Summer 8-15-2023

ANALYZING THE SYSTEM FEATURES, USABILITY, AND PERFORMANCE OF A CONTAINERIZED APPLICATION ON CLOUD COMPUTING SYSTEMS

Anoop Abraham
aabra03@jaguar.tamu.edu

ANALYZING THE SYSTEM FEATURES, USABILITY, AND PERFORMANCE OF A CONTAINERIZED APPLICATION ON CLOUD COMPUTING SYSTEMS

A Graduate Thesis

by

ANOOP ABRAHAM

Submitted to Office of Graduate Studies
Texas A&M University-San Antonio
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2023

Major Subject: Computer Science

ABSTRACT

Analyzing The System Features, Usability, And Performance Of A Containerized Application On Cloud Computing Systems

August 2023

Anoop Abraham, B.S, Texas A&M – San Antonio

Graduate Thesis Chair: Dr. Jeong Yang

This study analyzed the system features, usability, and performance of three serverless cloud computing platforms: Google Cloud's Cloud Run, Amazon Web Service's App Runner, and Microsoft Azure's Container Apps. The analysis was conducted on a containerized mobile application designed to track real-time bus locations for San Antonio public buses on specific routes and provide estimated arrival times for selected bus stops. The study evaluated various system-related features, including service configuration, pricing, and memory & CPU capacity, along with performance metrics such as container latency, Distance Matrix API response time, and CPU utilization for each service. Easy-to-use usability was also evaluated by assessing the quality of documentation, a learning curve for beginner users, and a scale-to-zero factor. The results of the analysis revealed that Google's Cloud Run demonstrated better performance and usability when compared to AWS's App Runner and Microsoft Azure's Container Apps. Cloud Run exhibited lower latency and faster response time for distance matrix queries.These findings provide valuable insights for selecting an appropriate serverless cloud service for similar containerized web applications.

Dedication

I want to express my gratitude to Dr. Jeong Yang, my thesis advisor, for her unwavering support, invaluable knowledge, and extensive experience. She helped me navigate and overcome the numerous challenges I faced throughout this project and pushed me to achieve my very best. Thanks to her, I gained an abundance of academic and professional opportunities. I also want to acknowledge my parents and my elder brother Aneesh Abraham, whose unwavering support made it possible for me to attend college. Lastly, I am grateful to God for granting me the determination and ability to persevere in achieving this accomplishment.

Contents

Introduction

1.1 Cloud Computing

In recent years, cloud computing has gained significant popularity as an efficient and cost-effective approach to hosting applications. With its scalability, flexibility, and pay-per-use model, cloud computing offers businesses and organizations a viable alternative to traditional on-premises hosting solutions. Cloud Computing services provide resources based on virtualization[1] and containerization[2]. Modern VM technologies allow a single server to be divided into multiple virtual Containers [3, 4, 5, 6]. Cloud computing has become a critical component of modern business operations, offering flexibility, scalability, and cost savings. However, the reliability of cloud-based systems is a key concern for organizations that rely on these systems for mission-critical applications. Service disruptions and outages can have serious consequences, including lost revenue, damaged reputation, and decreased productivity. Despite the importance of reliability, there needs to be more comprehensive and unbiased information on the reliability of different cloud service providers. Organizations need help making informed decisions about which provider to choose, which can lead to costly service disruptions and outages.

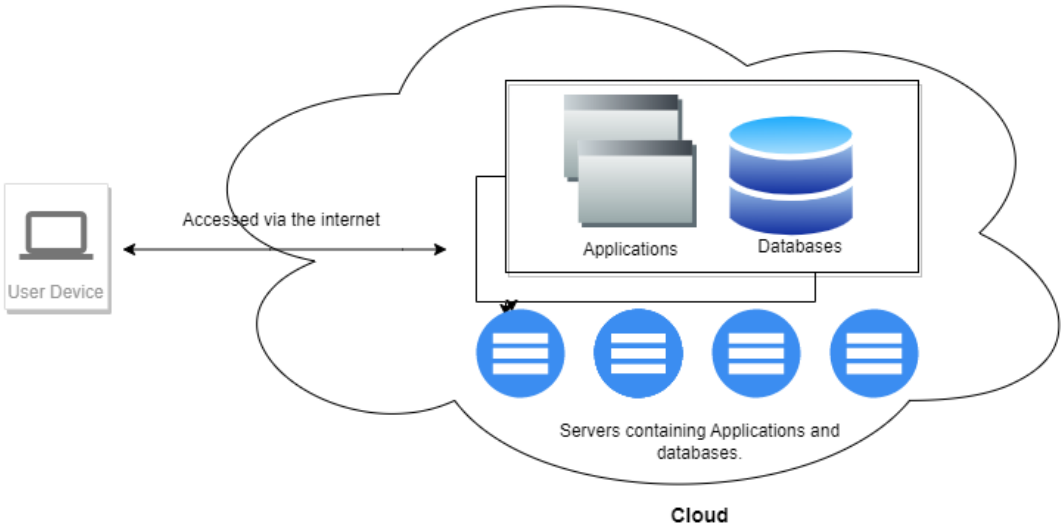

Figure 1.1: Serverless Cloud Computing.

1.2 Serverless Cloud Computing

Serverless computing is a next-generation service delivery approach in which service providers (SPs) offer just the resources required for the length of the requested services' execution[7, 8, 9]. Developers just need to build application code and deliver it to containers controlled by a cloud service provider[9]. The remainder will be handled by the cloud provider, who sets up the infrastructure required to run the code and scales it up and down on demand. Fig.1.1 shows a pictorial representation of a serverless cloud computing setup. Google Cloud, one of the leading cloud providers in the market, offers several hosting options, including serverless and server-based services.

1.3 Goals

This study aims to compare and evaluate serverless services offered by leading cloud providers, specifically Google Cloud Run, AWS App Runner, and Azure Container Apps, in terms of their features, pricing, scalability, performance, and ease of use for deploying and managing containerized applications. The goal is to determine the most suitable option for deploying a real-time vehicle location tracking web application and to identify any trade-offs or limitations of each service.

To conduct the comparative study, I will survey the existing literature on the topic and review the features and capabilities of serverless cloud services on AWS, GCP, and Azure. I will also conduct experiments to compare the performance and usability of running sample applications on all three services. Furthermore, I will investigate the support and documentation provided by each provider and the provider's reputation.

1.4 Contributions

In this thesis, a comparative analysis of different serverless services is presented, including a performance and usability study. These findings can aid future authors in discovering new research avenues for deploying a containerized application on a serverless cloud platform. The main contributions of this thesis are:

- A comparative analysis of the Cloud Run, App Engine, and Compute Engine services from Google Cloud Platforms for deploying containerized web applications with heavy use of Google Maps APIs.
- An overview of recent academic research related to serverless cloud computing from 2015 to 2023.
- A comparative analysis of the serverless service features offered by Cloud Run, App Runner, and Container Apps.
- A usability analysis of the serverless services offered by Cloud Run, App Runner, and Container Apps based on their ease of use and suitability for beginners in serverless computing.

- A performance analysis to deploy a containerized Django web application with heavy use of Google Maps APIs on the three serverless platforms: Cloud Run, App Runner, and Container Apps.
- Suggestions on the best serverless service to deploy containerized web applications with heavy use of Google Maps APIs.

1.5 Thesis Outline

The rest of this document consists of the following:

- **Chapter 2**: In this section, a comparison is made between three Google Cloud services - Cloud Run, App Engine, and Compute Engine - to determine their features and usability for deploying a containerized web application.
- **Chapter 3**: Provides the comparison results for serverless services from Google Cloud, Amazon Web Services, and Microsoft Azure.
- **Conclusions**: This section provides a summary of the research and results discussed in the previous two sections, highlighting the key findings.

A Comparative Analysis of Performance and Usability on Serverless and Server Based Google Cloud Services

This study compared the performance and usability of a containerized mo-bile-web application on serverless and server-based services in Google Cloud. The primary purpose of the application was to enable users to view the real-time location of San Antonio VIA buses on specific bus routes and obtain estimated arrival times for chosen bus stops when requested. The application was hosted in Google Cloud's serverless Cloud Run, App Engine, and server-based Compute Engine, and its latency and throughput were measured to analyze its performance. Google Cloud's Monitoring services were used to measure various metrics, including latency, throughput, and CPU utilization. Documentation and learning curve were evaluated for usability. Comparative analysis of these three Google services for hosting a containerized web application with heavy use of map-related APIs will assist individuals in selecting the appropriate cloud service for a for a similar type of application. The study results indicated that Cloud Run had the best performance and usability compared to App Engine and Compute Engine. It had the lowest latency of 47.24ms and highest throughput with sent and received rates of 9.47k/s and 11.15k/s, respectively. Additionally, it had the lowest container startup latency of 475.22ms and CPU utilization of 1%.

2.1 Introduction

The three most widely used cloud service offerings are IaaS, PaaS, and SaaS, also known as cloud service models or cloud computing service models, as shown in Fig.2.1. According to [10], IaaS, PaaS, and SaaS are not mutually exclusive and are often used in combination. Many mid-sized businesses use multiple types, and most large enterprises use all three.

Infrastructure as a Service (IaaS). IaaS is a cloud computing service model that provides virtualized computing resources over the internet[10, 11]. It enables customers to rent virtual machines, storage, and other computing resources on a pay-per-use basis instead of investing in and maintaining their own physical infrastructure. IaaS can be considered the original 'as a service' offering, as every leading cloud service provider, including AWS, Google Cloud, IBM Cloud, and Microsoft Azure, initially offered some form of IaaS. The benefits of IaaS

include higher availability, lower latency, improved performance, improved responsiveness, comprehensive security, and faster access to the latest technology [10]. Examples of IaaS offerings include AWS Elastic Computing Cloud (EC2), Microsoft Azure Virtual Machines, and Google Cloud Compute Engine.

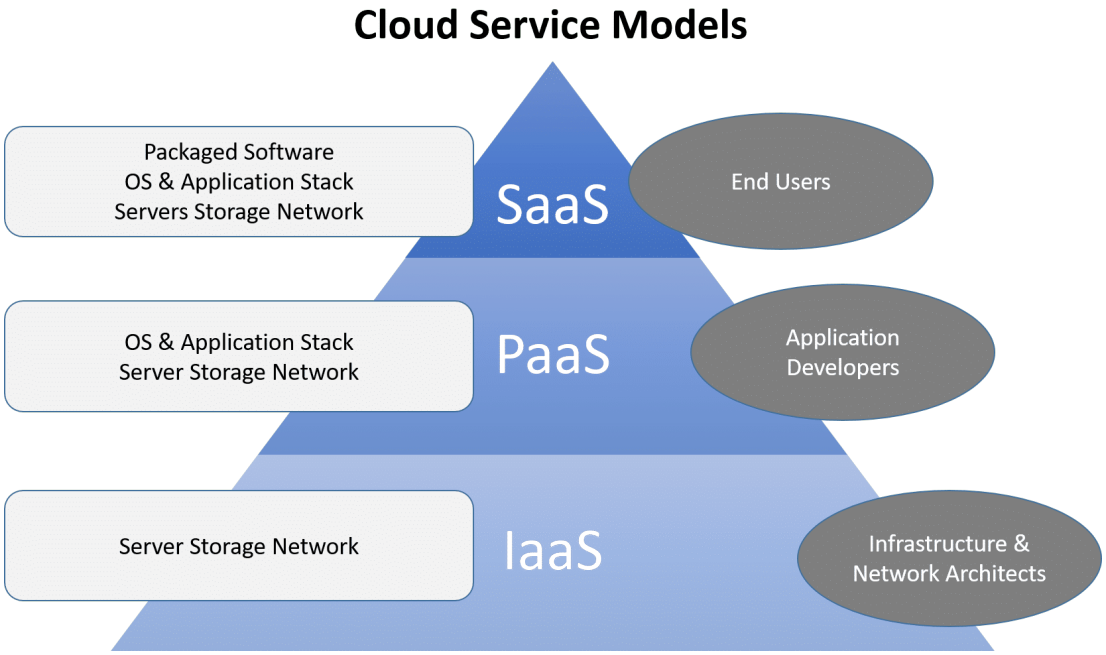

Figure 2.1: Cloud Service Models (IaaS, PaaS, SaaS)[8]

Platform as a Service (PaaS). PaaS provides a platform for developers to create and deploy applications without having to worry about the specific resources such as memory and processor required by their applications[10, 12, 13]. Some examples of PaaS offerings are AWS Elastic Beanstalk, Google App Engine, Microsoft Windows Azure, and Red Hat OpenShift on IBM Cloud[10, 12]. The primary benefit of PaaS (Platform as a Service) is the ability to build, test, deploy, run, update, and scale applications more efficiently and cost-effectively than traditional on-premises solutions.

Software as a Service (SaaS). SaaS (Software as a Service) is a type of cloud computing that enables users to access complete software applications through a web browser, desktop client, or mobile app[13, 14]. The software and all its infrastructure are hosted and managed by a SaaS vendor, which also manages all upgrades, patches, and security. This eliminates the need for the user to manage the infrastructure and software and enables them to access the application from anywhere with an internet connection [10]. SaaS offers several benefits, including minimal financial risk, easy scalability, and anytime/anywhere productivity. Examples of SaaS are Gmail, Google Drive, Dropbox, Slack, WebEx, and Zoom.

This paper presents a comparative analysis of the performance and usability of a containerized mobile-web application hosted on Google Cloud's serverless Cloud Run and App Engine and server-based Compute Engine. The study aims to provide insights into the performance and usability of using serverless and server-based Google services for hosting containerized applications and help individuals choose the appropriate cloud service for their hosting needs.

The rest of the paper is organized as follows: Section 2.2 summarizes some of the recent works in the field of cloud computing and talks about the different deployment models for cloud computing. Section 2.3 describes the three Google services used in this paper for comparison; Section 2.4 talks about the research methodology. Section 2.5 provides the analysis of the Google services explained in section 2.3, and Section 2.6 concludes the paper.

2.2 Related Work

In recent years, there have been several studies and surveys on serverless computing. For example, Zaid Al-Ali et al. proposed extending the scope of serverless computing beyond the currently limited framework to support a broader range of programming paradigms [15]. Mingyu Wu et al. conducted a comprehensive assessment of serverless computing, covering three areas: applications suited for serverless computing, performance challenges, and security concerns[16]. Theo Lynn et al. presented a list of parameters to evaluate different services from major primary cloud providers[7]. While various surveys exist on serverless computing, with different focuses and purposes, Yongkang Li et al. conducted a thorough review of the state-of-the-art challenges and opportunities of serverless cloud computing[17]. However, as cloud computing is a rapidly evolving field, some of the shortcomings identified in articles from two to three years ago may no longer be relevant in modern cloud computing.

Another study discusses Cloud Run in detail, including its pricing and the pricing of Cloud Build, a service frequently used with Cloud Run[18]. Another publication presents a tabulated comparative analysis of several function-as-a-service (FaaS) platforms provided by Google, AWS, and Microsoft[19]. There are numerous research studies that utilize Google Cloud to conduct their experiments and simulations. For instance, Yuping Shen in[20] leverages Google App Engine to host his study, while[21] also employs the same platform. The main reason for both researchers to opt for App Engine is due to its versatile and flexible nature. Laxmaiah et al. conducted a comparative analysis of Google App Engine, Amazon Web Services (AWS), and Microsoft Azure[22]. The study compared the services offered by these providers, focusing on aspects such as cloud services, runtime support, language support, and Service Level Agreement (SLA). In [23], Landoni, Marco, et al. describe how they utilized the Google Compute Engine to efficiently deploy and evaluate a Proof of Concept for their astrophysics study.

According to [24], Google Cloud Services is a cloud computing offering from Google that encompasses four main categories of services. The Computing category offers the Google Compute Engine, which provides users with the ability to select either Platform as a Service (PaaS) or Infrastructure as a Service (IaaS) to meet their specific needs. The Storage category includes Google Cloud Storage, a non-SQL, schema-less data storage solution, and Google Cloud SQL, which enables the processing of complex SQL queries and is connected to MySQL. In the Big Data category, users have access to fast and effective big data analysis through the highly scalable Big Query. Lastly, the Application Support category features a range of applications that allow for file viewing across multiple platforms, such as Gmail, Calendar, and Google Suite.

Google Cloud Run is a relatively new service from Google. It was first made

available in beta in April 2019. Research studies have utilized Google Cloud Run for its benefits in terms of scalability, security, and ease of deployment. For instance, [25] employs Google Cloud Run in its comparative study of the Container as a Service option and Kubernetes-based orchestration methods. The feasibility of using serverless containers in scientific computing is studied in [26]. The authors conclude that serverless containers can be effectively used for scientific workflows based on their experiments. They use CaaS offerings from various cloud service providers, including AWS Far-gate, Azure Container Instances, and Google Cloud Run, which is the one they use. K. Burkat et al. evaluated the capabilities of elastic containers and their suitability for scientific computing in the context of scientific workflows using AWS Fargate and Google Cloud Run infra-structures [26].

Our study aims to perform a comparison of three Google Cloud services – server-based Compute Engine and serverless Cloud Run and App Engine- to determine the most appropriate service for deploying a real-time vehicle location tracking web application.

2.3 Google Cloud Services

2.3.1 Cloud Run

Cloud Run is Google's serverless computing platform, introduced in August 2018. It is a fully managed serverless platform for letting users develop and deploy an application from a container image or directly from source code (and Google provides the container). Cloud Run comes with the following services: Traffic Management, Auto Scaling, Security, and Effective Pricing.

Traffic Management. Every deployment creates a new immutable revision, and customers can customize how incoming traffic is routed to their revisions. Customers can route incoming traffic to the latest or previous revision or even split traffic to multiple revisions simultaneously. Also, customers can even fine-tune the percentage of incoming traffic to each revision.

Auto Scaling. Cloud Run adds and removes container instances automatically to handle all incoming requests. Cloud Run is capable of what is commonly referred to as scale to zero: if there are no incoming requests to the service, even the last remaining container instance will be removed. This behavior is enabled by default: the default number of minimum instances is 0. Note that several minimum instances of 0 may result in cold starts, where latency is proportional to the time it takes for your container to start. Customers should set the number of minimum instances to at least one if their container takes more than 10 seconds to start because Cloud Run does not keep requests pending for longer than 10 seconds [8].

Security. A Cloud Run service can be reachable from the internet, and customers can restrict access by configuring a security policy. By default, all incoming traffic is allowed access.

Pricing. With Cloud Run, customers are charged for the CPU and memory allocated to a container instance rounded up to the nearest 100 milliseconds. With scale to zero (the default configuration), customers are not charged if their service is unused. There are two pricing models customers can enable: request-based or

instance-based. With request-based, if a container instance is not processing requests, customers are not charged, but when it is, customers pay a per-request fee. With instance-based, customers are charged for the entire lifetime of a container instance, but there is no per-request fee.

Free Tier. Google has a generous free tier: the first 180,000 vCPU-seconds per month are free, the first 360,000 GiB-seconds per month are free, and the first 2 million requests per month are free. Also, requests are only billed when they reach the container after successfully being authenticated. Requests denied by customers' security policies are not billed. When executing a build using Google's Cloud Build service with Cloud Run, the first 120 build-minutes per day are free[18]. Thereafter, the pricing varies but appears to start at $0.003 per build-minute.

2.3.2 App Engine

App Engine is Google's platform-centric solution as a type of PaaS. With App Engine, customers do not need to buy, build, or operate hardware infrastructure [27]. Customers need to select from either the App Engine flexible environment or the App Engine standard environment to run their applications in App Engine. They can also use both environments simultaneously. App Engine standard environment supports source code written in specific versions [27] of Python, Java, Node, PHP, and Go. The standard environment is suitable for applications that need to handle sudden and extreme traffic spikes. App Engine also can scale to zero. This helps reduce the cost incurred when the application is not serving any request.

App Engine Flexible environment deploys and manages the applications instances in a docker container on a Compute Engine Virtual Machine (VM). This option is suitable for applications that don't anticipate sudden spikes in traffic and need to scale up or down gradually. According to Google [27], the flexible environment is suitable for applications that satisfy the following characteristics.

- Source code that is written in a version of any of the supported programming languages: Python, Java, Node.js, Go, Ruby, PHP, or .NET.
- It runs in a Docker container with a custom runtime or source code written in other programming languages.
- Uses or depends on frameworks that include native code.
- Accesses resources or services of your Google Cloud project that reside in the Compute Engine network.

2.3.3 Compute Engine

Compute Engine is a server-based cloud computing service offered by Google that provides scalable computing resources for running virtual machines (VMs). It acts as a type of IaaS. With Compute Engine, users can create and run VMs in Google's data centers. It provides a high-performance infrastructure for running a variety of workloads, including large-scale data processing, complex scientific simulations, and web-based applications. The service is designed for enterprises,

startups, and research organizations that need access to scalable computing resources for their computing needs.

Compute Engine offers a range of features that make it an attractive choice for businesses and organizations. It provides flexible and scalable virtual machines, automatic load balancing, and real-time scaling, which makes it easy to handle sudden spikes in demand. Additionally, Compute Engine integrates with other Google Cloud services, such as Google Cloud Storage, Google BigQuery, and Google Kubernetes Engine, making it possible to build complex, multi-tiered applications. This integration helps organizations to manage their infrastructure more efficiently and reduces the time and effort required to deploy new applications and services.

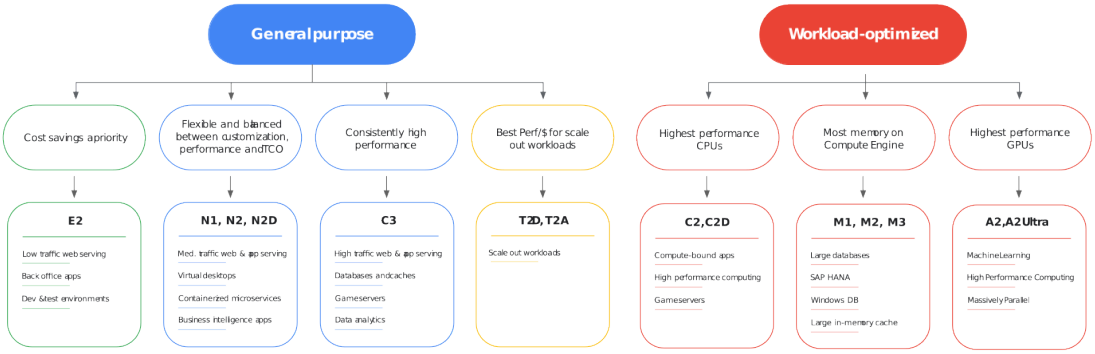

Figure 2.2: Different virtual machine types offered by Google as per[28].

GCE offers several instance types with varying levels of CPU, memory, and local storage to cater to different computing needs. GCE also provides options for users to configure the network, firewall, and access scopes for their instances. Furthermore, GCE provides pre-configured images for popular operating systems and application environments, making it easier to quickly deploy and run applications. In addition, GCE also offers custom machine types, sole-tenant nodes, and committed use contracts to meet the specific requirements of businesses and organizations. Fig. 2.2 illustrates the various offerings from Google, suited for specific purposes [28]. These offerings are straightforward virtual machine options from Google, providing users with greater control over the machine compared to other services.

2.4 Performance Evaluation Methodology

An experiment was employed with a containerized SmartSAT Django mobile-web application for evaluating the performance of the application on various serverless and server-based services available on Google Cloud. The application was slightly altered in its configuration aspect to enable its deployment on each service: serverless Cloud Run and App Engine and server-based Compute Engine. This application was chosen for its ease of deployment, as it is already developed and containerized as part of the project [29], making it compatible with Google's Cloud Run, App Engine, and Compute Engine services.

The SmartSAT application aims to provide critical services to San Antonio transit users through the San Antonio Transit [29]. With a user-friendly inter-

face, transit users can access the status of all running VIA buses on their preferred bus route. The application currently supports around ten VIA bus routes. However, only three routes were used as part of this experiment. Google Maps JavaScript API [29] was utilized to showcase the real-time location of the buses on the selected route. Furthermore, the SmartSAT application has the capacity to calculate and display the estimated arrival time at a selected bus stop on the route. The Google distance matrix API was used for this purpose, taking into consideration various factors such as traffic patterns, road accidents, and other elements that may impact the trip time.

Through the app, users can access the bus routes from the home screen of the application, where they can select their desired route. The application will then load all the bus stops on that route, including the order in which they appear. By clicking on any of the bus stop markers, users can view the estimated arrival time for that stop.

This application uses a Postgres database in Google Cloud SQL to maintain the data related to users and bus routes. The experiment was performed by hosting the application on the Cloud Run, App Engine, and Compute Engine services in Google Cloud. Apart from specific services-related configuration changes, the application used the same source code in all three deployed services. Docker was used to containerize the application. The latest version of the Compute Engine can directly deploy a container to it. This uses an optimized operating system called Container Optimized OS. The configuration details for each machine used in all three services are detailed in the evaluation section.

Below is a brief description of some of the performance matrices measured during the experiment.

- **Latency** is the time traveled by the request to and back from the server. This is also known as Network Latency or Network Delay.
- **Throughput** is the number of transactions per unit of time an application can handle. Generally denoted in requests per second (RPS), Transactions per second (TPS), hits per second, etc.

An Android emulator application was used to create the GPS movement of a bus in a selected bus route. The following steps were performed on each de-ployed service to measure the performance.

1. Navigate to the deployed service link from the Android emulator and log in to the website as a bus driver.
2. Start two different bus routes. (This will emulate the same scenario when a real bus moves in that route.)
3. Access the application as a normal user (Bus Rider) from different devices.
4. Click on a different bus stop icon so the website will show the Estimated Arrival time for that stop.
    (a) Repeat this task for both the active routes from different tabs opened in the web browser. This will mimic multiple people accessing the application.

(b) Once both buses reach their destination stops, stop the bus driving from the driver screen.

5. Navigate to the cloud monitoring option in Google Cloud for the corresponding service and note down the results.

2.5 Results Evaluation

This section presents the results of the comparative analysis of google cloud's serverless and server-based services in terms of performance and usability of the service. The usability of the services is associated with service configuration, pricing, and relevant system features as well as documentation and learning, and the performance of a containerized mobile app application featuring applications Latency, Throughput, and CPU utilization on each of the services. The containerized web application [29] was used to measure the application's latency, throughput, and CPU utilization when the application is hosted in Google's Cloud Run, App Engine, and Compute Engine. Table 1 and Table 2 show the results of the analysis. The numbers in the tables are mostly sourced from configuring the services.

2.5.1 Service Configuration, Pricing, and Features

The results in Table 1 focus on comparing the service configurations, pricing, and features. These services offer different configurations and pricing models, making them suitable for different types of applications and use cases. Cloud Run is a relatively new service and is designed for serverless deployment of containerized applications. It has a request-based and instance-based pricing model, with the first 120 minutes of build time per day being free. The instance used in this experiment had a memory capacity of 0.5 GB per container and a single. However, it has a high rating for documentation and a learning curve, making it easy for beginners to get started.

App Engine, on the other hand, has many instance classes, which can be selected based on the computing requirements of the application. The cost of using the service is $0.031611 per hour, and it has a moderate memory capacity of 256 MB. The documentation and learning curve for App Engine is rated as 3.5/5, indicating that it is not as straightforward as Cloud Run, but still easy to follow with adequate documentation available.

Compute Engine is a flexible and scalable computing service that provides virtual machines for running applications. It has a vCPU-based pricing model with a cost of $0.021811 per vCPU hour and offers a free tier with a limit of time. It has the highest memory capacity of 4 GB among the three services and 2 CPUs. However, it has a lower rating for documentation and learning curve, with a rating of 3.5/5 indicating that it may take more effort for users to configure the entire setup, especially for beginners.

Table 2.1: Comparison results on service configuration, pricing, and features.

Service Configuration	Google Cloud Run (Serverless)	Google App Engine (Serverless)	Google Compute Engine (Server-based)
Instance type	-	F1	e2-medium
No of CPU's	1	600 MHz	2
Memory	0.5GB / container	256 MB	4GB
Pricing			
Compute Time Unit	100 ms	per hour per instance	per vCPU hour
Pricing Models	request-based, instance-based	Many instance classes as shown on [30]	On Demand, Spot Price
Build Fee	First 120 minutes/day are free, pricing varies thereafter	$0.031611 per hour (us-central-1)	$0.021811 / vCPU hour (Predefined vCPU model)
Free Tier	First 180,000 vCPU-seconds/month, first 360,000 GiB-seconds/month, 2 million requests/month [31]	28 hours per day of "F" instances, 9 hours per day of "B" instances, 1 GB of egress per day [31]	Free Tier e2-micro instance. limit is by time.[31]
Features			
Documentation	4/5	3.5/5	3.5/5

2.5.2 Performance of Application

A containerized web application [29] was used to measure the application's latency and throughput when the application is hosted in Google's serverless Cloud Run, and server-based App Engine, and Compute Engine. The main functionality of the application is to show the real-time location of San Antonio VIA buses on selected bus routes and serve the estimated arrival time for select bus stops on user request [29].

Two different bus routes were emulated on the application with the help of an android device emulator. Then the application was accessed by various clients (bus riders) emulating more load on the service. We measured the performance

Table 2.2: Comparison results on performance matrices.

Service Configuration	Google Cloud Run (Serverless)	Google App Engine (Serverless)	Google Compute Engine (Server-based)
Latency (50%)	47.24ms	50.50 ms	-
Throughput	• Sent: 9.47k/s • Received: 11.15k/s	• Sent: 3.422KiB/s • Received: 2.241KiB/s	• Sent: 0.09KiB/s • Received: 0.16KiB/s
Container Startup Latency (50%)	475.22ms	4.944s	-
CPU Utilization (50%)	1%	6000 Megacycles/minutes	2%
Auto-scaled based on load	Yes	Yes	No

matrices using the Google Cloud Monitoring service. Table 2 shows the results of the performance matrices. The measured metrics include Latency, Throughput, Contain-er Startup Latency, CPU Utilization, and whether the service supports auto-scaling based on load.

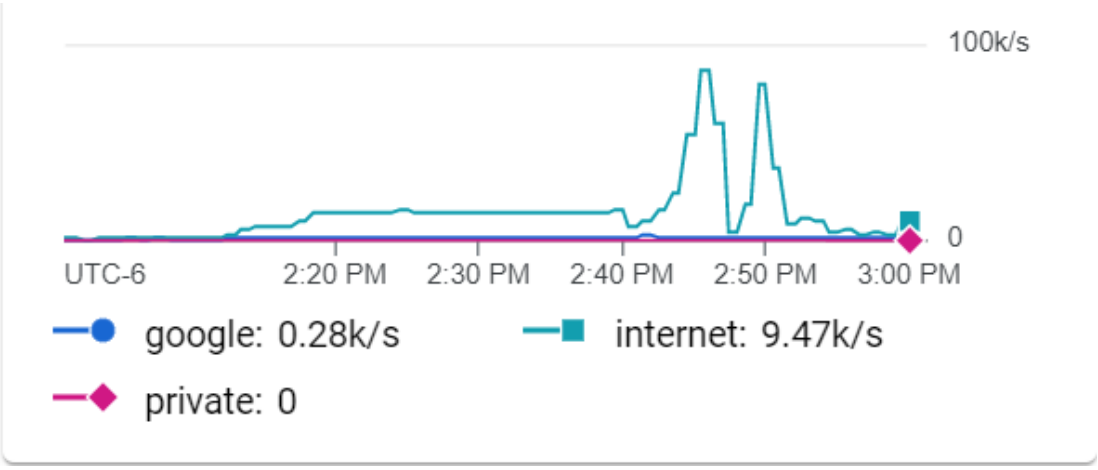

Figure 2.3: Cloud Run Service Throughput Sent Bytes

Cloud Run has the lowest Latency with a value of 47.24ms and the highest Throughput with a Sent rate of 9.47k/s and Received rate of 11.15k/s as shown

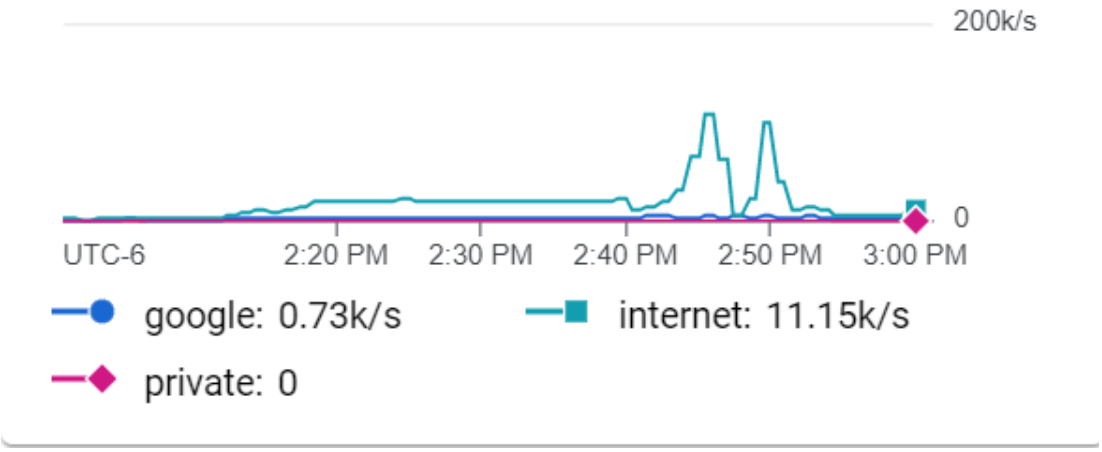

Figure 2.4: Cloud Run Service Throughput Received Bytes

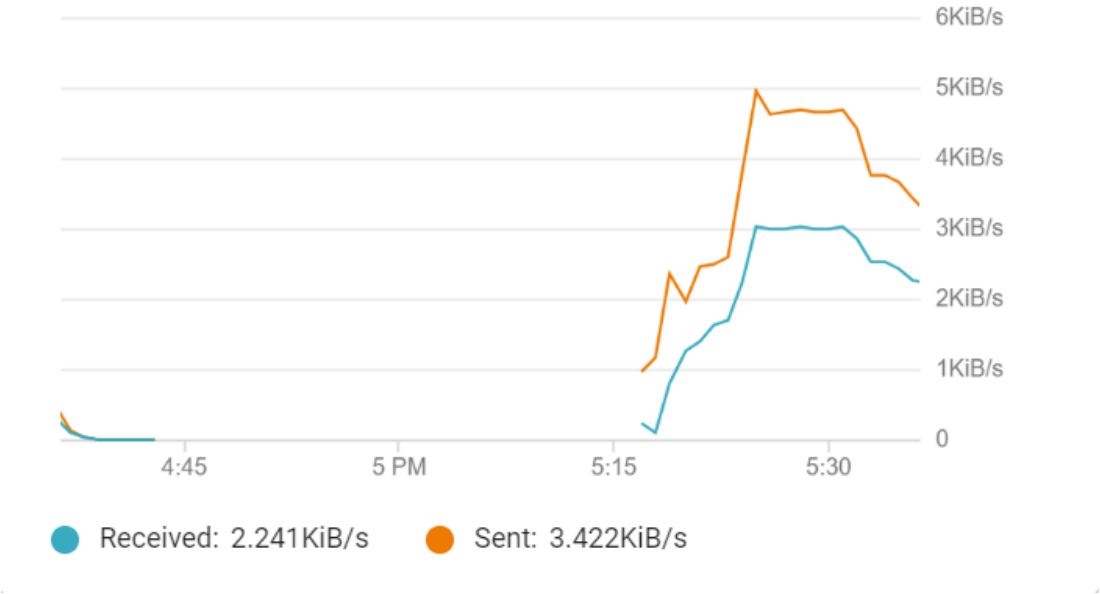

Figure 2.5: App Engine Throughput

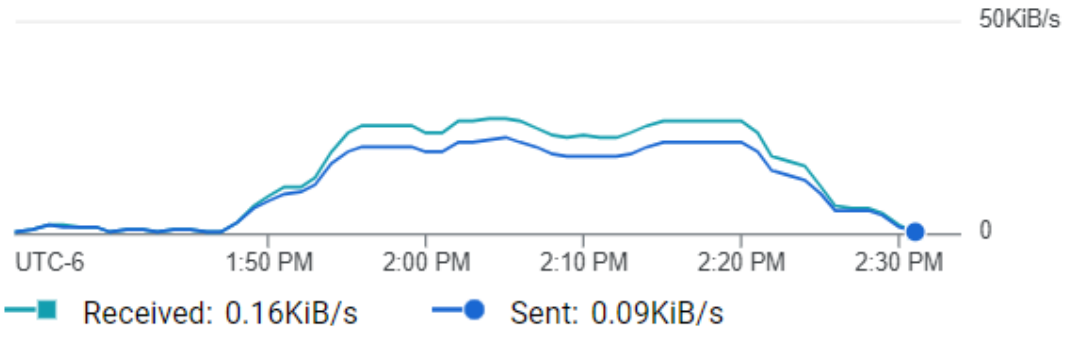

Figure 2.6: Compute Engine Throughput

in Fig. 2.3 and Fig. 2.4. It also has the lowest Container Startup Latency with a value of 475.22ms (Fig. 2.7) and its CPU utilization is 1% (Fig. 2.8). Cloud Run supports auto-scaling based on load.

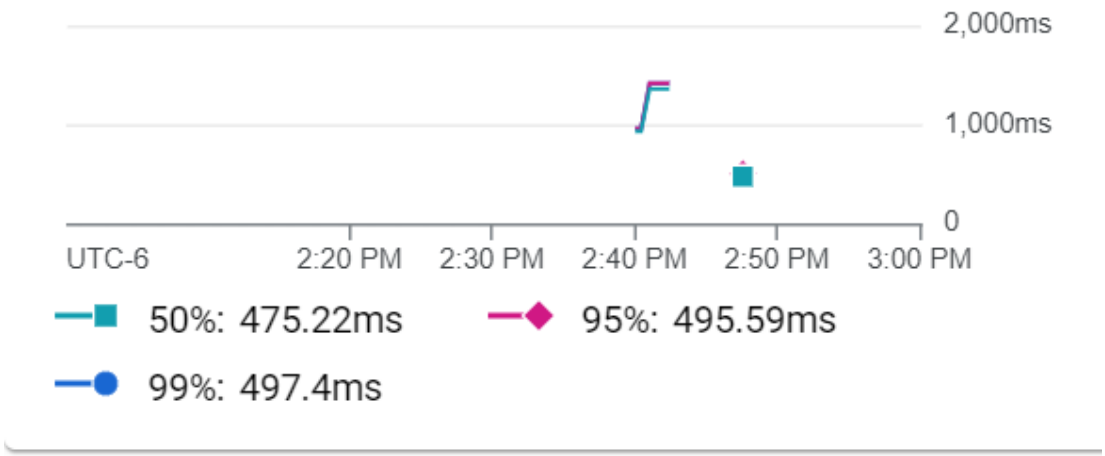

Figure 2.7: Startup Latency Cloud Run

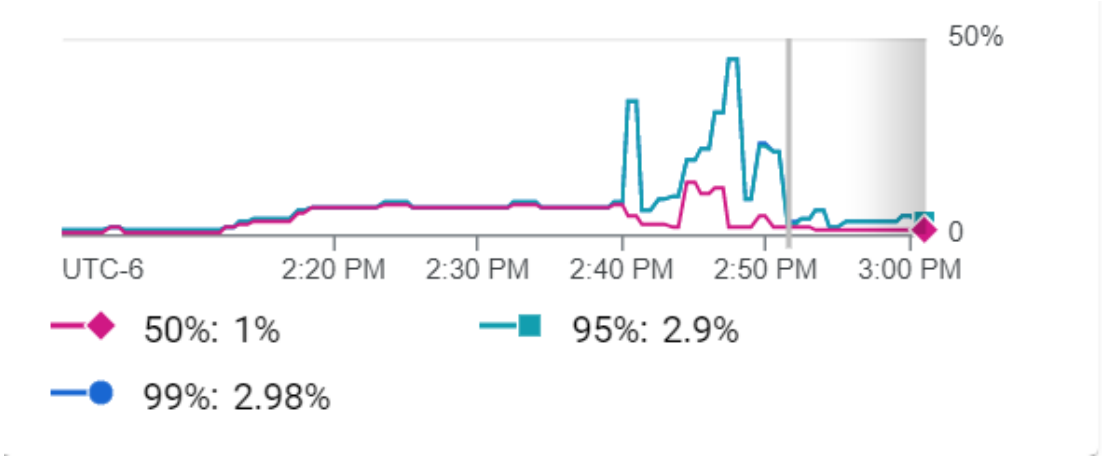

Figure 2.8: CPU Utilization Cloud Run

Figure 2.9: CPU Utilization App Engine

App Engine has a slightly higher Latency of 50.50ms compared to Cloud Run. Its Throughput Sent rate is lower with a value of 3.422KiB/s and a Received rate of 2.241KiB/s (Fig 2.5). The Container Startup Latency is higher at 4.944s. The

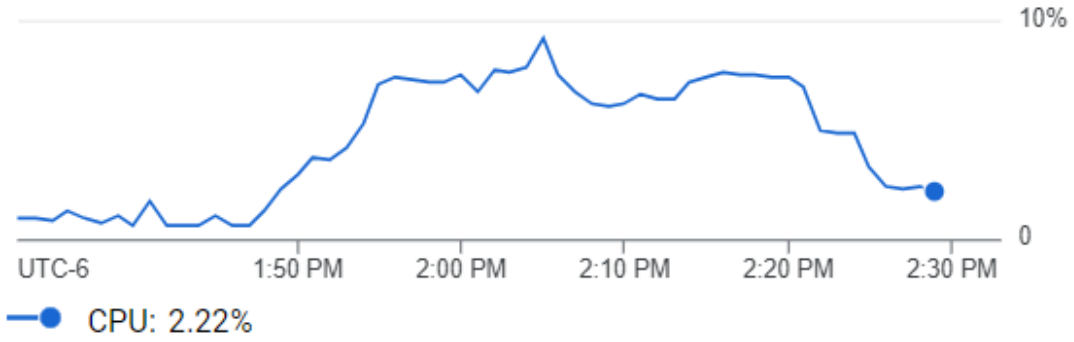

Figure 2.10: CPU Utilization Compute Engine

CPU utilization is 6000 Mega cycles/minutes (Fig 2.9). App Engine also supports auto-scaling based on load.

Compute Engine does not have a value for Latency or Container Startup Latency, but it has the lowest CPU utilization of 2% (Fig. 2.10). It does not support auto-scaling based on load and has the lowest Throughput Sent rate of 0.09KiB/s and Received rate of 0.16KiB/s (fig 2.6).

We found that the auto-scale feature from the App Engine and Cloud Run helped the application to quickly spin up additional instances so that it can serve the requests with less latency. We assumed that the Cloud Run service might perform better when we started the experiment. However, we observed that the latency was high for cloud-run service initially. This was mainly because of a quota limit hit by the Distance Matrix API used within the application. However, measuring the same values at a later stage showed that the latency was less for the Cloud Run service.

In conclusion, the serverless Cloud Run has the best performance compared to the other two services, based on the metrics. The choice between the services to deploy applications will depend on the specific requirements and priorities of the user, such as cost, ease of use, and scalability.

2.5.3 Limitation and Implication

The ratings of the documentation and learning curve presented in Table 1 were solely based on the author's experience during the study with the three cloud services. They may not necessarily reflect the objective views or opinions of others. The performance analysis mainly relied on the Cloud Monitoring service provided by the google cloud to measure the application's matrices. The lower CPU utilization values may be the result of less than adequate load on the web application. These are some of the improvements that we are planning to implement in a future date. The Compute Engine required an additional static IP to host the service with HTTPS support. This added more cost to the setup. Comparing the overall expense for the setup to host the same application over these three different services, Cloud Run costs the minimum $13.055 [32] with maximum benefits like HTTPS, auto-scaling, cloud logging, cloud monitoring, etc.

2.6 Conclusion and Future Work

In this study, the overall performance and usability of a containerized mobile-web application hosted on serverless and server-based services on Google Cloud were analyzed. The application's latency and throughput were measured when hosted on those three different platforms. The usability analysis focused on rating documentation and the learning curve on each of the three services.

The results of the analysis show that Cloud Run is a highly efficient and cost-effective option for deploying real-time web applications. With its request-based pricing model and free tier offering, users can save on costs compared to other Google Cloud services like Compute Engine and App Engine. The simplicity of the service also makes it an attractive choice for beginners, as users only need to provide the location of the application container to get started. Furthermore, the Cloud Run service is capable [15] of auto-scaling based on load, providing low latency and good throughput performance, as shown in the comparison data. In conclusion, Google Cloud Run offers a strong combination of cost-effectiveness, ease of use, and performance, making it a suitable option for real-time web application applications.

In our future study, we plan to expand the comparison of the overall performance and usability of the same containerized application on a serverless compute service (container without infrastructure) provided by two other Cloud services: AWS and Microsoft Azure. The serverless offerings we will be examining are AWS App Runner and Azure Container Apps.

Evaluating the Usability and Performance of a Containerized Mobile Web Application on Serverless Cloud Platforms

This study analyzed the features, usability, and performance of three serverless cloud computing platforms: Google Cloud's Cloud Run, Amazon Web Service's App Runner, and Microsoft Azure's Container Apps. The analysis was conducted on a containerized mobile application designed to track real-time bus locations for San Antonio public buses on specific routes and provide estimated arrival times for selected bus stops. The study evaluated various system-related features, including service configuration, pricing, and memory & CPU capacity, along with performance metrics such as container latency, Distance Matrix API response time, and CPU utilization for each service. The results of the analysis revealed that Google's Cloud Run demonstrated better performance and usability when compared to AWS's App Runner and Microsoft Azure's Container Apps. Cloud Run exhibited lower latency and faster response time for distance matrix queries. These findings provide valuable insights for selecting an appropriate serverless cloud service for similar containerized web applications.

3.1 Introduction

Cloud computing is a paradigm that enables the seamless addition and utilization of services over the Internet, with the unique capability of dynamic scalability. In the past, the cloud was frequently employed as a representation of a portion of the Internet that included certain infrastructure. However, in present times, the term "cloud" has evolved to serve as a metaphor for the wide range of services offered over the Internet[33]. The concept of cloud computing can date back to 1950 - 1960 [34, 35]. There were dramatic developments in this space after IBM introduced its new operating system named VM, which allowed its mainframe systems to have multiple virtual systems, or 'virtual machines (VM)', on a single physical node [35]. The National Institute of Standards and Technology (NIST) defines cloud computing as follows. "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction"[36].

Serverless cloud computing is a fairly new concept. According to [37], serverless cloud computing streamlines nearly all system administration tasks, effectively simplifying the cloud usage experience for programmers. When cloud computing became popular for the first time, developers were drawn to solutions like Amazon EC2 because they offered full control over application instances [37]. However, this also meant that developers were responsible for managing and scaling their applications, which could be time-consuming and complex. Additionally, developers often did not want to make significant changes to their existing code to make it cloud-friendly. As cloud computing matured, new technologies like FaaS (Function as a Service) emerged. FaaS provides a serverless computing platform that allows developers to run code without having to worry about infrastructure or scaling. This makes it ideal for running small, frequently-invoked tasks, such as processing payments or sending emails.

Currently, there are many cloud service providers available, with Google Cloud Platform (GCP), Amazon Web Services (AWS), and Microsoft (MS) Azure being the main players. Despite their current popularity, there is no active academic research available that provides a comparison of their usability and performance. Therefore, we have taken the initiative to conduct a comparative study of these services using a Django web application with a heavy reliance on Google Maps and Distance Matrix APIs. The analysis incorporated and focused on three serverless platforms: Google's Cloud Run, AWS' App Runner, and Azure's Container Apps. With this, the significant contributions of the research include:

- An overview of recent academic research related to serverless cloud computing from 2015 to 2023.
- A comparative analysis of the serverless service features offered by Cloud Run, App Runner, and Container Apps.
- A usability analysis of the serverless services offered by Cloud Run, App Runner, and Container Apps based on their ease of use and suitability for beginners in serverless computing.
- A performance analysis to deploy a containerized Django web application with heavy use of Google Maps APIs on the three serverless platforms: Cloud Run, App Runner, and Container Apps.
- Suggestions on the best serverless service to deploy containerized web applications with heavy use of Google Maps APIs.

The rest of the paper is organized as follows: Section 3.2 reviews a few academic studies in the cloud computing and serverless computing domain conducted from 2015 onward; Section 3.3 describes the serverless services used in the study; Section 3.4 discusses the Django application used for evaluating the performance of the services; Section 3.5 discusses the research methodology used in the study; Section 3.6 presents the results of the performance and usability analysis and Section 3.7 concludes the work.

3.2 Background on Serverless Computing

The idea of serverless computing has been around for many years, but it really started to take off in the early 2010s. This was due in part to the increasing popularity of cloud computing, as well as the development of new serverless platforms like AWS Lambda and Google Cloud Functions. In the early days, serverless computing was primarily used for small, event-driven tasks. However, as the technology has matured, it has become more widely adopted for a wider range of applications. Today, serverless computing is used by businesses of all sizes to build everything from simple web applications to complex microservice architectures.

There is a good amount of research happening in the serverless domain exploring its advantages and disadvantages. After conducting a thorough analysis of the literature published in this field from 2015 to 2023, a brief overview is presented below. In [38], Zaid Al-Ali et al. proposes a new abstraction for serverless computing named ServerlessOS. It abstracts away the details of resource management so developers can focus on their code. This allows processes to be seamlessly scaled across the data center, making serverless computing more serverless. R. Arokia Paul Rajan in [39] provides a comprehensive study of the serverless computing architecture and the working principle of the serverless computing reference model adapted by AWS Lambda.

Eivy and Weinman [40] analyzed the economics of serverless cloud computing, showing that the cost of a publicly hosted serverless service can grow quickly, even for low levels of traffic. They also noted that the free quota offered by most cloud service providers is insufficient for a decent public web service. Van Eyk et al. [41] discussed the early days of serverless computing and the obstacles and opportunities that existed at the time. Some of the obstacles, such as the lack of maturity of the technology and the limited availability of serverless platforms, have been addressed in recent years. However, new obstacles have emerged, such as the need to manage complex event-driven architectures.

With large organizations moving to the cloud, protecting the data in the cloud is equally important. In [42], Anaya Garde et. al. proposes a microservices-based approach to protect cloud-native assets using a serverless framework. This approach first discovers cloud assets and takes a snapshot-based backup of them. The backed-up assets can then be used to recover the asset in the event of a disaster or data loss. In [43], Mileski and Gusev used Google Cloud's serverless services to experiment with using serverless computing for real-time monitoring of thousands of patients with streaming electrocardiograms. This is an example of an embarrassingly parallel task, which means that it can be broken down into small, independent tasks that can be executed in parallel. They found that the serverless solution was able to achieve a speedup of almost 40 compared to a sequential execution on a virtual machine, and a speedup of 23 compared to a parallel execution using virtual machines.

Initially, serverless computing was largely associated with Function as a Service (FaaS) offerings from various cloud providers. However, there has been limited research in this area analyzing and comparing the different services in this domain [44, 45, 46, 47]. But, in 2023, there are more serverless offerings from most of the major cloud service providers. This major change happened with the

introduction of containers in serverless cloud computing [48]. In [49], Rodriguez Cortes et al. discusses the open issues of the serverless architecture.

3.3 Serverless Computing Services

Serverless cloud computing platforms like Google Cloud Run, AWS App Runner, and MS Azure Container Apps offer services on hassle-free containerized applications running without infrastructure management. These platforms prioritize scalability, reliability, and cost-effectiveness. Specifically, Cloud Run allows developers to run stateless containers via HTTP requests, App Runner enables running containerized web apps and API services, and Container Apps deploys containerized applications from code or containers without complex infrastructure orchestration. Table 3.1 summarizes the different features offered by these serverless services. Although they offer various features, they also share many similarities. The following describes the specific details of each service.

Table 3.1: Different features offered by three serverless services.

Feature	Google Cloud Run	AWS App Runner	Microsoft Azure Container Apps
Automatic Build and Deployment	Yes	Yes	Yes
Load Balancing	Yes	Yes	Yes
Scalability	Automatic	Automatic	Automatic
scaling to zero	Yes	No	Yes
General Purpose	Yes	No	Yes
Security	High	High	High
Cost	Competitive	Competitive	Competitive
Ability to Split traffic between different versions of the application.	Yes	Yes	Yes
Cloud Monitoring options.	Yes	Yes	Yes

3.3.1 Google Cloud Run

Google Cloud documentation defines Cloud Run as a serverless computing platform that allows developers to run stateless containers that are invocable via HTTP requests. Cloud Run is a fully managed service that abstracts away all infrastructure management, so developers can focus on what matters most —

building great applications [50]. Cloud Run was first announced in 2018 and made generally available in 2019. Cloud Run is built on top of the Kubernetes container orchestration platform [51]. Kubernetes is a popular open-source project that is used by many organizations to manage containerized applications. Cloud Run offers a number of benefits [52], including:

Traffic Management: Cloud Run's traffic management allows you to customize how incoming traffic is routed to your application revisions[50]. You can route traffic to the latest revision, a previous revision, or even split traffic between multiple revisions simultaneously. You can also fine-tune the percentage of incoming traffic that goes to each revision. This gives you a lot of flexibility in how you deploy and manage your applications, ensuring that they are always available and performing at their best.

Cost-effectiveness: Cloud Run charges customers for the CPU and memory allocated to a container instance, rounded up to the nearest 100 milliseconds. If a service is not in use, customers are not charged. This can lead to significant cost savings, especially for applications with variable or unpredictable usage patterns. There are two pricing models: request-based and instance-based. With request-based, customers are charged a per-request fee when a container instance is processing requests. With instance-based, customers are charged for the entire lifetime of a container instance, but there is no per-request fee.

Scalability: Cloud Run automatically adds and removes container instances to handle all incoming requests. This is known as auto-scaling. Cloud Run can scale to zero[50], which means that if there are no incoming requests, even the last remaining container instance will be removed. This is the default behavior, and the default number of minimum instances is 0. This can help to improve application performance and reliability, and it can also help to save money by avoiding over-provisioning of resources. However, setting the minimum number of instances to 0 may result in cold starts. A cold start occurs when a container is started for the first time since it was last shut down. Cold starts can add latency to requests because the container needs to load its application code and dependencies before it can start processing requests.

DevOps automation: Cloud Run can help to automate many of the tasks involved in DevOps, such as provisioning and scaling infrastructure, managing application deployments, and monitoring application health. This can free up DevOps teams to focus on other tasks, such as improving application performance and security.

Security: Cloud Run services are accessible from the internet by default. To restrict access, customers can configure a security policy that specifies which IP addresses or hostnames are allowed to access the service. If no security policy is configured, all incoming traffic is allowed access, which is not recommended for production environments. Customers should configure a security policy that restricts access to their services to only authorized users or systems.

Agility: Cloud Run can help to improve the agility of application development and deployment. By eliminating the need to provision and manage servers, developers can focus on building and testing applications. This can help to shorten the time to market for new applications.

3.3.2 AWS App Runner

AWS App Runner is a fully managed service that helps developers build, deploy, and run containerized web applications and API services without prior infrastructure or container experience [53]. It provides a high-level abstraction over the underlying infrastructure, allowing you to focus on the application code. App Runner supports a variety of programming languages and frameworks, including Node.js, Python, Java, Go, and . NET. It also supports a variety of container images, including Docker and Amazon ECS. App Runner also provides a number of features that make it easy to manage the applications, including:

Automatic scaling: App Runner's autoscale feature can help you improve the performance, reliability, and cost-effectiveness of your application. App Runner automatically scales your application up or down based on demand, so you only pay for the resources you use [53, 54]. App Runner can also automatically scale your application to handle spikes in traffic, so your users always have a good experience. Additionally, App Runner can automatically recover from failures by scaling your application back up. You can provide an auto-scaling configuration to customize the scaling behavior. If you do not provide one, App Runner provides a default configuration with recommended values. You can share a single auto-scaling configuration across multiple App Runner services to ensure they have the same auto-scaling behavior. Unlike Google Cloud Run, AWS App Runner doesn't allow the scale down to zero option.

Health checks: App Runner automatically performs health checks on your application. If your application fails a health check, App Runner will automatically restart it [54].

Logging and monitoring: App Runner provides detailed logs and metrics for your application. This helps you troubleshoot problems and identify performance bottlenecks [54].

Automatic Deployments: With App Runner, you can easily build and deploy your application in a matter of minutes. Simply connect App Runner to your code repository or container image registry to get started. App Runner will then monitor your repository for any updates to your source code or container image. Once an update is detected, App Runner will automatically build and deploy the new version of your application [54]. This feature is an efficient way to keep your application up to date, reducing the time it takes to deploy new features and bug fixes while improving the overall reliability of your application.

Load Balancing: App Runner offers the important capability of automatically balancing traffic across multiple containers [54]. This feature is highly significant in managing unexpected surges in traffic while ensuring optimal performance and availability. Given this, it proves to be a viable option for applications that require a robust traffic capacity and dependable availability.

Networking: App Runner provides users with the flexibility to tailor the interaction between their service, applications, and resources. Users are afforded the option to restrict access to their service solely within an Amazon VPC or enable the service to communicate with other AWS services within a VPC. This degree of control enables users to satisfy their security and networking compliance requirements [54].

3.3.3 Microsoft Azure Container Apps

Azure Container Apps is a fully managed environment that enables you to run microservices and containerized applications on a serverless platform[55]. It is a good choice for applications that need to be deployed quickly and easily and that can scale dynamically based on demand. Container Apps can be used to deploy API endpoints, host background processing applications, handle event-driven processing, and run microservices. Applications built on Azure Container Apps can dynamically scale based on HTTP traffic, event-driven processing, or CPU or memory load.

Serverless hosting: Container Apps provides a serverless architecture that eliminates the need for managing the underlying infrastructure. This translates to a worry-free experience for developers as they don't have to be concerned with server provisioning, capacity management, or application scaling. Instead, they can focus solely on developing their applications with ease.

Automatic scaling: Container Apps provide a solution to the problem of having to constantly monitor and adjust application capacity. With this technology, the applications can be scaled automatically based on various factors, such as CPU usage, memory usage, and network traffic [55]. This ensures that the applications are always accessible to users without any interruptions. The dynamic scaling feature of Azure Container Apps is a great feature for businesses that need to handle varying levels of demand for their applications. By removing the burden of manual capacity provisioning, companies can focus on other aspects of their operations that are critical to success.

Secured by default: Container Apps provide robust security features like role-based access control[55] and network isolation to keep your apps secure from unauthorized access. They're an excellent choice for maintaining app security.

Simple to use: Container Apps offers a user-friendly experience, featuring a straightforward interface that simplifies the process of deploying and managing your applications. This makes it an excellent choice for developers who are just starting with containerization. Through the Azure Container Apps portal, you can easily create and manage your applications. Additionally, you have the option to manage your applications using Azure CLI or Azure PowerShell [55].

Cost-effective: Deploying and scaling your applications is made cost-effective with Azure Container Apps. You only pay for the resources you utilize, with no upfront costs [55, 56]. The pricing for Azure Container Apps is determined by the number of containers you run and the amount of memory you consume.

Flexible: Azure Container Apps offers a versatile platform for deploying a diverse range of applications, making it an excellent choice for various use cases. You can deploy web applications, microservices, and serverless functions with ease using Azure Container Apps.

3.4 SmartSAT Django Web Application

SmartSAT is a customizable mobile app for San Antonio Transit that provides critical services to transit users. It is similar to other apps like Google Maps, Moovit, and Transit, but SmartSAT is designed specifically to improve the transit experience for lower-income people who rely on San Antonio's public VIA Transit

[57]. Its main objective is to provide real-time bus location information to reduce riders' wait times and thus the length of their daily commutes. The application tracks bus location in real-time, ultimately improving the prediction accuracy of bus arrival time. SmartSAT offers a variety of features that are not available on other apps, such as:

- Real-time bus arrival information
- Seat capacity information
- Instant alert messages on schedule changes
- Secure data collection and feedback from riders on their commute experience

These features can help lower-income people who rely on public transportation plan their trips more efficiently and avoid delays. They can also help riders provide feedback to VIA Transit so that the agency can improve its services.

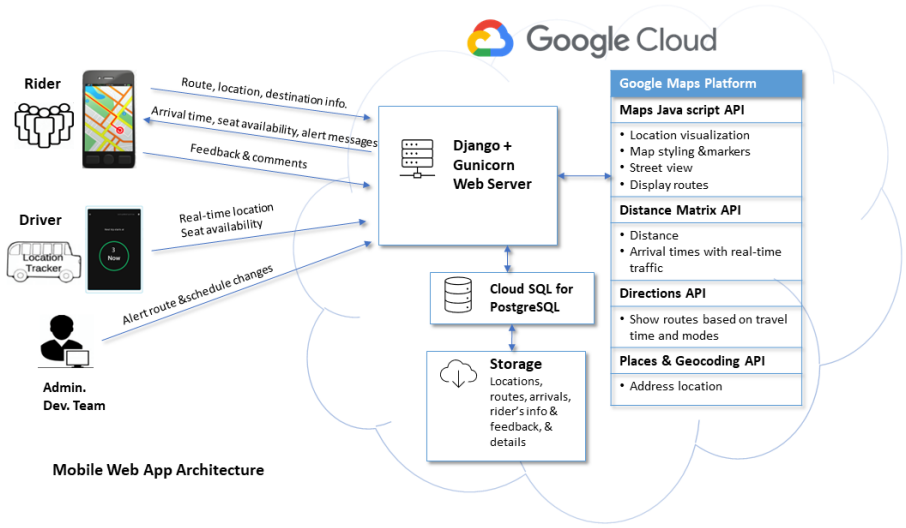

Figure 3.1: Architecture Diagram for the SmartSAT Application

Google Maps JavaScript API was used to showcase the real-time location of the buses on the selected route. Additionally, the SmartSAT app has the capacity to calculate and display the estimated arrival time at a selected bus stop on the route. The Google Distance Matrix API was used for this purpose, taking into consideration various factors such as traffic patterns, road accidents, and other elements that may impact the trip time. The app currently supports about ten VIA bus routes, but only three were used for the experiment of the performance analysis. Directions API was used to showcase routes based on travel time and modes as well as Places and Geocoding API for addressing current locations. Figure 3.1 shows the architecture diagram for the smartSAT application.

To use the app, users first select their desired bus route from the home screen. The app will then load all the bus stops on that route, including the order in

which they appear. By clicking on any of the bus stop markers, users can view the estimated arrival time for that stop. The app uses a Postgres database in Google Cloud SQL to maintain data related to users and bus routes.

3.4.1 Deployment of SmartSAT Application on Three Serverless Platforms

Using containers in computing has a rich and extensive history [58, 59]. Unlike hypervisor virtualization, where an intermediation layer enables running one or more independent machines virtually on physical hardware, containers operate in user space atop an operating system's kernel [60, 61]. Thus, container virtualization is often referred to as operating system-level virtualization [60]. This technology enables the creation of numerous isolated user-space instances on a single host.

Docker is an app platform that streamlines the process of building, deploying, and running apps by bundling all the necessary elements into containers. The advantages of this approach include easy mobility, efficient resource utilization, and a quick, streamlined development cycle [60, 62]. Apart form Docker there are many other container management options available as listed in [63]. But we decided to use docker because it was supported by the all the three services used in the study and it was famous among the developer community. Because of these benefits, we opted to package our application in a docker container when the application was deployed on each of the three serverless platforms: Google's Cloud Run, AWS's App Runner, and Azure's Container Apps. We followed the instructions outlined on [64] to establish the initial configuration of the docker file and build it for the Cloud Run platform. We then made minor tweaks to the docker file to address various application requirements. To guarantee deployment on other cloud services on App Runner and Container Apps, we modified the settings files to include the appropriate port numbers and parameter values. Using a container instead of a virtual machine is more cost-effective. With a container, the user is only billed for the time it is running, whereas a virtual machine is billed for the entire time it is powered on. This makes container deployment a more economical option in my case.

3.5 Research Methodology

The primary goal of the study was to analyze and evaluate the usability and performance of a containerized SmartSAT Django mobile web application [52, 57] on three serverless cloud services: Google's Cloud Run, AWS's App Runner, and Microsoft Azure's Container Apps. Toward this, the following three research questions were accordingly developed. The evaluation measured the results of the system features, usability, and performance of running the application on each of the serverless services.

- RQ1: What are the key differences in system features between serverless computing services on AWS, GCP, and Azure?

- RQ2: Which cloud provider among GCP, AWS, and Azure offers the most beginner-friendly documentation and learning resources for serverless computing services?
- RQ3: How do serverless computing services on AWS, GCP, and Azure differ in terms of the performance of running a containerized application?

For RQ1, we analyzed the system features of each service, considering their service configuration, pricing, and memory & CPU capabilities. For RQ2, we evaluated the documentation, learning curve, and Scale to Zero capabilities of each service, considering the perspective of a beginner serverless cloud user. Lastly, RQ3 focused on the performance of the containerized application, measuring factors such as container latency, API response time, and Container CPU utilization across all three services.

3.5.1 Performance Measurements of Application

For the performance measurements, the application was slightly modified to enable its deployment on each service. This application was chosen for its ease of deployment, as it was already developed and containerized as part of the project [52], making it compatible with the selected services. In order to create a required load on the application, an Android emulator was used to simulate the GPS (Global Positioning System) movement of a bus along a selected bus route. The following steps were performed on each deployed serverless platform: Cloud Run, App Runner, and Container Apps.

1. Navigate to the deployed service link from the Android emulator and log in to the website as a bus driver.
2. Start two different bus routes. (This will emulate the same scenario when a real bus moves in that route.)
3. Access the application as a normal user (Bus Rider) from different devices.
4. Click on a different bus stop icon so the website shows the estimated arrival time for that stop.
    (a) Repeat this task for both the active routes from different tabs opened in the web browser. This will mimic multiple people accessing the application.
    (b) Once both buses reach their destination stops, stop the bus driving from the driver screen.
5. Navigate to the cloud monitoring option in Google Cloud for the corresponding service and note down the results.

As the indicators of the performance metrics, 'Container Request Latency', 'Container CPU Utilization', and 'API Response Time' were measured on the performance analysis of the application during the study. Below is a detailed description of the metrics measured and Fig.3.2 presents a visual representation of their concepts.

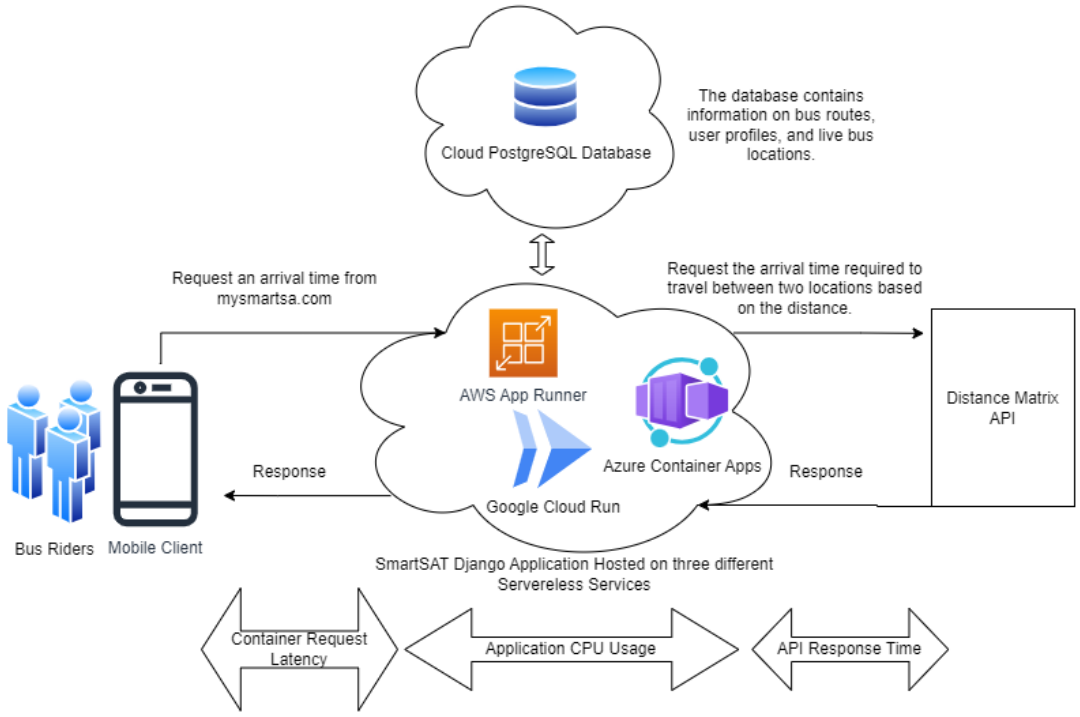

Figure 3.2: Visual representation of the concepts of Request Latency, CPU Usage, and API Response Time.

- **Container Request Latency:** When a client sends a request to a server and waits for the corresponding response, the time it takes for this process is known as request latency. This is a crucial performance metric in the realm of computer networks and distributed systems, as it affects the overall system performance. Factors such as distance between the client and server, network congestion, and hardware limitations can impact the latency. In this study, we conducted performance testing for three services from the same device and network. It's important to note that the latency values captured primarily reflect the container request latency and not the client-side network latency. Additionally, the cloud provider's network load can influence the service's response time to user requests. By maintaining consistent user-side network and hardware configurations, the latency measurements captured by the cloud provider's application monitoring services provide a clear picture of the application latency for the specific service.

- **API Response Time:** Response Time is defined as the duration between the time a user sends a request and the moment the system responds with the intended output. A pictorial representation of the same is shown in Fig. 3.2. Although response time may appear like latency, it encompasses both latency and processing time. The significance of response time in modern computing cannot be overstated, as it directly impacts the user experience. A slow response time can lead to frustration, decreased productivity, and negative perceptions of the system or application. Therefore, it is imperative to optimize response time to enhance user satisfaction. Various techniques such as caching, load balancing, and optimization of code can

be employed to reduce response time.

- **CPU Utilization:** When evaluating the performance of the containerized application, it is essential to consider its CPU usage. During the test load, the CPU usage for the container can provide valuable insights into its efficiency and effectiveness. By analyzing the CPU usage patterns, it is possible to identify any bottlenecks or performance issues that may be impacting the container's performance. It is important to note that CPU usage can vary depending on the workload and the hardware configuration of the system. Therefore, it is essential to establish a baseline for CPU usage and compare it against the actual usage during the test load. This comparison can help to identify any abnormal behavior that may be indicative of performance issues.

3.6 Evaluation Results

This section presents the results of the analysis in terms of system features, usability, and performance of the three serverless services: Google's Cloud Run, AWS's App Runner, and Microsoft Azure's Container Apps. Tables 3.2, 3.3, and 3.4 show the results of the analysis. The numbers in the table 3.4 are mostly sourced from configuring the serverless services.

3.6.1 Results of Analysis on System Features

Comparing system features on each of the services is associated with service configuration, pricing, and relevant system features such as memory and CPU. This analysis is to answer RQ1: What are the key differences in system features between serverless computing services on AWS, GCP, and Azure? Table 3.2 compares the service configurations, pricing, and system features of the three services. These services offer different configurations and pricing models, making them suitable for different types of applications and use cases.

Cloud Run is Google's serverless computing platform that allows for the easy deployment of containerized applications. It is priced on the basis of the number of requests and compute time used. The first 120 minutes of build time per day is free. This experiment used an instance with 0.5GB of memory per container and a single CPU. The application was able to handle a high volume of requests without any performance degradation.

AWS App Runner, on the other hand, does not offer free quotas. It costs around $0.064 per vCPU-hour and $0.007 per GB-hour. It supports two pricing models: a provisioned plan and an active container instance plan. The maximum number of instances for the service is limited to 25, but the maximum concurrency is 200, compared to 100 for Cloud Run. The maximum memory size available for a container is 12 GB, while Cloud Run has a capacity of 34 GB. App Runner also has a maximum CPU capacity of 4, while Cloud Run has 8. I found the learning curve for AWS to be steep.

Microsoft Container Apps is a relatively new service launched in May 2022 [65]. It offers two pricing options: one based on resource utilization and one based on a fixed pricing plan. The free tier quotas are the same for both Cloud Run and

Table 3.2: Comparison results on system features.

Service Configuration	Google Cloud Run	AWS App Runner	Microsoft Azure Container Apps
No of CPU's	1		1
Memory	0.5 GB / container		2.0 GB
Min number of Instances	0	1	0
Max number of Instances	100 by default, but depends on CPU and memory configurations	25	300
Max Concurrency per instance	1000	200	Supports custom value
Pricing			
Compute Time Unit	100 ms	1s	1s
Pricing Models	request-based, instance-based	Provisioned, and active container instances	Plans available based on resource consumption and a dedicated plan option
Free Tier	First 180,000 vCPU-seconds/month, first 360,000 GiB-seconds/month, 2 million requests/month	No Free tier, $0.064 / vCPU-hour $0.007 / GB-hour	The first 180,000 vCPU-seconds, 360,000 GiB-seconds, and 2 million requests each month are free
Memory & CPU			
Min Memory	1 GB	0.5 GB	0.5
Max Memory	34 GB	12 GB	4
Min vCPUs	<1	0.25	0.25
Max vCPUs	8	4	2

Container Apps. Container Apps support scaling to zero if the scaling setting is not based on CPU utilization. The maximum number of instances supported is 300, which is higher than the other two services. In this test, a configuration of 1 CPU core and 2 GB of memory was used for the container. There was a good amount of documentation available about getting started with hosting service, but due to the recent release of the service, there were not many external resources available about the service.

To summarize, Google Cloud Run offers more memory and CPU options and supports more requests per container instance than AWS App Runner. Microsoft

Container Apps has two pricing options and supports up to 300 instances per service. Ultimately, the choice of service will vary based on the individual needs of the user. Cloud run may be the preferable option for those requiring greater memory and CPU capabilities.

3.6.2 Results of Usability Analysis

The ISO 9241-11 [66] is the standard many uses for the usability testing of software systems. The framework comprises three components: System Effectiveness, which assesses the users' ability to accomplish the assigned tasks; System Efficiency, which gauges the resources required by the users to complete the tasks. And System Satisfaction, which records the users' opinions and feedback[67]. This standard defines the word usability as *"extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use"*[68]

This study focuses on analyzing the usability of serverless services, specifically from the perspective of a beginner cloud user. The author analyzed the effectiveness of deploying the application on each respective service. The ease or difficulty of learning and performing the deployment task on each service, as well as the time it took for the author to complete it, were considered measures of efficiency. The satisfaction experienced throughout the task, from beginning to end, is the determining factor for recommending services. Two significant factors that affect the time it takes for a user to complete tasks on a new system are documentation and the learning curve. The "scale-to-zero" feature eliminates the need for additional configurations on services to adjust their capacity based on the system's request load. These three factors can be attributed to the "effectiveness" and "efficiency," discussed in the ISO 9241-11.

Cloud Run, App Runner, and Container Apps are serverless container platforms that allow you to deploy and run containerized applications without having to worry about managing servers. Analyzing the easy-to-use usability of three serverless services is associated with its documentation, learning curve for beginners, and scale-to-zero capability. This is to address RQ2: Which cloud provider among GCP, AWS, and Azure offers the most beginner-friendly documentation and learning resources for serverless computing services? The results of the usability analysis are summarized in Table 3.3.

For this analysis, we have evaluated the quality of documentation available for various cloud services by primarily referring to their official documentation pages. Additionally, we have taken into consideration the availability of open learning resources on the Internet. Through the analysis, we have discovered that Google Cloud run service has the most detailed and user-friendly official documentation, along with several third-party resources readily available online. AWS App Runner service, on the other hand, has documentation that may be challenging for beginners to follow due to the abundance of information on the cloud console dashboard. However, there are still resources available online for application deployment on the service. Azure has a more straightforward cloud console and clear documentation compared to AWS, but due to its recent launch, there are fewer online resources available as compared to the other two services. It should be noted that these ratings were solely based on the author's experience

Table 3.3: Comparison results on ease-of-use usability.

Ease of use [1]	Google Cloud Run	AWS App Runner	Microsoft Azure Container Apps
Documentation	Excellent 4.5/ 5	Good 3 / 5	Good 3.5 / 5
The learning curve for a beginner	Gradual Learning curve. The availability of a lot of getting-started templates makes learning faster.	Pretty Steep. Too much information to grasp.	Gradual Learning Curve. External resources were less as it is relatively new.
Scale to zero	Yes[2]	No	Yes[2]

[1] The ratings of the documentation and learning curve presented were solely based on the author's experience during the study with the three cloud services. They may not necessarily reflect the objective views or opinions of others.

[2] Applications that scale on CPU or memory load can't scale to zero.

during the study with the three cloud services. They may not necessarily reflect the objective views or opinions of others.

Scale to zero is another factor that is considered for the usability analysis of the services. This feature allows a service to shut down the container if it doesn't receive any requests for a specified amount of time. This is particularly beneficial in terms of cost savings when the end user is not utilizing the application. While this capability is only available on GCP and Azure, it proves to be useful when scaling based on requests instead of CPU and memory usage. Container Apps support scaling to zero when the scaling setting is not based on CPU utilization.

In conclusion, as a beginner cloud service user, we found Google Cloud has the most detailed and user-friendly documentation for their serverless service, with several third-party resources available online. AWS App Runner's documentation may be challenging for beginners, but there are still online resources available. Azure has a straightforward cloud console and clear documentation, but fewer online resources compared to the other two services.

3.6.3 Results of Performance Analysis

The performance of the serverless service was evaluated using a containerized Django web application described in section 3.4. The performance metrics, 'Container Request Latency', 'Response Time (Distance Matrix API)', and 'CPU Utilization' of the application were measured when the application was hosted on each of the three serverless platforms. This analysis is to address RQ3: How do serverless computing services on AWS, GCP, and Azure differ in terms of the performance of running a containerized application? Table 3.4 shows the results of the evaluation.

To obtain the values of the metrics, two different bus routes were emulated in the application with the help of an Android emulator. Then the application was accessed by various clients (bus riders) emulating more load on the service. The 'Container Request Latency' is the time taken for a process when a client sends a request to a server and waits for the corresponding response. The latency

Table 3.4: Comparison results on Request Latency, Response Time, and CPU utilization.

Performance Metrics	Google Cloud Run	AWS App Runner	Microsoft Azure Container Apps
Request Latency (Container)	50 ms	91 ms	-
Response Time (Distance Matrix API)	63.7 ms	71.38 ms	66.8 ms
CPU Utilization	23%	47%	0.000773% [1]

Note: The corresponding graphs for each parameter can be found in Figures 3.3 - 3.8

[1] Azure App monitoring graph shows the values in millicores. Thus, this value is converted to a percentage using the formula 3.1.

values were captured primarily reflecting the container request latency and not the client-side network latency. API Response Time is the duration between the time a user sends a request and the moment the system responds with the intended output. This is to get Distance Matrix API's response to the user's request. The CPU utilization represents the CPU usage for the execution of the application during the test load. These three metrics were measured by using the Cloud Monitoring services provided by the corresponding cloud provider while the application was running.

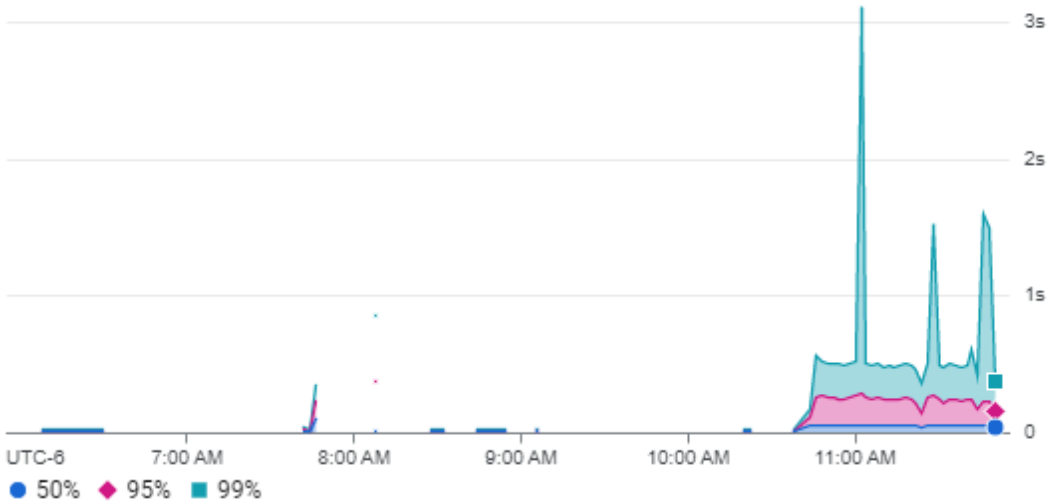

Figure 3.3: Request Latency for Google Cloud Run.

In terms of the 'Container Request Latency', Google's Cloud Run demonstrated a lower latency compared to AWS's App Runner. Cloud Run has latency, with an average request latency of 50 milliseconds (Fig. 3.3). AWS App Runner

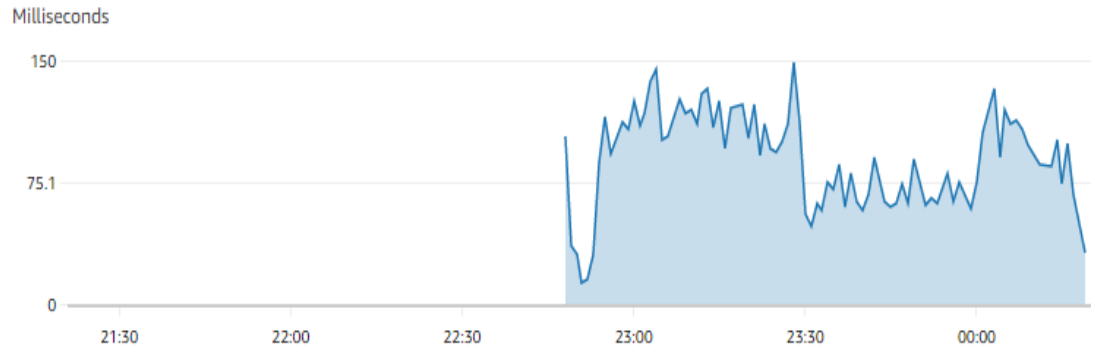

Figure 3.4: Request Latency for AWS App Runner.

has the higher latency, with an average request latency of 91 milliseconds (Fig. 3.4). Azure Container Apps monitoring services don't have the ability to capture the container latency for the applications deployed. Due to this, the values for the Azure container apps are not available. The total load on the cloud provider for a specific time can impact the performance of that provider. The performance test for the cloud run service and the remaining two services were performed on two days. We can't attribute any specific behavior from the google cloud as a key factor for better results on the latency value.

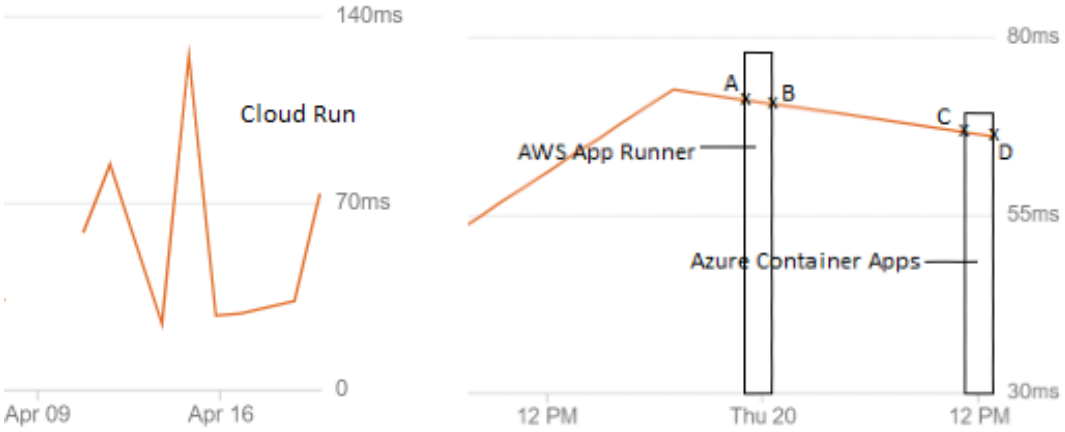

Figure 3.5: Response Time for Google Cloud Run, AWS App Runner, and Azure Container Apps (A=71.66, B=71.11, C=67.22, D=66.38).

Regarding the Response Time to Distance Matrix API, the average response time was better when the application was hosted on Cloud Run than being hosted on App Runner and Container Apps (Fig. 3.5). The response time for Cloud Run is 63.7 milliseconds on average, while App Runner and Container Apps have an average response time of 71.38 and 66.8 milliseconds, respectively. Tests for AWS were run from 10:50 PM to 12:30 AM and for Azure from 11:30 AM to 12:30 PM. Therefore, a single graph is presented to cover the results for both services during the test duration. As shown in Fig. 3.5 and Table 3.4, the average values of A and B represent the response time for AWS App Runner, while the averages of C and D represent the response time for Azure Container Apps. We observe that the better response time when hosted on Google Cloud Run may be because the Distance Matrix API was also from Google, which may have helped in faster

communication between the application and API.

In terms of CPU utilization, Google Cloud Run uses an average of 23 % (Fig. 3.6). AWS App Runner uses the most amount of CPU, with an average CPU utilization of 47% (Fig. 3.7). Azure Container Apps uses the least amount of CPU, with an average CPU utilization of 0.000773% (Fig. 3.8). As the Azure Container Apps monitoring graph shows the values in millicore, this value is converted to a percentage using the formula 3.1 below, as indicated in Table 3.4. The lower CPU Utilization values for Azure can be because of how the CPU is allocated for the application. Cloud Run and App Runner use CPU the entire time the container runs, whereas the Azure container apps request CPU only when it needs to process a request. So, directly comparing this with Cloud Run and AWS is not right when considering the total container CPU usage.

$$percentage = \frac{number\ of\ cores}{10^6} \times 100 \tag{3.1}$$

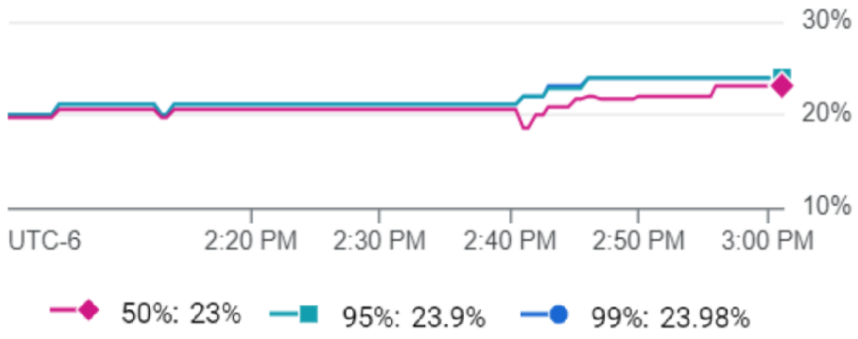

Figure 3.6: CPU Utilization for Google Cloud Run.

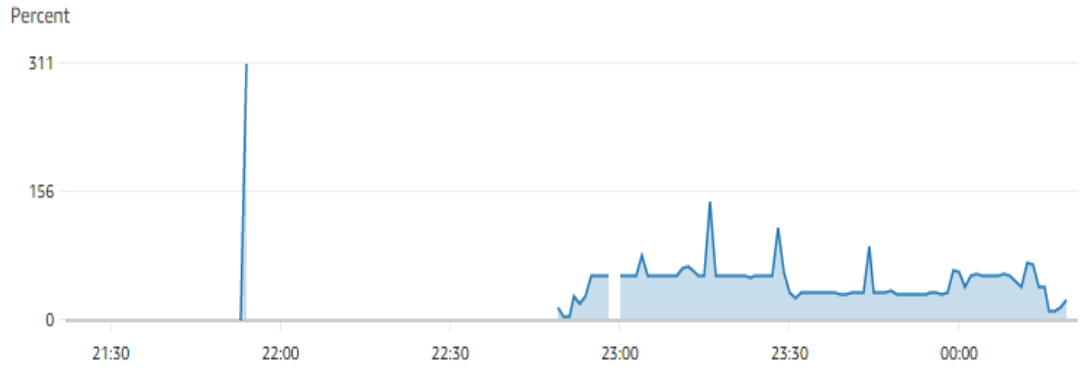

Figure 3.7: CPU Utilization for AWS App Runner.

Overall, the findings of the analysis revealed that Google's Cloud Run exhibited superior performance and usability compared to AWS's App Runner and Microsoft Azure's Container Apps on this particular application. Cloud Run demonstrated the lowest latency at 50 ms and a faster response time of 63.7

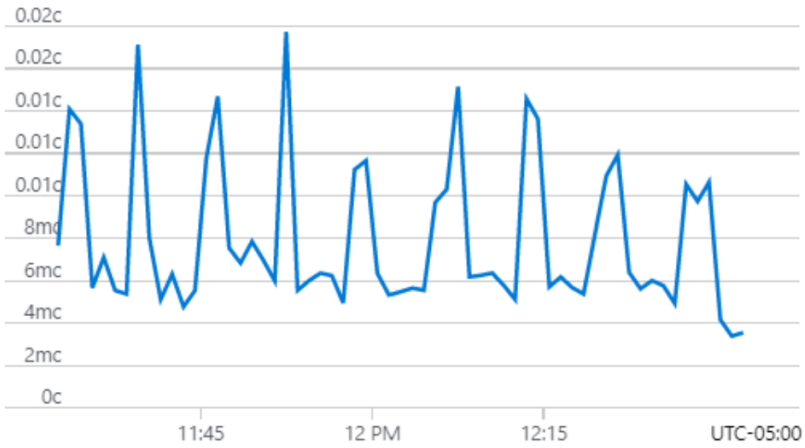

Figure 3.8: CPU Utilization for Azure Container Apps.

ms for distance matrix queries. These results provide valuable insights for individuals seeking to select an appropriate cloud service for similar containerized web applications. Google Cloud Run might be the best service to host a Django containerized web application with Google Maps and Distance Matrix API usage.

3.6.4 Limitation and Implication

The ratings for documentation and learning curve presented in Table 3.3, are derived solely from the author's experience with the three serverless cloud services throughout the in-depth study that includes the entire application deployment process as well as subsequent maintenance activities. It is important to note that these ratings may not necessarily reflect the objective views or opinions of others. The performance analysis was primarily conducted using the Cloud Monitoring service offered by Google, AWS, and Azure to measure the application's metrics. It is possible that lower CPU utilization values were due to insufficient load on the web application. In particular, the lower CPU Utilization values for Azure Container Apps can be because of how the CPU is allocated for the application. Cloud Run and App Runner use CPU the entire time the container runs, whereas the Azure Container Apps request CPU only when it needs to process a request. Thus, directly comparing this with Cloud Run and App Runner is not right when considering the total container CPU usage. Also, the performance matrix values are specific to a containerized mobile web application with heavy use of google maps APIs. These values may be different for other applications.

3.7 Conclusion

This research aimed to identify the key differences in system features, ease-of-use usability, and performance between AWS, GCP, and Azure serverless cloud computing systems with respect to a containerized mobile web application. By reviewing the academic research works that happened in the serverless computing domain in the last 6 to 8 years, we found that a lot of new research is happening in this area, especially in recent years. This paper focused on several which talked about the history of cloud and serverless computing and the comparative study

of different cloud services. After thoroughly examining the features of serverless services offered by GCP, AWS, and Azure, it was observed that the Google Cloud Run service stands out with its extensive range of memory and CPU options. It also supports a greater number of containers, instances per service, and maximum concurrency per instance compared to its competitors. The AWS App Runner service and Azure Container Apps were found to be the next best options. When it comes to ease of use and usability of services, Google Cloud Run provides the easiest-to-understand documentation and educational materials for beginners. Azure and AWS also offer helpful resources, but Google Cloud Run stands out as the most beginner-friendly option. From the performance analysis of the services, we found that the performance of the Google Cloud Run service was superior in regards to Container Request Latency and Distance Matrix API response time. Meanwhile, Azure Container Apps exhibited the lowest CPU memory utilization compared to the other two options.

From these findings, our suggestion is to opt for GCP as the primary choice, followed by Azure, for those developers who are new to the concept of serverless computing. On the other hand, AWS is a suitable option for developers already well-acquainted with serverless computing. These key contributions will help future academic researchers and developers when making a decision to choose a serverless cloud platform for deploying similar containerized applications.

The analysis of performance results in this comparative study was focused solely on the particular mobile web application. However, there is potential for conducting further research on other application domains and exploring a broader range of cloud providers beyond those examined in this study.

Summary & Conclusion

As the industry shifts towards cloud technology, more people are transitioning their applications to the cloud in order to stay up-to-date. It's crucial to select the most secure and cost-effective option for your application. Conducting a comparison of various serverless services offered by major cloud providers can assist researchers and organizations in selecting the best fit for hosting their applications.

In Chapter 2, we conducted a comparative analysis of three Google Cloud services, both serverless and server-based, for hosting a containerized mobile web application with heavy usage of Google Maps APIs. This section discusses various cloud deployment models along with the features of each service.

Chapter 3 builds upon the research presented in Chapter 2 by exploring serverless services from other providers such as AWS and Azure. The comparative analysis primarily focuses on the common features shared by all three cloud providers. After conducting the study, it was discovered that the Google Cloud Run service is the most appropriate option for hosting the containerized application that is heavily utilized in this study.

4.1 Conclusion

This research aimed to identify the key differences in system features, ease-of-use usability, and performance between AWS, GCP, and Azure serverless cloud computing systems. The first part of this study analyzed Google cloud run (serverless), Google App Engine (serverless), and Google Compute Engine (server-based) for their usability and performance of hosting a containerized application, we found that google cloud run performed better as compared to other serverless and server-based service options from google cloud platform itself.

The second part of the study compared AWS, GCP, and Azure serverless cloud computing systems using the same application, it was found that GCP's Cloud Run provides the best range of memory and CPU options, supports more containers and instances, and has the highest concurrency per instance. Additionally, it has the most user-friendly documentation. Google Cloud Run also performed the best in terms of Container Request Latency and Distance Matrix API response time. Meanwhile, Azure Container Apps had the lowest CPU memory utilization. Based on these findings, GCP is recommended for beginners to serverless computing, followed by Azure. AWS is better suited for experienced developers. However, additional research is needed to determine which cloud providers are best for other application domains.

Bibliography

[1] Ruozhou Yu, Xudong Yang, Jun Huang, Qiang Duan, Yan Ma, and Yoshiaki Tanaka. Qos-aware service selection in virtualization-based cloud computing. 2012. doi: 10.1109/APNOMS.2012.6356046.

[2] David Bernstein. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing*, 1, 2014. ISSN 23256095. doi: 10.1109/MCC.2014.51.

[3] Rabindra K. Barik, Rakesh K. Lenka, K. Rahul Rao, and Devam Ghose. Performance analysis of virtual machines and containers in cloud computing. 2017. doi: 10.1109/CCAA.2016.7813925.

[4] Franco Callegati, Walter Cerroni, Chiara Contoli, and Giuliano Santandrea. Performance of network virtualization in cloud computing infrastructures: The openstack case. 2014. doi: 10.1109/CloudNet.2014.6968981.

[5] Michael Seibold, Andreas Wolke, Martina Albutiu, Martin Bichler, Alfons Kemper, and Thomas Setzer. Efficient deployment of main-memory dbms in virtualized data centers. 2012. doi: 10.1109/CLOUD.2012.13.

[6] Ryan Shea and Jiangchuan Liu. Performance of virtual machines under networked denial of service attacks: Experiments and analysis. *IEEE Systems Journal*, 7, 2013. ISSN 19328184. doi: 10.1109/JSYST.2012.2221998.

[7] Theo Lynn, Pierangelo Rosati, Arnaud Lejeune, and Vincent Emeakaroha. A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms. volume 2017-December, 2017. doi: 10.1109/CloudCom.2017.15.

[8] Jussi Nupponen and Davide Taibi. Serverless: What it is, what to do and what not to do. 2020. doi: 10.1109/ICSA-C50368.2020.00016.

[9] Thirukovela Venkata Sarathi, Julakanti Sai Nischal Reddy, Peddaboinolu Shiva, Rounak Saha, Anurag Satpathy, and Sourav Kanti Addya. A preliminary study of serverless platforms for latency sensitive applications. *2022 IEEE International Conference on Electronics, Computing and Communication Technologies, CONECCT 2022*, 2022. doi: 10.1109/CONECCT55679.2022.9865790.

[10] IBM. Iaas vs. paas vs. saas | ibm. URL `https://www.ibm.com/topics/iaas-paas-saas`.

[11] Jianfeng Yang and Zhibin Chen. Cloud computing research and security issues. 2010. doi: 10.1109/CISE.2010.5677076.

[12] Chnar Mustafa Mohammed, Subhi Zeebaree, Chnar Mustafa Mohammed, and Subhi Zeebaree. Sufficient comparison among cloud computing services: Iaas, paas, and saas: A review. *International Journal of Science and Business*, 5:17–30, 2021. URL `https://EconPapers.repec.org/RePEc:aif:journl:v:5:y:2021:i:2:p:17-30`.

[13] Dimpi Rani and Rajiv Kumar Ranjan. A comparative study of saas , paas and iaas in cloud computing. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4, 2014.

[14] I. Hsun Chuang, Syuan Hao Li, Kuan Chieh Huang, and Yau Hwang Kuo. An effective privacy protection scheme for cloud computing. 2011.

[15] Zaid Al-Ali, Sepideh Goodarzy, Ethan Hunter, Sangtae Ha, Richard Han, Eric Keller, and Eric Rozner. Making serverless computing more serverless. volume 2018-July, 2018. doi: 10.1109/CLOUD.2018.00064.

[16] Mingyu Wu, Zeyu Mi, and Yubin Xia. A survey on serverless computing and its implications for jointcloud computing. 2020. doi: 10.1109/JCC49151.2020.00023.

[17] Yongkang Li, Yanying Lin, Yang Wang, Kejiang Ye, and Cheng Zhong Xu. Serverless computing: State-of-the-art, challenges and opportunities. *IEEE Transactions on Services Computing*, 2022. ISSN 19391374. doi: 10.1109/TSC.2022.3166553.

[18] Fakulta Informačních Technologií, Bakalářská Práce, and Autor PRÁCE SUPERVISOR Ing MARTIN HRUBÝ. Framework for a web internet service im-plemented in google cloud platform. *theses.cz*. URL `https://theses.cz/id/71d25b/25187.pdf`.

[19] Victor Juan Exposito Jimenez and Herwig Zeiner. Serverless cloud computing : A comparison between "function as a service" platforms. 2018. doi: 10.5121/csit.2018.80702.

[20] Yuping Shen. Data statistics of intelligent monitoring platform for preschool education cultural inheritance and practice resource allocation based on google cloud sharing algorithm. *4th International Conference on Inventive Research in Computing Applications, ICIRCA 2022 - Proceedings*, pages 1381–1384, 2022. doi: 10.1109/ICIRCA54612.2022.9985475. URL `https://www.researchgate.net/publication/366675241_Data_Statistics_of_Intelligent_Monitoring_Platform_for_Preschool_Education_Cultural_Inheritance_and_Practice_Resource_Allocation_based_on_Google_Cloud_Sharing_Algorithm`.

[21] Hassan B. Hassan, Saman A. Barakat, and Qusay I. Sarhan. Survey on serverless computing, 2021. ISSN 2192113X.

[22] MAHESH K, M. LAXMAIAH, and YOGESH KUMAR SHARMA. A comparative study on google app engine amazon web services and microsoft windows azure. *INTERNATIONAL JOURNAL OF COMPUTER ENGINEERING & TECHNOLOGY*, 10, 2019. ISSN 0976-6367. doi: 10.34218/ijcet.10.1.2019.007.

[23] M. Landoni, G. Taffoni, A. Bignamini, and R. Smareglia. Application of google cloud platform in astrophysics. 3 2019. doi: 10.48550/arxiv.1903.03337. URL https://arxiv.org/abs/1903.03337v1.

[24] Hera Arif, Hassan Hajjdiab, Fatima Al Harbi, and Mohammed Ghazal. A comparison between google cloud service and icloud. 2019. doi: 10.1109/CCOMS.2019.8821744.

[25] Senecca Miller, Travis Siems, and Vidroha Debroy. Kubernetes for cloud container orchestration versus containers as a service (caas): Practical insights. 2021. doi: 10.1109/ISSREW53611.2021.00110.

[26] Krzysztof Burkat, Maciej Pawlik, Bartosz Balis, Maciej Malawski, Karan Vahi, Mats Rynge, Rafael Ferreira Da Silva, and Ewa Deelman. Serverless containers - rising viable approach to scientific workflows. 2021. doi: 10.1109/eScience51609.2021.00014.

[27] Google. Choose an app engine environment | app engine documentation | google cloud, . URL https://cloud.google.com/appengine/docs/the-appengine-environments.

[28] Google. Compute engine: Virtual machines (vms) | google cloud, . URL https://cloud.google.com/compute/#section-10.

[29] Jeong Yang, Young Lee, William Noon, and Anoop Abraham. Demo abstract: Smartsat - a customizable secure app for san antonio transit pilot project. *MobiWac 2022 - Proceedings of the 20th ACM International Symposium on Mobility Management and Wireless Access*, pages 123–127, 10 2022. doi: 10.1145/3551660.3560910. URL https://dl.acm.org/doi/10.1145/3551660.3560910.

[30] Google. Pricing | app engine | google cloud, . URL https://cloud.google.com/appengine/pricing.

[31] Google. Free cloud features and trial offer | google cloud free program, . URL https://cloud.google.com/free/docs/free-cloud-features#free-tier-usage-limits.

[32] Google. Google cloud pricing calculator, . URL https://cloud.google.com/products/calculator/#id=04887044-5cb5-422e-b37b-7c6c1caa7c45.

[33] Jayachander Surbiryala and Chunming Rong. Cloud computing: History and overview. In *2019 IEEE Cloud Summit*, pages 1–7, 2019. doi: 10.1109/CloudSummit47114.2019.00007.

[34] Simson Garfinkel. *Architects of the information society: 35 years of the Laboratory for Computer Science at MIT*. MIT press, 1999.

[35] IBM Cloud IBM Cloud Team. A brief history of cloud computing, 2017. URL `https://www.ibm.com/cloud/blog/cloud-computing-history`.

[36] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.

[37] Eric Jonas, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar, Joao Carreira, Karl Krauth, Neeraja Yadwadkar, et al. Cloud programming simplified: A berkeley view on serverless computing. *arXiv preprint arXiv:1902.03383*, 2019.

[38] Zaid Al-Ali, Sepideh Goodarzy, Ethan Hunter, Sangtae Ha, Richard Han, Eric Keller, and Eric Rozner. Making serverless computing more serverless. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, pages 456–459. IEEE, 2018.

[39] R Arokia Paul Rajan. Serverless architecture-a revolution in cloud computing. In *2018 Tenth International Conference on Advanced Computing (ICoAC)*, pages 88–93. IEEE, 2018.

[40] Adam Eivy and Joe Weinman. Be wary of the economics of" serverless" cloud computing. *IEEE Cloud Computing*, 4(2):6–12, 2017.

[41] Erwin Van Eyk, Lucian Toader, Sacheendra Talluri, Laurens Versluis, Alexandru Uță, and Alexandru Iosup. Serverless is more: From paas to present cloud computing. *IEEE Internet Computing*, 22(5):8–17, 2018.

[42] Anaya Garde, Siddhi Gandhale, Rutuja Dharankar, Bhagya Shri Sangtani, Nutan Deshmukh, Shilpa Deshpande, Rahul Rathi, and Astitva Srivastava. Serverless data protection in cloud. In *2023 6th International Conference on Information Systems and Computer Networks (ISCON)*, pages 1–6. IEEE, 2023.

[43] Dimitar Mileski and Marjan Gusev. Serverless implementations of real-time embarrassingly parallel problems. In *2022 30th Telecommunications Forum (TELFOR)*, pages 1–4. IEEE, 2022.

[44] Theo Lynn, Pierangelo Rosati, Arnaud Lejeune, and Vincent Emeakaroha. A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 162–169. IEEE, 2017.

[45] Rakesh Veuvolu, Anirudh Suryadevar, T Vignesh, and Nikhil Reddy Avthu. Cloud computing based (serverless computing) using serverless architecture for dynamic web hosting and cost optimization. In *2023 International Conference on Computer Communication and Informatics (ICCCI)*, pages 1–6. IEEE, 2023.

[46] Mohit Sewak and Sachchidanand Singh. Winning in the era of serverless computing and function as a service. In *2018 3rd International Conference for Convergence in Technology (I2CT)*, pages 1–5. IEEE, 2018.

[47] N Saravana Kumar and Samy S Selvakumara. Serverless computing platforms performance and scalability implementation analysis. In *2022 International Conference on Computer, Power and Communications (ICCPC)*, pages 598–602. IEEE, 2022.

[48] Neil Savage. Going serverless. *Communications of the ACM*, 61(2):15–16, 2018.

[49] Luis Miguel Rodriguez Cortes, Edward Paul Guillen, and Wilson Rojas Reales. Serverless architecture: Scalability, implementations and open issues. 2022.

[50] Google Cloud Team. What is cloud run, 2018. URL `https://cloud.google.com/run/docs/overview/what-is-cloud-run`.

[51] Google Cloud Team. The most scalable and fully automated kubernetes service, 2018. URL `https://cloud.google.com/kubernetes-engine/`.

[52] Anoop Abraham and Jeong Yang. A comparative analysis of performance and usability on serverless and server-based google cloud services. In Kevin Daimi and Abeer Al Sadoon, editors, *Proceedings of the 2023 International Conference on Advances in Computing Research (ACR'23)*, pages 408–422, Cham, 2023. Springer Nature Switzerland. URL `https://doi.org/10.1007/978-3-031-33743-7_33`.

[53] AWS. Aws app runner, 2021. URL `https://aws.amazon.com/apprunner/`.

[54] AWS. Aws app runner features, 2023. URL `https://aws.amazon.com/apprunner/features/?refid=12eea001-bcfd-40ce-9788-748f73400e32`.

[55] Azure. Azure container apps overview, 2023. URL `https://learn.microsoft.com/en-us/azure/container-apps/overview`.

[56] Azure. Quotas for azure container apps, 2023. URL `https://learn.microsoft.com/en-us/azure/container-apps/quotas`.

[57] Jeong Yang, Young Lee, William Noon, and Anoop Abraham. Demo abstract: Smartsat - a customizable secure app for san antonio transit pilot project. In *Proceedings of the 20th ACM International Symposium on Mobility Management and Wireless Access*, MobiWac '22, page 123–127, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394802. URL `https://doi.org/10.1145/3551660.3560910`.

[58] Allison Randal. The ideal versus the real: Revisiting the history of virtual machines and containers. *ACM Comput. Surv.*, 53(1), feb 2020. ISSN 0360-0300. doi: 10.1145/3365199. URL `https://doi.org/10.1145/3365199`.

[59] Ell Marquez. The history of container technology, 2023. URL `https://www.pluralsight.com/resources/blog/cloud/history-of-container-technology`.

[60] James Turnbull. *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2014.

[61] Claus Pahl. Containerization and the paas cloud. *IEEE Cloud Computing*, 2(3):24–31, 2015. doi: 10.1109/MCC.2015.51.

[62] Docker Community. Docker overview, 2023. URL `https://docs.docker.com/get-started/overview/`.

[63] TAMAL DAS. The 9 best docker alternatives for container management, 2023. URL `https://www.makeuseof.com/best-docker-alternatives/`.

[64] Google Cloud. Running django on the cloud run environment, 2023. URL `https://cloud.google.com/python/django/run`.

[65] Azure. Generally available: Azure container apps, 2023. URL `https://azure.microsoft.com/en-us/updates/generally-available-azure-container-apps/`.

[66] Ergonomics of human-system interaction — part 110: Interaction principles, 2020.

[67] Wikipedia. Iso 9241, 2023. URL `https://en.wikipedia.org/wiki/ISO_9241`.

[68] Ergonomics of human-system interaction — part 11: Usability: Definitions and concepts, 2018.

VITA

Anoop Abraham, a native of Kanhangad, Kerala, India, was born on July 25th, 1995. He earned his Bachelor's degree in Computer Science from Christ University Faculty of Engineering, Karnataka, India, in 2017. Anoop then began his career as a Software Engineer at "Sunquest Information Systems" in Karnataka, India. Later, he pursued his passion for Computer Science and obtained a Master's degree from Texas A&M - San Antonio in August 2023.

PUBLICATIONS

Anoop Abraham, Daniel Livingston, Izabella Guerra, and Jeong Yang. Exploring the application of machine learning algorithms to water quality analysis. In 2022 IEEE/ACIS 7th International Conference on Big Data, Cloud Computing, and Data Science (BCD), pages 142–148, 2022. doi: 10.1109/BCD54882.2022.9900636.

Jeong Yang, Young Lee, William Noon, and Anoop Abraham. Demo abstract: Smartsat - a customizable secure app for san antonio transit pilot project. In Proceedings of the 20th ACM International Symposium on Mobility Management and Wireless Access, MobiWac '22, page 123–127, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394802. doi: 10.1145/3551660.3560910. URL https://doi.org/10.1145/3551660.3560910.

Anoop Abraham and Jeong Yang. A comparative analysis of performance and usability on serverless and server-based google cloud services. In Kevin Daimi and Abeer Al Sadoon, editors, Proceedings of the 2023 International Conference on Advances in Computing Research (ACR'23), pages 408–422, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-33743-7.

Anoop Abraham, Jeong Yang. Analyzing the System Features, Usability, and Performance of a Containerized Application on Serverless Cloud Computing Systems, 14 July 2023, PREPRINT (Version 1) available at Research Square [https://doi.org/10.21203/rs.3.rs-3167840/v1].