

CLOUD COMPUTING

VenusX - AI Powered chat App

## INDEX

No.	Title	Page no.
	GitHub Link	3
1	Purpose & goal of project	3
2	Scope	3
3	Key Features	5
4	System Architecture	6
5	Workflow explanation	6
6	Gantt chart	7
7	Screenshots	8
8	Result & discussion	12
9	Conclusion	13

## **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

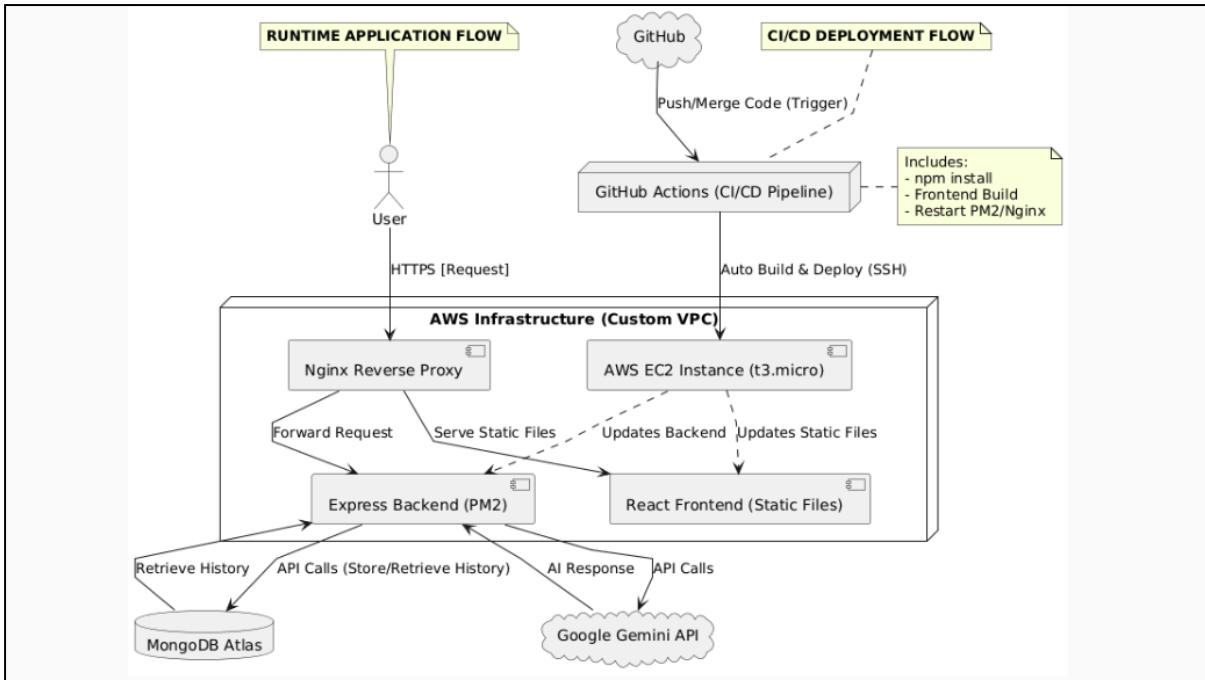
## **6. Automated CI/CD Pipeline**

- 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)

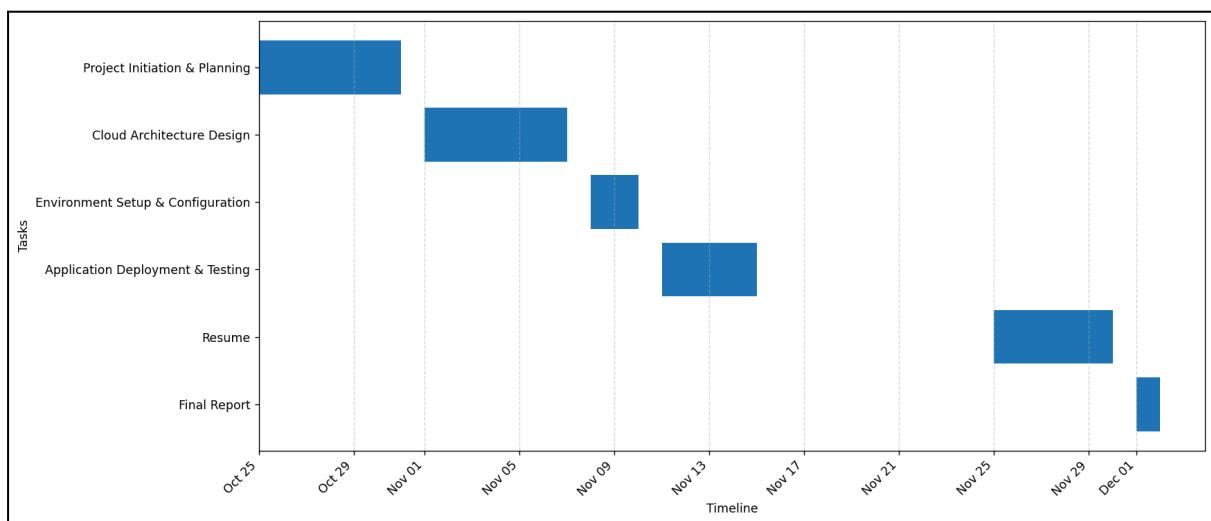
- **Access:** The **User** accesses the application via their **Browser** using the **Public DNS** provided by the **AWS EC2 Instance**. This initiates the secure **HTTPS** connection.
- **Routing:** The request first hits the **Nginx Reverse Proxy** running on the **AWS EC2 Instance**. Nginx performs 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

The screenshot shows the AWS EC2 Instances page. The left sidebar has sections for EC2 (Dashboard, AWS Global View, Events, Instances, Images, Elastic Block Store), VPC (VPC dashboard, Virtual private cloud, Security), Lambda, and CloudWatch Metrics. The main content area is titled 'Instances (2) Info' and lists two instances:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4
prod	i-02c9103a43b55489f	Stopped	t3.micro	-	<a href="#">View alarms</a>	ap-southeast-1c	-
new_instance	i-064e899b2540a760	Stopped	t3.micro	-	<a href="#">View alarms</a>	ap-southeast-1c	-

Below the table is a section titled 'Select an instance'.

## VPC

The screenshot shows the AWS VPC Your VPCs page. The left sidebar has sections for VPC (Your VPCs, Subnets, Route tables, Internet gateways, Egress-only Internet gateways, DHCP option sets, Elastic IPs, Managed prefix lists, NAT gateways, Peering connections, Route servers), Security (Network ACLs), and CloudWatch Metrics. The main content area is titled 'Your VPCs (1) Info' and lists one VPC:

Name	VPC ID	State	Encryption c...	Encryption control ...	Block Public...	IPv4
-	vpc-0c48aa6dfcddfa15	Available	-	-	Off	172

Below the table is a section titled 'Select a VPC above'.

## Inside EC2 instance

## Backend-runner

```
Scanning processes...
Scanning linux images...

Running kernel seems to be up-to-date.

No services need to be restarted.

No containers need to be restarted.

No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
ubuntu@ip-172-31-10-235:~$ mkdir backend-runner && cd backend-runner
ubuntu@ip-172-31-10-235:~/backend-runner$
```

## Configure github actions in EC2

## Backend is active

```

MONITOR in production.
$ 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.



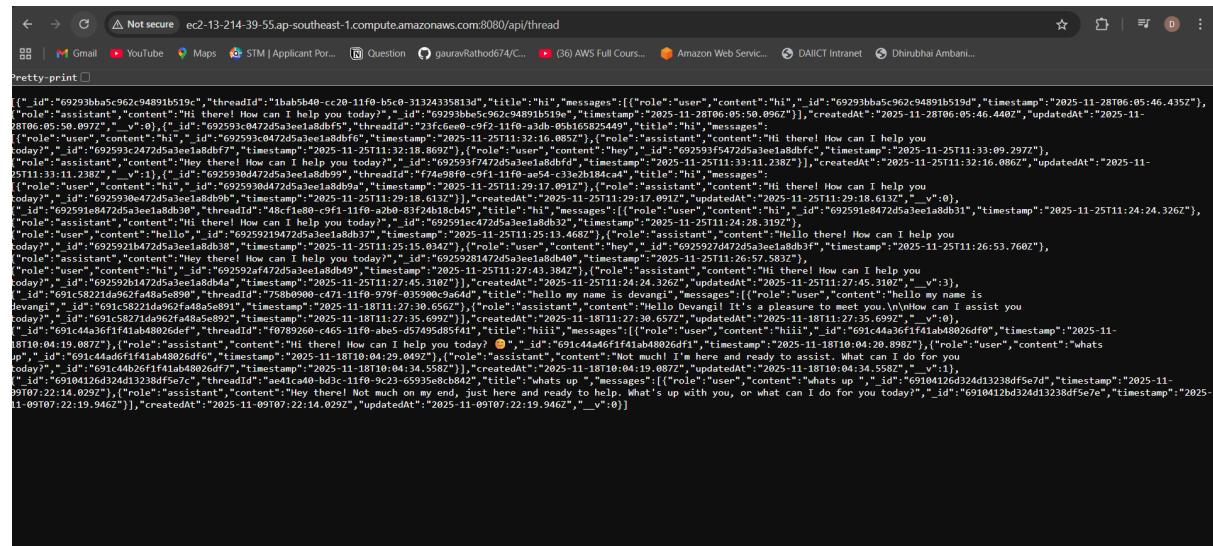
| <b>id</b> | <b>name</b> | <b>mode</b> | <b> </b> | <b>status</b> | <b>cpu</b> | <b>memory</b> |
|-----------|-------------|-------------|----------|---------------|------------|---------------|
| 0         | back_end    | fork        | 0        | online        | 0%         | 24.9mb        |


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

```

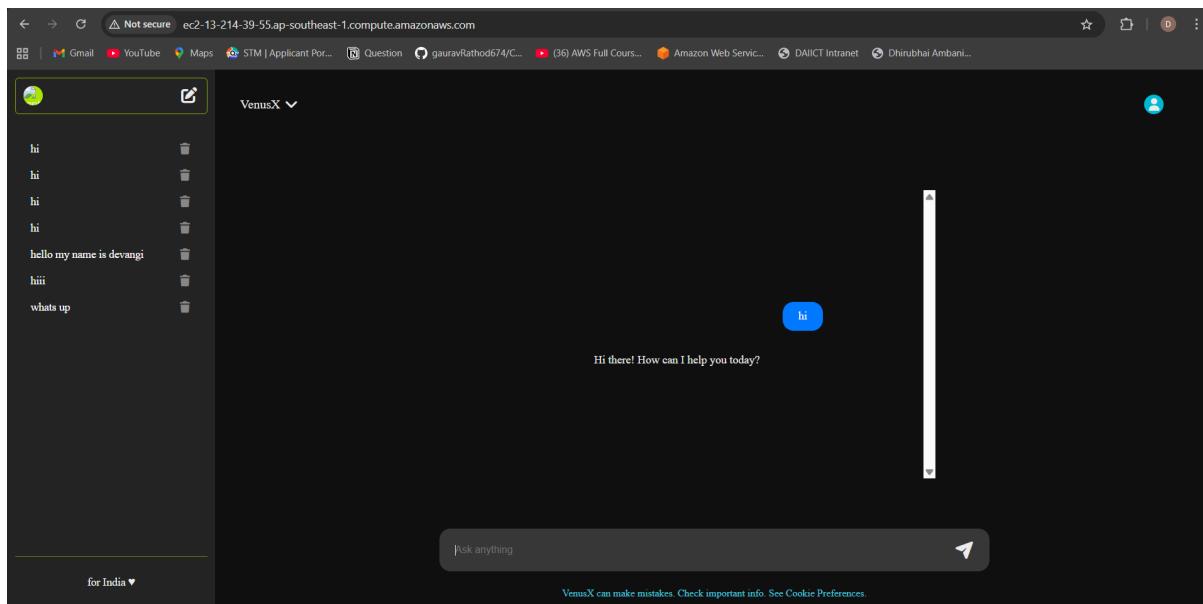
## Backend accessed via public DNS of EC2

<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

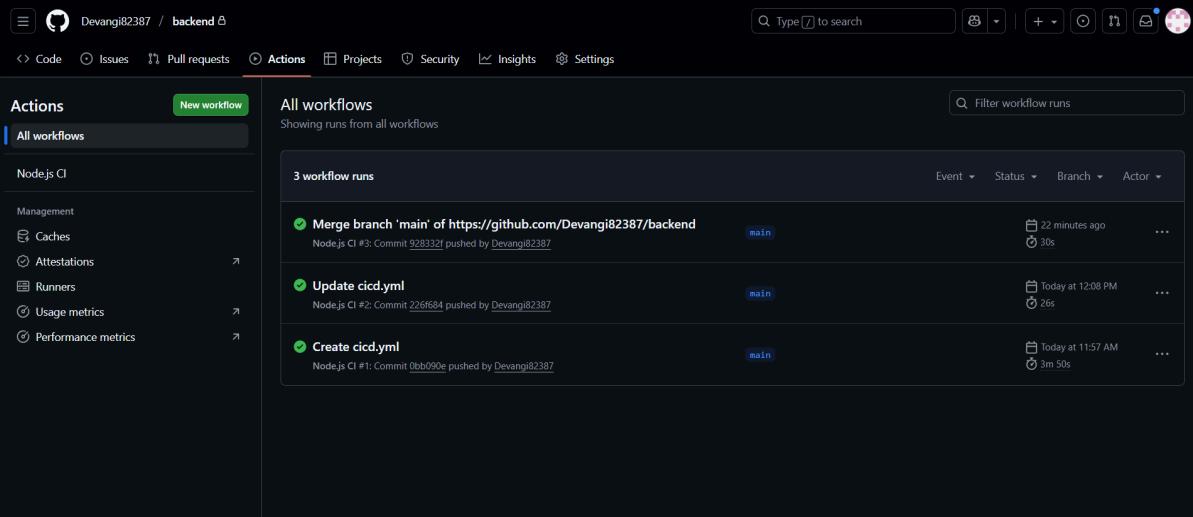
A screenshot of the MongoDB Atlas interface, specifically the "test.threads" collection. The collection has a storage size of 36KB, logical data size of 3.6KB, total documents of 8, and indexes total size of 72KB. The "Find" tab is selected. A query builder allows generating queries from natural language. A preview pane shows the document structure:

```
messages : Array (2)
  createdAt : 2025-11-28T06:58:41.895+00:00
  updatedAt : 2025-11-28T06:58:43.188+00:00
  __v : 0

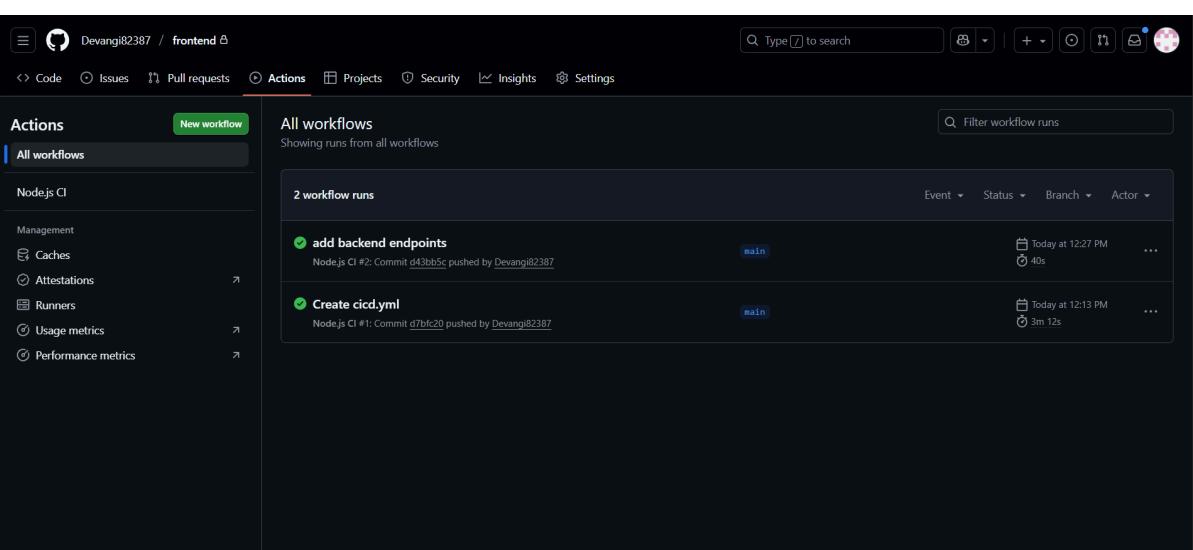
  _id: ObjectId('69295368ed4b7b61d0ac9867')
  threadId : "5cbbba50-cc2e-11f0-896b-0f37fa2894b4"
  title : "hello how are you"
  messages : Array (2)
    0: Object
    1: Object
      role : "assistant"
      content : "Hello! I'm doing well, thank you for asking.

      How are you today?"
```

## CICD pipeline



The screenshot shows the GitHub Actions pipeline for the 'backend' repository. It displays three workflow runs: 'Merge branch 'main'' (status: success, 22 minutes ago), 'Update cicd.yml' (status: success, Today at 12:08 PM), and 'Create cicd.yml' (status: success, Today at 11:57 AM). The sidebar on the left shows management options like Caches, Attestations, Runners, Usage metrics, and Performance metrics.

The screenshot shows the GitHub Actions pipeline for the 'frontend' repository. It displays two workflow runs: 'add backend endpoints' (status: success, Today at 12:27 PM) and 'Create cicd.yml' (status: success, Today at 12:13 PM). The sidebar on the left shows management options like Caches, Attestations, Runners, Usage metrics, and Performance metrics.

## 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.