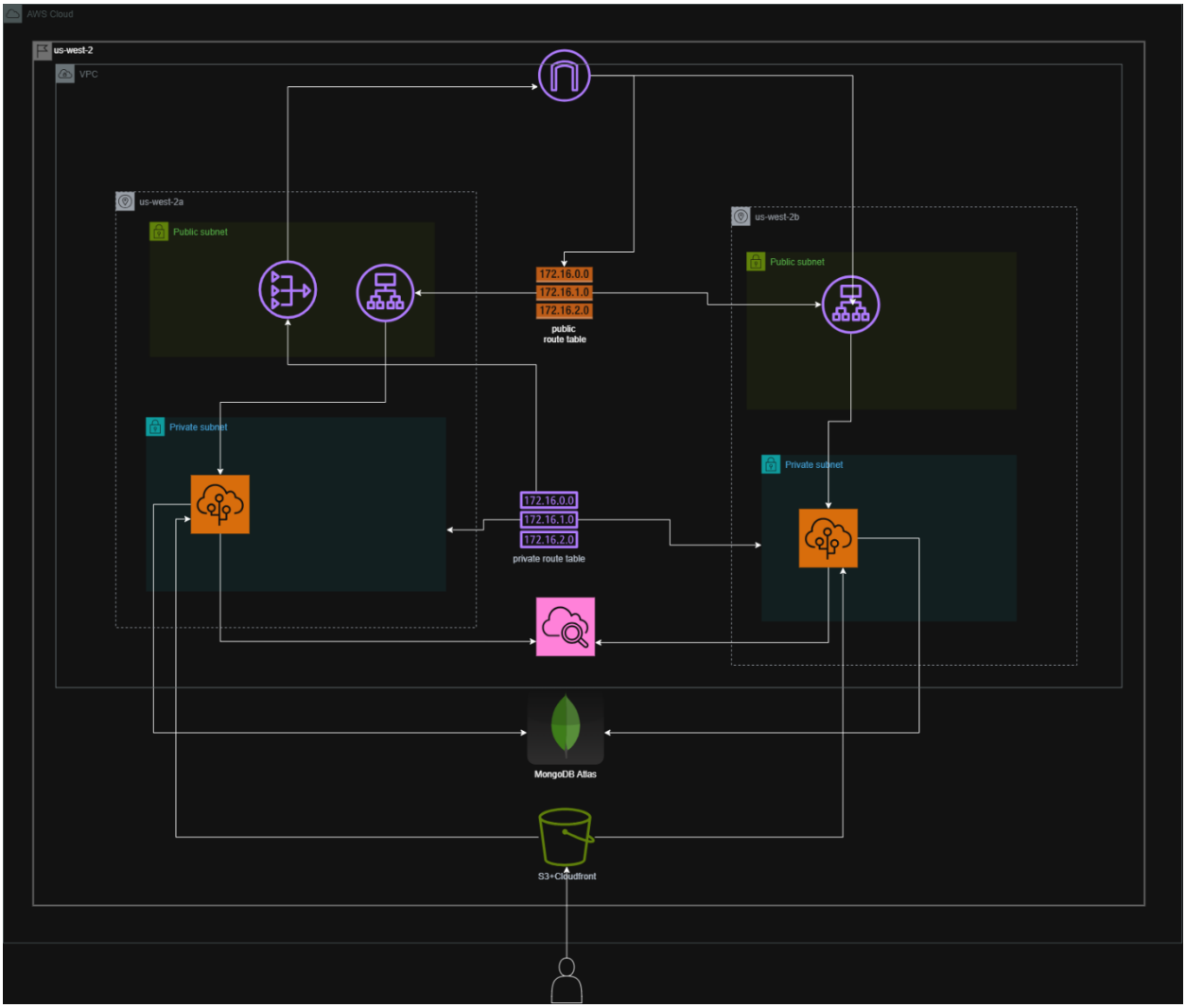
Sardar Noor Ul Hassan
Cloud Intern

Deploying a Production-Ready Node.js Application on AWS Elastic Beanstalk (Three-Tier Architecture)

1. Project Overview

This project demonstrates the end-to-end deployment of a **production-ready Node.js backend application** using **AWS Elastic Beanstalk**, integrated with a managed **MongoDB database (MongoDB Atlas)** and later extended into a **three-tier architecture** by introducing a **static frontend hosted on Amazon S3 and Amazon CloudFront**.

2. Architecture Summary (High Level)



Architecture Type: Three-Tier Architecture

Layers:

1. **Presentation Layer (Frontend)**
    - Static frontend hosted on Amazon S3
    - Delivered globally via Amazon CloudFront (HTTPS)
2. **Application Layer (Backend)**
    - Node.js REST API
    - Deployed on AWS Elastic Beanstalk (Node.js platform)

    - Exposed via Elastic Beanstalk load-balanced environment

3. **Data Layer (Database)**
    - MongoDB Atlas (managed MongoDB)
    - Secure connection via MongoDB URI stored in environment variables

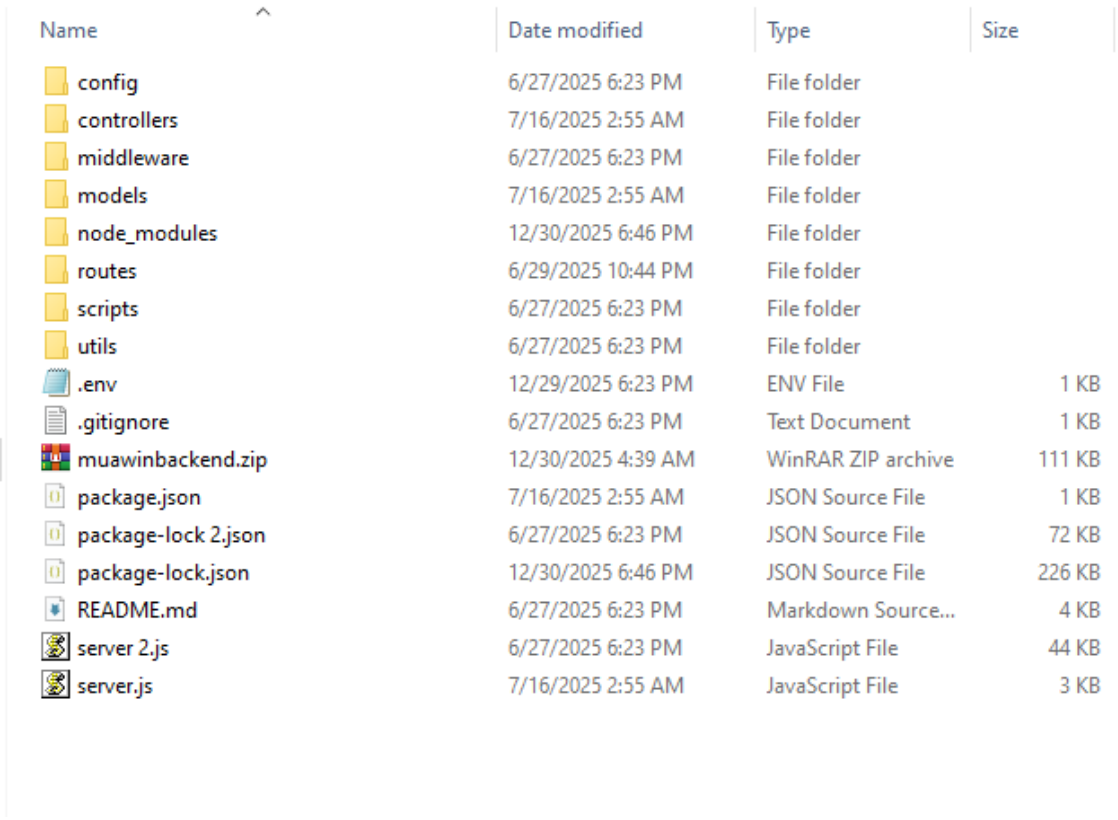
3. Prepare Node.js Application Code with Necessary Dependencies

The backend application was prepared locally with:

- Node.js runtime
- Express.js framework
- MongoDB (Mongoose) integration
- JWT-based authentication
- Modular folder structure:
    - controllers/
    - routes/
    - models/
    - middleware/
    - config/
    - utils/

All required dependencies were defined in:

- package.json
- package-lock.json



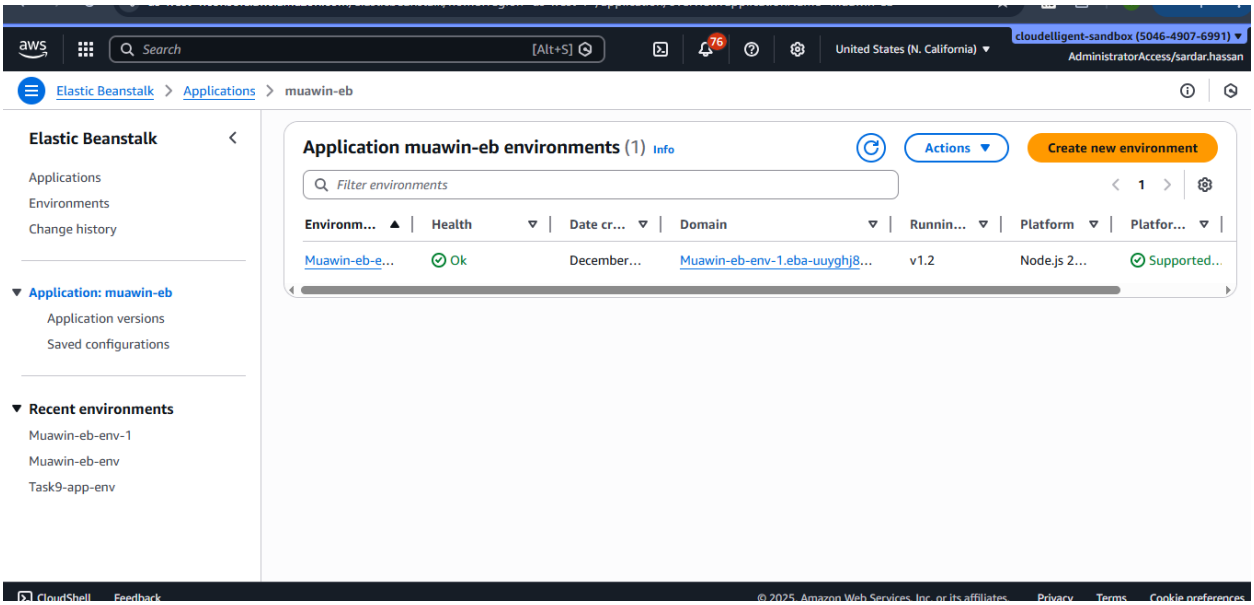
Name	Date modified	Type	Size
config	6/27/2025 6:23 PM	File folder	
controllers	7/16/2025 2:55 AM	File folder	
middleware	6/27/2025 6:23 PM	File folder	
models	7/16/2025 2:55 AM	File folder	
node_modules	12/30/2025 6:46 PM	File folder	
routes	6/29/2025 10:44 PM	File folder	
scripts	6/27/2025 6:23 PM	File folder	
utils	6/27/2025 6:23 PM	File folder	
.env	12/29/2025 6:23 PM	ENV File	1 KB
.gitignore	6/27/2025 6:23 PM	Text Document	1 KB
muawinbackend.zip	12/30/2025 4:39 AM	WinRAR ZIP archive	111 KB
package.json	7/16/2025 2:55 AM	JSON Source File	1 KB
package-lock 2.json	6/27/2025 6:23 PM	JSON Source File	72 KB
package-lock.json	12/30/2025 6:46 PM	JSON Source File	226 KB
README.md	6/27/2025 6:23 PM	Markdown Source...	4 KB
server 2.js	6/27/2025 6:23 PM	JavaScript File	44 KB
server.js	7/16/2025 2:55 AM	JavaScript File	3 KB

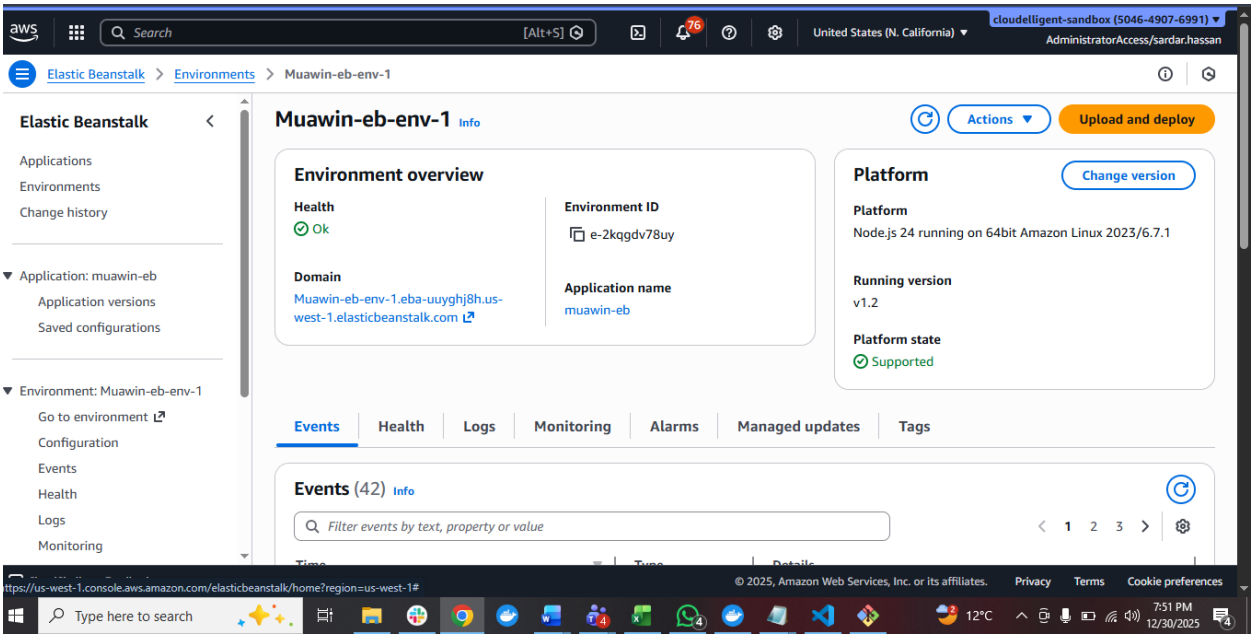
4. Create an Elastic Beanstalk Application in AWS Management Console

An Elastic Beanstalk application was created from the AWS Management Console.

Key actions:

- Navigated to Elastic Beanstalk service
- Created a new application
- Assigned a meaningful application name
- Selected **Web server environment**





5. Configure Elastic Beanstalk Environment with Platform Node.js

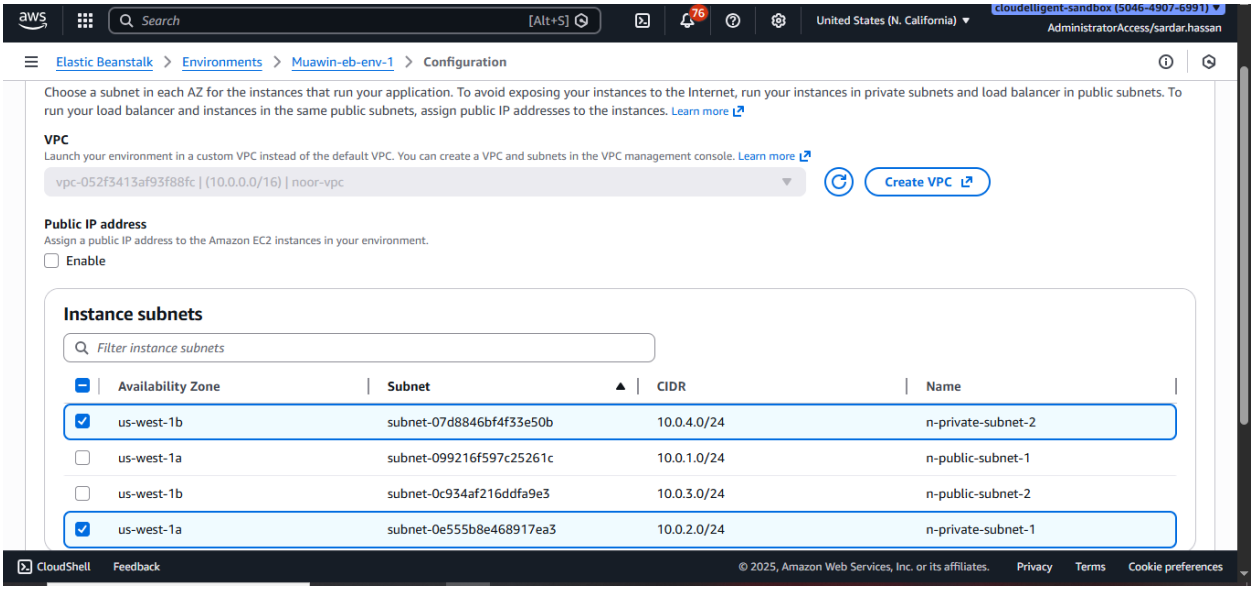
The Elastic Beanstalk environment was configured with:

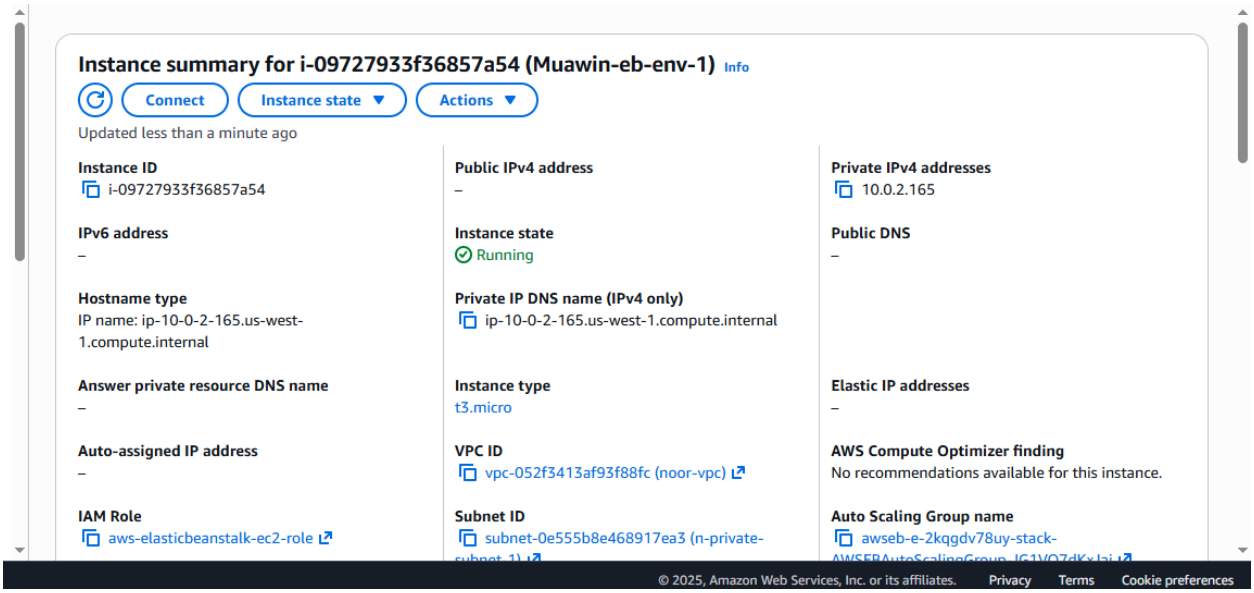
- **Platform:** Node.js
- **Environment type:** Load balanced
- **Proxy server:** Nginx

- **VPC:** noor-vpc
- **Subnets:** Private subnets for EC2 instances

Elastic Beanstalk automatically provisioned:

- EC2 instances
- Application Load Balancer
- Auto Scaling Group
- IAM roles
- Security groups



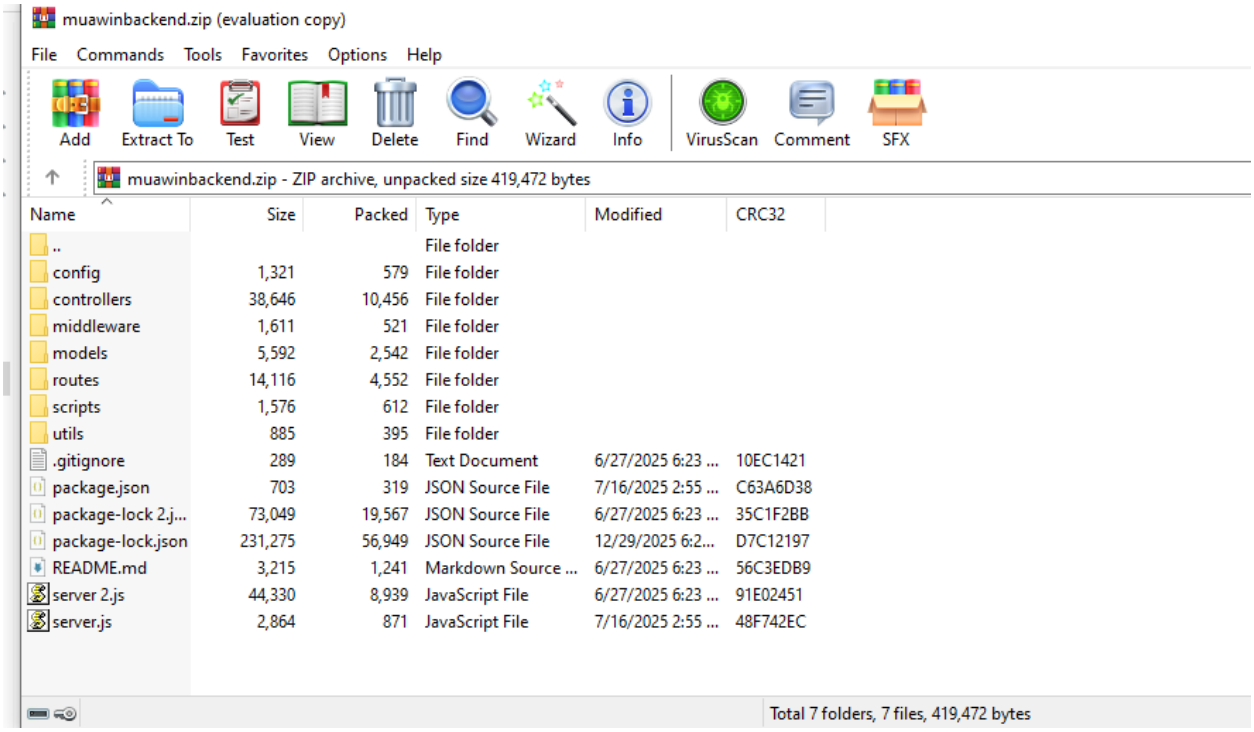


6. Package Application Code as a ZIP File for Deployment

The backend application was packaged as a ZIP archive:

- Included all application source files
- Excluded node_modules
- Included package.json and package-lock.json
- .env file was **not relied upon** for production secrets

This ZIP file became the deployment artifact for Elastic Beanstalk.



7. Upload and Deploy the Application Package to Elastic Beanstalk

The ZIP package was uploaded to the Elastic Beanstalk environment.

Elastic Beanstalk handled:

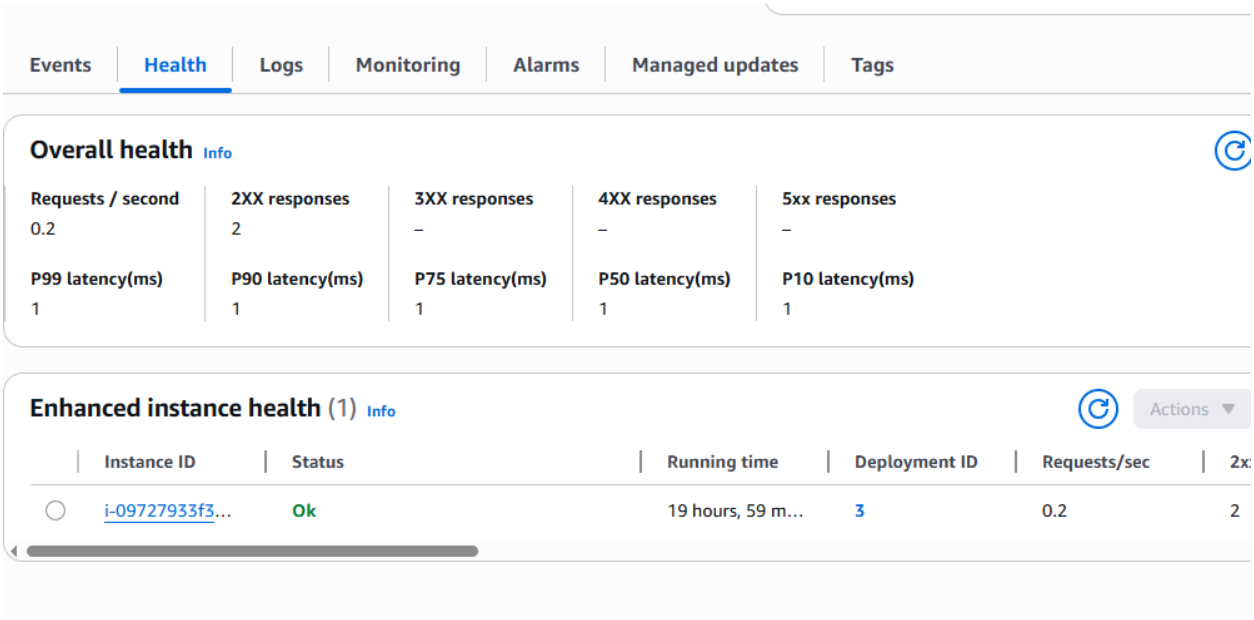
- Application extraction
- Dependency installation
- EC2 instance provisioning
- Application startup

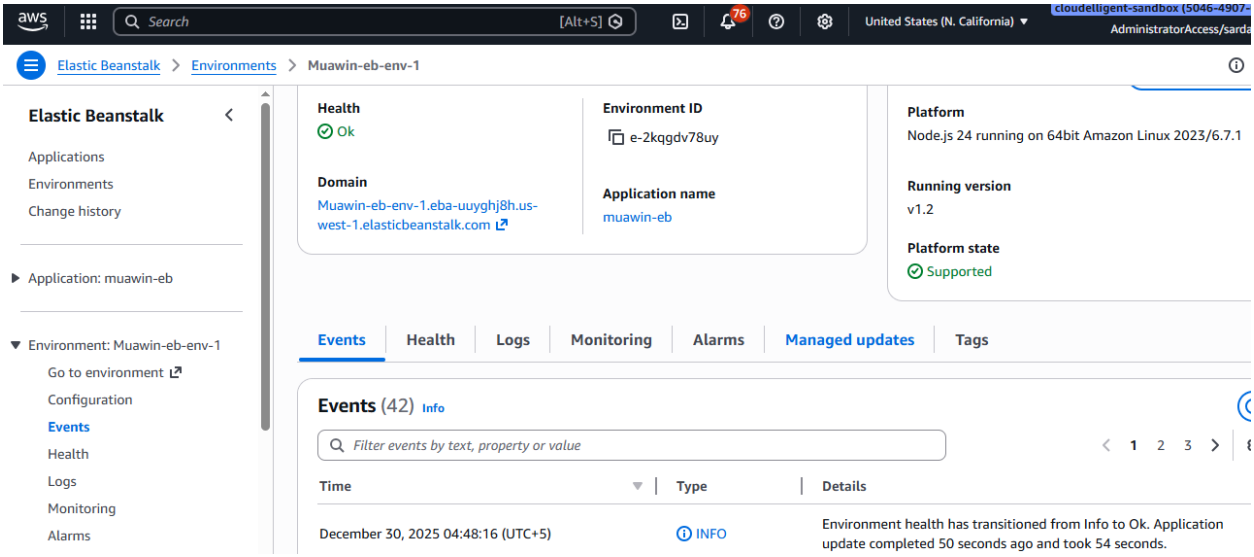
After deployment, the environment transitioned to a **Running** state.

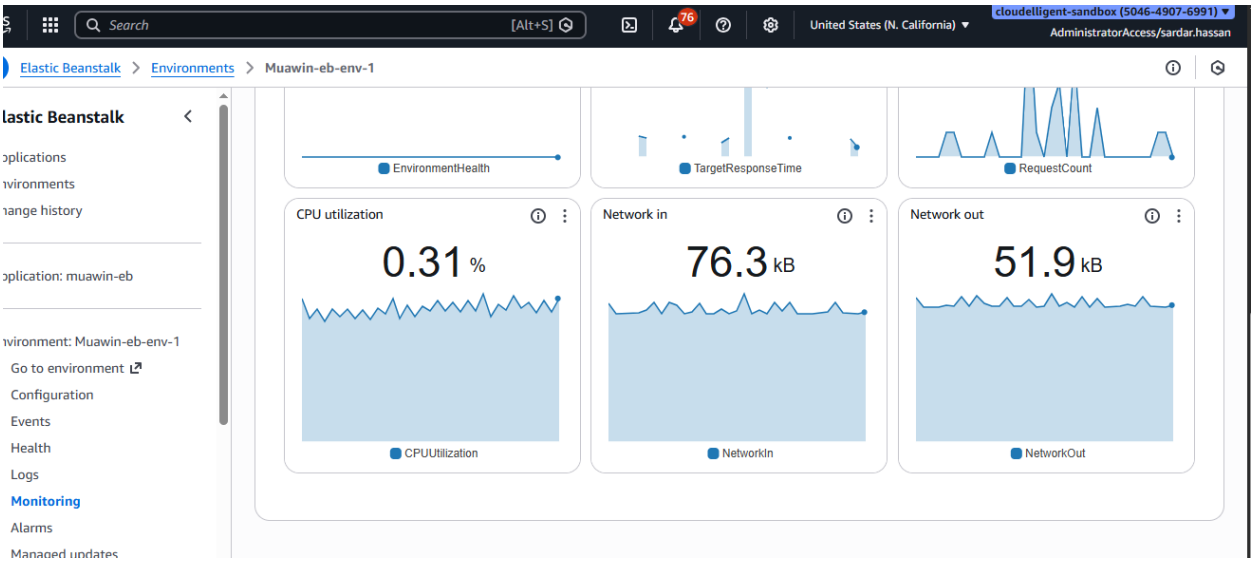
8. Monitor Deployment Status and Logs via Elastic Beanstalk Dashboard

Deployment and runtime monitoring were performed using:

- Elastic Beanstalk **Events**
- Elastic Beanstalk **Health dashboard**Application and instance logs


Events
Health
Logs
Monitoring
Alarms
Managed updates
Tags
Overall health Info
Requests / second
0.2
2XX responses
2
3XX responses
–
4XX responses
–
5xx responses
–
P99 latency(ms)
1
P90 latency(ms)
1
P75 latency(ms)
1
P50 latency(ms)
1
P10 latency(ms)
1
Enhanced instance health (1) Info
Actions
Instance ID
Status
Running time
Deployment ID
Requests/sec
2x
i-09727933f3...
Ok
19 hours, 59 m...
3
0.2
2


aws
Search
[Alt+S]
United States (N. California)
AdministratorAccess/sarda
Elastic Beanstalk
Environments
Muawin-eb-env-1
Elastic Beanstalk
Applications
Environments
Change history
Application: muawin-eb
Environment: Muawin-eb-env-1
Go to environment
Configuration
Events
Health
Logs
Monitoring
Alarms
Health
Ok
Environment ID
e-2kqgdv78uy
Domain
Muawin-eb-env-1.eba-uuyghj8h.us-west-1.elasticbeanstalk.com
Application name
muawin-eb
Platform
Node.js 24 running on 64bit Amazon Linux 2023/6.7.1
Running version
v1.2
Platform state
Supported
Events
Health
Logs
Monitoring
Alarms
Managed updates
Tags
Events (42) Info
Filter events by text, property or value
1 2 3
Time
Type
Details
December 30, 2025 04:48:16 (UTC+5)
INFO
Environment health has transitioned from Info to Ok. Application update completed 50 seconds ago and took 54 seconds.

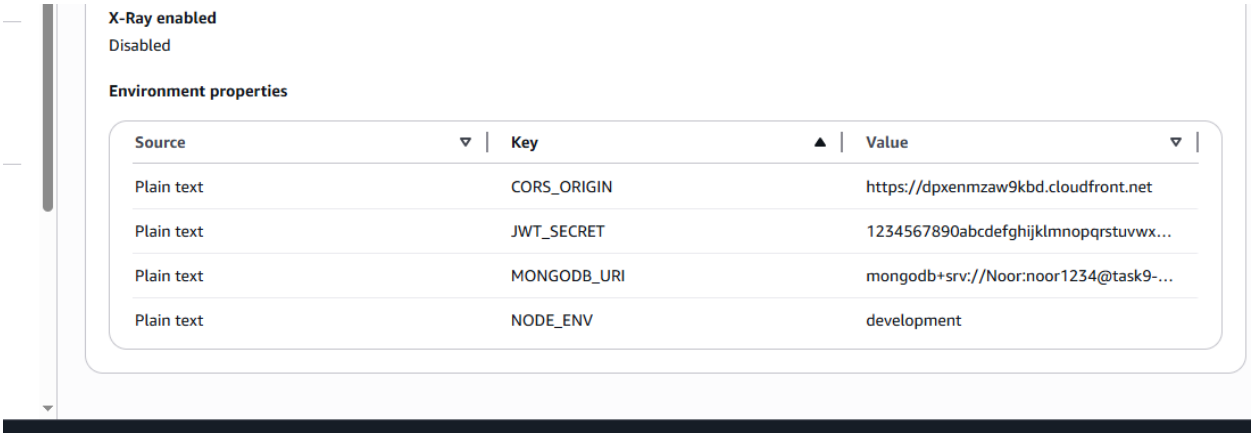


9. Configure Environment Variables and Scaling Options

Sensitive configuration values were securely configured using **Elastic Beanstalk environment properties**, including:

- MONGODB_URI
- JWT_SECRET
- NODE_ENV=production

These environment variables replaced local .env usage.

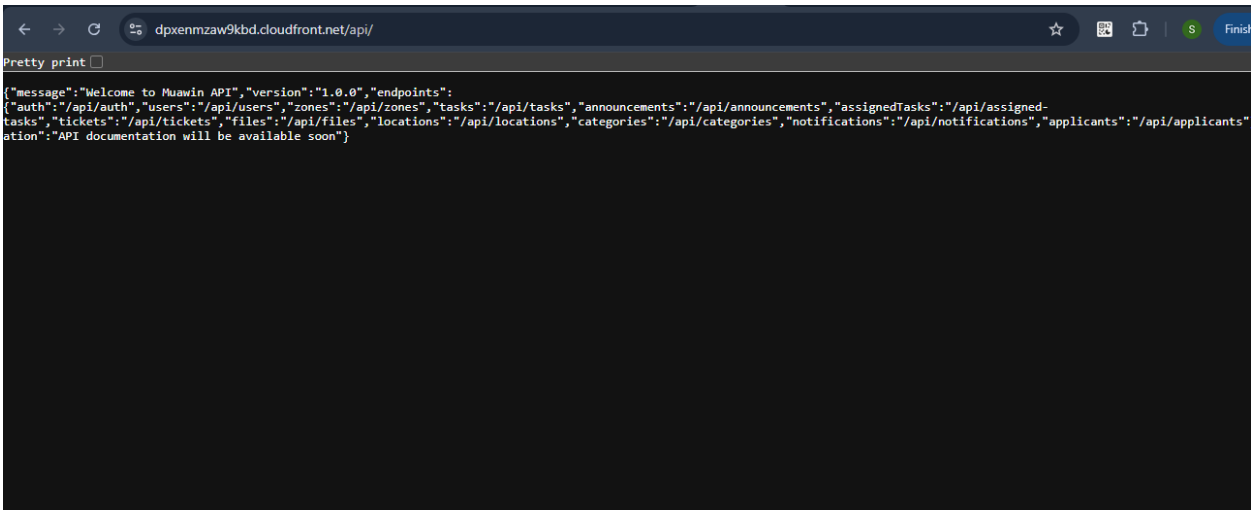


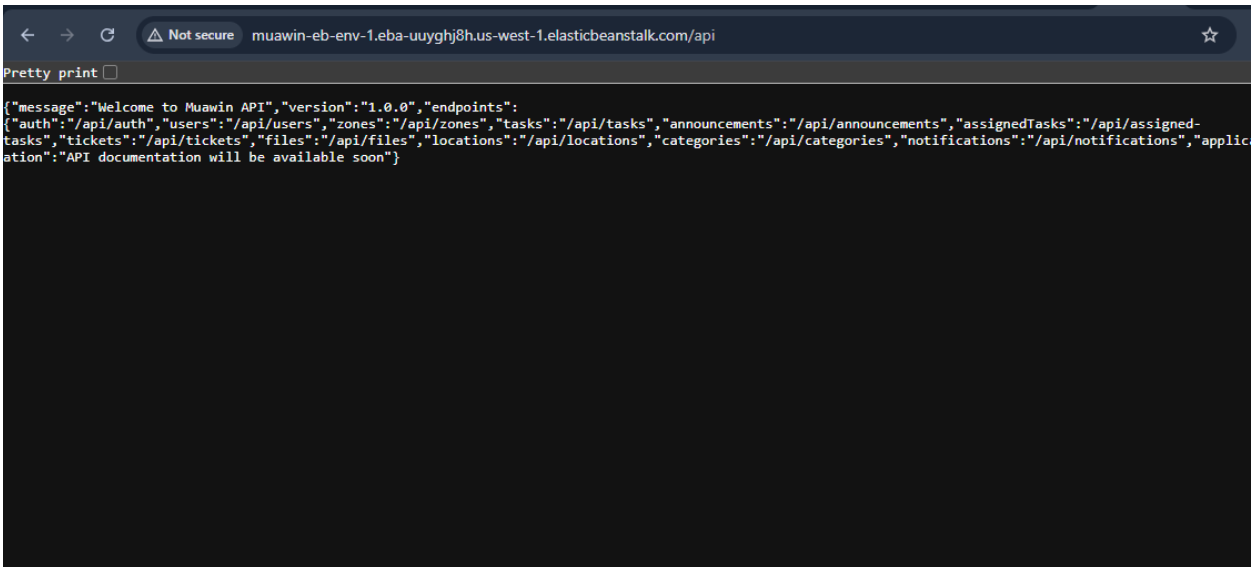
10. Test the Deployed Application URL

The backend application was tested using:

- Browser

- API endpoints

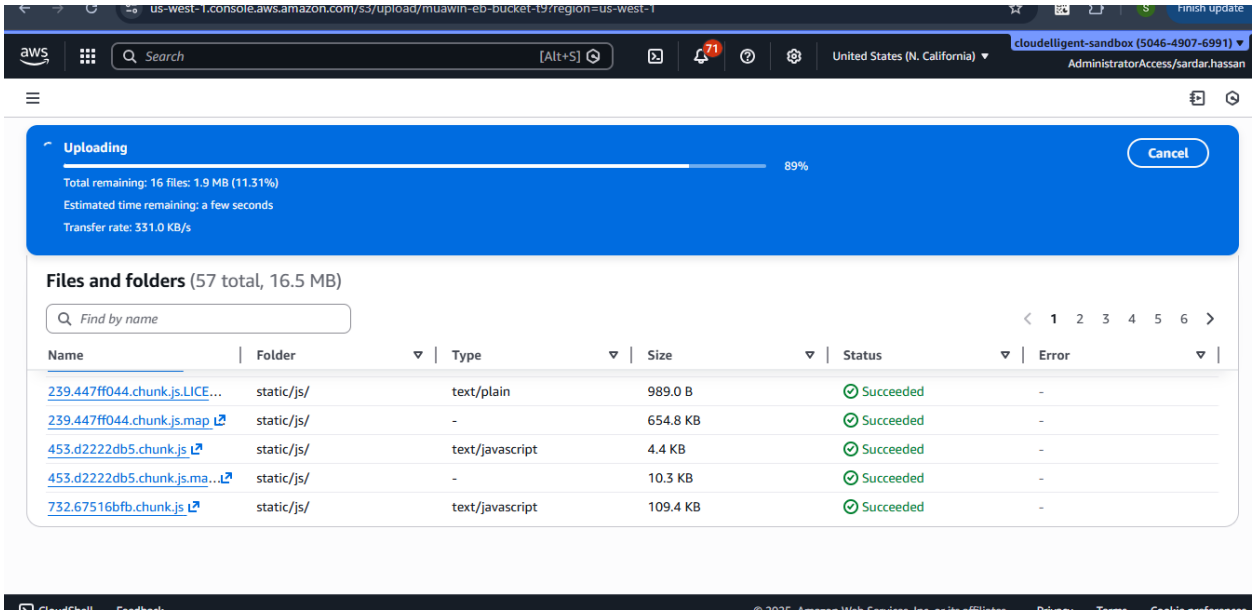


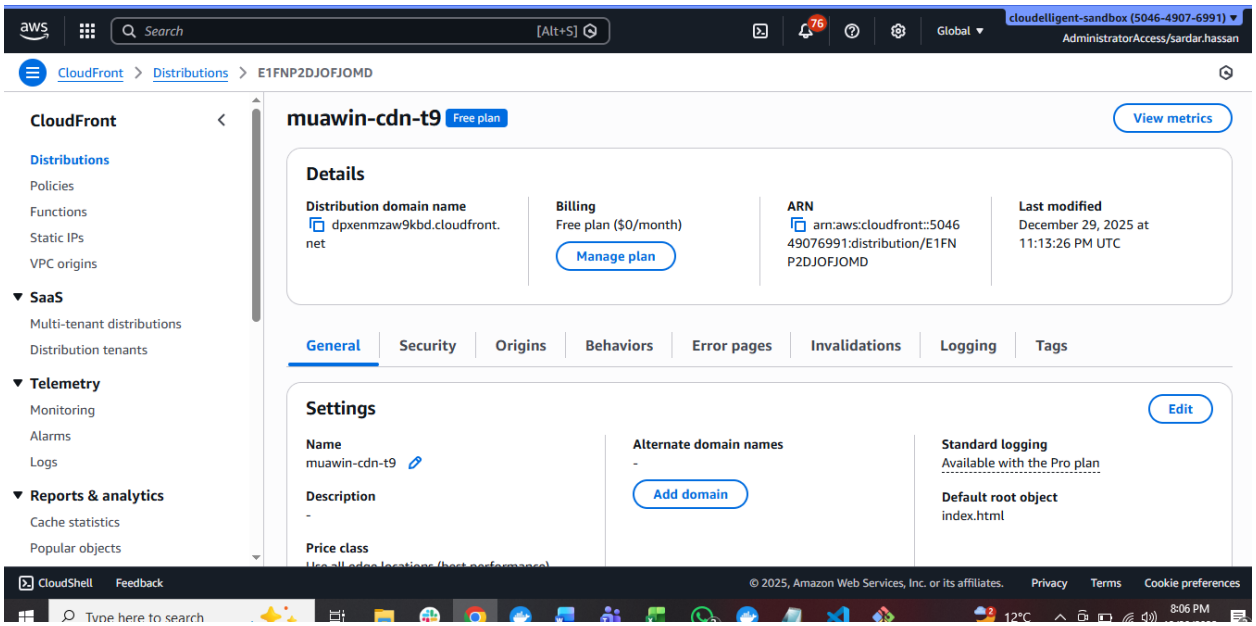


11. Frontend Deployment (Three-Tier Extension)

To complete the three-tier architecture:

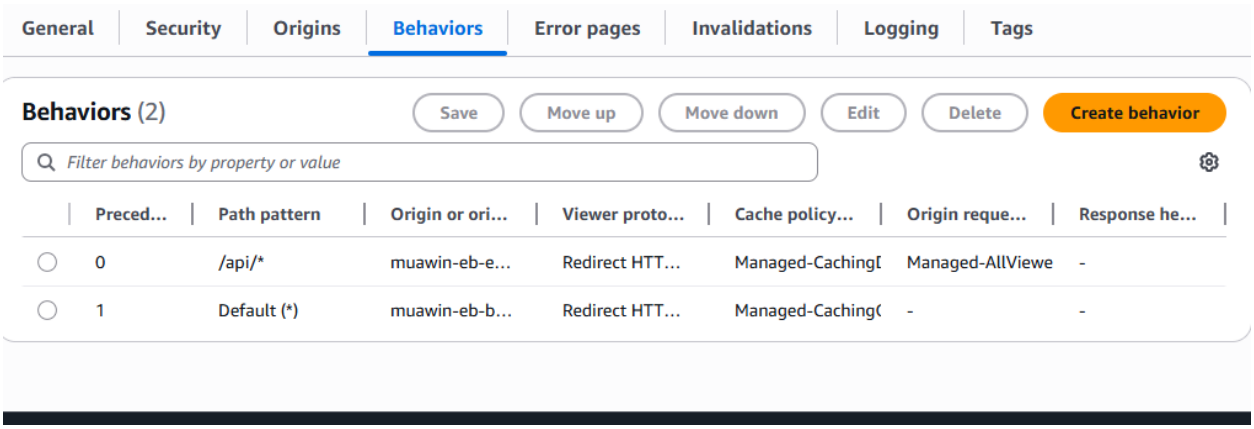
- Frontend static build files were generated
- Uploaded to an Amazon S3 bucket
- S3 static website hosting remained disabled
- Amazon CloudFront distribution was created





CloudFront Behaviors:

- Default (*) → S3 (Frontend)
- /api/* → Elastic Beanstalk backend



This enabled:

- HTTPS for frontend
- Secure routing of API calls to backend
- Elimination of mixed-content errors

12. CloudFront Cache Invalidation

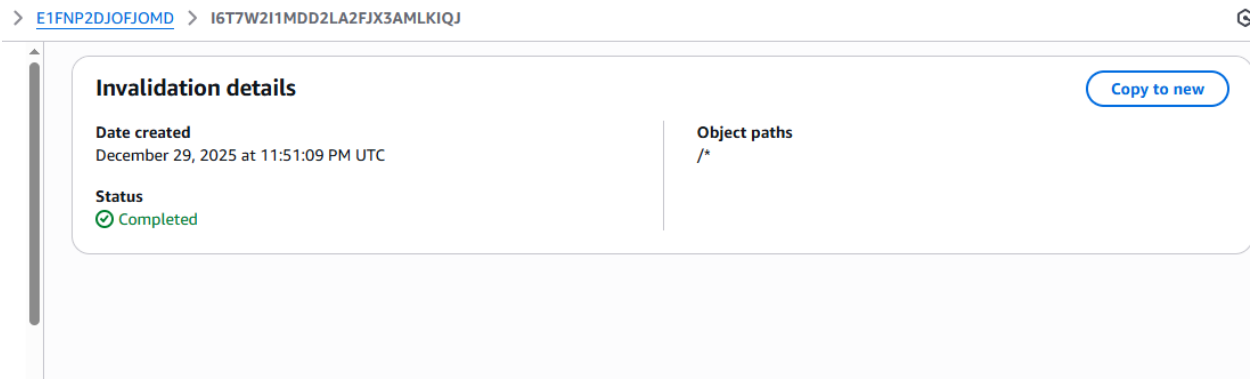
CloudFront cache invalidation was used to:

- Force refresh updated frontend files
- Clear cached assets after frontend changes

Invalidation paths used:

/*

This ensured the latest frontend build was served globally.

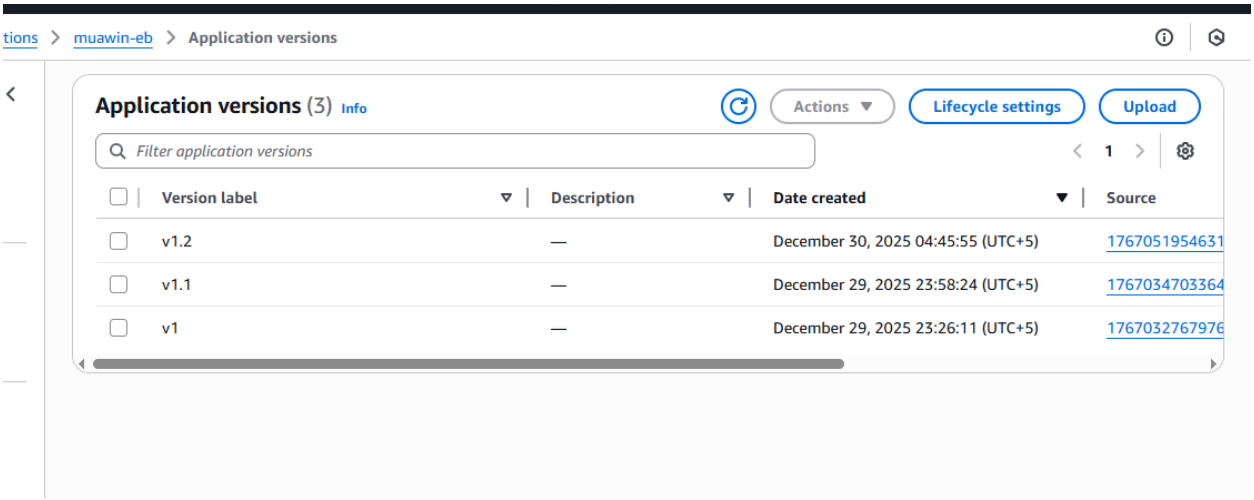


13. Update Application Code and Redeploy as Needed

Application updates followed a repeatable process:

1. Update code locally
2. Rebuild frontend or backend
3. Re-zip backend (if applicable)
4. Upload new version to Elastic Beanstalk
5. Invalidate CloudFront cache (frontend)

Elastic Beanstalk versioning allowed safe rollouts.



14. Set Up Monitoring and Alerts for Elastic Beanstalk Environment Health

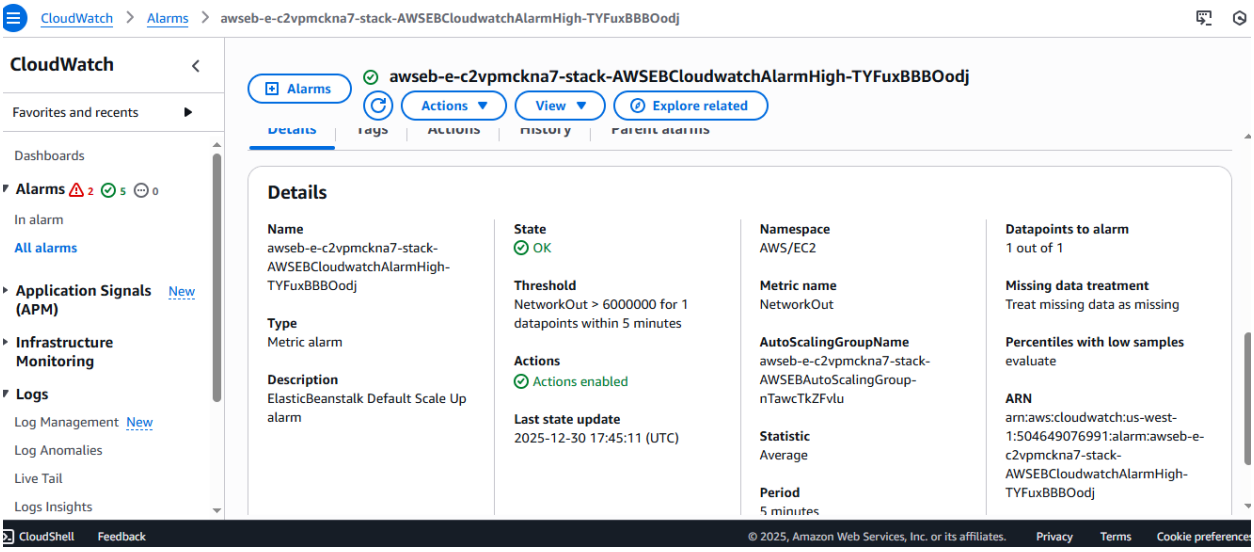
Elastic Beanstalk health monitoring was used to track:

- Instance health
- HTTP error rates
- Application responsiveness

Health transitions (OK → Severe → OK) were analyzed and resolved by:

- Fixing health check paths
- Ensuring backend root responses were valid

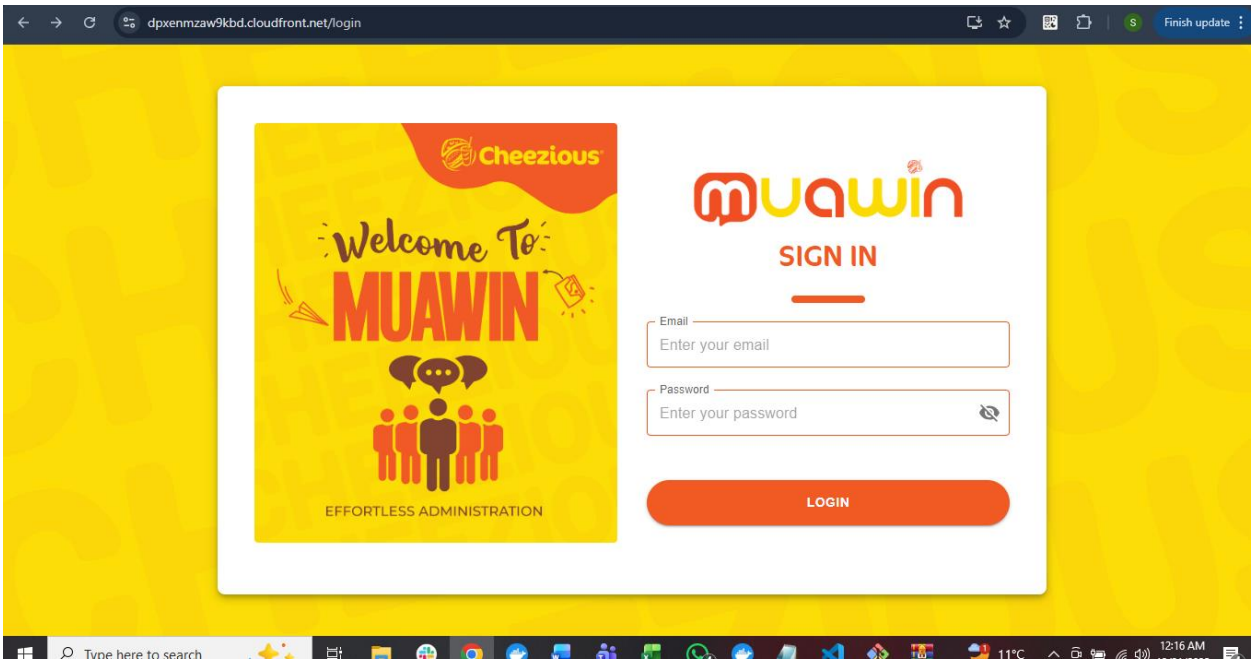
This validated production readiness.

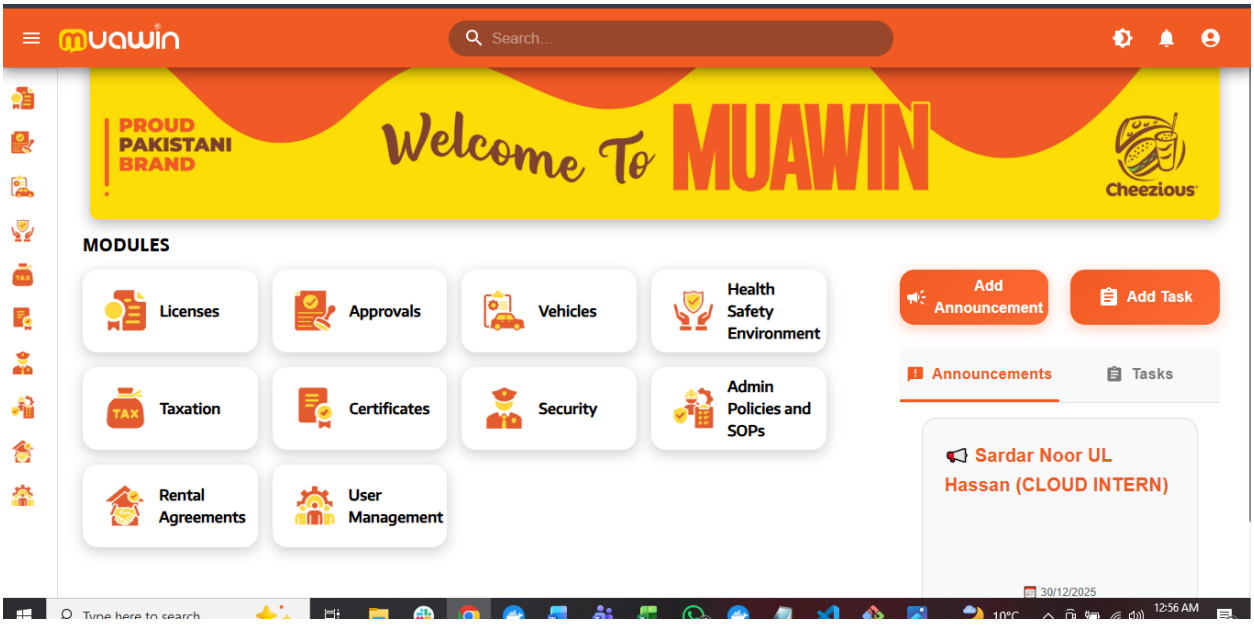


15. Final Outcome

At the end of this project:

- Backend Node.js API is live on Elastic Beanstalk
- Database is securely connected via MongoDB Atlas
- Frontend is served globally using S3 + CloudFront
- HTTPS is enforced end-to-end
- Application follows a clean three-tier architecture
- Deployment is scalable, monitored, and production-ready





16. Skills & Technologies Demonstrated

- AWS Elastic Beanstalk
- Node.js Production Deployment
- MongoDB Atlas Integration
- Amazon S3 Static Hosting

- Amazon CloudFront
- HTTPS & Security Best Practices
- Environment Variable Management

17. Conclusion

This project successfully transformed a local Node.js application into a **fully functional, cloud-hosted, production-grade system** using AWS managed services.