CSCI5409

Adv Topic in Cloud Computing

Term Project

Gitlab Link: https://git.cs.dal.ca/courses/2024-summer/csci4145-5409/jspatel

Table of Contents

Overview

Slackify is a cloud-based integration service made to enhance developer productivity by seamlessly connecting GitHub with Slack. The primary objective of Slackify is to streamline communication and collaboration within development teams by automatically synchronizing GitHub repository activities with Slack channels.

When a new issue is created in a GitHub repository, Slackify detects this event and automatically sets up a new Slack channel dedicated to that specific issue. This feature helps in centralizing discussions, tracking, and resolving issues in a focused environment. Conversely, when comments are made on existing issues, Slackify posts these comments in the relevant Slack channels, keeping all team members updated in real time.

Slackify's architecture takes advantage of cloud services to deliver its functionality. Users just need to provide their email, GitHub username and repository name, and GitHub, and Slack tokens.

By automating the creation of channels and notification processes, Slackify removes manual intervention from tracking GitHub activities, thus enhancing team efficiency and response times. The integration aims to keep development teams agile, ensuring they remain informed and can swiftly address issues as they arise.

Target Users

The primary users of Slackify are:

Development Teams: Those using GitHub for version control and Slack for communication.
Project Managers: Who needs to track issue resolutions and team discussions.
DevOps Engineers: Interested in integrating continuous monitoring of GitHub repositories into their workflows.
Open Source Projects: Efficiency across distributed and asynchronous contributors.
QA Teams: Who needs real-time updates on bug reports and issue tracking to streamline testing and quality assurance processes.
Technical Support Teams: Those who need to monitor and respond to issues reported by users in real-time, ensuring quick resolution and communication.

Performance Target

In Slackify, we target reliability and availability for our system. If an issue is created, the corresponding channel must exist with that issue number. For availability, we aim for 24/7 uptime since people can work on the same project from all over the world. It is our responsibility to ensure that everyone is up to date with the latest news about the repository. We also ensure availability by using cloud services, which help in disaster recovery and maintaining continuous operation even in the event of failures.

Requirements

I fulfilled the menu item requirements by choosing the following services:

Compute:
- EC2
- Lambda

Storage:
- RDS

Network:
- API Gateway

General:
- Secret Manager
- SNS

Services

API Gateway

Purpose: API Gateway[1] serves as the entry point for connecting with GitHub and Slack. It is the gateway through which users interact with Lambda, where the connection logic is handled.
Why API Gateway: Direct access to Lambda using its ARN is not feasible; a secure gatekeeper is needed. API Gateway provides this functionality and offers usage tracking through usage

plans, which is not available with Function URL. Additionally, API Gateway generally has better response times compared to Function URL.
Alternative - Function URL[2]: My application is a SaaS solution requiring usage tracking and control, which Function URL does not support.

Lambda

Purpose: Lambda[3] executes code for input verification, SNS subscription, and webhook creation. It also stores user details and tokens in RDS, with configuration data pulled from the Secret Manager.
Why Lambda: Lambda provides a serverless execution environment that automatically scales and charges only for compute time, which is ideal for tasks that need to be performed once per user.
Alternative - EC2[4]: Using EC2 for this functionality would involve running a continuously active instance, which would incur unnecessary charges since the app is idle most of the time.

EC2

Purpose: EC2 hosts the Spring Boot application that processes GitHub events and interacts with Slack.
Why EC2: I required a computing system without additional attached services. EC2 allows me to manage the core functionality of my Slackify application independently.
Alternative - Elastic Beanstalk[5]: Elastic Beanstalk includes many additional services like databases and security groups. Since I handle these components separately, EC2 is a better fit for my needs.

SNS

Purpose: SNS[6] is used for sending subscription confirmation emails to users, ensuring they confirm their email addresses before connecting their repositories to Slack.
Why SNS: SNS is suitable for sending plain text emails to individual users, which aligns with my requirements.
Alternative - SES[7]: SES is designed for bulk email campaigns with HTML formats and is more suitable for marketing purposes.

RDS

Purpose: RDS[8] stores user credentials, tokens, and repository-related data such as issues and comments securely.
Why RDS: I require an SQL database for this project, rather than a NoSQL option.
Alternative - Amazon Aurora: While Amazon Aurora offers similar functionalities, it is more expensive compared to RDS.

Secret Manager

Purpose: Secret manager[9] Securely stores and manages sensitive information like database credentials.
Why Secret Manager: Offers secure management of secrets with features like automatic rotation and fine-grained access control, crucial for maintaining the security of this application
Alternative - Systems Manager Parameter Store[10]: It is also a viable option for storing configuration data and secrets, Secret Manager provides additional features such as automatic rotation of credentials and more granular access policies, making it more suitable for managing sensitive information securely.

Deployment Model

All components of the Slackify application are hosted on a **public cloud[11]** infrastructure provided by AWS. This includes the compute resources (EC2 and Lambda), storage (RDS), and other services such as API Gateway, SNS, and Secret Manager. The application operates entirely within AWS's cloud environment, leveraging its scalable, secure, and managed services to ensure reliable performance and high availability.

Delivery Model

For the Slackify project, the overall delivery model is Software as a Service (SaaS)[12]. This SaaS model enables users to seamlessly connect GitHub with Slack without having to manage the underlying infrastructure. The application handles webhook creation, event processing, and notifications automatically, providing a smooth experience with minimal operational overhead. By delivering and maintaining the service as a fully managed solution, Slackify enables users to

focus on their GitHub activities and Slack communications while benefiting from a scalable, reliable integration platform.

API Gateway	Platform as a Service (PaaS)
Lambda	Function as a Service (FaaS)
EC2	Infrastructure as a Service (IaaS)
RDS	Platform as a Service (PaaS)
SNS	Platform as a Service (PaaS)
Secret Manager	Platform as a Service (PaaS)

Architecture

Diagram

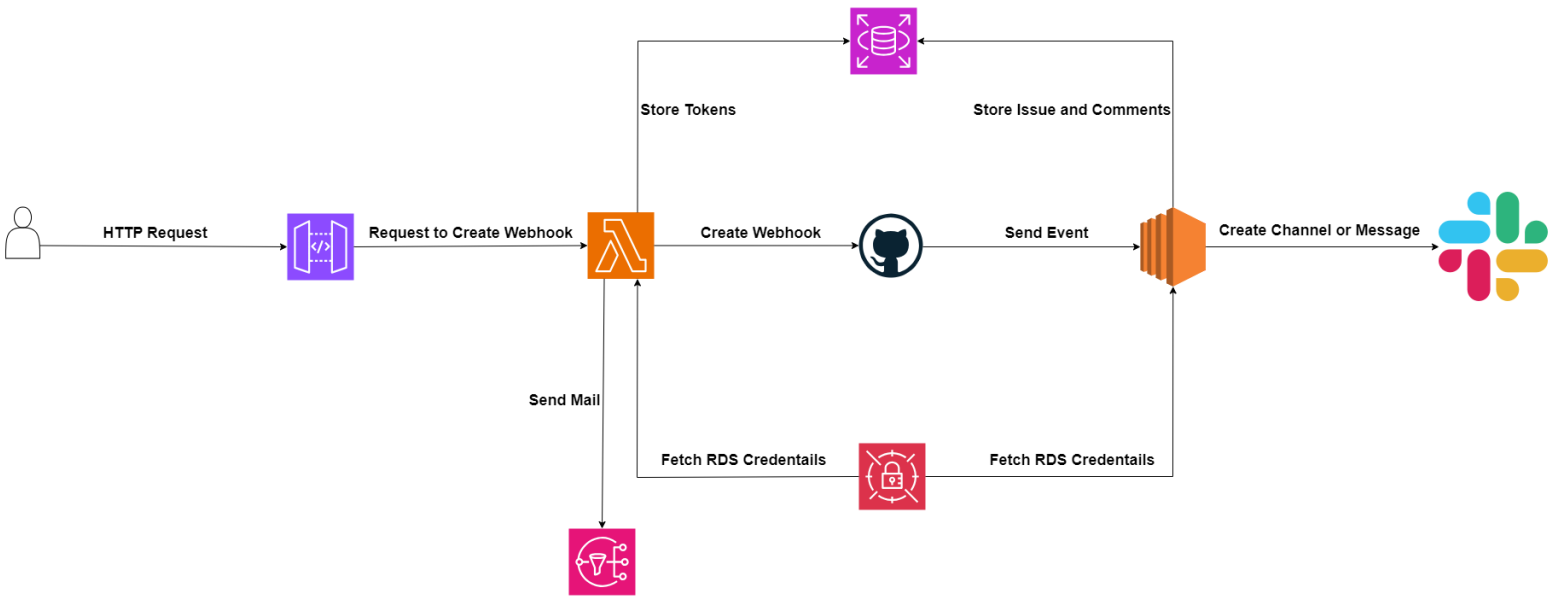

Figure 1: Slackify Architecture[13]

Flow of the Software

User Input

- The user provides their email, GitHub username, repository name, GitHub token, and Slack token.

API Gateway

- The request containing user inputs is sent to the API Gateway.

Lambda Function

- **Input Verification:** The Lambda function verifies the provided inputs.
- **Email Check:** The Lambda function checks if the user's email is already present.
- **Subscription Check:** The Lambda function checks if the user is already subscribed to the SNS topic.
    - ➔ *If the user is not subscribed:*
        - ❖ **Send Subscription Email:** The Lambda function uses SNS to send a subscription confirmation email to the user.
        - ❖ **Await Subscription Confirmation:** The user must confirm their subscription by clicking the link in the email.
        - ❖ **Repeat Request:** The user must resend the request. If the email is not confirmed, the Lambda function responds with "Please confirm the email first."
    - ➔ *If the user is subscribed:*
        - ❖ **Webhook Creation:** The Lambda function creates a webhook on the specified GitHub repository.
        - ❖ **Store Information:** The user's GitHub token, username, repository name, and Slack token are stored in an RDS database. RDS credentials are fetched from AWS Secrets Manager.
        - ❖ **Set Webhook:** The webhook is configured to send events to the IP address of the Spring Boot application running on EC2.

Event Handling

- **Receiving Events:** When an issue or comment event occurs in the GitHub repository, the webhook sends the event data to the Spring Boot application on EC2.
- **Validation:** The Spring Boot application verifies the received event data using information stored in the RDS database. RDS credentials are again fetched from AWS Secrets Manager.
- **Slack Integration:**

- ➔ **Issue Events:** For new issues, the application creates a new Slack channel. For reopened or closed issues, it sends a message to the appropriate channel indicating the issue status and the user responsible.
- ➔ **Comment Events:** For new comments on issues, the application sends the comment and the commenter's details to the relevant Slack channel.

Overall, the cloud mechanisms—API Gateway, Lambda, SNS, RDS, Secret Manager, and EC2—work in tandem to handle user input, securely store and manage data, process GitHub events, and facilitate communication with Slack, delivering a comprehensive and scalable integration solution.

Data Stored

RDS

RDS is used to securely store user information like user email addresses, the connection of GitHub and Slack status, and GitHub repository and event information. The data is organized in relational tables, which allows for efficient querying and management.

Secret Manager

The secret Manager stores sensitive information such as RDS credentials which include username, password, URL, port, and database name.

Programming languages

For the Slackify application, Java was used across all parts of the code. Here’s how Java was utilized in different parts of the application:

Lambda Functions: Java is used for implementing AWS Lambda functions to handle input verification, webhook creation, and interaction with SNS. Java's robust ecosystem and libraries support complex logic and integrations, making it suitable for any scalable system.

Spring Boot Application on EC2: Java is employed to develop the Spring Boot application running on EC2, which processes GitHub events and integrates with Slack. Spring Boot

simplifies the development of enterprise-level applications with its extensive support for RESTful APIs, database interactions, and integration with external services like Slack.

System Deployed to the Cloud

Initially, we provision the cloud infrastructure using CloudFormation, which automates the setup of our resources. Once the infrastructure is in place, we install the necessary tools on the EC2 instance, such as Docker. We then pull the Docker image from Docker Hub[14] and run it on the EC2 instance. For updates to the software, we create a new Docker image, push it to Docker Hub, and then pull and run this updated image on the EC2 instance. This approach allows us to deploy new versions of the application efficiently and ensures that our system remains up-to-date with minimal manual intervention.

Future Considerations
To further streamline and enhance the deployment process, we plan to integrate CI/CD pipelines. This will automate the process of building, testing, and deploying Docker images, reducing manual steps and improving deployment consistency.

Data Security in Application Architecture

In the Slackify application, data security is maintained through several key mechanisms. API Gateway is employed to manage access to the Lambda functions, ensuring that only authorized requests can interact with the backend services. This helps protect against unauthorized access and provides a layer of security between users and the application logic.

For managing sensitive information such as RDS credentials, Secret Manager is utilized. Secret Manager provides a secure environment for storing and accessing sensitive data by controlling access through fine-grained IAM policies.

However, there is a noted vulnerability in the current implementation regarding the storage of GitHub and Slack tokens. These tokens are stored in RDS in their raw form without additional encryption. This exposes them to potential risks if the database is compromised or if there are weaknesses in access controls.

To address this vulnerability, future improvements should include encrypting GitHub and Slack tokens before storing them in RDS. Leveraging encryption libraries or AWS Key Management Service (KMS)[15] to handle encryption and key management can help secure these tokens.

Cost for Reproducing Cloud Architecture in a Private Cloud

To reproduce the Slackify architecture in a private cloud while maintaining a similar level of availability and functionality as the AWS cloud implementation, the organization would need to acquire a combination of hardware, software, and services. Here's a rough estimate which is calculated from AWS Cost Explorer[16] of what would be required:

Compute Resources
Slackify utilizes AWS Lambda for serverless computing tasks, such as input verification and webhook creation. To replicate this, you would need high-performance servers capable of running your application logic, handling GitHub events, and managing Slack communications. This would involve deploying multiple servers to ensure scalability and redundancy.

Cost Estimate - $1403 (16 GB RAM)

Database Storage
AWS RDS is used for storing user credentials, tokens, and repository details securely. In a private cloud, you would need enterprise-grade relational database servers with high-availability features, including replication and automated backups.

Cost Estimate - $175 (2TB SSD)

Networking and API Management
AWS API Gateway manages access to our Lambda functions. To replicate this, you would need to invest in hardware load balancers and API management software to handle traffic, security, and rate limiting.

Cost Estimate - $1200 (12 GB RAM, 8 Core CPU)

Secret Management
AWS Secret Manager securely stores sensitive information like RDS credentials. In a private cloud, you would need hardware security modules (HSMs) or enterprise secret management software to ensure secure storage and handling of secrets.

Cost Estimate - $125

Notification Service
AWS SNS is used for sending subscription confirmation emails and notifications. To replicate this in a private cloud, you would need a notification service platform or build a custom notification system to handle email and messaging needs securely and efficiently.

Cost Estimate - $725

To reproduce the Slackify architecture in a private cloud, the total cost would range from approximately $3500 to $4000. This estimate includes the expenses for compute resources, database systems, networking equipment, secret management solutions, and notification services, aiming to deliver functionality and availability comparable to the AWS cloud implementation.

Monitoring Compute Resources to Control Cloud Costs

In the Slackify application, EC2 instances are critical for hosting the Spring Boot application that processes GitHub events and interacts with Slack. Since these instances are essential for core functionality and are actively processing messages to Slack, they can become significant cost drivers if not monitored carefully.

To manage costs for EC2 instances running the Slackify application, focus on setting up CloudWatch Alarms[17]. By configuring alarms for key metrics like CPU utilization and memory usage, you can monitor the performance of your instances in real time and detect any inefficiencies or unexpected spikes in resource usage. This helps ensure that you are not over-provisioning or under-utilizing resources, thus controlling costs effectively.

Future Evolution and Feature Expansion

As Slackify evolves, several enhancements could be implemented to further streamline communication between GitHub and Slack. One significant addition would be to expand event handling capabilities to include a broader range of GitHub activities beyond just issues and comments. This could involve incorporating events such as pull requests, commits, and user mentions. To achieve this, we would extend the Lambda function to handle these new event types and utilize API Gateway to manage the increased API calls.

Another potential feature is the development of a comprehensive reporting page. This page would provide project managers with an overview of all messages and notifications sent from

GitHub to Slack. It would aggregate data such as issue statuses, comments, and reactions, presenting them in an accessible format. To implement this, we would use AWS RDS for storing historical data and AWS Lambda to fetch and process this information. A web interface could be hosted on EC2, allowing managers to view reports and gain insights into communication patterns.

Additionally, we could introduce functionality to automatically invite users to relevant Slack channels based on their involvement in GitHub activities. For instance, when a new issue is created or a comment is made, users mentioned in these activities would be automatically added to the corresponding Slack channels. This would be managed through Lambda functions that interact with the Slack API, using AWS Secrets Manager to securely handle API tokens.

Another feature could involve notifying users when they are tagged in comments or mentions within GitHub. SNS would be employed to send these notifications, ensuring users are promptly informed of their mentions.

Finally, incorporating emoji handling for reactions to GitHub comments could enhance user engagement. Lambda functions would process reaction events and use the Slack API to reflect these emojis in Slack channels, making the communication more interactive and engaging.

These enhancements would improve Slackify's functionality and user experience, leveraging AWS's scalable services to handle increased complexity and ensure seamless integration.

References

[1] Amazon Web Services, "Amazon API Gateway Documentation," Amazon Web Services, 2024. [Online]. Available: https://docs.aws.amazon.com/apigateway/. [Accessed: Aug 07, 2024].

[2] Amazon Web Services, "Configuring AWS Lambda Function URLs," Amazon Web Services, 2024. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/urls-configuration.html. [Accessed: Aug 07, 2024].

[3] Amazon Web Services, "AWS Lambda Documentation," Amazon Web Services, 2024. [Online]. Available: https://docs.aws.amazon.com/lambda/. [Accessed: Aug 07, 2024].

[4] Amazon Web Services, "Amazon Elastic Compute Cloud (Amazon EC2) Documentation," Amazon Web Services, 2024. [Online]. Available: https://docs.aws.amazon.com/ec2/. [Accessed: Aug 07, 2024].

[5] Amazon Web Services, "Amazon Elastic Beanstalk Documentation," Amazon Web Services, 2024. [Online]. Available: https://docs.aws.amazon.com/elastic-beanstalk/. [Accessed: Aug 07, 2024].

[6] Amazon Web Services, "Amazon SNS Documentation," Amazon Web Services, 2024. [Online]. Available: https://docs.aws.amazon.com/sns/. [Accessed: Aug 07, 2024].

[7] Amazon Web Services, "Amazon SES Documentation," Amazon Web Services, 2024. [Online]. Available: https://docs.aws.amazon.com/ses/. [Accessed: Aug 07, 2024].

[8] Amazon Web Services, "Amazon RDS Documentation," Amazon Web Services, 2024. [Online]. Available: https://docs.aws.amazon.com/rds/. [Accessed: Aug 07, 2024].

[9] Amazon Web Services, "AWS Secrets Manager Documentation," Amazon Web Services, 2024. [Online]. Available: https://docs.aws.amazon.com/secretsmanager/. [Accessed: Aug 07, 2024].

[10] Amazon Web Services, "AWS Systems Manager Parameter Store," Amazon Web Services, 2024. [Online]. Available: https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-parameter-store.html. [Accessed: Aug 07, 2024].

[11] GeeksforGeeks, "Cloud Deployment Models," GeeksforGeeks, 2024. [Online].

Available: https://www.geeksforgeeks.org/cloud-deployment-models/. [Accessed: Aug 07, 2024].

[12] GeeksforGeeks, "Cloud-Based Services," GeeksforGeeks, 2024. [Online]. Available: https://www.geeksforgeeks.org/cloud-based-services/. [Accessed: Aug 07, 2024].

[13] diagrams.net, "Free Online Diagram Editor", https://app.diagrams.net/, [Accessed Aug 07, 2024].

[14] Docker, Inc., "Docker Hub," Docker, Inc., 2024. [Online]. Available: https://hub.docker.com/. [Accessed: Aug 07, 2024].

[15] Amazon Web Services, "AWS Key Management Service (KMS) Documentation," Amazon Web Services, 2024. [Online]. Available: https://docs.aws.amazon.com/kms/. [Accessed: Aug 07, 2024].

[16] Amazon Web Services, "AWS Pricing Calculator," Amazon Web Services, 2024. [Online]. Available: https://calculator.aws/#/. [Accessed: Aug 07, 2024].

[17] Amazon Web Services, "Create an Alarm That Sends an Email Notification," Amazon Web Services, 2024. [Online]. Available: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/AlarmThatSendsEmail.html. [Accessed: Aug 07, 2024].