AnswerAI Backend Architecture Documentation

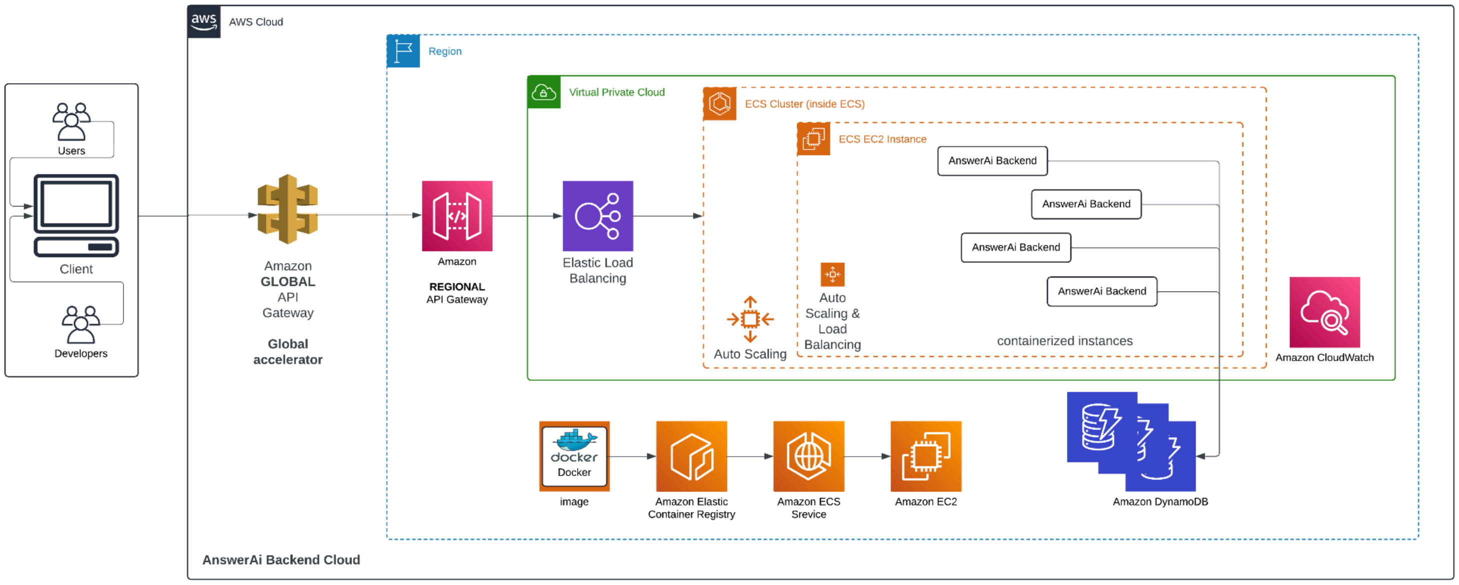
Introduction

This document provides an in-depth overview of the architecture and design of the Answer AI backend application. Leveraging the AWS cloud platform, the architecture incorporates various AWS services to ensure scalability, security, and reliability. The system utilizes microservices architecture, containerization, and database sharding for optimal performance and manageability.

Architecture Overview

The AnswerAI backend application is designed to efficiently handle incoming requests from users globally, process them, and interact with the underlying data storage. Below is an expanded explanation of the components and services, including recent additions and enhancements.

Architecture Design



AnswerAi - Backend AWS Service

Requirements & Scope

Functional Requirments

- Users can submit questions to the platform.
- The system should provide accurate and relevant answers to user questions.

Non-Functional Requirements

- **Availability:** Ensure redundancy, load balancing, auto-scaling, and proactive monitoring to maintain uninterrupted service.
- **Partition Tolerance:** Design for distributed environments with distributed databases and eventual consistency.

Components and Services

1. Global Accelerator

- **Purpose:** Improves the availability and performance of the application by directing user traffic to optimal endpoints.
- **Functionality:** The Global Accelerator utilizes the AWS global network to route user requests to the nearest AWS edge location. This reduces latency and improves the overall user experience.

2. Global API Gateway

- **Purpose:** Provides a single entry point for APIs deployed across multiple regions.
- **Functionality:** The Global API Gateway enables unified API management and deployment across different geographic regions. It ensures consistency and simplifies API access for clients.

3. Regional API Gateway

- **Purpose:** Handles API requests within specific AWS regions.
- **Functionality:** The Regional API Gateway is deployed within each AWS region to manage incoming API requests. It integrates with backend services and can route requests to different endpoints based on routing rules and configurations.

4. Elastic Load Balancer (ELB)

- **Purpose:** Distributes incoming application traffic across multiple targets, ensuring high availability and fault tolerance.

- **Functionality:** The ELB routes traffic to EC2 instances and containerized services within the ECS cluster. It dynamically scales to meet demand and ensures efficient resource utilization.

5. Virtual Private Cloud (VPC)

- **Purpose:** Provides a secure and isolated network environment for the application.
- **Functionality:** The VPC includes subnets, route tables, and security groups to control inbound and outbound traffic. It isolates the application infrastructure and enhances security.

6. Amazon Elastic Container Service (ECS)

- **Purpose:** Orchestrates the deployment and management of containerized applications.
- **Functionality:** ECS manages the lifecycle of Docker containers, including provisioning, scaling, and monitoring. It integrates with other AWS services for seamless deployment and operation.

7. Database Sharding (DynamoDB)

- **Purpose:** Improves scalability and performance of the database by distributing data across multiple shards.
- **Functionality:** DynamoDB is sharded to distribute data storage and processing load. This ensures that the database can handle increasing volumes of data and requests while maintaining low latency.

8. CloudWatch

- **Purpose:** Monitors and logs application and infrastructure performance.
- **Functionality:** CloudWatch collects metrics, monitors logs, and sets alarms to ensure the health and availability of the system. It provides visibility into system performance and enables proactive monitoring and troubleshooting.

Deployment Process

1. **Docker Image Creation:** Develop the Node.js application and containerize it using Docker.
2. **Image Registration:** Register the Docker image with Amazon ECR for easy retrieval and deployment.
3. **ECS Cluster Configuration:** Create an ECS cluster with auto-scaling enabled to manage container instances dynamically.
4. **EC2 Instance Setup:** Launch EC2 instances and register them with the ECS cluster to run containerized services.
5. **Service Deployment:** Deploy the containerized application to the ECS cluster, configuring load balancing and auto-scaling to handle varying loads effectively.

Security Considerations

- **VPC Isolation:** Deploy the application within a VPC to isolate the infrastructure and control network traffic.
- **API Gateway Security:** Implement authentication and authorization mechanisms to secure API endpoints and prevent unauthorized access.

Monitoring and Logging

- **CloudWatch Integration:** Monitor application performance, resource utilization, and system health using CloudWatch. Configure alarms and notifications to respond to performance issues and ensure high availability.

Conclusion

The AnswerAI backend architecture is designed to leverage the scalability, reliability, and security features of the AWS cloud platform. By incorporating global and regional services, database sharding, and robust monitoring capabilities, the application ensures high performance and availability for users worldwide. This documentation provides a comprehensive overview of the architecture, components, deployment process, and security considerations, facilitating understanding and collaboration among development and operations teams.