Introduction

This introduction provides an overview of the Software Architecture document for the Loyalty Program System, it includes the purpose, requirements analysis, scope, definitions, assumptions, and the proposed System

Purpose

This document provides an architectural proposal of the loyalty Program System. The primary purpose is to allow the project to adapt to production scale with minimum effort and budget.

This document is indented to capture the architectural proposal and highlight the benefits of implementing it.

Scope

The scope of this document is the architecture recommendation for the Loyalty System that can be implemented with minimal effort while ensure the system can adapt to production scale

Assumptions

* The System is only indented to serve customers in one region.
* Current SAML based identity will not be replaced.
* Frequency of accessing the stored statement is unknown, even if statements are aged, will keep it accessible with low latency, so will not archive it to Glacier, later on based on access pattern that can be revisited to reduce cost.
* The following regions doesn’t support Dynamodb Encryption at Rest: Chine(Beijing) , China(Ningxia)

References

* Oracle Golden Gate With Amazon RDS <https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Appendix.OracleGoldenGate.html>
* <https://docs.aws.amazon.com/aws-technical-content/latest/oracle-database-aws-best-practices/oracle-database-aws-best-practices.pdf?icmpid=link_from_whitepapers_page>
* <https://docs.aws.amazon.com/quickstart/latest/oracle-database/overview.html>
* <https://docs.aws.amazon.com/codedeploy/latest/userguide/welcome.html>
* <https://docs.aws.amazon.com/solutions/latest/real-time-web-analytics-with-kinesis/architecture.html>
* <https://aws.amazon.com/lambda/>
* API Gateway : <https://aws.amazon.com/blogs/aws/amazon-api-gateway-build-and-run-scalable-application-backends/>
* API Gateway swagger importer : <https://github.com/awslabs/aws-apigateway-importer>
* Web hosting: <https://aws.amazon.com/websites/> , <https://aws.amazon.com/elasticbeanstalk/>
* Amazon S3 storage classes: <https://aws.amazon.com/s3/storage-classes/>
* Achieving S3 data to Amazon Glacier <https://aws.amazon.com/blogs/aws/archive-s3-to-glacier/>
* S3 Share Object : <https://docs.aws.amazon.com/AmazonS3/latest/dev/ShareObjectPreSignedURL.html>
* CloudWatch logs insight : <https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/AnalyzingLogData.html>

Requirements Analysis

* and Android Apps.
* The Web and mobile interfaces call a common API defined in Swagger and implemented in Node.js. This needs to be robust and scalable given its central to the entire platform
* Monitoring API performance is mandatory through logs, alerts and dashboards.
* Database schema is fairly simple but will hold millions of transactional entries related to the customers rewards earnings and spends.
* One of the API methods manipulates (GET/ POST / PUT) customer details which are stored and must remain in an on-premise Oracle database
* The Oracle DB is the system of record/single point of truth for customer data, so we need to ensure in the new architecture that this DB is robust, secure, Stable
* The backend generated statements (PDF format) which need to be accessed from both the web portal and mobile apps, and need to be archived indefinitely
* Have the ability to analyse telemetry (Click/ tab stream) from the interface and perform sentiment analysis on it. //TODO
* Enhance user experience by enabling the use of social account.
* Ensure System is secure //TODO
* Use AWS automation capability to improve productivity with minimal effort

Proposed architecture

1. Build API using API Gateway, that will ensure your API is scalable, reliable, secure, performant and cost effective. Also, you can describe your API using swagger that’s already exist in the current solution and import it into the API Gateway through tools like API Gateway importer. Use regional endpoint and have CloudFront that point to these regional endpoints to improve performance.  
   Use Lambda authorizers to support current bearer token AUTH strategy used.
2. Host Web Portal using AWS Elastic Beanstalk, configure it with at multi EC2 instances, load balancing, auto scaling and application health monitoring. That will give us full control over the underling AWS resources.  
   EB provides several options that control the security of the application, I propose using [Service role](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/using-features.managing.security.html) to associate with EB.

1. Store all the transactions in DynamoDB, that will allow store and retrieve large volume of records in milliseconds, without the performance overhead and scale limitation of relational DB. It also fit the purpose as we don’t need to support ACID transactions.  
   All data stored in DynamoDB with minimal effort using [Amazon Dynamodb Encryption at Rest](https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/EncryptionAtRest.html)
2. Oracle DB need to be set in private VPN, so we ensure its secure and accessible from the cloud system.
3. Use active-active database replication between Oracle DB and Amazon RDS using Oracle Golden Gate which guarantee zero-downtime migration and upgrades, disaster recovery, data protection, and in-region and cross-region replication.
4. SNS/SQS need to be added to ensure all Put/Post requests will successfully be executed to the DB.  
   In case of failure, failed operations will be queued in the dead letter queue to be retried later.
5. Statements will be stored in S3 that’s configured as S3 Intelligent-Tiering, which is designed to optimize costs by automatically moving data to the most cost-effective access tier.
6. API need to be developed to retrieve these statements by retuning singed URL for the statement that grant time limited permission to download it.
7. Security should be explained in the above points ( KMS, S3 security, VPC with DB, encryption when possible, API security using S2S authentication , etc //TODO
8. Use Cloud formation to provision this architecture stack, that will ensure full automation to provision the production environment also help in provision other environment to be used for testing and UAT.
9. Use CodeDeploy to application deployment to EC2, ELB, Lambda and API Gateway.
10. AWS CodePipline to enable continuous delivery by modelling, visualizing and automating the steps required to release builds.
11. Analyse log data from different sources like VPC flow logs, Route 53 logs, Cloud Trail logs, lambda logs using CloudWatch logs insight , that will gives you fast, interactive queries and visualizations.
12. S2S todo
13. Route 53 TODO
14. S3 prefix for performance