MTECH SE4211
CLOUD COMPUTING
PROJECT REPORT

PinUp APPLICATION

TEAM MEMBERS

CHAN SU-WEN PHILEMON
EDUARD ANTHONY CHAI
PRANSHU RANJAN SINGH
SIDDHARTH PANDEY

1.0 SHORT DESCRIPTION

PinUp is a social media platform web application that allows users to upload pictures, follow friends and get feeds on friends' activity and like photos of friends. Users of this application are able to search for other users (friends) and follow them. The photos that a user posts will appear on the feed of the user's friend almost real-time.

2.0 TECHNOLOGY STACK OVERVIEW

Technology stack

	Description
Github	Version control.
Jenkins	Automate building and deployment process.
Docker	Used to containerized our application.
Docker Hub	Docker image repository.
GraphQL	Query language for API. It is used to put/fetch/modify application data.
ReactJS	Used to create our client application.
Semantic UI	React components library, enable us to develop our pages faster.
D3	Used to create data visualization in application's dashboard.
Apollo Client	GraphQL client.
AWS Amplify	Javascript library that enable our application to connect to AWS services.

Amazon Web Services stack

	Description
Cognito	Provides authentication, authorization, and user management for our application.
AppSync	Fully managed GraphQL service with real-time data synchronization.
Simple Storage Service (S3)	Used to store application images
DynamoDB	NoSQL database used to store application data
Elastic Compute Cloud (EC2)	VMs that is used to deploy our application and Jenkins
Elastic Beanstalk	Used to deploy our application, manage application's environment, provisioning, load balancing, scaling.
Lambda	Serverless compute that is triggered by some application's events such as user registration and add new post.
Rekognition	Used to detect object in our post's images and use the results as images' tags.

3.0 PROJECT HIGHLIGHT

3.1. CONTINUOUS INTEGRATION AND CONTINUOUS DEPLOYMENT (CI/CD)

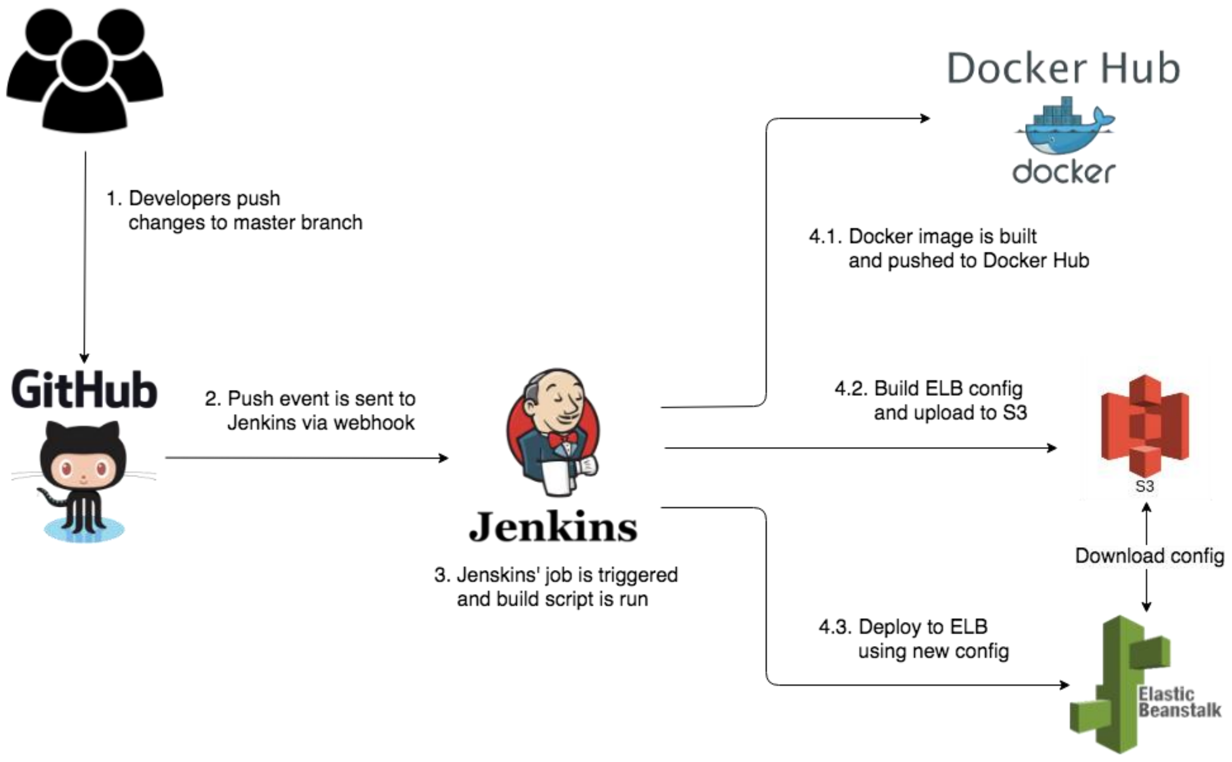
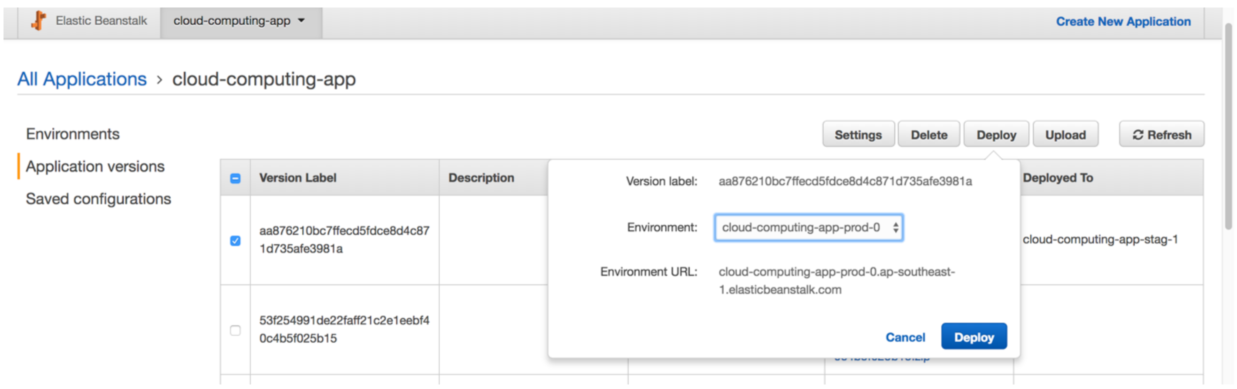

1. Developers push changes to master branch
GitHub
2. Push event is sent to Jenkins via webhook
Jenkins
3. Jenskins' job is triggered and build script is run
Docker Hub
docker
4.1. Docker image is built and pushed to Docker Hub
4.2. Build ELB config and upload to S3
S3
Download config
4.3. Deploy to ELB using new config
Elastic Beanstalk

Figure 1 *Continuos integration and continuous deployment pipeline*

CI/CD flow

1. We use Github as our source management and versioning. A new feature will be created in new branch. Pull request is created when the branch is ready to be merged to master branch.
2. Merge or push event is created when developer merge his branch to master branch. This event is then detected and trigger a webhook to send event to Jenkins.
3. Jenkins' job is triggered and pull the updated code from Github and run the build script.
4. The build script consists of 2 main tasks:
   a. Build docker image from our source code and push the image to Docker Hub
   b. After new image is pushed, Jenkins build a configuration file that is going to be used for deployment to Elastic Beanstalk and upload to S3
   c. Jenkins then send deployment request to Elastic Beanstalk where it will download the new config file from S3 and deploy to specified environment

The flow above will deploy the updated application to our staging environment. Manual deployment still needs to be done to deploy application to production environment. This can be done through Elastic Beanstalk console.



	Version Label	Description	Deployed To
✓	aa876210bc7ffecd5fdce8d4c871d735afe3981a		cloud-computing-app-stag-1
	53f254991de22faff21c2e1eebf40c4b5f025b15		

Version label: aa876210bc7ffecd5fdce8d4c871d735afe3981a

Environment: cloud-computing-app-prod-0

Environment URL: cloud-computing-app-prod-0.ap-southeast-1.elasticbeanstalk.com

Figure 2 *Deploying to production environment*

3.2. BACKEND

3.2.1 GRAPHQL: A QUERY LANGUAGE FOR API

GraphQL is a query language for your API, and a server-side runtime for executing queries by using a type system you define for your data. GraphQL isn't tied to any specific database or storage engine and is instead backed by your existing code and data.

How it works

1. Resources defines by a GraphQL schema
2. Client sends query and server orchestrates data
3. Efficient (network bandwidth)
4. Built-in API documentation

GraphQL features

1. No overfetching, no underfetching
2. Pagination
3. Structured data – data is structured as requested
4. Query driven
5. Realtime capability

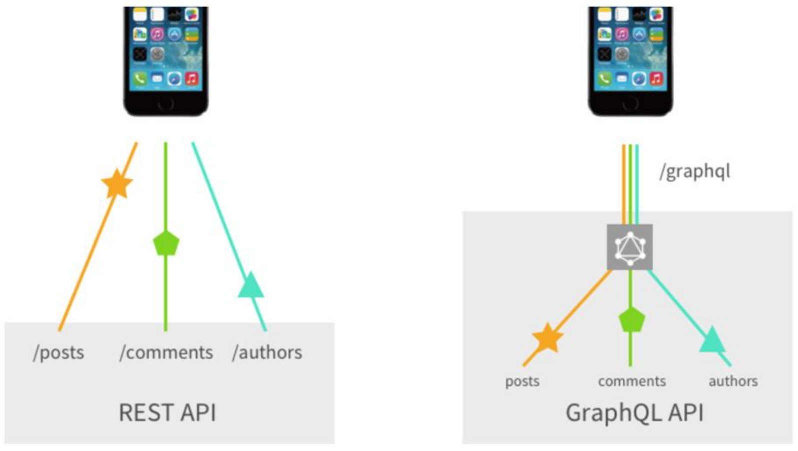


Figure 3 REST API vs GraphQL API

One of the main reason why we chose to use GraphQL over Rest API is the efficiency of the bandwidth. In this era, where the application needs to be fast and responsive, GraphQL offers capability to fetch the data as you needed. This make sure the data is not overly or under fetched. Below is comparison of REST and GraphQL using simple example for our home feed.

Data schema

```
type Post {
  id: ID!
  username: String
  profilePic: S3Object
  user: User
  caption: String
  likes: Int!
  file: S3Object
  tags: String
  createdAt: String
  liked: Boolean
}
```

Using REST API

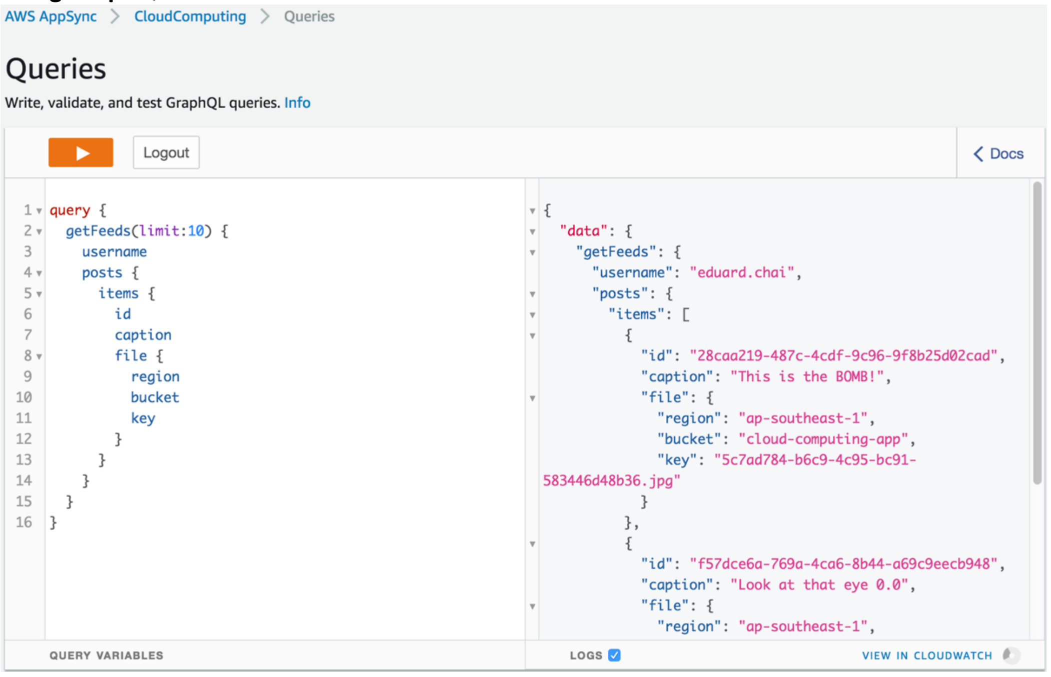
request

```
GET /feeds
```

response

```
{
  "results": [
    {
      "post": {
        "id": ...,
        "username": ...,
        ...
      }
    },
    {
      "post": {
        "id": ...,
        "username": ...,
        ...
      }
    },
    ...
  ]
}
```

Using GraphQL



Queries

Write, validate, and test GraphQL queries. Info

```
query {
  getFeeds(limit:10) {
    username
    posts {
      items {
        id
        caption
        file {
          region
          bucket
          key
        }
      }
    }
  }
}
```

```
{
  "data": {
    "getFeeds": {
      "username": "eduard.chai",
      "posts": {
        "items": [
          {
            "id": "28caa219-487c-4cdf-9c96-9f8b25d02cad",
            "caption": "This is the BOMB!",
            "file": {
              "region": "ap-southeast-1",
              "bucket": "cloud-computing-app",
              "key": "5c7ad784-b6c9-4c95-bc91-
583446d48b36.jpg"
            }
          },
          {
            "id": "f57dce6a-769a-4ca6-8b44-a69c9eecb948",
            "caption": "Look at that eye 0.0",
            "file": {
              "region": "ap-southeast-1",
```

From the example above we can see that REST API is over fetching the data. In REST API, the structured of the response data is normally set by the server. Most of the time, it causes client to fetch data that he might not need. But using GraphQL API, client can specify what data he needs from Post object.

3.2.2 AWS APPSYNC

AWS AppSync is a managed GraphQL server service that gives additional benefit of updates data for offline users as soon as they reconnect. However the offline capability is not why we use AppSync in the first place. We main reason is because it is managed by AWS and serverless by design. We don't need to spawn additional EC2 instance to host our GraphQL server. We are billed separately for query and data modification operations, and for performing real-time updates on your data.

AppSync Architecture

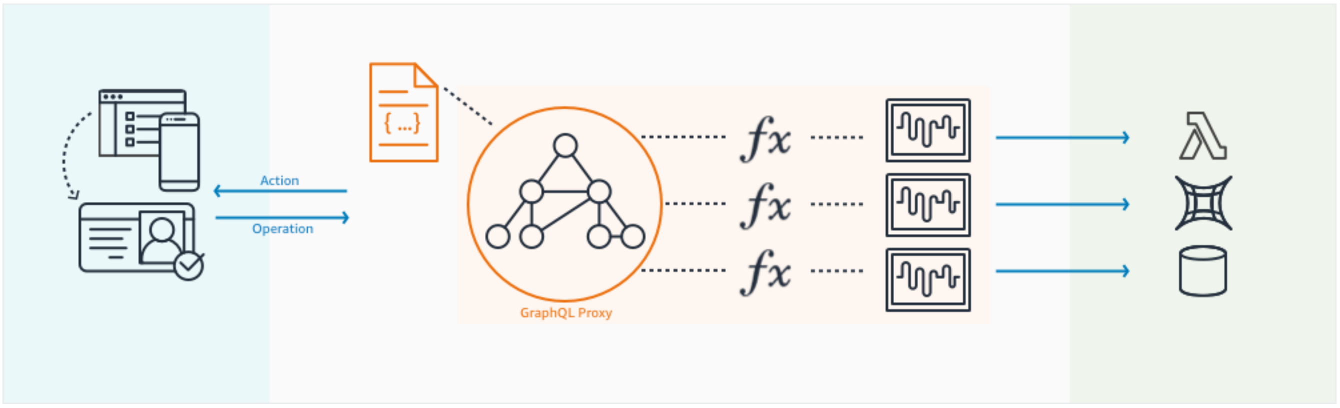


Figure 4 *AWS AppSync Architecture*

How it works

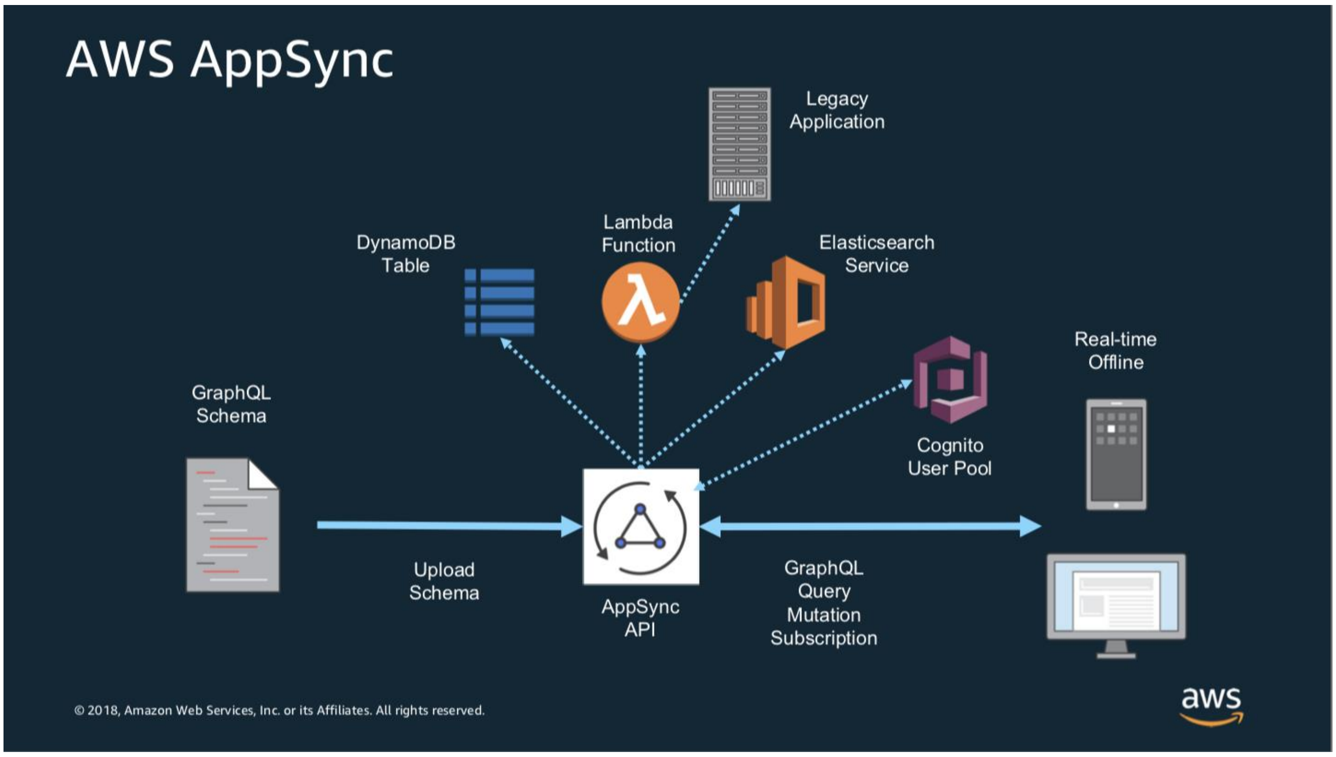


Figure 5 *AWS AppSync system overview*

3.3 APPLICATION ARCHITECTURE

Overview

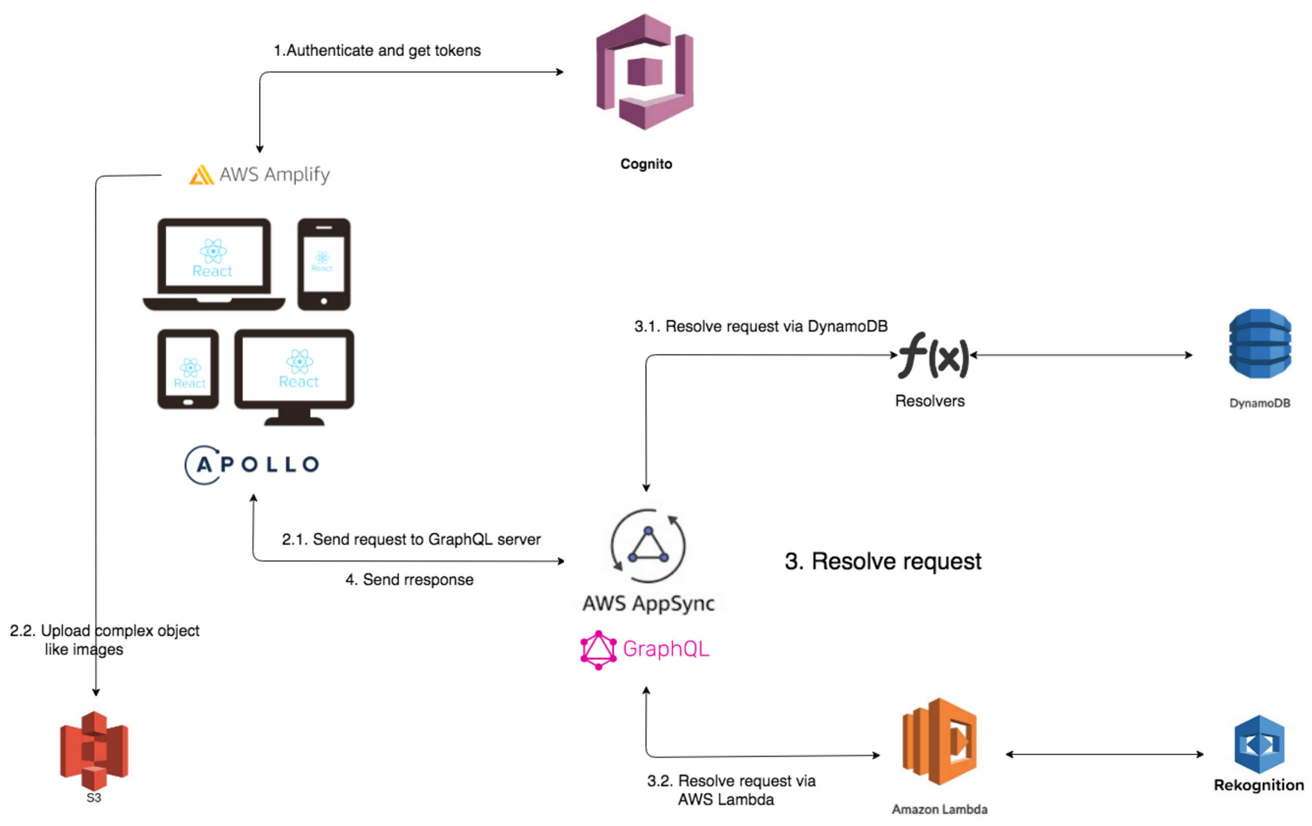

1.Authenticate and get tokens
Cognito
AWS Amplify
React
APOLLO
3.1. Resolve request via DynamoDB
f(x)
Resolvers
DynamoDB
2.1. Send request to GraphQL server
4. Send rresponse
AWS AppSync
GraphQL
3. Resolve request
2.2. Upload complex object like images
S3
3.2. Resolve request via AWS Lambda
Amazon Lambda
Rekognition

Figure 6 *PinUp application architecture*

Request/Response Flow

1. Users authenticate themselves using their credentials, it is verified against **AWS Cognito** user pools, authorization and token management is done using **AWS Amplify**.
2. **AWS AppSync** works as API gateway, through which most of the request will be routed. Complex objects will be uploaded to **S3** bucket directly.
3. For all other user activity like fetching feeds, like/unlike, follow/unfollow AWS AppSync resolvers will be invoked.
   - 3.1. **Dynamo DB** resolvers will be used to perform operation like query, mutation and subscription on schema objects.
   - 3.2. For image tag attribute, **AWS Lambda** resolver will be invoked which will communicate with **AWS Rekognition** to send the image as byte array and receive the corresponding tags.

Signup flow

1. User inputs his choice of username, password and email id. And clicks sign up.
2. To complete sign up process the email id must be verified. An email with the verification code which has to be entered to proceed with sign up is sent to user.
3. The user enters the verification code,
   - 3.1. Upon successful verification, AWS Lambda is triggered which add an entry to Dynamo DB table.

3.2. The User is redirected to log in form, where he can enter his credentials and log in

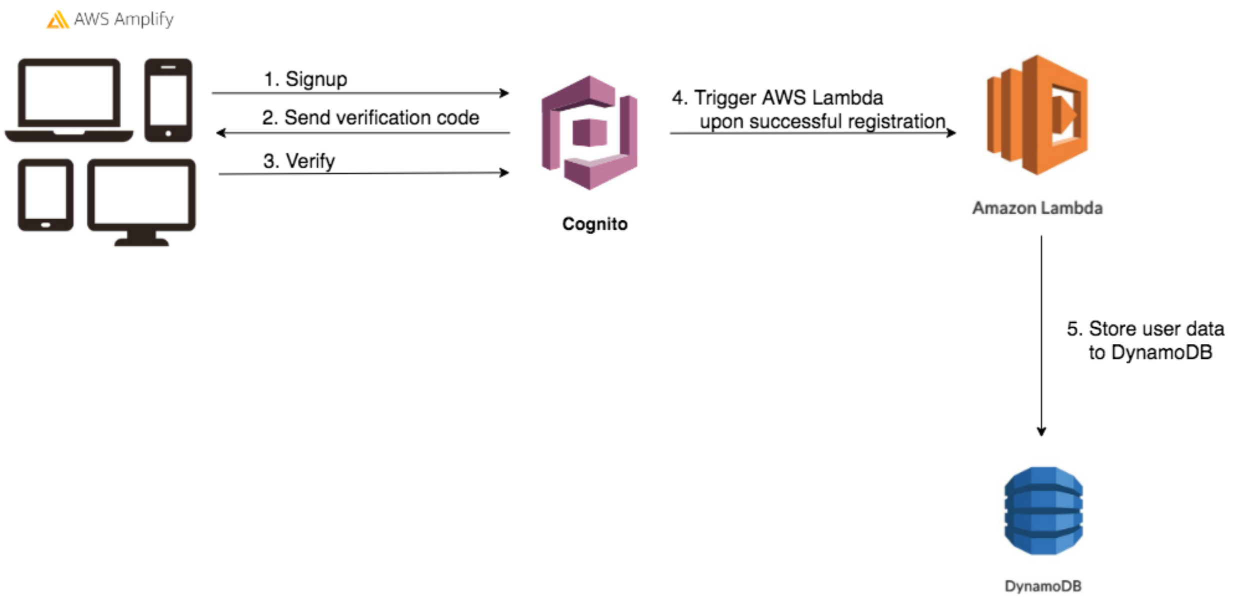

AWS Amplify
1. Signup
2. Send verification code
3. Verify
Cognito
4. Trigger AWS Lambda upon successful registration
Amazon Lambda
5. Store user data to DynamoDB
DynamoDB

Figure 7 *PinUp Signup flow*

Add new post flow

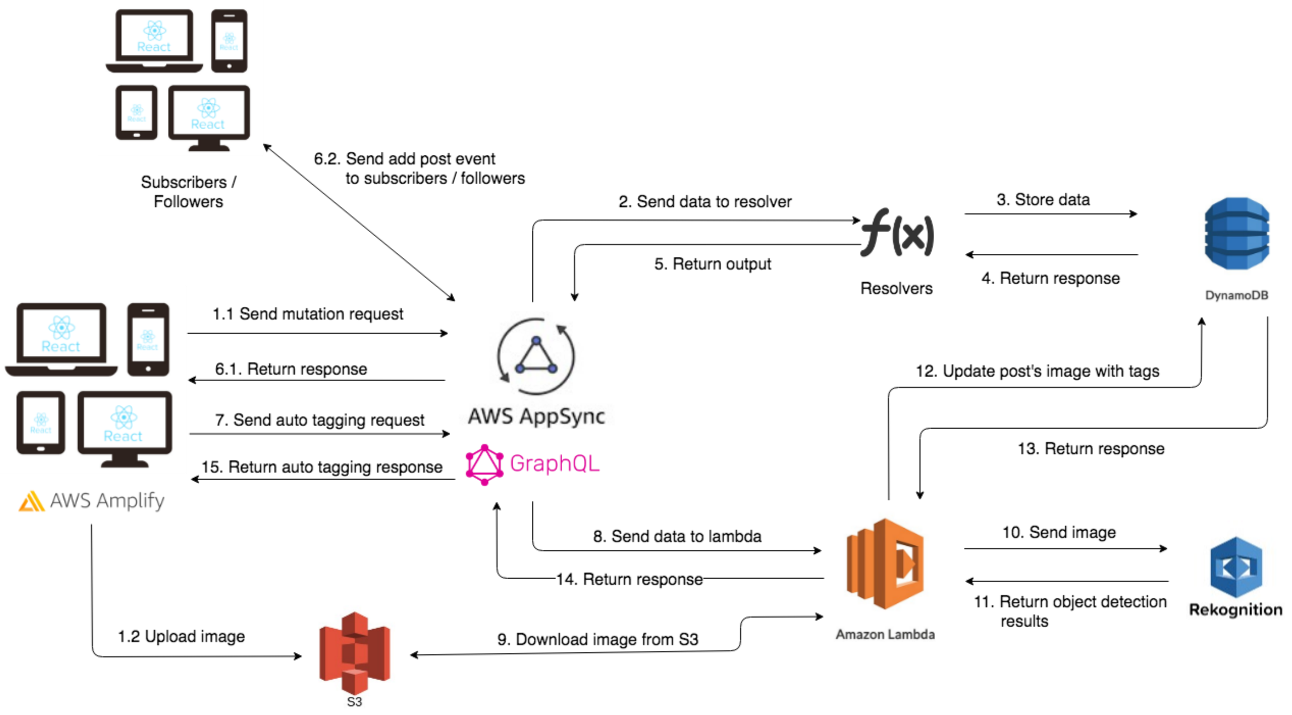


Figure 8 *PinUp Add new post flow*

1. User uploads the new image and caption.
   1.1. It sends a mutation request to AppSync server to add new post entry with corresponding caption.
   1.2. Meanwhile the image is also uploaded to S3 bucket.
2. AppSync delegates the operation to corresponding resolver.
3. The resolver runs the query to get/put/modify the record in the Dynamo DB table.

4. The output of the query is returned to resolvers, which then coverts into a specific output format and returns the output JSON.
5. AppSync then returns the response to client.
6. The client processes the response in a call back triggered after the promise object is received.
7. The triggered callback then sends the image tagging request to AppSync server.
8. AppSync delegates the work to image tag resolver which is lambda function and sends in image's S3 link as an argument.
9. The lambda downloads the images from S3 and converts it into a byte array.
10. The byte array is sent to AWS Rekognition which uses machine learning to tag the images.
11. AWS Rekognition returns the top 5 tags with more than 90% probability as request by lambda.
12. Lambda finally converts the tag into comma separated value. Executes a Dynamo DB post modification query to add tag to the post.
13. Dynamo DB returns the output to the query.
14. The lambda finally returns a successful tagging response back to resolver.
15. The resolver converts the response to json and sends to AppSync server. AppSync server return the added tags to client where it can further processed using callbacks.

3.4 OTHER FEATURES

Search Users

1. A user can search other users by typing the username in the search bar.
2. The search results are displayed as the user starts typing the characters. If the user keyed 'ed', all the users having 'ed' anywhere in their username are displayed.

Follow/Unfollow Users

1. The search results (from Search Users Feature) displays a user card, which contains a Follow/Unfollow button.
2. If the logged in user has not followed the searched user, the logged in user can follow by clicking on Follow button. The button changes to Unfollow.
3. To unfollow the searched user, click the Unfollow button.

Posts Timeline

1. The Posts Timeline of a user shows all their posts and followed users' posts.
2. All the posts are not loaded on users' timeline at once. As the user scrolls, other posts are loaded until there are no new posts to display.

Like/Unlike Post

1. A user can like/unlike a post by clicking on the heart button attached to each post.

My Profile Page

1. My Profile Page contains 3 tabs, 'My Posts', 'Followers' and 'Following'.
2. My Posts tab lists all the posts of the logged in user.
3. Followers tab shows the list of users that the logged in user has followed. The logged in user can Unfollow any of the listed users from this tab also.
4. Following tab shows the list of users that are following the logged in user. The logged in user can Follow any of the listed users from this tab also.

Dashboard

1. The Insights tab shows the dashboard. There are three tabs on the dashboard, 'Your Insights', 'Popular Posts' and 'Popular Users'.
2. Your Insights tab is specific to each user. It displays three visualizations:
    a. A line chart to display number of posts posted each day for past 7 days.
    b. A pie chart to display the number of likes on a specific post against the number of friends (users that follow the logged in user) of the user.
    c. A bar chart to display top 10 most generated tags along with their count.
3. Popular Posts tab is generic to all users. It displays two visualizations:
    a. A bar chart to display top 10 most liked posts along with their like count across all users.
    b. A bar chart to display top 10 most generated tags along with their count across all users.
4. Popular Users tab is generic to all users. It displays two visualizations:
    a. A bar chart to display top 10 most active users (based on number of posts only) along with their posts count across all users.
    b. A bar chart to display top 10 most followed users along with their followers count across all users.

Account Settings Page

1. The Account Settings Page allows user to manage their personal information. It consists of two tabs, 'Edit Profile' and 'Change Password'.
2. Edit Profile tab allows users to edit their personal information and update their profile picture.
3. Change Password tab allows users to change their password.

4.0 COST EVALUATION

4.1 AMAZON WEB SERVICES PRICING

In order to estimate the costs of running various Amazon Web Services, we assume that there are 3 posts per user a day, and 100 thousand Monthly Active Users (M.A.U.), activity per user per month is 1500 and image that the user uploads to S3 bucket is of 1 MB size.

Cloud Service	**Costs ($USD)**
AppSync	913.50
Cognito	0.0055
Standard S3 Storage	1905.98
DynamoDB	80.87
Elastic Compute Cloud (EC2)	395.49
Lambda	1.60
Rekognition	2.30
Grand Total	**3299.7455**

All the costs for cloud services mentioned above except Rekognition is according to Singapore Region. The cost of Rekognition is according to the Tokyo Region. A comprehensive cost evaluation is attached in **Appendix A**.

5.0 CONCLUSION AND LESSONS LEARNT

In this project, we developed a cloud based photo sharing application which enable users to build a social network by following each other, and share posts among themselves. The application was built using cloud services provided by AWS. Through the project we were able to demonstrate a use case of building a cloud native application that can leverage out of box services provided by cloud vendors.

We gained an understanding how cloud native application can fully leverage the cloud resources, making application easily scalable and more available. First-hand experience of building a serverless architecture was also grasped. For creation of the application various services such as AppSync, Cognito, Dynamo DB and many other services were explored and incorporated. Our learning included CI/CD, AWS Services, Lambdas and serverless architecture and using machine learning through cloud. We also learnt a how to budget a cloud application and do an in depth cost analysis.

APPENDIX A

Usage Assumptions	
Number of days in a Month	30
Number of Posts per user a day	3
Number of posts per user a Month	90
Total number of Posts per Month	9000000
Montly Active Users (MAUs)	100000
Activity Per User Per Month	1500
Total Activity Per Month	150000000
(activities include add post, browsing, like, unlike, follow, unfollow, search, signin, signout)	
Image Size (GB)	0.001

AppSync $4 per million ops, $0.09 per GB, $2 per million updates (assume all activity are data modifications)	**Costs ($)**
Data Modification Charges	600
Data Transfer Charges	13.50
Real-time update Charges	300
Connectivity Charges	nil
Total for AppSync:	**913.50**

AWS Cognito	**Costs ($)**
First 50,000 MAUs	0
Next 50,000 MAUs	0.0055
Next 9,0,000 MAUs	0
Next 9,000,000 MAUs	0
> 10,000,000 MAUs	0
Total for Cognito:	**0.0055**

Standard S3 Storage	**Costs ($)**
Total Size of Images in GB/Month	9000
First 50 TB/Month	12.5
Next 450 TB/Month	0
Over 500 TB/Month	0
Requests Pricing	3.6
Data Transfer In	1079.88
Data Transfer Out	810
Total for S3:	**1905.98**

DynamoDB 1 read capacity unit (RCU) for 4KB, $0.47 per WCU enough for 2.5 Million Writes Per Month 1 write capacity unit (WCU) for 1KB, $0.09 per RCU enough for 5.2 Million Writes Per Month $0.25 per GB per hour (1 month = approximately 720 hours)	**Costs ($)**
Provisioned Throughput (Write)	29.73
Provisioned Throughput (Read)	6.5
Indexed Data Storage (FriendsTable) 60 Bytes per entry	4.32
Indexed Data Storage (LikePostTable) 80 Bytes per entry	5.76
Indexed Data Storage (PostTable) 250 Bytes per entry	18
Indexed Data Storage (UserTable) 230 Bytes per entry	16.56
Total for DynamoDB:	**80.87**

Elastic Compute Cloud (EC2) (On demand) Assume without free tier 21.38$ per t2.small instance 10.69$ per t2.micro instance 171.0$ per t2.xlarge instance	**Costs ($)**
Staging (t2.micro) 1 instance	10.69
Production (t2.small) 3 instances	64.14
Production (t2.small) 4 instances	85.52
Production (t2.small) 5 instances	106.9
Production (t2.small) 6 instances	128.28
Production (t2.small) 7 instances	149.66
Production (t2.small) 8 instances	171.04
Production (t2.small) 9 instances	192.42
Production (t2.small) 10 instances	213.8
Jenkins (t2.extralarge) 1 instance	171
Total for EC2:	**395.49**

AWS Lambda (Free Tier)	**Costs ($)**
First 1 million Requests	0
$0.20 per million requests after	1.6
400,000 GB-seconds per month (3.2 M seconds of compute time)	0
Compute Time Thereafter	0
Total for Lambda:	**1.6**

AWS Lambda Rekognition First 1 million Images processed per month at $1.3 Next 9 million Images processed per month at $1 Next 90 million images processed per month at $0.8 Over 100 million images processed per month at $0.5	**Costs ($)**

1 M images	1.30
8 M images	1
N.A.	0
N.A.	0
Total for Rekogntion:	**2.30**

Grand Total (USD$):	**3299.7455**