AWS Project Solution

Description

The project Blog Page Application aims to deploy blog application as a web application written Django Framework on AWS Cloud Infrastructure.

Solution

Step 1: Create dedicated VPC and whole components

- VPC
    - Create VPC, named `Blog-VPC` CIDR blok is `10.0.0.0/16`
    - select `Blog-VPC` VPC, click `Actions` and `enable DNS hostnames` for the `Blog-VPC`.
- Create Subnets
    - Create a public subnet named `Blog-public-subnet-1A` under the vpc Blog-VPC in AZ us-east-1a with 10.0.10.0/24
    - Create a private subnet named `Blog-private-subnet-1A` under the vpc Blog-VPC in AZ us-east-1a with 10.0.11.0/24
    - Create a public subnet named `Blog-public-subnet-1B` under the vpc Blog-VPC in AZ us-east-1b with 10.0.20.0/24
    - Create a private subnet named `Blog-private-subnet-1B` under the vpc Blog-VPC in AZ us-east-1b with 10.0.21.0/24
- Set `auto-assign IP` up for public subnets. Select each public subnets and click Modify "auto-assign IP settings" and select "Enable auto-assign public IPv4 address"
- Click Internet gateway section on left hand side. Create an internet gateway named `Blog-IGW` and create.
- ATTACH the internet gateway `Blog-IGW` to the newly created VPC `Blog-VPC`. Go to VPC and select newly created VPC and click action ---> Attach to VPC ---> Select `Blog-VPC` VPC
- Route Table
    - Go to route tables on left hand side. We have already one route table as main route table. Change it's name as `Blog-public-RT`
    - Create a route table and give a name as `Blog-private-RT`.
    - Add a rule to `Blog-public-RT` in which destination 0.0.0.0/0 (any network, any host) to target the internet gateway `Blog-IGW` in order to allow access to the internet.
    - Select the private route table, come to the subnet association subsection and add private subnets to this route table. Similarly, we will do it for public route table and public subnets.
- Endpoint

    - Go to the endpoint section on the left hand menu
    - select endpoint
    - click create endpoint
    - service name : `com.amazonaws.us-east-1.s3`
    - VPC : `Blog-VPC`
    - Route Table : private route tables
    - Policy : `Full Access`
    - Create

Step 2: Create Security Groups.

1. ALB Security Group
2. EC2 Security Groups
3. RDS Security Groups
4. NAT Instance Security Group

Step 3: Create RDS

- Create a subnet group for our custom VPC. Click `subnet Groups` on the left hand menu and click `create DB Subnet Group`
- Select 2 Private Subnets in these subnets

Step 4: Create two S3 Buckets and set one of these as static website.

Go to the S3 Consol and lets create two buckets.

1. Blog Website's S3 Bucket

- Click Create Bucket

2. S3 Bucket for failover scenario

- Click Create Bucket
- Selects created `www.<YOUR DNS NAME>` bucket ---> Properties ---> Static website hosting
- Select `www.<YOUR DNS NAME>` bucket ---> select Upload and upload `index.html` and `sorry.jpg` files from given folder---> Permissions ---> Grant public-read access ---> Checked warning massage

Step 5: Copy files downloaded or cloned from `Blog` repo on Github

Step 6: Prepair your Github repository

- Create private project repository on your Github and clone it on your local. Copy all files and folders which are downloaded from our repo under this folder. Commit and push them on your private Git hup Repo.

Step 7: Prepare a userdata to be utilized in Launch Template

- Launch Template

```
#!/bin/bash
apt-get update -y
apt-get install git -y
apt-get install python3 -y
cd /home/ubuntu/
TOKEN=""
git clone https://$TOKEN@github.com/<USER_NAME>/<REPO-NAME>.git
cd /home/ubuntu/<REPO_NAME>
apt install python3-pip -y
apt-get install python3.7-dev libmysqlclient-dev -y
pip3 install -r requirements.txt
cd /home/ubuntu/<REPO_NAME>/src
python3 manage.py collectstatic --noinput
python3 manage.py makemigrations
python3 manage.py migrate
python3 manage.py runserver 0.0.0.0:80
```

Step 8: Write RDS database endpoint and S3 Bucket name in settings file push your own public repo on Github

- Write RDS and S3 Bucket Variables into Settings.py
- Check if this userdata is working or not. to do this create new instance in public subnet and check if it is working

Step 9: Create NAT Instance in Public Subnet

- To launch NAT instance, go to the EC2 console and click the create button.
- Select newly created NAT instance and enable stop source/destination check
- Go to private route table and write a rule

Step 10: Create Launch Template and IAM role for it

- Go to the IAM role console click role on the right hand menu than create role with `AmazonS3FullAccess policy`
- To create Launch Template, go to the EC2 console and select `Launch Template` on the left hand menu. Tab the Create Launch Template button.

```
Launch template name               : Blog_launch_template
Template version description       : Blog Web Page
Amazon machine image (AMI)         : Ubuntu 18.04
Instance Type                      : t2.micro
Key Pair                           : key.pem
Network Platform                   : VPC
Security Groups                    : Blog_EC2_sec_group
```

```
Storage (Volumes)                    : keep it as is
Resource tags                        : Key: Name   Value: Blog_web_server
Advance Details:
    - IAM instance profile           : Blog_EC2_S3_Full_Access
    - Termination protection         : Enable
    - User Data
                                     #!/bin/bash
                                     apt-get update -y
                                     apt-get install git -y
                                     apt-get install python3 -y
                                     cd /home/ubuntu/
                                     TOKEN=""
                                     git clone
https://$TOKEN@github.com/<USER_NAME>/<REPO-NAME>.git
                                     cd /home/ubuntu/<REPO_NAME>
                                     apt install python3-pip -y
                                     apt-get install python3.7-dev libmysqlclient-
dev -y
                                     pip3 install -r requirements.txt
                                     cd /home/ubuntu/<REPO_NAME>/src
                                     python3 manage.py collectstatic --noinput
                                     python3 manage.py makemigrations
                                     python3 manage.py migrate
                                     python3 manage.py runserver 0.0.0.0:80
                                     - create launch template
```

Step 11: Create certification for secure connection

Step 12: Create ALB and Target Group

- Go to the Load Balancer section on the left hand side menu of EC2 console. Click `create Load Balancer` button and select Application Load Balancer
- click create
- To redirect traffic from HTTP to HTTPS, go to the ALB console and select Listeners sub-section.

Step 13: Create Autoscaling Group with Launch Template

- Go to the Autoscaling Group on the left hand side menu. Click create Autoscaling group.

Step 14: Create Cloudfront in front of ALB

Go to the cloudfront menu and click start

- Origin Settings

```
Origin Domain Name          :Blog-ALB-1947210493.us-east-2.elb.amazonaws.com
Origin Path                 : Leave empty (this means, define for root '/')
Protocol                    : Match Viewer
HTTP Port                   : 80
```

```
HTTPS                        : 443
Minimum Origin SSL Protocol : Keep it as is
Name                         : Keep it as is
Add custom header            : No header
Enable Origin Shield         : No
Additional settings          : Keep it as is
```

Default Cache Behavior Settings

```
Path pattern                                  : Default (*)
Compress objects automatically                : Yes
Viewer Protocol Policy                        : Redirect HTTP to HTTPS
Allowed HTTP Methods                          : GET, HEAD, OPTIONS, PUT, POST,
PATCH, DELETE
Cached HTTP Methods                           : Select OPTIONS
Cache key and origin requests
- Use legacy cache settings
  Headers     : Include the following headers
    Add Header
    - Accept
    - Accept-Charset
    - Accept-Datetime
    - Accept-Encoding
    - Accept-Language
    - Authorization
    - Cloudfront-Forwarded-Proto
    - Host
    - Origin
    - Referrer
Forward Cookies                           : All
Query String Forwarding and Caching     : All
Other stuff                               : Keep them as are
```

- Distribution Settings

```
Price Class                               : Use all edge locations (best
performance)
Alternate Domain Names                    : www.<domain.name>.com
SSL Certificate                           : Custom SSL Certificate (example.com) ---
> Select your certificate creared before
Other stuff                               : Keep them as are
```

Step 15: Create Route 53 with Failover settings

- Come to the Route53 console and select Health checks on the left hand menu. Click create health check
- Configure health check

```
Name                : Blog health check
What to monitor     : Endpoint
Specify endpoint by : Domain Name
Protocol            : HTTP
Domain Name         : Write cloudfront domain name
Port                : 80
Path                : leave it blank
Other stuff         : Keep them as are
```

- Click Hosted zones on the left hand menu
- click your Hosted zone :
- Create Failover scenario
- Click Create Record
- Select Failover ---> Click Next

```
Configure records
Record name             : www.<YOUR DNS NAME>
Record Type             : A - Routes traffic to an IPv4 address and some AWS
resources
TTL                     : 300

---> First we'll create a primary record for cloudfront

Failover record to add to your DNS ---> Define failover record

Value/Route traffic to  : Alias to cloudfront distribution
                          - Select created cloudfront DNS
Failover record type    : Primary
Health check            : aws capstone health check
Record ID               : Cloudfront as Primary Record
-------------------------------------------------------------

---> Second we'll create secondary record for S3

Failover another record to add to your DNS ---> Define failover record

Value/Route traffic to  : Alias to S3 website endpoint
                          - Select Region
                          - Your created bucket name emerges ---> Select it
Failover record type    : Secondary
Health check            : No health check
Record ID               : S3 Bucket for Secondary record type
```

- click create records

Step 16: Create DynamoDB Table Go to the Dynamo Db table and click create table button

- Create DynamoDB table

Step 17: Create Lambda function -------> Create Role

- Before we create our Lambda function, we should create IAM role that we'll use for Lambda function. Go to the IAM console and select role on the left hand menu, then create role button

```
LambdaS3fullaccess,
Network Administrator
DynamoDBFullAccess
```

- go to the Lambda Console and click create function

Step 18: Create S3 Event and set it as trigger for Lambda Function

Go to the S3 console and select the S3 bucket named `Blogcbucket-name`.

- Go to the properties menu ---> Go to the Event notifications part
- Go to the code part and select lambda_function.py ---> remove default code and paste a code on below. If you give DynamoDB a different name, please make sure to change it into the code.

```
import json
import boto3

def lambda_handler(event, context):
    s3 = boto3.client("s3")

    if event:
        print("Event: ", event)
        filename = str(event['Records'][0]['s3']['object']['key'])
        timestamp = str(event['Records'][0]['eventTime'])
        event_name = str(event['Records'][0]['eventName']).split(':')[0][6:]

        filename1 = filename.split('/')
        filename2 = filename1[-1]

        dynamo_db = boto3.resource('dynamodb')
        dynamoTable = dynamo_db.Table('change me!!!!!')

        dynamoTable.put_item(Item = {
            'id': filename2,
            'timestamp': timestamp,
            'Event': event_name,
        })

    return "Lambda success"
```

- Click deploy and all set. go to the website and add a new post with photo, then control if their record is written on DynamoDB.