Case Study: Use of AWS Lambda for Building a Serverless Chat Application

Chapter · January 2020

DOI: 10.1007/978-981-15-0790-8_24

5 authors, including:

Brijesh Choudhary
3 PUBLICATIONS 7 CITATIONS
SEE PROFILE

Chinmay Pophale
3 PUBLICATIONS 7 CITATIONS
SEE PROFILE

Aditya Gutte
Northeastern University
3 PUBLICATIONS 7 CITATIONS
SEE PROFILE

Ankit Dani
Maharashtra Institute of Technology
3 PUBLICATIONS 7 CITATIONS
SEE PROFILE

Case Study: Use of AWS Lambda for Building a Serverless Chat Application

A. Dani, C. Pophale, A. Gutte,
B. Choudhary, and S.S. Sonawani

Department of Computer Engineering
Maharashtra Institute of Technology, Pune
ankitdani1997@gmail.com,chinmay2997@gmail.com,adityagutte@gmail.com,
brijesh.choudhary7@gmail.com,shilpa.sonawani@mitpune.edu.in

Abstract. Cloud Computing in recent times has unfolded as the most popular and efficient standard for managing and delivering useful services over the internet. Out of various cloud services and techniques, serverless computing represents an evolution of cloud based programming technologies, and is an attestation to the growth and large scale adoption of cloud concepts. In this era of increasing technological advancements large internet companies like Amazon, Netflix, and LinkedIn deploy big multistage applications in the cloud which can be developed, tested, deployed, scaled, operated and upgraded independently. However, aside from gaining agility and scalability, infrastructure costs are a major hindrance for companies following this pattern due to the increasing server loads and the need to increase server space. This is where serveless computing and services like AWS Lambda comes into picture. The paper delves into the functioning of AWS Lambda along with other existing AWS services through the development of a serverless chat application which supports scalability without the addition of new servers. The paper further explores the connection of different existing services in AWS like S3, DynamoDB, CloudWatch etc. with serverless technologies like Lambda.

Keywords: AWS, Lambda, S3, DynamoDB, Cognito, SNS, IAM, Serverless, node.js, CORS, API Gateway

1 Introduction

Organizations such as WhatsApp, Instagram and Facebook have long been promoting conversational UI (User Interface) based applications that are based on either text or voice or both. This has recently resulted in growing interest regarding how relevant technologies could be used to improve business in wide industry domains.

The latest trend in cloud computing constructs is the use of serverless architecture[1]. This fast developing cloud model includes the provision of a server and its managerial operations by the cloud provider itself which eliminates the need for the user to maintain, monitor and schedule servers. This greatly simplifies the methodology of deploying the code into production process. Serverless code is employed in conjunction with code deployed in generic code designs, like microservices. Majority of the serverless applications run in state-less compute blocks that are triggered by events. The pricing depends on the number of executions rather than a already purchased number of compute capacity servers.

This paper reports a month long case study where a chat application was built based on Amazon Web Services' Lambda framework which as mentioned above, is a rapidly evolving serverless computing[2] platform. The paper also covers explorations about various services and components of Amazon Web Services and their usage[3].

2 Literature Survey

For security and authentication numerous techniques provided by AWS were discovered including the use of a Single Sign On session. OTP based authentication was found to be more reliable and secure over HTTP based methods[4]. Cloud implications relating to infrastructure elasticity, load balancing, provisioning variation, infrastructure and memory reservation size[5] were studied. This study helps to implement abstraction for serverless architecture which enables efficient cross server data management, resource allocation and isolation amongst other things[6]. Different frameworks that cater to a certain criteria of real life applications[7] were also studied. The comparison between various cloud platforms was carried out[8]. Interactive uses of various Amazon cloud services were explored[9]. The evaluation of the viability of the services offered by AWS was checked. The major reason for the shift of paradigm to severless cloud computing applications was the Infrastructure cost comparison of running a web application on AWS Lambda. This paper [10] rightly compared the whereabouts. Till now the major cloud service models were IaaS(Infrastructre as a Service), SaaS(Software as a Service) and PaaS(Platform as a Service) but with the coming of serverless computing a new cloud service model is introduced i.e. FaaS(Function as a Service)[11].

3 Amazon Web Services

AWS is a cloud service platform, providing a plethora of services for the development software packages and varied applications. The developer is not responsible and receives hassle free solutions for accessibility, availability, scalability, security, redundancy, virtualization, networking or computing as Amazon handles them automatically looking into the requirements[12]. The below sections further emphasize upon the various services employed in the development of the chat application.

3.1 Simple Storage Service (S3)

Amazon S3 in simple terms is built as a storage for the Web. It is tailored to enable easier web-scaled computing for the developing community. It has a simple interface that is used for storage and retrieval any amount of data, at any time, universally on the web.

Fundamentally, it is a key blog store. Each created S3 object has data, a key, and metadata. The key name uniquely identifies the object in a bucket. Object metadata is a set of name-value pairs and is set at the time it is uploaded. Blogs are associated with unique key names which makes them easier to sort and access. They may contain information such as Meta data (e-tag), creation time information etc. Amazon S3 is extremely durable.

3.2 Lambda (λ)

AWS Lambda is a trigger-driven computing cloud platform that allows programmers to code functions on a pay-as-you-go basis without having to define storage or compute resources. One of the most prominent advantages of AWS Lambda is that it uses abstraction and segregates server management from end user. With its use, Amazon itself handles the servers, which enables a programmer to focus more on writing code.

Lambda follows the Function-as-a-Service (FaaS) model and allows any arbitrary function to run on AWS in a wide range of programming languages including Node.js, Python, Java and C#. Users can also utilize code compiler tools, such as Gradle or Maven, and packages to build functions on a serverless platform.

Why Lambda over other Services?
Lambda enables the creation of a new execution of functions in milliseconds. The time taken to execute a function, factors in Lambda. Comparing with other services for instance EC2 (Elastic Compute Cloud) wherein a server boots up the OS before doing any work, Lambda bills a user for just the time you took rounded up-to to the next 100 ms and bill is generated for just the power computed. On the other hand EC2 bills by the second. Servers are billed till the time they are shut down by the user. The user also might be forced to pay for the computing power he/she might not be using.

For Lambda, all you need to do is code, build and upload your function. And from there, triggers will execute it however it is required. In EC2, you have to pay for the servers OS boot time also. For example, if a user has a fast Lambda function, the bill is generated for 100 ms depending on how fast the Lambda function is. But for the same function, EC2 can charge for the minute. For scaling, Lambda can scale up to a thousand parallel executions. In EC2, a user can scale only by adding more servers. A user might wind up paying for the CPU time they might not use. It is just not as granular at how well you can scale the servers which means you end up wasting money. In lambda, AWS automatically keeps the server up to date. In EC2 you can restart the system with the updated image but that still takes time and has to be done explicitly. For all above discussed reasons, Lambda is chosen as an integral framework for the application.

Table 2.1. Comparison of AWS Lambda and EC2

Feature	**Lambda**	**EC2**
Time to execute	milliseconds	seconds to minutes
Billing increments	100 miliseconds	1 second
Configuration	Function	Operating System
Scaling Unit	Parallel function executions	Instances
Maintenance	AWS	AWS and User

3.3 Identity Access Management (IAM)

IAM manages users and groups for the user. It defines ”roles” which allows a user to define pre-packaged access sets. There is a functionality to assign roles to users and we also needed to assign roles to our Lambda function and your API gateway as we built our serverless application[13]. Policies in IAM are the actual access control descriptions. A developer adds a policy to the S3 bucket to actually provide public access to that bucket. This is where the details actually live. Policies cant do anything on their own. They need to be assigned to a role or a user or a group in order to do something. We need to attach the policy to that entity in order to underline how we interact with that thing. It also manages identity providers and user account settings. Encryption keys are something that AWS can manage through IAM and it will generate and provide access to encryption keys.

3.4 DynamoDB

DynamoDB is a NoSQL database provided by AWS. It is basically a key-value storage system. It does not typically provide ACID (atomicity, consistency, isolation, durability). There is a really good chance that data might be written but may be not immediately.
For retrieving items from DynamoDB:

– Get Item: It is the easiest way to retrieve an item. User just has to provide a key to use it. The result will be a single item or if the item does not exist, it will through an error.
– Query: Queries require a hash key. If the user does not have a sort key, there is no need to do a query. User can do a ”get item” with a hash key. The client can also put a constraint on sort key for retrieving a particular set of data.

- Scan: If you dont know a key, you can use a scan. It is a brute force approach in which the entire dataset is scanned. A filter can be applied to retrieve particular information from the dataset.

In DynamoDB, a user needs to set different capacities for read and write functionalities. DynamoDB gives an option of how much consistency is needed but the prices get increased gradually.

- 1 read unit = 1 consistent read upto 4 kb per second.
- 1 write unit = 1 write up to 1 kb per second.

3.5 API Gateway

Amazon API Gateway is a service offering that allows a developer to deploy non-AWS applications to AWS backend resources, such as servers. This allows two or more programs to communicate with each other to achieve greater efficiency. A user creates and manages APIs within API Gateway, which receives and processes concurrent calls. A developer can connect to other services such as EC2 instances, AWS Elastic Beanstalk and code from AWS Lambda. In order to create an API, a developer defines its name, an HTTP function, how the API integrates with services and how requests and transfers are handled. It accepts all payloads including JSON (JavaScript Object Notation) and XML (Extensible Markup Language). An AWS user can monitor API calls on a metrics dashboard in Amazon API Gateway and can retrieve error, access and debug logs from Amazon's CloudWatch. The service also allows an AWS user to maintain different versions of an API concurrently.

3.6 Cognito

Amazon Cognito is a User Management System used to manage user pools that controls user authentication and access for mobile applications. It saves and synchronizes client data, which enables an application programmer to focus on writing code instead of building and managing the back-end infrastructure.

Amazon Cognito collects a user's profile attributes into directories referred as user pools and associates information sets with identities and saves encrypted data as key-value pairs within Amazon Cognito sync storage.

3.7 CloudWatch and CloudFront

CloudWatch is used to provide data insights, graphical representation of resources used by the application. The dash boards provided by AWS for cloud watch has an umpteen options for data crunching. Cloud Watch is essential for resource monitoring and management for developers, system operators, site reliability engineers, and IT managers.

Amazon CloudFront is a middle-ware which resides in between a user request and the S3 data center in a specific region. CloudFront is used for low latency distribution of static and dynamic web content from S3 to the user. Amazon Cloud Front is a Content Delivery Network (CDN) proxies and caches web data at edge locations as close to users as possible.

4 Case Study: Serverless Chat Application

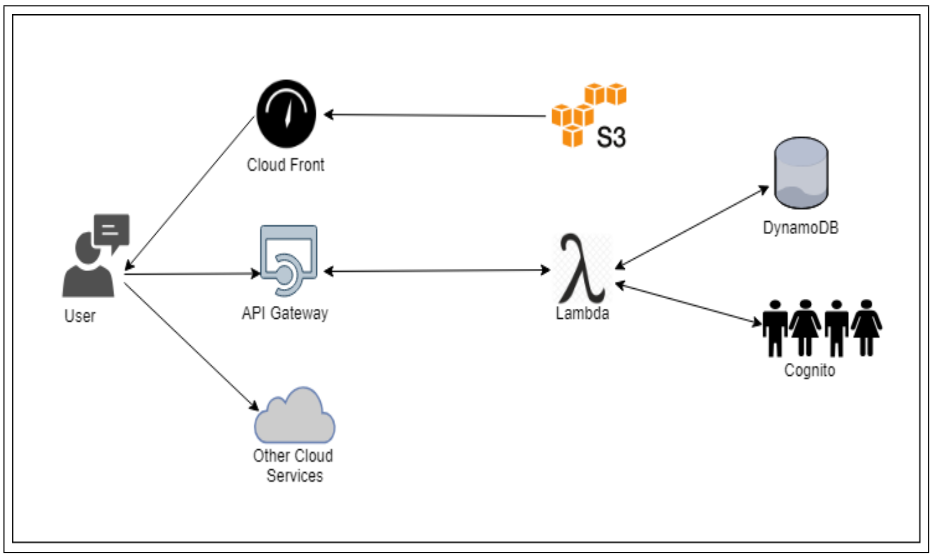

Fig. 1. Model for AWS Based Serverless Chat Application

This case study involves building a Serverless chat application to demonstrate serverless computing through the use of AWS. The application puts to use several of the functionalities & advantages of the Severless Computing platform that have been mentioned above. Following are the features of the web based chat application:

– User Authentication and cloud based security: Determines the authenticity of the user by verifying the email provided by OTP (One Time Password) generation.
– Real time messaging: Instant delivery of messages to the chat users without any delay.
– Session Monitoring: Allows the users to observe the time when a particular message was sent or received.
– Intuitive User Interface: Provides the user with a easy to use chat interface.
– Platform Independent: Can function on disparate devices such as both personal computers & mobile phones.

– Serverless: Does not involve management & maintenance of any type of servers.
– Scalable: The number of users can be augmented as required.

S3 and DynamoDB: A bucket is created in the AWS Simple Storage Service that consists of the source code of the web based chat application. The S3 bucket has been given the public access so that other Amazon cloud services are able to utilize the code. To maintain the content of the chats and the user sessions, tables have been created in the DynamoDB.

AWS Lambda: AWS lambda contains the JavaScript methods that are required to run the web application. It is the link between the S3, DynamoDB and AWS cognito that allows for the execution of the chat application program and resource management for the same. The AWS function used here is Dynamo.query() that has the following attributes: table name, projection expression (which acts like a select statement), expression attribute names (timestamp), key conversation expression (used for retrieving conversation id).

AWS Cognito and API Gateways: Amazon Resource Name(ARNs) is a naming system used to identify a resource. One of the several functionalities of API gateway are Models, Stages & Resources that are utilized for the application.

1. **Models:** It contains the JSON data of the structure of the conversation between the two users.
2. **Stages:** A stage is a named reference to a deployment, which is a snapshot of the API. You use a Stage to manage and test a particular deployment. We have used stages to configure logging, define stage variables and attach a canary release for testing.
3. **Resources:** It consists of the GET and POST methods that are used to request and send data to specific resources.

5 Conclusion and Future Scope

This paper has proposed the use of AWS Lambda for serverless Applications and merits of serverless computations. The research is supported by a case study of a serverless chat application built using AWS Lambda and other AWS cloud services. The architecture of the messenger application requires very less management and is cost efficient. Implementing the messenger application using the above mentioned technologies helped in grasping the usefulness and relevance of serverless computing and 'FaaS' in hosting dynamic applications with lots of user interaction. The future work of this messenger application includes creation of personalized groups and advanced encryption methods to enhance cyber security. Since Lambda cost model is based on the time required for the code execution, future scope also includes cost and pricing optimizations. Use of other NoSQL databases instead of DynamoDB may be tested in the future implementations of the application. Cross platform integration of AWS Lambda with other prominent serverless technology providers like Google Cloud and Microsoft Azure Functions, etc. Serverless computing with the help of AWS and other similar cloud services and their decreasing costs for the resources is here to stay for a long time changing the way we develop, test, deploy and maintain our applications.

References

1. M. Sewak and S. Singh, ”Winning in the Era of Serverless Computing and Function as a Service,” 2018 3rd International Conference for Convergence in Technology (I2CT), Pune, 2018, pp. 1-5
2. T. Lynn, P. Rosati, A. Lejeune and V. Emeakaroha, ”A Preliminary Review of Enterprise Serverless Cloud Computing (Function-as-a-Service) Platforms,” 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Hong Kong, 2017, pp. 162-169.
3. M. Villamizar et al., ”Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures,” 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Cartagena, 2016, pp. 179-182.
4. K. Swedha and T. Dubey, ”Analysis of Web Authentication Methods Using Amazon Web Services,” 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Bangalore, 2018, pp. 1-6.
5. W. Lloyd, S. Ramesh, S. Chinthalapati, L. Ly and S. Pallickara, ”Serverless Computing: An Investigation of Factors Influencing Microservice Performance,” 2018 IEEE International Conference on Cloud Engineering, Orlando, FL, 2018, pp. 159-169.
6. Z. Al-Ali et al., ”Making Serverless Computing More Serverless,” 2018 IEEE 11th International Conference on Cloud Computing, San Francisco, CA, 2018, pp. 456-459.
7. K. Kritikos and P. Skrzypek, ”A Review of Serverless Frameworks,” 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion, Zurich, 2018, pp. 161-168.
8. C. Kotas, T. Naughton and N. Imam, ”A comparison of Amazon Web Services and Microsoft Azure cloud platforms for high performance computing,” 2018 International Conference on Consumer Electronics, Las Vegas, NV, 2018, pp. 1-4.
9. H. Yoon, A. Gavrilovska, K. Schwan and J. Donahue, ”Interactive Use of Cloud Services: Amazon SQS and S3,” 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Ottawa, ON, 2012, pp. 523-530.
10. M. Villamizar et al., ”Infrastructure Cost Comparison of Running Web Applications in the Cloud Using AWS Lambda and Monolithic and Microservice Architectures,” 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), Cartagena, 2016, pp. 179-182.
11. P. Garca Lpez, M. Snchez-Artigas, G. Pars, D. Barcelona Pons, . Ruiz Ollobarren and D. Arroyo Pinto, ”Comparison of FaaS Orchestration Systems,” 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), Zurich, 2018, pp. 148-153.
12. S. Narula, A. Jain and Prachi, ”Cloud Computing Security: Amazon Web Service,” 2015 Fifth International Conference on Advanced Computing & Communication Technologies, Haryana, 2015, pp. 501-505.
13. G. McGrath and P. R. Brenner, ”Serverless Computing: Design, Implementation, and Performance,” 2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW), Atlanta, GA, 2017, pp. 405-410.