Wordpress Video Conversion application

BY : Aadrika Ishika Siddharth

We Have Implemented the 3 Tier Microservices Architecture Based Project.

Frontend: Wordpress

Backend: Terraform, Ansible and Lambda

Storage: RDS , S3

---

download and install terraform and ansible###########################

- curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo apt-key add -
- sudo apt-add-repository "deb [arch=amd64] https://apt.releases.hashicorp.com $(lsb_release -cs) main"
- sudo apt-get update && sudo apt-get install terraform
- apt install ansible -y

creating resourcess by terraform

- **ATTACH AmazonEC2FullAccess, AmazonElasticFileSystemFullAccess, AmazonRDSFullAccess role to your ec2**

- mkdir terraform-project
- cd terraform-project
- vim main.tf

```
#---------------------------------

#VPC

#---------------------------------

resource "aws_vpc" "Main" {

  cidr_block       = var.main_vpc_cidr
  instance_tenancy = "default"
  enable_dns_hostnames = true
  tags = {
   Name = "custom-vpc"
 }
}

resource "aws_internet_gateway" "IGW" {
   vpc_id =  aws_vpc.Main.id
}

resource "aws_subnet" "public1" {
  vpc_id =  aws_vpc.Main.id
  cidr_block = var.public_subnet1
  availability_zone = "us-east-1a"
 tags = {
   Name = "Public-Subnet-1"
 }
}

resource "aws_subnet" "public2" {
  vpc_id =  aws_vpc.Main.id
  cidr_block = var.public_subnet2
  availability_zone = "us-east-1b"
 tags = {
   Name = "Public-Subnet-2"
 }
}

resource "aws_subnet" "private1" {
  vpc_id =  aws_vpc.Main.id
  cidr_block = var.private_subnet1
  availability_zone = "us-east-1a"
 tags = {
   Name = "Private-Subnet-1"
 }
}

resource "aws_subnet" "private2" {
  vpc_id =  aws_vpc.Main.id
  cidr_block = var.private_subnet2
  availability_zone = "us-east-1b"
 tags = {
   Name = "Private-Subnet-2"
 }
}

resource "aws_route_table" "PublicRT" {
   vpc_id =  aws_vpc.Main.id
        route {
   cidr_block = "0.0.0.0/0"
   gateway_id = aws_internet_gateway.IGW.id
    }
}

resource "aws_route_table" "PrivateRT" {
  vpc_id = aws_vpc.Main.id
  route {
  cidr_block = "0.0.0.0/0"
  nat_gateway_id = aws_nat_gateway.NATgw.id
  }
}
```

```
 resource "aws_route_table_association" "PublicRTassociation1" {
   subnet_id = aws_subnet.public1.id
   route_table_id = aws_route_table.PublicRT.id
 }
resource "aws_route_table_association" "PublicRTassociation2" {
   subnet_id = aws_subnet.public2.id
   route_table_id = aws_route_table.PublicRT.id
 }

 resource "aws_route_table_association" "PrivateRTassociation1" {
   subnet_id = aws_subnet.private1.id
   route_table_id = aws_route_table.PrivateRT.id
 }

 resource "aws_route_table_association" "PrivateRTassociation2" {
   subnet_id = aws_subnet.private2.id
   route_table_id = aws_route_table.PrivateRT.id
 }
 resource "aws_eip" "nateIP" {
  vpc   = true
 }

 resource "aws_nat_gateway" "NATgw" {
  allocation_id = aws_eip.nateIP.id
  subnet_id = aws_subnet.public1.id
 }

# ----------------------
# Security Group
# ----------------------
```

```
resource "aws_security_group" "ec2" {
 name        = "EC2-sg"
 description = "Allow efs outbound traffic"
 vpc_id      = aws_vpc.Main.id
 ingress {
   cidr_blocks = ["0.0.0.0/0"]
   from_port = 22
   to_port = 22
   protocol = "tcp"
  }
  ingress {
   security_groups = [ aws_security_group.elb.id ]
   from_port = 80
   to_port = 80
   protocol = "tcp"
  }
  ingress {
   security_groups = [ aws_security_group.elb.id ]
   from_port = 443
   to_port = 443
   protocol = "tcp"
  }
 egress {
  from_port       = 0
  to_port         = 0
  protocol        = "-1"
  cidr_blocks     = ["0.0.0.0/0"]
 }
 tags = {
  Name = "EC2-sg"
  }
}

resource "aws_security_group" "efs" {
  name = "efs-sg"
  description= "Allows inbound efs traffic from ec2"
  vpc_id = aws_vpc.Main.id

  ingress {
    security_groups = [aws_security_group.ec2.id]
    from_port = 2049
    to_port = 2049
    protocol = "tcp"
  }

  egress {
    security_groups = [aws_security_group.ec2.id]
    from_port = 0
    to_port = 0
    protocol = "-1"
  }
  tags = {
   Name = "EFS-sg"
 }
 }

resource "aws_security_group" "rds" {
  name = "rds-sg"
  description= "Allows inbound RDS traffic from ec2"
  vpc_id = aws_vpc.Main.id
 ingress {
   security_groups = [aws_security_group.ec2.id]
   from_port = 3306
   to_port = 3306
   protocol = "tcp"
 }

 egress {
   security_groups = [aws_security_group.ec2.id]
   from_port = 0
   to_port = 0
   protocol = "-1"
 }
 tags = {
  Name = "RDS-sg"
 }
}

resource "aws_security_group" "elb" {
 name = "elb-sg"
 description= "Allows inbound elb traffic from route53"
 vpc_id = aws_vpc.Main.id

 ingress {
   cidr_blocks = [ "0.0.0.0/0" ]
   from_port = 443
   to_port = 443
   protocol = "tcp"
 }
 ingress {
   cidr_blocks = [ "0.0.0.0/0" ]
```

```
    from_port = 80
    to_port = 80
    protocol = "tcp"
  }

  egress {
    cidr_blocks = [ "0.0.0.0/0" ]
    from_port = 0
    to_port = 0
    protocol = "-1"
  }
  tags = {
   Name = "ELB-sg"
 }
}

#-----------------------------
#EFS
#----------------------------

resource "aws_efs_file_system" "efs" {
  creation_token = "efs"
  performance_mode = "generalPurpose"
  throughput_mode = "bursting"
  encrypted = "true"
 tags = {
    Name = "custom-efs"
  }
}
```

```
resource "aws_efs_mount_target" "efs-mt" {

   file_system_id  = aws_efs_file_system.efs.id
   subnet_id = aws_subnet.private1.id
   security_groups = [aws_security_group.efs.id]
 }

 resource "aws_efs_mount_target" "efs-mt1" {

   file_system_id  = aws_efs_file_system.efs.id
   subnet_id = aws_subnet.private2.id
   security_groups = [aws_security_group.efs.id]
 }

#--------------------------
#RDS
#-------------------------

resource "aws_db_instance" "default" {
  allocated_storage    = 30
  engine               = var.engine
  engine_version       = var.engine_version
  instance_class       = var.instance_class
  db_name              = var.name
  username             = var.username
  password             = var.password
  parameter_group_name = var.parameter_group_name
  db_subnet_group_name   = aws_db_subnet_group.default.name
  vpc_security_group_ids = [ aws_security_group.rds.id ]
  skip_final_snapshot       = true
```

```
}

resource "aws_db_subnet_group" "default" {
  name       = "main"
  subnet_ids = [aws_subnet.private1.id, aws_subnet.private2.id]

  tags = {
    Name = "DB-sg"
  }
}

#--------------------------
#EC2
#-------------------------
resource "aws_instance" "ec2" {
    ami = var.ami
    instance_type = var.instance_type
    subnet_id = aws_subnet.public1.id
    vpc_security_group_ids = [ aws_security_group.ec2.id ]
    key_name= "ab"
    associate_public_ip_address = true
    tags= {
        Name = "terraform_ec2"
    }
}

#--------------------------
#ELB
#-------------------------
resource "aws_elb" "classiclb" {
  name               = "classicelb"
```

```
 # availability_zones = ["us-east-1a", "us-east-1b"]
 subnets = [aws_subnet.public1.id, aws_subnet.public2.id]
 security_groups = [ aws_security_group.elb.id ]


listener {
   instance_port     = 80
   instance_protocol = "http"
   lb_port           = 80
   lb_protocol       = "http"
 }



 health_check {
   healthy_threshold   = 2
   unhealthy_threshold = 2
   timeout             = 3
   target              = "TCP:80"
   interval            = 10
 }

 instances                   = [aws_instance.ec2.id]
 cross_zone_load_balancing   = true
 idle_timeout                = 300
 connection_draining         = true
 connection_draining_timeout = 300

 tags = {
   Name = "classic-elb"
 }
```

```
}
```

- vim variables.tf

```
variable "ami" {}
variable "instance_type" {}
variable "main_vpc_cidr" {}
variable "public_subnet1" {}
variable "public_subnet2" {}
variable "private_subnet1" {}
variable "private_subnet2" {}
variable "engine" {}
variable "engine_version" {}
variable "instance_class" {}
variable "name"  {}
variable "username" {}
variable "password" {}
variable "parameter_group_name" {}
```

- vim terraform.tfvars

```
ami = "ami-04505e74c0741db8d"
instance_type = "t2.micro"
main_vpc_cidr = "10.0.0.0/16"
public_subnet1 = "10.0.0.0/24"
public_subnet2 = "10.0.2.0/24"
private_subnet1 = "10.0.1.0/24"
private_subnet2 = "10.0.3.0/24"
```

```
engine             = "mysql"
engine_version      = "5.7.37"
instance_class      = "db.t3.micro"
name               = "epam"
username           = "root"
password           = "root1234"
parameter_group_name = "default.mysql5.7"
```

- vim providers.tf

```
provider "aws" {
  region = "us-east-1"
}
```

- terraform init
- terraform plan
- terraform validate
- terraform apply

#########configuring wordpress via ansible

- come out of terraform directory
- mkdir ansible-play
- cd ansible-play
- vim key.pem   (copy paste the key here in .pem format of the host to which you want to connect)
- chmod 600 key.pem
- vim inventory.txt

```
node1 ansible_host=<publicorprivateipaddress of slave node launch from terraform>
ansible_ssh_private_key_file=/root/ansible-play/key.pem ansible_user=ubuntu
```

- cat > play-project.yaml

```
-
```

```
name: Installing wordpress using ansible

hosts: node1

become: true

tasks:

  - name: Update packagemanager

    shell: apt update -y

  - name: Install apache2

    apt: name=apache2 state=present

  - name: Install php

    apt: name=php state=present

  - name: python

    apt: name=python state=present

  - name: Install apache2-php5

    apt: name=libapache2-mod-php state=present

  - name: Install php-gd

    apt: name=php-gd state=present

  - name: Install php-mysql

    apt: name=php-mysql state=present

  - name: Install php-mbstring

    apt: name=php-mbstring state=present

  - name: Install php-xmlrpc

    apt: name=php-xmlrpc state=present

  - name: Install php-xml

    apt: name=php-xml state=present

  - name: Install php zip

    apt: name=php-zip state=present

  - name: Install unzip

    apt: name=unzip state=present

  - name: Install php-curl

    apt: name=php-curl state=present

  - name: Execute the command in remote shell; stdout goes to the specified file on the remote
```

```
    shell: rm -f index.html
  - name: delete file
    ignore_errors: yes
    file:
      state: absent
      path: /var/www/html/index.html
  - name: Download and Extract WorPress
    unarchive:
      src: https://wordpress.org/latest.tar.gz
      dest: /var/www/
      remote_src: yes
  - name: move contents of wordpress to the /var/www/html directory
    shell: mv /var/www/wordpress/* /var/www/html/
  - name: rewriting
    command: a2enmod rewrite
  - name: changing ownership on html directory
    command: chown -R www-data:www-data /var/www/html
  - name: setting correct permissions for wordpress files
    command: find /var/www/html -type d -exec chmod g+s {} \;
    command: chmod g+w /var/www/html/wp-content
    command: chmod -R g+w /var/www/html/wp-content/themes
    command: chmod -R g+w /var/www/html/wp-content/plugins
    command: cp /etc/apache2/sites-available/000-default.conf /etc/apache2/sites-
available/domain.com.conf
  - name: Inserting a line after a pattern in Ansible example
    lineinfile:
      path: /etc/apache2/sites-available/domain.com.conf
      line: ServerName domain.com
      line: ServerAlias www.domain.com
      insertafter: DocumentRoot /var/www/html
  - name: Enabling wordpress configuration file and disabling default conf file
    command: a2ensite domain.com.conf

    command: a2dissite 000-default.conf

  - name: Restart Apache

    service:

      name: apache2

      state: restarted
```

- ctrl+d
- ansible-playbook play-project.yaml -i inventory.txt

Come to AWS console

- **now go to route53**
- >>hosted domain
- >>create record
- >> create "A" record(don,t give any record name)
- >> select alias to classic load balancer
- >> selct your classicLB which got created by terraform
- >>click create

- **now got to certificate manager**
- >>request certificate
- >>give domain name(domainname, www.domainname, *.domainame)
- >>select dns resolution
- >>create after this click view certificate
- >> create route53 records
- >>create(this is for validation dns is owned by us)
- after 2-5 minutes you can see dns certificate got verfified

- **now come to your classic load balancer which got created by terraform**
- >>listener
- >>edit
- >>add HTTPS and add your ssl certificate

- **now come to wordpress ec2(terraform_ec2) created via terraform**
- >>security group
- >>edit inbound rules
- >> for HTTP,HTTPS select source as ELB-sg

- **now browse dns**
- >>you will see wordpress page

- >>login with following credential as they are mentioned in terraform.tfvarsfile

- ✓ DB NAME: epam
- ✓ USERNAME: root
- ✓ PASSWORD: root1234
- ✓ DATBASE: <endpoint of RDS install by terraform>

- now in next step give your login username and password of your choice
- after configuring wordpress
- install and activate following plugins

1. really simple ssl

2. wp offload media lite

- create s3 bucket and make it publicalyy accesible
- create iam role for s3fullaccess and attach to the wordpress server ec2
- click settings
- >>offload media lite
- >> my server is on aws and i would use IAM roles
- >>next
- >>(keep setting default and scroll down in advance setting select delete file from server)
- now click on post on left pane
- >>add new post
- >> click "+" button choose a image and upload and publish

Increasing Video File Size uploading

- cd /etc/php/7.4/apache
- ls
- vim php.ini
- search /upload there..add 3 line there

- upload_max_filesize = 250M
- post_max_size = 300M
- memory_limit = 2G
  :wq

- service apache2 restart

configuring lambda function to convert video format

- In wordpress name folderwhere you have to upload video as: videos/
- create folder in s3 bucket named: converted-videos
- go to SNS create 2 topic - error and complete using standard type
- also under access policy make publisher and subscriber everyone and then also create subscription using mail id for both
- confirm subscription in mail for both

- **now go to elastic transcoder service**
- click create new pipeline
- give name
- select your bucket which you have created previously
- select create console default role
- further for all option select your bucket created previously and storage as standard
- under notification service in completion and error event use existing sns topic
- which we previously created-complete and error respectively
- create
- copy pipeline id(will be used in environmental variable of lambda inside configuration section)   ###1650877539997-9vot77

- now go to IAM role and create role for LAmbda(select lambda service under role not ec2) with following access

1. AmazonElasticTranscoder_FullAccess
2. AmazonS3FullAccess
3. CloudWatchFullAccess
4. AmazonSNSFullAccess

- **Now come to lambda service and create a function**
- select "author from scratch"..nodejsx12
- select"use existing role"
- select role created in previous step
- create

- now scroll down add following code in index.js

```
'use strict';

var AWS = require('aws-sdk'),

    transcoder = new AWS.ElasticTranscoder({

        apiVersion: '2012-09-25',
        region: 'us-east-1'
    });
exports.handler = (event, context, callback) => {
    let fileName = event.Records[0].s3.object.key;
    var srcKey =  decodeURIComponent(event.Records[0].s3.object.key.replace(/\+/g, " "));
    var newKey = fileName.split('.')[0];
    console.log('New video has been uploaded:', fileName);
transcoder.createJob({
     PipelineId: process.env.PIPELINE_ID,
     Input: {
      Key: srcKey,
      FrameRate: 'auto',
      Resolution: 'auto',
      AspectRatio: 'auto',
      Interlaced: 'auto',
      Container: 'auto'
     },
     Output: {
      Key: getOutputName(fileName),
      ThumbnailPattern: '',
      PresetId: '1351620000001-000050',
      Rotate: 'auto'
     }
    }, function(err, data){
        if(err){
            console.log('Something went wrong:',err)
        }else{
            console.log('Converting is done');
        }
     callback(err, data);
    });
```

```
};
function getOutputName(srcKey){
 let baseName = srcKey.replace('videos/','');
 let withOutExtension = removeExtension(baseName);
 return 'converted-videos/' + withOutExtension + '.mp4';
}
function removeExtension(srcKey){
   let lastDotPosition = srcKey.lastIndexOf(".");
   if (lastDotPosition === -1) return srcKey;
   else return srcKey.substr(0, lastDotPosition);
}
```

- change region,preset-id, source folder name, destination folder name in code

1351620000001-000030	#480 4:3 pixel

- and click "deploy" button
- now go to configuration tab
- in left pane you can see option of environment variable, where you can add you pipeline id, in following format

PIPELINE_ID	<value of id>

- Now click "Add trigger" presented above
- select S3
- select bucketname
- (prefix and suffix are optional, don't do anything)
- now click create

➢ now upload video from wordpress less than 2MB(in left pane you can see media option) and then check elastictransocder status, it should be completed. also u can see cloudwatch logs and s3