# 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

# 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 project 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.

# Contents

# List of Figures

# List of Tables

## 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*

# 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, is estimated to contribute between 2.1% to 3.9% of global greenhouse gas emissions. Data centers, which are crucial to cloud computing infrastructures, account for about 200 TWh/yr, or about 1% of the global electricity consumption. (Freitag et al., 2021). This consumption is projected to increase dramatically increase, potentially accounting for 15% to 30% of electricity consumption in some countries by 2030.

In Norway, for example, Google is in the process of constructing a data center in Skien, expected to be fully operational by 2026. This facility alone is estimated to require 7.5TWh/yr, amounting to 25% of Google's total power consumption world wide.(TODO: SRC) Google has commited to powering this data center entirely with renewable energy, aiming for a net-zero carbon footprint. 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, D. 2021)

It is important to note is that these measurements and estimates comes with a certain level of uncertainty, yet they offer a glimpse into the current and future challenges of cloud computing's energy consumption. As demand for cloud services continues to rise, there is a pressing need to explore alternative technologies that can enhance energy efficiency without compromising on the performance, availability and scalability of that the users expect.

One prevalent method for building cloud applications today involves using serverless computing platforms, such as Amazon's *AWS Lambdas*, or Google's *Cloud Functions*. These platforms let developers deploy code into containers executed upon request. Despite the benefits, such as the ability to run software across different architectures, this approach introduces a startup latency for the containers. This latency is negligible for long-running services, but significant for on-demand functions.

This thesis proposes exploring WebAssembly and the WebAssembly System Interface (WASI) as innovative choices for deploying functions to public cloud services. WebAssembly, originally developed for efficient execution in web browsers, combined with WASI, enables WebAssembly to run on servers, potentially offering a more efficient way to package and deploy functions.

## Motivation

The emergence of cloud computing has reshaped the landscape of digital services, offering unparalleled scalability, flexibility, and efficiency. However, this transformation comes with significant environmental implications. With the increase in energy consumption in data centers, essential to cloud computing, there needs to be a critical evaluation of the sustainability of the industry.

Amidst growing concerns over the carbon footprint associated with cloud services, there's a need for new and innovative solutions that can balance the demands of the users and the impact on the planet. This thesis is motivated by the opportunity to contribute to this balance by exploring the potential of WebAssembly combined with the WebAssembly System Interface (WASI) for deploying functions in the cloud more efficiently.

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.

## Methodology

For this thesis, we will build a Functions-as-a-Service platform prototype, named Nebula for brevity, that we deploy to a typical Debian web server. To compare functions compiled and packaged into WebAssembly modules, we will write standalone functions in Rust that we both A. compile to WebAssembly+WASI and

B. build as ordinary Rust binaries and package into a lightweight Docker image. These WebAssembly modules and Docker images will be deployed to the web server, ready to be called by our test scripts.

On this server we will support running functions both as WebAssembly modules and Docker Images, using a respective runtime for each. For WebAssembly Nebula will use the Wasmtime project as runtime, while Docker relies on the Docker CLI tool. The act of calling these functions will be exposed as API endpoints, which allows a test script to call functions in succession in order to simulate user load.

Furthermore, a simple application will be written that hits this endpoint X amount of times with Y amount of different input values. Nebula will benchmark each individual function request based on the *Instant* crate in Rust, which in turn will be stored in a JSON file with metric information. This list of function results and metrics will lay the foundation for the findings the experiments will reveal, which we will go into detail later.

*Hopefully*, there will also be time to set up Nebula to run on a Raspberry PI that I can measure power consumption of during load of the web server, but this is paragraph won't make it into the final paper, it's more of a TODO to me, to not forget (how could I) to attempt to measure power consumption as well.

## Problem Statement

1. Investigate potential optimization by employing an alternative way of deploying functions on a Functions-as-a-Service platform.
2. Investigate potential energy savings from employing more efficient compute on cloud.

## Outline

This thesis has three parts: the introduction and background, the project, and discussions about the findings and future work. Each part is divided into chapters:

- Chapter 1 - Introduction, where the project is introduced, the motivation lying behind it, the problem statement provided and the methodology used for the project work.
- Chapter 2 - Background, where the history of cloud computing has led to the topic that is explored in this thesis. The concept of *three waves* of cloud computing is introduced, and other vendors in the market are explored.

- Chapter 3 - Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

# Background

## Cloud Computing: An Overview

Cloud computing, more commonly known as *the cloud*, refers to the delivery of computing services, such as storage, processing power, network, and software, served over the internet, instead of running on locally owned hardware (on-premise). For companies, this has proved to be a super power, where businesses can focus on deploying their own applications and services to their users without worrying about the underlying infrastructure.

Some benefits include:

- Reduced total cost of ownership: Cloud computing has enabled companies to take their computing needs to the next level. Startups who can't afford neither the cost or time required to build their own infrastructure, and larger companies that want to iterate faster and decrease their lead time from idea to production (Thomas, Dave. 2009).
- Scalability is one of the most significant benefits of cloud computing. As organizations expand, so do their customer needs and the complexity of their application infrastructure. Each additional feature brings with it additional costs, highlighting the importance of efficient resource management to optimize hardware investments in a cloud computing environment (Thomas, Dave. 2008).

Some challenges include:

- Cost: Managing the cost of cloud computing is an ongoing challenge, with Gartner predicting that through 2024, 60% of infrastructure and operations (I&O) leaders will encounter cloud costs that are higher than budgeted for (Rimol, M. 2021).

- Energy usage: The challenge of energy usage in cloud computing is a significant concern, with data centers alone accounting for approximately 1% of the world's electricity consumption (Freitag et al. 2021). This staggering statistic highlights the need to explore strategies and measures to mitigate energy usage in data centers. By reducing energy consumption, not only can costs be reduced, but also the carbon footprint associated with running the cloud can be lessened. Decreasing energy usage in data centers is expected to yield cost savings and contribute to the overall sustainability goals by reducing the cloud's environmental impact.

## Three waves of cloud compute

### The first wave: Virtual machines

In the era preceding cloud computing, companies bought, set up and managed their own infrastructure. This necessitated having in-house infrastructure engineers to maintain on-premise data centers or servers, leading to significant cost. Sensing the potential in offering managed infrastructure, Amazon launched its subsidiary, Amazon Web Services, during the mid-2000s.

The launch of AWS's Amazon S3 cloud service in March 2006, followed by Elastic Compute Cloud (EC2) in August the same year (Barr, J. 2006), marked a major turning point in application development and deployment, and popularized cloud computing. EC2, as an Infrastructure-as-a-Service platform, empowered developers to run virtual machines remotely. While similar services existed before 2006, Amazon's large customer base helped them gain significant traction, effectively bringing cloud computing to the mainstream.

### 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 (Sharma, 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, 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.

## The third wave: WebAssembly

## 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.

### 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.

## WebAssembly

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.

## 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.

# Project

# Approach

# Design

# Implementation

This is the chapter on implementing Nebula.

## Tech stack

Rust/Docker/Etc.

# Discussions and future works

| Header 1 | Header 2 |
|---|---|
| Row 1 | Data |
| Row 2 | Data |

Table 1: Example Table

## Some Graphs

## References