

IS698 Cloud Computing using AWS

PhotoSnap - Snapchat Clone

Under the guidance of Prof Samson Oni

Ankit Kumar Nath
LC46377

Table Of Contents

1. Introduction.....	3
2. Methodology.....	3
3. Architecture.....	3
4. Code/Output.....	4
4.1 Terraform.....	4
4.2 CloudFormation.....	7
4.3 Logger.....	11
4.4 EC2.....	12
4.5 User Testing.....	15
5. Challenges faced and solutions implemented.....	21
6. Future Works.....	21

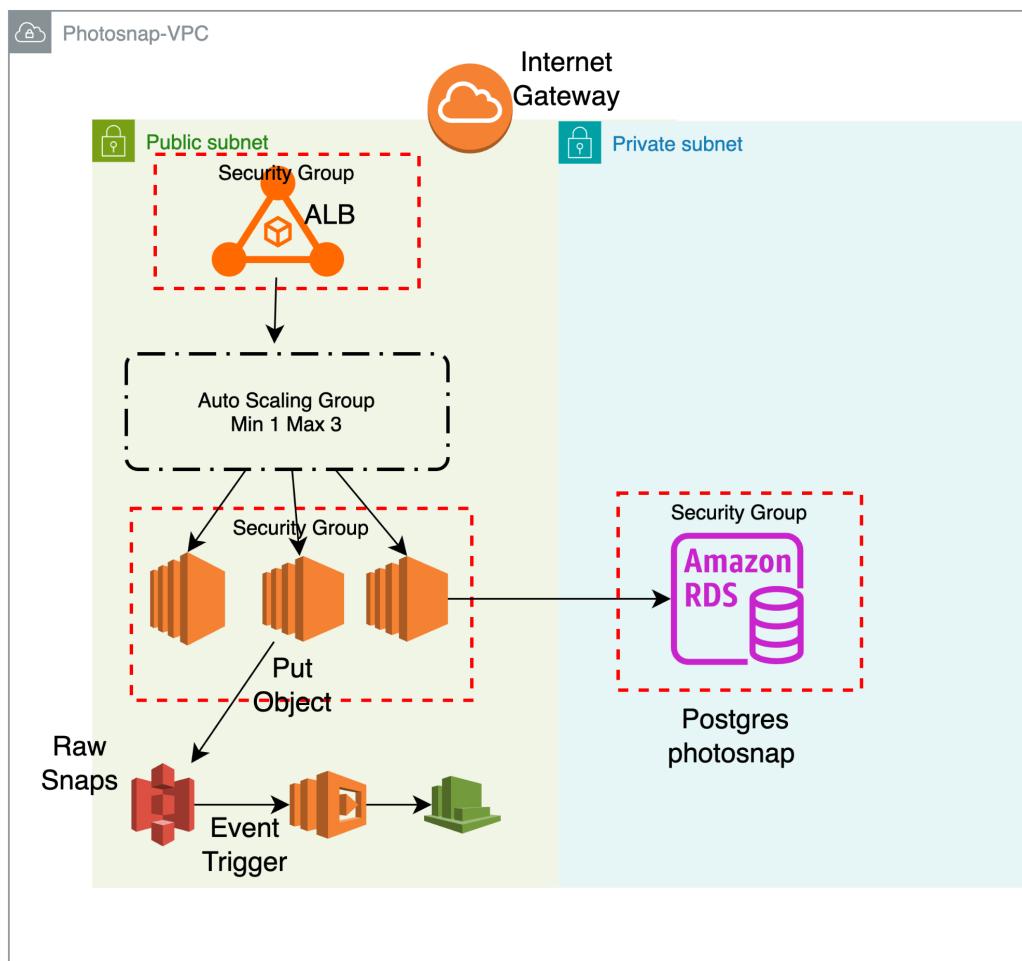
1. Introduction

Photosnap is a snapchat clone, where we have tried to replicate the core part of how snapchat works. A user would be able to login and signup into our app. The user will be able to choose an image and send it to other multiple users. The user can also view images that other users have sent. This was achieved through terraform, cloudformation and various **AWS** resources.

2. Methodology

The project is structured in such a way that we first setup the infrastructure through our terraform code, and then move onto the cloudformation for launching our **EC2** instances with the **Auto Scaling** feature. We also have a logging service which is achieved through **AWS Lambda** and can be viewed with help of **AWS Cloudwatch**. We will review the architecture followed by the implementation of the code.

3. Architecture

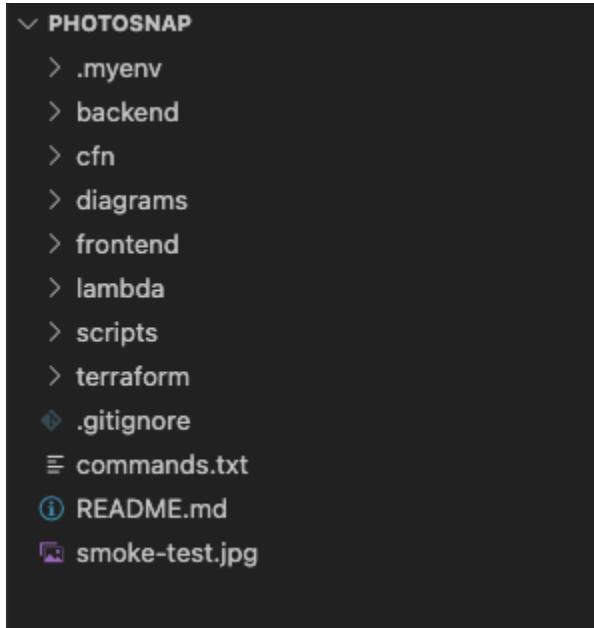


4. Code/Output

We start by setting up the project, we clone the [PhotoSnap](#) github repo. We can see the folder structure.

4.1 Terraform

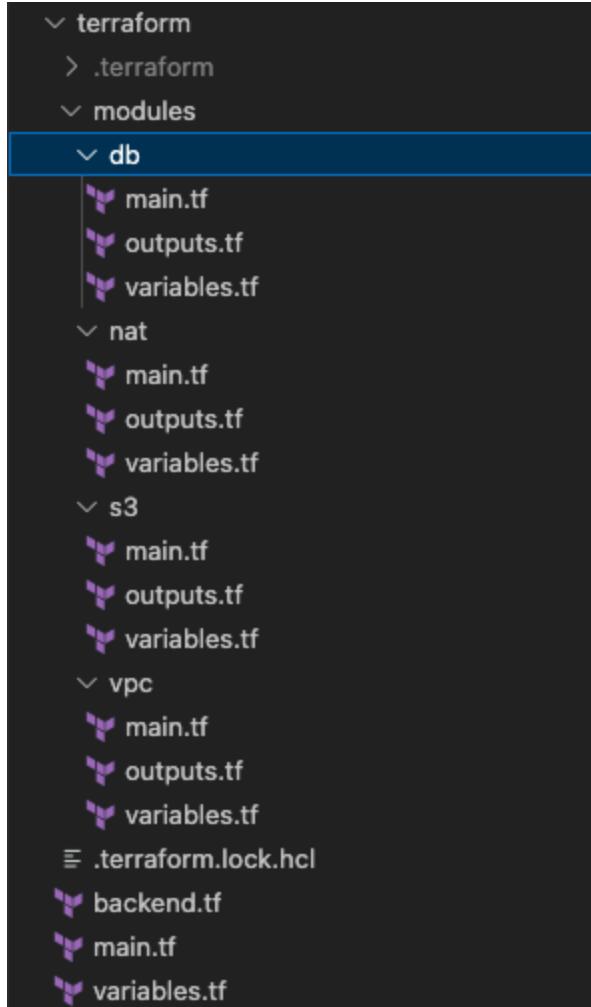
As we can see the whole project is divided into sub folders, where we have separate folders for backend, cloudformation, frontend, lambda, scripts and terraform.



Let's start with setting up the Infrastructure for this project.

We start with terraform. We have a modules folder where we have the database, the s3 bucket (for storing the terraform state, the snap raw images, the lambda logger code), the vpc setup and the NAT gateway for instances hosted on private subnets.

We have two private and public subnets in our VPC.



Once we have everything setup, we run the below command. The **db_password** has the password for our **RDS** and the variable **enable_nat** enables or disables the **NAT** gateway depending if we want NAT or not based on the environment.

Shell

```
terraform apply \
  -var="db_password=password" \
  -var="enable_nat=false" \
  -auto-approve
```

```

module.db.aws_db_instance.snapshots: Still creating... [5m0s elapsed]
module.db.aws_db_instance.snapshots: Still creating... [5m18s elapsed]
module.db.aws_db_instance.snapshots: Still creating... [5m28s elapsed]
module.db.aws_db_instance.snapshots: Still creating... [5m38s elapsed]
module.db.aws_db_instance.snapshots: Still creating... [5m48s elapsed]
module.db.aws_db_instance.snapshots: Still creating... [5m58s elapsed]
module.db.aws_db_instance.snapshots: Still creating... [6m0s elapsed]
module.db.aws_db_instance.snapshots: Still creating... [6m18s elapsed]
module.db.aws_db_instance.snapshots: Creation complete after 6m27s [id=db-XFCLKOK7FQKWNMJJX0V5XHCWQ]

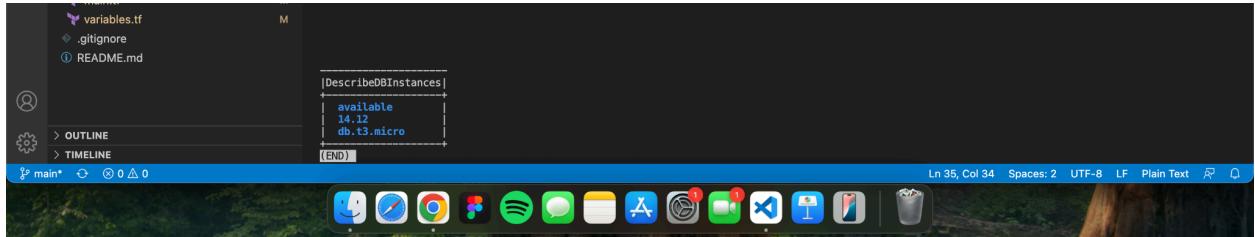
Apply complete! Resources: 16 added, 0 changed, 0 destroyed.

Outputs:

bucket_name = "photosnap-raw-snaps-4490772a"
db_endpoint = "photosnap-db.cktgkiokq2qf.us-east-1.rds.amazonaws.com"
private_subnet_ids =
  ["subnet-0ca19168146a01fb2",
  "subnet-00b27aecdc21ea1a",
]
public_subnet_ids = "subnet-013441305b454278b,subnet-002a69f372656adc"
vpc_id = "vpc-0ac3731a683126b80"
~Desktop/MS/ISE98/photosnap/terraform main>
> []

```

7m 3s 15:24:19



Your VPCs (2) <small>Info</small>						
Last updated less than a minute ago 🕒 Actions Create VPC						
	Name	VPC ID	State	Block Public...	IPv4 CIDR	IPv6 CIDR
□	-	vpc-05337bcae3a3cb2d4	Available	<input type="radio"/> Off	172.31.0.0/16	-
□	photosnap-vpc	vpc-0ac3731a683126b80	Available	<input type="radio"/> Off	10.0.0.0/16	-

Subnets (10) <small>Info</small>						
Last updated less than a minute ago 🕒 Actions Create subnet						
	Name	Subnet ID	State	VPC	Block Public...	IPv4 CIDR
□	private-10.0.4.0/24	subnet-00b27aecdc21ea1a	Available	vpc-0ac3731a683126b80 photosnap-vpc	<input type="radio"/> Off	10.0.4.0/24
□	-	subnet-0bf0fc268f5cae9e1b	Available	vpc-05337bcae3a3cb2d4	<input type="radio"/> Off	172.31.32.0/20
□	-	subnet-0eb432a1e91d22fd2	Available	vpc-05337bcae3a3cb2d4	<input type="radio"/> Off	172.31.0.0/20
□	-	subnet-0916bfb996342c33	Available	vpc-05337bcae3a3cb2d4	<input type="radio"/> Off	172.31.48.0/20
□	-	subnet-088482eba53dac4e8	Available	vpc-05337bcae3a3cb2d4	<input type="radio"/> Off	172.31.80.0/20
□	-	subnet-0fc250c1c259c9c7e	Available	vpc-05337bcae3a3cb2d4	<input type="radio"/> Off	172.31.16.0/20
□	public-10.0.1.0/24	subnet-013441305b454278b	Available	vpc-0ac3731a683126b80 photosnap-vpc	<input type="radio"/> Off	10.0.1.0/24
□	private-10.0.3.0/24	subnet-0ca19168146a01fb2	Available	vpc-0ac3731a683126b80 photosnap-vpc	<input type="radio"/> Off	10.0.3.0/24
□	-	subnet-0bc0cfafce0f03c5	Available	vpc-05337bcae3a3cb2d4	<input type="radio"/> Off	172.31.64.0/20
□	public-10.0.2.0/24	subnet-002a69f372656adc7	Available	vpc-0ac3731a683126b80 photosnap-vpc	<input type="radio"/> Off	10.0.2.0/24

Internet gateways (2) <small>Info</small>						
Last updated less than a minute ago 🕒 Actions Create internet gateway						
	Name	Internet gateway ID	State	VPC ID	Owner	
□	photosnap-igw	igw-04d3dcfe80f64164f	Attached	vpc-0ac3731a683126b80 photosnap-vpc	537124959881	
□	-	igw-0a0862bf7cb4c3f22	Attached	vpc-05337bcae3a3cb2d4	537124959881	

Databases (1)									
Group resources 🕒 Actions Modify Create database									
	DB identifier	Status	Role	Engine	Region ...	Size	Recommendations	CPU	
□	photosnap-db	Available	Instance	PostgreSQL	us-east-1a	db.t3.micro		6.5	

Name	AWS Region	IAM Access Analyzer	Creation date
cf-templates-c4yvxf2va155-us-east-1	US East (N. Virginia) us-east-1	View analyzer for us-east-1	April 9, 2025, 21:52:35 (UTC-04:00)
is698	US East (Ohio) us-east-2	View analyzer for us-east-2	February 5, 2025, 21:56:39 (UTC-05:00)
lambda-trigger-lab-lc46377	US East (N. Virginia) us-east-1	View analyzer for us-east-1	April 17, 2025, 16:25:14 (UTC-04:00)
lc46377-s3-bucket-example	US East (N. Virginia) us-east-1	View analyzer for us-east-1	February 20, 2025, 13:14:03 (UTC-05:00)
photosnap-lc46377-bucket	US East (N. Virginia) us-east-1	View analyzer for us-east-1	April 18, 2025, 23:41:24 (UTC-04:00)
photosnap-raw-snaps-4490772a	US East (N. Virginia) us-east-1	View analyzer for us-east-1	May 15, 2025, 15:17:37 (UTC-04:00)

Now we can see that our resources are set up and our infrastructure is ready for the next steps.

4.2 CloudFormation

In this we handle the EC2 instances and the Auto Scaling Groups and also have a separate template for lambda to get logs when an image is uploaded. Let's discuss the folder structure. The ec2-asg.yml handles the setting up the instance and the user-data and ASG takes care of the auto scaling. The s3-logger.yml handles the logging when an image is uploaded.

```
└─ cfn
  └─ ! ec2-asg.yml
  └─ ! s3-logger.yml
```

In our ec2-asg.yml we can see that we have this user data.

Shell

```
#!/bin/bash
set -e

# 1) Enable Python 3.8 & PostgreSQL 14 client
amazon-linux-extras enable python3.8 postgresql14
yum clean metadata -y

# 2) Install system packages
yum install -y git python3.8 postgresql

# 3) Create & activate a virtualenv
cd /home/ec2-user
python3.8 -m venv venv
source venv/bin/activate

# 4) Clone your backend & install Python deps
git clone https://github.com/lc46377/photosnap.git
cd photosnap/backend

# Upgrade pip, then install exactly your requirements
```

```
pip install --upgrade pip
pip install -r requirements.txt

# 5) Bootstrap your database schema (snaps + auth + friendships)
psql "host=${DbEndpoint} port=5432 user=${DbUsername}
password=${DbPassword} dbname=postgres sslmode=require" <<EOSQL
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

CREATE TABLE IF NOT EXISTS users (
    id          UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    username    TEXT UNIQUE NOT NULL,
    password_hash TEXT NOT NULL,
    created_at   TIMESTAMP WITH TIME ZONE DEFAULT now()
);

CREATE TABLE IF NOT EXISTS friend_requests (
    id          UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    from_user   UUID REFERENCES users(id) ON DELETE CASCADE,
    to_user     UUID REFERENCES users(id) ON DELETE CASCADE,
    status      TEXT      NOT NULL,  -- e.g.
"pending", "accepted", "rejected"
    created_at   TIMESTAMP WITH TIME ZONE DEFAULT now()
);

CREATE TABLE IF NOT EXISTS friendships (
    user_id     UUID REFERENCES users(id) ON DELETE CASCADE,
    friend_id   UUID REFERENCES users(id) ON DELETE CASCADE,
    created_at   TIMESTAMP WITH TIME ZONE DEFAULT now(),
    PRIMARY KEY(user_id, friend_id)
);

CREATE TABLE IF NOT EXISTS snaps (
    id          UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
    s3_key      TEXT NOT NULL,
    owner       UUID REFERENCES users(id) ON DELETE CASCADE,
    created_at   TIMESTAMP WITH TIME ZONE DEFAULT now()
);

CREATE TABLE IF NOT EXISTS snap_recipients (
    snap_id     UUID REFERENCES snaps(id) ON DELETE CASCADE,
    user_id     UUID REFERENCES users(id) ON DELETE CASCADE,
    viewed      BOOLEAN DEFAULT false,
    created_at   TIMESTAMPTZ DEFAULT now(),
    PRIMARY KEY(snap_id, user_id)
```

```

);
EOSQL

# 6) Export runtime env-vars
export SNAPS_BUCKET=${SnapsBucket}
export DB_ENDPOINT=${DbEndpoint}
export DB_USERNAME=${DbUsername}
export DB_PASSWORD=${DbPassword}
export AWS_REGION=${AwsRegion}

# 7) Launch your Flask app under Gunicorn
nohup /home/ec2-user/venv/bin/gunicorn app:app --bind 0.0.0.0:5000
&

```

So we install python 3.8 to avoid version conflicts, postgres, and the packages that we have in requirements.txt

```

Shell
Flask==2.2.5
boto3>=1.26.0,<2.0.0
SQLAlchemy==2.0.40
psycopg2-binary==2.9.10
gunicorn==23.0.0
Flask-Cors==3.0.10
flask-jwt-extended==4.6.0
werkzeug==3.0.6

```

After installing these packages, we set up the tables for our data. So for all EC2 instances, we set up these packages so that our backend can communicate with the frontend.

Before we run our cloud formation template, we need to export the variables so that we can run our CloudFormation template

```

Shell
export VPC_ID=$(terraform -chdir=terraform output -raw vpc_id)
export PUB_SUBNETS=$(terraform -chdir=terraform output -raw public_subnet_ids)
export BUCKET=$(terraform -chdir=terraform output -raw bucket_name)
export DB_ENDPT=$(terraform -chdir=terraform output -raw db_endpoint)

```

Run the cloud formation command for setting up the ec2-asg

Shell

```
aws cloudformation deploy \
--template-file cfn/ec2-asg.yml \
--stack-name photosnap-ec2-asg \
--parameter-overrides \
VpcId=$VPC_ID \
PublicSubnetIds="$PUB_SUBNETS" \
AmiId=ami-085386e29e44dacd7 \
KeyName=MyKeyPair \
SnapsBucket="$BUCKET" \
DbEndpoint="$DB_ENDPT" \
DbUsername=photosnap_user \
DbPassword=password \
AwsRegion=us-east-1 \
InstanceType=t2.micro \
--capabilities CAPABILITY_NAMED_IAM
```

Here we pass the public subnets, the database endpoints and the ec2 configurations so that all our scaled EC2 will follow the same configurations.

```
> export VPC_ID=$(terraform output -raw vpc_id)
export PUB_SUBNETS=$(terraform output -raw public_subnet_ids)
export BUCKET=$(terraform output -raw bucket_name)
export DB_ENDPOINT=$(terraform output -raw db_endpoint)
~/sketchup/IS599/PhotoSnap/main>
> aws cloudformation deploy \
--template-file cfn/ec2-asg.yml \
--stack-name photosnap-ec2-asg \
--parameter-overrides \
VpcId=$VPC_ID \
PublicSubnetIds="$PUB_SUBNETS" \
AmiId=ami-085386e29e44dacd7 \
KeyName=MyKeyPair \
SnapsBucket="$BUCKET" \
DbEndpoint="$DB_ENDPT" \
DbUsername=photosnap_user \
DbPassword=password \
AwsRegion=us-east-1 \
InstanceType=t2.micro \
--capabilities CAPABILITY_NAMED_IAM

Waiting for changeset to be created..
Waiting for stack create/update to complete
Successfully created/updated stack - photosnap-ec2-asg
```

Stacks (2)

Stack name	Status	Created time	Description
photosnap-s3-logger	CREATE_COMPLETE	2025-05-15 15:27:14 UTC-0400	S3 ? Lambda logging pipeline for PhotoSnap raw snaps
photosnap-ec2-asg	CREATE_COMPLETE	2025-05-15 15:24:58 UTC-0400	PhotoSnap Flask API on EC2 AutoScaling Group

4.3 Logger

We create a zip of our logger code and send it to the s3 bucket using the command

Shell

```
RAW_BUCKET=$(terraform -chdir=terraform output -raw bucket_name)
aws s3 cp lambda/s3_logger.zip s3://$RAW_BUCKET/lambda/s3_logger.zip
```

Once we have the logger code in the s3 we can run our cloud formation code for the logger. We have separated our ec2 and logger because they are independent of each other keeping it modular and abstracted.

Shell

```
aws cloudformation deploy \
--template-file cfn/s3-logger.yml \
--stack-name photosnap-s3-logger \
--parameter-overrides RawSnapsBucket=$RAW_BUCKET \
--capabilities CAPABILITY_NAMED_IAM
```

We have our architecture setup. Let's pull the ALB endpoint and set it in our frontend/.env file Our .env contains the url for which each api makes a request through.

Shell

```
export ALB_URL="http://${(aws cloudformation describe-stacks \
--stack-name photosnap-ec2-asg \
--query "Stacks[0].Outputs[?OutputKey=='LoadBalancerDNS'].OutputValue" \
--output text)}"
echo "ALB → $ALB_URL"
```

```
successfully created/updated stack photosnap-s3-logger
~/Desktop/M5/IS698/photosnap/main# 
> export ALB_URL="http://${(aws cloudformation describe-stacks \
--stack-name photosnap-ec2-asg \
--query "Stacks[0].Outputs[?OutputKey=='LoadBalancerDNS'].OutputValue" \
--output text)}"
echo "ALB → $ALB_URL"
ALB = http://photosnap-AppA-8v1z8nxECDD-1332185804.us-east-1.elb.amazonaws.com
~/Desktop/M5/IS698/photosnap/main# 
> █
```

1m 7s .myenv 15:28:20

.myenv 15:35:14

Let's do a smoke test to test if our apis are working! Using the below command, we first make a request to our ALB endpoint which points to our EC2 where our backend is hosted. We also save our response and also try to get back our downloaded data.

Shell

```
curl -v -X POST $ALB_URL/upload \
-H "Content-Type: application/json" \
-d '{"filename": "smoke-test.jpg"}'
```

```

RESP=$(curl -s -X POST $ALB_URL/upload \
-H "Content-Type: application/json" \
-d '{"filename":"smoke-test.jpg"})'
PUT_URL=$(jq -r .put_url <<<"$RESP")
curl --upload-file ./smoke-test.jpg "$PUT_URL" && echo "uploaded"

GET_URL=$(jq -r .get_url <<<"$RESP")
curl -s "$GET_URL" -o downloaded.jpg && echo "downloaded"

```

```

~/Desktop/MS/IS698/photosnap main*
> RAW_BUCKET=$(!terraform -chdir=terraform output --raw bucket_name)
~/Desktop/MS/IS698/photosnap main*
> aws s3 cp lambda/s3_logger.zip s3://$RAW_BUCKET/lambda/s3_logger.zip
upload: lambda/s3_logger.zip to s3://photosnap-raw-snaps-4490772a/lambda/s3_logger.zip
~/Desktop/MS/IS698/photosnap main*
> aws cloudformation deploy \
--template-file cfn/s3-logger.yaml \
--stack-name photosnap-s3-logger \
--parameter-overrides RawSapsBucket=$RAW_BUCKET \
--capabilities CAPABILITY_NAMED_IAM
Waiting for changeset to be created...
Waiting for stack create/update to complete
Successfully created/updated stack - photosnap-s3-logger
~/Desktop/MS/IS698/photosnap main*
> [REDACTED]

```

.myenv 15:25:27
.myenv 15:26:49
.myenv 15:26:56
1m 7s .myenv 15:28:20

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
~/Desktop/MS/IS698/photosnap main*
> aws s3api put-bucket-notification-configuration \
--bucket $RAW_BUCKET \
--notification-configuration '{"LambdaFunctionConfigurations": [
    {
        "Id": "InvokeS3Logger",
        "LambdaFunctionArn": "$(aws lambda get-function --function-name photoSnapS3Logger \
--query "Configuration.FunctionArn" --output text)""
    }
]}'
~/Desktop/MS/IS698/photosnap main*
> aws s3 cp ./smoke-test.jpg s3://$RAW_BUCKET/test-notify.jpg
upload: ./smoke-test.jpg to s3://photosnap-raw-snaps-2b4976be/test-notify.jpg
~/Desktop/MS/IS698/photosnap main*
> aws logs filter-log-events \
--log-group-name /aws/lambda/photoSnapS3Logger \
--limit 5 \
--query 'events[].message' \
--output text
~/Desktop/MS/IS698/photosnap main*
> [REDACTED]

```

23:52:10 1m 7s zsh terraform
zsh
23:52:57
23:53:36
9s 23:54:04
Whole Image 0B ⌂ ⌂

```

INIT START Runtime Version: python3.9.v91      Runtime Version ARN: arn:aws:lambda:us-east-1:runtim:c4127de9acf600313fc596a22245bc4
13de98744c008a2b8a38abb334f663c06
START RequestId: 013bf3d9-1675-43d7-a8d2-f4839f11c74e Version: $LATEST
[INFO] 2025-05-13T03:53:37.792Z 013bf3d9-1675-43d7-a8d2-f4839f11c74e New file uploaded: test-notify.jpg in bucket:
photosnap-raw-snaps-2b4976be
END RequestId: 013bf3d9-1675-43d7-a8d2-f4839f11c74e
REPORT RequestId: 013bf3d9-1675-43d7-a8d2-f4839f11c74e Duration: 38.10 ms Billed Duration: 39 ms Memory Size: 128 MB
Max Memory Used: 31 MB Init Duration: 59.91 ms
[END]

```

23:52:57
23:53:36
9s 23:54:04
Whole Image 0B ⌂ ⌂

4.4 EC2

Once we have everything setup, we need to check what's happening behind the scenes. We first login into our EC2 and actually see if the packages were installed properly along with the table creation.

We need to add our IP so that we can SSH into the EC2, because we wouldn't want to SSH from other IP's

```
Shell
```

```
curl -s https://checkip.amazonaws.com
```

```
[> curl -s https://checkip.amazonaws.com
73.191.110.121
~]
> [ 15:37:26
```

```
/Desktop/MS/IS698
15:38:16
ssh -i MyKeyPair.pem ec2-user@44.199.246.94
The authenticity of host '44.199.246.94 (44.199.246.94)' can't be established.
ED25519 key fingerprint is SHA256:jBZXpMthZKxpFcQx5uYtQ7eOCLMkYjm7QFRkYqtXsG8.
His key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '44.199.246.94' (ED25519) to the list of known hosts.

#_
Amazon Linux 2
\####\ AL2 End of Life is 2026-06-30.
\#/ ___
\~' '-->
/ A newer version of Amazon Linux is available!
\~-. / Amazon Linux 2023, GA and supported until 2028-03-15.
/m/ https://aws.amazon.com/linux/amazon-linux-2023/

package(s) needed for security, out of 7 available
Run "sudo yum update" to apply all updates.
ec2-user@ip-10-0-1-95 ~]$ [
```

The screenshot shows the AWS Management Console with the EC2 service selected. The Instances section displays three terminated instances. The first instance, i-01371e58ed7e9815e, was successfully terminated. The second instance, i-01c0a2616c2736a3f, is terminating. The third instance, i-053a36623897f27e, was terminated. All instances are of type t2.micro and are located in us-east-1b.

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public
	i-01371e58ed7e9815e	Terminated	t2.micro	-	View alarms +	us-east-1b	-	-
	i-01c0a2616c2736a3f	Shutting-down	t2.micro	-	View alarms +	us-east-1b	-	-
	i-053a36623897f27e	Terminated	t2.micro	-	View alarms +	us-east-1b	-	-

```
Shell
```

```
sudo tail -n 200 /var/log/cloud-init-output.log
```

```
IS698 — ec2-user@ip-10-0-1-95:~ — ssh -i MyKeyPair.pem ec2-user@44.1.2.144

Downloading zipp-3.20.2-py3-none-any.whl (9.2 kB)
Installing collected packages: zipp, urllib3, typing-extensions, Six, PyJWT, psycopg2-binary, packaging, MarkupSafe, jmespath, itsdangerous, greenlet, click, werkzeug, SQLAlchemy, python-dateutil, Jinja2, importlib-metadata, gunicorn, Flask, botocore, s3transfer, flask-jwt-extended, Flask-Cors, boto3
Successfully installed Flask-2.2.5 Flask-Cors-3.0.10 Jinja2-3.1.6 MarkupSafe-2.1.5 PyJWT-2.9.0 SQLAlchemy-2.0.40 Six-1.17.0 boto3-1.37.38 botocore-1.37.38 click-8.1.8 flask-jwt-extended-4.6.0 greenlet-3.1.1 gunicorn-23.0.0 importlib-metadata-8.5.0 itsdangerous-2.2.0 jmespath-1.0.1 packaging-25.0 psycopg2-binary-2.9.10 python-dateutil-2.9.0.post0 s3transfer-0.11.5 typing-extensions-4.13.2 urllib3-1.26.20 werkzeug-3.0.6 zipp-3.20.2
CREATE EXTENSION
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
Cloud-init v. 19.3-46.amzn2.0.5 finished at Thu, 15 May 2025 19:26:46 +0000. Data source DataSourceEc2. Up 62.81 seconds
[2025-05-15 19:26:46 +0000] [3411] [INFO] Starting gunicorn 23.0.0
[2025-05-15 19:26:46 +0000] [3411] [INFO] Listening at: http://0.0.0.0:5000 (3411)
[2025-05-15 19:26:46 +0000] [3411] [INFO] Using worker: sync
[2025-05-15 19:26:46 +0000] [3422] [INFO] Booting worker with pid: 3422
[ec2-user@ip-10-0-1-95 ~]$
```

We can also see the health status of our Target Group

Details

arn:aws:elasticloadbalancing:us-east-1:537124959881:targetgroup/photos-AlbTa-7E8OR60DZYYM/04c278a48612ed02

Target type Instance	Protocol : Port HTTP: 5000	Protocol version HTTP1	VPC vpc-0ac3731a683126b80		
IP address type IPv4	Load balancer photos-AppAL-8v13zBnxECDD				
1 Total targets	1 Healthy	0 Unhealthy	0 Unused	0 Initial	0 Draining
	0 Anomalous				

► **Distribution of targets by Availability Zone (AZ)**
Select values in this table to see corresponding filters applied to the Registered targets table below.

Targets | **Monitoring** | **Health checks** | **Attributes** | **Tags**

Registered targets (1) [Info](#) Anomaly mitigation: Not applicable [C](#) [Deregister](#) [Register targets](#)

Target groups route requests to individual registered targets using the protocol and port number specified. Health checks are performed on all registered targets according to the target group's health check settings. Anomaly detection is automatically applied to HTTP/HTTPS target groups with at least 3 healthy targets.

Filter targets	Instance ID	Name	Port	Zone	Health status	Health status details	Administr...	Override ...	Launch...	Anom...
<input type="checkbox"/>	i-035b3410dae56e861		5000	us-east-1a (us...)	Healthy	-	<input type="checkbox"/> No override	<input type="checkbox"/> No override i...	May 15, 2...	<input checked="" type="checkbox"/> No

We can also terminate our instance and check if it triggers a new EC2

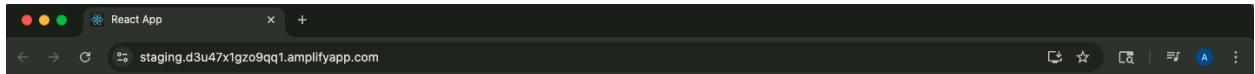
Instances (2) Info									
Find Instance by attribute or tag (case-sensitive) All states ▾									
	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	
<input type="checkbox"/>		i-035b3410dae56e861	Terminated View details Logs	t2.micro	-	View alarms +	us-east-1a	-	
<input type="checkbox"/>		i-08c76e032b827e576	Running View details Logs	t2.micro	2/2 checks passed View alarms +	View alarms +	us-east-1a	ec2-44-198-57-212.co...	

4.5 User Testing

The user can login into our application that is hosted on the AWS server using AWS Amplify

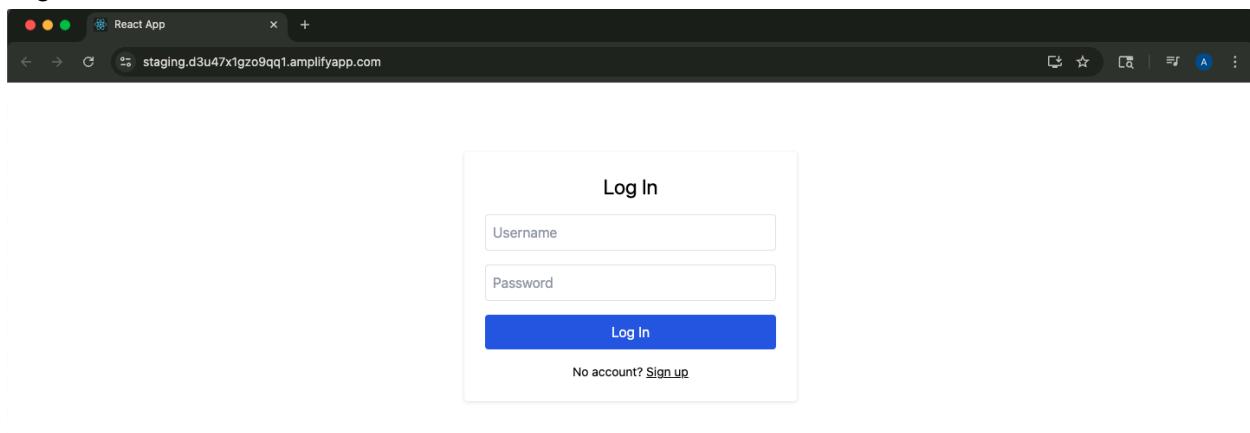
The screenshot shows the AWS Amplify console for the 'photosnap' app. On the left, there's a sidebar with 'Overview', 'Hosting', and 'App settings'. The main area has a heading 'Get to production' with three steps: 1. Add a custom domain, 2. Enable firewall protections, and 3. Connect new branches. Below this, there's a 'Branches' section with 'staging' listed as 'Deployed'. At the bottom, it shows a domain URL: <https://staging.d3u47x1gz09qq1.amplifyapp.com> and a 'Last deployment' timestamp of '1 minute ago'.

Create a account with us,



The screenshot shows the 'Sign Up' form from the previous browser screenshot. It has two input fields: 'Username' and 'Password', and a large blue 'Sign Up' button. Below the button, there's a link 'Have an account? [Log in](#)'.

Login into the account



Choose from list of users and choose a file to send, also the user can view the snap that they might have received

The application has a header with "PhotoSnap" on the left and "Log Out" on the right. Below the header are two main sections. The top section is titled "Upload a Snap" and contains a "Select Recipients:" field with "sam" and "a" listed, a "Choose File:" input field with "Choose file No file chosen", and a green "Upload Snap" button. The bottom section is titled "View a Snap" and contains a "Pick a Snap:" dropdown menu with the option "a - 0a7d76e9-6cd1-41be-9565-6f4f2b2d2314" and a blue "Show Snap" button.

We can check the list of users present in our database through the EC2

```
[ec2-user@ip-10-0-1-107 ~]$ export PGSSLMODE=require
[ec2-user@ip-10-0-1-107 ~]$
[ec2-user@ip-10-0-1-107 ~]$ psql \
>   --host="$DB_ENDPOINT" \
>   --port=5432 \
>   --username="$DB_USERNAME" \
>   --dbname="$DB_NAME" \
[>   -c "SELECT id, username FROM users;" \
          id           | username
+
-----+
facaab8a-2ab7-48ef-b06d-bd8cc0aedaf0 | ankit
6ae827b9-6776-41fc-a6a7-ae7e53427b29 | sam
(2 rows)
```

We can see that the api call was successful.

The screenshot shows a browser window for 'PhotoSnap' at localhost:3000. On the left, there's an 'Upload a Snap' form with a recipient list containing 'ankit' and 'sam'. A file input field shows 'Screenshot 2025-05-15 at 2.55.41AM' and a green 'Upload Snap' button. On the right, a modal window displays the message 'localhost:3000 says Uploaded! Snap ID: 23ad5898-585f-4657-a9cc-8f8aa1f39bde...' with an 'OK' button. Below the modal is a Network tab in the developer tools showing several requests, including preflight and XHR requests for 'Auth.jsx:14', 'UploadSnap.jsx:12', and 'UploadSnap.jsx:20'. The table in the Network tab has columns for Name, Status, Type, Initiator, Size, and Time. At the bottom of the developer tools, there's a 'What's new in DevTools 135' section.

So once the user sends snap, it goes to our s3 bucket and gets stored as the raw file. We can also view our image from the S3 bucket.

We can also view the snap sent by a different user

We have successfully developed a working snapchat clone that can send and receive images from other users.

We can destroy our AWS Infrastructure to avoid incurring cost 😂

```

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
    # (4 unchanged attributes hidden)
}

# module.vpc.aws_vpc.this will be destroyed
resource "aws_vpc" "this" {
  arn = "arn:aws:ec2:us-east-1:537124959881:vpc/vpc-019elc623e15ac085"
  cidr_block = "10.0.0.0/16"
  default_network_acl_id = "acl-0d27b79e86cc9ebfb"
  default_route_table_id = "rtb-0d8f44dd261b4fac"
  default_security_group_id = "sg-0774d60607358ae0"
  dhcp_options_id = "dopt-0b14cc6ca1c72094da"
  enable_dns_hostnames = true
  enable_dns_support = true
  enable_ip_address_usage_metrics = true
  id = "vpc-019elc623e15ac085"
  instance_tenancy = "default"
  ipv6_netmask_length = 0
  main_route_table_id = "rtb-0d8f44dd261b4fac"
  owner_id = "537124959881"
  tags = {
    Name = "photosnap-vpc"
  }
  tags_all = {
    Name = "photosnap-vpc"
  }
  # (4 unchanged attributes hidden)
}

Plan: 0 to add, 0 to change, 6 to destroy.

Changes to Outputs:
- private_subnet_ids = [
  - subnet-04cd314f9aa3d8a7",
] -> null
- public_subnet_ids = [
  - "subnet-066aedbbaf5f4874",
] -> null
  vpc_id = "vpc-019elc623e15ac085" -> null

module.vpc.aws_route_table_association_public_assoc["10.0.1.0/24"]": Destroying... [id=mrthbassoc-0f3181492a65a52eb]
module.vpc.aws_subnet.private["10.0.2.0/24"]": Destroying... [id=subnet-04cd314f9aa3d8a7]
module.vpc.aws_route_table_association_private_assoc["10.0.1.0/24"]": Destruction complete after 0s
module.vpc.aws_subnet.public["10.0.1.0/24"]": Destroying... [id=subnet-066aedbbaf5f4874]
module.vpc.aws_route_table_association_public_assoc["10.0.2.0/24"]": Destroying... [id=mrthbgw-0edc3b3b7a77745]
module.vpc.aws_subnet.private["10.0.2.0/24"]": Destruction complete after 0s
module.vpc.aws_subnet.public["10.0.1.0/24"]": Destruction complete after 0s
module.vpc.aws_route_table_public.": Destruction complete after 0s
module.vpc.aws_internet_gateway.ipw": Destruction complete after 0s
module.vpc.aws_vpc.this: Destroying... [id=vpc-019elc623e15ac085]
module.vpc.aws_vpc.this: Destruction complete after 0s

Destroy complete! Resources: 6 destroyed
~/Desktop/NS/T5698/photosnap/terraform/main> █

```

7s 00:06:13

Ln 34, Col 1 Spaces: 2 UTF-8 LF Plain Text ⌂ ⌂

We could also run certain scripts to get some insights if we face errors. The scripts are located in the script folder.

This is the `list_ec2.py` that lists current ec2 instances for us.

```

Python
import boto3

def main():
    ec2 = boto3.client("ec2")
    resp = ec2.describe_instances(
        Filters=[{"Name": "instance-state-name", "Values": ["running"]}]
    )
    for reservation in resp.get("Reservations", []):
        for inst in reservation.get("Instances", []):
            iid = inst.get("InstanceId")
            state = inst.get("State", {}).get("Name")
            typ = inst.get("InstanceType")
            print(f"{iid}\t{state}\t{typ}")

if __name__ == "__main__":
    main()

```

```
~/Desktop/MS/IS698/photosnap main*
> python3 scripts/list_ec2.py
i-0f4a5eee4f232c0ea    running t2.micro
~/Desktop/MS/IS698/photosnap main*
> █
```

list_upload_logs.py for fetching the logs from our lambda function.

Python

```
import boto3
import sys

def main():
    log_group = sys.argv[1] if len(
        sys.argv) > 1 else "/aws/lambda/photoSnapS3Logger"

    logs = boto3.client("logs", region_name="us-east-1")

    resp = logs.describe_log_streams(
        logGroupName=log_group,
        orderBy="LastEventTime",
        descending=True,
        limit=1
    )
    streams = resp.get("logStreams", [])
    if not streams:
        print(f"No log streams found in {log_group}!")
        return

    stream_name = streams[0]["logStreamName"]

    # Fetch up to 10 of the latest events
    events = logs.get_log_events(
        logGroupName=log_group,
        logStreamName=stream_name,
        limit=10,
        startFromHead=False
    )["events"]

    if not events:
        print("No log events found.")
        return

    # Print each message
    for evt in events:
```

```
    print(evt["message"])
```

```
if __name__ == "__main__":
    main()
```

```
~/Desktop/MS/IS698/photosnap main
> python3 scripts/list_upload_logs.py
~/aws/lambda/photosnapS3Logger
INIT_START Runtime Version: python:3.9.v91      Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:c4127de9acf60031fc596a22245bc413de98744c008a2b8a38abb334f663c06
START RequestId: 013bf3d9-1675-43d7-a8d2-f4839f11c74e Version: $LATEST
[INFO] 2025-05-13T03:53:37.792Z 013bf3d9-1675-43d7-a8d2-f4839f11c74e New file uploaded: test-notify.jpg in bucket: photosnap-raw-snaps-2b4976be
END RequestId: 013bf3d9-1675-43d7-a8d2-f4839f11c74e
REPORT RequestId: 013bf3d9-1675-43d7-a8d2-f4839f11c74e Duration: 38.10 ms      Billed Duration: 39 ms   Memory Size: 128 MB   Max Memory Used: 31 MB  Init Duration: 58.91 ms
~/Desktop/MS/IS698/photosnap main*
> |
```

5. Challenges faced and solutions implemented

Many challenges were faced during project implementation.

1. ALB was not set up properly because it needs at least 2 public subnets to be set up properly. Solution was to implement 2 public and 2 private subnets so that the ALB can be launched without any issues.
2. Postgres specific version needs t2.micro instead of t3.micro storage, resolved by switching from t3 to t2.micro.
3. SSH into the EC2 launched from the cloudformation template from my IP, resolved by adding my ip into the security group
4. The target groups were not healthy, resolved by checking the logs of the EC2.
5. Packages were not installed due to version mismatch of python, resolved by finding the right versions for the python 3.8
6. Insertion of data into table due to wrong schema mismatch, fixed the schema
7. CORS policy related errors, resolved by adding OPTIONS in flask app.

These challenges were resolved after carefully analyzing the logs, reading articles and with help of resources online.

6. Future Works

We have set up the infrastructure, and have connected the frontend and backend. Frontend needs proper design and a fix needed to CORS policy. Backend needs to handle more edge cases. Scalability test is also required for photosnap.

Frameworks & Tools: React for Frontend, Flask for Backend, [Draw.io](#) for Architecture, Git for version control.