**Deployment & Operations Guide Document**

CareConnect

University of Maryland Global Campus

SWEN 670 - Software Engineering Capstone

Dr. Mir Assadullah

August 3rd, 2025

Contributors: Diane Angeles, Torie Bias, Juan Gaucin Calderon, Luke Curran, Ashenafi Grbreegziabhere, Alyssa Marielle Harding, Astha Malla-Paudel, Maria Ramirez, Edwenson Raphael, Dat Truong, Alex Vecchioni, Christian Yawn

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 07/26/2025 | 1.0 | Initial Release | CareConnect Team |
| 08/03/2025 | 2.0 | Milestone 4 update | CareConnect Team |
|  |  |  |  |

**Sign-off Sheet**

|  |  |  |  |
| --- | --- | --- | --- |
| **Role** | **Name** | **Signature** | **Date** |
| Project Manager | Alireza Minagar |  |  |
| Team Lead | Alyssa Harding |  |  |
| Lead Tester | Juan Gaucin |  |  |
| Client | Dr. Mir Assadullah |  |  |

# Table of Contents

[Table of Contents 3](#_Toc205126746)

[1. Introduction 5](#_Toc205126747)

[1.1 Purpose 5](#_Toc205126748)

[1.2 Intended Audience 5](#_Toc205126749)

[1.3 Document Organization 5](#_Toc205126750)

[1.4 Project Documents 5](#_Toc205126751)

[1.5 Definitions, Abbreviations, and Acronyms 6](#_Toc205126752)

[1.6 References 7](#_Toc205126753)

[2. Source Control 7](#_Toc205126754)

[2.1 GitHub Access 7](#_Toc205126755)

[2.2 Repository Structure 7](#_Toc205126756)

[2.3 Pull Request Guidelines 8](#_Toc205126757)

[2.3.1 Prevent Pushing “Bad” Code 8](#_Toc205126758)

[2.4 Collaboration Methods 8](#_Toc205126759)

[2.4.1 Teams Channel 8](#_Toc205126760)

[2.4.2 Meetings 8](#_Toc205126761)

[2.4.3 Emails 9](#_Toc205126762)

[2.5 CI/CD Pipeline 9](#_Toc205126763)

[3. CareConnect Setup 10](#_Toc205126764)

[3.1 System Requirements 10](#_Toc205126765)

[3.2 Software Installation 10](#_Toc205126766)

[3.2.1 VS Code 10](#_Toc205126767)

[3.2.2 Android Studio 10](#_Toc205126768)

[3.2.3 Android Emulator 11](#_Toc205126769)

[3.2.4 IntelliJ 11](#_Toc205126770)

[3.2.5 Git 11](#_Toc205126771)

[3.2.6 GitHub Desktop 11](#_Toc205126772)

[3.2.7 Flutter + Dart 11](#_Toc205126773)

[3.2.8 AWS Account 12](#_Toc205126774)

[3.2.9 AWS Command Line Interface 12](#_Toc205126775)

[3.2.10 Terraform 12](#_Toc205126776)

[3.2.11 MySQL Database Engine 12](#_Toc205126777)

[3.3 Program Configurations 13](#_Toc205126778)

[3.3.1 Third-Party APIs 13](#_Toc205126779)

[3.3.1.1 Stripe 13](#_Toc205126780)

[3.3.1.2 Google Health Connect 14](#_Toc205126781)

[3.3.1.3 DeepSeek 14](#_Toc205126782)

[3.3.1.4 OpenAI 15](#_Toc205126783)

[3.3.1.5 Apple Health 16](#_Toc205126784)

[3.3.1.6 Fit Bit 16](#_Toc205126785)

[3.3.1.7 OpenFDA 16](#_Toc205126786)

[3.4 Compiling CareConnect 17](#_Toc205126787)

[3.4.1 Front-End 17](#_Toc205126788)

[3.4.1.1 Environment Variables 17](#_Toc205126789)

[3.4.2 Back-End 17](#_Toc205126790)

[3.4.2.1 Environment Variables 17](#_Toc205126791)

[3.5 Deployment to AWS 18](#_Toc205126792)

[4. Troubleshooting 19](#_Toc205126793)

[4.1 Common Deployment Errors 19](#_Toc205126794)

[4.2 Log Collection 20](#_Toc205126795)

# 1. Introduction

## 1.1 Purpose

This Deployment and Operations Guide provides a series of detailed instructions to deploy, configure, and maintain the CareConnect system. CareConnect is a healthcare administration program that assists in communication and job coordination among caregivers and patients to make healthcare an easy way to follow. This guide will assist system administrators, IT professionals, and support personnel in successfully implementing the CareConnect application and its monitoring and maintenance.

## 1.2 Intended Audience

The target audience of this guide will be IT administrators, deployment engineers, and members of the deployment support staff who will be in charge of the implementation, use, and troubleshooting of CareConnect. The guide outlines the steps of releasing the system to production, integrating it into (third-party) services, and suggestions on resolutions of common operational problems. This guide can also be useful to developers and testers as they seek to take the right direction in their deployments.

## 1.3 Document Organization

This guide is developed to facilitate the users through all stages of the CareConnect implementation and use cycle. It includes:

* Source Control: Details on managing code versions using Git.
* Software Installation: System requirements and installation instructions.
* Preparation for Use: Configurations for backend services and environment variables.
* Troubleshooting: Common issues and solutions for deployment.
* Post-Deployment Tasks: Verification steps and monitoring instructions.

## 1.4 Project Documents

This guide is one of several key documents in the CareConnect project. These documents are integral to understanding the overall system and ensuring that the development aligns with the project’s goals. The table below highlights the documents within the entire document package of CareConnect.

**Table 1**

*CareConnect Documentation Table*

| **Document** | **Version** | **Date** |
| --- | --- | --- |
| Project Plan | 1.0 | 05/31/2025 |
| Software Requirements Specification | 1.0 | 05/31/2025 |
| Technical Design Document | 1.0 | 06/14/2025 |
| Software Test Plan | 1.0 | 06/14/2025 |
| Programmer’s Guide | 1.0 | 07/26/2025 |
| Deployment & Operations Guide | 1.0 | 07/26/2025 |
| User Guide | 1.0 | TBD |
| Test Report | 1.0 | TBD |

***Note***.Documents with the date labeled TBD have not been delivered yet.

## 1.5 Definitions, Abbreviations, and Acronyms

* Caregiver: A user who manages health and care activities of a patient.
* Patient: A user as a recipient of care and organization of his/her healthy activities.
* API: Application Programming Interface, employed to connect CareConnect with the outside services.
* AWS: Amazon Web Services, the cloud infrastructure used for hosting CareConnect's backend services.
* RDS: Relational Database Service, AWS service to provision your database instance.
* IaC: Infrastructure as Code
* S3(SSS): Simple Storage Service
* SNS: Simple Notification Service, used to send messages and notify someone or an application based on subscriptions.
* Sate Machine: Instance of a Step Function
* API Gateway = AWS Service that help exposing HTTP Endpoint securely to the internet.
* Amplify: AWS Service that hosts website and deploy many type of applications. It brings simplicity but has a lot of limitations in terms of maintenance since almost all underlying resources are provisioned.
* HIPAA: Health Insurance Portability and Accountability Act, governing the privacy and security of health data.

## 1.6 References

Flutter documentation. (n.d.). https://flutter.dev/docs

AWS Documentation Overview. (n.d.). *Amazon Web Services, Inc*. https://aws.amazon.com/documentation/

# 2. Source Control

## 2.1 GitHub Access

The CareConnect project is managed using Git and hosted on GitHub under the following branch: <https://github.com/umgc/summer2025/tree/care-connect-develop>

This is the main development branch where all active work on the front-end, back-end, and infrastructure is integrated. Developers should use this branch as the base for creating feature branches and submitting pull requests.

The production branch of the CareConnect project can be located here: [umgc/summer2025 at care-connect-production](https://github.com/umgc/summer2025/tree/care-connect-production)

This is the branch that has a stable working build and deployment-ready code.

## 2.2 Repository Structure

CareConnect uses GitHub for source control. The repository is organized as follows:

https://github.com/umgc/summer2025/tree/care-connect-develop/careconnect2025

**Table 2**

*CareConnect Branch Names and Descriptions*

| Branch | Description |
| --- | --- |
| care-connect-production | The production branch, where stable releases are stored. |
| care-connect-develop | The development branch, where ongoing features and fixes are implemented. |
| feature/\* | Branches for individual features or bug fixes, created from dev and merged back after completion. |
| cc-<task #>-smallDescription | **“cc”** is abbreviated for CareConnect. This is used as a consistent project prefix in naming conventions to clearly identify that the task belongs to this project, especially when working across multiple repositories or services.  **“Task #”**: This relates to the function assigned from GitHub Project. It helps team members quickly reference and track the progress of a specific item in the development workflow  **“SmallDescription”:** Small explanation of a task. It helps maintain clarity and organization when viewing branches or commits. |

***Note****.* Table denotes branch naming style with description of each branch name.

## 2.3 Pull Request Guidelines

As you might understand, a repository provides the benefit of version control, although on top of that we agreed to use pull requests as standard for our continuous integrations. Based on the repository structure that you see above, new work would come from a contributor’s branch through a pull request, that is then submitted for review by the rest of the team but more importantly one of the leads, like a tech lead is required.

### 2.3.1 Prevent Pushing “Bad” Code

The pull request process is part of our safeguard mechanism on preventing unwanted code, code that could introduce bugs, security vulnerabilities or code that does not align with our patterns and standards for CareConnect.

## 2.4 Collaboration Methods

### 2.4.1 Teams Channel

1. Open Microsoft Teams (web or desktop)
2. Join the Joint Collab channel if invited
3. Join the “Joint Collab Care Connect” channel if invited
4. Use the Posts tab for discussions and announcements
5. Use the Files tab to access shared documents and folders
6. Tag team members with the @name to notify them

### 2.4.2 Meetings

1. Open Microsoft Teams-Use the desktop app or go to [https://teams.microsoft.com](https://teams.microsoft.com/)
2. Go to the Channel- Click on the "Joint Collab (Care Connect)" channel
3. Find the Meeting Link- Go to the “Posts” tab or Calendar tab in Teams. Click on the meeting listed for the current date and time
4. Click “Join”
5. Participate-Share updates, ask questions, and take notes if needed

### 2.4.3 Emails

1. Use verified email accounts to send correspondence
2. Check emails regularly and often
3. Use clean subject lines
4. Use email for general project topics

## 2.5 CI/CD Pipeline

CareConnect's deployment is automated using GitHub Actions and other AWS services. Therefore, proper configuration on Github is necessary for these workflows. When code is pushed to the care-connect-develop branch, GitHub Actions trigger the following:

* Test: Unit and integration tests are run using Flutter’s testing framework for the front-end and Java/Maven for the backend.
* Build:
  + Front-end: The application is built for web deployments as of now. But it will be compiled also for both Android and iOS platforms for mobile.
  + Backend: An artifact is created from the release of one of our two integration branches, care-connect-develop and care-connect-production. This artifact is a zip file built and ready for AWS Lambda deployment.
* Deploy:
  + Frontend: Once the build is created successfully for the web. A zip is created and pushed to AWS. That zip can then be used to update the branch needed from Amplify.
    1. The zip file is pushed to our IaC S3 bucket.
    2. A rule from EventBridge picks up that action/event from S3 to trigger a State Machine.
    3. The State Machine start a new Amplify deployment by pushing the S3 URI to the Amplify branch specified.
    4. An email is then published by an SNS topic. The deployment is done.
  + Backend: Once the build/release is created successfully:
    1. The zip file is pushed to our IaC S3 bucket.
    2. A rule from EventBridge picks up that action/event from S3 to trigger a State Machine.
    3. The State Machine updates the lambda code and publishes a new version of the lambda.
    4. While the new version is in the creation process. We loop to check if the new published version expected is ready. This loop has a maximum ten (10) minutes timeout.
    5. If the new version is Active the State Machine updates the API Gateway integration to use the created numbered version.
    6. An email is then published by an SNS topic. The deployment is done.

# 3. CareConnect Setup

## 3.1 System Requirements

Before installing CareConnect, ensure the following system requirements are met:

* Operating System: Ubuntu 22.04 LTS, Windows 11 or macOS 12 (Monterey) or later.
* RAM: 8GB (16GB recommended).
* Disk Space: 100GB minimum.
* Cloud: AWS account credentials and secrets with ample access to services like Lambda, Step Function, IAM, VPC, RDS, S3
* Software Dependencies: Git
  + Frontend: Flutter SDK, Android SDK (to target Android platform)
  + Backend: Java JDK 17 or later, Maven (or provided by your IDE), MySQL (For development)
  + Infrastructure: Terraform

## 3.2 Software Installation

### 3.2.1 VS Code

1. Install VS Code from Visual Studio Code’s website.
2. Install the Dart and Flutter extensions from the VS Code marketplace to enable Flutter development features like IntelliSense, debugging, and Flutter-specific tooling.
3. Configure workspace settings:

* Emulator path: Set the path for the Android Emulator in the settings.json file.
* Formatter: Use Dart-specific formatters for consistent code styling.

## 3.2.2 Android Studio

1. Download and install Android Studio from Android Developers.
2. Set up Android Virtual Devices (AVDs) for testing CareConnect on emulated Android devices.
3. Install Gradle and other required SDKs.

## 3.2.3 Android Emulator

Set up an Android Emulator using Android Studio.

* Configure AVDs for multiple device types (e.g., Pixel, Nexus) to ensure cross-device testing.
* Ensure hardware acceleration (via Intel HAXM or ARM) is enabled for optimal performance.

## 3.2.4 IntelliJ

1. Download and Install IntelliJ IDEA
2. Get the Project from GitHub
3. On the welcome screen, click "Get from Version Control" or go to File > New > Project from Version Control
4. Enter the Repository Link: Paste this GitHub URL: https://github.com/umgc/summer2025.git
5. Choose a folder where you want to save the project
6. Click "Clone"
7. Switch to the Correct Branch
8. In the bottom-right corner or Git panel, select: care-connect-develop
9. If you don’t see it, click "Fetch" to update the branch list
10. Pull the Latest Code: Go to VCS > Git > Pull or use the Pull button in the Git toolbar to get the newest updates

## 3.2.5 Git

1. Install Git
   * Download from: [https://git-scm.com](https://git-scm.com/) and follow the setup steps.
2. Open Terminal or Command Prompt
3. Clone the Repository: git clone https://github.com/umgc/summer2025.gitcd summer2025
4. Switch to Development Branch: git checkout care-connect-develop
5. Pull the Latest Changes:git pull origin care-connect-develop

## 3.2.6 GitHub Desktop

1. Install GitHub Desktop- Download from: [https://desktop.github.com](https://desktop.github.com/)
2. Sign In-Log in with your GitHub account.
3. Clone the Repository, Go to File > Clone Repository:

Paste: <https://github.com/umgc/summer2025.git> Choose a save location and click Clone

1. Switch to Development Branch: Select care-connect-develop from the branch dropdown
2. Get Latest Code-Click Fetch origin or Pull origin

## 3.2.7 Flutter + Dart

1. Install Dart SDK and Flutter SDK from Flutter’s official site.
2. Run flutter doctor to check for missing dependencies.
3. Install necessary packages:
4. flutter pub get

## 3.2.8 AWS Account

1. Get Access: Ask the project lead to give you IAM (AWS login) access.
2. You will get a username, password, and maybe access keys.
3. Login to AWS Console
4. Go to: <https://aws.amazon.com/console>
5. Use your IAM login to sign in.
6. Use the AWS CLI (Command Line)
7. Install AWS CLI: <https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2.html>
8. Run this to connect: aws configure
9. Enter your access key, secret key, region , and output format .
10. Deploy with Terraform -Go to the Terraform folder (e.g., /test/terraform\_aws)
11. Run:

* terraform init
* terraform plan
* terraform apply

## 3.2.9 AWS Command Line Interface

1. Install the AWS CLI
   * Verify the Installation
2. Open a terminal and run: aws –version
3. You will be asked to enter: AWS Access Key ID, AWS Secret Access Key. Default region Default output format (press Enter for json)

## 3.2.10 Terraform

Use Terraform to manage infrastructure provisioning in AWS. Ensure the following:

* Install Terraform from Terraform’s website.

Configure the AWS provider in the Terraform files to manage resources such as S3, RDS, and Lambda functions.

## 3.2.11 MySQL Database Engine

1. Setup MySQL (Local or Cloud)

You have two options:

Local Development:

Install MySQL locally (use MySQL Installer).

Create a database: CREATE DATABASE careconnect;

Or:

AWS Deployment:

Go to Amazon RDS.

Create a new MySQL 8.x instance.

Enable public access if needed for dev/testing.

Set DB name (e.g., careconnect), username, and password.

2. Add MySQL Connection Info to Your Project

In Spring Boot, open your application.properties ensure you at least have these properties

careconnect.db.url=${JDBC\_URI}

careconnect.db.username=${DB\_USER}

careconnect.db.password=${DB\_PASSWORD}  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update

* We use environment variables to improve the security and integration since some properties are environment specific.
* Provide the values for the environment variable. Replace JDBC\_URI your jdbc:mysql://<HOST>:<PORT>/careconnect, DB\_USER with your <YOUR\_USERNAME>, and DB\_PASSWORD with your <YOUR\_PASSWORD> from your MySQL setup.
* To deploy on AWS Lambda and use the RDS instance please set the values respectively as they will be translated with SSM Parameter Store, when the app is initializing.

3. Run Your Backend

Once MySQL is running and credentials are in place:

./mvnw spring-boot:run

The app will connect to MySQL and automatically create necessary tables.

4. Tools You Can Use

MySQL Workbench – for visual database access

DBeaver – cross-platform SQL tool

AWS RDS Console – monitor production DB

## 3.3 Program Configurations

### 3.3.1 Third-Party APIs

#### 3.3.1.1 Stripe

Step 1: Create a Stripe Account

1. Go to [https://stripe.com](https://stripe.com/)
2. Click Start now or Sign up.
3. Complete the business verification process.

Step 2: Get API Keys

1. Go to Developers > API Keys in the Stripe Dashboard.
2. Copy the following:
   1. Publishable key (for frontend)
   2. Secret key (for backend)
3. Store them securely:
   1. STRIPE\_SECRET\_KEY → in backend .env or application.properties
   2. STRIPE\_PUBLISHABLE\_KEY → in frontend environment config

Step 3: Add Stripe to Backend

Step 4: Create Payment Endpoint

Step 5: Add Stripe to Frontend (Flutter Example)

Step 6: Test with Stripe Test Cards

Step 7: Monitor Payments

### 3.3.1.2 Google Health Connect

Step 1: Check Requirements

Must target Android API 28+ (Android 9+)

The Health Connect app must be installed from Google Play

User must give explicit permission

Step 2: Add Health Connect Dependency

In app/build.gradle:

dependencies {

implementation "androidx.health.connect:connect-client:1.1.0-alpha05" // Or latest

}

Step 3: Request Permissions in Code

Step 4: Read Health Data

After permissions are granted, read data like this:

val response = healthConnectClient.readRecords(Steps::class)

for (record in response) {

Log.d("CareConnect", "Steps: ${record.count}")

}

Step 5: Write Health Data

To log new data (e.g., custom step entry):

val stepsRecord = Steps(

count = 1000,

startTime = Instant.now().minus(1, ChronoUnit.HOURS),

endTime = Instant.now()

)

healthConnectClient.insertRecords(listOf(stepsRecord))

Step 6: Test on Supported Devices

Step 7: Handle Denied Permissions

### 3.3.1.3 DeepSeek

1. Choose Access Method
   1. Option 1: Local or lightweight hosted model
   2. Option 2: API call to a deployed DeepSeek endpoint (e.g., AWS Lambda or container)
2. Prepare Input
   1. Format user or health data into a simple text prompt:

{  
 "prompt": "Summarize this health data: sleep 6 hours, heart rate elevated for 3 days, medication missed twice.",  
 "max\_tokens": 100  
}

1. Call DeepSeek Endpoint (Sample Python Logic)
   1. import requests  
      response = requests.post("https://api.deepseek.example.com/generate", json={  
       "prompt": "Summarize patient: 3 days elevated HR, missed meds.",  
       "max\_tokens": 80  
      })  
      print(response.json()["output"])
   2. In production, this is handled within a Spring Boot microservice and wrapped in a secure Lambda call.
2. Receive and Display Output
3. Security & Compliance
   1. Data sent to DeepSeek is anonymized (no names or IDs)
   2. Not used for permanent storage—only real-time inference
   3. Access is logged and encrypted in transit

### 3.3.1.4 OpenAI

1. Create an OpenAI Account
   1. Visit https://platform.openai.com
   2. Sign in and create an API key under your user settings
   3. Copy the API key (keep it secure)
2. Choose a Model
   1. Common models used:
   2. gpt-3.5-turbo – For fast, general-purpose reasoning
   3. gpt-4 – For deeper, more nuanced understanding
3. Send a Request to OpenAI
   1. Use a backend service (like Spring Boot) to call the OpenAI API.
4. Use the Response
   1. The response from OpenAI will contain a summarized or generated message that you can:
   2. Display to caregivers
   3. Store in a recommendation log
   4. Trigger a notification
5. Secure Your Integration
   1. Store the API key in environment variables or secure config files
   2. Limit usage to only necessary prompts to control costs
   3. Monitor usage via OpenAI dashboard

### 3.3.1.5 Apple Health

1. Enable HealthKit
   1. Open your Xcode project.
   2. Go to your app target > Signing & Capabilities.
   3. Click + Capability, and add HealthKit.
2. Request Authorization
3. Read Health Data
4. Display or Sync to CareConnect Backend
   1. Collected data is sent securely to your backend using HTTPS.
   2. You can use a Flutter method channel if integrating with Dart code.

### 3.3.1.6 Fit Bit

1. Create a Fitbit Developer Account
   1. Go to: https://dev.fitbit.com
   2. Sign in with a Fitbit account
   3. Create a new app (click Manage > Register an App)
2. Register Your App
   1. Fill in required fields:
   2. OAuth 2.0 Application Type: Server
   3. Callback URL: e.g., https://your-backend.com/fitbit/callback
   4. OAuth 2.0 Scopes: Choose based on data needs: activity, heartrate, sleep, profile
   5. Save your Client ID and Client Secret
3. Implement OAuth 2.0 Flow
4. Retrieve Data Using Fitbit API
5. Sync with Backend

### 3.3.1.7 OpenFDA

1. Choose a Relevant OpenFDA Endpoint
   1. Examples include:
   2. Drug Labeling: <https://api.fda.gov/drug/label.json>
   3. Drug Adverse Events: <https://api.fda.gov/drug/event.json>
   4. Drug NDC (Product Code): <https://api.fda.gov/drug/ndc.json>
2. Send an API Request
3. Parse and Use the Data
4. Display or Alert in UI

## 3.4 Compiling CareConnect

### 3.4.1 Front-End

#### 3.4.1.1 Environment Variables

Configure environment variables for the CareConnect application:

* **API Keys**: Secure API keys for third-party services (e.g., Stripe, Firebase).
* **Database URLs**: Ensure correct database URLs for Amazon RDS, DynamoDB, and S3.

Example .env file: API\_KEY=api\_key DATABASE\_URL=database\_url

### 3.4.2 Back-End

CareConnect integrates with multiple backend services:

* **AWS RDS:** Stores relational data such as user profiles and patient records when deployed to AWS. RDS is not accessible outside of the AWS network created for CareConnect.
* **AWS S3:** Used for storing media files (e.g., images, reports).
* **Stripe API:** Manages billing and subscriptions for caregivers.

#### 3.4.2.1 Environment Variables

Configure environment variables for the CareConnect application:

* **API Keys**: Secure API keys for third-party services (e.g., Stripe, Firebase).
* **Database URLs**: Ensure correct database URLs for Amazon RDS, DynamoDB, and S3.

Example .env file: API\_KEY=api\_key DATABASE\_URL=database\_url

## 3.5 Deployment to AWS

This section explains how to deploy CareConnect manually; the other option is the automation of the CI/CD pipelines. To achieve the deployment, you can follow the README.md(s) under the project. However, we will provide the main steps here. You will start with creating the resources using Terraform. A prerequisite of this process is having AWS CLI setup on your computer see 3.2.9 AWS Command Line Interface.

1. Create the infrastructure

* General note on the variable: You can see what variables are needed for each application inside of the *variables.tf* file at the root level of the application, at the same level with the *main.tf*.
* Update the backend S3 bucket in all the other terraform applications: general, db, compute; to match the name of the bucket you are creating in the s3\_tfsate application.
  1. Create the IaC S3 bucket

That S3 bucket will be used for the IaC overall. I will store the Terraform states, it will archive the artifacts of frontend and the backend builds. We will use your account number for uniqueness, since S3 bucket name is unique across AWS.

* + 1. Go inside the *careconnect2025/terraform\_aws/s3\_tfstate/* folder.
    2. Execute the terraform commands to create the S3 bucket. You will have to pass a variable file to the command.
  1. Create the general resources

That is the second terraform application in the project that will create most of the infrastructure including the networking (VPC, Security Groups, Subnets…), the Database instance, the IAM role and an internal storage S3 Bucket to list those only.

* + 1. Go inside the *careconnect2025/terraform\_aws/general/* folder.
    2. Go to the main.tf Update the S3 bucket name to match the one you created from the s3\_tfstate folder.
    3. Execute the terraform commands to create the underlying common resources. You will have to pass a variable file to the command.
  1. Create the Database

There is a separate terraform application for the creation of the RDS instance. This isolation helps to reduce the dependency between the rest of the resources. To share data across those multiple applications we use the *terraform remote state* to inject the outputs of other applications. As an example, the security group of the RDS was created in the general app it only allows ingress from the security group of the compute (Lambda) resources. That is where remote state comes into play.

* + 1. Go to the *careconnect2025/terraform\_aws/db/* folder.
    2. Go to the main.tf Update the S3 bucket name to match the one you created from the s3\_tfstate folder.
    3. Execute the terraform commands to create the RDS instance along with secure Parameter Store resources that hold credentials and the JDBC URL that will be fetched by Spring Boot app. It will also at a policy that allow the IAM APPROLE access to the created parameters. You will have to pass a variable file to the command.
  1. Create the compute  
     This terraform application will create a Lambda Function along a new published version. It also creates the routes and integration to the API Gateway that will the published version. Since SnapStart is enabled, it is recommended to have a published version to take advantage of it.
     1. Go to the *careconnect2025/terraform\_aws/compute/* folder.
     2. Go to the main.tf Update the S3 bucket name to match the one you created from the s3\_tfstate folder.
     3. Execute the terraform commands to create the Lambda and related resources form application. You will have to pass a variable file to the command. Remember to include all the necessary environment variables for the Lambda, some are pass automatically from other resources.

1. Deploy the front-end.
   1. Ensure you have the environment variables needed for the front-end in place. Some of these are, the CC\_BASE\_URL\_WEB, OPENAI\_API\_KEY, DEEPSEEK\_API\_KEY and deepSeek\_uri. Normally you would put them in a “*.env*” file inside of the *careconnect/frontent/*. CC\_BASE\_URL\_WEB and other CC\_BASE\_URL that are expected should be the base URL of your HTTP API endpoints.
   2. As of right now the process is to go on the AWS console, go to the Amplify app in the Amplify service and the Deploy updates to the branch that you want to. Our amplify app is not connected to a repository as of now, you will use a zip file of the project to deploy it. This process will be automated.
2. Deploy the backend
   1. Upload the zip file of the packaged Spring Boot app to S3. Because it is over 100MB.
   2. If you deployed the infrastructure all the way with all the terraform applications, you can ignore the rest of this section. The pipeline should automatically pick up the zip, update the Lambda and update the API Gateway integration.
   3. Copy the full Object URL that you uploaded. I will look like that: https://cc-iac-us-east-1.s3.us-east-1.amazonaws.com/cc-backend-jars/cc-pre-m3-demo-build-7-26-0.0.1-SNAPSHOT-lambda-package.zip
   4. Go to Lambda Function, choose the function and Upload from Amazon S3 location
   5. Once the lambda is updated, Go to Actions and Publish new version.
   6. Copy the version ARN
   7. Go to API Gateway, inside of your API go to the Lambda integration, Edit to add the new Function version ARN.

# 4. Troubleshooting

## 4.1 Common Deployment Errors

* **Emulator fails to boot**: Ensure Intel HAXM or ARM acceleration is enabled. Reinstall the emulator if necessary.
* **Backend 404 errors**: Ensure API endpoints are correctly deployed and the routes match those defined in the backend services.

## 4.2 Log Collection

Use Flutter’s debugging tools for frontend logs and AWS CloudWatch for backend service logs.  
 To collect logs:

* **Frontend**: Use Android Studio’s Logcat for device logs.
* **Backend**: We use Cloudwatch logs to collect the logs for the backend on AWS. We have multiple LogGroups, one for the API, one for the Lambda. That helps to better monitor and debug in case of an issue as we will be able to check multiple stages of the integration. And obviously the log level depends on the configuration on the application properties we used, and the log entries provided by the developer in the code.