Stori Tech Challenge – Data Engineer – Sebastián Lozada

Contenido

Strengths and Weaknesses of Architecture

The architecture ensures **secure access** by restricting RDS through Security Groups and using IAM Roles to limit permissions across Lambda, API Gateway, and S3. It provides **scalability** as S3 and Lambda automatically handle growing workloads, and Redshift offers fast analytics as data increases.

The **serverless approach with Lambda** eliminates the need for server management, scaling based on demand and reducing operational overhead. The **event-driven workflow** triggers Lambda functions upon S3 uploads, ensuring real-time processing with minimal latency.

By **separating responsibilities**, S3 handles storage, RDS stores metadata, and API Gateway acts as the interface for uploads, improving maintainability. **Real-time analytics** is supported through AWS Rekognition, extracting tags automatically for easy insights via RDS queries.

The architecture is **flexible and extensible**, allowing integration with tools like Looker or adding services such as Glue or Redshift if needed. **CloudWatch monitoring** ensures visibility into system performance, simplifying troubleshooting. Finally, it is **cost-effective** with a pay-per-use model, making it ideal for fluctuating workloads.

Architecture faces challenges, misconfigurations in permissions can lead to access issues, and real-time RDS queries may introduce latency. Cost management is essential to prevent unexpected expenses from traffic surges or growing storage. Additionally, maintaining data consistency across S3, RDS, and analytics tools can be challenging, with network connectivity issues posing risks to seamless operations.

The architecture, while functional, is still pending full implementation and offers areas for improvement in future designs. Key recommendations include adding moderation to prevent the upload of inappropriate content, implementing file type restrictions to only accept certain formats, and introducing verification mechanisms to detect duplicate uploads. Additionally, setting file size limits will help control storage consumption, and user logging can be integrated to track and manage who uploads content, enhancing security and accountability.

Looker Studio Dashboard

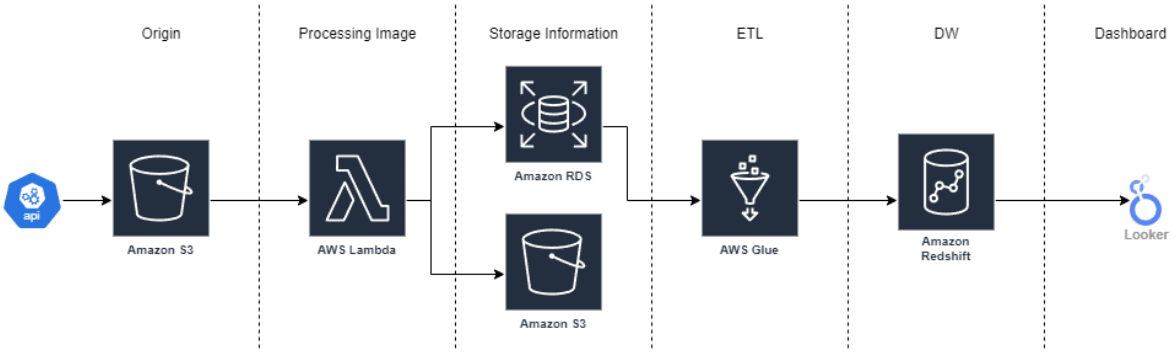
https://lookerstudio.google.com/reporting/080dbbf3-20fe-4c11-a3ef-f29647eedf08/page/EaFGE

Tech Challenge

> Hello! Welcome to Stori's Data Engineering coding assessment. Thank you for applying to Stori. This technical exercise is intended to take up to 3 days to complete. Coding Challenge: You find a time machine and go back in time! The year is 2000 and social media is just taking off. The hottest thing is a thumbnail generator: you input a big picture and get a small picture as output. Please build a thumbnail generator with the architecture that you like using the cloud services that you like. Deploy that service with the deployment tool of your choice (CDK, Cloudformation, SAM, Serverless Framework, Terraform, etc) and share your code with us: either put it in Python file(s) and email it back OR put it on GitHub and share the repo link. Write a paragraph (at most 1 page) of the strengths and weaknesses of the architecture you built. If you have time and interest, you can make your thumbnail generator special by adding cool new features and then explaining the cool features in your paragraph or video. Good luck! May you build an amazing thumbnail generator.

Architecture

This architecture displays a cloud-based pipeline using **AWS services** to generate image thumbnails, store metadata, and perform analytics. Below is a breakdown of the components and the steps involved in this solution that is implemented using IaC with Terraform:



Origin of the image

API Gateway

- **Purpose**: Acts as the front-facing service to receive requests for image uploads from the users
- **How it Works**:
  - Receives HTTP requests to upload images (PUT /upload).
  - Saves the image directly into a S3 Bucket of the original images.

Storage of the Original images

Amazon S3 (Original Images Bucket)

- **Purpose**: Stores the **original images** uploaded via the API Gateway.
- **How it Works:**
  When an **image is uploaded** to the source bucket, it triggers an **S3 event notification**.
  The event invokes the **AWS Lambda function** to process the image.

Processing the image

AWS Lambda (Thumbnail Generator)

- **Purpose:** The core logic to process images, generate thumbnails, and store metadata.
- **How it Works:**
  - Triggered by the S3 Event when an image is uploaded to the source bucket.
  - Create a 128x128 thumbnail using the Pillow library.
  - Analyzes the image using AWS Rekognition to generate image tags.

- Upload the thumbnail to the destination S3 bucket.
- Insert metadata (image key, tags, thumbnail link, size of the file) into Amazon RDS for structured storage.

Storage Information

Amazon S3 (Thumbnails Images Bucket)

- **Purpose**: Stores the **thumbnails** generated by the Lambda function.
- **How it Works:** The event invokes the **AWS Lambda function** to process the image.

Amazon RDS (MySQL Database)

- **Purpose:** Stores metadata about the processed images, including:
  - **Image Key:** Unique identifier of the image.
  - **Tags:** Image tags generated by Rekognition.
  - **Thumbnail URL:** Link to the stored thumbnail.
  - **Size:** File size in MB.
  - **Timestamp:** When the image was processed.
- **How it Works:**
  1. The Lambda function connects to **Amazon RDS MySQL**.
  2. If the database and table don't exist, the Lambda function creates them.
  3. Metadata for each processed image is inserted into the **RDS table**.

ETL and Data Preparation

AWS Glue

- **Purpose**: Extracts metadata from RDS and prepares it for analysis in Redshift.
- **How it Works:**
  1. **AWS Glue** connects to the **RDS MySQL instance** and extracts the metadata.
  2. The data is transformed (if needed) and loaded into **Amazon Redshift** for analytical processing.

Data Warehouse:

Amazon Redshift

- **Purpose**: Stores processed image metadata in a structured format for deeper analysis and reporting.

- **How it Works:**
    1. AWS Glue loads the metadata from RDS into Redshift.
    2. Redshift acts as the data warehouse, providing fast querying and analytics capabilities.

Data Visualization Tool

Looker Studio

- **Purpose**: Visualizes data stored in Redshift for insights.
- **How it Works:**
    1. **Looker** connects to **Redshift** to retrieve image processing insights, such as:
        - Frequency of uploaded images.
        - Popular image tags based on Rekognition analysis.
        - Storage trends (e.g., number of thumbnails generated).

Summary of Workflow:

1. **User uploads an image** via the API Gateway or uploads it to the S3 Bucket.
2. **The image is stored** in the **source S3 bucket**.
3. **S3 event triggers Lambda**, which:
    - Generates a thumbnail.
    - Upload the thumbnail to the **thumbnails bucket**.
    - **Analyzes the image** with Rekognition.
    - **Stores metadata** in **RDS MySQL**.
4. **AWS Glue** extracts metadata from RDS and loads it into **Redshift**.
5. **Looker** connects to Redshift to provide insights and analytics.

Terraform Implementation

Structure

1. Configure AWS Credentials using the Access Key and the Secret Key in the console to make the deployment.
2. Create a main.tf file with the code that is going to configure the different services.
3. Execute:
    a. terraform init - To initialize terraform.
    b. terraform plan – To review before applying the changes.
    c. terraform apply – To apply the changes.

Configuration Details

1. **AWS Provider Configuration**
   - This sets the region to **us-west-2**.
2. **S3 Buckets**:
   - **source_images bucket** stores the original images.
   - **thumbnails bucket** stores the generated thumbnails.
3. **IAM Roles and Policies**:
   - **Lambda Execution Role**: Allows Lambda to assume the role and interact with S3.
   - **S3 Access Policy**: Grants access to the Lambda function to read from and write to both buckets.
   - **Rekognition Policy**: Allows Lambda to call the **Rekognition** service for image analysis.
4. **RDS MySQL Setup**:
   - **RDS MySQL Database**: A MySQL instance is created, accessible publicly for this demonstration.
   - **Security Group**: Allows **MySQL traffic** (port 3306) from all IPs (should be restricted in production).
5. **Lambda Layers**:
   - **Pillow Layer**: For image processing.
   - **MySQL Connector Layer**: For connecting Lambda to the MySQL database.
6. **Lambda Function**:
   - **Processes images** uploaded to S3.
   - **Uploads thumbnails** to S3.
   - **Insert metadata** (image key, tags, thumbnail link) into **RDS MySQL**.
7. **API Gateway**:
   - **Setup for Uploads**: Provides a RESTful endpoint to trigger image uploads.
   - **Integration with Lambda**: Uses the **PUT** method to trigger Lambda.
8. **CloudWatch Log Groups**:
   - Collects logs for **Lambda function** and **API Gateway** for monitoring.
9. **S3 Notification**:

- Configure **S3** to trigger **Lambda** whenever a new object is created in the **original images bucket**.