GROUP 2

Cloud-Based Agentic Solution Architecture

Statement of Work

MiMinions.ai

Geetha Narasimha
Taral Kotwal
Sandeep Parwal
Aditya Mhatre

Table of Contents

1. Document Version

Version	Description	Date	Author(s)
1	Statement of Work	01/02/2025	Sandeep Parwal Geetha Narasimha Taral Kotwal
2	Statement of Work - Updated	23/02/2025	Sandeep Parwal
3	Statement of Work – Updated	16/03/2025	Aditya Mhatre
4	Statement of Work – Updated	05/04/2025	Aditya Mhatre
5	Statement of Work – Updated	15/04/2025	Sandeep Parwal

2. Team Contact Information

Group 2

Ife Agboola - iagboola@bowvalleycollege.ca - **Supervisor**

Sandeep Parwal – s.parwal175@mybvc.ca – Team Lead

Geetha Narasimha – g.narasimha064@mybvc.ca - Assistant Team Lead

Taral Kotwal – t.kotwal365@mybvc.ca

Aditya Mhatre - a.mhatre623@mybvc.ca

3. Description

MiMinions.ai is seeking to explore and design a cloud-based architecture that can support general agentic solutions. The project aims to leverage cloud technologies to create a scalable, flexible, and efficient framework that can be adapted for various agentic applications. Learners will investigate existing cloud solutions, evaluate their suitability for agentic tasks, and propose an architecture that integrates these technologies e8ectively. The project will focus on identifying key components such as data storage, processing capabilities, and communication protocols that are essential for agentic solutions. By applying their classroom knowledge in cloud computing and system architecture, learners will gain practical experience in designing a robust and adaptable cloud-based system.

4. Proposed Solution

As per the requirements obtained from the Client, following are the assumptions/analysis we have made for now:

- MiMinions - We have to provide multiple AI agentic models which take specific tasks from the users and give solutions.
- Every Queries can be processed using Open API (rest API) at the backend requested by the chatbot interface (assuming we need to provide the chatbot interface)
- These queries are generally related to AWS services or specific task services where users can ask any open question or in generic it must respond.
- Database entities would be designed for **a**. storing these queries, **b**. Metadata of the AI model, **c**. User information
- We have discussed the different agents in our latest call (19-02-2024) with the client. We have shown a demo on Agricultural Help Centre AI Agent.
- We have proposed a draft AWS architecture diagram with a client where he will share some feedback on it.
- Client proposed to integrate the RAG, GraphQL and AWS ECR.
- Client has shared the AWS account with us, where we have registered and will use the services in future.
- Finalization of AWS architecture and database model
- Refinement of event-driven processing using Apache Kafka
- Updated cost estimation (V1 & V2) for cloud services
- Project development focusing on homepage and backend integration
- Implementation of Lambda-based Image Processing Agent
- Kafka evaluated for use via AWS MSK for event-driven architecture; implementation planning completed.
- Flask used for backend development; deployed locally and to AWS EC2 instance (currently single-instance setup).
- Chatbot supports registration, login, file upload (admin only), and file querying from S3/DynamoDB.

5. Stakeholders

Client Name: Roland Ding
Role: Founder of MiMinions.ai
Email Id: shengxio@miminions.ai

6. Proposed Technology

As of now the Technology stack is not yet finalized but we discussed it with the Client and came up with the preferences:

- Programming language Framework: Python (Client suggested), ~~Node Js~~

- Database: ~~MySql~~, ~~Postgres~~ ,DynamoDB
- Third Party: AI Models – OpenAI, ~~Azure, Gemini~~
- Server: AWS Cloud and Services
- Tools: Postman Collections, Draw.io, Miro.io
- Project Management Tool: JIRA
- Repository Tool: GitHub
- Apache Kafka (via AWS MSK) – Researched for asynchronous processing.
- Flask (Python framework) – Backend implementation.
- AWS CodeGuru – For repository code analysis.
- Terraform – Infrastructure provisioning and future CI/CD automation.

7. Assumptions

A list of assumptions that could be related to the scope of the project:

7.1. Non-Technical Assumptions

- The project team consists of both technical and non-technical members, requiring clear documentation and communication.
- The architecture should be designed with ease of use and adaptability for various agentic applications.
- The solution should align with budget considerations while ensuring performance.
- The project will be broken down into manageable phases with clear deliverables.
- The final architecture should support various agentic applications without deep technical expertise required.
- Security and governance policies must be integrated into the system as per client requirements.
- The project must comply with AWS cost optimization best practices to ensure affordability.
- The system must support multi-user access and role-based permissions for different levels of access

7.2. Technical Assumptions

- Serverless functions like AWS Lambda might be used for event-driven processing.
- The system may use a combination of SQL (e.g., PostgreSQL, MySQL) and NoSQL (e.g., DynamoDB, MongoDB) databases.
- Different AI Agents – Chatbot (Proposed and Discussed but not yet finalized)
    - Agricultural Help Centre
    - Image Processing
    - Language Model
- ~~Students will/will not have access to the client server~~
- ~~Client provided the AWS account details with us~~.
- APIs will be designed using REST or GraphQL, secured with authentication and rate limiting.

- Logging, monitoring, and alerting will be implemented.
- Automation using GitHub Actions, Jenkins, or GitLab CI/CD for seamless deployment.
- Apache Kafka is being researched for event-driven services, but feasibility is still under review.
- The system must support scalable AI model integration, enabling future updates without major architectural changes.
- IAM policies must be refined to avoid issues like AWS 403 errors encountered in Sprint 3.

8. Deliverables

By the end of the project, we aim to deliver:

Deliverable	**Description**
Document Structure and Preparation	Documents structured and prepared - SOW - Stage Plan of ongoing sprints - Risk and Issues - MOM - Weekly Status Report
Stage Planning	Document structured and planning has been provided for Sprint Planning
Kickoff and Communication with Client	Though it is an on-going progress, what was planned for this Sprint or Stage has been covered.
Cloud Architecture	Cloud architecture blueprint with documentation.
Deployment Scripts	Infrastructure-as-Code (IaC) scripts for deployment (Terraform, AWS CDK, etc.).
Agent Orchestration	Agent orchestration service with working examples
APIs	APIs for agent interaction (e.g., agent control, task execution).
Security and Access Control	Security and access control implementation
Monitoring	Implementation of Monitoring and logging tools and setup
Sprint 1	Completed
Sprint 2	Completed
Sprint 3	Completed
Sprint 4	Completed
Codebase - GitRepo	https://github.com/MiMinions-ai/bvc-project

9. Requirements

The client expects the MiMinions.ai team to design a scalable, cloud-based architecture that supports general agentic solutions. The team is responsible for evaluating existing cloud technologies, ensuring modularity, security, and cost-effectiveness while proposing an adaptable framework for various agentic applications. The ultimate goal of PoC (proof of concept) is to create a resilient, efficient system that enables intelligent agents to automate tasks, collaborate effectively, and make data-driven decisions. By leveraging cloud-native technologies, the client aims to optimize costs, enhance scalability, and establish a competitive, future-proof solution for agentic workloads.

10. Lower Priority Features

- Agent Customization & UI Enhancements
- Auto-adaptive agents that learn dynamically over time.
- Advanced Monitoring & Logging
- Expansion of Cloud Services – Multiple AZs

11. Out of Scope

Item Description
Development of advanced agent logic or AI models beyond standard API calls.
UI/Frontend development (this scope is backend/cloud-focused).
Edge computing or on-premise deployments.
Full-scale production deployment (this project focuses on an MVP or prototype).

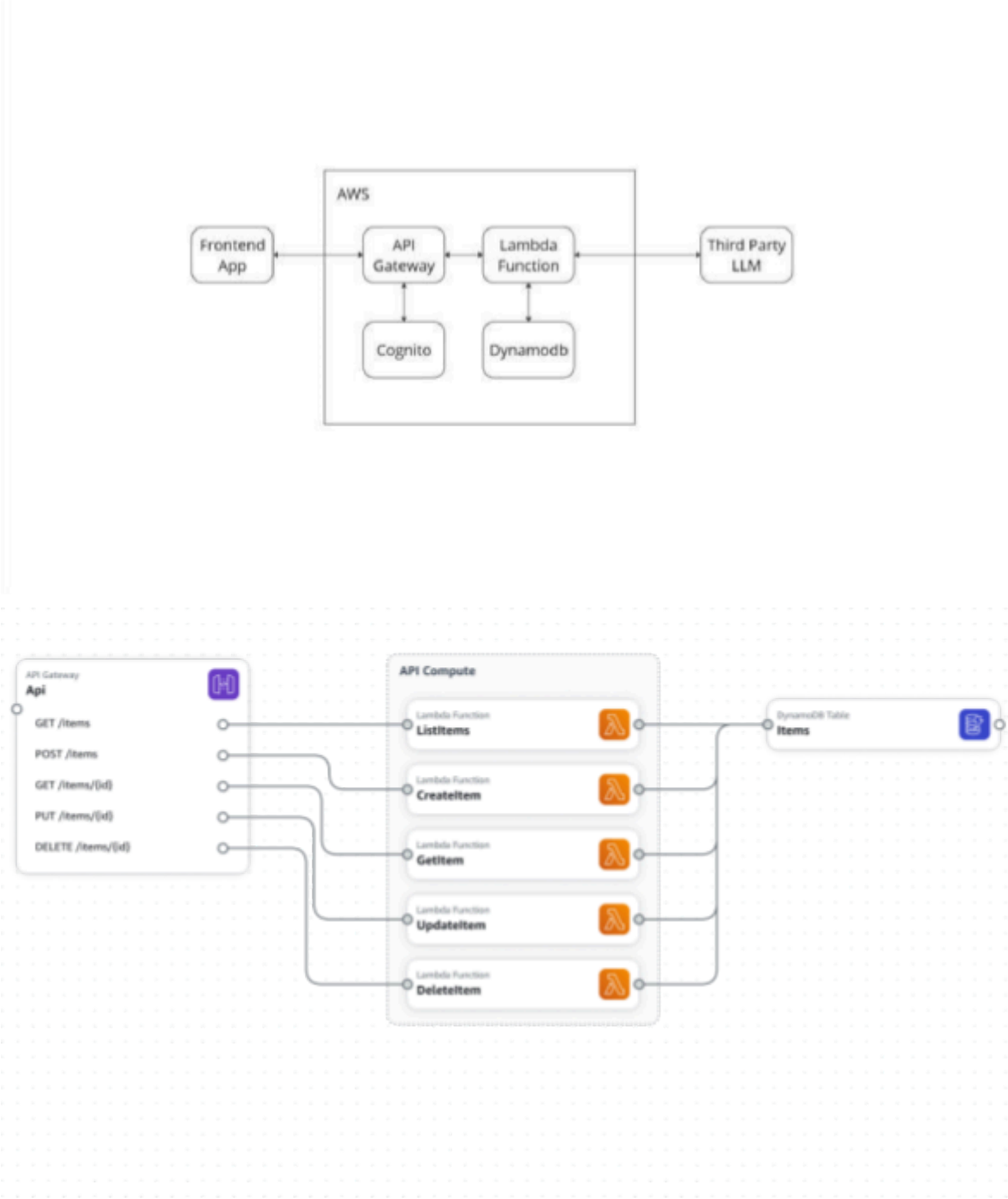
12. Existing System

N/A

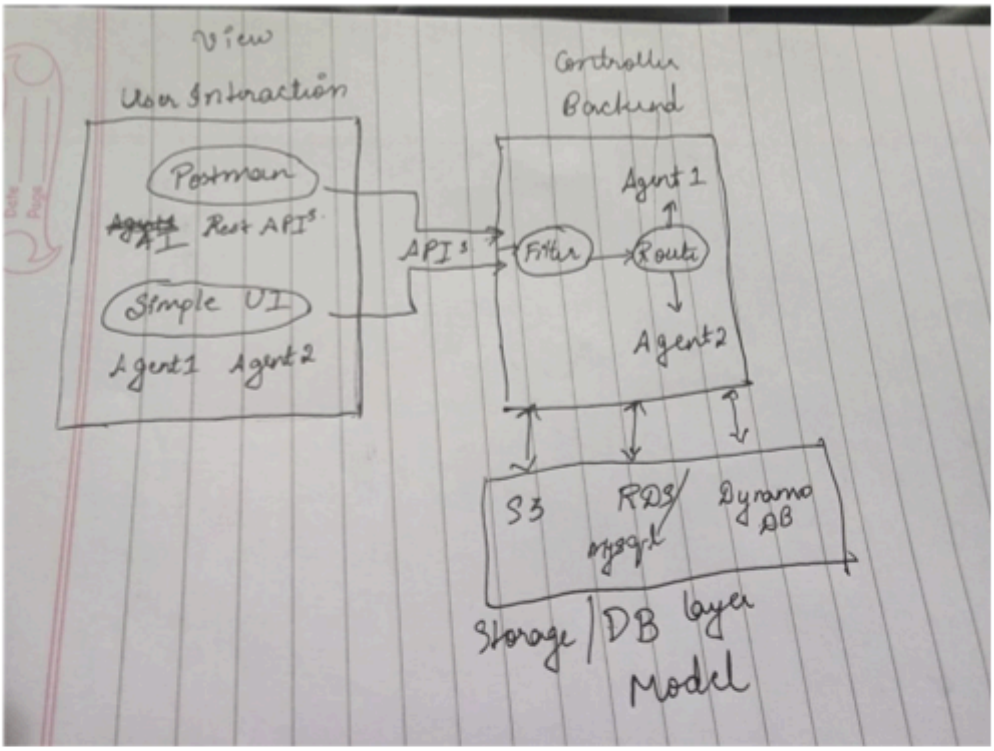
13. Diagrams

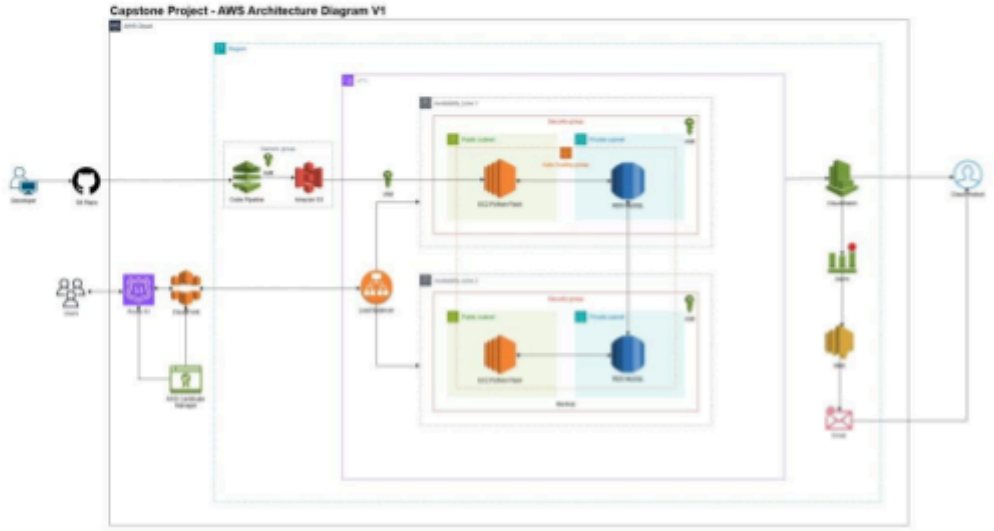
A draft diagram shared by client and team members.

13.1. Client Diagrams

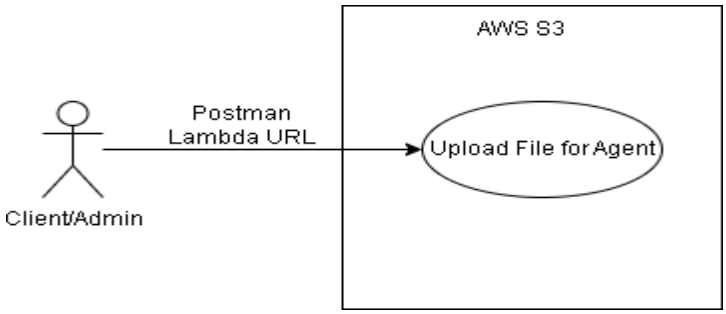


13.2. Team Diagrams

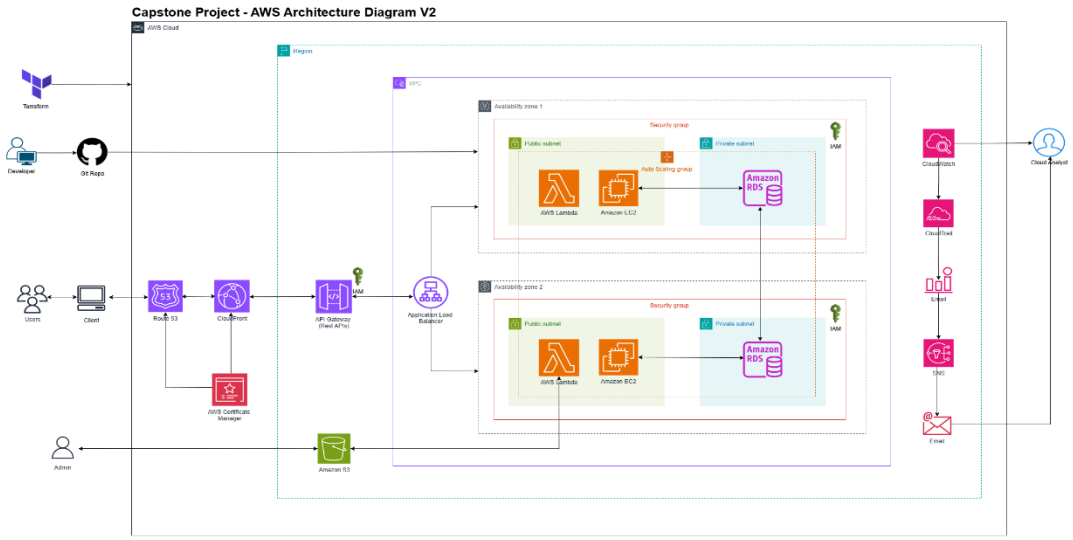




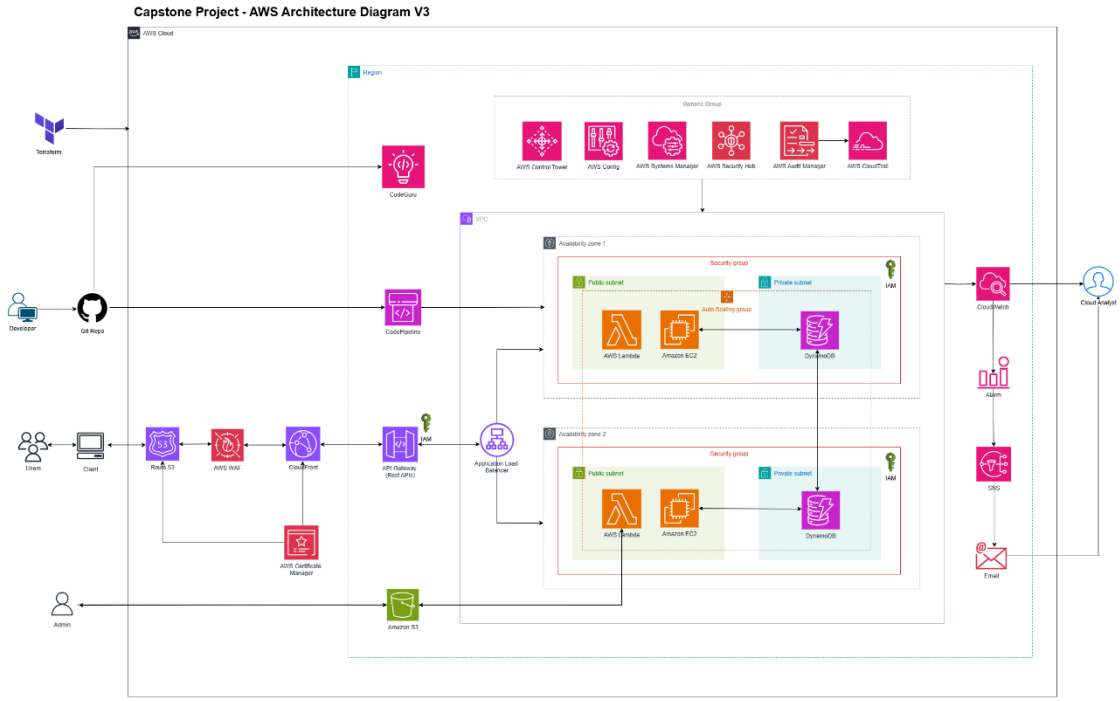
Use case diagram



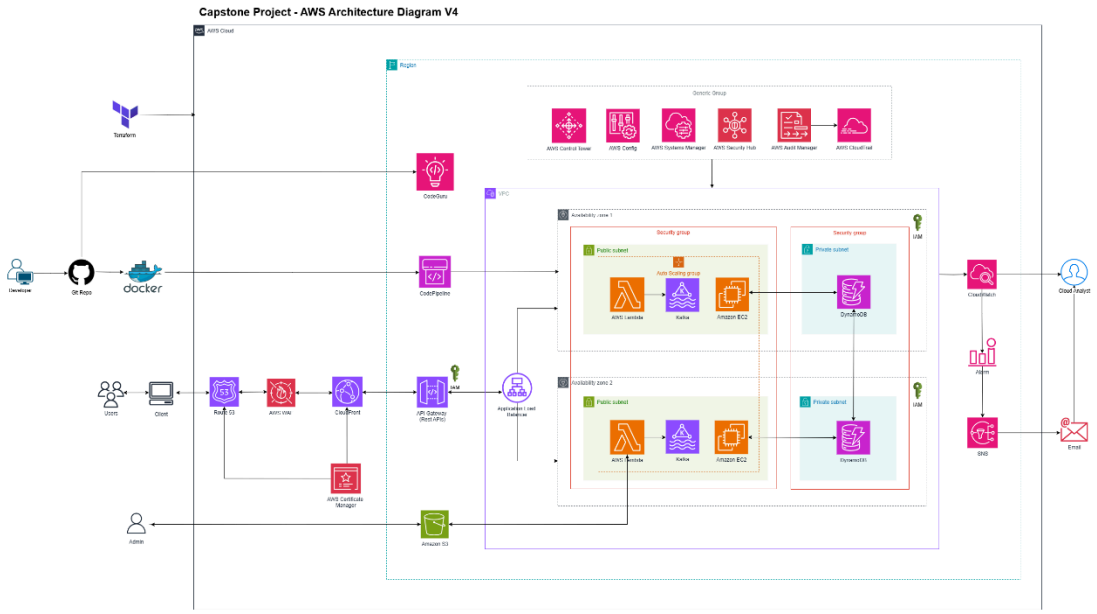
Updated architecture diagram-v2



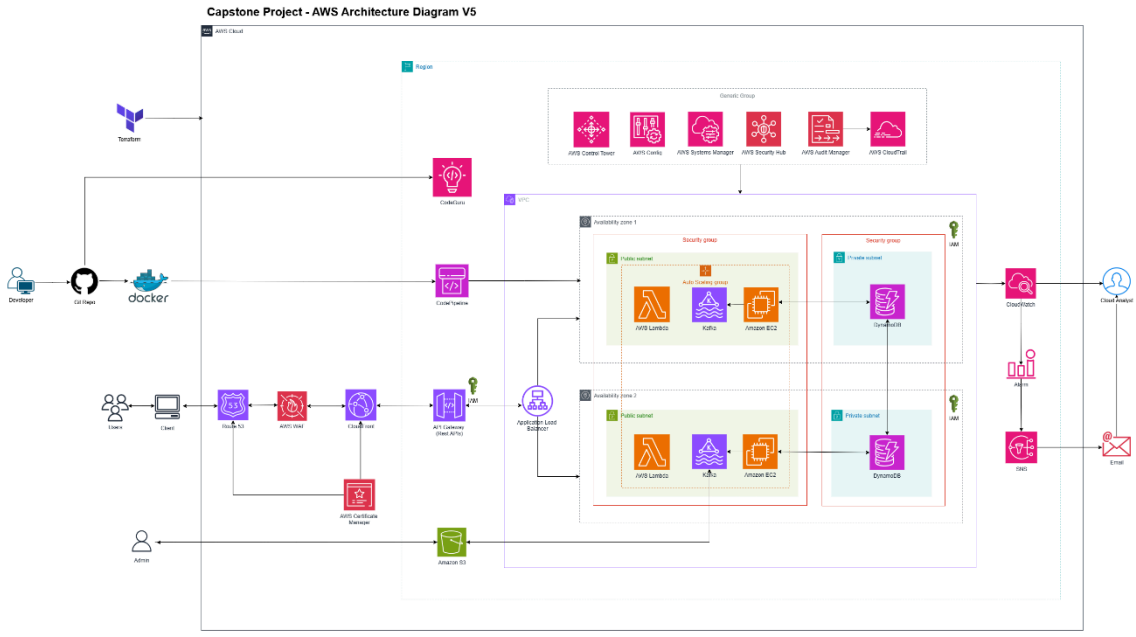
Updated architecture diagram-v3



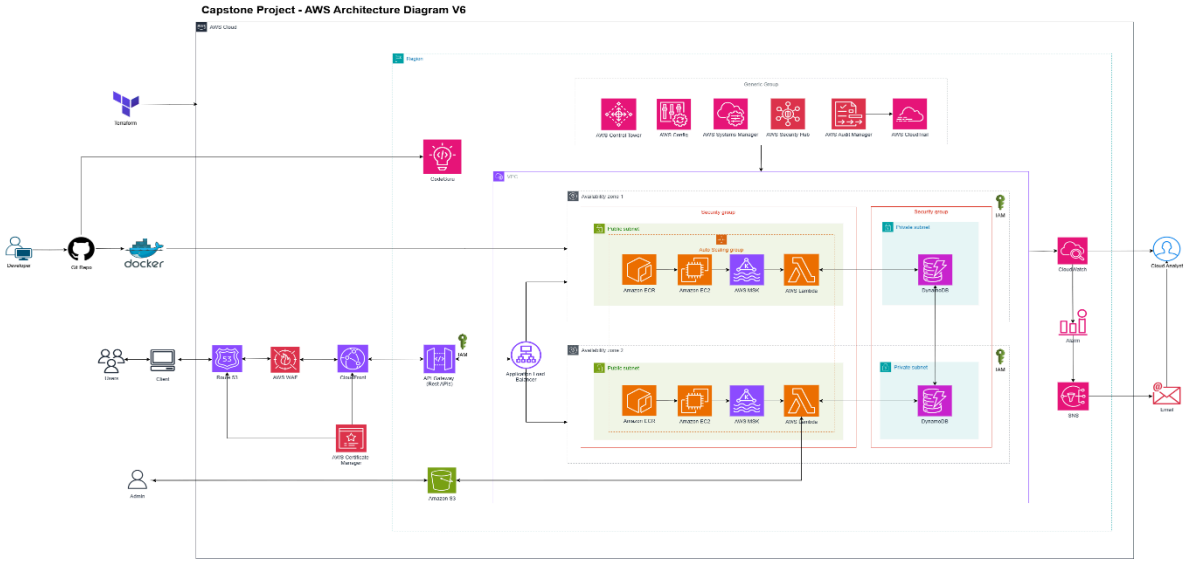
Updated architecture diagram v4



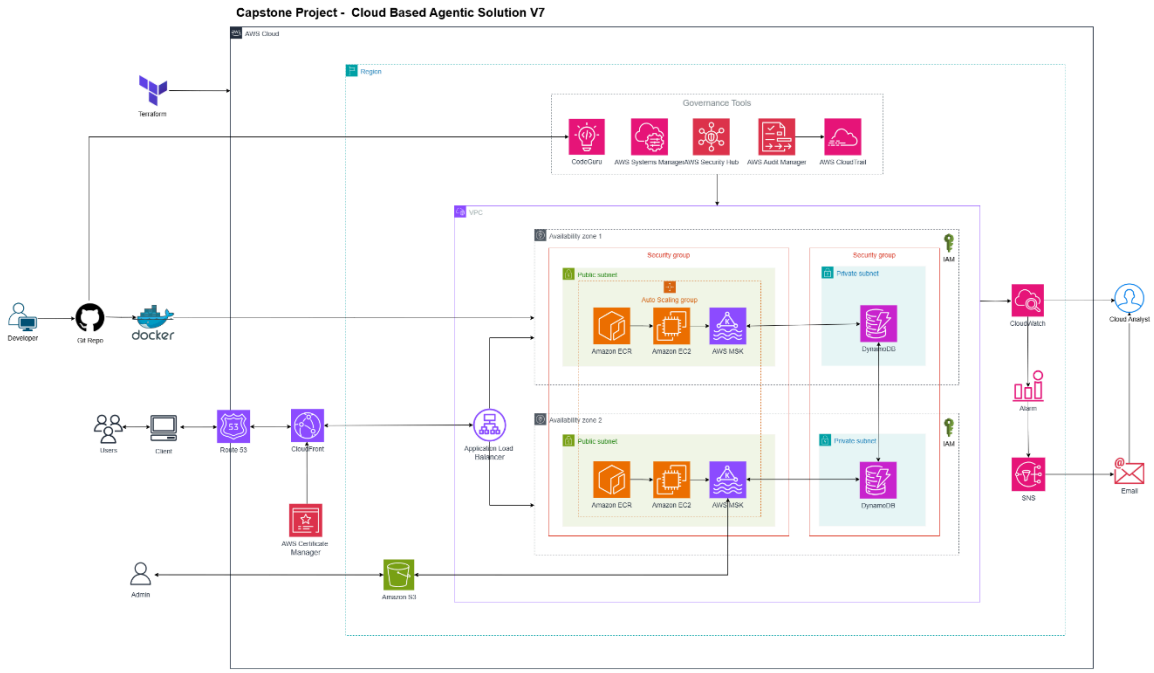
Updated architecture diagram v5



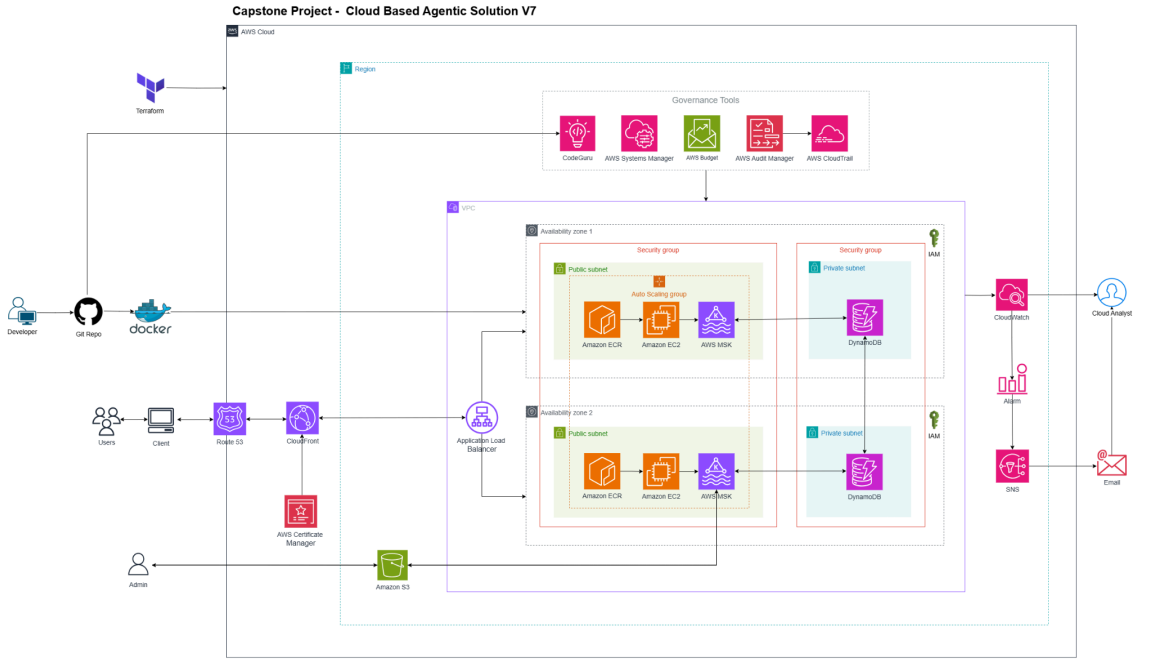
Updated architecture diagram v6



Updated architecture diagram v7



Final - Architecture Diagram



14. High-Level Schedule

Here is the high-level schedule:

#	**Milestone Description**	**Estimated Completion**
Phase 1	Research & Architecture Design	Week 2
Phase 2	Cloud Infrastructure Setup	Week 4
Phase 3	Agent Orchestration & API Development	Week 6
Phase 4	Security Implementation	Week 7
Phase 5	Testing & Deployment	Week 8

15. High-Level Work Breakdown Structure (WBS)

Owner	**Description**	**Completion Date**
Team	Project Initiation	07/02/2025
Team	Research and Requirement Analysis	15/02/2025
Team	Architecture Design	14/04/2025
Team	Proof of Concept – Development	15/03/2025
Team	Testing	15/04/2025
Team	Deployment	15/04/2025
Team	Documentation, Knowledge Sharing & Handover	15/04/2025
Team	Sprint 1	02/02/2025
Team	Sprint 2	23/02/2025
Team	Sprint 3	16/02/2025
Team	Sprint 4	15/04/2025

16. Risks and Issues

Risk or Issue Description	**Potential Impact**	**Mitigation/Resolution Steps**
Knowledge Gaps in the Team	The mix of technical and nontechnical members may slow down progress if knowledge sharing is insufficient.	Conduct regular knowledge transfer sessions to ensure both technical and nontechnical team members are aligned.
Timeline Delays	Dependencies on third-party services, unexpected technical issues, or resource	Define project milestones clearly and manage scope effectively to prevent delays.

	constraints could delay project milestones.	
Scalability Challenges	The architecture may struggle to handle large numbers of agentic tasks efficiently, requiring proper auto-scaling mechanisms.	
Security Vulnerabilities	Unauthorized access, data leaks, or cyber threats could compromise the system if proper security measures (IAM, encryption, firewalls) are not implemented.	Use industry standards for security, and cloud architecture to ensure reliability.
Testing & Validation Gaps	Insufficient testing could lead to undetected security flaws, performance issues, or system failures post-deployment.	Implementation of proper testing strategies to validate performance, security, and scalability.
Cost Overruns	Unoptimized cloud resource usage may lead to unexpected costs, making the solution financially unviable.	Continuously monitor and optimize cloud resources to avoid unnecessary expenses.
Too Many AWS Services Could Increase Costs	Budget overruns due to excess service usage	Optimize service selection and regularly monitor AWS billing
Chatbot May Not Always Give Correct Answers	User experience may suffer due to inaccurate AI responses	Continuous model training and feedback mechanisms to improve accuracy
Delays in Completing the Project	Project deadlines may be missed	Define clear milestones, conduct weekly progress tracking, and mitigate risks early
Technical Issues Prevented the Demo	Client unable to evaluate system performance	Debug and resolve all identified issues before the next scheduled client meeting
Unable to Specify IAM Role for Lambda	Lambda function cannot execute due to permission issues	Work with AWS administrators to request necessary IAM permissions
Need to Compare MongoDB vs.	Database selection is pending, causing delays	Conduct performance benchmarking and cost analysis for all three databases

DynamoDB vs. MySQL for Suitability		
Research Required on Apache Kafka and Alternatives	Uncertainty in event-driven processing solution	Compare Apache Kafka with AWS SQS and RabbitMQ for best suitability
Code Review May Be Delayed	Potential slowdowns in development progress	Schedule periodic follow-ups with the client to ensure timely feedback and integration
Kafka Integration May Be More Complex Than Anticipated	Implementation challenges could arise	Break down integration into smaller phases and explore alternative streaming solutions
Feasibility of the Custom Agent Feature Remains Uncertain	Project scope may need re-evaluation	Conduct additional discussions with the client to define feasibility and expected outcomes
AWS Sandbox Limitations Could Impede Development Progress	Testing constraints could slow down iterations	Request additional AWS resources or migrate testing to a full AWS environment
Integrating Governance Policies May Prove Challenging	Security and compliance requirements may not be met	Define governance policies early and integrate them incrementally into the system
Successful Project Completion Depends on Timely Kafka Research	Delays in Kafka feasibility could hinder design	Allocate dedicated resources to complete Kafka research within the project timeline
Scaling DynamoDB May Cause Performance Issues with High Transaction Volumes	Database performance bottlenecks	Optimize schema design, implement partitioning strategies, and configure auto-scaling
Errors in AWS Services Could Disrupt the Project Timeline and Stability	System failures and instability	Regular testing, real-time monitoring, and proactive troubleshooting to resolve issues
Post-login Agent Instance Creation May Require Additional Resources and Further Research	Potential complexity in resource management	Conduct feasibility studies and explore cost-effective solutions for instance creation
Scope Expansion	May lead to time and resource constraints if not managed effectively	Define clear objectives and prioritize tasks based on client needs

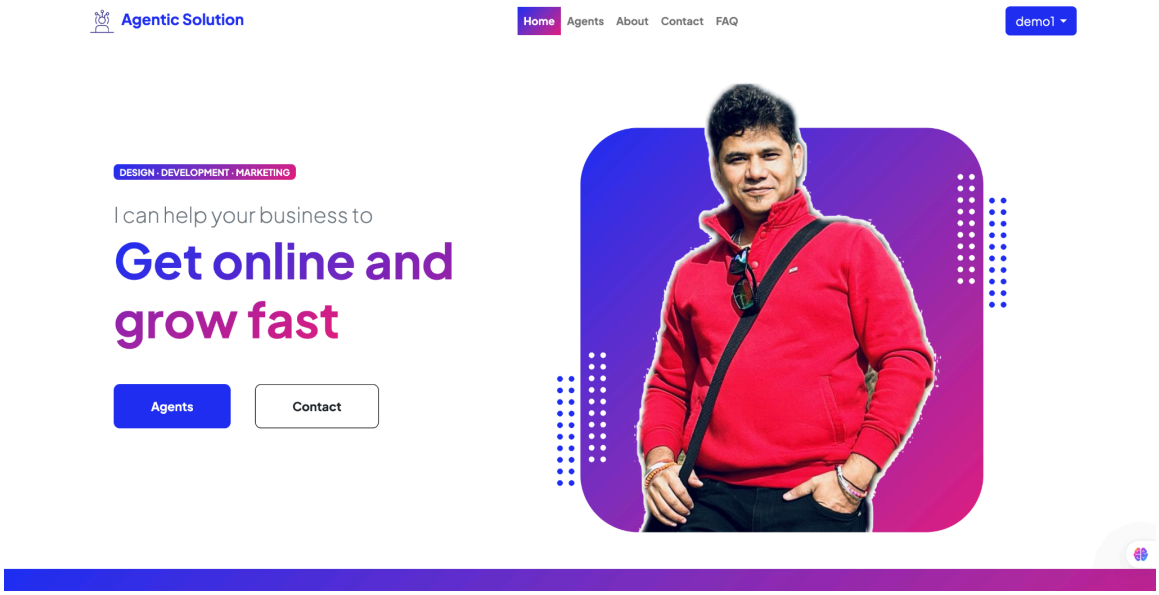
Dependency on Kafka Integration	Delays in Kafka integration could block other dependent services	Parallelize Kafka research with other development tasks and define fallback mechanisms
Terraform CI/CD Pipeline Deployment Issues	Could affect deployment timelines and service testing	Set up isolated test environments for CI/CD trials before final deployment
Security Configuration Gaps in API	Increased risk of unauthorized access or data leaks	Conduct security audits and implement encryption, IAM roles, and access policies
Managing Multiple AWS Services (Kafka, Terraform, CI/CD)	May increase system complexity and require additional maintenance efforts	Assign clear responsibilities for each service and implement robust monitoring and documentation
Scaling Chatbot and Kafka Implementation	Might require fine-tuning and additional resources	Perform load testing and auto-scaling validation; provision scalable infrastructure
PDF Storage and Retrieval Across Kafka, S3, DynamoDB	Could lead to synchronization challenges	Design event-driven workflows with retry mechanisms and proper logging
Security Risks from Separate Assistant Management API	Including authentication, authorization, and data protection concerns	Apply OAuth2, encryption, token validation, and regular security audits
Kafka and AWS Service Deployment Without Cost Monitoring	Unexpected budget overruns	Set up AWS Budgets, Billing Alarms, and cost usage reports
Integration of Kafka, S3, DynamoDB, and Chatbot	To prevent delays or message loss	Use event queues, logging, backpressure controls, and performance profiling
Terraform-based CI/CD Deployment Automation	May introduce unforeseen bugs or delays in releases	Test infrastructure-as-code thoroughly and maintain version control with rollback plans

17. Ethical Impact

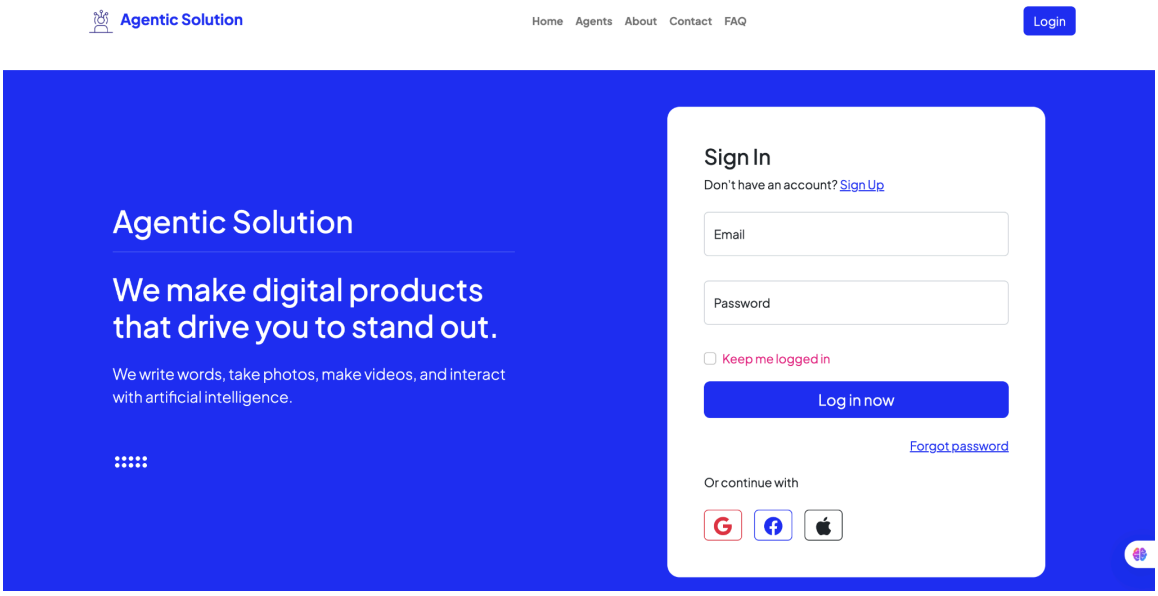
- Replacing human agents with AI agents may reduce employment opportunities.
- AI limitations could lead to poor customer experiences in high-stakes situations.
- IT staff may need re-skilling due to automation.
- Automation may reduce reliance on human roles in backend operation and code review (AWS CodeGuru).
- Security and data privacy must be enforced for file uploads and chatbot interactions.
- Access control and IAM refinement are essential to uphold data ethics.

18. Application Output - Evidences

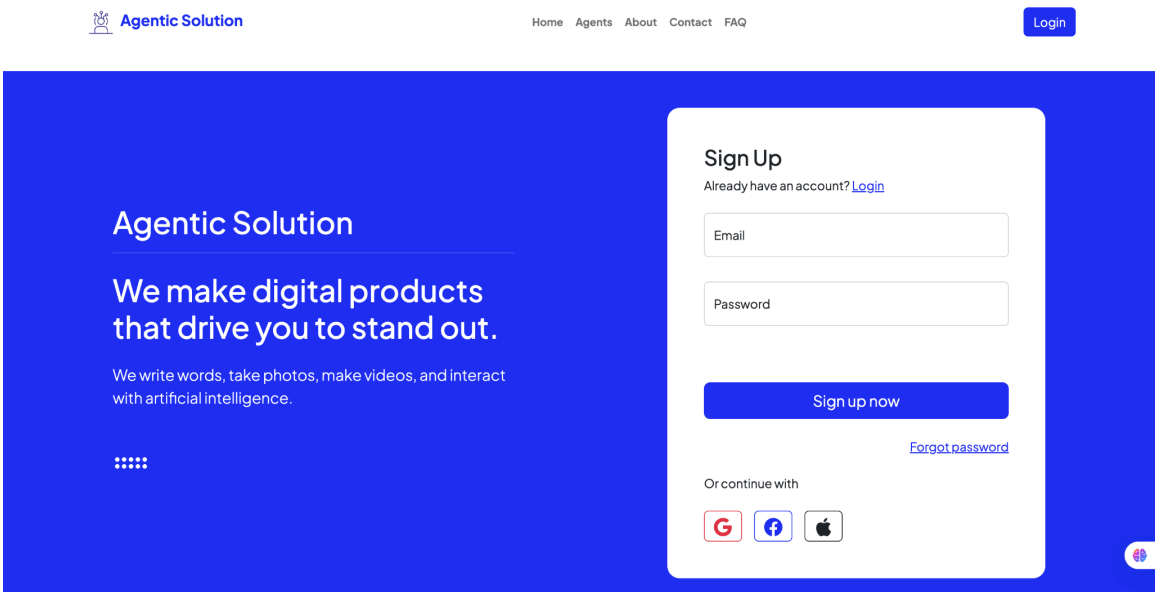
Homepage



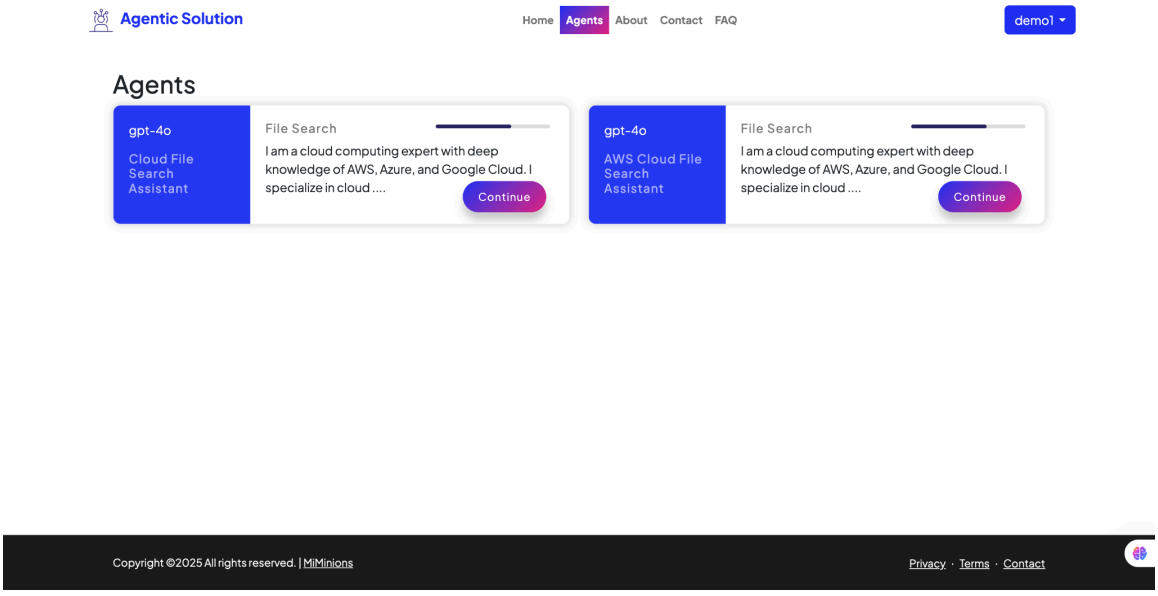
Login



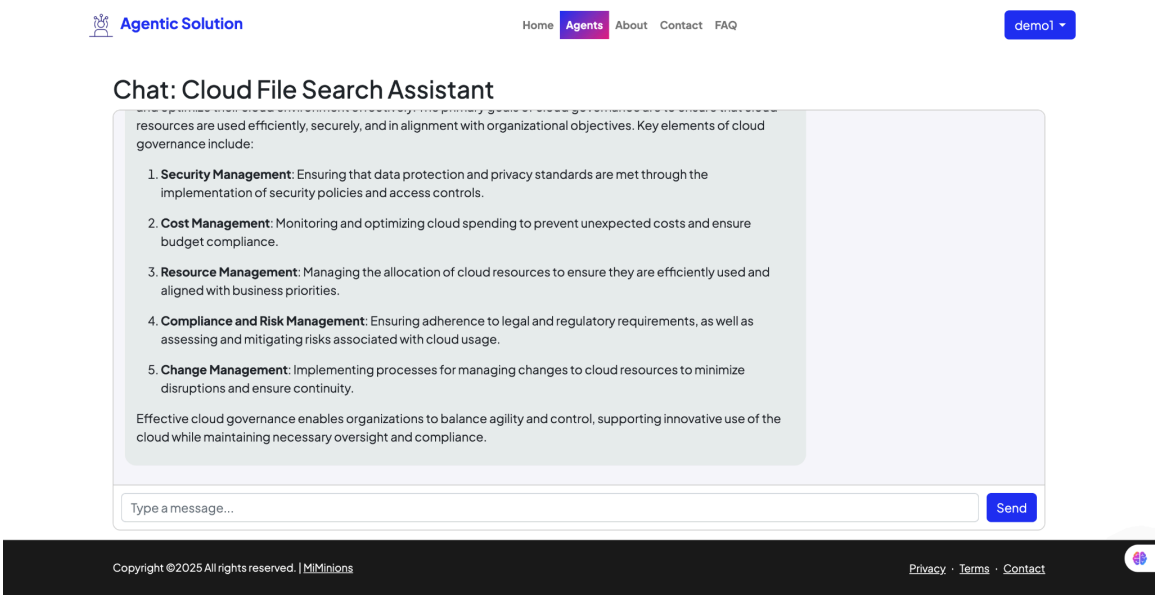
Registration



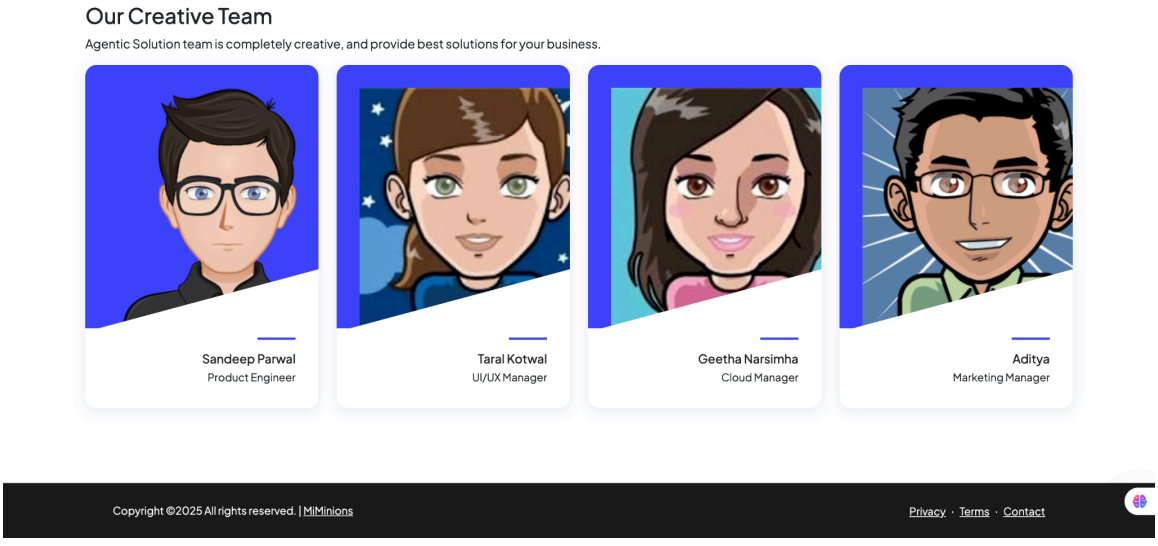
Agents



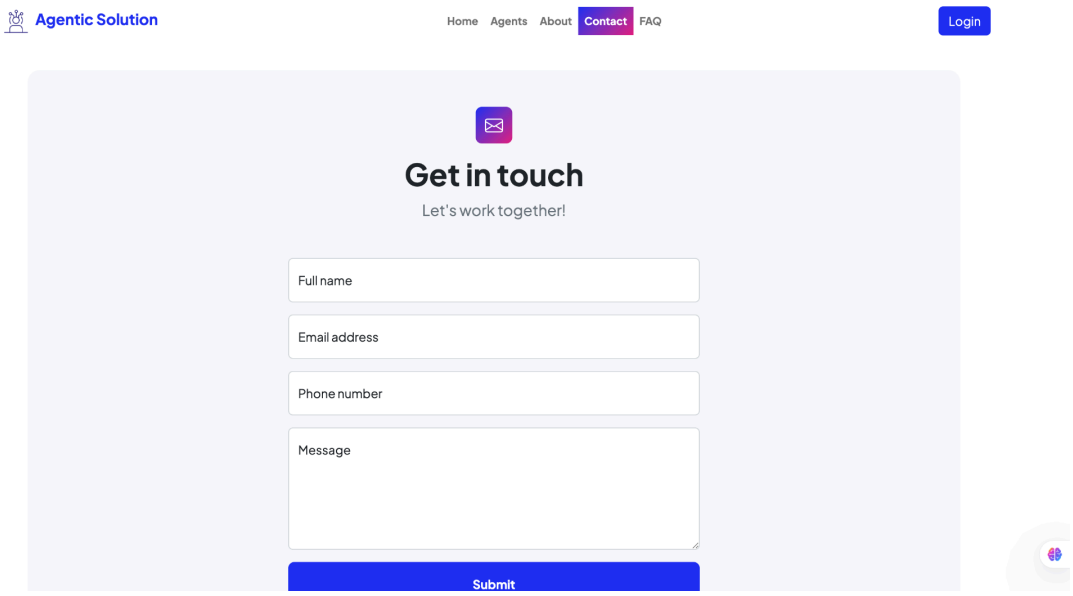
Chat with Agent



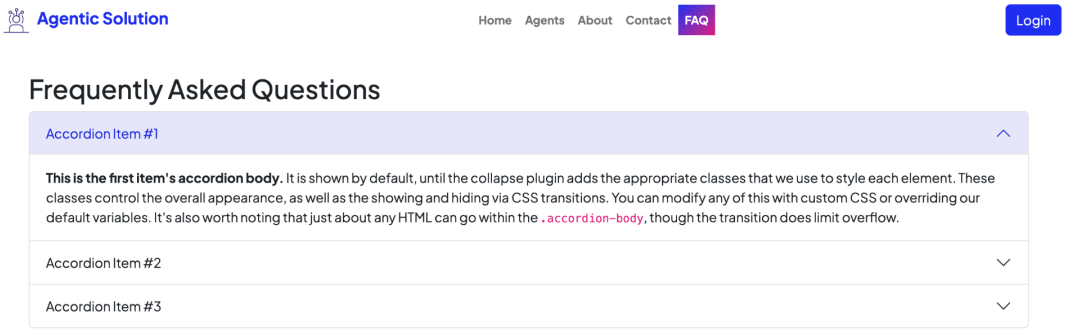
About Us



Contact Us

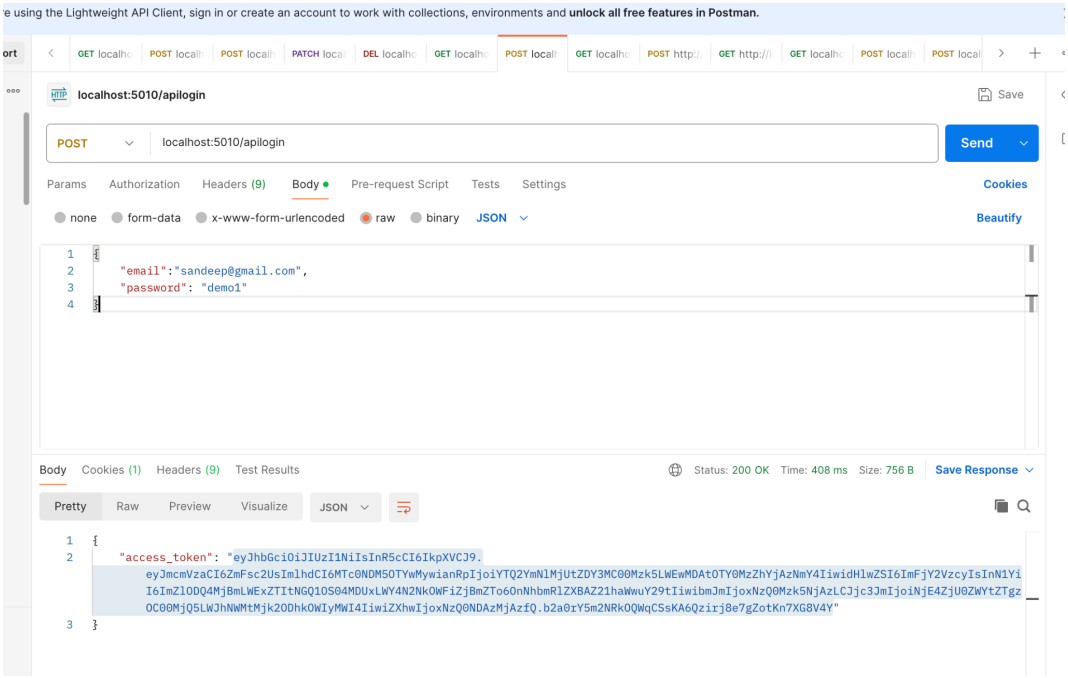


FAQs

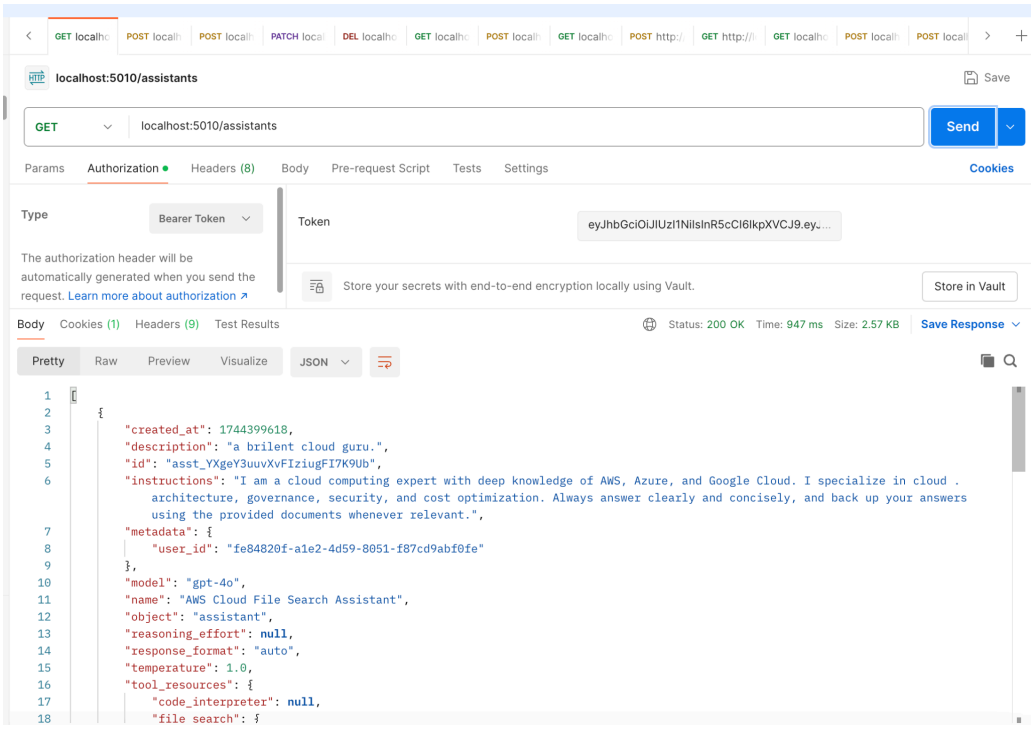


POSTMAN - Admin API

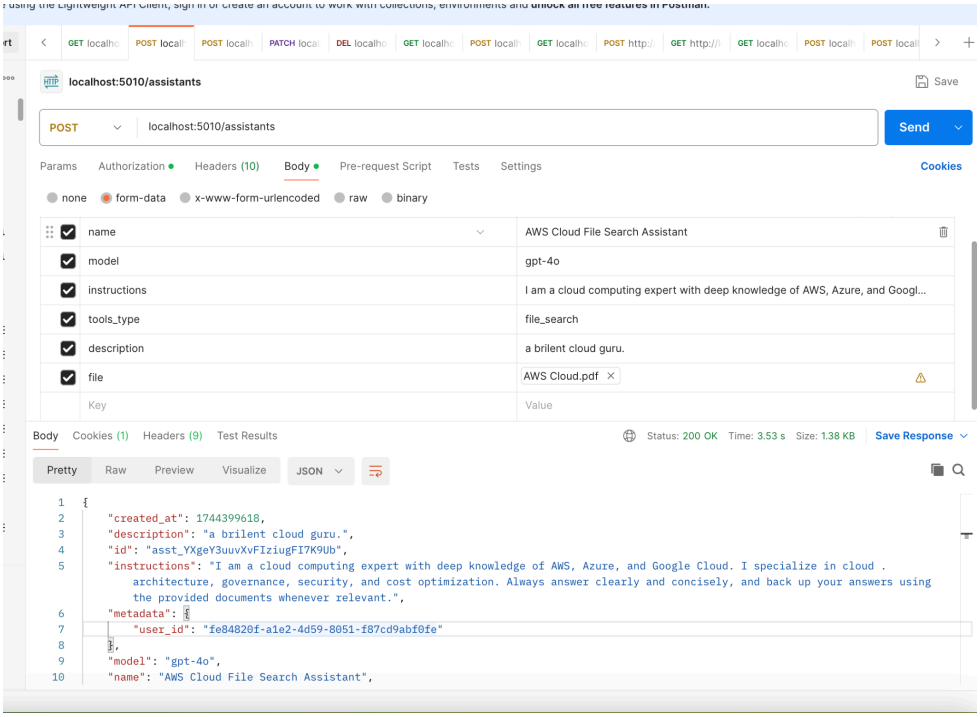
Login



List Agents



Create Agent



19. Deployment Process

Steps to create Docker image and push to Docker hub

1. Clone the project to local and create DockerFile and docker-compose files in the root directory of the project
2. Open cmd and run this command from the root directory with project path

C:\Users\GEETHA NARASIMHA\Desktop\CCP_WinterSem\Capstone Project\bvc-project>docker build -t miminions-ai .

Or

docker-compose build

1. Push the image to docker hub - docker login
2. Tag the image - docker tag bvc-project-flask_app sparwal175/miminions-app:v1.5
3. Push the docker image to docker hub repository - docker push sparwal175/miminions-app:v1.5

Steps to Push Docker image to ECR

1. Create AWS sso session in CLI - aws configure sso
2. Confirm the session related info like shown here
3. Pull the docker image from docker hub - docker pull sparwal175/miminions:latest
4. Login to aws profile - aws sso login --profile AdministratorAccess-248189910523 and validate the code
5. Login to ecr - aws ecr get-login-password --region us-east-2 --profile AdministratorAccess-248189910523 | docker login --username AWS --password-stdin 248189910523.dkr.ecr.us-east-2.amazonaws.com
6. Tag the docker image to the ecr repo - docker tag sparwal175/miminions-app:v1.5 111.dkr.ecr.us-east-2.amazonaws.com/miminions:latest
7. Push the docker image to ecr - docker push 111.dkr.ecr.us-east-2.amazonaws.com/miminions:latest

Steps to push the image from ECR to EC2

- Create EC2 – using terraform (create keypair and give the name in script, instance size : 20GB)
- Modify EC2 adding AIM roles to access ECR
    - docker run -d -p 80:5010 –name miminions-project-v2 -e AWS_REGION=us-east-1 -e AWS_ACCESS_KEY_ID=AKIA2FK66V5xxxxxxx -e AWS_SECRET_ACCESS_KEY=xxxxxxx.dkr.ecr.us-east-2.amazonaws.com/miminions:latest
- Check the OS version - cat /etc/os-release – Amazon Linux 2023
- Based on the OS version run the below commands –
    - sudo dnf update -y
    - sudo dnf install -y docker
    - sudo systemctl start docker
    - sudo systemctl enable docker
    - sudo usermod -aG docker ec2-user
    - newgrp docker
    - docker --version
- login to ecr in ec2 - aws ecr get-login-password --region us-east-2 | docker login --username AWS --password-stdin 248189910523.dkr.ecr.us-east-2.amazonaws.com
- pull the docker image from ecr - docker pull 248189910523.dkr.ecr.us-east-2.amazonaws.com/miminions:latest

- Run the container - docker run -d -p 80:80 248189910523.dkr.ecr.us-east-2.amazonaws.com/miminions:latest
- If different port - docker run -d -p 80:5010 --name miminions-project-v1 248189910523.dkr.ecr.us-east-2.amazonaws.com/miminions:latest
- Use public IP of EC2 or public DNS and run it.

20. Limitations

- **Admin Centric**: The system restricts PDF uploads to admins.
- **Event-driven Processing**: Lambda function.

21. Recommendations

- Use tools like **Kafka(MSK)** to enable real-time processing and autonomous task triggering.
- Integrate **SageMaker Pipelines** for continuous training, testing, and deployment of AI models.
- Enable **Cross-Cloud Communication** to operate across multi-cloud platforms (e.g., AWS, Azure, GCP).