A Serverless Architecture for Wireless Body Area Network Applications

Pangkaj Chandra Paul(✉), John Loane, Fergal McCaffery, and Gilbert Regan

Regulated Software Research Centre, Dundalk Institute of Technology, Dundalk, Co. Louth, Ireland
paulp@dkit.ie

Abstract. Wireless body area networks (WBANs) have become popular for providing real-time healthcare monitoring services. WBANs are an important subset of Cyber-physical systems (CPS). As the amount of sensing devices in such healthcare applications is growing rapidly, security, scalability, availability and privacy are a real challenge. Adoption of cloud computing is growing in the healthcare sector because it can provide high scalability while ensuring availability and affordable healthcare monitoring services. Serverless computing brings a new era to the design and deployment of event-driven applications in cloud computing. Serverless computing also helps the developer to build a large application using Function as a Service without thinking about the management and scalability of the infrastructure. The goal of this paper is to propose a dependable serverless architecture for WBAN applications. This architecture will improve the dependability of WBAN applications through ensuring scalability, availability, security and privacy by design, in addition to being cost-effective. This paper presents a detailed price comparison between two leading cloud service providers. Additionally, this paper reports on the findings from a case study which evaluated security, scalability and availability of the proposed architecture. This evaluation was conducted by load testing and rule-based intrusion detection.

Keywords: Wireless body area network · Cloud computing · Serverless architecture

1 Introduction

With the rapid growth of wireless communication and sensor technology, Wireless body area network (WBAN) applications are an increasingly important technology in providing healthcare services. WBAN applications can provide an affordable healthcare service with real-time monitoring [1]. A WBAN application can provide long-term health monitoring of a patient's physiological states including body temperature, blood pressure and heart rate without constraining their normal activities. These sensor-based applications can be used to monitor patients with different chronic diseases such as diabetes, hypertension, and cardiovascular disease [2]. In [3], the authors proposed a solar-powered sensor-based smartphone healthcare application to display data from

Y. Papadopoulos et al. (Eds.): IMBSA 2019, LNCS 11842, pp. 239–254, 2019.
https://doi.org/10.1007/978-3-030-32872-6_16

multiple sensor nodes. Sensors and smartphones can be combined with cloud computing to provide smart and affordable healthcare systems.

Cloud computing is a model which provides on-demand self-service for provisioning resources and rapid elasticity with minimal management effort and service provider interaction [4]. Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) are three types of service model available in cloud computing. Currently, Amazon web services (AWS), Microsoft Azure, Google and IBM are the leading cloud service providers. According to the Gartner magic quadrant 2018 report, AWS and Azure are recognised as leaders in IaaS.

In [5], the authors propose a remote healthcare application developed using a combination of Android apps and cloud computing to provide medical services for older adults. In a healthcare application, it is necessary to ensure minimal latency while exchanging information between sensor devices and servers. This minimal latency will increase the availability of patient health record for providing real-time healthcare service. In [6], the authors presented a cloud-based smart healthcare monitoring system using a docker container-based virtual environment to reduce latency and bandwidth.

As WBANs have limited memory, energy and computing power, a scalable high-performance computing and storage infrastructure is required to provide real-time data processing and storage. Serverless computing started a new era in the cloud computing industry, allowing minimum maintenance and providing cost-effective infrastructure for application development. Serverless computing is a cloud computing execution model where a cloud provider will run the server and dynamically manage the resource allocation. Serverless computing only charges for execution time, which helps in developing a cost-effective service. The goal of this paper is to present a serverless architecture for developing a dependable cloud-assisted WBAN application. By dependable we mean the application will be secure, available, scalable and ensure privacy.

The rest of the research paper is organised as follows; Sect. 2 briefly describes current trends in cloud computing in WBAN. Section 3 details the proposed serverless architecture, while Sect. 4 presents the implementation of the proposed architecture. Load testing results and attack mitigations for the proposed architecture are presented in Sect. 5. Finally, Sect. 6 concludes the paper by detailing future work.

2 Cloud Computing in WBAN

A fundamental issue in a WBAN healthcare application is the effective and efficient management of a large amount of data generated from sensor nodes. Cloud infrastructure can provide scalability of data storage, perform data analysis and give access to the user's health records [7]. In [8], the authors proposed a SaaS approach called BodyCloud. This SaaS approach supports the storage and management of sensor data streams for sensor-based healthcare applications. It also provides offline and online processing of stored data by using Google PaaS infrastructure, which allows for rapid prototyping of applications, easy customisation of architectural components, and scalability. In [9], the authors present cloudlet-based efficient WBAN healthcare applications which provide reliable large-scale sensor data to the end user. The

proposed prototype consists of a virtual machine which provides scalable data storage and processing infrastructure for large-scale WBAN systems. Sensor nodes used in a WBAN application can have different data transmission rates which require optimal resources for computing to avoid performance degradation or data loss. In [10], the authors proposed a cloud-based experimental framework named Cloud-WBAN, which will automatically adjust computing resources based on data volume and application type.

In [11], the authors proposed a green cloud-assisted WBAN based health monitoring service by adjusting the sleep time of sensor nodes for energy saving. The authors proposed to use the cloud-based MapReduce algorithm to analyse sensing frequency of decentralised data transmission between cloud and sensor nodes. In [12], the authors proposed a virtual hospital architecture by integrating WBAN and software-defined networking (SDN) in cloud computing to provide a better quality of service. As cloud computing provides scalability, elasticity and cost efficiency, the SDN will add further dimensions by providing adaptability and high bandwidth capability.

As sensor nodes of a WBAN application generate large amounts of data, cloud computing can provide a scalable storage option in addition to assisting with processing data in real-time. Cloud computing can also help with quick prototyping and deployment of the application. Furthermore, easy customisation of cloud infrastructure will help with feature enhancement of WBAN applications.

3 Proposed Serverless Architecture

Serverless computing is getting popular as a new and compelling paradigm for the development of cloud-based applications, largely due to the recent migration of enterprise applications to containers and microservices [13]. In the traditional cloud computing scenario, the healthcare application provider will pay a fixed and recurring cost, whether the application is used or not. In serverless computing, the user will only pay per-execution, not for the idle time. Serverless computing helps the developer to build a larger application using Function-as-a-Service (FaaS) platforms where each component of the application can scale separately. It also gives the flexibility to develop an application without thinking about managing infrastructure.

To develop system architecture, we first need to gather requirements and define the use cases. In this research paper, we choose a fitness tracking application designed by a mid-size enterprise, Company A, located in Ireland. This fitness tracking application consists of a wearable device which sends sensor data to a mobile application. This mobile application then transmits the data received from the sensors to a cloud-based backend application for further analysis. The user can access previously uploaded sensor data through the mobile application. Additionally, a user management process needs to be in place to manage sign in, sign up, and profile updates. This section presents an overview of the services required to develop the fitness tracking application, along with the cost structure of providing these required services from two leading cloud providers, that is AWS and Azure.

3.1 Domain Name System Service

Amazon Route 53 is a highly available, scalable and cost-effective Domain Name System (DNS) service for translating a domain name to an IP address. It can be used to manage user traffic globally through a variety of routing types, including latency-based routing or Geo DNS [14]. Additionally, it can also connect user requests to other AWS services such as Elastic load balancer, Amazon S3, CloudFront and API Gateway. The Azure DNS service also provides a similar service by using the Microsoft global network of name servers along with anycast networking. To provide high availability and faster performance, each DNS query is resolved by the closest available DNS server [15]. AWS Route 53 and Azure DNS have a similar monthly charge which is based on the number of hosted zones. AWS Route 53 ensures availability and traffic management using latency-based and geoproximity based routing protocols.

3.2 User Management and Authentication Service

Amazon Cognito is an authentication, authorisation and user management service for web and mobile applications. The user can sign up and sign in using their user name and password, without building and managing a backend solution or any infrastructure to handle identity management [16]. The Cognito service can save authentication information locally inside the device, which will allow applications to work offline. In the Azure cloud environment, Azure Active Directory (Azure AD) B2C is a business-to-consumer identity management service [17]. This service helps to customise and control how the user will communicate with the application. Azure AD B2C was developed using OpenID connect and OAuth2.0 protocols to provide security tokens and secure access to resources. This authentication and authorisation service will ensure privacy by preventing unauthorised access of personally identifiable information.

AWS Cognito charges are based on the number of monthly active users, while Azure AD B2C charges for each authentication. Both services have additional charges for enabling a multi-factor authentication service.

3.3 Content Delivery Service

A content delivery web service is used to deliver content to end users with low latency, high data transfer speeds, and no minimum usage commitments. When a user places a request for content, it will be automatically routed to the nearest edge location, so content is delivered with the best possible performance. Both cloud providers have content delivery services named AWS CloudFront and Azure CDN. Azure CDN serves the content from 30 point of presence (PoP) server locations worldwide [18], while AWS CloudFront serves content from 79 PoP server locations across 49 countries [19]. CloudFront supports dedicated custom SSL certificates and field level encryption.

AWS CloudFront provides content delivery from more PoP server locations compared to Azure CDN. Both service providers have a different pricing model based on the origin of the request, but in CloudFront there is no charge for the first 2,000,000 HTTP/HTTPS requests and 50 GB data transfer out per month for the first year.

3.4 Serverless Computing Service

Developing applications using serverless architectures requires event-driven or micro computing services to virtually run code for any application or backend service without the need to provision or manage servers. This service also needs to provide high scalability and availability with zero hardware and system administration.

AWS Lambda is an event-driven computing service which helps to build a serverless backend system to handle requests from the web and mobile applications using API Gateway [20]. The Lambda service helps to run and trigger code in parallel processes and scales with the size of the workload. By integrating Cognito services, Lambda can authenticate each request by using access tokens. Lambda supports several programming languages, including Java, Go, PowerShell, Node.js, C#, Python, and Ruby.

Azure Functions or Azure Service Fabric can be used to develop a serverless application using event-driven or micro computing services [21]. Azure Functions can directly integrate with mobile or web applications without attaching an application gateway. Azure Functions support C#, JavaScript, F# and Python in preview mode which is only available on request. Preview mode is excluded from the Microsoft service level agreement and might not be brought forward into general release.

Azure Functions provide different pricing models such as per execution, resource consumption and premium plan, whereas AWS Lambda only has a pay-as-you-go pricing model. However, AWS Lambda supports more programming languages than Azure Functions, which allows more flexibility during development of the application.

3.5 API Management Service

An API management service is required to publish APIs to integrate web or mobile applications with serverless backend services. The Amazon API Gateway and Azure API Management services are fully managed services which makes it easier for developers to create, publish, maintain, monitor, and secure RESTful application programming interfaces (APIs) at any scale and to expose backend and frontend HTTPs endpoints [22]. The Amazon API Gateway uses the Amazon CloudFront edge location service and can therefore provide lower latency responses when compared to Azure. The Azure API management service has three different pricing plans whereas the AWS API Gateway charges per request [23]. Additionally, the AWS API gateway supports multiple stages for API development, which provides better API lifecycle management when compared to Azure.

3.6 Database Service

In a serverless application, it is better to have a database with low latency that requires zero maintenance. Amazon DynamoDB is a fully managed fast and flexible cloud NoSQL database service. It is suitable for all applications which require single-digit millisecond latency at any scale [24]. This database supports both document and key-value data models. In DynamoDB, the user only needs to create a database table and set throughput. The rest of the database management tasks such as hardware or software

provisioning, autoscaling, and automatic partitioning will be handled by AWS. The Azure Cosmos DB is a fully managed, globally distributed, multi-model database service with high scalability and single-digit read-write latency with multiple NoSQL supports such as document, graph database and key-value data models [25].

In DynamoDB the user is charged per read and write request, whereas Cosmos DB charges for provisioned throughput and consumed storage by the hour. Furthermore, the databases are distinguished by their backup processes, as Cosmos DB provides automatic backup whereas it is a manual process with DynamoDB.

3.7 Web Application Firewall

Finally, a firewall service will be required to protect web and mobile applications from common web exploits which could affect application availability, compromise security, or consume excess resources. The AWS Web Application Firewall (WAF) provides control over which traffic to allow or block to the web application by defining customisable web security rules. WAF charges per rule [26]. By creating a custom rule, the WAF can block common attack patterns, such as distributed denial-of-service (DDoS) attack, SQL injection or cross-site scripting. The WAF can integrate with other services such as CloudFront, Elastic load balancer and the API gateway. A lambda function can be used to analyse the CloudFront access log and automatically update security rules in the WAF.

In the Azure cloud platform, the WAF service can be enabled as part of the Application Gateway [27]. This Application Gateway WAF service is based on the Core Rule Set 3.0 provided by the Open Web Application Security Project (OWASP). This WAF service does not provide any protection against DDoS attacks. To protect the application from DDoS attacks in the Azure cloud platform, a separate service named Azure DDoS Protection needs to be enabled. It comes with a fixed monthly charge, whereas AWS WAF charges are based on the number of rules created.

3.8 SSL/TLS Certificate

SSL/TLS certificates are used to secure communication between two entities in the system. AWS certificate manager (ACM) provides easy provisioning, management and deployment of public or private SSL/TLS certificates. ACM also provides easy certificate integration with other AWS services such as elastic load balancer, CloudFront and API Gateway. Azure only provides a public certificate for the Azure CDN and App services. Both service providers provide public certificates free of charge. There is an additional charge for private certificates.

3.9 Cost Comparison Between Azure and AWS

In this section, a cost comparison between the selected AWS and Azure services is presented. This comparison is based on different parameters such as the number of users, database size and read and write requests per second. During the cost calculation, a pricing calculator provided by the respective cloud providers for the Ireland region was used. As AWS and Azure use different pricing models, in some cases, an

adjustment will be required for the selected parameters. For example, the AWS API management charge is based on the number of requests per second, whereas the Azure API management has four tiers including developer, basic, standard and platinum. The basic tier was selected for Azure API management. AWS Cognito charges for the number of active users in a month, whereas Azure AD B2C charges for the number of authentication requests. Based on Company A's business goal to have 50,000 monthly active users with an average of five authentication requests per user, 50,000 monthly active users for AWS Cognito and 250,000 authentication requests for Azure are considered in the calculation. Additionally, one Web access control list (WEB ACL) and 15 custom rules for AWS WAF, 10 TB data transfer for content delivery and a database size of 50 GB was selected for the calculation. Table 1 outlines the cost for individual services of Azure and AWS.

Table 1. Azure and AWS monthly cost comparison

Service name	Azure	AWS
Domain name system service	$6.50	$6.50
User management and authentication service	$560.00	$0
Content delivery service	$828	$870
Serverless computing service	$96.80	$2.30
API management service	$250.62	$5.00
Database service	$70.90	$56.89
Web application firewall	$3456	$26.00
SSl/TLS certificate (public)	$0	$0
Support plan	$100	$100
Total	$5368.82	$1,066.69

During the cost analysis, we notice a large difference in the API management and WAF services. For API management, Azure requires the combination of Application gateway and API management services which results in higher costs compared to AWS API Gateway. AWS API Gateway charges are based on the number of requests, whereas Azure charges are based on the tier subscription and the number of instances. For WAF, Azure provide a package that secures web and infrastructure for a fixed monthly price. For AWS, the user needs to configure web security rules which cost $1 per rule. To secure the infrastructure with AWS, the user can rely on AWS with zero cost. For user management and authentication services, AWS Cognito charges are based on monthly active users and no charge will be required with free tier support, but Azure B2C will charge $560 for 250,000 authentications.

3.10 Summary of Comparison Between AWS and Azure

After reviewing the services from AWS and Azure, we notice some key differences in terms of cost and features. AWS will provide more availability in terms of content delivery due to having more PoP than Azure. AWS Lambda supports more

programming language options than Azure Functions service. A summary of the comparisons between AWS and Azure is presented in Table 2.

3.11 System Architecture

After reviewing the available features and cost comparison the AWS cloud platform was selected to develop the serverless architecture as it will provide larger programing language support, lower latency for content delivery, easy management of WAF, costs less and the developer was more familiar with AWS. To develop the fitness tracking application, the design of the core backend application system started by adding AWS Cognito. Lambda and DynamoDB were selected to process user requests and store data. To connect the backend application with mobile applications API gateway was deployed and attached with CloudFront to ensure wider availability. Additionally, the integration of Lambda functions allows for the analysis of CloudFront access logs Finally, Route 53 with an SSL certificate issued from the Certificate manager will be connected with CloudFront. The serverless architecture is illustrated in Fig. 1.

4 Implementation of the Proposed Architecture

This section describes the configuration process for the different AWS services contained within the proposed architecture.

4.1 Configuration of AWS Services

An AWS Cognito user pool is created to manage all user accounts and configured to handle end user sign in and sign up requests. The sign up process requires an email address and username, along with other attributes related to the application such as name, address, birthdate, gender and phone number. When a user successfully signs in, Cognito will provide a JWT token with a one-hour expiration time limit. Therefore, the mobile app will be configured to request a token refresh operation before the token expiration time. Each table in DynamoDB is created by assigning a name and primary key with partition and sort keys for better scalability and availability. To minimise the database cost, each table was provisioned with a capacity of five reads and writes per second. To ensure scalability, based on DynamoDB best practice guidelines, auto-scaling was configured with a target utilisation of 70%. Finally, encryption at rest is set up by assigning a key from the AWS Key Management Service (KMS).

The AWS Lambda platform supports several programming languages such as .NET, Go, Java, Python, Node.js and Ruby to create functions. During this implementation, all functions were developed using Node.js 8.10. Based on application benchmarking, functions to process and retrieve data were configured with 256 MB memory and 10 s timeout. The rest of the functions related to other use cases such as user profile creation, getting and updating endpoints and used a minimum of 128 MB memory with a 5 s timeout. Each function is designed to be invoked by requests coming from the API gateway. Additionally, a domain name is registered in AWS

Table 2. Comparison summary between AWS and Azure

Service name	AWS	Azure
DNS service	AWS Route53: • Latency-based and geoproximity based routing protocols • Pricing model: cost per hosted zone and number of requests	Azure DNS: • DNS query resolved by the closest available DNS server • Pricing model: cost per hosted zone and number of requests
User management and authentication	AWS Cognito: • Offline and online authentication support • Pricing model: charge based on monthly active users	Azure AD B2C: • Only support online authentication • Pricing model: charge per authentication
Content delivery service	AWS CloudFront: • 79 PoP server locations • Pricing model: charge based on the origin of the request and data transfer rate with free-tier support for first year	Azure CDN: • 30 PoP servers worldwide • Pricing model: charge based on the origin of the request and data transfer rate with no free-tier support
Serverless compute service	AWS Lambda: • More supported languages and all generally available for use • Pricing model: pay per execution and memory consumption	Azure function: • Less supported language with preview mode • Pricing model: pay per execution and memory consumption or premium plan
API management service	AWS API gateway: • Multiple API lifecycle stages support • Better response time and lower latency with CloudFront • Pricing model: pay per request	Azure API management: • No lifecycle stage support for API • Pricing model: three different pricing plans: developer, standard and premium
Database service	AWS DynamoDB: • Document and key-value data models • Manual backup • Pricing model: pay per read and write request	Azure Cosmos DB: • Document, graph database and key-value data models • Automatic backup • Pricing model: pay per provisioned throughput and consumed storage
Web application firewall	AWS WAF: • Customisable web security rules • Standalone service can be integrated with other AWS services • Implement DDoS protection by analysing CloudFront log • Pricing model: pay per web security rule	Azure WAF: • Web security rules not customizable and managed by the service provider • Only available with the Application gateway • For DDoS protection require Azure DDoS Protection service • Pricing model: a fixed monthly charge

(*continued*)

Table 2. (*continued*)

Service name	AWS	Azure
SSL/TLS Certificate	AWS ACM: • Central certificate management for other AWS services • Pricing model: no charge for the public certificate. Additional charge for a private certificate	• Certificates are managed separately for Azure CDN and App service • Pricing model: no charge for the public certificate. Additional charge for a private certificate

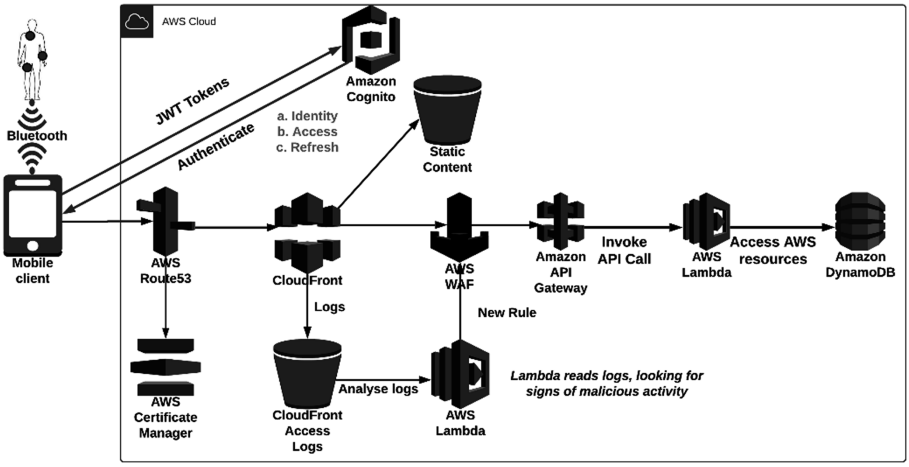

Fig. 1. Proposed serverless architecture for WBAN applications

Route 53 to route end-user requests using CloudFront. To enable HTTPS, a public certificate was assigned from the ACM.

4.2 Deploy RESTful API Using API Gateway

The API gateway exposes the AWS Lambda functions as a RESTful API. A new REST API is created by assigning a name with edge optimised endpoint option to serve from the end user's nearest location. As the WAF service is not fully integrated with the API gateway, the CloudFront access log will be used with the WAF service for intrusion detection. To fulfil each end user request, the following steps are necessary to create a RESTful API using the API gateway:

1. Create the API gateway resource with POST method and attach to associated Lambda function;
2. Configure the API gateway to use Cognito user pool as an authoriser to validate user requests using JWT tokens before invoking any Lambda function;
3. Deploy API gateway resources with a stage name called "Prod" and collect the URL;

4. Create a CloudFront web distribution with HTTPS;
   a. Add alternate domain name and respective SSL certificate from the ACM list;
   b. Create a root origin entry with default behaviours using the step 3 URL;
   c. Assign an S3 bucket to store the access log and create distribution;
5. Finally, configure Route 53 entry with respective CloudFront distribution.

This proposed architecture uses the OWASP top 10 recommendations for intrusion detection and prevention. An AWS CloudFormation template was used to deploy the WEB ACL, condition types and rules. Additionally, a lambda function was used to analyse the CloudFront access log to identify the source of DDoS attacks and automatically update the security rules in the WAF.

5 Performance Analysis of Proposed Architecture

Performance of the proposed architecture was evaluated by load testing and carrying out a vulnerability assessment. Load testing will evaluate the scalability and availability of the system. A vulnerability assessment will help to identify the weaknesses, potential areas of intrusion, and configuration issues in the system. It will also help to implement proper countermeasures for identified vulnerabilities to ensure the availability and security of the proposed architecture. To evaluate the system, load testing and vulnerability assessments were conducted in two phases: (1) In-house and (2) Penetration testing service provider (PTSP). Due to having limited resources for creating real-world scenarios for load testing and limited knowledge for conducting a vulnerability assessment in-house, we consulted with several PTSPs. A PTSP was selected based on budget and experience.

In the following sections, we first provide the load test results and then describe how the proposed architecture is affected by common web exploits such as distributed denial-of-service (DDoS) and SQL injection attacks.

5.1 Load Testing Results

A load test is used to evaluate how the application or REST API backend will perform with hundreds or thousands of concurrent users requests, and respective data volumes in a real-life scenario. Load testing was performed for two scenarios: (1) users will first download the mobile app and sign up for an account; (2) a user signs in to the mobile app and starts sending sensor data along with other profile metadata. Table 3 presents the list of REST API endpoints used during the load test.

Table 3. List of rest API endpoints for load testing

Scenario 1 (Sign Up)	Scenario 2 (Sensor data transmission)
• Cognito: SignUp endpoint • Cognito: InitiateAuth endpoint • API:/user/registration • API:/user/profile	• Cognito: Sign In • Cognito: InitiateAuth • API:/user/profile • API:/sensordata/upload • API:/sensordata/get • API:/user/profile/update

In-House Load Testing: It is recommended to use a modern, powerful and easy to use tool for load testing. A custom bash script with the help of AWS SDK (Command line version) was designed to test the sign up and sign in processes. Additionally, the ab benchmarking tool (Apache HTTP server benchmarking tool) was used to generate adequate traffic for testing API endpoints. During the test process, 100 sample users were created with randomly generated emails and passwords using a bash script. All users were successfully created in the AWS Cognito User pool. No exceptions or time-outs were noticed during this test. To assess the scalability and availability of the API, the ab benchmarking tool was used with ten concurrent users, each generating 200 API requests. The authentication tokens were used to verify each API request. During testing, 15% of the requests for one of the API endpoints timed-out due to throughput issues with the DynamoDB tables. Therefore, target utilisation was reduced to 60% for DynamoDB tables related to this endpoint. After the reconfiguration of DynamoDB the same test was run again and no timeout issues were noticed.

PTSP: To perform the load test, the PTSP used the Artillery tool with different combinations of arrival rates and durations. Artillery is a modern, powerful and easy-to-use distributed load testing toolkit. Distributed load testing will help to create real-world scenarios by generating traffic from different locations worldwide. The arrival rate is the number of incoming users per second. Generally, this is ramped up evenly from a start point to an endpoint throughout the test period. During the load test three rounds of tests were conducted with (1) arrival rate starting with 1 and ending with 5 for 300 s (henceforth known as Arrival rate A) (2) arrival rate starting with 5 and ending with 10 for 900 s (henceforth known as Arrival rate B) (3) arrival rate starting with 5 and ending with 10 for 1800 s (henceforth known as Arrival rate C). Table 4 illustrates the load test results for both scenarios for different arrival rates.

Table 4. Load test results for two scenarios by the PTSP

	Scenario 1 (Sign Up)	Scenario 2 (Sensor data transmission)
Arrival rate A: Start: 01 End: 05 Duration: 300 s	No timed-out requests Latency: <0.5 s	No timed-out requests Latency: <1.0 s
Arrival rate B: Start: 05 End: 10 Duration: 900 s	No timed-out requests Latency: <0.5 s	~10% requests timed-out Latency: >5.0 s (for ~10% requests) Test result after DynamoDB reconfiguration: No Timed-out requests Latency: <1.0 s
Arrival rate C: Start: 05 End: 10 Duration: 1800 s	No timed-out requests Latency: <0.5 s	~25% requests timed-out Latency: >20.0 s (for ~25% requests) Test result after DynamoDB reconfiguration: No timed-out requests Latency: <1.0 s

For scenario 1, a significant load was created with over 10,000 users signing up over 30 min. No issues were encountered with either timed-out requests (HTTP 504) or high latency. For scenario 2, no issues were encountered with either timed-out requests or high latency for Arrival rate A, however, for Arrival rate B, 10 percent of requests encountered a latency greater than 5 s and thus timed-out. For Arrival rate C more than 25% of requests encountered a latency greater than 20 s and thus timed-out. The key finding is that the DynamoDB takes a little time to scale, and the sudden high-traffic spikes caused the time-outs and throughput problems. To mitigate this issue, an adjustment was made in DynamoDB. Using the auto-scaling configuration feature, the minimum read and write capacity per second was increased to 10, and the target utilisation was reduced to 55% for Arrival rate B. For Arrival rate C the minimum read and write capacity per second was increased to 20 and the target utilisation was reduced to 45%. After making these configuration changes a similar test was run again for scenario 2 for both Arrival rates B and C, resulting in latency being reduced to <1.0 s and no requests timed-out.

5.2 Vulnerability Assessment

In-House Assessment: A denial-of-service scenario was created using the ab benchmarking tool which generated 400 requests from 30 concurrent users. Additionally, an IP address-based security rule was configured in the AWS WAF to prevent more than 100 requests per minute from an address. Results indicate that the lambda function automatically identified the IP address which generated more than 100 requests per minute. Finally, this lambda function also updated the source IP address in the WAF block list. The result shows that the proposed architecture assists to ensuring the availability of the system by preventing more than 100 requests from the same source over a short period.

Assessment by PTSP: The PTSP uses manual and automated methods to assess and perform vulnerability testing to attempt to gain access or compromise the service. The tools and methods used for exploitation during penetration testing are the same as those commonly used by people trying to compromise systems with malicious intent. Before testing begins, clear ground rules were established for stop points of the testing process, which will help to prevent unexpected damage to systems. For instance, when testing an API which contains an SQL injection flaw, it is enough to identify the compromise without attempting to obtain further access to the database servers. Network requests are relayed through several tools for manual and automated inspection, to allow listening and watching what the platform was doing. These data dumps are then taken into different tools and tested for any injection points and manual investigation. Table 5 presents the list of tools used during the vulnerability assessment process:
Additionally, manual and scripted testing was used to examine the results found during automated testing. Below are some of the major vulnerabilities found during the assessment process along with possible solutions.

Potential Denial of Service Points: During testing, there were several potential DDoS points found. These are requests that timeout within 10 s due to malformed data inside

Table 5. List of tools used for vulnerability assessments

Name	Description
OWASP ZAP	The open web application security project - Zed Attack Proxy (ZAP) is a penetration testing tool for finding vulnerabilities in applications
BURP SUITE	Burp Suite is a platform for performing security testing of applications
NMAP	Nmap (Network mapper) is a free and open source utility for network exploration or security auditing
SSLSCAN	SSLScan tests for different SSL exploits, such as heartbleed and the POODLE vulnerability, it also tests the cipher suites and key exchanges
HYDRA brute force	Hydra is a rapid dictionary attacker which can be configured against over 50 different protocols. It is most commonly used for brute forcing user accounts to test for weak passwords
KALI LINUX	Kali is a Debian-derived Linux distribution designed for digital forensics and penetration testing installed with hundreds of different tools

the payload. These can be run multiple times in multiple threads, driving up the usage and putting stress and strain on the service.

Solution: Action was taken in the API endpoints backend lambda code to handle potential malformed data gracefully by assessing each field from the payload. Additionally, a proper HTTP response was added to allows the user to retry a request later.

Security Misconfiguration – Stack Traces Enabled :During testing, it was discovered that stack traces were enabled for some API endpoints.

Solution: Stack traces were turned off in the lambda code base, and logging was copied to an encrypted AWS S3 bucket for future analysis from AWS CloudWatch.

After making the necessary changes in the lambda code and infrastructure to address the issues found during the assessment process, we informed the PTSP. A re-test of the updated system was unable to reproduce the vulnerabilities.

In summary, load testing and vulnerability assessments are required to evaluate system availability, scalability and security. In-house testing helped to identify issues and implement countermeasures in the early stages of the development lifecycle. The DynamoDB throughput bottleneck issue was identified by both in-house and PTSP. This issue required reconfiguration of the DynamoDB. Additionally, the PTSP identified other issues which required code changes in the Lambda functions.

6 Conclusion

Cloud computing is becoming a popular way to develop WBAN based healthcare applications which provide real-time monitoring. The recent introduction of serverless computing in the cloud paradigm helps developers to build more dependable applications which are highly scalable, available and cost-effective. In this paper, we presented a serverless architecture using AWS serverless computing to develop a dependable WBAN based healthcare application which is secure, highly scalable and

available. Serverless computing applications can be developed without thinking about the maintenance of the infrastructure. Furthermore, as the cost model for serverless computing is based on execution time, the cost of the core backend services will be minimised. We also performed load testing and vulnerability assessment by in-house and PTSP to test the security, scalability and availability of the proposed architecture. Load tests indicated some initial latency and time-out problems which were resolved by the reconfiguration of DynamoDB. Additionally, the mitigation of DDoS attacks using the WAF was tested to verify the availability of the application. Future work will involve extending the architecture by integrating AWS CloudTrail for privacy governance, AWS Kinesis Data Analytics and the AWS EMR service to perform big data analysis.

Acknowledgement. This work was supported with the financial support of the Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern and Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (www.lero.ie). Additionally, this work was partly funded by the DEIS H2020 project (Grant Agreement 732242).

References

1. Bouazizi, A., Zaibi, G., Samet, M., Kachouri, A.: Wireless body area network for e-health applications: overview. In: International Conference on Smart, Monitored and Control Cities (2017)
2. Taha, M.S., Rahim, M.S.M., Hashim, M.M., Johi, F.A.: Wireless body area network revisited. Int. J. Eng. Technol. **7**, 3494–3504 (2018)
3. Shaji, J.E., Varghese, B., Varghese, R.: A health care monitoring system with wireless body area network using IoT. Int. J. Recent Trends Eng. Res. **3**, 112–117 (2017)
4. Mell, P., Grance, T.: The NIST definition of cloud computing (2011)
5. Luarasi, T., Durresi, M., Durresi, A.: Healthcare based on cloud computing. In: Proceedings - 16th International Conference on Network-Based Information Systems. NBiS 2013, pp. 113–118 (2013)
6. Kavita, J., Srichandan, S., Ashok, K.T., Sahoo, L.B., Bhabendu, K.M., Debasish, J.: An IoT-cloud based smart healthcare monitoring system using container based virtual environment in Edge device. In: ICETIETR, pp. 1–7 (2018)
7. Fortino, G., Di Fatta, G., Pathan, M., Vasilakos, A.V.: Cloud-assisted body area networks: state-of-the-art and future challenges. Wireless Netw. **20**, 1925–1938 (2014). https://doi.org/10.1007/s11276-014-0714-1
8. Fortino, G., Parisi, D., Pirrone, V., Di Fatta, G.: BodyCloud: a SaaS approach for community body sensor networks. Future Gener. Comput. Syst. **35**, 62–79 (2014)
9. Quwaider, M., Jararweh, Y.: Cloudlet-based efficient data collection in wireless body area networks. Simul. Model. Pract. Theory **50**, 57–71 (2015)
10. Bhardwaj, T., Sharma, S.C.: Cloud-WBAN: an experimental framework for cloud-enabled wireless body area network with efficient virtual resource utilization. Sustain. Comput. Inform. Syst. **20**, 14–33 (2018)
11. Chiang, H.P., Lai, C.F., Huang, Y.M.: A green cloud-assisted health monitoring service on wireless body area networks. Inform. Sci. (Ny) **284**, 118–129 (2014)

12. Al Shayokh, M., Kim, J.W., Shin, S.Y.: Cloud based software defined wireless body area networks architecture for virtual hospital. In: 10th EAI International Conference on Body Area Networks, pp. 4–7 (2015)
13. Baldini, I., et al.: Serverless computing: current trends and open problems. In: Chaudhary, S., Somani, G., Buyya, R. (eds.) Research Advances in Cloud Computing, pp. 1–20. Springer, Singapore (2017). https://doi.org/10.1007/978-981-10-5026-8_1
14. AWS: AWS Route53. https://aws.amazon.com/route53/
15. Azure: Azure DNS. https://azure.microsoft.com/en-in/services/dns/
16. AWS: AWS Cognito. https://aws.amazon.com/cognito/
17. Azure: AD B2C. https://azure.microsoft.com/en-us/services/active-directory-b2c/
18. Azure: Azure CDN. https://azure.microsoft.com/en-gb/services/cdn/
19. AWS: AWS CloudFront. https://aws.amazon.com/cloudfront/
20. AWS: AWS Lambda – Serverless Compute. https://aws.amazon.com/lambda/
21. Azure: Azure Functions. https://azure.microsoft.com/en-gb/services/functions/
22. AWS: Amazon API Gateway. https://aws.amazon.com/api-gateway/
23. Azure: API Management. https://azure.microsoft.com/en-us/services/api-management/
24. AWS: Amazon DynamoDB. https://aws.amazon.com/dynamodb/
25. Azure: Azure Cosmos DB. https://azure.microsoft.com/en-us/services/cosmos-db/
26. AWS: AWS WAF - Web Application Firewall. https://aws.amazon.com/waf/
27. Azure: WAF. https://docs.microsoft.com/azure/application-gateway/waf-overview/