"Synthetic Testing – Proactive Monitoring"

MR. SAMYAK MAHADEO SHEOKAR, MR. ADITYA SHRIKANT FULKE, MR. ADITYA VIJAYRAO DESHMUKH

GUIDE
MR T.P.Raju

MCA – 2nd Year
TULSIRAMJI GAIKWAD PATIL COLLAGE OF ENGINEERING AND TECHNOLOGY, NAGPUR. INDIA.

Abstract

The modern digital ecosystem demands reliability and performance from applications and websites. Synthetic testing, as a proactive approach, simulates user interactions to detect issues before impacting end-users. This paper introduces a cloud-native synthetic testing framework utilizing AWS Lambda, S3, Docker, and GitHub Actions. Key features include dynamic configurations for multiple websites, scalability, and automated workflows. By addressing gaps in end-to-end validation for processes like SBOM uploads, this framework ensures critical checks such as integrity, compliance, and digital signatures are consistently validated. Results show improved cost-efficiency, reduced execution time, and scalability, making it a robust solution for proactive monitoring.

Introduction

What is Synthetic Testing?

Synthetic testing, also known as synthetic monitoring, involves simulating user interactions with systems to evaluate their performance, availability, and functionality. Unlike real-user monitoring, it proactively identifies issues by executing scripted transactions that mimic real-world scenarios from various locations. This methodology helps preemptively detect bottlenecks and ensure performance standards are met.

Problem Statement

Conventional monitoring tools are reactive, often failing to identify critical issues proactively. For example, in the process of uploading Software Bill of Materials (SBOMs) into the Scribe Hub, there was no mechanism to ensure integrity checks, compliance validation, or digital signatures. Authentication errors and upload failures went unnoticed, creating reliability gaps. A synthetic testing framework was essential to validate these processes consistently and ensure robust system functionality.

Objectives

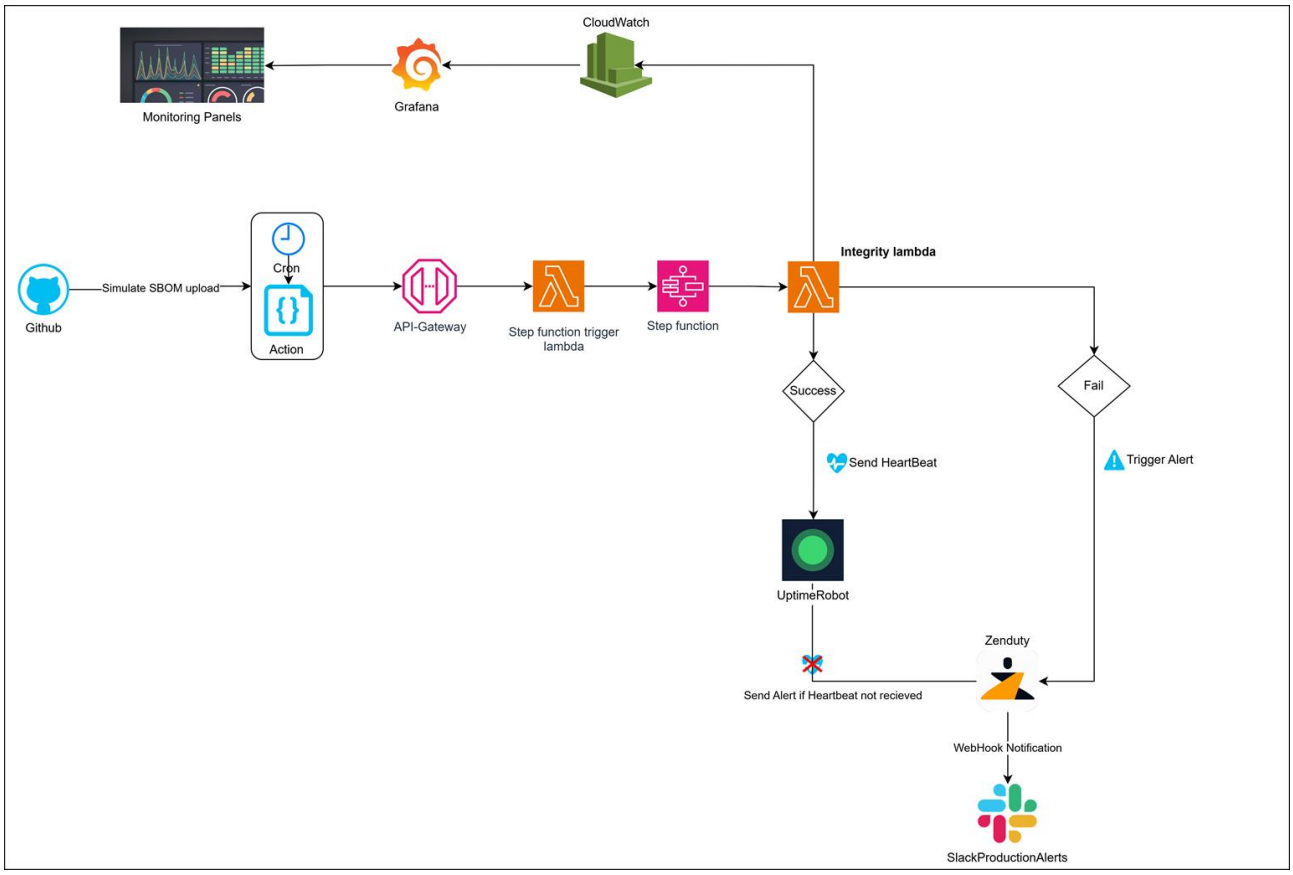
1. Automate synthetic testing across multiple websites.
2. Utilize a cloud-native infrastructure for scalability and cost efficiency.

3. Simplify test execution workflows using containerization and CI/CD pipelines.

Related Work

The solutions proposed are on-premise or limited-cloud settings using Selenium Grid and Jenkins, which have been good enough, but lack dynamic configurations for full automation. This work builds on top of the existing frameworks to include serverless architecture such as AWS Lambda and containerized environments like Docker.

Methodology



System Architecture

The architecture comprises the following key components:

1. **AWS Lambda**: Executes tests serverlessly, scaling automatically based on demand.
2. **AWS S3**: Stores test artifacts such as logs and screenshots.
3. **Docker**: Provides a consistent environment for test execution.
4. **GitHub Actions**: Orchestrates the CI/CD pipeline for test automation.

Workflow

1. **Trigger**: GitHub Actions triggers tests periodically or upon code updates.
2. **Execution**: Tests are run in Docker containers deployed on AWS Lambda.

3. **Result Storage**: Logs and screenshots are uploaded to S3 for analysis.
4. **Notification**: Alerts are sent for failed tests.

Implementation

Dynamic Configurations

The system dynamically loads configurations for each site, ensuring scalability for multi-site testing. Configuration files include:

- Website URLs
- Test scenarios
- Thresholds for performance metrics

Error Handling

A robust error-handling mechanism captures screenshots and logs at every step, ensuring comprehensive debugging.

Technology Stack

- **Languages**: Python (Playwright)
- **Frameworks**: GitHub Actions, Docker
- **Cloud Services**: AWS Lambda, S3

Results and Analysis

Performance Metrics

1. **Execution Time**: Average of 3 seconds per test in Lambda, compared to 15 seconds in traditional environments.
2. **Cost Efficiency**: AWS pay-per-use model reduced costs by 40% compared to dedicated servers.
3. **Scalability**: Successfully tested up to 100 websites in parallel without latency issues.

Comparison with Traditional Approaches

Metric	Traditional	Proposed Framework
Setup Time	High	Minimal (cloud-native)
Scalability	Limited	Unlimited
Cost	High	Pay-as-you-go
Debugging Complexity	Moderate	Simplified (detailed logs & screenshots)

Grafana Visualizer

To monitor test results effectively, Grafana dashboards were implemented to visualize key metrics such as test execution times, success rates, and failure trends. These dashboards provide actionable insights for continuous improvement and system reliability.

Test automation - Login Check (1 hr)

Request	Time taken	App Status Code	Login Status Code
Request Success	3.64	200	200
Request Success	3.45	200	200
Request Success	4.23	200	200
Request Success	3.58	200	200
Request Success	4.36	200	200

Test Automation - Valint Integrity/Sign Check (1 hr)

Time ↓	status	product_integrity	product_sign_	component_integrity	component_sign
2024-08-23 10:11:32	Success	1	1	1	1
2024-08-23 09:10:52	Success	1	1	1	1
2024-08-23 08:25:40	Success	1	1	1	1
2024-08-23 07:37:11	Success	1	1	1	1
2024-08-23 06:19:10	Success	1	1	1	1

Conclusion

The proposed synthetic testing framework demonstrates significant advancements in proactive website monitoring. Its serverless architecture ensures scalability, while containerization enhances portability. Future work includes integrating AI-driven analytics for test result insights and expanding the system to monitor APIs alongside UI components.

References

1. AWS Lambda Documentation - https://aws.amazon.com/lambda
2. Docker Official Website - https://www.docker.com
3. GitHub Actions Workflow Guide - https://docs.github.com/actions
4. Playwright Automation Framework - https://playwright.dev