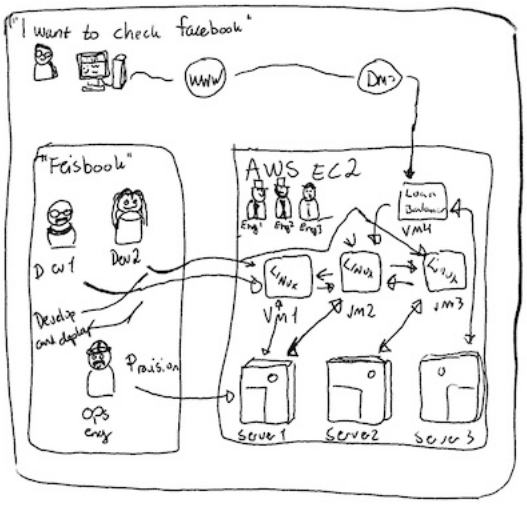

Figure 2.2: Example of ”Feisbook” building their services on EC2

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

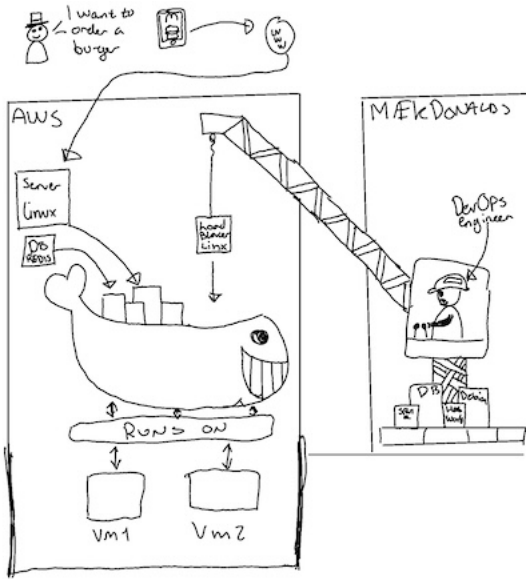

Figure 2.3: DevOps engineer deploying services as containers on AWS

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 enable this, the WebAssembly System Interface project was birthed. 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: An quick summary

Cloud computing, more commonly known as "*the cloud*", refers to the delivery of computing resources served over the internet, instead of running on locally owned hardware (on-premise). 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.

For many companies, the cloud has proved to be a super power, where companies can focus on deploying their own applications and services to their users without worrying about the underlying infrastructure. Some of these benefits include:

Table 3.1: Cloud Computing Benefits

Benefit	Description
Reduced total cost of ownership	Cloud computing enables companies to reduce upfront infrastructure costs and optimize expenses as they scale. Important for startups who can't afford to manage their own infrastructure. **?**.
Scalability	Offers the ability to efficiently scale operations and resources in response to changing demand **?**.
Portability	Cloud services provide

However, in the field of ICT, one truth prevails: the inevitability of trade-offs. Every decision in software engineering involves weighing different factors against each other, such as performance against simplicity, or between speed of delivery and robustness **?**. This rings especially true in the world of cloud computing where, depending on the scale of your company, the way you set up your services on the cloud can have large implications. Some of these trade-offs include:

Table 3.2: Cloud Computing Challenges

Benefit	Description
Cost management	Despite its potential for cost savings, managing and optimizing cloud expenses remains a challenge for many organizations. **?**
Energy consumption	The environmental impact and energy usage of data centers pose significant sustainability challenges. Data centers alone account for approximately 1

3.2 Virtualization

3.3 Virtual Machines

3.4 Containers and Docker

3.5 Container orchestration

3.6 Serverless and Function-as-a-Service (FaaS)

Building your own developer platform on top of Kubernetes, much like building your own infrastructure, also entails a significant cost. Often, developers wish to launch specialized smaller services, without having to grapple with complicated orchestration. This led to the emergence of the Serverless model. Despite its somewhat misleading name, serverless doesn't imply the absence of a server. Instead, it means that the responsibility of server management has shifted from the developer to a third party provider.

From the advancements of serverless, we get its subset, Functions-as-a-Service, or FaaS. Companies already in the cloud game decided to develop their own FaaS platforms to attract developers interested in just writing their functions and running them, and not worry about anything underneath.

3.7 Major vendors in Serverless

The concept of "the cloud" isn't owned by any single organization, but rather, through the collective effort of industry players including Amazon, Microsoft, Google, Alibaba and DigitalOcean, among others. This essay delves into some challenges faced by the biggest three vendors: Amazon, Google and Microsoft.

Amazon Web Services (AWS) provides AWS Lambdas, a technology that hinges on their proprietary Firecracker - a streamlined virtualization technology for executing functions. Interestingly, for this thesis, is that Amazon's Prime Video streaming service transitioned recently from a serverless architecture to a monolithic system to meet specific service demands. One might question whether this reflects the suitability of serverless systems for cloud computing, or for specific use cases like theirs (Kolny, M. 2023. Accessed 29.05.23). Some discussions suggest that their need to process videos frame by frame led to astronomical costs on their sibling company's FaaS, Amazon Lambda.

Google provides Google Cloud Functions, which allow developers to write and execute functions in languages such as Node.js, Python, Go and execute them in response to events. Google's approach to function execution centeres around container technology (Wayner, P. 2018. Accessed 29.05.23).

Microsoft's Azure Functions is a Faas platform that enables developers to create and execute functions written in languages like C#, JavaScript, Python. Similar to Google, they also harness the power of containers to execute these functions.

3.8 WebAssembly

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.

WebAssembly (Wasm) is a binary instruction format designed as a stack-based virtual machine. It aims to be a portable target for the compilation of high-level languages like Rust, C++, Go and many others, enabling deployment on the web for client and server applications. Originally designed and developed to complement JavaScript in the browser, it now expands its scope to server-side applications, thanks to projects like WebAssembly System Interface (WASI), which provides a standarized interface for WebAssembly modules to interface with a system.

WebAssembly's design provides advantages over traditional deployments methods in the context of cloud native applications:

Efficiency and Speed: Wasm was designed to be fast, enabling near-native performance. Its binary format is compact and designed for quick decoding, contributing to quicker startup times, an important aspect for server-side applications. The performance gains could lead to less CPU usage, thereby improving energy efficiency.

Safety and Security: WebAssembly is designed to be safe and sandboxed. 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, irrespective of the environment.

Language Support: A large amount of programming langauages 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.

3.8.0.1 WASM+WASI: Towards Energy-efficient FaaS Platforms

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.

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

Bao, Wenlei, Changwan Hong, Sudheer Chunduri, Sudheer Chunduri, Sriram Krishnamoorthy, Sriram Krishnamoorthy, Sriram Krishnamoorthy, Louis-Noël Pouchet, Fabrice Rastello, and P. Sadayappan. 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.

Durieux, Thomas. 2024. "Empirical Study of the Docker Smells Impact on the Image Size." https://doi.org/10.1145/3597503.3639143.

Freitag, Charlotte, Mike Berners-Lee, Kelly Widdicks, Bran Knowles, Gordon S. Blair, and Adrian Friday. 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.

Mytton, David. 2020. "Hiding Greenhouse Gas Emissions in the Cloud." *Nat. Clim. Chang.* 10 (8): 701–1. https://doi.org/10.1038/s41558-020-0837-6.

Chapter 11

Appendices