Developing a Serverless Application using services such as AWS Lambda, Azure Functions, or Google Cloud Functions.

A PROJECT REPORT

Submitted by

D. DEEPAK (192210646)

Under the guidance of

DR. GNANA SOUNDARI

in partial fulfilment for the completion of

COURSE CSA1590- CLOUD COMPUTING AND BIG DATA ANALYTICS FOR VIRTUAL CLUSTERS

SIMATS ENGINEERING

THANDALAM

JULY 2024

BONAFIDE CERTIFICATE

Certified that this project report titled "**Developing a Serverless Application using services such as AWS Lambda, Azure Functions, or Google Cloud Functions.**" is the bonafide work of **D. Deepak (192210646)** who carried out the project work under my supervision as a batch. Certified further, that to the best of my knowledge the work reported here in does not form any other project report.

Date: Project supervisor: Head of Department:

ABSTRACT

Serverless computing, also known simply as "serverless," represents a significant transformation in how applications are designed, deployed and executed. Contrary to traditional methods where developers manage servers, serverless allows them to focus only on coding, trusting cloud providers with infrastructure responsibilities. This paradigm shift results in dynamic resource allocation, where users are billed based on actual resource consumption rather than pre-allocated capacities. The article elucidates serverless computing's nature, highlighting its core components - Function as a Service (FaaS) and Backend as a Service (BaaS).The benefits of serverless are underscored, including cost efficiency, inherent scalability, rapid development, and reduced operational demands. Yet, it is not without limitations. Concerns such as "cold starts," potential vendor lock-in, restricted customization, and specific security vulnerabilities are discussed. Practical serverless applications include web applications, data processing, IoT backends, chatbots, and ephemeral tasks. In conclusion, while serverless computing heralds a new age in cloud technology, businesses are encouraged to discerningly evaluate its pros and cons, mainly as the landscape evolves. The future is serverless, prompting organizations to determine their readiness for this revolution.

Keywords: Serverless Computing, Function as a Service (FaaS), Backend as a Service (BaaS), Scalability, Cold Starts, and Vendor Lock-in.

INTRODUCTION

In the rapidly advancing field of cloud computing, serverless architecture has emerged as a revolutionary approach, reshaping the way applications are conceived, built, and operated. This report details a project dedicated to developing a serverless application utilizing three leading serverless platforms: AWS Lambda, Azure Functions, and Google Cloud Functions. These services epitomize the serverless paradigm by abstracting away the complexities of infrastructure management, allowing developers to concentrate on crafting application logic and functionality. Serverless computing provides the capability to execute code in response to various events or triggers—such as HTTP requests, database modifications, or scheduled tasks—without the need for manual resource provisioning or scaling. This project aims to leverage these platforms to construct a robust, scalable application that efficiently handles dynamic workloads and integrates seamlessly with other cloud services, such as databases, storage solutions, and messaging systems.

The core objectives of the project include developing individual serverless functions to perform essential business operations, configuring event triggers to automate function execution, and integrating these functions with a suite of cloud-based resources to form a cohesive application architecture. Furthermore, the project will delve into optimizing function performance to balance cost and efficiency, addressing potential issues like cold start latency, and implementing stringent security measures to safeguard application data. The report will also explore the implications of vendor lock-in and strategies to mitigate its impact, providing a

comprehensive analysis of the benefits and challenges associated with serverless computing. By the end of this report, the goal is to demonstrate how serverless technologies can streamline application development processes, reduce operational complexity, and deliver a scalable, cost-effective solution to meet modern computational demands.

Advantages of Serverless Computing

Cost Efficiency

One of the most enticing attributes of serverless computing is its cost efficiency. The pay-as-you-go model ensures businesses are billed based on actual resource consumption rather than allocated server capacity (Schleier-Smith et al.,2021). This approach eliminates the expenditure tied to idle server time, allowing companies to optimize costs in real-time. Furthermore, the absence of a dedicated infrastructure to manage translates to a significant decrease in associated overheads. Organizations adopting serverless reported a 60% reduction in infrastructure management costs compared to traditional cloud models (Li et al., 2021). Further, the one-time deployment cost of Serverless computing is 68% less than server-based computing, as seen in the following table.

Scalability

Serverless computing offers inherent scalability. Cloud providers automatically handle scaling based on the application's demand, ensuring seamless operation without manual intervention (Cloudflare, n.a.). This feature is particularly beneficial for applications that experience fluctuating traffic loads. For instance, e-commerce sites during sale events or startups experiencing rapid user growth can leverage serverless to meet dynamic demands without incurring the costs and challenges of pre-scaling.

One-time Development Cost for Server-based (EC2) Compared with Serverless (QWS lambda)			
Development	Server-based	Serverless	Difference
Days to Deploy	~25 days	~8 days	~17 days
One-time Upfront	$38,300	$12,300	$26,000
Monthly Cost	$640	$205	$(435)
			Serverless is 68% cheaper than server-based

Table 1. Comparison of Deployment Cost

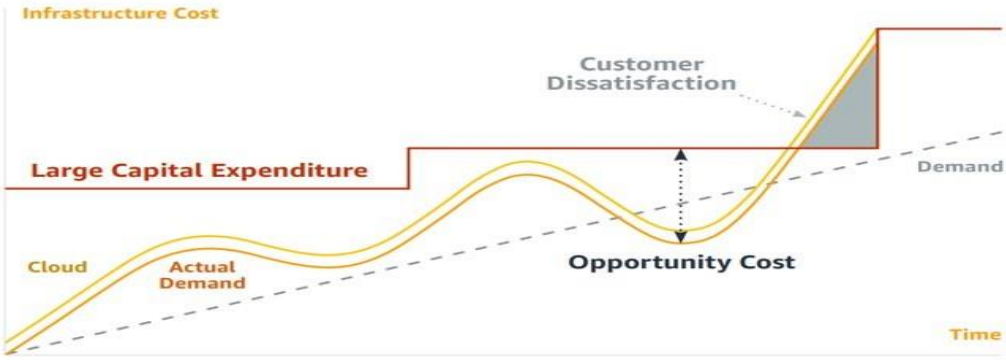

Figure 1. Benefits of Cost Savings in Dynamic Scaling

Improved Development Speed

Serverless architectures speed up software development (Rosenbaum, 2017). Without the need to manage infrastructure, developers can focus solely on writing and refining code, drastically reducing development cycles (Castro et al., 2019). Additionally, with the immediate execution environment offered by serverless platforms, rapid deployment, and iteration become the norm. This quick turnaround means businesses can respond faster to market changes, user feedback, and emerging trends.

Reduced Operational Overheads

The operational demands in traditional computing models, ranging from server management to software patching, often divert valuable time and resources from core business objectives. By outsourcing these tasks to cloud providers, serverless computing essentially eradicates these overheads (Castro et al., 2019). Deployment becomes a straightforward process, devoid of the complexities tied to server configurations. Moreover, rollback, reverting to previous versions of applications in case of errors or issues, is vastly simplified (Lloyd, 2022). This ensures businesses can maintain continuity in face of challenges.

Limitations and Concerns of Serverless Computing

While serverless computing offers numerous advantages, it's essential to understand the associated challenges and concerns.

Cold Starts

One of the most discussed limitations in serverless computing is the phenomenon of cold starts. A cold start occurs when a new instance of a function is initiated, and there's a latency involved before the Function begins executing, primarily because the environment needs to be set up from scratch (Vahid Inia, Farahani, & Aliee, 2020). This latency can affect performance, particularly in applications where swift response times are critical. For user-facing applications, the additional latency introduced by cold starts can result in sub-optimal user experiences, potentially impacting user retention and engagement.

Vendor Lock-in

Adopting serverless architectures often means aligning with a specific provider's toolset, infrastructure, and services. This dependence can lead to vendor lock-in, where migrating to a different platform becomes time-consuming and costly due to incompatible interfaces and

configurations (Taibi, Spillner, & Warch, 2021). This lock-in can limit flexibility, making it challenging for organizations to adapt to evolving business needs or leverage better offerings in the market. Limited Customization and Control As serverless platforms manage the underlying infrastructure, they often impose certain restrictions on runtime environments, memory limits, and execution durations. This can limit the scope for deep customization, which might be essential for some niche applications. While simplifying many operations, serverless frameworks may not cater to highly specialized use cases, necessitating workarounds or alternative solutions (Baldini et al., 2017).

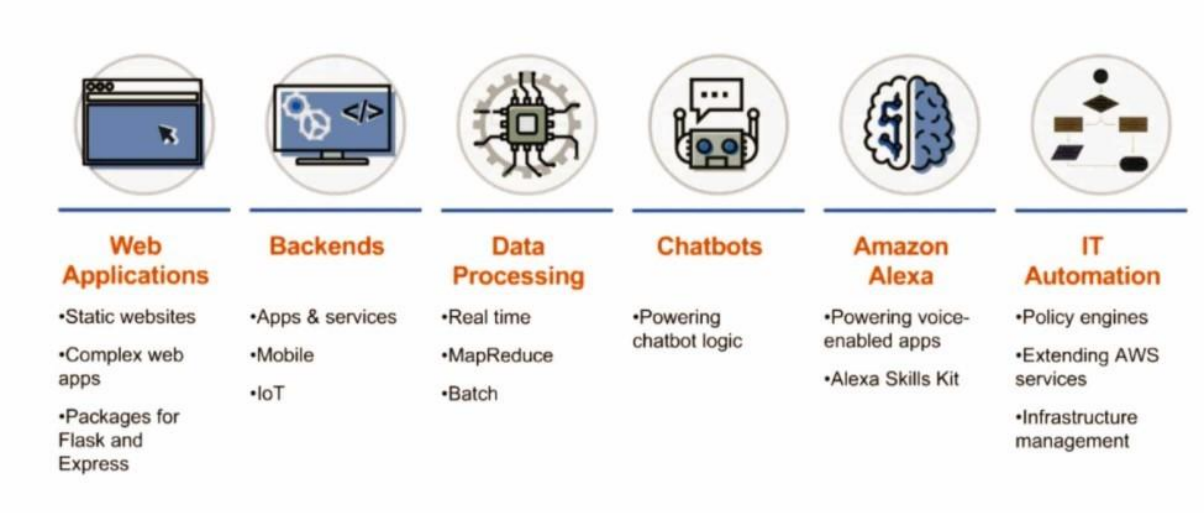

Figure 2. Use Case of Serverless Computing

Security Concerns

With serverless, while the provider primarily handles infrastructure security, the application's security remains a shared responsibility. This model means developers are responsible for securing their application code, configurations,and third-party libraries (Marin, Perino, & Di Pietro, 2022). Serverless environments may also introduce unique security vulnerabilities. For instance, the dynamic nature of serverless may expose applications to event-data injections or potential unauthorized function invocations (Marin, Perino, & Di Pietro, 2022). Functions, especially those interacting with other cloud services, can become possible entry points for attackers, making a solid case for rigorous security practices in serverless architectures (O'Meara, & Lennon, 2020).

Use Cases for Serverless Computing

The use cases for serverless computing are numerous; however, the four prominent use cases are web applications, data processing, IoT Backends, Chatbots and Services, and Ephemeral tasks.

Web Applications

Serverless computing has proven instrumental in the modern web development. Web applications, particularly those requiring dynamic content generation, can leverage serverless functions to serve content in real-time without server management overhead (Castro et al.,

2019). This approach offers scalability, ensuring the application remains responsive during traffic surges. Leveraging serverless, developers can focus solely on front-end development, knowing that the serverless provider optimizes and maintains the backend processes.

Data Processing

The sheer volume and velocity of data demand efficient processing solutions in today's digital world. Serverless frameworks cater to this need by enabling data transformation, analysis, and processing (Goli et al., 2020). As data streams in, serverless functions are invoked to process and analyse the data instantaneously, making it especially beneficial for applications requiring immediate insights. For instance, serverless computing systems can be applied in the financial sector to fasten data processes while reducing costs (Goli et al., 2020).

IoT Backend

The Internet of Things (IoT) is characterized by sporadic data bursts from numerous devices. Managing a consistent server setup for such erratic data inflows can be cumbersome. Serverless platforms shine here, seamlessly handling sudden data influxes and ensuring efficient processing and storage without manual intervention (Cicconetti, Conti, & Passarella, 2021).

Chatbots and AI Services

In customer service and support, AI-powered chatbots have gained immense traction. Serverless provides an ideal environment for these chatbots, ensuring quick responses by eliminating the need for maintaining a complete backend (Lehvä, Mäkitalo, & Mikkonen, 2018). The inherent scalability of serverless ensures that these bots can cater to many users simultaneously, offering consistent performance.

Ephemeral Tasks

Intermittent Jobs, such as CRON jobs or other scheduled tasks, can be inefficient on traditional servers due to resource wastage during idle times. With serverless, these tasks can be executed on-demand, ensuring optimal resource usage and cost efficiency (Lynn et al., 2017).

1. Error Handling

A. Lambda Functions

1. **Error Handling in Code:**
    - **Try-Catch Blocks:** Use try-catch blocks to handle exceptions in your code. Ensure that you capture and log meaningful error messages.
    - **Error Objects:** Return detailed error objects with information about the error. For example, in Node.js:

```
try {

} catch (error) {
```

```
  console. error('Error occurred:', error. message);
  return { statusCode: 500, body: JSON.stringify({ error: 'Internal Server Error' }) };
}
```

2. **Retries:**
    - **Automatic Retries:** AWS Lambda automatically retries failed invocations, but ensure idempotency in your functions to handle retries safely.
    - **Dead Letter Queues (DLQs):** Configure DLQs to capture failed events. This allows you to inspect failed events and troubleshoot issues.
3. **Error Handling Patterns:**
    - **Custom Error Classes:** Create custom error classes to categorize different types of errors and handle them appropriately.
    - **Validation:** Implement input validation to catch issues before they trigger downstream errors.
4. **Third-Party Libraries:**
    - Use libraries or frameworks that support robust error handling and recovery mechanisms.

B. API Gateway

1. **Integration Responses:**
    - Configure integration responses in API Gateway to handle different HTTP statuses and map them to meaningful error messages.
2. **Custom Error Messages:**
    - Customize error messages returned by API Gateway to provide more informative feedback.

2. Logging

A. AWS Lambda Logging

1. **Using CloudWatch Logs:**
    - **Console Logging:** Use console.log in Node.js, print in Python, or equivalent methods to log information, errors, and debug messages.

```
console.log('This is a log message');
console.error('This is an error message');
```

    - **Structured Logging:** Use structured logs (e.g., JSON format) to make it easier to parse and analyze log data.

console.log(JSON.stringify({ level: 'info', message: 'Processing event', event }));

2. **Log Levels:**
    - **Debug:** Log detailed information useful for debugging.
    - **Info:** Log general operational information.
    - **Warning:** Log potential issues that might need attention.
    - **Error:** Log errors and exceptions that affect functionality.
3. **Log Management:**
    - **Log Retention:** Configure CloudWatch Logs retention policies to manage log storage costs.
    - **Log Insights:** Use CloudWatch Logs Insights to query and analyze log data.

B. API Gateway Logging

1. **Access Logs:**
    - Enable access logging in API Gateway to log request and response data.
    - Configure log formats to include relevant request and response information.
2. **Execution Logs:**
    - Enable execution logging to capture detailed execution data for debugging.

3. Monitoring and Alerts

1. **CloudWatch Metrics:**
    - **Custom Metrics:** Publish custom metrics to monitor application-specific performance indicators.
    - **Standard Metrics:** Monitor Lambda function metrics such as invocation count, duration, error count, and throttles.
2. **Alarms:**
    - Set up CloudWatch Alarms to notify you of critical issues, such as high error rates or function timeouts.
3. **Dashboards:**
    - Create CloudWatch Dashboards to visualize key metrics and logs in one place.

4. Best Practices

1. **Error Reporting:**
    - Use error reporting tools and services (e.g., Sentry, Datadog) to get real-time error notifications and detailed stack traces.

2. **Consistent Logging Format:**
    - Ensure consistent logging formats across different functions and services to simplify log aggregation and analysis.
3. **Security Considerations:**
    - Avoid logging sensitive information or personal data to comply with privacy regulations.
4. **Documentation:**
    - Document error handling and logging strategies as part of your project documentation to ensure team-wide consistency.

CONCLUSION

serverless computing represents a pivotal shift in how applications are developed and deployed, delivering numerous benefits such as cost efficiency, automatic scaling, and streamlined operations. By abstracting the underlying infrastructure management, serverless platforms enable developers to focus on writing code and accelerating feature delivery, thus enhancing overall productivity. This model significantly reduces costs through a pay-as-you-go pricing structure, where users are charged only for the compute resources consumed during function execution and eliminates the expense of idle infrastructure.

The scalability inherent in serverless computing ensures that applications can dynamically adjust to varying loads without manual intervention, which improves performance and reliability while minimizing the need for capacity planning. However, this approach is not without its challenges. Issues such as cold start latency, which can affect response times, and the potential for vendor lock-in require careful consideration and management. Effective error handling and logging are crucial for diagnosing issues and maintaining system health, necessitating the implementation of comprehensive monitoring and alerting mechanisms.

Best practices for managing serverless environments include employing structured logging for easier analysis, configuring CloudWatch Alarms for real-time issue detection, and using Dead Letter Queues (DLQs) to capture failed events. Adhering to these practices ensures robust application performance and reliability. In summary, while serverless computing offers significant operational efficiencies and opportunities for innovation, it requires a strategic approach to overcome its limitations and fully harness its benefits. This balance between leveraging serverless advantages and addressing its challenges is key to achieving sustained success in modern cloud-based application development.

REFERENCES

1.Adam, J. (2022). Serverless architecture for increased Dev Productivity. K&C. Retrieved from https://kruschecompany.com/serverless-architecture-for-modern-apps-providers-and-caveats/

2.Arora, G., Tayal, A. & Sembhi, R. (2021). Determining the Total Cost of Ownership: Comparing Serverless and Server-based Technologies. Retrieved from https://www.strategist-hub.com/pdfs/203755_AWS_MAD_TCO_eBook_2021_Final.pdf .

3.O'Meara, W. & Lennon, R.G. (2020). Serverless Computing Security: Protecting Application logic. In 2020 31st Irish Signals and Systems Conference (ISSC). https://doi.org/10.1109/issc49989.2020.9180214 .

4.Lee, H., Satyam, K. & Fox, G. (2018). Evaluation of production serverless computing environments. In 2018 IEEE 11th International Conference on Cloud Computing (CLOUD). https://doi.org/10.1109/cloud.2018.00062 .

5.Eismann, S., Scheuner, J., Eyk, E., Schwinger, M., Grohmann, J., Abad, C., & Iosup, A. (2021). Serverless applications: Why, when, and how?," IEEE Software, 38(1), 32–39. https://doi.org/10.1109/ms.2020.3023302.

6.Cloudflare. (n.a.). Why use serverless computing? Pros and cons of serverless. Retrieved from https://www.cloudflare.com/learning/serverless/why-use-serverless/.

7. Cicconetti, C., Conti, M. & Passarella, A. (2021). A decentralized framework for serverless edge computing in the internet of things. IEEE Transactions on Network and Service Management, 18(2) 2166–2180. https://doi.org/10.1109/tnsm.2020.3023305.

8. Castro, P., Ishakian, V., Muthusamy, V., & Slominski, A. (2019). The rise of serverless computing. Communications of the ACM, 62(12), 44–54. https://doi.org/10.1145/3368454 .