AWS S3

Prepared by Sriganesh Pera

S3 means Simple Storage Service

Amazon S3 stores data as objects within buckets. An object consists of a file and optionally any metadata that describes that file.

S3 is not database like RD or DynamoDB which functions on Block -Level storage

You can retrieve data at any point of time. It gives any developer access to the same highly scalable, reliable, fast, inexpensive data storage infrastructure that Amazon uses to run its own global network of web sites.

Advantages to Amazon S3

Create Buckets – Create and name a bucket that stores data. Bucket names must be unique.

Store data in Buckets – Each object can hold up to 5 TB. Each object is stored and retrieved using a unique developer-assigned key. You can upload as many objects as possible.

Download data – Download your data or enable others to do so.

Permissions – Grant or deny access to others who want to upload or download data into your Amazon S3 bucket.

Standard interfaces – Use standards-based REST and SOAP interfaces designed to work with any Internet-development toolkit.

S3 Concepts:

1.Buckets: A bucket is a container for objects stored in Amazon S3. For example, if the object named photos/puppy.jpg is stored in the johnsmith bucket, then it is addressable using the URL
http://johnsmith.s3.amazonaws.com/photos/puppy.jpg

You can configure a bucket when you add an object to it as Amazon S3 generates a unique version ID to the object.When an object is successfully uploaded it will generate http 200 status code.

2.Objects: Objects are elements stored in a bucket which consists of both data and metadata

3. keys: A key is the unique identifier for an object within a bucket.Every object in a bucket has only one key.

For ex in the URL http://doc.s3.amazonaws.com/2006-03-01/AmazonS3.wsdl

doc = bucket

2006-03-01/AmazonS3.wsdl = key

4.Regions: Bucket stored in a region will never leave the region unless you transfer the bucket to the desired region.

5.Amazon S3 Data Consistency Model:

This implementation of the **PUT** operation adds an object to a bucket. You must have WRITE permissions on a bucket to add an object to it.

This implementation of the **GET** operation retrieves objects from Amazon S3. To use GET, you must have READ access to the object.

Amazon S3 provides **read-after-write consistency** for PUTS of new objects in S3 bucket and **eventual consistency** for overwrite PUTS and DELETES in all regions.

What is Read-After-Write Consistency?

With read-after-write consistency, a newly created object or file or table row will immediately be visible, without any delays.

No Stale Reads Possible. Read after Write Consistency is Atomic.

What is eventual consistency?

If you make any change to an object systems will take time to make sure that data is visible to all its users. This is opposite to read-after-write consistency.

So if you updated/deleted an object in S3 you can see it immediately (read-after-write consistency), but if you update your object it takes little time(eventual consistency) to see the affected change.

Reading Stale data possible in eventual consistency. But guarantees lower read latency and lower read throughput.

S3 Objects consists of following:

1.Key :The name of the object

2.Value :This is simply data made up of sequence of bytes

3.Version ID :Versioning

4.Metadata :Data about Data

5.Subresources

Access Control Lists : Granular permissions for individual files in Buckets and also applicable for buckets as well.

S3 Performance:

Amazon scale requests at a very high rate.

Bursts in the number of requests per second:

- >300 PUT/LIST/DELETE
- >800 GET

You need to contact Amazon to prepare and avoid temporary limits.

If consistenly get high number of requests per second

- >100 PUT/GET/DELETE
- >300 GET

Follow best practice guidelines to avoid overwhelming the I/O capacity of a partition

S3 Buckets:

1.An AWS account can own up to 100 S3 Buckets

2. No limit on number of objects stored in bucket

3.Bucket Name Restrictions:

1. Must be unique

2.Should comply with DNS naming conventions

3.Shouldn't be an IP address format(192.68.1.)

4.Can only contain lower case letters,numbers,periods and hyphens,but must not start with period/hyphen

S3 Objects:

Object size :

Minimum of 0 bytes

Maximum of 5 TB

Objects larger than 5 GB requires Multipart Upload API:

1.Can upload independently, in any order, and in parallel

2.If any part fails to upload, you can retransmit that part

3.You can pause and upload

4.You can upload objects as they are being created.

5.Objects are reassembled after calling MultiPartUpload API

Objects can be encrypted before being saved to disk, and decrypted when downloaded.

LAB 1: Working with S3 Buckets:

1.Go to AWS Console and click on S3

2.Click on Create Bucket:

I'm using console's older interface if you have problems with new interface/older interface you can switch the way you like it.

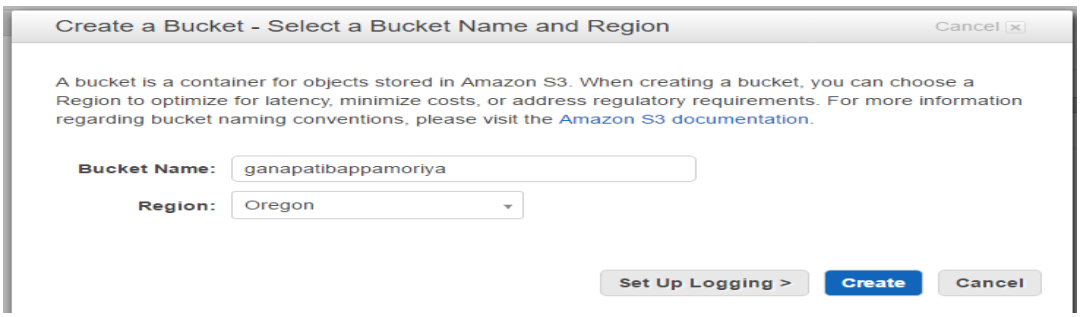
Now click on create Bucket and select your favored region and give a unique name as this name can be addressable as URL(DNS) for static web hosting so unique name is necessary.

The rules for DNS-compliant bucket names are:

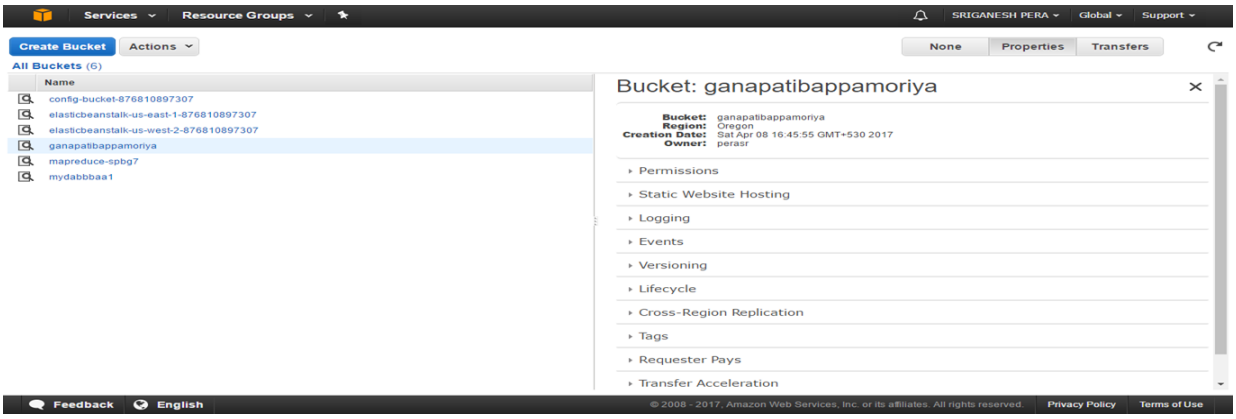
1.Bucket names must be at least 3 and no more than 63 characters long.

2.Bucket names can contain lowercase letters, numbers, periods, and/or hyphens. Each label must start and end with a lowercase letter or a number.

3. Bucket names must not be formatted as an IP address (e.g., 192.168.1.1)



By clicking on properties tab you can see the info of your empty bucket

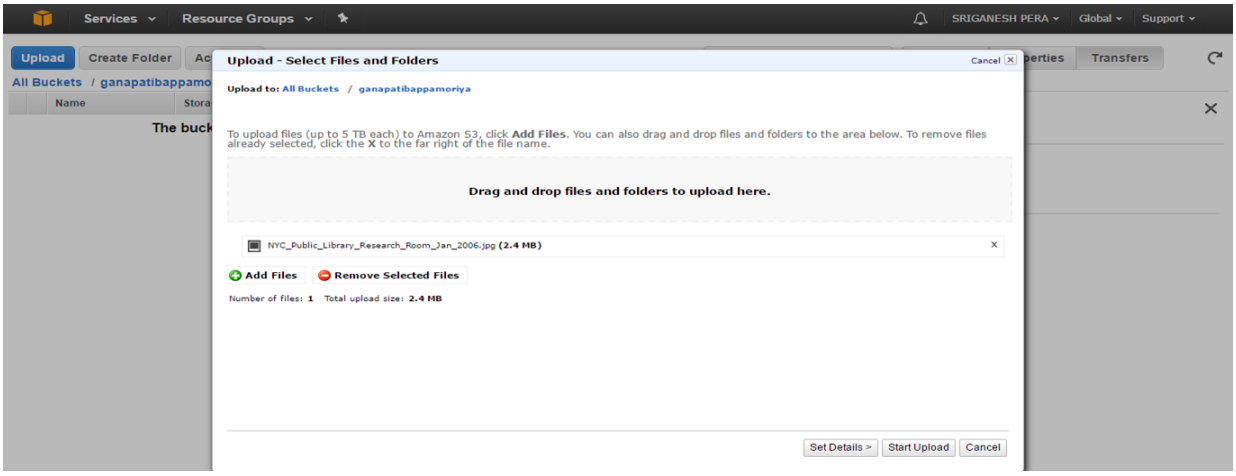


3.Create a folder inside your bucket

This folder need not be an unique name. I've created and named it as myfolder.

4.Upload object into your bucket:

Upload any file/image into your bucket.I've uploaded an image file into my newly created bucket.I'm uploading an image file for this lab.



5.Set the S3 file the we uploaded as publicly accessible:

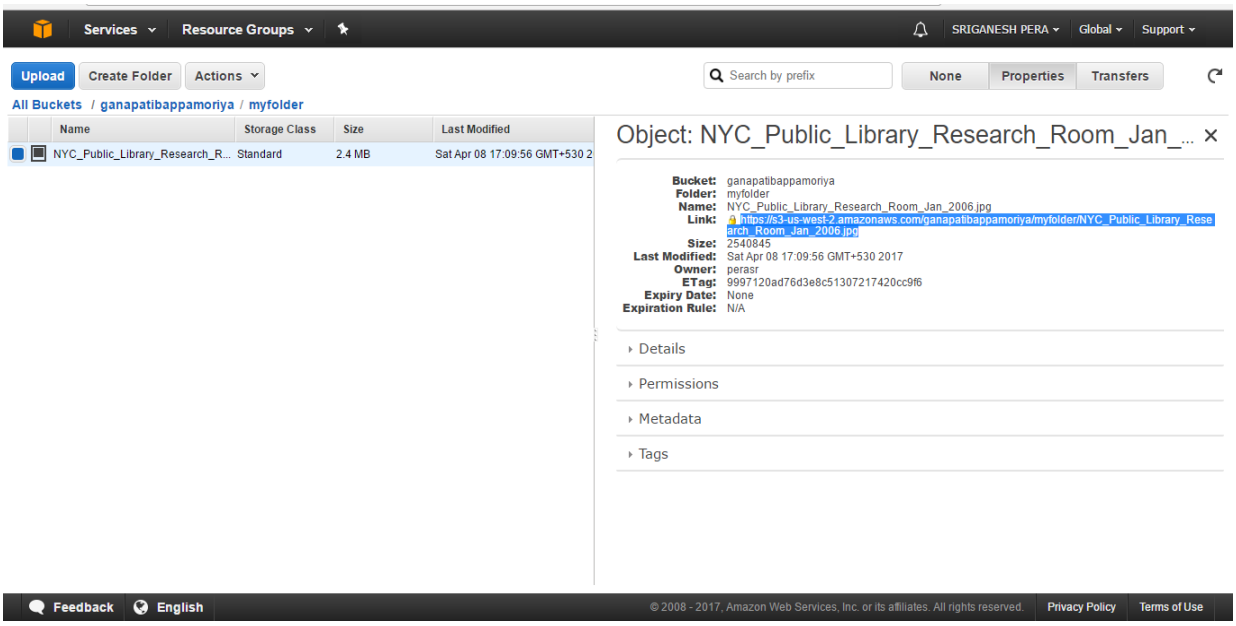
All uploaded files are private by default and they can only be viewed or edited by you. In order to illustrate this point, complete the following steps:

1.Select the newly file.

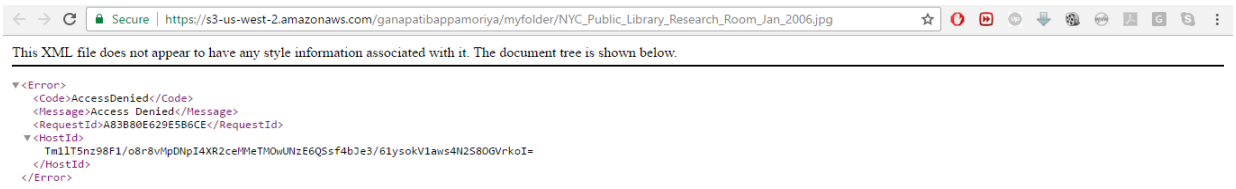
2.Switch to the Properties tab (in the upper right portion of the console).

3. In the Link field, right-click and copy the URL shown.

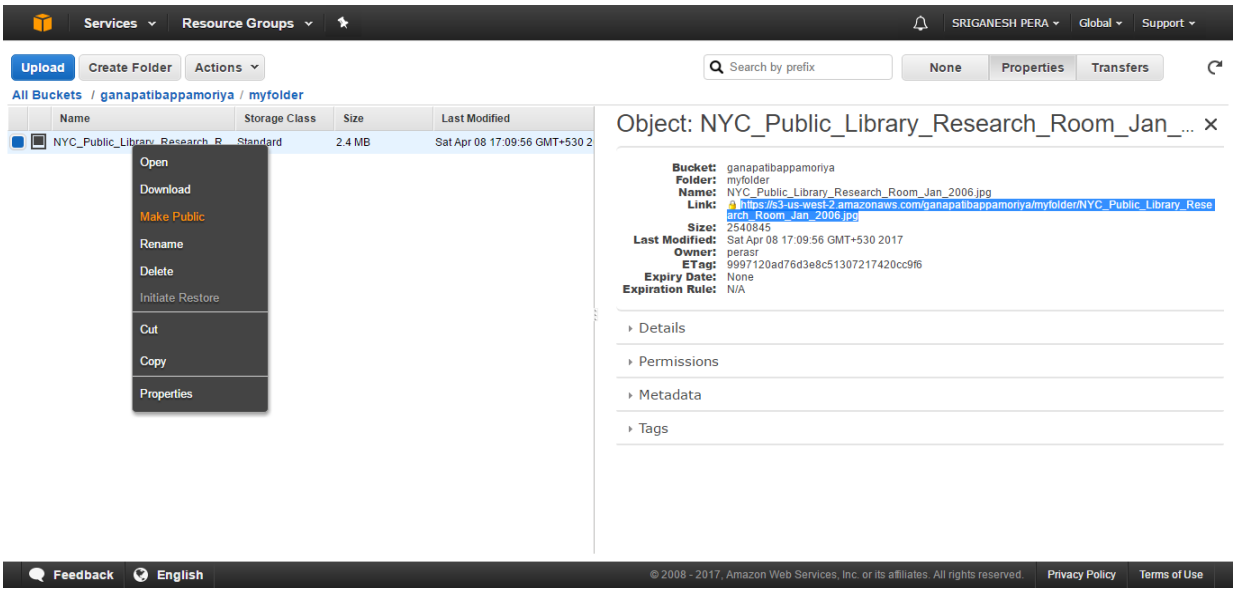
4. Paste the URL into the address field of a new browser tab. You should see “Access Denied” rather than the image



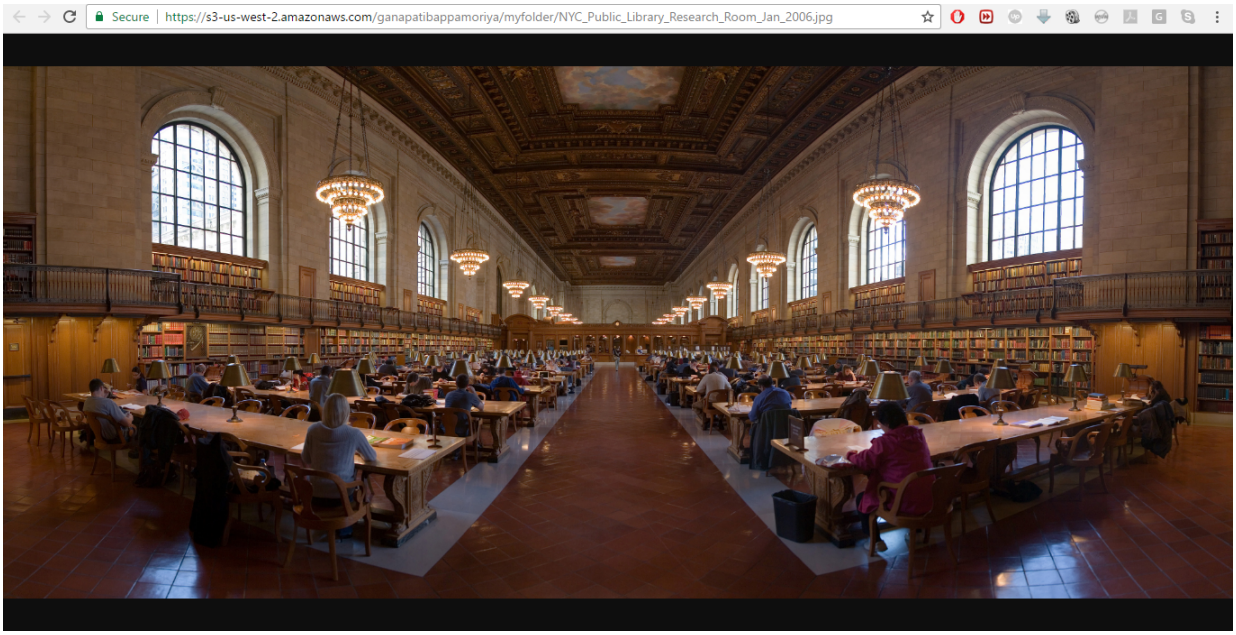
If you click on the link you will get this page “Access denied”



Now right click on the file you have just uplaoded and click on ‘Make public’



Now you click on properties agains and click the URL you will see your image in the browser.



6. Change metadata of an S3 object:

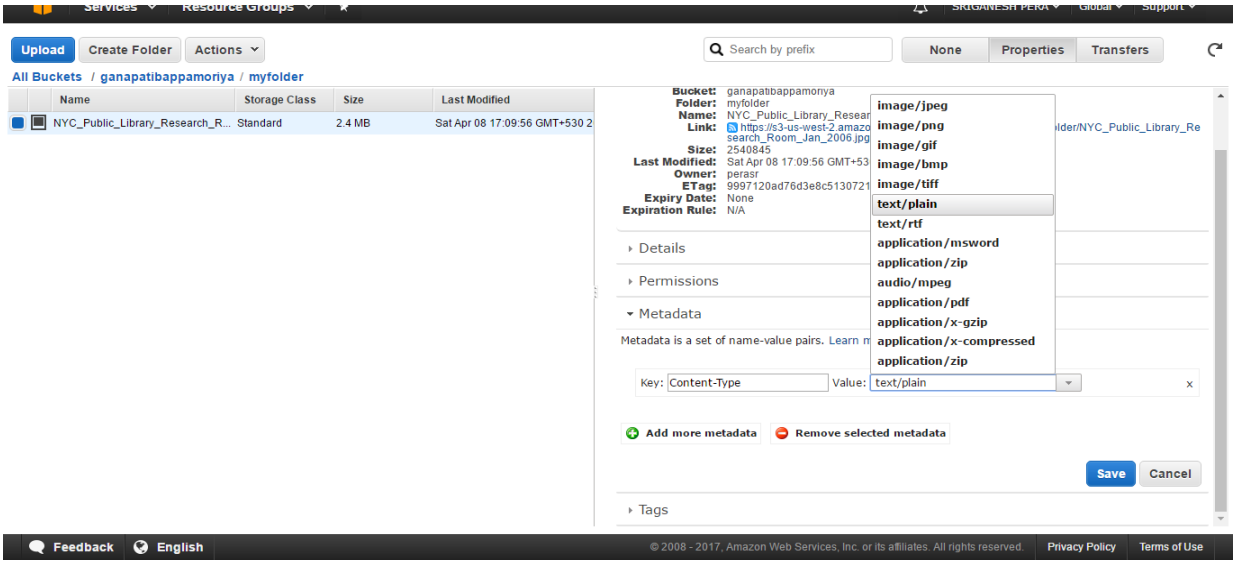
Each object in Amazon S3 has a set of key/value pairs representing its metadata. There are two types of metadata: "System metadata" (e.g. Content-Type and Content-Length) and custom "User metadata". User metadata is stored with the object and returned with it. As an example, you might have your own taxonomy for various images, such as “logo”, “screenshot”, “diagram”, "flowchart" and so on.

Let's change the Content-Type of our image to "text/plain".

1.Click on the image object, select the Properties pane and then expand Metadata.

2. Select text/plain as the new Content-Type value.

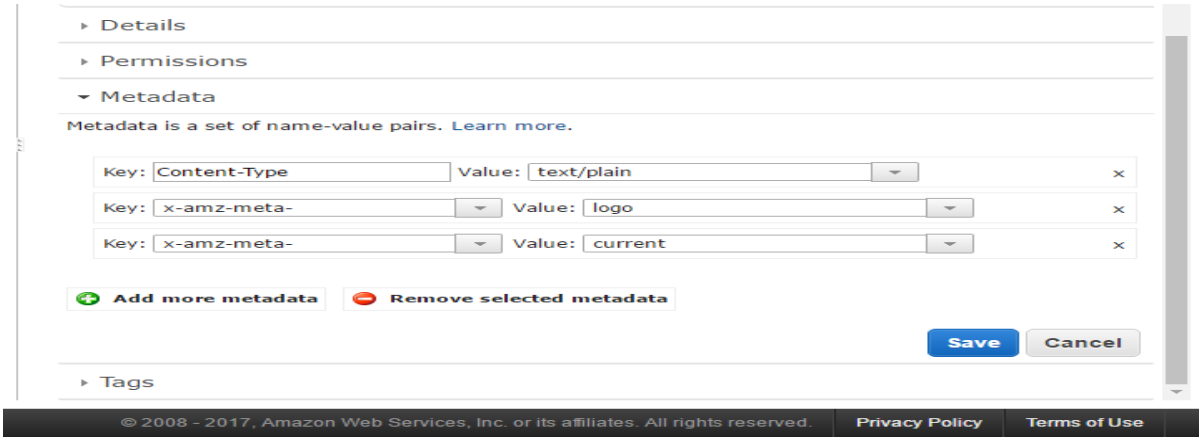
3. Click on Save.



Of course, changing the Content-Type of a PNG image to text/plain is not very useful, but the point is to see how simple it is to change system or user metadata associated with objects in S3. Now let's add custom user metadata. User metadata must be prefaced with "x-amz-meta-". Follow these steps to add a custom user type for imagetype and imagestatus:

1. Click the **Add more metada** icon.
2. For the **Key** value enter: *x-amz-meta-imagetype*
3. For the **Value** enter: *logo*
4. Click **Save**.

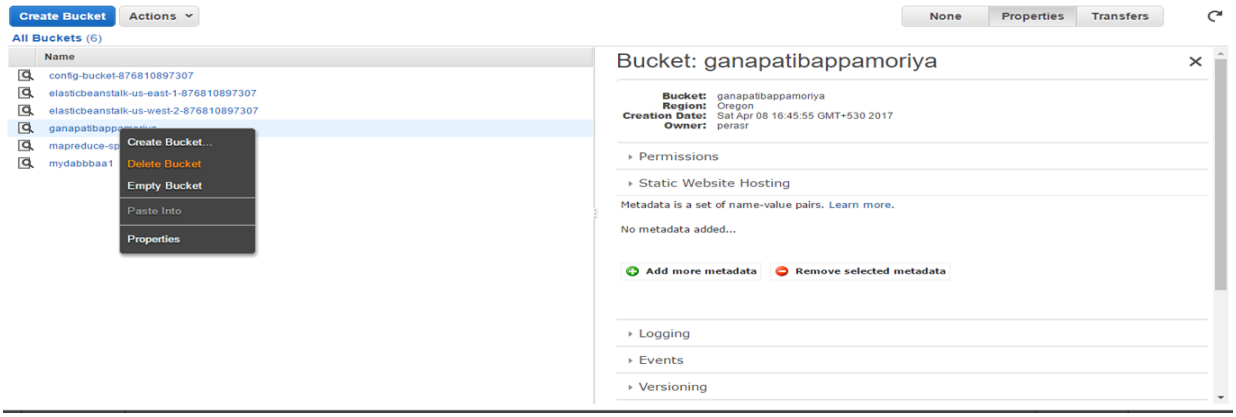
Repeat the process, but this time enter x-amz-meta-imagestatus and current for the Key/Value pair. Click Save and your logo Metadata should look likethe following:



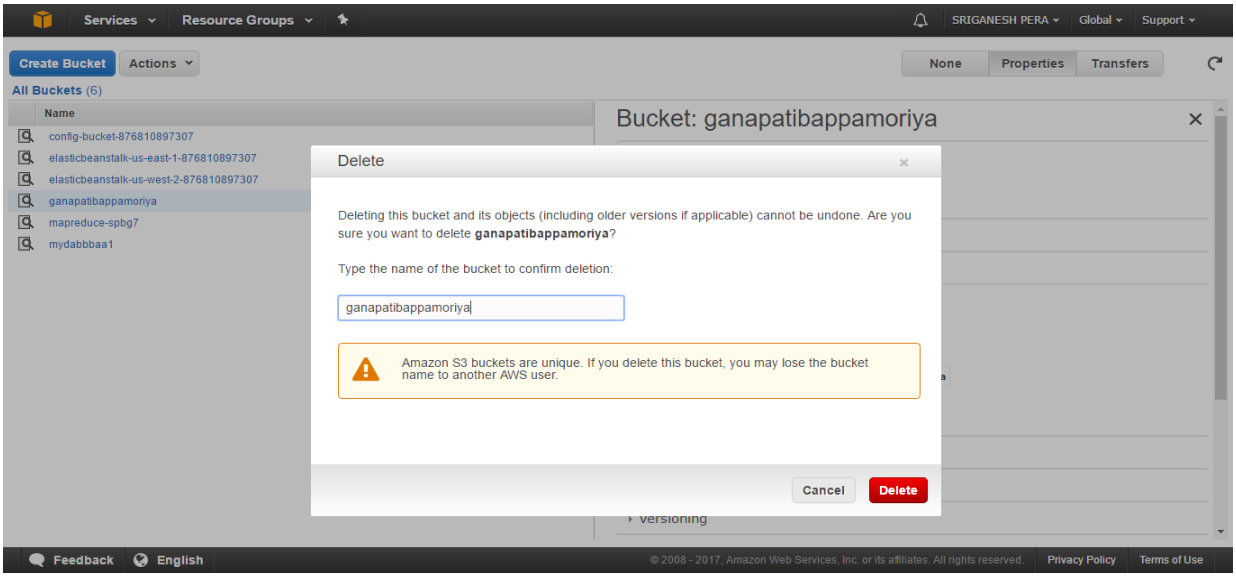
7.Delete the bucket:

When you delete the bucket the entire data gets deleted.

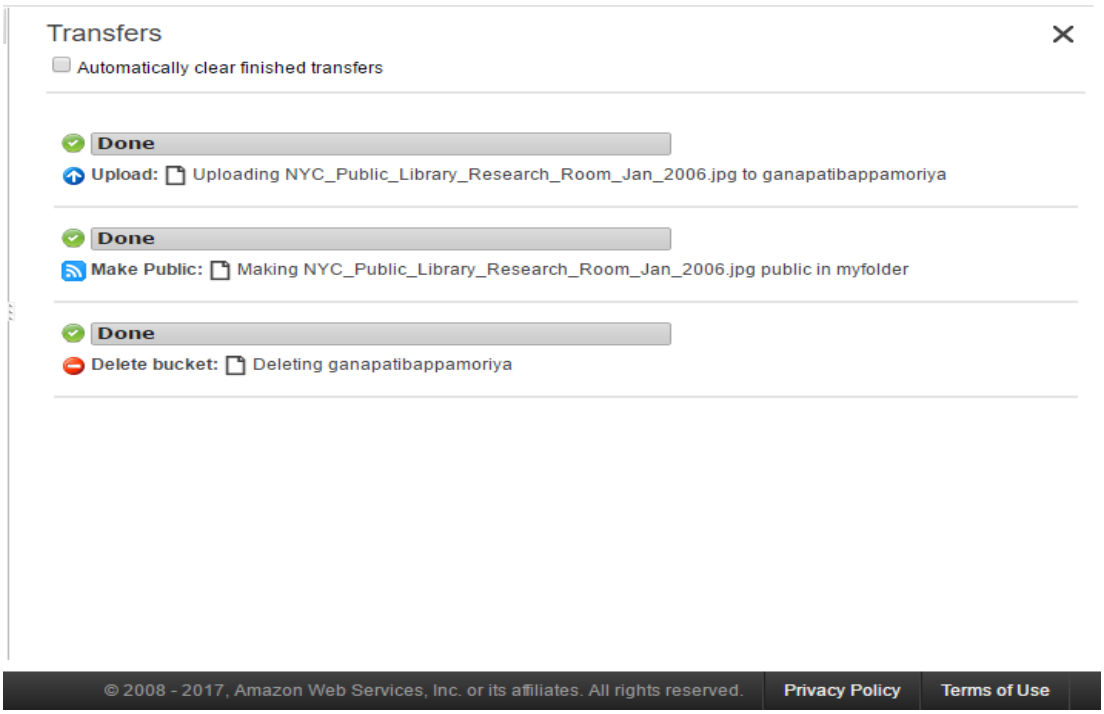
Right Click on the bucket name and click delete



AWS makes sure that you are deleting the right bucket so it asks you to specifcally type the name of the bucket that you want to delete.



As you can see my bucket is deleted now



DNS

Domain Name System resolves IP address to human friendly names.

It converts IP address to human readable format like from 192.168.32.1 to google.com which is a domain name.

TLD:

Top-Level Domain

A top-level domain, or TLD, is the most general part of the domain.

Common top-level domains are "com", "net", "org", "gov", "edu", and "io".

Host:

In WWW.example.com:

WWW = host

Example.com = domain

SubDomain:

A "subdomain" refers to any domain that is part of a larger domain.

In Ubuntu.com “Ubuntu “ is a sub-domain of .com

Or in www.tripadvisor.com/forums

“forums” is the SLD (Sub level domain) of the tripadvisor.com

Fully Qualified Domain Name:

Fully qualified domain Name(FQDN) is the complete domain name.

Ex: mail.missouri.edu.

In this FQDN:

mail = host

Missouri = domain and .edu = TLD and . = root server

A FQDN is an absolute name that specifies its location in relation to the absolute root of the domain name system. This means that it specifies each parent domain including the TLD

Another ex:

www.missouri.edu. is the FQDN for the University of Missouri on the web.

Mizzou.edu

Mail.google.com →

Name server:

A name server translates domain names into IP addresses. It's a part of DNS.

Google.com -> 192.168.32.1

Zone File

A zone file is a simple text file that contains the mappings between domain names and IP addresses. Zone files are present in DNS name servers.

.com

192.162.32.1 -> Google.com -> Ip

202.55.23.120 -> Apple.com -> ip

Facebook.com -> ip

Records:

A group of DNS records forms a zone file

Domain name resolver or DNS resolver:

Domain Name Resolvers are the computers which are used by ISPs to respond to a user request to resolve a domain name. "Resolving a domain name" refers to the translation of a domain name into an IP Address

How DNS works:

Step 1: user types google.com in his browser

Step 2: Your computer connects through your ISP.

Step3: Your ISP checks in its cache whether this IP is already stored or not. If it is stored, then your ISP connects you to the website and if it is not then your ISP domain name resolver forwards request to the **root server** '.'

What are these Root servers or Root name servers:

Root servers are at the top of DNS hierarchy. Every DNS request is first received by the root server and this root server delegates the **requestor (ISP DNS resolver)** to the TLD server.

Root servers themselves don't know where these domains are stored so they direct the requestor to TLD.

Every DNS name has a dot at the end before DNS preprocess a request

www.google.com.

This dot is the root server which sits on the top of DNS hierarchy.

Since the TLD for google.com is .com so the root name server directs the requestor to IP address of .com TLD server.

Step 4: The ISP's DNS resolver visits the .com nameserver, using the IP address it got from the root nameserver. It asks the .com nameserver, "Where can I find the nameserver for google.com?"

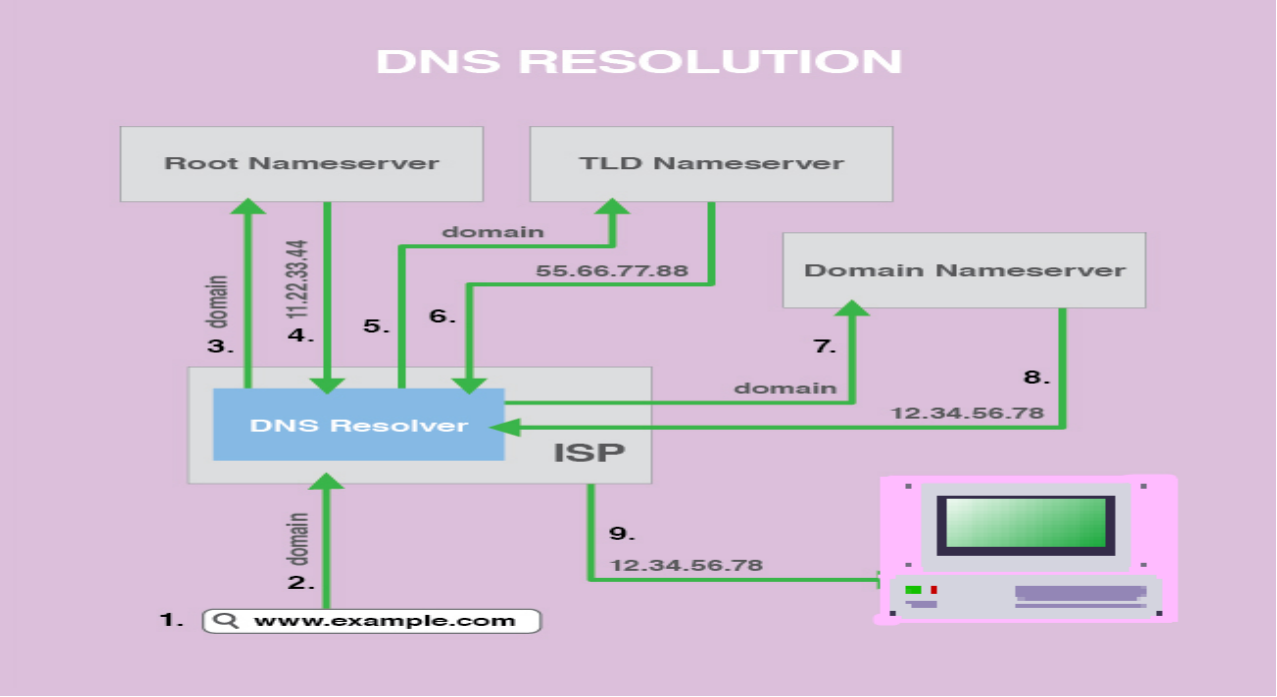
Step 5: The .com nameserver(TLD) responds with the IP address for the google.com nameserver.

Step 6: The ISP's DNS resolver visits your domain's nameserver and reads the zone file.

Step 7: The zone file shows which IP address goes with the domain google.com

Step 8: Now your ISP has the IP address of google.com. ISP stores this IP address in this its cache.

Step 9: It connects you to our PC



Records: In general it contains info about IP address

SOA Records

A start of authority (SOA) record is information stored in a domain name system (DNS) zone about that zone and about other DNS records.

SOA is a **mandatory record** in all zone files.

Format example:

```
domain.com.   IN SOA ns1.domain.com. admin.domain.com. (

                                              12083    ; serial number

                                                3h      ; refresh interval

                                                30m     ; retry interval

                                                3w      ; expiry period

                                                1h      ; negative TTL

)
```

domain.com = root of the zone; This specifies that the zone file is for the domain.com. domain

IN = internet

SOA = start of authority

Ns1.domain.com = primary master server

admin.domain.com.: This is the email address of the administrator for this zone. The "@" is replaced with a dot in the email address

12083: This is the serial number for the zone file. Every time you edit a zone file, you must increment this number for the zone file to propagate correctly

3h: This is the refresh interval for the zone.

30m: This is the retry interval for this zone.

3w: This is the expiry period. If a slave name server has not been able to contact the master for this amount of time, it no longer returns responses as an authoritative source for this zone.

1h: This is the amount of time that the name server will cache a name error if it cannot find the requested name in this file.

NS Records

This record type defines the name servers that are used for this zone.

AWS Route 53 creates 4 NS records. These records specify the name servers used for this zone.

'A' record:

An 'A' record stores IPV4 address for a DNS name. If a client requests for the IP address of a website the DNS server looks for its IP in 'A' record and sends back the IP address to the client.

AAAA record:

An AAAA record stores the IPV6 address unlike A record which stores IPV4 address.

Example format:

```
ns1.domain.com.      IN   A        111.222.111.222
```

ns1.domain.com is the domain name

IN = internet and A= record type and associated is its IP address

CNAME or Alias record:

CNAME stands for canonical name. It is an alias for other domain name.

The canonical name (CNAME) resource record creates an alias (synonymous name) for the specified FQDN.

For ex: for the FQDN mail.google.com(primary)

CNAME records are :

www.googlemail.com,

www.gmail.com,

www.google.com/mail

All these are CNAME records that point to IP address of mail.google.com

Mail exchange Record (MX)

Every domain has a mail server which tells where to receive an incoming email from a client.

A mail exchanger record (MX record) is a type of resource record in the Domain Name System that specifies a **mail server** responsible for accepting email messages on behalf of a recipient's domain, and a preference value used to prioritize mail delivery if multiple mail servers are available.

MX record looks like this:

```
    IN  MX  10   mail.domain.com.
```

IN = Internet

MX = Mail Exchange

10 = priority number. If there are multiple mail servers involved for a domain, the lowest numbered MX record gets the highest priority.

Mail.domain.com = mail server which receives your email

The MX record should generally point to a host defined by an A or AAAA record, and not one defined by a CNAME.

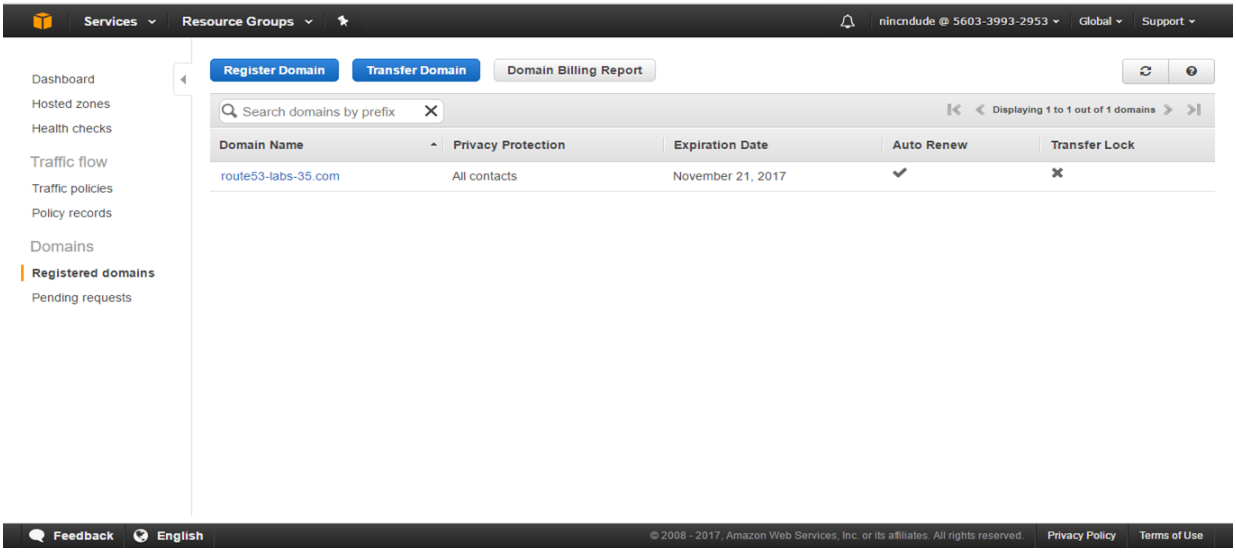
So, let's say that we have two mail servers. There would have to be records that look something like this:

```
        IN  MX  10  mail1.domain.com.
        IN  MX  50  mail2.domain.com.
mail1   IN  A       111.111.111.111
mail2   IN  A       222.222.222.222
```

Amazon Route 53 lab:

Step 1: Register a domain

Click on Route 53 in AWS console and then click on registered domain and register a domain. It may cost up anywhere between 11-50 depending upon the domain you want.



As you can see I already have a registered domain.

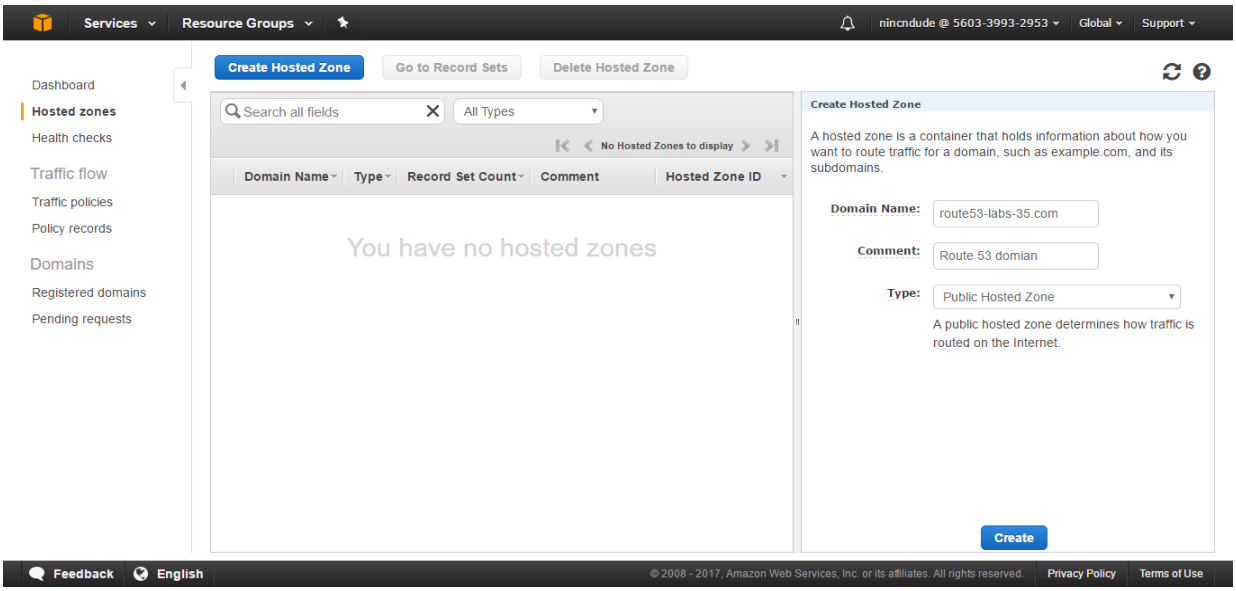
Step 2: Create a hosted zone

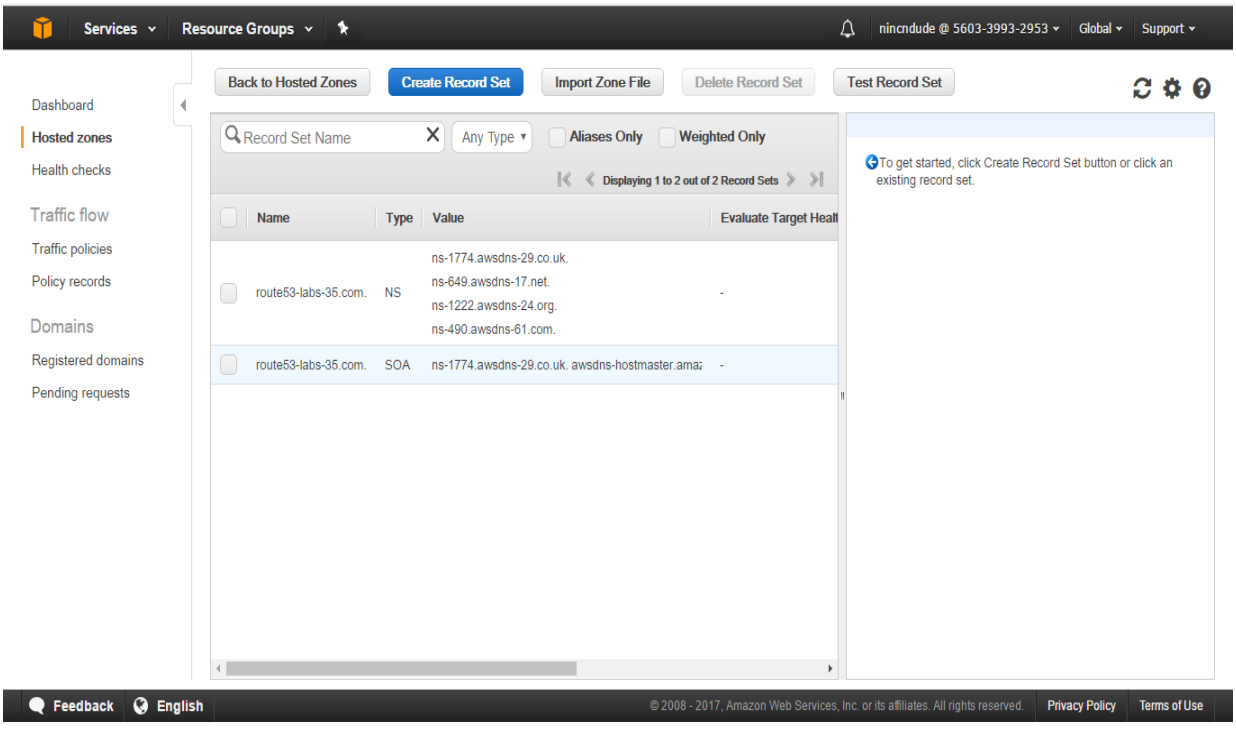
Click on hosted zones and click on create hosted zone

Then a side box appears asking you to type the name of the domain you want to create along with comments(optional) and type of domain (Public or Private)

So, the domain name should be the domain you just created and type whatever the comments you want to and choose the type as: **Public Hosted zone** as it should be accessible by anyone.

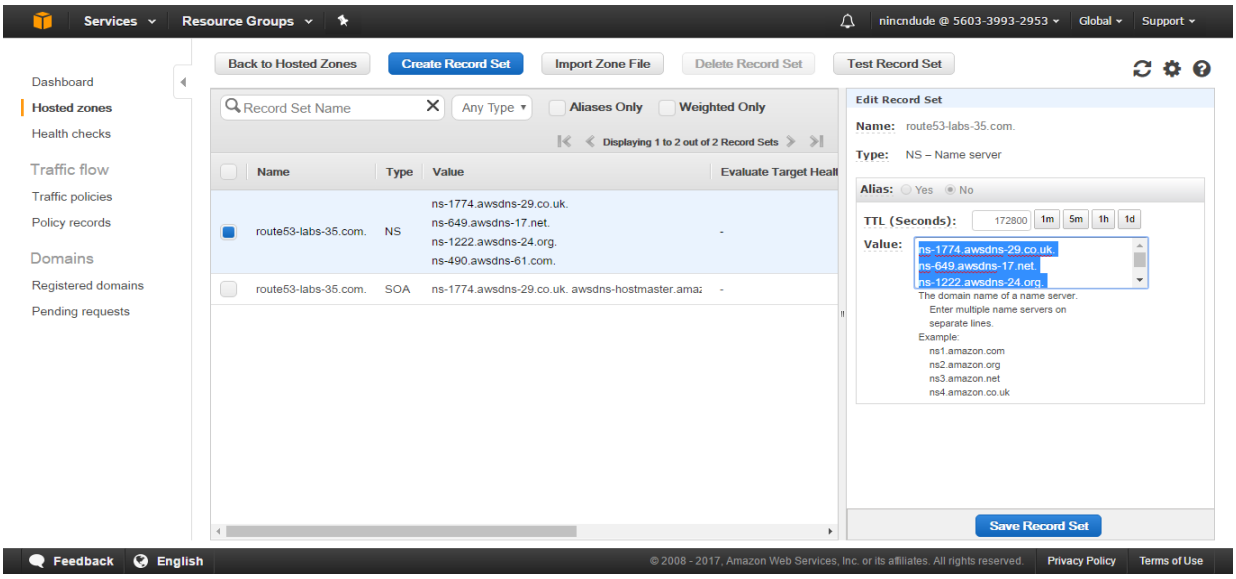
Then click on **Create**





As you notice two types of records are created SOA and NS records(which are mandatory)

Now click on ns record and you will notice a side box appears.Now copy the all the 4 NS servers in the value box and save it in a text file.

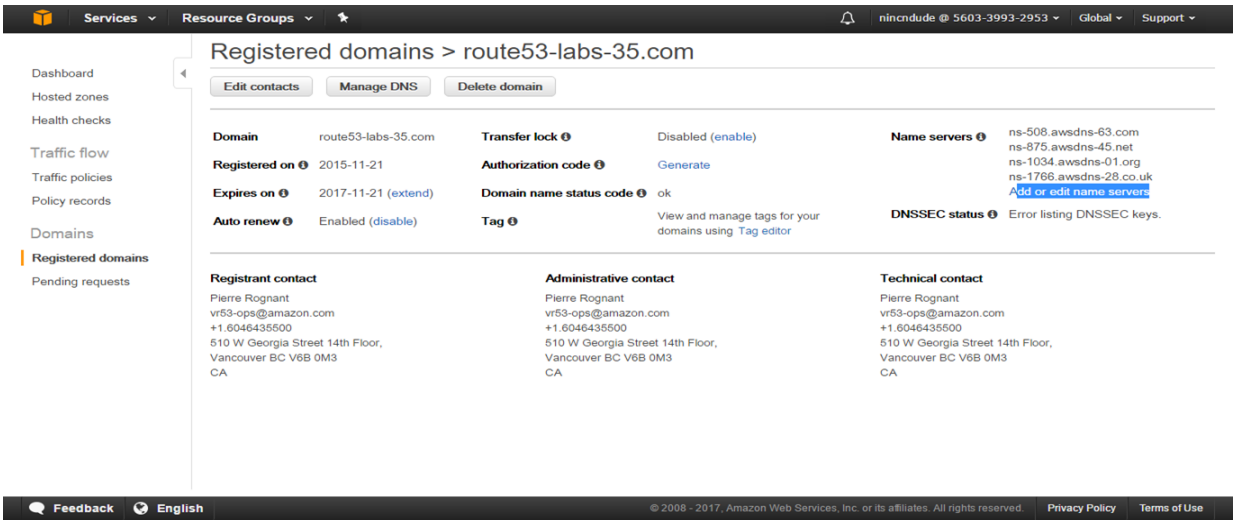


As you notice at the end of every NS server you will a see a '.'.Its called root server as discussed earlier and it doesn't matter whether you copy that or not because by default.

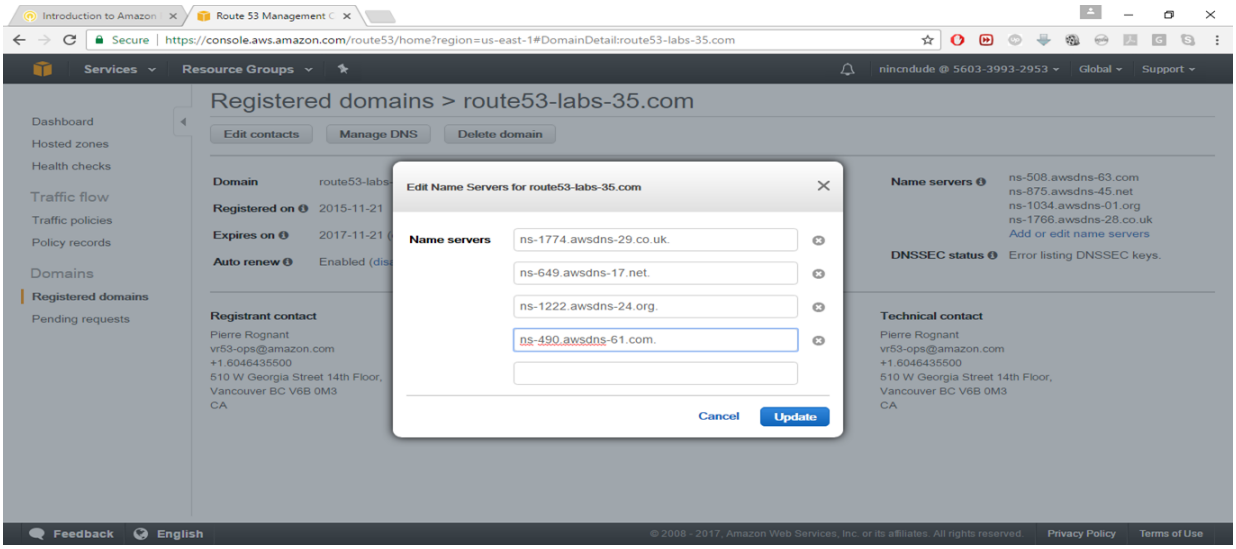
Step 3: Attaching your name servers to the registerd domain

Click on registerd domain and click on your domain and you will see the information about your domain.Now click on the **Add or edit name servers**

Remove all the defualt values and paste the valued you copied on your notepad



So ,your name servers should reflect the name servers you got when you created hosted zone



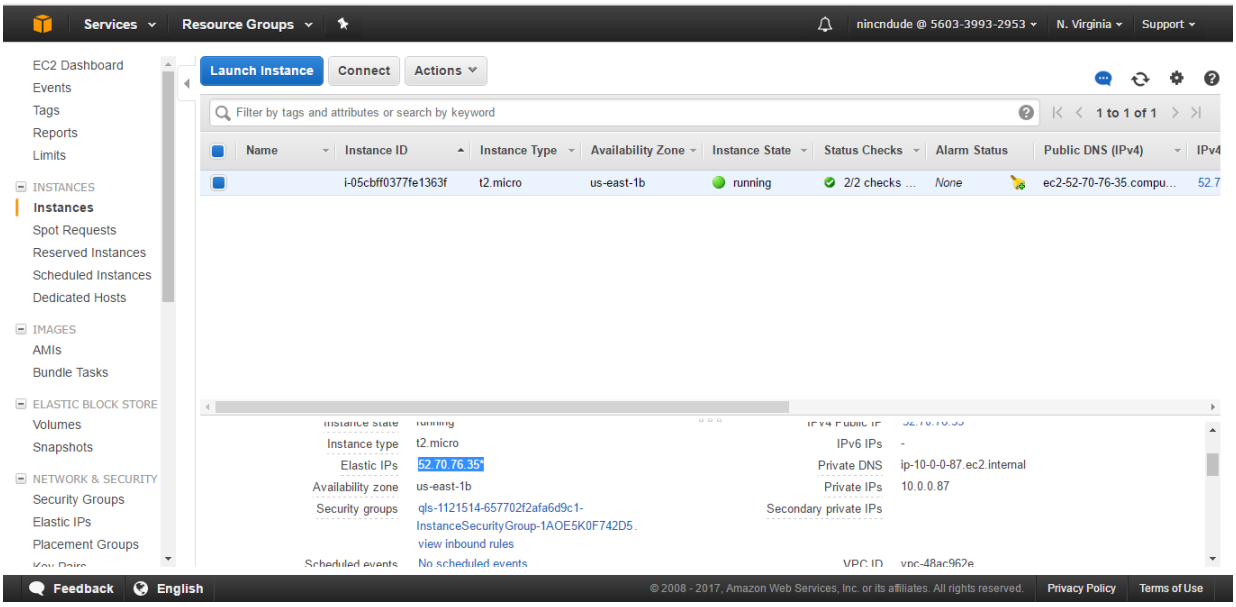
Now click update once you're fininshed.

So, what we have done so far is the we registered a domain and associated that domain with a hosted zone.

Now we will create an **EC2 instance with an Elastic IP** and create DNS records

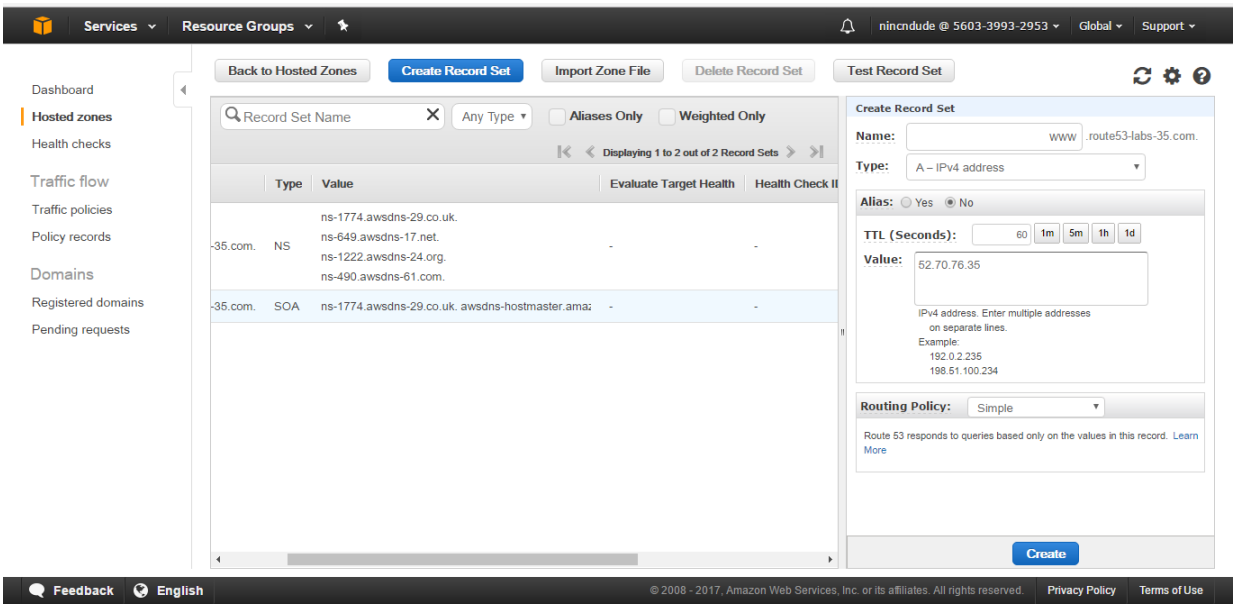
Step 4: Creating DNS records for EC2 instance

Launch an EC2 instance and associate and elastic IP to it.



As you can see I already have a launched instance with an Elastic ip associated with it. Now copy the elastic IP and save it in a note pad.

Now click on Route 53 from your browser and click on **hosted zone and** now click on your domain name and click on **create record set.**



Now a side box opens up and you fill in the details as shown:

Name :www(type)

Type: IPV4 adress(choose)

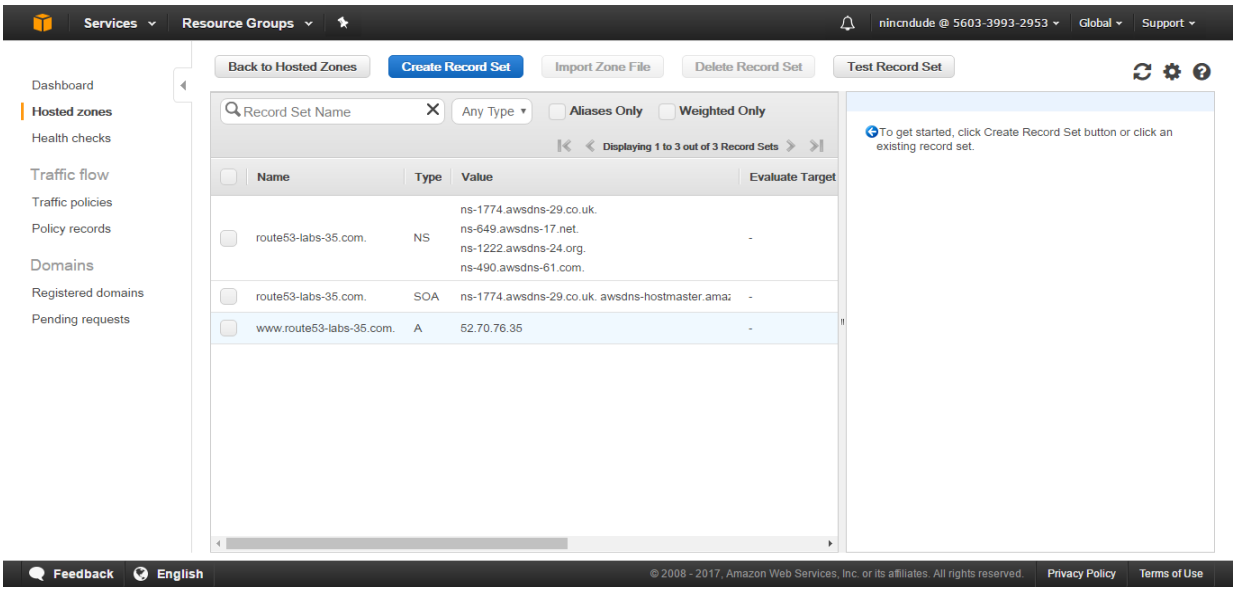
Alias: No(choose)

TTL :60(type)

Value : copy the elastic IP you have created

Routing policy: simple(choose)

Click on create



As you can see from the screen shot our domain is associated with an elastic Ip.

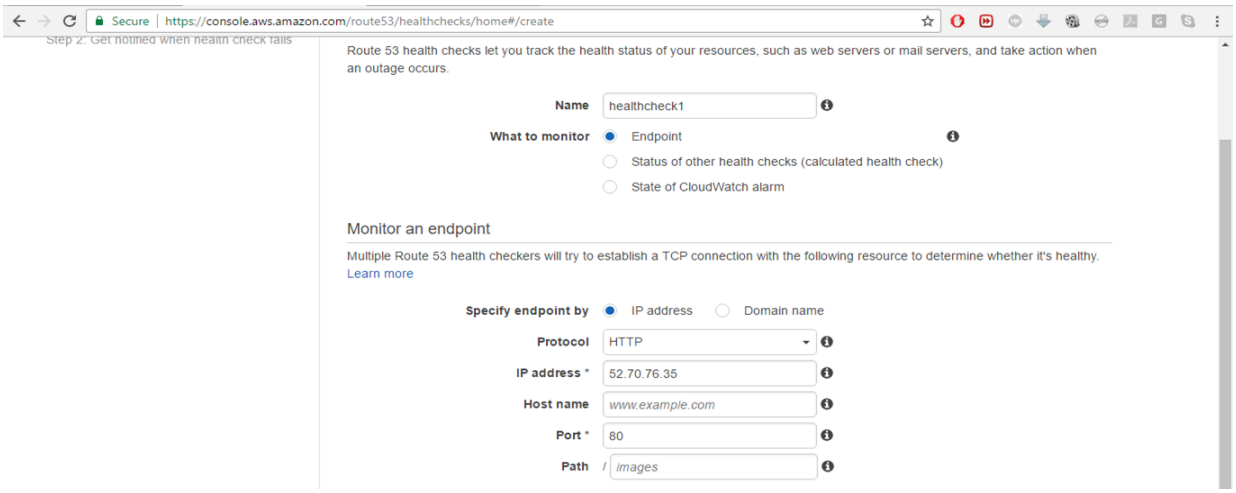
Now copy the domain and paste it in your browser you will see your domain works.I customized my domain like this “Hello! This is your EC2 web server.It’s nice to see you.”

So what we have done so far is that we associated our elastic IP to our domain.

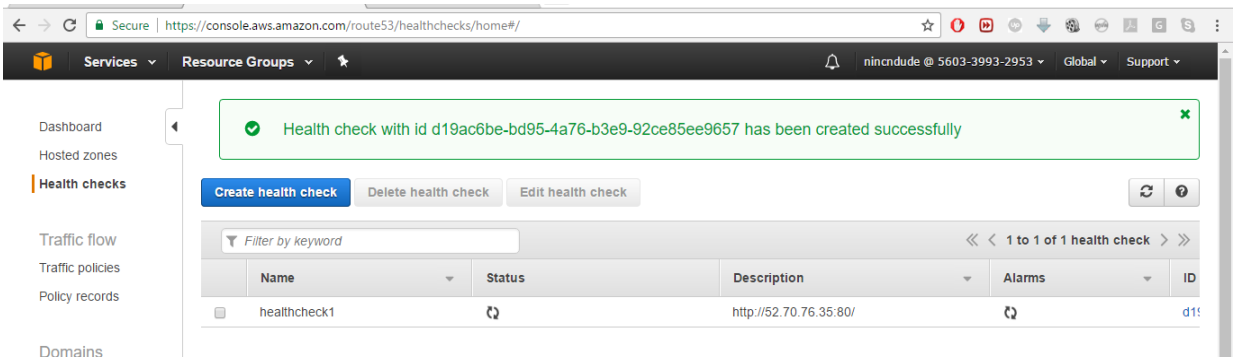
Step 5: Add a route 53 Health check to your Amazon EC2 server

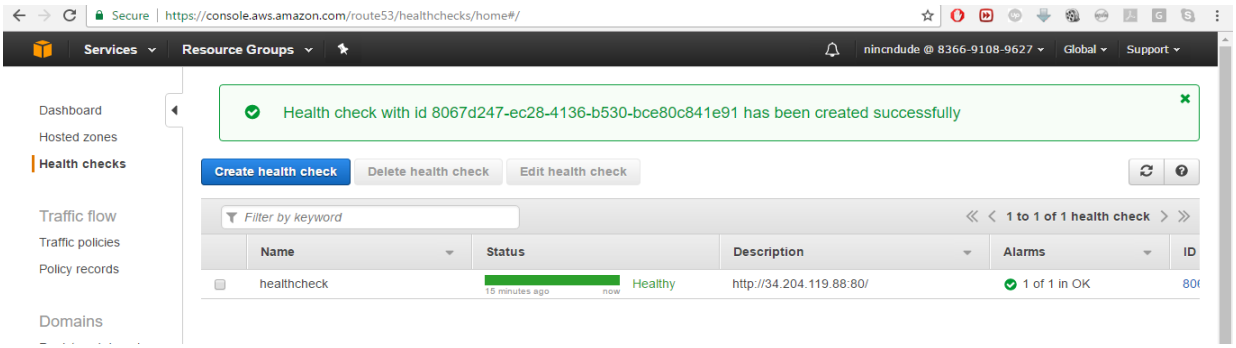
Click on Health checks on your Route 53 console and click on create Health check and in the Name box type healthcheck or your desired name

And set what to monitor is endpoint and chose Protocol :HTTP IP adress: and port :80 and click next



Click on create alarm and chose new SNS topic to send notification and write the topic name :Route53healthcheck and type your recipient email address and click create health check.First it will show as unknown then after a minute or 2 it shows healthy



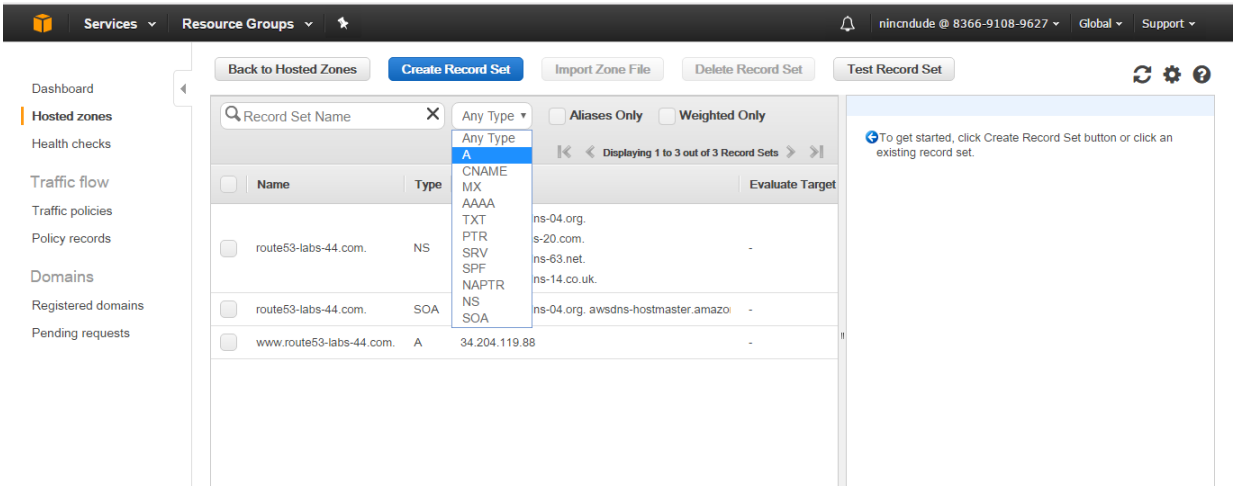


First it will show unknown and after a minute or two if you refresh the screen you will see ‘healthy’.

Step 6: Configuring a DNS failover to Amazon S3

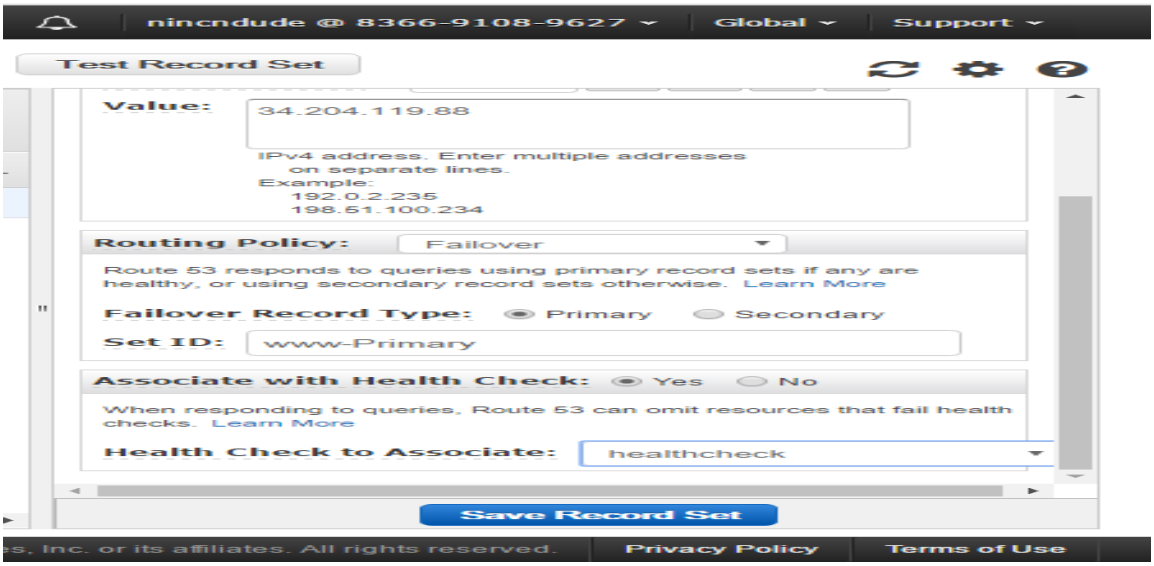
Now what we do we create an alias for our domain leveraging S3 features.In case if our main domain is down for some reasons the S3 powered site will act as back up

Go to hosted zones in Route 53 console and click on the domian name and in the type column click on the ‘A’ record.



Now click on the domain name and a side-box will pop-up .Now change the routing policy to fail over

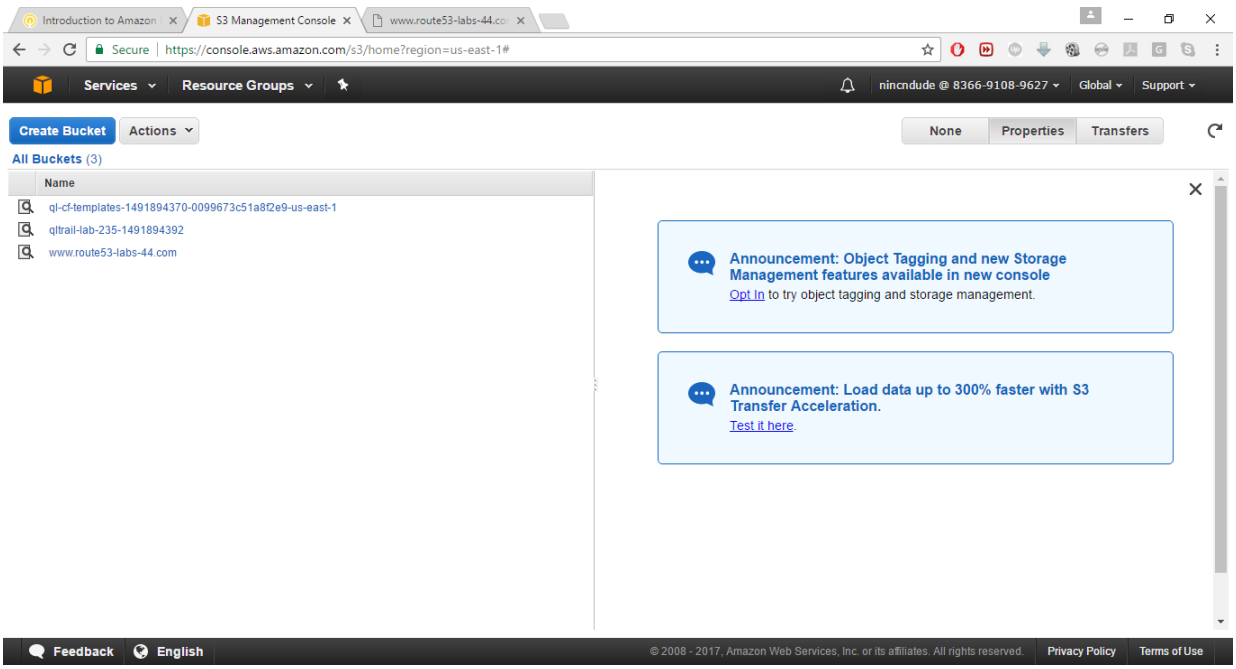
And choose the fail over record type as **‘Primary’** and click **‘yes’** for the associate health check and select the healthcheck you just created and click on **save record set.**

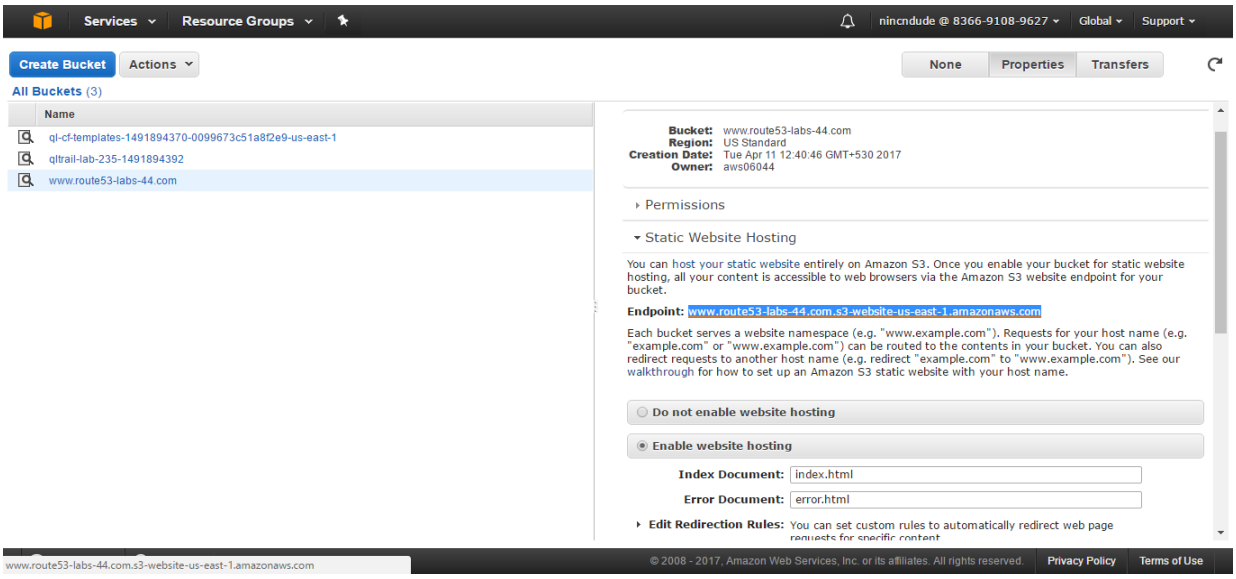


Now that we have configured the **primary endpoint**. Now what we have to do is that we will create secondary endpoint or back up endpoint.This can be done by creating an S3 bucket matching the same domain name.

Now go to S3 feature using AWS console and create a bucket with the same name

and click on properties and enable web hosting and upload index and error html files with some text written static into it.





Now copy the end-point and save it in a text file

Now go back to Route 53 and click on hosted zone and click on domain name and click on create record set.Now what we do is we create an alias record set and point our S3 endpoint to the domian name as a failover.

After you click on create record set you will see a side box opens u

Now you type the following:

Name : www

Type : IPV4

Alias :Yes

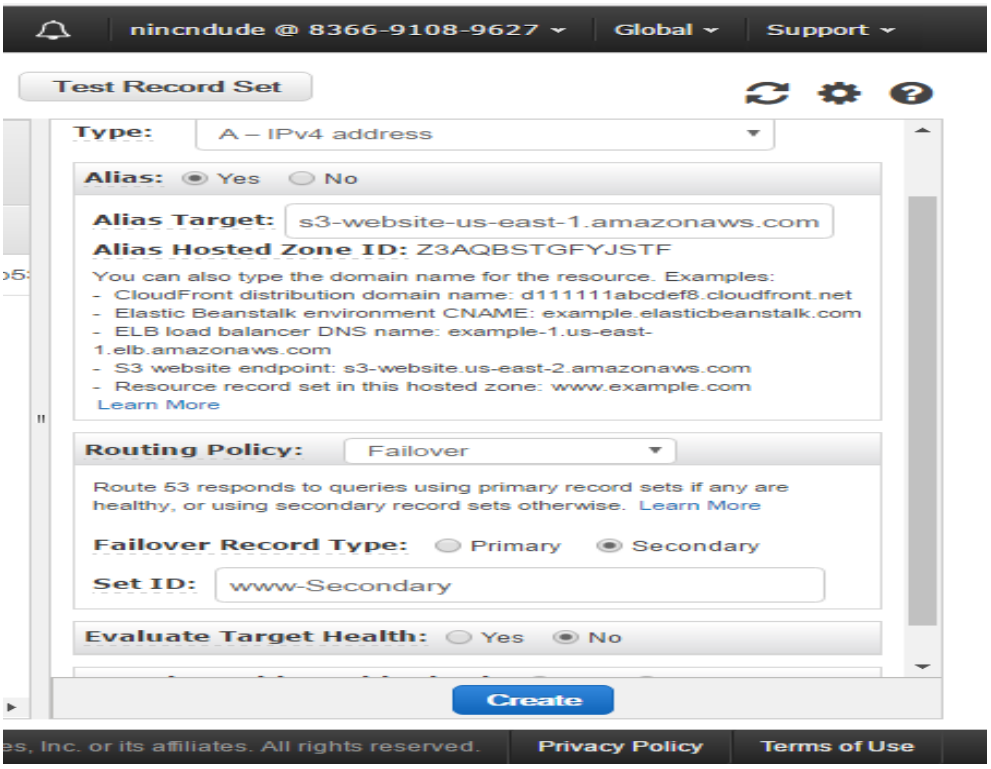
Alias Target: click on the field and select the S3 endpoint

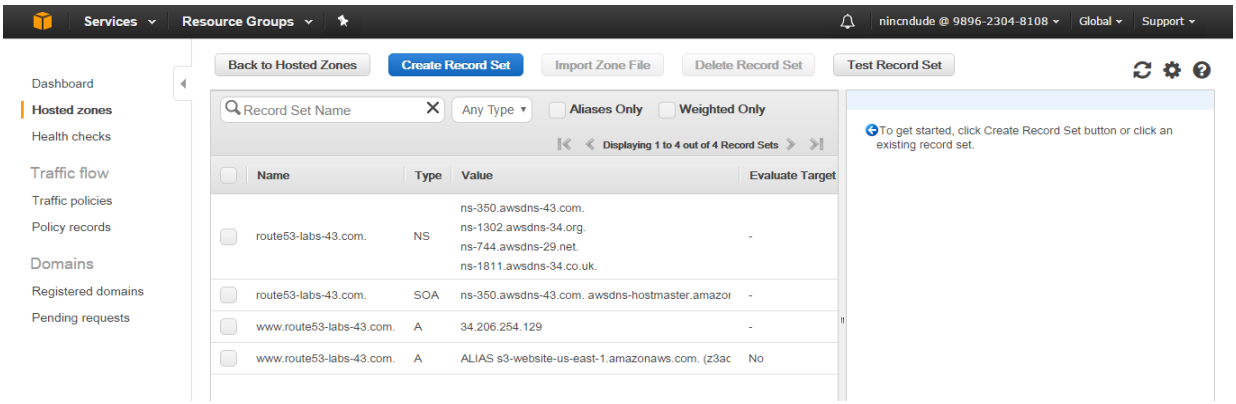
Routing policy: failover

Failover record type: secondary

Associate health check: No

And then click save record set



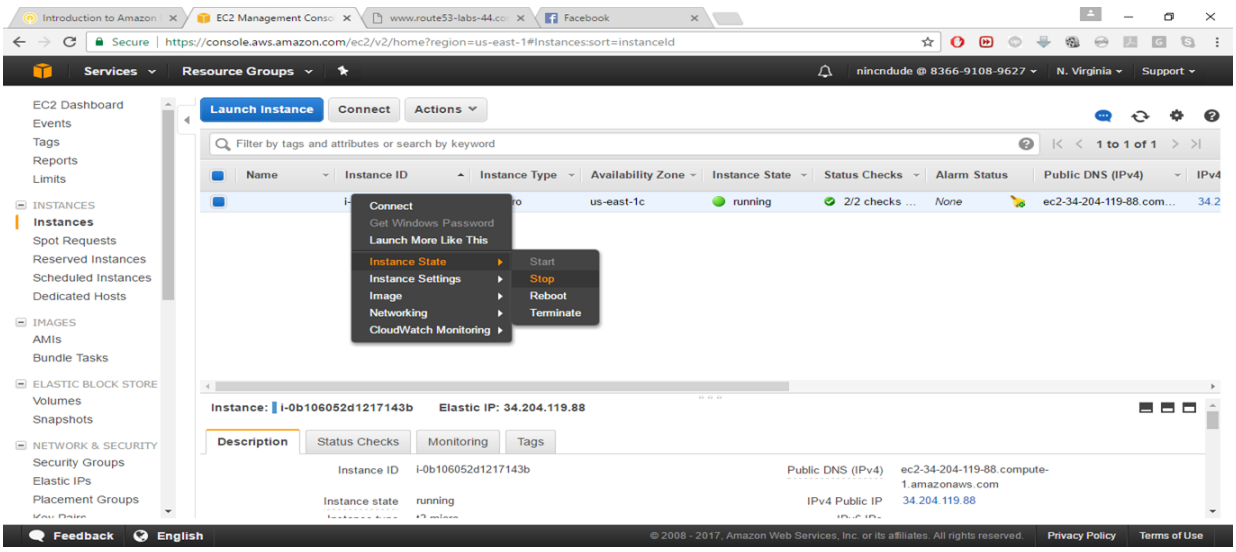


As you can see S3 is our Alias or CNAME record for our domain(FQDN)

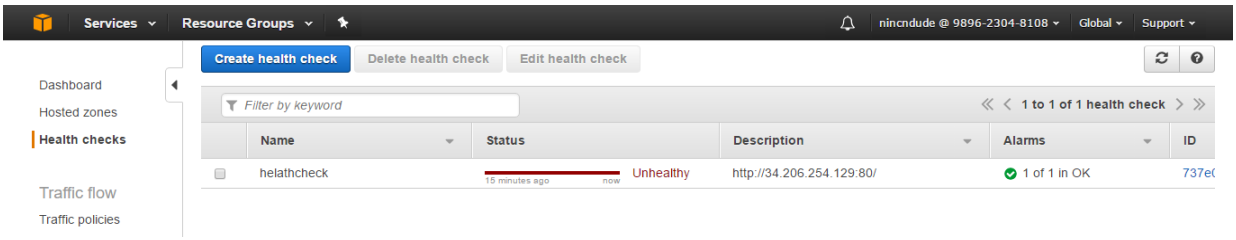
So uptill now what we have done is first we set our main domain routing policy to failover and configured that as primary .Now we created an S3 bucket and enabled static web hosting and pointed that bucket to our domain as seconodary endpoint for a DNS failover

Step 7:Now we test how DNS failover and health check works incase of our domain is down

Now go to EC2 instance and stop the instance



Now go to Route 53 and health check.You will notice health check is unhealthy and probably you will recive a message to your email based on SNS settings



Now click on the domain and you will notice this (as written in Index.html)

So when our primary domain is down our S3 end point acts as fail over.

Now clean up:

Turn off all your instances

Conclusion:

So what we have done:

1.Registerd a primarydomain

2.Created a hosted domain

3.Associated NS records to the domain

4.Launched an instance and associated Elastic IP to it

5.Created a resource record and associated elastic IP with primary domain

6.Cofigured health check to our primary domain

7.Configured DNS failover to Amazon S3

8.Tested health chek and DNS fail over by stopping EC2 servers

S3 concepts:

Enable CORS Configuration

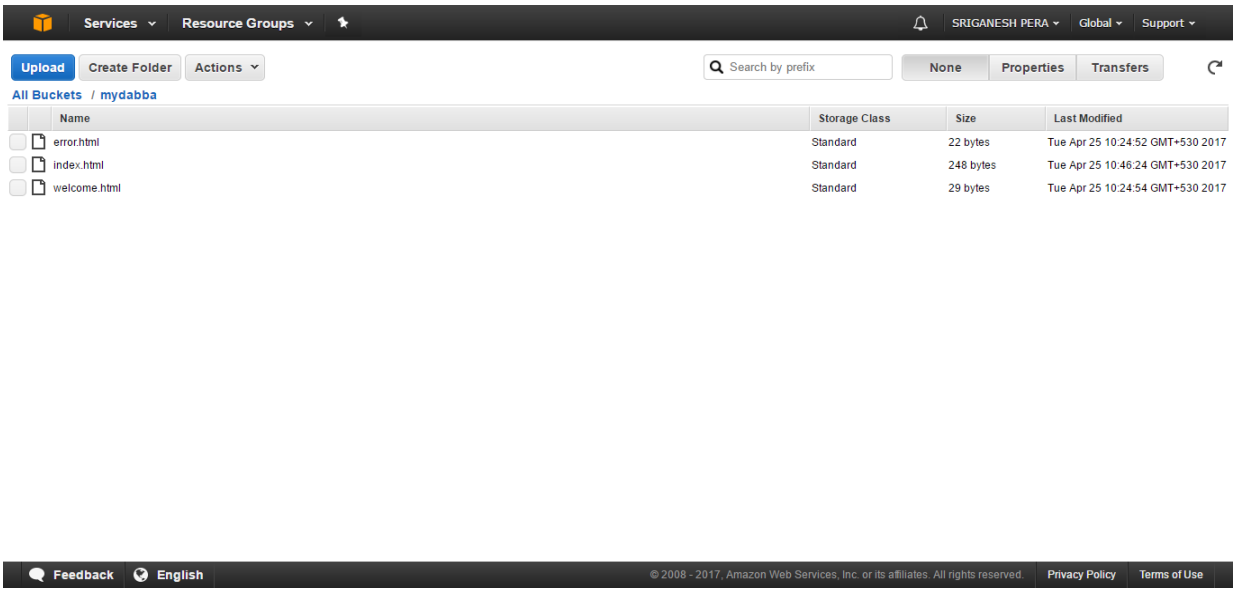
Cross-origin resource sharing (CORS) defines a way for client web applications that are loaded in one domain to interact with resources in a different domain.

That means you can access resources from one bucket when you hosted a client-side web app using a different bucket.

To use that you should enable CORS feature.

Lab Objective: Accessing objects from a different bucket by enabling CORS

Step 1: Create a bucket in S3 and load 3 files index.html, welcome.html, and error.html and enable static web-hosting feature



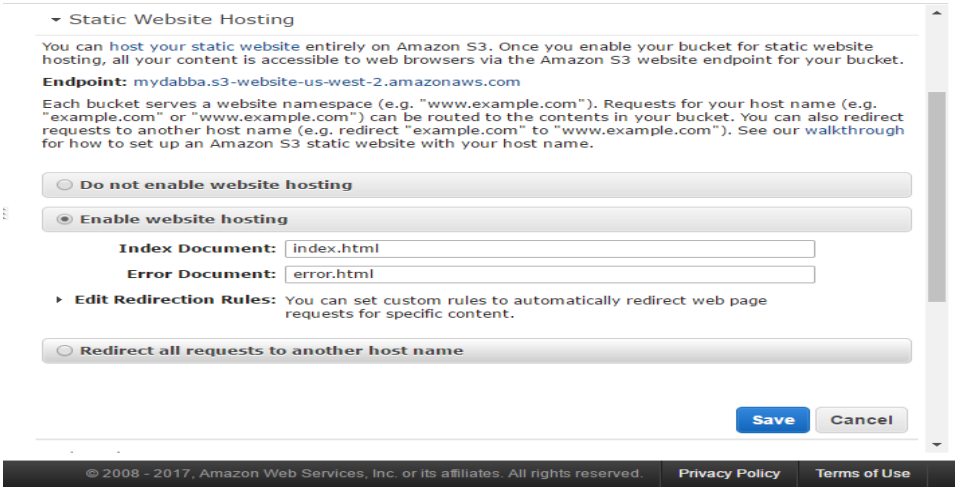
As you notice in mydabba bucket is the first bucket I create and I have uploaded 3 html files with some text written on it.

And I have made them avaliable to 'Public'.(To make objects public click on properties of each object and then click on add permission and set grantee to **'Everyone'**

Now enable static hosting:

Go to the bucket and click on properties and click on enable stastic hosting and in the index document and error document column

Type this as shown:

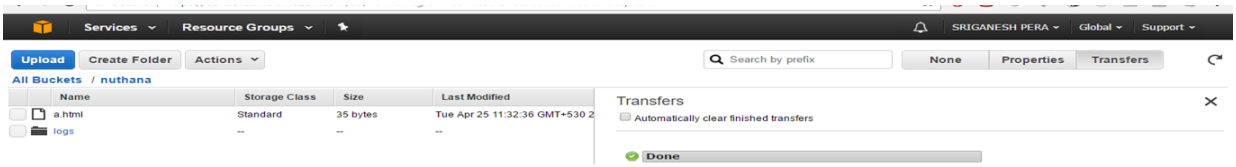


Now click on your index .html S3 link you will find this:

As you may notice the index.html has loaded the content both from index.html and welcome.html

I used Jquery script to enable this functioning.

Step 2: Create a second bucket and upload a file name A.html and enable static webhosting ONLY without typing the index and error document as mentioned above (reason: we have not uploaded index/error document here)



As you can see I have uploaded A.html file only in the second bucket named **'nuthana'** .

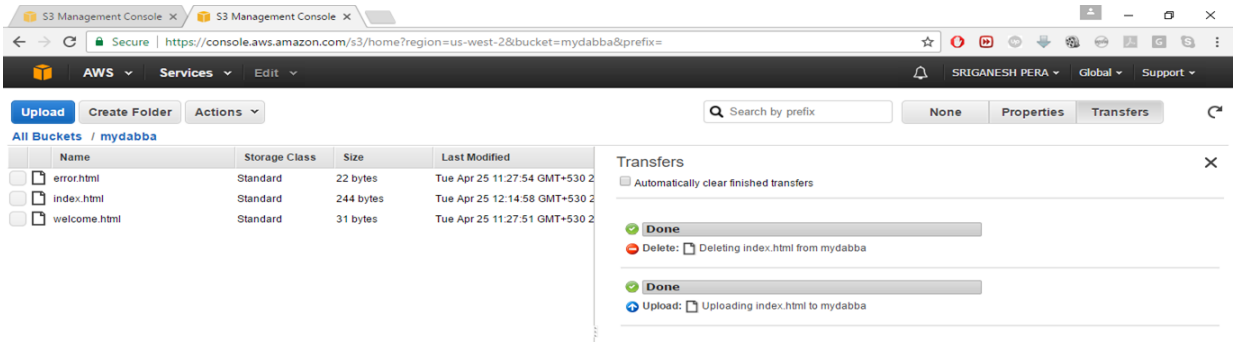
Make A.html availaible to public as well and copy the s3 static link of A.html

As you can see A.html has one line which says" Content loaded from Nuthana bucket"

Now copy this s3 link for A.Html and paste in index.html in the load funtion in the code editor.

The load function now has the S3 link for A.html

Now go back to the first bucket('mydabba') and delete the index.html file and upload the edited index.html file and make it public



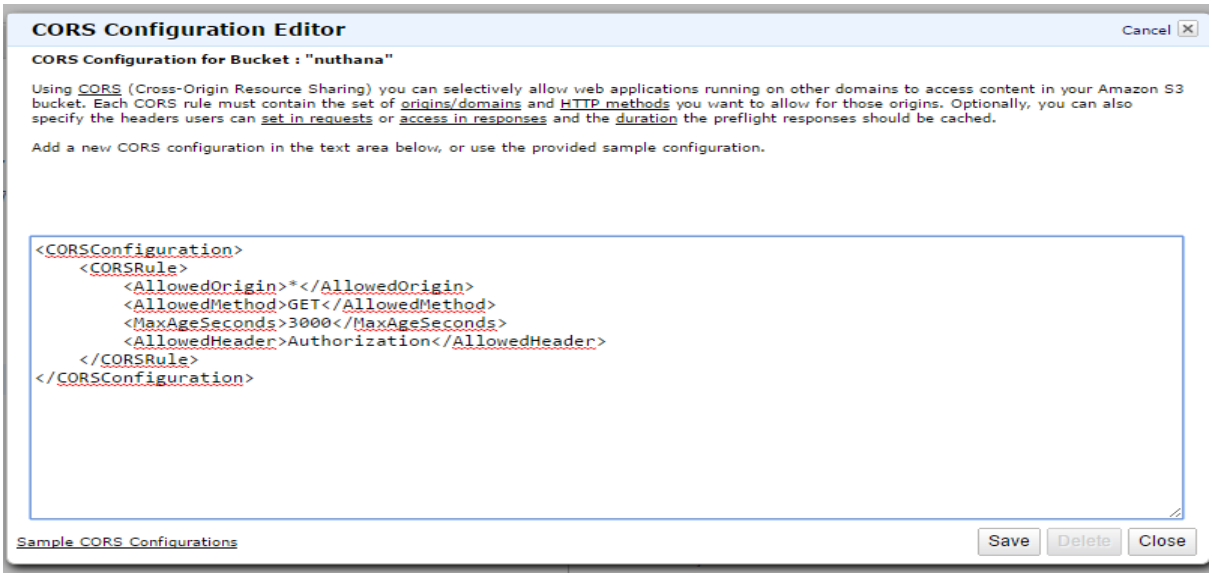
If you test that s3 endpoint link for index.html you will only notice that content of welcome.html gone and if you click on inspect element you will find the error 'No Access-Control -Allow-Origin header present'.

Step 3: Enable Cors for Second bucket (Nuthana)

Now what happens if we enable CORS origin is that we can easily access the A.html file within the index.html from the first bucket 'mydabba'

Go to properties of Nuthana bucket and click on permissions and click on **Add CORS Configuration**

You will notice an editor with some XML code written in it.



In this XML code:

<AllowedOrigin> : '*' means any bucket can access the contents of the bucket 'Nuthana'

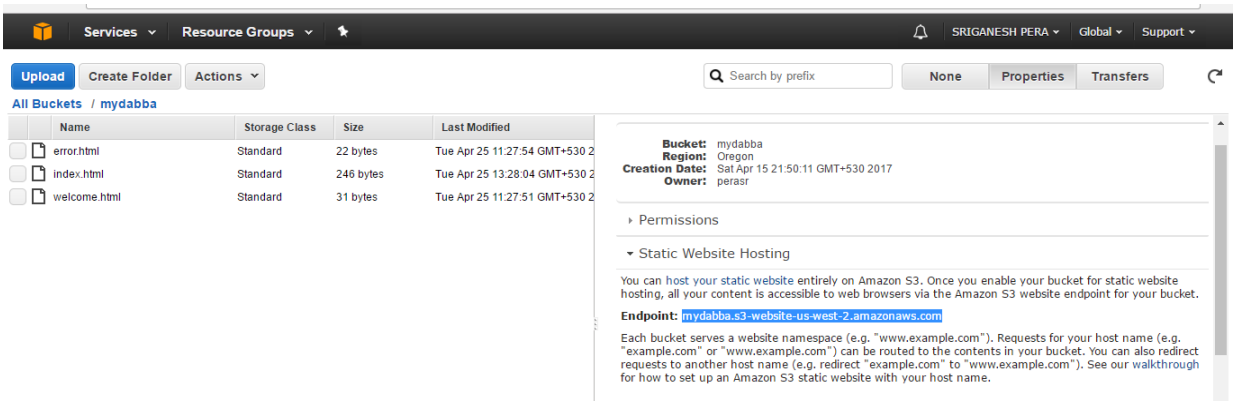
<AllowedMethod>: 'Nuthana' bucket allows only GET requests from any bucket

You can use other methods such as :POST,DELETE,LIST and HEAD

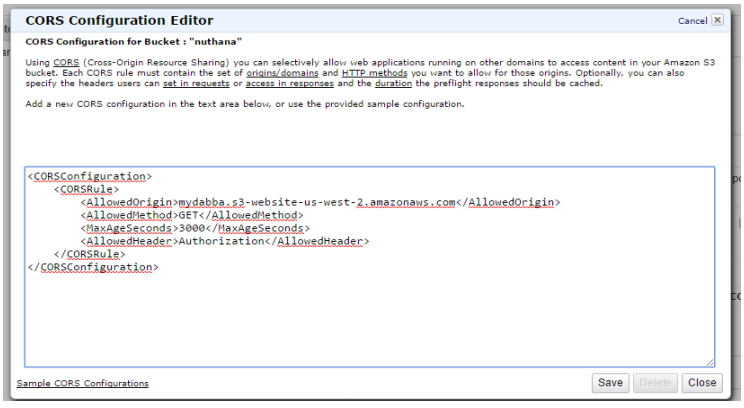
<MaxAgeSeconds> : Time required by the browser to Cache an Amazon S3 resource

Now what we do is we change the origin from '*' to s3 endpoint of the first bucket(mydabba) so that only the first bucket can have access for Cross origin resource sharing to the second bucket(Nuthana)

Copying endpoint of first S3 bucket(mydabba)



Pasting it in the CORS configuration of second S3 bucket(Nuthana):



And click Save.

Step 4: Testing CORS enabled

Now go back to the first bucket(mydabba) and click on Index.html file and you will notice the content of A.Html is loaded into index.html

Lab is success

S3 bucket and IAM policies:

AWS gives full permissions to the owner of a resource (bucket, object)

Resource owner can grant access to others, even cross-account, but the bucket owner who is paying bills can deny or modify objects regardless of who owns them.

Within S3 there are a number of ways to set permissions on your S3 resources, and two of these options are Bucket Policies and User Policies. Bucket policies are applied directly to a bucket within S3 itself, whereas user policies are set within IAM (Identity & Access Management)

Bucket policy and user policy are two of the access policy options available for you to grant permission to your Amazon S3 resources. Both use JSON-based access policy language.

S3 Bucket policies in a nutshell:

1.It's a resource based policy as we attach resources to the policies

2.Use JSON file for attaching a policy

3.Can grant other AWS accounts or IAM users permission for the buckets and objects inside.

4.Should be used to manage cross-account permissions for all Amazon S3 permissions

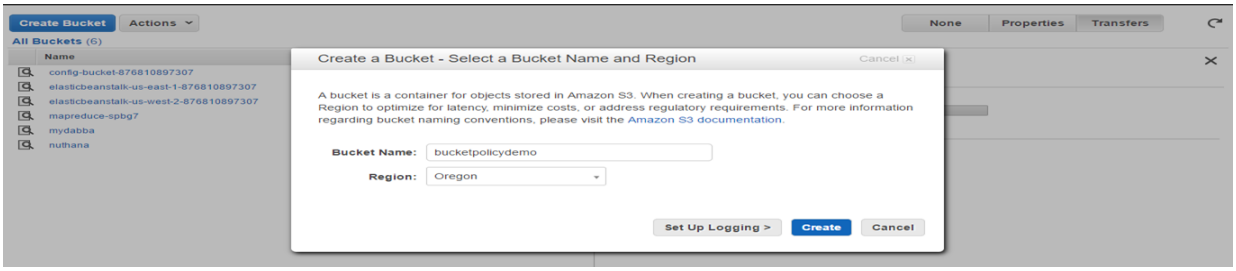
5.Limited to 20 kb in size.

Jaya shyamala- A(admin access) A creates bucket Ganesh -B(admin access)

ACL B can upload objects in the bucket created by A and owned by you!

Lab: Using bucket policy to restrict access

Step 1: Create a bucket

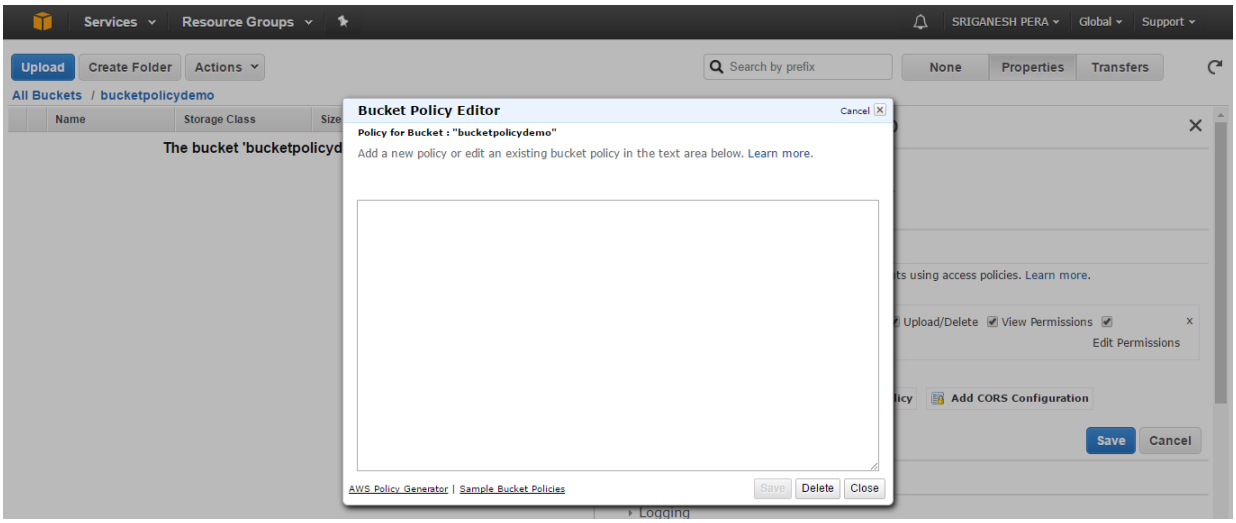


Step 2:Create a Bucket policy within S3 with IP Address conditions

1.Go to properties of the created bucket

2.Select permissions and selet add bucket policies

3.Select **AWS Policy generator** at the bottom of new window that appears



This takes you to a new web page which is the AWS Policy Generator.

The AWS Policy Generator is a tool that enables you to create policies that control access to your resources.

Now under step 1: **Select S3 bucket policy**

Under Step 2: **Effect, select Deny**

Under Step 2: **Principal, enter *

The Principal dictates the user, account or service that is policy will apply to.

Under Step 2: Actions, select **Tick-box All Actions**

Under Step 2**: Amazon Resournce Name (ARN)**, enter arn:aws:s3:::bucketpolicydemo/*

Note: bucketpolicydemo is the name of the bucket I created. You should replace that part 'bucketpolicydemo' with your own bucket name.

ARN: Amazon Resource name

ARNs are set up in the pattern of arn:paritition:service:region:accout-id-resouce.

'Partition' – This relates to the partition that the resource is found in. For standard AWS regions, this section would be 'aws.'

'Service' – This reflects the specific AWS service, for example 's3.'

'Region' – This is the region where the resource is located. Some services do not need a region specified, so this can sometimes be left blank.

'Account-ID' – This is your AWS Account ID (without hyphens). Again, some services do not need this information, and so can be left blank.

'Resource' – The value of this field will depend on the AWS service you are using. For example, if I were using the Action: "Action":"s3:*", then here we have used the bucket name that I wanted the permission to apply to arn:aws:s3:::bucketpolicydemo/*.

Now select **Add conditions(Optional)**

This section allows you to add specific conditions to the Policy. These conditions allow you to define a greater granularity to your policy to only execute under certain conditions and keys.

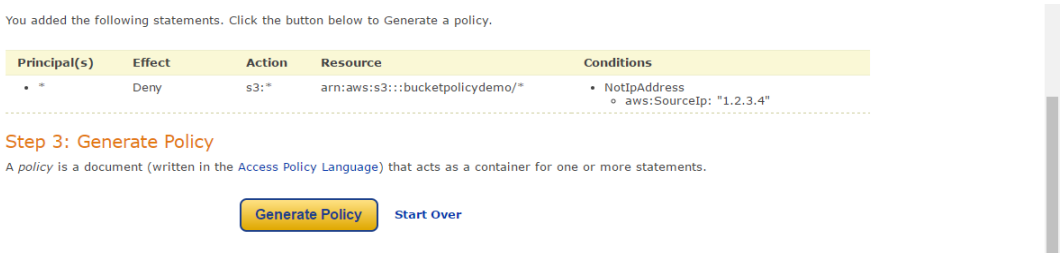
Under Condition select **NotIpAddress**

Under Key select **aws:SourceIp**

Under Value enter **1.2.3.4**

Click on the yellow **Add Condition** button

Click on the yellow **Add Statement** button

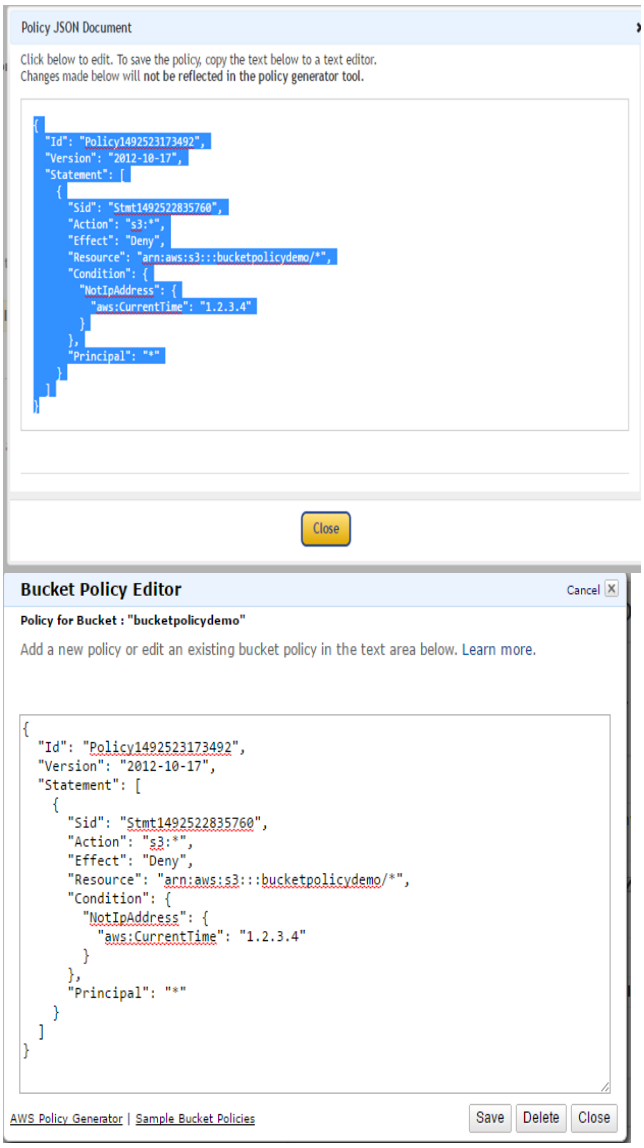


Now you 'll see an overview of what we have done

We have now created a bucket policy that will deny access to all S3 functions within the 'bucketpolicydemo' bucketfor anyone whose IP Address is NOT 1.2.3.4

Now under **step 3: Click on generate policy**

Now a policy is generated using JSON. Copy that policy and paste it in the bucket policy editor window in our S3 console and click **save.**



Now you have created a bucket policy for S3 which blocks access to the bucket 'bucketpolicydemo' from any IP exept 1.2.3.4

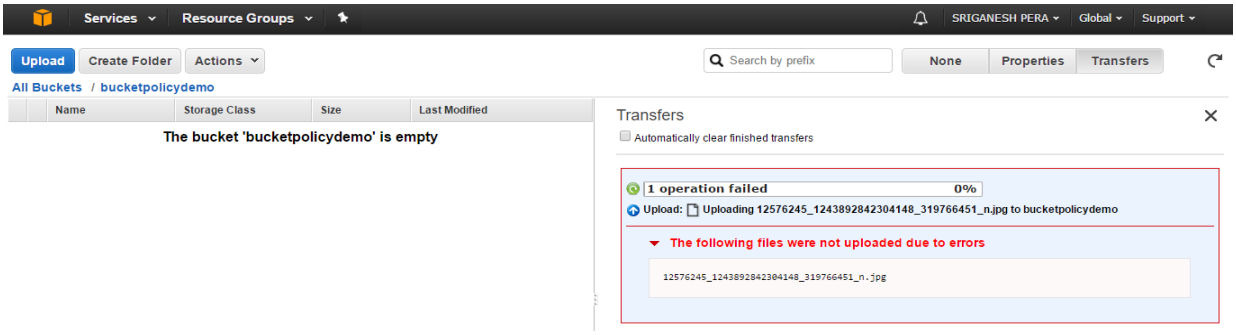
Step 3: Test the policy

a. Select your bucket and try to create a folder

It wont work ☺

b.Try to upload any file

It will give you an error message like this as my IP is not 1.2.3.4

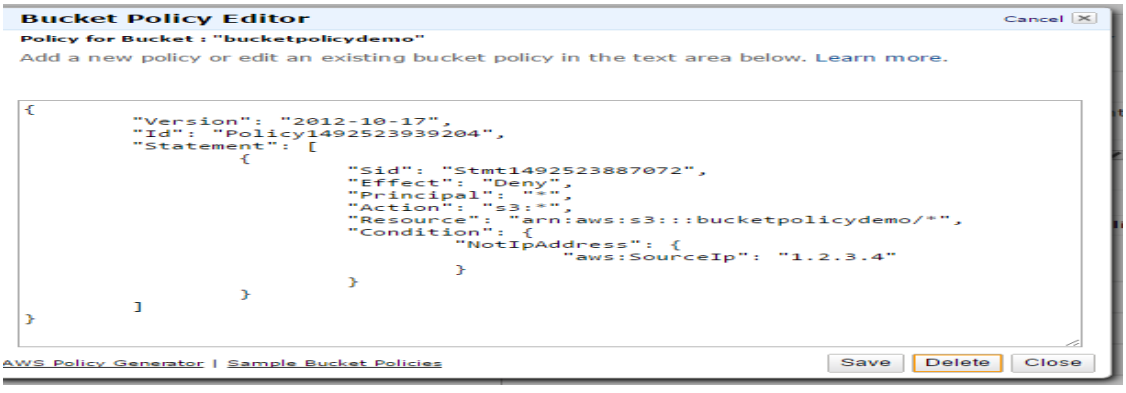


Lab 2: Create a Bucket Policy Within S3 with Encryption Conditions

In this step we will create a bucket policy which ensures that any uploaded object that does not have **server-side AES256 encryption** is denied.

AWS documentation states: Server-side encryption is about data encryption at rest—that is, Amazon S3 encrypts your data at the object level as it writes it to disks in its data centers and decrypts it for you when you access it. As long as you authenticate your request and you have access permissions, there is no difference in the way you access encrypted or unencrypted objects.

Step 1: Select the bucket which we created in previous lab and click on **properties** and click on **edit bucket policy** and click on **delete.**



Step 2: select properties then click on add bucket policy and click on AWS policy generator and it will take you to new window 'Aws policy generator' for specifying conditions just like in lab 1.

Now under step 1 : Select policy type choose S3 Bucket policy

Under step 2

Effect : deny

Principle : *

AWS service : Amazon S3

Actions: Put object

Resource: arn:aws:s3:::bucketpolicydemo/*

Note: bucketpolicydemo must be replaced with your own bucket

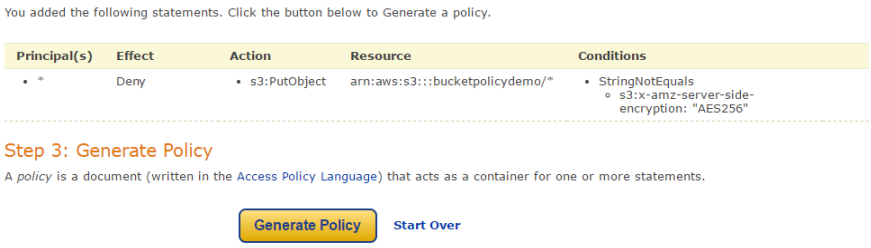
Select **Add conditons**

Conditions : ‘stringnotequals’

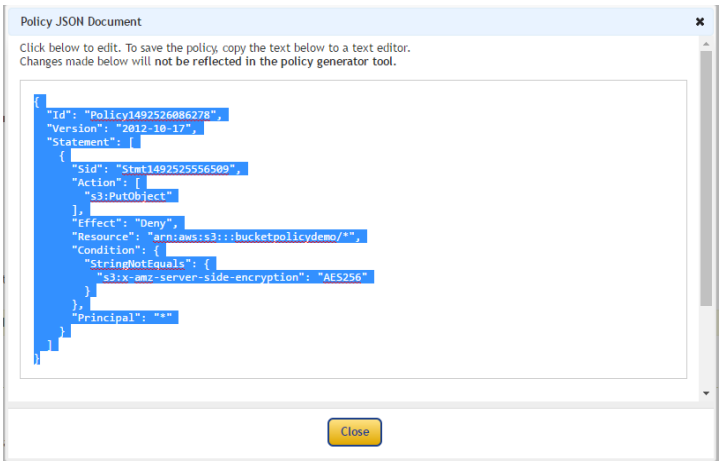
Key: s3:x-amz-server-side-encryption

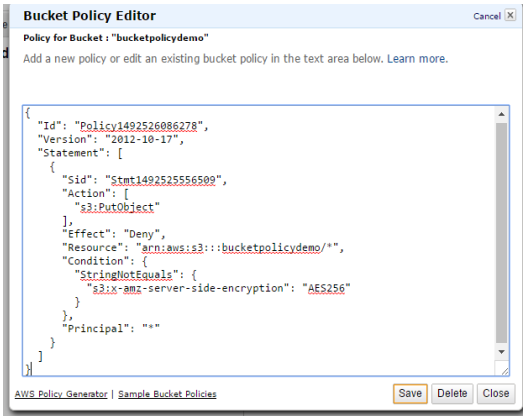
Value : AES 256

And then click on Add condition and Add statement button you will see this:



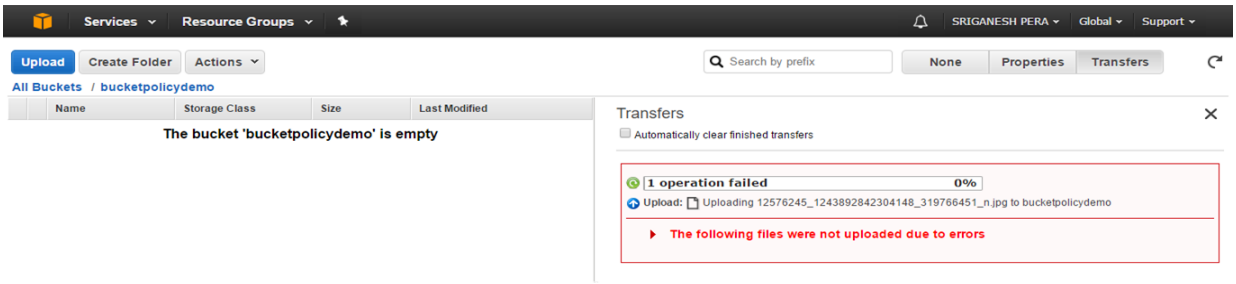
Now click on generate policy copy the JSON policy and paste it in your ‘bucketpolicydemo’ editor and click **save.**





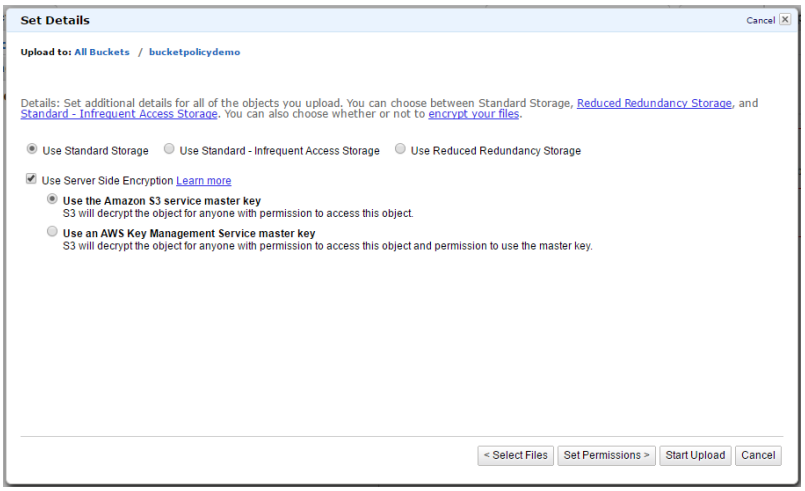
Step 3: Testing your policy

a. Try to upload a file from your local computer and see what happens

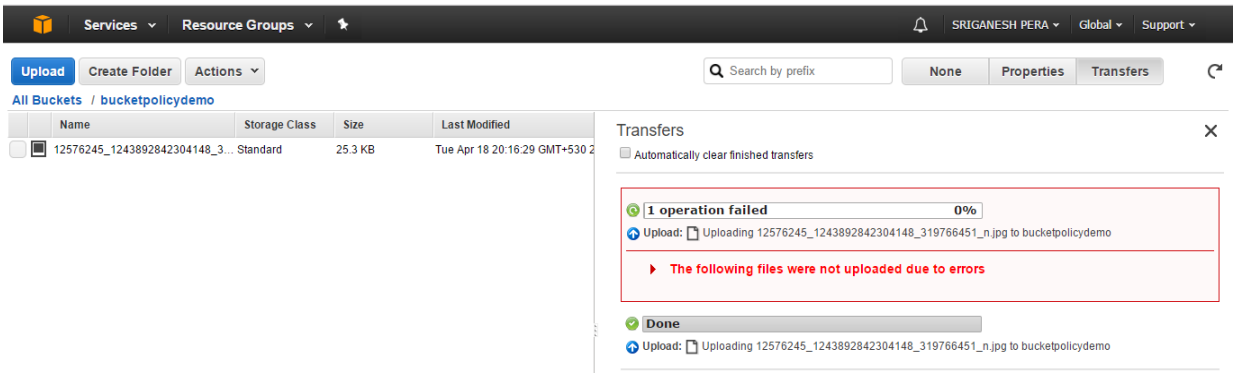


The upload will fail as we have not enabled SSE on the object which shows that the condition we applied within the bucket is working.

b.Now upload the same object and before you click on upload select ‘set details’ and Tick ‘Use server side Encryption’ and select ‘Use the Amazon S3 service master key’ and click upload.



Yay yay! My object is uploaded



As you can see my object is uploaded there.

Logging S3 API calls:

In one of the previous labs we have done making API calls to S3. As you know API calls can be made anyone (Bucket owner,developers,or even contractors)

We can actually record whoever makes API calls to S3 buckets using:

1.Cloud Trail: It logs API calls as well through Console for Bucket level operations and those logs are stored in a S3 bucket of your choice

2.Amazon S3 server access logs: It records info for Object level operations:

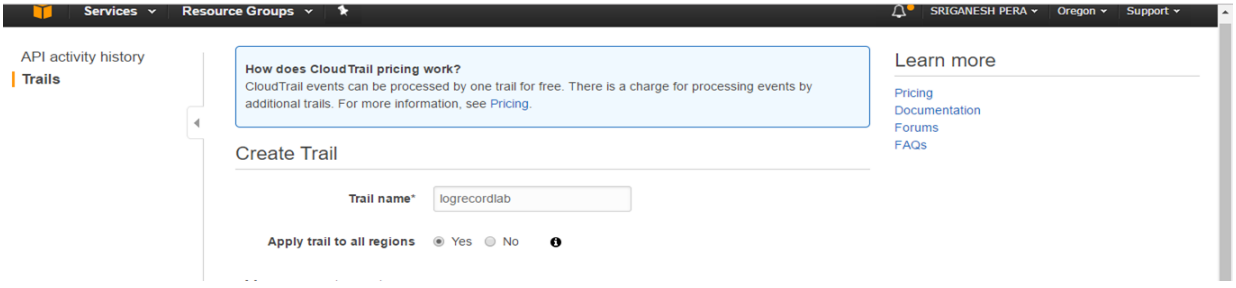
1. Who made request
2. IP address of the source
3. And time of access

Lab 1 : Recording logs in Amazon cloud trail using S3 API's

Step 1: Click on Cloud trail in AWS console

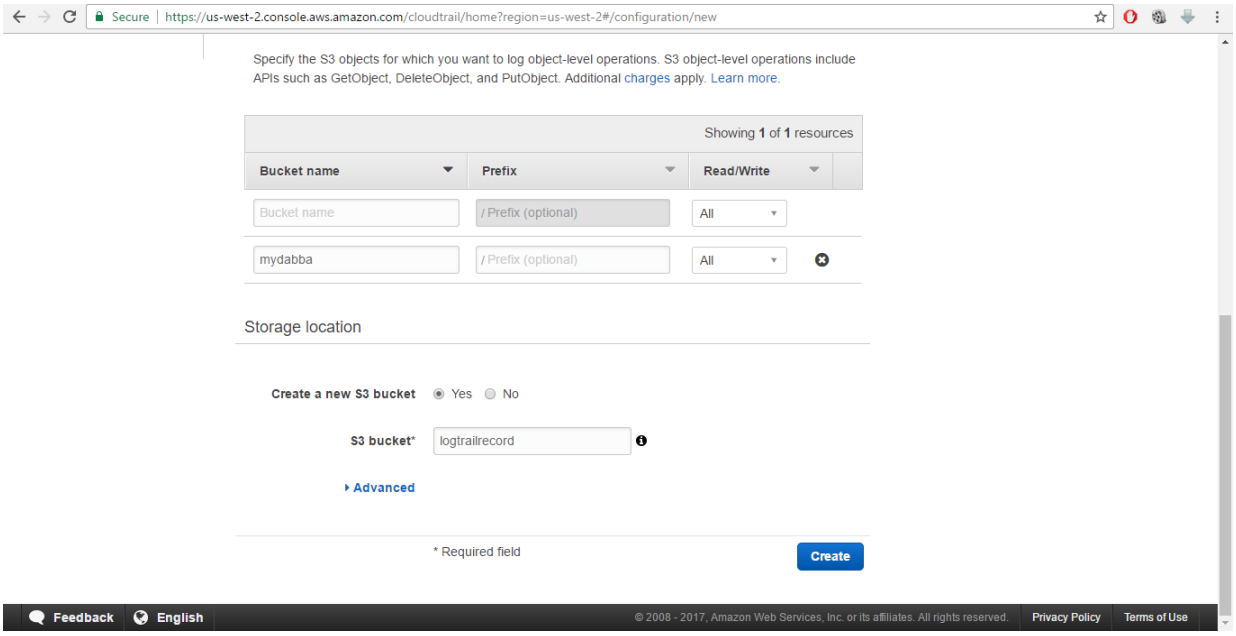
a. Then click on **Trails** and **add new trail**

b. Then give a trailname and select apply to all regions



c. Now leave the bucket name empty

d. Create a buket in which you will find all the recorded log trails and click create.



Step 2 : Make some API activity history

If you click on API activity hoistory you will probably find it empty if you have not enabled this before

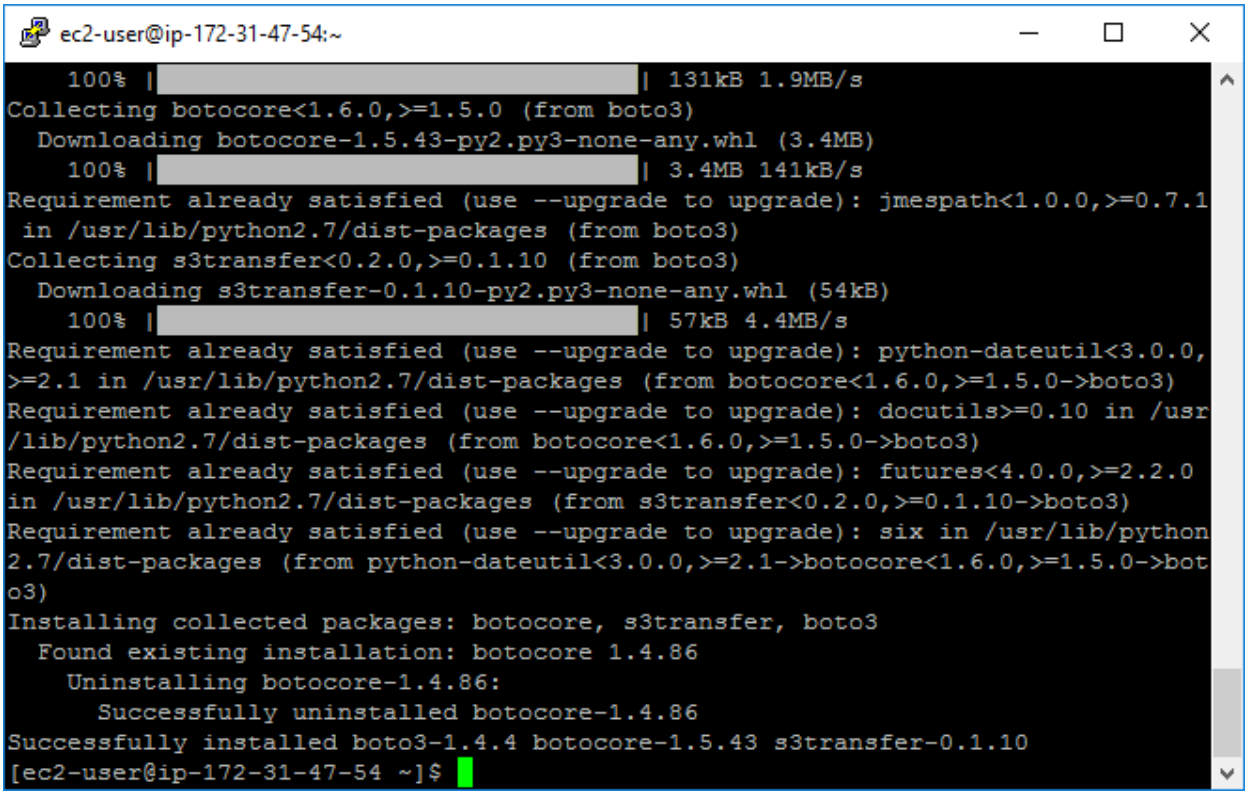
Now let make some api activity to the bucket we have chosen to record the logs

What I will do now?

1.Create an EC2 instance ,ssh into it and install boto3 SDK on the server

2. Create a new bucket with boto3 SDK

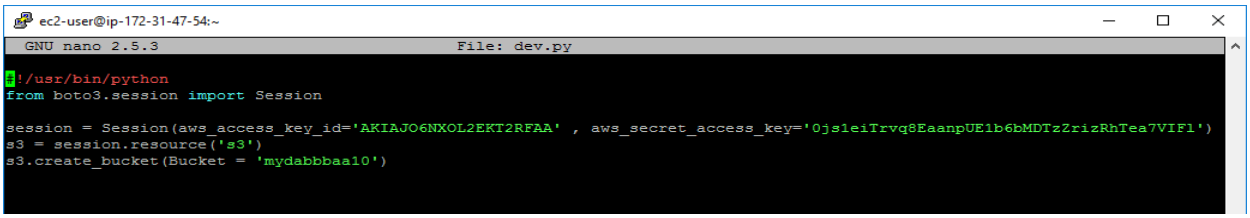
1.Create an EC2 instance ,ssh into it and install boto3 SDK on the server



2. Create a new bucket with boto3 SDK

Now create new python file dev.py using the command

$ nano dev.py and write the following code(which creates a Bucket using Python SDK) in it



As you notice from the code those access keys and secret access keys are used from an IAM user who has admin access.So to get that you need to create an IAM userand assign the user Admin access

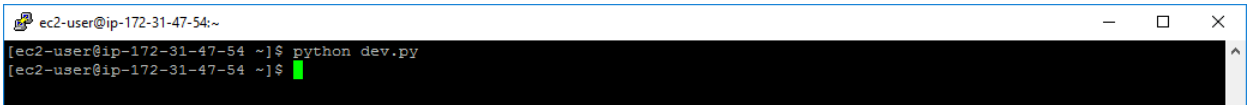
In the code you can clearly see I created a new Bucket named ‘mydabbaa10’.

Now save the file and exit from the editor and run the code

So to run the code you have to write the following the commands:

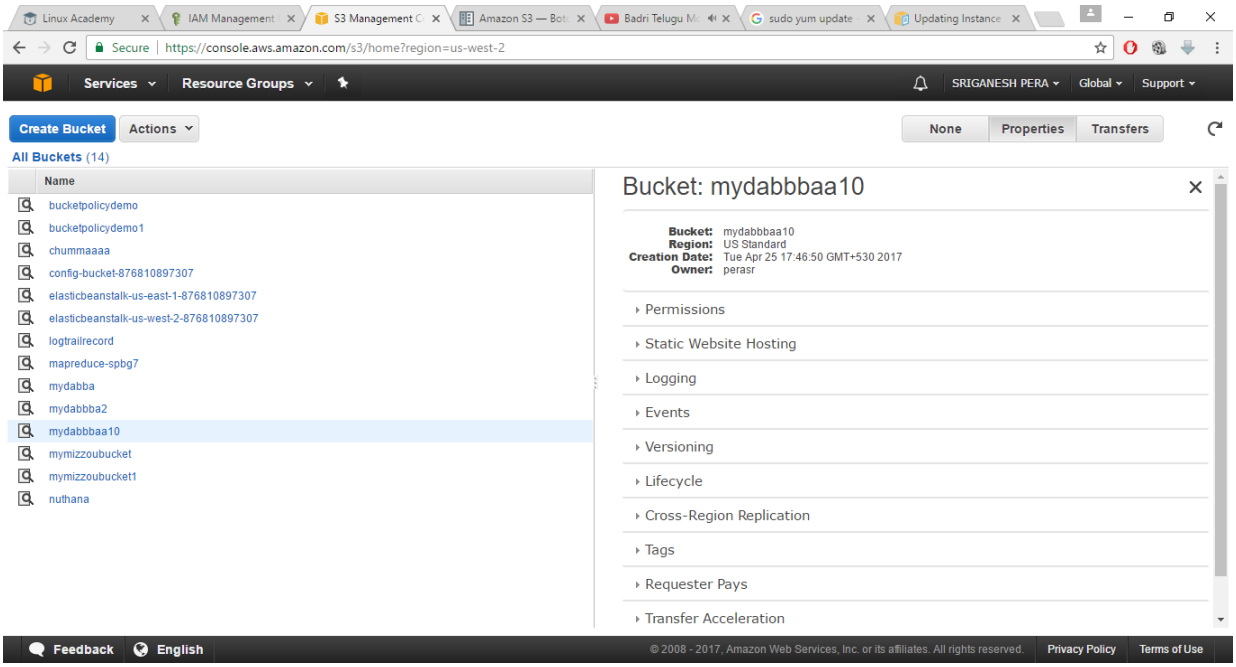
Chmod u+x dev.py

Python dev.py



Since I didn’t get any error I presume a new bucket(‘mydabba10’) has been created in my S3 console.

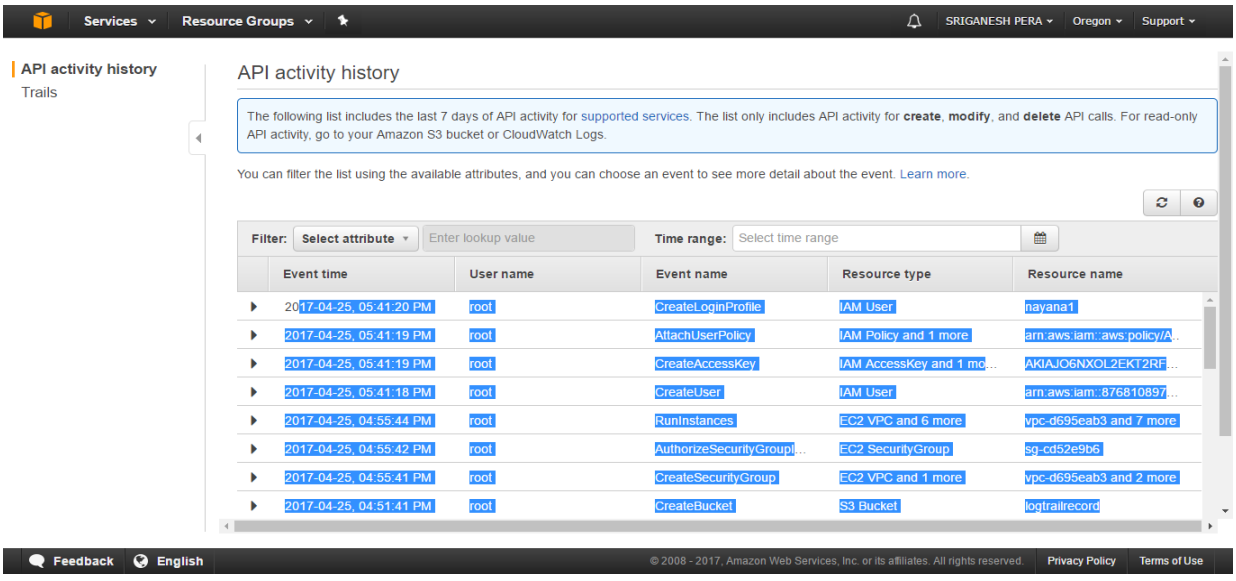
Lets check it out.



So my new bucket is created.

Step 3: Check cloud trail logs whther the current API activity is logged or not.

Now go back to Cloud trail feature and click on API activity



As you can see It recorded everything:

1. IAM user,attaching policy and creating access keys fro the user

2. EC2 isntance launching

3. S3 bucket creation

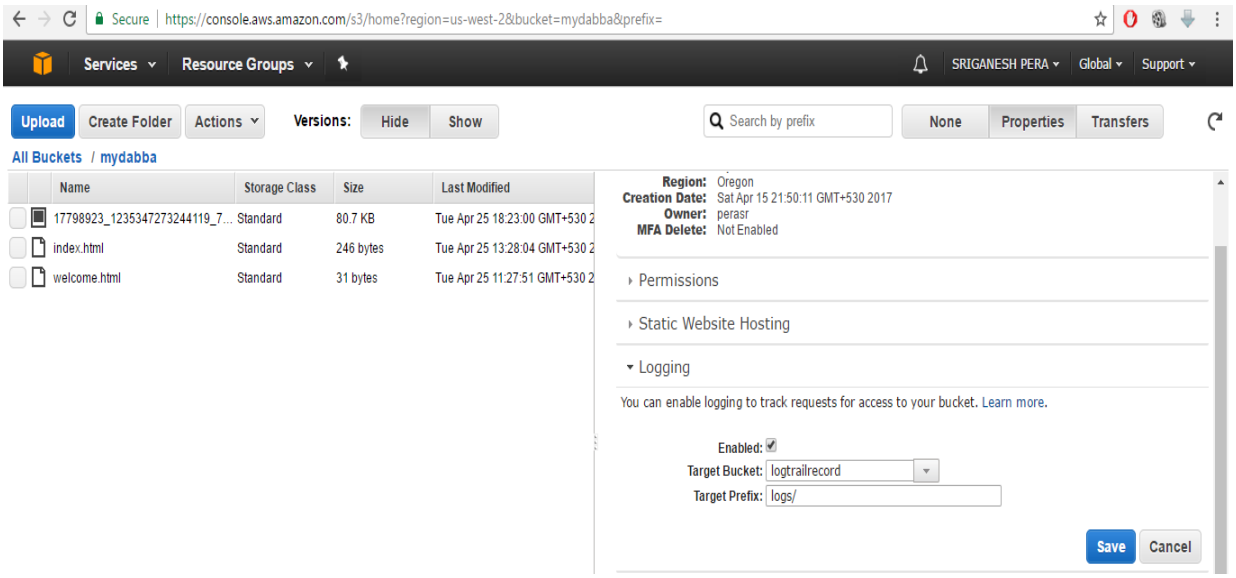
So,cloud trail enables us to record any changes we made to S3 at a bucket level as well as object level for some additional charges.

Now enabling server access logging specifclly records changes made to objects

Lab 2: Server access logging

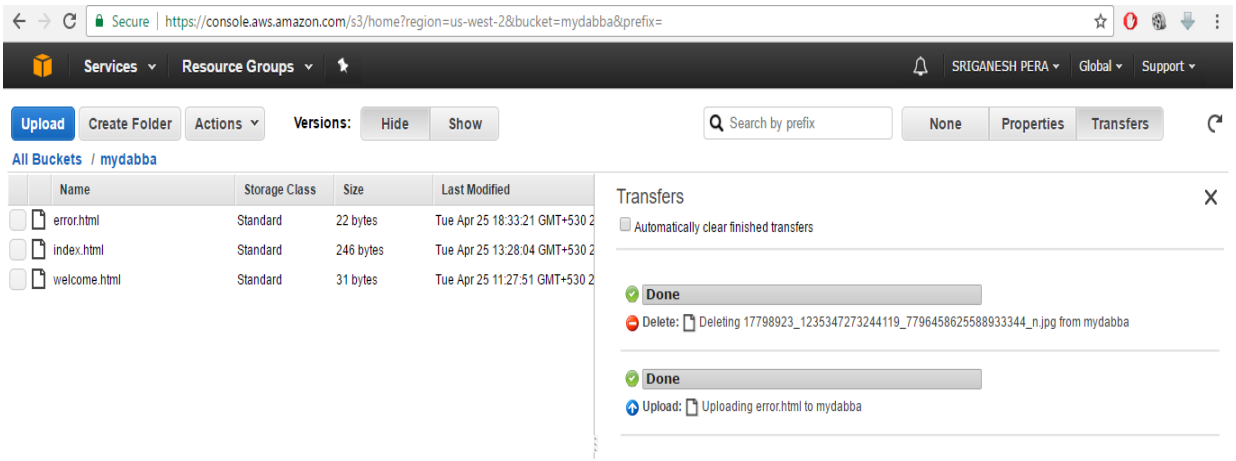
Step 1: Go to S3 console and click on the bucket for which you want to enable server access logging

Step 2 Go to properties and click on logging and check enable and chose Destination bucket where those logs are stored and click Save.



Step 3 :Now make some changes on the objects of the selected bucket.

Here I will delete an object and upload a new object

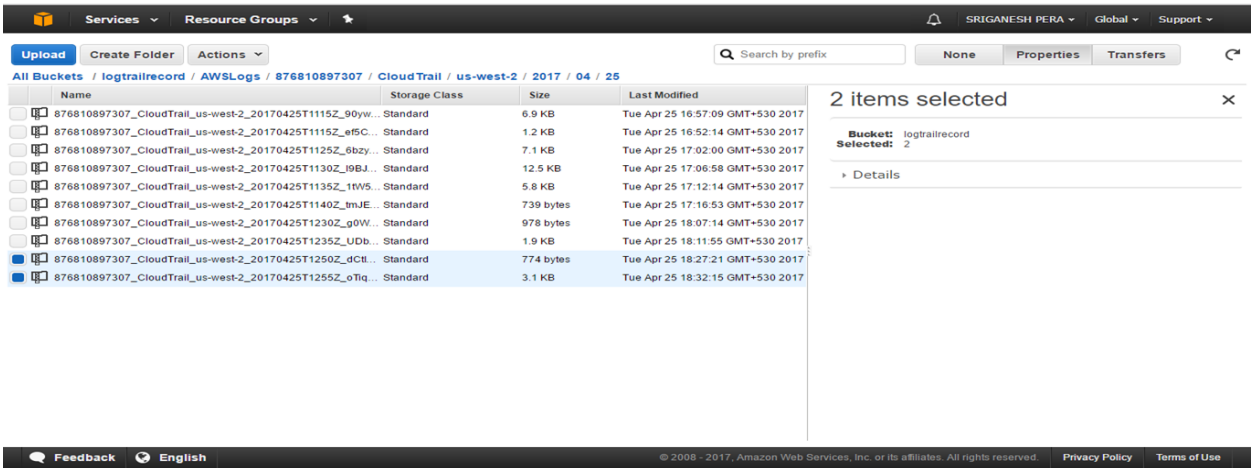


As you can see I deleted one object and uplaoded a new object

Now these Object level changes are recorded and stored in the target bucket we have chosen in the step 2.

But in this case you wont see the changes immediatley .It might take min 1 hour to see those changes.

<u>After 1 hour:Those 2 changes are recorded</u>



-

This is how you can log changes to S3 buckets and objects.

~~~~~~~~~~~~~~~~~~~~~~~~End of lab~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~