Cloud Photo Repository

CmpE 272-Team 23-Project Report

Poojitha Eragamreddy, Syed Imran Ahmed
Computer Engineering Department,
SJSU
San Jose, CA, USA
Poojithareddy93@gmail.com,
syedimran.ahmed@sjsu.edu

Shruthi Kulai, Niveditha Bhandary
Computer Engineering Department,
SJSU
San Jose, CA, USA
shruthi.kulai@sjsu.edu,
niveditha.bhandary@sjsu.edu

***Abstract*— In cloud Storage, data is stored on remote servers and can be accessed by the end user based on the need. Cloud service providers are responsible for managing data and making it available to the users. With the advent of technology, people are increasingly using multiple devices (like PCs, Laptops, Mobile Phones, Tablets etc.) to capture memories through device camera. These numerous images are continuously stored on different devices and it is difficult for end user to keep track of the photos present in the devices. The requirement is to utilize cloud storage to store all the images clicked from different devices in a sorted manner. Our project involves having a single web interface to sync information about images to the cloud server.**

Keywords— Cloud Storage, Web interface, Cloud Server, Cloud Service Provider, Technology, Multiple Devices, Photos.

I. INTRODUCTION

Cloud Computing Services host Storage feature making it easy for the end users to store their respective data in devices on the Cloud without wasting the storage on the particular device. The cloud Application server is owned by cloud service provider and he takes up the responsibility of maintaining physical environment. The Cloud Service providers are responsible for making data accessible and present whenever the user requires it.

Now a days with the advent of Technology, People who are enthusiastic in taking photos, click them from various devices like Camera, Phone, Laptop, IPad, tablets etc. After few days of click, they forget which photo is in which device. There are service providers like Google and Apple who provide their own cloud storage services Google Photos and iCloud respectively, but there is no central repository to search or know about which photo is in which device and also the sorting options are limited.

Through Cloud Photo Repository people can directly click their photo from the device and upload them to the cloud. It will keep track of all our photos which were taken based on the device and will provide us with the list of the photos on any device by accessing it through the web application.

In this report, we will explain the working feature of our project and the technologies we used to develop it.

II. ARCHITECTURE

Our Cloud Photo Repository is deployed on Amazon Elastic Compute Cloud (EC2) Instance which provides us with the infrastructure to host our servers and configure the virtual machine as per our wish. Our application is a three layered architecture based system:

Fig. 1: Photo Cloud Storage from Various Devices

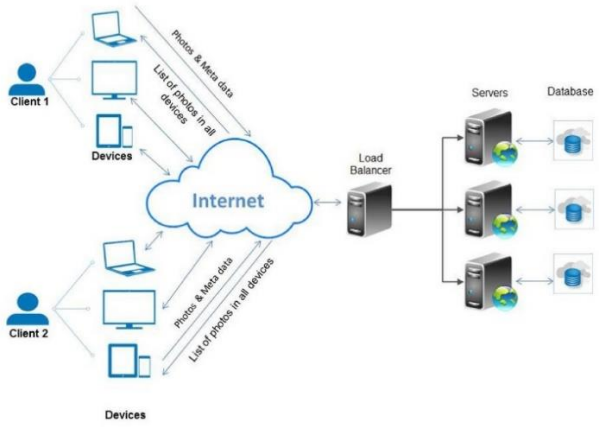

Fig. 2: Architecture Design

A. First Layer

It consists of the users and the devices that they use to access the application on EC2. The devices can be of any type which has the internet connectivity like mobile phones, tablets, Computer etc. The functionality of it will be to take the photos and upload it to the central Cloud Photo Repository using the web based application that we have developed.

B. Second Layer

The second layer will consist of the application itself which is deployed on the cloud infrastructure which in our case is Amazon EC2 Linux based instance. This application will accept the connections from the web and process the images and then send it to the database at the backend. It will give us the features of the web based UI developed in accordance with the dynamic features that work well with any kind of devices.

C. Third Layer

The third layer consists of the databases, which we have used as Mongo database (dB), on another Amazon EC2 instances. Here we have used the database scaling technique to elastically auto scale the database when the traffic on our website is high and reduce the database instance when our website sees low traffic. This elastic auto scaling feature has been provided by Amazon web services. We have put our database behind the load-balancer which almost guarantees that the network traffic will be distributed among the instances under the load-balancer. We have used the auto scaling of 5-10 for the heavy traffic. This improves the efficiency and reliability of the availability of our application in the case of partition or network breakdown of the machines.

III. IMPLEMENTATION

A. Frontend

For the frontend part or the UI part of our application we are using Embedded Java Script (EJS) which is a client side template language along with the bootstrap which is used for Hyper Text Markup language (HTML), Cascading Style Sheets(CSS) and Java Script (JS) framework for developing the responsive website, which can easily accommodate to the mobile layout. Along with these we are also using the Angular JS which is for binding the data with the HTML.

To use the EJS the JavaScript code can be written between the <% %> which is executed. In order for the HTML to add the results JavaScript code is written between <%= %>.

In order to include the bootstrap to our application and use its features we need to include the Content Delivery Network (CDN) support for it which can be found at its website. It gives us with the 'bootstrap.min.css' which handles all the attributes of responsive website.

Similarly, for the Angular JS we need to include the 'angular.min.js' which will give us the angular way of scripting our HTML page. Angular is extended by HTML using the ng-directives. The 'ng-app' directive defines that the particular application is an angular JS based application. The 'ng-model' directive binds the HTML controls like input, select, text-area to the HTML view.

B. Backend

In the backend we are using the Node Java Script (Node JS) framework, which is again based on the JavaScript. Along with that we are using the mongo dB as our database to store the images and the user's data.

In the node JS part we are using the express as our server, which is minimal and flexible for our Node JS application. There are various node JS modules that we have used with our application in accordance to cater to our needs.

The "async" module is used in our application to make the asynchronous behavior of Node JS as synchronous for certain part of the application.

The "bcrypt-nodejs" is used to give us the salt and the hash code generator for the password when the user signs up.

The "body-parser" is used to parse the Java Script Object Notation (JSON) string and give us the pretty looking output.

The "ejs" is used to EJS templates as the ejs engine.

The 'mongoose' is used to connect to the database which in our case is mongo dB, it gives us the module wrapper over the database, which we can use to have queries over the database.

The 'multer'[2] is the core module for our application; it is used to upload the images from the form based HTML page. It has Application program Interface (API)'s which tells us when the images are uploaded and when the task is finished. It allows us to open a stream for uploading of the images. We have configured the multer[2] to accept the images in the Joint Photographic Group (JPG) and the Portable Network Group (PNG) format currently, along with the limit on the number of images that can be uploaded at a time which we have set it to 5. The user can upload more images in chunks of 5 images per upload. There is also a limit on the size of each image being uploaded which currently we have set it to 5 MB.

The 'Passport' module is used for login and signup, it gives us the flexibility to implement the api's for the user signup, it also gives us the features to sign-up using Facebook, LinkedIn, twitter etc. But for now we have created our own sign-up and login, which we'll be storing in the database that uses the passport module.

C. Database

The database that we have used here for our purpose is the mongo dB, it is a Non-Relational (NoSQL) document based database, which is a Consistency and Partition Tolerance (CP)

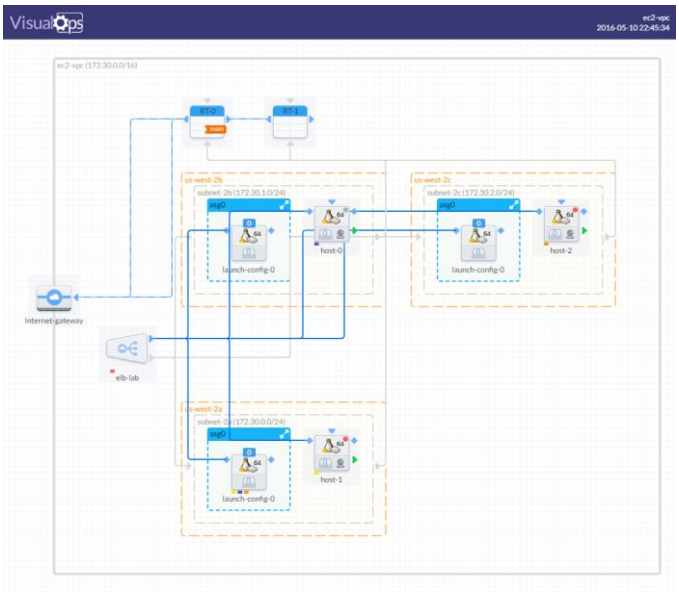

Fig. 3: Database architecture behind the EC2

system, that is it is more towards the consistency when there is a partition in the network. The reason to choose this database is that our system should be consistent every time and it should not give us the stale data at any stage of our application process. For the sake of the availability of our system we have implemented the replica sets which are in the different zones so that if one node goes down for any reason, others should function properly. Mongo dB provides the replication option that can be configured using the mongo config file.

The above architecture depicts the real time structure behind the EC2 Load Balancer that we have used. The virtual machines in the EC2 instances are placed in different Availability Zones (AZ), under the Virtual Private Cloud (VPC) which have three different subnets and the only traffic allowed to these machines is through the internet gateway which has specific ports open for the incoming traffic, specifically the *Port: 27017* which is required by the mongo dB database. Our database is behind this gateway which in turn provides a level of security by liming the traffic on specific port.

IV. Definitions

In this section we go over the flow of our application and more in depth working of for the Cloud Photo Repository. Our implementation consists of Index Page, User Sign-up, User Log-in, User Home Page, User Photos Page, User Profile Page.

A. *Index Page*

This is our landing page which shows up whenever the client or the user hits our link to the website. This page contains nothing but the Login and Signup buttons which takes user to another page according to the option that he chooses.

B. *SignUp Page*

Once the user is through the home page, he can come to the signup page which will ask for the email address and the password that the user wants to keep. This page assures that the user signs up with a unique E-mail id and returns proper error messages if the E-mail id already exists in our system. This assures that there are no two users with the same unique mail id.

The password field for the user to sign up also has the validations, which restricts the user to keep a very simple or a single letter password. We have used the regular expression to check if the password matches certain criteria's or not. The expression is:

$$"^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])(?!.*\s).*\$"$$

There is another field for the confirm password which reassures that the user has typed the same password as earlier and he remembers it correctly.

Once this information is verified the user can click on the Signup button which will register the user to our system and will land him to his home page.

The schema that we have used for the registration of the user is:

```
var userSchema = mongoose.Schema({
   local          : {
    email      : String,
    password : String,
    devices    :  [{ type: String,index:true }]
   }
});
```

As we can see that from the above schema we have used mongoose module for Node JS. The fields required are email, password and devices. The devices field is an array of devices that the user wants to register with our system. This is not a mandatory field and the user can register his device after signing up from his home page.

C. *Login Page*

The user or the client uses this page to login to the website. Here the user has to mandatorily fill in two fields namely Email Id and Password. There is a client side validation on these fields using the form validation features provided by HTML5. The values entered by the user are validated with his/her credentials stored in the Mongo DB.

Once the user successfully logs in, he lands on the user home page. If the login fails, the user is shown the same page with an error message on top of the screen.

D. UserHome Page

This page is shown to the user once he successfully logins in to his account. Here, the user has options to register his devices, and upload photos from his device.

Register your device field: This field lets the user register his/her devices with the application. These devices are used to uniquely identify the photos uploaded. The field is a text field, however the user can only enter letters and numbers here. This check is done using the expression

[a-zA-Z0-9]

Once the devices are registered, a message is displayed to the user "Your device is registered successfully". The devices which are registered for the user are then populated in the drop down menu. The user has to select one of the devices from this dropdown to indicate that this is the device from where he will be uploading the picture. Once the user selects his device, the 'Select files to upload' button will be enabled. Using the HTML5 input field of type 'file'. On click of the button, user is provided a window dialog to browse his images.

The schema that is used for uploading of the images is:

```
var ImageDataSchema = mongoose.Schema({
   url: {type: String, trim: true},
   thumb: {type: String, trim: true},
   bin: {type: String, trim: true},
   contentType: {type: String, trim: true}
});
```

The above Schema has the url of the image, the thumbnail of the image, the image in the binary form itself and the content type which is either JPG or PNG.

In mobiles, this also provides a feature which opens the camera to click and upload photos instantaneously. Either single or multiple images may be selected. Once the photos are uploaded to the server, a message is shown to the user indicating successful upload. The uploaded images are displayed as thumbnails on this page.

E. Photos Page

This page provides the user with an interface to view all his photos in a single place. On load of this page, the user is shown a list of all his devices which he registered. Each link to the device displays all the pictures which were uploaded from that particular device. The display shows a thumbnail of the collection of images. Clicking this version of the image opens the actual image in a different tab in the browser.

The Schema that we have used to build this is the nested schema model which uses the schema of image. It is:

```
var DeviceSchema = mongoose.Schema({
   email: {type: String,trim: true},
   name: {
      type: String,
      trim: true
   },
   kind: {
      type: String,
      enum: ['thumbnail', 'detail']
   },
   url: {type: String, trim: true},
   createdAt: {type: Date, required: true, default: Date.now()},
   imgs: [ImageDataSchema]
});
```

The above schema has the email-id which is unique to the user, through which we can identify for which user this device is registered. Then it has the name field which tells the name of the registered device, also the kind of image it has either the thumbnail or the detailed image. Then it has the images array, which contains the images schema, so this device can have multiple images inside it, so to list down those images we have this schema.

F. Profile Page

This page is a profile page which provides account information for the user. The details shown here are the unique Id generated for each user, along with his email address.

Along with these features, the user is allowed to log out from the web application, at any time once he is logged in. Only an authenticated user is allowed to view the UserHome, Photos and Profile Pages. Each of these pages has easy navigation to the other pages. This is implemented as a navigation bar on the top right portion of the screen. On mobile devices, the navigation bar is initially hidden from the user. The user can view the buttons by clicking on the navigation menu icon on the top right of the screen.

V. RESULTS

Our project involves having a single web interface to sync images clicked in different devices. Searching favorite photo has become easier with our feature to sort images based on device name. We have also included a feature "click and upload" through which the user can click a photo using his mobile device and upload it to the cloud directly. The uploaded images are compressed for storage optimization. The images under each device category is sorted based on date. The UI of our application is user friendly. Navigation buttons and bars are easy to understand and use, also they are consistent throughout the website. We have made our application responsive to provide the best experience to the users. We have used tools like Valgrind to detect memory leaks and threading bugs.

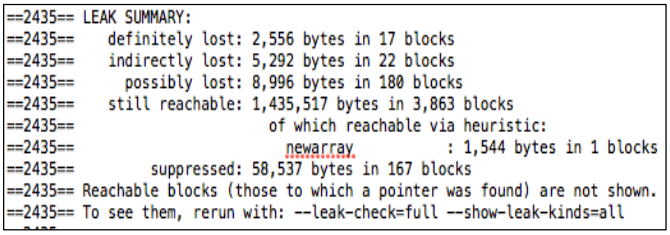

Fig. 4: Valgrind Memory leak summary

VI. FUTURE ROADMAP

A. *Sort photos based on Image Recognition*

We can add ability to assign names to faces and organize the pictures based on the person in the photos. We can also sort photos based on different things in photo, such as “bridges”, “mountains”, “dogs” etc.

B. *Photo sharing Feature*

Using this feature, user can share his photos with other users. Basic photo sharing feature would include allowing user to email photos by dragging and dropping photos into a pre-defined template.

C. *Photo Editing Feature*

Using this feature, user can edit his favorite photos within the web app and save the changes.

D. *Sorting based on time and location*

Using this feature the user can sort his images based on the time and location of the photos.

VII. CONCLUSION

With the dawn of digital camera and ever increasing capacity of storage, the world of photography is entering the era of abundance. Anyone can shoot as many pictures as they want to and on several devices. But for most amateur photographers, sorting and organizing photos becomes an issue. We try to solve this issue with our Cloud Photo Repository project.

Our application can be used to upload photos from different devices to Amazon EC2 server and sort uploaded photos based on device name. Our system also includes compression technique for storage optimization. Navigation through different pages of the web app is explained in the Definitions section. Future roadmap is discussed which includes future enhancements to our application.

VIII. ACKNOWLEDGMENT

This Project is made possible through the help and support from everyone. We would like to thank Prof. Rakesh Ranjan for his support, encouragement and advice. Secondly, we would like to thank our group members for their contribution and support. Finally, we express our gratitude towards our families and friends for their kind co-operation and encouragement which helped us in completion of this project.

IX. REFERENCES

[1] Reference for IEEE Template-IEEE Advancing Technology http://www.ieee.org/conferences_events/conferences/publishing/templates.html

[2] Multer library for multipart image upload: https://github.com/expressjs/multer

Demo link:
http://ec2-52-33-199-228.us-west-2.compute.amazonaws.com:5000/

GitHub link:
https://github.com/Imran-syed/272-Cloud-Photo-Repository