Day 10 Internship Report

Contact Manager Application Deployment on AWS

Intern Name: Arsalan Sharief

Company: Signiance Technology

Role: DevOps Intern

Day: 10

Repository Link:

https://github.com/arsalan-signiance/internship-day10-11-12

(Pending, will be available on 13-02-2026)

Objective of the Day

The primary objective for today was to deploy the full-stack Contact Manager application on AWS using a production-ready architecture. This involved orchestrating multiple AWS services to ensure scalability, security, and performance. Key goals included implementing ECS Fargate for containerized backend hosting, setting up an Application Load Balancer (ALB) for traffic distribution, hosting the static frontend on S3, and utilizing CloudFront as a CDN with secure origin access via OAC.

Tasks Completed Today

1. Backend Deployment Using ECS Fargate

We successfully containerized and deployed the Flask backend application using AWS Elastic Container Service (ECS) with the Fargate launch type to eliminate server management overhead.

- Built the Docker image for the Flask backend and pushed it to Docker Hub.
- Initialized an ECS Cluster using the Fargate capacity provider.
- Defined a Task Definition with the following specifications:
  - CPU: 0.25 vCPU
  - Memory: 0.5 GB
  - Container Port: `5000`
- Configured necessary IAM roles including the Task Execution Role (for pulling images and logging) and the Task Role (for AWS service interaction).
- Launched the ECS Service and verified the backend health check endpoint: `/api/health` returned `{"status":"ok"}`.

2. Application Load Balancer (ALB) Setup

An Application Load Balancer was configured to manage incoming traffic and route it to the backend containers securely.

- Provisioned an ALB in the public subnets to accept internet traffic.
- Created a Target Group configured for IP type targets on port `5000`.
- Set the health check path to `/api/health` to ensure traffic is only routed to healthy containers.
- Attached the ECS service to the ALB and verified that the target group status transitioned to Healthy.

3. VPC & Networking Configuration

A custom Virtual Private Cloud (VPC) was designed to provide network isolation and proper routing.

- Created a custom VPC containing 2 Public Subnets and 2 Private Subnets across different Availability Zones.
- Configured the Internet Gateway for public access and a NAT Gateway to allow private instances to access the internet for updates.
- Updated Route Tables to properly direct traffic between subnets and gateways.
- Implemented Security Groups:
  - ALB SG: Allows inbound traffic on port 80 (HTTP).
  - ECS SG: Allows traffic on port 5000 only from the ALB Security Group.

4. Frontend Deployment on S3

The frontend application was prepared for cloud hosting and stored securely in Amazon S3.

- Updated frontend JavaScript code to replace `localhost` references with relative `/api` paths to support the reverse proxy configuration.
- Created a private S3 bucket specifically for frontend hosting.

- Uploaded application assets: `index.html`, `app.js`, `style.css`, and `maintenance.html`.
- Configured CloudFront Origin Access Control (OAC) to restrict direct S3 bucket access, ensuring all traffic flows through the CDN.

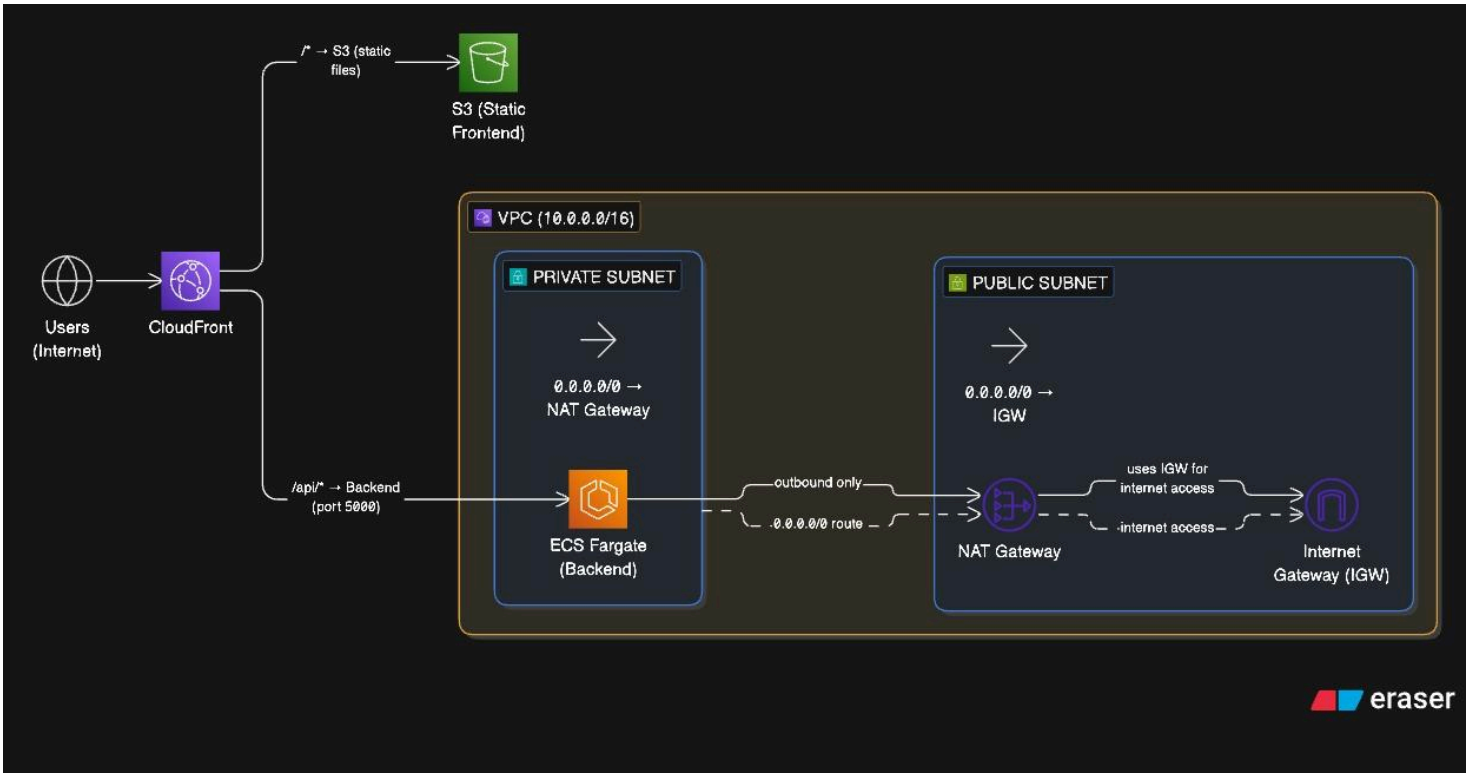
5. CloudFront CDN Configuration

CloudFront was set up as the global entry point for the application to handle caching and routing.

- Created a new CloudFront distribution with two distinct origins:
  - S3 Origin: Serves static frontend files.
  - ALB Origin: Serves backend API requests.
- Configured behavior rules:
  - Path pattern `/api/*` forwards traffic to the ALB.
  - Caching was disabled for API routes to ensure real-time data.
- Enabled automatic HTTPS redirection for security.
- Set the default root object to `index.html`.

Final Architecture Built

The following diagram illustrates the production architecture deployed today:



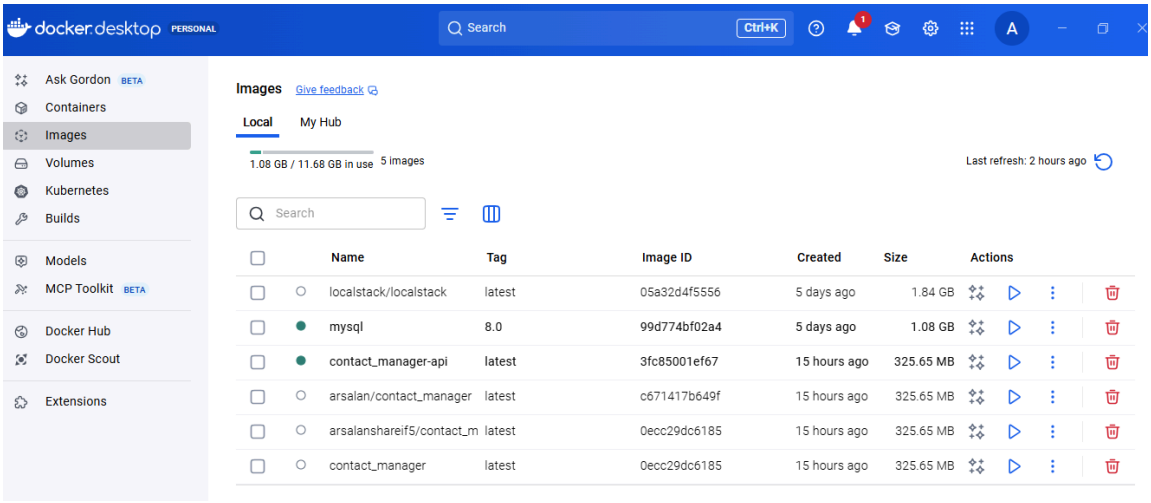
Key Learnings

- Gained practical experience in orchestrating ECS Fargate services behind an Application Load Balancer.
- Understood the critical importance of Security Groups chaining to isolate backend resources from direct internet access.
- Learned how CloudFront Behaviors effectively route traffic to different origins based on path patterns.
- Clarified the distinction between an Origin (where content lives) and a Behavior (how content is served) in CDN configurations.
- Implemented S3 security best practices using Origin Access Control (OAC) rather than enabling public bucket access.
- Solved frontend-backend integration challenges by utilizing relative API paths to avoid CORS issues in production.

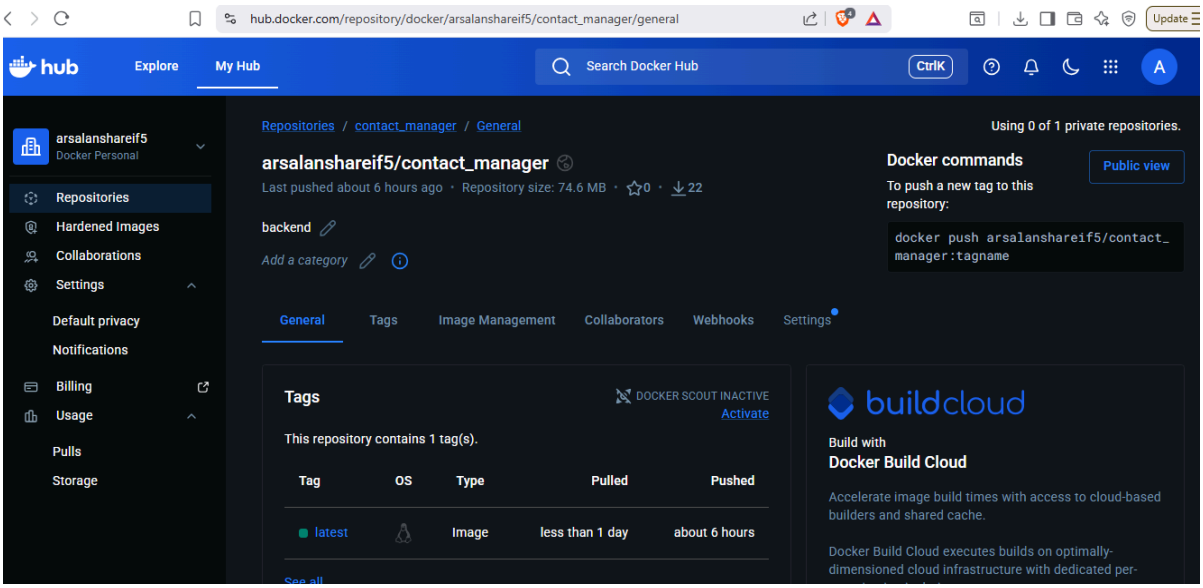
Result

- Backend successfully deployed on ECS and returning healthy status.
- Frontend securely accessible via the CloudFront domain.
- API requests successfully routed via CloudFront `/api/*` to the backend.
- Full stack application is running in a robust, production-like AWS architecture.

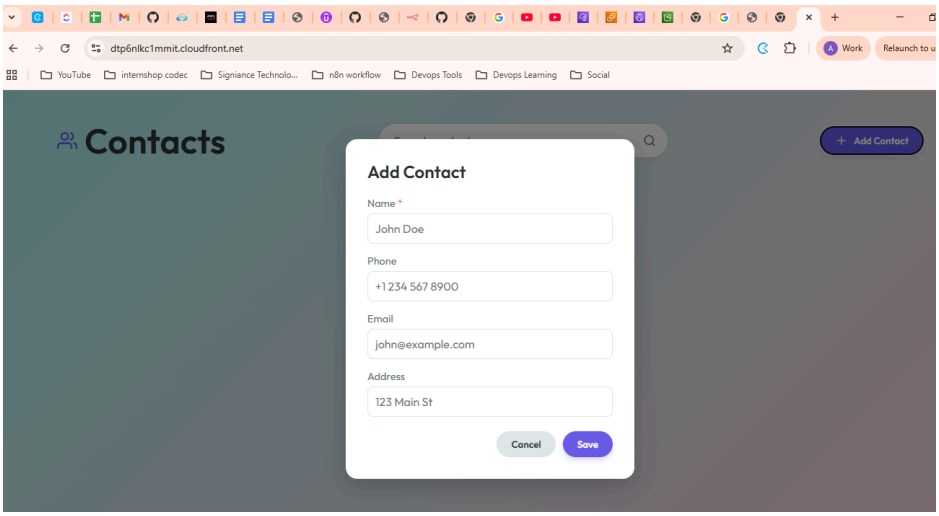
Screenshots



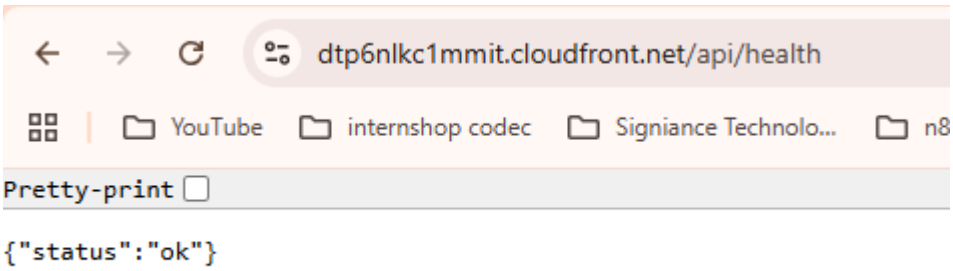
Docker Desktop interface showing the Docker images listed `contact_manager`



Docker Hub repository page for `arsalanshareif5/contact_manager`, displaying repository details, last push time, tags (latest).



Web application live using CloudFront URL



Browser window displaying a health check API endpoint (`/api/health`) returning a JSON response:
`{"status":"ok"}` indicating the backend service is running successfully.

Prepared by: ArsalanSharief
DevOps Intern – Signiance Technology