ANDERSEN

Architecture Vision Document for My Log

Issued: 15.05.2024

Version: 1.0

Rondo ONZ 1, 00-124, Warsaw, Poland

Revision History

Date	Version	Description	Author
2024-06-01	0.1	Analyze and fill architectural drivers.	Ahramenko Artem
2024-06-02	0.2	Generated and filled Executive Summary and Architectural Drivers.	Ahramenko Artem
2024-06-03	0.3	Filled gaps and updated Architectural Drivers.	Ahramenko Artem
2024-06-04	0.4	Filled Context, Container and Component MyLog app sections.	Ahramenko Artem
2024-06-14	0.5	Added Dev and Prod environments	Ahramenko Artem
2024-06-15	0.6	Added Implementation Roadmap	Ahramenko Artem
2024-06-16	0.7	AVD first Iteration review	Ahramenko Artem
2024-06-25	1.0	Milestones correction, presentation creation	Ahramenko Artem

1 Introduction

1.1 Vision Purpose

This document provides a high-level description of the technical solution for the MyLog project. This solution was conceived with the goal of addressing all requirements described in the MyLog Vision & Scope Document. The current document describes the major modules that will make up the solution within the initial scope, as well as interaction and integration aspects.

This document is intended to:

- Overview high-level project description.
- Describe delivery and operations processes.
- Data backups and restore processes.
- Recommend technology stack.
- Provide some decisions explanations.
- Recommend technology stack.
- Support on-boarding processes for new members of the development team.

Any changes to the document after agreeing on the initial version should be tracked in the Revision history table.

1.2 Business Context

The customer is an international global company that developed and supports several on-premise e-commerce products. These products should be installed in customer data centers but may send some data to our central server in the cloud. The goal is to develop a new, highly available, global application (native mobile + web) to monitor and analyze logs from our e-commerce products. This application will provide near real-time data and aggregated statistics to help our customers understand their system's performance and issues.

1.3 Glossary

The Definitions section lists the acronyms and terms used in this document which might possess lesser familiarity or double meaning to the reader.

Abbreviation	Transcript
AWS	Amazon Web Services
SSO	Single Sign-on
RPS	Requests per second
ELB	Elastic Load Balancer
HTTP	HyperText Transfer Protocol
TTL	Time To Live

1.4 References

The References section provides a complete list of all the documents referenced elsewhere in this document.

#	Version	Date	Document Name	Published by
	<0.1>	<05/15/2024>	MyLogTask	Ahramenko Artem
	<0.2>	<05/15/2024>	MyLogCalculation	Ahramenko Artem
	<0.5>	<06/04/2024>	C4	Ahramenko Artem
	<0.6>	<06/04/2024>	Deployment, CI/CD, Database Diagrams	Ahramenko Artem
	<0.7>	<06/05/2024>	Project Estimation	Ahramenko Artem
	<0.8>	<06/05/2024>	Infrastructure Estimation	Ahramenko Artem
	<0.9>	<07/05/2024>	Database	Ahramenko Artem

2 Executive Summary

2.1 High Level Description

The goal is to develop a new, highly available, global application (native mobile + web) to monitor and analyze logs from our e-commerce products. This application will provide near real-time data (updating every second) and aggregated statistics to help our customers understand their systems' performance and issues.

2.2 Key Decision

There are key decisions that were made during the discovery phase:

- Usage of the AWS Cloud platform.
- Support for web and mobile-native cross-platform applications sharing the server side.
- Integration with the existing SSO provider solution.
- Inclusion of an Admin Console as part of the “MyLog” application.
- The release date is 3 months after the development phase starts.

2.3 Key Risk

There are key risks that were identified during the discovery phase:

- Data storage costs: Storing logs for six months could lead to significant storage costs.
- Performance and scalability under high load: Ensuring near real-time performance (less than 2-4 seconds) under peak load conditions (up to 300 messages per minute per product) may be challenging and has a significant impact on the architecture.
- Integration with existing SSO providers: This can be unpredictable due to potentially unavailable documentation during the discovery phase.
- Global performance consistency: Ensuring consistent performance across different geographical regions may be challenging due to network latency and varying internet speeds.

3 Architectural Drivers

3.1 Business Goals

ID	Business Goal Description
BG-1	Create a global SaaS application “MyLog” with an availability rate of at least 99.99% and the ability to scale up to 10,000 and more active senders simultaneously within 3 months including performance testing..
BG-2	Develop a simple and intuitive cross platform(web + mobile-native) interface with the ability to select and filter data by applications and periods within the scope of “MyLog” application.
BG-3	Develop and integrate an admin console for configuring applications and managing users as a part of “MyLog” application.
BG-4	Ensure the “MyLog” application can process up to 80.000 messages per second and retain 99.99% of logs for analysis.
BG-5	“MyLog” application must be available around the world with approximately the same performance(2-4 seconds) to send logs regardless of the country of origin.
BG-6	Ensure integration with existing SSO to use current customer accounts without requiring re-registration, improving user convenience and reducing transition barriers.
BG-7	Support the English language in the initial phase, with the subsequent ability to add support for other languages, enabling international scalability of the application.

3.2 Major Features

ID	Feature Description
MF-1	Auth and SSO provider integration.
MF-2	AWS infrastructure setup and configuration.
MF-3	Processing, storing and displaying logs(including filtration and aggregation)
MF-4	Admin console
MF-5	MyLog application implementation

3.3 Design Constraints

ID	Description	Priority
CT-1	The application must be hosted on AWS cloud.	High
CT-2	Support web and mobile-native cross platform applications sharing the server side.	High
CT-3	A user interface will be implemented only in English with future support of others.	High
CT-4	The performance must be approximately the same(2-4 seconds) independently on a country of an application/user.	High
CT-5	SSO - integration with a current provider solution.	High
CT-6	The application must handle up to 80,000 messages per second.	High
CT-7	The application must provide real-time data processing with a delay of less than 2 seconds under normal conditions and up to 4 seconds during peak loads.	High
CT-8	Develop and integrate an admin console for configuring applications and managing users as a part of “MyLog” application.	Medium
CT-9	The release date is 3 months after the development phase starts, including performance testing and bug fixing	Medium

CT-10	Browser versions requirements: ● Chrome - latest version ● Safari - latest version	Medium
CT-11	Screen resolutions requirements: ● Desktop: ○ 1920 x 1080 ● Mobile native app: From 5 to 7 inches.	Medium
CT-12	The system should support Android platform 8.0 version and higher, iOS 13.0	Medium
CT-13	The application should store data for 6 months.	Medium
CT-14	Implement robust data protection mechanisms, including encryption and secure access controls.	Soft

3.4 Architectural Concerns

ID	Description	Priority	Complexity
CN-1	Ensure the system can handle up to 40,000 messages per second without significant delays	High	High
CN-2	Ensure the system maintains 99.99% availability	High	High
CN-3	Fault tolerance and disaster recovery mechanisms	High	High
CN-4	Worldwide consistent performance (2-4 seconds)	High	High
CN-5	Select technologies and architectural patterns to achieve low latency and high performance	High	Medium
CN-6	Ensuring a simple and intuitive interface for selecting and filtering data	High	Medium
CN-7	Technologies and tools to ensure real-time data processing with a delay of less than 2 seconds	High	Medium
CN-8	Ensuring scalability and reliability during peak loads	High	Medium
CN-9	Ensuring the security of the admin console within the primary app.	High	Medium

CN-10	Storing and managing more than 15 TB of data per month	High	Low
CN-11	Security measures to protect data and prevent breaches	Medium	Medium
CN-12	Implementation data encryption and access control	Medium	Medium
CN-13	Integration the current SSO mechanism and ensure authentication security	Medium	Medium
CN-14	Integration the new application with existing systems and AWS infrastructure	Medium	Medium
CN-15	Organizing the development and testing process to meet the 3-month deadline	Medium	Medium
CN-16	Project management methodologies and tools to ensure adherence to deadlines and quality standards	Medium	Medium
CN-17	Multiple languages support	Medium	Low
CN-18	Strategies for data archiving and deletion to optimize storage costs	Medium	Low
CN-19	Monitoring and reporting	Medium	Low
CN-20	Select a reference architecture	Medium	Low

3.5 Quality Attributes and Quality Attribute Scenarios

A Quality Attribute Scenario is an unambiguous and testable requirement for one or more Solution Quality Attributes such as Performance, Usability, Maintainability and others. The scenario consists of six parts: Source of Stimulus, Stimulus, Environment, Artifact, Response, testable and accurate Response Measure.

ID	Source of Stimulus	Stimulus	Response	Response Measure	Business Priority	Complexity
Availability						
QA-1	Users	Request to access the system	The system should be available 24/7	99.99% system availability(Year server downtime 52.56 min)	High	High
QA-2	System	Failure occurs	The system should recover from failures without data loss	Logs are preserved and correctly processed after recovery	High	High
Data Management						
QA-3	Developers	Delete data older than 6 months	Data is deleted	Data is deleted according to policies	Medium	Low
Maintainability						
QA-4	Developers	New feature deployment	The system updates with a new version	Updates integrated with zero downtime, worst case under 1 minute	Medium	Medium
QA-5	Developers	Review of codebase	Codebase should follow best practices and coding standards	Code reviews pass without major issues	Medium	Medium
QA-6	Developers	New feature deployment	The system should support automated deployment processes	Deployment processes are automated and reduce manual errors	Medium	Medium

Monitoring						
QA-7	Developers/ Admins/ DevOps	Monitoring of system performance	The system should provide real-time monitoring and reporting of system performance and security metrics	Real-time monitoring and reports are available	Medium	Low
Performance						
QA-8	Users	Retrieval of log data	The system should retrieve data and display logs	Request time is less than 2 seconds under normal conditions, up to 4 seconds during peak loads	High	High
QA-9	Users	Use a system from a distant region	The system should retrieve data and display logs	CDN works, request time is less than 4 seconds	High	High
Reliability						
QA-10	System	Unexpected downtime or performance degradation	The system should log and alert administrators about any unexpected downtime or performance degradation	Alerts are sent out and logged correctly immediately after failure	High	Low
Scalability						
QA-11	Business growth	Increase in number of users	The system should be able to handle a growing number of users without performance degradation	No performance degradation with growing number of users	High	Medium
QA-12	System traffic	Increase in log messages	The system should scale to support up to 40,000 messages per second by adding additional server instances	System supports 40,000 messages per second without performance degradation	High	High
Security						

QA-13	Admin	Access to sensitive functions and data	Implement role-based access control	Access is restricted based on roles. Users can’t see Admins’ features.	Medium	Medium
QA-14	Users	Authentication to the system	Integration of the current SSO mechanism and ensuring authentication security	SSO mechanism is integrated and secure	Medium	Medium
QA-15	Users	Transmission of credentials during login	At all times, the credentials entered by the user during log-in are transferred to the server over an encrypted channel	Credentials are transferred securely without being sniffed by third parties	High	Medium
Integrity						
QA-16	Users	Consistency of authentication data	Ensure that all user authentication data remains consistent and accurate when integrated with the existing SSO system	Authentication data is consistent and accurate	High	Medium
Usability						
QA-17	Admins	Configuration of applications and user management	Provide an admin console that allows administrators to configure applications and manage users efficiently	Admin console is intuitive and efficient	Medium	Medium
QA-18	Users	Usage of the system in different languages	Support multiple languages to enable international scalability of the application	Multiple languages are supported	Medium	Low
QA-19	Users	Filtering and sorting of data	Users should be able to filter and sort data within the 'MyLog' application and also create their own dashboards	Data filtering, sorting, and dashboard creation are supported	High	Medium
QA-20	Users	Learning to use the user interface	Ensure the user interface is intuitive and requires minimal training for new users	Less than a 5 minutes training required for new users	High	Medium

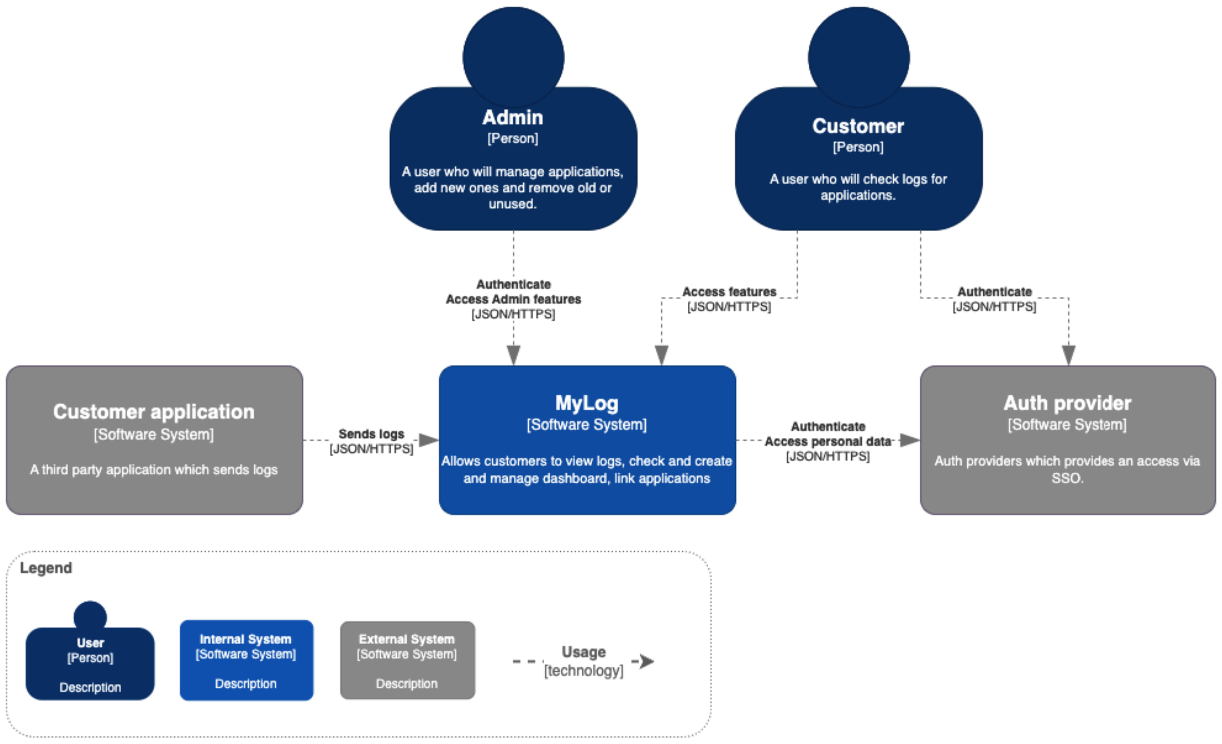
4 Solution Architecture

4.1 System Context View

4.1.1 Intent

The view defines the primary solution components collaborating with the external systems and services. It is driven by the **Business Case**.

4.1.2 Representation



4.1.3 Elements

Internal Elements

Name	Description
Customer	An external user of “MyLog” application
Admin	An external user of “MyLog” application with admin features
MyLog	Software system provides access to logs

External Elements

Name	Description
User	An external user of a third-party system
Customer application	A third-party system which sends logs to “MyLog” application
Auth provider	The company’s internal application which provides access to SSO.

4.1.4 Rationale

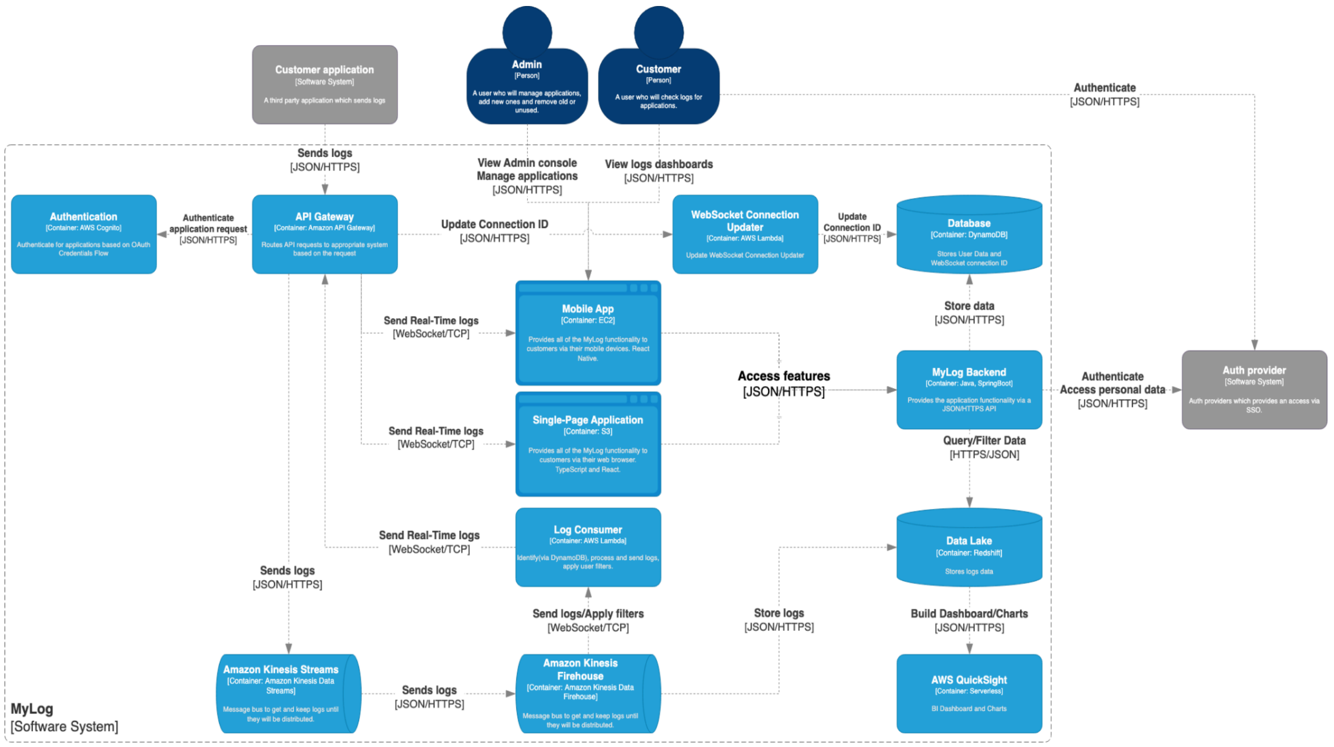
The section above provides a high-level view of which system needs to be designed, and also gives an opportunity to perceive the scale of the application by displaying third-party / additional systems and services.

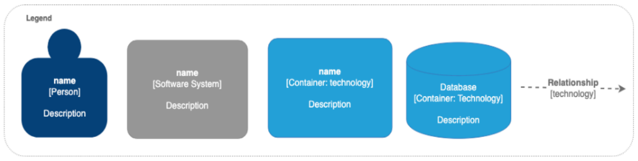
4.2 Container View

4.2.1 Intent

This view describes the major modules of the “MyLog” system as a whole.

4.2.2 Representation


Customer application
[Software System]
A third party application which sends logs
Admin
[Person]
A user who will manage applications, add new ones and remove old or unused.
Customer
[Person]
A user who will check logs for applications.
Authenticate
[JSON/HTTPS]
Sends logs
[JSON/HTTPS]
View Admin console
Manage applications
[JSON/HTTPS]
View logs dashboards
[JSON/HTTPS]
Authentication
[Container: AWS Cognito]
Authenticate for applications based on OAuth Credentials Flow
Authenticate application request
[JSON/HTTPS]
API Gateway
[Container: Amazon API Gateway]
Routes API requests to appropriate system based on the request
Update Connection ID
[JSON/HTTPS]
WebSocket Connection Updater
[Container: AWS Lambda]
Update WebSocket Connection Updater
Update Connection ID
[JSON/HTTPS]
Database
[Container: DynamoDB]
Stores User Data and WebSocket connection ID
Send Real-Time logs
[WebSocket/TCP]
Mobile App
[Container: EC2]
Provides all of the MyLog functionality to customers via their mobile devices. React Native.
Store data
[JSON/HTTPS]
Access features
[JSON/HTTPS]
MyLog Backend
[Container: Java, SpringBoot]
Provides the application functionality via a JSON/HTTPS API
Authenticate
Access personal data
[JSON/HTTPS]
Auth provider
[Software System]
Auth providers which provides an access via SSO.
Send Real-Time logs
[WebSocket/TCP]
Single-Page Application
[Container: S3]
Provides all of the MyLog functionality to customers via their web browser. TypeScript and React.
Query/Filter Data
[HTTPS/JSON]
Send Real-Time logs
[WebSocket/TCP]
Log Consumer
[Container: AWS Lambda]
Identify(via DynamoDB), process and send logs, apply user filters.
Data Lake
[Container: Redshift]
Stores logs data
Sends logs
[JSON/HTTPS]
Send logs/Apply filters
[WebSocket/TCP]
Store logs
[JSON/HTTPS]
Build Dashboard/Charts
[JSON/HTTPS]
Amazon Kinesis Streams
[Container: Amazon Kinesis Data Streams]
Message bus to get and keep logs until they will be distributed.
Sends logs
[JSON/HTTPS]
Amazon Kinesis Firehouse
[Container: Amazon Kinesis Data Firehouse]
Message bus to get and keep logs until they will be distributed.
AWS QuickSight
[Container: Serverless]
BI Dashboard and Charts
MyLog
[Software System]



4.2.3 Elements

Internal Elements

Name	Description
Customer	An external user of “MyLog” application
Admin	An external user of “MyLog” application with admin features
Customer Application	A third-party applications which send logs
MyLog Backend	Provides the application functionality via a JSON/HTTPS API
Mobile App	Provides all of the MyLog functionality to customers via their mobile devices.
Single-Page Application	Provides all of the MyLog functionality to customers via their web browser.
Database	DynamoDB that stores users data, settings, WebSocket keys
Redshift	Data Lake for logs data
QuickSight	BI tool to create dashboard and charts
Log Consumer	Process and send logs to API Gateway WebSocket connection
Amazon Kinesis Streams	Message bus to get and keep logs until they will be distributed.

Amazon Kinesis Data Firehouse	Message bus to get and keep logs until they will be distributed.
API Gateway	Routes API requests to appropriate system based on the request

External Elements

Name	**Description**
Customer application	A third-party system which sends logs to “MyLog” application
Auth provider	The company’s internal application which provides access to SSO.

4.2.4 Rationale

1. Kappa pattern to process logs streaming: it offers a scalable and cost-effective solution for processing and analyzing large volumes of log data in real-time, which is essential for a data streaming application. Lambda’s serverless architecture automatically scales to handle varying loads, processes data asynchronously, and integrates seamlessly with AWS services like Kinesis Data Firehouse and API Gateway. This approach allows our application to efficiently manage and process logs with minimal operational overhead, ensuring high availability and responsiveness for real-time monitoring and alerting.

2. WebSockets to provide real-time data: can be used to deliver logs in real-time to clients. WebSockets provide bidirectional communication, allowing clients to send filter criteria and receive only relevant logs.
3. Log Consumer is a serverless Lambda solution which is responsible for optional logs processing/transformation, an application and its WebSocket Connection ID identification, and real-time data delivery via API Gateway.
4. WebSocket connection updater is responsible for providing updates from client-side of connection, different filters, connection id's and so on.
5. The Main Application Service is responsible for several things:
    a. Query Handling: Processes read queries from clients, fetching and aggregating log data as needed. This includes responding to complex queries that may involve filtering, sorting, and aggregating log data.
    b. REST API: Provides a RESTful API for accessing log data, user management, and other application functionalities.
    c. Integration with SSO: Manages authentication and authorization using Single Sign-On (SSO), ensuring secure access to the application and its data.
    d. Business Logic: Implements the core business logic of the application, such as application management, dashboard functionalities, and data visualization.
    e. Admin Console: Includes functionality for configuring applications and managing users, providing a user-friendly interface for administrative tasks.
6. The Message Bus is a crucial component in the system because it provides a reliable and scalable way to handle large volumes of log data in real-time and transfer them further. By integrating a Message Bus, the system can efficiently manage and process high-throughput log data streams, ensuring reliability, scalability, and real-time data delivery, which are important for our requirements. I propose using Amazon Kinesis as the message bus here. For more details, please check the Message Bus Decision View.

7. I propose using RDS for PostgreSQL as a relational database to store users' related settings, attach new applications to them, filters for logs, users' charts and things related to users' security including admin roles. Database scheme.
8. I propose using Redshift as a Data Lake solution. Redshift offers seamless integration with other AWS services, enabling efficient storage, processing, and analysis of large volumes of data. It supports both structured and semi-structured data formats, allowing flexibility in how data is ingested and queried. For more details about the chosen database, please check the Database Decision View.

Benefits of Serverless Architecture

- Scalability: With services like Kinesis Data Streams (KDS), Kinesis Data Firehose (KDF), and AWS Lambda, our log processing application can automatically scale to handle varying loads. Whether it's a sudden spike in log data or a gradual increase in usage, serverless architecture adjusts resources seamlessly without manual intervention.
- Cost Efficiency: Serverless services like Lambda and API Gateway follow a pay-as-you-go model, meaning you only pay for the compute time and resources you actually use. This eliminates the need for over-provisioning and reduces costs, especially during periods of low activity.
- Reduced Operational Overhead: Since there's no need to manage or maintain servers, a team can focus on developing and improving the application rather than dealing with infrastructure. AWS handles tasks like server provisioning, patching, and maintenance.
- High Availability: Serverless services are designed to be highly available and fault-tolerant. AWS automatically handles redundancy and failover, ensuring a log processing application remains available and responsive even under adverse conditions.

- Integration and Flexibility: With services like API Gateway and Cognito, we can easily integrate authentication, authorization, and other features without managing the underlying infrastructure. This allows for quicker development and more flexibility in how your application interacts with users and other services.
- Quick Iteration and Deployment: Serverless architecture enables rapid development and deployment cycles. Lambda functions, for example, can be updated and deployed quickly, allowing to iterate on features and improvements without lengthy deployment processes.
- Security: AWS serverless services come with built-in security features, such as encryption at rest and in transit, along with fine-grained access control through IAM and Cognito. This ensures that the log processing application maintains a strong security posture with minimal effort.

4.3 MyLog Backend Component View

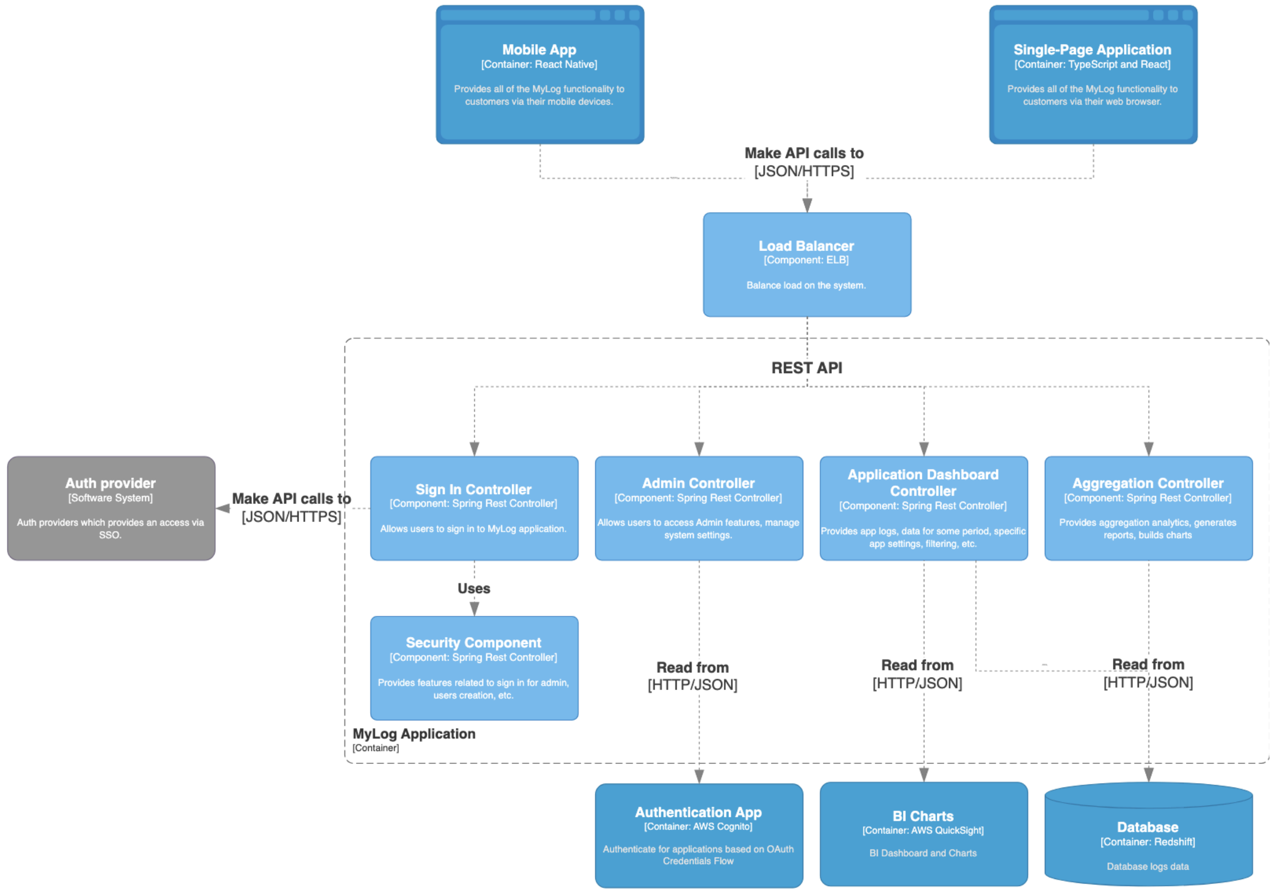
4.3.1 Intent

The Component diagram shows how a container is made up of a number of "components", what each of those components are, their responsibilities and the technology/implementation details.

4.3.2 Context

The view context is defined by the view Container View where this section represents decomposition of the **MyLog component**.

4.3.3 Representation


Mobile App
[Container: React Native]
Provides all of the MyLog functionality to customers via their mobile devices.
Single-Page Application
[Container: TypeScript and React]
Provides all of the MyLog functionality to customers via their web browser.
Make API calls to
[JSON/HTTPS]
Load Balancer
[Component: ELB]
Balance load on the system.
REST API
Auth provider
[Software System]
Auth providers which provides an access via SSO.
Make API calls to
[JSON/HTTPS]
Sign In Controller
[Component: Spring Rest Controller]
Allows users to sign in to MyLog application.
Admin Controller
[Component: Spring Rest Controller]
Allows users to access Admin features, manage system settings.
Application Dashboard Controller
[Component: Spring Rest Controller]
Provides app logs, data for some period, specific app settings, filtering, etc.
Aggregation Controller
[Component: Spring Rest Controller]
Provides aggregation analytics, generates reports, builds charts
Uses
Security Component
[Component: Spring Rest Controller]
Provides features related to sign in for admin, users creation, etc.
Read from
[HTTP/JSON]
Read from
[HTTP/JSON]
Read from
[HTTP/JSON]
MyLog Application
[Container]
Authentication App
[Container: AWS Cognito]
Authenticate for applications based on OAuth Credentials Flow
BI Charts
[Container: AWS QuickSight]
BI Dashboard and Charts
Database
[Container: Redshift]
Database logs data

4.3.4 Elements

Internal Elements

Name	Description
Load balancer	Balance the load on the system among instances(if we have more than one)
Sign In Controller	Allow users to sign in to MyLog application.

Admin Controller	Allows users to access Admin features, manage system settings.
Application Dashboard Controller	Provides app logs, data for some period, specific app settings, filtering, etc.
Aggregation Controller	Provides aggregation analytics, generates reports, builds charts
Security Component	Provides features related to sign in for admin, users creation, etc.

External Elements

Name	**Description**
Auth provider	The company's internal application which provides access to SSO.
Database	Stores logs were sent by applications
Auth Database	Stores users, apps, filters and access keys
Cache	Distributed cache to get recent logs

4.3.5 Rationale

The representation above showed MyLog backend component internals, I'd like to highlight some points of it:

1) Having Load Balancer gives some advantages:

- **High Availability and Fault Tolerance**: distributes traffic across multiple instances, ensuring continued operation if one instance fails.
- **Scalability**: easily add or remove instances based on demand, supporting automatic scaling.
- **Improved Performance**: ensures even distribution of traffic, preventing overload of individual servers and speeding up request processing.
- **Security**: offers additional security features such as DDoS protection, SSL/TLS termination, and certificate management.
- **Flexibility in Configuration**: allows for customizable routing rules based on URL, IP address, and other parameters, optimizing traffic flow.

As we host on AWS cloud, I propose to use ELB balancer and its ALB configuration because of HTTP protocol support.

2) Authentication flow: for usual users we will use an SSO provider via the REST API. For admin users we don't have such an opportunity and also having a unified admin for all systems is not a very secure solution, so for admin we create users ourselves , provide our own JWT token.

4.4 Deployment View

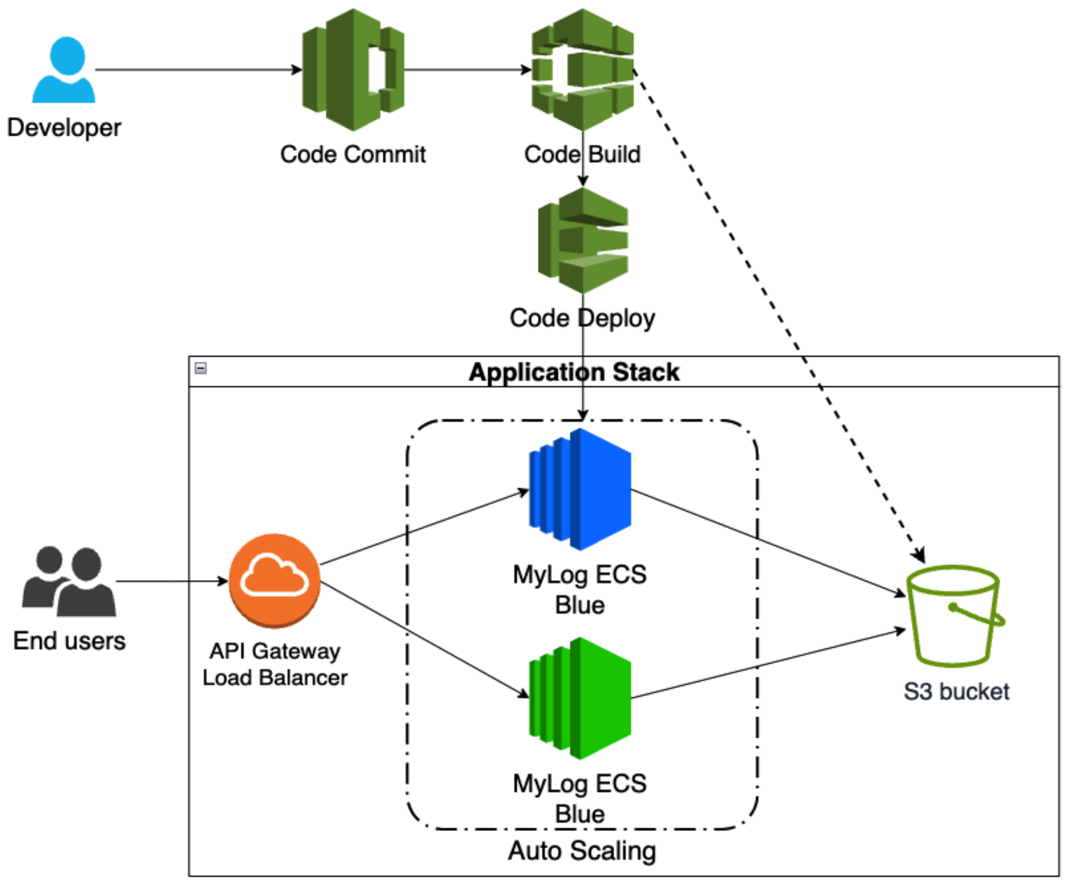
4.4.1 Intent

The deployment view diagram is designed to help us understand the deployment process of the main application. It provides a visualization of the infrastructure components and their interactions, facilitating better planning and management of the deployment.

4.4.2 Context

This diagram shows the deployment process for our main backend application "MyLog"

4.4.3 Representation



4.4.4 Elements

Internal Elements

Name	Description
MyLog EC2 Instance	The instance of our server application “MyLog”
API Gateway	Routes API requests to appropriate system based on the request
Load Balancer	Allows a system to split load between instances
Auto Scaling Group	Allows a system to scale(increase and decrease number of instances) depending on a current load.
S3	File system for saving application artifacts.

External Elements

Name	Description
Code Commit	The process of saving changes made to a project's source code in a shared repository
Code Build	Fully managed continuous integration service that compiles source code, runs tests, and produces ready-to-deploy software packages logs were sent by applications

Code Deploy	A deployment service that automates application deployments

4.4.5 Rationale

1. A developer commits code changes from their local repo to the CodeCommit repository. The commit triggers CodePipeline execution.
2. CodeBuild execution begins to compile source code, install dependencies, run custom commands, and create deployment artifacts as per the instructions in the Build specification reference file.
3. During the build phase, CodeBuild copies the source-code artifact to the Amazon S3 file system and maintains two different directories for current (green) and new (blue) deployments.
4. After successfully completing the build step, CodeDeploy deployment kicks in to conduct a Blue/Green deployment to a new Auto Scaling Group.
5. During the deployment phase, CodeDeploy mounts the S3 file system on new ECS instances as per the CodeDeploy AppSpec file reference and conducts other deployment activities.

4.5 Decision View <Database>

To define which database we need first of all we should find out the appropriate type of it. Let's compare the most appropriate of them. Relational, Columnar, NoSQL Document, Search Databases

I excluded Search Databases because we have a constraint not to implement Elastic and similar solutions here. Significant points for us:

Data Processing Speed

Scalability

Data Structure Flexibility

Filtering Capabilities

Reliability and Fault Tolerance

Parameter	Relational (SQL)	Columnar	Document NoSQL
Data Processing Speed	Medium: suitable for transactions and simple queries, but not optimized for large volumes of logs	High: optimized for analytical queries and large data volumes	High: optimized for fast access to semi-structured data
Scalability	Limited: horizontal scaling is possible but complex	High: supports distributed architecture and clusters	High: supports horizontal scaling and distributed clusters
Data Structure Flexibility	Rigid schema: requires predefined data schema	Flexible: supports semi-structured data and complex structures	Very flexible: supports unstructured and semi-structured data like JSON
Filtering	Basic: supports simple queries	Advanced: supports	Very advanced:

Capabilities	and filters	complex analytical queries and aggregations	supports complex filters and full-text search
Reliability and Fault Tolerance	High: supports replication and ACID transactions	High: supports replication and fault tolerance	High: supports replication, fault tolerance, and clustering

We can see that relational databases are not the best options here because of their disadvantages. We don’t need a strict relational model here because of the absence of a big amount of models and relations among them. Our primary flow is log -> application -> user. Columnar and NoSQL databases look better, I’d like to compare some popular representatives of them. ClickHouse for columnar,Amazon Redshift(because we are on AWS) and MongoDB. Requirements which are significant for us:

Performance for billions records(filtering, aggregation)

Cost

Integration into our system

Durability and Replication

Data Ingestion

Scalability

Data Compression(because 15TB per month)

Feature	**Amazon Redshift**	**ClickHouse**	**MongoDB**
Performance for billions of records	Good performance for typical data warehousing queries, but may face latency issues with very large datasets and complex filtering/aggregation	Excellent performance for large-scale filtering and aggregation due to columnar storage architecture	Good performance for typical filtering and indexing, but may face latency with complex aggregation

Cost	Can be expensive, especially with high query loads and large datasets	Generally more cost-effective for large-scale data storage and querying	Flexible pricing: self-managed or managed (Atlas) with variable costs based on usage
Speed of Filtering Large Volumes of Data	Good performance, but may experience latency with extremely large datasets and complex queries	Excellent performance due to its columnar storage and efficient indexing	Decent performance, but can experience latency with very large datasets and complex aggregation queries
Integration into our system	Excellent integration with AWS ecosystem and services like Kinesis, **QuickSight** for charts, etc...	Requires more manual integration, but can be integrated with other systems via connectors and custom solutions	High: flexible integration options, wide language support, and extensive ecosystem, especially with Atlas on AWS
Durability and Replication	**Automatically** replicates data across multiple nodes for durability, managed by AWS	Provides data replication and high availability, but requires manual setup and maintenance	Built-in replication with Replica Sets, strong consistency models, high durability, and automatic failover
Data Ingestion	Good for batch ingestion, supports streaming with Kinesis integration, but not optimized for real-time ingestion	Highly efficient batch and real-time data ingestion, optimized for fast data processing	High-speed data ingestion, flexible schema design, suitable for both batch and near real-time ingestion
Scalability	**Automatically** scales by adding nodes to the cluster, managed scaling by AWS	Scales horizontally with distributed architecture, requires manual configuration for scaling	Scales horizontally with sharding, flexible scaling options, easy to scale out with MongoDB Atlas
Operational Cost	No	ec2 instances manage, replicas, sharding	ec2 instances manage, replicas, sharding

Data Compression	Support LZO, Zstandart	Support LZ4, Zstandart	Supports various compression options, including WiredTiger storage engine compression, Zstandard

So, Amazon Redshift is a good solution in terms that we use other AWS applications and it has perfect integration with them. MongoDB and ClickHouse are also great solutions but require high operational and support cost to cover all significant requirements out of the box. My proposal is **Redshift**.

Data distribution: The size of compressed data for 6 months will be approximately 40TB, which is quite a huge volume for one instance of a database. Even if we can store it there, we can't work with that data efficiently. I propose using a "data distribution" supported by Redshift. It allows us to split all data by pieces. My suggestion is to separate data by user id, it means that the requests from the specific user will always go to a specific piece.

Snapshots: To increase reliability and fault tolerance, the best practice is to keep data not only in one place. A snapshot is a duplicate of our database that we will use as soon as something happens to the primary database.

Time Series Optimization: To work with time series data in Redshift, it is recommended to use sorting keys by time fields. This may resemble time partitioning in other DBMSs.

Old data deletion: Set up Amazon Redshift Scheduler to add TTL.

4.6 Decision View <Message Bus>

In a high-performance and scalable architecture, a message bus is essential for decoupling components and ensuring efficient communication between different parts of the system. For our use case, which involves processing and analyzing large volumes of user logs, I'd like to compare the main approaches in the IT industry to implement a queue. Pub/Sub queue and Producer/Consumer queue by next parameters:
Performance (40k messages/second)
High Scalability
Complexity
Unique Message Consumption

Parameter	Pub/Sub	Producer/Consumer Queues
Performance (40k messages/second)	High: Designed to handle high throughput with multiple subscribers consuming messages concurrently	High: Efficiently manages high throughput with multiple consumers processing messages in parallel
High Scalability	High: Easily scalable by adding more subscribers, supports horizontal scaling	High: Supports horizontal scaling by adding more consumers, suitable for distributed systems
Unique Message Consumption	Not guaranteed: Messages are broadcast to all subscribers, which can result in duplicate processing	Guaranteed: Each message is consumed by only one consumer, ensuring no duplicates
Complexity	Moderate: Requires setup of topics and subscription management, but many	Moderate to High: Setting up multiple queues, handling message distribution,

	managed services simplify this	and ensuring fault tolerance can add complexity

The main difference is that for the pub/sub queue we can't ensure uniqueness, which is an important requirement for our message bus. I propose to proceed with **Producer/Consumer queues.**
The second step is to select the most appropriate implementation for our architecture, I'd like to compare 4 implementations: Apache Kafka, RabbitMQ, Amazon SQS,Amazon Kinesis Data Streams. The most significant requirements are:
Performance (80k requests/second)
Scalability
Integrity (AWS Integration)
Reliability
Post-Processing (Data Modeling)
Cost
Complexity

Parameter	Apache Kafka	RabbitMQ	Amazon SQS	Amazon Kinesis
Performance (80k rps)	Very high: Can handle high throughput with proper tuning and scaling	High: Good for high throughput, but may require complex configuration for very high loads	High: Scales with usage, but may need careful setup for very high loads	Very high: Designed for real-time streaming and can handle high throughput
Scalability	High: Easily scales horizontally by adding more brokers and partitions	High: Supports clustering and sharding for scalability	High: Scales horizontally by adding more queues	High: Scales by adding more shards, easily integrates with other AWS services

Integrity (AWS)	Moderate: Can be integrated with AWS services using connectors or third-party tools	Moderate: Can be integrated with AWS but requires more configuration	High: Native AWS service with seamless integration	High: Native AWS service with seamless integration
Reliability	High: Data replication and fault tolerance with multiple brokers	High: Supports message durability and acknowledgments, can be configured for high reliability	High: Provides guaranteed delivery and durability, managed by AWS	High: Built-in fault tolerance and data durability, managed by AWS
Post-Processing (Data Modeling)	High: Supports complex stream processing with Kafka Streams or integration with Apache Spark	Moderate: Can be used with additional processing tools, but not as seamless as Kafka or Kinesis	Moderate: Integrates with AWS Lambda and other processing tools, but with some limitations	High: Seamless integration with AWS Lambda, Kinesis Data Analytics, and other processing services
Cost	Variable: Costs include infrastructure, scaling, and operational overhead	Variable: Costs include infrastructure, scaling, and operational overhead	Cost scales with usage but higher than Kinesis for huge data volume	Cost scales with usage
Complexity	High: Requires setup, configuration, and maintenance, especially for scaling and integration	Moderate to High: Requires setup and management of clustering, but has good documentation and community support	Low: Managed service with minimal setup and operational overhead	Low to Moderate: Managed service, but setup can be complex depending on use case

RabbitMQ and AmazonSQS are good solutions but may be complex to set up and pretty expensive for our data volumes, also we can’t consume it as a stream of data.

Kafka and Kinesis are quite similar solutions, but Kinesis has advantages due to having high integrity with other AWS technologies and requires less manual setup because of the same reason. My proposal is Amazon Kinesis.

Integration with AWS API Gateway: because of high integrity with other AWS applications we can send our log via AWS API Gateway without any additional services.

Partition key: I propose to use application id as partition key.

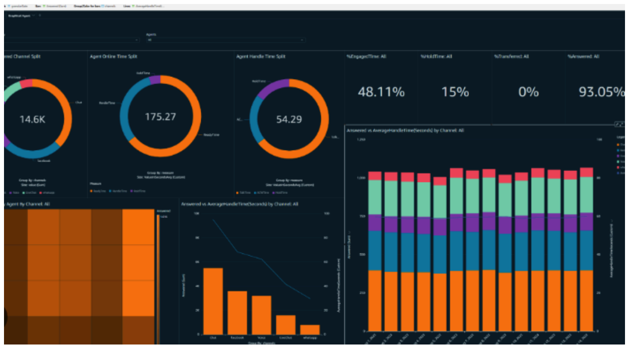
The flow: An application sends logs -> AWS API Gateway receives them and sends them to AWS Kines for further processing.

4.7 Decision View <QuickSight>

I propose to use QuickSight for BI and data visualization of the logs stored in Redshift for the following reasons:

- Embedded Analytics: QuickSight provides easy options for embedding dashboards and visualizations directly into your web or mobile applications. This means you can integrate powerful, interactive charts and graphs into your front-end without having to develop and maintain custom visualization code.
- Reduced Development Effort: By leveraging QuickSight's embedded analytics capabilities, you reduce the need to build and maintain custom front-end data visualization features, saving both time and resources.
- User-Friendly Interface: QuickSight offers an intuitive, drag-and-drop interface for creating dashboards and reports, making it accessible to both technical and non-technical users.
- Seamless Integration with AWS: QuickSight integrates seamlessly with Redshift and other AWS services, allowing you to quickly connect to your data sources and start building visualizations without needing to set up complex data pipelines.

Examples:



Sheet 1
Historical opportunity pipeline
Current open pipeline by weighted revenue
YTD Opportunities by Stage and Segment
Prospect
Contracting
Lead
Startup
SMB
Enterprise

5 Operation plan

5.1 Infrastructure

5.1.1 Infrastructure concepts

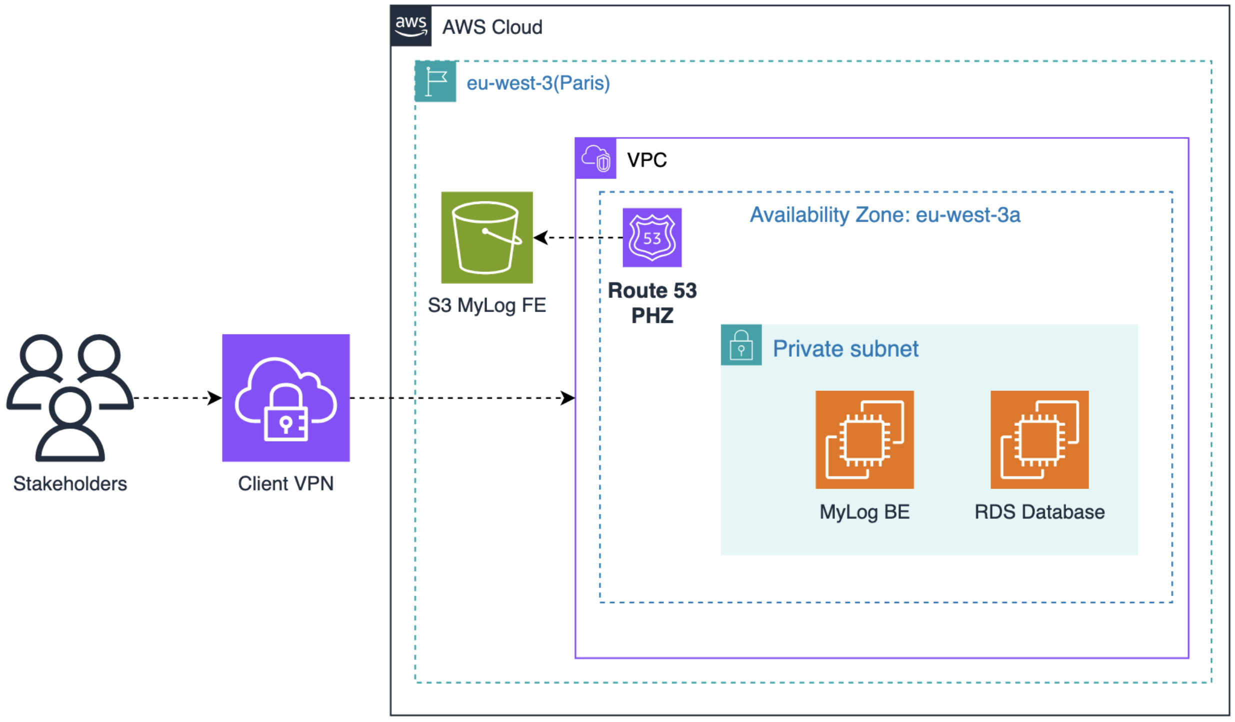
According to Design Constraints (CT-1) were decided to use AWS as a cloud provider for hosting MyLog applications. AWS has a number of the following advantages:

- Easy to use - AWS is designed to allow application providers, ISVs, and vendors to quickly and securely host your applications – whether an existing application or a new SaaS-based application. You can use the AWS Management Console or well-documented web services APIs to access AWS's application hosting platform.
- Flexible - AWS enables you to select the operating system, programming language, web application platform, database, and other services you need. With AWS, you receive a virtual environment that lets you load the software and services your application requires. This eases the migration process for existing applications while preserving options for building new solutions.
- Reliable - With AWS, you take advantage of a scalable, reliable, and secure global computing infrastructure, the virtual backbone of Amazon.com's multi-billion dollar online business that has been honed for over a decade.
- Scalable and high-performance - Using AWS tools, Auto Scaling, and Elastic Load Balancing, your application can scale up or down based on demand. Backed by Amazon's massive infrastructure, you have access to compute and storage resources when you need them.
- Secure - AWS utilizes an end-to-end approach to secure and harden our infrastructure, including physical, operational, and software measures.

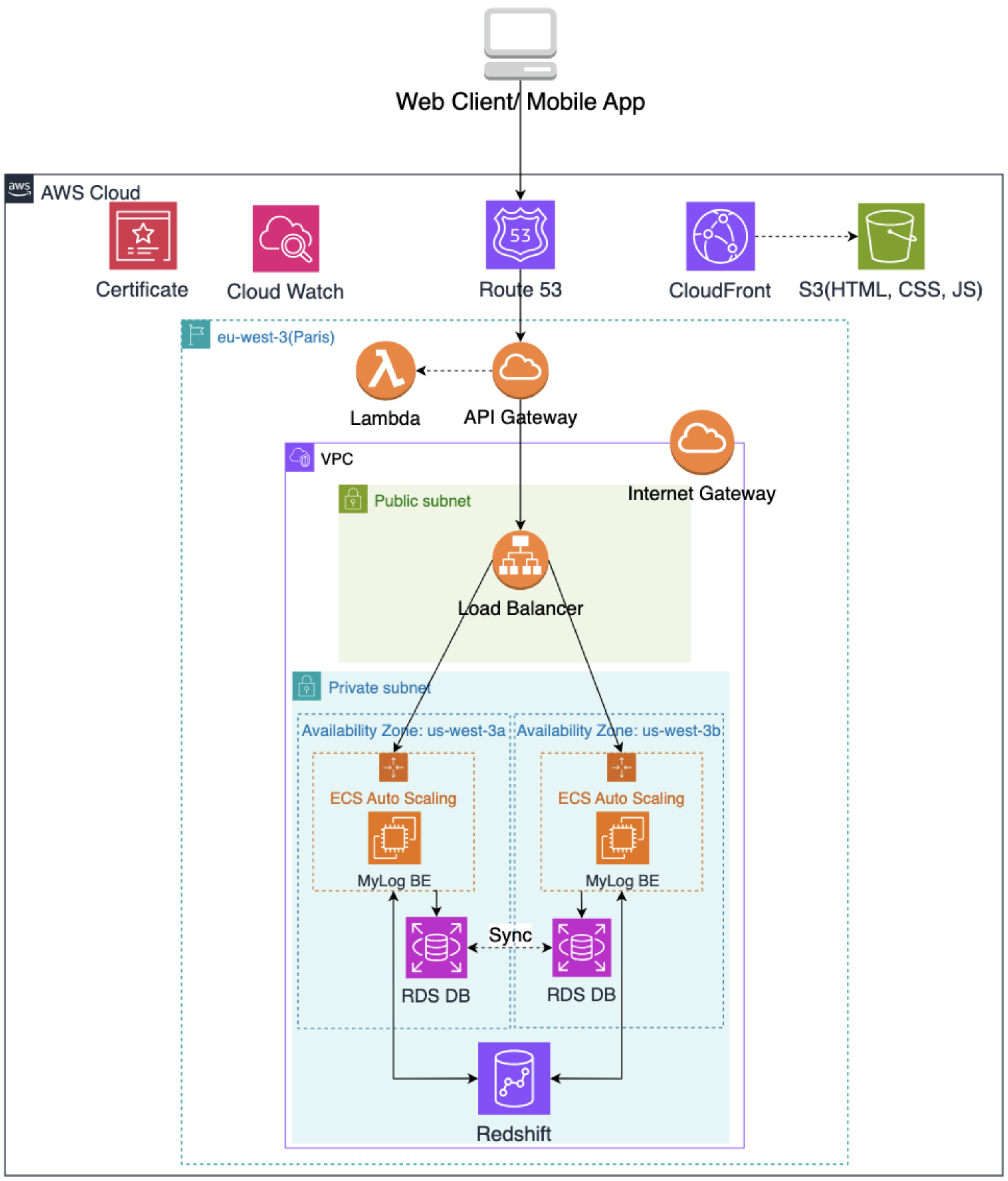
- Cost-Effective - You pay only for the compute power, storage, and other resources you use, with no long-term contracts or up-front commitments. For more information on comparing the costs of other hosting alternatives with AWS.

5.1.2 Environments

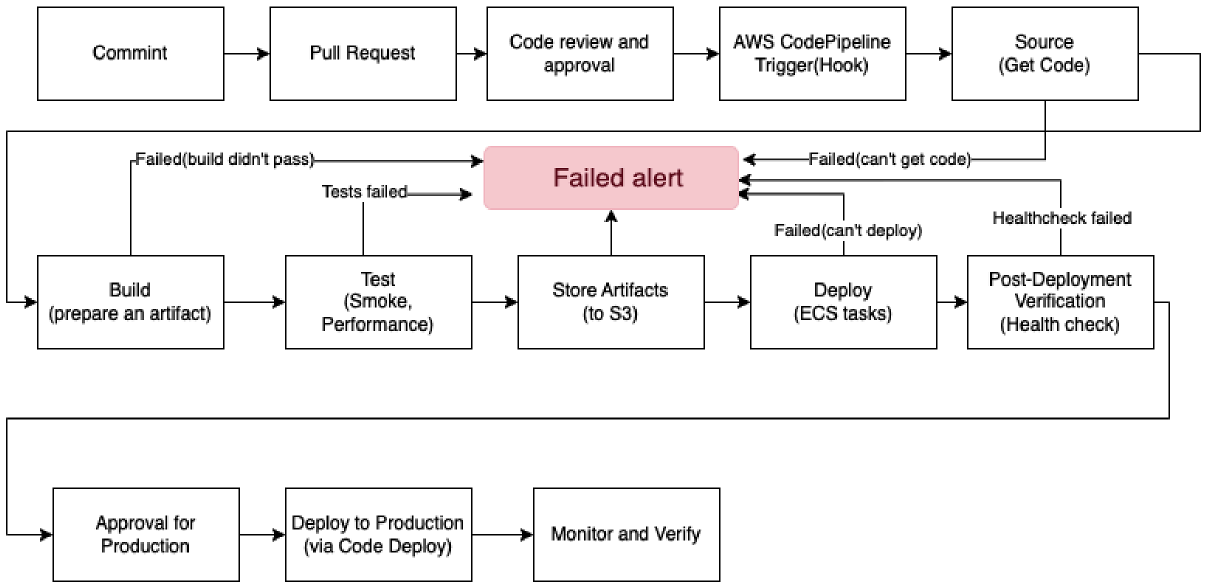
5.1.2.1 Development and Staging environment



5.1.2.2 Production environment



5.2 Continuous integration and continuous delivery



To update EC2 instances inside ASG(Auto scaling group) we should use Deployment Group. It will automatically deploy new artifacts on our instances based on chosen type(Blue/Green deployment) and other settings(like how many instances at a time we want to update) which helps us rely on AWS here and not to do anything manually.

5.3 Logging, Monitoring, Notifying

To collect logs, monitor applications, environments and notify stakeholders I propose to use Amazon CloudWatch. It is a comprehensive monitoring and observability service provided by AWS. It is designed for DevOps engineers, developers, site reliability engineers (SREs), and IT managers to gain insights into their applications and infrastructure. CloudWatch collects monitoring and operational data in the form of logs, metrics, and events, providing a unified view of AWS resources, applications, and services running on AWS and on-premises servers.

Key Features of Amazon CloudWatch

1. Metrics: Collect and track metrics from AWS services and custom applications.
2. Logs: Collect, monitor, and analyze log files from Amazon EC2 instances, AWS CloudTrail, and other sources.
3. Alarms: Set up alarms to automatically trigger actions based on metric thresholds.
4. Dashboards: Create customizable dashboards to visualize metrics and logs in a single view.
5. Events: Respond to changes in your AWS resources by triggering notifications or automating actions based on events.
6. CloudWatch Logs Insights: An interactive query service for analyzing log data.
7. CloudWatch Synthetics: Monitor your endpoints and APIs with canary tests to ensure application uptime and performance.

5.4 Backup, Restore and Disaster Recovery

Kinesis Data Streams

When a server associated with Amazon Kinesis Data Streams fails, Kinesis Data Streams continues to operate due to its highly available and fault-tolerant architecture. Here's what happens in various aspects:

Architecture and Fault Tolerance of Kinesis Data Streams

1. Data Durability:
   - Data in Kinesis Data Streams is stored in replicated shards, distributed across multiple nodes in different Availability Zones within an AWS region.
   - This replication ensures data protection against failures of individual nodes or servers.
2. Data Producers:
   - If a data producer (e.g., a server sending data to the stream) fails, it simply stops sending new data to the stream until it recovers.
   - Data that has already been sent to Kinesis Data Streams remains safe and accessible to consumers.
3. Data Consumers:
   - If a server acting as a data consumer (e.g., a server reading data from the stream) fails, other consumers can continue reading data from the stream.
   - The Kinesis Client Library (KCL) and other consumer libraries can be configured to automatically rebalance shards among the remaining available consumers.
   - Once the failed server recovers, it can resume reading data from where it left off using checkpoints that track which data has been processed.
4. Automatic Scaling and Rebalancing:

- Kinesis Data Streams supports automatic scaling and rebalancing of load across available nodes, minimizing the impact of a single node failure on the system's performance and availability.

5. **Monitoring and Alerts**:
   - AWS CloudWatch can be used to monitor Kinesis Data Streams metrics and set up alerts for failures or performance issues.
   - In case of detected issues, actions can be automatically triggered, such as restarting servers or notifying system operators.

MyLogBE applications

When an instance in an Amazon EC2 Auto Scaling Group (ASG) fails, the Auto Scaling service automatically takes steps to replace the failed instance and maintain the desired capacity and health of the application. Here's what happens:

1. **Instance Health Check**:
   - **Health Checks**: Auto Scaling periodically performs health checks on instances within the ASG. These health checks can be either EC2 status checks or ELB (Elastic Load Balancer) health checks, if the ASG is associated with an ELB.
   - **Detection of Failure**: If an instance fails a health check, it is marked as unhealthy.
2. **Termination and Replacement**:
   - **Termination**: Once an instance is marked as unhealthy, Auto Scaling terminates the failed instance.
   - **Launch Replacement**: Auto Scaling then launches a new instance to replace the terminated one. This ensures that the ASG maintains its desired capacity.
3. **Rebalancing**:

- **Load Distribution**: If the ASG is spread across multiple Availability Zones, Auto Scaling attempts to launch the replacement instance in the same Availability Zone as the terminated instance to maintain balanced distribution.
- **Scaling Policies**: Any scaling policies associated with the ASG continue to operate, adjusting the number of instances as needed based on the defined metrics and policies.

4. **Configuration Consistency**:
    - **Launch Configuration/Template**: The new instance is launched using the ASG's launch configuration or launch template, ensuring that the new instance has the same configuration (AMI, instance type, security groups, etc.) as the original one.
    - **User Data Scripts**: Any user data scripts defined in the launch configuration/template are executed on the new instance, ensuring it is properly configured upon launch.
5. **Elastic Load Balancer (ELB) Integration**:
    - **Registration**: If the ASG is associated with an ELB, the new instance is automatically registered with the ELB.
    - **Health Checks**: The ELB performs health checks on the new instance, and only routes traffic to it once it passes these checks.
6. **Monitoring and Notifications**:
    - **CloudWatch Alarms**: Auto Scaling integrates with Amazon CloudWatch, allowing you to set up alarms to notify you of changes in instance health or capacity.
    - **SNS Notifications**: You can configure Amazon SNS (Simple Notification Service) to send notifications when Auto Scaling launches or terminates instances, helping you stay informed about changes in your ASG.

6 Implementation Roadmap

I propose to use Scrum with two-week sprints for this project to ensure frequent and regular feedback loops. This approach allows the team to adapt quickly to changing requirements and priorities, enhancing our responsiveness to client needs. The short sprint duration promotes continuous improvement and maintains a high level of focus and productivity within the team. It also provides regular opportunities for stakeholder engagement and progress review, fostering transparency and alignment. Lastly, two-week sprints help in identifying and addressing issues early, minimizing risks and improving overall project quality. In the case of a 3 month project we can assume having 7 Sprints (SP).

6.1 Implementation Milestones

ID	Business Goal Description	Success Criteria Description
M-0	Project Kickoff and Planning	Project plan approved, team roles defined, and initial setup done
M-1	Dev/Stage Env Infrastructure Setup	AWS dev infrastructure set up with VPC, subnets, and security groups
M-2	Kinesis Data Stream Setup	AWS Kinesis Data Stream set up to get logs from the applications.
M-3	Data Storage and Management	ClickHouse cache and database set up with replication and cluster configured
M-4	Log Consumer Development	Log Consumer is implemented to get logs from Kinesis and distribute logs further to WebSocket

M-5	My Log Application Development	Core backend services and APIs implemented and tested. Initial versions of mobile and web dashboards developed and tested
M-6	Security and Compliance	Security measures implemented and compliance checks passed
M-7	Integration and Testing	All components integrated and system tested end-to-end
M-8	Monitoring and Logging	CloudWatch and other monitoring tools configured and tested
M-9	Prod Env Infrastructure Setup	AWS prod infrastructure set up with VPC, subnets, and security groups
M-10	Performance Optimization	System optimized for high load and performance metrics met
M-11	User Acceptance Testing (UAT)	UAT completed with user feedback incorporated
M-12	Final Deployment and Launch	System deployed to production and launched successfully
M-13	Post-Launch Support and Maintenance	Post-launch monitoring and support processes in place

6.2 Technology Stack

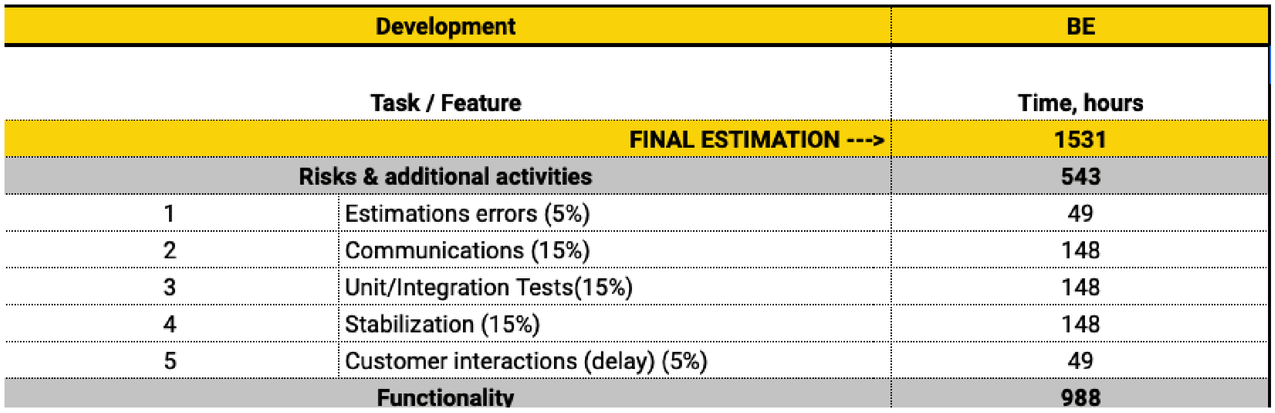
Technology	Function
Backend	
Java, v. 17 (LTS)	Java is a high-level, class-based, object-oriented programming language designed for portability and cross-platform compatibility. It is widely used for building enterprise-scale applications, web applications, and Android mobile apps. Java's "write once, run anywhere" (WORA) capability makes it a popular choice for developers.
Spring Boot	Spring Boot is an open-source framework based on the Spring framework, designed to simplify the development of Java applications. It provides a pre-configured, opinionated setup that eliminates much of the boilerplate code and configuration needed to set up a Spring application. Spring Boot allows developers to create stand-alone, production-grade Spring-based applications that can run with minimal configuration.
Amazon Kinesis Data Streams	Amazon Kinesis Data Streams is a real-time, scalable data streaming service provided by AWS. It allows you to collect, process, and analyze large streams of data records in real time. Kinesis Data Streams is commonly used for real-time analytics, log and event data collection, and processing clickstream data.
ClickHouse	ClickHouse is a high-performance columnar database management system designed for online analytical processing (OLAP) of queries. It is optimized for handling large volumes of data and providing real-time analytics. ClickHouse achieves high performance through its columnar storage format, data compression, and the ability to process queries in parallel.

WebSocket	WebSocket is a communication protocol that provides full-duplex, bidirectional communication channels over a single TCP connection. It is designed for real-time, low-latency communication between a client (such as a web browser) and a server. WebSockets are commonly used for applications requiring real-time updates, such as live chat, gaming, and financial trading platforms.
Frontend and Mobile	
React	Is a popular open-source JavaScript library developed by Facebook for building user interfaces, particularly single-page applications. It allows developers to create reusable UI components and manage the state of their applications efficiently. React uses a virtual DOM to optimize updates and rendering, making it highly performant for dynamic and interactive web applications.
TypeScript	TypeScript is a strongly typed, open-source programming language developed by Microsoft. It is a superset of JavaScript, meaning it extends JavaScript by adding static types, which can improve code quality and developer productivity. TypeScript compiles down to plain JavaScript, ensuring compatibility with existing JavaScript libraries and frameworks.
React Native	React Native is an open-source framework developed by Facebook for building mobile applications using JavaScript and React. It allows developers to create natively-rendered mobile apps for iOS and Android using a single codebase. React Native provides a rich set of components and APIs that allow developers to leverage native mobile functionalities, ensuring high performance and a native look and feel.

DevOps	
Eureka	Eureka is a service discovery tool developed by Netflix, part of the Netflix OSS suite. It allows microservices to register themselves and discover other services in a distributed system. Eureka Server acts as a registry, while Eureka Client can register services and query the server for other available services, enabling load balancing and failover.
ELB	Elastic Load Balancing (ELB) is a scalable load balancing service provided by AWS. It automatically distributes incoming application traffic across multiple targets, such as Amazon EC2 instances, containers, and IP addresses, in one or more Availability Zones. ELB helps ensure fault tolerance and improves the availability and scalability of applications by distributing the load and detecting unhealthy targets to reroute traffic.
API Gateway	API Gateway is a managed service provided by AWS that allows developers to create, publish, maintain, monitor, and secure APIs at any scale. It acts as a front door for applications to access data, business logic, or functionality from backend services, such as AWS Lambda, EC2 instances, or any web application. API Gateway handles tasks such as traffic management, authorization and access control, monitoring, and API version management.
Amazon Route 53	Amazon Route 53 is a scalable and highly available Domain Name System (DNS) web service provided by AWS. It is designed to route end-user requests to internet applications by translating human-readable domain names into IP addresses. Route 53 also supports health checks, domain registration, and traffic management, including routing policies like latency-based routing, geo DNS, and weighted round-robin, to direct traffic efficiently based on various criteria.

AWS CodeCommit	AWS CodeCommit is a fully managed source control service that hosts secure Git repositories. It allows teams to collaborate on code in a scalable and reliable manner. CodeCommit integrates seamlessly with other AWS services and supports standard Git commands, providing a familiar experience for developers.
AWS CodeBuild	AWS CodeBuild is a fully managed continuous integration service that compiles source code, runs tests, and produces software packages ready for deployment. CodeBuild scales continuously and processes multiple builds concurrently, eliminating the need for a dedicated build server.
AWS CodeDeploy	AWS CodeDeploy is a fully managed deployment service that automates application deployments to various computer services such as Amazon EC2, AWS Fargate, AWS Lambda, and on-premises servers. It helps maximize uptime during the deployment process by enabling features like blue/green deployments, rolling updates, and automated rollback.

6.3 Project Estimation



Development		BE
Task / Feature		Time, hours
	FINAL ESTIMATION --->	1531
Risks & additional activities		543
1	Estimations errors (5%)	49
2	Communications (15%)	148
3	Unit/Integration Tests(15%)	148
4	Stabilization (15%)	148
5	Customer interactions (delay) (5%)	49
Functionality		988

[Project Estimation]

Additional Notes:

1) Proposals to start for FE and Mobile developers after the first Sprint because during the first Sprint and partially second BE developer will be busy with a LogConsumer application which doesn’t require the front end part.
2) Proposals to Performance QA start after the second Sprint when at least basic components will be added and performance tests can be created.
3) Proposal to Manual QA starts after the third Sprint when FE developers prepare basic UI pages to test.

6.4 Infrastructure Estimation

Infrastructure Estimation

6.5 Team Skillset

Role	Skillset
Software Architect	• Java • Spring • NoSQL • Cloud experience (AWS/GCP/Azure/etc.) • Kafka/Kinesis/RabbitMQ • Docker • Technical Debt Management • System analysis and design • ADR • Git • WebSocket
Senior Back End Engineer	• Java 11 • Spring Boot • AWS Platform • Kafka/Kinesis/RabbitMQ • WebSocket • NoSQL DB • Docker • Git
Senior Back End Engineer	• Java 11 • Spring Boot • Web Security • NoSQL DB • Docker • Git
Senior DevOps Engineer	• AWS • Deployment Group, VPC • Docker • Kafka/Kinesis/RabbitMQ • SonarQube • NoSQL
Senior Frontend Engineer	• TypeScript • React • WebSocket • Git • Web Security • Cloud experience (AWS/GCP/Azure/etc.)

	• Docker
Senior Mobile Engineer	• React Native • TypeScript • WebSocket • Git • Web Security • Cloud experience (AWS/GCP/Azure/etc.) • Docker
Business Analytic	• Confluence • Jira • Draw.io (BPMN UML) • Miro • Balsamiq
Project Manager	• Agile Methodologies (Scrum/Kanban) • Atlassian Jira / Confluence • Asana • MS Project • Team Gantt • Miro
Middle Manual QA Engineer	• TestRail • Postman • Bug tracking system • DevTools • Jira • Swagger • Amazon Cloudwatch
UI/UX Designer	• Figma • Miro • Adobe Photoshop • Adobe Color • Jira
Performance QA Engineer	• JMeter • Postman • Fiddler • Git • Jira • Amazon Cloudwatch

6.6 Team Structure

Role	Responsibility	Count	FTE
Solution Architect	• System analysis and design • System requirements specification • Document and monitor requirements needed to institute proposed requirements • Provide detailed specifications for proposed solutions • General Technical Leadership • Negotiation with customers	1	0.5
Senior Back End Engineer	• Technical communication • Code review • Backend implementation • Being transparent with the team about challenges, failures, and successes. • Writing progress reports and delivering presentations to the relevant stakeholders. • SSO and security integration.	1	1.5
Senior Frontend Engineer	• Technical communication • Code review • Frontend implementation • Be involved and participate in the overall application lifecycle • Main focus on coding and debugging • Collaborate with Backend developers • Provide training, help and support to other team members • Build high-quality reusable code that can be used in the future • Develop functional and sustainable web applications with clean codes • Troubleshoot and debug applications	1	1
Middle Manual QA Engineer	• Plan, execute, and oversee inspection and testing of incoming and outgoing product to confirm quality conformance to specifications and quality deliverables	1	0.5

	• Analyze and investigate product complaints or reported quality issues to ensure closure in accordance with company guidelines and external regulatory requirements		
Business Analytic	• Identify the steps or tasks to support the implementation of new features • Design the new features to implement • Analyze the impact of implementing new features	1	0.5
Project Manager	• Designing and applying appropriate project management standards • Managing the production of the required deliverables • Planning and monitoring the project • Preparing and maintaining project, stage and exception plans as required • Managing project risks, including the development of contingency plans • Monitoring overall progress and use of resources, initiating corrective action where necessary	1	0.5
Performance QA Engineer	• Plan, execute, and oversee inspection and testing of incoming and outgoing product to confirm quality conformance to specifications and quality deliverables • Analyze and investigate product complaints or reported quality issues to ensure closure in accordance with company guidelines and external regulatory requirements	1	0.5
UI/UX Designer	• Investigating user experience design requirements for our suite of digital assets • Developing and conceptualizing a comprehensive UI/UX design strategy for the brand • Producing high quality UX design solutions through wireframes, visual and graphic designs, flow diagrams, storyboards, site maps, and prototypes • Designing UI elements and tools such as navigation menus, search boxes, tabs, and widgets for our digital assets	1	0.5

	• Testing UI elements such as CTAs, banners, page layouts, page designs, page flows, and target links for landing pages • Providing advice and guidance on the implementation of UX research methodologies and testing activities in order to analyze and predict user behavior • Adhering to style standards on typography and graphic design.		
Senior Mobile Engineer	• Technical communication • Code review • Mobile Frontend implementation • Be involved and participate in the overall application lifecycle • Main focus on coding and debugging • Collaborate with Backend developer • Provide training, help and support to other team members • Build high-quality reusable code that can be used in the future • Develop functional and sustainable web applications with clean codes • Troubleshoot and debug applications	1	1
Senior DevOps Engineer	• Cloud infrastructure setup • Deployment setup • Build pipeline review	1	1