**SWINBURNE UNIVERSITY OF TECHNOLOGY**

SYDNEY CAMPUS

**MASTER OF INFORMATION TECHNOLOGY (SOFTWARE DEVELOPMENT)**

**COS70008 –Technology Innovation Project**

# Innovation Concept

**Submitted to**

Dr. Ayesha Binte Ashfaq

**Submitted by Team 4**

Subham B K [104755498]

Prajwol Dhungana [104819239]

Manisha Nagarkoti [104767587]

Submission Date: 3rd July 2024

Word Count: 2024

## Table of Contents

## TEAM CONTRIBUTION

| Student | Contribution |
|---|---|
| Subham Bk (104755498) | • Execute Summary, Project Scope<br>• Client requirement<br>• Design concept( design 3)<br>• Draw Figure for component 1 iteration 2, 3 |
| Manisha Nagarkoti (104767587) | • Project Overview, Objective<br>• Client requirement<br>• Design concept 1(preliminary design)<br>• Draw component 1, 2, 3<br>• Draw Figure for component 1 iteration 1<br>• Documentation |
| Prajwol Dhungana(104819239) | • Client requirement<br>• Design Concept 1(contribution to implementation setup)<br>• Design 2 |

***Table 1: Team contribution***

## 1. EXECUTIVE SUMMARY

This project aims to create a strong DevOps solution using AWS services to automate the monitoring and deployment of web applications. The main goal is to monitor how quickly websites load and how often they are available, focusing on a specific set of websites listed in a JSON file. Using AWS CDK and Lambda functions, the system automatically checks these websites at regular intervals. It then uses CloudWatch to show these metrics visually and sends email alerts if any of the metrics go beyond acceptable limits. All monitoring data is stored in a database, where it can be analyzed later to improve operations and respond quickly to any issues.

## 2. INTRODUCTION

### 2.1. PROJECT OVERVIEW

This project focuses on automating website monitoring and maintenance using AWS services to ensure reliable and efficient performance. It incorporates the implementation of a CI/CD pipeline for automated deployments, comprehensive testing, and continuous deployment capabilities without human intervention. The system will employ Lambda functions for monitoring and a web crawler for assessing website metrics, coupled with automated rollback procedures to address service degradation. All project artifacts, including code and documentation, will be managed in a version-controlled repository on GitHub. It consist of three main components:

1. Building a Monitoring Application
2. Multi-Stage Pipeline with AWS CDK
3. Monitoring the Monitoring Application and Auto Rollback Feature

### 2.2. OBJECTIVE

- Develop and deploy an automated CI/CD pipeline using AWS services.
- Use Lambda function to monitor and assess critical metrics of public web applications.
- Implement unit, integration, and functional tests for comprehensive code quality assurance.
- Utilize AWS CloudWatch for real-time monitoring of web application metrics.
- Set up SMS notifications using AWS SNS to alert about critical issues.
- Automate rollback procedures to ensure service reliability and continuity.
- Store alert message in a database for analysis and auditing purposes.
- Document all project activities and resources comprehensively on GitHub.

### 2.3. SCOPE

- Deploy a Lambda function using AWS CDK in a specified AWS Region.
- Monitor critical metrics like latency and availability for selected public web applications.
- Enhance the Lambda function to include web crawling capabilities, evaluating websites stored in an S3 bucket.
- Set up AWS CloudWatch alarms to monitor performance and availability thresholds, triggering notifications (via SNS) for any anomalies.
- Store monitoring logs and metrics data in a DynamoDB database for thorough analysis and auditing.
- Develop and configure a multi-stage pipeline (Beta/Gamma and Prod) using AWS CDK for automated testing, deployment, and rollback management.
- Write, execute, and automate unit and integration tests tailored for the web crawler functionality.
- Implement procedures to automatically revert to the last stable deployment in response to alerts indicating performance degradation.
- Maintain comprehensive documentation, including README files and GitHub repositories, ensuring transparency and promoting collaboration.

## 3. CLIENT REQUIREMENT

1. **Canary Function Development**
   - Develop a canary function within a Lambda using AWS CDK to monitor the health and performance of the web application periodically.

2. **Identification and Definition of Metrics**
   - Identify critical metrics essential for monitoring the selected web application.

3. **Implementation of Monitoring**
   - Implement monitoring of the web application based on identified metrics using the canary function developed in step 1.
   - Configure alarms within the monitoring service to notify stakeholders when metrics drop below predefined thresholds.

4. **Version Control with GitHub**
   - Utilize GitHub for version control to track and manage code changes throughout the development lifecycle.

5. **Extension of Canary Function**
   - Expand the existing canary Lambda function to include web crawling capabilities.
   - Configure the web crawler to retrieve a custom list of websites stored in JSON format within an S3 bucket

6. **Scheduling and Configuration of Web Crawler**
    - Schedule the web crawler to run periodically at a 5-minute interval to ensure consistent monitoring of designated websites.

7. **Definition of Metrics for Crawled Websites**
    - Define metrics to monitor for each website crawled, such as response time, status codes, and content changes.

8. **Setting up Alarms**
    - Establish alarms in the monitoring service to alert stakeholders of potential service degradation when metrics fall below predefined thresholds.

9. **Configuration of Notification Service**
    - Configure the notification service to send alerts to designated email addresses whenever alarms are triggered, ensuring prompt awareness of issues.

10. **Multi-Stage Pipeline Creation**
    - Use AWS CDK to create a multi-stage pipeline (Beta, Gamma, and Production) deploying project code in a single AWS region.
    - Each stage should include bake times for artifact preparation, code review processes, and test blockers to ensure deployment quality.

11. **Unit/Integration Testing**
    - Develop comprehensive unit and integration tests specifically tailored for the web crawler.
    - Integrate these tests into the CI/CD pipeline to validate crawler functionality and reliability.

12. **Monitoring and Alarming**
    - Implement metrics monitoring to track the operational health of the web crawler, including memory usage and processing time per run.
    - Configure alarms to trigger notifications for potential service degradation based on predefined metric thresholds.

13. **Rollback Automation**
    - Implement automated rollback mechanisms within the pipeline to revert to the last stable build in response to metric alarms or deployment failures.
    - Ensure automated rollback processes occur seamlessly without human intervention to minimize downtime and mitigate risks.

14. **Documentation Management**
    - Manage README files and GitHub repositories.
    - Provide comprehensive documentation and guidance on deploying, configuring, and maintaining the pipeline and associated components.

## 4. DESIGN CONCEPT

### 4.1. DESIGN CONCEPT 1: INITIAL IMPLEMENTATION

#### 4.1.1. CONTRIBUTION TO IMPLEMENTATION SETUP

1. **Repository Setup:**
   - GitHub for version control.
   - All code and documentation are maintained within this repository.

2. **Coding Standards:**
   - JavaScript as the primary programming language.
   - Followed JavaScript coding standards and best practices.

3. **Commenting Standards:**
   - All code is well-documented with comments.

4. **Environment Setup:**
   - Created an AWS account.
   - Using Visual Studio Code (VS Code) as the code editor.
   - Installed necessary packages using the command line, such as AWS CLI, CDK, and npm.
   - Linked the AWS account with VS Code to enable direct deployment and management from the code editor.
   - Linked VS Code with a GitHub account to maintain version control of the code.

#### 4.1.2. PRELIMINARY DESIGN

The project is further divided into three main components. They are:

- Component 1: Build a Monitoring Application
- Component 2: Multi-Stage Pipeline with CDK
- Component 3: Monitoring the Monitoring Service and Auto Rollback Feature

#### 4.1.2.1. COMPONENT 1: BUILD A MONITORING APPLICATION

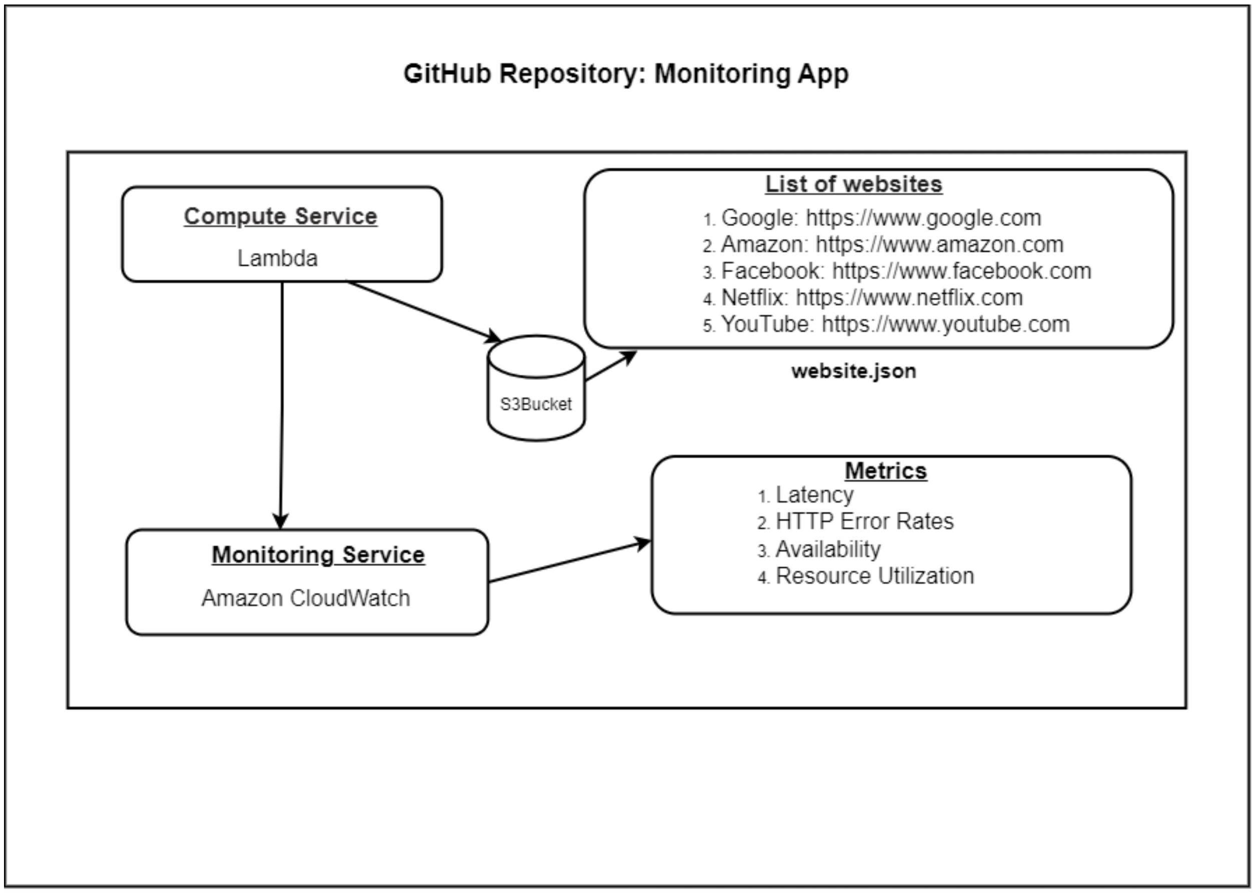
GitHub Repository: Monitoring App
Compute Service
Lambda
List of websites
1. Google: https://www.google.com
2. Amazon: https://www.amazon.com
3. Facebook: https://www.facebook.com
4. Netflix: https://www.netflix.com
5. YouTube: https://www.youtube.com
website.json
S3Bucket
Metrics
1. Latency
2. HTTP Error Rates
3. Availability
4. Resource Utilization
Monitoring Service
Amazon CloudWatch

*Figure 1: Monitoring Application Service (Iteration 1)*

##### 4.1.2.1.1. METHODOLOGY

- Develop a Lambda function using AWS CDK.[1]
- Identify and monitor key metrics for a public web application.
- Select a list of websites and store them in a JSON file.
- Create an S3 bucket using CDK and store the JSON file of websites in it.[2]
- Implement version control and documentation on GitHub.

##### 4.1.2.1.2. DESIGN CONSTRAINT

- SNS cannot directly send messages to RDS because RDS doesn't natively subscribe to SNS notifications.
- Each SNS notification would require manual processing and transformation before being stored in RDS, leading to potential delays and increased complexity in handling data flow.
- RDS requires schema definition and management, unlike DynamoDB which offers scheme less flexibility.
- Limited to one AWS region.
- Crawler must run on a 5-minute cadence without fail.

#### 4.1.2.1.3. Specification

- S3 bucket for storing website list which is in json file.
- Develop a Lambda function that acts as a web crawler, calling websites from a list stored in an S3 bucket. The function will periodically check the health and performance metrics (latency, availability) of these websites.
- Use RDS to store logs of each check performed by the Lambda function, including timestamps, metrics, and any anomalies detected.
- Implement AWS CloudWatch to monitor the performance metrics collected by the Lambda function. Set up alarms to trigger when metrics exceed predefined thresholds, indicating potential issues.
- Utilize AWS Lambda for executing the web crawler without needing to manage server infrastructure, ensuring scalability and cost-efficiency.
- Metrics: Availability, Latency, resources utilization and HTTP error rates.

#### 4.1.2.1.4. Vulnerability Analysis

- Ensure the Lambda function is secure and resilient to attacks.
- Monitor for unexpected spikes in metrics that may indicate security issues.
- Secure the S3 bucket to prevent unauthorized access.

#### 4.1.2.1.5. Justification of Design

1. **Use of AWS Lambda for Serverless Architecture**

- Eliminates the need for server provisioning and management.
- Facilitates rapid deployment and updates, supporting CI/CD practices.
- Removes the complexity of managing infrastructure, focusing solely on code.
- Ensures each function execution is independent, enhancing reliability and scalability.

2. **Use of AWS CloudWatch for Alarm and Monitoring**

- Provides detailed insights into performance and operational health.
- Allows creation of custom metrics and alarms tailored to specific needs.
- Seamlessly integrates with Lambda, SNS, and DynamoDB.
- Triggers automated actions such as invoking Lambda functions or sending SNS notifications when alarms are triggered.

#### 4.1.2.2. Component 2: Multi-Stage Pipeline with CDK

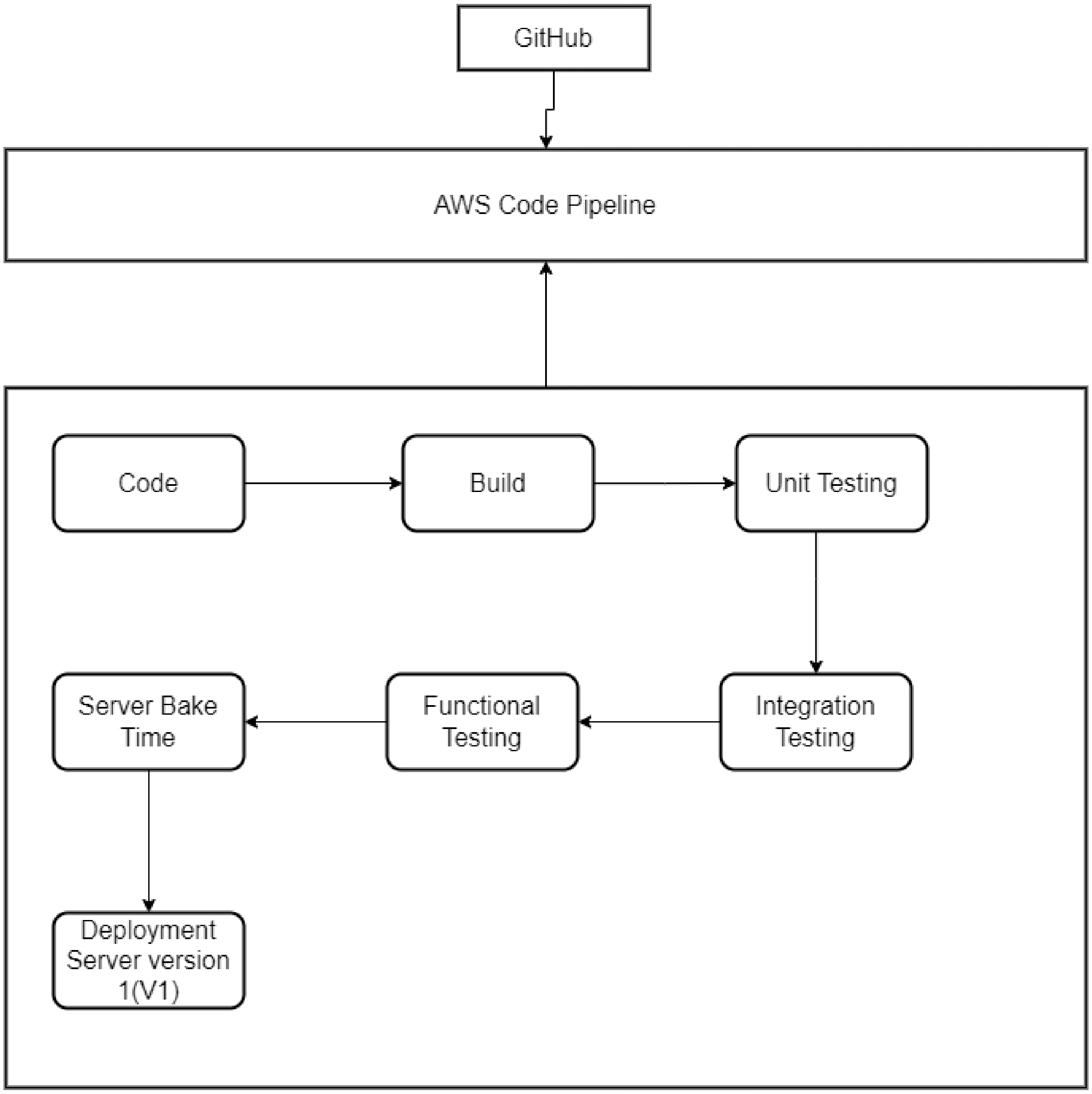

***Figure 2: CDK CICID Pipeline (Iteration 1)***

#### *4.1.2.3. Component 3: Monitoring the Monitoring Service and Auto Rollback Feature*

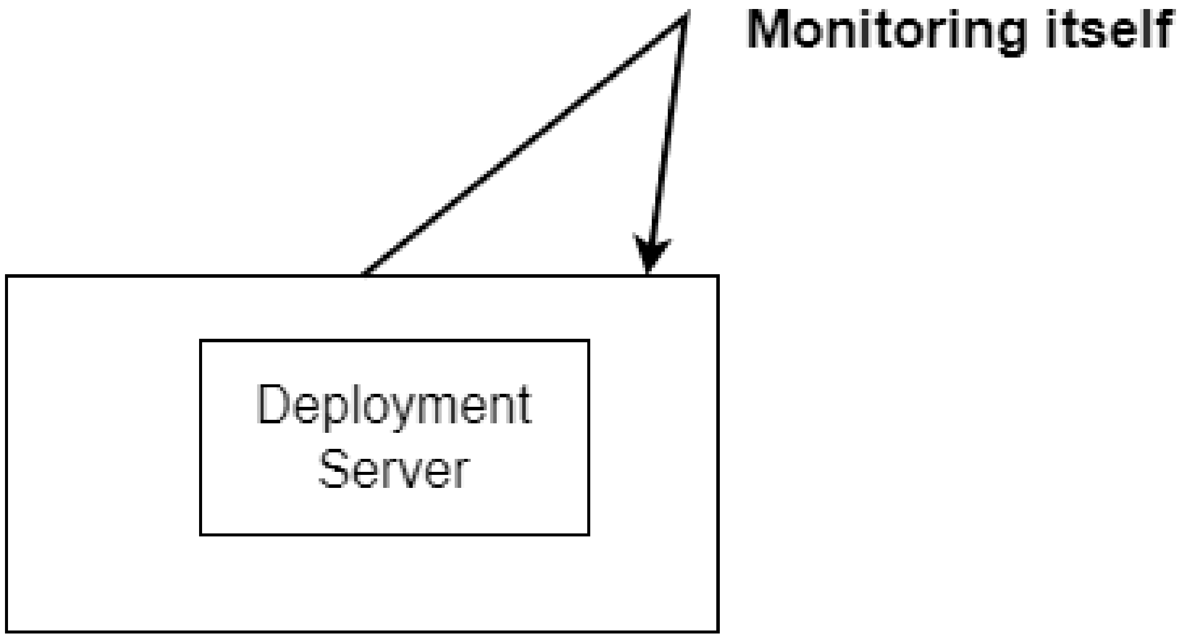

***Figure 3: Monitoring the Monitoring Service (Iteration 1)***

## 4.2. DESIGN 2

### 4.2.1. Component 1: Build a Monitoring Application

GitHub Repository: Monitoring App

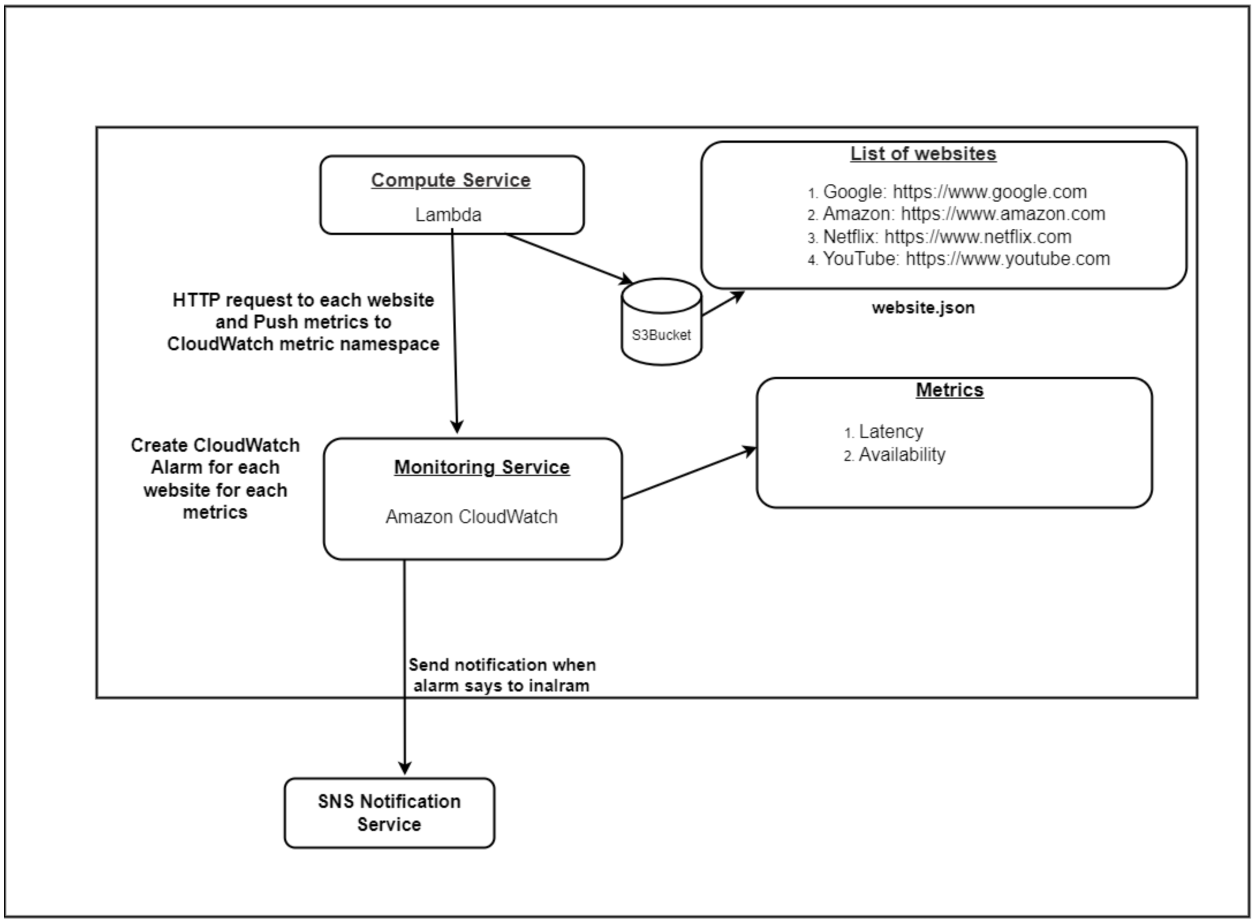
Compute Service
Lambda
List of websites
1. Google: https://www.google.com
2. Amazon: https://www.amazon.com
3. Netflix: https://www.netflix.com
4. YouTube: https://www.youtube.com
website.json
S3Bucket
HTTP request to each website and Push metrics to CloudWatch metric namespace
Create CloudWatch Alarm for each website for each metrics
Monitoring Service
Amazon CloudWatch
Metrics
1. Latency
2. Availability
Send notification when alarm says to inalram
SNS Notification Service

*Figure 4: Monitoring Application Service (Iteration 2)*

#### 4.2.1.1. *Methodology*

- Develop a Lambda function using AWS CDK.
- Create cloud metrics and publish metrics via lambda.
- Extend the canary Lambda function to include web crawling capabilities.
- Configure the lambda function to read a custom list of websites from an S3 bucket and monitor them periodically.
- Set up CloudWatch alarms for metrics.
- Configure Amazon Simple Notification Service (SNS) to send email alerts to stakeholders when CloudWatch alarms are triggered.

#### 4.2.1.2. DESIGN CONSTRAINT

- Limited to one AWS region.
- Crawler must run on a 5-minute cadence without fail.
- Manage resource utilization to avoid excessive costs.

#### 4.2.1.3. SPECIFICATION

- Custom CloudWatch metrics
- CloudWatch Alarm to monitor the metrics.
- SNS for email notification [3]

#### 4.2.1.4. VULNERABILITY ANALYSIS

- Unauthorized user shouldn't receive the mail.
- Protect lambda and CloudWatch service from the unauthorized user.

#### 4.2.1.5. JUSTIFICATION OF DESIGN

- **Broadcast to Multiple Subscribers**: SNS standard topics allow a single message to be sent to multiple subscribers simultaneously.
- **Real-time Notifications**: SNS delivers messages in near real-time, making it suitable for immediate alerts.
- **Multiple Delivery Protocols**: SNS supports various protocols (HTTP/S, email, SMS, Lambda), providing flexibility in message delivery.
- **High Throughput**: SNS standard topics can handle a high volume of messages per second.

## 4.3. DESIGN 3

### 4.3.1. COMPONENT 1: BUILD A MONITORING APPLICATION

GitHub Repository: Monitoring App

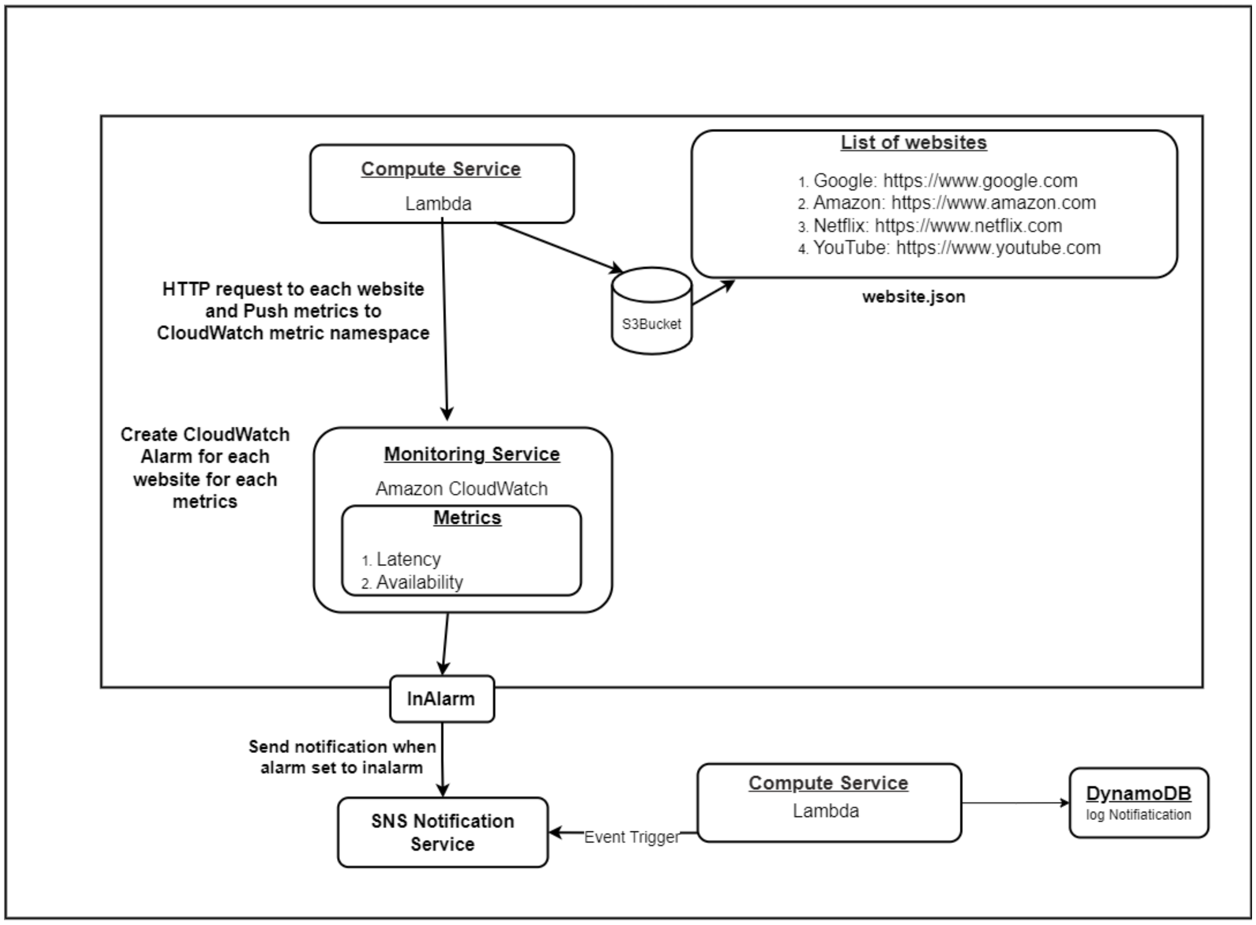
Compute Service
Lambda
List of websites
1. Google: https://www.google.com
2. Amazon: https://www.amazon.com
3. Netflix: https://www.netflix.com
4. YouTube: https://www.youtube.com
S3Bucket
website.json
HTTP request to each website and Push metrics to CloudWatch metric namespace
Create CloudWatch Alarm for each website for each metrics
Monitoring Service
Amazon CloudWatch
Metrics
1. Latency
2. Availability
InAlarm
Send notification when alarm set to inalarm
SNS Notification Service
Event Trigger
Compute Service
Lambda
DynamoDB
log Notifiatication

*Figure 5: Monitoring Application Service (Iteration 3)*

#### 4.3.1.1. Methodology

- Develop a Lambda function using AWS CDK.
- Create cloud metrics and publish metrics via lambda.
- Lambda function to get trigger from SNS topic.
- Lambda function to store notification log to DynamoDB.

#### 4.3.1.2. Design Constraint

- Manage resource utilization to avoid excessive costs.
- Limited to one AWS region.
- No logs records are stored in Database unless notification is invoke.
- Won't invoke lambda function if notification is not sent.

#### 4.3.1.3. Specification

- DynamoDB to store SNS logs.
- Lambda to insert data for SNS log in DynamoDB when new SNS triggers.

#### 4.3.1.4. Vulnerability Analysis

- Implementing IAM roles and policies to restrict access.
- Ensuring data privacy and security.
- Resource and cost management.

#### 4.3.1.5. Justification of design

The project's design offers strategic benefits by setting an event trigger from SNS to call a Lambda function and storing notification logs in DynamoDB.

1. **Instantaneous Event Reaction:**
    - **SNS to Lambda Trigger**: The SNS to Lambda Trigger method allows Lambda functions to be promptly notified of new messages posted to SNS topics.

2. **Cost-effectiveness and Serverless Scalability:**
    - **Lambda**: Lambda functions automatically scale to accommodate incoming alerts, charging only for real execution time, ensuring cost optimization in line with variable notification volumes without server setup or management.

3. **Smooth Combination and Adaptability:**
    - **Integration with DynamoDB:** AWS's DynamoDB, a managed NoSQL database service, can be seamlessly integrated with Lambda, enabling Lambda to log notification details into DynamoDB tables for reliable and long-lasting storage.
4. **Excellent Durability and Availability:**
    - **DynamoDB**: The system ensures the reliability and longevity of stored notification logs through built-in replication and multi-AZ deployment options, thereby enhancing the integrity of logging and monitoring processes.[4]

5. **Analysis and Auditing:**
    - **DynamoDB enables easy auditing** and analysis of previous notification data, allowing for querying and reporting, providing insights into operational data, trends, and patterns related to notifications.

## 5. REFERENCE

*[1] Lambda — AWS Cloud Development Kit 2.147.3 documentation* 2020, Amazon.com, viewed 3 July 2024, <https://docs.aws.amazon.com/cdk/api/v2/python/aws_cdk.aws_ses_actions/Lambda.html#lambda>.

*[2] CloudFront Origins for the CDK CloudFront Library — AWS Cloud Development Kit 2.147.3 documentation* 2020, Amazon.com, viewed 3 July 2024, <https://docs.aws.amazon.com/cdk/api/v2/python/aws_cdk.aws_cloudfront_origins/README.html#s3-bucket>.

*[3] What is Amazon SNS? - Amazon Simple Notification Service* 2024, Amazon.com, viewed 3 July 2024, <https://docs.aws.amazon.com/sns/latest/dg/welcome.html>.

*[4] What is Amazon DynamoDB? - Amazon DynamoDB* 2024, Amazon.com, viewed 3 July 2024, <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html>.