**High-Level Architecture**

The system follows a serverless microservices architecture designed to perform AI-driven code reviews for GitLab Merge Requests. It consists of three event-driven services communicating via SQS and DynamoDB Streams:

1. **Webhook Listener** – triggers analysis pipeline when an MR is created or updated
2. **Gen-AI Review Service** – performs AI-based code review and stores the result
3. **GitLab Comment Poster** – posts the AI review as a GitLab MR comment

**Layered Architecture Overview**

|  |  |
| --- | --- |
| Layer | Description |
| Presentation Layer | GitLab Merge Request UI — the entry point for developers |
| Application Layer | AWS Lambda functions handling MR events, reviews, and comments |
| Business Logic Layer | Contains core AI review logic, vector search (FAISS), embedding management, and OpenAI integration |
| Data Access Layer | Interfaces with GitLab APIs, DynamoDB, and the FAISS vector index |
| Infrastructure Layer | Built on AWS services (Lambda, SQS, DynamoDB, ECR); Docker images stored in ECR and run in Lambda containers |

**Layered Architecture Diagram**

**A diagram of a software process

AI-generated content may be incorrect.**

**Service 1 : gitlab-webhook-listener**

**A diagram of a software process

AI-generated content may be incorrect. A diagram of software development

AI-generated content may be incorrect.**

**Service 2: Pre-processor & gen-AI**

**A diagram of a diagram of a writer

AI-generated content may be incorrect.**

**Service 3 : gitlab-writer**

**System Architecture Patterns**

* **Architectural Style:** Serverless Microservices
* **Design Patterns:** 
  + Event-Driven Architecture
  + Retrieval-Augmented Generation (RAG)
  + Lambda Container Pattern
* **Communication Protocols**
  + REST (GitLab API)
  + HTTPS (OpenAI API)
  + Internal AWS service calls
* **Message Queuing:** 
  + AWS SQS (between Webhook Listener and Review Service)
  + AWS DynamoDB Streams (between Review Service and Comment Poster)

**Infrastructure Architecture**

|  |  |
| --- | --- |
| Component | Details |
| Deployment Environment | AWS Cloud |
| Container Strategy | Docker images stored in Amazon ECR |
| Load Balancing | AWS Lambda handles automatic scaling; SQS ensures load buffering |
| Content Delivery | Not applicable; no static web content. |
| Monitoring & Logging | AWS CloudWatch for logs and metrics. |

**Network Architecture**

|  |  |  |
| --- | --- | --- |
| Component | | Description |
| **Network Topology** | | Distributed, event-driven with asynchronous message passing |
| **API Gateway** | | Exposes a secure HTTPS endpoint for GitLab Webhooks |
| **Service Mesh** | | Not used (no Kubernetes); loose coupling via AWS SQS and Streams |
| **Security Boundaries** | IAM role-based access control for each Lambda Secrets (GitLab tokens, OpenAI keys) stored securely in environment variables. | |

**Data Architecture**

|  |  |
| --- | --- |
| Component | Description |
| Data Sources | GitLab Merge RequestsLocal indexed codebase (default branch) |
| Data Pipeline | |  | | --- | |  |  |  | | --- | | Codebase is pre-processed and embedded using OpenAI  Vector data stored in FAISS index | |
| Data Warehouse/Lake | Not used explicitly  Embedding storage simulated with FAISS + JSON + Pickle metadata |
| Real-time Processing | Yes — reviews are generated in near real-time based on incoming MR events |

**Disaster Recovery & Business Continuity**

|  |  |
| --- | --- |
| Component | Strategy |
| Backup Strategy | Implementation is planned as a future enhancement to support disaster recovery. |
| Recovery Procedures | Lambdas can be redeployed from ECR |
| Failover Mechanisms | AWS SQS retry logic and Lambda concurrency scaling act as automatic failover |
| Business Continuity Plan | Entire pipeline is stateless and recoverable; messages can be replayed from DLQ or reprocessed with minimal downtime |