EDUZONE: International Peer Reviewed/Refereed Multidisciplinary Journal (EIPRMJ), ISSN: 2319-5045
Volume 12, Issue 2, July-December, 2023, Available online at: www.eduzonejournal.com

Advances in Cloud Operations: Insights into Serverless Computing Dynamics and Potential

Mayank Jindal

Independent Researcher, USA

ABSTRACT

Serverless architecture is a revolutionary approach in cloud computing and has provided agility to businesses to handle engineering and technological operations. It is especially beneficial for event-driven architecture because of the on-demand invocation-based processing model resulting in simplified development. This is different from traditional cloud server management featuring automatic scaling to meet various application developments that could be easily integrated with other cloud services. However, a serverless system has its very own specific issues like cold start where there might be an initial delay in executions and performance optimization is needed for a long lasting effect of serverless function. This paper offers a deep dive into AWS Lambda as a demonstration to show the features and performance optimization strategies. It will include best known methods for utilizing serverless for scalable and innovative solutions highlighting some key points like functionalities, lifecycle management problems and other challenges.

Keywords: Cloud Services, Serverless Computing, Function as a Service (FaaS), Cold Start, Event-Driven Architecture, Auto-Scaling, AWS Lambda.

INTRODUCTION

Automating operations are a must for a fast paced environment in today's business world to increase efficiency and for optimal growth opportunities [1]. Servers that could perform automated computational tasks are absolutely needed to enable productivity. However, such infrastructure was considered a larger investment for businesses due to resource limitations and larger costs. It comes with high maintenance and limitations such as need of physical spaces for data centers, security and hiring talents [2].

Cloud computing is a revolutionary solution for businesses which offers hassle free automation without owning or managing servers but with renting servers. It helps improve business models for greater scalability and removes the need of a complex and resource-intensive chore of managing servers at a lower cost. Cloud computing provides effective ways to enhance productivity and help build healthier profit margins by eliminating massive infrastructure and diverse skill sets needed for data center maintenance so businesses can concentrate on their core products and services[3].

Cloud computing services are generally provided in three main models as shown by Table 1[4, 5].

Table 1: Service models used by cloud computing platforms

Model	**Description**
Infrastructure as a Service(IaaS)	This model provides businesses physical and virtual servers, storage and networking resources to run computations and programs in the cloud.
Platform as a Service (PaaS)	PaaS is a hybrid cloud platform which allows companies to easily create, operate, maintain and control applications.
Software as a Service (SaaS)	In the SaaS model, cloud providers provide packaged, cloud-hosted software or programs.

Cloud Service providers have started offering services as a new model called Function as a service (FaaS)[6]. FaaS allows companies to focus on their code by enabling computation in response to an event without encountering cloud server management issues. While FaaS shares some features with PaaS but FaaS offers greater agility. FaaS frees developers from the headache of scalability planning, allowing applications to grow automatically as demand increases.

Amazon Web Services had introduced the first serverless platform; AWS Lambda in 2014 and similar abstractions could now be found available on all major cloud computing platforms [7]. This paper will provide an in-depth discussion of the characteristics of a major FaaS offering - AWS Lambda, any related usage issues and recommended solutions to address them. This review will help understand the importance of FaaS and specifically AWS Lambda today in the technological landscape.

Features

AWS Lambda operates on an invocation-based processing model. Unlike constantly running traditional servers, Lambda functions are initiated only upon receiving a request. This on-demand execution model aligns well with event-driven architectures while allowing more efficient resource utilization.

For example, consider a machine learning model designed to determine loan application approvals based on applicant details. A Lambda function can be created to process this decision-making script, triggering only when there is a new loan application submission.

This approach translates into a cost-effective "pay-per-use" model, where organizations incur expenses only during the active computation time of the Lambda function.[8]

Lambda could be seamlessly integrated with other AWS services which is especially helpful in facilitating complex applications and workflows. There are various AWS services such as Amazon Simple Queue Service (SQS) for queuing, Amazon Simple Storage Service (S3) for object storage, and Amazon DynamoDB for NoSQL database operations that could be integrated with Lambda functions to execute necessary actions.

Other than compatibility with additional AWS services, auto-scaling is another prominent feature of AWS Lambda which allows it to dynamically adjust resources based on the volume of requests. Lambda can automatically scale in various scenarios where applications experience fluctuating workloads by creating multiple instances of the function to handle requests independently and thereby enhancing efficiency and avoiding resource wastage [9].

AWS Lambda offers support for various runtimes across multiple programming languages by catering to the diverse needs of different tasks. For instance, Python is widely used in machine learning due to its extensive library support.

AWS Lambda accommodates this versatility, supporting languages like Node.js, Python, Ruby, Java, Go, and .NET Core. This flexibility enables organizations to adopt a serverless architecture without being restricted to a specific programming language.

Lambda lifecycle

The lifecycle of an AWS Lambda function is a crucial aspect of its architecture which governs how it handles requests and executes code. AWS Lambda initializes the function's lifecycle when it receives a request.

This lifecycle can be summarized in the following steps:

1. Downloading the Code: After receiving a request, Lambda then proceeds to download the code associated with a particular function.
2. Creation of Execution Environment: The service sets up an execution environment for that particular function. This environment decides where the function will be running.
3. Running Initialization Code: Before executing the main handler code, Lambda runs any initialization code. This stage typically includes tasks such as importing dependencies, setting up configurations, and initializing connections to other services.
4. Executing the Handler Code: Finally, the function's main handler code is executed to process the request.

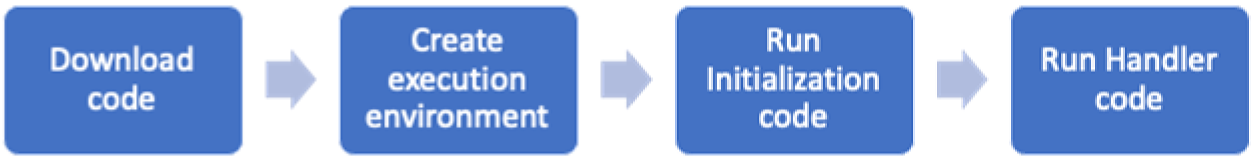


Figure 1: Lifecycle of AWS Lambda function

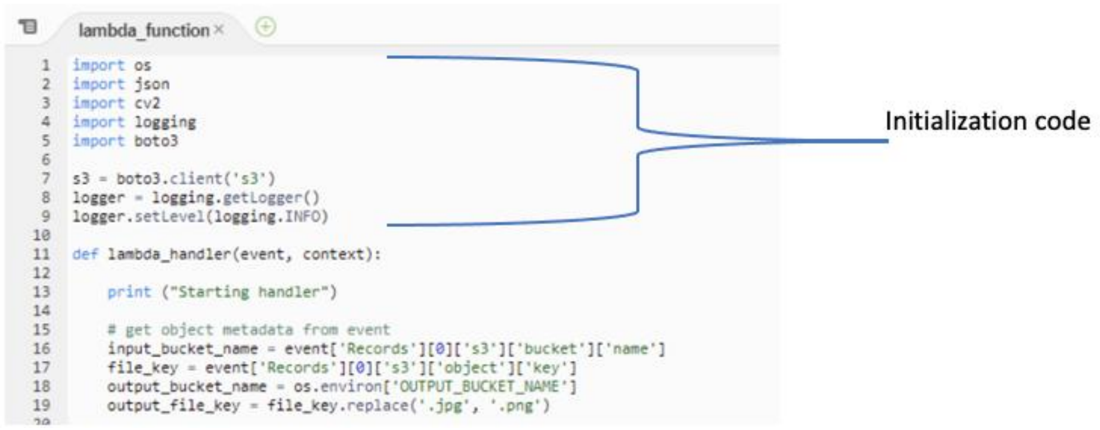
The first three steps in this lifecycle are commonly referred to as the "cold start". A cold start occurs when a function is invoked after being idle for a significant period of time or when scaling up in response to an increased traffic when necessitating the creation of new execution environments. The duration of a cold start can vary significantly which typically ranges from 100 milliseconds to over a second, and is influenced heavily by the initialization code[10].

After the cold start phase, AWS Lambda retains the execution environment by including initialized code objects for some time. This retention is intended to improve performance for subsequent requests. When a second request is served by the same execution environment, it is referred to as a "warm start". Warm starts are generally faster as they bypass the initialization phase.

However, Cold starts typically occur in under 1% of invocations but their frequency and duration can vary based on factors like traffic patterns and the specific configuration of the Lambda function. Moreover, understanding the lifecycle of AWS Lambda is essential for optimizing function performance which is crucial for reducing the impact of cold starts and ensuring efficient handling of requests.

Initialization Code and Global Variables

Initialization code is the code which runs prior to the actual handler code in the function and it includes things like importing dependencies, setting up configuration, and initializing connections to the other services . It has typically the biggest impact on the duration of a cold start.



```
import os
import json
import cv2
import logging
import boto3

s3 = boto3.client('s3')
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):

    print ("Starting handler")

    # get object metadata from event
    input_bucket_name = event['Records'][0]['s3']['bucket']['name']
    file_key = event['Records'][0]['s3']['object']['key']
    output_bucket_name = os.environ['OUTPUT_BUCKET_NAME']
    output_file_key = file_key.replace('.jpg', '.png')
```

Figure 2: Initialization code in a sample AWS Lambda function

Global variables are shared across invocations if served by the same execution environments. It is recommended to avoid global variables for storing request specific information as it improves initialization performance. Also, it could prevent variable leaks across invocations.

Provisioned Concurrency

Provisioned concurrency is a feature in AWS Lambda designed to mitigate the latency issues which are often associated with cold starts. It operates by keeping Lambda functions initialized and warm while ensuring that they are ready to respond swiftly (typically in double-digit milliseconds) when operating at a provisioned scale[11]. The essence of provisioned concurrency lies in its pre-invocation setup. In contrast to the conventional on-demand invocation model where setup activities occur during the cold start phase, these activities are performed beforehand with provisioned concurrency. This means that the initialization processes such as loading the code into the execution environment and running the initialization code are completed prior to any invocations. Consequently, a system can be quicker in responding without the typical delays of a cold start when a function is invoked. But this increased readiness and performance come at an additional cost. Provisioned concurrency incurs charges based on the amount of concurrency that is provisioned and the duration for its maintenance. This cost is above and beyond the standard pay-per-use model for Lambda invocations which makes it a factor to be considered for businesses when prioritizing response times and performance in their serverless applications.

Moreover, provisioned concurrency provides an innovative solution to organizations that can help ensure that their critical Lambda functions are always ready to execute with minimal latency and thereby providing a more consistent and reliable user experience. This feature is particularly beneficial for applications with predictable traffic patterns where the balance between performance and cost can be effectively managed.

Technical limitations and Cost

AWS Lambda has specific technical limitations that need to be considered. These include the maximum runtime of 15 minutes for a function, a payload limit of 6MB, and memory limitations of under 10GB. While some limits, such as the default 1,000 concurrent executions can be increased upon requesting AWS support, other limits are hard set by AWS and cannot be changed.

The pricing model for AWS Lambda is distinctively structured around the usage pattern of the functions[12]. It primarily comprises two components:

1. Number of Requests: Charges are incurred based on the total number of requests for functions. It counts towards this metric every time a function is invoked.
2. Execution Time: The cost also depends on the duration of code execution which is calculated from the time function begins executing until it returns or otherwise terminates, and is typically rounded up to the nearest 100ms. The amount of memory allocated to function plays a role in determining this part of the cost.

CONCLUSION

In conclusion, serverless computing has become a disruptive cloud technology innovation that can solve modern computing issues and provides greater flexibility, efficiency and affordability. It has been successfully illustrated by platforms such as AWS Lambda and is compatible with event-driven architectures which offer agility and enable development without intensive infrastructure of conventional server management. Although serverless computing presents its own set of challenges including cold start issues but also provide extended advantages like scalability, integration capabilities and cross-platform support which are of major significance. The serverless paradigm is an important new paradigm shift from conventional computing and is characterized by more effective and innovative cloud-based solutions. While technology evolves, serverless computing will become much more important to the world of cloud and company activities.

REFERENCES

[1]. Andiyappillai, N. (2021). An analysis of the impact of automation on supply chain performance in logistics companies. IOP Conference Series: Materials Science and Engineering, 1055(1), 012055. https://doi.org/10.1088/1757-899X/1055/1/012055
[2]. Knapp, K. J., Denney, G. D., & Barner, M. E. (2011). Key issues in data center security: An investigation of government audit reports. Government Information Quarterly, 28(4), 533–541. https://doi.org/10.1016/j.giq.2010.10.008
[3]. Saini, H., Upadhyaya, A., & Khandelwal, M. K. (2019). Benefits of cloud computing for business enterprises: A review (SSRN Scholarly Paper 3463631). https://doi.org/10.2139/ssrn.3463631
[4]. Wulf, F., Lindner, T., Strahringer, S., & Westner, M. (2021). Iaas, paas, or saas? The why of cloud computing delivery model selection : vignettes on the post-adoption of cloud computing. 6285–6294. https://opus4.kobv.de/opus4-oth-regensburg/frontdoor/index/index/docId/791
[5]. I. Odun-Ayo, M. Ananya, F. Agono and R. Goddy-Worlu, "Cloud Computing Architecture: A Critical Analysis," 2018 18th International Conference on Computational Science and Applications (ICCSA), Melbourne, VIC, Australia, 2018, pp. 1-7, doi: 10.1109/ICCSA.2018.8439638.

[6]. Paul, A., & Haldar, M. (2023). Introduction to serverless. In A. Paul & M. Haldar (Eds.), Serverless Web Applications with AWS Amplify: Build Full-Stack Serverless Applications Using Amazon Web Services (pp. 1–22). Apress. https://doi.org/10.1007/978-1-4842-8707-1_1

[7]. Jangda, A., Pinckney, D., Brun, Y., & Guha, A. (2019). Formal foundations of serverless computing. Proceedings of the ACM on Programming Languages, 3(OOPSLA), 149:1-149:26. https://doi.org/10.1145/3360575

[8]. Werner, S., Girke, R., & Kuhlenkamp, J. (2021). An evaluation of serverless data processing frameworks. Proceedings of the 2020 Sixth International Workshop on Serverless Computing, 19–24. https://doi.org/10.1145/3429880.3430095

[9]. Baarzi, A. F., Zhu, T., & Urgaonkar, B. (2019). Burscale: Using burstable instances for cost-effective autoscaling in the public cloud. Proceedings of the ACM Symposium on Cloud Computing, 126–138. https://doi.org/10.1145/3357223.3362706

[10]. Manner, J., Endreß, M., Heckel, T., & Wirtz, G. (2018). Cold start influencing factors in function as a service. 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion), 181–188. https://doi.org/10.1109/UCC-Companion.2018.00054

[11]. Chahal, D., Palepu, S. C., & Singhal, R. (2022). Scalable and cost-effective serverless architecture for information extraction workflows. Proceedings of the 2nd Workshop on High Performance Serverless Computing, 15–23. https://doi.org/10.1145/3526060.3535458

[12]. Boscain, S. (2023). Aws cloud: Infrastructure, devops techniques, state of art. [Laurea, Politecnico di Torino]. https://webthesis.biblio.polito.it/26672/