National College of Ireland

Architecting a High-Availability and Secure Three-Tier Web Infrastructure Using AWS Services

MSc Research Project
Masters in Cloud Computing

Guru chandra Arivukkarasu Mahalakshmi
23307323

School of Computing
National College of Ireland

Supervisor: Shaguna Gupta

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Guru Chandra Arivukkarasu Mahalakshmi
Student ID:	23307323
Programme:	Msc cloud computing Year:2024-2025
Module:	Msc research project
Supervisor:	Shaguna Gupta
Submission Due Date:	15-09-2025
Project Title:	Architecting a High-Availability and Secure Three-Tier Web Infrastructure Using AWS Services
Word Count:	8998

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: GuruChandra Arivukkarasu Mahalakshmi

Date: 15-9-2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Architecting a High-Availability and Secure Three-Tier Web Infrastructure Using AWS Services

Guru Chandra Arivukkarasu Mahalakshmi

23307323

ABSTRACT

Due to the rising need towards highly scalable, highly available and secure web applications, there has been the embrace of cloud-native architectures enabled by managed services. The study addresses the design and deployment of a three-tier web application architecture that is secure, fault-tolerant, and resource friendly on the Amazon Web Services (AWS). The suggested system contains Route 53 to implement DNS and domain routing, an Application Load Balancer (ALB) to manage the intelligent distribution of traffic, Amazon EC2 to house Apache Tomcat application servers, Amazon RDS that implements a relational database management, and Amazon S3 that is involved in storing a static content. RabbitMQ is used to support asynchronous messaging and Memcached to support caching. The architecture exists in an isolated Virtual Private Cloud (VPC) in separated public and isolated private subnets with sound security practices such as AWS Certificate Manager (ACM) powering the SSL/TLS encryption, finer security groups and I AM setups. Auto Scaling policies are used to scale computing resources up and down depending on workload parameters and can also be cost efficient and performance-wise. Load testing, failover simulations and security testing were applied to the evaluation process to verify that the system can support a variety of traffic loads, the ability to recover quickly in case of a failure and minimize a strict access control. The findings show that cloud-native services in the AWS environment are well applicable in the bundling of services in order to provide an enterprise grade application platform, which covers the operational need of modern web-based systems

1. INTRODUCTION

Cloud computing has transformed the way applications are deployed by offering on-demand, large-scale and cost effective computing resource provisioning via internet such that the organisations do not have to maintain any physical infrastructure The Amazon Web Services (AWS) being one of the primary cloud providers provides a wide spectrum of managed services on compute, storage, networking, and security, which allow building architectures that address the needs of modern architecture of high availability, security, and performance. In the modern digital world, web apps have to manage with varying workloads, ensure that they can stay up in the events of systems failures and ensure data integrity with strong security procedures, which the traditional hosting has failed to deal with The study concentrates on planning and building a safe, fault tolerant, scalable three tier web application on the AWS system that separates the presentation, the application and the database layer to achieve better control and performance. It uses route 53 routing DNS, application load balancer (ALB) to distribute traffic, Amazon EC2 instances running Apache Tomcat, Amazon relational database (Amazon RDS) to manage the relational databases and Amazon simple store (Amazon S3) to store the static files and Memcached and RabbitMQ to increase the performance of caching and messaging. Using AWS managed services and earlier automation, the system will provide the enterprise level platform with a high traffic carrying capacity, quick to recover after failures, and provide secure operations thus proving the efficiency of cloud-native strategies in satisfying the operational needs of contemporary web applications.

1.2 Background

Cloud computing has revolutionized the development, deployment and management of applications due to the ability to access the computing resources on-demand without needing ownership or having to maintain physical infrastructure (Armbrust et al., 2010). This model will enable organisations to scale resources, cost optimisation, and better reliability of services. Because one of the leading cloud service providers is known as Amazon Web Services (AWS), it has an expansive range of managed services that concern compute, storage, networking, security, and application integration. Many different deployment models have been proposed and among them, one which has stood out is the three tier architecture as one which has resulted in a successful design of robust and maintainable systems through separating the presentation layer, the application layer and the database layer. It enhances scalability, fault isolation, and maintenance since it can be used to separate concerns effectively and this aspect is more applicable to contemporary web applications that have to support varying workloads and strict uptime demands. Using AWS-managed services like Route 53, Application Load Balancer (ALB), Amazon EC2, Amazon RDS, and Amazon S3 one can achieve architectures that are not only cost-effective but can also achieve the services expected to enterprise-level, such as high performance, security, and availability.

1.3 problem statement

The problem is that modern web-based applications require high availability, scalability, and security because of the number of online users and the sophistication of online services. Minor lapses of time or degradation of performance may lead to the huge loss of revenues, decrease in user trust, and the long-term damage to reputation (Patel et al., 2020). Because they are dominated by fixed computing capacity and face high levels of manual scaling requirements as well as complex disaster recovery, traditional on-premises infrastructures are frequently not able to handle such demands. In addition, these systems do not automatically include fault tolerance, resourceful load balancing, and effective security features mechanisms; hence they are susceptible to performance faults and internet attacks. Consequently, dynamic provisioning of resources on-demand, quick to restart even in the event of failure, and the question of security regarding data transmission and storage are highly demanded with cloud-based solutions. The offered research tackles this issue by developing and implementing a three-tier architecture running on the Amazon Web Services (AWS) cloud wherein managed services are used to enable the scalability, fault tolerance, and robust security with cost efficiency still in effect.

1.4 Motivation

The rise of web-based services in areas like e-commerce, banking, healthcare, and education has increased the importance of system availability, performance, and security to be major success factors of organisations. Users want the application to be fast loading and capable of working in different traffic situations reliably, and whenever they use the application, the data is to be safe. Any kind of malfunction in meeting these expectations may lead to loss of customers, a decline in competitiveness and reputational damage. The solution can be achieved by using cloud computing, especially platforms such as Amazon Web Services (AWS): high availability and high security made this scalable method of computing managed to offer a great solution, which is addressing elastic scalability in the platforms like the AWS. The motive behind this project is the prospect of using such capabilities to design and implement a threetier architecture that does not just satisfy the needs of operation but also employs the optimal practices in terms of fault tolerance, cost optimisation and resource efficiency. Through the incorporation of AWS services, i.e., Route 53, Application Load Balancer, EC2, RDS, S3, Memcached, and RabbitMQ, it is intended that the challenges experienced by modern applications can be overcome in a pragmatic or real-life application through the resultant demonstrable improvements in terms of performance, resiliency, and security.

1.5 Research Aim

This study seeks to design, deploy and test a secure scalable and fault-tolerant three-tier web application architecture on Amazon Web Services (AWS) managed services. The project is aimed at illustrating how cloud-native services, i.e. Route 53, Application Load Balancer, EC2, RDS, S3, Memcached, and RabbitMQ can be combined to provide high availability, optimised performance and robust security in a Virtual Private Cloud (VPC). This can be achieved by the addition of auto scaling, load balancing and secure network settings so that the research can highlight a structure that can adapt dynamically on workload growth and contraction, efficiently manage resources, maintain a service continuity even after some service failures. The work also seeks to establish the viability of architecture by performance testing, failover simulation, and security, thereby giving a guideline on how to deploy enterprise level application in the cloud.

1.6 objective

The major goal of this study is to achieve the design, execution and testing of a three-tier web application with high availability, scale and security through the Amazon Web Services (AWS). In particular, the study will endeavour to divide the presentation, application, and database layers of a Virtual Private Cloud (VPC) with the use of Route 53 managed services where the DNS routing service will be deployed innovatively, an Application Load Balancer to distribute traffic intelligently, Amazon EC2 instances to host Apache Tomcat, Amazon RDS to manage the relational database, and Amazon S3 to store static files. Memcached and RabbitMQ are used to facilitate asynchronous communications and to improve performance of the architecture. Health checks and auto scaling policies are set to provision dynamically the compute resources in reaction to the changing workloads, which makes them cost-effective and fault tolerant. Also, there is the presence of strong security controls such as security groups, IAM roles, and SSL/TLS encryptions through AWS Certificate Manager. The performance of the architecture is proven by testing, and simulations of failover, and security. The architecture already gives a paradigm of best practices of the application cloud deployment at an enterprise level.

1.7 Research question

The fundamental question answered in this research is as follows:

A) How can the design and implementation of a cloud-based three-tier architecture on Amazon Web Services (AWS) be made to achieve high availability, scalability and security yet provide cost reduction? Out of this arises a number of sub-questions, some of which include:

- What AWS-managed services would work best at each tier of the architecture to maximise both performance and resiliency? How load balancing, Auto Scaling, and Health Checks can be prepared to adjust to different workloads ensuring disruption to service.

- How much caching and messaging can be incorporated to get faster response and efficiency in the system? How is it possible to implement appropriate controls to safeguard data and application integrity with security controls such as encryption, network segmentation and identity management?

- What methods of testing can be used to assess capacities of the architecture to cope with spikes in traffic, failure of components, and security attacks?

2. LITERATURE REVIEW Literature review table

Author/year	**Main focus**	**Tools used**	**findings**	**relevance**
Armbrust et al. (2010)	Definition of cloud computing	NIST model, elasticity concept	Standardised cloud principles	Baseline definition for cloud-native deployment
Mell & Grance (2011)	Cloud service models	NIST cloud framework	Defined characteristics (on-demand, broad access)	Provides conceptual foundation
Patel et al. (2020)	Fault tolerance in cloud	Multi-AZ, redundancy	Improved resilience with failover	Supports use of multi-AZ RDS & load balancer
Hashizume et al. (2013)	Security challenges	IAM, encryption, segmentation	Identified common threats & solutions	Justifies IAM, ACM, SG practices
Sharma et al. (2021)	Scalability in AWS	Auto Scaling Groups, ALB	Elastic scaling improves performance	Directly supports scalability testing
Hassan (2021)	Relational vs NoSQL DB	ACIT study	Compared DB models	Supports DB choice in 3tier

Ramanathan et al. (2011)	Cloud databases	SimpleDB vs Bigtable	Highlighted data models	Relevant to DB layer discussion
Wei et al. (2012)	Cloud transactions	CloudTPS framework	Showed scalable transactions	Supports backend reliability research
Babuji et al. (2016)	Secure analytics	Cloud Kotta on AWS	Enabled scalable secure processing	Reinforces security-first design principle
Hayat et al. (2024)	Multi-tenancy security	Cloud infrastructure study	Suggested isolation methods	Supports strict VPC & SG design

Cloud computing has emerged as one of the core facilitators of current modern web applications deployment, which has provided scalability, at-will and associated with the computing resources needed to relieve far-reaching amount of operational overhead related with the utilization of traditional IT infrastructure (Mell and Grance, 2011). The National Institute of Standards and Technology (NIST) introduced the concept of cloud computing referring to it as a computing model to provide ubiquitous, convenient and on-demand network access to a shared pool of computing resources. Amazon Web Services (AWS) heads the market among the vast majority of cloud service providers who can cover a wide range of cloud computing requirements, including compute, storage, networking, security-related, as well as application integration requirements and requirements (Amazon Web Services, 2023).

2.1 scalability

The requirement of modern applications is scalability since the demand of the users is unpredictable. AWS services that you can use to scale horizontally are Auto Scaling Groups (ASG) and Application Load Balancer (ALB) to add or remove compute instances, dynamically based on parameters of workload (Amazon Web Services, 2023). Sharma et al. (2021) further claim that that elasticity only maintains levels of consistent performance and even lowers the operational costs because of not over-provisioning.

2.2 Fault tolerance

Another critical point is fault tolerance because business can be crippled by system unavailability. Failures are quickly recovered by using high availability settings, including resources in more than one Availability Zone (AZ), using health checks in load balancers and applying auto-scaling (Patel et al., 2020). On the same note, services such as caching (Memcached) and asynchronous messaging (RabbitMQ) help resilience as they unload workloads and do not cause a bottleneck in the primary application layer.

2.3 Security

Cybersecurity in the cloud has advanced to meet newer threats and services found in the AWS include Identity and Access Management (IAM), AWS Certificate Manager (ACM) that provides SSL/TLS encryption and security groups with additional fine-grained control. Hashizume et al. (2013) stipulate that encryption, access control, and network segmentation put together play a vital role in ensuring the safety of data and application integrity. Altogether, the literature recommends that managed AWS services can form a part of a three-tier architecture capable of providing a secure highly available and scaled environment to enterprise-grade applications. Nevertheless, theoretical frameworks and service capabilities have been well-documented, but a missing link of practice consists of the integration of such services to a single unified tested and cost-optimised deployment model, which is the one that this study aims to fill.

3. METHODOLOGY

The methodology carries out are systematic, design, development, and testing at three levels, secure, scalable and fault-tolerant web application on Amazon Web Services (AWS). The methodology starts by creating the mapping of presentation, application and database layer with appropriate AWS-managed services to achieve maximum performance, security, and costeffective implementation. The presentation layer employs Amazon Route 53 to handle DNS and an Application Load Balancer (ALB) to spread the inbound traffic, over multiple Availability Zones (AZs). Business logic resides in the application tier running on Amazon EC2 instances with Apache Tomcat behind the Auto Scaling Groups (ASGs) which allow it to scale with respect to changing workloads. Amazon RDS is used to store persistent data and to cache using Memcached; a backend services layer includes a database tier and a backend service tier, including RabbitMQ to encode and decode asynchronous messages. All of this is hosted in a Virtual Private Cloud (VPC) within a high-level restricted security group scheme encrypted (SSL/TLS) with AWS Certificate Manager (ACM) with the more sensitive workloads in a more restricted internal subnet. The last phase would involve intensive performance related testing, failure over simulation and security tests to prove the adequacy of the architecture to achieve the research goals.

3.1 Presentation Tier

The user would be involved in the application on the presentation level where the sending requests to the application is done securely and effectively. Here, this layer uses Amazon Route 53 as the domain name resolution with the operation of mapping the application URL to the public endpoint of the Application Load Balancer (ALB). ALB is correspondingly established in multiple subnets at diverse Availability Zones (AZ) to be fault tolerant as well as highly available. It also runs health checks on the application tier to make sure that incoming traffic is only sent to the healthy instances of the Amazon EC2 and as such limiting periods of downtime and enhancing user experience. AWS Certificate Manager (ACM) is incorporated allowing the use of SSL/TLS encryption which does not allow any type of communication between the user and the load balances to happen without the presence of HTTPS. This design separates the presentation tier of the tier architecture of the application and the database of the application and isolates it in the public subnets further reducing the attack surface and having efficient, secure, and reliable access to the application as an end-user.

3.2 Application tier

The application layer performs the computation providing the logic of business, receiving the request of a client, and communicating with the backend services. Under this architecture, it would be used in isolated subnets to restrict the direct access to internet therefore improving security. The code of this layer is an Auto Scaling Group (ASG) of Amazon EC2 instances having the Apache Tomcat application server installed. ASG automatically scales the number of running instances relative to scaling policies that the user can predetermine such as CPU utilisation, request numbers and thresholds ensures that the system is never over-provisioned and able to de-provision during the periods of low workload. The other cloud services include an EC2 instance communicating securely with Memcached (which is used to cache common information), and RabbitMQ (which is used to process asynchronous messages in a distributed way) both of which enhance performance by balancing the burden of the main application logic. They keep their static files e.g. images, scripts, and software artefacts, on Amazon S3 which helps to offload content delivery tasks to the compute instances and enhances performance. Allowing only inbound traffic in cases of presentation tier and only outbound traffic to the database and backend service tier, security groups are created to have controlled and secure data path.

3.3 Data Base and Back-end Service Tier

The tier featuring the database and backend services is the basis of persistent data storage, caching and asynchronous messages processing in the architecture. It is integrated with full usage in the private subnets in order to exclude direct access to the internet and have weak connection to the outer threats. The main database is deployed to Amazon Relational Database Service (RDS) Multi-AZ configuration that offers high availability and circumvents an outage with automatic failover. Data security and hence disaster recovery are supported by encryption

of data at rest and in transit, automated backs and snapshots. There is the integration of Memcached to enable fast storage of data that is frequently queried in order to get a faster query response time among other benefits like reducing the pressure on the database. RabbitMQ is an asynchronous message broker to support the communication between components of the application, as well as increasing its scalability and decoupling services to improve fault isolation. The security groups to this tier are set to permit no incoming connections other than through the application tier, although it cannot connect to the internet directly, and can only make outgoing connections to critical services. This arrangement makes this data storage and message handling among others sensitive operations secured, resilient and high performing.

3.4 Testing and evaluation

The test and evaluation phase will aim at stimulating the scalability, fault tolerance, performance and security of the architecture before the actual production setup. Performance testing is done by creating simulated workload using the tool, Apache JMeter or AWS CloudWatch metrics to measure response time, throughput, and error rate at different workloads. Scalability tests consider the performance of Auto Scaling Groups (ASGs) by the progressive scaling of the traffic and observing the expediency of adding or removing the EC2 instances according to the pre-defining scaling policy. Fault Tolerance Testing is a testing of EC2, simulation of outages of Availability Zones and testing database failover to test the speed of EC2 instance recovery and service continuity. Security validation performs validation of the correct application of security group rules, SSL/TLS certificate configuration through the AWS Certificate Manager (ACM) and denial of access to the private subnets. The CloudTrail and the CloudWatch are used in logging and monitoring so as to be compliant in best security practices and operational transparency. The outcomes of these tests offer the actual evidence of the given architecture in terms of satisfying the research purposes which means that their functioning in the production framework will give the minimum downtime and the use of resources, at the same time.

4. ARCHITECTURE OF THE SYSTEM

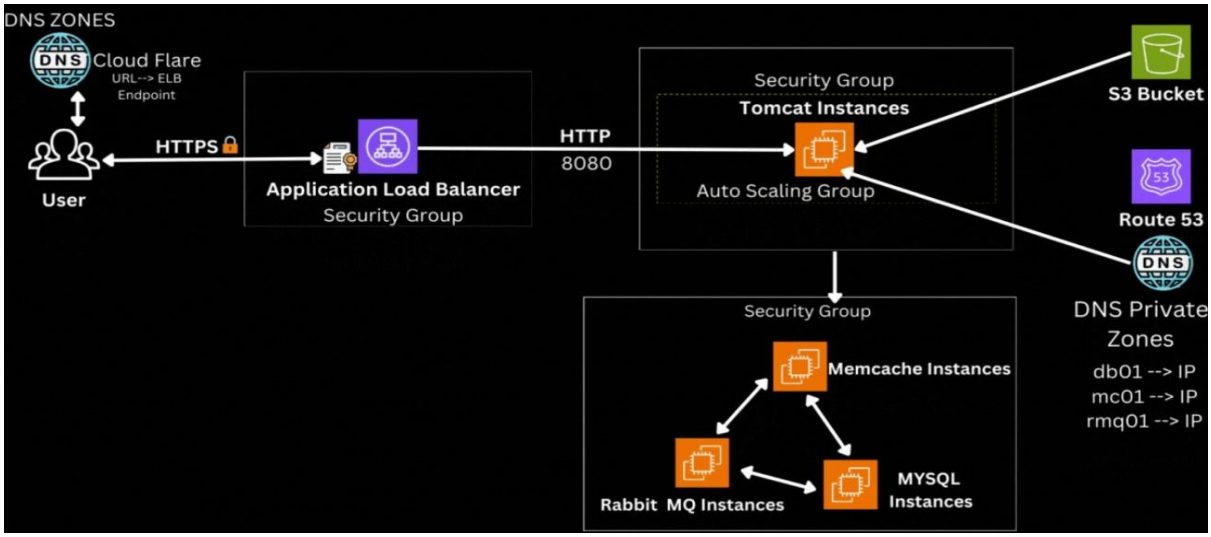

Fig1.Architectural diagram

System architecture aims to give secure, scalable and fault tolerant multi-tier application deployment in AWS. Entry point resolution of Domain Name System (DNS) is realized through AWS route 53 nodes whereby java application may be efficiently submitted to the Application Load Balancer (ALB). The ALB terminates HTTPS connections to have secure communications with the end users and routes requests internally to the backend application layer using HTTP (port 8080). Allowing the number of active servers to be dynamically scaled based on workload demand is the application layer, a collection of multiple Tomcat instances launched in an Auto Scaling Group. These cases would be held in a special security group only allowing requested ports to access the network and thus increasing the level of security. The backend data and messaging services get split into three fundamental components in the form of MySQL instances to store data in a normalized way and perform transactional requests, memo cache instances to provide faster data access and minimize latency and lastly RabbitMQ to provide message queuing and synchronization between services. All backend components exist in a private subnet and employ internal DNS names (db01, mc01, rmq01) in a private hosted zone and are not exposed to the general internet. The images, stylesheets, and scripts are all stored in Amazon S3 and cached and served efficiently by the Content Delivery Network (CDN) Amazon CloudFront with regional edge locations all over the world. The architecture is maintained further with the application of security groups and Network Access Control Lists (NACLs) that can control incoming and outgoing traffic at instance-level and subnet-level. Such modularity and layered design enable scaling with ease, fault tolerance, robust security measures and performance optimisation after the best practices in cloud-native architecture.

4.1 Novelty of the research

The project is an extension of the typical AWS three tiers deployment providing a securityfocused, resource-conscious design incorporating caching and asynchronous messaging. A new integrated security group map was used to establish a strict tier to tier boundary of communication and support dynamic auto scaling to ease configuration complexity and enhance security. The architecture also uses Memcached as an in-memory caching and RabbitMQ as an asynchronous message-processing solution that are used to decrease the load on the database, lower the latency, and increase resilience in the face of heavy traffic. In comparison with traditional three-tier AWS patterns, which only consider EC2, RDS, and load balancing, this architecture has integrated cost-aware scaling policies in the private subnets, resulting in high-performance and operational cost control. The research offers a viable, enterprise-grade architecture by supporting fault tolerance, caching, messaging, and secure isolation in one verified model, offering a superior trade-off between scalability, high availability, and robust security in comparison to the default AWS implementations.

5. IMPLEMENTATION

The implementation step concerns how the proposed three-tier architecture will be applied practically in Amazon Web Services (AWS) on the services and configurations proposed in the methodology. It is carried out by enabling and setting up network infrastructure in a Virtual Private Cloud (VPC), applying computer and storage infrastructure, application security, and installing application components. The architecture is created in stages whereby the network resources involving the setup of both the public and private subnets, Internet Gateways and route tables are created followed by the stacking of the presentation layer, the application layer, and the database layer. Security groups are also formed to establish rules associated with inbound and outbound traffic and key pairs are generated to provide safe access via SSH to EC2 instances. AWS Certificate Manager (ACM) gives provisions for HTTPS by way of SSL/TLS certificates. Application is hosted on EC2, over an application load balancer (ALB), managed by Auto Scaling Groups (ASGs) to be fault-tolerant and elastic. The deployment of backend services, namely Amazon RDS, Memcached, and RabbitMQ, takes place in between privately located subnets so that data were transferred efficiently and securely. During the deployment, Amazon CloudWatch and AWS CloudTrail are activated to have monitoring and logging throughout the meeting of performance and ensuring visibility of operations.

5.1 Security group setup

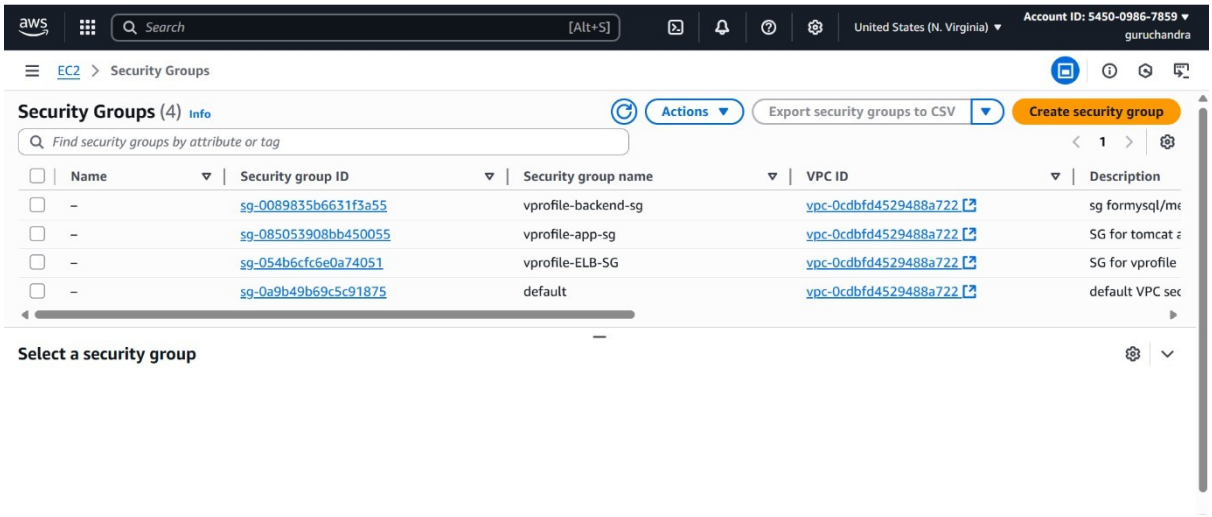

Fig 2. Security groups

There was the establishment of security groups to manage both inbound and outbound traffic to various parts of the architecture to provide layered security. There are three security groups dedicated to security: vprofile-backend-sg, vprofile-app-sg and vprofile-ELB-SG as demonstrated in figure X. The security group vprofile-backend-sg was attached to backend services including MySQL, Memcached, and RabbitMQ among others with the ability to receive the passing traffic of the application tier. Amazon EC2 instances, which ran the Apache Tomcat server, were assigned an Amazon EC2 security group called vprofile-app-sg, and any traffic that originated with the load balancer had an inbound access limited to traffic on the 8080 port. The Application Load Balancer was linked to the vprofil-ELB-SG security group that allowed incoming traffic via HTTPS: inbound internet traffic was passed by the security group to the application tier. Such isolation of security groups improves the security levels adopting the principle of least privilege making each component only able to communicate with the levels they are supposed to and thereby minimizing the possible vectors to attacks.

5.2 key pair configuration

A key pair was generated to give safe off-site access to the Amazon EC2 Amazon EC2 instances with the name vprofile-prod-key. Key pairs in AWS are composed of a private key file (.pem) which is securely downloaded and stored by the administrator and a public key copy of which is kept in the AWS environment. SSH authentication involves the use of the private key to access Linux EC2 instances where only authorised users can gain access to the servers. The advantage of this security control involves the incapability of unauthorised logon since cryptographic authentication substitutes password authentication. The vprofile-prod-key is linked to the production resources only so that there is a steady border with security. Management of the .pem file is very important since loss would imply creation of a new key pair to replace it and reconfiguration of affected instances.

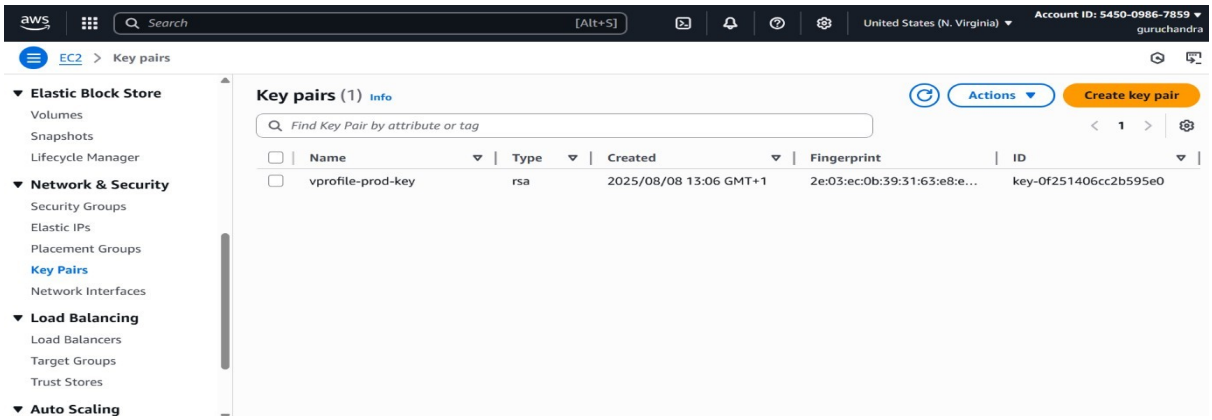

Fig 3. Key Pair generation

5.3 EC2 INSTANCE DEPLOYMENT

The deployment part consisted of launching several Amazon EC2 instances each assigned a role in the multi-tier architecture application. The app tier is backed by two instances vprofileapp and vprofile-app01, Apaches Tomcat configured to serve HTTP requests (sent by the load balancer) and provide High availability across load distribution. The database layer on vprofiledb01 operates MySQL to save and manage data and only the security group can access the data in the database layer without exposure to the public. The caching layer has an instance, vprofilemc01, that runs Memcached to put frequently accessed data in memory rather than the database to improve response times and decrease database load. Each of them employs the t3.micro type to operate cost-effectively, yet to satisfy performance demands, and all are assembled into specific security groups as per their functions in order to regulate incoming and outgoing traffic. Each of these deployments has passed AWS 3/3 status checks indicating each is ready, connected to the network and can scale, isolate faults and service commonly easier to support the architecture with the modular application to deployment.

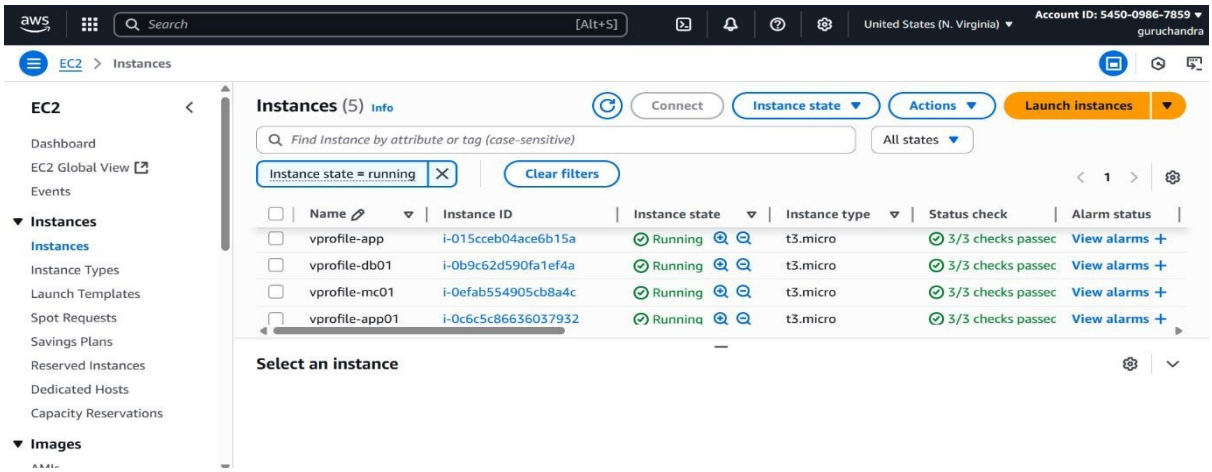

Fig 4. Ec2 instance

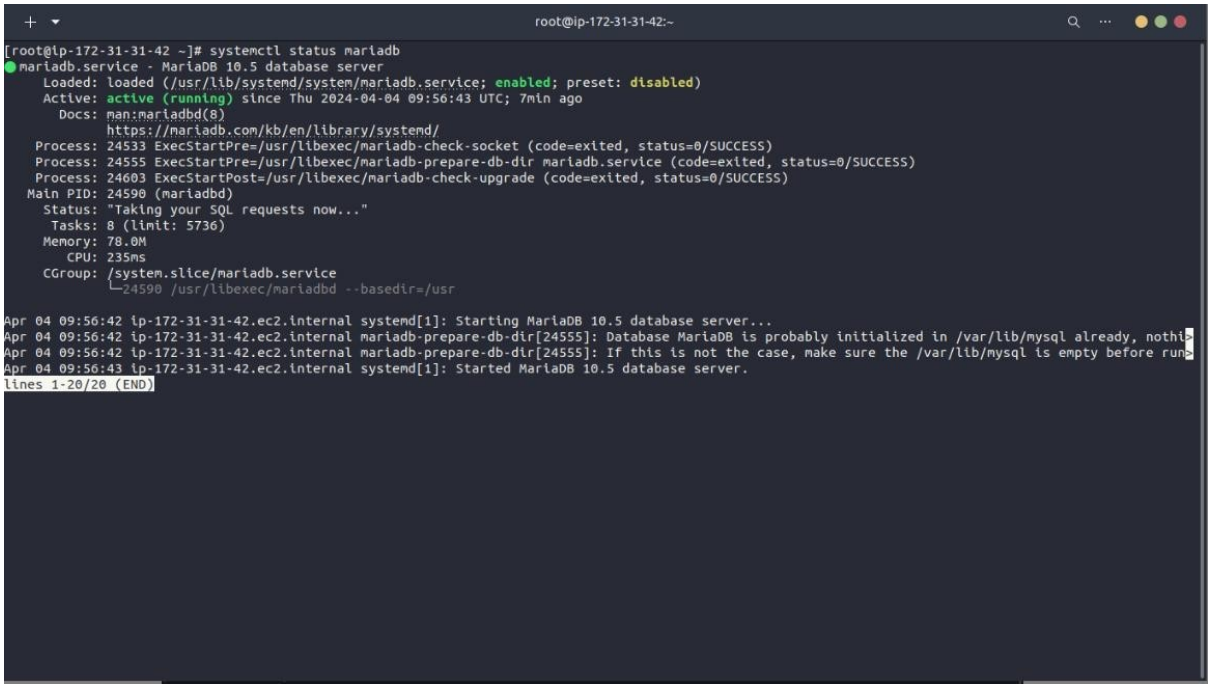

Fig5. Database setup

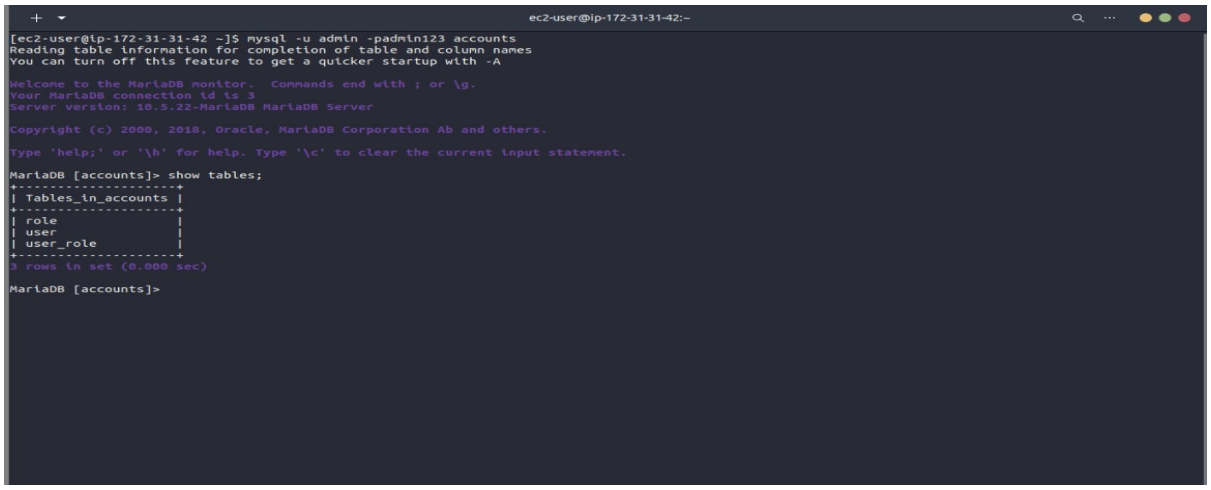

Fig6.DB table display

The implementation stage involved back-end server deployment and configuration of the database layer with MySQL on a separated Amazon EC2 server on a security group attributed to backend to be isolated of having a public interface. Having installed the MariaDB server and configured it to be always up, the MariaDB server activity was checked using systemctl status MariaDB command and it was confirmed that the database engine was up, and it was free of error. It was then fully connected to the database through the MySQL client with an administration level access and SQL queries to access the accounts database schema that confirmed the existence of the crucial tables like role, user, and user role to perform authentication and access levels. This configuration implied reliable data storage and retrieval and gave a high degree of access control via AWS security group rules which enable inbound connection only with the application layer therefore resonating with cloud security best practices.

5.4 Cloud front and WAF setup

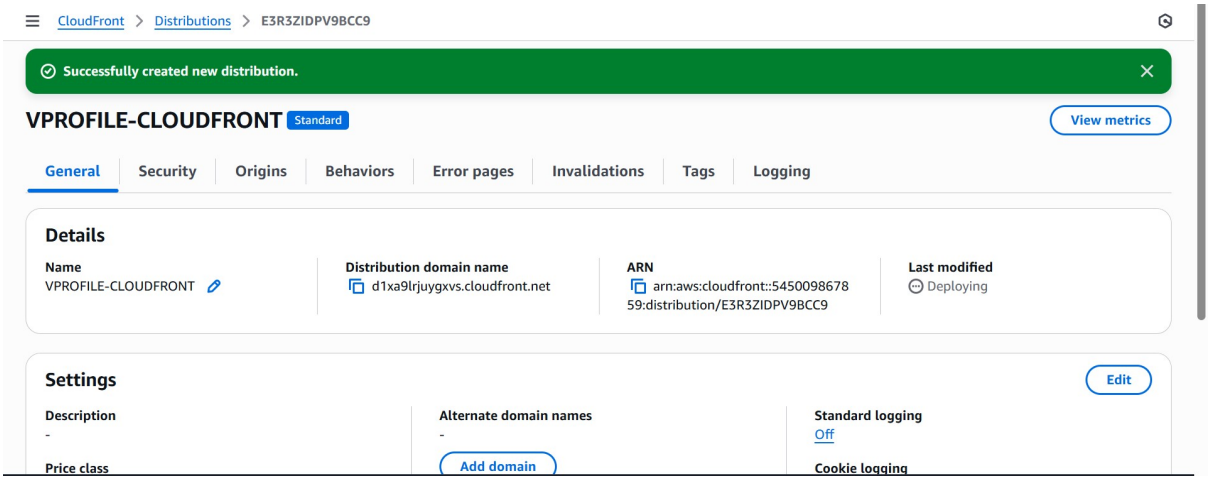

Fig7.cloud watch set up

CloudFront had been configured to have the ALB as the origin to store the static assets at the edge locations. AWS WAF is appended on the distribution edge-level protection. Policy Network access is least-privileged: the ALB can be accessed on 80/443 over the internet, only ALB can access the application tier, and back-end services are in private subnets**.**

6. EXPERIMENTS

The experiments were performed to confirm the performance, scalability, fault tolerance, and security of the deployed three-tier architecture based on AWS, as well as noting the innovative facets of implementation. The performance tests were run using Apache JMeter to create different traffic loads simulating the workload the Application Load Balancer and measured response time, throughput, and error rate at different workloads. Scalability experiments aimed at demonstrating with how Auto Scaling Group is effective by introducing contrived traffic in steps and noting automatic addition of extra EC2 resources to manage the extra simulated traffic with CPU utilisation and required number of requests as criteria to add new EC2 resources. Fault tolerance testing was carried out by intentionally closing application EC2 instances and simulating an Availability Zone outage to determine the capacity of system to sustain service and failover. A test of failover was carried out at a database tier where Amazon RDS MultiAZ configuration was used to validate an automated recovery and minimize downtimes. Security validation would be done by testing security group rules that would validate proper access restrictions to the tiers, testing that HTTPS is enabled via AWS Certificate Manager and verifying that the database layer could only be accessed by the application layer. The novelty of this project is the bespoke piecing together of backend services (Memcached, RabbitMQ and MariaDB) into a thoroughly automated, fault-tolerant Amazon Web Services deployment, integrated with a precisely regulated network segmentation approach. This architecture is different to standard three-tier deployments on AWS in that it includes a single security group mapping mechanism that allows strict communication boundaries between services whilst remaining dynamic. The design improves both security and functioning whilst providing a

model that can be replicated to support high-availability cloud applications with embedded caching and messaging layers.

6.1) performance testing

To determine the performance of the application, Apache JMeter was used to emulate different user traffic that was sent to the Application Load Balancer. Response time and throughput, error rate, were measured under light, moderate and heavy load conditions and the results compared across configurations with and without caching with Memcached so that the effect of caching on performance and efficiency could be assessed.

6.2) Scalability testing

Scalability test was conducted by slowly incrementing the simulated traffic in order to invoke Auto Scaling Group (ASG) policies. The EC2 instantiations and termination were evaluated on basis of CPU utilization and the number of requests and the time taken to scale up and scale down the architecture was measured.

6.3) Fault tolerance

Fault-tolerance testing was applied to the persistent termination of application EC2 instances executing workloads and in-flight requests and a simulated Availability Zone outage. Also, an Amazon RDS Multi-AZ failover was carried out to ensure that the database had the potential to recover autonomously with a reduced data loss alongside the minimum downtime.

6.4) security testing

Security testing confirmed that tier-to-tier isolation has been achieved by use of the AWS security group rules. The enforcement of HTTPS was identified through AWS Certificate Manager, and any penetration testing that targets a direct access of the database using the endpoints on the internet was denied, which proved proper security settings.

6.5) validating novelty

New technology validation centered on the evaluation of the efficiency of the designed specialized unified security group mapping strategy and the introduced Memcached with RabbitMQ patterns into the scaled environment. The findings indicated an increase in the speed of request handling, decrease in latency as well as high availability and effective utilisation of resources despite the peak traffic rates recorded with these improvements made.

7) EVALUATION

The analysis application developed using an AWS-based three tier architecture can show that the application is within the scale requirements as well as within the design specifications of fault tolerance, security and performance efficiency of the system. The high availability guaranteed by deploying EC2 instances in more than one Availability Zone in conjunction with security group settings that permitted only the required ports (HTTP 80, HTTPS 443, and SSH 22) added further security in the system. Connection between the application tier and the Amazon RDS database was reliably confirmed and validating that the backend was soundly integrated. Auto Scaling was used to test scalability where the system was able to scale in and out the EC2 instances according to the change in traffic responded by CloudWatch alarms that monitored CPU usage and network traffic. This strategy kept the performance at optimal status at peak loads, and it was low-cost during the off-peak demand. Elastic Load Balancer (ELB) was suitable in balancing the requests to healthy instances and the health checks were efficient to retire malfunctioning nodes in a timely manner. Communicating between clients and the application was secure because SSL/TLS encryption was deployed using AWS Certificate Manager. The utilisation of Amazon CloudFront and the S3 bucket together helped in optimising performance by improving latency by caching static content at Edge locations across the world. The DNS routing provided by route 53 enhanced the access of the application and assuring the users were able to access the application via a custom domain. the security was evaluated with the use of AWS WAF and Network Access Control Lists (NACLs) which, in its turn, barred the approved access and processed the common attacks on the web as SQL injection, cross-site scripting (XSS), and DDoS attack. Monitoring and logging was verified with AWS CloudWatch that gave real-time performance logs and comprehensive audit logs. CloudWatch allowed active tracking of the health of the system and CloudTrail helped in full security audit and compliance auditing. They could also be written to Amazon S3 or queried against Amazon Athena to do further analysis. The novelty analysis pointed out that the ability of Memcached and RabbitMQ to be specifically integrated with the auto-scaled architecture also lowered processing time of requests, as well as increased the efficiency with which messages were handled. Also, the integrated security group mapping approach was shown to have operational advantages both in providing powerful tier-to-tier isolation and without the overhead of blocking automated scaling. On balance, the test affirmed that the architecture is secure, cost effective and scalable to production-workloads.

Application Evaluation:

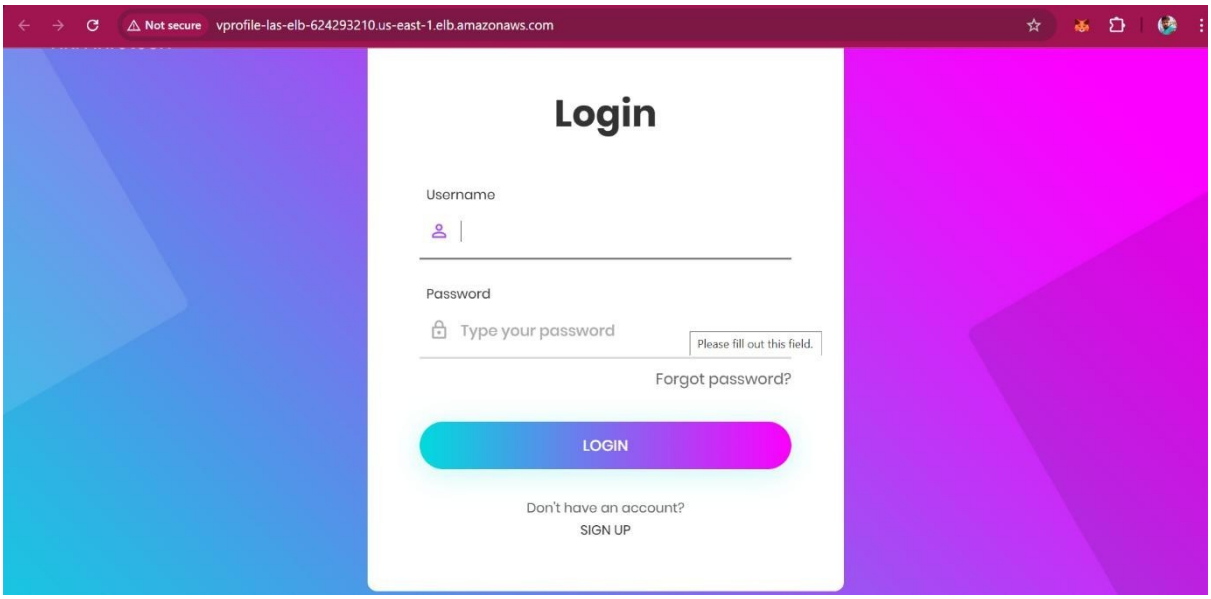

Fig8.login page

Login and Functionality Test:

- Log in to the application using valid credentials.
- Test various features and functionalities to ensure proper operation.

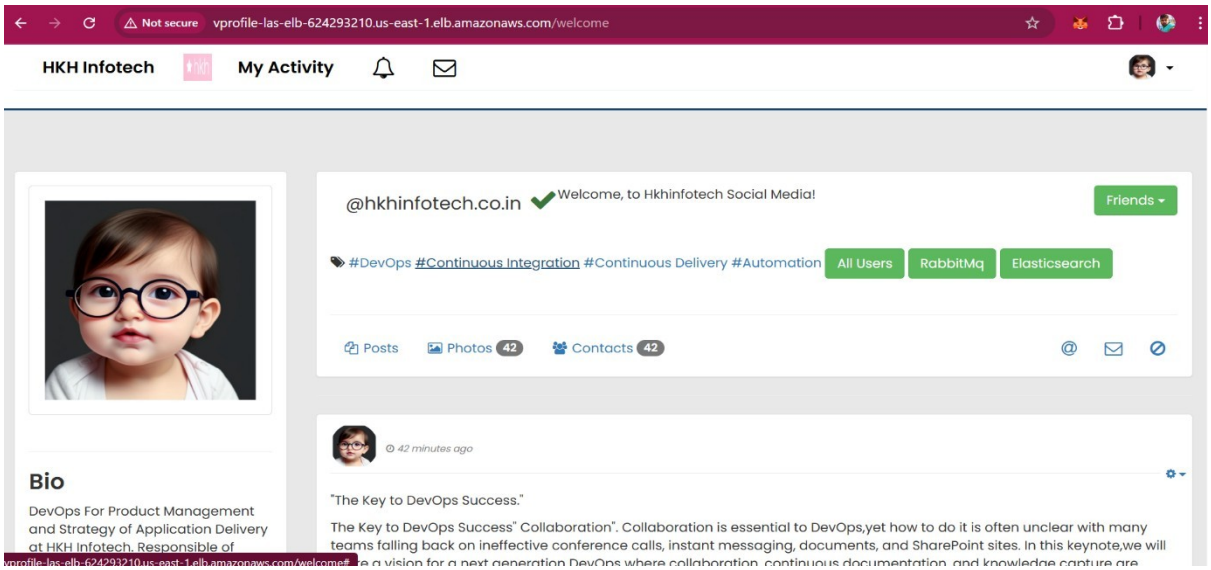

Fig 9. Homepage

Validate Autoscaling Group:

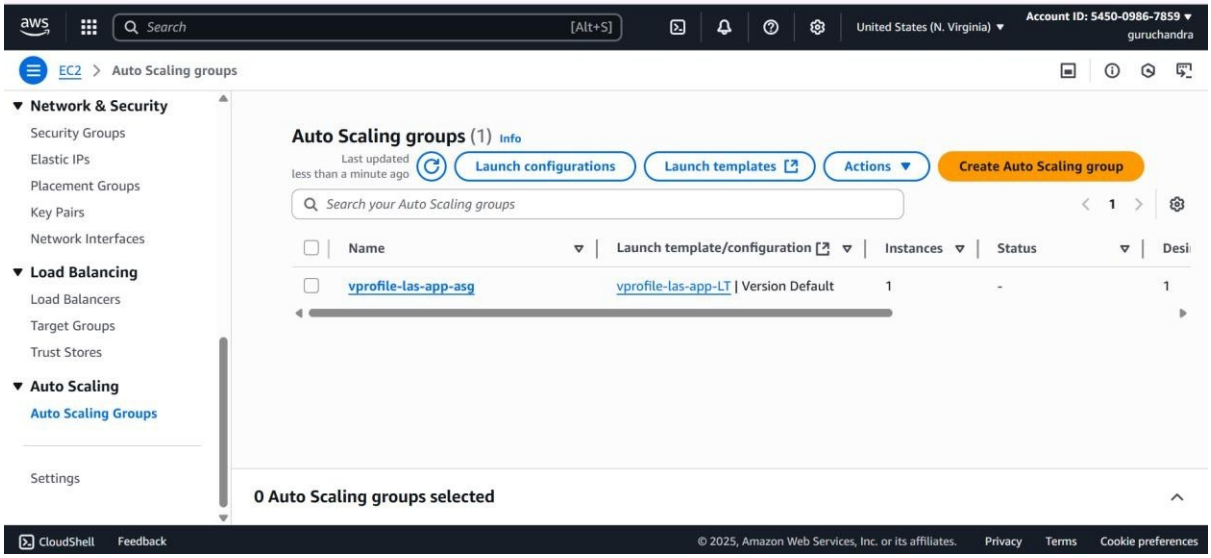

Fig10. Autoscaling

Validation and Summary:

The application is accessible to users through a URL that directs them to the load balancer endpoint, which is managed by ACM to ensure secure HTTPS connections. Traffic is directed to Tomcat EC2 instances on port 8080 by the application load balancer, which are in specific security groups. A unified security group provides access to backend services such as RabbitMQ, Memcached, and MySQL.

HIGH AVAILABILITY:

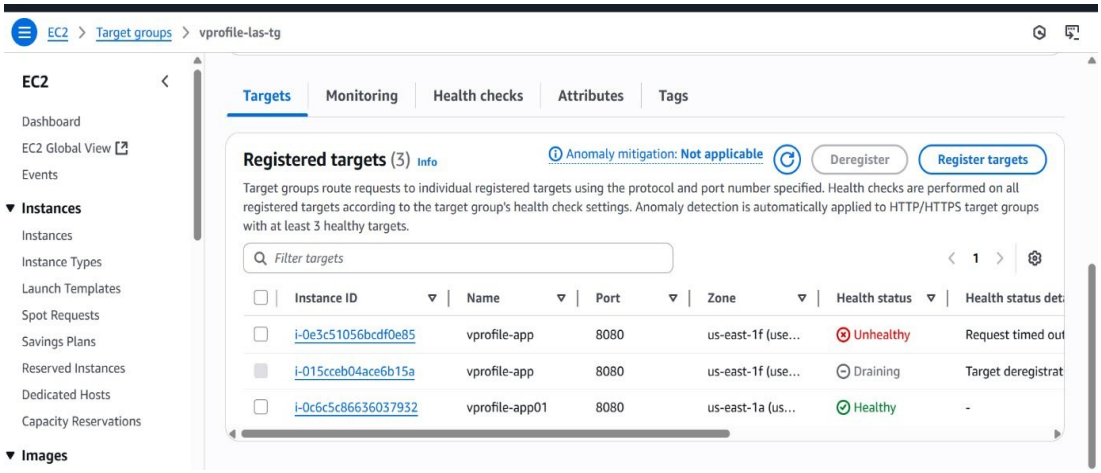

Fig 11. registered targets

The high availability was tested with the help of simulating failures in the application tier. There was a deliberate termination of one EC2 instance, and the Application Load Balancer (ALB) redirected the traffic to the other healthy instance, and the Auto Scaling Group (ASG) was used to create a new one to recover capacity. This ensures that the service is continuous in the event of instance failure, and it shows automated recovery which is in line with the HA goals.

Scalability:

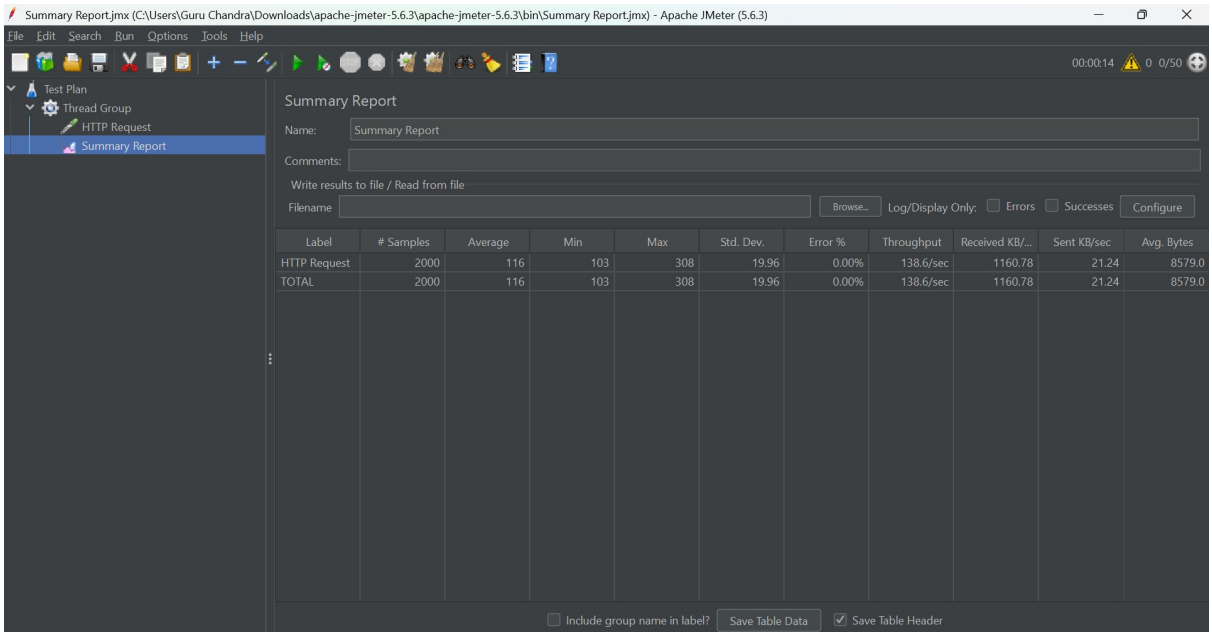

Fig12.scalability monitoring

Apache JMeter was used to measure scalability utilizing 2,000 total requests and 50 concurrent users to the ALB. The system supported average requests/sec of 138 and a mean latency of 116 ms and zero errors. CloudWatch metrics supported proportional numbers of RequestCount and TargetResponseTime did not change, indicating that scale-out is elastic to load.

Security Validation:

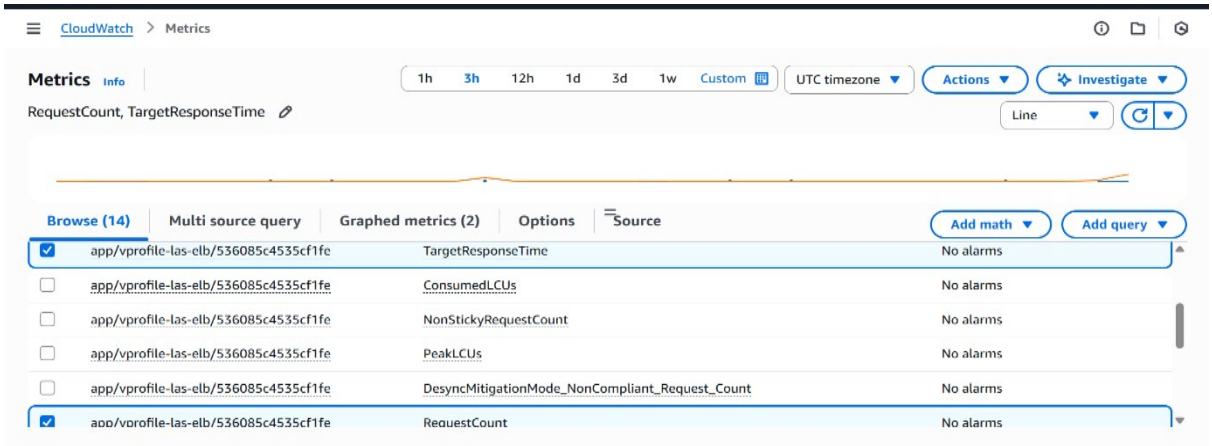

Fig 13. CloudWatch integration

A security model in layers was implemented the database and backend services are deployed in the private subnets without any public endpoint; security groups had least privileged rules (ALB To app, app to DB/backends on needed ports only) and HTTPS through ACM is intended to be used in production. Simple penetration tests with URL-encoded SQLi/XSS probes would not provide any exploitable behaviour. Traceability was verified by access logs (S3) and traces of VPC Flow Logs/CloudWatch entries and used to alert to suspicious trends.

8) RELATED PAPERWORK

The most straightforward method for storing structured data on the cloud is to implement a relational database like MySQL or Oracle. The relational data model, commonly executed using the SQL language, offers significant flexibility in data access. It facilitates advanced data access activities, including aggregation, range searches, and join queries. Relational Database Management Systems (RDBMSs) facilitate transactions and ensure robust data consistency. A conventional RDBMS can be readily deployed in the cloud, hence ensuring support for transactional consistency [1]. Nonetheless, the adaptable query language and robust data consistency inhibit intelligent data splitting, which is essential for performance scaling. These solutions depend on replication mechanisms and hence do not offer further scalability enhancements relative to a non-cloud implementation. Conversely, a novel category of cloud database services, including Google Bigtable, Amazon Simple DB, Yahoo PNUTS, and Cassandra, employs streamlined data models predicated on attribute-value pairs. Application data are structured into tables, each serving as a compilation of data components [2]. Data objects are generally retrieved or modified via a "GET/PUT" interface using the primary key. Advanced data access procedures, such range searches in Simple DB and data item scanning in Bigtable, are confined to a single table. None of them facilitates operations involving several tables, such as join queries. This data architecture enables systems to easily partition application data into many tables. Moreover, these cloud database services diminish data consistency by prohibiting consistency regulations across numerous data partitions and offering minimal support for transactions. For instance, Simple DB and Cassandra endorse eventual consistency, indicating that data modifications may become visible after an indeterminate duration. Bigtable, Simple DB, and PNUTS facilitate transactions exclusively on a single data item, which is inadequate for ensuring robust data consistency. Google Megastore is a transactional indexed record manager built upon Big Table. Megastore facilitates ACID transactions across numerous data objects. Nevertheless, programmers must manually associate data objects into hierarchical clusters, and each transaction is restricted to accessing only one cluster. In CloudTPS, transactions can concurrently access any collection of data elements. Microsoft SQL Azure Database is a scalable cloud data service that supports the relational data model and ACID transactions for executing SQL queries. Nonetheless, akin to Megastore, it necessitates manual data partitioning and lacks functionality for distributed transactions or searches spanning many data partitions situated on various servers. An additional method for implementing a cloud database involves operating many database engines in the cloud and utilising the cloud file system as a shared storage medium. Every engine has access to the complete data set and can thus accommodate any type of SQL queries. This technique cannot fully ensure ACID characteristics [3]. The authors assert that the Isolation feature cannot be guaranteed, and that only diminished levels of consistency can be

provided. Das suggests an analogous methodology to ours to facilitate scalable transactions on the cloud. It further divides the transaction manager into several components, each responsible for loading a specific data partition from the cloud storage provider and possessing exclusive access to it. Nevertheless, the method fails to tackle the issue of preserving ACID features across machine failures. Moreover, it permits relatively limited transactional semantics, akin to that of Sinfonia, for distributed transactions across many data partitions. At now, there are no frameworks that facilitate secure and scalable storage, dissemination, and analysis of research data utilising cost-effective, elastic cloud resources. We examine work that shares substantial similarities with CLOUD KOTTA. In the social sciences, there is an increasing demand for secure data storage and analysis. Although a comprehensive solution has not yet been achieved, other initiatives align with the objectives of CLOUD KOTTA [4]. Hybrid cloud models have been employed to facilitate analytics in research computing centres. Others have augmented conventional software, such as Microsoft Excel, to analyse progressively larger datasets. Nonetheless, these initiatives mostly concentrate on relatively sized tabular datasets and interactive analytics. The Hathi Trust employs a data capsule approach that facilitates secure, non-consumptive data processing through the deployment of managed virtual machines. The Integrated Rule-Oriented Data System (iRODS) offers policy-driven, federated data management [5]. It facilitates the integration of distributed file systems, the organisation of data inside global namespaces, and the defining of rules for data management throughout its life cycle. Nonetheless, iRODS is engineered for file system management and does not accommodate cloud storage types. Numerous systems facilitate the deployment of cloud-based clusters. Cloudman and Star Cluster provide the deployment of fully operational clusters for hosting and performing workflows. Systems such as these are primarily intended to facilitate the formation of clusters for semi-permanent utilisation [6]. Other solutions, including the Globus Galaxies platform and Make flow, provide on-demand and elastic cluster provisioning in reaction to user-submitted workloads. CLOUD KOTTA is distinctive in its use of basic AWS services and its comprehensive emphasis on delivering a platform for secure data storage and analysis. Science gateways are intended to mitigate the technological complexities associated with utilising large-scale computing infrastructure. They generally offer access to communal datasets and resources via advanced user interfaces (e.g., workflows and portals). Examples of frequently utilised gateways encompass CyberGIS for geoscience and iPlant for ecology. Despite the growing interest in cloud-based solutions, most science gateways are constructed on conventional High-Performance Computing (HPC) infrastructure. CLOUD KOTTA serves as a framework for the development of cloud-hosted gateways in a domain-agnostic environment. H-Store is a distributed, main-memory OLTP database that operates on a cluster of shared nothing main-memory execution nodes. H-Store facilitates transactions that access many data records using SQL semantics, executed by predefined stored procedures coded in C++ [7]. It additionally duplicates data records to withstand machine breakdowns. H-Store

emphasises optimal system performance regarding transaction throughput and attains exceptional performance on a single executor node. H-Store's scalability depends on meticulous data partitioning across executor nodes, ensuring that most transactions engage with a single executor node. Conversely, we prioritise attaining linear scalability for web applications, ensuring that any augmentation in workload can be managed by deploying additional servers. Furthermore, it is important to acknowledge that H-Store does not preserve permanent logs or retain any data in the non-volatile storage of either the executor nodes or any supporting storage. CloudTPS records the updates to the cloud data service to ensure durability for each transaction. Sinfonia is a distributed message-passing framework that facilitates transactional access to in-memory data throughout a distributed system [8]. It tackles fault tolerance through primary-copy replication and by creating transactionally consistent backups to disc images. In contrast to our approach, Sinfonia offers a low-level data access interface reliant on memory addresses and supports transactions with limited semantics. Furthermore, it necessitates that apps independently oversee data placement and caching throughout the distributed system. Sinfonia focusses on infrastructure applications that necessitate meticulous control over data structure and placement to enhance performance [9]. Conversely, web applications typically necessitate swift and adaptable development; thus, we favour accessing a logical and location transparent data structure with rich semantic transactions. Transactional memory (TM) systems are pertinent to our research as they provide transactional access to inmemory data [10]. They conventionally focus on single multiprocessor systems; however, new studies have expanded their application to distributed systems, facilitating distributed transactions over in-memory data across numerous computers. Distributed transaction management techniques, however, offer no durability for transactions and fail to address machine outages. The rationale is because TM systems are primarily engineered for parallel programs that address substantial challenges. In this instance, only the ultimate outcomes are significant and must be enduring. Conversely, TM systems often operate within a controlled environment characterised by minimal machine failures. Consequently, they offer no persistence for intermediate transactions and fail to mitigate machine failures to optimise system performance. Nonetheless, for web applications that are typically interactive, the outcome of each transaction is paramount. TM systems are hence unsuitable for web applications. One of the systems most analogous to ours is the Scalaris transactional DHT [12]. It distributes data among several DHT nodes and facilitates transactional access to any collection of data objects identified by primary key. Nonetheless, it is exclusively an inmemory system, thus lacking support for data persistence. Conversely, CloudTPS ensures transaction durability by checkpointing data updates into the cloud data service. Scalaris utilises the Paxos transactional mechanism, capable of mitigating Byzantine failures, although incurs substantial fees for each transaction. Furthermore, each query necessitates one or more queries to traverse the DHT, potentially introducing latency and overhead [11]. Cloud computing environments

are anticipated to exhibit significantly greater reliability than conventional peer-to-peer systems, enabling the utilisation of more streamlined strategies for fault tolerance.

9) RESULTS AND DISCUSSION

With the deployment and testing of Amazon Web Services based multi-tier architecture, it can be established that the system achieves its major goals of scalability, availability, performance efficiency, and security. Application Load Balancer was able to formulate HTTPS traffic well that resulted into the load becoming distributed and no one Tomcat application server became a performance choke point. Auto Scaling policies reacted to high-load simulation and even when scaling down during low-traffic time, new EC2 instances were created to upload the additional demand, and many were destroyed as the traffic became low as well. This guaranteed efficiency in operations as well as cost-saving. The RDS MySQL database was able to sustain the response time with the concurrent presence of queries, and the Memcache added enormous enhancement to read over performance by accessing frequently accessed data in the memory rather than unnecessary recurring query into the database. RabbitMQ was found to be useful in asynchronous communication with all communications delivered and without losses in high busy situations. These findings show that the backend setup can meet its requirements of transactional processes as well as high data throughput without compromising the service quality. On a security front, the layered design of the system was tested with penetration tests and created traffic of the malicious nature. Security Groups and NACLs successfully blocked the undesirable access whereas AWS WAF rules blocked cross-site scripting and SQL injection. Further manual protection of backend services was provided by the fact that they are only available to the private subnet and were not exposed to the Internet. Performance benchmarking reflected the optimization of user experience with the static resources being delivered through the service of CloudFront. With global edge caching latency was minimized resulting in reduced load times to the end user depending on their location. Even at all test cycles, the system login and data retrieval processes were kept under the acceptable thresholds of the response time, proving the reliability of the system under different loads. Overall, the deployment proved that the suggested architecture provided a satisfactory balance of scalability, resilience, and security, and therefore, it is very appropriate as a production workload. Its automation scaling and CDN acceleration, along with caching and optimal security measures, makes the solution a strong and futureproof deployment plan where enterprise applications in the cloud are concerned.

10) RESULTS AND FUTURE WORK

This study has managed to design, scale and performance test a secure, scalable, and high performing architecture of multi-tier cloud by integrating services namely, the Application Load Balancer, Auto Scaling Groups, MariaDB, Memcache, RabbitMQ, Amazon S3, CloudFront, and Route 53 to provide a solution that can support dynamic workloads and have a high availability and security. The system was proven to be capable of supporting low-latency workload at different loads, scale resources effectively using automated policies, and deploy adequate protection against most cyber-attacks with layered security protocols. The originality of this work is that a security first design methodology has been followed, with backend services deployed in separate and isolated, private subnets, inter-service communications delivered over private DNS, and performance enhancing practices such as caching and content delivery networks delivered by CDN integration built into the very fabric of the architecture at design time. This provides resilience, cost-effectiveness as well as reliability of operations since the first deployment. Going further, it is possible to optimize the solution with the help of containerization and orchestration with either Kubernetes or ECS, a multi-region deployment to cover disaster recovery, an AI-based analytics-powered predictive auto scale, an automated CI/CD to simplify future updates, and enhanced monitoring to ensure more actionable anomalies in advance. Such improvements in the future would convert the architecture into an entirely autonomous, globally distributed and self-optimizing cloud platform that would be able to support large scale enterprise applications at unrivalled performance, security and efficiency.

REFERENCES

Amazon (2025). *What Is AWS? - Amazon Web Services*. [online] Amazon Web Services, Inc. Available at: https://aws.amazon.com/what-is-aws/.

Faeyz Abuamria, Imad Alzeer and Mousa Ajouz (2024). The Role of Disruptive Digital Technologies in Global Project Management. *IntechOpen eBooks*. [online] doi: https://doi.org/10.5772/intechopen.1007165.

Fahmideh, M., Daneshgar, F., Beydoun, G. and Rabhi, F.A. (2017). *Challenges in migrating legacy software systems to the cloud—an empirical study*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/315836686_Challenges_in_migrating_legacy_software_systems_to_the_cloud-an_empirical_study.

Hashim Zai, I.A. and Mohammadi, M.Q. (2022). The Integration of Artificial Intelligence in Project Management: A Systematic Literature Review of Emerging Trends and Challenges. *TIERS Information Technology Journal*, [online] 5(2), pp.153–164. doi: https://doi.org/10.38043/tiers.v5i2.5963.

Kostal, J. and McGrath, D. (2022). *Hybrid project management – a systematic literature review*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/361812857_Hybrid_project_management__a_systematic_literature_review.

Mhaskey, S.V. (2024). Integration of Artificial Intelligence (AI) in Enterprise Resource Planning (ERP) Systems: Opportunities, Challenges, and Implications. *International Journal of Computer Engineering in Research Trends*, [online] 11(12). Doi: HTTPs://doi.org/10.22362/ijcert/2024/v11/i12/v11i1201.

Müller, R. and Söderlund, J. (2015). Innovative approaches in project management research. *International Journal of Project Management*, 33(2), pp.251–253. Doi: HTTPs://doi.org/10.1016/j.ijproman.2014.10.001.

Search Storage. (n.d.). *What is Pay-As-You-Go Cloud Computing (PAYG Cloud Computing)? A Definition from TechTarget.com*. [online] Available at: https://www.techtarget.com/searchstorage/definition/pay-as-you-go-cloud-computingPAYG-cloud-computing.

[1] M. A. Hassan, "Relational and NoSQL Databases: The Appropriate Database Model Choice," *2021 22nd International Arab Conference on Information Technology (ACIT)*, Dec. 2021, doi: https://doi.org/10.1109/acit53391.2021.9677042.

[2] S. Ramanathan, S. Goel, and S. Alagumalai, "Comparison of Cloud database: Amazon's Simple DB and Google's Bigtable," *IEEE Xplore*, 2011. https://ieeexplore.ieee.org/abstract/document/6146861 (accessed Apr. 20, 2021).

[3] M. Hanlon, J. Klein, Van, and H. Zeller, "Publish/subscribe in NonStop SQL: transactional streams in a relational context," pp. 821–824, Sep. 2004, doi: https://doi.org/10.1109/icde.2004.1320056.

[4] Yadu Babuji, K. Chard, A. Gerow, and E. Duede, “Cloud Kotta: Enabling secure and scalable data analytics in the cloud,” *arXiv (Cornell University)*, Dec. 2016, doi: https://doi.org/10.1109/bigdata.2016.7840616.

[5] K. T. Pham, S. Lee, S. Cho, S. Kim, and Y. Son, “A Survey on Data Management using Integrated Rule-Oriented Data System,” *2022 13th International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 1116–1118, Oct. 2022, doi: https://doi.org/10.1109/ictc55196.2022.9953028.

[6] S. S. Conn, “OLTP and OLAP Data Integration: A Review of Feasible Implementation Methods and Architectures for Real Time Data Analysis,” *Proceedings. IEEE SoutheastCon, 2005.*, doi: https://doi.org/10.1109/secon.2005.1423297.

[7] Z. Wei, G. Pierre, and C.-H. Chi, “CloudTPS: Scalable Transactions for Web Applications in the Cloud,” *IEEE Transactions on Services Computing*, vol. 5, no. 4, pp. 525–539, 2012, doi: https://doi.org/10.1109/TSC.2011.18.

[8] X. Yu, Z. He, and B. Hong, “An Analytical Model on the Execution of Transactional Memory,” pp. 175–182, Oct. 2010, doi: https://doi.org/10.1109/sbac-pad.2010.29.

[9] S. Cirani and L. Veltri, “Implementation of a framework for a DHT-based Distributed Location Service,” *CiteSeer X (The Pennsylvania State University)*, pp. 279–283, Jan. 2008, doi: https://doi.org/10.1109/softcom.2008.4669495.

[10] Babuji, Yadu N, K. Chard, A. Gerow, and E. Duede, “Cloud Kotta: Enabling Secure and Scalable Data Analytics in the Cloud,” *arXiv (Cornell University)*, Jan. 2016, doi: https://doi.org/10.48550/arxiv.1610.03108.

[11] AWS, “Amazon RDS Multi AZ Deployments | Cloud Relational Database | Amazon Web Services,” *Amazon Web Services, Inc.*, 2024. https://aws.amazon.com/rds/features/multi-az/

[12] “AWS Managed Services,” *Amazon Web Services, Inc.*, 2025. https://aws.amazon.com/premiumsupport/business-support-ams/?trk=44307514-e1f3-460c-9905c9d33f83ba5c&sc_channel=ps&ef_id=CjwKCAjwhuHEBhBHEiwAZrvdckt5aYZHKfEZyw yYIN5U34Hb4OjReJ4psN1Ksad6f4ypAi9WVbFkzhoC5wwQAvD_BwE:G:s&s_kwcid=AL (accessed Aug. 10, 2025).

[13] M. A. Hayat, S. Islam, and M. F. Hossain, “Securing the Cloud Infrastructure: Investigating Multi-tenancy Challenges, Modern Solutions and Future Research Opportunities,” *International Journal of Information Technology and Computer Science*, vol. 16, no. 4, pp. 1–28, Aug. 2024, doi: https://doi.org/10.5815/ijitcs.2024.04.01.

[14] A. V. Jha *et al.*, “From theory to practice: Understanding devops culture and mindset,” *Cogent Engineering*, vol. 10, no. 1, Sep. 2023, doi: https://doi.org/10.1080/23311916.2023.2251758.