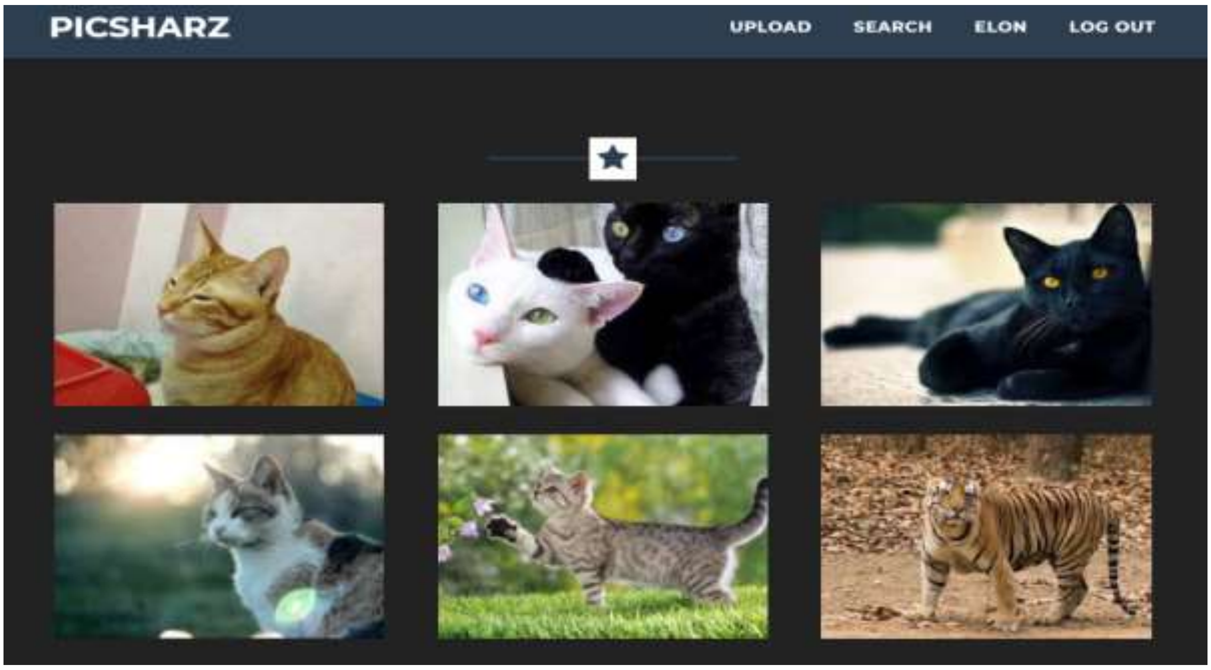
MTECH SE-4211(CC) PROJECT REPORT

PICSHARZ



FT TEAM 02
TEAM MEMBERS

ANURAG CHATTERJEE
BHUJBAL VAIBHAV SHIVAJI
LIM PIER
PRATYUSH MISHRA
TSAN YEE SOON

Table of Contents

1 Executive Summary 3
2 System Functionalities 3
3 Solution Outline 4
3.1 System Architecture and AWS Services 4
3.2 Data flow design 5
3.3 Novel Features in Picsharz 7
3.4 DevOps 8
3.5 Cost Projection 9
4 Conclusion 9
4.1 Lessons learnt 9
4.2 Future Enhancements 10

1 Executive Summary

In an era where most applications that migrate to the cloud adopt a lift and shift approach, we have designed and developed ground up a serverless application that is completely cloud-native using Amazon Web Services (AWS). We have built **Picsharz**, a social photo sharing application that we envision would cater seamlessly to 12,000 monthly active users.

The application heavily leverages the Platform as a Service (PaaS) offerings of AWS – Lambdas as serverless compute, Dynamo DB as NoSQL document database, EC2 to host the website, etc. We also utilise some of the innovative offerings from AWS like Rekognition and Elastic search for some of our functionalities. Picsharz provides most of the features that one would expect from a social image sharing application like user management, secured access to assets, interactions with the user interface and notifications on significant events.

Picsharz is also accompanied by useful aggregate analytics on the usage patterns that can be efficiently viewed by the administrators of the application using interactive visualisations. Finally, the development of the app was streamlined using code deploy such that any changes that were committed to source control would immediately reflect on the website hosted on EC2 automatically.

We have developed a serverless cloud-native application to compete with the best photo sharing apps that are available.

2 System Functionalities

Upload Pictures and Automatic Tagging
Picsharz allows for the uploading of pictures to share with the users who are following you. Upon upload, a thumbnail for the picture will be generated, and automated tags will be generated for it. Do not upload explicit images. The system administrator will be notified immediately should the system detect that explicit images have been uploaded.

Search for and Follow Friends
If you have a friend you know on Picsharz, all you have to do is to click on the search button and search for the name of your friend. You will be able to find your friend's profile page and follow him/her. If one day, you are no longer friends with this user, you have the option of unfollowing him/her.
Or, if you are interested in guitars, you can search for "guitar", and all guitar-related images will appear. From here, you will also be able to see the users that upload guitar images frequently.

Friend Activity Feed
You will get an active feed of the pictures that your friends that you are following have uploaded. These are the thumbnails, and if you want to get a higher resolution picture, you can click on the image to get to the image screen, where you can see more details about the image.

Like a Picture
If you like a picture, you have the option of letting your friend know by clicking on "Like". The image will display the number of people who have liked the image thus far.

User Management
Picsharz handles the sign-up, log-in and authentication process from the web application. By default, users who are not logged in will see a feed of the most recent images. When you are logged in, you will see the friend activity feed.

Administrator Dashboard
The dashboard covers the administration data and analytics of the Picsharz application. This dashboard will only be available to administrators via Amazon QuickSight.

Automated Backup
The Picsharz database will be backed up automatically on a regular basis.

3 Solution Outline

3.1 System Architecture and AWS Services

Amazon Web Services has been leveraged to take advantage of the various PaaS offerings that power the backend of the solution.

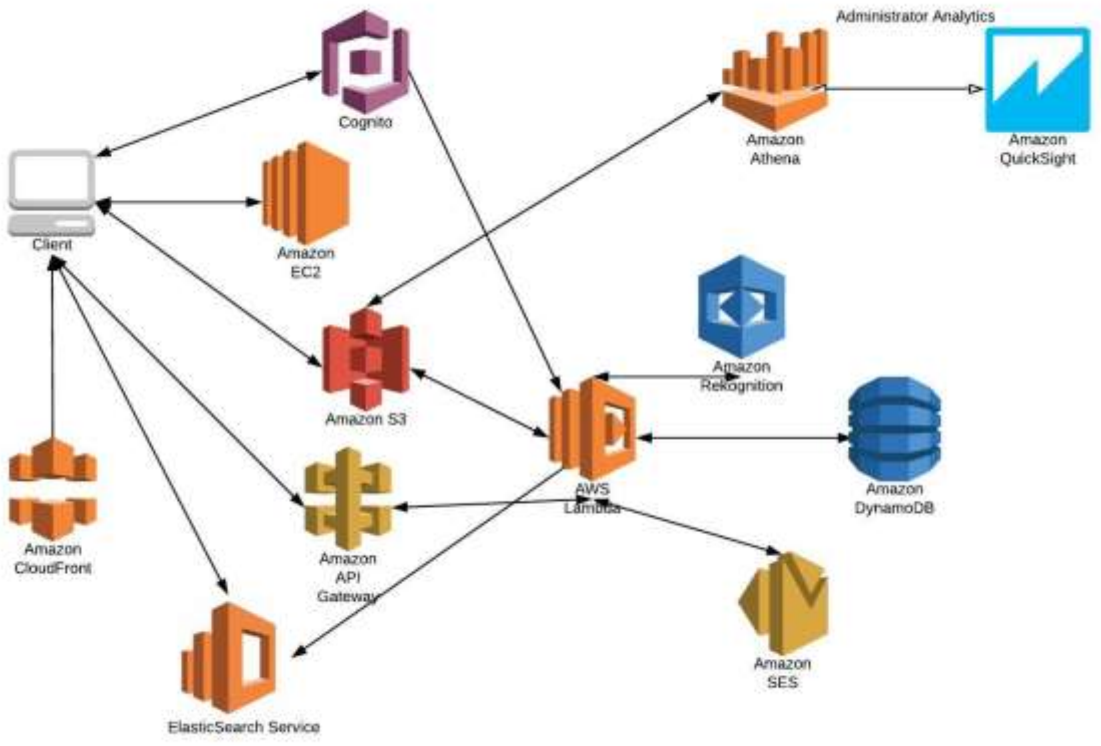


Figure 1 Backend architecture for Picsharz

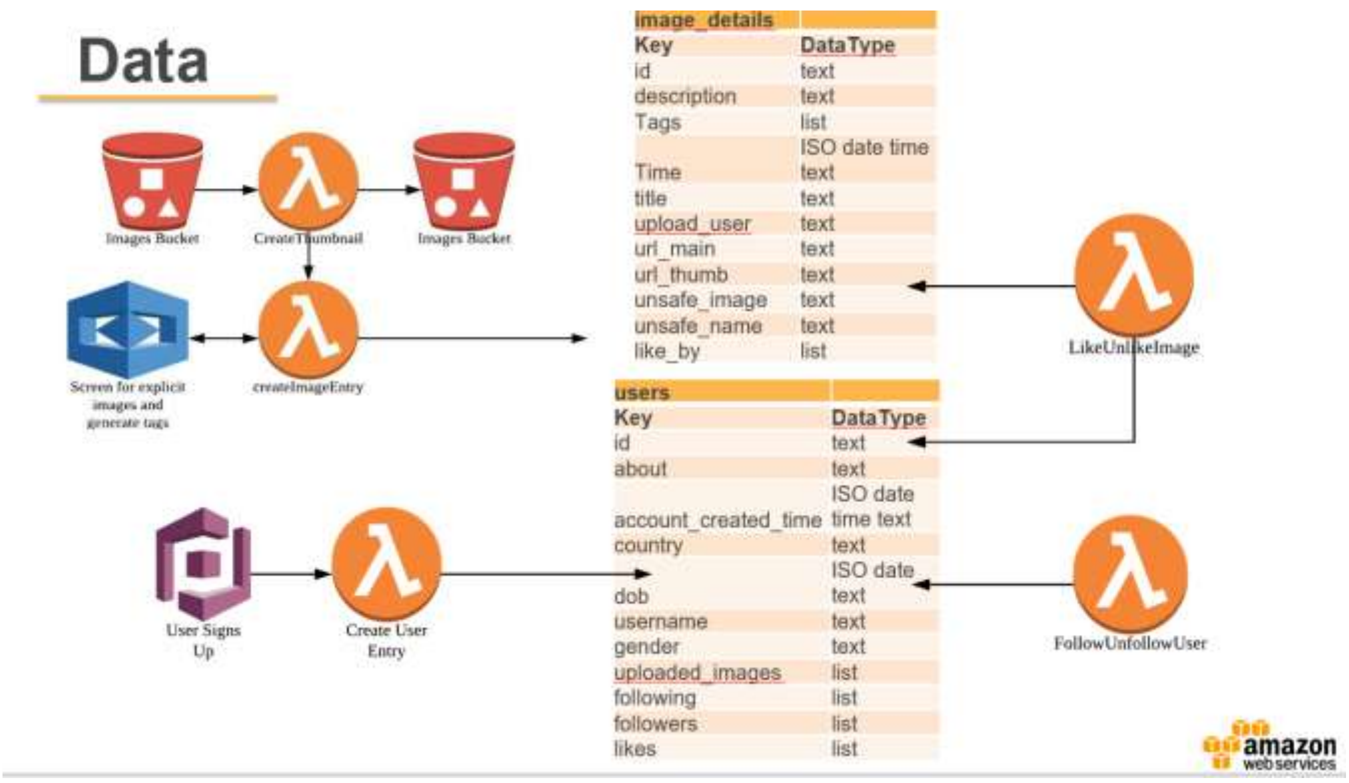
The below is a brief description of how the different managed services from AWS has been leveraged to build the backend for the system based on the above image.

- The front-end components for the end users – HTML, CSS and JavaScript are deployed to an **AWS EC2** instance from where the end users access the same.
- **Cognito** has used for the user management flow, i.e. user sign up, sign in and sign out. Internally it also connects with AWS Simple Email Service (SES) to send verification codes via email after a user signs up. A lambda is also set up to trigger after user sign up that will sync the user details to DynamoDB.
- **AWS S3** is leveraged to store the images that are uploaded by the users. There are two private buckets. The first bucket stores the image that is uploaded by the user as is with the high

resolution. The second bucket stores a thumbnail of the image that is generated by a lambda which triggers after the main image is stored in the S3 bucket.

- **Cloudfront CDN** is leveraged to store the most popular images that are accessed when a user accesses PicSharz.
- **API gateway** is leveraged to act as the main controller to route client requests to the appropriate lambdas. The APIs exposed via API gateway is authenticated to prevent anonymous access.
- The main server-side functionalities – generation of user feed, image upload, thumbnail creation, following users, like/unlike images are handled via **lambdas**. The lambdas together with the API gateway are used to create a microservice architecture.
- **Simple Email Service (SES)** is used to send emails to users when they are being followed or to the moderator when an explicit image is uploaded via the lambdas.
- **AWS Rekognition** is leveraged for the machine learning features that are utilised to generated tags for the images and to flag whether an image contains explicit content.
- **Dynamo DB** is the central transactional database that contains two collections – user details to contain the details of the users and their interactions with the system and image details to contain the image specific details and what are the interactions that are performed on the image.
- **Elastic search service** is leveraged to index all the text that is input while creating a user or uploading an image including the auto-generated tags. Streams are enabled for both the collections in the Dynamo DB that trigger lambdas when new documents are added to the collections which sync the stream data into the Elastic search server.
- **Amazon Athena** is leveraged to analyse the Dynamo DB collections back up in the S3 buckets using standard SQL
- **Amazon QuickSight** is used to build a custom cost analysis visualisation and aggregate user analytics for the administrators of the system that utilise the queries from Amazon Athena.

3.2 Data flow design



image_details	
Key	DataType
id	text
description	text
Tags	list
	ISO date time
Time	text
title	text
upload_user	text
url_main	text
url_thumb	text
unsafe_image	text
unsafe_name	text
like_by	list

users	
Key	DataType
id	text
about	text
	ISO date
account_created_time	time text
country	text
	ISO date
dob	text
username	text
gender	text
uploaded_images	list
following	list
followers	list
likes	list

Figure 2 Data flow design

The above image shows the database design and interaction of lambdas with tables. The primary tables are as follows:
1. **users**: This table stores the details of all the users like country, id, his followers, following, liked images.
2. **image_details**: This table stores the details of all the images like image id, uploaded by, image title, tags, time.

Following are the interactions between lambdas with the database:

1. Whenever new user signs up, CreateUserEntry lambda gets triggered, and user entry is made inside **users** table.

2. If a user uploads a new image, then a thumbnail of the image gets created using CreateThumbnail lambda and then image entry gets created in the **image_details** table by using createImageEntry lambda.

3. If User A follows or unfollows User B, then FolllowUnfollow lambda gets triggered and "following" field for User A as well as "followers" field for User B are updated simultaneously.

4. Whenever user likes or unlikes an image, then FollowUnfollowUser lambda gets triggered, and "like_by" field in **image_details** table as well as "likes" field in **users** table are updated simultaneously.

3.3 Novel Features in Picsharz

Serverless, Cloud-Native Architecture

Picsharz is built on a fully serverless, cloud-native architecture. Our usage of AWS Lambda, as well as managed AWS services like DynamoDB allows for built-in scaling. By default, AWS Lambda can handle 1000 total concurrent executions. If our user base increases to a point where we need more than this, AWS will add 500 concurrent executions per minute until your account safety limit has been reached, or the number of concurrently executing functions is sufficient to successfully process the increased load. The sole purpose of our existing EC2 server is to serve the web pages, and we have set this to auto-scale should there be increased traffic.

Machine Learning using AWS Rekognition

Picsharz makes use of AWS Rekognition to do auto-tagging of images with Machine Learning. As soon as a photo is uploaded to the AWS, the lambda that handles the image upload triggers AWS Rekognition for two purposes. The first is to auto-generate image tags that describe the content of the image. The second functionality is to flag the image out for inappropriate content and inform the system administrator should such images be detected.

Searching using AWS Elastic Search Service

Picsharz makes use of AWS Elastic Search Service to do a full text search on tags and names. Lambdas have been developed to sync the text content, including the auto generated tags whenever there is a new user or image uploaded, from the Dynamo DB streams to the Elastic Search server. The search results are displayed on an interactive search results page.

Notification Using Simple Email Service

Emails are sent to users when someone on the Picsharz network follows them. Also, when explicit images are uploaded by a user, moderators are notified and alerted immediately via email.

Backup of DynamoDB to S3 using Data Pipeline

Data Pipeline allows us backups back-dated longer than what DynamoDB offers (35 days). It converts DynamoDB data into JSON or CSV formats for analytics and machine learning, in the process separating analytical queries from production traffic on our DynamoDB table. This allows us to preserve DynamoDB read capacity units (RCUs) for important production requests.

Cognito for User Log-In and Authentication

Cognito allows users that are signed-in to it to be linked to the federated pool for authentication to AWS services. This allows users to request an authentication token that they can use to authenticate the calls to the API Gateway. In this manner, the authentication design is kept simple and contained within AWS.

3.4 DevOps

3.4.1 Continuous Deployment using CodeDeploy

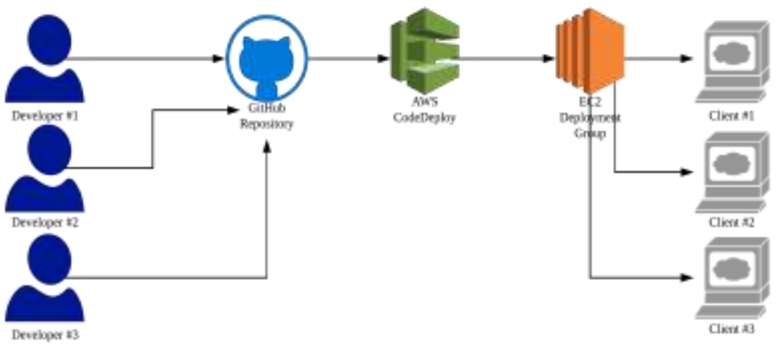


Figure 3 Pisharz Continuous Deployment

We practice Continuous Deployment in Picsharz. Each of the developers is pushing to a GitHub Repository. As soon as a commit is pushed to the "master" branch in GitHub, AWS CodeDeploy takes the commit and deploys it to an EC2 Deployment Group. If no EC2 servers are running, CodeDeploy launches the deployment group and sets up the environment for the web server. Otherwise, CodeDeploy updates the web server pages with the latest commit changes. Because the servers are in a deployment group and roll-out can be controlled, we can serve the clients with no downtime.

3.4.2 Cloud Formation used to deploy AWS Cost and Usage Report Data

We do like to load our AWS Cost and Usage data up to the QuickSight Analytic service to create a share-able dashboard for authorised us. To do this, we would like to deploy serverless solutions (Lambdas) and the relevant resources that process the Extraction, Transform and Load operations, creating Athena queries at the end that can be used by Quicksight.

For this new ETL operation, we explored using CloudFormation following AWS Big Data Blog instructions to create and manage the collection of AWS resources and provisioning. The diagram below is the representative of the resources diagram after creating the YAML codes in CloudFormation. The IaaC are executed, and the resources are created and can also be rollback easily if needed or if there are errors. The lambda functions are draw from AWS Big Data Blog tutorial.

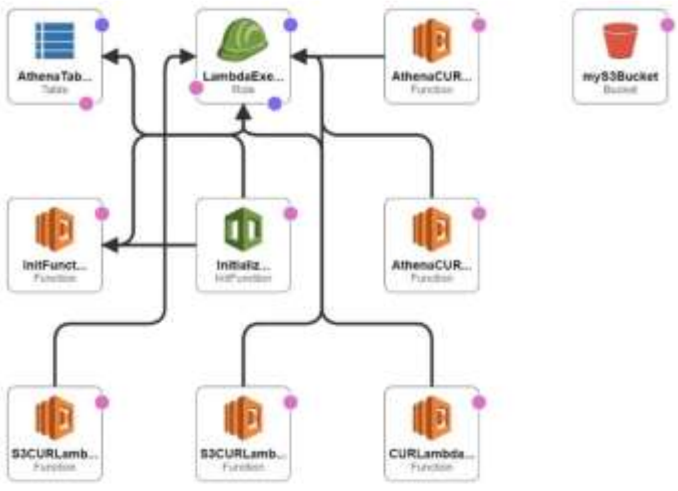


Figure 4 Architecture Formed using CloudFormation to get Cost and Usage Report

3.5 Cost Projection

Region	Registered Users	M.A.U.
US East (N. Virginia)	40,000	12,000

Figure 5

Lambdas	Rekognition	S3 Ops & Store	S3 Data Out	S3 Glacier	Dynamo DB
$ 11.06	$ 720.00	$ 186.88	$ 2,597.76	$ 7.50	$ 8.59

Elastic Search	API Call	Athena	EMR	SES	Analytics	WebServer
$ 52.56	$ 50.86	$ 31.50	$ 5.28	$ 7.90	$ 45.37	$ 72.00

Total = USD 3797.262 Per Month

Figure 6

The above shows our baseline cost projection. This scenario assumes eight months of operations using US East-1 AWS region (average 250-350 ms latency-good enough for our application), and Picsharz managed to sign up 40,000 users, about the population of NUS. Following similar online picture sharing website Imgur.com which has about 30% Monthly Active Users (M.A.U.) from their total registered base, we also forecast picsharz will have about 12,000 M.A.U.

We also assume a few points about the average active users, such as in each month the user will upload about 30 pictures, log in 60 times. Our worksheet further breakdown calculation to the exact cost details such as each login will view five feeds which in turns process 20 S3 got of thumbnails images, 1 API call and triggered one 128MB lambda for 800ms.

From the cost breakdown, we can see that S3 Data Out consume is the single most expensive component. This cost assumes the worst-case situation of no CDN caching, and each S3 get request has a data transfer associated with it.

For the development of Picsharz, we were charged US0.014 by AWS, spending mostly on the free-tier provided by AWS.

4 Conclusion

4.1 Lessons learnt

The below are the lessons that have been learnt during the development of the application:

a. Care must be taken when following the AWS official tutorials as the resources that are generally asked to be provisioned may not be suitable for the desired use case. The tutorials may ask to engage more costly AWS services which can be avoided in some cases.
b. AWS services are updated periodically, and so one needs to do some prior research to find the optimal service for a specific use case. Tutorials from blogs outside of AWS may not always be using the best way to go about doing a task.
c. Fined tuned configuration of lambdas is not always possible. For example, if you already know that your application requires specifically 2 GPUs and 256gigs of memory to run, lambdas may not be the best choice.

d. Lambda based architecture promotes vendor lock-in. The lambdas as such cannot be easily ported to another vendor which can be done with traditional web apps hosted on an EC2 server. However, tools like Cloud Foundry may be able to provide vendor agnostic tools to use serverless architectures across platforms.

4.2 Future Enhancements

1. **Further Leverage CloudFront:**

Popular images that require to be displayed large number of times can be moved to CloudFront CDN to get better cost benefits for those images. A list of ~24 such images will be maintained and updated at low frequency (1 update/day). This will ensure that in case large numbers of users attempt to see the popular image, either by search or by loading their feeds, the GET requests are not triggered for the images, and we will not be charged for the same.

2. **Enhanced Moderation and Censorship:**

Currently, Amazon recognition classifies images as *Suggestive, Safe* or *Explicit Nudity.* Due to our policy of light-handed censorship, images tagged "Suggestive" would be temporarily invisible from the feed pending review by the moderator. Explicit images can be outright marked and deleted from the database without moderator intervention.

3. **Automatic Lambda Deployment in Continuous Deployment:**

In the future, the lambda deployment can be part of the Continuous Deployment. This can be done via using CodePipeline, CodeBuild and CloudFormation to create a Deployment Stack. During the project, we had tested this with another account and verified that it was able to create a Lambda using the above tools. However, due to time constraint, we were unable to integrate this with the Picsharz project's deployment.

4. **Tags Analysis:**

Our current tags Analysis calculates the frequency of a discussion by creating a word bag of image name, image tags and image description; however, this approach does not produce very good results. Our future approach involves running the word bag through a stop word filter into a text mining machine developed by us with **AWS Sagemaker** since *AWS Comprehend* is not suitable for such operations; or, requires a more comprehensive overview from us.

5. **Asynchronous Processing**

Currently the lambdas associated with image creation, which perform the machine learning tasks like tag generation, explicit content filtering are triggered synchronously before inserting the image details. While this process is sufficient, even preferred, for small-scale operations, at larger scale or high volume of upload, these lambdas are likely to slow down the process of adding image details to the DynamoDB to be available for users.

In such scenarios, it would be beneficial to prioritize image thumbnail creation and image database insertion tasks. The test for image suitability and the automatic generation of tags can be pushed to processing queues created with Amazon Simple Queue Service and handled asynchronously.