

Development of prototype CRM platform using Amazon Web Services

|  |  |
| --- | --- |
| **Project Group** | |
| Stuart Salvara | stuartsalvara1@aol.com |
| [team member name, redacted] | [team member email, redacted] |
| [team member name, redacted] | [team member email, redacted] |

|  |  |
| --- | --- |
| SUPERVISED BY | [project leader name, redacted] |
| CLIENT’S DELEGATE FULL NAME | [client delegate name, redacted] |
| CLIENT’S DELEGATE EMAIL ADDRESS | [client delegate email, redacted] |

|  |  |
| --- | --- |
| **Document Submission Date** *18/08/2024* | ***Final Document Wordcount***  2,857 |

Contents

[Abstract 2](#_Toc357320234)

[Introduction 3](#_Toc1573742312)

[Methodology 3](#_Toc1540773841)

[Substantive Chapter 1 4](#_Toc1002366229)

[Substantive Chapter 2 7](#_Toc47249016)

[Database Design 9](#_Toc1937797166)

[Frontend Development 9](#_Toc294205538)

[Backend Development 9](#_Toc1771060529)

[Security and Authentication 10](#_Toc57714150)

[Infrastructure as Code with CloudFormation 10](#_Toc623657866)

[Conclusion 11](#_Toc2024258370)

[Recommendations 11](#_Toc359441858)

[Bibliography 12](#_Toc25257489)

[Appendix 13](#_Toc1839273473)

# Abstract

This project focuses on the development of a prototype Customer Relationship Management (CRM) platform for [client name, redacted], utilizing AWS tools and services. The primary objective is to automate existing manual processes, thereby enhancing transparency and efficiency. The methodology involves creating a web application that integrates essential CRM functionalities such as client management, account history tracking, and communication logs. The findings suggest that this prototype, with its cost-effective and scalable architecture, can significantly improve [client name, redacted], operational workflows and client management capabilities.

# Introduction

The objective of this project is to address the operational inefficiencies faced by [client name, redacted], due to manual processes in managing client relationships. As a leading IT and business consulting firm, [client name, redacted], requires a robust CRM system to manage its vast clientele effectively. This project aims to develop a prototype CRM platform that automates these processes, leveraging AWS's cloud services for a scalable, efficient, and transparent solution.

The project is designed to create a proof-of-concept web application incorporating key CRM functionalities, including client management, account history, and communication logs. By automating these processes, [client name, redacted], can expect to save valuable time and reduce costs, while also increasing the transparency and efficiency of their operations. This report outlines the project's scope, objectives, and methodology, focusing on the integration of AWS tools to achieve the desired outcomes. The project aims to fill the existing gap in [client name, redacted], workflow by providing a scalable and cost-effective solution that enhances overall client management and operational efficiency.

# Methodology

Based on the project briefing we were tasked with building a CRM system incorporating AWS tools and services. As we considered which AWS resources would be best suited for this project, we had to first consider what would be the framework for creating our CRM model and therefore the following questions were raised.

* What existing CRM models existed that we could use as a model template for the model that we wanted to create?
* What would be the main components for our web application infrastructure? (EC2 instances, Auto Scaling Groups, subnets etc)
* How many tiers would exist for our web application CRM model? Example being a two-tier Infrastructure model vs a three-tier model.
* What AWS services would we require for the model and what would be the rationale for using each service?
* What would be the projected cost of the web application CRM model and what other considerations would be required in terms of functionality, scalability, latency/availability and storage for our proof-of-concept CRM model.

In answering these questions, we began researching existing CRM models. The company Salesforces for us in achieving this objective and is regarded as a globally leading CRM system. Salesforce provides CRM software and applications focused on sales, customer service, marketing automation, e-commerce, analytics, and application development. As part of our project brief, we had to include a table of contents, index page and a login function in our CRM. The Salesforce CRM had all these features.

We had to also outline what the main components would be for our web application infrastructure. We decided that we would use a three-tier application (*Appendix 1 - The three tier web application model)* which would be best suited for our CRM model based on the considerations of scalability, security, and storage capacity in terms of the AWS services for what we would require we researched the services that would investigate the following areas :-

* Storage
* Compute
* Core Services
* Networking
* Security
* Monitoring and Analytics
* Development and Management tools
* Cost

We decided on at least one service for each area. A full listing of the AWS services which we utilized including rationale for their use are contained in the document *(Appendix 2 - AWS services for the Project).*

# Substantive Chapter 1

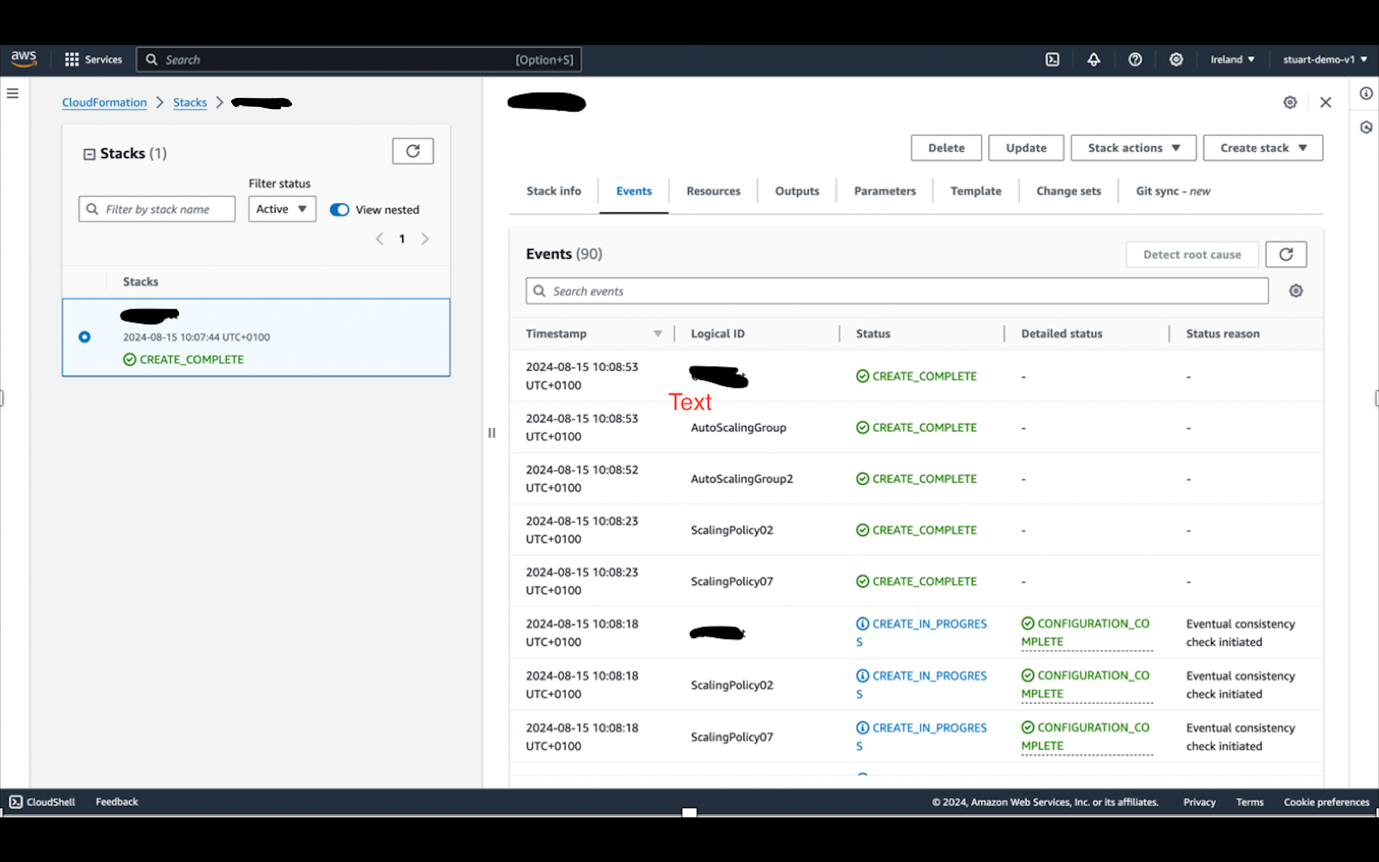
Once we had decided on the infrastructure that would be created including all the associated AWS services that would be deployed, we set about trying to have a better understanding of the associated costs for each of these services using the AWS pricing calculator. We were able to make basic assumptions on the configuration of our CRM model in terms of storage, computing power, scalability based on similarly sized CRM models and the general basic functionality features that we wanted our CRM model to have. (*Appendix 3 - AWS pricing calculator estimate for the project)* contains all the AWS services that we utilized and their estimated costs per month. Some of the estimated cost numbers were guestimates and did not consider certain variables such as seasonal differences in demand to access to our CRM as well as changes to our database storage demands based on the numbers of the clientele.

Once we had established an approximate pricing for the AWS services we would incorporate as part of our CRM we sought to deploy those resources in AWS CloudFormation. As part of the project brief CloudFormation is an AWS service that can automatically create resources in the cloud in a secure manner. Resources can be created using infrastructure as code in which the code for each resource e.g security groups is written onto a template and uploaded as a stack into the CloudFormation function of the AWS management console. The role of CloudFormation in creating our resources is explained in more depth in the section “Infrastructure as code with CloudFormation”

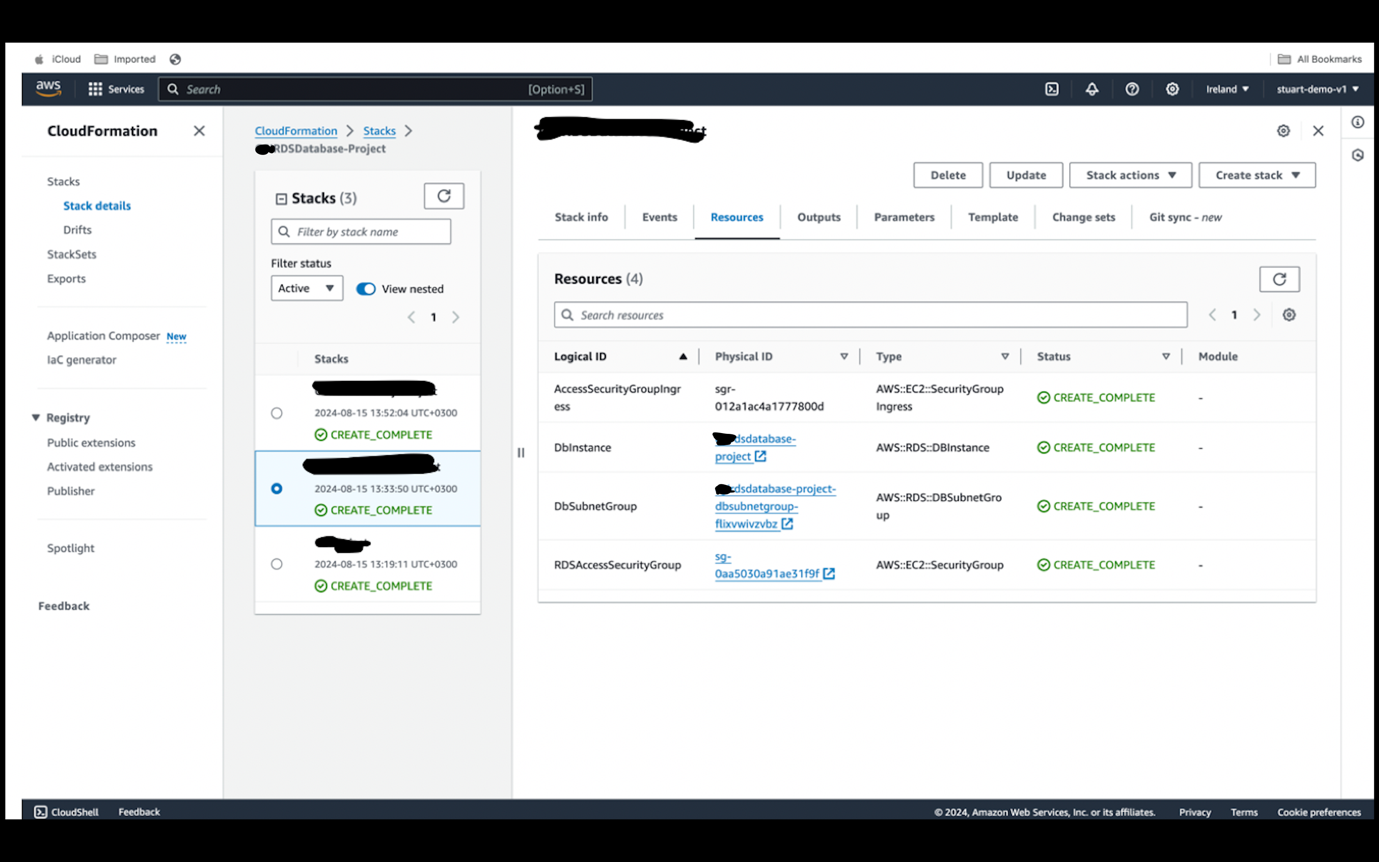
We were able to utilize some of the many templates that exist on GitHub which is where we selected our CloudFormation template, and we found three templates which had all the resources that we believed would be required for the three-tier web application. The first template had all the components within the VPC we required except for the NAT gateway. The second template that we uploaded as a stack was the RDS database. This contained the dummy data that we were permitted to use to upload to the RDS database. The last template we created was used to create the NAT Gateway used to connect the private subnets to the internet.

These templates gave us all the components that we needed. The next step was to save these templates as YAML files to be uploaded to CloudFormation as a stack, please refer to (*Appendix 4 - YAML code template*). Once we uploaded these templates, we were able to let CloudFormation as an automating tool configure the components to the size, CPU power and CIDR block (IP addresses). A full list of the template resources listing is below. One of the components of our system is the RDS database which would be used to host our data for the CRM.

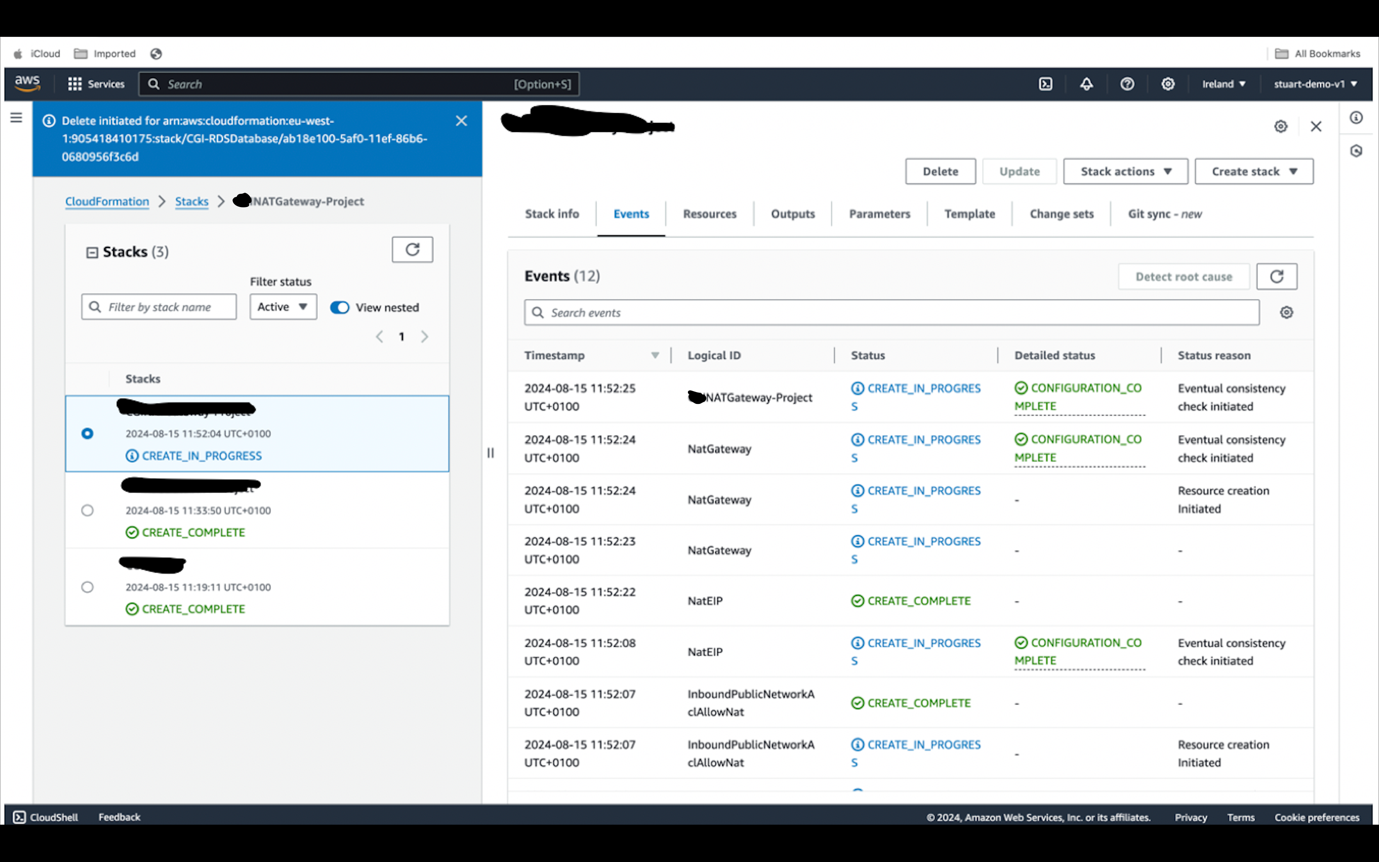
* Subnets
* Application Loading Balancer (x2)
* Auto Scaling group (x2)
* EC2 Instances x4
* RDS Database
* Public Subnets (x2)
* Private Subnets (x4)
* NAT Gateway
* Internet Gateway
* VPC



*Fig 1.1 - The above image shows the CloudFormation stack containing the resources needed to create our three-tier web application model e.g security groups, subnets, EC2 instance.*



*Fig 1.2 - The above image shows the CloudFormation stack containing the RDS database instance and associated resources*



*Fig 1.3 - The above image shows the NAT gateway created using another CloudFormation stack. The NAT gateway is used to connect our private subnets to the internet.*

# Substantive Chapter 2

Once we uploaded the stack (template) via CloudFormation using the management console our CRM system was automatically generated using a URL that directly connected to our database.

A screenshot of a computer

Description automatically generated

*Fig 1.4 - The above image is the Prototype CRM system that we have created utilizing the three tier web application resources and associated AWS resources.*

The main task at this point was to test our CRM based on the functionality we had to ensure that the CRM model had based on our project brief. This included the following items

* Front Page
* Table of Contents
* Database
* IAM

We also had to ensure that the AWS services that we listed in “**AWS Services for the project**” were fully integrated into the functionality of the CRM model. As part of this process after the three-tier architecture system was created, we had to add the AWS services to the architecture that we required to have a complete secure, scalable and had the required storage space. These services are outlined clearly in the “**AWS services for the project**” file. Our CRM system had to be built on top of the three-tier architecture using serverless AWS services which would reduce the need for managing physical servers, lowered operational costs, and ensured that the system could scale automatically to meet demand. The core components of the system include:

* **Amazon RDS**: A relational database service that stores all customer-related data, including client information, contact details, account history, deals, calls, emails, and reminders.
* **AWS Lambda**: Serverless compute service that handles all backend processing and CRUD (Create, Read, Update, Delete) operations for the database.
* **Amazon API Gateway**: A fully managed service that allows developers to create, publish, maintain, monitor, and secure APIs. It acts as the front door for applications to access data and business logic in the CRM.
* **Amazon S3**: A scalable object storage service used to store static assets such as HTML, CSS, and JavaScript files that make up the CRM’s user interface.
* **Amazon Cognito**: A service that manages user authentication and authorization, providing secure access to the CRM system.
* **AWS CloudFormation**: An infrastructure-as-code service that automates the provisioning and management of AWS resources used by the CRM system

### **Database Design**

At the heart of our CRM system is a relational database managed by Amazon RDS. The database schema is designed to handle various aspects of customer relationship management, including:

* **Clients Table**: Stores primary information about each client, such as their name, contact details, and unique identifier.
* **Contacts Table**: Maintains a list of contacts associated with each client, including their roles, phone numbers, and email addresses.
* **Account History Table**: Records all interactions and transactions between the client and the organization, providing a historical overview of the client’s relationship with the company.
* **Deals Table**: Tracks opportunities and deals, including their status, value, and associated client information.
* **Calls Log Table**: Logs all telephone interactions with clients, including the date, time, and purpose of the call.
* **Emails Log Table**: Maintains a record of email communications with clients, capturing the content and metadata of each email.
* **Reminders Table**: Stores reminders and alerts for follow-up actions, ensuring that important tasks are not overlooked.

This database schema is implemented using SQL scripts or Object-Relational Mapping (ORM) tools, depending on the development approach.

### **Frontend Development**

The user interface (UI) of the CRM system is designed to be intuitive and easy to navigate. It is developed using standard web technologies, including HTML, CSS, and JavaScript. (*Appendix 5 – Front end code in JS (JavaScript*)

The key pages include:

* **Front Page**: The landing page that provides an overview of the CRM system and quick access to its various functionalities.
* **Table of Contents**: A navigation menu that allows users to easily access different sections of the CRM, such as clients, contacts, account history, deals, calls, emails, and reminders.
* **Client Information Form**: A form that allows users to input and view client information, contact details, and logs.

These static assets are stored in an Amazon S3 bucket, ensuring high availability and fast load times. The UI communicates with the backend via RESTful APIs, allowing users to perform CRUD operations on the data stored in the database.

### **Backend Development**

The backend of the CRM system is powered by AWS Lambda, which provides serverless compute resources that automatically scale to handle varying levels of demand. Each Lambda function is responsible for a specific set of operations, such as fetching client data, updating contact information, or deleting a record from the database. (*Appendix 6 – Lambda Function in JS (JavaScript*)

The backend is exposed to the frontend through Amazon API Gateway, which routes HTTP requests to the appropriate Lambda function. This setup ensures that the system is modular and easily extensible, allowing new features to be added without significant changes to the existing codebase.

The Lambda functions handle various CRUD operations, including:

* **GET /clients**: Retrieves a list of all clients from the database.
* **GET /clients/{id}**: Retrieves detailed information about a specific client.
* **POST /clients**: Adds a new client to the database.
* **PUT /clients/{id}**: Updates the information of an existing client.
* **DELETE /clients/{id}**: Deletes a client from the database.

Each Lambda function is triggered by an API request and interacts with the RDS database to perform the necessary operations. The use of Lambda functions eliminates the need for dedicated servers, reducing costs and simplifying maintenance.

### **Security and Authentication**

Security is a critical aspect of the CRM system. Amazon Cognito is used to manage user authentication and authorization. Users must authenticate themselves using their credentials before accessing the system, and their permissions are controlled based on their roles within the organization.

Cognito integrates seamlessly with AWS Lambda and API Gateway, ensuring that only authorized users can perform specific operations. For example, only users with the appropriate permissions can delete records or access sensitive client data.

In addition to Cognito, other AWS security best practices are implemented, including:

* **Encryption at Rest**: All data stored in Amazon RDS and S3 is encrypted using AWS Key Management Service (KMS).
* **Encryption in Transit**: Communication between the frontend, backend, and database is secured using HTTPS, ensuring that data is encrypted during transmission.
* **IAM Roles**: AWS Identity and Access Management (IAM) roles are used to control access to AWS resources, ensuring that only authorized services and users can interact with the CRM system.

### **Infrastructure as Code with CloudFormation**

AWS CloudFormation is used to automate the deployment and management of the CRM system’s infrastructure. A CloudFormation template defines all the resources needed for the system, including the RDS database, Lambda functions, API Gateway, S3 bucket, and Cognito user pool.

By using CloudFormation, the entire infrastructure can be provisioned consistently and repeatably with minimal manual intervention. This approach reduces the risk of configuration errors and ensures that the system can be easily replicated across different environments, such as development, testing, and production.

# Conclusion

The three tier web application infrastructure we have outlined in this document has all the components required for full functionality. We have explained all the components of this architecture and how we were able to create the resources for the three-tier web application.

The CRM system described in this document provides a comprehensive solution for managing customer relationships. By leveraging AWS’s serverless architecture, the system is scalable, cost-effective, and secure, making it well-suited for businesses of all sizes. The modular design allows for easy expansion and customization, ensuring that the system can evolve to meet the changing needs of the organization. Whether managing client information, tracking deals, or logging communications, this CRM system offers the tools necessary to enhance customer engagement and drive business growth.

In terms of the scalability of our CRM system we have incorporated the AWS service Amazon S3 which gives us scalable storage. We have also used the Amazon pricing calculator to estimate the costs of all the AWS services that we require, and we will use the service AWS Cognito and IAM access management to provide user access control and user authentication for our AWS services.

# Recommendations

Having created or CRM system with full functionality. We can state definitively that we have created a fully functioning CRM that can be used by our client [client name, redacted], for all the purposes set out in our project brief. We recommend that [client name, redacted], adopt this infrastructure framework and utilize the CRM model that we have created to help its broader business manage their interactions with customers to improve relationships and grow their business operations. As well as the obvious considerations of sales, marketing and customer support this CRM model achieves all of the objectives that a CRM model should have in terms of contact management, task management and reporting and analytics.

As a team we feel that we have had the opportunity to create a working CRM model in an efficient manner given the tools available to us via CloudFormation. However, we also feel as a team that if we had more time possibly another 2 weeks, we would have been able to more critically assess alternative CRM models such as Saleforce’s CRM offering which we could use to model our approach to this project briefing on. In addition, if we had another two weeks to complete the project, we possibly could have been able to give a more accurate assessment of the cost impact of the AWS services we deployed as part of our CRM model, rather than rely on guesstimate costs. To further explain this the AWS pricing calculator gives an assessment of the cost of AWS services over a specific period of time but does not accurately assess the cost impact of extremely high external demand (traffic) to our CRM model as well as other potential variables which could significantly change the cost projection over the course of a year.

Finally, as a team project we would like to state that this has been a great learning experience and as a group we would like to thank [client name, redacted], and IN4 Group for giving us the opportunity to tackle this challenge and to prove that we are capable of creating a functional infrastructure model that can be utilized by an organization that size of [client name, redacted].

# Bibliography

1. Baraqheart. (n.d.). Deploy highly available and secure 2-tier application on AWS. Medium. Retrieved August 6, 2024, from <https://baraqheart.medium.com/deploy-highly-available-and-secure-2-tier-application-on-aws-9b30f96cc3de>
2. Building a CRM application on AWS for Medicaid patient intake. (2024, March 29). Medium. Retrieved August 6, 2024, from <https://medium.com/@purushothamkamath/building-a-crm-application-on-aws-for-medicaid-patient-intake-7d93dec2a3cc>
3. Building a 3-tier web application architecture with AWS. (2023, November 1). Medium. Retrieved August 6, 2024, from <https://medium.com/@aaloktrivedi/building-a-3-tier-web-application-architecture-with-aws-eb5981613e30>
4. Build a 2-tier web application using AWS core services (VPC, EC2, RDS, S3). (2022, May 3). AWS in Plain English. Retrieved August 6, 2024, from <https://aws.plainenglish.io/build-a-2-tier-web-application-using-aws-core-services-vpc-ec2-rds-s3-40cb03f88b77>
5. Build a 3-tier architecture. (2021, May 3). AWS in Plain English. Retrieved August 6, 2024, from <https://aws.plainenglish.io/aws-3-tier-architecture-4366d514a62f>
6. Introduction. (n.d.). AWS Workshops. Retrieved August 6, 2024, from <https://catalog.us-east-1.prod.workshops.aws/workshops/85cd2bb2-7f79-4e96-bdee-8078e469752a/en-US/introduction>
7. Architecture Overview. (n.d.). AWS Workshops. Retrieved August 6, 2024, from <https://catalog.us-east-1.prod.workshops.aws/workshops/85cd2bb2-7f79-4e96-bdee-8078e469752a/en-US/introduction#architecture-overview>
8. Welcome to AWS CloudFormation. (n.d.). AWS Documentation. Retrieved August 6, 2024, from <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html>
9. Parham, K. (2022) Create a 3-Tier Architecture via AWS CloudFormation. Available at: <https://levelup.gitconnected.com/create-a-3-tier-architecture-via-aws-cloudformation-b25e450e8acd> (Accessed: 14 August 2024).

# Appendix

**Appendix 1**

***The three tier web application model***

A screenshot of a computer

Description automatically generated

**Appendix 2**

***AWS Services for the Project***

**Presentation Tier – Frontend (1st)**

**Storage**

1. **Amazon S3 (Simple Storage Service)**:
   * **Purpose**: For scalable storage.
   * **Description**: Ideal for storing files, backups, and static content such as images and documents.

**Application Tier – Backend (2nd)**

**Compute**

1. **Amazon EC2 (Elastic Compute Cloud)**:
   * **Purpose**: For scalable computing capacity.
   * **Description**: Allows you to rent virtual servers (instances) to run the CRM application.
2. **Lambda (Serverless computing)**
   * **Purpose**: For serverless computing.
   * **Description**: compute service that handles all backend processing and CRUD (Create, Read, Update, Delete) operations for the database.

**Database Tier (3rd)**

**Core Services**

1. **Amazon RDS (Relational Database Service)**:
   * **Purpose**: For managing relational databases.
   * **Description**: Supports MySQL, PostgreSQL, Oracle, and SQL Server among others. It provides automated backups, software patching, automatic failure detection, and recovery.

**Networking**

1. **Amazon VPC (Virtual Private Cloud)**:
   * **Purpose**: For network management.
   * **Description**: Allows the launch of AWS resources in a logically isolated virtual network.

**Security**

1. **AWS IAM (Identity and Access Management)**:
   * **Purpose**: For managing user access.
   * **Description**: Allows securely controlled access to AWS services and resources for our users.
2. **AWS Cognito**:
   * **Purpose**: For user authentication.
   * **Description**: Provides user sign-up, sign-in, and access control.

**Monitoring and Analytics**

1. **Amazon CloudWatch**:
   * **Purpose**: For monitoring.
   * **Description**: Provides monitoring for AWS cloud resources and applications, collects, and tracks metrics, and sets alarms.

**Development and Management Tools**

1. **AWS CloudFormation**:
   * **Purpose**: For resource provisioning.
   * **Description**: Helps to model and set up our AWS resources so that we can spend less time managing those resources and more time focusing on the applications.

**Cost**

1. **AWS Pricing Calculator:**
   * **Purpose**: To cost projection
   * **Description**: To give us an estimated cost of the utilization of AWS services in creating our CRM model as well as all components of the three-tier web application model.

**Appendix 3**

***AWS pricing calculator estimate for the project (Costs projected in $USD on a per monthly basis)***

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Monthly cost | Total 12 months cost | Currency |  |  |  |  |  |  |
| 7164.4 | 85,972.80 | USD |  |  |  |  |  |  |
|  | \* Includes upfront cost |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| Region | Description | Service | Upfront | Monthly | First 12 months total | Currency | Status | Configuration summary |
| Europe (Ireland) | Project | Amazon API Gateway | 0 | 111 | 1332.00 | USD |  | Cache memory size (GB) (None), WebSocket message units (thousands), HTTP API requests units (millions), Average size of each request (34 KB), REST API request units (millions), Average message size (32 KB), Requests (100 per month) |
| Europe (Ireland) | Project | Amazon CloudWatch | 0 | 15.0015 | 180.02 | USD |  | Number of Metrics (includes detailed and custom metrics) (50), GetMetricData: Number of metrics requested (50), Number of other API requests (100) |
| Europe (Ireland) | Project | S3 Standard | 0 | 1177.6 | 14131.20 | USD |  | S3 Standard storage (50 TB per month), PUT, COPY, POST, LIST requests to S3 Standard (50), GET, SELECT, and all other requests from S3 Standard (50) |
| Europe (Ireland) | Project | Data Transfer | 0 | 0 | 0.00 | USD |  |  |
| Europe (Ireland) | Project | Amazon EC2 | 0 | 1476.644 | 17719.73 | USD |  | Tenancy (Dedicated Instances), Operating system (Linux), Workload (Consistent, Number of instances: 4), Advance EC2 instance (t3.nano), Pricing strategy (On-Demand Utilization: 100 %Utilized/Month), Enable monitoring (disabled), DT Inbound: Not selected (0 TB per month), DT Outbound: Not selected (0 TB per month), DT Intra-Region: (0 TB per month) |
| Europe (Ireland) | Project | AWS Lambda | 0 | 0 | 0.00 | USD |  | Architecture (x86), Architecture (x86), Invoke Mode (Buffered), Amount of ephemeral storage allocated (512 MB), Number of requests (1000 per month) |
| Europe (Ireland) | Project | AWS CloudFormation | 0 | 1367.85 | 16414.20 | USD |  | Number of third-party extensions managed (500), Average duration per operation (30 seconds), Total number of operations per extension (100 per day) |
| Europe (Ireland) | Project | VPN Connection | 0 | 1333.5 | 16002.00 | USD |  | Working days per month (22), Number of Site-to-Site VPN Connections (25), Number of subnet associations (2) |
| Europe (Ireland) | Project | Network Address Translation (NAT) Gateway | 0 | 526.56 | 6318.72 | USD |  | Number of NAT Gateways (1) |
| Europe (Ireland) | Project | Transit Gateway | 0 | 138.9 | 1666.80 | USD |  | Number of Transit Gateway attachments (1) |
| Europe (Ireland) | Project | Gateway Load Balancer | 0 | 56.8 | 681.60 | USD |  | Number of Availability Zones that Gateway Load Balancer is deployed to (1), Number of Gateway Load Balancer Endpoints (1), Total processed bytes (5 TB per month), Average number of new connections/flows (10 per minute), Average connection/flow duration (5 minutes) |
| Europe (Ireland) | Project | Amazon Cognito | 0 | 540.25 | 6483.00 | USD |  | Optimization Rate for Token Requests (0.1), Optimization Rate for App Clients (0.1), Advanced security features (Enabled), Number of monthly active users (MAU) (5), Number of token requests (2), Number of app clients (100) |
| Europe (Ireland) | Project | AWS IAM Access Analyzer | 0 | 8 | 96.00 | USD |  | Number of accounts to monitor (10), Average roles per account (2), Average users per account (2), Number of analyzers per account (1) |
| Europe (Ireland) | Project | Amazon RDS for MySQL | 0 | 412.30000000000000 | 4947.60 | USD |  | Storage amount (30 GB), Storage for each RDS instance (General Purpose SSD (gp2)), Nodes (2), Instance type (db.m1.medium), Utilization (On-Demand only) (100 %Utilized/Month), Deployment option (Multi-AZ), Pricing strategy (OnDemand) |

**Appendix 4**

YAML template codes (3 templates)

***YAML template to create the CloudFormation stack for the following resources***

* VPC
* Route Table
* Internet Gateway
* Attaches Internet Gateway to the VPC you just created
* 2 Public Subnets
* Scaling Policy for Public Subnets
* Route Table for Public Subnets
* Associate the Public Subnets to the Route Table
* Security group with HTTP & SSH to open to 0.0.0.0/0
* Launch Template with Bootstrap script to install & launch Apache
* Autoscaling Group for Public Subnets
* Scaling Policy
* 4 Private Subnets — two in us-east1a & two in us-east1b
* Autoscaling Group for two Private Subnets in the Application tier

<https://gist.githubusercontent.com/KinzP/8b2c83003bbf1688a90dca5ce597d039/raw/70bb39b05b44e9fb9e2b5361661beb7368f2fb23/gistfile1.txt>

***YAML template to create the CloudFormation stack for RDS Database tier***

* RDS Database Tier

<https://gist.githubusercontent.com/KinzP/f35870bcca868b9f103fcc1bd05385c4/raw/c959b359060d67d6ae41b1a0866b3f5fb4bbea05/RDS%20MySql>

***YAML template to create the CloudFormation stack for NAT Gateway***

* NAT Gateway

***https://gist.githubusercontent.com/KinzP/66bcff17ad694fa60ac3da3e57aa6b43/raw/56b771eabf0db71e4108eb2ceed5d16ccba51ea4/NATGateway***

**Appendix 5**

Front end code in JS (JavaScript)

A screenshot of a computer program

Description automatically generated

**Appendix 6**

Lambda function (Backend Code) in JS (JavaScript)

A screenshot of a computer program

Description automatically generated