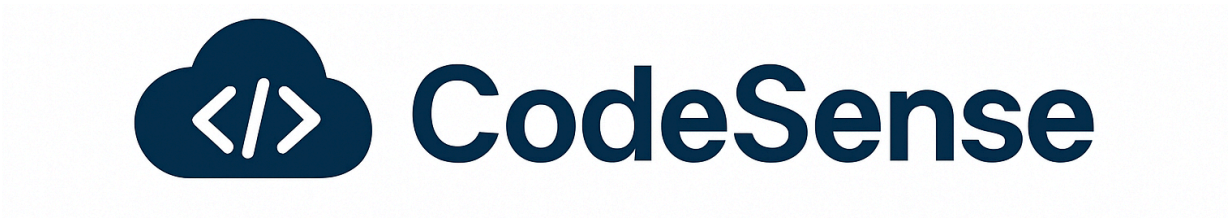

Cloud-Based AI-Powered Code Review and Sanity Check Service

CSCI 4253/5253 Data Center Scale Computing Project

Team Members:

- Atharva Patil - atpa5127@colorado.edu
- Mihir Chauhan - mich3215@colorado.edu

1. Project Goals

CodeSense is striving to create a cloud native, smart service, which automatically analyses the code whenever it is uploaded to a Github repository. The system will give prior catches of typical problems at the beginning of the development lifecycle to guarantee superior quality and acceleration speeds of the iteration. Combining the principle of automation of clou native and artificial intelligence, CodeSense ensures the effortless code review procedure, which allows developers to reduce the number of human interventions.

To be more exact, the project will:

- Auto-check the quality of the code that is bound to the events of the push button in GitHub.
- Smart recommendations such as the feedback in the format of AI-generated improvement recommendations on codes, bugs, and style adjustments.
- Show a distributed and event-driven architecture comprising of serverless and asynchronous messaging.
- Demonstrate scalability/reliability of the cloud infrastructure with several repositories and commits running concurrently.
- Enhance CI/CD by integrating into the day-to-day workflow of the developers.

To start with, CodeSense would be a virtual code reviewer to help in maintaining a clean, efficient, and error-free codebase by demonstrating the true strength of cloud technologies in software automation.

2. Software and Hardware Components

Software Components:

- GitHub webhook events (push/commit signals): These events are received and dispatched to the backend Lambda functions by AWS API Gateway.
- AWS Lambda: Servelessly executes code analysis, syntax validation scripts and static analysis scripts, communicates with AI services to receive suggestions.
- Amazon S3: Reports, logs and artifacts at the analysis have been stored in a manner that they can be accessed and monitored easily.
- Amazon SQS: It executes the queues of parallel independent tasks in order to provide scalability and fault-tolerance across the analysis requests.
- Amazon SES / SNS: Dispatches email or push messages containing summaries of the code review to the developers.
- Amazon Bedrock / OpenAI API: Provides intelligent response on the intelligent analysis of code and AI.
- GitHub: It is a store of information on repositories and a webhook event is raised when a code push happens.
- CloudWatch (AWS): It is used in monitoring and debugging of performance of applications and logging.

Hardware Components:
It does not require any physical hardware since all the parts are then hosted on the cloud. The developers will be tested and deployed using AWS in the local machines of regular standards. All the compute, storage and message processing will be hosted on AWS cloud.

3. Architectural Diagram

CodeSense system architecture: it is an event-driven, stateless workflow that runs automatically to analyze code each time a GitHub commit occurs. The figure below shows the connection between all significant elements:

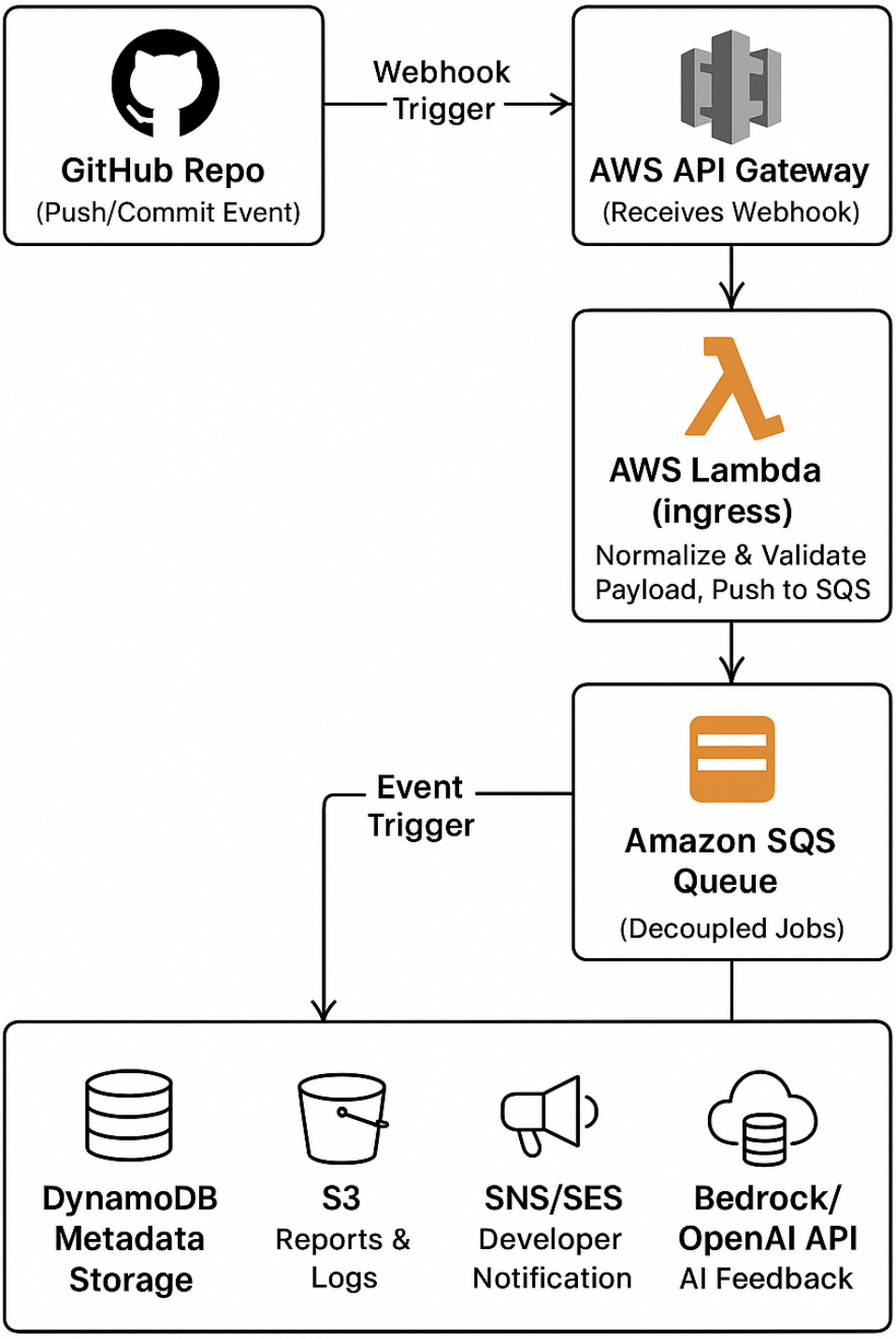

Fig: Project Architecture

Data Flow:

1. Push - GitHub Webhook - API Gateway - developer This pushes code to the GitHub Webhook (API Gateway, p. 1).
2. Gateway messages Lambda - Message in SQS.
3. Lambda worker: Job - AI feedback - save results in DynamoDB/S3.
4. SES/SNS provides summary notification to developer.

It shows the use of a completely serverless, scalable, and asynchronous pipeline with the capacity to support multiple concurrent code review requests.

4. Component Interaction Description

CodeSense workflow is an event based process and it is efficient and isolating between components.

Step-by-Step Flow:

- GitHub Commit Event:
  - The developer makes new code additions to a monitored GIT repository. The repository is wired with a webhook, a payload that provides an event to the API Gateway endpoint of the project.
- AWS API Gateway:
  - API Gateway safely receives the webhook event and makes its authenticity. It forwards the request to an AWS Lambda function (Ingress Lambda) that reads and checks the payload.
- AWS Lambda (Ingress):
  - Ingress Lambda extracts commit data, and forwards a job message to Amazon SQS. This is to ensure that the system can support the multiple pending commits that can operate asynchronously without lost events.
- Amazon SQS:
  - The message queue is also used as a buffer as it helps decouple the webhook ingestion and the processing logic. Each new commit will use a message and these will trigger the Worker Lambda which will be analyzed.
- AWS Lambda (Worker):
  - The Worker Lambda receives the message of the job, reads the corresponding repository or commit data and performs the tools of the static code analysis (linters, syntax, and other validators and so on). It then sends some of the code and context to Amazon Bedrock or OpenAI API to receive AI generated suggestions of reviews.
- Amazon Bedrock / OpenAI API:
  - The smart suggestions provided by the AI service consists of the descriptions of bugs, the best practices or the stylistic changes, depending on the code context analysis. Results outputs are then combined with the output of the results of the static analysis to form a completed review report.
- Amazon S3:

    - The artifacts stored in the S3 storage are all reports on the analysis, logs, and AI recommendations. These come in form of pre-matched URL with developers.
- Amazon SES / SNS:
    - After processing, the synopsis of the findings is sent to the developer (SES) or pushed (SNS). It includes a short description of the message and a general report with its full content in S3.
- Monitoring and Debugging:
    - All the Lambda and all the AWS elements capture their events and metrics into Amazon cloudwatch which can be utilized by the developers to trace executions, error detection, and performance.

This interactive stream of interaction contributes to CodeSense being a modular, resilient and event-driven system. The functions of the services offered by AWS are different: API Gateway is applied to implement the secure ingress, SQS is applied to ensure the reliability, scalability, Lambdas is applied to implement the calculations and orchestrations, DynamoDB and S3 are applied to ensure the persistence, and Bedrock/OpenAI is applied to offer the AI intelligence which makes the system unique.

5. Debugging and Testing Strategy

In order to ensure that CodeSense remains reliable and accurate, one will have to approach the process of debugging and testing with the structure that would guarantee that the infrastructure of the clouds is verified as well as the logic of the AI-based analysis is adequate. Tests based verification workflow, a combination of local testing, and AWS monitoring tools will be used in the project.

- **1. Unit and Integration testing.**
    - Local testing AWS SAM CLI and pytest of every AWS Lambda function (Ingress and Worker) will be used to validate the logic before it is deployed.
    - False messages will be applied to make sure that the parsing, validation, and the generation of SQS messages are correct by simulating a GitHub webhook event.

- **2. Logging and Monitoring**
    - There will be a structured logging of each Lambda function, that will document significant events such as the creation of a job, AI requests, and report storage.
    - The Amazon CloudWatch logs and Amazon CloudWatch metrics will be used to monitor the execution times, the error rates as well as the invocation counts.

- **3. AI Feedback Validation**
    - The recommendations suggested by AI generated by either Amazon Bedrock or OpenAI API will be measured on sample codebases to obtain a contextual fit and useful feedback.

    - The review of the output of the AI will be performed manually first of all to make sure that the suggestions comply with the standard practice of the code quality.

- **4. Data Validation and Consistency Checks**
    - Post processing scripts will report and validate stored metadata in DynamoDB and generated files in S3.
    - verify hash id check or special job id check will ensure that all the analysis of commit are linked to the report stored which will make CodeSense robust, scalable and testable and will be transparent in the manner the results are retrieved and conveyed.

6. Compliance with Project Requirements

The CodeSense project is particularly designed to pursue and exceed the course goal of implementing four variant types of cloud technologies. It offers a combination of multiple AWS services - each of which plays a critical part in the entire workflow - alongside benefits improved through AI of Amazon Bedrock or the Open AI API.

Use of Four (or More) Cloud Technologies

- **AWS API Gateway:**
  The safe scaling entry point of incoming webhook events of GitHub. It does authentication, request throttling and routing to the backend Lambda functions.

- **AWS Lambda:**
  A serverless architecture is used to do all backend. A webhook ingestion, job creation, and code analysis functions are carried out by use of lambda functions. This service ensures scalability, and it also does not need manual infrastructure management.

- **Amazon SQS (Simple Queue Service):**
  Plays the role of the asynchronous job queue which isolates the ingestion and processing layers. It provides scaling, fault tolerance, reliability by buffering requests as well as parallel workloads.

- **Amazon S3 (Simple Storage Service):**
  Reports, logs and output of analysis were made by the stores. It is also cost efficient and durable hence it is most appropriate to be used to carry on with structured and unstructured data produced by the system..

- **Amazon SES / SNS:**
  Prepares reports overview and connects through review to programmers. It is possible to

deliver the feedback to a subscribed endpoint or an inbox of a developer through these services without any problems.

- **Amazon Bedrock / OpenAI API:**
  Provides a contextual analysis using AI and feedback in natural language of submissions. This kind of integrating contributes to the sheer analysis of the code, where the developers have a clue as to why certain changes in code are recommended.

These aspects constitute a multi faceted and dependent consumption of the offerings of cloud computing- including compute (Lambda), messaging (SQS), data persistence (S3, DynamoDB), artificial intelligence (Bedrock/OpenAI), and communication (SES/SNS).

Alignment with Cloud Project Requirements

The project meets the major educational aims proving:

- **Event based, distributed architecture**: Webhook, SQS queues, and asynchronous Lambda triggers.
- **Serverless architecture and scalability**: Using fully managed AWS services without specifying server configuration.
- **Metadata persistence and report storage**: DynamoDB and S3 respectively.
- **AI and automation**: With Bedrock/OpenAI, use intelligent suggestions on code.
- **CI/CD relevance**: The integration with GitHub workflows guarantees the applicability in the real development pipeline.

Summary of Cloud Technologies Used

Category	**AWS/External Service**	**Function**
Compute	AWS Lambda	Executes code review logic
API Gateway	AWS API Gateway	Handles webhook requests
Messaging	Amazon SQS	Job queuing and decoupling
Storage	Amazon S3	Stores reports and logs
Database	Amazon DynamoDB	Stores job and analysis metadata

Notifications	Amazon SES / SNS	Sends email and push updates
AI Services	Amazon Bedrock / OpenAI API	Generates intelligent review suggestions

Table: Summary of Cloud Technologies Used

With this design, CodeSense achieves the same level of technical richness and complexity demanded of a cloud computing project and, at the same time, can be practical, scalable and have an instructional value that is clearly evident.

7. Scope and Ambition Discussion

The CodeSense project will be a big but feasible project regarding the timely completion (semester). It also incorporates the API Gateway, Lambda, SQS, DynamoDB, S3, SNS/SES, and Bedrock/OpenAI using a mix of multiple AWS services to construct a fully serverless and scalable smart code review pipeline. They all are necessary building blocks to achieve automation, asynchronous processing and AI-driven insights, which are reflective of an architecture which is used in production-grade systems.

The project ambition is the ambition to automatize and optimize the processes of code review using the assistance of AI and maintain the best practices that are cloud-native. CodeSense is also able to use machine learning through AWS Bedrock or OpenAI APIs to test the quality of code, possible bugs, and optimizations - simulating how professional CI / CD systems can be combined with intelligent review systems like SonarQube or GitHub Copilot.

Education/technical wise, the system has been found to be competent in a number of big areas:

- Serverless computing (AWS Lambda)
- Event-based design (API Gateway + SQS + SNS)
- Affinities: data storage and management (Cloud storage, DynamoDB, S3).
- The integration of AI (Bedrock/OpenAI)
- DevOps automation (GitHub Webhooks to analyze on a continuous basis)

Despite the current scope of the project, to create a working end-to-end pipeline of one programming language (e.g., Python), the project remains small enough and can be finished within the time constraints of one semester. Its modular architecture ensures that each AWS component may be deployed and tested in stages and it can be tracked and collaborate as a team in manageable progress milestones.

Potential Stretch Goals:

- Translation, to more programming languages (Java, C++, JavaScript, etc.).

- Developing a web-based dashboard that will display review statistics and AI feedback of DynamoDB and S3.
- Integrating the GitHub Actions/CI/CD pipelines with automatic deployment of the code review.
- The sentiment or tone analysis of the comments of reviews to stimulate positive feedback.
- Enabling the customization of AI models to be specialized to a set of coding standards or group conventions.

To conclude, CodeSense is a middle ground between innovation and feasibility - it stretches AI-driven code intelligence to the limits but is still feasible within the framework of a semester-long project.

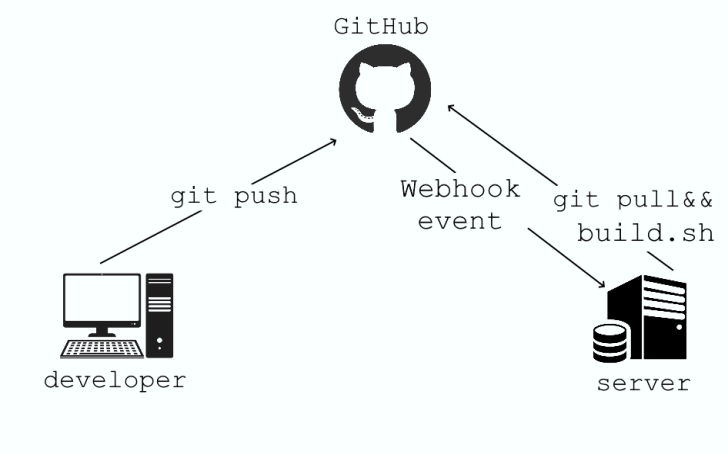

Fig. Github Webhooks Demo

8. REFERENCES

1. GitHub Docs. “Webhooks.” https://docs.github.com/en/webhooks
2. GitHub Docs. “Webhook events and payloads.” https://docs.github.com/en/webhooks/webhook-events-and-payloads
3. GitHub Docs. “Validating webhook deliveries.” https://docs.github.com/en/webhooks/using-webhooks/validating-webhook-deliveries
4. Amazon Web Services. “Amazon API Gateway Documentation.” https://docs.aws.amazon.com/apigateway/
5. AWS Prescriptive Guidance. “Integrate Amazon API Gateway with Amazon SQS to handle asynchronous REST APIs.” https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/integrate-amazon-api-gateway-with-amazon-sqs-to-handle-asynchronous-rest-apis.html
6. Amazon Web Services. “AWS Lambda Documentation.” https://docs.aws.amazon.com/lambda/
7. AWS Lambda User Guide. “What is AWS Lambda?” https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
8. Amazon Web Services. “Amazon Simple Queue Service (SQS) Documentation.” https://docs.aws.amazon.com/sqs/
9. SQS Developer Guide. “What is Amazon SQS?” https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html
10. Amazon S3 User Guide. “Download and upload objects with presigned URLs.” https://docs.aws.amazon.com/AmazonS3/latest/userguide/using-presigned-url.html
11. Amazon S3 User Guide. “Sharing objects with presigned URLs.” https://docs.aws.amazon.com/AmazonS3/latest/userguide/ShareObjectPreSignedURL.html
12. Amazon Web Services. “Amazon DynamoDB Documentation.” https://docs.aws.amazon.com/dynamodb/
13. DynamoDB Developer Guide. “What is Amazon DynamoDB?” https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html
14. Amazon SES Developer Guide. “What is Amazon SES?” https://docs.aws.amazon.com/ses/latest/dg/Welcome.html
15. Amazon SNS Developer Guide. “What is Amazon SNS?” https://docs.aws.amazon.com/sns/latest/dg/welcome.html
16. Amazon CloudWatch User Guide. “What is Amazon CloudWatch?” https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html
17. AWS X Ray Developer Guide. “What is AWS X Ray?” https://docs.aws.amazon.com/xray/latest/devguide/aws-xray.html
18. AWS X Ray with Lambda. “Visualize Lambda function invocations using AWS X Ray.” https://docs.aws.amazon.com/lambda/latest/dg/services-xray.html
19. Amazon Bedrock User Guide. “Submit a single prompt with InvokeModel.” https://docs.aws.amazon.com/bedrock/latest/userguide/inference-invoke.html