Wrocław University of Science and Technology
Faculty of Information and Communication Technology

Field of study: **CS**

MASTER THESIS

Web services performance testing with the use of CMS and CDN systems.

Dominik Ilski

Supervisor
Dr inż. Jakub Długosz

Keywords: Web testing, CDN, CMS

WROCŁAW 2023

ABSTRACT

This thesis presents a comprehensive performance analysis of Cloudflare Pages and Vercel, two popular website hosting platforms. The study focuses on the user experience by examining key performance metrics such as Cumulative Layout Shift (CLS), Speed Index, and Time to First Byte (TTFB). The data was collected using Lighthouse and spanned across three regions: Asia, Europe, and the United States.

The results suggest subtle performance differences between the two platforms. For basic websites, Cloudflare Pages and Vercel exhibited comparable performance. However, for more complex, heavy websites, Vercel displayed better metrics. Interestingly, during anomaly events, Cloudflare Pages demonstrated more stability, indicating potential robustness to such occurrences.

Websites were created using JAMstack architecture and also because of that CMS service and static site generator.

The study also uncovers regional disparities in performance, with Asia generally outperforming the other regions by approximately 10%. These observations point to the need for further research to understand the underlying factors contributing to such variations.

This study contributes valuable insights into the performance dynamics of Cloudflare Pages and Vercel and lays the groundwork for future research in this domain. The findings may assist website owners in making informed decisions when choosing their hosting platform.

STRESZCZENIE

Niniejsza praca przedstawia kompleksową analizę wydajności Cloudflare Pages i Vercel, dwóch popularnych platform hostingowych. Badanie koncentruje się na doświadczeniu użytkownika, badając kluczowe wskaźniki wydajności, takie jak skumulowana zmiana układu (CLS), indeks prędkości i czas do pierwszego bajtu (TTFB). Dane zostały zebrane za pomocą Lighthouse i obejmowały trzy regiony: Azję, Europę i Stany Zjednoczone.

Wyniki sugerują subtelne różnice w wydajności między dwiema platformami. W przypadku podstawowych stron internetowych Cloudflare Pages i Vercel wykazywały porównywalną wydajność. Jednak w przypadku bardziej złożonych, ciężkich witryn Vercel

wyświetlał lepsze wskaźniki. Co ciekawe, podczas zdarzeń związanych z anomaliami strony Cloudflare wykazywały większą stabilność, co wskazuje na potencjalną odporność na takie zdarzenia.

Strony powstały w oparciu o architekturę JAMstack, a także dzięki temu serwisowi CMS i generatorowi statycznych stron.

Badanie ujawnia również regionalne dysproporcje w wynikach, przy czym Azja generalnie przewyższa inne regiony o około 10%. Obserwacje te wskazują na potrzebę dalszych badań w celu zrozumienia podstawowych czynników przyczyniających się do takich różnic.

Badanie to dostarcza cennych informacji na temat dynamiki wydajności Cloudflare Pages i Vercel oraz kładzie podwaliny pod przyszłe badania w tej dziedzinie. Odkrycia mogą pomóc właścicielom stron internetowych w podejmowaniu świadomych decyzji przy wyborze platformy hostingowej.

SPIS TREŚCI

INTRODUCTION

Web services have become an integral part of modern digital infrastructure, serving as the backbone for numerous applications across diverse sectors. The performance of these services significantly impacts user experience, influencing key metrics such as user engagement, satisfaction, and ultimately, the success of the digital service. Performance is particularly crucial for content-heavy services, where the delivery speed of static and dynamic content can greatly affect the perceived quality of the service.

Content Management Systems (CMS) and Jamstack have long served as essential tools for web content management and development, facilitating the creation, management, and modification of digital content. More recently, Content Delivery Networks (CDN) have emerged as an increasingly popular solution to enhance the delivery of web content to users as they are greatly suited for jamstack websites. By distributing content across various nodes in different geographical locations, CDNs aim to reduce latency and improve site load times, thereby enhancing user experience and service performance.

Despite the rising prominence of CDN systems, there is a need for empirical research to assess the real-world benefits and potential limitations of these systems. This is particularly relevant in comparison to traditional web hosting services, which continue to play a significant role in web content delivery. The motivation of this research lies in addressing this gap, aiming to provide a comprehensive evaluation of web services performance with the use of CMS and CDN systems.

The scope of this research encompasses web services performance in three major regions: the United States, Europe, and Asia. This geographical focus ensures a broad coverage of the major digital markets, providing a comprehensive view of CDN performance across diverse network infrastructures and user bases.

The thesis proposes that the use of CDN systems can significantly improve web services performance, and aims to find suitable solutions. To better showcase the gains of CDNs systems test comparing CDN "hosted"website and traditionally tested website will be conducted, but this will not be part of the thesis of this work, as there is plenty of literature explaining this matter in detail that will be discussed in the Research chapter. Through rigorous testing and analysis, this research aims to either validate or challenge this proposition, thereby contributing to the body of knowledge on effective web content delivery strategies. The subsequent chapters of this thesis provide a detailed account of the research methodology, findings, and implications for both theory and practice.

THE AIM OF THE THESIS

The primary goal of this thesis is to evaluate the performance of various Content Delivery Networks (CDNs) and compare them to help evaluate which solution to choose according to the use case. The study will focus on the delivery of text and image content from static websites generated using Gatsby [9], a popular static site generator, and a Content Management System (CMS) Flotiq [10].

The performance of the web services will be measured using Google Lighthouse [15], a comprehensive tool for assessing the quality of web pages. Serverless functions or servers deployed in different regions, namely North America, Europe, and Asia, will be utilized to carry out these performance tests.

In essence, the research aims to determine if the use of CDN systems can significantly improve web services performance, and then try to indicate the best CDN in terms of latency and metrics provided by a lighthouse which are going to be explained later in this work.

The results of this research are anticipated to contribute to the field of web services by offering insights into the effectiveness of various CDN systems in comparison with traditional web hosting. It aims to guide decision-making processes for businesses and individuals looking to optimize their web service performance, particularly when deploying JAMstack websites.

While the primary focus is on latency, it is expected that the findings of this research may stimulate further investigations into other aspects of web service performance, such as throughput and availability, thus contributing to a more comprehensive understanding of the subject.

THE SCOPE OF THE THESIS

This research centers around the performance evaluation of various Content Delivery Networks (CDNs) measured by "user-friendly"metrics. The scope includes the analysis of static websites developed with Gatsby, a widely-used static site generator, and hosted on the CMS platform.

The geographical regions under consideration for this study are North America, Europe, and Asia, chosen for their diverse network infrastructures and substantial digital markets. These regions also represent a significant proportion of the global user base, making the findings relevant on a broader scale.

The structure of this thesis is as follows:

- **Abstract:** A concise summary of the thesis in both Polish and English.
- **Table of Contents:** An overview of the chapters and sections in the thesis.

- **Introduction:** This section provides the research motivation, aim, scope, and thesis structure.
- **Problem Background:** Detailed discussion on cloud computing service models, CDN and CMS as SaaS, web services architecture, and performance testing.
- **Research:** Description of the research methodology, including planning, design, deployment, and tests.
- **Analysis:** A analysis of the gathered data, with conclusions.
- **Conclusions:** A summary of the findings and their implications.
- **Future Work:** Discussion of potential future research directions.
- **Bibliography:** A list of the sources cited in the thesis.
- **List of Tables and Figures:** A reference list for the tables and figures included in the thesis.
- **Appendix:** Additional information such as full listings, configuration settings, full diagrams, etc.

The research is expected to provide valuable insights into the effectiveness of various CDN systems, helping to inform decisions related to web service optimization, particularly for Jamstack websites. The thesis may also stimulate further research into other aspects of web service performance.

1. PROBLEM BACKGROUND

This chapter provides a detailed exploration of the main concepts pertinent to the research problem. It sets the stage for the research by delving into the foundational knowledge necessary to comprehend the context and significance of the research problem. This section is important as it will explain what kind of services are going to be used and how are they going to contribute to the solution being tested.

1.1. CLOUD COMPUTING SERVICE MODELS

This section will introduce the three primary cloud computing service models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). Each model will be explained, and their respective advantages and disadvantages will be discussed.

1.1.1. IaaS

Infrastructure as a Service (IaaS) is one of the three primary cloud computing service models[40]. IaaS provides users with virtualized computing resources over the internet. This model is essentially an outsourcing method that is used for hosting websites, web apps, and other IT infrastructure components such as storage, servers, and network.

IaaS platforms provide highly scalable resources that can be adjusted on-demand. This makes it an ideal option for temporary, experimental, or changing workloads that are unpredictable. The cost of deploying IaaS is typically variable depending on the usage of the resources.

Users have direct control of their infrastructure, and they are responsible for managing aspects such as runtime, data, middleware, and operating systems. The cloud provider, on the other hand, manages the underlying cloud infrastructure, including the physical hardware and virtualization environment [28].

1.1.2. PaaS

Platform as a Service (PaaS) is another core model of cloud computing services. The PaaS model provides a platform that enables developers to build, test, deploy, and manage the development of software applications without the complexity of building and maintaining the infrastructure typically associated with the process [45].

In the PaaS model, the cloud provider manages the underlying infrastructure, including the network, servers, operating systems, and storage. Meanwhile, users manage the applications and services they develop, and in some cases, the configuration settings of the application-hosting environment [45].

PaaS solutions serve to abstract much of the need for managing hardware and operating systems, allowing developers to focus more on the coding and design of their applications. This can significantly speed up the development process and reduce the costs associated with it.

PaaS platforms typically include features for application design, application development, testing, and deployment as well as services such as team collaboration, web service integration, and database integration. It's a suitable choice for developers who want to manage the application and leave the infrastructure management to the provider.

1.1.3. SaaS

Software as a Service (SaaS) is the third core model of cloud computing services. SaaS provides users with access to the provider's cloud-based software applications over the Internet. These applications are typically accessed through a web browser, with the data and software hosted on the provider's servers [30].

In the SaaS model, the cloud provider manages the underlying infrastructure, including network, servers, operating systems, storage, and even individual software capabilities. The users are only responsible for using the software application and, in some cases, configuring the application settings within the software itself [30].

SaaS solutions allow users to outsource the entire infrastructure and software management to the provider, thus minimizing the effort and cost associated with software maintenance, patching, and upgrades. SaaS platforms usually offer software applications for various business functions such as email, Customer Relationship Management (CRM), Enterprise Resource Planning (ERP), Human Resources Management (HRM), and more.

It's a suitable choice for businesses that want to start using software applications quickly and at a minimal upfront cost. However, the level of customization and control over the software may be limited compared to the other cloud computing models.

1.2. CDN AND CMS AS SAAS

This section will discuss Content Delivery Networks (CDNs) and Content Management Systems (CMS) as instances of the SaaS model. The working principles, and benefits.

1.2.1. CDN

A Content Delivery Network (CDN) is a distributed network of servers that work together to provide fast delivery of internet content. A CDN allows for the quick transfer of assets needed for loading Internet content including HTML pages, javascript files, stylesheets, images, and videos [29].

The primary function of a CDN is to improve web performance by reducing the time taken to transmit content and rich media to the user's browser. This is achieved by caching content at various points in a network, known as Point of Presence (PoP), and delivering the content to a user from the nearest PoP [29].

CDNs are also valuable for protecting websites against malicious attacks such as Distributed Denial of Service (DDoS) attacks [41].

Fundamentally, the goal of a CDN is to serve content to end-users with high availability and high performance. As the volume of information being disseminated online grows, and as businesses become increasingly global, CDNs are becoming more important for distributing content in a fast and reliable manner.

While the benefits of using CDNs are clear, it's important to also consider the trade-offs. One of the main considerations involves the increase in DNS lookup time.

DNS lookup is the process where a domain name (the human-friendly URL that is typed into the browser) is converted into an IP address that corresponds to the server that hosts the website content. When a CDN is involved, an extra DNS lookup is usually needed to direct the request to the nearest edge server. This extra step may introduce a slight delay in the initial response time, especially if the DNS server handling the request is slow or overloaded.

However, this is generally a minimal delay, and the benefits of serving content from a geographically closer location typically outweigh the costs of the extra DNS lookup. In many cases, modern browsers and DNS servers can cache the results of DNS lookups, which means that this extra lookup only happens once, on the first request, and subsequent requests do not have this additional delay.

1.2.2. CMS

A Content Management System (CMS) is a software application that allows users to create, manage, and modify digital content [27]. CMSs are typically used for enterprise content management (ECM) and web content management (WCM). An ECM facilitates collaboration in the workplace by integrating document management, digital asset management, and records retention functionalities, and providing end users with role-based access to the organization's digital assets.

CMSs are designed to reduce the need for hand coding, making it easier for users without technical expertise to create and manage their own content [27]. They typically

provide a user-friendly interface where users can create and edit content, set up workflows for content creation and publication, and manage the organization and display of content.

By utilizing a CMS, users can focus more on the content and design of their websites or applications, without needing to worry as much about the technical aspects of web development, as they have become more popular over the years [42], especially with the use of JAMstack architecture.

1.3. WEB SERVICES

This section will delve into the architecture of web services and the methods of access, such as REST and SOAP. It will also discuss the performance of web services and how performance testing is typically conducted.

1.3.1. Web Service Architecture

Web services architecture is a key component of modern, distributed computing. It serves as a framework for data exchange between disparate systems over the internet. The architecture relies on a set of standards, protocols, and specifications to ensure seamless interaction and interoperability between different software applications, regardless of the programming language or platform used in their development [39].

The basic web services architecture comprises three roles, namely the Service Provider, the Service Requestor, and the Service Registry, and operates through four basic operations - Publish, Find, Bind, and Invoke.

- **Service Provider:** The service provider creates the web service and makes its interface and implementation accessible to clients over the internet.
- **Service Requestor:** The service requestor, or the client application, is the user of the web service. The requestor locates entries in the broker's registry using various find operations and binds to the service provider in order to invoke one of its web services.
- **Service Registry:** The service registry, or broker, is a searchable database that contains information about available web services. The service provider publishes the service description into the service registry allowing other applications to discover them [39].

When it comes to operations:

- **Publish:** The service provider publishes a service's description to the service registry. This description details what the service does, where it resides, and how it can be invoked.
- **Find:** The service requestor queries the service registry to locate the desired web service.

- **Bind:** The service requestor uses the service description obtained from the service registry to bind to the service implementation and invoke the web service.
- **Invoke:** The service requestor sends a service request to the service provider. The provider processes the request and returns the service results to the requestor.

The web service architecture is built around communication via messages, typically written in XML format for platform-neutral data exchange. The messages are transported via common internet protocols such as HTTP, SMTP, or FTP [39].

The primary web service protocols include SOAP, REST, and JSON-RPC or XML-RPC, although the latter two are often classified separately from traditional web services. Of these, REST and SOAP are the most common, and they each offer different approaches to creating web services, with their unique advantages and trade-offs This architecture enables developers to build scalable and flexible systems, leveraging the benefits of reuse and interoperability of web services.

1.3.2. REST and SOAP

Web services primarily use two different methods of access: REST and SOAP. These methods, or protocols, define how web services communicate with each other and with clients.

1.3.2.1. REST

Representational State Transfer (REST) is an architectural style for designing networked applications. REST relies on a stateless, client-server communication protocol, almost always HTTP. In REST, a client makes a request to a server, and the server returns a response. This request/response cycle is stateless. That is, the server does not retain any information about the client between requests[49].

RESTful web services use HTTP methods explicitly and include the following constraints:

- Stateless: Each request from a client must contain all the information needed by the server to fulfill the request. The server cannot store the information provided by the client in one request and use it in another request.
- Client-Server: The client and the server are separate entities that communicate over a network. The client is responsible for the user interface and user experience, while the server provides the data and handles most of the business logic.
- Cacheable: Clients can cache server responses to improve performance.
- Layered System: The architecture allows an application to be composed of layered components, each providing specific functionality.

1.3.2.2. SOAP

Simple Object Access Protocol (SOAP) is a messaging protocol specification for exchanging structured information in the implementation of web services in computer networks. It uses XML for its message format, which allows for programmatic communication between applications over a network, regardless of the platform, language, or object model used by the applications [49].

Unlike REST, which is stateless, SOAP supports both stateless and stateful operations. A SOAP message is an ordinary XML document containing the following elements:

- An Envelope element that identifies the XML document as a SOAP message.
- A Header element that contains header information.
- A Body element that contains call and response information.
- A Fault element that provides information about errors that occurred while processing the message.

SOAP can be used for messaging, RPC (Remote Procedure Calls), or to build web services. It supports different types of communication such as one-way messaging, request/response, and duplex message exchange patterns.

Despite their differences, both REST and SOAP aim to achieve the same goal - to provide a means for different applications or software components to communicate and share data with one another. The choice between using REST or SOAP depends on the specific requirements and constraints of the project at hand.

1.3.3. JAMstack Architecture

JAMstack is a modern web development architecture based on client-side JavaScript, reusable APIs, and prebuilt Markup [43]. The term JAMstack stands for JavaScript, APIs, and Markup, which are the three core components of this architecture.

JAMstack's strength lies in its ability to serve pre-rendered files and static assets, enabling high performance, robust scalability, and greater security [48]. By pre-rendering files, the server load is significantly reduced as most of the computing work is shifted to the client side. The use of APIs allows JAMstack applications to be decoupled from the backend, promoting modularity and flexibility.

However, JAMstack has its own set of limitations. The architecture might not be suitable for applications that require dynamic server-side functionalities, as all pages are pre-rendered at build time. Furthermore, sites with numerous pages might face extended build times due to the nature of static site generation [48].

Static site generators, such as Gatsby, play a crucial role in the JAMstack architecture. Gatsby transforms source data, which could be from a file system, APIs, or a Content Management System (CMS), into a static website [9]. This results in a highly optimized

site that can be served over a Content Delivery Network (CDN), maximizing performance and scalability.

Content Management Systems (CMS) like Flotiq functions as data sources for static site generators. Flotiq provides a headless CMS, which means it only provides the backend content management functionality through an API, separating it from the frontend presentation layer [10]. This aligns with the decoupled nature of JAMstack, allowing developers to choose their preferred technologies for the frontend while still enabling efficient content management.

In conclusion, JAMstack represents a paradigm shift in web development, promoting a decoupled architecture for better performance, scalability, and security. Static site generators and headless CMS are integral components of this architecture, serving as tools for content creation and optimization.

1.3.4. Website Performance and Performance Testing

Website performance is a key factor in ensuring a satisfactory user experience. A major factor in this performance is the efficiency with which web content is delivered to the end user. Traditional web hosting serves all content from a single location, which can lead to inefficiencies and delays, especially when users are geographically distant from the hosting server [37].

A modern solution to this problem is the use of a Content Delivery Network (CDN). A CDN is a global network of "edge servers"which distribute the content from multiple locations, enabling faster and more reliable access for users around the world [47]. In a CDN, the majority of the static and dynamic content is served from the location closest to the user, reducing download times significantly compared to traditional web hosting [38].

1.3.4.1. Performance Testing

Performance testing, particularly for websites, is a crucial aspect of assessing the effectiveness of a CDN. For instance, one case study revealed that deploying a CDN on a site hosted by DigitalOcean resulted in a decrease in website load time by 38.24% [3]. Similarly, using Google's CDN for font delivery instead of local web font hosting resulted in a 50 ms decrease in load time [3]. These examples illustrate the significant impact that a CDN can have on website performance.

1.3.4.2. Website Performance Metrics

It's important to measure and monitor a set of key performance metrics to ensure that website of interest is delivering the best possible user experience. This thesis will look into standards created by Google already mentioned in this work i.e., lighthouse [15].

1.3.4.3. Google Lighthouse

Google Lighthouse is an automated, open-source tool for improving the quality of web pages. It can be run against any web page, public, or requiring authentication. Lighthouse has audits for performance, accessibility, progressive web apps, and SEO [15].

In the context of performance testing, Lighthouse evaluates multiple aspects that are in the detail presented in ch. 1.3.4.4 such as the time it takes to load the first contentful paint (FCP – the point at which the first text or image is painted), time to interactive (TTI – time since the page is fully interactive), speed index (how quickly the contents of a page are visibly populated).

The Performance Score provided by Lighthouse is a weighted average of the metric scores. It gives an overall idea of the page performance as perceived by a user. In the case of CDNs, Lighthouse can be used to measure the impact of using a CDN on the website's performance score. By comparing the performance scores with and without using a CDN, the effectiveness of the CDN can be acessed. Lighthouse also provides opportunities and diagnostics for more insights about the page's performance. These can be used to identify potential improvements to the performance. Given its comprehensiveness and ease of use, Google Lighthouse is a critical tool for performance testing, providing a multitude of valuable insights.

1.3.4.4. Tested parameters

Parameters that are going to be analyzed in this thesis:

1. First Meaningful Paint
2. First Contentful Paint
3. Cumulative Layout Shift
4. Potential FID
5. Speed Index
6. Time to Interactive
7. Time to first byte

Here are explanations of evaluated parameters:

1. **First Meaningful Paint**
   The time required to render the first content in the DOM - it could be a header, text, or anything visible to the user. According to Google Lighthouse, it should fall within the following ranges:
   - from 0 to 2s - good,
   - from 2s to 4s - average,
   - over 4s - poor [33].

2. **First Contentful Paint**

   The time required to render the first text, image, vector graphic, etc. According to Google Lighthouse, it should fall within the following ranges:

   - from 0 to 2s - good,
   - from 2s to 4s - average,
   - over 4s - poor [32].

3. **Cumulative Layout Shift**

   The sum of shifts of individual elements on the web page, causing that with the loading of subsequent elements, previously displayed ones may shift. This drastically impacts user experience, often causing, for example, a mistaken click on the wrong button that appeared in a place where there was another one before. According to Google Lighthouse, it should fall within the following ranges:

   - from 0 to 0.1 - good,
   - from 0.1 to 0.25 - improvement suggested,
   - over 0.25 - poor [31].

4. **Potential FID**

   Max Potential FID (more precisely, Max Potential First Input Display) denotes the worst case of FID (First Input Delay) that a user can experience. First Input Delay is the time in which a user can already interact with the website, e.g., by clicking a link or button. According to Google Lighthouse, it should fall within the following ranges:

   - from 0 to 130ms - fast,
   - from 130ms to 250ms - average,
   - over 250ms - slow [34].

5. **Speed Index**

   The speed index indicates how quickly the page fills up with visible content. According to Google Lighthouse, it should fall within the following ranges:

   - from 0 to 4.3s - good,
   - from 4.4s to 5.8s - average,
   - over 7.3s - poor[35].

6. **Time to Interactive**

   Time to full interactivity is the time after which the site becomes fully interactive. According to Google Lighthouse, it should fall within the following ranges:

   - from 0 to 3.8s - good,
   - from 3.9s to 7.3s - average,
   - over 7.3s - poor [36].

   A page is considered fully interactive when:

   - The page displays useful content, which is measured by **First Contentful Paint** [32],

- event handler is already registered for most visible items,
- Page responds to user interactions within **50ms** [36].

7. **Time to first byte**
   Time to First Byte (TTFB) is a metric that measures the time between the request for a resource and when the first byte of response begins to arrive. It is a foundational metric for measuring connection setup time and web server responsiveness. TTFB can help identify when a web server is too slow to respond to requests and, in the case of navigation requests for an HTML document, it precedes every other meaningful loading performance metric [24]. It should fall within the following ranges:
   - from 0 to 800ms - good,
   - from 800ms to 1800ms - needs improvement,
   - over 1800ms - poor [24].

Conclusions

The web development landscape has seen significant evolution with the advent of technologies like Content Delivery Networks (CDNs), Content Management Systems (CMSs), and the JAMstack architecture. These technologies aim to address the challenges of website performance, scalability, and security in the contemporary Internet era. However, understanding the impact of these technologies on website performance metrics, and choosing the appropriate tools and services requires rigorous scientific analysis. CDNs, such as Cloudflare and Vercel, have emerged as viable alternatives to traditional web hosting solutions, leveraging geographically dispersed servers to deliver content faster and more efficiently to users worldwide. Similarly, CMSs, like Flotiq, provide developers with a platform to manage website content effectively, particularly when working within the JAMstack architecture. The JAMstack architecture itself represents a paradigm shift in web development, promoting a decoupled architecture for better performance, scalability, and security. Despite the apparent benefits of these technologies, their effectiveness is contingent on various factors, including the size of the website, the geographical location of the users, and the specific CDN and CMS services used. Thus, it becomes essential to empirically test and compare these technologies to derive conclusive insights.

The subsequent research chapter will delve into the planning, design, deployment, and testing of static websites using different CDNs, with a particular focus on Cloudflare, and Vercel. The study will employ a comparative analysis approach to measure and evaluate website performance metrics depicted in detail in ch. 1.3.4.4. Through systematic testing and analysis, this research aims to provide a comprehensive understanding of how CDNs, CMSs, and the JAMstack architecture impact website performance, thereby enabling informed decision-making for web developers and researchers in the field. Sum up of the tests and showcase is shown in Table 1.1.

Tabela 1.1. Website performance metrics and their depiction

Label	Code	Description	Possible Values
M1	FMP	**First Meaningful Paint** The time required to render the first content in the DOM.	0 - 2s (good) 2s - 4s (average) > 4s (poor)
M2	FCP	**First Contentful Paint** The time required to render the first text, image, vector graphic.	0 - 2s (good) 2s - 4s (average) > 4s (poor)
M3	CLS	**Cumulative Layout Shift** The sum of shifts of individual elements on the web page.	0 - 0.1 (good) 0.1 - 0.25 (average) > 0.25 (poor)
M4	FID	**First Input Display** First Input Delay is the time in which a user can already interact with the website.	0 - 130ms (good) 130ms - 250ms (average) > 250ms (poor)
M5	SI	**Speed Index** The speed index indicates how quickly the page fills up with visible content.	0 - 4.3s (good) 4.4s - 5.8s (average) > 7.3s (poor)
M6	TTI	**Time to Interactive** Time to full interactivity is the time after which the site becomes fully interactive.	0 - 3.8s (good) 3.9s - 7.3s (average) > 7.3s (poor)
M7	TTFB	**Time to First Byte** Time to First Byte measures the time between the request for a resource and when the first-byte of the response arrive.	0 - 0.8s (good) 0.8s - 1.8s (average) > 1.8s (poor)

2. RESEARCH

This chapter details the comprehensive investigation undertaken to discern the performance characteristics of web hosting through Content Delivery Network (CDN) services, using Cloudflare and Vercel as representative, in comparison to themselves and a traditional web hosting. It delineates the various stages of the research process, from the initial methodology and planning phases through to the design, deployment, testing, and the ensuing analysis and discussion of the results.

The 'Planning' section begins by outlining the comprehensive research methodology that guided the entire study. It covers the research approach, the selection of tools and metrics for data collection and performance evaluation, and the methods employed for data analysis. The planning phase also involves detailing the functional and non-functional requirements of the web application to be deployed for the study, as well as presenting the preliminary concepts that influenced the design and development process.

In the 'Designing' section, the final concept for the web application is defined based on the criteria outlined in the planning stage. This section covers the process of designing the architecture of the web application, selecting the appropriate tools and services for its development and deployment, and ensuring its compatibility with both the CDN and traditional web hosting environments.

The 'Deployment' section describes the process of deploying the web application on both Cloudflare, Vercel and the traditional hosting platform. It discusses the steps taken to ensure equivalent conditions in both environments, and any challenges or issues encountered during the deployment phase.

The 'Tests' section presents the actual performance tests conducted on the web application in both hosting environments. It provides a detailed account of the testing conditions, the performance metrics captured, and the tools and methodologies used for data collection and analysis.

Finally, the 'Discussion' section synthesizes and interprets the results of the performance tests. It offers a comparative analysis of the performance characteristics of CDN hosting via Cloudflare, Vercel and traditional web hosting, discussing the implications of the findings and providing recommendations for future research. This chapter thus provides a holistic view of the research process, offering significant insights into the comparative performance of CDN and traditional web hosting.

2.1. PLANNING STAGE

The primary objective of this research is to carry out a comparative assessment of websites deployed using Content Delivery Networks (CDNs) from Cloudflare [5] and Vercel [25], in contrast to a website hosted on a conventional server based in Europe. The traditional server setup will be deployed using AWS EC2 [22].

The initial step involves the creation of two distinct static websites using the JAMstack architecture ch. 1.3.3 and the Gatsby framework [9]. The complexity of these websites will vary, with one being simple, incorporating minimal content, and the other being more elaborate, integrating several images with an overall size of approximately 10MB.

These websites will be deployed utilizing CDN services from Cloudflare and Vercel, and a conventional server based in Europe. To comprehensively test the capabilities of these CDNs, performance evaluations will be performed from various geographical locations worldwide.

To accommodate this, the AWS platform will be employed, specifically the Amazon Elastic Container Service (ECS). ECS is a scalable, high-performance container orchestration service that supports Docker containers, enabling applications to run on a managed cluster of Amazon EC2 instances.

The orchestration of these containers will be managed using Terraform, an open-source Infrastructure as Code (IaC) software tool. Terraform facilitates the expression of infrastructure setup as code in a simple, human-readable language, ensuring a consistent and repeatable infrastructure setup across different regions.

The Docker containers will execute a script that employs Google Lighthouse, a tool designed to assess the quality of web pages. Lighthouse will evaluate several crucial performance parameters of the websites. Additionally, a headless Google Chrome browser will be activated prior to testing. To optimize resource utilization, an AWS task with a cron configuration will be created, which will initiate the container. The container is configured to terminate itself after the test execution.

The output from these performance tests will be stored in AWS S3 [4], a scalable object storage service for data archiving and backup. Upon completion of all tests, these reports will be retrieved, transformed, and analyzed to derive valuable insights about the performance of the websites under different hosting solutions.

The performance tests will span a duration of one week, with each test being initiated at six-hour intervals. This methodology ensures the collection of a substantial volume of performance data under various network conditions and times, enhancing the robustness of the subsequent analysis.

In conclusion, this plan leverages potent technologies and methodologies, including Gatsby, AWS ECS, Terraform, Docker, and Google Lighthouse, to conduct a detailed and

comprehensive assessment of websites' performance when hosted using CDN services and a conventional server.

2.2. DESIGNING

2.2.1. Design of the Test Websites

In order to evaluate the performance of different hosting solutions, two test websites are designed: a text-based website and a more heavy website. Both of these websites are designed as single page applications, due to their increasing prevalence and the different performance challenges they present.

Basic Website

This version of the website is primarily text-based, with an interactive button that changes state upon clicking. This site is designed to be less than 1MB in size, allowing for an evaluation of the performance of the hosting solutions with minimal content. This site serves as a baseline for comparison with the more heavy website. Despite its simplicity, it is responsive (its layout adapts to different screen dimensions of end users) and optimized for mobile devices (mobile-first design), reflecting the a significant proportion of web traffic that comes from mobile users[14].

Heavy Website

The heavy website includes text, images, and an interactive form. Images are embedded directly into the site, to evaluate the ability of the CDN to cache this type of content. This site is also optimized for mobile devices, reflecting the common trend towards mobile web browsing. The total size of this site is approximately 10MB, allowing for an evaluation of performance with a more typical web page size.

2.2.1.1. Static-Generated and Dynamic-Generated Websites

Static-Generated Websites

Static-generated websites, also known as static sites, are websites where the content is generated during the build process and then served to the user as is. Each page is a standalone HTML file that represents a unique URL on the website. This means every time a user requests a page, the server delivers the same HTML file to every user, regardless of the nature of the request or the user's individual characteristics [44].

This method has several advantages:

- **Performance:** Because the server is merely retrieving and sending a file, static sites typically load faster than dynamic ones.
- **Security:** Static sites are inherently more secure as they don't rely on database calls or server-side processing.
- **Scalability:** Scaling is as simple as serving the same files to more users, often via a Content Delivery Network (CDN).
- **Development and Hosting Costs:** Static sites are generally simpler to develop and cheaper to host than their dynamic counterparts.

However, static sites are not ideal for situations where content changes frequently or needs to be personalized for individual users.

Dynamic-Generated Websites

Dynamic-generated websites, on the other hand, generate pages on-the-fly based on the user's request. When a user requests a page, the server processes the request, queries a database if needed, assembles the HTML file, and then sends it to the user's browser. This allows for a high degree of personalization and interactivity, as the content can change in response to user inputs or other factors [44].

Dynamic websites have their own advantages:

- **Interactivity:** Dynamic sites can provide a highly interactive experience, as they can react to user inputs and preferences.
- **Personalization:** Content can be tailored to individual users, making dynamic sites ideal for social media, and other user-focused applications.
- **Freshness:** The content of the website can change constantly without the need for a rebuild, making dynamic sites ideal for news sites, blogs, and any site where the content changes regularly.

However, dynamic websites can be slower, more complex to develop, and more expensive to host than static sites. They also face more security risks due to their reliance on server-side processing and databases.

2.2.1.2. Choice of Static Site Generator

Given that both test websites are designed as static websites, a static site generator is required. After reviewing various options, Gatsby, a popular choice in the JAMstack ecosystem, was selected due to its wide usage, strong community support, and its ability to handle the interactive elements required in the test websites.

2.2.1.3. Gatsby as a Static Site Generator

Gatsby is a modern static site generator that uses React [21] and GraphQL [16], offering a new approach to building websites and applications. It allows developers to build high-performing websites with a rich set of features out of the box.

Unlike traditional static site generators that merely produce HTML files, Gatsby produces static HTML files based on React components. This means that while the initial load of a Gatsby site serves as static files, subsequent interactions on the site are handled by React in the browser, effectively turning the site into a single-page application (SPA) [9].

Gatsby leverages GraphQL at build time to create these static sites. With GraphQL, developers can query the exact data they need for a page in a single request, eliminating over-fetching and under-fetching of data. This data can come from a wide variety of sources, including headless CMSs, SaaS services, APIs, databases, and the file system. Gatsby's rich plugin ecosystem makes it easy to connect these data sources [9].

The result is a static site that is fast, secure, and easy to scale. Because the pages are pre-built, they can be served directly from a CDN, which reduces server load and makes them inherently more performant than server-rendered pages.

Gatsby also includes modern web standards such as Webpack [26], and Babel [1], making it a robust tool for developers. It supports Progressive Web App (PWA) features, which means Gatsby sites can work offline and feel like native apps [9].

However, Gatsby does have its limitations. The build times can become lengthy for large sites. Also, being a static site, real-time features would require additional solutions. Despite these, Gatsby stands as a powerful choice for building high-performance static websites.

2.2.1.4. Interactive Elements

Both websites include interactive elements, reflecting the dynamic nature of modern websites. Despite being static sites, the use of client-side JavaScript allows for dynamic changes to the website after it has been loaded by the browser. This allows for the inclusion of interactive elements such as buttons and forms, which can be used to simulate user interaction and evaluate performance under more realistic conditions.

2.2.1.5. Considerations

While the design of the test websites takes into account many factors, it should be noted that this study is primarily focused on performance, rather than aspects such as SEO. Therefore, while some SEO considerations may be inherent in the design choices (such as the use of Gatsby, which has good SEO support), they are not the primary focus of this study.

2.2.2. Selection of Deployment Solutions

The process of selecting an appropriate deployment solution, that includes CDN, for the designed websites required careful consideration and evaluation of various factors. These factors included the popularity of the solution, cost, support for Gatsby, and the overall focus on performance.

2.2.2.1. Chosen solutions

Cloudflare [5] and Vercel [25] were chosen as primary solutions for several reasons. Firstly, they both enjoy a significant share of the market, which is indicative of their reliability and the quality of service they offer. Secondly, they both provide free options, which aligns with the aim of employing cost-effective solutions for this project [2]. Thirdly, they provide native support for Gatsby, the chosen static site generator. Lastly, both of these solutions have a strong focus on performance, which is an essential factor for the successful delivery of content on the web.

2.2.2.2. Considered solutions

Two additional solutions were also considered for comparison purposes: Netlify [6] and GitHub Pages [13]. Netlify is similar to both Vercel and Cloudflare in terms of its features, however, it was not chosen as a primary solution due to its less significant market presence. GitHub Pages, while popular, lacks some of the advanced features offered by the other solutions and thus was deemed less suitable for the requirements of this project [2].

2.2.2.3. Global distribution

The deployment solutions were also evaluated for their ability to distribute content globally. This is crucial for the project, as the performance tests will be conducted from various geographical locations (US, Europe, Asia). Both Cloudflare and Vercel possess a globally distributed network of servers, which facilitates rapid content delivery irrespective of the user's location.

2.2.2.4. Security of the deployed website

Regarding security, JAMstack architecture ensures a high level of security by default, which reduces the need for additional security measures. However, the chosen solutions further enhance security by providing features such as DDoS protection and TLS encryption.

Conclusion

The selected deployment solutions will be validated through performance tests, the details of which will be discussed in the subsequent section.

2.2.3. Design of the Test Environment

The design of the test environment is focused on providing a robust and automated performance testing system. This system aims to capture and analyze the critical performance metrics of the deployed websites.

2.2.3.1. Testing Tools

The primary tool chosen for performance testing is Lighthouse, a widely recognized open-source, automated tool designed by Google for improving the quality of web pages. The choice of Lighthouse is motivated by its comprehensive performance metrics, ease of integration, and the fact that it can run programmatically as a Node.js module. This makes it suitable for the automated testing system which is required in this project. The test requires running the Google Chrome browser that will be used by Lighthouse. The browser will be provided inside the docker image. The automated tests will be triggered by AWS container task manager every six hours in 3 different regions for a week, which will run the Lighthouse tests and save results as a JSON file to S3 storage.

2.2.3.2. Test Metrics

The metrics to be tracked during testing are comprehensive and cover a range of performance aspects:

- First Meaningful Paint
- First Contentful Paint
- Cumulative Layout Shift
- Potential First Input Delay
- Speed Index
- Time to Interactive
- Time to First Byte

These metrics were chosen due to their relevance for user-perceived website performance and their representation in the Lighthouse performance score.

2.2.3.3. Automation

Automation is a vital component in this testing environment. Docker containers will be utilized to initiate the Lighthouse tests and process the results. Docker [7] was chosen due to its compatibility with various platforms, the ability to package and distribute applications in containers, and its efficient resource isolation features. Furthermore, containers will be orchestrated using Amazon Elastic Container Service (ECS), allowing for robust management and operation of these containers at scale. The script used to create the registry is provided inside the Appendix B and will be closer explained in ch. 2.3.2.

2.2.3.4. Deployment

The automated deployment of the testing environment will be handled using Terraform [23], an open-source Infrastructure as Code (IaC) [46] tool that provides a consistent CLI workflow for managing cloud services. Terraform was selected due to its broad provider support, robust community, and its capacity to manage intricate and interdependent infrastructure. The code used to deploy the solutions automatically is provided in Appendix A.

2.2.3.5. Data Storage

After the tests are executed, the data needs to be stored for later analysis. For this purpose, AWS S3 (Simple Storage Service) will be employed. AWS S3 was selected because of its durability, scalability, security, and integration with other AWS services. It allows easy retrieval and analysis of the test data when required.

2.2.3.6. Challenges and Considerations

Designing an automated test environment involves several considerations and potential challenges. The reliability of the testing process is paramount, and it's essential to ensure the accuracy of the results obtained. As such, the choice of Lighthouse as a testing tool, with its extensive metrics and automated testing capabilities, contributes significantly to the reliability of the testing process.

Moreover, the decision to use AWS Elastic Container Service for automation and AWS S3 for storage also provides benefits in terms of scalability, manageability, and cost-effectiveness, but it also ties the project to a particular cloud provider. While AWS offers a comprehensive suite of services, this choice may limit future flexibility if a switch to another cloud provider becomes desirable.

In conclusion, the design of the test environment aims to provide an automated, robust, and scalable solution for performance testing of the deployed websites. The chosen tools and services align with these goals, providing the means to capture and analyze critical performance metrics.

2.2.4. Design of Test Execution

The testing frequency is designed to strike a balance between comprehensiveness and efficiency. It is crucial to have regular performance data to enable an accurate analysis of the deployment solutions' effectiveness. However, excessively frequent tests can consume unnecessary resources without significantly improving the data quality.

The choice of running tests every two hours is based on several considerations. First, it allows for a sufficient number of data points to be collected over a given day, enabling the capturing of any temporal trends or fluctuations in website performance. This could include peak load times or any inconsistencies that may occur during server maintenance periods.

Second, with the geographic distribution of the tests across the US, EU, and Asia, this frequency ensures that performance data is collected from different regions in various time zones. It provides a comprehensive view of the global performance of the deployment solutions, capturing any region-specific anomalies that may arise.

Finally, it is essential to consider the computational cost and the potential impact on the tested websites. Running performance tests involves generating and processing substantial amounts of data. While it is crucial to ensure the tests' thoroughness, it is equally important not to overload the system with frequent, heavy testing. The chosen frequency aims to balance these factors, providing regular and comprehensive performance data without unnecessarily straining resources.

The testing frequency and the specific metrics chosen for evaluation are not set in stone. They can be adjusted based on the evolving needs of the project, the observed performance of the deployment solutions, and any specific issues that may arise during the testing process.

2.2.5. Design of Test Data Storage and Analysis

The design of the test data storage and analysis component is a vital part of the testing framework. It ensures that the test results are stored effectively and that meaningful insights can be extracted from them. The primary focus for this part of the framework is on efficient storage, ease of access, and comprehensive analysis.

For storage Amazon's Simple Storage Service (S3) was choosen. This choice was driven by the need for a durable, secure, and scalable storage solution. S3 provides these features, along with cost-effectiveness for large amounts of data. Furthermore, the test results generated by Lighthouse are stored in JSON format, which is readily supported by S3.

The data in the S3 bucket [4] is organized based on the website name, region, and the time of test execution. This organization helps to quickly identify and access the required test results for a given website and region, and for a specific time period.

Data is encrypted both in transit and at rest, providing an additional layer of security.

The test data is analyzed using a combination of automated and manual methods. Automated methods are used for preliminary analysis and alerting, such as detecting significant changes in performance metrics. Manual methods are used for deeper, ad-hoc analysis.

The analysis aims to provide insights into the performance of the websites and to identify any potential bottlenecks or issues. The focus is on understanding the trends in the performance metrics, comparing the performance across different regions, and identifying any anomalies.

For visualization of the analysis results, standard tools such as graphs and charts are

used. They provide a visual representation of the performance metrics, making it easier to understand the trends and patterns.

Overall, the design of the test data storage and analysis component is intended to provide a comprehensive view of the website performance, enabling informed decisions for optimization and improvement.

Tests design table

Sum up of the tests and showcase was shown in the table below [Table 2.1]

Tabela 2.1. Tests shortcut

Website	Test Kind	Location	Time	Period
Basic Website	Performance	US EU Asia	06:00	1 Week
Heavy Website			12:00	
Basic Website (Standard Host)			18:00	
			24:00	

2.3. DEPLOYMENT

The deployment stage of this project marked a shift from the conceptual design and planning stages to a more technical implementation phase. The chosen architectural design, technological stacks, and tools were implemented, with comprehensive code for scripts and applications being written. The following subsections provide detailed insight into each step of the deployment process.

2.3.1. Creation of the Node.js Script

The Node.js script plays a central role in the deployment phase. Its purpose is to execute the core task of running Lighthouse tests and subsequently, storing the reports generated from these tests in an Amazon S3 bucket.

AWS SDK and Puppeteer

The script leverages several Node.js [19] packages to execute its functions. Among them are the AWS SDK [12] and Puppeteer [20]. The AWS SDK enables interactions with Amazon S3, allowing the script to push the Lighthouse reports to the cloud. Puppeteer, on the other hand, is a Node.js library that provides a high-level API to control Chrome or Chromium over the DevTools Protocol. Code fragment showcasing the usage of puppeteer has been shown in Listing 2.1. The code starts the browser process and closes

it after finishing the tests. It is used in this script to launch a headless Chrome browser programmatically, which is crucial for running the Lighthouse tests.

```
import puppeteer from "puppeteer";

//...
    chrome = await puppeteer.launch({
      args: ["--no-sandbox", "--disable-gpu",
      ↪  "--remote-debugging-port=9222"],
      executablePath: "/usr/bin/google-chrome",
      headless: "new",
      ignoreHTTPSErrors: true,
    });
    runnerResult = await runLighthouseTest(chrome);
  }

  await chrome.close();
  //...
};
```

Listing 2.1: Puppeteer usage in Node.js script

Lighthouse Tests

Code Listing 2.2 showcases the function `runLighthouseTest()` that encapsulates the Lighthouse test execution. This function is configured to run tests focused on the performance category. The runLighthouse() function is a higher-level function that initiates a headless Chrome instance via Puppeteer, triggers the Lighthouse tests through `runLighthouseTest()`, and handles any errors that may occur during this process.

Environment Variables

The script is designed to be flexible and reusable by employing environment variables such as WEBSITE_URL, REGION, WEBSITE_NAME, and S3_BUCKET_NAME. These variables, set during the node process initiation, inform the script about the website URL to be tested and the S3 bucket where the report should be sent.

Report Generation and Handling

The Lighthouse tests generate JSON reports, which are saved to the local filesystem before being uploaded to the S3 bucket. The files are named following a convention that includes the website name, the region, and the date and time of the test. After a successful upload to S3, the local report file is deleted to conserve resources within the Docker

```
import lighthouse from "lighthouse";

//...

const runLighthouseTest = async () => {
  const options = {
    output: "json",
    onlyCategories: ["performance"],
    port: 9222,
    preset: "desktop",
    "disable-full-page-screenshot": true,
    chromeFlags: [
      "--headless",
      "--disable-gpu",
      "--no-sandbox",
      "--disable-dev-shm-usage",
    ],
  };
  return await lighthouse(process.env.WEBSITE_URL, options, {
    extends: "lighthouse:default",
    settings: {
      onlyAudits: [
        "first-meaningful-paint",
        "first-contentful-paint",
        "cumulative-layout-shift",
        "max-potential-fid",
        "speed-index",
        "interactive",
        "network-server-latency",
        "server-response-time",
      ],
    },
  });
};

//...

runLighthouse().catch(console.error);
```

Listing 2.2: Lighthouse test in Node.js script

container. In case of any failure during the upload, the script crashes without deleting the local report, thereby allowing for error detection and troubleshooting.

Challenges

The primary challenge encountered during the creation of this script was ensuring that the headless browser launched via Puppeteer was operational and the same for all the tests.

2.3.2. Docker Deployment

Docker [7] serves as a crucial component in the deployment process of this project. The primary role of Docker in this context is to create an image [7] that encapsulates all the necessary dependencies and configurations, ensuring the project can run in any environment that supports Docker. This image is subsequently pushed to the AWS registry (ECR) [11], from which it can be initiated as a Docker container at a given time in a specified AWS region. Appendix B contains Dockerfile that explains what the base image contains, and how it is set up for the tests. Subsequent subsections will explain this Listing in the great detail.

Base Image Selection

The base image for the Dockerfile [8] is Ubuntu 20.04. This selection is influenced by the popularity and widespread usage of this Linux distribution, which in turn translates to extensive community support and ease in troubleshooting.

Package Installation

Several packages are installed in the Docker image to ensure the proper functioning of the project. These include curl, unzip, wget, gnupg, Node.js, AWS CLI, and Google Chrome.

- *curl, unzip, wget, gnupg*: These utilities support various operations within the Docker image, including file downloads, extraction, and scheduled tasks.
- *Node.js*: The Node.js runtime is necessary to execute the project's core script, which is written in JavaScript and runs on Node.js. This script is the essence of the project, performing the actual performance testing.
- *AWS CLI*: The AWS Command Line Interface (CLI) is a unified tool to manage AWS services. In this case, it is used to push the generated reports to an S3 bucket.
- *Google Chrome*: Puppeteer, a headless browser used for performance testing in this project, requires a Chrome or Chromium browser installation. Hence, Google Chrome is installed in the Docker image.

Working Directory and AWS Credentials

The working directory for the Docker image is set to /app. An AWS credentials file is copied into the Docker image during its creation. This file is necessary for the AWS CLI to authenticate and authorize operations against the AWS resources.

Entrypoint Script

Listing 2.3.2 is crucial to the operation of the Docker container. The purpose of this script is to run the Node.js performance testing script for each website to be tested. This script is set as the entrypoint for the Docker container, meaning it is automatically run every time the Docker container is initiated.

```
#!/bin/bash
declare -a websites=("https://web-performance-testing.pages.dev"
  ↪ "https://web-performance-testing-2.pages.dev"
  ↪ "https://web-performance-testing-two.vercel.app"
  ↪ "https://web-performance-testing-heavy.vercel.app")
declare -a websites_names=("basic_pages" "heavy_pages" "basic_vercel"
  ↪ "heavy_vercel")

for i in "${!websites[@]}"; do
  WEBSITE_URL="${websites[$i]}" WEBSITE_NAME="${websites_names[$i]}" node
    ↪ /app/lambda/index.js
done
exit 0
```

Listing 2.3: entrypoint.sh script

In the Listing 2.3.2, the URLs of the websites to be tested are declared in an array, and a loop is used to iterate over this array. For each website, the Node.js script is run with the website's URL and name passed as environment variables. Once all the websites have been tested, the script exits with a status code of 0, indicating successful execution and end of life for the container.

2.3.2.1. Building and Publishing the Docker Image

Once the Dockerfile was complete, the Docker image was built using the `docker build` command. The image was then published to Docker Hub using the `docker push` command, making it available for use in any environment that can run Docker containers, including the AWS ECS service.

2.3.2.2. Configuring the AWS ECS Service with Terraform

The AWS ECS Service is configured using Terraform, an Infrastructure as Code (IaC) tool that allows for simple management and versioning of infrastructure. In this case, the Terraform code provided is designed to set up and manage several AWS services necessary for the deployment of the Docker containers that perform website performance testing.

Resource Allocation and Management

The first part of the Terraform configuration involves the definition of various AWS resources. AWS ECS (Elastic Container Service) is used to manage and orchestrate the Docker containers. An ECS cluster named "web-testing-performance"is declared using the **aws_ecs_cluster** resource.

The **aws_ecs_task_definition** resource is used to describe the tasks that will run in the ECS cluster. The **container_definitions** attribute specifies properties for the Docker container, such as the Docker image to be used (which is the image that was built and uploaded to AWS ECR in the Docker section), and environment variables that are necessary for the running of the script, including the AWS region and the name of the S3 bucket for storing test reports.

```
resource "aws_ecs_cluster" "cluster" {
  name = "web-testing-performance"
}

resource "aws_ecs_task_definition" "task" {
  family                   = "testing"
  network_mode             = "awsvpc"
  requires_compatibilities = ["FARGATE"]
  cpu                      = "4096"
  memory                   = "8192"
  execution_role_arn       = aws_iam_role.ecs_task_execution_role.arn

  container_definitions = jsonencode([{
    name : "web-performance-testing-${var.region}",
    image : var.docker_image,
    essential : true,
    environment : [
      { "name" : "REGION", "value" : var.region },
      { "name" : "S3_BUCKET", "value" : var.s3_bucket }
    ]
  }])
}
```

Resource Allocation for Adequate Performance

When running performance tests, it is crucial to ensure that the hardware on which the tests are executed is not a limiting factor. The Google Lighthouse team, creators of one of the most popular web performance testing tools, strongly recommend running tests on adequate hardware to minimize variability in test results. According to their guidelines [18], test environments should not be resource-constrained, as this can introduce additional latency and variability, which can skew the results.

In the context of AWS Fargate, which is used to run the ECS tasks in this project, the CPU value is expressed in units where 1024 is equivalent to a single vCPU. The chosen value of 4096 for CPU in the **aws_ecs_task_definition** resource thus corresponds to 4 vCPUs. This allows the performance testing script to run in a multi-threaded mode, making full use of modern multi-core processors. Such a setup can handle the computational demands of running performance tests, especially if the tests involve running complex page rendering and JavaScript execution tasks.

The memory value for the task is set to 8192, which corresponds to 8GB of RAM. This is an ample amount to run performance tests on most websites. The Chrome browser, which is used by many web performance testing tools, can be memory-intensive, especially when loading and interacting with complex web pages. Having sufficient memory ensures that the browser and testing scripts can function optimally [18], without being impeded by memory swapping or other resource constraints.

Choosing these values for CPU and memory ensures that the test environment has sufficient resources to handle the load of performance testing, and helps minimize variability and latency in the test results. Listing 2.3.2.2 shows resources assignment to the specific testing task.

```
resource "aws_ecs_task_definition" "task" {
    family = "testing"
    network_mode = "awsvpc"
    requires_compatibilities = ["FARGATE"]
    cpu = "4096"
    memory = "8192"
    execution_role_arn =
    ↪ aws_iam_role.ecs_task_execution_role.arn
}
```

Listing 2.4: Resource assignment

IAM Roles and Permissions

The **aws_iam_role** resource is used to create an IAM role for ECS tasks. This role grants the ECS service permissions to access other AWS service resources. The associated

aws_iam_role_policy_attachment binds the **AmazonECSTaskExecutionRolePolicy** to the created role, which provides the necessary permissions for ECS to execute tasks on behalf of the role.

Scheduling and Running Tasks

Finally, the Terraform configuration uses CloudWatch events to schedule the ECS tasks. The **aws_cloudwatch_event_rule** resource creates a rule that triggers every six hours, as defined by the **schedule_expression** attribute. The **aws_cloudwatch_event_target** resource then associates this rule with the ECS task, specifying that the task should be run in the ECS cluster each time the rule triggers. Listing 2.3.2.2 explains scheduling the test tasks and when the task is supposed to fire.

```
resource "aws_cloudwatch_event_rule" "every_six_hours" {
  name                = "every-six-hours"
  description         = "Fires every six hours"
  schedule_expression = "cron(0 6,12,18,0 1-30 6 ? *)"
}

resource "aws_cloudwatch

_event_target" "run_task_every_six_hours" {
  rule      = aws_cloudwatch_event_rule.every_six_hours.name
  target_id = "run-ecs-task"
  arn       = aws_ecs_cluster.cluster.arn
  role_arn = aws_iam_role.ecs_task_execution_role.arn

  ecs_target {
    task_count          = 1
    task_definition_arn = aws_ecs_task_definition.task.arn
  }
}
```

Listing 2.5: Scheduled test task

Deployment across regions

The AWS infrastructure is deployed across different regions using a bash script. The script iterates over an array of AWS regions, setting the appropriate environment variables for each region. Then, for each region, Terraform is initialized, and the configuration is applied using the `terraform apply -auto-approve` command. This command creates all the necessary AWS resources as per the Terraform configuration file and starts the execution of the ECS tasks. Listing 2.3.2.2 is a script that deploys terraform code for all

different regions. It is required as it is not possible to deploy to different regions with only one configuration.

```
#!/bin/bash
DOCKER_IMAGE=$1

cd ./terraform_configs/ap-southeast-1/ || exit
bash tf_deploy.sh "$DOCKER_IMAGE"
cd ../eu-west-1/ || exit
bash tf_deploy.sh "$DOCKER_IMAGE"
cd ../us-east-1/ || exit
bash tf_deploy.sh "$DOCKER_IMAGE"

```

Listing 2.6: Bash script for all AWS regions

Listing 2.3.2.2 is a bash script that is responsible for deploying a single region. `regions` variable specifies what region code is deployed to.

This approach ensures a global coverage of web performance testing, taking into account the geographical differences in website performance.

Deploying the ECS Task

Finally, the ECS task was deployed by applying the Terraform configuration. After the task was running, the Node.js script within the Docker container started interacting with websites as expected.

Each of these steps played a crucial role in deploying the solution, demonstrating how the conceptual design and planning translated into a working, practical implementation. Any problems encountered during deployment were addressed and resolved, proving the practicality and effectiveness of the solution.

2.3.3. Analysis of Webpages and their Deployment Process

In this section, we'll discuss the deployment processes of two distinct webpages, their size, and structure. The webpages under consideration are hosted on Vercel and Cloudflare Pages. They are designed to serve different use cases, with one being a relatively basic website, and the other being heavier, with more content and larger file sizes.

Basic website

The first webpage, available at `https://web-performance-testing.pages.dev`, is a simple single-page application. It comprises a form at the beginning, followed by text content and a few images. The total size of this website is approximately 6.1 megabytes,

```bash
#!/bin/bash
DOCKER_IMAGE=$1
S3_BUCKET_NAME=website-test-reports
regions=("ap-southeast-1")

echo "$DOCKER_IMAGE"

# Iterate over the regions
for region in "${regions[@]}"
do
  export AWS_DEFAULT_REGION=$region
  export AWS_REGION=$region
  export TF_VAR_region=$region
  export TF_VAR_s3_bucket=$S3_BUCKET_NAME
  export TF_VAR_docker_image=$DOCKER_IMAGE
  # initialize Terraform
  terraform init
  # Apply the Terraform configuration
  terraform apply -auto-approve
done
```

Listing 2.7: Code of script tf_deploy.sh

which is primarily due to the embedded images. This webpage serves as a representative of standard webpages with a moderate amount of content.

Heavy website

The second webpage, found at `https://web-performance-testing-2.pages.dev`, is a heavier version. It features six images up front, some text, and a form. Its total size is considerably larger, at about 25.6 megabytes. This webpage is an example of heavy webpages that utilize substantial visual content, such as image galleries.

Deployment Process

The deployment process for both webpages follows the same general steps, though they are hosted on different platforms. The websites are developed locally and then pushed to a GitHub repository. After the push, the repository is linked to the respective hosting service (Vercel or Cloudflare Pages), and the build parameters are configured. Parameters used are showcased inside Table 2.2 Once the setup is complete, the webpages are built, deployed, and distributed on a Content Delivery Network (CDN). After the deployment, the URL is returned to the user. All Available URLs after all deployments are shown in table 2.3

Tabela 2.2. Build parameters

Parameter Name	Content
Build command	gatsby build
Output directory	/public
Install command	npm install
Root directory	/basic-website
Environment variables	FLOTIQ_API_KEY

Tabela 2.3. Website web adress

Distributor	Website kind	Url
Vercel	Basic	https://web-performance-testing-two.vercel.app
	Heavy	https://web-performance-testing-heavy.vercel.app
Cloudflare	Basic	https://web-performance-testing.pages.dev
	Heavy	https://web-performance-testing-2.pages.dev

Webpage Structure

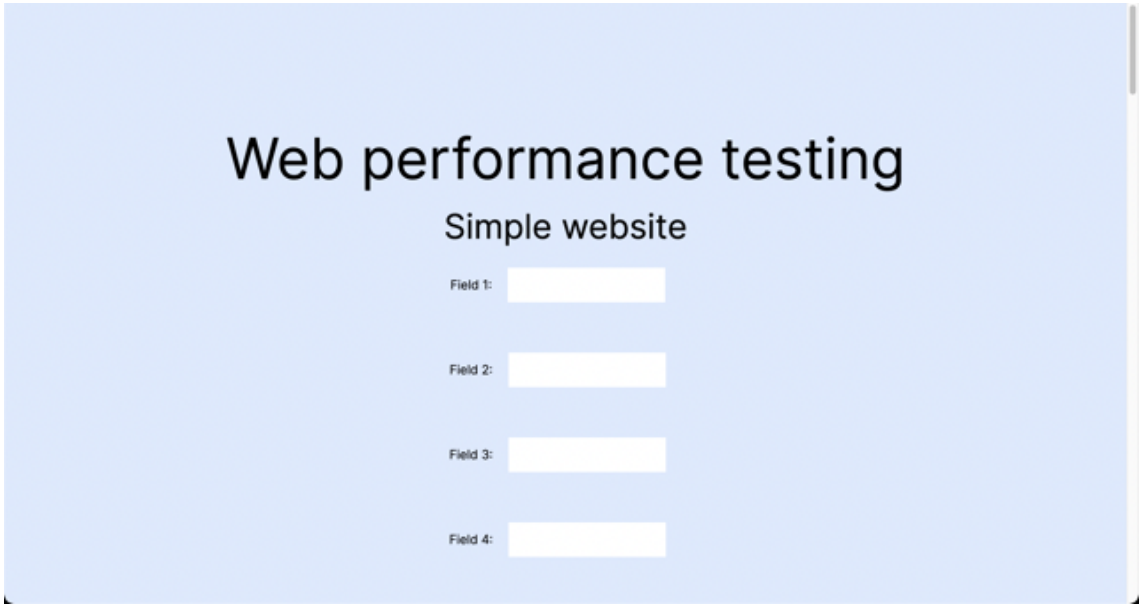
Both webpages are structured as single-page applications (SPAs). SPAs work inside a browser and require only a single page load. After the initial page load, SPAs update and render sections of the page in response to user actions, leading to a fluid user experience. The SPAs for these webpages are composed of images, text, and a form, offering a diverse array of content types.

Basic website Structure

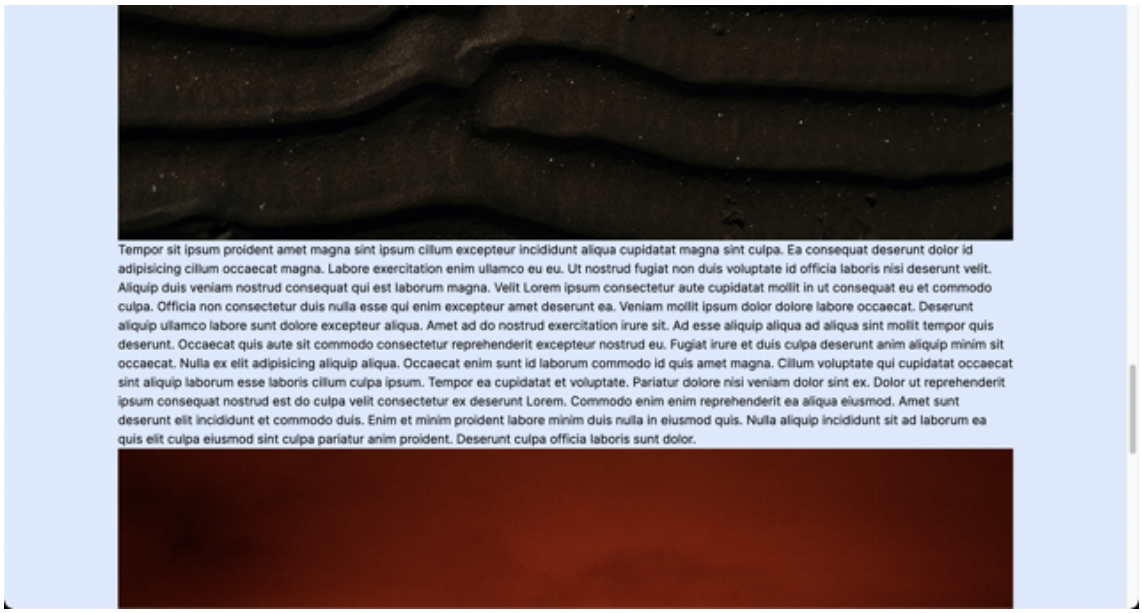
The simpler webpage is designed with a focus on minimalism. It starts with a form, which offers an interactive element for users. Following the form, there is text content, providing information to the visitor. The webpage is concluded with a few images, which serve as visual content. The placement of these components is strategic, offering a balance between text, interactive elements, and visual content. Implemented website is showcased in Figures 2.1 - 2.3.

Heavy website Structure

The heavier webpage adopts a more visual-oriented structure. It starts with six images, immediately offering substantial visual content to the user. Following the image section, the webpage features text content, serving to inform the visitor. Like the simpler webpage, it also includes a form, providing an interactive element. The more significant quantity of images on this webpage, along with their front placement, signifies the importance of visual content in this design. Implemented website is showcased in Figures 2.4 - 2.7.



Rysunek 2.1. Basic website part 1

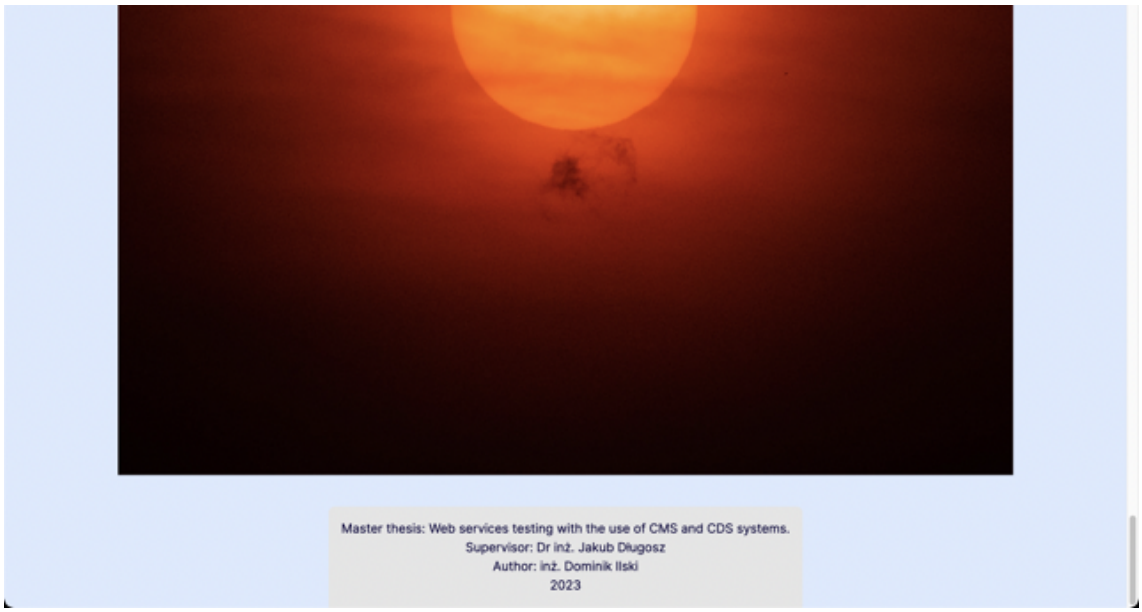


Rysunek 2.2. Basic website part 2

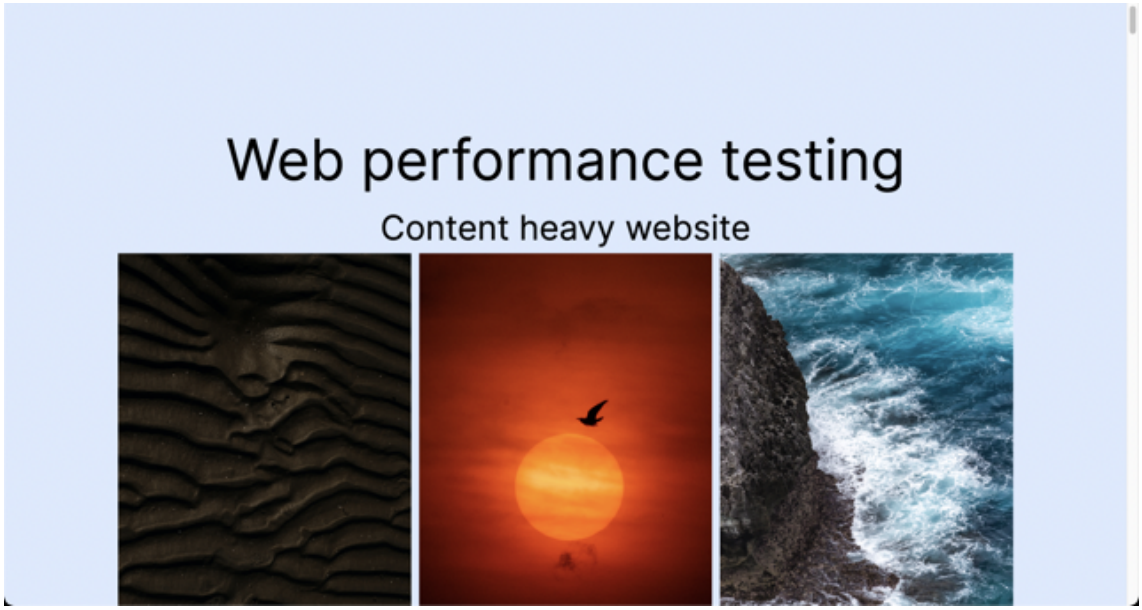
Resource Weight and Placement

The weight of resources and their placement on a website are vital factors affecting the webpage's performance. In this case, images contribute significantly to the weight of both webpages, especially the heavier website.

The placement of resources also plays a significant role in page load times. For the basic website, the form, which is a relatively lightweight resource, is placed at the beginning, followed by text and images. This structure allows the page to load faster initially, delivering important content to the user as quickly as possible.



Rysunek 2.3. Basic website part 3



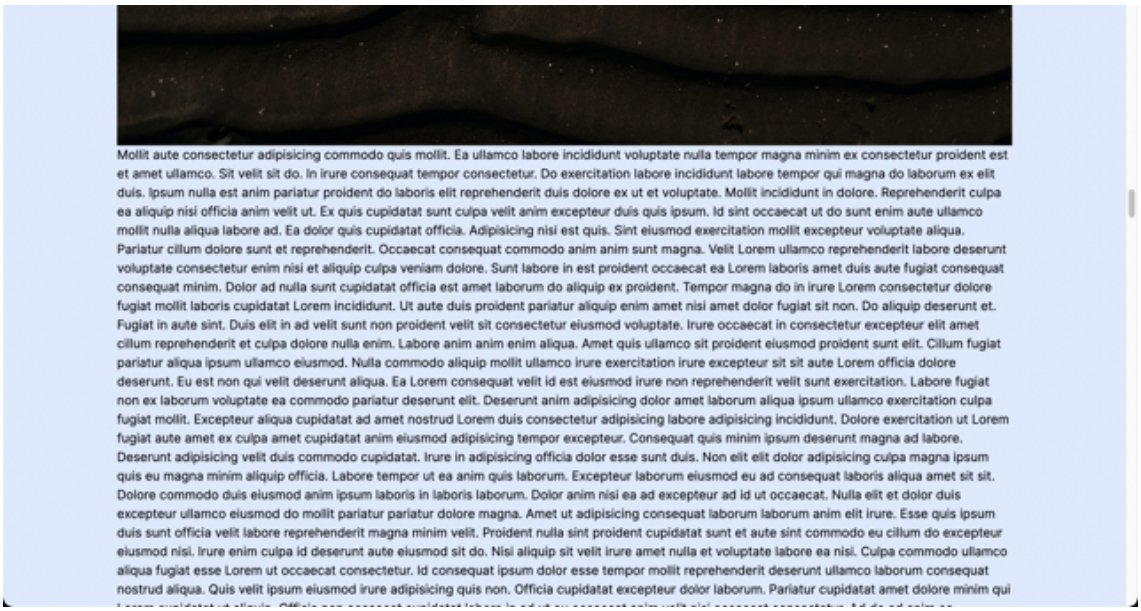
Rysunek 2.4. Heavy website part 1

On the other hand, the heavy website places the images at the beginning, which means the initial page load time is longer. However, this structure is by design, as the images are the main content of this webpage.

The structure and design of both webpages provide valuable insights into how the weight and placement of resources can impact webpage performance, and how designers can balance these factors to create engaging user experiences.

In summary, the deployment and design of these webpages demonstrate the different strategies that can be adopted when developing and hosting webpages, and how these decisions can impact the final user experience. The balance between resource weight and

Rysunek 2.5. Heavy website part 2

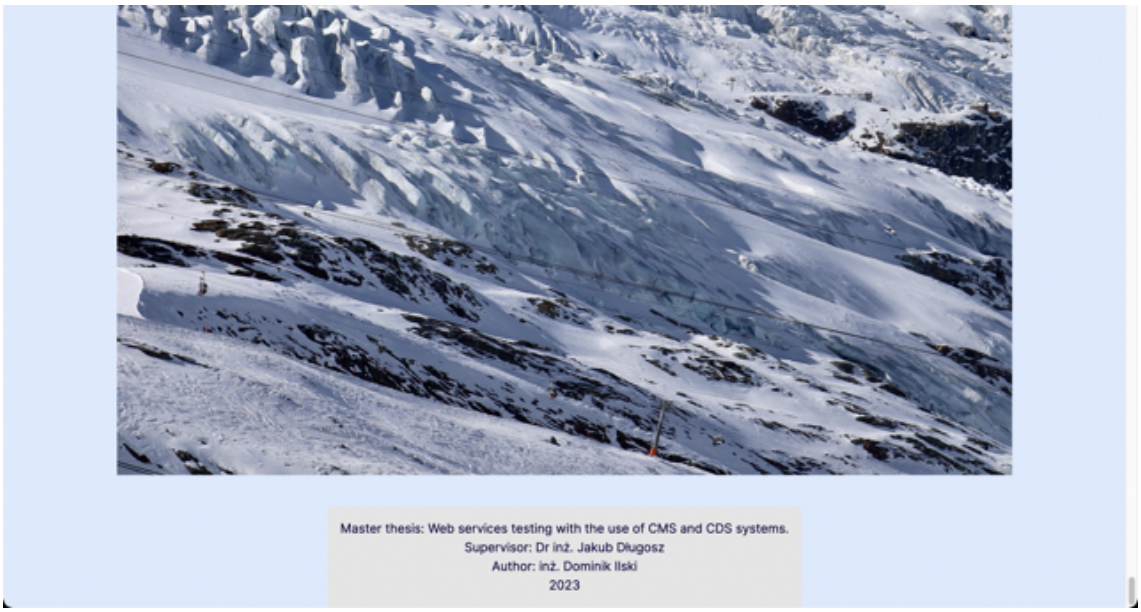


Rysunek 2.6. Heavy website part 3

placement, combined with the power of modern deployment platforms, allows developers to effectively serve content to end users, no matter how simple or complex their needs may be.

Tabela 2.4. Website size, name, and content

Website Name	Content	Size (MB)
basic website	Form, Short Text, 2 Images	6.1
heavy website	6 Images, Long Text, Form	25.6



Rysunek 2.7. Heavy website part 4

2.4. TESTS

The testing phase of the project was an essential part of the development cycle. It allowed to validate the performance of the deployed websites in different regions and under different hosting solutions. The tests were systematically conducted and meticulously recorded to ensure the reliability and reproducibility of the results.

2.4.1. Test Classification and Parameters

The tests were divided into groups based on the region where they were performed and the hosting solution used. Specifically, it was conducted tests in three regions: Asia, the US, and the EU. The parameters for each group of tests have been previously discussed in the ch. 1.3.4.4.

2.4.2. Number and Frequency of Tests

In each group, tests were conducted four times per day. This testing frequency was maintained consistently over a period of one week, allowing to monitor the performance of the websites under different conditions and at different times of the day.

2.4.3. Test Procedure

The procedure for setting up and carrying out the tests was as described in the ch. 2.3. This approach ensured that the tests accurately reflected the real-world usage and performance of the websites. To ensure that tests were run correctly during the testing process, manual analysis was conducted on randomly selected results. The tool used for

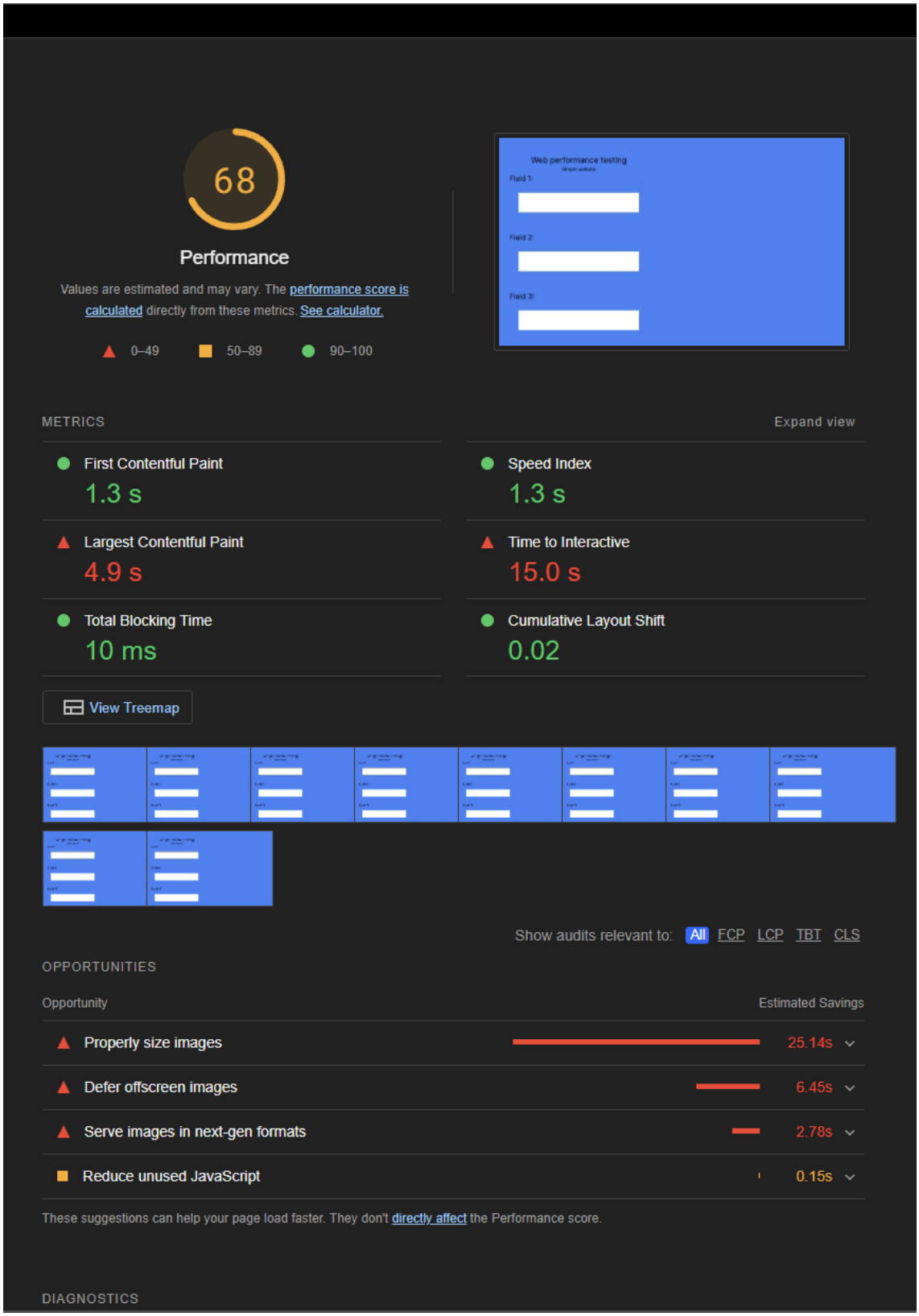
that purpose was Lighthouse Report Viewer [17]. It allows for the analysis of raw data produced from the Lighthouse testing tool. Figure 2.8 displays a report that was randomly selected and manually downloaded from S3.

2.4.4. Automation and Test Results

The testing process was automated as described in the ch. 2.3.2.2. This helped to ensure the consistency and reliability of the tests.

While the testing process proceeded smoothly with no significant problems encountered, it is worth noting that testing is a dynamic process. Even when tests do not yield unexpected results or reveal issues that need to be addressed, they still provide valuable information that can inform future development and testing efforts. In this case, the tests served to confirm the effectiveness of the deployment strategy and the performance of the websites across different regions and hosting solutions.

Given the smooth execution of the tests and the consistency of the results, there were no specific tests that stood out as particularly significant or impactful. However, the overall process of systematic, repeated testing was in itself a key aspect of the project's success. The rigorous testing process provided a solid foundation of evidence supporting the performance and reliability of the deployed websites.



Rysunek 2.8. Lighthouse report from basic website displayed in Lighthouse report viewer.

3. ANALYSIS

The objective of this study is to conduct a comprehensive performance analysis of two different types of websites - a basic website and a more content-rich, "heavy"website - deployed using two different providers, Vercel and Cloudflare Pages, across three different regions: Asia (ap-southeast-1), Europ (eu-west-1), and North America (us-east-1).

The analysis will be focused on seven key performance metrics: First Meaningful Paint, First Contentful Paint, Cumulative Layout Shift, Max Potential FID, Speed Index, Interactive, and Server Response Time, which were discussed in detail in ch. 1.3.4.4. Each of these metrics provides a different perspective on the performance of the websites, with some focused on the initial rendering of the page, others on layout stability, and others on user interaction responsiveness.

Each subsection of this analysis will delve into the performance results for a specific region, analyzing each performance metric in turn. Within each metric, will be comparing the performance of the simple and heavy websites deployed using Vercel and Cloudflare Pages. This comparative analysis will provide a nuanced view of how the choice of website type, provider, and deployment region impacts website performance.

The performance results are presented through tables and visual charts, providing both a detailed and a high-level view of the performance landscape.

By the end of this analysis, there is an aim to provide a comprehensive understanding of how the performance of static websites varies depending on the website type, provider, and region. This knowledge will be invaluable for web developers and businesses aiming to optimize the performance of their websites and provide the best user experience.

3.1. REGION: ASIA

3.1.1. First Meaningful Paint

Beginning with the Asia region (ap-southeast-1), the data (Fig. 3.1) illustrates variances in FMP values across different times of the day, for both the basic and heavy websites, deployed using either Vercel or Cloudflare Pages.

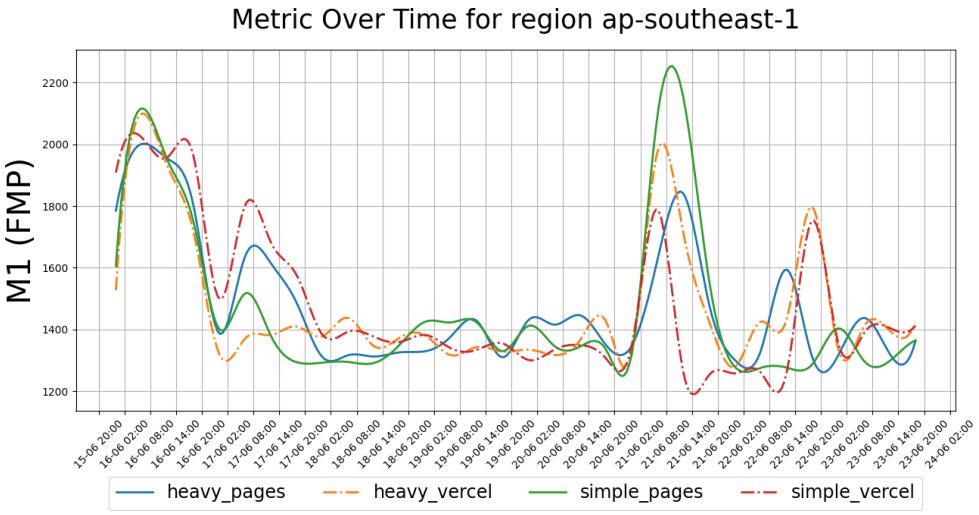
Upon analysis, it can be observed that for the "heavy"websites, the FMP values demonstrate a downward trend from the 16th to the 23rd of June, 2023. Specifically, the FMP for the heavy website on Cloudflare Pages drops from approximately 1785ms on the 16th of

June to roughly 1316ms on the 23rd of June. A similar trend is noted for the heavy website on Vercel, decreasing from about 1527ms to approximately 1338ms over the same period.

Comparatively, the "basic"websites show a slightly different trend. The FMP for the basic website on Cloudflare Pages is approximately 1606ms on the 16th and slightly increases to 1404ms on the 23rd, while the basic website on Vercel also displays an increase from around 1908ms to 1358ms in the same time span.

Therefore, it can be inferred from this data that "heavy"websites generally displayed an improved FMP over the observed period, which suggests optimizations or fewer demands on resources over time. On the other hand, the "basic"websites exhibited a slight increase in FMP, which could be due to additional content, increased server load, or other factors that may have affected the website's performance.

However, it is essential to note that these trends may not be universal across all regions or providers, given the inherent variability of website performance metrics due to network conditions, server load, and other factors. As such, these findings underscore the need for ongoing performance monitoring and optimization, particularly for websites aiming to provide an optimal user experience across diverse geographical locations and various deployment platforms. Figure 3.1 shows the trends mentioned.



Rysunek 3.1. First Meaningful Paint in Asia.

3.1.2. First Contentful Paint

The First Contentful Paint (FCP) data inside Figure 3.2 presented provides interesting insights into the performance differences between the two website types - "heavy"and "simple and the two deployment providers, Vercel and Cloudflare Pages, in the Asia region (ap-southeast-1). As with the First Meaningful Paint, smaller FCP values are indicative of better performance as this metric represents the time until the first piece of DOM content is painted, such as text or an image.

From the data, it is noticeable that the FCP times for both "heavy"and "simple"websites are generally comparable, with a range of approximately 1300 ms to 2100 ms across the given dates. There are no substantial or systematic differences that could suggest a clear superiority of one website type over the other.

Considering the performance across deployment providers. When focusing on the "heavy"websites, the FCP times for both Vercel and Cloudflare Pages fluctuate around similar values, with no clear trends. This suggests that for "heavy"websites with more content and images, the choice of deployment provider does not significantly impact the FCP time.

However, there are some days when significant differences appear, such as on June 21st, when Cloudflare Pages' FCP for "heavy"sites increases to approximately 2000 ms. On June 22nd, there is also a considerable spike for the Vercel-deployed "heavy"site, reaching nearly 1800 ms. These could be due to network or server issues on these specific days, or higher traffic in this area in general.

For "simple"websites, again, the FCP times fluctuate without a clear pattern. However, there are instances when Cloudflare Pages perform substantially better than Vercel, such as on June 21st, when the FCP time for Cloudflare Pages drops to around 1250 ms, compared to nearly 1800 ms for Vercel.

It is worth noting that, as the CDN solution is being tested here, some performance improvements could be due to the efficient distribution of images, especially for the "heavy"website. This could be an interesting point for further investigation.

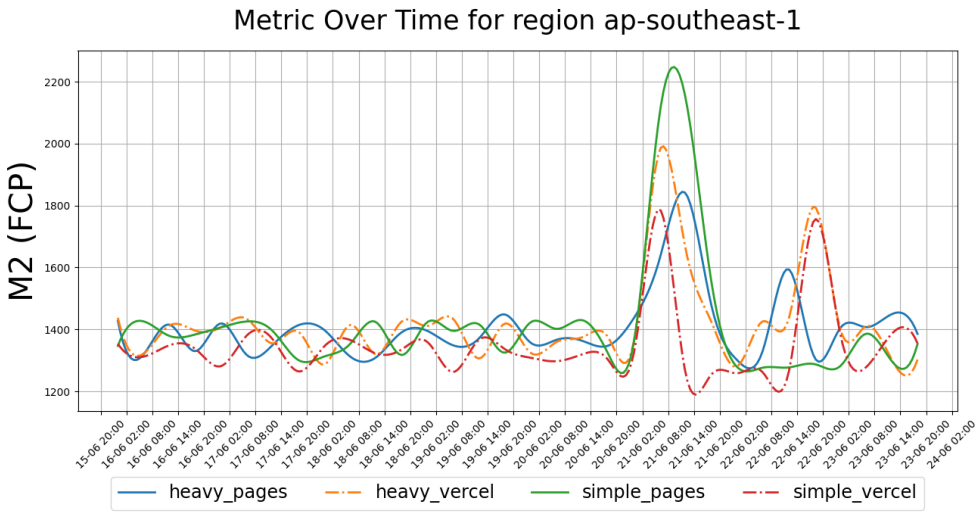
In summary, while there are day-to-day variations, overall, neither website type nor provider appears to have a consistently substantial impact on FCP times. Both website types and both providers show a range of FCP times that are broadly comparable. While there are moments when one provider performs significantly better than the other, these do not constitute a consistent pattern.

3.1.3. Cumulative Layout Shift

The Cumulative Layout Shift (CLS) data presented in Figure 3.3 helps in evaluating the visual stability of the two website types - "heavy"and "simple across the two deployment providers, Vercel and Cloudflare Pages, in the Asia region (ap-southeast-1). Lower CLS values are desirable as they indicate fewer unexpected layout shifts, leading to a better user experience.

For both the "heavy"and "simple"websites, there is a clear distinction in the CLS values. The "heavy"websites have higher CLS values, with most values falling in the range of approximately 0.13 to 0.35. This is expected given their higher content and image volume, which can lead to more frequent and significant layout shifts.

The "simple"websites, on the other hand, have consistently lower CLS values, ranging



Rysunek 3.2. First Contentful Paint in Asia.

mostly between 0.015 and 0.020. This suggests better visual stability, which can be attributed to their lesser content volume leading to fewer layout shifts.

Considering the deployment providers, for the "heavy"websites, the CLS values for Vercel and Cloudflare Pages are generally comparable, indicating similar levels of visual stability. There is, however, a significant spike in the CLS value for Vercel on June 21st, reaching about 0.35. This could suggest a specific issue or change at this time that led to an increase in layout shifts.

For "simple"websites, again, the CLS values are similar for both providers, suggesting comparable visual stability. It should be noted that the CLS values for the "simple"websites are consistent, with minor fluctuations across the period.

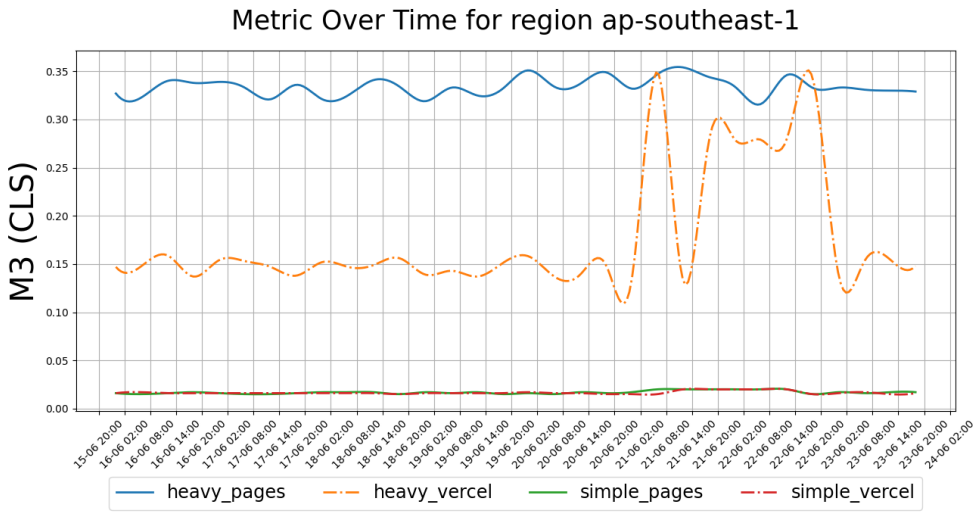
To summarize, the "heavy"websites have higher CLS values, indicating more frequent or significant layout shifts. The "simple"websites, with their lesser content volume, demonstrate better visual stability with consistently lower CLS values. There doesn't seem to be a significant difference between the deployment providers regarding the CLS values.

3.1.4. Max Potential FID

The Max Potential First Input Delay displayed in Figure 3.4 metric is used to measure the responsiveness of a web page. It indicates the longest duration a user might have to wait for the page to become responsive after their first interaction. Lower Max Potential FID values are desirable as they mean users would typically not have to wait long after interacting with the page for the first time.

In the given data, there are two types of websites - "heavy"and "simple hosted on two deployment providers, Vercel and Cloudflare Pages, in the Asia region (ap-southeast-1).

From the data, it's clear that "heavy"websites have considerably higher Max Potential FID values compared to "simple"ones, which is consistent with expectations. Heavy pages

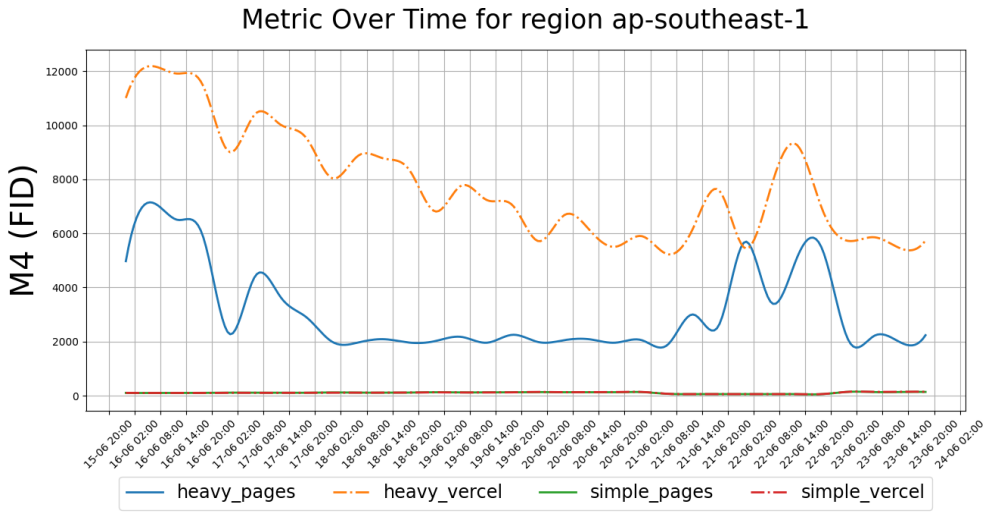


Rysunek 3.3. Cumulative Layout Shift in Asia.

generally have more content, which can increase the potential delay before the page becomes interactive. The Max Potential FID values for the "heavy"websites mostly fall in the range of thousands (around 2000 to 12000), while for the "simple"websites, the values are mostly in the range of hundreds (around 50 to 140).

Comparing the hosting providers, it appears that the Max Potential FID values are generally comparable for "simple"websites across Vercel and Cloudflare Pages. This suggests that the choice of deployment provider may not have a significant impact on this particular performance metric for the basic type of website.

There is a big difference when it comes to content-heavy websites. From the start of the tests, the value slowly shrinks, only showing increases from the 21st to the 23rd which is observable in other charts (3.1 - 3.7.



Rysunek 3.4. Max Potential FID in Asia.

3.1.5. Speed Index

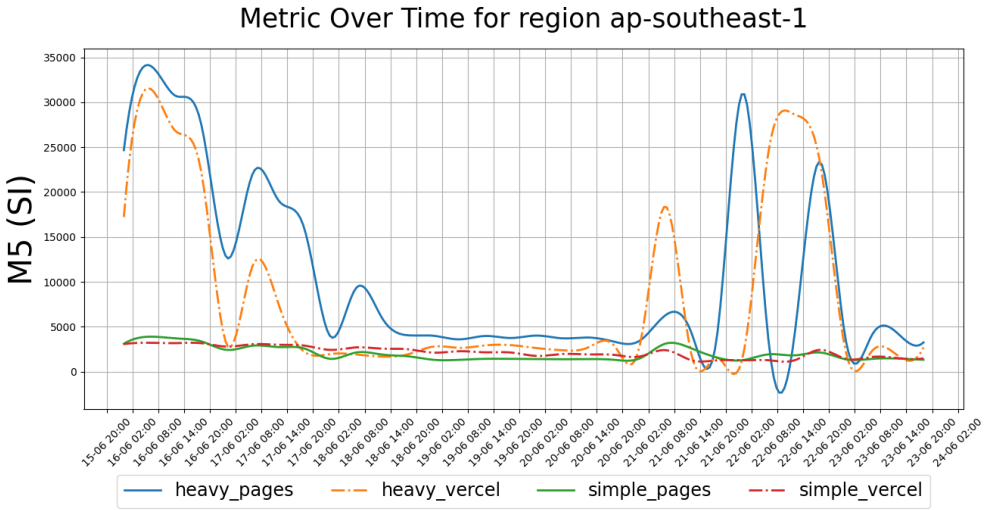
According to the Speed Index data (Fig. 3.5), there is a noticeable trend where both "heavy"and "simple"pages hosted on Vercel generally perform better (i.e., have lower Speed Index values) than those hosted on Cloudflare Pages.

The Speed Index for "heavy"pages fluctuates significantly during the observed period, reaching its highest values at the beginning of the measurement, then significantly dropping around 17th June, and slightly increasing again towards the end of the period.

On the other hand, the Speed Index for "simple"pages remains relatively steady throughout the period, indicating a more consistent user experience.

One exception to the general trend is on the 21st and 22nd of June when there are noticeable spikes in the Speed Index for "heavy"pages hosted on Vercel. This could indicate a temporary performance issue.

The data reinforces the importance of page complexity, as "simple"pages consistently have a lower Speed Index compared to "heavy"pages across both platforms, resulting in faster load times and potentially better user experience.

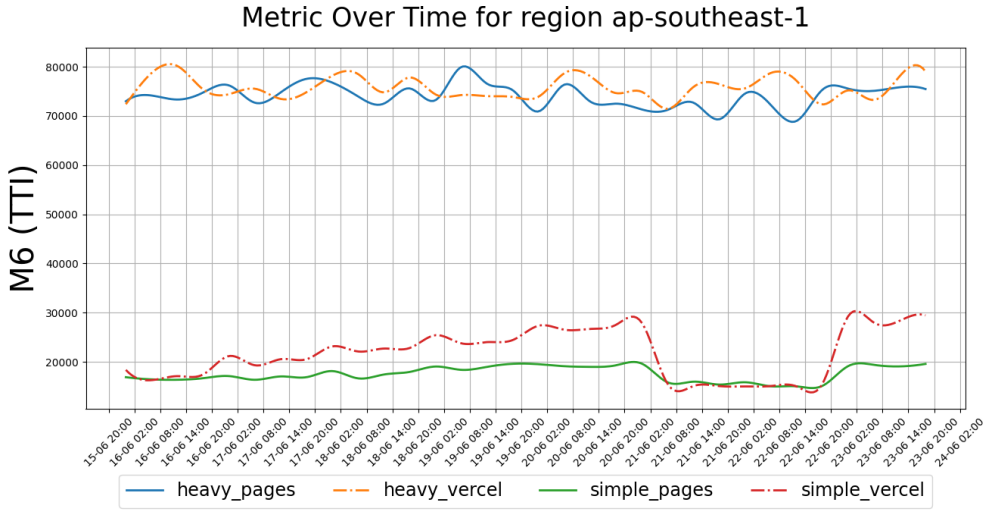


Rysunek 3.5. Speed Index in Asia.

3.1.6. Time to Interactive

The Time to Interactive Data (Fig. 3.6) for the "heavy"and "simple"websites hosted on Vercel and Cloudflare Pages in the Asia region (ap-southeast-1) shows interesting trends. The TTI for heavy pages fluctuates around a mean value of approximately 112000-120000 milliseconds across the hosting platforms, with a significant decrease on the 21st and 22nd June, followed by an increase on the 23rd. The TTI for simple pages is significantly lower than that for heavy pages, with less fluctuation over the course of the study. However, like the heavy pages, the TTI for simple pages also drops significantly on the 21st and

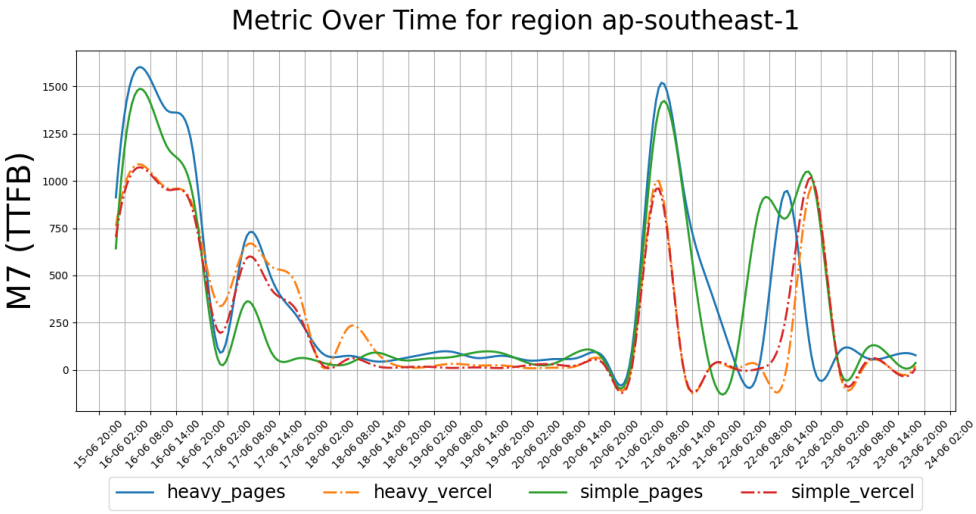
22nd June, and rises again on the 23rd. Comparing the hosting providers, both Vercel and Cloudflare Pages show similar trends for the TTI for both heavy and simple pages, suggesting comparable performance. Notably, the TTI values for Vercel are slightly lower than those for Cloudflare Pages in most cases, suggesting a slight performance edge for Vercel. The results highlight that both the complexity of the page (i.e., "heavy"vs "simple") and the hosting provider can impact the TTI significantly, but further analysis might be needed to determine the exact causes of the fluctuations observed in the TTI over time.



Rysunek 3.6. Time to Interactive in Asia.

3.1.7. Time to first byte

The Time to First Byte data (Fig. 3.7 shows significant variability for both the "heavy"and "simple"websites. The TTFB for the heavy pages varies widely, ranging from the low tens to nearly 1600 milliseconds, indicating inconsistency. However, a general decrease in TTFB is observed starting from June 17th, with the lowest values on June 18th and 19th, before it increased again on June 21st and fluctuated for the remaining dates. The TTFB for the simple pages also shows significant variability, with values ranging from around 30 to almost 1500 milliseconds. Like the heavy pages, the TTFB for the simple pages decreases significantly starting from June 17th but sees some increases on the 21st, 22nd, and 23rd. When comparing the hosting providers, both Vercel and Cloudflare Pages show similar trends and ranges for the TTFB for both heavy and simple pages. However, the TTFB for heavy pages on Vercel tends to be lower than on Cloudflare Pages, while for the simple pages, the TTFB values are quite comparable between the two providers. The variability and changes in the TTFB values over time could be due to many factors, including changes in server load, network conditions, and changes in the websites themselves. The significant drop in TTFB starting from June 17th might indicate some kind of optimization or change in the network conditions but would need further investigation to confirm.



Rysunek 3.7. Time to first byte in Asia.

3.2. REGION: EUROPE

3.2.1. First Meaningful Paint

From the FMP data (3.8), what can be observed is that both "heavy"and "basic"pages hosted on Vercel generally have lower FMP values compared to those hosted on Cloudflare pages in the Europe region, which implies a faster-perceived loading speed.

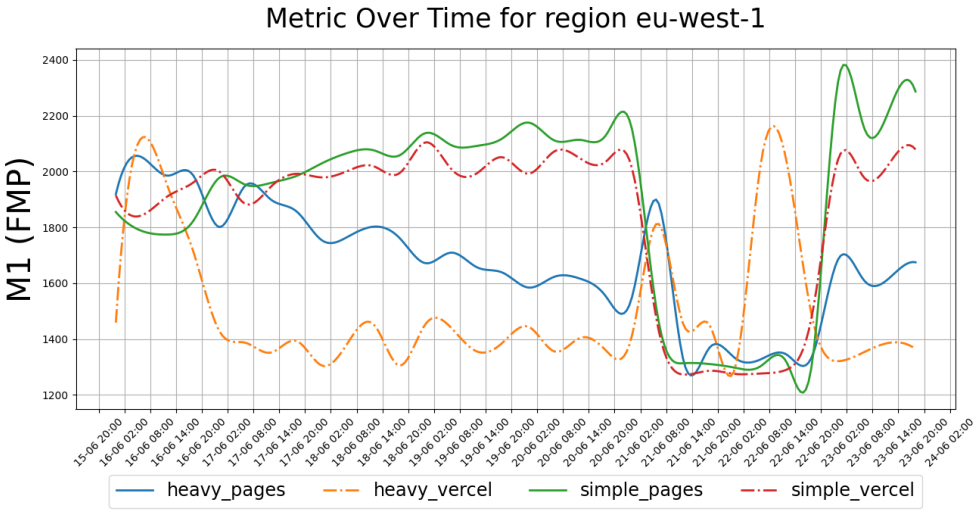
Similar to the Speed Index, the FMP for "heavy"pages exhibits more fluctuations over the observed period compared to "basic"pages. However, a significant decrease in FMP values is noticed around 21st June, which then gradually increases towards the end of the period.

In contrast, the FMP values for "basic"pages remain relatively steady, with a noticeable increase on 23rd June. This shows that basic pages generally provide a more consistent user experience in terms of perceived loading speed.

An exception is seen on 21st June, where there is a noticeable spike in FMP for "heavy"pages hosted on Vercel. This could indicate a temporary performance issue and requires further investigation.

In general, the FMP data emphasizes the significance of page complexity on perceived loading speed, as "basic"pages consistently outperform "heavy"pages in both hosting environments.

A clear pattern can be seen thru the studied charts. Because of that form now analysis will be provided in points.



Rysunek 3.8. First Meaningful Paint in Europe.

3.2.2. First Contentful Paint

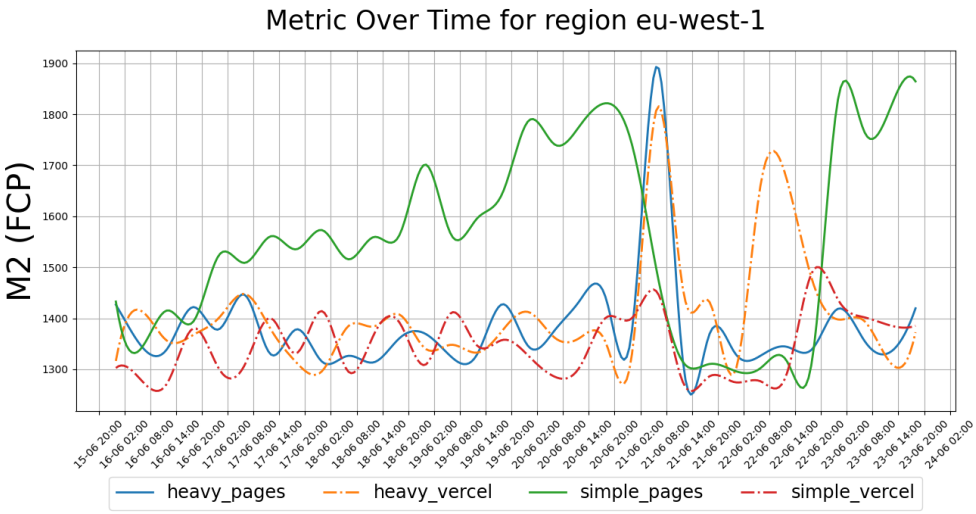
Analyzing the FCP data (Fig. 3.9) for the "heavy"and "simple"pages hosted on Cloudflare pages and Vercel in the Europe region, several points can be drawn:

- Both "heavy"and "simple"pages hosted on Vercel tend to have lower FCP values compared to those hosted on Cloudflare pages, suggesting faster rendering times.
- The FCP for "heavy"pages fluctuates more over the period compared to "simple"pages, indicating the complexity of the pages can affect the rendering speed.
- A significant decrease in FCP values can be observed around 21st June, which then gradually increases towards the end of the period.
- The FCP values for "simple"pages remain relatively stable, but there is a noticeable spike on 23rd June.
- On 21st June, there is an unusual spike in FCP for "heavy"pages hosted on Vercel, which could indicate a temporary performance issue.
- In general, "simple"pages consistently provide a faster rendering time compared to "heavy"pages in both hosting environments.

3.2.3. Cumulative Layout Shift

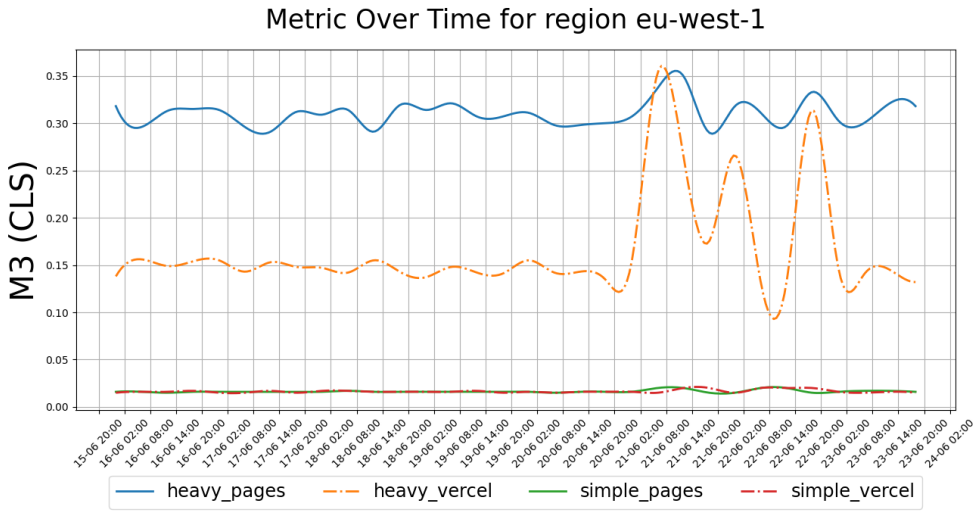
Here's a summarized analysis of the data (Fig. 3.10) from "heavy"and "simple"pages hosted on Cloudflare pages and Vercel in the Europe region:

- "Heavy"pages generally have a significantly higher CLS score compared to "simple"pages on both hosting platforms, indicating they are less stable and may provide a poorer user experience.



Rysunek 3.9. First Contentful Paint in Europe.

- The CLS scores for "heavy"pages hosted on Cloudflare pages are generally higher than those hosted on Vercel, suggesting that the Cloudflare pages hosted pages may have less visual stability.
- There's a significant increase in the CLS score for "heavy"pages hosted on Vercel on 21st June, which might indicate a temporary issue causing increased layout shifts.
- "Simple"pages on both platforms consistently have a low CLS score, which suggests a stable layout with less unexpected shifting for users.



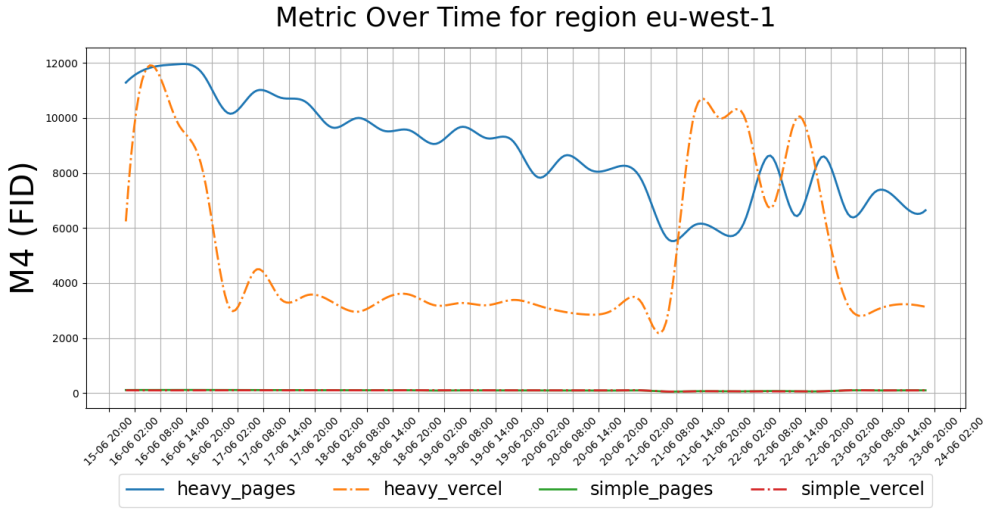
Rysunek 3.10. Cumulative Layout Shift in Europe.

3.2.4. Max Potential FID

In this analysis, comparison of the data (Fig. 3.11) for the Max Potential FID for both ’heavy’ and ’simple’ pages hosted on two platforms, Cloudflare pages (eu-west-1) and Vercel (eu-west-1).

- The Max Potential FID is significantly higher for ’heavy’ pages compared to ’simple’ pages across both hosting platforms, implying that users might experience longer interactivity delays on ’heavy’ pages.
- ’Heavy’ pages on Cloudflare pages consistently exhibit higher Max Potential FID scores than those on Vercel, suggesting that ’heavy’ pages hosted on Cloudflare pages could be less responsive to user inputs.
- There’s a noticeable dip in the Max Potential FID scores for both ’heavy’ and ’simple’ pages across the two platforms on June 21, likely due to a change in page design or improvement in server response times.
- Overall, ’simple’ pages, regardless of the hosting platform, have maintained a very low Max Potential FID, indicating quick responsiveness to user inputs.

In conclusion, while both ’heavy’ pages show room for improvement in their Max Potential FID scores, Cloudflare pages hosted pages may require additional optimization to enhance user interactivity.



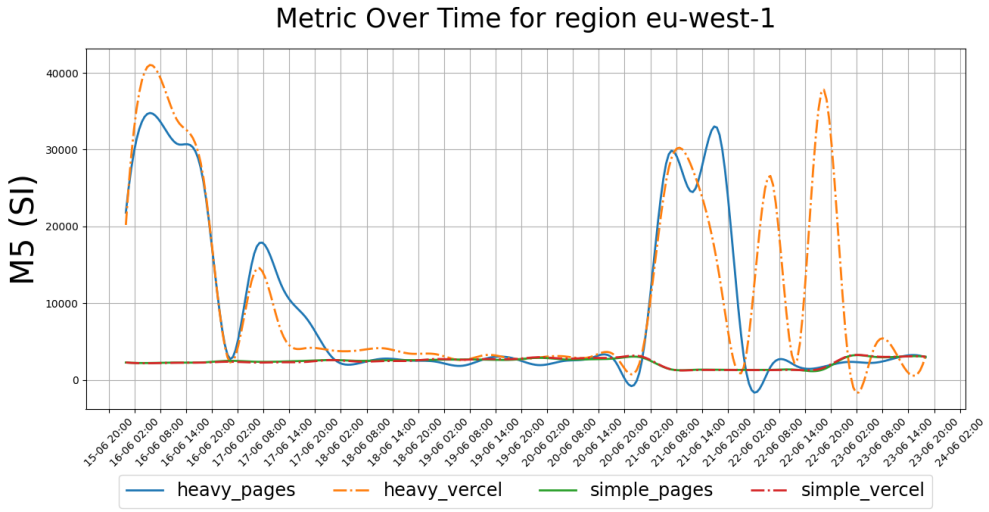
Rysunek 3.11. Max Potential FID in Europe.

3.2.5. Speed Index

Analyzing the Speed Index data (Fig. 3.12) for the "heavy"and "simple"pages hosted on Cloudflare pages and Vercel in the Europe region, several point can be drawn:

- Across all days, 'heavy' pages consistently show a higher Speed Index score than 'simple' pages on both Cloudflare pages (eu-west-1) and Vercel (eu-west-1). This suggests that users typically perceive 'heavy' pages as loading more slowly than 'simple' pages.
- On June 17, there's a significant drop in the Speed Index score for both 'heavy' and 'simple' pages on both platforms. This could indicate a change in the design of the pages, optimisation of their load time, or improved server response time.
- From June 21 onwards, there's a sharp rise in Speed Index scores for 'heavy' pages on both platforms, which suggests a slowdown in perceived loading time. This could be due to increased network congestion, a change in page design, or server-side issues.
- 'Simple' pages, regardless of the hosting platform, generally maintain a low Speed Index, implying a fast perceived loading time. However, there's a noticeable increase in their Speed Index scores from June 23 onwards, which may need further investigation.
- Between Cloudflare pages and Vercel, 'heavy' pages on Vercel occasionally show significantly higher Speed Index scores (e.g., on June 22 and 23). This could imply that users perceive these pages as loading more slowly compared to those hosted on Cloudflare pages.

In conclusion, while both 'heavy' and 'simple' pages show room for improvement in Speed Index scores, 'heavy' pages may require more significant optimization to enhance user perception of load time. Further, the platform used for hosting may also impact this metric, with Vercel sometimes performing less optimally for 'heavy' pages.



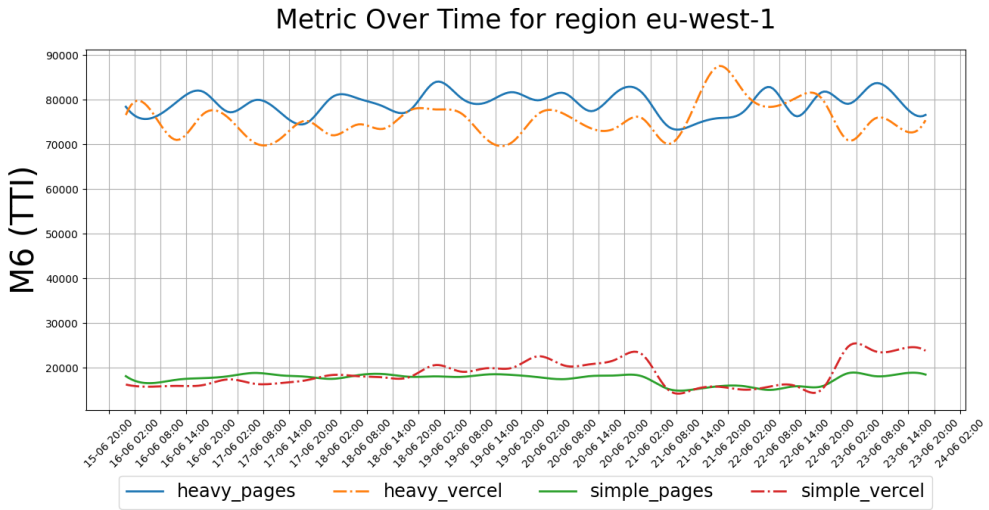
Rysunek 3.12. Speed Index in Europe.

3.2.6. Time to Interactive

Here's a summarized analysis of the data (Fig. 3.13) from "heavy"and "simple"pages hosted on Cloudflare pages and Vercel in the Europe region:

- The 'Time to Interactive' (TTI) metric, across all days and both platforms (Cloudflare pages and Vercel), shows a similar pattern as the Speed Index, with 'heavy' pages taking significantly longer than 'simple' pages. This suggests that 'heavy' pages may contain more elements or complex scripts that need to load before the page becomes fully interactive.
- On June 17, 'heavy' pages across both platforms display a noticeable dip in TTI scores, mirroring the trend observed in the Speed Index data. This could indicate improvements in the interactive readiness of the pages, perhaps because of more images being cashed by the CDN.
- From June 21, the TTI scores for 'heavy' pages experience a steep rise on both platforms. Interestingly, this happens simultaneously with a drop in the TTI scores for 'simple' pages, indicating some changes in the load and interaction behaviors of the pages on this date.
- 'Simple' pages generally maintain lower TTI scores, indicating a faster interaction readiness. However, they also show a noticeable increase in TTI scores from June 23 onwards, which merits further investigation.
- Vercel shows a tendency to have higher TTI scores than Cloudflare pages for 'heavy' pages (e.g., on June 22 and 23), suggesting that 'heavy' pages on Vercel may take longer to become interactive.

In conclusion, 'heavy' pages demonstrate a greater need for optimization to reduce their TTI scores and improve user experience. Similar to the Speed Index data, the hosting platform may also influence TTI, with Vercel occasionally showing less optimal performance for 'heavy' pages.



Rysunek 3.13. Time to interactive in Europe.

3.2.7. Time to first byte

In this analysis, there is a comparison of the Time to the first byte data (Fig. 3.14) for both 'heavy' and 'simple' pages hosted on two platforms, Cloudflare pages (eu-west-1) and Vercel (eu-west-1).

- The Time to First Byte (TTFB) metric, across all days and both platforms (Cloudflare pages and Vercel), shows a higher time for 'heavy' pages compared to 'simple' pages. This indicates that 'heavy' pages take more time to start rendering, which could be due to larger resource sizes or more server-side processing.
- Notably, TTFB for 'heavy' pages shows a steep drop on June 18 across both platforms, reaching its lowest point on June 19. This trend indicates a significant improvement in the start-render times of 'heavy' pages, perhaps due to server-side optimizations or improvements in resource delivery.
- On June 22, a sudden spike in TTFB scores for 'heavy' pages can be seen on Vercel, while it remains fairly low on Cloudflare pages. This could suggest issues on Vercel's side on this particular day, which increased the time it took for 'heavy' pages to start rendering.
- 'Simple' pages maintain lower TTFB scores throughout the given period, which indicates their faster start-render times. However, a substantial increase in TTFB scores for 'simple' pages is observed on June 22, particularly on Vercel, which might warrant further investigation.
- Comparing the two platforms, Cloudflare pages generally tends to have higher TTFB scores for 'heavy' pages, suggesting that 'heavy' pages start rendering slower on Cloudflare pages than on Vercel. However, Vercel's performance seems to be more variable, with sudden spikes in TTFB scores on certain days (e.g., June 22).

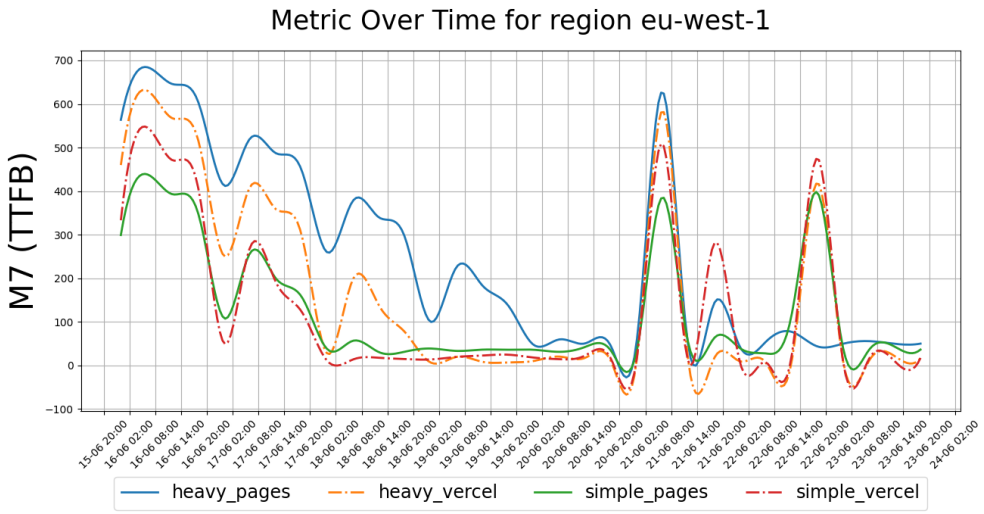
In summary, both the page type and the hosting platform significantly influence TTFB. 'Heavy' pages typically take longer to start rendering, and while Cloudflare pages consistently shows higher TTFB scores for these pages, Vercel's performance appears less consistent. Further investigation might be needed to identify the causes behind these trends.

3.3. REGION: USA

3.3.1. First Meaningful Paint

From the FMP data (3.15), observations are as follows:

- Across all days and platforms (Cloudflare pages and Vercel), 'heavy' pages consistently show higher FMP values compared to 'basic' pages. This indicates that 'heavy' pages



Rysunek 3.14. Time to first byte in Europe.

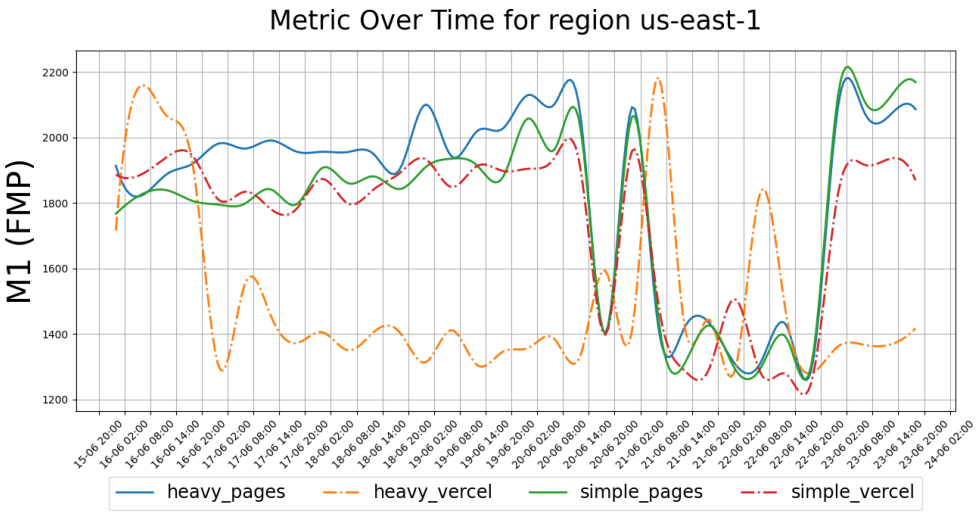
take longer to display meaningful content to users, possibly due to larger resource sizes or more complex rendering.

- On June 20, there is a notable drop in FMP scores for 'heavy' pages across both platforms. This suggests an improvement in the rendering performance of 'heavy' pages, resulting in a faster display of meaningful content.
- However, on June 18, there is a sudden decrease in FMP scores for 'heavy' pages on Cloudflare pages, while Vercel shows a relatively stable trend. This discrepancy could be due to variations in server-side optimizations or resource delivery between the two platforms.
- 'Basic' pages consistently maintain lower FMP values, indicating faster rendering and display of meaningful content to users. However, on June 20, there is a noticeable increase in FMP scores for 'basic' pages on both platforms, indicating a potential slowdown in rendering performance.
- Comparing the two platforms, Cloudflare pages generally exhibit higher FMP scores for both 'heavy' and 'basic' pages, suggesting that pages hosted on Cloudflare pages take longer to display meaningful content compared to Vercel.

3.3.2. First Contentful Paint

Analyzing the FCP data (Fig. 3.16) for the "heavy"and "simple"pages hosted on Cloudflare pages and Vercel in the US region, several points can be drawn:

- Across all days and platforms (Cloudflare pages and Vercel), 'heavy' pages consistently show higher FCP values compared to 'simple' pages. This suggests that 'heavy' pages take longer to render and display the first content to users, possibly due to larger resource sizes or more complex rendering.



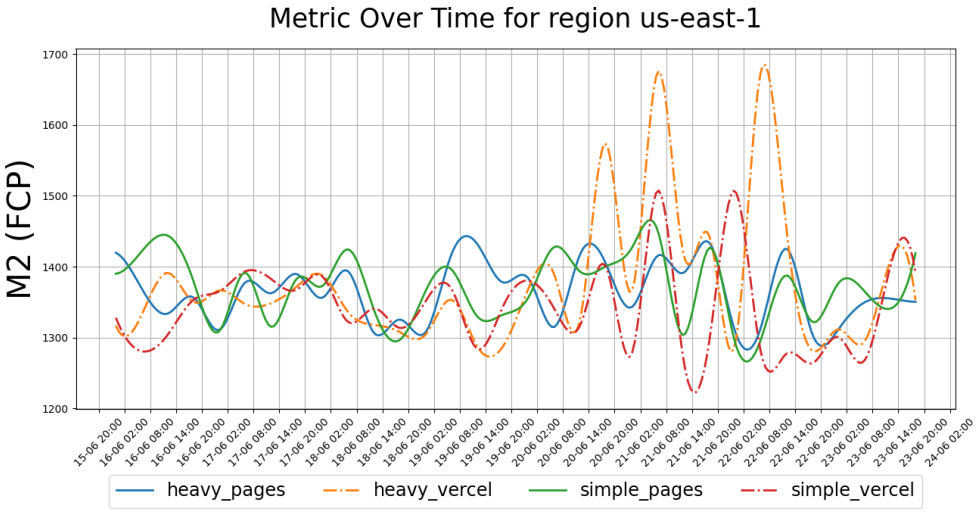
Rysunek 3.15. First Meaningful Paint in United States

- On June 20, there is a notable drop in FCP scores for 'heavy' pages across both platforms. This indicates an improvement in the rendering performance of 'heavy' pages, resulting in faster display of the first content.
- However, on June 18, there is a sudden decrease in FCP scores for 'heavy' pages on Cloudflare pages, while Vercel shows a relatively stable trend. This difference could be attributed to variations in server-side optimizations or resource delivery between the two platforms.
- 'Simple' pages consistently maintain lower FCP values, indicating faster rendering and display of the first content to users.
- Comparing the two platforms, Cloudflare pages generally exhibit higher FCP scores for both 'heavy' and 'simple' pages, suggesting that pages hosted on Cloudflare pages take longer to render the first content compared to Vercel.

3.3.3. Cumulative Layout Shift

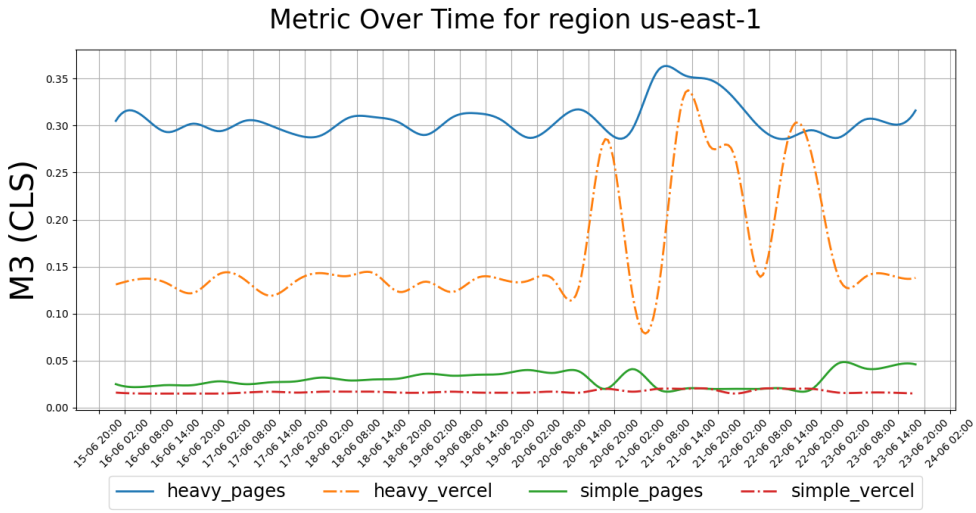
Here's a summarized analysis of the data (Fig. 3.17) from "heavy"and "simple"pages hosted on Cloudflare pages and Vercel in the US region:

- The Cumulative Layout Shift (CLS) metric measures the visual stability of a web page by calculating the sum of all individual layout shift scores.
- Across all days and platforms (Cloudflare pages and Vercel), 'heavy' pages consistently exhibit higher CLS values compared to 'simple' pages. This suggests that 'heavy' pages are more prone to layout shifts, which can lead to a less stable user experience.
- On June 21, there is a significant increase in CLS scores for 'heavy' pages on both platforms. This indicates a decrease in visual stability, potentially caused by layout shifts during page rendering.



Rysunek 3.16. First Contentful Paint in the United States.

- 'Simple' pages generally maintain lower CLS values, indicating better visual stability with fewer layout shifts during page loading.
- Comparing the two platforms, Cloudflare pages generally shows higher CLS scores for both 'heavy' and 'simple' pages, indicating a higher likelihood of layout shifts on Cloudflare pages-hosted pages compared to Vercel.
- It is important to note that CLS values can vary depending on factors such as the placement of dynamic content, image loading, and third-party scripts.

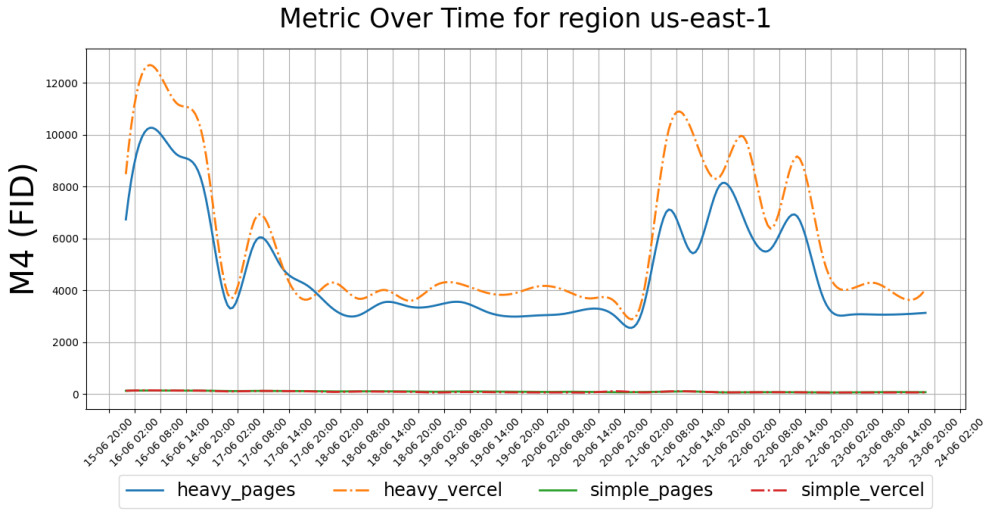


Rysunek 3.17. Cumulative Layout Shift in the United States.

3.3.4. Max Potential FID

In this analysis, data (Fig. 3.18) for the Max Potential FID for both 'heavy' and 'simple' pages hosted on two platforms, Cloudflare pages and Vercel in United States region is compared.

- The Maximum Potential FID metric measures the maximum delay between a user's first interaction and the response of the web page.
- 'Heavy' pages consistently have higher Maximum Potential FID values compared to 'simple' pages across all days and platforms (Cloudflare pages and Vercel). This suggests that 'heavy' pages are more likely to have longer delays in responding to user interactions.
- On June 21, there is a significant increase in Maximum Potential FID for both 'heavy' and 'simple' pages on Cloudflare pages, indicating a higher likelihood of delayed responses to user interactions.
- Comparing the two platforms, 'heavy' pages on Vercel generally have higher Maximum Potential FID values compared to Cloudflare pages, indicating a higher likelihood of long delays in responding to user interactions on Vercel-hosted pages.



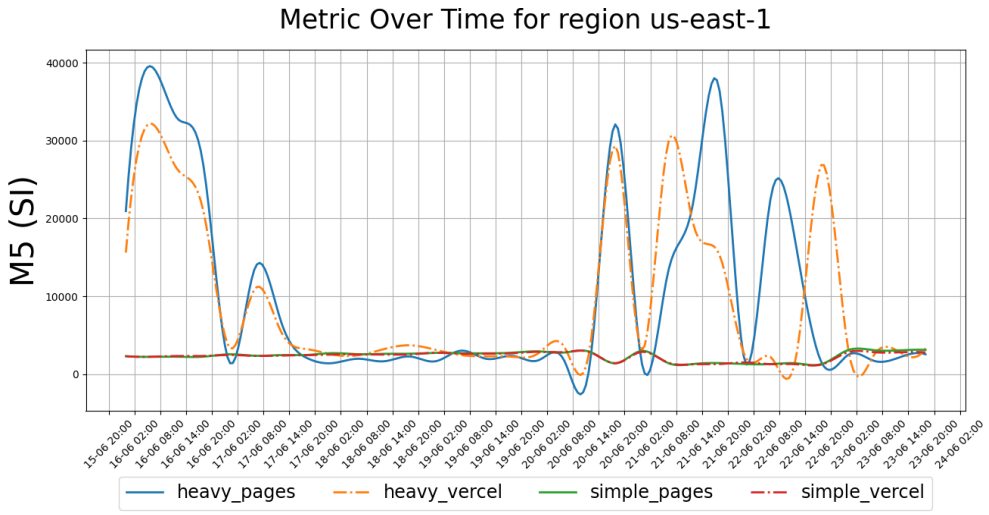
Rysunek 3.18. Max Potential FID in the United States.

3.3.5. Speed Index

Analyzing the Speed Index data (Fig. 3.19) for the "heavy"and "simple"pages hosted on Cloudflare pages and Vercel in the US region, several points can be drown:

- The Speed Index metric measures how quickly the contents of a web page are visibly populated.

- 'Heavy' pages consistently have higher Speed Index values compared to 'simple' pages across all days and platforms (Cloudflare pages and Vercel). This suggests that 'heavy' pages take longer to load and display their content to users.
- On June 20, there is a significant increase in Speed Index for 'heavy' pages on Cloudflare pages, indicating a higher loading time for these pages compared to other days.
- Comparing the two platforms, 'heavy' pages on Vercel generally have higher Speed Index values compared to Cloudflare pages, indicating a longer loading time for Vercel-hosted pages.
- It is important to note that Speed Index values can vary depending on factors such as page size, network conditions, and server response times.



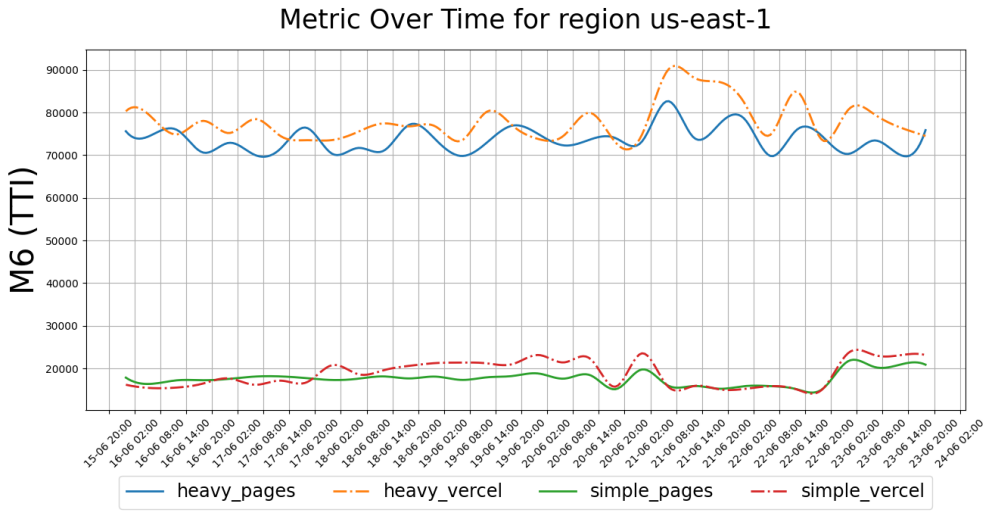
Rysunek 3.19. Speed Index in the United States.

3.3.6. Time to Interactive

Here's a summarized analysis of the data (Fig. 3.20) from "heavy"and "simple"pages hosted on Cloudflare pages and Vercel in the US region:

- The Time to Interactive metric measures the time it takes for a web page to become fully interactive, allowing users to interact with the page and perform actions.
- 'Heavy' pages consistently have higher Time to Interactive values compared to 'simple' pages across all days and platforms (Cloudflare pages and Vercel). This suggests that 'heavy' pages take longer to become fully interactive.
- On June 20, there is a significant decrease in Time to Interactive for 'simple' pages on Cloudflare pages, indicating a faster interactive experience for these pages compared to other days.

- Comparing the two platforms, 'heavy' pages on Vercel generally have higher Time to Interactive values compared to Cloudflare pages, indicating a longer waiting time for the pages to become fully interactive when hosted on Vercel.



Rysunek 3.20. Time to Interactive in the United States.

3.3.7. Time to first byte

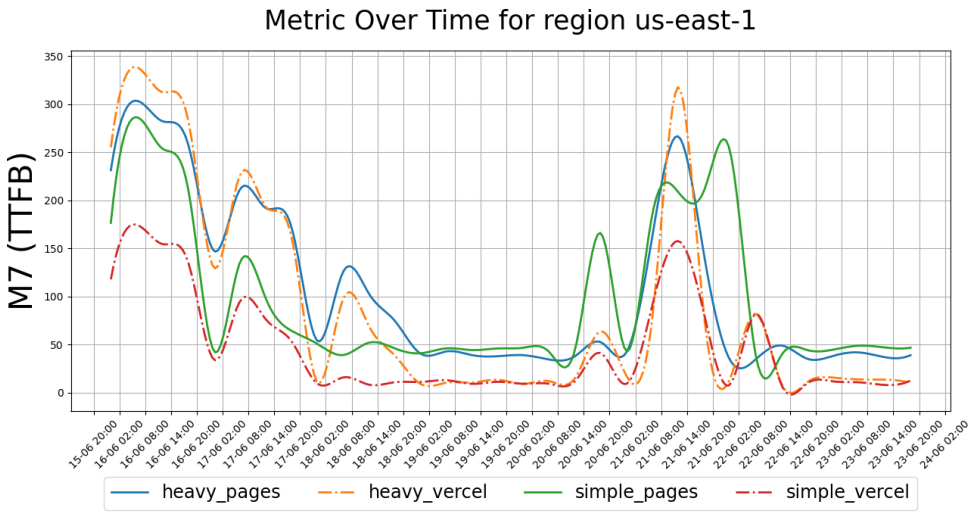
In this analysis, a comparison of the Time to the first byte data (Fig. 3.21) for both 'heavy' and 'simple' pages hosted on two platforms, Cloudflare pages and Vercel.

- 'Heavy' pages generally have higher TTFB values compared to 'simple' pages across all days and platforms (Cloudflare pages and Vercel). This suggests that 'heavy' pages take longer for the server to generate and send the initial response.
- On June 17, there is a significant decrease in TTFB for 'simple' pages on both Cloudflare pages and Vercel, indicating faster server response times for these pages compared to other days.
- Comparing the two platforms, TTFB values are generally higher on Vercel compared to Cloudflare pages, indicating potentially slower server response times when hosting pages on Vercel.

3.4. PERFORMANCE ON 21ST - 23TH OF JUNE

3.4.1. Performance Metrics

The metric that was not explained earlier, but is used in case of anomaly is the Performance Score (M8). It is a weighted average of several metric scores. It gives a high-level

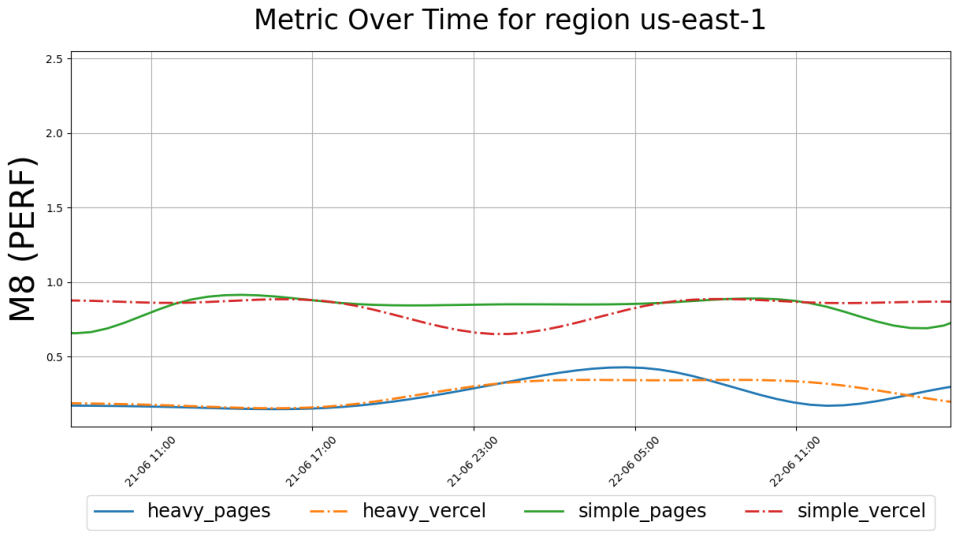


Rysunek 3.21. Time to first byte in the United States.

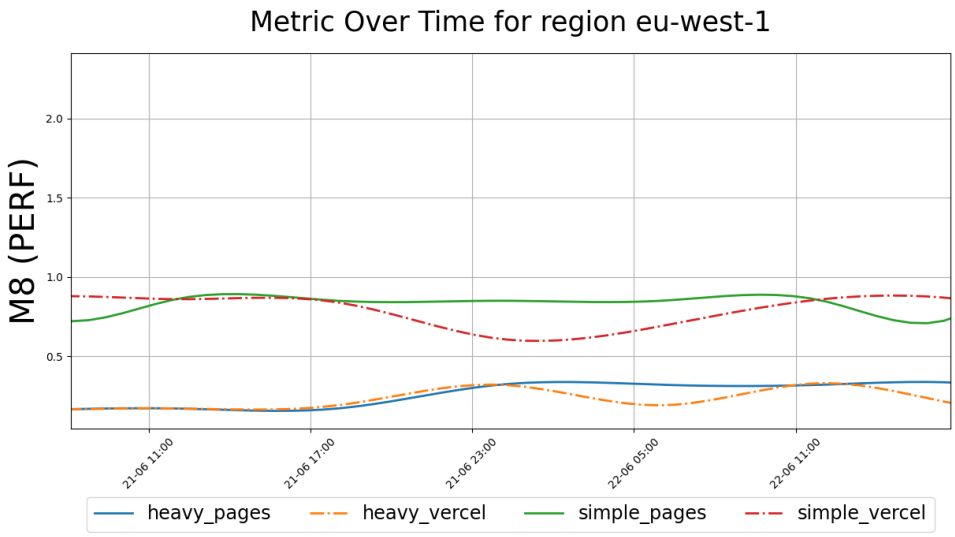
overview of a site's speed and performance. The score is measured on a scale from 0 to 1, with higher scores representing better performance. This part of the study is to try to investigate the anomaly, without trying to get too deep into it, as it requires a much wider time frame to get conclusions out of it. Still, it is worth mentioning.

3.4.2. Anomaly

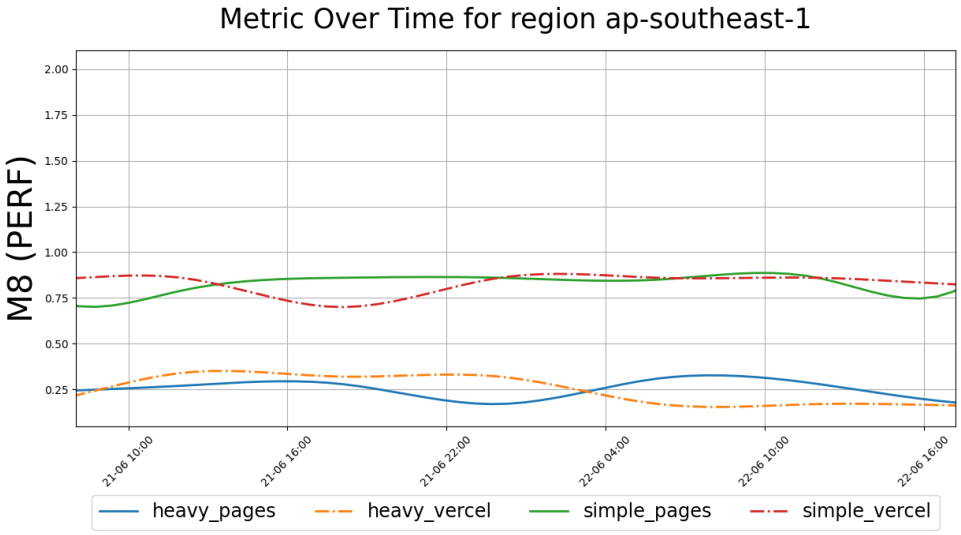
In this section, data was analyzed from the most critical moment of the study. Figures 3.22 - 3.24 display performance scores from three discussed regions. The overall score for heavy websites is 10% better in the Asia region compering to the EU and US. No crucial differences in the case of regions for basic websites were observed. Cloudflare Pages are more stable during this period. Amplifying better performance overall.



Rysunek 3.22. Performance score in the United States anomaly.



Rysunek 3.23. Performance score in Europe during anomaly.



Rysunek 3.24. Performance score in Asia anomaly.

4. CONCLUSIONS

4.1. OVERVIEW OF FINDINGS

This study sought to investigate the performance of Cloudflare Pages and Vercel, with a particular emphasis on user experience. The primary metrics of consideration (ch. 1.3.4.4) included e.g. Cumulative Layout Shift (CLS), Speed Index, and Time to First Byte (TTFB).

The analyses conducted in this research revealed several noteworthy patterns. One of the key findings is that Vercel tends to exhibit slightly superior performance compared to Cloudflare Pages as shown in Table 4.1, altho during the anomaly Cloudflare was a bit more stable (Fig. 3.22 - 3.24) than Vercel and also shown a bit better results. As mentioned before to conduct any concrete data from the anomaly much wider time frame of the test is required.

Tabela 4.1. Performance of Providers by Parameters and Regions

Metric	Cloudflare Pages			Vercel		
	Asia	EU	US	Asia	EU	US
CLS (unitless)	0,333	0,310	0,306	0,175	0,165	0,161
Speed Index (ms)	10464	9022	10051	8230	10642	9138
TTFB (ms)	368	243	106	260	161	91
Performance (0-1)	0,2245	0,2288	0,2206	0,2145	0,2040	0,2222

4.2. REGIONAL DIFFERENCES

An interesting trend observed in this study pertains to the performance differences across regions. The regions under investigation were Asia, Europe, and the United States. The results indicate that the performance scores in Asia were generally the highest, outperforming other regions by approximately 10%. This observation calls for a more detailed investigation into the factors that might be contributing to this regional disparity.

4.3. WEBSITE COMPLEXITY

The complexity of the websites hosted also played a significant role in determining the performance of the two platforms. For basic websites, there was no substantial difference in

performance between Cloudflare Pages and Vercel. However, for heavier websites, Vercel displayed better metrics, suggesting that it might be more suitable for hosting such websites.

Despite this, it's worth noting that Cloudflare Pages exhibited more stability during the anomaly that started on the 22nd. This resilience might imply that Cloudflare Pages could be a more reliable option for heavier websites in the face of unexpected events. Nevertheless, further research is warranted to solidify these observations.

5. FUTURE WORK

While the current research has provided insightful results about the performance of JAMstack websites across different hosting solutions and geographical regions, there is a wealth of opportunities for further exploration and study in this area. The findings of this research have significant implications not only for the future of web development but also for creating a robust testing method for future cases of testing web performance.

5.1. EXPLORATION OF ADDITIONAL SERVICES

The current study considered two hosting solutions: Vercel and Cloudflare Pages. However, there are numerous other services in the market that could be explored in future studies. These include both established platforms such as Netlify and AWS Amplify, and emerging technologies in the serverless computing space. An exploration of these services could provide a broader perspective on the performance and suitability of different hosting solutions for JAMstack websites.

5.2. CHANGES IN LOCATION

While this study is focused on three regions (ap-southeast-1, us-east-1, and eu-west-1), the expansion of the geographical scope to include other regions can provide a more comprehensive understanding of the performance of JAMstack websites globally. Particularly, regions with varying levels of Internet infrastructure development could present interesting contrasts and insights.

5.3. MORE EXTENSIVE JAMSTACK PAGES

The complexity of the websites tested in this study was fairly minimal. Future research could look into testing more complex JAMstack pages, including those with dynamic content and user interaction elements. This would provide valuable insights into how well different hosting solutions can handle the demands of more sophisticated JAMstack applications.

5.4. PRACTICAL APPLICATIONS

The findings of this study can have numerous practical applications. They can aid in the assessment of the efficiency of services offered by Internet service providers, and contribute to the development of strategic plans for implementing Internet services. Furthermore, these findings can serve as a basis for creating applications designed to test and monitor the performance of JAMstack websites across different hosting solutions and regions.

In conclusion, while this study has made significant strides in understanding the performance of JAMstack websites, the journey is far from over. The future holds great promise for further exploration and discovery in this rapidly evolving field.

5.5. FINAL REMARKS

In conclusion, this study provides useful insights into the performance dynamics of Cloudflare Pages and Vercel across different regions and website complexities. The findings underscore the need for further exploration and could guide future research in this area. While the choice between the two platforms depends on several factors, the insights derived from this study could serve as a valuable reference for website owners.

SUMMARY

This comprehensive study provides an in-depth analysis of the performance of two popular Content Delivery Network (CDN) providers, Cloudflare Pages, and Vercel, with a specific focus on website hosting across various geographical locations. The objective of this research was to understand the comparative performance of these CDN providers under different conditions and with the use of JAMstack architecture and CMS system, how they impact user experience, and the robustness of websites.

To achieve this, a range of performance metrics were considered, including Cumulative Layout Shift (CLS), Speed Index, Time to First Byte (TTFB), and an aggregate Performance Score. These metrics were obtained using Google's Lighthouse tool, a reliable and widely used tool for performance evaluation.

The study was conducted with two types of websites: basic and content-heavy. The results indicated that while there was little difference in the performance of the two providers for basic websites, Vercel offered superior metrics for content-heavy websites, suggesting a better user experience. However, Cloudflare Pages showed greater stability during an observed anomaly, indicating a possible advantage in robustness and resilience.

Tests were conducted on docker machines distributed in three different locations. The tests were conducted in a time frame of a week. Results were saved to S3 bucket.

In terms of geographical location, the performance of both providers was found to be best in Asia, with a performance increase of approximately 10% across all metrics. When it comes to basic websites there is no big difference when it comes to the provider of the CDN service. The difference was found when conducting content-heavy websites, with the conclusion that Vercel is better suited for this kind of website.

The study concludes with suggestions for future research, particularly in the context of the observed anomaly. More research is needed to determine whether the anomaly was a one-time occurrence or a regular event and to understand its impact on the overall performance of the providers.

In conclusion, this work provides valuable insights into CDN provider performance its influence on user experience, offering a valuable resource for developers and businesses aiming to optimize their website performance across different geographic regions. It contributes to the broader understanding of the performance of CDN providers and offers a solid foundation for further research in this area.

BIBLIOGRAFIA

[1] *Babel · babel*, https://babeljs.io/. (Accessed on 06/21/2023).

[2] *The best hosting options for your static site (for 2023) | spinal*, https://spinalcms.com/blog/static-site-hosting/. (Accessed on 06/21/2023).

[3] *Cdn hosting vs traditional web hosting - keycdn*, https://www.keycdn.com/blog/content-delivery-networks. (Accessed on 06/12/2023).

[4] *Cloud object storage – amazon s3 – amazon web services*, https://aws.amazon.com/s3/. (Accessed on 06/15/2023).

[5] *Cloudflare pages*, https://pages.cloudflare.com/. (Accessed on 06/15/2023).

[6] *Develop and deploy websites and apps in record time | netlify*, https://www.netlify.com/. (Accessed on 06/15/2023).

[7] *Docker: Accelerated, containerized application development*, https://www.docker.com/. (Accessed on 06/22/2023).

[8] *Dockerfile reference | docker documentation*, https://docs.docker.com/engine/reference/builder/. (Accessed on 06/22/2023).

[9] *The fastest frontend for the headless web | gatsby*, https://www.gatsbyjs.com/. (Accessed on 06/12/2023).

[10] *Flotiq - api-first headless cms*, https://flotiq.com/. (Accessed on 06/12/2023).

[11] *Fully managed container registry – amazon elastic container registry – amazon web services*, https://aws.amazon.com/ecr/. (Accessed on 06/22/2023).

[12] *Getting started in node.js - aws sdk for javascript*, https://docs.aws.amazon.com/sdk-for-javascript/v3/developer-guide/getting-started-nodejs.html. (Accessed on 06/21/2023).

[13] *Github pages | websites for you and your projects, hosted directly from your github repository. just edit, push, and your changes are live.*, https://pages.github.com/. (Accessed on 06/15/2023).

[14] *Global mobile traffic 2022 | statista*, https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/. (Accessed on 06/15/2023).

[15] *Googlechrome/lighthouse: Automated auditing, performance metrics, and best practices for the web.*, https://github.com/GoogleChrome/lighthouse. (Accessed on 06/12/2023).

[16] *Graphql | a query language for your api*, https://graphql.org/. (Accessed on 06/21/2023).

[17] *Lighthouse report viewer*, https://googlechrome.github.io/lighthouse/viewer/. (Accessed on 06/22/2023).

[18] *lighthouse/docs/variability.md at main · googlechrome/lighthouse*, https:

//github.com/GoogleChrome/lighthouse/blob/main/docs/variability.md#run-on-adequate-hardware. (Accessed on 06/22/2023).

[19] *Node.js*, https://nodejs.org/en. (Accessed on 06/21/2023).

[20] *Puppeteer | puppeteer*, https://pptr.dev/. (Accessed on 06/21/2023).

[21] *React*, https://react.dev/. (Accessed on 06/21/2023).

[22] *Secure and resizable cloud compute – amazon ec2 – amazon web services*, https://aws.amazon.com/ec2/. (Accessed on 06/15/2023).

[23] *Terraform by hashicorp*, https://www.terraform.io/. (Accessed on 06/15/2023).

[24] *Time to first byte (ttfb)*, https://web.dev/ttfb/. (Accessed on 13/06/2023).

[25] *Vercel: Develop. preview. ship. for the best frontend teams*, https://vercel.com/. (Accessed on 06/15/2023).

[26] *webpack*, https://webpack.js.org/. (Accessed on 06/21/2023).

[27] Barker, D., *Web content management: Systems, features, and best practices* ("O'Reilly Media, Inc.", 2016).

[28] Bhardwaj, S., Jain, L., Jain, S., *Cloud computing: A study of infrastructure as a service (iaas)*, International Journal of engineering and information Technology. 2010, tom 2, 1, str. 60–63.

[29] Buyya, R., Pathan, M., Vakali, A., *Content delivery networks*, tom 9 (Springer Science & Business Media, 2008).

[30] Cusumano, M., *Cloud computing and saas as new computing platforms*, Communications of the ACM. 2010, tom 53, 4, str. 27–29.

[31] Developers, G., *Cumulative layout shift (cls)*, "https://web.dev/cls/".

[32] Developers, G., *First contentful paint (fcp)*, "https://web.dev/fcp/".

[33] Developers, G., *First meaningful paint*, "https://web.dev/first-meaningful-paint/".

[34] Developers, G., *Max potential first input delay*, "https://web.dev/lighthouse-max-potential-fid/".

[35] Developers, G., *Speed index*, "https://web.dev/speed-index/".

[36] Developers, G., *Time to interactive*, "https://web.dev/interactive/".

[37] Dreibholz, T., Rathgeb, E.P., *On improving the performance of reliable server pooling systems for distance-sensitive distributed applications*, w: *Kommunikation in Verteilten Systemen (KiVS) 15. Fachtagung Kommunikation in Verteilten Systemen (KiVS 2007) Bern, Schweiz, 26. Februar–2. März 2007* (Springer, 2007), str. 39–50.

[38] Goel, U., Wittie, M.P., Steiner, M., *Faster web through client-assisted cdn server selection*, w: *2015 24th International conference on computer communication and networks (ICCCN)* (IEEE, 2015), str. 1–10.

[39] Gottschalk, K., Graham, S., Kreger, H., Snell, J., *Introduction to web services architecture*, IBM systems Journal. 2002, tom 41, 2, str. 170–177.

[40] Jin, H., Ibrahim, S., Bell, T., Gao, W., Huang, D., Wu, S., *Cloud types and services*, Handbook of cloud computing. 2010, str. 335–355.

[41] Jung, J., Krishnamurthy, B., Rabinovich, M., *Flash crowds and denial of service attacks: Cha-*

racterization and implications for cdns and web sites, w: *Proceedings of the 11th international conference on World Wide Web* (2002), str. 293–304.

[42] Lavrnić, I., i in., *Analyzing the potential mechanism for measurements-the most popular open source web content management system*, w: *Sinteza 2017-International Scientific Conference on Information Technology and Data Related Research* (Singidunum University, 2017), str. 85–89.

[43] Markovic, D., Scekic, M., *Understanding jamstack and its perception in web development*. 2021.

[44] Petersen, H., *From static and dynamic websites to static site generators*, university of TARTU, Institute of Computer Science. 2016.

[45] Qian, L., Luo, Z., Du, Y., Guo, L., *Cloud computing: An overview*, w: *Cloud Computing: First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings 1* (Springer, 2009), str. 626–631.

[46] Rahman, A., Mahdavi-Hezaveh, R., Williams, L., *A systematic mapping study of infrastructure as code research*, Information and Software Technology. 2019, tom 108, str. 65–77.

[47] Vakali, A., Pallis, G., *Content delivery networks: Status and trends*, IEEE Internet Computing. 2003, tom 7, 6, str. 68–74.

[48] Zammetti, F., Zammetti, F., *What is jamstack all about?*, Practical JAMstack: Blazing Fast, Simple, and Secure Web Development, the Modern Way. 2020, str. 1–17.

[49] Zur Muehlen, M., Nickerson, J.V., Swenson, K.D., *Developing web services choreography standards—the case of rest vs. soap*, Decision support systems. 2005, tom 40, 1, str. 9–29.

SPIS RYSUNKÓW

LIST OF LISTINGS

SPIS TABEL

Appendix

A. TERRAFORM CODE

```
variable "region" {
  description = "AWS region"
  type        = string
}

variable "s3_bucket" {
  description = "S3 bucket name"
  type        = string
}

variable "docker_image" {
  description = "ECR Docker image URL"
  type        = string
}

terraform {
  backend "local" {}
}

provider "aws" {
  region = var.region
}

resource "aws_ecs_cluster" "cluster" {
  name = "web-testing-performance"
}

resource "aws_ecs_task_definition" "task" {
  family                   = "testing"
  network_mode             = "awsvpc"
  requires_compatibilities = ["FARGATE"]
  cpu                      = "4096"
  memory                   = "8192"
  execution_role_arn       = aws_iam_role.ecs_task_execution_role.arn

  container_definitions = jsonencode([{
    name : "web-performance-testing",
    image : var.docker_image,
```

```
    essential : true,
    environment : [
      { "name" : "REGION", "value" : var.region },
      { "name" : "S3_BUCKET", "value" : var.s3_bucket }
    ]
  }])
}

resource "aws_iam_role" "ecs_task_execution_role" {
  name = "ecs_task_execution_role"

  assume_role_policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Principal": {
        "Service": "ecs-tasks.amazonaws.com"
      },
      "Effect": "Allow",
      "Sid": ""
    }
  ]
}
EOF
}

resource "aws_iam_role_policy_attachment"
 ↪  "ecs_task_execution_role_policy_attachment" {
  role       = aws_iam_role.ecs_task_execution_role.name
  policy_arn =
   ↪  "arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy"
}

resource "aws_cloudwatch_event_rule" "every_six_hours" {
  name                = "every-six-hours"
  description         = "Fires every six hours"
  schedule_expression = "cron(0 6,12,18,0 1-30 6 ? *)"
}

resource "aws_cloudwatch_event_target" "run_task_every_six_hours" {
  rule      = aws_cloudwatch_event_rule.every_six_hours.name
  target_id = "run-ecs-task"
  arn       = aws_ecs_cluster.cluster.arn
  role_arn = aws_iam_role.ecs_task_execution_role.arn

```

```
  ecs_target {
    task_count          = 1
    task_definition_arn = aws_ecs_task_definition.task.arn
  }
}
```

B. DOCKERFILE

```
# Start from a base image with Ubuntu
FROM ubuntu:20.04

ENV REGION us-east-1
ENV S3_BUCKET website-test-reports
# Install required packages
RUN apt-get update && apt-get install -y curl unzip wget gnupg cron

# Install Node.js
RUN curl -sL https://deb.nodesource.com/setup_16.x | bash - && \
    apt-get install -y nodejs

# Install AWS CLI
RUN curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
↪  "awscliv2.zip" && \
    unzip awscliv2.zip && \
    ./aws/install && \
    rm awscliv2.zip

# Install Google Chrome
# Install Google Chrome Stable and fonts
# Note: this installs the necessary libs to make the browser work with
↪  Puppeteer.
RUN apt-get update && apt-get install gnupg wget -y && \
    wget --quiet --output-document=-
    ↪  https://dl-ssl.google.com/linux/linux_signing_key.pub | gpg
    ↪  --dearmor > /etc/apt/trusted.gpg.d/google-archive.gpg && \
    sh -c 'echo "deb [arch=amd64] http://dl.google.com/linux/chrome/deb/
    ↪  stable main" >> /etc/apt/sources.list.d/google.list' && \
    apt-get update && \
    apt-get install google-chrome-stable -y --no-install-recommends && \
    rm -rf /var/lib/apt/lists/*

# Set the working directory in the container to /app
RUN mkdir -p ~/.aws

# Copy the .credentials file to the container
COPY .credentials /root/.aws/credentials
```

```
34
35 COPY crontab crontab
36
37 WORKDIR /app
38
39 # Copy all files
40 COPY ./ .
41
42 RUN mkdir -p /app/reports
43
44 # Copy the entrypoint script
45 COPY entrypoint.sh /usr/local/bin/entrypoint.sh
46
47 # Make the entrypoint script executable
48 RUN chmod +x /usr/local/bin/entrypoint.sh
49
50 # Set the entrypoint script as the entrypoint for the container
51 ENTRYPOINT ["/usr/local/bin/entrypoint.sh"]
```

C. NODE SCRIPT

```
import AWS from "aws-sdk";
import lighthouse from "lighthouse";
import puppeteer from "puppeteer";
import fs from "fs";
import util from "util";

const writeFile = util.promisify(fs.writeFile);

const s3 = new AWS.S3();

console.log("Starting Lighthouse test...");

const runLighthouseTest = async () => {
  const options = {
    output: "json",
    onlyCategories: ["performance"],
    port: 9222,
    preset: "desktop",
    "disable-full-page-screenshot": true,
    chromeFlags: [
      "--headless",
      "--disable-gpu",
      "--no-sandbox",
      "--disable-dev-shm-usage",
    ],
  };
  return await lighthouse(process.env.WEBSITE_URL, options, {
    extends: "lighthouse:default",
    settings: {
      onlyAudits: [
        "first-meaningful-paint",
        "first-contentful-paint",
        "cumulative-layout-shift",
        "max-potential-fid",
        "speed-index",
        "interactive",
        "network-server-latency",
        "server-response-time",
```

```
      ],
    },
  });
};

const runLighthouse = async () => {
  let chrome;
  let runnerResult;

  try {
    console.log("Launching Chrome headless...");
    // chrome = await chromeLauncher.launch({chromeFlags: ['--headless,
    ↪  --disable-gpu'], chromePath: '/usr/bin/google-chrome'});
    chrome = await puppeteer.launch({
      args: ["--no-sandbox", "--disable-gpu",
      ↪  "--remote-debugging-port=9222"],
      executablePath: "/usr/bin/google-chrome",
      headless: "new",
      ignoreHTTPSErrors: true,
    });

    runnerResult = await runLighthouseTest(chrome);
  } catch (error) {
    console.error("Error running Lighthouse test. Retrying...", error);
    if (chrome) {
      await chrome.close();
    }
    console.log("Launching Chrome headless...");
    chrome = await puppeteer.launch({
      args: ["--no-sandbox", "--disable-gpu",
      ↪  "--remote-debugging-port=9222"],
      executablePath: "/usr/bin/google-chrome",
      headless: "new",
      ignoreHTTPSErrors: true,
    });
    runnerResult = await runLighthouseTest(chrome);
  }

  await chrome.close();

  console.log("Writing report to file...");
  const currentTime = new Date().toLocaleString("pl-PL", {
    timeZone: "Europe/Warsaw",
  });
  const [date, hour, year, time] = currentTime.split(/[\/\s,:]+/);
  const reportName = `${process.env.WEBSITE_NAME}-${process.env.REGION}-` +
`${date.replaceAll('.','-')}-${hour}.json`;
```

```
  const reportJSON = JSON.stringify(runnerResult.lhr);
  await writeFile('./reports/' + reportName, reportJSON, 'utf8')
  console.log(`Report written to ${reportName}`);

  console.log("Uploading report to S3...");
  const s3Params = {
    Bucket: process.env.S3_BUCKET_NAME,
    Key: reportName,
    Body: reportJSON,
  };

  await s3.putObject(s3Params).promise();
  console.log("Lighthouse test complete!");
  fs.unlinkSync('./reports/' + reportName);
  console.log(`Report file ${reportName} removed`);

  console.log("Lighthouse test complete!");
};

runLighthouse().catch(console.error);
```