Automated Code Review System Design

Comprehensive Architectural Blueprint

# Prepared by:

* Gary A. Smith
* Virtually Non-Existent LLC

# Prepared for:

* Gen AI Capstone
* Interview Kickstart

# Date:

* May 11, 2025

Table of Contents

[1. Introduction 3](#_Toc197856314)

[2. Overall Description 3](#_Toc197856315)

[2.1 System Functions 3](#_Toc197856316)

[2.2 User Characteristics 3](#_Toc197856317)

[2.3 Design Constraints 3](#_Toc197856318)

[2.4 Design Decisions 3](#_Toc197856319)

[2.5 Next Steps 4](#_Toc197856320)

[3. System Architecture 4](#_Toc197856321)

[3.1 Logical Design 4](#_Toc197856322)

[3.2 Architecture Design 5](#_Toc197856323)

[3.3 Cloud Infrastructure Components 5](#_Toc197856324)

[3.4 Application Components 5](#_Toc197856325)

[3.5 System Interactions 5](#_Toc197856326)

[3.6 Data Flow 6](#_Toc197856327)

[4. Detailed Design 6](#_Toc197856328)

[4.1 Module Specifications 6](#_Toc197856329)

[**4.2 Database Design** 11](#_Toc197856330)

[5. Build and Deployment 11](#_Toc197856331)

# Introduction

This system design document will cover the design, architecture, implementation, deployment and support of the Automated Code Review project.

The goal of the application is to save time and to improve the quality of code reviews. Code reviewers are typically busy people and may do a less than through code-review. This tool will save time by allowing the reviewer to simply review the review and submit an approval decision.

The tool is integrated with Github and all processes from the opening of the pull request to the final reviewed displayed on the review page are fully automated, performant, and automatically scalable.

# Overall Description

The system is fully hosted in the cloud on AWS. Custom system components are written using common languages and frameworks such as .Net C# and Python.

An LLM, currently Anthropic Sonnet 3.7, is the engine of the code-review.

### 2.1 System Functions

The primary function of the system is to automatically trigger a code-review by an LLM and to deliver the code-review in near real-time to the reviewer. To enable the reviewer to submit decisions.

### 2.2 User Characteristics

Users of the system include Software Development Engineers – SDE’s and Development Lead or peer reviewers.

### 2.3 Design Constraints

* Host the entire system within AWS to provide a blueprint for an enterprise grade implementation.
* Minimal cost to operate and to process reviews.
* Must be performant from the user perspective.
* Page loads < 2 seconds, API response < 500ms.

### 2.4 Design Decisions

* Bedrock was used initially to experiment with on the RAG implementation. While it was easy to use, took care of loading style documents from the web, chunking, embedding and ingestions, it also had a cost while not in use so I removed it.
* Reviews are stored in DynamoDB to create optimal response time for the Reviewer UI. The review takes about 15 seconds in total, so this was done on the backend before publishing to the UI. When the user clicks on a PR item in the UI details are fetched from the database instead of being generated on the fly.

### 2.5 Next Steps

* Implement RAG using Chroma and deploy to AWS.
* Add LangChain and memory to capture user feedback.
* Implement LiteLLM to enable a fallback approach.
* Implement and SQS queue to fix a problem with distributed messaging in the current architecture.
* Make the review editable and run a diff to capture changes to the feedback mechanism.
* Provide a reward system such as XP and a leaderboard to encourage feedback to improve the model.

# System Architecture

### 3.1 Logical Design

A diagram of a service

AI-generated content may be incorrect.

### 3.2 Architecture Design

A diagram of a software service

AI-generated content may be incorrect.

### 3.3 Cloud Infrastructure Components

* Github
* Github Actions
* Route53 – Domain Name Services
* API Gateway
* Application Load Balancer
* Lambda
* Simple Notification Service
* Dynamo DB
* Anthropic Sonnet 3.7
* Amazon Bedrock (Removed)
* Review Web Page

### 3.4 Application Components

* Github Workflow – initial trigger of the automated system.
* Request Service – processes raw requests.
* Review Service – orchestrates the code-review and persist.
* Notifications Service – interacts with reviewer web page.
* Reviewer’s Web page – Landing page for the review to evaluate and decision automated code-reviews.

### 3.5 System Interactions

* Github

The system is highly integrated with Github to trigger the initial workflow and via the use of Github APIs to gather Pull Request Details.

* Anthropic Sonnet 3.7

The core component of the system is the Sonnet LLM to produce code reviews.

### 3.6 Data Flow

1. An SDE logs into Github and creates a new PR.
2. The Open PR event triggers a custom Github workflow.
3. The Github workflow makes a REST POST call to an AWS based Python Lambda, the "Request Service".
4. The Request Service receives Pull Request metadata from the Github POST.
5. The Request Service packages the PR metadata into an SNS message and posts to an SNS Topic "Request".
6. The "Review Service", another AWS based Python Lambda, subscribes to the SNS request topic and receives the message posted by the Request Service.
7. The Review Service:
   1. Makes a call to Github to gather details, including diffs for each to the files included in the PR.
   2. Makes a call to a chroma vector store to get coding style context.
   3. Makes a call to an LLM such as Anthropic Sonnet 3.7 submitting the code diffs and the style context to get a code review.
   4. Stores the PR metadata and the review from the LLM in a DynamoDB table "PRReview".
   5. Posts PR metadata to an "Review" SNS topic.
8. The Notifications Service, an ECS hosted container, a .Net C# WebAPI, is a subscriber to to the SNS Review topic.
9. The Notifications Service receives the SNS Review message.
10. If there are any Reviewer UI clients connected via websockets, the Notifications service pushes the new PR item to the Review UI.
11. Reviewers login to the Reviewer UI, an HTML/Javascript/Bootstrap application also running in a container in ECS.
12. Upon first launch of the Reviewers UI, the UI makes a REST GET call to the Notifications Service to retrieve all open PRs currently stored in DynamoDB.

# Detailed Design

### 4.1 Module Specifications

#### Github Actions Automated PR Workflow

**System Function:**

* + Event driven workflow triggered on Pull Request Open Events.

**Type**: Github Actions Workflow

**Language**: YAML

**Interfaces**:

**Event:**

**On Pull Request**

* + **Opened**
  + **Reopened**

**Uses curl to POST to the request service**

#### Request Service

**System Function:**

* + Receives a call from the Github Actions Automated PR workflow with an incoming payload containing Pull Request metadata

**Type**: Lambda

**Language**: Python 3.11

**Interfaces**:

* + POST <https://api.codeominous.com/prrequest>
  + Send Message SNS “Request” Topic

Sample Input Payload:

A computer screen shot of a program

AI-generated content may be incorrect.

* Review Service

**System Function:**

* + Receives an SNS “Request” topic message from the Request Service.
  + Gathers pull request details from Github.
  + Pulls contextual info from the vector store.
  + Submits a code-review request to LLM.
  + Persists code review and PR metadata to Dynamo DB.
  + Publishes new review event to the SNS Topic “Review”.

**Type**: Lambda

**Language**: Python 3.11

**Interfaces**:

* + SNS Subscription on the “Request” Topic
  + Vector Store
  + Dynamo DB
  + SNS Publish to the
  + Send Message SNS “Review” Topic.
* Notifications Service

**System Function:**

* + Receives an SNS “Review” topic message and pushes a new pull request item to the Reviewer web page via web sockets in near real-time.
  + Exposes a REST API for the Reviewer UI to retrieve all open PRs.

**Type**: ECS managed Docker Container

**Language**: .Net 9, C# 12 webapi

**Interfaces**:

* + Web socket connection to the Reviewer UI.

**URL**: https://notifications.codeominous.com

* + GET /routes

Returns a list of endpoints supported by the API

* + GET /health

Health check endpoint

* + GET /openprs

Used from the Review UI “List” view at startup to retrieve all open pull requests.

* + GET /details

Used from the Review UI “Detail” view when the user selects a pull request list item to get review details.

* + POST /feedback

Stubbed function used from the Reviewer UI to submit feedback concerning the quality of the automated code review.

* + POST /decision

Used from the Reviewer detail view to submit an approval decision; Approve or Request Changes.

* + Receive

Method subscribed to the SNS Review Topic to receive new pull request reviews.

Sample SNS “Review” message Payload:

A close-up of a computer screen

AI-generated content may be incorrect.

* Review UI

**System Function:**

User Interface for Tech Lead or Peer reviewers to review the automated code reviews, to submit feedback, and submit an approval decision.

**Type**: Kestral hosted web page

**Language**: HTML, Javascript, Bootstrap

**Interfaces**:

<https://notifications.codeominous.com/index.html>

Two views; “List” and “Detail”.

List View:

A screenshot of a computer

AI-generated content may be incorrect.

Detail View:

A screenshot of a computer

AI-generated content may be incorrect.

### 4.2 Database Design

Type: Amazon Dynamo DB

Tables: PRReviews

Items:

* prId (string): Primary key. Composite key <Repo>#<Pull Request Number>
* metadata (string): pull request metadata from Github
* prState (string): Current PR state - “Open” or “Closed”
* review (string): detailed review text from the LLM.
* reviewTitle (string): Title of the review from Github

# Build and Deployment

Infrastructure

Cloud Formation

aws cloudformation deploy \

--template-file template.yaml \

--stack-name pr-notifications-stack \

--parameter-overrides file://parameters.json \

--capabilities CAPABILITY\_NAMED\_IAM \

--region us-east-1

Request and Review Service Lambdas

1. Build

sam build

2. Deploy

**\*\* From each lambda project folder**

**pr-request**

**sam deploy ^**

**--s3-bucket codeominous-artifacts ^**

**--stack-name pr-request ^**

**--region us-east-1 ^**

**--capabilities CAPABILITY\_IAM ^**

**--parameter-overrides Stage=dev SnsTopicArn=arn:aws:sns:us-east-1:238338230919:pr-review-standard**

**curl https://<pr-request-api-id>.execute-api.us-east-1.amazonaws.com/dev/health**

**pr-review**

**sam deploy ^**

**--s3-bucket codeominous-artifacts ^**

**--stack-name pr-review^**

**--region us-east-1 ^**

**--capabilities CAPABILITY\_IAM ^**

**--parameter-overrides Stage=dev SnsTopicArn=arn:aws:sns:us-east-1:238338230919:pr-review-standard**

Notifications Service and Review UI

1. Login to ECS

**aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 238338230919.dkr.ecr.us-east-1.amazonaws.com**

1. Build the image from the project folder

**docker build -t notifications-service .**

1. **Tag the image**

docker tag notifications-service:latest 238338230919.dkr.ecr.us-east-1.amazonaws.com/notifications-service:latest

1. Deploy the new image to ECS

**docker push 238338230919.dkr.ecr.us-east-1.amazonaws.com/notifications-service:latest**