CLOUD COMPUTING

VenusX - AI Powered chat App

INDEX

GitHub Link:

Frontend: https://github.com/Devangi82387/frontend.git

Backend: https://github.com/Devangi82387/backend.git

Purpose:

The purpose of this project is to develop and deploy an AI-powered chat application on a secure and scalable cloud environment. The system provides real-time responses using the Gemini API, manages chat history, and ensures automated, reliable deployments through a CI/CD pipeline using GitHub Actions.

Goal:

1. Build a full-stack MERN chat application integrated with the Gemini API.
2. Deploy the application on AWS cloud (EC2, VPC, pm2, nginx)
3. Set up a fully automated CI/CD pipeline using GitHub Actions.
4. Maintain a secure and scalable deployment infrastructure.
5. Enable users to chat with AI, view history, and manage conversations.

Scope:

1. Frontend (React)

- Clean UI for chat interface
- Sidebar to display saved chat history
- Option to delete conversations
- REST API calls to backend

2. Backend (Node.js + Express)

- API to send user prompts to Gemini
- API to store, retrieve, delete chat history
- Middleware for handling API errors
- Secure environment variable management

3. Database (MongoDB)

- Store chat sessions and messages
- Timestamped history management
- Uses mongoDB atlas cloud database

4. Cloud Infrastructure (AWS)

- EC2(t3.micro) instance hosting backend and frontend
- Custom VPC for secure networking
- Security groups for controlled inbound/outbound access
- Nginx/PM2 for process & server management

5. CI/CD Pipeline (GitHub Actions)

- Auto-deploy code to EC2 instance on push/merge
- Install dependencies
- Build frontend
- Restart backend service automatically
- Full end-to-end automation

6. AI Integration

- Connect backend with Gemini API
- Parse and store AI responses

Key features:

1. AI Chat Using Gemini API

Provides intelligent, real-time responses to user queries through natural language processing using the Gemini API.

2. Conversation History Management

Maintains all past chats in a sidebar, allowing users to easily revisit, continue, or delete individual conversation sessions.

3. MERN Full-Stack MVC Architecture

Built using React, Node.js, Express, and MongoDB to deliver a scalable, responsive, and maintainable full-stack application.

4. Cloud Deployment on AWS

- Hosted on EC2 instance
- Database connection through secure VPC
- Nginx reverse proxy setup
- PM2 for backend process management

5. Secure Cloud Networking

- Custom VPC
- Public and private subnets
- Security groups controlling SSH/HTTP/HTTPS

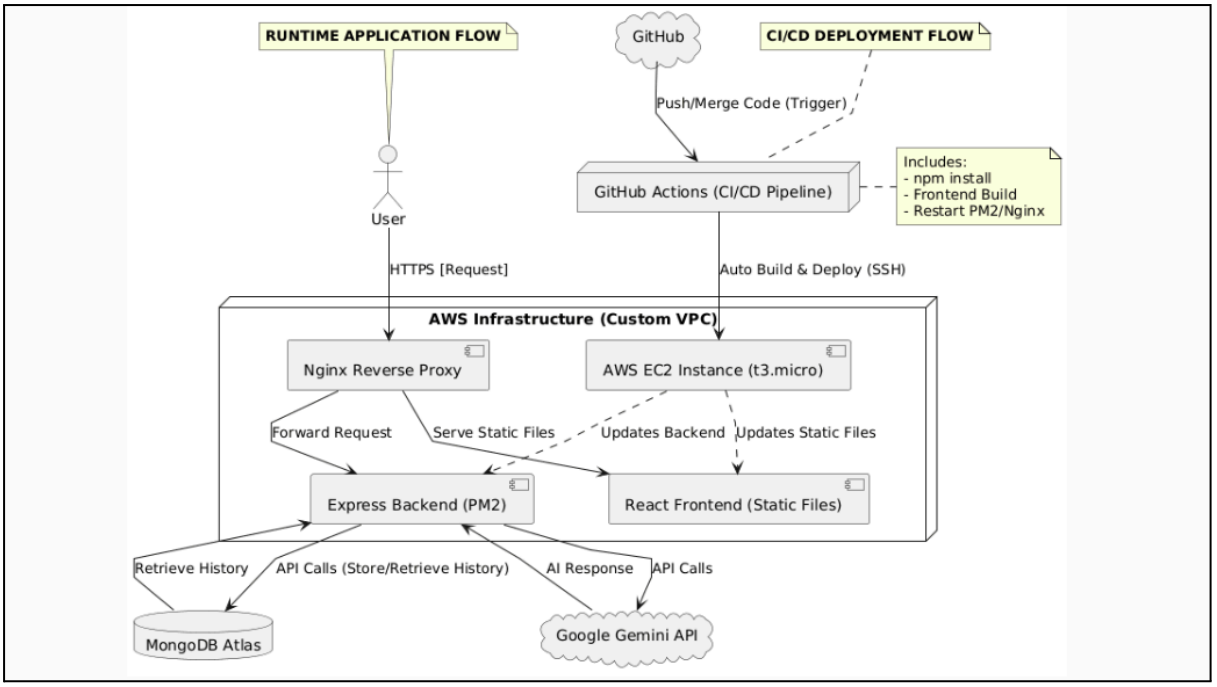
6. Automated CI/CD Pipeline

- GitHub Actions triggers on push/main
- Automatically builds frontend
- Updates backend on EC2
- Runs npm install + restarts services
- Zero manual deployment effort

7. Environment Management

- .env variables stored securely on server
- API keys not exposed

System Architecture:



Workflow Explanation:

1. Runtime Application Flow (User Interaction)

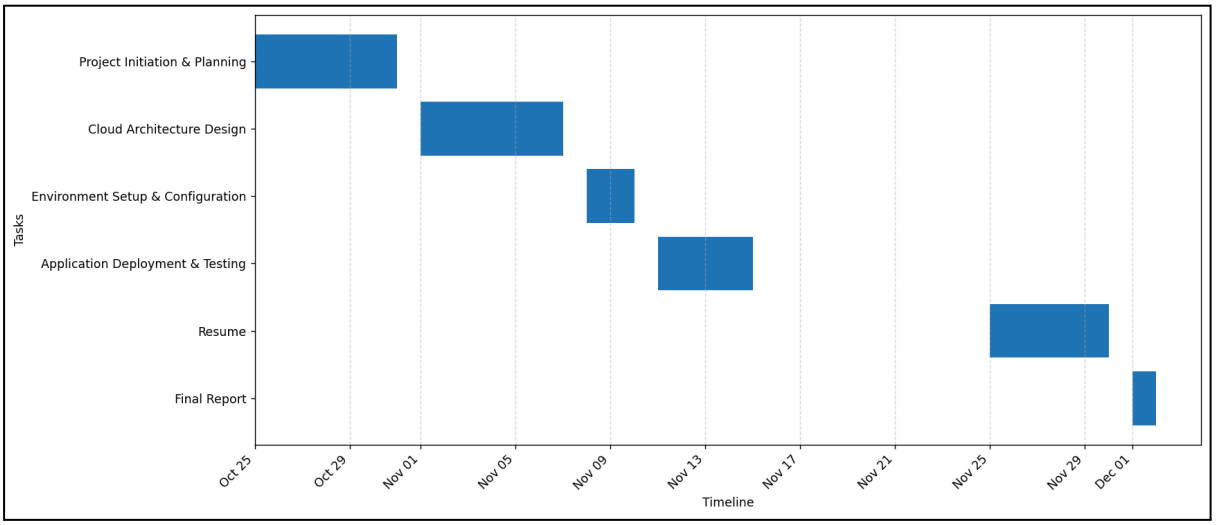
- **Access:** The **User** accesses the application via their **Browser** using the **Public DNS** provided by the **AWS EC2 Instance**. This initiates the secure **HTTPS** connection.
- **Routing:** The request first hits the **Nginx Reverse Proxy** running on the **AWS EC2 Instance**. Nginx performs two key functions:
    - It **Serves Static Files** (the **React Frontend**) directly for the initial load and static assets.
    - It **Forwards Request** (API calls) to the **Express Backend** (managed by PM2).
- **AI Chat:** When a user submits a prompt, the Express Backend receives the request and communicates with the **Google Gemini API** to generate the **AI Response**, which is then returned to the user via Nginx.
- **Data Management:** The Express Backend also manages chat persistence by communicating with the **MongoDB Atlas** database to **Store/Retrieve History** for the user's ongoing conversation sessions.

2. CI/CD Deployment Flow (Automation)

- **Trigger:** Any **Push or Merge Code** event into the designated branch of the **GitHub Repository** automatically triggers the **GitHub Actions (CI/CD Pipeline)**.
- **Build & Deploy:** GitHub Actions performs the necessary steps (including npm install, **Frontend Build**) and then executes an **Auto Build & Deploy** command.
- **Update:** The code is securely transferred and updated on the **AWS EC2 Instance**. Once the files are updated:

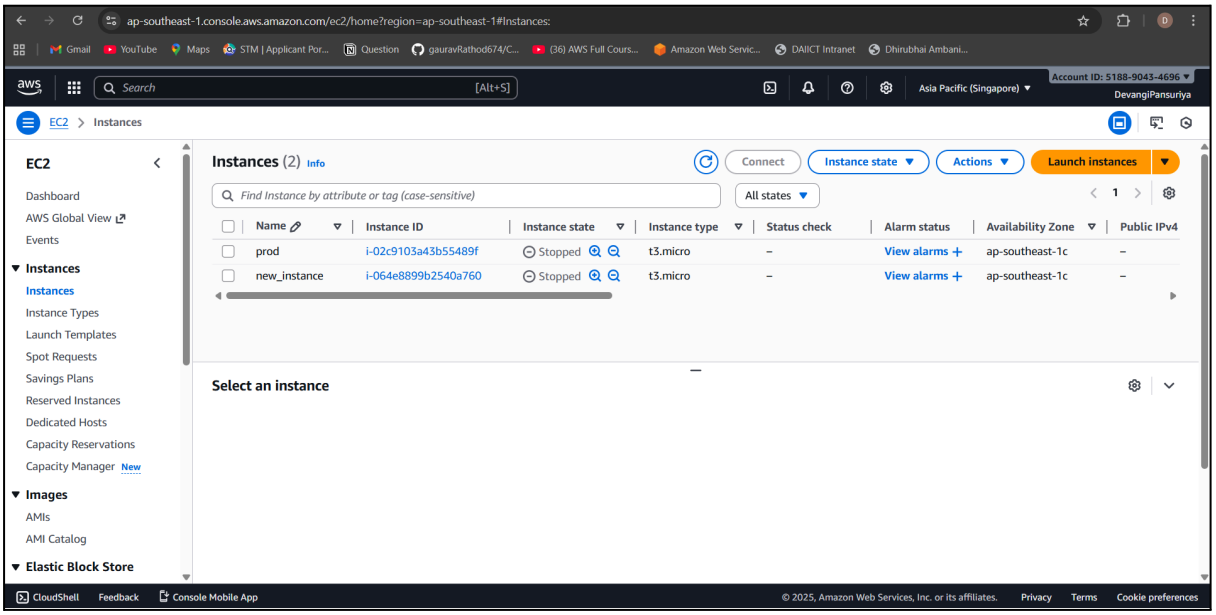
    - The pipeline restarts the **Express Backend (PM2)** service to load the new code.
    - It signals **Nginx** to reload, ensuring the updated **React Frontend** static files are served immediately.
- **Reliability:** This fully automated process ensures that code changes are reliably moved from GitHub to the production EC2 environment with **zero manual deployment effort**.

Gantt Chart:

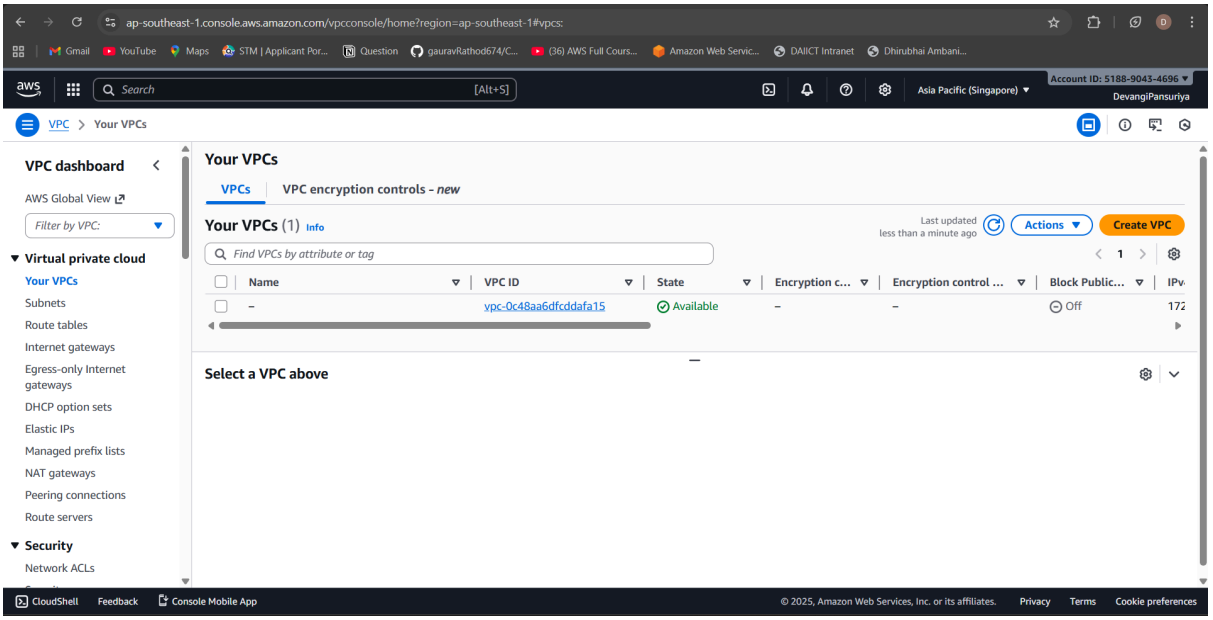


Screenshots:

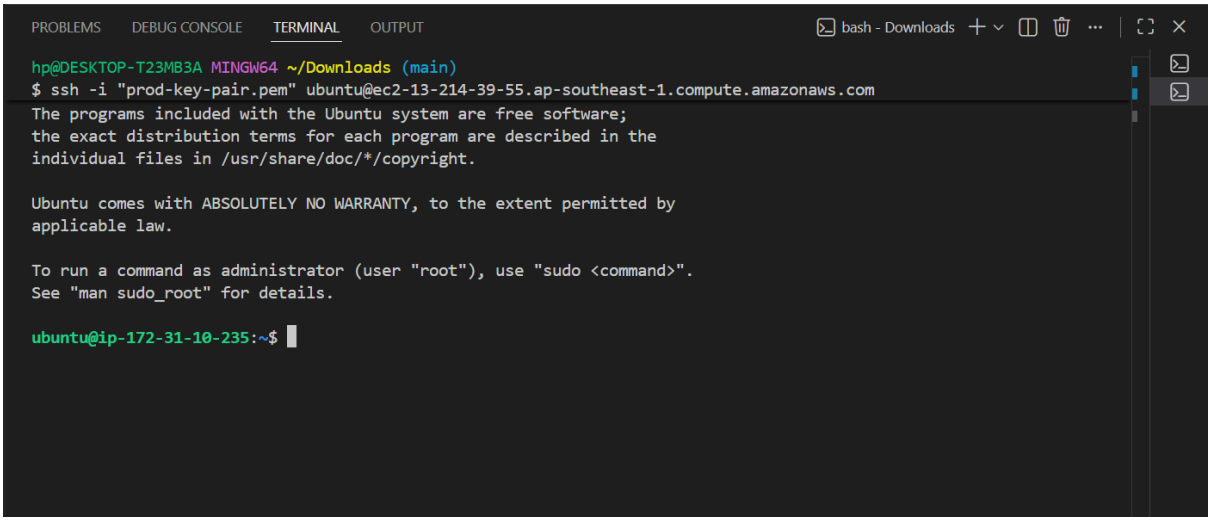
EC2 instance created



VPC

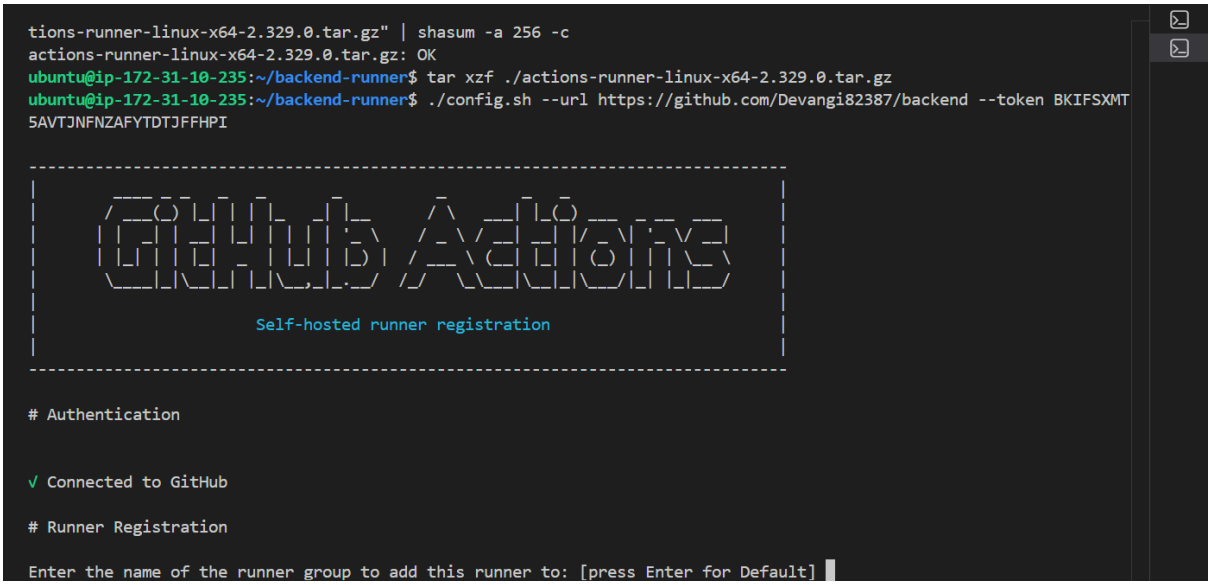


Inside EC2 instance



Backend-runner

Configure github actions in EC2



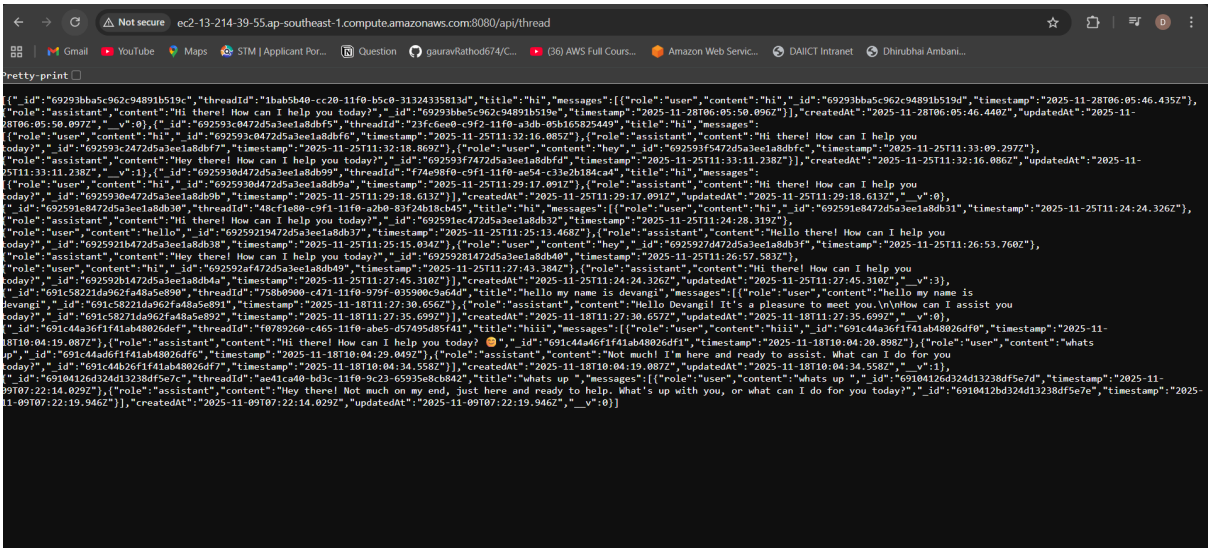
Backend is active

```
$ pm2 monitor

Make pm2 auto-boot at server restart:
$ pm2 startup

To go further checkout:
http://pm2.io/

             ------------

[PM2] Spawning PM2 daemon with pm2_home=/root/.pm2
[PM2] PM2 Successfully daemonized
[PM2] Starting /home/ubuntu/backend-runner/_work/backend/backend/server.js in fork_mode (1 instance)
[PM2] Done.
```

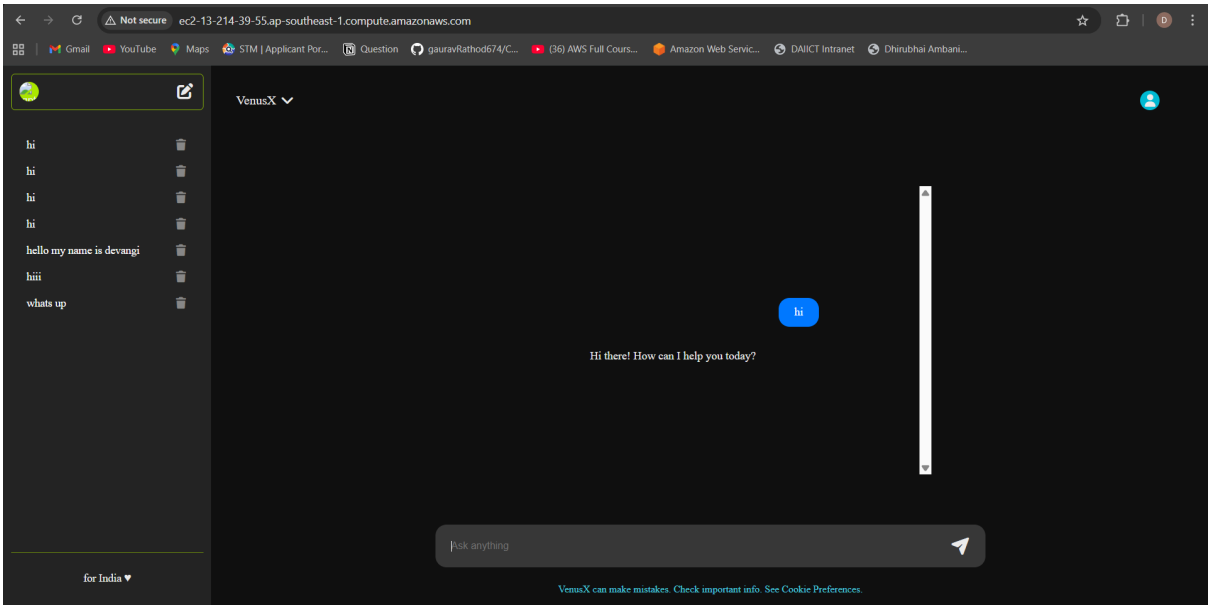
id	name	mode	↺	status	cpu	memory
0	back_end	fork	0	online	0%	24.9mb

```
ubuntu@ip-172-31-10-235:~/backend-runner/_work/backend/backend$
```

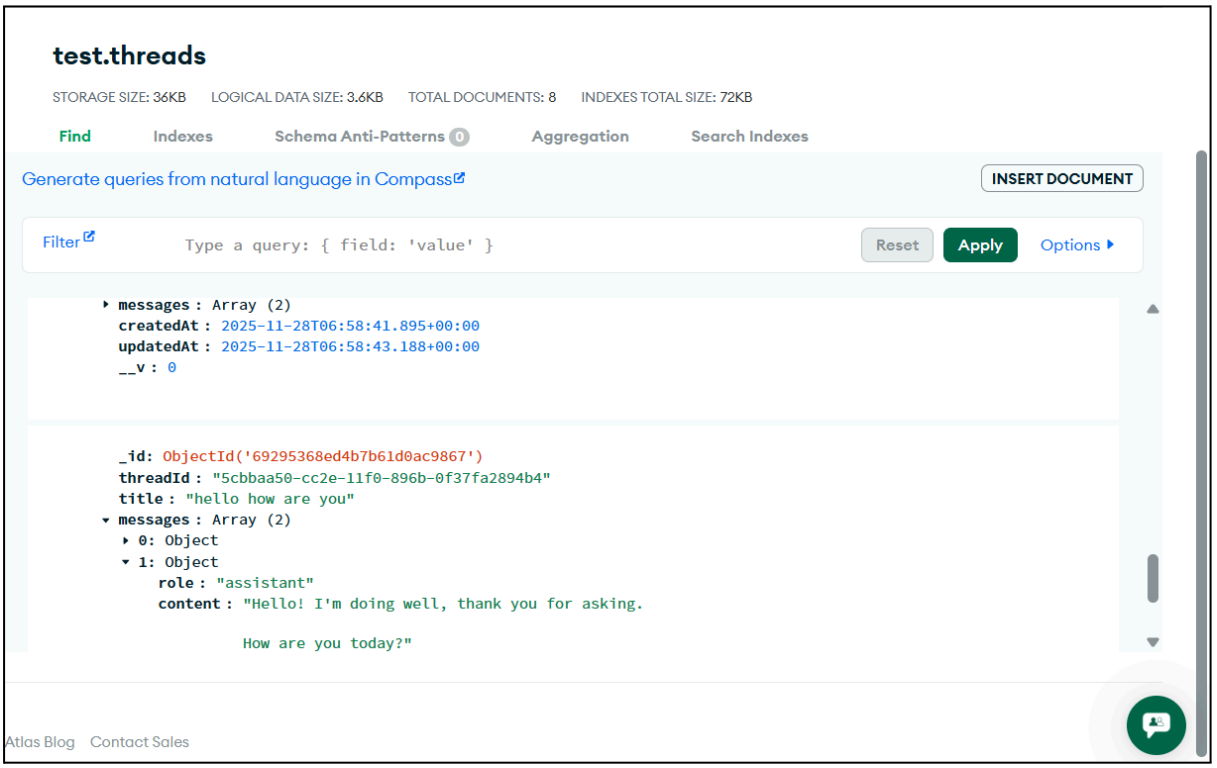
Backend accessed via public DNS of EC2
http://ec2-13-214-39-55.ap-southeast-1.compute.amazonaws.com:8080/api/thread



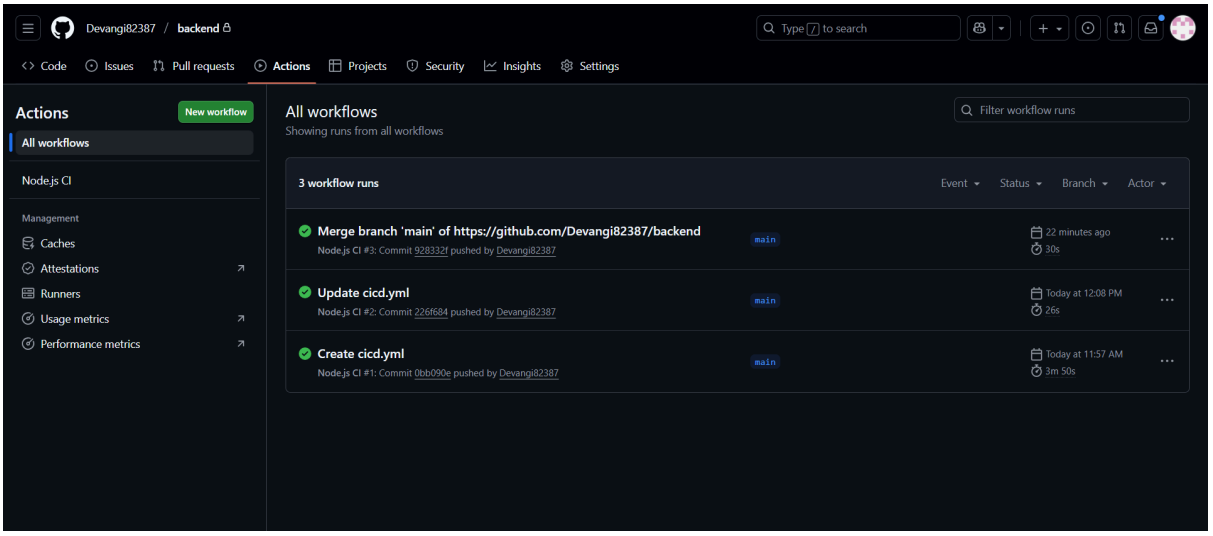
Front End accessed via public DNS of EC2
http://ec2-13-214-39-55.ap-southeast-1.compute.amazonaws.com

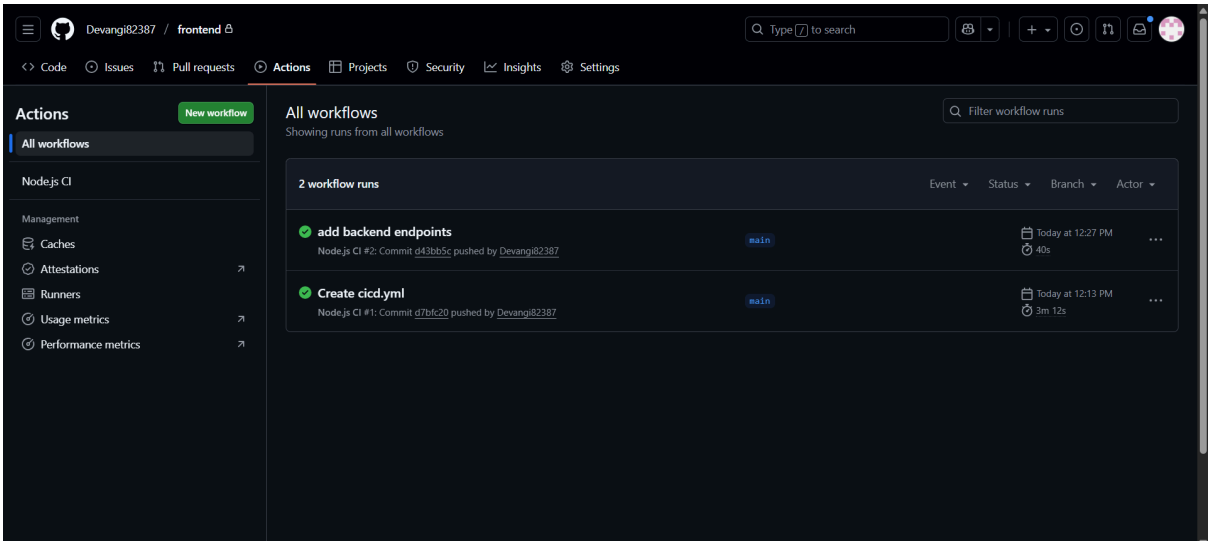


mongoDB atlas



CICD pipeline





Result & Discussion:

The VenusX application was successfully developed and deployed on a cloud-based environment using AWS infrastructure. The CI/CD automation pipeline worked as expected, enabling seamless deployment upon every push to the main branch. The system demonstrated stable functionality during testing across the following criteria:

- **Real-time AI Chat** response using the Gemini API was accurate and responsive under various user prompts.

- **Frontend and backend communication** performed smoothly with secure REST API integration.

- **Cloud deployment** using EC2, Nginx, PM2, and VPC ensured high availability and structured network security.

- The **conversation history feature** worked as intended, enabling users to store, retrieve, and delete messages.

- **MongoDB Atlas** successfully persisted user data, demonstrating smooth cloud database integration.

Overall performance testing confirmed that the system responded within acceptable latency levels, and automation via GitHub Actions ensured error-free, repeatable deployments.

Conclusion:

VenusX successfully delivers an AI-powered cloud-hosted chat system following modern full-stack and DevOps principles. The project demonstrates how cloud computing, automation pipelines, and AI technologies can be combined to build a scalable and intelligent application.

Through this implementation, the team gained hands-on experience with:

- MERN full-stack development
- Cloud infrastructure configuration
- CI/CD automation
- AI-driven conversational models

The project achieves its goals and lays a strong foundation for future enhancements such as multi-user authentication, mobile-friendly UI, voice interaction, and enhanced AI capabilities.