**CSE565 – Cloud Computing Project**

**Mobile Security Solution on Cloud**

Team No - 9

(Mentor : Vishrut Mehta)

*Deepak Jiwal*

*Dilwar Hossain*

*Jitender Yasaswi*

*Ganesh*

*1. Problem:*

To design a cloud framework that allows mobile phones to post images on a minute basis and also retrieve the images to the end user with minimum delay.

*2. Proposed framework:*

Figure 1 – Architecture Diagram

*3. AWS Services Used:*

*3.1. Relational Database service (RDS):*

RDS is a web service that makes it easy to set up, operate, and scale a relational database in the cloud.

We use RDS for storing following information.

*3.1.1 Device Info*

For every device (User), we maintain a table named User in the database which has the following fields:

ID, NAME, PASSWORD, CLICK DELAY IN SECONDS, UPLOAD INTERVAL IN SECONDS

CLICK DELAY IN SECONDS is an adjustable attribute which defines the time interval between two successive clicks by the camera.

UPLOAD INTERVAL is also an adjustable attribute which defines a session of time so that images captured in that session are to be uploaded altogether at the end of that session.

A sample tuple is given below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **NAME** | **PASSWORD** | **CLICK DELAY IN SECONDS** | **UPLOAD INTERVAL IN SECONDS** |
| himalaya | Himalaya | 1234 | 60 | 1800 |

*3.1.2 Session Info*

For every session we maintain a table named Session in the database which has the following fields:

ID, USERID, START TIME, END TIME and LOCATION.

A sample tuple is given below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **USER ID** | **START TIME** | **END TIME** | **LOCATION** |
| d07d7163-ebf6-404c-b3e7-643e3c003813 | Himalaya | 2014-09-23 08:00:00 | 2014-09-23 08:29:59 | East-IIITH |

*3.1.3 Configuration*

|  |  |
| --- | --- |
| Engine | MySQL 5.6.19a |
| Storage | 5GB |
| Availability zone | US West (Oregon) |
| Configuration | 1 VCPU, 1GB RAM |

*3.2 Simple Storage Service (S3):*

S3 is a file storage web service, used to store the images captured from the device. All the images taken in a session are stored in a folder named with the session identifier. Every session for a device reside in a directory named with the device identifier.

*3.2.1 Storage Hierarchy*

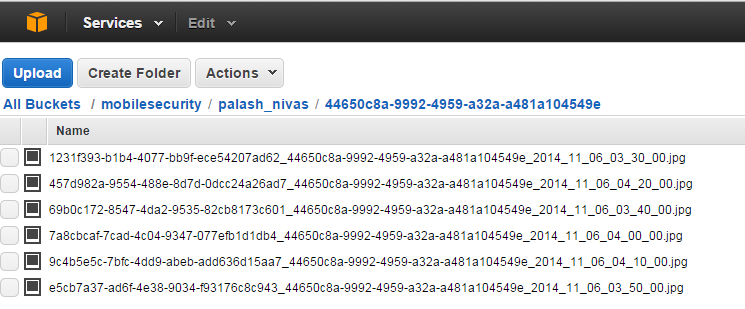


Figure 2 – S3 storage for a session.

As you can see, the directory structure starts with the bucket name (‘mobilesecurity’), followed by the device name (‘palash\_nivas’), and followed by session identifier (‘44650c8a-9992-4959-a32a-a481a104549e’).

File name follows the pattern: IMAGE-ID\_SESSION-ID\_SNAPED-AT-TIMESTAMP.jpg

This is helpful in optimizing the database storage, as we may not need to store one entry for each image. Since the scale of image upload is very high for our application, this optimization is very effective.

*3.3 Elastic compute cloud (EC2) & Elastic Block Storage (EBS):*

EC2 provides virtual servers on the cloud. An instance is created, attached to EBS, which provides memory for the server. We installed Apache Tomcat on the machine, which is HTTP server to host the back end service and web page.

*3.3.1 Configuration*

|  |  |
| --- | --- |
| Public DNS | ec2-54-201-117-236.us-west-2.compute.amazonaws.com |
| Availability zone | US West (Oregon) |
| EBS size | 8GB |
| Amazon Machine Image | Amazon Linux AMI (HVM) 3.14 Kernel |

*3.4 Amazon Cloud Front*

Amazon CloudFront is a content delivery network that speeds up distribution of static and dynamic web content, for example, .html, .css, .php, and image files, to end users. We use it only to power our images so that they will be served faster i.e the distribution is created for the complete S3 bucket (‘mobilesecurity’) which host all the images.

*3.4.1 Configuration*

|  |  |
| --- | --- |
| Image URL before using CloudFront | https://**s3-us-west-2.amazonaws.com**/mobilesecurity/admin/ |
| Image URL after using CloudFront | https://**d3iq5ru0cfmp88.cloudfront.net**/mobilesecurity/admin/... |

*4. Project Components*

*4.1 Android Application (Secure.apk)*

Secure is the android application that captures and uploads photos periodically to cloud. This app. is developed completely in Java, using Android SDK.

The application automatically starts capturing pictures, and uploading them periodically (defined by uploadIntervalInSeconds in the device table).

If the uploadIntervalInSeconds is 30 minutes and the clickDelayInSeconds is 1 minute, it clicks pictures every minute, and starts uploading every 30 minute, by creating a session (/post/createSessionInfo) and then it pushes the image (/post/imageInfo). For this example, there are 31 (1+30) Rest calls are made for a complete transfer of images taken in a session.

*4.2 Web Page (SecureUI.war)*

SecureUI is the front end web application, used by security administrator to view images stored in cloud, in gallery form. It's completely built in Java using Google Web Toolkit(GWT), which is a compiler that converts Java code to efficient Javascript code. This service is deployed in tomcat running in the EC2 instance.

The web page allows one to:

* Login to a specific device. (/post/loginInfo)
* Login as a security admin user. (/post/loginInfo)
* Signup for a new device. (/post/signUpInfo)
* Search images by specifying filters such as start date of session, end date of session, device name and device location.

*4.3 Rest Back End Service (RestService.war)*

RestService is a REST based back end service, developed completely in Java. It uses Jersey library to parse and generate the JSON request and responses respectively. This service accepts request from both android application and web page.

*4.3.1 Restful Endpoints*

*4.3.1.1. Device login:*

Request-

POST 'post/loginInfo'

Content-Type: application/json

{

"userid":"library",

"password":"abxc3xx"

}

Response-

{

"name": "library",

"userid": "library",

"clickDelayInSeconds": "20",

"uploadIntervalInSeconds": "20",

"password": "678",

"message": "Login Success.",

"code": "200"

}

*4.3.1.2. Sign up for new device:*

Request-

POST 'post/signUpInfo'

Content-Type: application/json

{

"name":"nivas",

"id":"admin",

"clickDelayInSeconds":"30",

"uploadIntervalInSeconds":"50",

"password": "6781"

}

Response-

{

"message": "Signup success.",

"code": "200"

}

*4.3.1.3. Create a session (done by uploader before initiating the actual upload):*

Request-

POST 'post/createSessionInfo'

Content-Type: application/json

{

"userId":"admin",

"startTime":"2014-12-12 00:00:00"

}

Response-

{

"sessionId": "7e65ec99-c952-4768-a88c-b551e4ed6837",

"message": "Session created successfully.",

"code": "200"

}

*4.3.1.4 Upload the image (done by uploader after creating a session):*

Request-

POST 'post/createSessionInfo'

Content-Type: application/json

{

"userId":"admin",

"sessionId":"7e65ec99-c952-4768-a88c-b551e4ed6837",

"snapedAt":"2014-12-12 00:00:00",

"location":"centre-iiith",

"data":"base64 rep of data"

}

Response-

{

"message": "Image Upload success.",

"code": "200"

}

*5. Problems faced:*

* **Connecting to EC2, RDS instances.** (Firewall issues) – We were unable to connect to the EC2 and RDS nodes as those ports were in blocked state. We solved by using a open network (airtel data card).
* **Defining security groups for services over ports**. – We defined inbound and outbound rules for each resource in the cloud.
* **Remote connection to EC2.** - We solved by using tools such as Putty (to remotely execute commands on a EC2 instance) and WinSCP (to transfer the service and web app. wars)
* **Database storage optimization.** – We improved the database storage by storing session information, and using it to fetch all image specific details from S3. Thus we successfully avoided having one entry for each image in the database.
* **Parsing JSON Payload** – We solved by using Jersey which seamlessly support exposing our data in a variety of media types, hiding the low-level details of the client-server communication.