Sample application Assessment

**Overview:**

Took a sample application from Github, deployed it on Elastic Beanstalk with Multi-AZ (HA).

DynamoDB to store user data (run a value from DB).

Multi-AZ RDS to store thumbnails. Elastic cache to cache the DB queries

S3 will store videos and also contains java application.

Elastic Transcoder to transcode video when user uses various devices like tablet, pc, etc.

Added SSL certificate using ACM with ELB

There are cloudformation templates nested stacks to deploy this infrastructure

**List of AWS Resources**

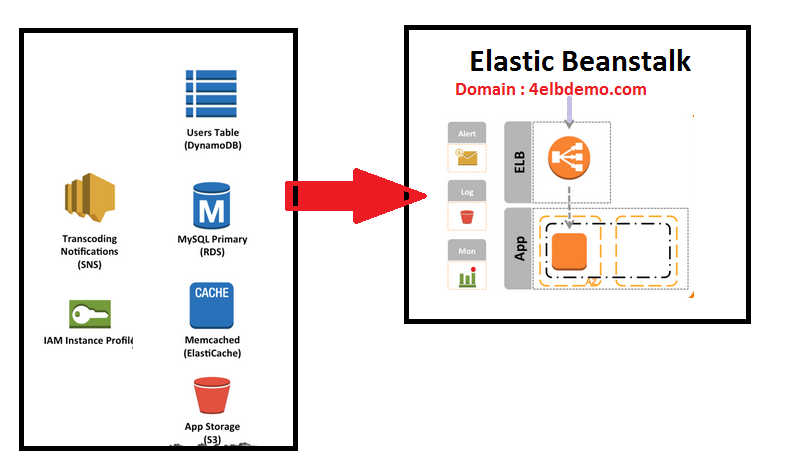
1. **Elastic Beanstalk**
2. **Cloud Formation**
3. **DynamoDB**
4. **RDS**
5. **Elastic Cache**
6. **Amazon Elastic Transcoder**
7. **S3**
8. **Route53 and AWS Certificate Manager**
9. **SNS**
10. **Cloud watch**

**Note : We have taken Sample Java Application which is used to upload , convert and share video via a web browser. We are integrating with AWS Resources**

**About the Infrastructure:**

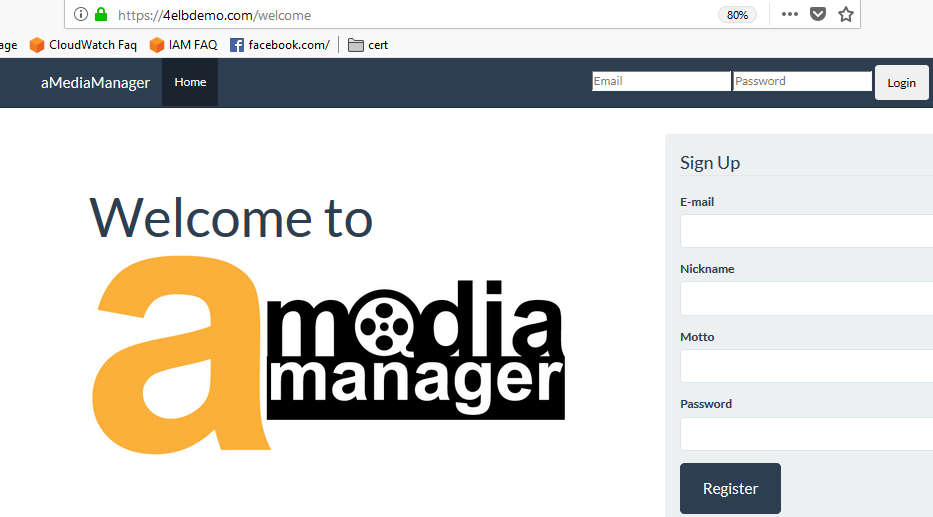
The Java application uses a number of AWS services for data storage, video conversion, security, and messaging, including Amazon S3 for uploaded videos and app logs; Amazon RDS for storing searchable video metadata; Amazon Elastic Cache for caching database queries; and Amazon DynamoDB for storing user profile information.

Here’s an overview of our Elastic Beanstalk application and the other AWS Resources it requires. We’ll model all of them in a CloudFormation template:

****

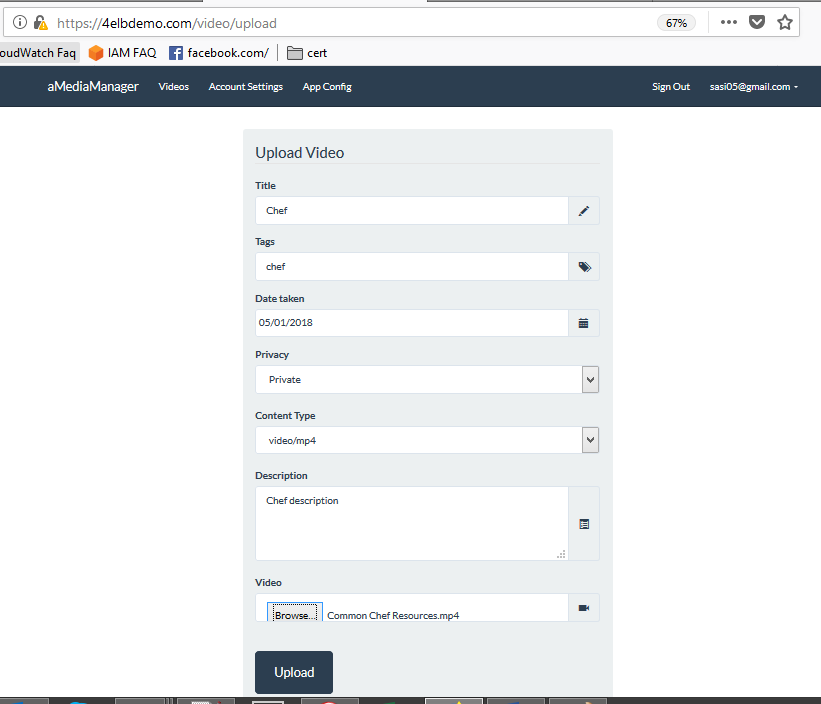
**Regarding Application :**

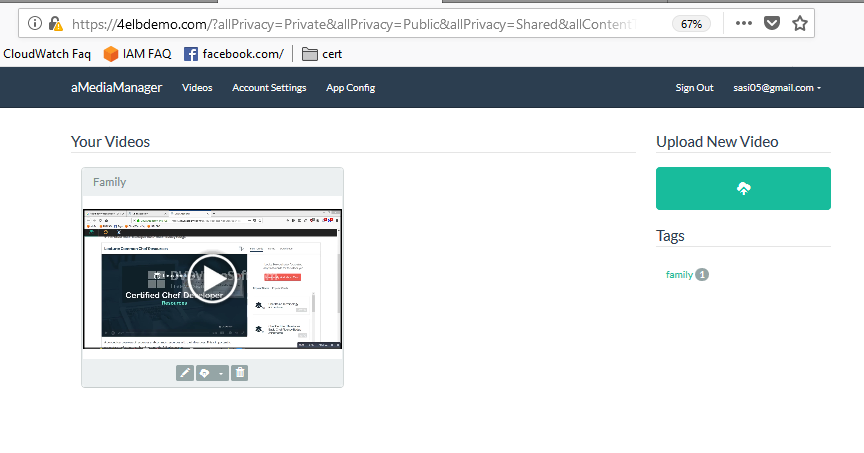
1. **User Login**



1. **Video Upload ,Conversion and Management**

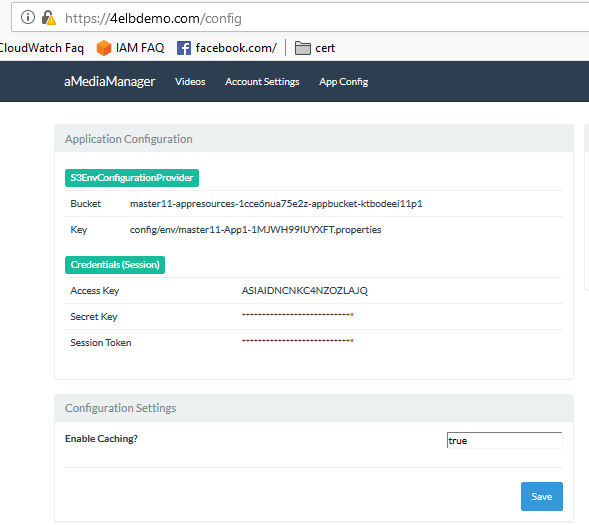
Videos are uploaded and stored directly to S3, while Elastic Transcoder creates a video thumbnail and creates a streaming version. Video metadata, including tags, is stored in RDS and cached in ElastiCache.





1. **Admin Console**

The console also provides information about the application configuration and AWS credentials being used by the app.



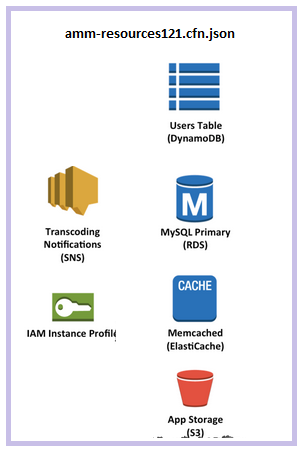
**CloudFormation template:**

we compose our entire app and infrastructure from 3 templates

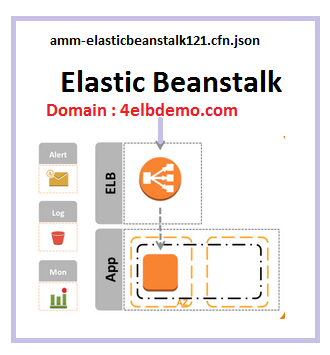
1. **amm-master121.cfn.json :** This is the parent template and it only defines two Resources, both of them embedded CloudFormation stacks. To run the entire application, run this stack

Two resources.

* 1. **amm-resources121.cfn.json:** This template defines all of the dependencies for our application, including RDS databse, ElastiCache cluster, DynamoDB table, S3 bucket, IAM roles, etc. It is the first child stack created by amm-master121.cfn.json, and it outputs the IDs of everything it creates.

****

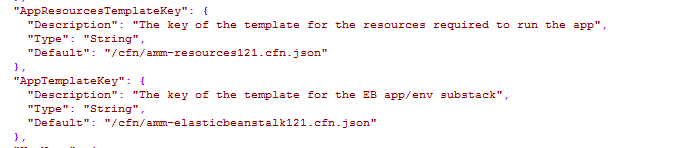
* 1. **amm-elasticbeanstalk121.cfn.json:** This template defines the Elastic Beanstalk Application and Environment that runs our app code. It takes as inputs the IDs of its resource dependencies (DB hostname, S3 bucket name, etc). It is a child stack defined in amm-master121.cfn.json, and that parent template provides its inputs by referencing the outputs of amm-resources121.cfn.json

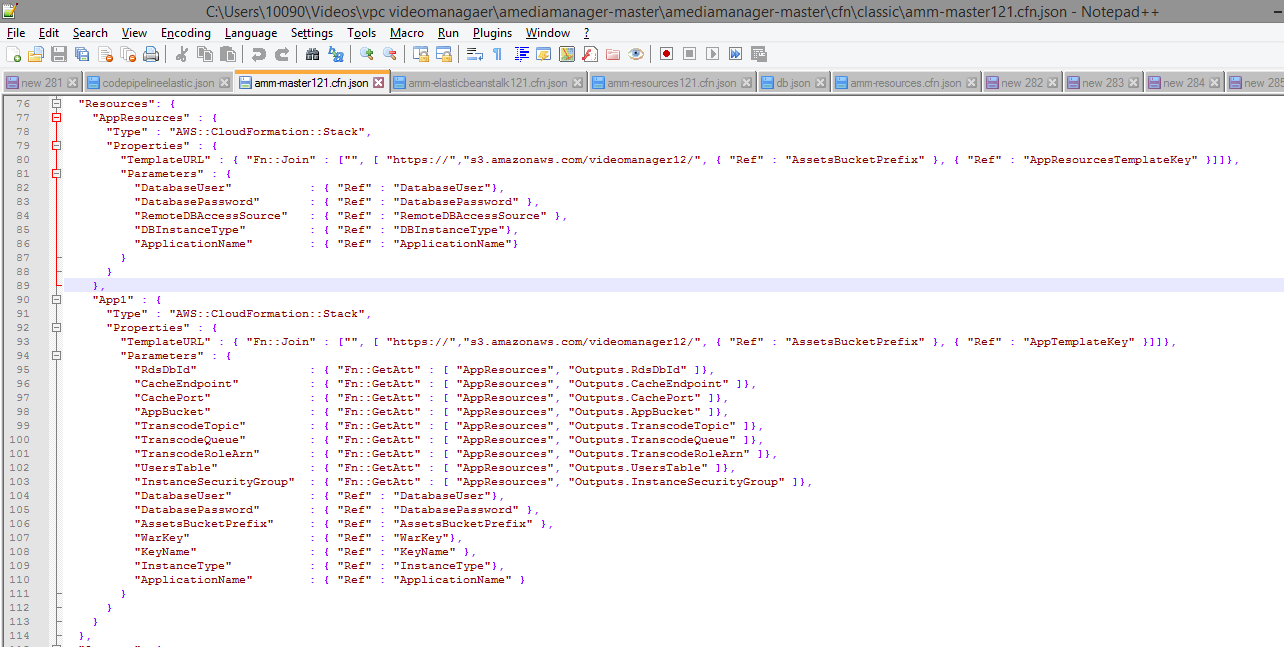
****

Here we are using one stack to bring All **amm-master121.cfn.json**

**amm-master.cfn.json is the parent template**. It defines several input parameters and just 2 resources. Those resources are both embedded CloudFormation stacks – AppResources and App1

Where **App Resources** and **App1**  where we have given the path which is stored in s3 bucket .





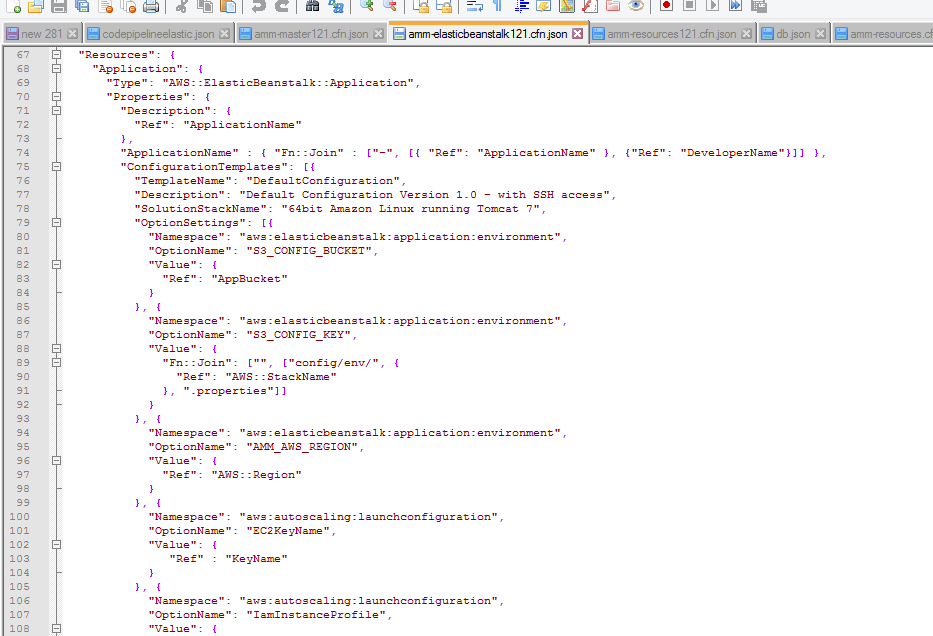
The AppResources resource in the parent template points to the amm-resources121.cfn.json in some S3 bucket, while the App1 resource points to the amm-elasticbeanstalk121.cfn.json template. The App1 resource depends on the AppResources template as we can see from the Fn::GetAtt in App1, so this means that CloudFormation will create the AppResources stack (i.e., the am-resources121.cfn.json template) first. This will create all of the dependencies our application requires, including RDS, DynamoDB, SQS, Security Groups, IAM Role, etc, and the values for each of these resources will be included in the stacks Outputs:



## Using Elastic Beanstalk with CloudFormation

Launching the amm-master121.cfn.json template causes the amm-resources121.cfn.json template to run, which creates all of the dependencies for our app. When it’s done, we’re finally ready to deploy and run our application code in Elastic Beanstalk. We model and define that Elastic Beanstalk environment in the amm-elasticbeanstalk121.cfn.json template file.

In the amm-elasticbeanstalk.cfn.json template we define both an Application and an Environment resource. This snippet shows the Application:



The Application’s Properties include:

ApplicationName: The name of the app as it will appear in the Elastic Beanstalk console

ConfigurationTemplates: Here we define one template named DefaultConfiguration. This defines configuration settings for an Environment within the Application.

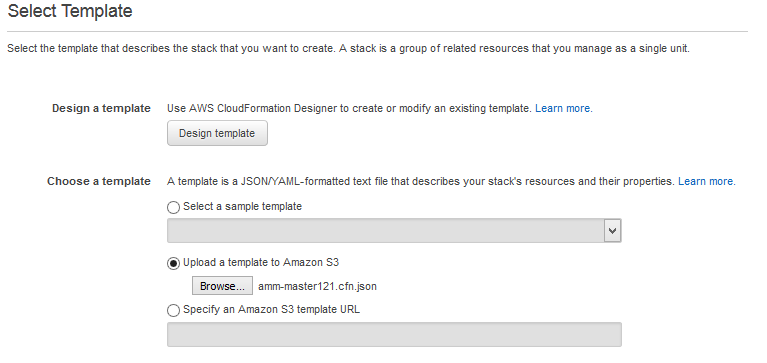
SolutionStackName: This property of the DefaultConfiguration templates means that any Environment launched with this template will be of type 64bit Amazon Linux running Tomcat 7

OptionSettings: This collection of settings will be applied to any Environment launched with the DefaultConfiguration template.

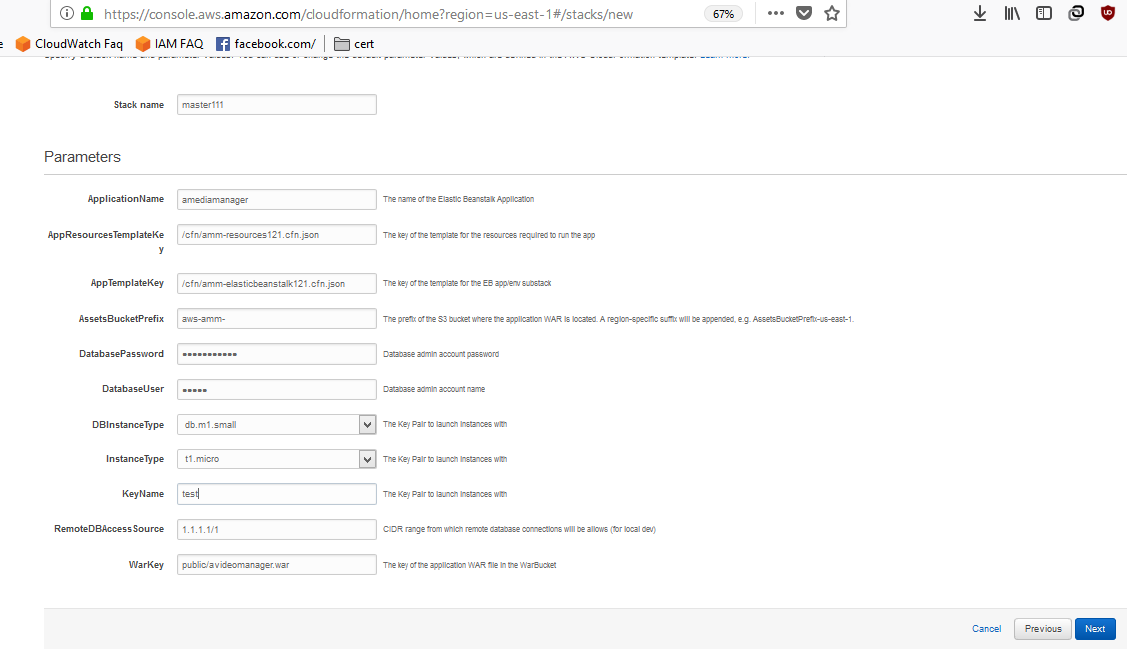
## The Result

After you deploy the amm-master121.cfn.json stack and it completes,

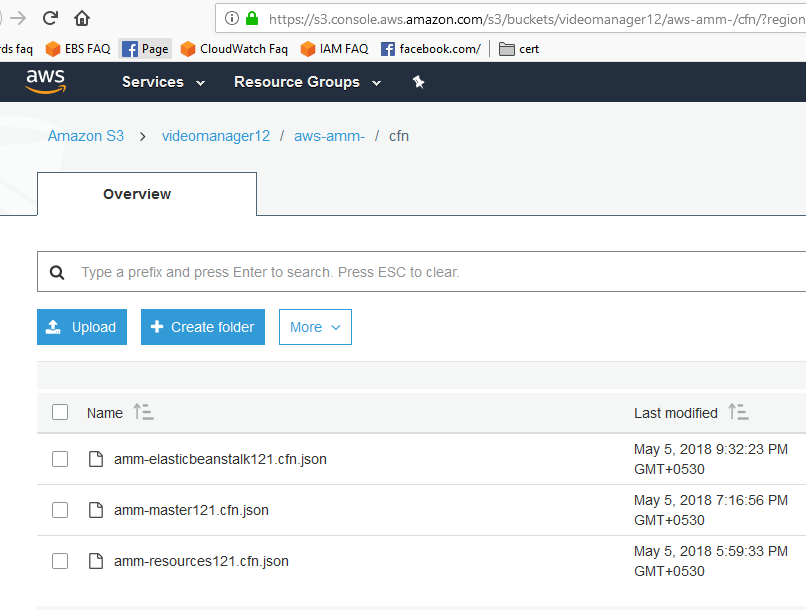
Step1 : upload a template amm-mster121.cfn.json in cloudofrmation.

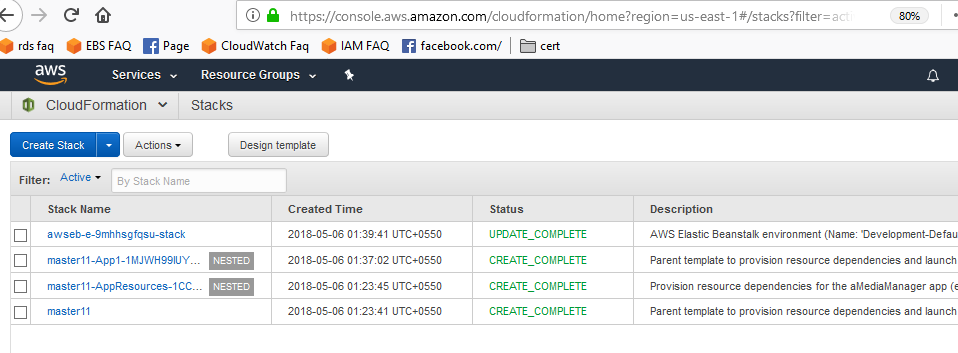


Step2 : We need to fill the parameters

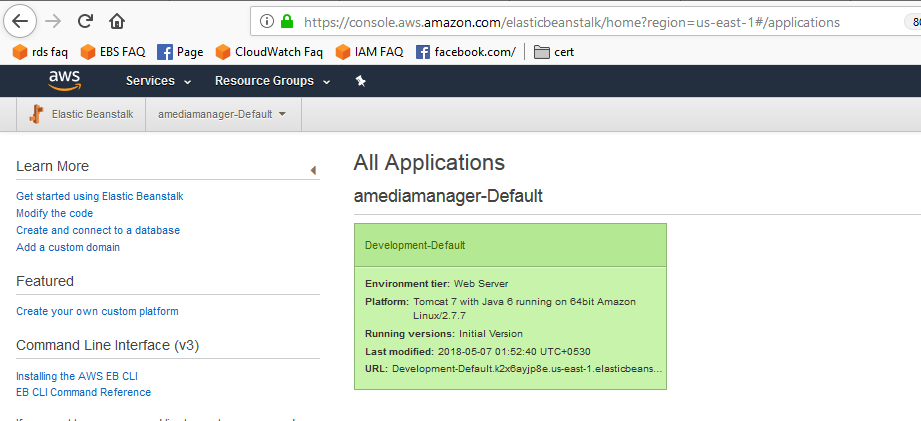


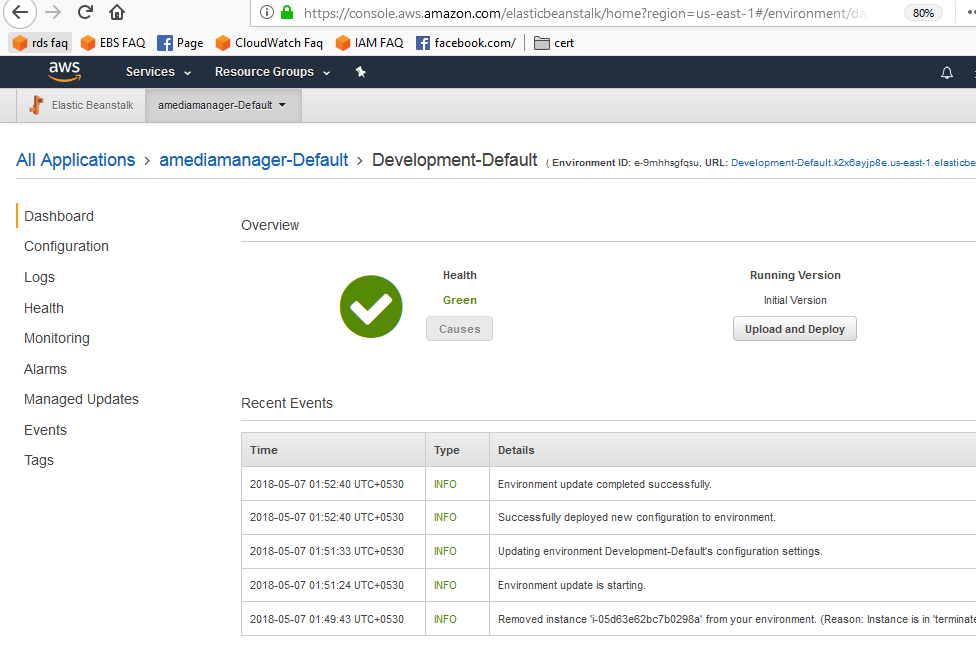
Step3 : Master will Launch both App Resources and App1 it will call from s3 bucket



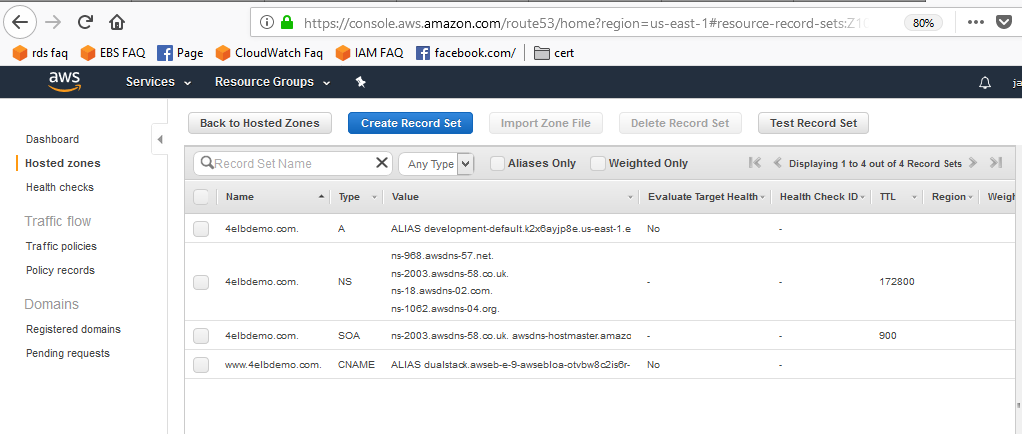


## Step 4 : Successfully launch in elasticbeanstalk url “[Development-Default.k2x6ayjp8e.us-east-1.elasticbeanstalk.com](http://Development-Default.k2x6ayjp8e.us-east-1.elasticbeanstalk.com)”

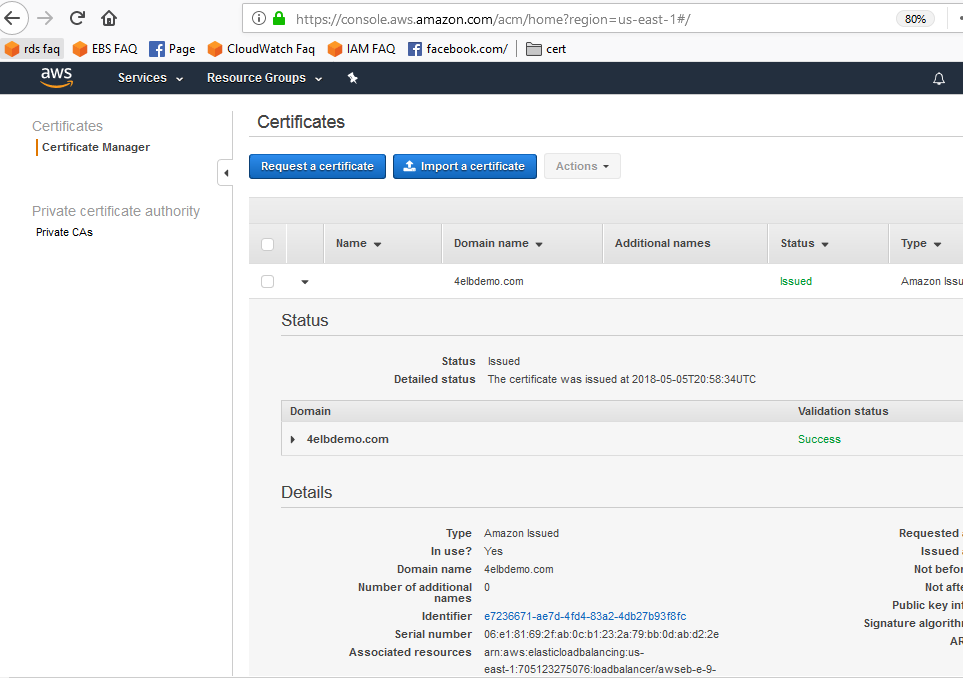


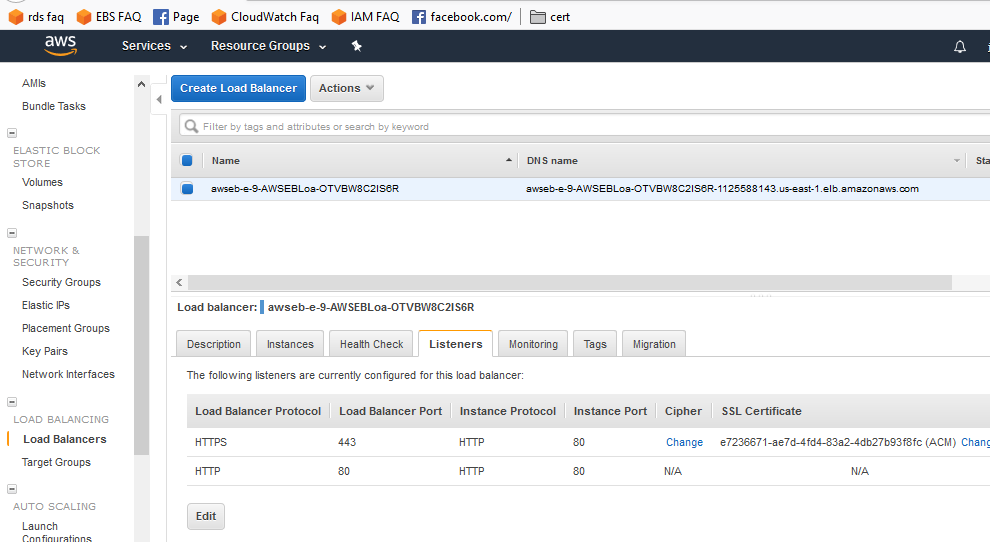


**Step 5** We are using Route 53 , and purchase the domain redirect to another domain “4ebldemo.com”

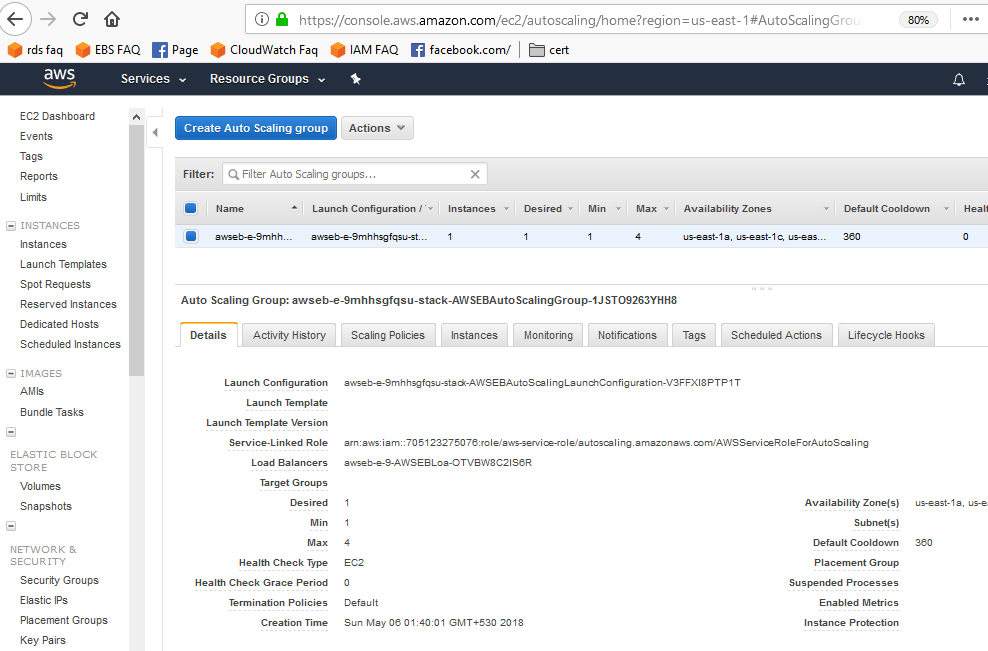


**Step6 :** Using AWS Certficate manager we have change to <http://4ebldemo.com> to <https://4ebldemo.com> and added in

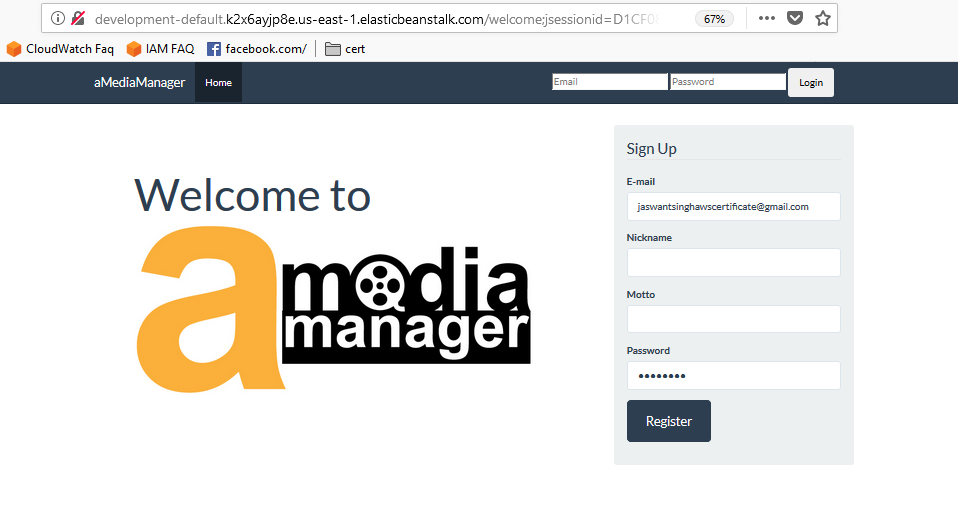


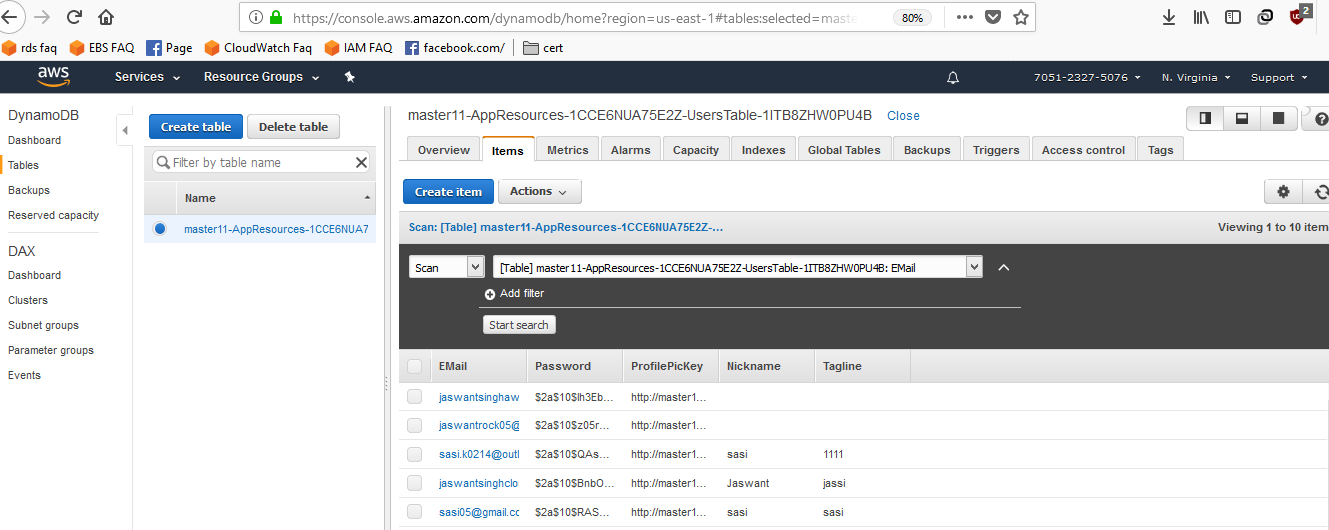


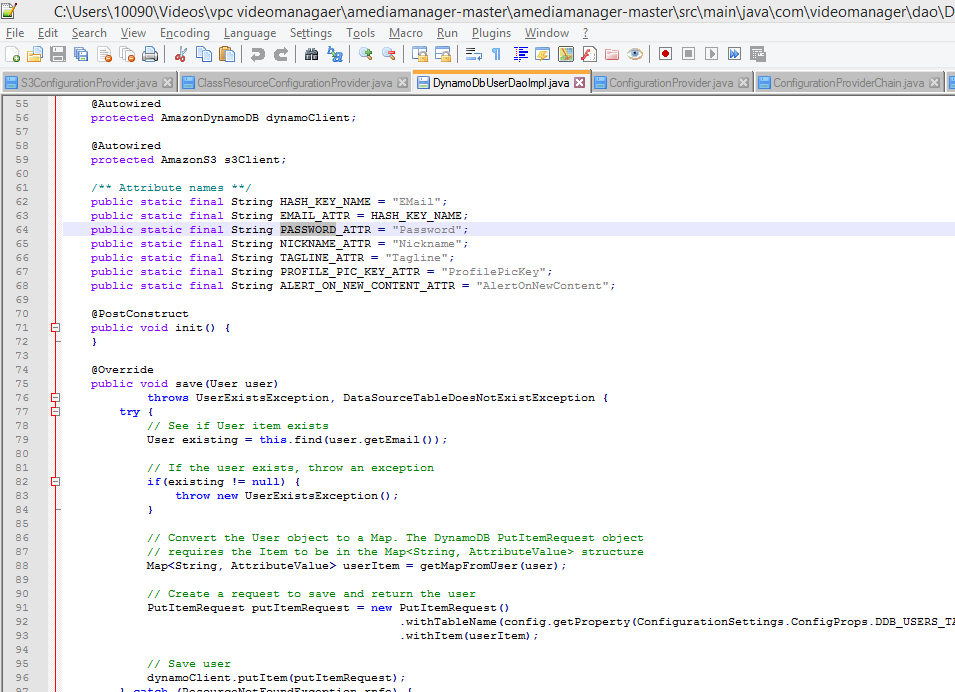
**Step7 :** We have added autoscaling in cloudformation stack



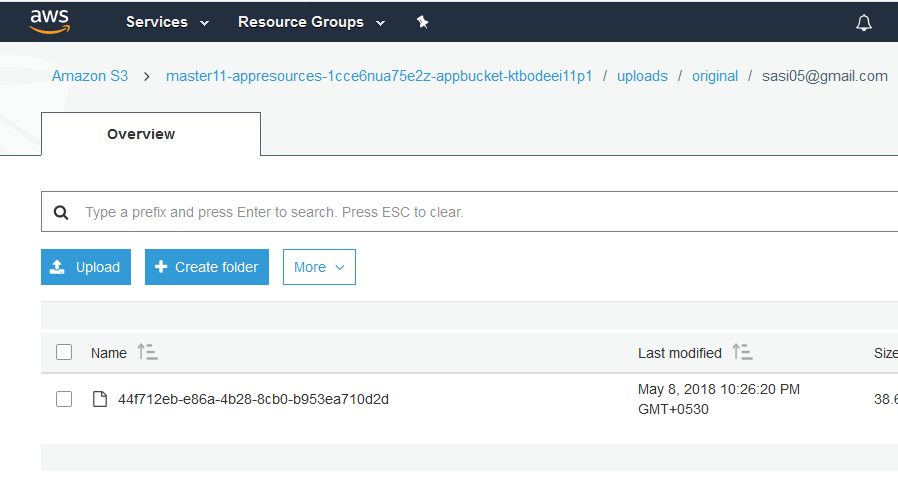
**Step8 :** At the login page of media manager the all Sign up data store in dynamo db with all detail , with encrypted password . Amazon DynamoDB for storing user profile information



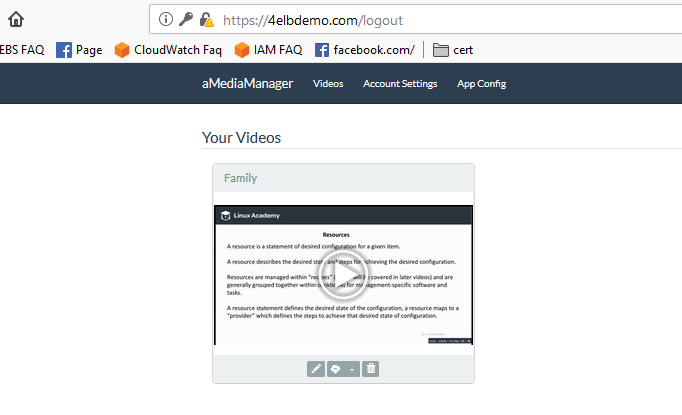
****

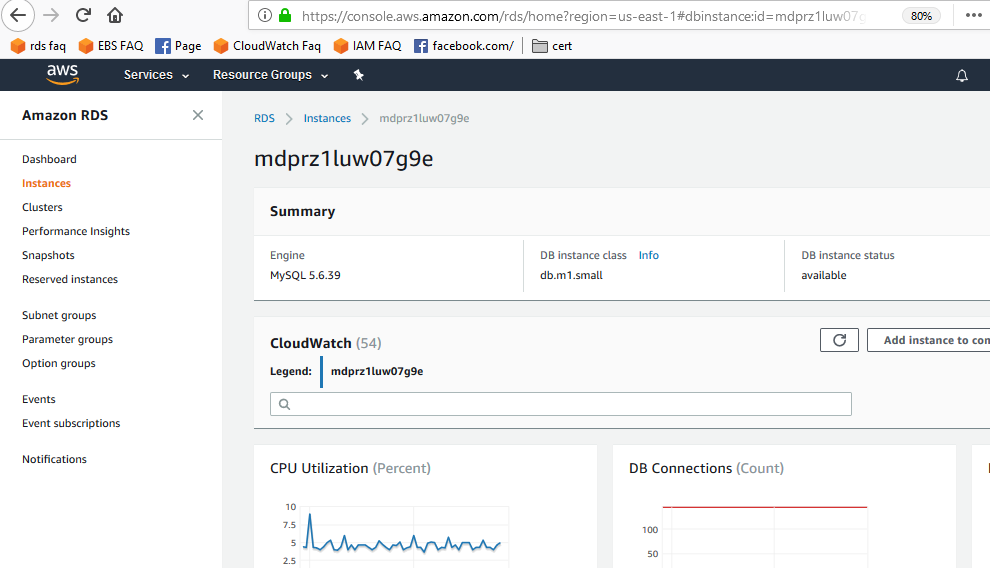


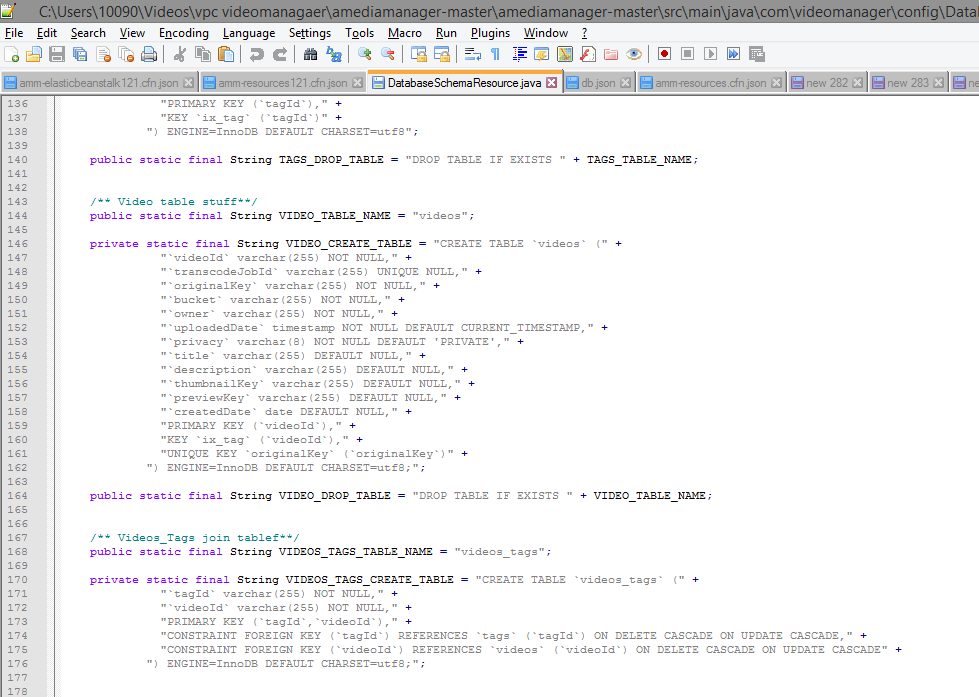
**Step9 :After login the media manager page we can upload the video , we can share the video , we can make private or public . it will stores in s3 bucket .**

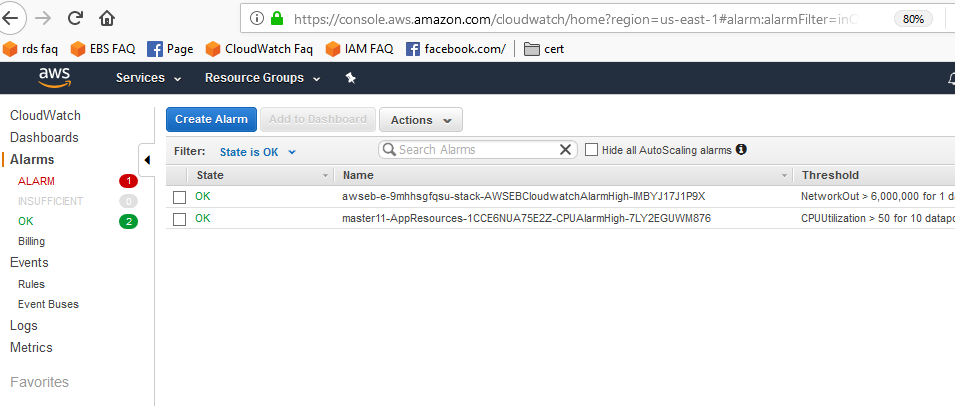


**Step 10** Rds is creating thumbnail of the video ,we have configure cpu in cloud watch Amazon RDS for storing searchable video metadata

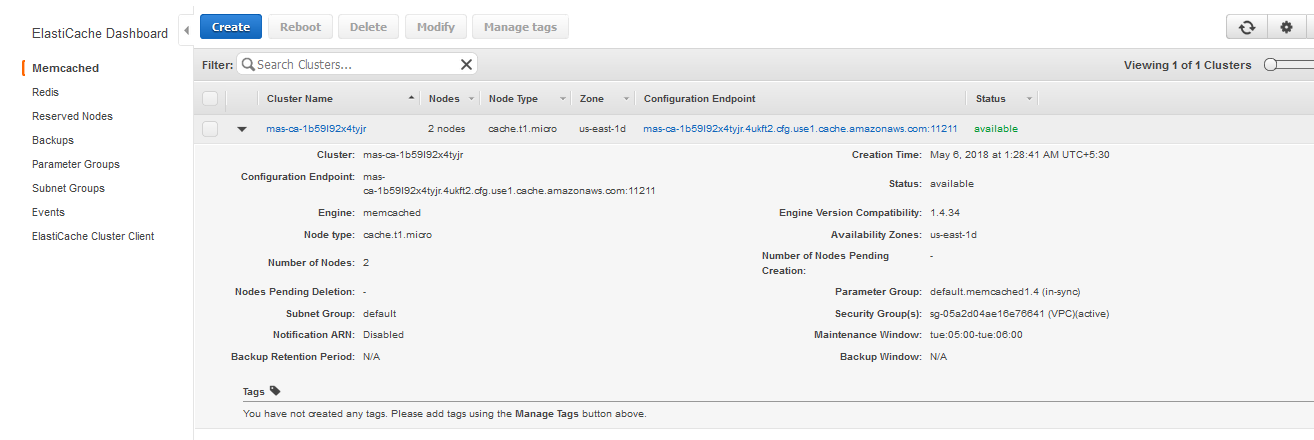








**Step11 : Amazon ElastiCache for caching database queries**



**Step 12 :Amazon Elastic Transcoder :**

Amazon Elastic Transcoder is media transcoding in [the cloud](https://aws.amazon.com/what-is-cloud-computing/). It is designed to be a highly scalable, easy to use and a cost effective way for developers and businesses to convert (or “transcode”) media files from their source format into versions that will playback on devices like smartphones, tablets and PCs.

Amazon Elastic Transcoder manages all aspects of the media transcoding process for you transparently and automatically. There’s no need to administer software, scale hardware, tune performance, or otherwise manage transcoding infrastructure. You simply create a transcoding “job” specifying the location of your source media file and how you want it transcoded. Amazon Elastic Transcoder also provides transcoding presets for popular output formats, which means that you don’t need to guess about which settings work best on particular devices.

