Testing Serverless Applications with AWS Lambda: An Automatic Move to Serverless Architectures

Shamiksha Mishra [1], Abdullah Alenizi [2], Subrata Dutta [3]

1. Department of Computer Science & Engineering, NIT Jamshedpur, India, 2019ugcs033@nitjsr.ac.in
2. Department of IT, College of Computer & Information Science, Majmaah University, Saudi Arabia-11952
3. Department of Computer Science & Engineering, NIT Jamshedpur, India

Abstract

In serverless computing, servers' computing resources are distributed dynamically by the cloud service provider. Consumers are charged based on the usage of resources, not on prepurchase computing capability. Programming models, abstractions, and platforms for cloud services and technologies need to evolve. This paper aims to provide large scalability and low configuration costs for cloud applications. This paper explores testing strategies for a system that allows users to request rides on unicorns from the Wild Rydes fleet. The proposed application to build, deploy and tested, for serverless services developed for the Amazon Web Services cloud platform. The results have been obtained for the parameters Duration, Error count & success rate, Call, Throttle, Total concurrent execution and compare with the existing work. The results have a less duration, with very high success rate with zero error. This paper will help users to help in state of art transportation service so that people can travel faster and easier.

Keywords: Serverless computing;Lambda; DynamoDB.

Introduction

Companies like Apache Open Whisk, Azure Function, Google Cloud Functions, and Open Lambda were among the first to offer serverless computing, a cloud-based service in which application logic is divided into functions and run in response to events [1]. These events are frequently triggered inside between cloud platform services as well as outside. This makes it possible for developers to swiftly and simply create distributed apps across several cloud providers. Applications described by their events are triggered by actions and events in serverless computing. Because events are handled in response to event streams, this language is similar to active database systems. These ideas are fully embraced by serverless function platforms by dispersing their event-processing logic across their clouds.

In addition to event-driven infrastructure, container management, and software development techniques are now being discussed. With serverless computing for multi-level elasticity and graphics processing unit (GPU) virtualization, scalable event-driven computing is possible. [2]. IoT applications benefit from serverless computing, which overlaps with edge and Fog computing infrastructures. Serverless computing allows large applications to be decomposed into smaller functions. This allows individual scaling of application components but creates a new problem of managing a large number of functions [3]. The field of serverless computing is extremely active. The point at which the use of serverless or virtual machines becomes more cost-effective has been calculated in several studies. Serverless computing is becoming increasingly important. It translates naturally into a microservices architecture and is on a growth and adoption trajectory [4,5]. It is seen as the next wave of cloud computing services. More and more mobile and Internet-of-Things (IoT) apps are powered by serverless computing, which is rapidly spreading across different cloud providers. Serverless cloud computing, which simplifies the management of intricate internal infrastructures and simply concentrates on data analytics solutions, is the solution to these issues. The most popular program is known as Amazon Web Services (AWS). Users from all over the world can access a wide range of infrastructure and cloud solutions through the app [6].

Data analytics has its own set of challenges, including those related to the infrastructure needed to carry out the analytics activities, the expense of doing so, as well as infrastructure, storage, and security. The Internet of Things (IoT), 5G Internet, smart cities, and other upcoming technologies all rely on cloud computing services to handle and store more data. The vulnerabilities and security concerns of the cloud paradigm will therefore increase as a result of the heterogeneity of new firms adopting the aforementioned technologies [7,8]. The biggest obstacle to effective data analysis is the infrastructure needed to handle the massive amounts of data. Its infrastructure consists of powerful processing units that deliver great performance in terms of execution time, sizable storage systems, data saved across multiple locations, and effective embedded software systems.

It is expected that as the cloud becomes more widespread, the associated problems can be solved and serverless computing will dominate the future of cloud computing. It is important that the system be extensible, so that different data sources and providers can be easily incorporated (e.g., metering systems), and that it be scalable so that the system can be used from smaller installations (e.g., a building) to very large installations (likely entire neighborhoods), with deployment costs proportional to installation size.

Motivation

The motivation behind this work is to develop a testable serverless application using AWS Lambda. Using the AWS Lambda computing service, users can run code without having to set up or maintain servers. Codes run on a highly available computing infrastructure using Lambda, and Lambda manages all aspects of computing resource management, such as capacity provisioning, server and OS maintenance, and auto-scaling. The Lambda platform enables the operation of virtually any type of back-end service and application. The proposed system combines AWS Amplify, Amazon Cognito, API Gateway, AWS Lambda, and Amazon DynamoDB. The AWS Amplify service makes it easy to build full-stack applications on AWS using tools and features tailored for front-end web and mobile. Amazon Cognito can authenticate, authorize, and manage web and mobile application, users. APIs can be created, published, maintained, monitored, and secured using Amazon API Gateway. APIs can be provided for custom client applications as well as APIs for third-party app developers. A key benefit of DynamoDB is encryption at rest, so sensitive data is protected without making encryption at rest cumbersome or complex. On-demand backups are possible with DynamoDB. To analyze a large amount of data using serverless architecture patterns that reduce the operational complexity of running and managing applications, this research attempts to research serverless cloud computing platforms (AWS).

Our main contributions to this research work are as follows:

- Addressing the serious issues of first-generation serverless computing that bring its potential for automatic scaling into conflict with the two main streams in computing - data-centric and distributed computing - as well as with open source and specialized hardware.
- To build, deploy and testing the proposed applications without configuring or managing the servers.
- To allow users to request rides on unicorns from the Wild Rydes fleet.
- To provided errorless and cost-effective application

The rest of the article is organized as follows. Section two contains the current state of the art developed by various researchers. Section three describes the research methodology and proposed framework. Section 4 discusses the experimental analysis and results. In section 5, summarization the conclusion of the paper is done.

Related Work

The idea of serverless computing in the IT industry holds significant potential for extending its capabilities to a broader range of industries. Therefore, the implementation of serverless computing is not limited to infrastructure improvements [9,10]. The future of cloud computing will be driven as much by business factors as by technological advances. Cloud customers are choosing serverless computing because it allows them to focus on industry- or domain-specific issues rather than server management or distributed systems issues. Due to the robustness of this consumer promise, serverless computing has a very good chance of being mainstream in the future [11].

It is used for many different things, such as serverless messaging, training neural networks [12], processing videos [13], and large data [14, 15, 16]. Without a doubt, both the general public and experts can benefit from their efforts. This is due to the critical relevance of comprehending how these technologies operate. Big Data analytics are incredibly important in today's online environment, particularly when it comes to the analysis of data from multi-tenant systems that are connected over the Internet and produce a lot of multi-structured data. In [15], the authors focus on Amazon Web Service while discussing multi-tenant data analytics of the serverless cloud architecture (AWS). There are two different application kinds involved. How well Big Data functions under the constraints of time, traffic, and data size. One generates static data while the other generates live dynamic data.

A tried-and-true Spark execution engine called Flint [16] uses Amazon Lambda to offer a pure pay-as-you-go pricing model. Without the necessity for a real Spark cluster, a developer can utilize PySpark as usual with Flint.In addition to the primary Spark data processing engine, Apache Spark [17] is a robust all-in-one analytics engine for machine learning and large-scale distributed computing that includes libraries for SQL, machine learning, graph computation, and stream processing. Applications involving analytics, machine learning, and artificial intelligence can benefit from Spark. Amazon Virtual Private Cloud (Amazon VPC [18] permits the construction of a logically isolated section of the AWS cloud when some AWS services are launched on an extremely virtual network. Users have complete control over how subnets and networks are set up.

To reduce overall data volume and adhere to privacy laws, the increasing adoption of new Internet-of-Things (IoT) devices necessitates more efficient bandwidth consumption, lower latency, and data pre-processing closer to the source. Even though open-source programs and commercial serverless cloud providers already exist [19].The authors discussed how cloud computing and its platforms are evolving, with serverless computing emerging as the next stage. In [20], the authors systematically reviewed several research papers on serverless computing and described various techniques to reduce execution time, cost, or both. The design, implementation, and deployment of serverless applications face additional obstacles, and the serverless computing platforms available today are far from ideal. To the best of our knowledge, these difficulties have not been thoroughly explored. This paper is the first to thoroughly explore how to capture the difficulties developers face when building serverless applications to fill this knowledge gap.

Amazon's AWS Lambda was the first broadly used FaaS platform, even though serverless architecture has been around for more than ten years [21]. However, Google and Microsoft also provide their own FaaS services, known as Google Cloud Functions (GCF) [22] and Azure Functions [23], respectively. Researchers still create serverless programs using Amazon Lambda today. Similar features and advantages are provided by Google Cloud Functions, Azure Functions, and Amazon Lambda. Writing serverless apps can be done in a wide variety of languages. The support for programming languages varies among AWS Lambda, Azure Functions, and Google Cloud Functions. All services can run serverless Java and Python functions natively. Yet, there are distinctions in addition to that. Go and Ruby can only be used with AWS Lambda and Google Cloud Functions, whereas JavaScript and TypeScript are only available with Azure Functions.

It's imperative that do more than just communicate theories and notions. Instead, it is now necessary to weigh the advantages and disadvantages of serverless computing, take into account how far the industry has come, and decide what still has to be done and improved.

It is possible to extend serverless computing's capabilities beyond the IT industry to a wide range of other industries. In this way, serverless computing goes beyond improving infrastructure. The future of cloud computing is driven equally by business factors and technological advances. Rather than focusing on server management or distributed systems, cloud customers choose serverless computing because of its flexibility. Serverless computing has a very good chance of becoming mainstream in the future because of its robust consumer promise.

The main research gap addressed in this proposed work is highlighted below:

- First-generation serverless computing, with its autoscaling potential at odds with prevailing trends in modern computing, including data-centric and distributed computing, as well as open source and custom hardware, must be addressed.
- To Builds and deploys application without configuring or managing the underlying servers.
- Serverless breaks down applications into smaller and smaller pieces, known as decomposition. This will lead to better observability across applications.
- As the cloud adoption rate grows, we predict the issues related to it can be resolved and serverless computing will grow to dominate the future of cloud computing.
- Containers enable serverless applications to run within fewer attack points than traditional architectures and to have only one set of credentials to access them.
- It is essential that the system be extensible, so that differing data sources and providers can be incorporated easily (such as measurement systems), and that it be scalable, allowing the system to be used from smaller installation (e.g., one building) to very large installations (probably entire neighborhoods), with deployment costs proportional to installation size.

Research Method

The objectives mentioned in the previous section would be fulfilled by using the following research process which is shown in Fig. 1 below.

Research Process

- Research gap and critical analysis
- Problem formulation
- Framework for serverless computing
- Build a serverless application

- Validate the result

Serverless architecture

Services can be developed and run without having to be in control of the underlying infrastructure thanks to a technique for creating software called serverless architecture. By writing code and deploying it in this way, a cloud service provider will create servers to operate any scale of applications, databases, and storage systems. Function as a Service (FaaS) is one of the most well-liked serverless concepts. Each function responds to a trigger, such as an HTTP request or an incoming email, by carrying out a certain action. Users provide their functions and triggers in a cloud provider account during standard testing methods. Depending on the circumstance, the cloud provider either creates a new server or executes the function on a server that is already running when a function is called. Fig. 2 shows the architecture of serverless computing.

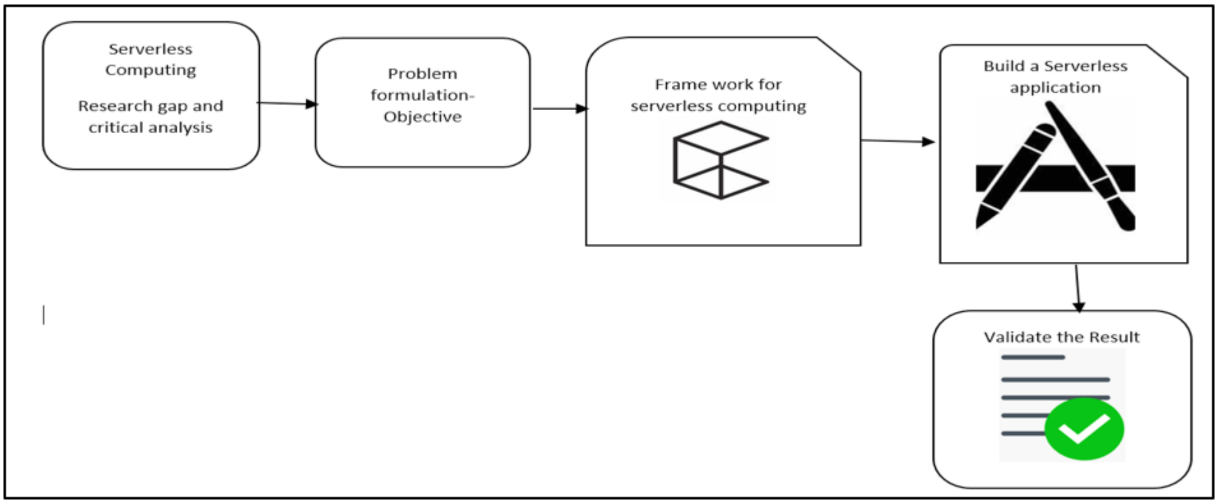

Serverless Computing
Research gap and critical analysis
Problem formulation-Objective
Frame work for serverless computing
Build a Serverless application
Validate the Result

Fig. 1. Research Process

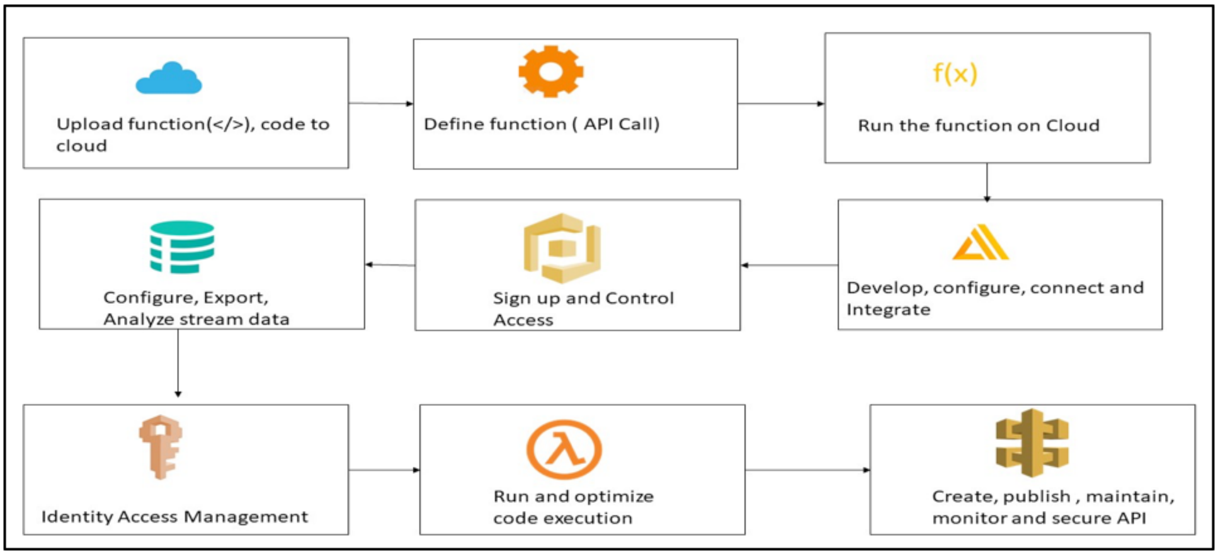

Upload function(</>), code to cloud
Define function (API Call)
f(x)
Run the function on Cloud
Develop, configure, connect and Integrate
Sign up and Control Access
Configure, Export, Analyze stream data
Identity Access Management
Run and optimize code execution
Create, publish , maintain, monitor and secure API

Fig.2. Architecture of Serverless Computing

Experimental Setup Analysis

Users were able to submit requests for rides on unicorns from the Wild Rydes fleet via a custom app. Users can specify where they want to be picked up through an HTML-based interface, and a RESTful web service submits the request and sends a nearby unicorn to the backend. Users have the option to register and sign in through the application, in addition to being able to do so before requesting a ride. The application uses AWS Lambda, Amazon API Gateway, Amazon DynamoDB, AWS Cognito, and AWS Amplify Console as its architecture[21]. HTML, CSS, JavaScript, and picture files are hosted in the amplifier Console and then loaded into the user's browser. Lambda and API Gateway are used to send and receive data from a public API via JavaScript. By allowing user management and authentication, Cognito secures the backend API. The Lambda function of the API can use DynamoDB's persistence layer as the last service to store data.

Fig. 3 shows the proposed serverless computing architecture based on AWS Lambda, Amazon API Gateway, Amazon DynamoDB, AWS Cognito, and AWS Amplify Console. Amplify Console hosts static web resources such as HTML, CSS, JavaScript, and picture files and loads them into users' browsers. Using Lambda and API Gateway, JavaScript is used in the browser to communicate with a public API.

By allowing user management and authentication, Cognito secures the backend API. The Lambda function of the API can use DynamoDB's persistence layer as the last service to store data. Wild Rydes fleet application links to a RESTful Web service on the back end, providing users with an HTML-based user interface that lets them specify the location where they want to be picked up to submit the request and send a nearby unicorn. Users can log in and register with the service before requesting a ride.

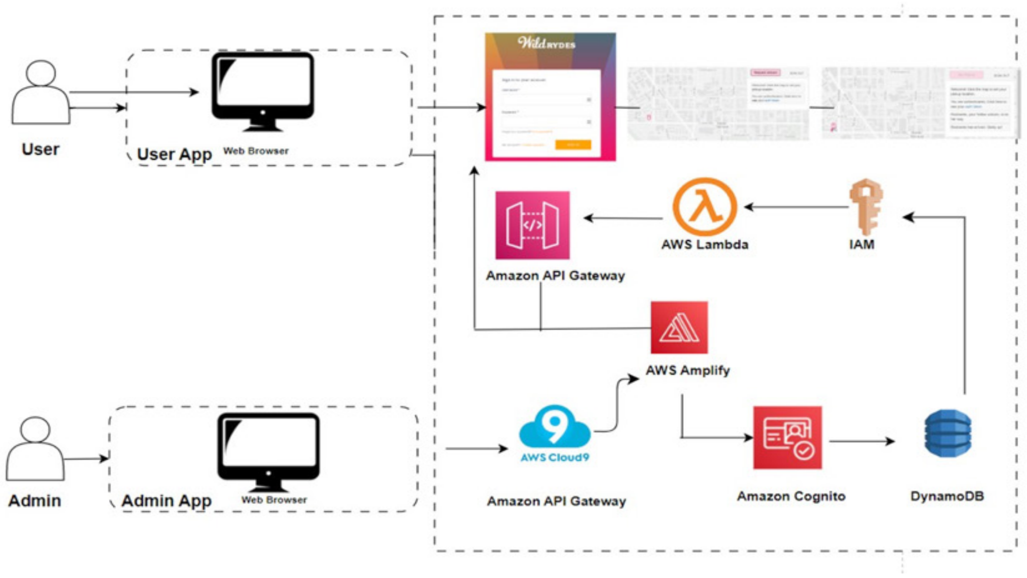


Fig.3. Serverless Computing Architecture: Wild Ryde's fleet using several AWS services

4.1 Analytics of Real-Time Data Streams

The cloud platform offered by Amazon Web Services serves as the foundation for the implementation's infrastructure (AWS). AWS services give customers and tenants the ability to swiftly build the infrastructure they need to suit their business demands. AWS manages and maintains these services. It blends;

A. a. Analyzing real-time data streams
B. b. Data analytics for dynamic web applications STEPS

1. 1. Install and configure the AWS command line interface
2. 2. Setting up the AWS Cloud9 IDE
3. 3. Static Web Hosting

- Creating the Git repository
- Deploying the site using the AWS Amplify Console

4. User management

- Creating an Amazon Cognito user pool and integrating an app with our user pool
- Update the site configuration
- Validate the deployment
- Create a new user for our user pool

5. Create a serverless backend

- Create a Lambda function to process requests
- Create an Amazon DynamoDB table
- Create an IAM role for the function
- Validate the implementation.

6. RESTful API

- Create a new Rest API,
- Deploy the API
- Update the site configuration,
- Validating the implementation.

Install and Configure the AWS Command Line Interface

Using pip, first install the AWS shell, which enables us to access the AWS command line interface. Amazon will request an AWS access key ID, and download an access key file by logging into an AWS account, going to security credentials, access keys, and clicking "Create new access key."

AWS Cloud9 IDE Setup

The AWS Cloud9 integrated development environment (IDE) allows us to develop, run, and debug code directly from our browsers. Code editors, debuggers, and terminals are included. There is no need to install or configure any files on laptop, as Cloud9 is already equipped with the most important tools for common programming languages. It is possible to access AWS resources from the Cloud9 environment using the same user account as used to log in to the AWS Management Console

Static Web Hosting

Amazon Amplify hosts static web resources such as HTML, CSS, JavaScript, and image files for static web hosting. These resources are loaded into the user's browser. We will then deploy the website we just committed to Git through the Amazon Amplify console. Setting up a location for our static web application's code is handled by the Amplify console, which also provides several features to ease the lifecycle of this application and promote best practices.

User Management

To secure the backend API, Cognito provides user management and authentication features. To manage our users' accounts, we will set up an Amazon Cognito user pool in the next step. Next step is to set up websites where users can sign up as new users, validate their email addresses, and log in to the website. Visitors to a website are asked to create an account. Only need the email address and password to register. In this app, Amazon Cognito can be set up to demand extra qualities. After registration, Amazon Cognito sends users a confirmation email with a verification code. After receiving the verification code, users must enter their email addresses and password on our website. Through the Amazon Cognito console, we can also verify user accounts with fake email addresses.

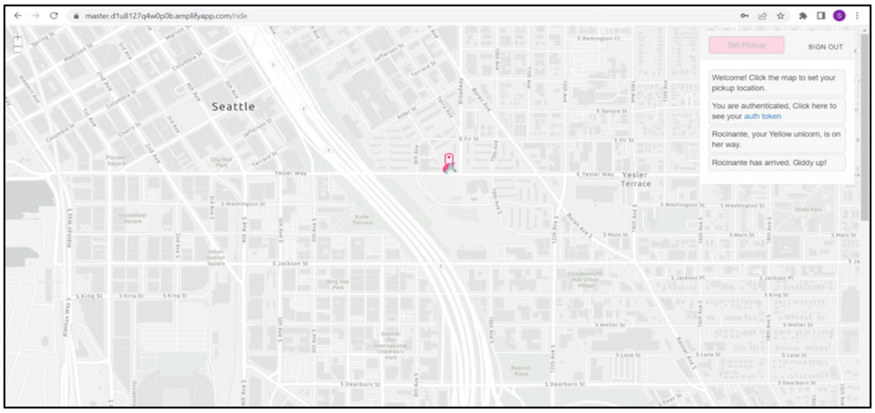
Login is possible after users confirm their account (either by email verification or by manual confirmation through the console). To log in, users must enter their username (or email address) and password. JSON Web Tokens (JWTs) are returned by Amazon Cognito after communicating with the JavaScript function, authenticating with SRP, and retrieving JSON Web Tokens (JWTs). Using JWTs, in the next step we will authenticate against the RESTful API that we created with Amazon API Gateway using information about the user's identity. There are two ways for users to sign in to Amazon Cognito. Another choice is to use Cognito User Pools for the login and sign-in features in our application, or Cognito Identity Pools for user authentication through SAML identity solutions, identity systems, or social identity providers like Facebook, Twitter, or Amazon. A user pool powers the given registration and sign-in pages.

Build a Serverless Backend

By using the lambda functions of the API, Amazon DynamoDB provides a persistence layer for storing data. In this step, we will create a backend process for our web application using AWS Lambda and Amazon DynamoDB. In the first step, we deployed a browser application that allows users to request unicorns to be shipped to the desired location. A cloud-based service is invoked by JavaScript running in the browser to fulfil these requests. Fig. 3 shows the serverless architecture using Amazon Lambda and DynamoDB. A Lambda function has been implemented that is called every time a unicorn is requested. When the front-end application requests a unicorn, the function selects one from the fleet, records the request in DynamoDB, and then provides details about the unicorn dispatched. The next step is to implement this connection. In this step, we will only isolate and test our function.

RESTful API

With Lambda and API Gateway, JavaScript is executed in the browser to communicate with a public backend API. With Amazon API Gateway, the Lambda function generated in the preceding step is made available as a RESTful API. Using our Amazon Cognito user pool from the previous step, it is secured. AJAX calls the exposed APIs by adding JavaScript client-side to our static-hosted website. The static website launched in the first stage already has a page prepared to connect with it using the API that was implemented in this step. The map-based interface located at /ride.html can be used to make a unicorn ride request. Users can choose their pick-up location on a map and request a ride after logging in via the /signin.html page by clicking the "Request Unicorn" button in the top right corner of the website. A stream was created in Kinesis into which information was written and from which information was read to allow users to select W. Built on a serverless architecture, Wild Rydes backend services are easy to use and cost-effective to maintain, allowing us to reliably meet the needs of our ever-growing user base. Unicorns have the advantage of being fast, secure, and reliable. Their numbers have increased dramatically recently, making mass transportation more accessible. Wild Rydes are produced by pairing idle unicorns with idle ryders within a short travel distance. A key factor is the shortest time to destination and proximity to the destination. Our serverless architecture streamlines and lowers the cost of scaling our backend services, enabling Wild Rydes to more consistently satisfy the demands of its steadily growing user base. Fig 4 and Fig. 5 shows, after choosing a location on a map and clicking on request unicorn, the lambda function was assigned a unicorn and display all those details on the application and also saved them on AWS Dynamodb.



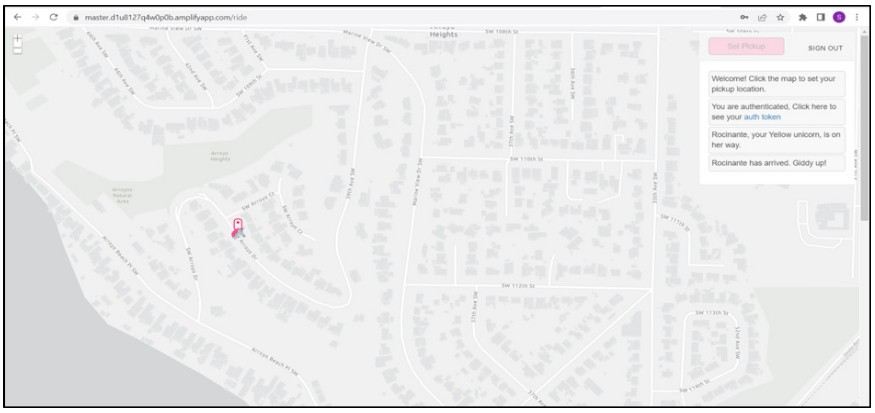
Welcome! Click the map to set your pickup location.

You are authenticated, Click here to see your auth token

Rocinante, your Yellow unicorn, is on her way.

Rocinante has arrived. Giddy up!

Fig.4. Request unicorn: location 1



Welcome! Click the map to set your pickup location.

You are authenticated, Click here to see your auth token

Rocinante, your Yellow unicorn, is on her way.

Rocinante has arrived. Giddy up!

Fig.5. Request unicorn: location 2

Results & Discussion

In Kinesis, a stream was created into which information was written and from which information was read to allow users to follow Wild Ryde's unicorns on a live map. For data analysis, two different types of programs were run to provide a foundation. Some of the experimental results include time delay, data processing performance on the Amazon platform, data aggregation using an aggregate matrix function, and performance analysis for both types of datasets. Amazon CloudWatch receives runtime metrics for Lambda's functions. The metrics displayed provide an overall view of all function runtimes. To view metrics for the unqualified resource, select Filter by. To view metrics for a specific function version or alias, select Aliases or Versions, select the alias or version, and then select Monitor. Logs generated by Lambda functions are automatically stored in Amazon CloudWatch Logs. Logging statements can be used to validate code. Click the Monitor section to view logs for a specific function version or alias.

Performance Metrics

- Duration
- Error count & success rate
- Call
- Throttle
- Total concurrent execution

A performance metric provides information about the performance of a single function call. For example, the Duration metric indicates how many milliseconds a function spends processing an event. The Average and Max metrics provide information about how much time is required for a function call. The Average and Max metrics give an idea of how fast function processes events. The latency for the real-time stream of the producer of the analytics application data stream is between 12.40 and 13.00 milliseconds (Fig.6). The obtained results are better than the work reported in [15]. The continuous peaks in the real-time data stream, where data production is proportional to time, are recorded.

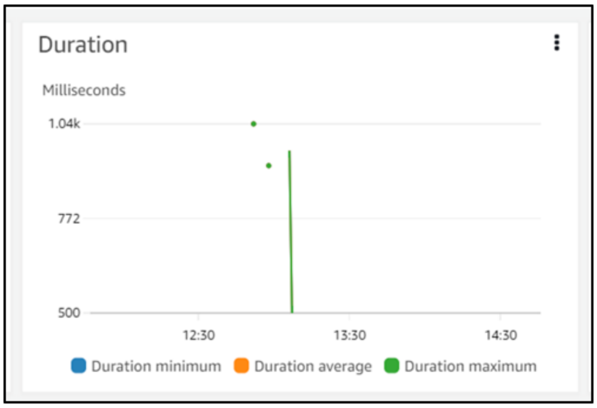


Fig.6. Function spends milliseconds processing an event (Duration metric)

The call metric of the Lambda function indicates the result of its execution. When Lambda returns an error from a function, it sends a 1 to the Errors metric. Consider summing the Errors metric with a period of 1 minute to determine the number of function errors per minute, as shown in Fig.7, success rate is 100%.

Invocations (Calls) that result in a function error - the number of times that function is called. Lambda runtime exceptions and exceptions are thrown by code including function errors. When timeouts or configuration errors occur, the runtime returns errors. By dividing errors by calls, cal-

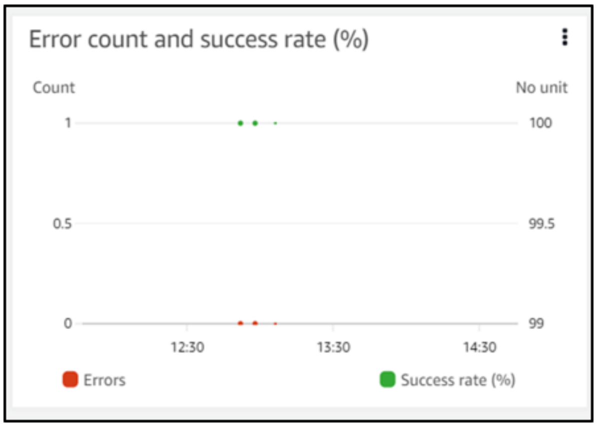


Fig.7. Error count and success rate

culation of error rate can be done. Error metrics include a timestamp that indicates when the function was called, not when the error occurred.

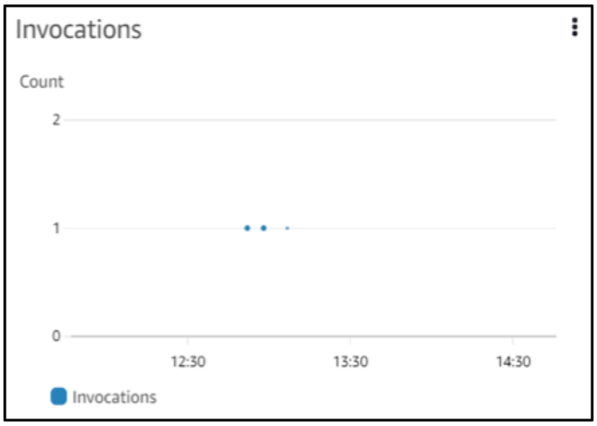


Fig.8. Invocations that result in an error

The total number of calls to function code, including both successful and unsuccessful calls. A call record is not created in response to a throttled call request or a call error. The value of Invocations represents the total number of calls that have been resolved.

A throttled call request or a call error does not result in the creation of a call record. The total number of calls that have been resolved is represented by the value of Invocations. Fig.9 shows there is no counting of throttled requests or other invocation errors. The concurrency metric reports

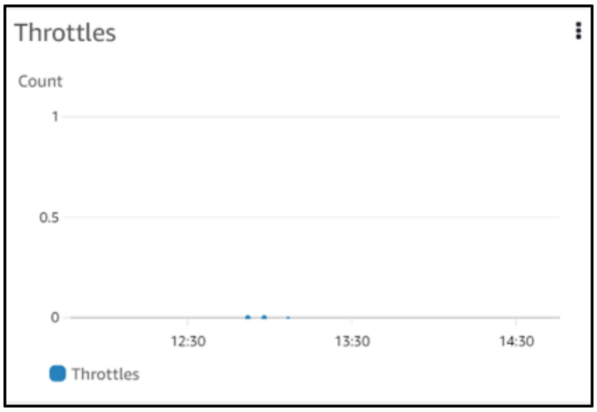


Fig.9. Invocation request

the number of instances processing events across functions, versions, aliases, or regions in a Lambda account. These metrics can be viewed along with the Max statistic to see where they are concerning concurrency limits. The number of function instances processing events at once is shown in Fig. 10. If the reserved concurrency limit for the function or the concurrent execution limitation for the Region is surpassed, more invocation requests will be throttled. The AWS Kinesis application was used to perform real-time data analysis. SQL queries are required for the analysis program to provide the appropriate analysis results.

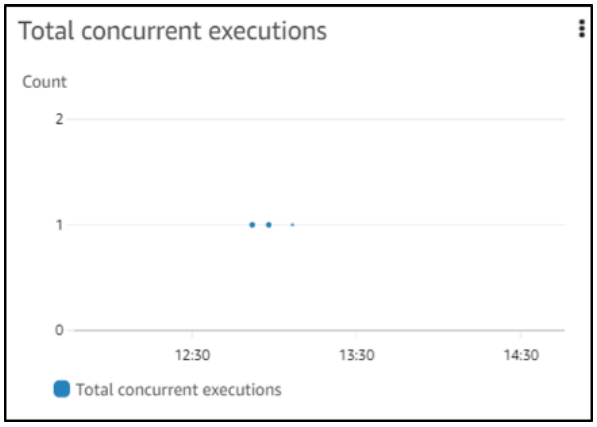
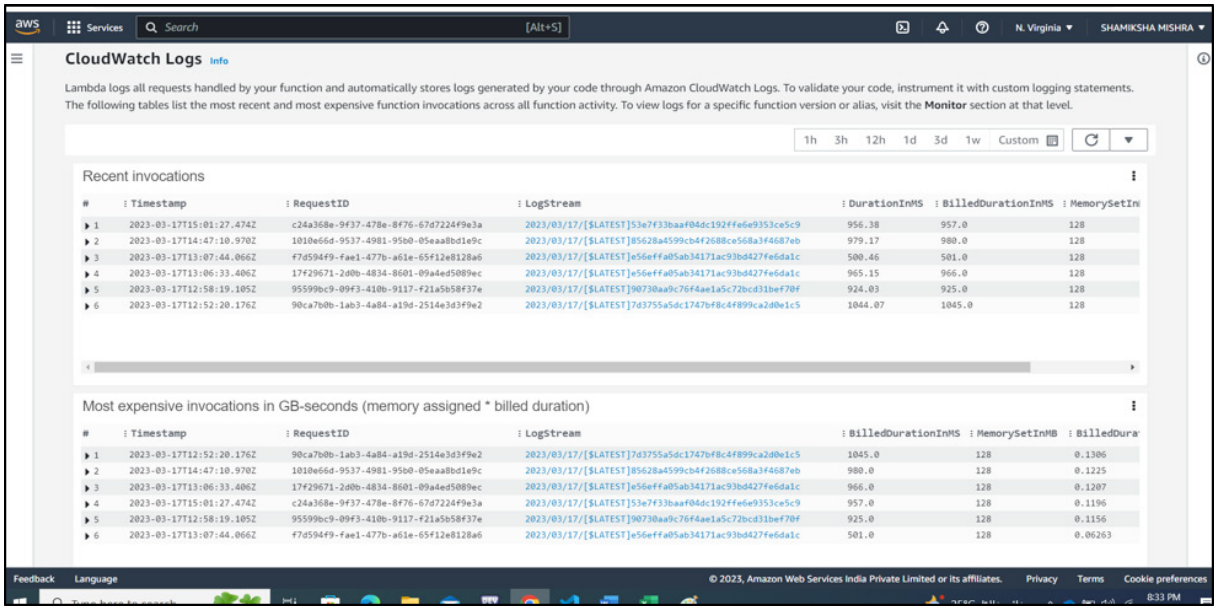


Fig.10. Total Concurrent execution

The Kinesis analytics application generated the aggregated data sets after performing an analysis of the raw input data. In

Kinesis, a feed was made to follow the unicorns of Wild Ryde on the real-time map. Before analysis, the input stream was used to collect the source data, which was made up of duplicate records and unaggregated records. We filtered the aggregated datasets for analysis after using data analytics to assess the data from these datasets. In Fig.11, two datasets with aggregated columns like minMagicPoints and maxMagicPoints are displayed in place of magic points.

Data continuously evolves, grows, and be-



CloudWatch Logs Info

Lambda logs all requests handled by your function and automatically stores logs generated by your code through Amazon CloudWatch Logs. To validate your code, instrument it with custom logging statements. The following tables list the most recent and most expensive function invocations across all function activity. To view logs for a specific function version or alias, visit the **Monitor** section at that level.

Recent invocations

#	Timestamp	RequestID	LogStream	DurationInMS	BilledDurationInMS	MemorySetIn
1	2023-03-17T15:01:27.474Z	c24a368e-9f37-478e-8f76-67d7224f9e3a	2023/03/17/[$LATEST]53e7f33baaf04dc192ffe6e9353ce5c9	956.38	957.0	128
2	2023-03-17T14:47:10.970Z	1010e66d-9537-4981-95b0-05eaa8bd1e9c	2023/03/17/[$LATEST]85628a4599cb4f2688ce568a3f4687eb	979.17	980.0	128
3	2023-03-17T13:07:44.066Z	f7d594f9-fae1-477b-a61e-65f12e8128a6	2023/03/17/[$LATEST]e56effa05ab34171ac93bd427fe6da1c	500.46	501.0	128
4	2023-03-17T13:06:33.406Z	17f29671-2d0b-4834-8601-09a4ed5089ec	2023/03/17/[$LATEST]e56effa05ab34171ac93bd427fe6da1c	965.15	966.0	128
5	2023-03-17T12:58:19.105Z	95599bc9-09f3-410b-9117-f21a5b58f37e	2023/03/17/[$LATEST]90730aa9c76f4ae1a5c72bcd31bef70f	924.03	925.0	128
6	2023-03-17T12:52:20.176Z	90ca7b0b-1ab3-4a84-a19d-2514e3d3f9e2	2023/03/17/[$LATEST]7d3755a5dc1747bf8c4f899ca2d0e1c5	1044.07	1045.0	128

Most expensive invocations in GB-seconds (memory assigned * billed duration)

#	Timestamp	RequestID	LogStream	BilledDurationInMS	MemorySetInMB	BilledDura
1	2023-03-17T12:52:20.176Z	90ca7b0b-1ab3-4a84-a19d-2514e3d3f9e2	2023/03/17/[$LATEST]7d3755a5dc1747bf8c4f899ca2d0e1c5	1045.0	128	0.1306
2	2023-03-17T14:47:10.970Z	1010e66d-9537-4981-95b0-05eaa8bd1e9c	2023/03/17/[$LATEST]85628a4599cb4f2688ce568a3f4687eb	980.0	128	0.1225
3	2023-03-17T13:06:33.406Z	17f29671-2d0b-4834-8601-09a4ed5089ec	2023/03/17/[$LATEST]e56effa05ab34171ac93bd427fe6da1c	966.0	128	0.1207
4	2023-03-17T15:01:27.474Z	c24a368e-9f37-478e-8f76-67d7224f9e3a	2023/03/17/[$LATEST]53e7f33baaf04dc192ffe6e9353ce5c9	957.0	128	0.1196
5	2023-03-17T12:58:19.105Z	95599bc9-09f3-410b-9117-f21a5b58f37e	2023/03/17/[$LATEST]90730aa9c76f4ae1a5c72bcd31bef70f	925.0	128	0.1156
6	2023-03-17T13:07:44.066Z	f7d594f9-fae1-477b-a61e-65f12e8128a6	2023/03/17/[$LATEST]e56effa05ab34171ac93bd427fe6da1c	501.0	128	0.06263

© 2023, Amazon Web Services India Private Limited or its affiliates.

Fig.11. Aggregated data after data analytics

comes more complex with every activity in our ever-evolving, vast, and frustratingly complex technological world. In the modern economy, data is one of the most valuable commodities, but without organization, segmentation, and interpretation, it is practically worthless.

Conclusion and Future work

In this study, we build, deploy and tested the methods and techniques for Amazon Lambda serverless applications. After that, we performed data analysis on these multitenant systems using real-time data streams and dynamic website click data. A system performance metric can be used to determine the performance of a single function call. In a real-time data stream, there are constant peaks where data creation increases over time. During execution, the Lambda function's call metric displays the results. Lambda runtime and code both throw exceptions when a function fails. If a timeout or configuration issue occurs, the runtime will return an error. An error or throttled call request does not result in the creation of a call record. Calls are represented by the value of Invocations, which represents the total number of calls that have been resolved. To get the right analysis results, SQL queries are needed. Real-time tracking of Wild Ryde unicorns was created using Kinesis. Prior to analysis, the input stream was used to gather the source data, which included duplicate and unaggregated records. Data analytics was used to analyze these datasets, and the aggregated datasets were filtered after analysis. Instead of magic points, two datasets are created with aggregated columns (miniMagicPoints and maximumMagicPoints). Because the platform is self-managed, users can concentrate on using the data rather than managing the platform's environment.In response to customer demands, cloud sites can expand or contract. A lack of storage space would have slowed down the internal system when executing tests with a large amount of data.

Future Work

A major focus of future research will be addressing various security problems related to cloud security and examining the latest developments. A better development environment, more efficient application assembly lines, and improved monitoring tools are available. A promising perspective, serverless integrates well with legacy systems and architectures, can come together with other technologies like Edge, and is integrated with legacy systems. There are many organizations that would benefit from serverless computing. By reducing the number of things your teams need to think about, you still allow them to develop whatever custom application functionality you require. Through the combination of the best architecture and an application, organizations can build the most innovative infrastructure for a high-performance operation.

References

[1] Li, Y., Lin, Y., Wang, Y., Ye, K., & Xu, C. Z. (2022). Serverless computing: state-of-the-art, challenges, and opportunities. IEEE Transactions on Services Computing.

[2] Naranjo, D. M., Risco, S., de Alfonso, C., Pérez, A., Blanquer, I., & Moltó, G. (2020). Accelerated serverless computing based on GPU virtualization. Journal of Parallel and Distributed Computing, 139, 32-42.

[3] Kjorveziroski, V., Filiposka, S., & Trajkovik, V. (2021). Iot serverless computing at the edge: A systematic mapping review. Computers, 10(10), 130.

[4] Sadek, J., Craig, D., & Trenell, M. (2022). Design and Implementation of Medical Searching System Based on Microservices and Serverless Architectures. Procedia Computer Science, 196, 615-622.

[5] Kjorveziroski, V., Bernad Canto, C., Juan Roig, P., Gilly, K., Mishev, A., Trajkovik, V., & Filiposka, S. (2021). IoT serverless computing at the edge: Open issues and research direction. Transactions on Networks and Communications.

[6] AL-Jumaili, A. H. A., Muniyandi, R. C., Hasan, M. K., Paw, J. K. S., & Singh, M. J. (2023). Big Data Analytics Using Cloud Computing Based Frameworks for Power Management Systems: Status, Constraints, and Future Recommendations. Sensors, 23(6), 2952.

[7] El Kafhali, S., El Mir, I., & Hanini, M. (2022). Security threats, defense mechanisms, challenges, and future directions in cloud computing. Archives of Computational Methods in Engineering, 29(1), 223-246.

[8] Tabrizchi, H., & Kuchaki Rafsanja-

ni, M. (2020). A survey on security challenges in cloud computing: issues, threats, and solutions. The journal of supercomputing, 76(12), 9493-9532.

[9] Golec, M., Ozturac, R., Pooranian, Z., Gill, S. S., & Buyya, R. (2021). IFaaSBus: A security-and privacy-based lightweight framework for serverless computing using IoT and machine learning. IEEE Transactions on Industrial Informatics, 18(5), 3522-3529.

[10] Casale, G., Artac, M., Van Den Heuvel, W. J., van Hoorn, A., Jakovits, P., Leymann, F., ... & Zhu, L. (2020). Radon: rational decomposition and orchestration for serverless computing. SICS Software-Intensive Cyber-Physical Systems, 35, 77-87.

[11] Chleier-Smith, J., Sreekanti, V., Khandelwal, A., Carreira, J., Yadwadkar, N. J., Popa, R. A., ... & Patterson, D. A. (2021). What serverless computing is and should become: The next phase of cloud computing. Communications of the ACM, 64(5), 76-84.

[12] Andi, H. K. (2021). Analysis of serverless computing techniques in cloud software framework. Journal of IoT in Social, Mobile, Analytics, and Cloud, 3(3), 221-234.

[13] Shafiei, H., Khonsari, A., & Mousavi, P. (2022). Serverless computing: a survey of opportunities, challenges, and applications. ACM Computing Surveys, 54(11s), 1-32.

[14] Muller, L., Chrysoulas, C., Pitropakis, N., & Barclay, P. J. (2020). A traffic analysis on serverless computing based on the example of a file upload stream on aws lambda. Big Data and Cognitive Computing, 4(4), 38.

[15] Ali, M. H., Hosain, M. S., & Hossain, M. A. (2021). Big Data analysis using BigQuery on cloud computing platform. Australian JofEng Inno Tech, 3(1), 1-9.

[16] Kim,Y.,&Lin,J.(2018,July).Serverless data analytics with flint. In 2018 IEEE 11th International Conference on Cloud Computing (CLOUD) (pp. 451-455). IEEE.

[17] https://spark.apache.org/

[18] Sharma, V., Nigam, V., & Sharma, A. K. (2020). Cognitive analysis of deploying web applications on microsoft windows azure and amazon web services in global scenario. Materials Today: Proceedings.

[19] Computers | Free Full-Text | IoT Serverless Computing at the Edge: A Systematic Mapping Review (mdpi.com)

[20] An empirical study on challenges of application development in serverless computing https://ieeexplore.ieee.org/abstract/document/9305905/

[21] Serverless Computing - AWS Lambda - Amazon Web Services (https://aws.amazon.com/lambda)

[22] Cloud Functions | Google Cloud (https://cloud.google.com)

[23] Azure Functions – Serverless Functions in Computing | Microsoft Azure (https://acloudguru.com/azure/functions)