A QUANTITATIVE STUDY ON THE PERFORMANCE AND SCALABILITY OF JAMSTACK IN COMPARISON TO A MONOLITHIC WEB ARCHITECTURE

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Bachelor's thesis

Häme University of Applied Sciences (HAMK)

Bachelor's Programme in Information and Communication Technology

Spring, 2023

Sam Whitley

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and Communication Technology — Abstract

Author	Sam Whitley	Year 2023
Subject	A Quantitative Study on the Performance and Scalability of Jamstack in Comparison to a Monolithic Web Architecture	
Supervisor	Petri Kuittinen	

Although the monolithic web architecture is still widely used and dominant today, developers are always actively seeking ways to deliver static content in ways that give greater performance, while using fewer moving parts and which requires far less maintenance. This is where the Jamstack web architecture comes in. Not only does it promise faster site performance, but also increased security and scalability, as well as making it more fun to develop and create for. Whilst this has been proven many of times already in different publications, the real question still looms: *How much faster and scalable is Jamstack?*

The aim of this research study was not only limited to gathering more quantitative data on the performance and scalability of the Jamstack web architecture, but also to compare it to the monolithic web architecture of WordPress and analysis how much better the Jamstack architecture is in terms of performance and scalability.

To test and evaluate the performance and scalability of the two web architectures, a total of eight Linux-based virtualised environments were tested, half of which were created using Jamstack and the other WordPress. These environments were then deployed on different virtualisation technologies, WSL2 and VM, using the different web servers, such as Apache and NGINX, from the software stacks of LAMP and LEMP.

In summary, the research findings suggest that the Jamstack web architecture provided clear performance benefits when serving its content statically, especially under heavier loads, compared to the monolithic web architecture of WordPress.

Keywords	Comparison, Jamstack, monolithic web architecture, performance, scalability
Pages	52 pages and appendices 53 pages

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintätekniikan koulutus (AMK-tutkinto)		Tiivistelmä
Tekijä	Sam Whitley	Vuosi 2023
Työn nimi	Kvantitatiivinen tutkimus Jamstackin suorituskyvystä ja skaalautuvuudesta verrattuna monoliittiseen verkkoarkkitehtuuriin	
Ohjaaja	Petri Kuittinen	

Vaikka monoliittinen verkkoarkkitehtuuri on edelleen laajalti käytössä ja hallitseva, ohjelmistokehittäjät etsivät aina aktiivisesti tapoja tuottaa staattista sisältöä entistä suorituskykyisemmällä tavalla, jossa käytetään vähemmän liikkuvia osia ja paljon vähemmän ylläpitoa. Jamstack-verkkoarkkitehtuuri palvelee juuri tätä asiaa. Se lupaa paitsi nopeampaa sivuston suorituskykyä, myös turvallisuuden ja skaalautuvuuden parantamista sekä mukavampaa kehittämistä ja luomista. Vaikka tämä on todistettu jo lukemattomia kertoja eri julkaisuissa, todellinen kysymys on edelleen: *Kuinka paljon nopeampi ja skaalautuva Jamstack on?*

Tämän tutkimuksen tavoitteena ei ollut ainoastaan kerätä lisää kvantitatiivista tietoa Jamstack-verkkoarkkitehtuurin suorituskyvystä ja skaalautuvuudesta, vaan myös verrata sitä WordPressin monoliittiseen verkkoarkkitehtuuriin ja analysoida, kuinka paljon parempi Jamstack-verkkoarkkitehtuuri on suorituskyvyn ja skaalautuvuuden kannalta.

Näiden kahden verkkoarkkitehtuurin suorituskyvyn ja skaalautuvuuden testaamiseksi ja arvioimiseksi testattiin yhteensä kahdeksan Linux pohjaista virtualisoitua ympäristöä, joista puolet luotiin Jamstackilla ja puolet WordPressillä. Nämä ympäristöt otettiin sitten käyttöön eri virtualisointitekniikoilla, WSL2:lla ja VM:llä, ja niissä käytettiin eri verkkopalvelimia, kuten Apachea ja NGINXiä, LAMP ja LEMP ohjelmistopinoista.

Yhteenvetona tutkimustuloksista voidaan todeta, että Jamstack-verkkoarkkitehtuuri tarjosi selkeitä suorituskykyetuja, erityisesti raskaammissa kuormituksissa, verrattuna WordPressin monoliittiseen verkkoarkkitehtuuriin.

Avainsanat	Jamstack, monoliittinen verkkoarkkitehtuuri, suorituskyky, skaalautuvuus, vertailu
Sivut	52 sivua ja liitteitä 53 sivua

Acknowledgements

I would like to take this opportunity to thank my thesis supervisor, Petri Kuittinen, for his help and guidance throughout the entire thesis process. He has provided me with advice and feedback and has always promptly responded to my gruelling long essay-like messages, never leaving any of my questions or messages go unanswered. Furthermore, where credit is due, I would also like to credit and thank him for suggesting the topic on the Jamstack architecture.

-Sam Whitley

List of Acronyms

API:	Application Programming Interface
CDN:	Content Delivery Network
CLI:	Command Line Interface
CMS:	Content Management System
CrUX:	Chrome User Experience Report
CSS:	Cascading Style Sheets
CSV:	Comma Separated Values
DOM:	Document Object Model
FCP:	First Contentful Paint
FID:	First Input Delay
GPSI:	Google PageSpeed Insights
HAR:	HTTP Archive
HTML:	HyperText Markup Language
JSON:	JavaScript Object Notation
LAMP:	Linux, Apache, MySQL, PHP
LEMP:	Linux, NGINX, MySQL/MariaDB, PHP/Perl/Python
PHP:	Hypertext Preprocessor
PHP-FPM:	FastCGI Process Manager
PLT:	Page Load Time
REST:	Representational State Transfer
SPSS:	Statistical Package for the Social Sciences
SQL:	Structured Query Language
SSG:	Static Site Generator
STDOUT:	Standard Output
TTFB:	Time to First Byte
VM:	Virtual Machine
VU:	Virtual User
W3Tech:	World Wide Web Technologies
WSL:	Windows Subsystem for Linux

Contents

Figures, Tables, and Equations

Appendices

1 Introduction

Web development has definitely come a long way since Web 1.0, the early stages of the World Wide Web's evolution when, back then, websites consisted mainly of just plain static pages (Sharma, 2022). As of the year 2023, web development has seen significant advancements. For example, web browsers have become more powerful, JavaScript has matured, and WebAssembly has gained more importance (Biilmann & Hawksworth, 2019, p. v). Unfortunately, these advancements have also raised the user expectations for faster sites and application responses, requiring developers as well as businesses to move away from the monolithic architecture and explore new approaches to developing sites and applications that perform as fast as possible, while ensuring the security and scalability of them. (Biilmann & Hawksworth, 2019, p. v; Vistola, 2021) One such approach that has gained momentum in recent years is Jamstack, which aims to not only make sites perform faster, but to also make them more secure, scalable, as well as fun to develop and create for. (Biilmann & Hawksworth, 2019, p. vi)

1.1 Previous Studies

Several publications have emerged that compare multiple user-centric performance metrics of Jamstack sites. In the article *A Look at Jamstack's Speed by the Numbers (2019)*, Artem Denysov, collected and analysed user-centric performance metrics of different content managements systems (CMSs) and a content delivery network (CDN) hosted Jamstack site from the *Chrome User Experience Report* (CrUX) and *The HTTP Archive*, such as Time to First Byte (TTFB), First Contentful Paint (FCP), and First Input Delay (FID). According to the article, the results indicated that Jamstack sites generally outperformed CMSs due to the benefits such as statically serving pages with a CDN, as well as the reduced TTFB time, which are known for offering better performance. (Denysov, 2019)

Another publication, a Finnish bachelor's thesis by Markus Matilainen (2020), on the other hand, compared the page load times of three sites. The first site used WordPress, the second used WordPress with Gatsby, and the third used a Netlify CMS with Gatsby. The author of the thesis used different performance testing tools such as Lighthouse, Pingdom, and GTmetrix to measure the page load times. According to the thesis, it was revealed that using a static website considerably improved the page load times compared to only using CMSs. Furthermore, the performance results on the mobile version of the site were even more evident compared to its desktop counterpart. (Matilainen, 2020)

1.2 Objectives and Questions

The objective of this research study was to gather more numerical data, also known as quantitative data, on the performance and scalability of the Jamstack architecture and compare it to the monolithic architecture of WordPress by utilising various open-source performance testing tools, like Grafana k6 and Sitespeed.io in combination with the different web servers from software stacks such as LAMP (Linux, Apache, MariaDB, and PHP) and LEMP (Linux, NGINX, MariaDB, and PHP). More specifically, this research study hopes to answer the following research questions bellow:

Question 1: *How does the performance of the Jamstack web architecture compare to the monolithic web architecture of WordPress in terms of different user-centric performance metrics?*

Question 2: *How does the scalability of the Jamstack web architecture compare to the monolithic web architecture of WordPress in handling different patterns of traffic?*

1.3 Scope and Limitations

The scope of this research study was only limited in comparing the performance and scalability of Jamstack and the monolithic web architecture, WordPress. It will not cover any other web architecture(s) or any other aspects of web development, such as security, cost (whether it be direct or indirect costs), or the developer experience. Instead, it will solely focus on the certain user and server-centric metrics, like the browser and visual metrics, which are metrics used in measuring “how quickly the site can load and display all of its visual elements on the screen” (Walton, 2022), as well as HTTP and iteration metrics, which are Grafana k6’s built-in metrics used in measuring “how a system performs under different test conditions”. (Grafana k6, 2023c)

2 Jamstack

This chapter hopes to provide readers with a brief overview of the Jamstack architecture and the technologies surrounding it. It will cover different aspects such as its definition, advantages and disadvantages, as well as its current state. By the end of this chapter, readers should have a general understanding of some of the inner workings of Jamstack and the reasons for its increasing adoption and popularity amongst web developers.

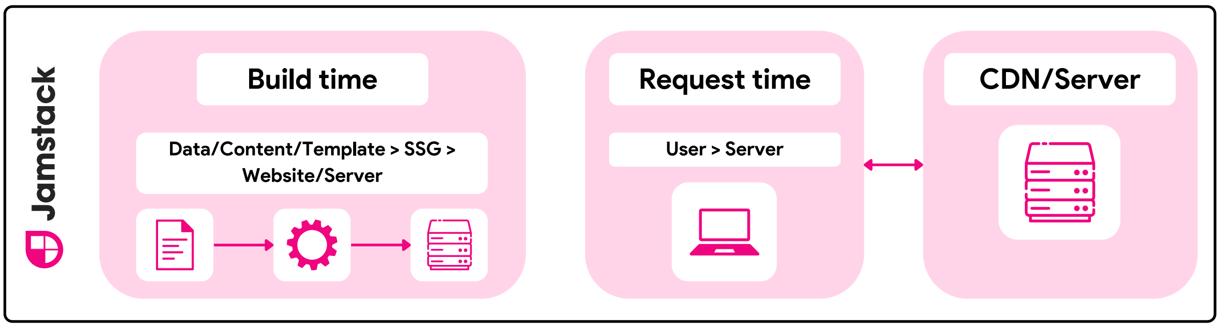
For any readers interested in delving deeper into the meaning or the technologies behind Jamstack, it is recommended to read either the "*Modern Web Development on the Jamstack*" or "*The Jamstack Book*". These books provide a comprehensive understanding of Jamstack by presenting multiple practical examples of building various Jamstack websites, as well as including a case study that demonstrates a company’s migration from a monolithic architecture to the Jamstack architecture.

2.1 Definition

Jamstack (formerly stylised as JAMstack) is a modern approach to web architecture that focuses on delivering fast, secure, and scalable static sites and applications with dynamic-like content. (Wallis, 2022) This approach is originally based on three technologies: JavaScript, APIs, and Markup, which forms the acronym of “Jam” in Jamstack. (Biilmann & Hawksworth, 2019, p. vii) At its core, Jamstack emphasises decoupling the web experience layer (client-side) from the data and business logic (server-side), a shift from a monolithic architecture model where these abstraction layers are tightly coupled together. (Biilmann & Hawksworth, 2019, p. 1; Jamstack.org, n.d.) Furthermore, Jamstack also emphasises in pre-rendering its static files and assets using a Static Site Generator (SSG) and serving them directly from a Content Delivery Network (CDN). (Biilmann & Hawksworth, 2019, p. viii)

Figure 1 below shows an example of a Jamstack architectural process. The figure, which was adapted from Bejamas, a software company specialised in Jamstack development, demonstrates how a Jamstack website functions. In the figure, it is shown that pre-built markup and optimised assets are served faster because there is no need to query the database as the files are already complied and served to the browser from a CDN. This drastically reduces the cumbersome Jamstack workflow hindrances and excess maintenance. (Kostrzewa, 2020)

Figure 1. Jamstack Architectural Process (Kostrzewa, 2020)



Note. Figure 1 was created by Sam Whitley using the free online graphic tool Canva (https://www.canva.com/). The design was adapted from Bejamas, originally created by Denis Kostrezewa. Copyright 2023 by Bejamas.io (https://bejamas.io/blog/jamstack/).

2.2 Advantages

As previously mentioned, Jamstack offers many advantages in addition to the faster performance, improved security, and scalability benefits. For example, hosting is more affordable and, in some cases, free (e.g., GitHub Pages) due to requiring far less server-side processing than its monolithic counterparts. This, in turn, should allow front end developers to focus more on the front-end aspects of development, leading to a quicker, more focused, and more cost-effective development process. Ultimately, this results in a better developer experience and more efficient development. (WTF Is Jamstack?, n.d.; Wallis, 2022; Falconer, 2022)

2.3 Disadvantages

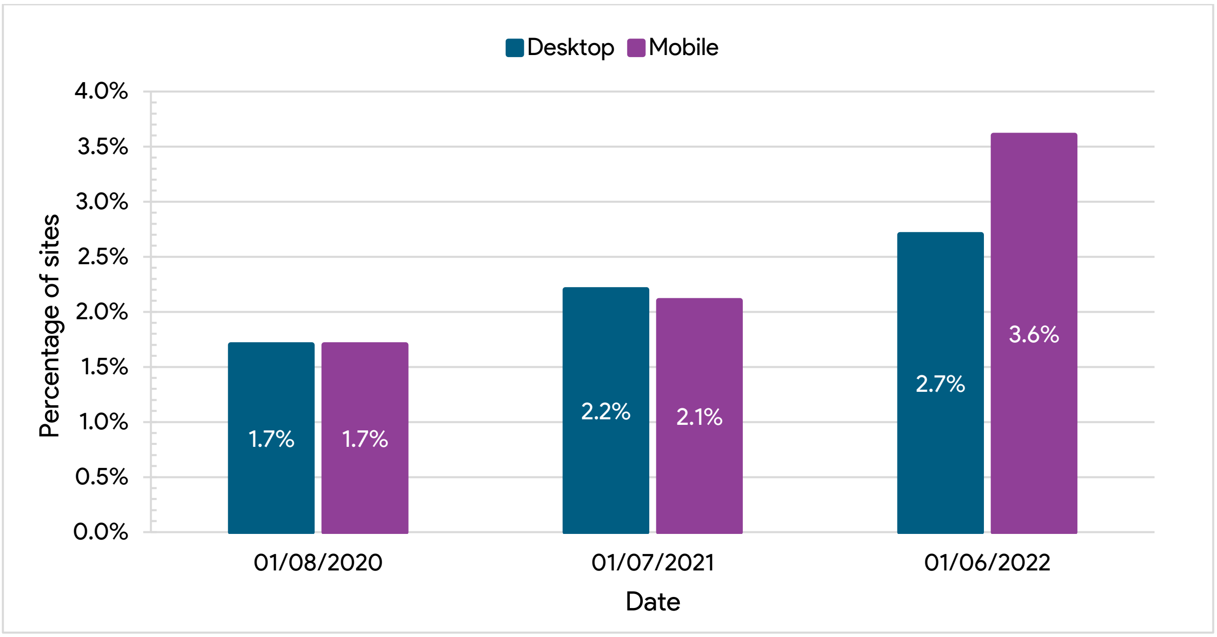
With its advantages, Jamstack also has its disadvantages. According to *The Jamstack Book*, written by Raymond Camden and Brian Rinaldi, while there are improved Jamstack tools (tools such as Next.js, Nuxt, and Gatsby) and services that allow users to make almost any kind of site, there might be certain situations where the Jamstack approach may not make much sense. For example, websites that rely heavily on user-generated content or where the content is continuously updated, might not be ideal for Jamstack, as it can be overly complex, difficult to implement, or it may even negate some of the overall performance benefits of Jamstack. Similarly, dashboard applications that rely heavily on server-side processing by utilizing different application programming interfaces (APIs) to populate charts and data tables might make perfect sense for Jamstack. However, this may put unnecessary load on the user and may not be an optimal solution. (Camden & Rinaldi, 2022, p. 9)

2.4 Current State

In recent years, Jamstack has become increasingly popular as developers, as well as companies, are seeking greater reliability, scalability, and security from their websites. Major brands such as PayPal, Nike, and Shopify have adopted the architecture by migrating several of their websites to Jamstack. In addition, large companies like Microsoft and Cloudflare have launched their own Jamstack offerings, such as Microsoft Azure's Static Web Apps service and Cloudflare's Cloudflare Pages. (Kostrzewa, 2020; Biilmann, 2021; Krzywda, 2021)

According to an annual report called *The Web Almanac*, which is a comprehensive yearly report on the state of the web, has revealed that the Jamstack adoption has seen a steady increase in recent years. For example, from 2020 to 2022 the report shows that desktop Jamstack websites have experienced a 58.8% increase from 1.7% to 2.7%. Mobile Jamstack sites, on the other hand, saw an increase of 111.8% from 1.7% to 3.6%, as shown in Figure 2. Figure 2 illustrates the percentage of Jamstack sites on desktop and mobile from 2020 to 2022. (Voss & Alam-Naylor, 2022)

Figure 2. The Growth of the Jamstack (Voss & Alam-Naylor, 2022)



Note. Adapted from "*The Web Almanac 2022, Part III Chapter 19, Jamstack*" by The HTTP Archive. Copyright 2023 by Web Almanac. Licensed under Apache 2.0. (https://almanac.httparchive.org/en/2022/jamstack#the-growth-of-the-jamstack)

3 Monolithic Architecture

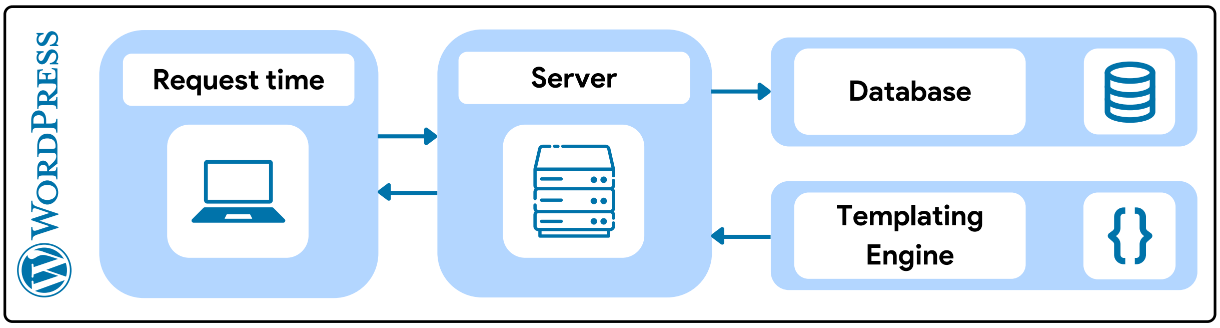
This chapter will provide readers with a brief overview of the monolithic architecture. It will cover aspects such as its definition, advantages and disadvantages, the monolithic content management system (CMS), WordPress, as well as the current state of WordPress.

For readers interested in delving deeper into the monolithic web architecture or how it compares to newer web architectures, like the microservices architecture, it is recommended to read either the article "*Microservices vs monolithic architecture*" written by Chandler Harris or the article "*Monolithic architecture*" written by Rahul Awati and Ivy Wigmore.

It should also be noted that while Jamstack websites do not fall under the microservices architecture, they do fall under the micro frontend architecture category, which is a combination of microservices and the frontend. (Dziuba, 2021a; Dziuba, 2021b) According to Anna Dziuba, VP of Delivery at Relevant Software, each static page with HTML and JavaScript is considered a micro frontend that can be utilized by any REST API, even those built upon the microservices architecture. (Dziuba, 2021a)

Figure 3 below shows an example of a monolithic web architecture process. The figure, which was adapted from Bejamas, a software company specialised in Jamstack development, demonstrates how a monolithic website functions using a LAMP stack as the example. When a user requests a page, the server first queries a database and then combines the results with the data from the page's markup and plugin to generate an HTML document in the browser. (Kostrzewa, 2020)

Figure 3. Monolithic Web Architecture Process (Kostrzewa, 2020)



Note. Figure 3 was created by Sam Whitley using the free online graphic tool Canva (https://www.canva.com/). The design was adapted from Bejamas, originally created by Denis Kostrezewa. Copyright 2023 by Bejamas.io (https://bejamas.io/blog/jamstack/).

3.1 Definition

A monolithic architecture is a traditional software program model that is self-contained and independent from other applications. (Harris, n.d.) In this context, the term "*monolithic*" means that something is composed all-in-one piece, according to Awati & Wigmore, 2022. Additionally, the same term can also refer to something either being too large or unable to be changed. (Awati & Wigmore, 2022; Cambridge Dictionary, n.d.)

3.2 Advantages

The monolithic architecture offers several advantages over other traditional web architectures, including Jamstack, which explains why many applications, as well as websites, are still reliant and created on this development model. For instance, organizations that opt for this kind of architectural approach often experience faster development speeds since the application is built on a single code base, which refers to the application's source code. Another advantage of the architecture is its ease of deployment, as one executable file or directory can make deploying an application much easier. (Awati & Wigmore, 2022; Harris, n.d.; Sheldon, 2023)

In addition to all of this, the monolithic architecture can even sometimes offer faster performance due to the centralized code bases and repositories where one application programming interface (API) can often perform the same function as numerous APIs perform with a microservice approach. Because of this, end-to-end testing is more simplified since a monolithic application is one simple, centralized unit compared to a distributed application. Finally, since the application’s code is in one place, it is easier to debug something by following requests. (Harris, n.d.)

3.3 Disadvantages

Netflix, the popular streaming service, is a prime example of how a monolithic application can be quite effective, as mentioned in the previous chapter. However, this type of architecture can lead to several disadvantages in the long term. As a monolithic application becomes larger in scale it often experiences slower and much more complex development. For example, making minor changes even to a single function in a code will require compiling and testing the entire platform, which counteracts the agile approach some developers favour. Additionally, individual components cannot be scaled, which might present scalability challenges. Another disadvantage could be that an error in any module could affect the entire application's availability, reducing its reliability. Finally, changes in the framework or language would also affect the entire application, making those changes often expensive and time-consuming, which might present a barrier to technology adoption. (Harris, n.d.)

3.4 WordPress

WordPress is a popular open-source content management system (CMS) that is used in creating, modifying, and maintaining websites. (Domantas, 2023) Based on PHP and MySQL, it was initially released in 2003 as a blogging tool by Matt Mullenweg and Mike Little. (Javatpoint, n.d.) Since then, WordPress has grown into the most widely used content management system to date with a CMS market share of close to two-thirds (63.3%) of websites that are used today, as shown in Table 1. (W3Techs, 2023b)

Table 1 illustrates the percentages of websites using various content management systems (CMSs). 31.8% of websites do not utilize any content management systems which W3Techs tracks. WordPress accounts for 43.2% of all websites, giving it a market share of 63.3% in terms of share of website management systems. W3Techs reports are updated on a daily basis. (W3Techs, 2023b)

Table 1. Usage Statistics of CMSs (W3Techs, 2023b)

CMS	CMS Market Share	Website Share
WordPress	63.3%	43.2%
None	N/A	31.8%
Shopify	5.5%	3.8%
Wix	3.7%	2.5%
Squarespace	3.1%	2.1%
Joomla	2.7%	1.8%
Drupal	1.8%	1.2%
Adobe Systems	1.7%	1.1%
Google Systems	1.2%	0.8%
PrestaShop	1.1%	0.7%
Bitrix	1.1%	0.7%

Note. Data adapted from "*Usage Statistics of Content Management Systems*" by World Wide Web Technology Surveys, 27th March 2023. Copyright 2023 by Q-Success DI Gelbmann GmbH. (https://w3techs.com/technologies/overview/content_management)

3.5 Current State of WordPress

Since its release exactly 20 years ago in 2003, WordPress has evolved from being a simple blogging tool to a fully-fledged CMS, with the capability of being used as an e-commerce website. Because of this, WordPress' versatility in having the ability to create different types of sites, such as blogs and e-commerce sites, has caused its CMS market share to increase throughout the years from 54.3% in 2012 to an impressive 63.3% in 2023, with no signs of slowing down according to W3Techs' yearly trends on CMSs. Another reason for its continued growth could be the introduction of their REST API, which allows developers to use it in a headless way, as a backend, whilst separating the frontend where only the frontend-focused technologies can be used. (Moreira, 2020; W3Techs, 2023a)

Although there have been other content management systems, such as Shopify, that briefly saw its CMS market share increase from 0.3% in 2014 to 6.6% in 2022; it is highly unlikely for Shopify or any other CMS to surpass, let alone compete against WordPress in terms of market share in the near future. (W3Techs, 2023a)

WordPress is widely used by many, as prominent big-name brands, such as Microsoft (Microsoft News Center), Sony (Sony Music), and CNN (CNN Press Room) have been using WordPress to run most of their blogs, as well as websites. (Ahmed, 2023; WPBeginner, 2023; Dodson, 2016)

4 Methodology

This chapter aims to provide readers with an overview of the methodology used during the research study. It will cover different aspects such as the research approaches, research design, reliability and validity, hardware and software configurations, as well as the tools used.

4.1 Research Approaches

A research approach is an approach used in collecting, analysing, and interpreting data. There are primarily three research approaches to choose from, which are *quantitative research*, *qualitative research*, or "*mixed methods research*". (Budert-Waltz, 2021)

Quantitative research involves collecting and analysing the numerical data to describe, predict, or control a phenomenon. The analysis of numerical data is complex and should be addressed systemically. (Budert-Waltz, 2021) Quantitative research also relies on using deductive reasoning, which is a logical approach that starts with one or more general statements and works its way towards reaching a logical conclusion. (Budert-Waltz, 2021; Sirisilla, 2023)

Qualitative research, on the other hand, involves collecting, analysing, as well as interpreting comprehensive narrative and visual data to gain insights into a particular phenomenon. Qualitative research encompasses many aspects of a phenomenon while striving to study them as they exist naturally. (Budert-Waltz, 2021) Qualitative research also relies on using inductive reasoning, referred as induction, which involves the construction or evaluation of prepositions from specific examples. (Budert-Waltz, 2021; Sirisilla, 2023)

Finally, the “mixed methods research” combines both quantitative and the qualitative approaches bringing together both data types into one study. “Mixed methods research” builds on the relationship and the strength that exists between quantitative and qualitative research methods, providing greater insight into the phenomenon being researched. (Budert-Waltz, 2021)

Table 2. The Differences Between Quantitative and Qualitative Research (Streefkerk, 2023)

Quantitative Research	Qualitative Research
Focuses on testing hypotheses and theories	Focuses on exploring ideas and formulating a theory or hypothesis
Analysed through math and statistical analysis	Analysed by summarizing, categorizing, and interpreting
Mainly expressed in numbers, graphs, and tables	Mainly expressed in words
Requires many respondents	Requires few respondents
Closed (multiple choice) questions	Open-ended questions
Key terms: testing, measurement	Key terms: understanding, context, complexity

Note. Adapted from “*Qualitative vs. Quantitative Research | Differences, Examples & Methods*” by Raimo Streefkerk, 2023. (https://www.scribbr.com/methodology/qualitative-quantitative-research/)

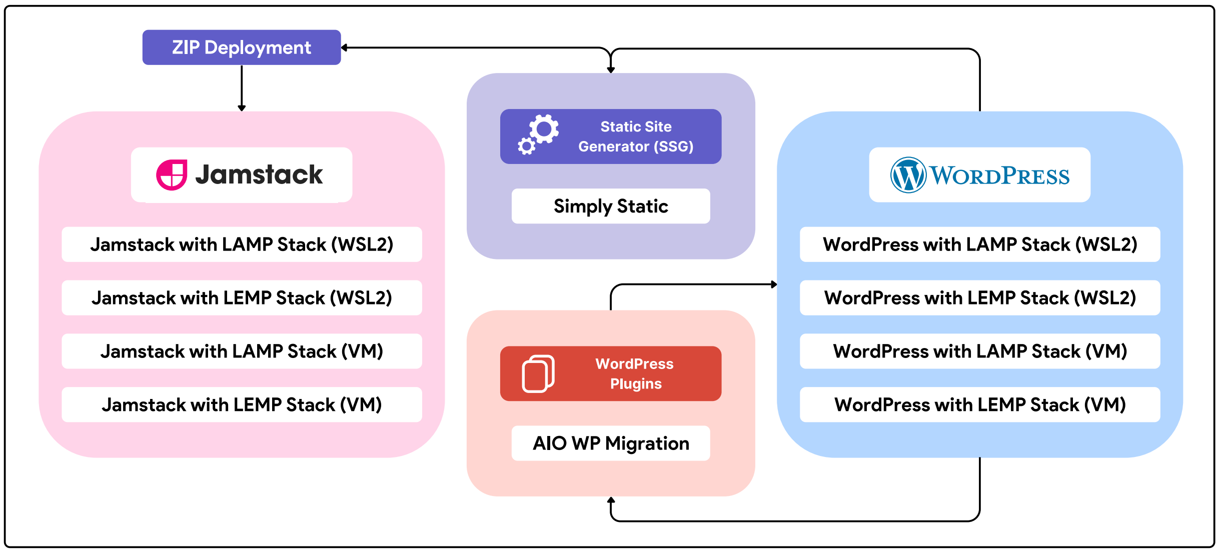
4.1.1 Research Approach Selection

The quantitative research approach was chosen as the objective of the research study was to gather more numerical data, also known as quantitative data, on the performance and scalability of both web architectures; Jamstack and the monolithic web architecture WordPress.

4.2 Research Design

To ensure a fair and accurate comparison between the two architectures, *Simply Static*, a static site generator (SSG), was used to generate a static site from the WordPress site. The generated static site then served as a foundation and was deployed to the rest of the Jamstack environments using ZIP. Additionally, the WordPress plugin called *All-in-One WP Migration* was also used to migrate one WordPress site to all the other WordPress sites, whilst ensuring that every site across all environments, including Jamstack, were identical one-to-one copies of one another. Finally, all environments were hosted locally on a virtual machine (VM), as well as on a Windows Subsystem for Linux (WSL2). This was done to eliminate any sort of external factors, such as network latency or any fluctuations in the internet speed as that could affect the test results.

Figure 4. Testing Environments Flowchart (Whitley, 2023d)


ZIP Deployment
Jamstack
Jamstack with LAMP Stack (WSL2)
Jamstack with LEMP Stack (WSL2)
Jamstack with LAMP Stack (VM)
Jamstack with LEMP Stack (VM)
Static Site Generator (SSG)
Simply Static
WordPress Plugins
AIO WP Migration
WordPress
WordPress with LAMP Stack (WSL2)
WordPress with LEMP Stack (WSL2)
WordPress with LAMP Stack (VM)
WordPress with LEMP Stack (VM)

Note. Figure 4 was created by Sam Whitley using the free online graphic design tool Canva (https://www.canva.com/).

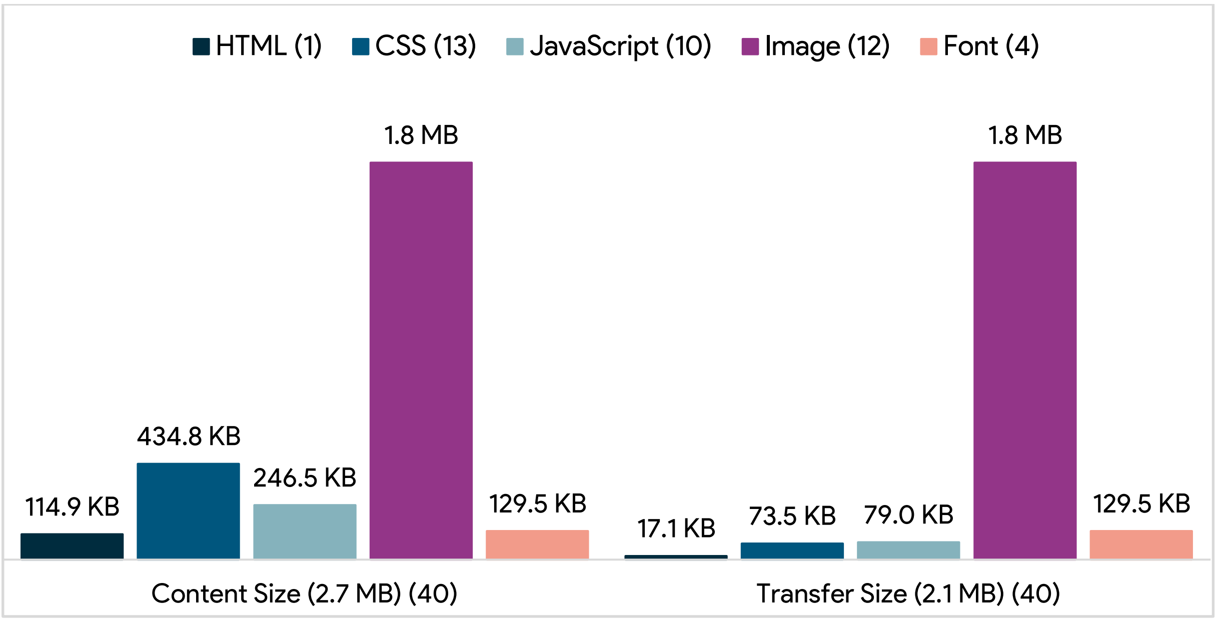
4.3 Reliability and Validity

In research, especially quantitative research, the researcher should always consider using reliability and validity when creating their research design, planning their methods, and writing their results. Neglecting to do so can not only lead to various types of bias in the research findings but can negatively affect the work in question. (Middleton, 2023) Reliability refers to the consistency of the measure. A high precision indicates that the measurement method produces similar results under the same circumstances. In other words, if a researcher measures, for example, an item or person multiple times, they want to obtain reproducible, comparable values. (Frost, n.d.) Validity, however, is a broader concept than reliability and refers to whether the measurements reflect on what they claim to measure. Researchers, for example, must always question themselves whether their results reflect what they think they measure, whether something else entirely has occurred, and ensure that an instrument measures exactly what it is intended. (Frost, n.d.) Even though reliability and validity are somewhat closely related, they refer to distinct concepts, for example, a measurement can be reliable without being valid, and likewise, if a measurement is valid, then it is usually also reliable. (Middleton, 2023)

To ensure the reliability of all tests conducted during the research study, each test was conducted for a prolonged amount of time on all WordPress and Jamstack sites. For instance, when using Sitespeed.io, 100 iterations of each test was ran, meaning each website was tested precisely 100 times. The tests conducted on desktop typically took around 10 seconds per iteration, or roughly 16 minutes and 40 seconds to complete 100 iterations. Mobile tests, on the other hand, took twice as long, or 20 seconds per iteration, and 33 minutes and 20 seconds to complete 100 iterations. Furthermore, these tests were conducted on four WordPress environments and four Jamstack environments, with the only difference between them being that each environment utilized a different virtualization technology (WSL2 or VM) and web server (Apache or NGINX).

Regarding the validity of all tests conducted during the research study, the website page weight (meaning the size of a page) was set at 2.1 MB. This decision was based on a report on page weights by The HTTP Archive, 2023, which showed that this value is the closest to the p50 (median) transfer size value for all desktop and mobile pages worldwide, which is 2199.45 kB. This value was calculated (as an average) of the median desktop page weight of 2340.6 kB and the median mobile page weight of 2058.3 kB, calculated in Equation 1. Furthermore, this can be viewed in more detail in Figure 5.

Figure 5. Requests and Sizes per Content Type (Whitley, 2023a)



Note. This is a clustered column chart that comprises of two charts, content size and transfer size. Each column displays a data measurement unit, which represents each content type's size. Furthermore, the chart legend (located at the top part of the chart) displays a numerical value after each content name, which indicates the number of requests.

Equation 1. Median of Desktop and Mobile Page Weight

$$Median\ (p50) = [(2340.6\ kB)\ +\ (2058.3\ kB)\ /\ 2\ =\ 2199.45\ kB]$$

Even though the reliability and validity of the research study have been assessed, it is important to note that other factors, such as hardware and network configurations, could still impact the test results. Several steps have been taken to mitigate these possibilities, which are listed below in Table 3.

Table 3. Measures Taken to Ensure Reliability and Validity

Aspect	Measures Taken by the Author
k6/Results	Ensured that all of the test results, even the questionable ones, were valid. For example, specific tests, such as the average-load tests on WSL2 LAMP and LEMP produced questionable results. To ensure the validity of this, the author verified that all of the packages and services were running on the same version (besides for the web server) and ran those tests again, which resulted in the same outcome.
k6/Smoke Test	Ensured that all the testing environments did not throw any errors when under minimal load.
Server/ Database	Ensured that all WordPress websites used the same SQL database, in this case, MariaDB.
Sitespeed.io/ Browser	Ensured that Sitespeed.io tested each page with a fresh browser profile (caching disabled), for every iteration.
Sitespeed.io/ Setup	Ensured that a test was performed on the host OS (the author's primary OS), under the same circumstances and on the same day. During the tests, no other programs were running nor did the author use the computer during that time.
Website/ Content Type	Ensured that each page content type, including third-party resources, like Google Web Fonts, have been hosted locally.
Website/HTTP Protocol	Ensured that all tests used the same HTTP protocol, in this case, HTTP/1.1. Chrome and Firefox only support HTTP/2 for HTTPS connections according to Hogan & Garnett, 2021.
Website/IP Address	Ensured that all tests used a local IP address, rather than the loopback address, also known as localhost.
Website/Page Weight	Ensured that each environment had the same exact page weight. Also ensured that the page weight is close as possible to the median according to The HTTP Archive, 2023.

Note. Table 3 outlines the measures taken to ensure the reliability as well as the validity of all of the test results.

4.4 Hardware and Software Configurations

This chapter will provide readers with all of the hardware and software configurations used to run all the performance and scalability tests conducted during this research study. This includes the computer hardware specifications, software stack configurations for the WSL2 and VM testing environments, Sitespeed.io and Grafana k6 settings, as well as several commands used to run the tests.

Table 4. Hardware Specifications

Component	Specification
CPU	AMD Ryzen 9 5900X 3.7 GHz 12-Cores
GPU	Gigabyte GeForce RTX 2070 SUPER 8GB WINDFORCE OC 3X
Motherboard	Asus ROG STRIX B550-F GAMING (WI-FI)
Operating System	Microsoft Windows 11 Education 64-bit (10.0.22621, Build 22621)
RAM	Kingston FURY Renegade DDR4-3600 C16 32GB (2x16GB)
Storage	Western Digital Black SN850 1TB

Note. All the tests performed on Sitespeed.io and k6 were conducted locally on the author's personal computer using a wired Ethernet connection.

Table 5. Testing Environments

Environment ID	Environment Name
L1	WordPress with LAMP Stack (VM)
L2	WordPress with LAMP Stack (WSL2)
L3	~~Jamstack (Netlify Local Development Environment)~~
L4	Jamstack with LAMP Stack (WSL2)
L5	Jamstack with LAMP Stack (VM)
L6	WordPress with LEMP Stack (VM)
L7	Jamstack with LEMP Stack (VM)
L8	WordPress with LEMP Stack (WSL2)
L9	Jamstack with LEMP Stack (WSL2)

Note. All the local testing environments (L1 through L9) were set up on the author's personal computer in a VM or on a WSL2 environment. Environment L3 was scrapped due to the lack of a suitable WordPress counterpart. The "L" in the environment ID is an abbreviation for "Local".

Table 6. LAMP and LEMP Configuration [WSL2 and VM]

Name	LAMP Stack Configuration	LEMP Stack Configuration
Database	MariaDB 10.10.3	MariaDB 10.10.3
Kernal/ Architecture	WSL2: Linux 5.15.90.1-microsoft-standard-WSL2 x86_64 VM: Linux 5.10.0-21-amd64 x86_64	WSL2: Linux 5.15.90.1-microsoft-standard-WSL2 x86_64 VM: Linux 5.10.0-21-amd64 x86_64
Operating System	Debian GNU/Linux 11 (Bullseye)	Debian GNU/Linux 11 (Bullseye)
PHP	PHP 8.1.15	PHP 8.1.15
Web Server	Apache 8.2.15	NGINX 1.23.3
WordPress	WordPress 6.1.1	WordPress 6.1.1

Note. Apart from the web server and kernel/architecture, both LAMP and LEMP stack configurations are identical on both WSL2 and VM environments.

Table 7. Sitespeed.io Runtime Settings [Desktop] (Whitley, 2023b)

Details	Configuration/Version
Browser	Chrome 112.0.5615.49
Browsertime	17.8.0
Connectivity	No connectivity settings
Number of runs	100
OS	Windows 11
Sitespeed.io	27.3.0
User Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Safari/537.36
View port	1366x708

Note. Please be aware that all the desktop performance tests were run only using the Chrome browser. All the tests were executed for 100 iterations (number of runs), taking an average time of ≈1000 seconds (equivalent to 16 minutes and 40 seconds) for each environment.

Table 8. Sitespeed.io Runtime Settings [Mobile] (Whitley, 2023c)

Details	Configuration/Version
Android version	12
Browser	Chrome 112.0.5615.48
Browsertime	17.8.0
Connectivity	No connectivity settings
Number of runs	100
OS	Android
Phone model	GM1913
Sitespeed.io	27.3.0
User Agent	Mozilla/5.0 (Linux; Android 12; GM1913) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/112.0.0.0 Mobile Safari/537.36
View port	412x797

Note. Please be aware that all mobile performance tests were run on a physical Android phone using only the Chrome browser. All tests were executed for 100 iterations (number of runs), taking an average time of ≈2000 seconds (equivalent to 33 minutes and 20 seconds) for each environment.

Table 9. Grafana k6 Load Test Types (Grafana k6, 2023b)

Test Type	Description
Smoke Test	Smoke tests ensure that the script works and that the system is performing as expected under minimal load.
Average-Load Test	Average-load tests assesses how the system is performing under expected normal conditions.
Stress Test	Stress tests assesses how well the system is performing under extreme loads when exceeding the expected averages.
Spike Test	Spike tests validate the behaviour and survival of the system when subjected to a sudden, short, and massive increase in activity.

Note. Adapted from "*Test Types, Load Test Types*" by Grafana k6, 2023. Copyright 2023 by Grafana Labs (https://k6.io/docs/test-types/load-test-types/). All the Grafana k6 test scripts were modified from the pre-existing example and tutorial scripts available on the Grafana k6 website in the documentation section (https://k6.io/docs/examples/).

Table 10. Grafana k6 Version (Whitley, 2023)

Name	Version
Grafana k6	v0.43.1 (2023-02-27T10:53:03+0000/v0.43.1-0-gaf3a0b89, go1.19.6, windows/amd64)

Note. All of the tests were performed on the author's personal computer.

Table 11. Grafana k6 Script Configurations

Script	Script Configuration
All Scripts	
All of the scripts are configured to make a single HTTP GET request every second. It then ensures that each check returns a 200-status code and that the content type is text/HTML. Additionally, it also checks that the homepage body includes the text "Lorem ipsum dolor"; dummy text to ensure that the page has loaded correctly. Examples of these checks can be found in Appendix D. At the end of each test, the script outputs the results into three locations: standard output (stdout), to an easy-to-read HTML report that aggregates the test results (the same as stdout), and onto a CSV file that includes all of the granular data (detailed data) from each test.	
smokeTest.js	This script uses three virtual users (VUs) during the test, which it loops for one minute.
averageLoadTest.js	This script utilizes a stage array that starts by ramping up the number of VUs to 100 over the next five minutes. It then maintains 100 VUs for 30 minutes before then ramping back down to zero VUs over the next five minutes.
stressTest.js	This script works similarly to the averageLoadTest.js script, but with a longer ramp-up duration of 10 minutes as well as an increased number of VUs from 100 to 200. It then maintains 200 VUs for 30 minutes before ramping back down to zero VUs over the next five minutes.
spikeTest.js	This script works by ramping up the number of VUs to 2000 over the next two minutes, before ramping back down to zero in the next minute.

Note. All the Grafana k6 test scripts were modified from all the pre-existing example and tutorial scripts available on Grafana k6's website in the documentation section (https://k6.io/docs/examples/).

4.5 Research Tools Used

This chapter will provide readers with an overview of the tools used in the research study. It will briefly describe each tool on how it was used, and why it was chosen.

4.5.1 Sitespeed.io

Sitespeed.io is an open-source web performance testing tool that helps users monitor and enhance their website's performance. (Sitespeed.io, 2023) It comprises of multiple open-source tools, such as Coach, which identifies performance issues with the website; Browsertime, which automates the execution of JavaScript in the browser to gather performance metrics; and PageXray, which converts HAR files into a more readable and user-friendly JSON format. (Hedenskog, 2021) Additionally, it also supports several third-party plugins from Google, such as GPSI (Google PageSpeed Insights) and Google Lighthouse. (Hedenskog, 2017) Figure 6 is a screenshot of Sitespeed.io running a performance test on one of the local virtual machines for one iteration.

Figure 6. Sitespeed.io Performance Test (Whitley, 2023)

```
Mozilla Firefox 106.0
Microsoft Edge 106.0.1370.52
[2023-03-27 18:05:25] INFO: Versions OS: linux 5.10.0-21-amd64 nodejs:
v16.16.0 sitespeed.io: 26.1.0 browsertime: 16.17.0 coach: 7.1.2
[2023-03-27 18:05:25] INFO: Running tests using Chrome - 100 iterations(s)
[2023-03-27 18:05:26] INFO: Testing url http://192.168.100.157/ iteration 1
[2023-03-27 18:05:35] INFO: Take after page complete check screenshot
[2023-03-27 18:05:35] INFO: Take cumulative layout shift screenshot
[2023-03-27 18:05:36] INFO: Take largest contentful paint screenshot
[2023-03-27 18:05:39] INFO: Get visual metrics from the video
[2023-03-27 18:05:40] INFO: http://192.168.100.157/ 41 requests, TTFB: 221ms,
firstPaint: 400ms, firstVisualChange: 467ms, FCP:, DOMContentLoaded: 618ms,
LCP: 461ms, CLS: 0.0024, TBT: 0ms, Load: 626ms, speedIndex: 582ms,
visualComplete85: 600ms, lastVisualChange: 600ms
[2023-03-27 18:05:42] INFO: HTML stored in /sitespeed.io/sitespeed-
results/192.168.100.157/2023-03-27-18-05-25
```

Note. The tool shown in Figure 6 is a standard output (stdout) of Sitespeed.io, created by Peter Hedenskog (https://github.com/sitespeedio/sitespeed.io).

The command shown in Figure 7 runs a 100-iteration test of environment L1. The command also includes two parameters: the *'name'* parameter, which gives the test a name, and the *'urlAlias'* parameter, which gives an alias to an URL and is mainly used to help distinguish between tests when reviewing the HTML reports.

Figure 7. Example Sitespeed.io Desktop Command (Whitley, 2023)

```
sitespeed.io http://192.168.100.157/ --video --visualMetrics --cpu
--visualElements -n 100 --name L1Desktop --urlAlias L1Desktop
```

Note. The tool shown in Figure 7 is a command from Sitespeed.io, created by Peter Hedenskog (https://github.com/sitespeedio/sitespeed.io).

In this research study, the open-source tool Sitespeed.io was used to measure the web performance of both the WordPress and Jamstack sites. Sitespeed.io was chosen over other performance testing tools, such as Pingdom or GTmetrix, because of its ease of use, feature-rich nature, open-source status and, most importantly, its ability to conduct all of the tests locally.

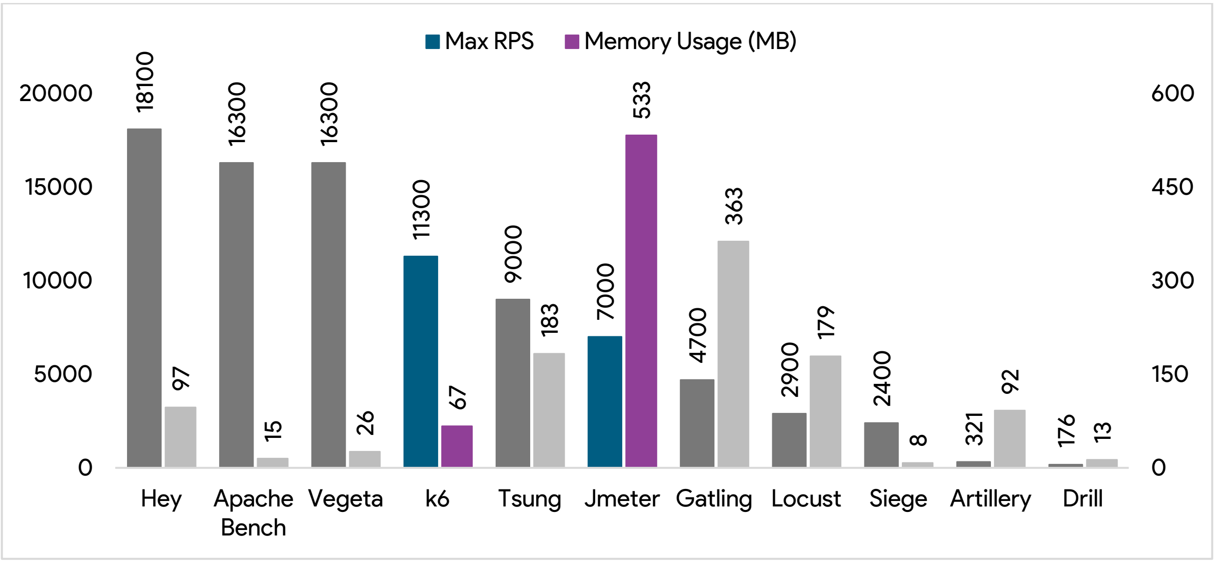
Because of its ease of use, Sitespeed.io allows users to run automated tests with a single command, as shown above in Figure 7. As previously mentioned in the reliability and validity chapter, depending on the platform in question, each test iteration can take anywhere from 10 to 20 seconds, and a full 100-iteration test can take between 17 to 33 minutes. After a test, Sitespeed.io generates a static HTML report that displays a large amount of numerical data on the different timing metrics. In addition, Sitespeed.io's other tools, such as Browsertime, the Coach, and PageXray, generate JSON and HAR files, which can be used to compare metrics from different tests using a website such as *compare.sitespeed.io*. (Hedenskog, 2022) Finally, the numerical data generated by Sitespeed.io was then imported into Excel, where it was visualised into several different figures and tables.

4.5.2 Grafana k6

Grafana k6 is an open-source load testing tool that was developed back in 2017 by Load Impact, which was then later acquired by Grafana in 2021. (Grafana k6, n.d.; Gustafsson, 2021) The tool allows users to test the performance and reliability of various services, such as websites, microservices, and APIs using a range of use cases that include load testing, browser testing, chaos and resilience testing, as well as performance and synthetic monitoring. (Grafana k6, n.d.) Although Grafana k6, similar to Sitespeed.io, supports browser testing through the xk6-browser extension, Sitespeed.io was chosen because it appeared to provide more numerical data compared to the xk6-browser extension. (Grafana k6, 2023a)

While there are several other popular alternatives to Grafana k6, such as the widely used open-source testing tool Apache JMeter, Grafana k6 was still chosen because it was newer, used a CLI (Computer Language Interface), as well as had better resource utilization according to Figure 8, were Grafana Labs compares Grafana k6 with several other open-source load testing tools. (van der Hoeven, 2021)

Figure 8. Max Traffic Generation Capability of Several Load Testing Tools (Lönn, 2020)



Note. Data adapted from "*Open-Source Load Testing Tool Review 2020*" by Ragnar Lönn at Grafana Labs, 2020. Copyright 2020 by Grafana Labs. (https://k6.io/blog/comparing-best-open-source-load-testing-tools/)

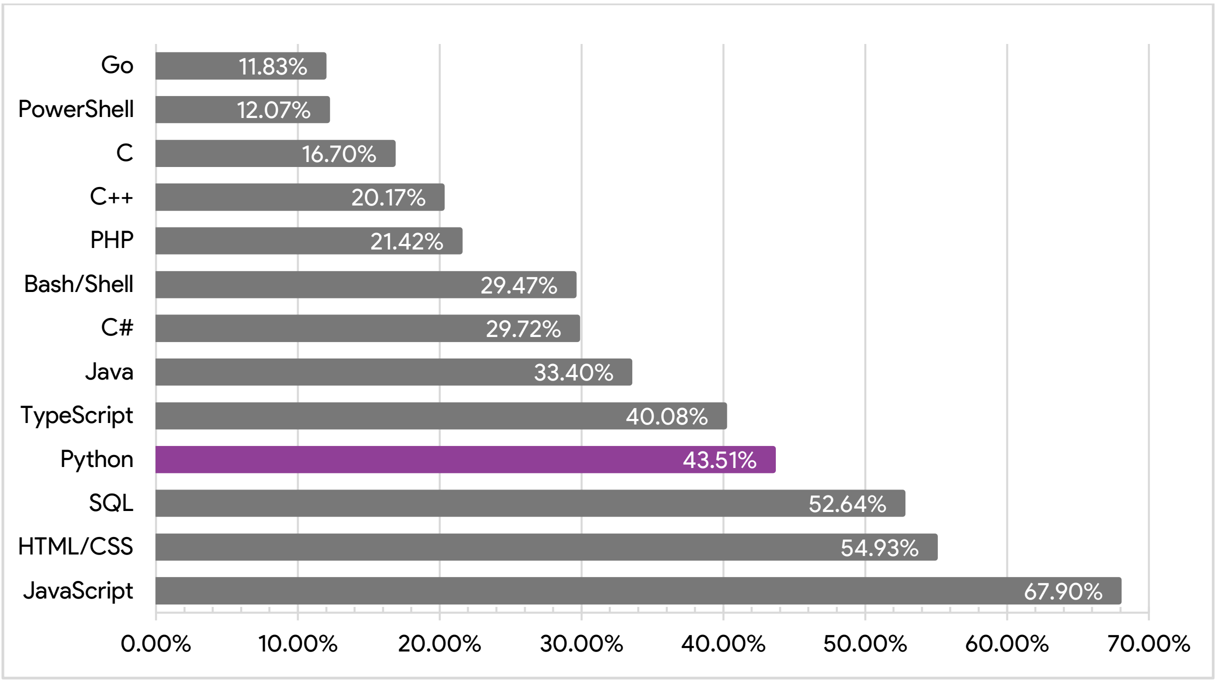
In this research study, Grafana k6 was used to perform several different types of load tests, which consisted of smoke testing, load testing, stress testing, and spike testing. Each test was implemented using JavaScript and was always designed with a specific testing objective in mind, which can be viewed in greater detail in Appendix E. All tests were conducted on the main page of the site (index.html) across all testing environments. At the end of each test, summarized results were outputted to multiple locations, including "stdout" (standard output) and as a HTML report generated using the k6-reporter, an extension created by Ben Coleman (benc-uk). The HTML report displayed results such as request groups, checks, HTTP metrics, and other statistics in a clear and easy-to-understand fashion. (Coleman, 2021)

4.5.3 Python

Python, one of the world's most popular programming languages has been employed in developing numerous technologies, such as Netflix's recommendation algorithm and software for autonomous vehicles. As a general-purpose programming language, Python is used in a range of applications, such as in data science, web and software development and automation. (Coursera Inc, 2022)

According to a recent developer survey conducted in May 2022 by Stack Overflow, Python was named the fourth most used programming language by professional developers with 43.51% votes, while JavaScript topped the list with 67.90% votes for the tenth consecutive time, as shown in Figure 9. (Stack Overflow, 2022)

Figure 9. Most Popular Technologies (Professional Developers) (Stack Overflow, 2022)



Note. Adapted from "*Stack Overflow Developer Survey 2022*" by Stack Overflow, 2022. Copyright 2023 by Stack Overflow. (https://survey.stackoverflow.co/2022/#most-popular-technologies-language-prof)

In this research study, Python, along with the data analysis and manipulation library Pandas, was used to analyse the data, briefly visualise it, and output the resulting data from Grafana k6 to a new CSV file, which can then be imported into Excel for further analysis and to create the final charts for the thesis. Furthermore, Python was also used to split the CSV data into multiple files by column name to bypass GitHub's 100 MB single file upload size limit.

Figure 10 illustrates a Python script that reads a CSV file generated by Grafana k6. It then parses the data based on the metric name in question, in this case, '*http_req_duration*', '*time*', and '*metric_value*' and resamples it to 30-second intervals. Finally, the data was then visualized through the print function and adjustments to the resampling time interval were made before outputting the data back into a CSV format.

Figure 10. CSV Output Python Script

```
import matplotlib.pyplot as plt
import os
from datetime import datetime

df = pd.read_csv('test_results.csv', index_col=0,
parse_dates=['timestamp'], date_parser=lambda x:
pd.to_datetime(float(x), unit='s'))

http_req_duration = df[df.index == 'http_req_duration']
http_req_duration.insert(0, 'metric_name', 'http_req_duration')
http_req_duration = http_req_duration.set_index('timestamp')
http_req_duration.loc[:, 'time'] = http_req_duration.index.time
http_req_duration = http_req_duration[['metric_name', 'time',
'metric_value']]
data_df = http_req_duration.resample('30s').mean(numeric_only=True)
data_df['metric_name'] = 'http_req_duration'
data_df = data_df[['metric_name', 'metric_value']]
print(data_df)
data_df.to_csv('testOutput.csv')
```

Note. The Python code in Figure 10 was created based on the tutorials and documentation from the following sources: *Data Independent*, *Robin Horn*, *Pandas' library documentation*, and *Python's documentation*.

4.5.4 Microsoft Excel

Microsoft Excel, released back in 1985, is a widely used spreadsheet software created by the Microsoft Corporation for organizing and manipulating data into columns and rows. (The Editors of Encyclopaedia Britannica, 2022; Techopedia, 2020) With Excel, users can perform various different mathematical functions on the data using numerous formulas to then visualize it through different charts. (The Editors of Encyclopaedia Britannica, 2022; Techopedia, 2020)

The research study utilised the spreadsheet software Excel in gathering and analysing the data from the performance testing tools Sitespeed.io and Grafana k6. In addition, Excel was then used to visualise that data through various tables and figures.

Initially, it was originally planned to use the statistical analysis software SPSS (Statistical Package for the Social Sciences) or R (the programming language) for data analysis and visualization for the research study. However, this plan was later abandoned due to the high cost of the SPSS license and the steep learning curve associated with R.

5 Results and Analysis

This chapter will provide readers with a summary of the performance and scalability metrics obtained from Sitespeed.io's and Grafana k6's test results. It includes tables and figures comparing both the web architectures performance and scalability, together with a brief analysis of several of the results obtained. For readers interested in viewing additional metrics, such as descriptive statistics tables, these can be found in the appendix section. In addition, miscellaneous data, such as result outputs (outputs like CSV, stdout, HTML reports), Excel files, and scripts, are available in the author's GitHub repository (https://github.com/Sam-Whitley/thesis).

5.1 Performance Results

As briefly mentioned in previous chapters, Sitespeed.io was used to test the performance differences between the two web architectures, Jamstack and the monolithic web architecture, WordPress, to determine how much of a performance difference there was between the two from a numerical perspective. To test this, a total of eight environments were used, half of which were WordPress environments, whereas the other half were Jamstack environments. These environments were then further divided into two separate technology virtualization groups, WSL2 and VM, alongside with different web servers, such as Apache and NGINX. This approach was chosen to not only help ensure the accuracy and reliability of the test results, but to also make a fair and accurate comparison between the two architectures using different environments with different virtualization groups and web servers. Furthermore, all the Jamstack and WordPress sites were exact one-to-one copies of each other. This was done again by first migrating a WordPress site to all other WordPress sites using the *All-in-One WP Migration* WordPress plugin, which exports and imports the WordPress site's database, media files, plugins, and themes. Next, a static site generator like *Simply Static* was then used to convert the existing WordPress site into a static site. The resulting static site was then exported as a ZIP file and imported into the different Jamstack environments, where it was statically hosted using either Apache or NGINX.

Once each of these environments were set up, the performance tests were run using Git Bash, an application for Windows that emulates the Git command line experience. (Atlassian, n.d.) Git Bash was used because Sitespeed.io's commands did not work on either Command Prompt or PowerShell command-line interfaces (CLIs). The command, as shown in Figure 7, was used to run the desktop performance tests. When running the mobile tests however, slight modifications were made to the command, which was the removal of the *'visualMetrics'* and *'video'* flags, as these options did not work for some reason, even after much troubleshooting.

These web performance tests were divided into two categories and were conducted over a two-day period. The first category that was tested was the desktop performance of the sites, which was run on the 12th of April 2023 and was completed in roughly three hours and 28 minutes, taking an average of 26 minutes per environment. The second category, on the other hand, tested the mobile version performance of the sites and was run on the following day, the 13th of April 2023 and was completed in about four hours and 48 minutes, taking an average of 36 minutes per environment. After the completion of all of the performance tests, the test results were then outputted as a CLI output (output of the command) and a HTML report, from which the most important metric data were chosen for this thesis, while the unused metrics were uploaded to the author's GitHub repository.

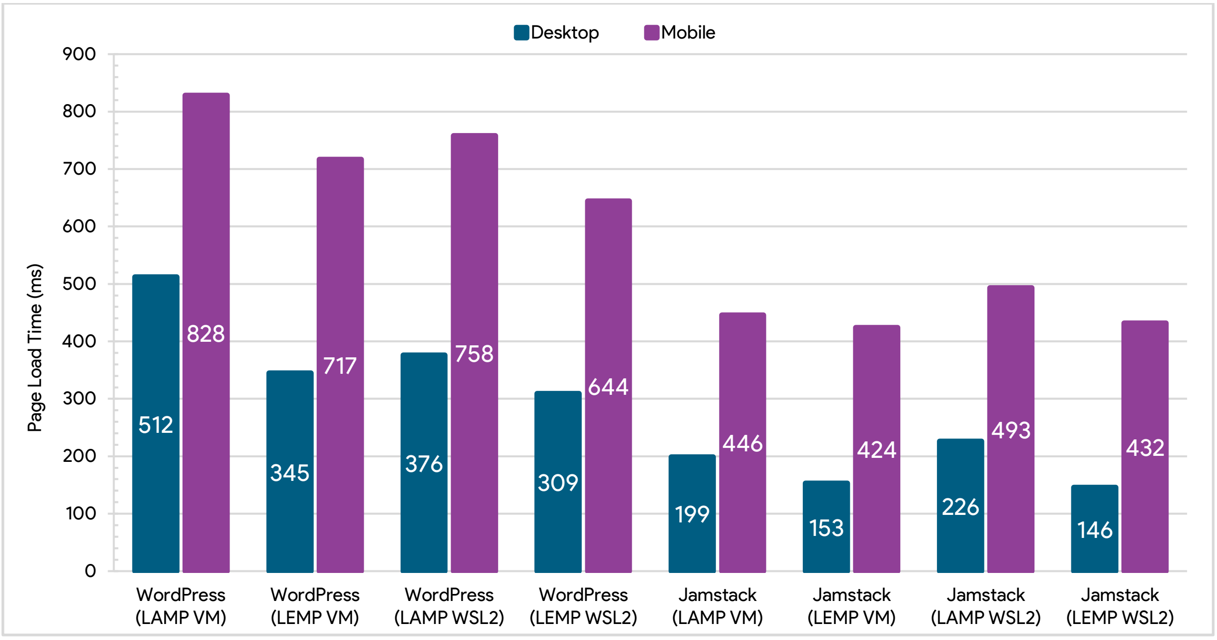
5.1.1 Page Load Time Results

After analysing Figure 11, which showed the Page Load Time results of both the desktop and mobile WordPress and Jamstack sites, it was clearly evident that the Jamstack (LEMP WSL2) site had the quickest performance out of all the desktop tests, with a median value of 146 milliseconds. Meanwhile, the Jamstack (LEMP VM) site had the fastest performance out of all the mobile tests, with a median value of 424 milliseconds, which was 278 milliseconds (2.90 times) slower compared to the desktop site. Both these Jamstack environments (LEMP WSL2 and LEMP VM) had similar performance when calculating their desktop and mobile site tests together, with only a single millisecond difference between the two. In contrast, the WordPress (LAMP VM) site was the slowest among both the desktop and mobile tests, with a median value of 512 milliseconds on the desktop site. Whereas the mobile site had a median value of 828 milliseconds, which was 316 milliseconds (1.62 times) slower compared to the desktop site.

Furthermore, when calculating the median value of the Page Load Time for all WordPress desktop sites using the median formula in Equation 2, located in Appendix A, the result was 360.50 milliseconds. Whereas for Jamstack desktop sites the median value was 176.00 milliseconds, which meant that desktop Jamstack sites were 184.50 milliseconds (2.05 times) quicker than WordPress desktop sites. Similarly, when calculating the median Page Load Time for all the WordPress mobile sites, the value was 737.50 milliseconds, whereas for Jamstack mobile sites it was 439.00 milliseconds, meaning that Jamstack sites were 298.50 milliseconds (1.68 times) quicker than WordPress mobile sites.

Upon further analysis of Figure 11, it became clear that both the desktop and mobile tests followed a similar pattern where the web server, NGINX, performed the quickest compared to Apache. In contrast, the Page Load Times on the desktop sites were faster than those on mobile sites. Interestingly enough, WordPress on WSL2 was the quickest compared to a VM, both for desktop and mobile sites. However, this was quite the opposite when compared to Jamstack, where the Jamstack VMs actually performed quicker than WSL2s, again both for the desktop and mobile sites.

Figure 11. Sitespeed.io – Page Load Time (Desktop and Mobile) [Median] (Whitley, 2023a)



Note. The following results were generated by Sitespeed.io. Lower is better. Page Load Time (PLT) refers to "the time it takes to load a page, from initiation of the page view to the load completion in the browser". (Sitespeed.io, 2022a) The test consisted of 100 iterations, meaning the page was tested 100 times for each environment.

5.1.2 Timing Metrics

Before proceeding to examine Figure 12, as well as Figure 13, it is important to understand how to read and interpret the following figures in question. Each test environment in the figures has a set of two chart types: *Scatter* (a marker with a horizontal line) and *Clustered Column* (a vertical bar with either a lighter grey or darker grey colour). Each scatter marker represents a timing metric, either it being Time to First Byte (TTFB), First Contentful Paint (FCP), Last Visual Change (Page Load Time on the mobile figure), or Fully Loaded. The vertical bar, on the other hand, represents the different phases in the loading of a page. The first lighter grey bar measures "the amount of time between the moment a user requests a web page to the moment the first byte of the response arrives" (Wagner, 2017, p. 27), while the darker grey bar represents the area between TTFB and Fully Loaded. Below in Table 12 is a brief description of each timing and visual metric used in Figure 12 and Figure 13 according to Sitespeed.io's metric documentation.

Table 12. Sitespeed.io's Metric Documentation (Sitespeed.io, 2022a)

Metric	Description
Time to First Byte (TTFB)	Measures "the time it takes for the network and the server to generate and send the HTML. It was collected using the Navigation Timing API". (Sitespeed.io, 2022a)
First Contentful Paint (FCP)	Measures "the time from navigation to when the browser renders the first bit of content from the DOM (Document Object Model)". (Sitespeed.io, 2022a)
Last Visual Change (Visually Complete)	Measures "the time when something changes for the last time within the viewport". (Sitespeed.io, 2022a)
Page Load Time [Only in Figure 13]	Measures "the time it takes for a page to load from initiation of the page view to load completion in the browser". (Sitespeed.io, 2022a)
Fully Loaded	Measures "the time when all of the assets on the page are downloaded. The value comes from the latest response in the HAR (HTTP Archive) file". (Sitespeed.io, 2022a)

Note. Adapted from "*Metrics Collected by Sitespeed.io*" (https://www.sitespeed.io/documentation/sitespeed.io/metrics/)

After analysing the *Fully Loaded* timing metric of the desktop results in Figure 12, it is still evident that the Jamstack (LEMP VM) site performed the quickest, while the WordPress (LAMP VM) site performed the slowest. This is also apparent in the mobile tests shown in Figure 13, were the Jamstack (LEMP VM) site performed the quickest, albeit only by a millisecond compared to Jamstack (LEMP WSL2), while the WordPress (LAMP VM) site once again performed the slowest. The slow performance that is seen with WordPress sites in both Figure 12 and Figure 13 is caused by the long *Time to First Byte* (TTFB), which is likely caused by a slow server response time due to the fact that monolithic sites have to always generate the HTML on the server for each request, while Jamstack sites are pre-rendered, meaning the server only has to serve those static files. (Falconer, 2022)

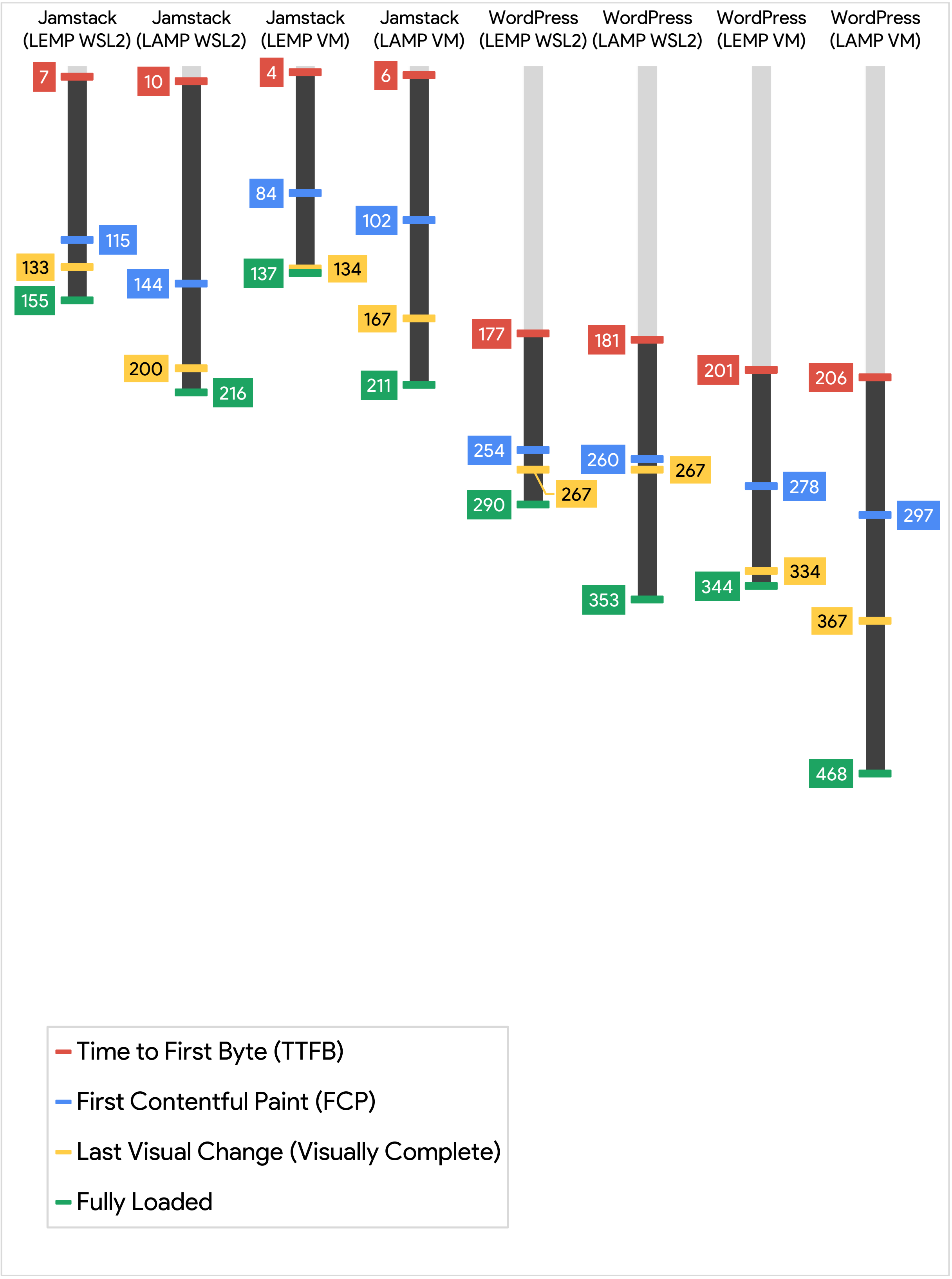
When calculating the *Time to First Byte* metric of all the WordPress and Jamstack desktop sites, the median value for all WordPress desktop sites was 191 milliseconds, while for Jamstack desktop sites it was only 6.5 milliseconds, meaning that Jamstack desktop sites were significantly faster compared to WordPress desktops sites with a difference of 184.50

milliseconds (29.38 times). Similarly for mobile sites, the median value for all WordPress mobile sites was 249 milliseconds, while for Jamstack mobile sites it was only 37.50 milliseconds, meaning that Jamstack mobile sites were 211.50 milliseconds (6.64 times) faster compared to WordPress mobile sites, which is significantly less compared to the difference between WordPress and Jamstack desktop sites.

Upon closer analysis of Figure 12, it is apparent that there are some interesting results when examining the time between the TTFB and Fully Loaded metrics of all WordPress and Jamstack desktop sites. When calculated together, the median value for all WordPress sites was 157.50 milliseconds, while Jamstack sites were slightly slower with a median value of 175.50 milliseconds. One possible explanation for this can be found in the waterfall chart for Jamstack (LEMP WSL2) and WordPress (LEMP WSL2) in Appendix C. When comparing these waterfall figures, which are generated from Sitespeed.io’s HAR (HTTP Archive format) comparison website, it is apparent that WordPress loads several of its website contents, such as its CSS, JavaScript, and font files, faster compared to Jamstack by a margin of five to almost 30 milliseconds per content request. However, Jamstack can quickly recoup for this performance loss with its faster TTFB load time.

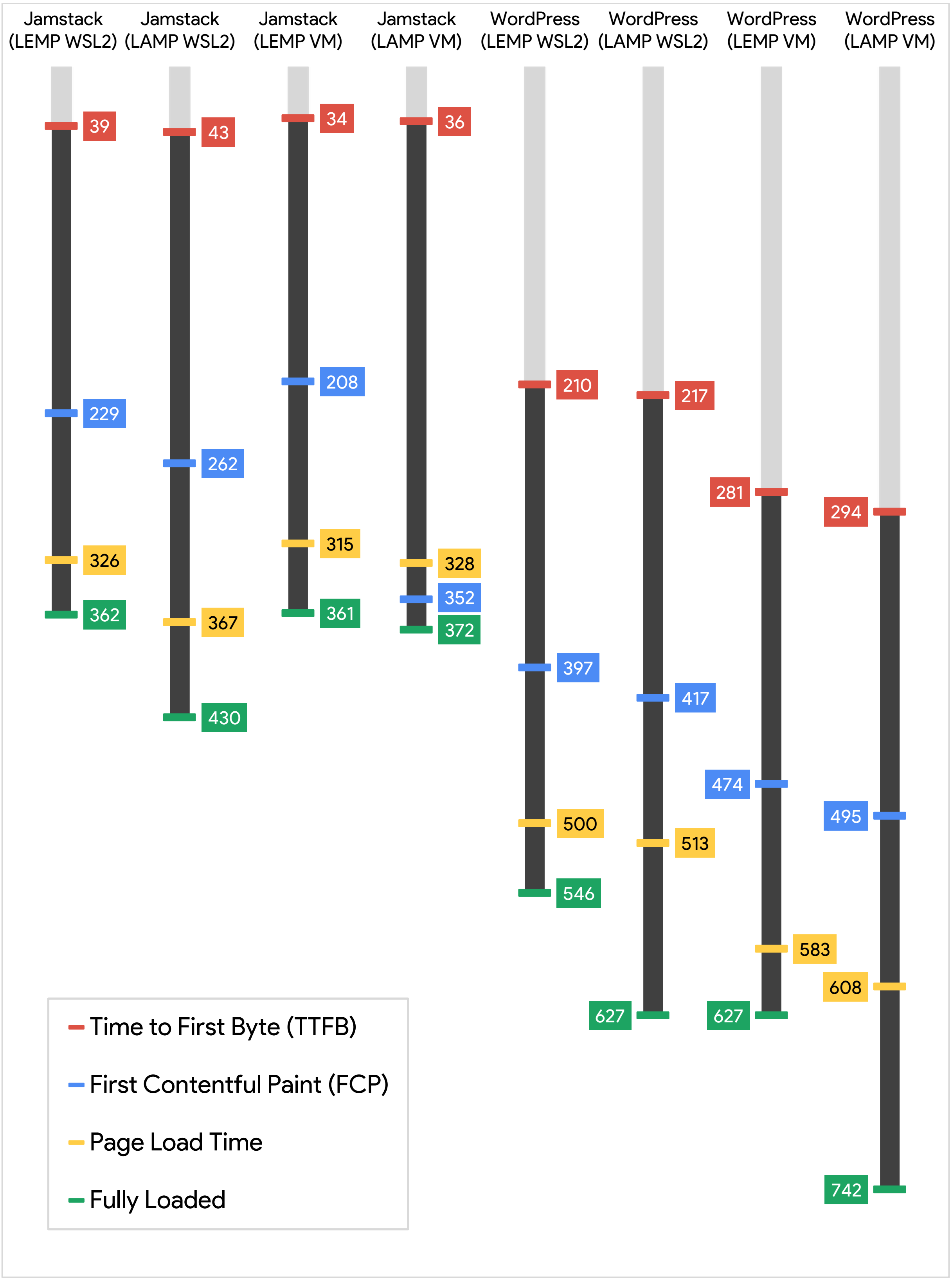
Furthermore, the next figure, Figure 13, shows a somewhat similar situation to that of Figure 12 when examining the time between TTFB and the Fully Loaded metrics of all WordPress and Jamstack mobile sites. Although Jamstack mobile sites were slightly faster than WordPress mobile sites, with a median value of 331.50 milliseconds compared to WordPress’ median value of 378.00 milliseconds, the waterfall figures in Appendix C still showed a familiar pattern to the desktop waterfall figures where WordPress was still able to load most of its website’s contents faster than Jamstack, again by a margin of five to 30 milliseconds.

Figure 12. Sitespeed.io – Timing Metrics (Desktop) [Median, p50] (Whitley, 2023a)



Note. The metric results were generated by Sitespeed.io. The unit of time used in Figure 12 is in milliseconds (ms). Lower is better. The metric results are in median (p50). Additional data is available from the author's GitHub repository (https://github.com/Sam-Whitley/thesis).

Figure 13. Sitespeed.io – Timing Metrics (Mobile) [Median, p50] (Whitley, 2023a)



Note. The metric results were generated by Sitespeed.io. The unit of time used in Figure 13 are in milliseconds (ms). Lower is better. Metric results are in median (p50). Additional data is available from the author's GitHub repository (https://github.com/Sam-Whitley/thesis).

5.2 Scalability Results

After the performance tests, the scalability of both web architectures was tested using the load testing tool, Grafana k6, with various types of load tests. Typically, these load tests simulated the different levels of traffic to determine how each web architecture handled different loads at different durations. Depending on the specific load test in question, most involved ramping up the number of virtual users (VUs) over a certain period of time, maintaining that specific number of VUs, before then ramping down the number of VUs back to zero.

Like Sitespeed.io, the load tests were run through a command-line interface (CLI), but instead of using Git Bash, the command was executed through the PowerShell. These load tests were then run using a command with three distinct options (also known as flags); *'IP_ADDRESS'*, *'ENVNAME'*, and *'CSV=fileName'*, which allowed for the options to be changed through the input rather than having to modify the script itself after each load test, as shown in Figure 14 below.

Figure 14. Example Grafana k6 Command (Whitley, 2023)

```
k6 run -e IP_ADDRESS=192.168.100.157 -e ENVNAME=L1
-o csv=fileName=csv/L1-smokeTest.csv smokeTest.js
```

Note. The tool shown in Figure 14 is a command used to run a smoke test. Created by Grafana Labs (https://k6.io/docs/get-started/running-k6/).

These load tests were conducted over a two-day period, similar to the web performance tests. However, since the VUs combine both desktop and mobile users, there was no need to separate them into different categories. This allowed for the scalability of the sites to be tested using only a single category of users, VUs, or better known as Virtual Users. VUs are essentially execution agents that load the script and its contents and are responsible for running it, practically emulating a user performing the script. (Crevon, 2022)

The tests were divided into different load test categories based on their type. These load tests were smoke testing, average-load testing, stress testing, and spike testing. The first set of load tests, consisted of smoke, average-load, and spike tests, which were all conducted on the 25th of April 2023, while the stress tests were conducted on the following day, the 26th of April 2023. In addition, the number of virtual users and the duration of each load test may vary depending on the type of test. Further details regarding the load test script descriptions and configurations for each test are found in Table 9 and Table 11, while the load test scripts are located in Appendix E.

Before proceeding to the following chapters, it is important to note that only the most important metrics and data have been chosen. However, only the Success Rate figures are covered, since there are too many figures and tables to cover. Moreover, additional data of these are available in appendices F through I, as well as in the GitHub repository.

It is also important to mention that the "Content type is text/HTML" check has been excluded from the load test Success Rate charts (e.g., in Figure 15) and from the Check tables (e.g., in Table 15). This is because it had a success rate of 100% for the most part, which greatly affected the overall success rate score. This can be seen in Appendix I, specifically in the "Spike Test L8" section, where the "Status is 200" and "Verify Homepage Text" checks both showed a success rate of 5.19%. However, with the "Content type is text/HTML" check being 100%, the overall success rate was significantly skewed, resulting in an overall success rate of 36.79%, not 5.19%, similarly to the other two checks.

5.2.1 Smoke Test

Smoke tests are typically performed to check and verify that a test script does not contain any errors or to establish a baseline for the performance metrics of a system’s response under minimal load. In this case, the smoke tests were used to check and verify that the environments (as well as the sites) did not produce any errors when under minimal load. This was always done before running any of the other primary load testing scripts. (Grafana k6, 2023d)

When analysing the Success Rate figure shown in Figure 15, it was evident that all the environments passed the smoke test, resulting in a total success rate of 100% for each environment. However, a closer analysis of the success count revealed that all the Jamstack environments had a combined median value of 360 checks, whereas WordPress only had 306 checks, meaning that Jamstack had 54 (or 1.17 times) more checks than WordPress. Interestingly enough, when comparing the WordPress environments' virtualisation groups and web servers to each other, WordPress (LAMP VM) had 306 checks, six more than WordPress (LEMP VM) with 200 checks. However, this was quite the opposite on WSL2, where WordPress (LEMP WSL2) had 312 checks, again six more than WordPress (LAMP WSL2), with 306 checks.

Table 13. Smoke Test [Requests] (Whitley, 2023a)

Environments	req_duration (Median)	req_waiting (Median)	req_receiving (Median)	iteration_duration (Median)
WP (LAMP VM)	180.26 ms	168.78 ms	8.83 ms	1.18 s
WP (LEMP VM)	207.46 ms	192.01 ms	9.69 ms	1.20 s
WP (LAMP WSL2)	166.39 ms	165.39 ms	1.03 ms	1.17 s
WP (LEMP WSL2)	162.00 ms	160.89 ms	1.05 ms	1.16 s
JS (LAMP VM)	8.66 ms	537.84 µs	7.99 ms	1.00 s
JS (LEMP VM)	8.86 ms	540.15 µs	8.15 ms	1.00 s
JS (LAMP WSL2)	6.41 ms	4.07 ms	2.11 ms	1.01 s
JS (LEMP WSL2)	4.03 ms	2.86 ms	1.06 ms	1.00 s

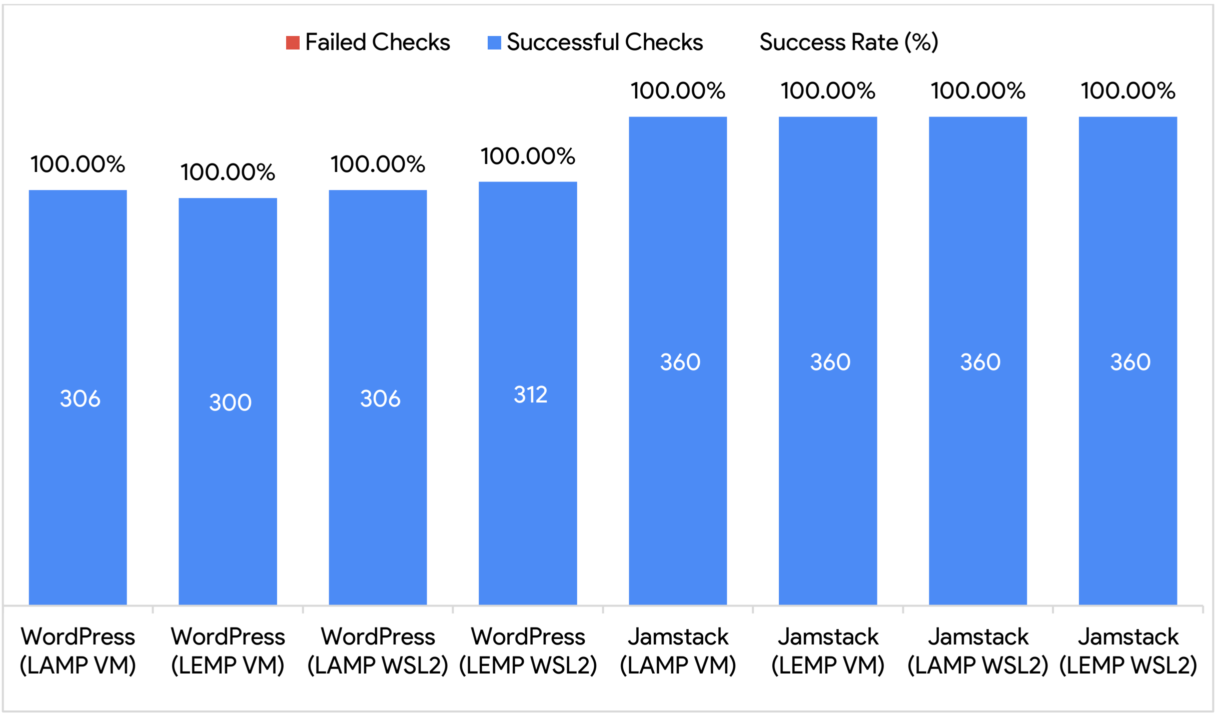
Note. The metric results in the table are in median (p50) and some are in different units of time, such as in seconds (s), milliseconds (ms), microseconds (µs), and sometimes even in nanoseconds (ns). Additional data is available in the author’s GitHub repository (https://github.com/Sam-Whitley/thesis).

Table 14. Smoke Test [Other Stats] (Whitley, 2023a)

Environments	Requests (Total)	Requests (Rate, /s)	Data Sent/Received (Total)	Data Sent/Received (Rate, /s)
WP (LAMP VM)	153	2.51/s	12 kB / 18 MB	204 B / 297 kB
WP (LEMP VM)	150	2.46/s	12 kB / 18 MB	199 B / 291 kB
WP (LAMP WSL2)	153	2.55/s	13 kB / 18 MB	219 B / 302 kB
WP (LEMP WSL2)	156	2.56/s	13 kB / 19 MB	220 B / 304 kB
JS (LAMP VM)	180	2.97/s	15 kB / 21 MB	240 B / 247 kB
JS (LEMP VM)	180	2.97/s	15 kB / 21 MB	231 B / 247 kB
JS (LAMP WSL2)	180	2.96/s	16 kB / 21 MB	255 B / 346 kB
JS (LEMP WSL2)	180	2.97/s	16 kB / 21 MB	256 B / 3438 kB

Note. The ”/s” in the table is an abbreviation for ”per second (s)”. Additional data is available in the author’s GitHub repository (https://github.com/Sam-Whitley/thesis).

Figure 15. Smoke Test [Success Rate] (Whitley, 2023a)



Note. The data label in the middle of the column bar represents a numerical value calculated by combining the results of both the *"Status 200 (OK)"* and *"Verify Homepage"* checks.

Table 15. Smoke Test [Checks] (Whitley, 2023a)

Environments	Status 200 (OK) Successful Checks	Status 200 (OK) Failed Checks	Verify Homepage Successful Checks	Verify Homepage Failed Checks	Success Rate (%)
WP (LAMP VM)	153	0	153	0	100.00 %
WP (LEMP VM)	150	0	150	0	100.00 %
WP (LAMP WSL2)	153	0	153	0	100.00 %
WP (LEMP WSL2)	156	0	156	0	100.00 %
JS (LAMP VM)	180	0	180	0	100.00 %
JS (LEMP VM)	180	0	180	0	100.00 %
JS (LAMP WSL2)	180	0	180	0	100.00 %
JS (LEMP WSL2)	180	0	180	0	100.00 %

Note. This table does not include the "*Content type is text/HTML*" check, as including it would significantly affect the overall success rate. This check is only available in the Check tables in Appendix F. Additional data is available in the author's GitHub repository (https://github.com/Sam-Whitley/thesis).

5.2.2 Average-Load Test

Typically, average-load tests are used to test and identify early signs of degradation during a ramp-up or full load period. They can also be used to ensure that the system still meets the performance standards after a system change (code and infrastructure). In this case, the average-load test was used to assess the performance of the environment and site under typical load conditions. The name “average-load test” is used by Grafana k6 to avoid confusion since a “load test” might refer to all types of tests that simulate traffic. In some testing conversations, the average-load test might also be called a “day-in-a-life” or volume test. (Grafana k6, 2023e)

After analysing the Success Rate figure shown in Figure 16, it was apparent that even after the smoke test, all the environments still passed all of the checks with flying colours with a total success rate of 100%. Out of all the Jamstack environments, Jamstack (LEMP VM) scored the highest number of checks, with a check count of 416 506. While Jamstack (LAMP WSL2) and Jamstack (LEMP WSL2) achieved similar check counts, scoring 410 340 and 413 688 checks respectively. The Jamstack environment with the lowest number of checks was the Jamstack (LAMP VM) environment, falling slightly behind the other Jamstack environments with a score of 323 580 checks, which was 92 926 (or 1.29 times) less than Jamstack (LEMP VM).

Moreover, when comparing both WordPress and Jamstack environments, the WordPress environments scored considerably less, with a median check count of 141 512, which was 270 502 (or 2.91 times) less than the median check count of 412 014 of the Jamstack environments. However, there was one WordPress environment that stood out from the rest. This environment was WordPress (LAMP WSL2), which had a pass count of 335 052, which was only 11 472 (or 1.04 times) more than the lowest performing Jamstack environment, Jamstack (LAMP VM). It is unclear why WordPress (LAMP WSL2) outperformed WordPress (LEMP WSL2) by such a large margin, despite the fact that WordPress (LEMP WSL2) should have had the edge with its slightly newer and better-performing web server, NGINX.

To verify the validity of this result, the WordPress (LAMP WSL2) environment was checked to ensure that it had the same packages, services, and WordPress settings and versions as the other WordPress environments. After this check, the same average-load test was rerun, which ultimately resulted in the same outcome. Afterwards, a new temporary environment was created to run several load tests on it. However, this temporary environment produced the same consistent results as WordPress (LAMP WSL2), which validated the accuracy of the environment's results. Subsequently, this issue was later researched, and an article from Hackr.io was discovered that revealed that there should not be a significant performance difference between the two web servers, Apache and NGINX, if the content type is dynamic. According to the benchmark mentioned in the article, done by Speedemy, both NGINX and Apache performed similarly when it came to displaying dynamic content since almost all of the request processing time is spent in the PHP runtime environment rather than on the core part of the web server. (Krishnan, 2022)

Ultimately, after much investigation there was no clear answer as to why WordPress (LAMP WSL2) performed better than WordPress (LEMP WSL2). As a result, the environment results for WordPress (LAMP WSL2) could be deemed invalid and left out based on how the other three WordPress environments performed.

Table 16. Average-Load Test [Requests] (Whitley, 2023a)

Environments	req_duration (Median)	req_waiting (Median)	req_receiving (Median)	iteration_duration (Median)
WP (LAMP VM)	2.26 s	2.23 s	14.20 ms	3.26 s
WP (LEMP VM)	2.24 s	2.20 s	19.40 ms	3.24 s
WP (LAMP WSL2)	255.10 ms	253.62 ms	1.56 ms	1.25 s
WP (LEMP WSL2)	2.35 s	2.35 s	1.08 ms	3.35 s
JS (LAMP VM)	272.14 ms	4.56 ms	255.07 ms	1.27 s
JS (LEMP VM)	7.59 ms	549.10 µs	6.79 ms	1.00 s
JS (LAMP WSL2)	17.56 ms	8.58 ms	8.54 ms	1.02 s
JS (LEMP WSL2)	12.76 ms	7.51 ms	3.81 ms	1.01 s

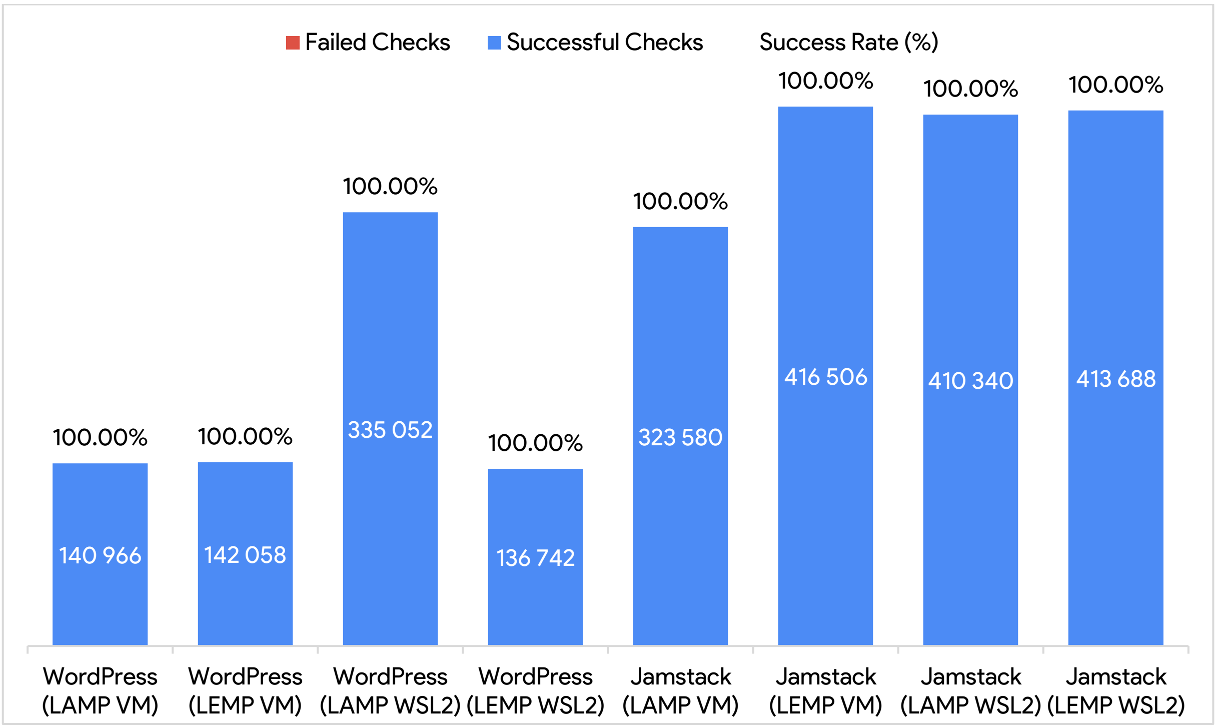
Note. The metric results in the table are in median (p50) and some are in different units of time, such as in seconds (s), milliseconds (ms), microseconds (µs), and sometimes even in nanoseconds (ns). Additional data is available in the author’s GitHub repository (https://github.com/Sam-Whitley/thesis).

Table 17. Average-Load Test [Other Stats] (Whitley, 2023a)

Environments	Requests (Total)	Requests (Rate, /s)	Data Sent/Received (Total)	Data Sent/Received (Rate, /s)
WP (LAMP VM)	70 483	29.37/s	5.7 MB / 8.3 GB	2.4 kB / 3.5 MB
WP (LEMP VM)	71 029	29.59/s	5.8 MB / 8.4 GB	2.4 kB / 3.5 MB
WP (LAMP WSL2)	167 526	69.77/s	14 MB / 20 GB	6.0 kB / 8.3 MB
WP (LEMP WSL2)	68 371	28.49/s	5.9 MB / 8.1 GB	2.4 kB / 3.4 MB
JS (LAMP VM)	161 790	67.40/s	13 MB / 19 GB	5.5 kB / 7.9 MB
JS (LEMP VM)	208 253	86.74/s	17 MB / 24 GB	7.0 kB / 10 MB
JS (LAMP WSL2)	205 170	85.47/s	18 MB / 24 GB	7.4 kB / 10 MB
JS (LEMP WSL2)	206 844	86.16/s	18 MB / 24 GB	7.4 kB / 10 MB

Note. The ”/s” in the table is an abbreviation for ”per second (s)”. Additional data is available in the author’s GitHub repository (https://github.com/Sam-Whitley/thesis).

Figure 16. Average-Load Test [Success Rate] (Whitley, 2023a)



Note. The data label in the middle of the column bar represents a numerical value calculated by combining the results of both the *"Status 200 (OK)"* and *"Verify Homepage"* checks.

Table 18. Average-Load Test [Checks] (Whitley, 2023a)

Environments	Status 200 (OK) Successful Checks	Status 200 (OK) Failed Checks	Verify Homepage Successful Checks	Verify Homepage Failed Checks	Success Rate (%)
WP (LAMP VM)	70 483	0	70 483	0	100.00 %
WP (LEMP VM)	71 029	0	71 029	0	100.00 %
WP (LAMP WSL2)	167 526	0	167 526	0	100.00 %
WP (LEMP WSL2)	68 371	0	68 371	0	100.00 %
JS (LAMP VM)	161 790	0	161 790	0	100.00 %
JS (LEMP VM)	208 253	0	208 253	0	100.00 %
JS (LAMP WSL2)	205 170	0	205 170	0	100.00 %
JS (LEMP WSL2)	206 844	0	206 844	0	100.00 %

Note. This table does not include the "*Content type is text/HTML*" check, as including it would significantly affect the overall Success Rate. This check is only available in the Check tables in Appendix G.

5.2.3 Stress Test

The stress test operates similarly to an average-load test, with the main difference being the higher load. Although the load pattern of the stress test slightly resembles that of an average-load test, the stress test has a longer ramp-up period to proportionally account for the higher load. Similarly, when the test reaches the designated load level, it must remain at that level for a slightly longer duration compared to an average-load test. This type of load test was used to test and verify the scalability and reliability of the environments and sites under heavier conditions with twice as many virtual users (VUs) compared to the previous test. (Grafana k6, 2023f)

After analysing the next Success Rate figure, Figure 17, it is again apparent that the figure followed a similar pattern to that of the earlier test, the average-load test, with a few notable differences. For instance, this was the first load test in which several checks failed, specifically in the first three WordPress environments. WordPress (LAMP VM) had 158 012 successful checks and 69 456 failed checks, while WordPress (LEMP VM) had 161 556 successful checks and 23 398 failed checks. The WordPress (LAMP WSL2) environment, on the other hand, was the only environment out of the WordPress environments to have no failed checks, but it had the lowest number of successful checks of the WordPress environments with 156 676 successful checks.

Moreover, when turning our attention to the Jamstack environments, it was apparent that not a single environment encountered a failed check, where each achieved a total success rate of 100%. This is in contrast to the WordPress environments, which achieved a median success rate of only 93.50%, with the lowest-scoring environment receiving a total of 69.47%. Upon closer analysis of Figure 17, the Jamstack environment, Jamstack (LAMP VM), had the fewest number of successful checks, with only 482 822, compared to Jamstack (LEMP VM), which scored the highest, with 877 680 successful checks.

Table 19. Stress Test [Requests] (Whitley, 2023a)

Environments	req_duration (Median)	req_waiting (Median)	req_receiving (Median)	iteration_duration (Median)
WP (LAMP VM)	4.03 s	3.98 s	7.99 ms	5.03 s
WP (LEMP VM)	4.67 s	4.59 s	19.74 ms	5.68 s
WP (LAMP WSL2)	1.29 s	1.28 s	1.61 ms	2.29 s
WP (LEMP WSL2)	5.68 s	5.68 s	1.11 ms	6.68 s
JS (LAMP VM)	1.17 s	70.68 ms	1.08 ms	2.18 s
JS (LEMP VM)	562.67 ms	6.95 ms	537.86 ms	1.56 s
JS (LAMP WSL2)	19.69 ms	9.54 ms	9.57 ms	1.02 s
JS (LEMP WSL2)	13.94 ms	8.09 ms	4.79 ms	1.01 s

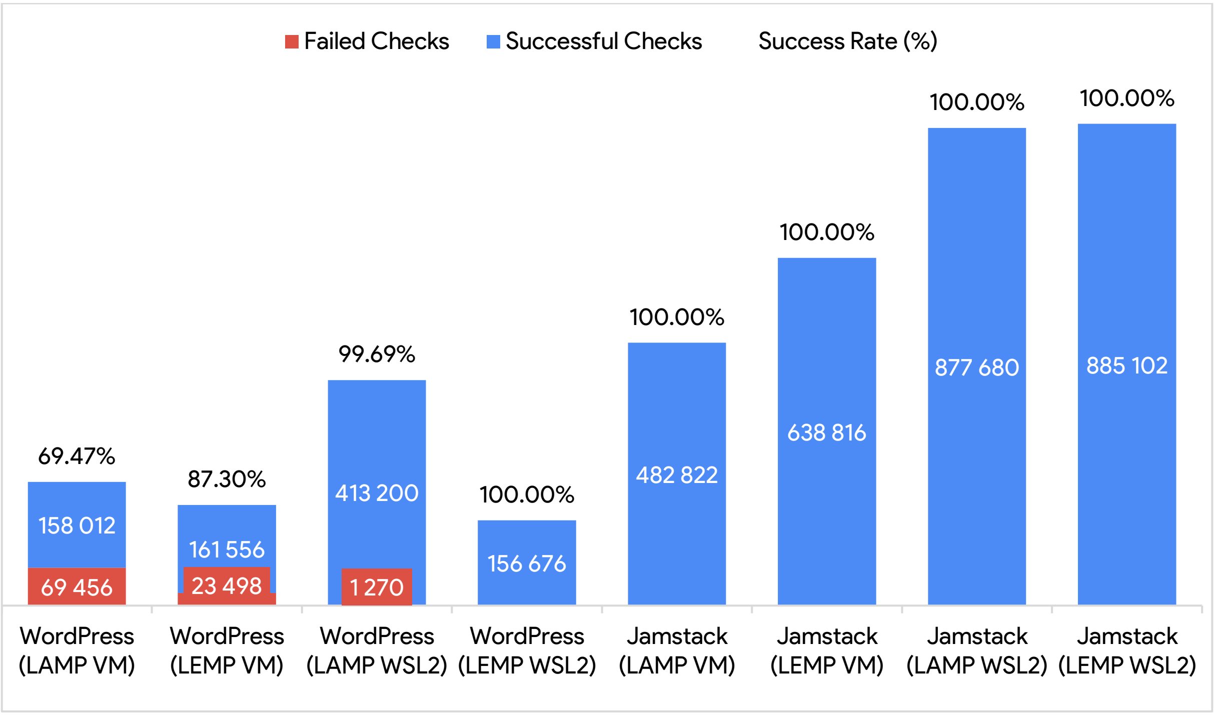
Note. The metric results in the table are in median (p50) and some are in different units of time, such as in seconds (s), milliseconds (ms), microseconds (µs), and sometimes even in nanoseconds (ns). Additional data is available in the author's GitHub repository (https://github.com/Sam-Whitley/thesis).

Table 20. Stress Test [Other Stats] (Whitley, 2023a)

Environments	Requests (Total)	Requests (Rate, /s)	Data Sent/Received (Total)	Data Sent/Received (Rate, /s)
WP (LAMP VM)	113 734	42.11/s	9.2 MB / 9.4 GB	3.4 kB / 3.5 MB
WP (LEMP VM)	92 527	34.26/s	7.5 MB / 9.6 GB	2.8 kB / 3.5 MB
WP (LAMP WSL2)	207 235	76.74/s	18 MB / 25 GB	6.6 kB / 9.1 MB
WP (LEMP WSL2)	78 338	29.01/s	6.7 MB / 9.3 GB	2.5 kB / 3.4 MB
JS (LAMP VM)	241 411	89.39/s	20 MB / 28 GB	7.2 kB / 11 MB
JS (LEMP VM)	319 408	118.26/s	26 MB / 37 GB	9.6 kB / 14 MB
JS (LAMP WSL2)	438 840	162.50/s	38 MB / 51 GB	14 kB / 19 MB
JS (LEMP WSL2)	442 551	163.88/s	38 MB / 52 GB	14 kB / 19 MB

Note. The ”/s” in the table is an abbreviation for ”per second (s)”. Additional data is available in the author's GitHub repository (https://github.com/Sam-Whitley/thesis).

Figure 17. Stress Test [Success Rate] (Whitley, 2023a)



Note. The data labels in the middle of the column bar represents a numerical value calculated by combining the results of both the *"Status 200 (OK)"* and *"Verify Homepage"* checks.

Table 21. Stress Test [Checks] (Whitley, 2023a)

Environments	Status 200 (OK) Successful Checks	Status 200 (OK) Failed Checks	Verify Homepage Successful Checks	Verify Homepage Successful Checks	Success Rate (%)
WP (LAMP VM)	79 006	34 728	79 006	34 728	69.47 %
WP (LEMP VM)	80 778	11 749	80 778	11 749	87.30 %
WP (LAMP WSL2)	206 600	635	206 600	635	99.69 %
WP (LEMP WSL2)	78 338	0	78 338	0	100.00 %
JS (LAMP VM)	241 411	0	241 411	0	100.00 %
JS (LEMP VM)	319 408	0	319 408	0	100.00 %
JS (LAMP WSL2)	438 840	0	438 840	0	100.00 %
JS (LEMP WSL2)	442 551	0	442 551	0	100.00 %

Note. This table does not include the "*Content type is text/HTML*" check, as including it would significantly affect the overall Success Rate. This check is only available in the Check tables in Appendix H.

5.2.4 Spike Test

Finally, the spike tests were used to test and identify whether the environments and sites could withstand a sudden and massive rush of traffic. This type of test is commonly used to simulate events, such as a product launch (e.g., a PlayStation 5 launch) or seasonal sales (e.g., a Black Friday or Christmas sale). In this case, the tests were used to put both web architectures to the ultimate test and evaluate how they performed under extremely high traffic loads. (Grafana k6, 2023g) Compared to earlier tests, the spike test used exactly 10 times more virtual users (VUs) than the stress test or 20 times more VUs than the average-load test.

When analysing the fourth and final Success Rate figure, Figure 18, significant changes were observed when compared to previous figures. The figure showed a staircase-like pattern, with the leftmost environments performing the fewest checks, while the rightmost performed the most checks, which was not observed in previous figures. Additionally, this was also the first chart where both WordPress and Jamstack architectures had failing checks.

By comparing both web architectures together, it became clear which environment performed better in handling potential traffic spikes. In the spike test, the Jamstack environments held up extremely well, despite the large number of VUs. Among the Jamstack environments, the best performing environment was Jamstack (LEMP WSL2), which had a total of 185 330 successful checks and only 1 336 failed checks, resulting in an impressive total success rate of 99.28%. On the other hand, the Jamstack (LAMP VM) environment had the fewest number of checks, with 36 792 successful checks and 927 failed checks, but still maintained a relatively high total success rate of 97.54%.

When analysing the WordPress environment in the same figure, Figure 18, it was clearly evident that the majority of them had a significant number of failures, with a median total success rate of only 14.25%, compared to 98.30% of all Jamstack environments. When comparing the best performing WordPress environment, WordPress (LAMP WSL2), which had 22 306 successful checks and 107 342 failed checks, to the best Jamstack environment, Jamstack (LEMP WSL2), it showed that Jamstack had 163 024 (or 8.31 %) more successful checks than WordPress, highlighting the significant difference in the scalability between the two architectures under extremely high traffic loads.

Although technically WordPress (LEMP WSL2) produced the most checks overall out of all the environments, most of them were failed checks. The reason why WordPress (LEMP WSL2) might have produced more failed checks than the total check count of Jamstack (LEMP WSL2) can probably be seen in the log files of the environment. Upon analysing the log files, it was found that the environment was hit with numerous "Resource temporarily unavailable" errors, also known as a "502 Bad Gateway" error. This error might be caused by the PHP-FPM (FastCGI Process Manager) being unable to process the huge amounts of requests, according to an article by Xiao Guoan. By default, the Linux kernel's “net.core.somaxconn” setting, which defines the maximum number of connections allowed to a socket file, has a value of 4096, meaning the maximum number of users. Before kernel 5.4, this value was 128. Therefore, the Linux kernel might not be to blame in this regard, as the maximum number of users limit was well above the VUs limit of the spike test. However, increasing the number of child processes in the PHP-FPM may potentially slightly alleviate this issue. (Guoan, 2022)

Table 22. Spike Test [Requests] (Whitley, 2023a)

Environments	req_duration (Median)	req_waiting (Median)	req_receiving (Median)	iteration_duration (Median)
WP (LAMP VM)	7.91 s	7.90 s	0 s	8.91 s
WP (LEMP VM)	1.26 s	1.26 s	0 s	2.26 s
WP (LAMP WSL2)	0 s	148.31 ms	505.40 µs	2.44 s
WP (LEMP WSL2)	1.25 ms	1.18 ms	0 s	1.00 s
JS (LAMP VM)	6.15 s	163.60 ms	5.88 s	7.58 s
JS (LEMP VM)	3.97 s	106.83 ms	3.82 s	5.02 s
JS (LAMP WSL2)	21.22 ms	10.09 ms	8.81 ms	1.02 s
JS (LEMP WSL2)	27.08 ms	14.43 ms	10.59 ms	1.02 s

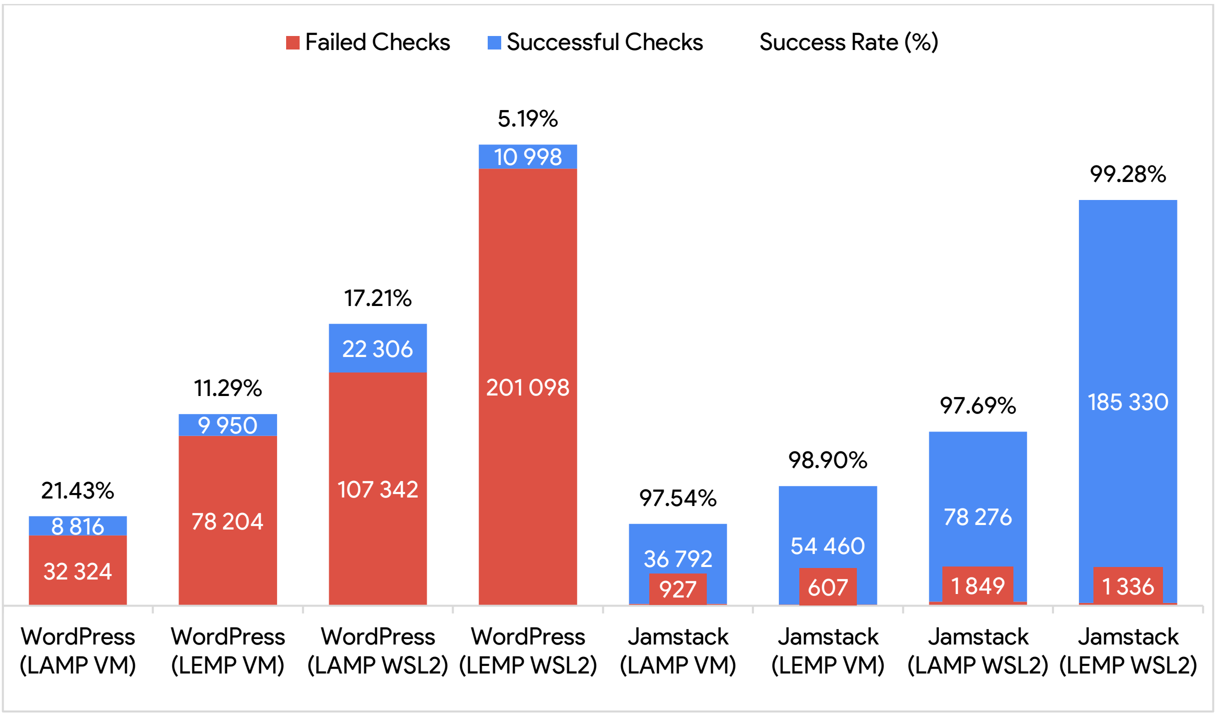
Note. The metric results in the table are in median (p50) and some are in different units of time, such as in seconds (s), milliseconds (ms), microseconds (µs), and sometimes even in nanoseconds (ns). Additional data is available in the author's GitHub repository (https://github.com/Sam-Whitley/thesis).

Table 23. Spike Test [Other Stats] (Whitley, 2023a)

Environments	Requests (Total)	Requests (Rate, /s)	Data Sent/Received (Total)	Data Sent/Received (Rate, /s)
WP (LAMP VM)	20 570	112.88/s	1.7 MB / 566 MB	9.2 kB / 3.1 MB
WP (LEMP VM)	44 077	243.47/s	3.6 MB / 618 MB	20 kB / 2.4 MB
WP (LAMP WSL2)	73 413	403.52/s	4.8 MB / 1.4 GB	27 kB / 7.9 MB
WP (LEMP WSL2)	106 048	554.74/s	9.1 MB / 682 MB	48 kB / 3.6 MB
JS (LAMP VM)	19 323	106.93/s	1.6 MB / 2.2 GB	8.8 kB / 12 MB
JS (LEMP VM)	27 837	153.85/s	2.3 MB / 3.2 GB	13 kB / 18 MB
JS (LAMP WSL2)	40 987	224.89/s	3.6 MB / 4.6 GB	20 kB / 25 MB
JS (LEMP WSL2)	94 001	447.62/s	8.2 MB / 11 GB	39 kB / 52 MB

Note. The ”/s” in the table is an abbreviation for ”per second (s)”. Additional data is available in the author's GitHub repository (https://github.com/Sam-Whitley/thesis).

Figure 18. Spike Test [Success Rate] (Whitley, 2023a)



Note. The data labels in the middle of the column bar represents a numerical value calculated by combining the results of both the *"Status 200 (OK)"* and *"Verify Homepage"* checks.

Table 24. Spike Test [Checks] (Whitley, 2023a)

Environments	Status 200 (OK) Successful Checks	Status 200 (OK) Failed Checks	Verify Homepage Successful Checks	Verify Homepage Failed Checks	Success Rate (%)
WP (LAMP VM)	4 408	16 162	4 408	16 162	21.43 %
WP (LEMP VM)	4 975	39 102	4 975	39 102	11.29 %
WP (LAMP WSL2)	11 153	62 260	11 153	45 082	17.21 %
WP (LEMP WSL2)	5 499	100 549	5 499	100 549	5.19 %
JS (LAMP VM)	18 396	927	18 396	0	97.54 %
JS (LEMP VM)	27 230	607	27 230	0	98.90 %
JS (LAMP WSL2)	39 138	1 849	39 138	0	97.69 %
JS (LEMP WSL2)	92 665	1 336	92 665	0	99.28 %

Note. This table does not include the "*Content type is text/HTML*" check, as including it would significantly affect the overall Success Rate. This check is only available in the Check tables in Appendix I.

6 Conclusion

This research study aimed to gather more quantitative data on the performance and scalability of the Jamstack architecture and compare it to the monolithic architecture of WordPress. More specifically, this research study aimed to answer the following research questions:

Question 1: *How did the performance of the Jamstack web architecture compare to the monolithic web architecture of WordPress in the terms of different user-centric performance metrics?*

Question 2: *How did the scalability of the Jamstack web architecture compare to the monolithic web architecture of WordPress in handling different patterns of traffic?*

To test and evaluate the performance and scalability of the two architectures, a total of eight Linux-based virtualised environments were tested, half of which were created using Jamstack and the other WordPress. These environments were then deployed on different virtualisation technologies, WSL2 and VM, using the different web servers, Apache and NGINX, from the software stacks of LAMP and LEMP, respectively. This was repeated to ensure the reliability as well as the validity of the test results.

Based on the performance results that were gathered and analysed, the Page Load Time (PLT) metric results showed that Jamstack loaded pages 1.68 times faster on mobile, whereas desktop loaded pages 2.05 times faster when compared to WordPress. Moreover, when comparing the Time to First Byte (TTFB) metric, Jamstack had a clear advantage in server response time with a 6.64 time faster TTFB on mobile and a staggering 29.88 time faster TTFB on desktop, which would explain the improved page load time of the Jamstack architecture. However, it is worth noting that when examining the time between TTFB and Fully Loaded metrics, Jamstack loaded some of its website content, such as its CSS, JavaScript, and font files slower compared to WordPress, being 1.11 times slower on desktop and 1.14 times slower on mobile, as clearly evident in the waterfall figures in Appendix C.

Fortunately, Jamstack can quickly recoup for this performance loss with its faster TTFB load time, again both on desktop and mobile sites.

When regarding the scalability results, the research study included various types of quantitative data, such as request metrics and checks. However, due to the large number of figures, tables, and the overall amount of data, only the Success Rate figures were covered. These Success Rate figures showed the number of successful and failed checks of each load test, providing insight into how each architecture performed under different types of traffic loads. Based on the scalability results of the Success Rate figures, the load tests, in this case, the smoke test, showed that Jamstack had 1.17 times the number of successful checks compared to WordPress during minimal load. During an average-load test, Jamstack had 2.91 times the number of successful checks compared to WordPress. As the number of virtual users (VUs) and the test duration increased, Jamstack’s advantaged grew as well, with Jamstack getting 4.75 times the number of successful checks than WordPress during the stress test. Finally, during the most rigorous load test, the spike test, Jamstack had close to 6.64 times more successful checks than WordPress, highlighting the significant difference in the scalability between the two architectures under extremely high traffic loads.

In summary, it can be concluded that the Jamstack architecture provided clear performance benefits when serving its content statically, especially under heavier loads, compared to the monolithic web architecture of WordPress. Furthermore, the research study generated more, potentially new and unresearched quantitative data of both web architectures. This data could be further analysed by examining, for example, the HAR (HTTP Archive) files from the performance tests, which contain different network traffic information about the browser’s interactions with a site. (Seal, 2022) Alternatively, the CSV files from the scalability tests, which contains granular data points (detailed data) of the load tests, could be analysed in greater detail to further validate or potentially uncover any aspects that were missed or overlooked by the author.

Additionally, a question remained on whether the Jamstack architecture would still have the upper hand in performance when utilising application programming interfaces (APIs) and microservices, such as an e-commerce Jamstack website, against the monolithic web architecture of WordPress, which could be a potential area for further research. Another potential area of research could involve comparing the build times of different static site generators (SSGs) against each other, as some are notorious for their slow build time. While there have been publications on this, notably a CSS-Tricks article written by Sean C Davis, the article might be slightly outdated by now.

References

Ahmed, R. (2023, January 25). *Why Biggest Brands in the World Use WordPress and Who They Are? [Why WordPress is So Popular]*. Retrieved May 4, 2023, from weDevs: https://wedevs.com/blog/103311/top-brands-using-wordpress/

Atlassian. (n.d.). *Git Bash*. Retrieved May 8, 2023, from Atlassian: https://www.atlassian.com/git/tutorials/git-bash

Awati, R., & Wigmore, I. (2022, May). *What Is Monolithic Architecture in Software?* Retrieved March 21, 2023, from TechTarget: https://www.techtarget.com/whatis/definition/monolithic-architecture

Bhandari, P. (2023, January 20). *How to Calculate Standard Deviation (Guide) | Calculator & Examples*. Retrieved May 9, 2023, from Scribbr: https://www.scribbr.com/statistics/standard-deviation/

Biilmann, M. (2021, May 3). *The Evolution of Jamstack*. Retrieved March 23, 2023, from Smashing Magazine: https://www.smashingmagazine.com/2021/05/evolution-jamstack/

Biilmann, M., & Hawksworth, P. (2019). *Modern Web Development on the JAMstack* (First Edition ed.). (J. Pollock, A. Rufino, & E. Dayton, Eds.) O'Reilly Media, Inc. Retrieved March 1, 2023, from https://www.netlify.com/pdf/oreilly-modern-web-development-on-the-jamstack.pdf

Budert-Waltz, T. (2021, December 28). *What Is Research Methodology?* Retrieved March 13, 2023, from Study.com: https://study.com/learn/lesson/research-methodology-examples-approaches-techniques.html

Cambridge Dictionary. (n.d.). *Meaning of Monolithic in English*. Retrieved March 23, 2023, from Cambridge Dictionary: https://dictionary.cambridge.org/dictionary/english/monolithic

Camden, R., & Rinaldi, B. (2022). *The Jamstack Book.* (K. S. Johnson, L. Lazaris, M. Batinić, A. Marinkovich, & M. Mitchell, Eds.) New York: Manning Publications Co. doi:9781617298882

Coleman, B. (2021, March 16). *K6 HTML Report Exporter v2*. Retrieved April 8, 2023, from GitHub: https://github.com/benc-uk/k6-reporter

Coursera Inc. (2022, November 14). *What Is Python Used For? A Beginner's Guide*. Retrieved March 28, 2023, from Coursera: https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python

Crevon, T. (2022, June 8). *What Are Iterations, VUs, and How They Relate?* Retrieved May 15, 2023, from Grafana k6: https://community.k6.io/t/level-easy-what-are-iterations-vus-and-how-they-relate/3887/3

Denysov, A. (2019, November 1). *A Look at JAMstack's Speed, By the Numbers*. Retrieved March 16, 2023, from CSS-Tricks: https://css-tricks.com/a-look-at-jamstacks-speed-by-the-numbers/

Dodson, M. (2016, September 28). *WordPress In Action: Microsoft*. Retrieved May 5, 2023, from Torque Magazine: https://torquemag.io/2016/09/wordpress-microsoft/

Domantas, G. (2023, February 21). *What Is WordPress? An Overview of the World's Most Popular CMS*. Retrieved March 11, 2023, from Hostinger Tutorials: https://www.hostinger.com/tutorials/what-is-wordpress

Dziuba, A. (2021a, November 17). *Do JAMstack Websites Use the Microservice Architecture*. Retrieved March 27, 2023, from Relevant Software: https://relevant.software/blog/jamstack-review/#Do_JAMstack_websites_use_the_microservice_architecture

Dziuba, A. (2021b, August 27). *What Are Micro Frontends?* Retrieved March 27, 2023, from Relevant Software: https://relevant.software/blog/scale-frontend-micro-frontends-architecture/#What_are_micro_frontends

Falconer, J. (2022, November 26). *Jamstack Basics: What Is Jamstack, and Is It the Future of Web Development? [Better Performance]*. Retrieved May 9, 2023, from Variables.sh: https://www.variables.sh/what-is-jamstack/

Fayock, C. (2022, March 31). *New to Jamstack? Everything You Need to Know to Get Started*. Retrieved May 28, 2023, from Snipcart: https://snipcart.com/blog/jamstack

Frost, J. (n.d.). *Reliability vs Validity: Differences & Examples*. Retrieved March 20, 2023, from Statistics by Jim: https://statisticsbyjim.com/basics/reliability-vs-validity/

Gaur, A. (2022, May 6). *Median Formula for Even, Odd, and Grouped Data*. Retrieved May 9, 2023, from Adda247: https://www.adda247.com/school/median-formula/

Grafana k6. (2023a). *Browser Module Documentation*. Retrieved April 6, 2023, from Grafana k6: https://k6.io/docs/javascript-api/k6-browser/

Grafana k6. (2023b). *Load Test Types*. Retrieved May 1, 2023, from Grafana k6: https://k6.io/docs/test-types/load-test-types/

Grafana k6. (2023c). *Using k6 [Metrics]*. Retrieved May 2, 2023, from Grafana k6: https://k6.io/docs/using-k6/metrics/

Grafana k6. (2023d, April 24). *What Is Smoke Testing? How to Create a Smoke Test in k6*. Retrieved May 15, 2023, from Grafana k6: https://k6.io/docs/test-types/smoke-testing/

Grafana k6. (2023e, April 24). *What Is Load Testing? How to Create a Load Test in k6*. Retrieved May 15, 2023, from Grafana k6: https://k6.io/docs/test-types/load-testing/#average-load-testing-in-k6

Grafana k6. (2023f, April 24). *What Is Stress Testing? How to Create a Stress Test in k6*. Retrieved May 15, 2023, from Grafana k6: https://k6.io/docs/test-types/stress-testing/#stress-testing-in-k6

Grafana k6. (2023g, April 24). *What Is Spike Testing? How to Create a Spike Test in K6*. Retrieved May 15, 2023, from Grafana k6: https://k6.io/docs/test-types/spike-testing/#spike-testing-in-k6

Grafana k6. (n.d.). *k6 Documentation*. Retrieved March 8, 2023, from Grafana k6: https://k6.io/docs/

Guoan, X. (2022, December 1). *How to Fix Common NGINX Web Server Errors*. Retrieved May 21, 2023, from LinuxBabe: https://www.linuxbabe.com/linux-server/how-to-fix-common-nginx-errors

Gustafsson, R. (2021, June 29). *Our Exciting Next Step - k6 Is Now Part of Grafana Labs!* Retrieved March 8, 2023, from k6: https://k6.io/blog/joining-grafana-labs/

Harris, C. (n.d.). *Microservices vs. Monolithic Architecture*. Retrieved March 21, 2023, from Atlassian: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith

Hedenskog, P. (2017, November 9). *Sitespeed.io/plugins*. Retrieved March 9, 2023, from GitHub: https://github.com/sitespeedio/plugins

Hedenskog, P. (2021, December 20). *Sitespeed.io - Documentation*. Retrieved March 9, 2023, from Sitespeed.io: https://www.sitespeed.io/documentation/

Hedenskog, P. (2022, September 23). *Results and Examples of What You Will Get if You Run Sitespeed.io*. Retrieved April 6, 2023, from Sitespeed.io: https://www.sitespeed.io/examples/#sitespeedio

Hogan, B., & Garnett, A. (2021, December 9). *How to Set Up Nginx With HTTP/2 Support on Ubuntu 20.04*. Retrieved April 3, 2023, from DigitalOcean: https://www.digitalocean.com/community/tutorials/how-to-set-up-nginx-with-http-2-support-on-ubuntu-20-04

Inch Calculator. (n.d.). *Relative Standard Deviation Calculator*. Retrieved May 9, 2023, from Inch Calculator: https://www.inchcalculator.com/relative-standard-deviation-calculator/

Jamstack.org. (n.d.). *What Is Jamstack?* Retrieved March 17, 2023, from Jamstack.org: https://jamstack.org/

Javatpoint. (n.d.). *WordPress History*. Retrieved March 11, 2023, from Javatpoint: https://www.javatpoint.com/wordpress-history

Kostrzewa, D. (2020, December 14). *Jamstack Explained*. Retrieved March 23, 2023, from Bejamas: https://bejamas.io/blog/jamstack/

Krishnan, A. (2022, December 13). *NGINX vs Apache: Head to Head Comparison*. Retrieved May 16, 2023, from Hackr.io: https://hackr.io/blog/nginx-vs-apache#differences-between-apache-and-nginx

Krzywda, K. (2021, December 9). *Why Are Developers Choosing Jamstack More Often?* Retrieved March 23, 2023, from Naturaily: https://naturaily.com/blog/why-are-developers-choosing-jamstack-more-often

Lönn, R. (2020, March 4). *Open Source Load Testing Tool Review 2020*. Retrieved April 4, 2023, from Grafana k6: https://k6.io/blog/comparing-best-open-source-load-testing-tools/#max-traffic-generation-capability

Matilainen, M. (2020). *Comparison of Usage of Different Content Management Systems.* Bachelor's thesis. Retrieved March 17, 2023, from https://www.theseus.fi/bitstream/handle/10024/354520/Matilainen_Markus.pdf

Middleton, F. (2023, January 30). *Reliability vs. Validity in Research*. Retrieved March 20, 2023, from Scribbr: https://www.scribbr.com/methodology/reliability-vs-validity/#understanding-reliability-vs-validity

Moreira, R. (2020, July 2). *From Monolithic to Headless: How and Why You Should Adapt Your WordPress Stack*. Retrieved May 3, 2023, from Medium: https://medium.com/pixelmatters/from-monolithic-to-headless-how-and-why-we-adapted-our-wordpress-stack-309f0536007e

Pawlik, K., & Czernia, D. (2022, December 5). *Percentage Change Calculator*. Retrieved May 9, 2023, from Omni Calculator: https://www.omnicalculator.com/math/percentage-change

Rechneronline. (n.d.). *Convert Percent and Factor*. Retrieved May 9, 2023, from Rechneronline: https://rechneronline.de/anteil/percent-factor.php

Seal, R. (2022, May 27). *A Comprehensive Guide on HAR Files*. Retrieved May 26, 2023, from Keysight: https://www.keysight.com/blogs/tech/nwvs/2022/05/27/a-comprehensive-guide-on-har-files

Sharma, M. (2022, November 4). *Comparison Between Web 1.0, Web 2.0 and Web 3.0*. Retrieved March 15, 2023, from GeeksforGeeks: https://www.geeksforgeeks.org/web-1-0-web-2-0-and-web-3-0-with-their-difference/

Sheldon, R. (2023, February). *What Is Codebase (Code Base)?* Retrieved March 21, 2023, from TechTarget: https://www.techtarget.com/whatis/definition/codebase-code-base

Sirisilla, S. (2023, January 19). *Inductive and Deductive Reasoning — Strategic Approach for Conducting Research*. Retrieved March 13, 2023, from Enago Academy: https://www.enago.com/academy/inductive-and-deductive-reasoning/

Sitespeed.io. (2022a, January 21). *Metrics - Sitespeed.io*. Retrieved April 17, 2023, from Sitespeed.io: https://www.sitespeed.io/documentation/sitespeed.io/metrics/

Sitespeed.io. (2022b). *Sitespeed.io - Compare HAR Files*. Retrieved May 11, 2023, from Sitespeed.io: https://compare.sitespeed.io/

Sitespeed.io. (2023, March 7). *Welcome to the Wonderful World of Web Performance*. Retrieved March 9, 2023, from Sitespeed.io: https://www.sitespeed.io/

Stack Overflow. (2022). *Stack Overflow Developer Survey 2022*. Retrieved March 28, 2023, from Stack Overflow: https://survey.stackoverflow.co/2022/#most-popular-technologies-language-prof

Streefkerk, R. (2023, May 1). *Qualitative vs. Quantitative Research | Differences, Examples & Methods*. Retrieved from Scribbr: https://www.scribbr.com/methodology/qualitative-quantitative-research/#the-differences-between-quantitative-and-qualitative-research

Techopedia. (2020, August 25). *Techopedia Explains Microsoft Excel*. Retrieved March 8, 2023, from Techopedia: https://www.techopedia.com/definition/5430/microsoft-excel#techopedia-explains-microsoft-excel

The Editors of Encyclopaedia Britannica. (2022, October 20). *Microsoft Excel*. Retrieved March 8, 2023, from Encyclopedia Britannica: https://www.britannica.com/technology/Microsoft-Excel

The HTTP Archive. (2023, March 1). *Page Weight*. Retrieved April 3, 2023, from HTTP Archive: https://httparchive.org/reports/page-weight#bytesTotal

van der Hoeven, N. (2021, January 2021). *Comparing k6 and JMeter for Load Testing*. Retrieved April 6, 2023, from Grafana k6: https://k6.io/blog/k6-vs-jmeter/

Vistola, L. (2021, July 23). *Shifting From Monolithic Application Development*. Retrieved March 15, 2023, from DevOps.com: https://devops.com/the-move-away-from-monolithic-application-development/

Voss, L., & Alam-Naylor, S. (2022, October 13). *The 2022 Web Almanac*. Retrieved March 8, 2023, from HTTP Archive: https://almanac.httparchive.org/en/2022/jamstack#the-growth-of-the-jamstack

W3Techs. (2023a, May 3). *Market Share Yearly Trends for Content Management Systems*. Retrieved May 3, 2023, from World Wide Web Technology Surveys: https://w3techs.com/technologies/history_overview/content_management/ms/y

W3Techs. (2023b, March 11). *Usage Statistics of Content Management Systems*. Retrieved March 11, 2023, from World Wide Web Technology Surveys: https://w3techs.com/technologies/overview/content_management

Wagner, J. L. (2017). 2.3.1 Viewing Timing Information. In S. Kline, I. Martinovic´, N. Watts, K. Sullivan, & S. Wilkey (Eds.), *Web Performance in Action: Building Faster Web Pages* (p. 27). New York: Manning Publications Co. doi:ISBN: 9781617293771

Wallis, J. (2022, April 14). *What Is Jamstack Architecture? [Here's A Quick Overview For People In a Hurry!]*. Retrieved March 6, 2023, from WEBO Digital: https://webo.digital/blog/what-is-jamstack-architecture-an-overview/

Walton, P. (2022, May 11). *User-Centric Performance Metrics [Types of Metrics]*. Retrieved May 2, 2023, from Web.dev: https://web.dev/user-centric-performance-metrics/#types-of-metrics

Whitley, S. (2023a, May 1). *Sam Whitley's Thesis GitHub Repository*. Retrieved April 26, 2023, from GitHub: https://github.com/Sam-Whitley/thesis

Whitley, S. (2023b, April 12). *Sitespeed.io Runtime Settings [Desktop]*. Retrieved April 12, 2023, from GitHub: https://sam-whitley.github.io/thesis/sitespeed/desktop/L1/settings.html

Whitley, S. (2023c, April 13). *Sitespeed.io Runtime Settings [Mobile]*. Retrieved April 13, 2023, from GitHub: https://sam-whitley.github.io/thesis/sitespeed/mobile/L1/settings.html

Whitley, S. (2023d, April 29). Testing Environment Flow Chart. Created using Canva (https://www.canva.com/).

Whitley, S. (2023e, May 15). *Sam Whitley's Grafana k6 Scripts*. Retrieved May 15, 2023, from GitHub: https://github.com/Sam-Whitley/thesis/tree/main/k6/scripts

WPBeginner. (2023, January 17). *40+ Most Notable Big Name Brands That Are Using WordPress [Why Do So Many Big Name Brands Use WordPress?]*. Retrieved May 4, 2023, from WPBeginner: https://www.wpbeginner.com/showcase/40-most-notable-big-name-brands-that-are-using-wordpress/

WTF Is Jamstack? (n.d.). *What Is Jamstack?* Retrieved March 3, 2023, from WTF Is Jamstack?: https://jamstack.wtf/#what-is-jamstack

Appendix A. Mathematical Formulas

Equation 2. Example Data List and Median Formula (Gaur, 2022)

$$[Data\ list = (x_1, x_2 \dots x_n)]$$

$$[Median = Med(X) = \begin{cases} X\left[\frac{n}{2}\right], & if\ n\ is\ even \\ \frac{(X\left[\frac{n-1}{2}\right] + X\left[\frac{n+1}{2}\right]}{2}, & if\ n\ is\ odd \end{cases}$$

Note. Median formula adapted from Adda247 (https://www.adda247.com/school/median-formula/)

Equation 3. Percentage Change Formula (Pawlik & Czernia, 2022)

$$\%\ Change = 100x\,\frac{(final - initial)}{|initial|}$$

Note. Percentage change formula adapted from Omni Calculator (https://www.omnicalculator.com/math/percentage-change)

Equation 4. Percent and Factor Formula (Rechneronline, n.d.)

$$\left[Factor = \frac{percentual\ value + 100}{100}\right]$$

$$[Percentual\ value = 100 * factor - 100]$$

Note. Percent and factor formula adapted from Rechneronline (https://rechneronline.de/anteil/percent-factor.php)

Equation 5. Relative Standard Deviation (RSD) Formula (Inch Calculator, n.d.)

$$RSD = \left|\frac{\sigma}{\mu}\right| \times 100\%$$

Note. σ = Population Standard Deviation, μ = Population Mean. Formula adapted from (https://www.inchcalculator.com/relative-standard-deviation-calculator/)

Equation 6. Standard Deviation (SD) Formula (Bhandari, 2023)

$$\sigma = \sqrt{\frac{\sum(x-\mu)^2}{N}}$$

Note. σ = Population Standard Deviation, $\sum$ = Sum of…, x = Each value, μ = Population Mean, N = Number of Values in the Population. Formula adapted from (https://www.scribbr.com/statistics/standard-deviation/)

Equation 7. Excel Standard Deviation (SD) Formula

$$[= STDEV.P(B3\colon CW3)]$$

Equation 8. Example Excel Relative Standard Deviation (RSD) Formula

$$STDEV.P\left(\frac{B3\colon CW3}{AVERAGE(B3\colon CW3)}\right) \times 100$$

Appendix B. Sitespeed.io – Timing Metric Summary

Sitespeed.io Metric Summary - [L1 WordPress with LAMP Stack (VM)] (Whitley, 2023a)

Metric	Median (Min; Max)	Mean (SD)	RSD %
Timing Metrics [Desktop]			
First Paint and FCP	297 (254-344)	293 (20)	6.69
Largest Contentful Paint	297 (254-344)	293 (20)	6.69
Load Event End	382 (319-435)	382 (26)	6.80
Server Response Time	204 (170-248)	201 (17)	8.57
Time to First Byte	206 (172-250)	202 (17)	8.50
Page Load Time	382 (318-435)	382 (26)	6.83
First Visual Change	300 (234-367)	289 (25)	8.83
Last Visual Change	367 (300-434)	358 (34)	9.50
Speed Index	359 (294-426)	350 (34)	9.66
Visual Readiness	67 (33-134)	69 (26)	37.34
Visual Complete 95	367 (300-434)	356 (36)	9.97
Visual Complete 99	367 (300-434)	358 (34)	9.50
Timing Metrics [Mobile]			
First Paint and FCP	495 (387-941)	501 (65)	12.91
Largest Contentful Paint	618 (503-1642)	635 (116)	18.25
Load Event End	609 (488-1643)	627 (129)	20.53
Server Response Time	271 (180-378)	270 (39)	14.34
Time to First Byte	294 (200-385)	292 (38)	12.98
Page Load Time	608 (487-1642)	626 (129)	20.56

Note. Lists several timing metrics (https://github.com/Sam-Whitley/thesis). Median (Min; Max) and Mean (SD) values are in milliseconds (ms).

Sitespeed.io Metric Summary - [L2 WordPress with LAMP Stack (WSL2)] (Whitley, 2023a)

Metric	Median (Min; Max)	Mean (SD)	RSD %
Timing Metrics [Desktop]			
First Paint and FCP	260 (248-272)	258 (6)	2.39
Largest Contentful Paint	260 (248-272)	258 (6)	2.39
Load Event End	278 (267-296)	278 (6)	2.29
Server Response Time	179 (168-193)	179 (6)	3.21
Time to First Byte	181 (170-195)	181 (6)	3.17
Page Load Time	277 (266-295)	278 (6)	2.29
First Visual Change	267 (233-333)	258 (18)	7.15
Last Visual Change	267 (267-333)	281 (18)	6.25
Speed Index	267 (263-333)	275 (16)	5.83
Visual Readiness	33 (0-66)	23 (16)	70.05
Visual Complete 95	267 (267-333)	277 (16)	5.92
Visual Complete 99	267 (367-333)	277 (16)	5.92
Timing Metrics [Mobile]			
First Paint and FCP	417 (372-942)	432 (68)	15.79
Largest Contentful Paint	530 (450-1164)	549 (83)	15.20
Load Event End	514 (463-1129)	538 (83)	15.35
Server Response Time	200 (173-591)	215 (55)	25.35
Time to First Byte	217 (192-422)	232 (40)	17.21
Page Load Time	513 (462-1127)	537 (83)	15.36

Note. Lists several timing metrics (https://github.com/Sam-Whitley/thesis). Median (Min; Max) and Mean (SD) values are in milliseconds (ms).

Sitespeed.io Metric Summary - [L4 Jamstack with LAMP Stack (WSL2)] (Whitley, 2023a)

Metric	Median (Min; Max)	Mean (SD)	RSD %
Timing Metrics [Desktop]			
First Paint and FCP	144 (133-157)	142 (4)	3.09
Largest Contentful Paint	144 (133-157)	142 (4)	3.09
Load Event End	209 (184-247)	204 (14)	6.67
Server Response Time	8 (7-10)	8 (1)	6.02
Time to First Byte	10 (10-12)	10 (1)	4.83
Page Load Time	208 (184-247)	203 (14)	6.69
First Visual Change	133 (133-167)	136 (10)	7.11
Last Visual Change	200 (167-234)	194 (18)	9.02
Speed Index	194 (163-228)	189 (17)	8.78
Visual Readiness	67 (33-100)	58 (17)	28.64
Visual Complete 95	200 (167-234)	194 (18)	9.02
Visual Complete 99	200 (167-234)	194 (18)	9.02
Timing Metrics [Mobile]			
First Paint and FCP	262 (219-375)	267 (31)	11.56
Largest Contentful Paint	375 (328-531)	389 (40)	10.23
Load Event End	368 (324-564)	379 (44)	11.53
Server Response Time	19 (15-70)	22 (8)	38.13
Time to First Byte	43 (26-84)	45 (8)	18.42
Page Load Time	367 (323-563)	378 (44)	11.55

Note. Lists several timing metrics (https://github.com/Sam-Whitley/thesis). Median (Min; Max) and Mean (SD) values are in milliseconds (ms).

Sitespeed.io Metric Summary - [L5 Jamstack with LAMP Stack (VM)] (Whitley, 2023a)

Metric	Median (Min; Max)	Mean (SD)	RSD %
Timing Metrics [Desktop]			
First Paint and FCP	102 (78-126)	101 (9)	9.17
Largest Contentful Paint	102 (78-126)	101 (9)	9.17
Load Event End	209 (154-284)	209 (21)	9.87
Server Response Time	5 (4-13)	6 (2)	29.85
Time to First Byte	6 (5-14)	6 (1)	22.70
Page Load Time	209 (153-284)	209 (21)	9.88
First Visual Change	100 (67-134)	100 (22)	22.10
Last Visual Change	167 (100-267)	171 (35)	20.73
Speed Index	161 (97-256)	163 (35)	21.64
Visual Readiness	67 (0-167)	70 (31)	43.56
Visual Complete 95	167 (100-267)	169 (37)	22.18
Visual Complete 99	167 (100-267)	170 (36)	20.89
Timing Metrics [Mobile]			
First Paint and FCP	209 (184-296)	216 (22)	10.07
Largest Contentful Paint	352 (283-453)	356 (35)	9.86
Load Event End	329 (292-579)	348 (58)	16.71
Server Response Time	15 (10-69)	19 (11)	58.38
Time to First Byte	36 (27-95)	39 (10)	26.02
Page Load Time	328 (291-578)	347 (58)	16.73

Note. Lists several timing metrics (https://github.com/Sam-Whitley/thesis). Median (Min; Max) and Mean (SD) values are in milliseconds (ms).

Sitespeed.io Metric Summary - [L6 WordPress with LEMP Stack (VM)] (Whitley, 2023a)

Metric	Median (Min; Max)	Mean (SD)	RSD %
Timing Metrics [Desktop]			
First Paint and FCP	278 (254-648)	282 (38)	13.52
Largest Contentful Paint	278 (254-648)	282 (38)	13.52
Load Event End	341 (309-720)	345 (40)	11.54
Server Response Time	200 (178-568)	205 (38)	18.46
Time to First Byte	201 (179-570)	206 (38)	18.42
Page Load Time	341 (308-720)	345 (40)	11.57
First Visual Change	267 (234-634)	278 (39)	13.98
Last Visual Change	334 (300-700)	331 (44)	13.38
Speed Index	301 (267-694)	320 (46)	14.30
Visual Readiness	67 (0-133)	53 (23)	43.47
Visual Complete 95	300 (267-700)	324 (47)	14.55
Visual Complete 99	334 (300-700)	330 (44)	13.44
Timing Metrics [Mobile]			
First Paint and FCP	473 (386-1055)	478 (72)	15.01
Largest Contentful Paint	603 (507-1288)	608 (82)	13.41
Load Event End	584 (497-1286)	594 (85)	14.27
Server Response Time	264 (193-564)	267 (48)	18.13
Time to First Byte	281 (210-681)	287 (55)	19.21
Page Load Time	583 (496-1284)	593 (85)	14.28

Note. Lists several timing metrics (https://github.com/Sam-Whitley/thesis). Median (Min; Max) and Mean (SD) values are in milliseconds (ms).

Sitespeed.io Metric Summary - [L7 Jamstack with LEMP Stack (VM)] (Whitley, 2023a)

Metric	Median (Min; Max)	Mean (SD)	RSD %
Timing Metrics [Desktop]			
First Paint and FCP	84 (77-91)	82 (3)	4.25
Largest Contentful Paint	84 (77-91)	82 (3)	4.25
Load Event End	135 (122-165)	136 (7)	5.47
Server Response Time	3 (2-5)	3 (1)	22.11
Time to First Byte	4 (3-5)	4 (0)	4.95
Page Load Time	134 (122-165)	136 (7)	5.49
First Visual Change	67 (67-100)	73 (13)	17.67
Last Visual Change	134 (100-167)	123 (21)	17.14
Speed Index	90 (67-161)	111 (22)	19.61
Visual Readiness	51 (0-100)	50 (22)	43.02
Visual Complete 95	100 (67-167)	115 (23)	20.35
Visual Complete 99	134 (100-167)	122 (22)	17.73
Timing Metrics [Mobile]			
First Paint and FCP	208 (184-297)	215 (23)	10.75
Largest Contentful Paint	340 (262-788)	351 (68)	19.36
Load Event End	316 (286-992)	347 (93)	26.80
Server Response Time	13 (9-57)	18 (11)	60.65
Time to First Byte	34 (21-82)	37 (11)	30.32
Page Load Time	315 (285-990)	346 (93)	26.85

Note. Lists several timing metrics (https://github.com/Sam-Whitley/thesis). Median (Min; Max) and Mean (SD) values are in milliseconds (ms).

Sitespeed.io Metric Summary - [L8 WordPress with LEMP Stack (WSL2)] (Whitley, 2023a)

Metric	Median (Min; Max)	Mean (SD)	RSD %
Timing Metrics [Desktop]			
First Paint and FCP	254 (248-279)	257 (7)	2.71
Largest Contentful Paint	254 (248-279)	257 (7)	2.71
Load Event End	274 (264-298)	276 (7)	2.63
Server Response Time	175 (169-195)	178 (6)	3.43
Time to First Byte	177 (171-197)	179 (6)	3.42
Page Load Time	274 (263-297)	276 (7)	2.63
First Visual Change	267 (233-300)	252 (18)	7.12
Last Visual Change	267 (267-300)	280 (16)	5.77
Speed Index	267 (263-300)	276 (16)	5.62
Visual Readiness	33 (0-34)	28 (13)	45.29
Visual Complete 95	267 (267-300)	279 (16)	5.65
Visual Complete 99	267 (267-300)	279 (16)	5.65
Timing Metrics [Mobile]			
First Paint and FCP	397 (362-676)	416 (51)	12.38
Largest Contentful Paint	513 (431-821)	528 (60)	11.38
Load Event End	502 (453-805)	520 (61)	11.70
Server Response Time	191 (173-314)	206 (38)	18.45
Time To First Byte	210 (183-350)	226 (39)	17.11
Page Load Time	500 (452-804)	519 (61)	11.70

Note. Lists several timing metrics (https://github.com/Sam-Whitley/thesis). Median (Min; Max) and Mean (SD) values are in milliseconds (ms).

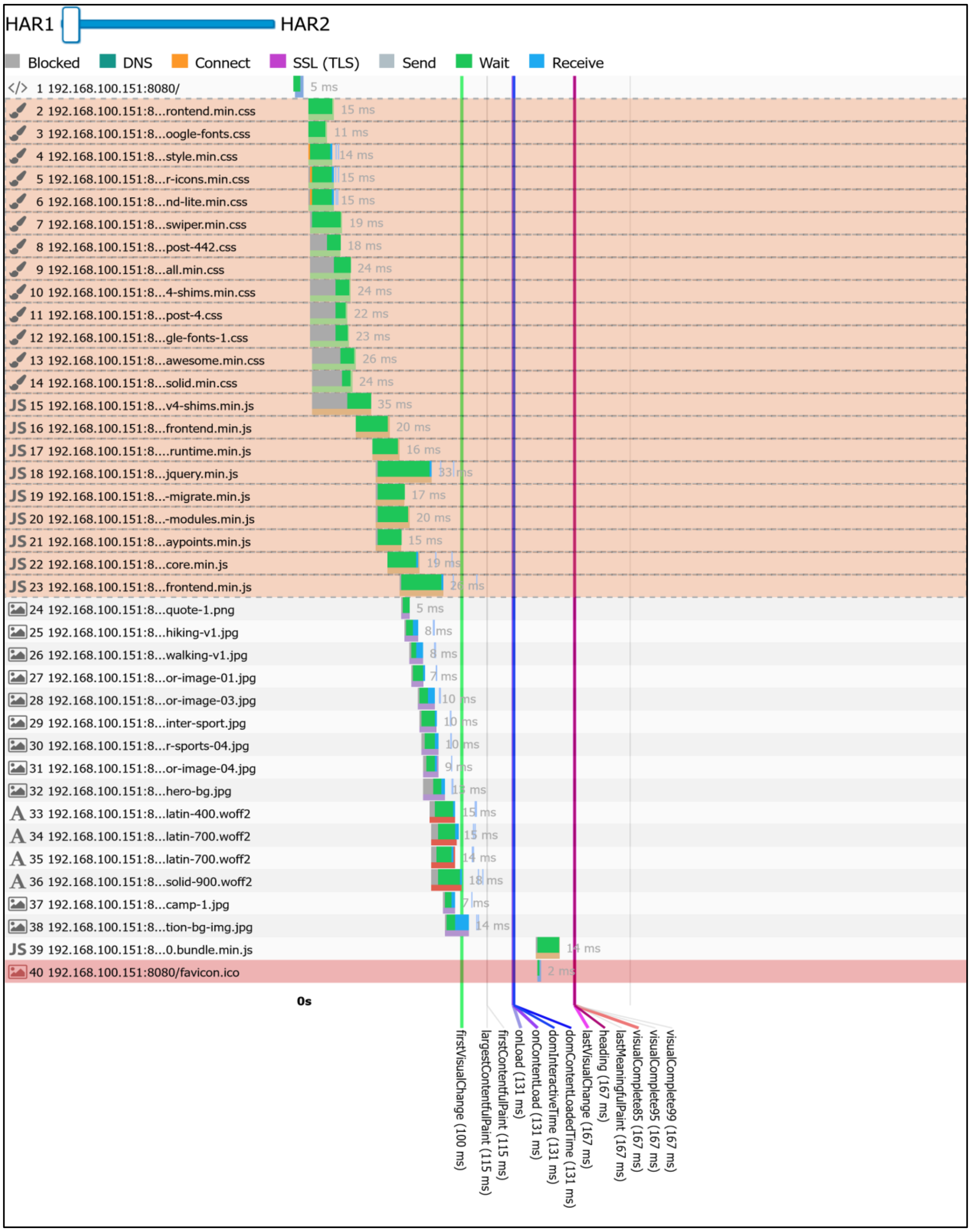
Metric Summary - [L9 Jamstack with LEMP Stack (WSL2)] (Whitley, 2023a)

Metric	Median (Min; Max)	Mean (SD)	RSD %
Timing Metrics [Desktop]			
First Paint and FCP	115 (96-133)	112 (7)	6.66
Largest Contentful Paint	115 (96-133)	112 (7)	6.66
Load Event End	134 (117-154)	134 (8)	5.87
Server Response Time	5 (5-6)	5 (0)	3.89
Time to First Byte	7 (6-8)	7 (0)	4.13
Page Load Time	134 (117-153)	134 (8)	5.88
First Visual Change	100 (100-134)	114 (16)	14.39
Last Visual Change	133 (100-167)	146 (17)	11.71
Speed Index	133 (100-164)	142 (17)	11.95
Visual Readiness	33 (0-67)	32 (16)	51.01
Visual Complete 95	133 (100-167)	145 (18)	12.45
Visual Complete 99	133 (100-167)	145 (18)	12.45
Timing Metrics [Mobile]			
First Paint and FCP	229 (187-707)	239 (57)	23.79
Largest Contentful Paint	341 (284-841)	356 (63)	17.77
Load Event End	327 (291-892)	349 (79)	22.52
Server Response Time	16 (12-202)	23 (23)	102.22
Time to First Byte	39 (23-291)	45 (28)	63.70
Page Load Time	326 (290-891)	348 (78)	22.56

Note. Lists several timing metrics (https://github.com/Sam-Whitley/thesis). Median (Min; Max) and Mean (SD) values are in milliseconds (ms).

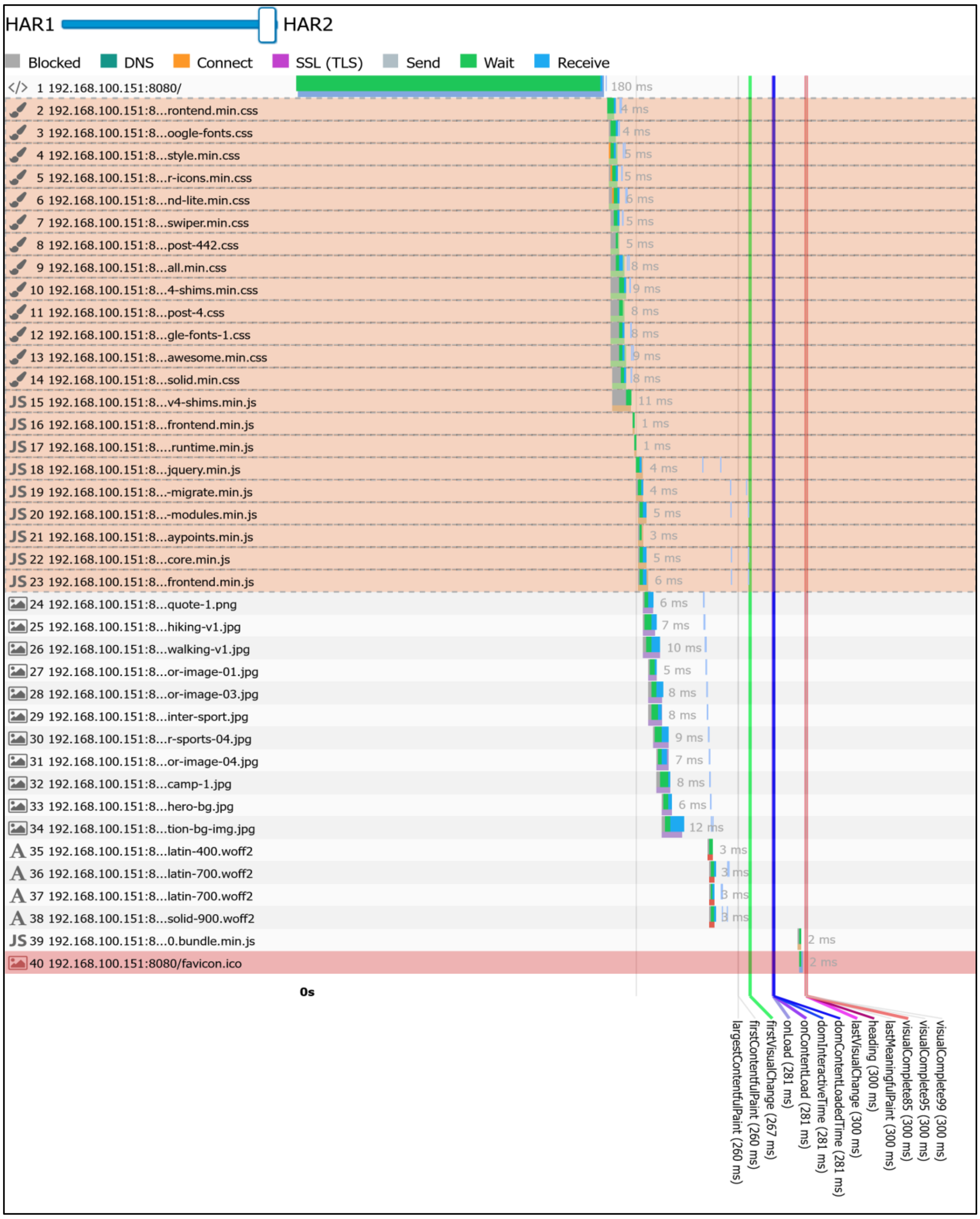
Appendix C. Sitespeed.io – Waterfall Graphs

Waterfall Graph - Jamstack (LEMP WSL2) [Desktop] (Sitespeed.io, 2022b; Whitley, 2023a)



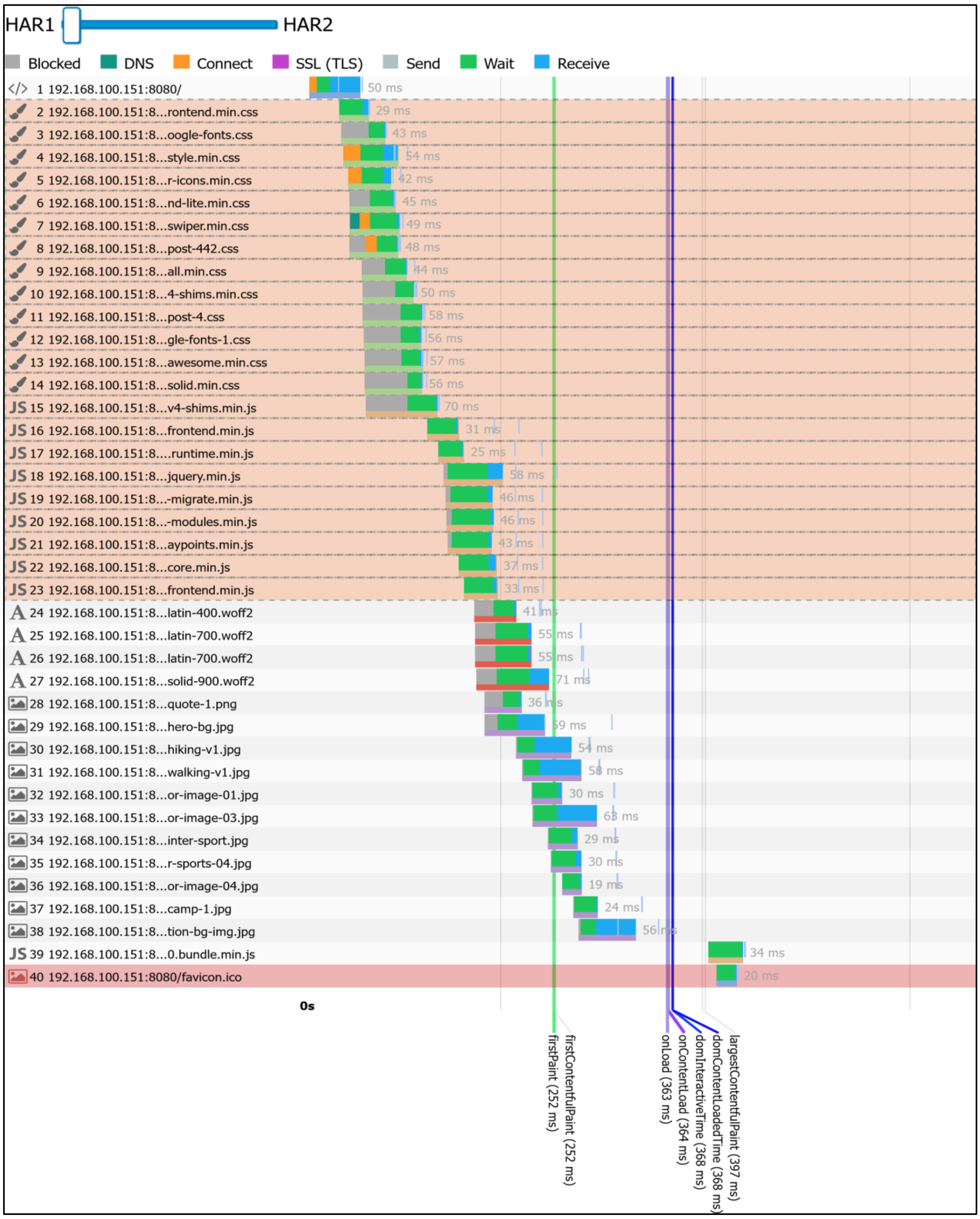
Note. This figure shows a waterfall chart of the site Jamstack (LEMP WSL2) [Desktop], which was adapted from Sitespeed.io's HAR file comparison website (https://compare.sitespeed.io/), created by the Sitespeed.io team. The HAR file used was from the author's GitHub repository (https://github.com/Sam-Whitley/thesis).

Waterfall Graph - WordPress (LEMP WSL2) [Desktop] (Sitespeed.io, 2022b; Whitley, 2023a)



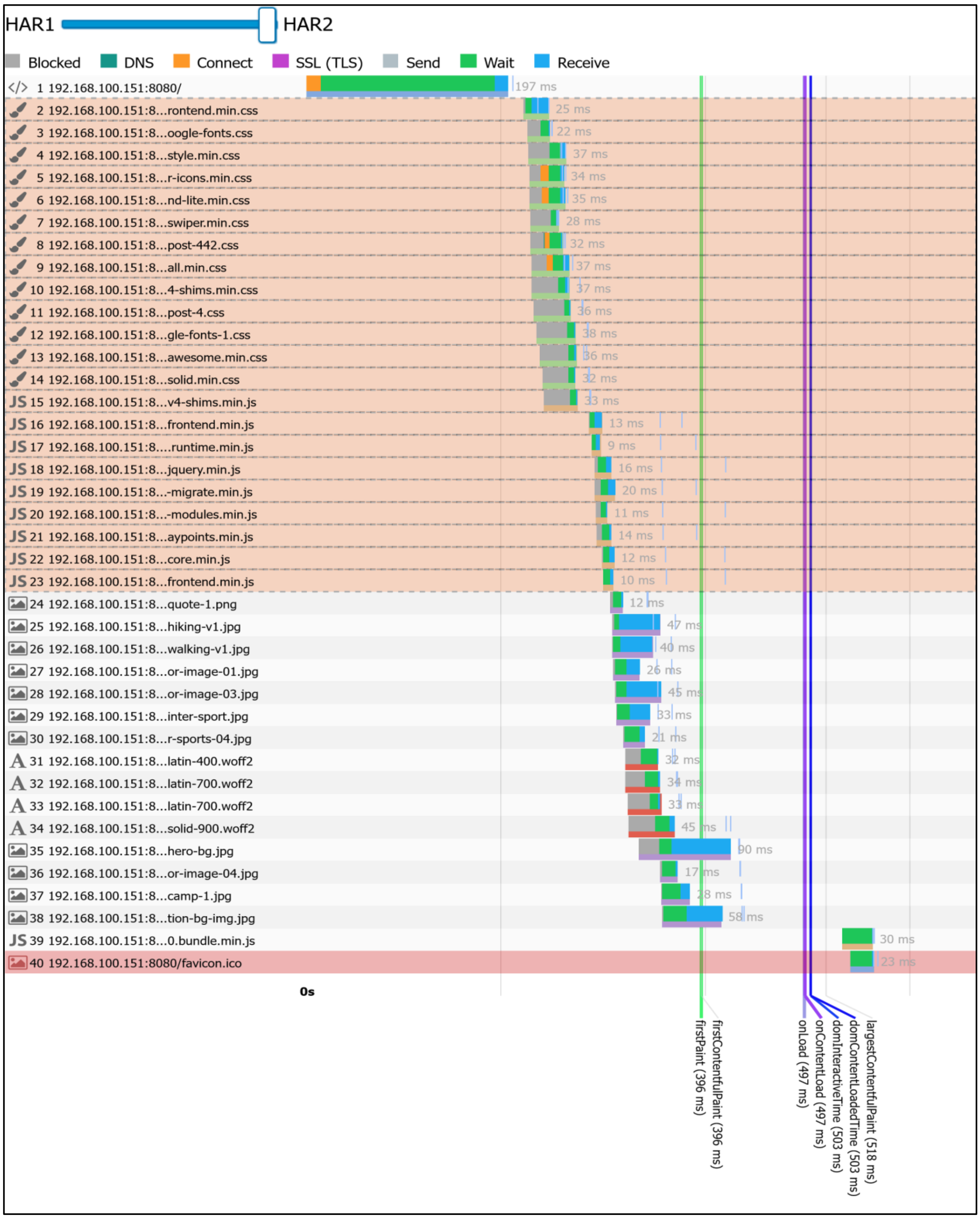
Note. This figure shows a waterfall chart of the site WordPress (LEMP WSL2) [Desktop], which was adapted from Sitespeed.io's HAR file comparison website (https://compare.sitespeed.io/), created by the Sitespeed.io team. The HAR file used was from the author's GitHub repository (https://github.com/Sam-Whitley/thesis).

Waterfall Graph - Jamstack (LEMP WSL2) [Mobile] (Sitespeed.io, 2022b; Whitley, 2023a)



Note. This figure shows a waterfall chart of the site Jamstack (LEMP WSL2) [Mobile], which was adapted from Sitespeed.io's HAR file comparison website (https://compare.sitespeed.io/), created by the Sitespeed.io team. The HAR file used was from the author's GitHub repository (https://github.com/Sam-Whitley/thesis).

Waterfall Graph - WordPress (LEMP WSL2) [Mobile] (Sitespeed.io, 2022b; Whitley, 2023a)



Note. This figure shows a waterfall chart of the site WordPress (LEMP WSL2) [Desktop], which was adapted from Sitespeed.io's HAR file comparison website (https://compare.sitespeed.io/), created by the Sitespeed.io team. The HAR file used was from the author's GitHub repository (https://github.com/Sam-Whitley/thesis).

Appendix D. Grafana k6 – Check Examples

smokeTest.js - Wrong IP Address (Whitley, 2023a)

```
          /\      |‾‾| /‾‾/   /‾‾/
     /\  /  \     |  |/  /   /  /
    /  \/    \    |     (   /   ‾‾\
   /          \   |  |\  \ |  (‾)  |
  / __________ \  |__| \__\ \_____/ .io

  execution: local
     script: smokeTest.js
     output: csv (L1-smokeTest.csv)

  scenarios: (100.00%) 1 scenario, 1 max VUs, 35s max duration (incl. graceful stop):
           * default: 1 looping VUs for 5s (gracefulStop: 30s)

WARN[0002] Request Failed
error="Get \"http://192.168.100.151:8080/\": dial tcp 192.168.100.151:8080: connectex:
No connection could be made because the target machine actively refused it."

WARN[0005] Request Failed
error="Get \"http://192.168.100.151:8080/\": dial tcp 192.168.100.151:8080: connectex:
No connection could be made because the target machine actively refused it."

INFO[0006] [k6-reporter v2.3.0] Generating HTML summary report  source=console
     ✗ status is 200
      ↳  0% — ✓ 0 / ✗ 2
     ✗ content type is text/html
      ↳  0% — ✓ 0 / ✗ 2
     ✗ verify homepage text
      ↳  0% — ✓ 0 / ✗ 2

     checks.....................: 0.00%   ✓ 0        ✗ 6
     data_received..............: 0 B     0 B/s
     data_sent..................: 0 B     0 B/s
     http_req_blocked...........: avg=0s min=0s med=0s max=0s p(90)=0s p(95)=0s
     http_req_connecting........: avg=0s min=0s med=0s max=0s p(90)=0s p(95)=0s
   ✓ http_req_duration..........: avg=0s min=0s med=0s max=0s p(90)=0s p(95)=0s
     http_req_failed............: 100.00% ✓ 2        ✗ 0
     http_req_receiving.........: avg=0s min=0s med=0s max=0s p(90)=0s p(95)=0s
     http_req_sending...........: avg=0s min=0s med=0s max=0s p(90)=0s p(95)=0s
     http_req_tls_handshaking...: avg=0s min=0s med=0s max=0s p(90)=0s p(95)=0s
     http_req_waiting...........: avg=0s min=0s med=0s max=0s p(90)=0s p(95)=0s
     http_reqs..................: 2       0.332844/s
     iteration_duration.........: avg=3s min=3s med=3s max=3s p(90)=3s p(95)=3s
     iterations.................: 2       0.332844/s
     vus........................: 1       min=1      max=1

running (06.0s), 0/1 VUs, 2 complete and 0 interrupted iterations
default ✓ [======================================] 1 VUs  5s
```

Note. This smoke test demonstrates testing an environment (in this case environment L1) with the wrong IP address, which resulted in failed checks and a failed test.

smokeTest.js – Error Establishing a Database Connection (WordPress) (Whitley, 2023a)

```
          /\      |‾‾| /‾‾/   /‾‾/
     /\  /  \     |  |/  /   /  /
    /  \/    \    |     (   /   ‾‾\
   /          \   |  |\  \ |  (‾)  |
  / __________ \  |__| \__\ \_____/ .io

  execution: local
     script: smokeTest.js
     output: csv (csv/L6-smokeTest.csv)

  scenarios: (100.00%) 1 scenario, 1 max VUs, 35s max duration (incl. graceful stop):
           * default: 1 looping VUs for 5s (gracefulStop: 30s)

INFO[0005] [k6-reporter v2.3.0] Generating HTML summary report  source=console
     ✗ status is 200
      ↳  0% — ✓ 0 / ✗ 5
     ✓ content type is text/html
     ✗ verify homepage text
      ↳  0% — ✓ 0 / ✗ 5

     checks.....................: 33.33%  ✓ 5        ✗ 10
     data_received..............: 14 kB   2.8 kB/s
     data_sent..................: 405 B   81 B/s
     http_req_blocked...........: avg=111.06µs min=0s      med=0s      max=555.3µs
p(90)=333.18µs p(95)=444.23µs
     http_req_connecting........: avg=111.06µs min=0s      med=0s      max=555.3µs
p(90)=333.18µs p(95)=444.23µs
   ✓ http_req_duration..........: avg=3.23ms   min=2.7ms   med=3.3ms   max=3.77ms
p(90)=3.72ms   p(95)=3.74ms
     http_req_failed............: 100.00% ✓ 5        ✗ 0
     http_req_receiving.........: avg=588.26µs min=167.5µs med=533.8µs max=1.16ms
p(90)=1.01ms   p(95)=1.08ms
     http_req_sending...........: avg=0s       min=0s      med=0s      max=0s
p(90)=0s       p(95)=0s
     http_req_tls_handshaking...: avg=0s       min=0s      med=0s      max=0s
p(90)=0s       p(95)=0s
     http_req_waiting...........: avg=2.65ms   min=2.48ms  med=2.51ms  max=3.23ms
p(90)=2.95ms   p(95)=3.09ms
     http_reqs..................: 5       0.995172/s
     iteration_duration.........: avg=1s       min=1s      med=1s      max=1s
p(90)=1s       p(95)=1s
     iterations.................: 5       0.995172/s
     vus........................: 1       min=1      max=1

running (05.0s), 0/1 VUs, 5 complete and 0 interrupted iterations
default ✓ [======================================] 1 VUs  5s
```

Note. This smoke test demonstrates testing an environment (in this case environment L6). While some checks and data results were received, the test clearly indicated which resulted in failed checks and a failed test.

smokeTest.js – Successful Smoke Test (Whitley, 2023a)

```
          /\      |‾‾| /‾‾/   /‾‾/
     /\  /  \     |  |/  /   /  /
    /  \/    \    |     (   /   ‾‾\
   /          \   |  |\  \ |  (‾)  |
  / __________ \  |__| \__\ \_____/ .io

  execution: local
     script: smokeTest.js
     output: csv (L1-smokeTest.csv)

  scenarios: (100.00%) 1 scenario, 1 max VUs, 35s max duration (incl. graceful stop):
           * default: 1 looping VUs for 5s (gracefulStop: 30s)

INFO[0006] [k6-reporter v2.3.0] Generating HTML summary report  source=console
     ✓ status is 200
     ✓ content type is text/html
     ✓ verify homepage text

     checks.........................: 100.00% ✓ 15       ✗ 0
     data_received..................: 591 kB  99 kB/s
     data_sent......................: 405 B   68 B/s
     http_req_blocked...............: avg=360.36µs min=0s       med=0s       max=1.8ms
p(90)=1.08ms   p(95)=1.44ms
     http_req_connecting............: avg=360.36µs min=0s       med=0s       max=1.8ms
p(90)=1.08ms   p(95)=1.44ms
   ✓ http_req_duration..............: avg=195.88ms min=185.09ms med=193.03ms
max=217.3ms p(90)=207.68ms p(95)=212.49ms
       { expected_response:true }...: avg=195.88ms min=185.09ms med=193.03ms
max=217.3ms p(90)=207.68ms p(95)=212.49ms
     http_req_failed................: 0.00%   ✓ 0        ✗ 5
     http_req_receiving.............: avg=7.83ms   min=5.46ms   med=6.46ms   max=11ms
p(90)=10.67ms  p(95)=10.84ms
     http_req_sending...............: avg=0s       min=0s       med=0s       max=0s
p(90)=0s       p(95)=0s
     http_req_tls_handshaking.......: avg=0s       min=0s       med=0s       max=0s
p(90)=0s       p(95)=0s
     http_req_waiting...............: avg=188.05ms min=179.05ms med=185.26ms
max=206.3ms p(90)=198.41ms p(95)=202.35ms
     http_reqs......................: 5       0.835372/s
     iteration_duration.............: avg=1.19s    min=1.18s    med=1.19s    max=1.21s
p(90)=1.2s     p(95)=1.21s
     iterations.....................: 5       0.835372/s
     vus............................: 1       min=1      max=1

running (06.0s), 0/1 VUs, 5 complete and 0 interrupted iterations
default ✓ [======================================] 1 VUs  5s
```

Note. This smoke test demonstrates testing an environment (in this case environment L6) that passes all of the checks.

Appendix E. Grafana k6 – Load Testing Scripts

Smoke Testing Script (smokeTest.js) (Grafana k6, 2023d; Whitley, 2023e)

```javascript
import http from 'k6/http';
import { check, sleep } from 'k6';
import { htmlReport } from 'https://raw.githubusercontent.com/benc-uk/k6-
reporter/main/dist/bundle.js';
import { textSummary } from 'https://jslib.k6.io/k6-summary/0.0.1/index.js';

const envNameMap = {…
};

export const options = {
  vus: 3,
  duration: '1m',
};

export default function () {
  const res = http.get(`http://${__ENV.IP_ADDRESS}`);
  check(res, {
    'status is 200': (r) => r.status === 200,
    'content type is text/html': (r) => r.headers['Content
Type'].includes('text/html'),
    'verify homepage text': (r) => r.body.includes('Lorem ipsum dolor'),
  });
  sleep(1);
}

function getFullEnvName(envAlias) { return envNameMap[envAlias] || envAlias; }

export function handleSummary(data) {
  const fullEnvName = getFullEnvName(__ENV.ENVNAME);
  return {
    [`reports/${fullEnvName}.html`]: htmlReport(data, { title: `[Smoke Test |
${fullEnvName}]`}),
    stdout: textSummary(data, { indent: ' ', enableColors: true }),
  };
}
```

Note. This smoke test ensure that the system is performing as expected under minimal load. (Grafana k6, 2023b)

Average-Load Testing Script (averageLoadTest.js) (Grafana k6, 2023e; Whitley, 2023e)

```
import http from 'k6/http';
import { check, sleep } from 'k6';
import { htmlReport } from 'https://raw.githubusercontent.com/benc-uk/k6-reporter/main/dist/bundle.js';
import { textSummary } from 'https://jslib.k6.io/k6-summary/0.0.1/index.js'

const envNameMap = {…
};

export const options = {
  stages: [
    { duration: '5m', target: 100 },
    { duration: '30m', target: 100 },
    { duration: '5m', target: 0 },
    // 40m
  ],
};

export default function () {
  const res = http.get(`http://${__ENV.IP_ADDRESS}`);
  check(res, {
    'status is 200': (r) => r.status === 200,
    'content type is text/html': (r) => r.headers['Content-Type'].includes('text/html'),
    'verify homepage text': (r) => r.body.includes('Lorem ipsum dolor'),
  });
  sleep(1);
}

function getFullEnvName(envAlias) {
  return envNameMap[envAlias] || envAlias;
}

export function handleSummary(data) {
  const fullEnvName = getFullEnvName(__ENV.ENVNAME);
  return {
    [`reports/${fullEnvName}.html`]: htmlReport(data, {title: `[Average-Load Test | ${fullEnvName}]`}),
    stdout: textSummary(data, { indent: ' ', enableColors: true }),
  };
}
```

Note. This average-load test assesses how the system is performing under expected normal conditions. (Grafana k6, 2023b)

Stress Testing Script (stressTest.js) (Grafana k6, 2023f; Whitley, 2023e)

```javascript
import http from 'k6/http';
import { check, sleep } from 'k6';
import { htmlReport } from 'https://raw.githubusercontent.com/benc-uk/k6-reporter/main/dist/bundle.js';
import { textSummary } from 'https://jslib.k6.io/k6-summary/0.0.1/index.js'

const envNameMap = {…
};

export const options = {
  stages: [
    { duration: '10m', target: 200 },
    { duration: '30m', target: 200 },
    { duration: '5m', target: 0 },
    // 45m
  ],
};

export default function () {
  const res = http.get(`http://${__ENV.IP_ADDRESS}`);
  check(res, {
    'status is 200': (r) => r.status === 200,
    'content type is text/html': (r) => r.headers['Content-Type'].includes('text/html'),
    'verify homepage text': (r) => r.body.includes('Lorem ipsum dolor'),
  });
  sleep(1);
}

function getFullEnvName(envAlias) {
  return envNameMap[envAlias] || envAlias;
}

export function handleSummary(data) {
  const fullEnvName = getFullEnvName(__ENV.ENVNAME);
  return {
    [`reports/${fullEnvName}.html`]: htmlReport(data, {title: `[Stress Test | ${fullEnvName}]`}),
    stdout: textSummary(data, { indent: ' ', enableColors: true }),
  };
}
```

Note. This stress test assesses how well the system is performing under extreme loads when exceeding the expected averages. (Grafana k6, 2023b)

Spike Testing Script (spikeTest.js) (Grafana k6, 2023g; Whitley, 2023e)

```
import http from 'k6/http';
import { check, sleep } from 'k6';
import { htmlReport } from 'https://raw.githubusercontent.com/benc-uk/k6-reporter/main/dist/bundle.js';
import { textSummary } from 'https://jslib.k6.io/k6-summary/0.0.1/index.js'

const envNameMap = {
};

export const options = {
  stages: [
    { duration: '2m', target: 2000 },
    { duration: '1m', target: 0 },
    // 3m
  ],
};

export default function () {
  const res = http.get(`http://${__ENV.IP_ADDRESS}`);
  check(res, {
    'status is 200': (r) => r.status === 200,
    'content type is text/html': (r) => r.headers['Content-Type'].includes('text/html'),
    'verify homepage text': (r) => r.body.includes('Lorem ipsum dolor'),
  });
  sleep(1);
}

function getFullEnvName(envAlias) {
  return envNameMap[envAlias] || envAlias;
}

export function handleSummary(data) {
  const fullEnvName = getFullEnvName(__ENV.ENVNAME);
  return {
    [`reports/${fullEnvName}.html`]: htmlReport(data, {title: `[Spike Test | ${fullEnvName}]`}),
    stdout: textSummary(data, { indent: ' ', enableColors: true }),
  };
}
```

Note. The spike test validates the behaviour and survival of the system when subjected to a sudden, short, and massive increase in activity. (Grafana k6, 2023b)

Appendix F. Grafana k6 – Smoke Test Results

Smoke Test [L1 - WP with LAMP Stack (VM)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	180.26 ms (163.80 ms - 259.73 ms)	188.87 ms	221.79 ms	239.23 ms
req_waiting	168.78 ms (157.65 ms - 244.95 ms)	177.49 ms	208.21 ms	221.90 ms
req_connecting	0 s (0 s - 1.2 ms)	19.26 µs	0 s	0 s
req_tls_handshaking	0 s (0 s - 0s)	0 s	0 s	0 s
req_sending	0 s (0 s - 0s)	0 s	0 s	0 s
req_receiving	8.83 ms (5.06 ms - 78.78 ms)	11.38 ms	16.01 ms	21.68 ms
req_blocked	0 s (0 s - 1.2 ms)	19.26 µs	0 s	0 s
iteration_duration	1.18 s (1.16 s - 1.26 s)	1.18 s	1.22 s	1.24 s

Smoke Test [L1 - WP with LAMP Stack (VM)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	153	2.513721/s
Data Sent/Received	12 kB / 18 MB	204 B / 297 kB

Smoke Test [L1 - WP with LAMP Stack (VM)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures
Total Checks	459	0
Status is 200 (OK status)	153	0
Content type is text/HTML	153	0
Verify Homepage Text	153	0

Smoke Test [L2 - WP with LAMP Stack (WSL2)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	166.39 ms (162.20 ms - 180.17 ms)	167.08 ms	170.83 ms	174.29 ms
req_waiting	165.39 ms (161.14 ms - 179.28 ms)	166.08 ms	169.89 ms	173.01 ms
req_connecting	0 s (0 s - 0s)	0 s	0 s	0 s
req_tls_handshaking	0 s (0 s - 0s)	0 s	0 s	0 s
req_sending	0 s (0 s - 0s)	0 s	0 s	0 s
req_receiving	1.03 ms (503.7µs - 2.58 ms)	1 ms	1.57 ms	1.63 ms
req_blocked	0 s (0 s - 0s)	0 s	0 s	0 s
iteration_duration	1.17 s (1.16s - 1.19 s)	1.17 s	1.18 s	1.18 s

Smoke Test [L2 - WP with LAMP Stack (WSL2)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	153	2.548485/s
Data Sent/Received	13 kB / 18 MB	219 B / 302 kB

Smoke Test [L2 - WP with LAMP Stack (WSL2)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures
Total Checks	459	0
Status is 200 (OK status)	153	0
Content type is text/HTML	153	0
Verify Homepage Text	153	0

Smoke Test [L4 - Jamstack with LAMP Stack (WSL2)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	6.41 ms (4.77 ms - 17.51 ms)	7.13 ms	11.42 ms	13.66 ms
req_waiting	4.07 ms (2.66 ms - 14.63 ms)	4.95 ms	7.80 ms	10.69 ms
req_connecting	0 s (0 s - 530.20 µs)	8.83 µs	0 s	0 s
req_tls_handshaking	0 s (0 s - 0s)	0 s	0 s	0 s
req_sending	0 s (0 s - 529.59 µs)	7.03 µs	0 s	0 s
req_receiving	2.11 ms (517 µs - 6.28 ms)	2.17 ms	2.71 ms	3.67 ms
req_blocked	0 s (0 s - 530.20 µs)	8.83 µs	0 s	0 s
iteration_duration	1.01 s (1 s - 1.02 s)	1.01 s	1.02 s	1.02 s

Smoke Test [L4 - Jamstack with LAMP Stack (WSL2)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	180	2.959218/s
Data Sent/Received	16 kB / 21 MB	255 B / 346 kB

Smoke Test [L4 - Jamstack with LAMP Stack (WSL2)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures
Total Checks	540	0
Status is 200 (OK status)	180	0
Content type is text/HTML	180	0
Verify Homepage Text	180	0

Smoke Test [L5 - Jamstack with LAMP Stack (VM)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	8.66 ms (6.39 ms - 26.24 ms)	9.67 ms	12.51 ms	14.29 ms
req_waiting	537.84 µs (503.80 µs)	815.42 µs	1.36 ms	1.68 ms
req_connecting	0 s (0 s - 718.20 µs)	11.97 µs	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 0 s)	0 s	0 s	0 s
req_receiving	7.99 ms (5.86 ms - 22.49 ms)	8.86 ms	11.44 ms	12.61 ms
req_blocked	0 s (0 s - 718.20 µs)	11.97 µs	0 s	0 s
iteration_duration	1 s (1 s - 1.02 s)	1.01 s	1.01 s	1.01 s

Smoke Test [L5 - Jamstack with LAMP Stack (VM)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	180	2.966327/s
Data Sent/Received	15 kB / 21 MB	240 B / 247 kB

Smoke Test [L5 - Jamstack with LAMP Stack (VM)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures
Total Checks	540	0
Status is 200 (OK status)	180	0
Content type is text/HTML	180	0
Verify Homepage Text	180	0

Smoke Test [L6 - WordPress with LEMP Stack (VM)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	207.46 ms (174.13 ms - 286.28 ms)	214.61 ms	251.63 ms	263.36 ms
req_waiting	192.01 ms (161.28 ms - 243.18 ms)	191.83 ms	211.77 ms	218.24 ms
req_connecting	0 s (0 s - 670 µs)	12.41 µs	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 148.40 µs)	989 ns	0 s	0 s
req_receiving	9.69 ms (5.41 ms - 66.24 ms)	22.78 ms	56.32 ms	57.80 ms
req_blocked	0 s (0 s - 670 µs)	13.40 µs	0 s	0 s
iteration_duration	1.20 s (1.17 s - 1.28 s)	1.21 s	1.25 s	1.26 s

Smoke Test [L6 - WordPress with LEMP Stack (VM)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	150	2.457366/s
Data Sent/Received	12 kB / 18 MB	199 B / 291 kB

Smoke Test [L6 - WordPress with LEMP Stack (VM)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures
Total Checks	450	0
Status is 200 (OK status)	150	0
Content type is text/HTML	150	0
Verify Homepage Text	150	0

Smoke Test [L7 - Jamstack with LEMP Stack (VM)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	8.86 ms (7.04 ms - 19.85 ms)	9.54 ms	12.53 ms	14.31 ms
req_waiting	540.15 µs (54.50 µs - 2.94 ms)	715.13 µs	1.33 ms	1.71 ms
req_connecting	0 s (0 s - 909 µs)	15.15 µs	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 94.10 µs)	522 ns	0 s	0 s
req_receiving	8.15 ms (6.52 ms - 18.36 ms)	8.82 ms	11.07 ms	13.22 ms
req_blocked	0 s (0 s - 909 µs)	15.15 µs	0 s	0 s
iteration_duration	1 s (1 s - 1.02 s)	1.01 s	1.01 s	1.01 s

Smoke Test [L7 - Jamstack with LEMP Stack (VM)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	180	2.968861/s
Data Sent/Received	15 kB / 21 MB	231 B / 247 kB

Smoke Test [L7 - Jamstack with LEMP Stack (VM) [Checks] (Whitley, 2023a)

Checks	Passes	Failures
Total Checks	540	0
Status is 200 (OK status)	180	0
Content type is text/HTML	180	0
Verify Homepage Text	180	0

Smoke Test [L8 - WordPress with LEMP Stack (WSL2)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	162 ms (156.20 ms - 179.11 ms)	162.58 ms	166.82 ms	169.23 ms
req_waiting	160.89 ms (155.15 ms - 178.05 ms)	161.45 ms	165.74 ms	168.01 ms
req_connecting	0 s (0 s - 0 s)	0 s	0 s	0 s
req_tls_handshaking	0 s (0 s - 0s)	0 s	0 s	0 s
req_sending	0 s (0 s - 0 s)	0 s	0 s	0 s
req_receiving	1.05 ms (505.29 µs - 2.17 ms)	1.13 ms	1.57 ms	1.70 ms
req_blocked	0 s (0 s - 0 s)	0 s	0 s	0 s
iteration_duration	1.16 s (1.15 s - 1.18 s)	1.17 s	1.17 s	1.17 s

Smoke Test [L8 - WordPress with LEMP Stack (WSL2)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	156	2.562256/s
Data Sent/Received	13 kB / 19 MB	220 B / 304 kB

Smoke Test [L8 - WordPress with LEMP Stack (WSL2)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures
Total Checks	468	0
Status is 200 (OK status)	156	0
Content type is text/HTML	156	0
Verify Homepage Text	156	0

Smoke Test [L9 - Jamstack with LEMP Stack (WSL2)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	4.03 ms (3.06 ms - 5.53 ms)	4.02 ms	4.69 ms	4.80 ms
req_waiting	2.86 ms (2.08 ms - 3.94 ms)	2.88 ms	3.28 ms	3.46 ms
req_connecting	0 s (0 s - 0 s)	0 s	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 129.10 µs)	717 ns	0 s	0 s
req_receiving	1.06 ms (510.80 µs - 2.18 ms)	1.14 ms	1.59 ms	1.65 ms
req_blocked	0 s (0 s - 0 s)	0 s	0 s	0 s
iteration_duration	1 s (1 s - 1.01 s)	1 s	1.01 s	1.01 s

Smoke Test [L9 - Jamstack with LEMP Stack (WSL2)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	180	2.973877/s
Data Sent/Received	16 kB / 21 MB	256 B / 348 kB

Smoke Test [L9 - Jamstack with LEMP Stack (WSL2)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures
Total Checks	540	0
Status is 200 (OK status)	180	0
Content type is text/HTML	180	0
Verify Homepage Text	180	0

Appendix G. Grafana k6 – Average-Load Test Results

Average-Load Test [L1 - WordPress with LAMP Stack (VM)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	2.26 s (159.61 ms - 3.18 s)	1.98 s	2.55 s	2.62 s
req_waiting	2.23 s (150.95 ms - 3.08 s)	1.95 s	2.52 s	2.59 s
req_connecting	0 s (0 s - 21 ms)	10.87 µs	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 2.63 ms)	2.7 µs	0 s	0 s
req_receiving	14.20 ms (4.38 ms - 717.74 ms)	27.52 ms	61.75 ms	94.97 ms
req_blocked	0 s (0 s - 21.18 ms)	12.27 µs	0 s	0 s
iteration_duration	3.26 s (1.15 s - 4.18 s)	2.98 s	3.56 s	3.62 s

Average-Load Test [L1 - WordPress with LAMP Stack (VM)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	70 483	29.366839/s
Data Sent/Received	5.7 MB / 8.3 GB	2.4 kB / 3.5 MB

Average-Load Test [L1 - WordPress with LAMP Stack (VM)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures
Total Checks	211 449	0
Status is 200 (OK status)	70483	0
Content type is text/HTML	70483	0
Verify Homepage Text	70483	0

Average-Load Test [L2 - WordPress with LAMP Stack (WSL2)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	255.10 ms (157.67 ms - 339.07 ms)	249.83 ms	266.10 ms	271.13 ms
req_waiting	253.62 ms (157.16 ms - 337.04 ms)	248.37 ms	264.58 ms	269.60 ms
req_connecting	0 s (0s - 6.81 ms)	4.3 µs	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 1.63 ms)	9.28 µs	0 s	0 s
req_receiving	1.56 ms (0 s - 26.97 ms)	1.45 ms	2.11 ms	2.24 ms
req_blocked	0 s (0 s - 6.81 ms)	7.02 µs	0 s	0 s
iteration_duration	1.25 s (1.16 s - 1.34 s)	1.25 s	1.27 s	1.27 s

Average-Load Test [L2 - WordPress with LAMP Stack (WSL2)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	167 526	69.769101/s
Data Sent/Received	14 MB / 20 GB	6.0 kB / 8.3 MB

Average-Load Test [L2 - WordPress with LAMP Stack (WSL2)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures
Total Checks	502 578	0
Status is 200 (OK status)	167 526	0
Content type is text/HTML	167 526	0
Verify Homepage Text	167 526	0

Average-Load Test [L4 - Jamstack with LAMP Stack (WSL2)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	17.56 ms (2.61 ms - 65.05 ms)	19.25 ms	32.46 ms	36.38 ms
req_waiting	8.58 ms (1.46 ms - 38.15 ms)	9.56 ms	15.51 ms	17.34 ms
req_connecting	0 s (0 s - 3.07 ms)	2.8 µs	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 3.15 ms)	3.72 µs	0 s	0 s
req_receiving	8.54 ms (502.20 µs - 47.99 ms)	9.68 ms	17.59 ms	20.80 ms
req_blocked	0 s (0 s - 4.65 ms)	4.06 µs	0 s	0 s
iteration_duration	1.02 s (1 s - 1.07 s)	1.02 s	1.03 s	1.04 s

Average-Load Test [L4 - Jamstack with LAMP Stack (WSL2)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	205 170	85.47/s
Data Sent/Received	18 MB / 24 GB	7.4 kB / 10 MB

Average-Load Test [L4 - Jamstack with LAMP Stack (WSL2)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures
Total Checks	615 510	0
Status is 200 (OK status)	205 170	0
Content type is text/HTML	205 170	0
Verify Homepage Text	205 170	0

Average-Load Test [L5 - Jamstack with LAMP Stack (VM)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	272.14 ms (4.73 ms - 757.20 ms)	298.41 ms	646.93 ms	668.89 ms
req_waiting	4.56 ms (0 s - 177.38 ms)	28.38 ms	81.52 ms	93.20 ms
req_connecting	0 s (0 s - 121.88 ms)	217.47 µs	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 1 ms)	2.73 µs	0 s	0 s
req_receiving	255.07 ms (3.82 ms - 750.46 ms)	270.02 ms	580.43 ms	600.42 ms
req_blocked	0 s (0 s - 121.88 ms)	218.32 µs	0 s	0 s
iteration_duration	1.27 s (1 s - 1.75 s)	1.29 s	1.64 s	1.66 s

Average-Load Test [L5 - Jamstack with LAMP Stack (VM)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	161 790	67.40/s
Data Sent/Received	13 MB / 19 GB	5.5 kB / 7.9 MB

Average-Load Test [L5 - Jamstack with LAMP Stack (VM)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures
Total Checks	486 370	0
Status is 200 (OK status)	161 790	0
Content type is text/HTML	161 790	0
Verify Homepage Text	161 790	0

Average-Load Test [L6 -WordPress with LEMP Stack (VM)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	2.24 s (160.07 ms - 3.43 s)	1.95 s	2.66 s	2.74 s
req_waiting	2.20 s (152.62 ms - 3.42 s)	1.91 s	2.60 s	2.68 s
req_connecting	0 s (0 s - 5.38 ms)	1.19 µs	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 1 ms)	1.3 µs	0 s	0 s
req_receiving	19.40 ms (4.13 ms - 780.99 ms)	46.21 ms	113.22 ms	161.93 ms
req_blocked	0 s (0s - 5.38 ms)	1.66 µs	0 s	0 s
iteration_duration	3.24 s (1.16 s - 4.44 s)	2.95 s	3.66 s	3.74 s

Average-Load Test [L6 -WordPress with LEMP Stack (VM)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	71 029	29.59/s
Data Sent/Received	5.8 MB / 8.4 GB	2.4 kB / 3.5 MB

Average-Load Test [L6 -WordPress with LEMP Stack (VM)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures
Total Checks	213 087	0
Status is 200 (OK status)	71 029	0
Content type is text/HTML	71 029	0
Verify Homepage Text	71 029	0

Average-Load Test [L7 - Jamstack with LEMP Stack (VM)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	7.59 ms (3.84 ms - 94.69 ms)	8.62 ms	12.10 ms	14.65 ms
req_waiting	549.10 µs (0 s - 42.15 ms)	904.02 µs	1.66 ms	2.10 ms
req_connecting	0 s (0 s - 3.49 ms)	884 ns	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 2.48 ms)	2.70 µs	0 s	0 s
req_receiving	6.79 ms (3.17 ms - 91.48 ms)	7.71 ms	10.85 ms	13.23 ms
req_blocked	0 s (0 s - 3.49 ms)	1.50 µs	0 s	0 s
iteration_duration	1 s (1 s - 1.09 s)	1 s	1.01 s	1.01 s

Average-Load Test [L7 - Jamstack with LEMP Stack (VM)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	208 253	86.74/s
Data Sent/Received	17 MB / 24 GB	7.0 kB / 10 MB

Average-Load Test [L7 - Jamstack with LEMP Stack (VM)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures
Total Checks	624 759	0
Status is 200 (OK status)	208 253	0
Content type is text/HTML	208 253	0
Verify Homepage Text	208 253	0

Average-Load Test [L8 - WordPress with LEMP Stack (WSL2)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	2.35 s (154.04 ms - 3.37 s)	2.07 s	2.36 s	2.37 s
req_waiting	2.35 s (153.23 ms - 3.37 s)	2.06 s	2.36 s	2.36 s
req_connecting	0 s (0 s - 703.70 µs)	293 ns	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 1.54 ms)	14.30 µs	0 s	0 s
req_receiving	1.08 ms (0 s - 8.21 ms)	1.35 ms	2.03 ms	2.15 ms
req_blocked	0 s (0 s - 1.5 ms)	4.1 µs	0 s	0 s
iteration_duration	3.35 s (1.15 s - 4.38 s)	3.07 s	3.36 s	3.37 s

Average-Load Test [L8 - WordPress with LEMP Stack (WSL2)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	68 371	28.49/s
Data Sent/Received	5.9 MB / 8.1 GB	2.4 kB / 3.4 MB

Average-Load Test [L8 - WordPress with LEMP Stack (WSL2)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures
Total Checks	205 113	0
Status is 200 (OK status)	68 371	0
Content type is text/HTML	68 371	0
Verify Homepage Text	68 371	0

Average-Load Test [L9 - Jamstack with LEMP Stack (WSL2)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	12.76 ms (2.10 ms - 57.35 ms)	13.75 ms	24.19 ms	26.91 ms
req_waiting	7.51 ms (1.50 ms - 41.36 ms)	8.73 ms	16.19 ms	18.30 ms
req_connecting	0 s (0 s - 658.80 µs)	329 ns	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 3.87 ms)	3.15 µs	0 s	0 s
req_receiving	3.81 ms (0 s - 33.82 ms)	5.01 ms	10.38 ms	12.30 ms
req_blocked	0 s (0 s - 1.27 ms)	1.36 µs	0 s	0 s
iteration_duration	1.01 s (1 s - 1.05 s)	1.01 s	1.02 s	1.02 s

Average-Load Test [L9 - Jamstack with LEMP Stack (WSL2)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	206 844	86.16/s
Data Sent/Received	18 MB / 24 GB	7.4 kB / 10 MB

Average-Load Test [L9 - Jamstack with LEMP Stack (WSL2)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures
Total Checks	620 532	0
Status is 200 (OK status)	206 844	0
Content type is text/HTML	206 844	0
Verify Homepage Text	206 844	0

Appendix H. Grafana k6 – Stress Test Results

Stress Test [L1 - WordPress with LAMP Stack (VM)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	4.03 s (505.7 µs - 6.70 s)	2.95 s	5.39 s	5.52 s
req_waiting	3.98 s (504.4 µs - 6.41 s)	2.93 s	5.35 s	5.47 s
req_connecting	0 s (0 s - 19.67 ms)	275.54 µs	836.70 µs	1.47 ms
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 560.37 ms)	185.93 µs	0 s	212.60 µs
req_receiving	7.99 ms (0 s - 1.25 s)	28.94 ms	104.31 ms	149.65 ms
req_blocked	0 s (0 s - 19.67 ms)	299.28 µs	921.50 µs	1.50 ms
iteration_duration	5.03 s (1 s - 7.70 s)	3.96 s	6.40 s	6.52 s

Stress Test [L1 - WordPress with LAMP Stack (VM)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	113 734	42.11/s
Data Sent/Received	9.2 MB / 9.4 GB	3.4 kB / 3.5 MB

Stress Test [L1 - WordPress with LAMP Stack (VM)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures	Success Rate (%)
Total Checks	271 746	69 456	79.64 %
Status is 200 (OK status)	79 006	34 728	69.47 %
Content type is text/HTML	113 734	0	100.00 %
Verify Homepage Text	79 006	34 728	69.47 %

Stress Test [L2 - WordPress with LAMP Stack (WSL2)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	1.29 s (2.10 ms - 2.37 s)	1.16 s	1.73 s	1.81 s
req_waiting	1.28 s (1.56 ms - 2.37 s)	1.16 s	1.73 s	1.81 s
req_connecting	0 s (0 s - 5.27 ms)	5.78 µs	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 4.07 ms)	11.06 µs	0 s	0 s
req_receiving	1.61 ms (0 s - 100.92 ms)	3.85 ms	9.49 ms	15.40 ms
req_blocked	0 s (0 s - 5.27 ms)	9.06 µs	0 s	0 s
iteration_duration	2.29 s (1 s - 3.37 s)	2.17 s	2.74 s	2.82 s

Stress Test [L2 - WordPress with LAMP Stack (WSL2)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	207 235	76.74/s
Data Sent/Received	18 MB / 25 GB	6.6 kB / 9.1 MB

Stress Test [L2 - WordPress with LAMP Stack (WSL2)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures	Success Rate (%)
Total Checks	620 435	1270	99.79 %
Status is 200 (OK status)	206 600	635	99.69 %
Content type is text/HTML	207 235	0	100.00 %
Verify Homepage Text	206 600	635	99.69 %

Stress Test [L4 - Jamstack with LAMP Stack (WSL2)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	19.69 ms (2.61 ms - 120.14 ms)	21.79 ms	36.69 ms	42.79 ms
req_waiting	9.54 ms (1.38 ms - 64.89 ms)	10.60 ms	17.55 ms	20.15 ms
req_connecting	0 s (0 s - 1.26 ms)	3.14 µs	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 1.04 ms)	3.69 µs	0 s	0 s
req_receiving	9.57 ms (503.70 µs - 95.08 ms)	11.18 ms	20.69 ms	25.65 ms
req_blocked	0 s (0 s - 1.26 ms)	4.46 µs	0 s	0 s
iteration_duration	1.02 s (1 s - 1.12 s)	1.02 s	1.04 s	1.04 s

Stress Test [L4 - Jamstack with LAMP Stack (WSL2)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	438 840	162.50/s
Data Sent/Received	38 MB / 51 GB	14 kB / 19 MB

Stress Test [L4 - Jamstack with LAMP Stack (WSL2)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures	Success Rate (%)
Total Checks	1 316 520	0	100.00 %
Status is 200 (OK status)	438 840	0	100.00 %
Content type is text/HTML	438 840	0	100.00 %
Verify Homepage Text	438 840	0	100.00 %

Stress Test [L5 - Jamstack with LAMP Stack (VM)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	1.17 s (4.77 ms - 1.55 s)	864.03 ms	1.33 s	1.35 s
req_waiting	70.68 ms (0 s - 282.87 ms)	74.90 ms	162.40 ms	178.35 ms
req_connecting	0 s (0 s - 259.61 ms)	653.40 µs	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 265.13 ms)	125.63 µs	0 s	0 s
req_receiving	1.08 s (3.92 ms - 1.5 s)	789 ms	1.2 s	1.23 s
req_blocked	0 s (0 s - 259.61 ms)	695.51 µs	0 s	0 s
iteration_duration	2.18 s (1 s - 2.55 s)	1.86 s	2.33 s	2.35 s

Stress Test [L5 - Jamstack with LAMP Stack (VM)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	241 411	89.39/s
Data Sent/Received	20 MB / 28 GB	7.2 kB / 11 MB

Stress Test [L5 - Jamstack with LAMP Stack (VM)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures	Success Rate (%)
Total Checks	724 233	0	100.00 %
Status is 200 (OK status)	241 411	0	100.00 %
Content type is text/HTML	241 411	0	100.00 %
Verify Homepage Text	241 411	0	100.00 %

Stress Test [L6 - WordPress with LEMP Stack (VM)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	4.67 s (505.40 µs - 11.49 s)	3.86 s	6.02 s	6.48 s
req_waiting	4.59 s (505.40 µs - 11.45 s)	3.78 s	5.88 s	6.33 s
req_connecting	0 s (0 s - 12.99 ms)	2.11 µs	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 976 µs)	1.65 µs	0 s	0 s
req_receiving	19.74 ms (0 s - 2.96 s)	84.37 ms	231.82 ms	374.13 ms
req_blocked	0 s (0 s - 13.64 ms)	2.64 µs	0 s	0 s
iteration_duration	5.68 s (1 s - 12.49 s)	4.86 s	7.02 s	7.48 s

Stress Test [L6 - WordPress with LEMP Stack (VM)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	92 527	34.26/s
Data Sent/Received	7.5 MB / 9.6 GB	2.8 kB / 3.5 MB

Stress Test [L6 - WordPress with LEMP Stack (VM)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures	Success Rate (%)
Total Checks	254 083	23 498	91.53 %
Status is 200 (OK status)	80 778	11 749	87.30 %
Content type is text/HTML	92 527	0	100.00 %
Verify Homepage Text	80 778	11 749	87.30 %

Stress Test [L7 - Jamstack with LEMP Stack (VM)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	562.67 ms (3.67 ms - 947.55 ms)	409.24 ms	802.80 ms	820.06 ms
req_waiting	6.95 ms (0 s - 437.09 ms)	13.13 ms	28.99 ms	37.56 ms
req_connecting	0 s (0 s - 36.67 ms)	7.26 µs	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 983.20 µs)	1.81 µs	0 s	0 s
req_receiving	537.86 ms (1.73 ms - 894.7 ms)	396.11 ms	780.87 ms	797.86 ms
req_blocked	0 s (0 s - 36.67 ms)	7.68 µs	0 s	0 s
iteration_duration	1.56 s (1 s - 1.94 s)	1.40 s	1.80 s	1.82 s

Stress Test [L7 - Jamstack with LEMP Stack (VM)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	319 408	118.26/s
Data Sent/Received	26 MB / 37 GB	9.6 kB / 14 MB

Stress Test [L7 - Jamstack with LEMP Stack (VM)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures	Success Rate (%)
Total Checks	958 224	0	100.00 %
Status is 200 (OK status)	319 408	0	100.00 %
Content type is text/HTML	319 408	0	100.00 %
Verify Homepage Text	319 408	0	100.00 %

Stress Test [L8 - WordPress with LEMP Stack (WSL2)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	5.68 s (154.17 ms - 5.75 s)	4.74 s	5.70 s	5.71 s
req_waiting	5.68 s (152.60 ms - 5.75 s)	4.74 s	5.70 s	5.70 s
req_connecting	0 s (0 s - 616.60 µs)	464 ns	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 1.51 ms)	13.48 µs	0 s	0 s
req_receiving	1.11 ms (0 s - 10.84 ms)	1.39 ms	2.11 ms	2.20 ms
req_blocked	0 s (0 s - 1.73 ms)	4.44 µs	0 s	0 s
iteration_duration	6.68 s (1.15 s - 6.75 s)	5.75 s	6.70 s	6.71 s

Stress Test [L8 - WordPress with LEMP Stack (WSL2)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	78 338	29.01/s
Data Sent/Received	6.7 MB / 9.3 GB	2.5 kB / 3.4 MB

Stress Test [L8 - WordPress with LEMP Stack (WSL2)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures	Success Rate (%)
Total Checks	235 014	0	100.00 %
Status is 200 (OK status)	78 338	0	100.00 %
Content type is text/HTML	78 338	0	100.00 %
Verify Homepage Text	78 338	0	100.00 %

Stress Test [L9 - Jamstack with LEMP Stack (WSL2)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	13.94 ms (2.06 ms - 79.26 ms)	14.75 ms	24.31 ms	27.52 ms
req_waiting	8.09 ms (1.03 ms - 40.66 ms)	9.19 ms	16.16 ms	18.22 ms
req_connecting	0 s (0 s - 1.05 ms)	333 ns	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 3.69 ms)	3.12 µs	0 s	0 s
req_receiving	4.79 ms (0 s - 54.03 ms)	5.55 ms	10.65 ms	12.88 ms
req_blocked	0 s (0 s - 1.07 ms)	1.28 µs	0 s	0 s
iteration_duration	1.01 s (1 s - 1.07 s)	1.01 s	1.02 s	1.02 s

Stress Test [L9 - Jamstack with LEMP Stack (WSL2)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	442 551	163.88/s
Data Sent/Received	38 MB / 52 GB	14 kB / 19 MB

Stress Test [L9 - Jamstack with LEMP Stack (WSL2)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures	Success Rate (%)
Total Checks	1 327 653	0	100.00 %
Status is 200 (OK status)	442 551	0	100.00 %
Content type is text/HTML	442 551	0	100.00 %
Verify Homepage Text	442 551	0	100.00 %

Appendix I. Grafana k6 – Spike Test Results

Spike Test [L1 - WordPress with LAMP Stack (VM)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	7.91 s (1.33 ms - 45.42 s)	8.32 s	14.33 s	17.78 s
req_waiting	7.90 s (1.12 ms - 45.41 s)	8.30 s	14.30 s	17.71 s
req_connecting	1.41 ms (0 s - 121.69 ms)	2.63 ms	6.64 ms	9.24 ms
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 2.58 s)	2.64 ms	104.60 µs	508.05 µs
req_receiving	0 s (0 s - 6.2 s)	16.04 ms	26.28 ms	87.98 ms
req_blocked	1.51 ms (0 s - 121.69 ms)	2.69 ms	6.74 ms	9.29 ms
iteration_duration	8.91 s (1 s - 46.43 s)	9.32 s	15.33 s	18.78 s

Spike Test [L1 - WordPress with LAMP Stack (VM)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	20 570	112.88/s
Data Sent/Received	1.7 MB / 566 MB	9.2 kB / 3.1 MB

Spike Test [L1 - WordPress with LAMP Stack (VM)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures	Success Rate (%)
Total Checks	29 386	32 324	47.62 %
Status is 200 (OK status)	4 408	16 162	21.43 %
Content type is text/HTML	20 570	0	100.00 %
Verify Homepage Text	4 408	16 162	21.43 %

Spike Test [L2 - WordPress with LAMP Stack (WSL2)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	0 s (0 s - 16.96 s)	785.42 ms	2.57 s	3.73 s
req_waiting	148.31 ms (0 s - 16.94 s)	781.75 ms	2.56 s	3.72 s
req_connecting	1.57 ms (0 s - 2.77 s)	278.23 ms	1.20 s	1.71 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0s - 37.21 ms)	47.23 µs	0 s	518.40 µs
req_receiving	505.40 µs (0s - 886.22 ms)	3.62 ms	4.83 ms	14.26 ms
req_blocked	1.58 ms (0 s - 2.77 s)	278.27 ms	1.20 s	1.71 s
iteration_duration	2.44 s (47.71 ms - 20.19 s)	2.49 s	3.98 s	5.36 s

Spike Test [L2 - WordPress with LAMP Stack (WSL2)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	73 413	403.52/s
Data Sent/Received	4.8 MB / 1.4 GB	27 kB / 7.9 MB

Spike Test [L2 - WordPress with LAMP Stack (WSL2)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures	Success Rate (%)
Total Checks	78 541	124 520	38.68 %
Status is 200 (OK status)	11 153	62 260	15.19 %
Content type is text/HTML	56 235	17 178	76.60 %
Verify Homepage Text	11 153	45 082	19.83 %

Spike Test [L4 - Jamstack with LAMP Stack (WSL2)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	21.22 ms (2.63 ms - 1 m 0 s)	3.63 s	54.88 ms	49.62 s
req_waiting	10.09 ms (1.53 ms - 1 m 0 s)	10.09 ms	25.10 ms	49.60 s
req_connecting	0 s (0 s - 3.11 ms)	24.15 µs	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 588.40 µs)	8.15 µs	0 s	0 s
req_receiving	8.81 ms (0 s - 78.41 ms)	10.88 ms	22.73 ms	28.25 ms
req_blocked	0 s (0 s - 3.11 ms)	29.11 µs	0 s	504 µs
iteration_duration	1.02 s (1 s - 1 m 0 s)	4.58 s	1.05 s	50.63 s

Spike Test [L4 - Jamstack with LAMP Stack (WSL2)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	40 987	224.89/s
Data Sent/Received	3.6 MB / 4.6 GB	20 kB / 25 MB

Spike Test [L4 - Jamstack with LAMP Stack (WSL2)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures	Success Rate (%)
Total Checks	117 414	3 698	96.95 %
Status is 200 (OK status)	39 138	1 849	95.49 %
Content type is text/HTML	39 138	1 849	95.49 %
Verify Homepage Text	39 138	0	100.00 %

Spike Test [L5 - Jamstack with LAMP Stack (VM)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	6.15 s (0 s - 1 m 0 s)	8.53 s	17.03 s	28.98 s
req_waiting	163.60 ms (0 s - 3.89 s)	209.87 ms	600.57 ms	784.91 ms
req_connecting	160.41 ms (0 s - 16.23 s)	329.23 ms	1.16 s	1.19 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 1.60 s)	6.97 ms	0 s	96.49 µs
req_receiving	5.88 s (0 s - 59.69 s)	8.31 s	16.61 s	28.59 s
req_blocked	162.07 s (0 s - 16.23 s)	334.25 ms	1.17 s	1.19 s
iteration_duration	7.58 s (1 s - 1 m 0 s)	9.81 s	18.69 s	31.19 s

Spike Test [L5 - Jamstack with LAMP Stack (VM)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	19 323	106.93/s
Data Sent/Received	1.6 MB / 2.2 GB	8.8 kB / 12 MB

Spike Test [L5 - Jamstack with LAMP Stack (VM)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures	Success Rate (%)
Total Checks	55 188	1 854	96.75 %
Status is 200 (OK status)	18 396	927	95.20 %
Content type is text/HTML	18 396	927	95.20 %
Verify Homepage Text	18 396	0	100.00 %

Spike Test [L6 - WordPress with LEMP Stack (VM)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	1.26 s (0 s - 33.38 s)	3.27 s	8.22 s	16.59 s
req_waiting	1.26 s (0 s - 32.89 s)	3.25 s	8.13 s	16.50 s
req_connecting	0 s (0 s - 11.58 ms)	49.51 µs	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 988.70 µs)	3.78 µs	0 s	0 s
req_receiving	0 s (0 s - 14.21 ms)	15.22 ms	14.21 ms	90.09 ms
req_blocked	0 s (0 s - 11.74 ms)	52.70 µs	0 s	0 s
iteration_duration	2.26 s (1 s - 34.38 s)	4.27 s	9.22 s	17.59 s

Spike Test [L6 - WordPress with LEMP Stack (VM)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	44 077	243.47/s
Data Sent/Received	3.6 MB / 618 MB	20 kB / 2.4 MB

Spike Test [L6 - WordPress with LEMP Stack (VM)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures	Success Rate (%)
Total Checks	54 027	78 204	40.86 %
Status is 200 (OK status)	4 975	39 102	11.29 %
Content type is text/HTML	44 077	0	100.00 %
Verify Homepage Text	4 975	39 102	11.29 %

Spike Test [L7 - Jamstack with LEMP Stack (VM)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	3.97 s (4.94 ms - 1 m 0 s)	5.72 s	11.61 s	15.92 s
req_waiting	106.83 ms (0 s - 2.49 s)	165.17 ms	412.21 ms	610.08 ms
req_connecting	0 s (0 s - 7.11 s)	23.74 ms	0 s	99.16 ms
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 1.03 ms)	4.88 µs	0 s	0 s
req_receiving	3.82 s (4.21 ms - 59.91 s)	5.56 s	11.31 s	15.55 s
req_blocked	0 s (0 s - 7.11 s)	23.75 ms	0 s	99.17 ms
iteration_duration	5.02 s (1 s - 1 m 0 s)	6.73 s	12.66 s	16.93 s

Spike Test [L7 - Jamstack with LEMP Stack (VM)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	27 837	153.85/s
Data Sent/Received	2.3 MB / 3.2 GB	13 kB / 18 MB

Spike Test [L7 - Jamstack with LEMP Stack (VM)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures	Success Rate (%)
Total Checks	81 690	1 214	98.54 %
Status is 200 (OK status)	27 230	607	97.82 %
Content type is text/HTML	27 230	607	97.82 %
Verify Homepage Text	27 230	0	100.00 %

Spike Test [L8 - WordPress with LEMP Stack (WSL2)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	1.25 ms (0 s - 18.33 s)	800.02 ms	2.55 ms	2.03 s
req_waiting	1.18 ms (0 s - 18.32 s)	799.91 ms	2.49 ms	2.03 s
req_connecting	0 s (0 s - 26.84 ms)	6.79 µs	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 2.76 ms)	7.61 µs	0 s	0 s
req_receiving	0 s (0 s - 48.16 ms)	104.36 µs	507.70 µs	1 ms
req_blocked	0 s (0 s - 26.84 ms)	10 µs	0 s	0 s
iteration_duration	1 s (1 s - 19.33 s)	1.80 s	1.01 s	3.03 s

Spike Test [L8 - WordPress with LEMP Stack (WSL2)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	106 048	554.74/s
Data Sent/Received	9.1 MB / 682 MB	48 kB / 3.6 MB

Spike Test [L8 - WordPress with LEMP Stack (WSL2)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures	Success Rate (%)
Total Checks	117 046	201 098	36.79 %
Status is 200 (OK status)	5 499	100 549	5.19 %
Content type is text/HTML	106 048	0	100.00 %
Verify Homepage Text	5 499	100 549	5.19 %

Spike Test [L9 - Jamstack with LEMP Stack (WSL2)] [Requests] (Whitley, 2023a)

Request Metrics	Median (Min; Max)	Mean	P90	P95
req_duration	27.08 ms (2.08 ms - 1 m 0 s)	937.17 ms	286.73 ms	371.26 ms
req_waiting	14.43 ms (1.03 ms - 1 m 0 s)	887.25 ms	106.93 ms	147.56 ms
req_connecting	0 s (0 s - 1.5 ms)	7.97 µs	0 s	0 s
req_tls_handshaking	0 s (0 s - 0 s)	0 s	0 s	0 s
req_sending	0 s (0 s - 1 ms)	4.4 µs	0 s	0 s
req_receiving	10.59 ms (0 s - 59.77 ms)	49.91 ms	165.24 ms	216.56 ms
req_blocked	0 s (0 s - 2.1 ms)	9.86 µs	0 s	0 s
iteration_duration	1.02 s (1 s - 1 m 0 s)	1.92 s	1.28 s	1.37 s

Spike Test [L9 - Jamstack with LEMP Stack (WSL2)] [Other Stats] (Whitley, 2023a)

Other Stats	Total	Rate (per second)
Iterations/Requests	94 001	447.62/s
Data Sent/Received	8.2 MB / 11 GB	39 kB / 52 MB

Spike Test [L9 - Jamstack with LEMP Stack (WSL2)] [Checks] (Whitley, 2023a)

Checks	Passes	Failures	Success Rate (%)
Total Checks	277 995	2 672	99.05 %
Status is 200 (OK status)	92 665	1 336	98.58 %
Content type is text/HTML	92 665	1 336	98.58 %
Verify Homepage Text	92 665	0	100.00 %