IEEE Report

Subject: Cloud Functions
Author: David Darigan
Number: C00263218

Title Page

- Title of the Report
- Authors' Names
- Affiliations
- Date

Abstract

This report reviews the the impact of cloud functions (also known as ‘serverless functions’) on cloud computing. It will look at the advantages, disadvantages, limitations and challenges of using cloud functions over traditional server-based computing

1. Introduction

Networked Devices previously used a traditional monolithic model where all of the infrastructure was based on

- Traditional Servers
- Problems
- Monolith

2. Overview of Cloud Functions

What is serverless / cloud computing
What are cloud functions
Example of Cloud Function Providers (Google, AWS, MongoDB)
Microservices, Isolation, Scalable

3. System Architecture

- Core Components and Workflow
- Execution Model

4. Use Cases and Applications

The following is a list of benefits that Cloud Functions offer over traditional server-based computing

1. **Real-Time Data Processing**

    a. **AWS Lambda**: Process real-time data streams from sources like AWS Kinesis or DynamoDB Streams. For example, transforming incoming data for analytics or triggering alerts based on specific conditions.

    b. **Google Cloud Functions**: Handle real-time data from Google Cloud Pub/Sub or Firebase, such as processing user events in real-time for analytics or personalized notifications.

    c. **MongoDB Realm Functions**: Automatically process database changes in MongoDB Atlas, such as updating related records or aggregating data when a new document is inserted.

2. **Event-Driven Application**

    a. **AWS Lambda**: Respond to various AWS events, like file uploads to S3, changes in DynamoDB, or user actions in AWS Cognito. For instance, generating thumbnails for uploaded images or sending confirmation emails on user registration.

    b. **Google Cloud Functions**: Trigger functions on events from Google Cloud services, such as Cloud Storage file changes or Firestore database updates. An example is automatically backing up data when a new file is added to Cloud Storage.

    c. **MongoDB Realm Functions**: React to MongoDB database operations, like inserts, updates, and deletes. Use cases include enforcing data validation rules or sending real-time notifications to users when their data is updated.

3. **Backend for Mobile & Web App**

    a. **AWS Lambda**: Build serverless backends for mobile and web applications, handling HTTP requests via Amazon API Gateway. This can include user authentication, data retrieval, and business logic processing.

    b. **Google Cloud Functions**: Serve as the backend for Firebase-based applications, managing authentication, database interactions, and business logic without maintaining servers.

c. **MongoDB Realm Functions**: Integrate directly with MongoDB databases, providing a seamless way to execute backend logic in response to client actions, such as validating user inputs or aggregating data before returning it to the client.

4. **Scheduled Tasks**

   a. **AWS Lambda**: Run scheduled tasks using Amazon CloudWatch Events, like cleaning up old data, generating reports, or performing routine maintenance.

   b. **Google Cloud Functions**: Execute functions on a schedule using Google Cloud Scheduler for tasks like sending out periodic notifications or performing regular data backups.

   c. **MongoDB Realm Functions**: Automate recurring tasks within MongoDB Atlas, such as scheduled data exports or periodic data analysis jobs.

5. **Microservices**

   a. **AWS Lambda**: Develop microservices that are triggered by different events or HTTP requests, enabling a modular and scalable application architecture.

   b. **Google Cloud Functions**: Create microservices that interact with other Google Cloud services and external APIs, fostering a decoupled and maintainable system.

   c. **MongoDB Realm Functions**: Implement microservices that directly interact with MongoDB collections, allowing for efficient data operations and service orchestration within a serverless environment.

5. Limitations & Challenges

1. **Cold Start Latency**

   One common challenge is the cold start latency associated with cloud functions. When a function is invoked after a period of inactivity, there may be a delay as the cloud provider initializes the execution environment. This can impact the responsiveness of applications, especially for time-sensitive operations.

2. **Execution Duration Limits**

   Cloud function providers impose limits on the maximum duration a function can run. If a function exceeds this limit, it may be terminated abruptly, leading to incomplete processing of tasks. Long-running or compute-intensive tasks may need to be broken down into smaller units or migrated to other compute services.

3. **Resources**

   Cloud functions typically have resource constraints, such as memory and CPU limits, which may affect the performance and scalability of applications. Developers need to optimize their code and configuration settings to ensure efficient resource utilization and avoid performance bottlenecks.

4. **Vendor Lock-In**

   Adopting cloud functions from a specific provider may result in vendor lock-in, making it d difficult to migrate applications to another platform in the future. Developers should carefully consider the long-term implications of vendor lock-in and design their architectures with portability in mind.

5. **Debugging**

   Debugging and monitoring cloud functions can be challenging, especially in distributed and event-driven architectures. Limited visibility into function execution and resource utilization may hinder troubleshooting efforts and impact the reliability of applications.

6. **Adopting Complexity**

   Integrating cloud functions into existing workflows and infrastructure can add complexity to the operational environment. Managing dependencies, orchestrating workflows, and ensuring interoperability with other services require careful planning and coordination.

7. **Cost Management**

   While cloud functions offer a pay-as-you-go pricing model, the cost of running functions can accumulate quickly, especially for applications with high throughput or resource-intensive workloads. Monitoring and optimizing costs is essential to avoid unexpected expenses and ensure cost-effective operation.

6. Conclusion

- Summary of Key Points
- Final Thoughts and Recommendations

References

- List of Cited Works in IEEE Style

Acknowledgments (if applicable)

- Acknowledgment of Contributions from Individuals and Institutions

This format allows you to cover the main aspects of cloud functions succinctly while keeping within the 6-page limit. Each section should be concise and focused, providing enough detail to convey the key points without unnecessary elaboration.