CodeSense

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 aims to create a smart, cloud-based service that will automatically analyze the source code with every commit to a GitHub repository. The system is to identify typical problems during the initial stages of the development process to guarantee superior quality of code and increase the rate of the iteration. Integrating the automation of cloud-native with the help of AI-driven insights, CodeSense simplifies the code review process and helps developers to minimize manual overhead.

To be more specific, the project will:

- Automate quality checks of code by connecting with the events of the push button in GitHub.
- Use AI to give intelligent feedback in the form of suggestions to improve code, find bugs and suggest stylistic enhancements.
- Showcase distributed, event-based architecture based on serverless components and asynchronous messaging.
- Demonstrate the scalability and reliability of the cloud infrastructure, with several repositories and commits at the same time.
- Facilitate CI/CD improvement by fitting in the regular developer workflows.

Primarily, CodeSense is a virtual code reviewer that would assist in keeping codebases cleaner, more efficient, and error free as it would demonstrate the actual power of cloud technologies in software automation.

2. Software and Hardware Components

Software Components:

- AWS API Gateway: GitHub webhook events (push/commit triggers) are received and sent to the backend Lambda functions.
- aws lambda: Runs code analysis serverlessly, executes syntax validation scripts, and static analysis scripts, and interacts with AI services to get recommendations.
- Amazon S3: Reports, logs, and artifacts in the analysis have been stored in such a way that they can be easily accessed and monitored.
- Amazon DynamoDB: It is a NoSQL database where the metadata on code reviews, job status, and past results of analysis are stored.
- Amazon SQS: It runs the queues of parallel independent tasks to maintain scalability and fault-tolerance among various analysis requests.
- Amazon SES / SNS: Sends email or push messages with summaries of the code review to developers.
- Amazon Bedrock / OpenAI API: Offers smart feedback on the intelligent review of code and AI.
- GitHub: It serves as a repository of information concerning repositories and generates a webhook event whenever there is a code push.
- CloudWatch (AWS): It is applied to the monitoring and debugging of the performance of applications as well as logging.
- Development Tools Python SDK (boto3), AWS CLI, GitHub CLI, Visual Studio Code, and Docker (local testing).

Hardware Components:

It does not need any physical hardware because all the components are then hosted in the cloud. The testing and deployment will occur in standard local machines by the developers through AWS. Every compute, storage and message processing will be under AWS cloud infrastructure.

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:

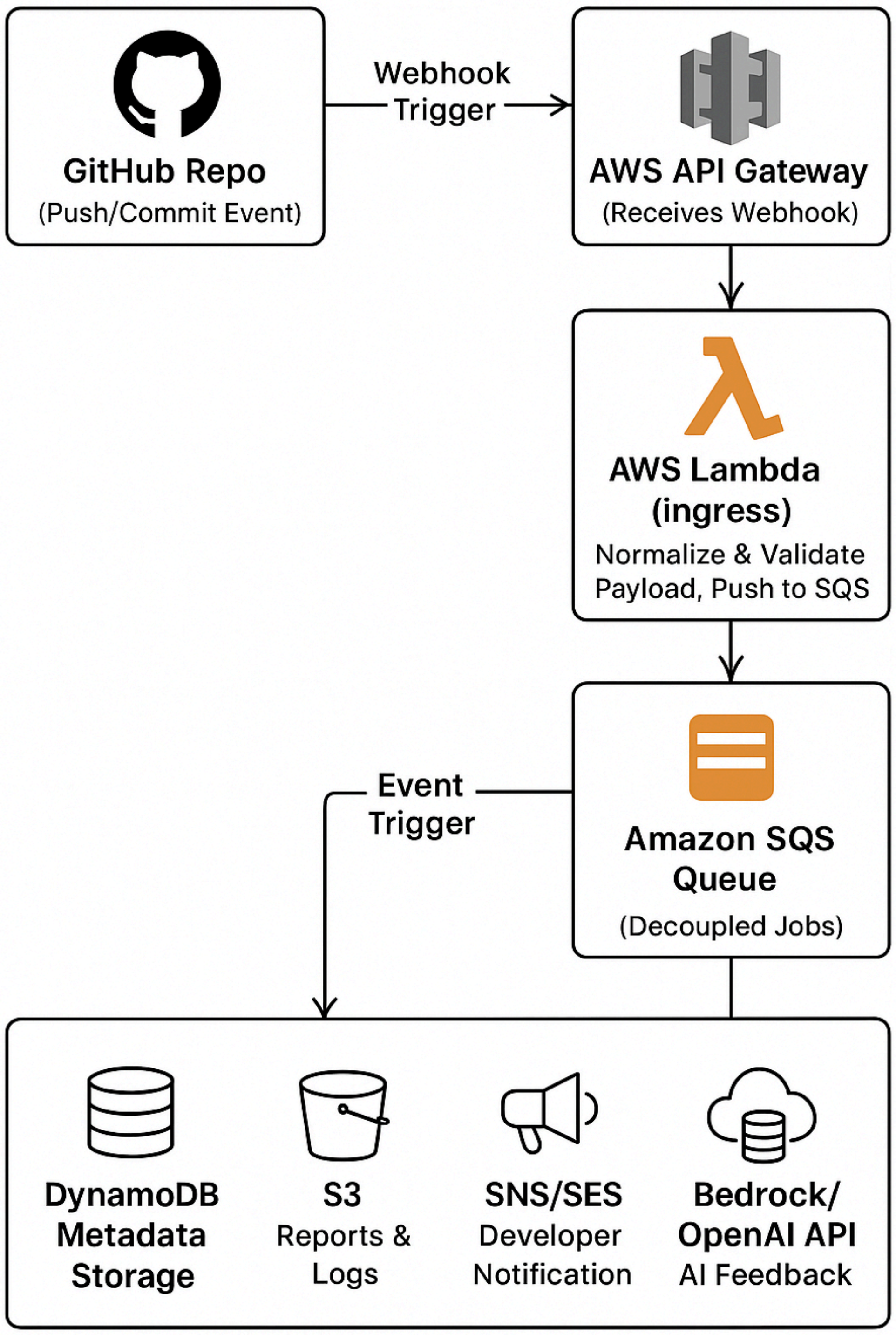

Fig: Project Architecture

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 is efficient and isolating between components.

Step-by-Step Flow:

- **GitHub Commit Event:**
  The developer commits new code to a supervised GitHub repository. The repository has a webhook, which sends an event payload to the API Gateway endpoint of the project.

- **AWS API Gateway:**
  The webhook event is safely received by API Gateway, which establishes its authenticity. It sends the request to an AWS Lambda function (Ingress Lambda) which reads and verifies the payload.

- **AWS Lambda (Ingress):**
  The Ingress Lambda pulls out commit information and sends a job message to Amazon SQS. This is to make sure that the system is able to support multiple concurrent commits which proceed asynchronously without lost events.

- **Amazon SQS:**
  The message queue serves as a buffer, which decouples the webhook ingestion with the processing logic. Every new commit will produce a message and this will invoke the Worker Lambda to be analyzed.

- **AWS Lambda (Worker):**
  The Worker Lambda accesses the job message, retrieves the related repository or commit data, and executes the tools of the static code analysis (linters, syntax, and other validators and so on). It then forwards part of the code and context to Amazon Bedrock or OpenAI API to get AI generated review suggestions.

- **Amazon Bedrock / OpenAI API:**
  The intelligent recommendations offered by the AI service include the explanations of bugs, the best practices, or the stylistic modifications, based on the analysis of the code

context. The outputs of the results are then added to the output of the static analysis to create a complete review report.

- **Amazon DynamoDB:**
  DynamoDB allows storing the processed metadata such as commit ID, repository name, timestamps, and summary of findings and making rapid queries and retrieval. This allows the construction of dashboards or monitoring previous performance.

- **Amazon S3:**
  All reports on the analysis, logs, and AI recommendations are stored in S3 as artifacts. These are available as presigned URLs with developers.

- **Amazon SES / SNS:**
  Once processed, the summary of the results is emailed to the developer (SES) or pushed (SNS). It contains a brief overview of the message and a link to the entire report in S3.

- **Monitoring and Debugging:**
  Every Lambda and every AWS element records their events and metrics to Amazon CloudWatch, which can be used by developers to trace executions, track error detection, and performance.

This flow of interaction makes CodeSense a modular, resilient and event-driven system. The services provided by AWS have different functions: API Gateway is used to perform secure ingress and SQS ensures reliability, scalability, Lambdas is used to perform computations and orchestration, DynamoDB and S3 are used to maintain persistence, and Bedrock/OpenAI is used to provide the AI intelligence that makes the system unique.

5. Debugging and Testing Strategy

To guarantee the reliability and accuracy of CodeSense, it is necessary to approach the process of debugging and testing with the structure that would ensure that both the infrastructure of the clouds is checked and the logic of the AI-based analysis is correct. The project will use test-based verification workflows, a mix of local testing, and AWS monitoring tools.

- **1. Unit and Integration testing.**
  - The logic will be validated by local testing AWS SAM CLI and pytest of each AWS Lambda function (Ingress and Worker) prior to deployment.
  - False messages will be used to ensure that the parsing, validation, and the creation of SQS messages are all correct by emulating a GitHub webhook event.

  - End-to-end testing will be conducted by making sure that Lambda, SQS, S3 and DynamoDB are communicating as intended.

- **2. Logging and Monitoring**
    - Each Lambda function will be tracked with a structured logging, which will record major events like job creation, AI requests, and storage of reports.
    - Execution times, error rates and invocation counts will be tracked using Amazon CloudWatch Logs and CloudWatch Metrics.
    - Alerts will be set to inform the team about the anomalies e.g. SQS queue buildup, Lambda timeouts or AI API errors.

- **3. AI Feedback Validation**
    - The output of AI-generated recommendations provided by Amazon Bedrock or OpenAI API will be evaluated against sample codebases to achieve a contextual fit and helpful feedback.
    - The initial step of review of the AI output will be done manually in order to ensure that the suggestions meet the standard practices of code quality.
    - There will be feedback loops that will be used to streamline AI prompts and response parsing.

- **4. Data Validation and Consistency Checks**
    - Stored metadata in DynamoDB and generated files in S3 will be reported and validated by post processing scripts.
    - Checking of hash or special job IDs will guarantee that every analysis of commit is associated to the report stored.

- **5. System Testing on AWS**
    - The complete system will be tested in a managed AWS facility, and end-to-end tests will be used to simulate a variety of simultaneous commit events to test scalability.
    - SQS and Lambda concurrency will be stress-tested to ensure that the system is able to support realistic workloads.
    - The notification channel through SES/SNS will be tested to ensure that the email has been successfully delivered and report links are available.
- **6. Continuous Monitoring and Debugging Tools**
    - The end to end request flows, latency bottlenecks, and component interactions will be followed with AWS X-Ray.
    - Infrastructure will be managed reproducibly by CloudFormation templates, and rollback is easily achieved, as well as environmental consistency.
    - The logs and traces will be useful in case of errors in order to isolate the fault to components (e.g. API Gateway event, Lambda runtime, or AI API latency)

- **7. Developer Testing and Iteration**
    - The team will have a private GitHub test repository where they will do continuous integration tests, and upon which, they will push small commits and monitor real-time responses, which CodeSense gives.

    - A tested dataset that is versioned and has tests with different code examples will guarantee that the system is able to work across the languages and styles of code.

This debugging and testing approach will keep CodeSense strong, expandable, and testable, and be transparent in the way the results are obtained and told.

6. Compliance with Project Requirements

The CodeSense project is specifically planned to address and surpass the course objective of implementing four different types of cloud technologies. It combines various AWS services - each with an essential role in the entire workflow - with AI-enhanced benefits of Amazon Bedrock or the Open AI API.

Use of Four (or More) Cloud Technologies

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

- **AWS Lambda:**
  Uses a serverless architecture to compute all backend. Lambda functions are used to perform a webhook ingestion, job creation, and code analysis functions. Scalability is guaranteed by this service, as well as manual infrastructure management is not necessary.

- **Amazon SQS (Simple Queue Service):**
  Performs the role of the asynchronous job queue that separates the ingestion and processing layers. It offers scalability, fault tolerance and reliability through buffering requests and handling parallel workloads effectively.

- **Amazon DynamoDB:**
  The NoSQL database used to store the metadata of the jobs, commit identifiers, timestamps and analysis summaries. The low-latency access enables it to efficiently query and support developers with dashboards.

- **Amazon S3 (Simple Storage Service):**
  Stores created reports, logs and analysis outputs. It is cost effective and long lasting, which is why it is best suited to continue structured and unstructured data generated by the system.

- **Amazon SES / SNS:**
  Sends reports summary and links via review to developers. These services allow delivery of feedback to an inbox of a developer or a subscribed endpoint without

difficulties.

- **Amazon Bedrock / OpenAI API:**
  Offers an AI-powered contextual analysis and natural-language feedback of code submissions. Such integration adds a level of intelligence to the mere analysis of the code, making the developers have a clue why some changes in code are suggested.

These elements form a multifaceted and interdependent usage of the services of cloud computing- including compute (Lambda), messaging (SQS), data persistence (S3, DynamoDB), AI (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 ambitious yet realistic in terms of being completed on time (semester). It uses a combination of several AWS services API Gateway, Lambda, SQS, DynamoDB, S3, SNS/SES, and Bedrock/OpenAI to build a completely serverless, scalable, and intelligent code review pipeline. They are all essential building blocks to realize automation, asynchronous processing and AI-driven insights, which are indicative of an architecture which is used in production-grade systems.

The ambition of the project is the desire to automate and optimize code review processes with the help of AI and retain cloud-native best practices. Using AWS Bedrock or OpenAI APIs, CodeSense uses machine learning to test the quality of code, potential bugs, and optimizations - emulating how professional CI / CD systems can be paired with intelligent review tools such as SonarQube or GitHub Copilot.

Education/technical wise, the system has proven competence in several major 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)

Although the existing scope is to develop a working end-to-end pipeline of a single programming language (e.g., Python), the project is still sufficiently limited and feasible within the timeframe of one semester. Its modular architecture guarantees that every AWS component can be deployed and tested progressively and it is possible to follow manageable progress milestones and work together as a team.

Potential Stretch Goals:

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

- Creating a web-based dashboard to represent review statistics and AI feedback of DynamoDB and S3.
- Adding the GitHub Actions/CI/CD pipelines with automatic deployment of code review.
- The analysis of sentiment or tone in the comments of reviews to encourage positive feedback.
- Allowing the customization of AI models to be specialized to certain 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.

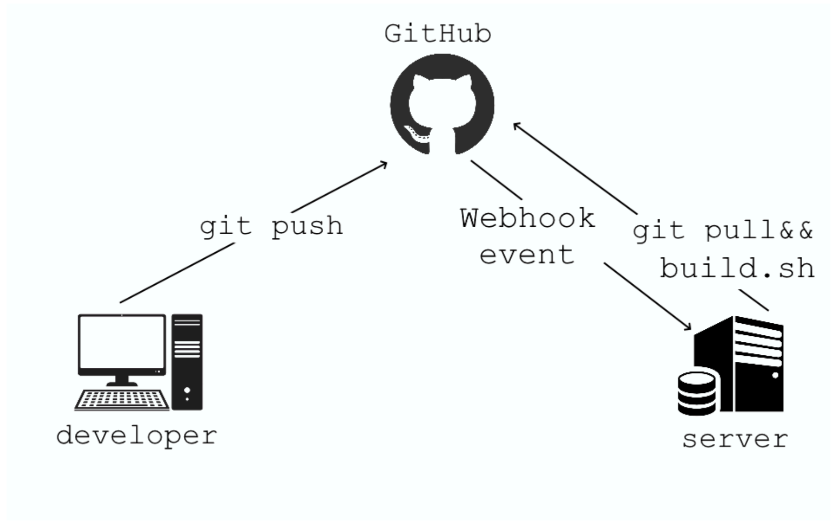

Fig. Github Webhooks Demo

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
20. OpenAI Platform. “Text generation guide.” https://platform.openai.com/docs/guides/text-generation
21. AWS SAM CLI Guide. “Introduction to testing with sam local invoke.” https://docs.aws.amazon.com/se
rverless-application-model/latest/developerguide/using-sam-cli-local-invoke.html
22. AWS SAM CLI Guide. “Introduction to testing with sam local start-api.” https://docs.aws.amazon.com/
serverless-application-model/latest/developerguide/using-sam-cli-local-start-api.htm