Migrated On Premise Application to AWS

AWS services

Amazon CloudFront, AWS Aurora, RDS, Amazon Simple Storage Service (Amazon S3), IAM Roles, VPC, Internet gateway, Security Group, etc.

Description

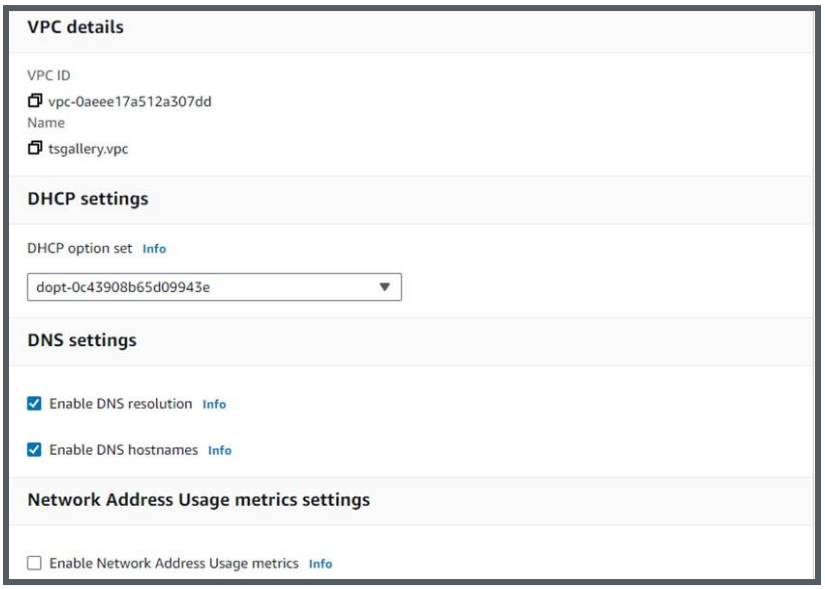
Here we have an on-premise web server (in this project its hosted on EC2), static content was first transferred to S3 bucket and then content was delivered via CloudFront. Database was shifted from single instance to highly available and fault tolerant serverless database (Aurora).

#1 - VPC Setup

In this site cloud network is being setup for resources used in entire project.

Created VPC

Created VPC named **tsgallery.vpc** which has IPv4 CIDR block **10.11.0.0/16**
We need to ensure checkbox for **DNS hostnames** is enabled.

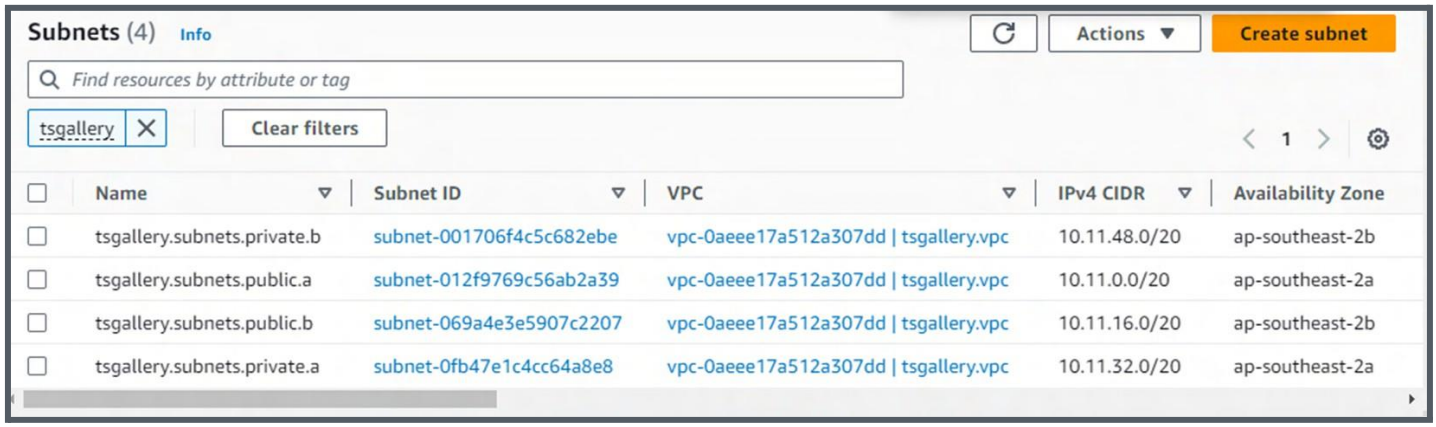


The DNS hostnames attribute determines whether instances launched in the VPC receive public DNS hostnames that correspond to their public IP addresses.

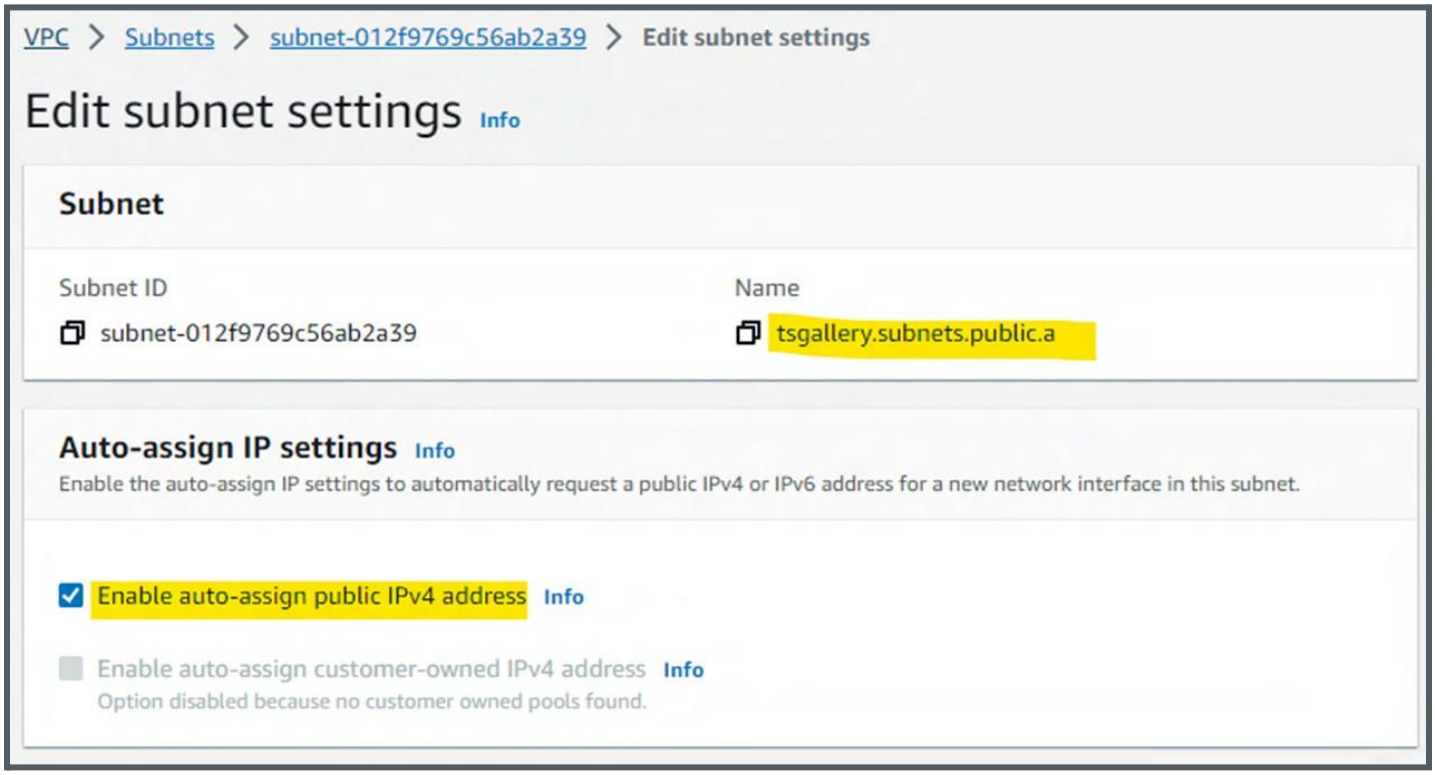
Created Subnets

Created 4 new subnets, 2 were private and 2 public.
All subnets were associated with VPC created.

Subnet name	IPv4 CIDR Block	Availability Zone
tsgallery.subnets.public.a	10.11.0.0/20	*a
tsgallery.subnets.public.b	10.11.16.0/20	*b
tsgallery.subnets.private.a	10.11.32.0/20	*a
tsgallery.subnets.private.b	10.11.48.0/20	*b



Both Public subnets needs to be assigned IP address
So Enabled **AUTO ASSIGN PUBLIC IP Address** for both public subnets.

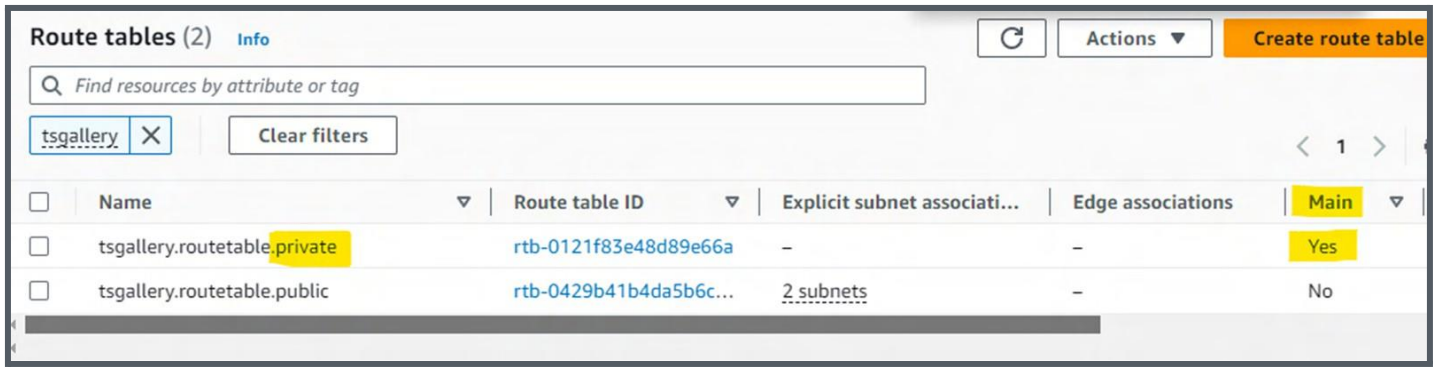


Created Internet Gateway

Created Internet gateway and attached to VPC - tsgallery.internetgateway

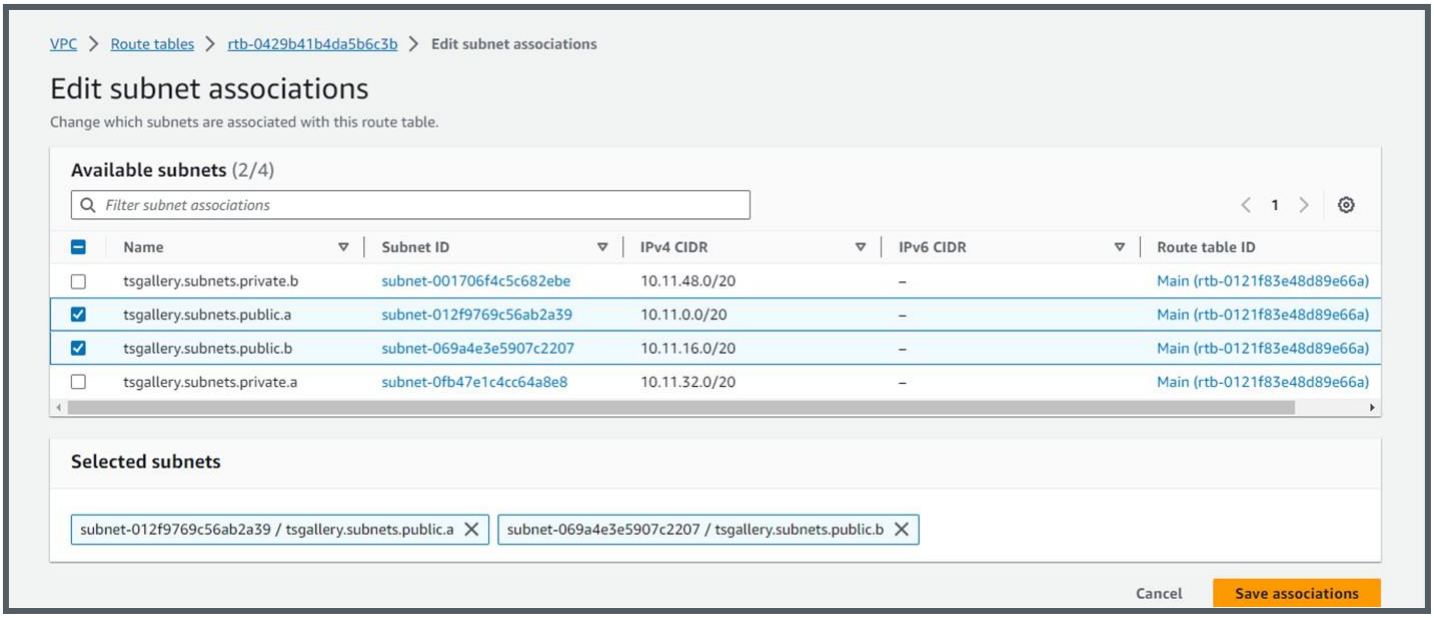
Securing Private Subnets

The default route table is renamed as Private, No subnet explicitly mentioned meaning it will assign this MAIN Default route table to newly created subnets in this VPC.



Adding a new public route

In the Route Editor table provided 0.0.0.0/0 as Destination, selected Internet Gateway we created from the Target dropdown.
Associated both public subnets to the newly created public route table.



Create an IAM Role

This IAM role will be used by your application to interact with AWS Services.
Role is for EC2, Policies of **AmazonS3FullAccess** and **AmazonSSMManagedInstanceCore** is provided to Role.

Step 1: Select trusted entities Edit

Trust policy

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Principal": {
                "Service": [
                    "ec2.amazonaws.com"
                ]
            }
        }
    ]
}
```

Step 2: Add permissions Edit

Permissions policy summary

Policy name	Type	Attached as
AmazonS3FullAccess	AWS managed	Permissions policy
AmazonSSMManagedInstanceCore	AWS managed	Permissions policy

The SSMManagedInstanceCore is required to allow Systems Manager to open sessions directly on an EC2 instance and for its Patch Manager to apply security updates automatically for you

#2 - EC2 Servers

In this lab I created a new Amazon EC2 instance to host a web server.
Installed web server, database and other support tools on that web server.
Then test if it is working before migrating to AWS.

Created SSH Certificate

Created a new **KeyPair** for this EC2, Key pair is secure and convenient way to connect to EC2s.

Key pairs (1) Info Actions Create k

Search 1

Name	Type	Created	Fingerprint	ID
tsa-ap-southeast-2	rsa	2023/09/17 18:28 GMT+5:30	7e:5d:64:78:cb:00:b3:bd:48:bf:af:8e:06:...	key-038b27b7ad2654aca

Created E2 Instance

Created webserver on EC2 Instance with instance type **m6g.medium**
Associated the key pair created in the previous step.
This on premise webserver is in the same VPC we created, where a public subnet is provided.
Public subnet associated here is **tsgallery.subnets.public.a**
Firewall Security Group associated had permission to allow HTTP requests access to the webserver from everywhere.
Associated **IAM Role** “tsgallery.roles.serverroles”

Added user data this script runs when the instance boots up for the first time.

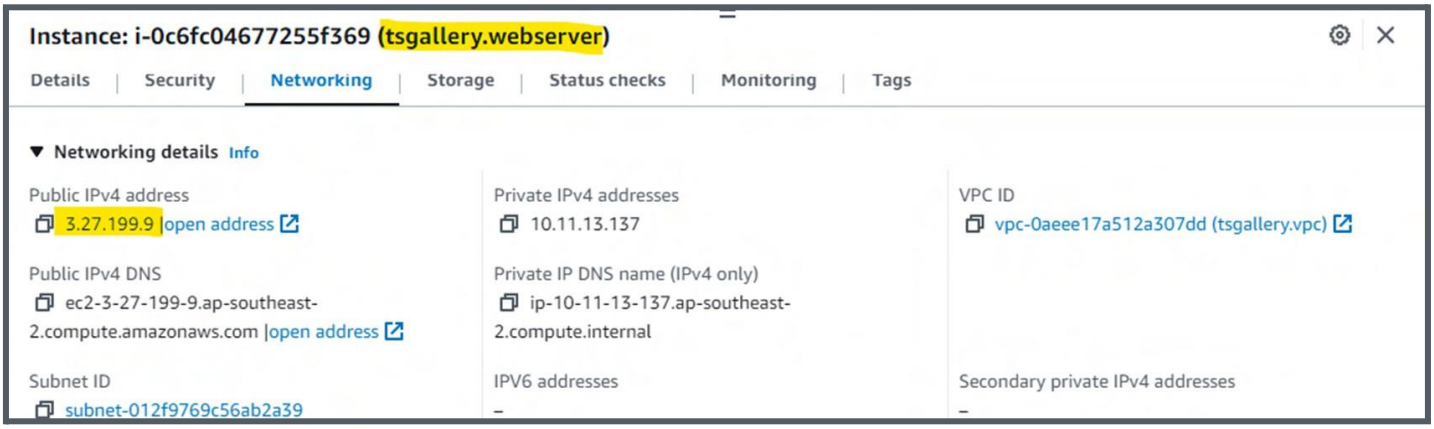
```
#!/bin/bash
echo ==== Starting UserData Script ====
curl -k -o /root/setup.sh
http://labassets.awstechshift.cloud/prod/migrate/2/setup.sh
chmod +x /root/setup.sh
sudo -i /root/setup.sh
echo ==== Finished UserData Script ====
```

The setup script itself does the following:

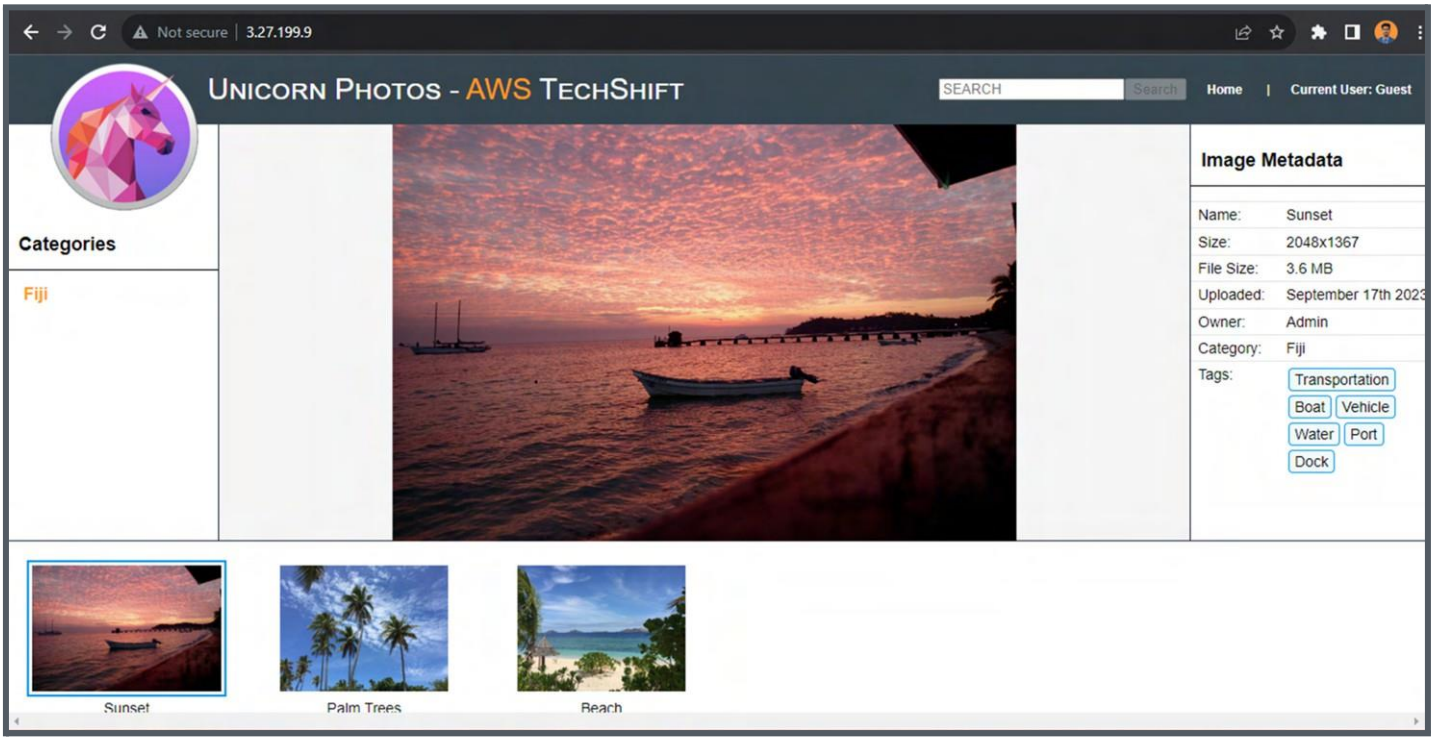
- Update the server
- Download the files needed for your server
- Install MySQL and run the initialization SQL script
- Download and configure NodeJS
- Download Forever utility
- Install and configure NGinx
- Copy all the server files to the correct locations
- Start the server

Access our Site

Site is accessible after the instance state is changed to RUNNING and the server completes running the User Data script.
Currently the firewall is configured to access via HTTP protocol so URI with HTTPS wont work here.



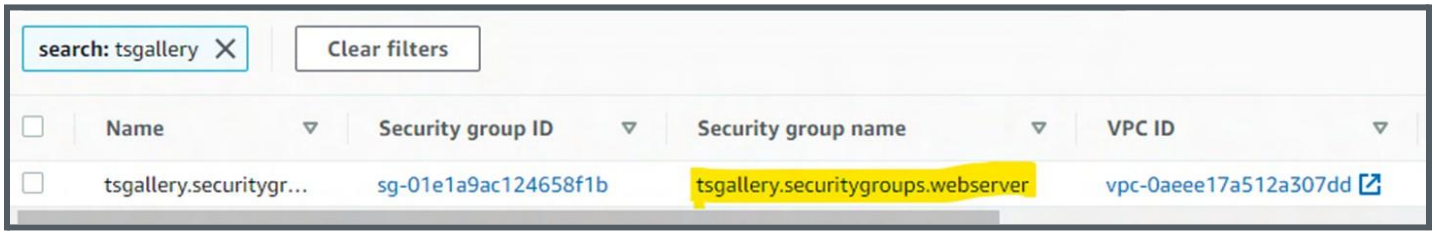
Here we can see a photo gallery application is running.
Administration page is accessible from the top right corner by clicking on **Current User: Guest.**
Username is **admin** and the **password** is **awstechshift**



Adding a Tag to our Security Group

Tagging our resources is the best practice. There is no tags to the security group because it was created using the instance wizard template.

Tag added to security group here is **tsgallery.securitygroups.webserver**



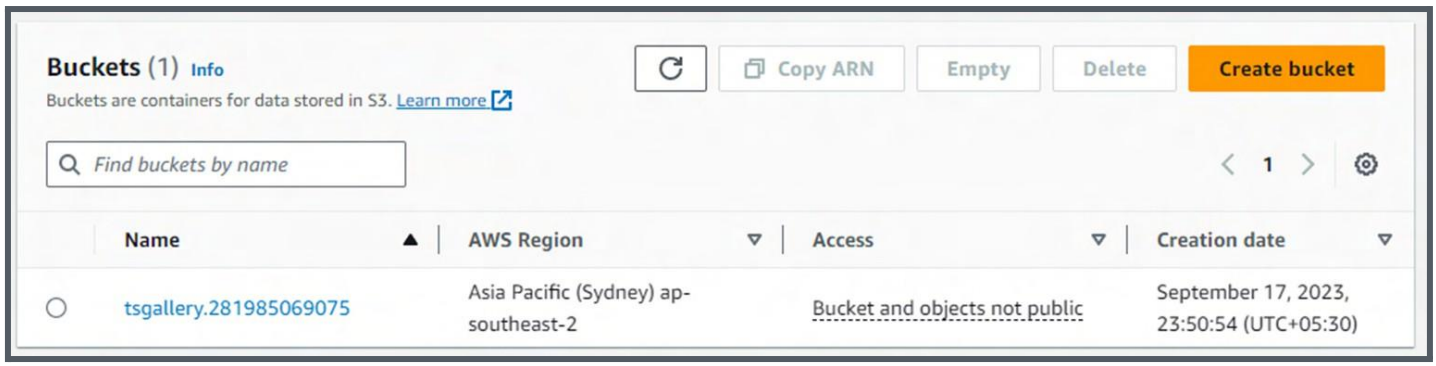
#3 - S3 & Cloudfront

Objectives

Here I moved the static portions of the web application from the WebServer to a S3 bucket which was served using Amazon CloudFront for distribution.

Storing images on S3 Bucket

Created S3 bucket in the same region where EC2 web server is running.



Copied images and updated to the server.

SSH session was initiated for EC2 web server, this was the step: Systems Manager > Session Manager > Start Session.

Below script is used to update the server.

```
sudo su -
cd /opt/ts_gallery
sh update.sh
```

Update Script does this:

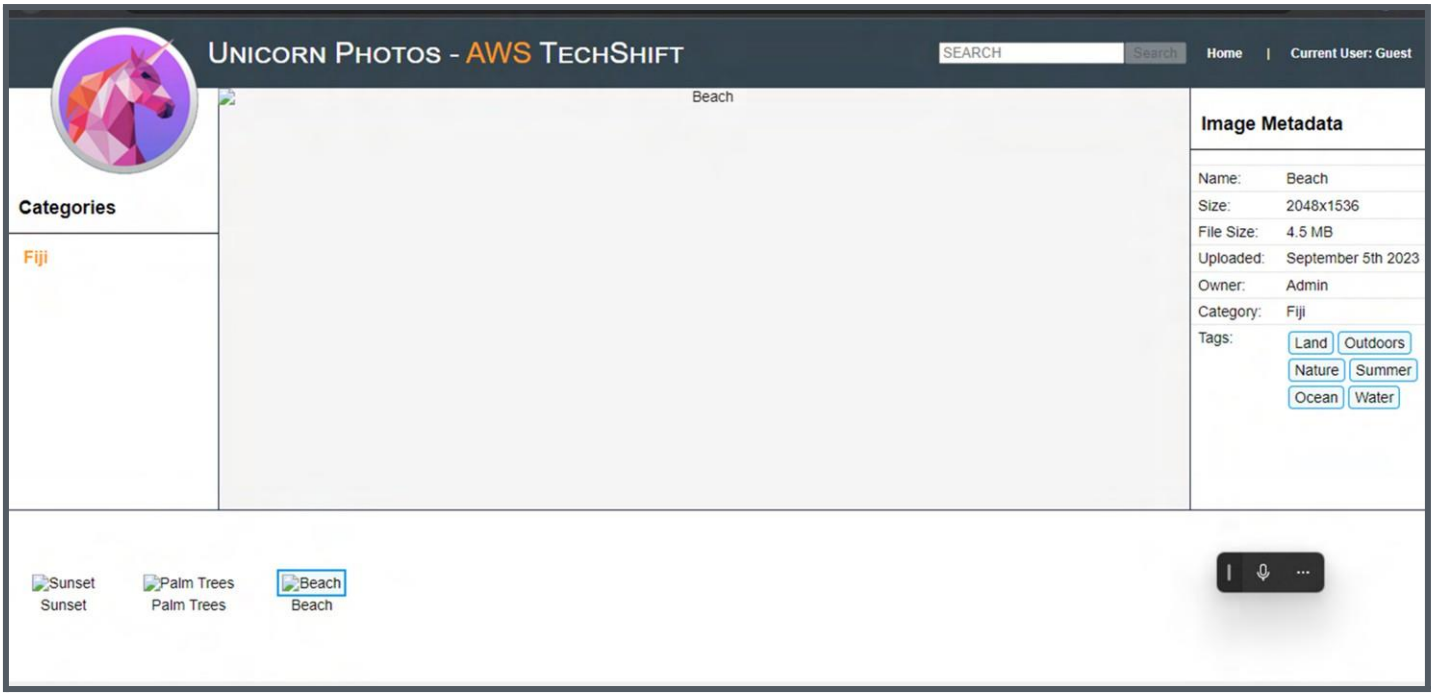
- Copy the images to S3

- Remove the images from the server
- Update the server config to reference S3 bucket
- Update the filesystem server code to use S3
- Restart the server

```
[root@ip-10-11-13-137 ts_gallery]# cd /opt/ts_gallery
[root@ip-10-11-13-137 ts_gallery]# sh update.sh
=====================
  TechShift Migrate
=====================

1. Update S3
7. Exit
Enter choice [1 to 7]: 1
--------------------------
 Starting S3 Migration
--------------------------
Enter your bucket name and press <ENTER>: tsgallery.281985069075

s3_migrate: ==== Validating access to tsgallery.281985069075 ====
s3_migrate: Bucket s3://tsgallery.281985069075 found and permissions appears ok
s3_migrate: Lock file created
s3_migrate: ==== Moving images to tsgallery.281985069075 ====
s3_migrate: Backup images
s3_migrate: Copy images to S3

Images have now been deleted from the server. To confirm the files are not on the server,
return to the browser window with the photo gallery open and refresh it. You should see
the site load with broken images.

Press any key to continue updating server . . .
s3_migrate: ==== Update the server ====
s3_migrate: Add the AWS-SDK package
```

```
Enter choice [1 to 7]: 1
--------------------------
 Starting S3 Migration
--------------------------
Enter your bucket name and press <ENTER>: tsgallery.281985069075

s3_migrate: ==== Validating access to tsgallery.281985069075 ====
s3_migrate: Bucket s3://tsgallery.281985069075 found and permissions appears ok
s3_migrate: Lock file created
s3_migrate: ==== Moving images to tsgallery.281985069075 ====
s3_migrate: Backup images
s3_migrate: Copy images to S3

Images have now been deleted from the server. To confirm the files are not on the server,
return to the browser window with the photo gallery open and refresh it. You should see
the site load with broken images.

Press any key to continue updating server . . .
s3_migrate: ==== Update the server ====
s3_migrate: Add the AWS-SDK package
s3_migrate: Backup config to config.s3.bak.js
s3_migrate: Add region and secret name to the aws block in config.js
s3_migrate: ==== Update the server code to use S3 ====
s3_migrate: Backup www to www.s3.bak.js
s3_migrate: Update dependancy injected file system with s3 version
s3_migrate: Restarting server
s3_migrate: Server update done
--------------------------
 S3 Migration Finished
--------------------------
```

At this step if we refresh the web page we should see the broken images as seen in the below snap.

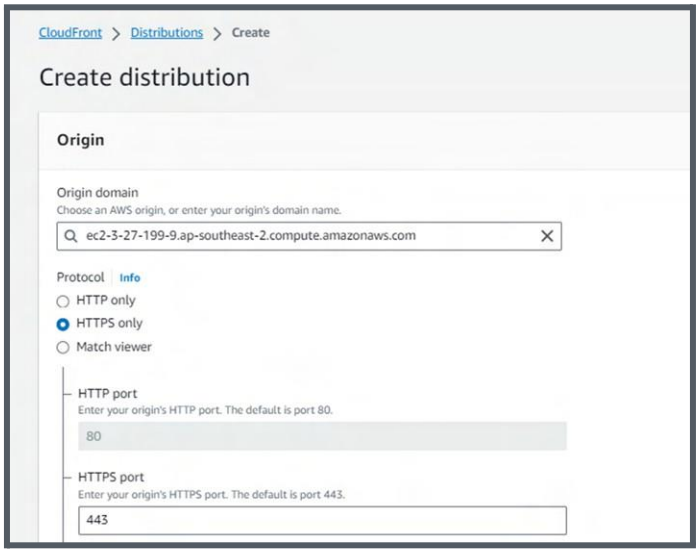


Serving using CloudFront

Here I did set up a CloudFront distribution to serve the static assets and route the API requests to the server.

In the Origin Settings section entered the website URI (EC2 instance's Public IPv4 DNS), into the Origin Domain and Enable Origin Shield was set as No.

In the **Default cache behavior** section, under Allowed HTTP Methods selected GET, HEAD, OPTIONS, PUT, POST, PATCH, DELETE. Ensure Cache policy and origin request policy is selected and selected Caching Disabled from the Cache Policy dropdown.



In the Default root object text box, specify **index.html** and click Create Distribution.

At this stage, the distribution only has the server in it. we need to add the images from S3.
With the proper distribution ID, we will create Origin.
For Origin Domain selected the S3 bucket.
In the Origin access section > Legacy access identities. Create a new OIA by clicking on **Create new OIA**. In the Bucket policy section, selected **Yes, update the bucket policy.**

Here now created new behavior that links to new origin. Clicked the **Behaviors tab**, then click **Create Behavior.**

Entered images/uploads/*.* as the Path Pattern and selected S3 bucket for the Origin or Origin Group.

Now we need to get the new cloudfront URI for our photo gallery. If we receive an error stating that our connection is not private, or get an AccessDenied message, S3 is still updating its internal DNS. Our setup is correct, we need to wait until the update completes.

#4 - VPC Setup

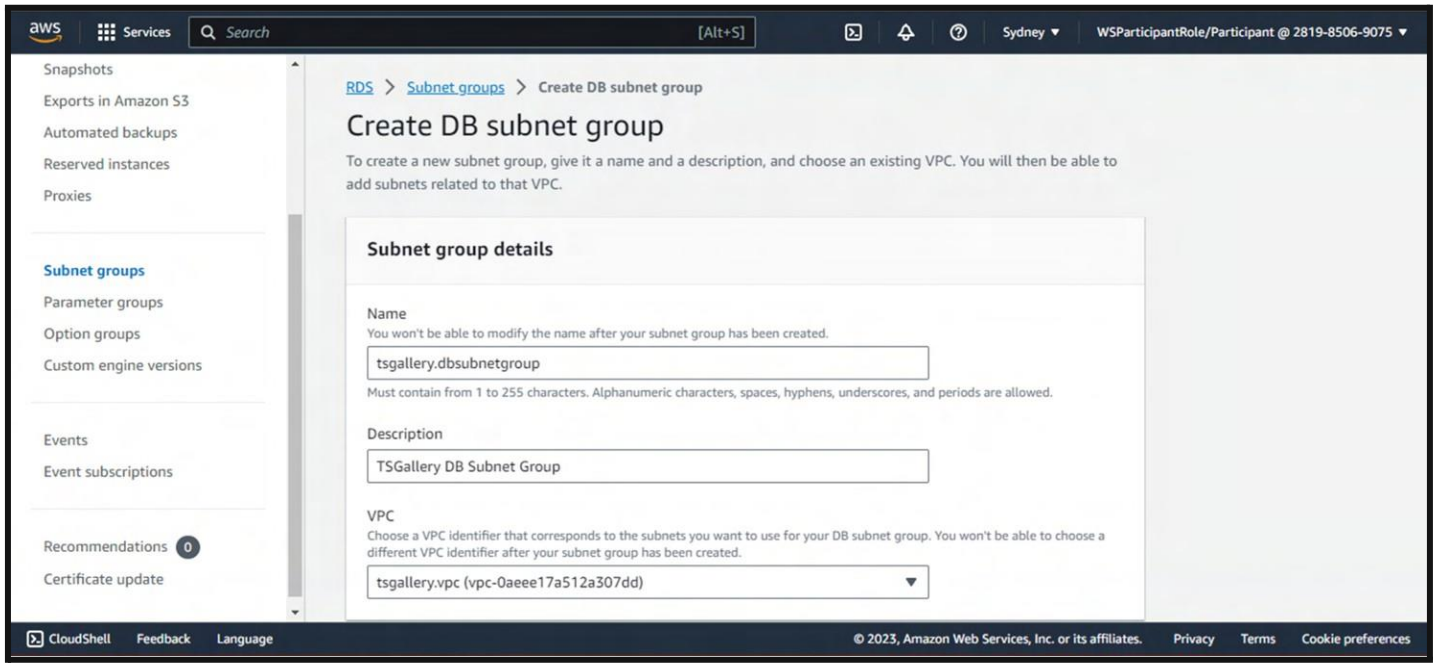
Here I moved the database off the single instance to highly available and fault tolerant serverless database.

- Created an Aurora Serverless database
- Exported data from your single instance
- Imported data into Aurora Server-less
- Updated the app code to use the new database

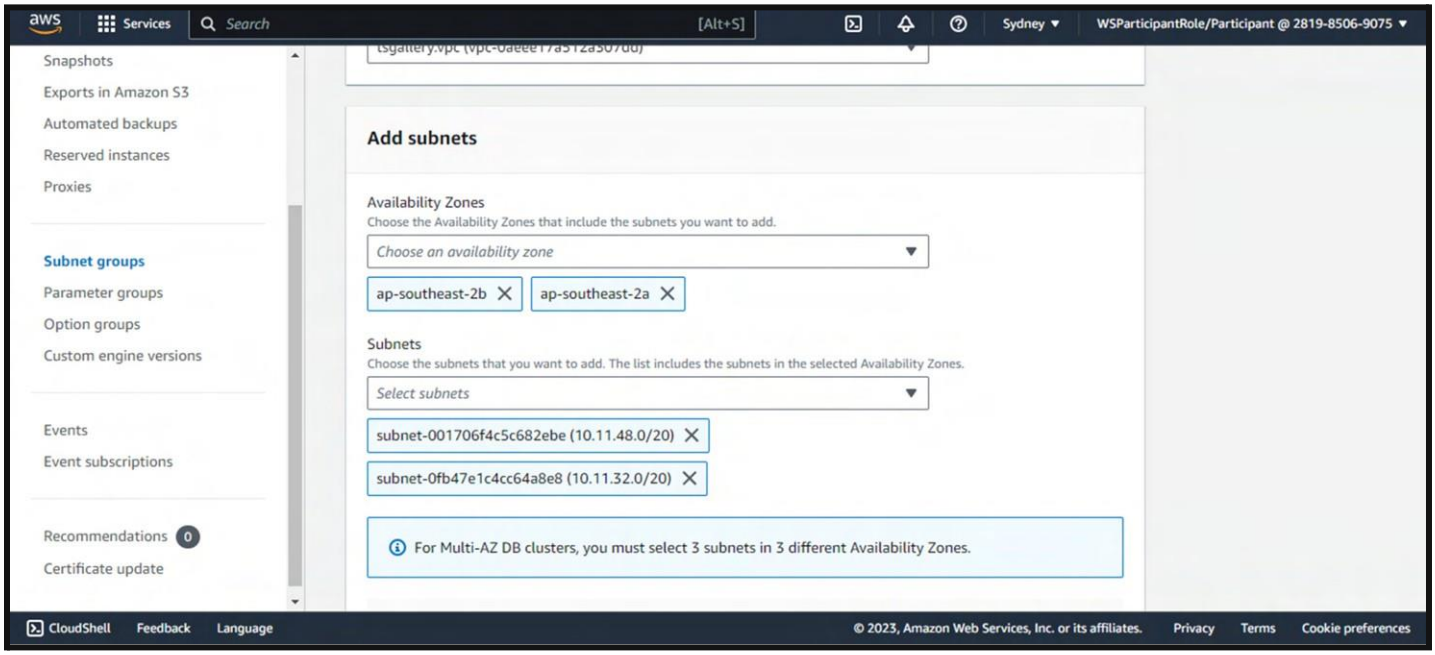
Create a Database subnet group

An RDS database needs to be located in a database subnet group. A database subnet group is simply a set of VPC subnets that the database will use to host the various nodes and endpoints.

Navigated to **RDS Service** > **Subnet Groups** > **Create DB Subnet Group**
I named it as tsgallery.dbsubnetgroup

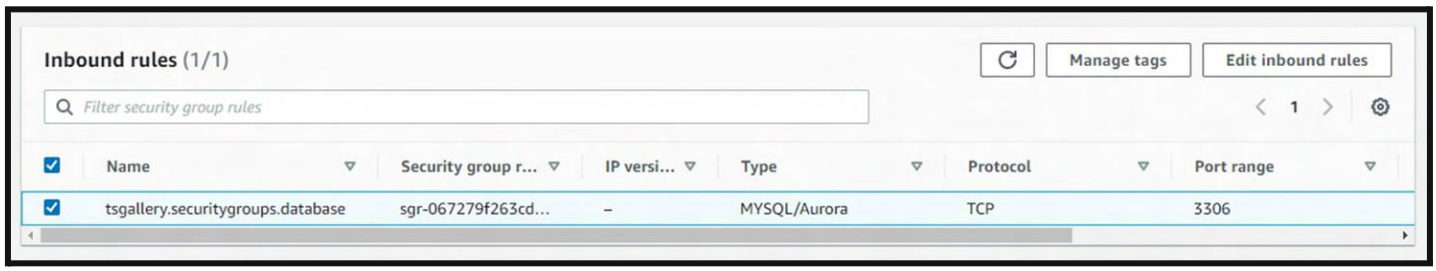


Here I associated the both private subnets I had created in the initial stage.



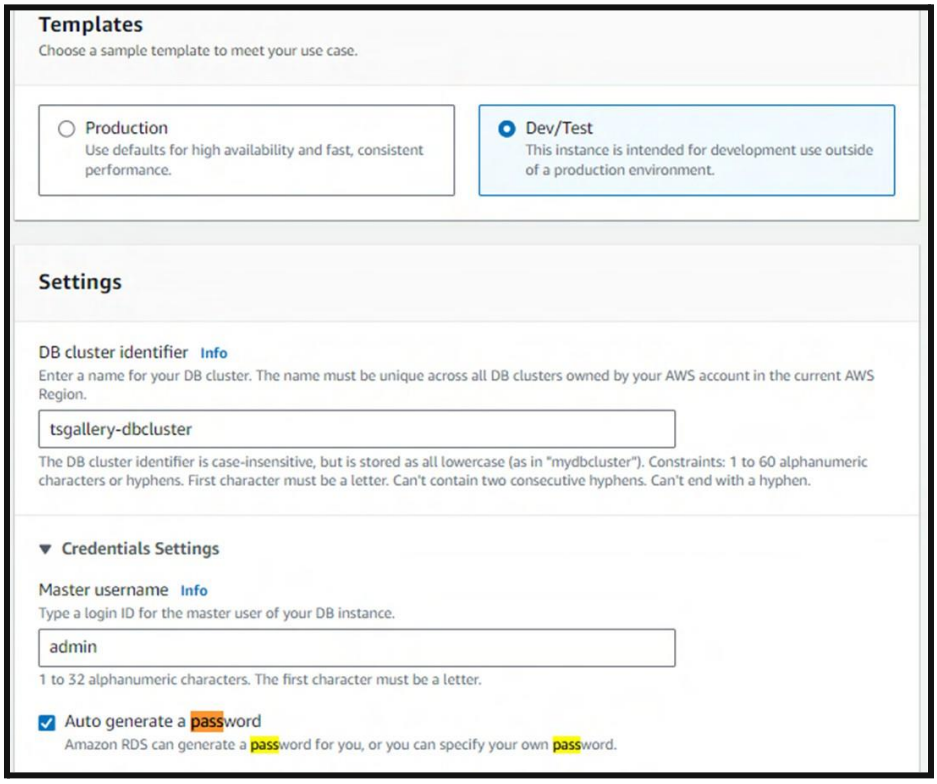
Create a private security group

Created a new private Security Group to accept inbound requests of type MYSQL/Aurora.

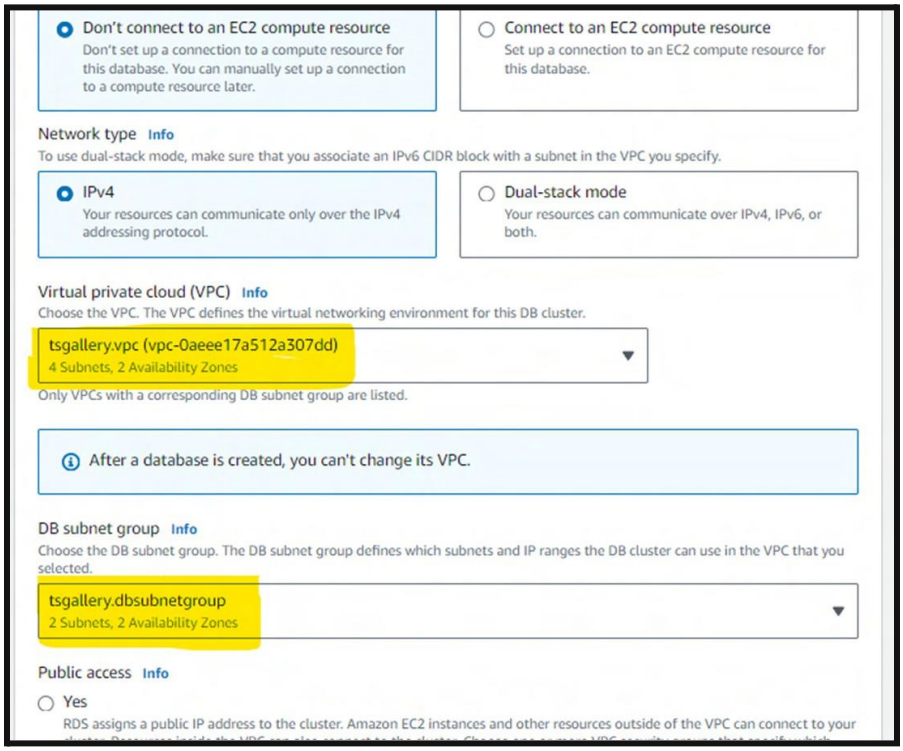


Added Aurora Database

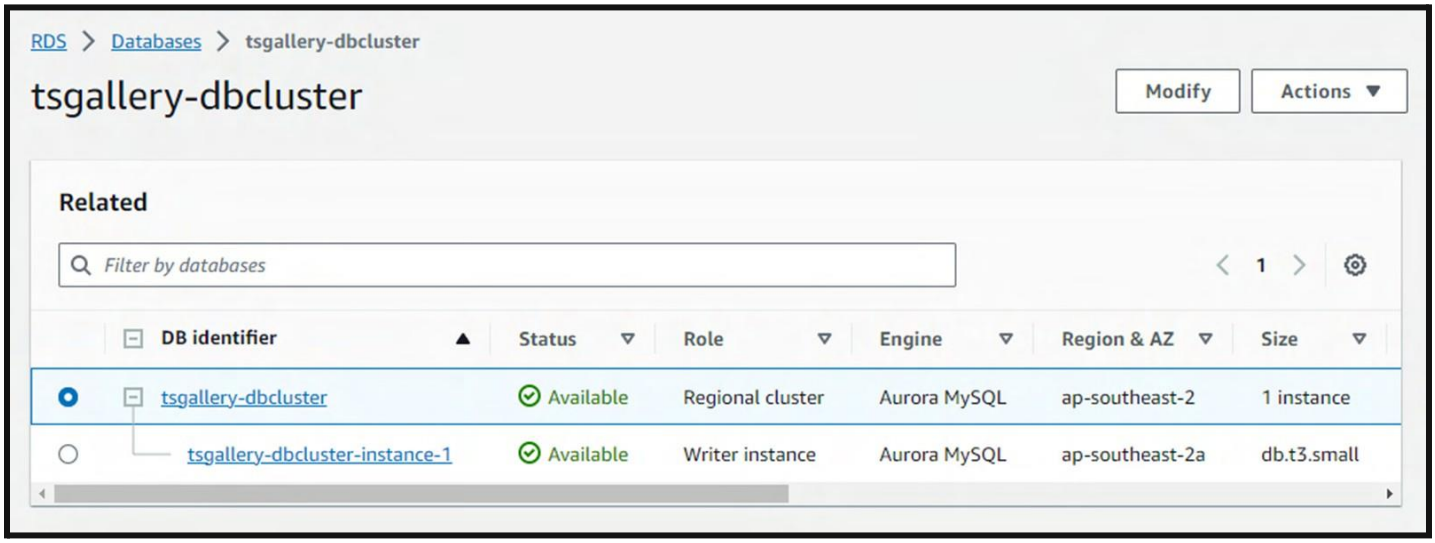
- Navigated to RDS Service > Databases > **Create Database.**
- **Standard Create** needs to be selected with **Amazon Aurora** as the engine type.
- Then **Amazon Aurora with MySQL compatibility**



- **DB cluster identifier - tsgallery-dbcluster**
- Checked the **Auto generate a password** option
- In the Instance Configuration section, select the **Burstable classess** instance and leave the default value of **db.t3.small** as the instance type.
- VPC here will be **tsgallery.vpc** and DB Subnet group is set to **tsgallery.dbsubnetgroup**

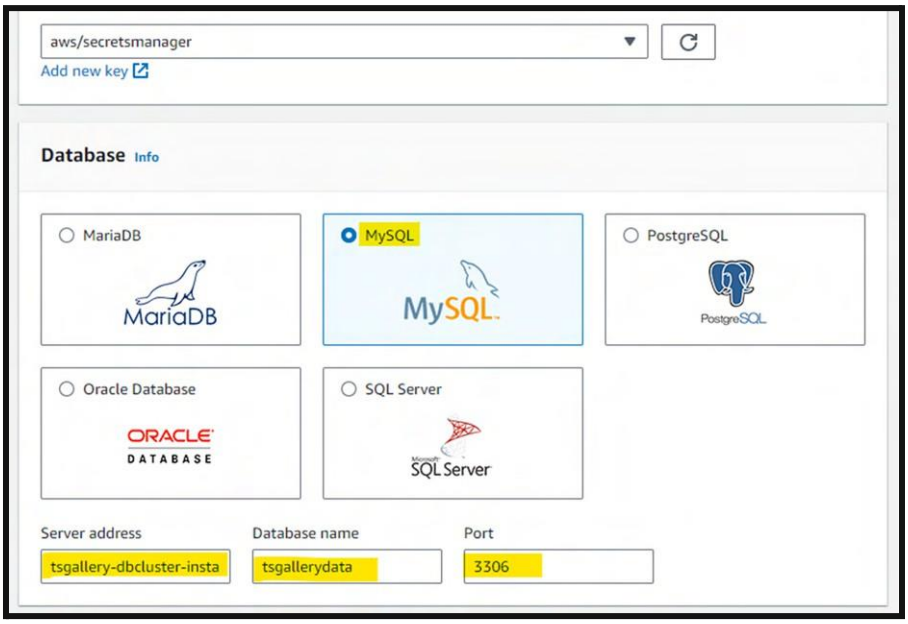


- Security group should be **tsgallery.securitygroups.database**, then deselected the **Default** security group.
- Under Monitoring, unchecked the **Enable Enhanced monitoring.**
- Unchecked the **Enable deletion protection.**.
- Finally here hit the **Create database.**
- In the **View credential details** button we will get the auto generated credentials.
- Cluster details can be found in **DB Identifier**. Endpoint and port values can be seen in **Connectivity & security tab.**



Use secret manager

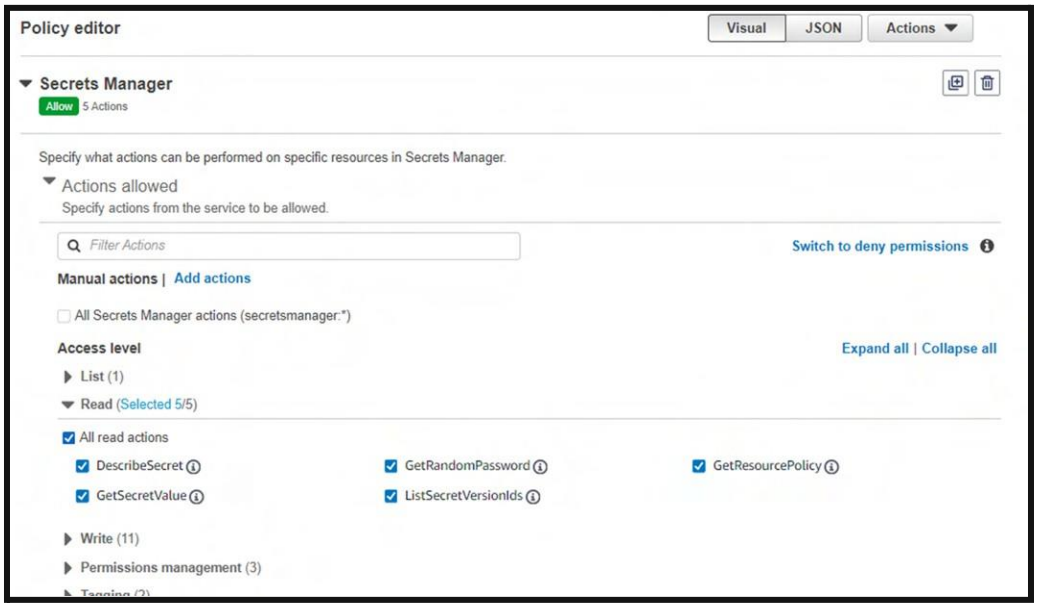
- Navigated to **Secrets Manager** > **Store a new secret.**
- Option **Credentials for other database** is selected and enter the **User name and Password**.
- Selected **MySQL** for the **Select which database this secret will access** option.
- Provided the **Endpoint** and **Port** of Aurora database.
- Database name as **tsgallerydata.**
- Enter **tsgallery.secrets.dbcluster** as the Secret name and the Description.



Gave server access to Secrets Manager

Adding access so that server can read secrets, this is done by updated IAM role with new permissions.

- Selected **Add permissions** and then option of **Create inline policy**.
- Selected **Secrets Manager** Service with **Read** checkbox and **All resources** needs to be selected.
- This new policy is named as **tsgallery.policies.readsecrets**



Moved the existing data and Update the server

Update script is used to migrate data and update the server.

```
sudo su -
cd /opt/ts_gallery
sh update.sh
```

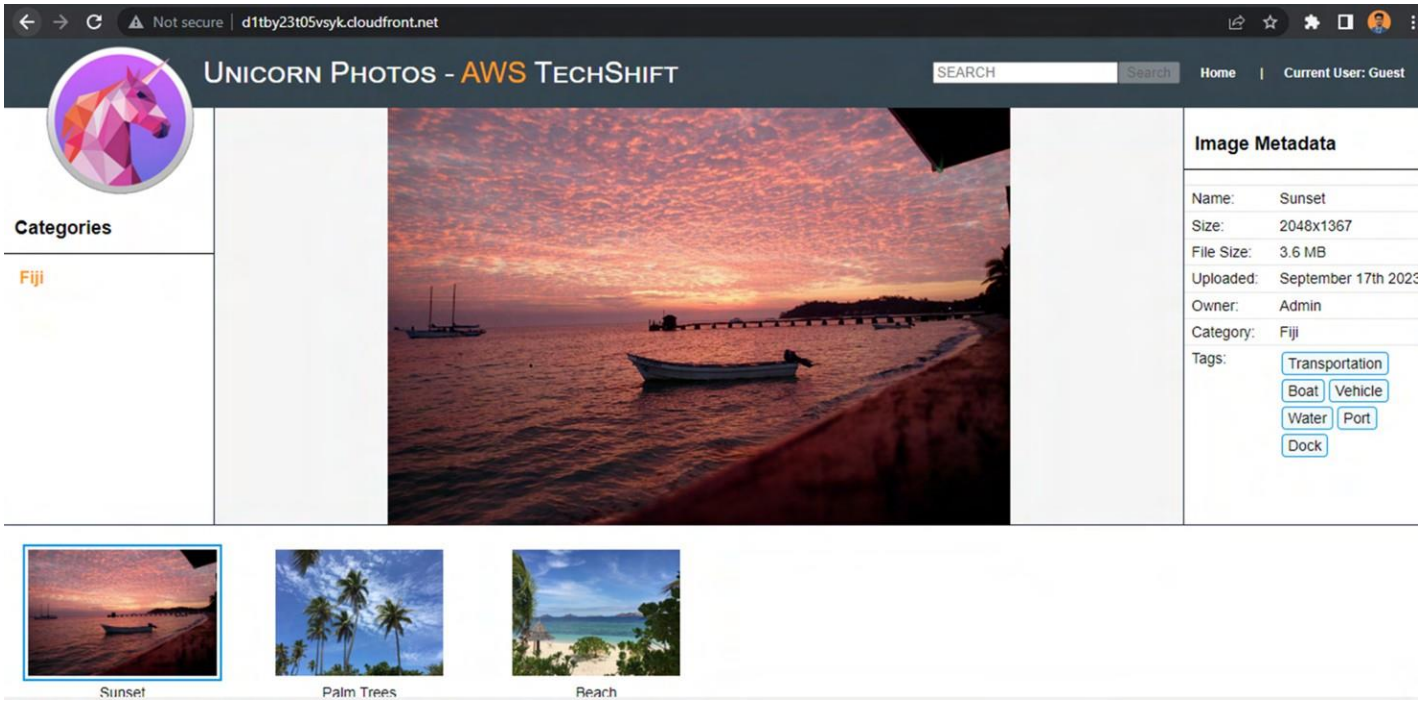
The update.sh script will do the following:

- Setup new schema in Aurora
- Copy data from local MySQL server
- Update the server config to reference the secret and remove hard coded database credentials
- Update the filesystem server code to use secret to get credentials
- Restart the server
- Shutdown MySQL server on local host as it's no longer needed

```
  TechShift Migrate
====================

2. Rollback S3
3. Update RDS
7. Exit
Enter choice [1 to 7]: 3
--------------------------
 Starting RDS Migration
--------------------------
Enter your secret name and press <ENTER>: tsgallery.secrets.dbcluster

rds_migrate: Retrieve secret tsgallery.secrets.dbcluster from ap-southeast-2
rds_migrate: ==== Moving data to tsgallery-dbcluster-instance-1.cb0gku2mg9ez.ap-southeast-2.rds.amazonaws.com ====
rds_migrate: Lock file created
rds_migrate: Create new schema tsgallerydata on tsgallery-dbcluster-instance-1.cb0gku2mg9ez.ap-southeast-2.rds.amazonaws.com
rds_migrate: Export data from MySQL localhost
rds_migrate: Push data to tsgallery-dbcluster-instance-1.cb0gku2mg9ez.ap-southeast-2.rds.amazonaws.com
rds_migrate: ==== Update the server config ====
rds_migrate: Backup config to config.rds.bak.js
rds_migrate: Add secret name to the aws block in config.js
rds_migrate: Removing un-needed db configuration block from config.js
rds_migrate: ==== Update the server code to retrieve database credentials before creating database pool ====
rds_migrate: Backup app to app.rds.bak.js
rds_migrate: Removing existing database configuration block
rds_migrate: Adding SecretsManager enabled database configuration block
rds_migrate: Restarting server
rds_migrate: Disable MySQL currently running on server
rds_migrate: Removing lock files
--------------------------
 RDS Migration Finished
--------------------------
[root@ip-10-11-13-137 ts_gallery]#
```

```
  TechShift Migrate
======================

4. Rollback RDS
5. Update Elastic Beanstalk
7. Exit
Enter choice [1 to 7]: 5
----------------------------------
 Starting Elastic Beanstalk Setup
----------------------------------
eb_migrate: ==== Validating access to tsgallery.281985069075 ====
eb_migrate: Bucket s3://tsgallery.281985069075 found and permissions appears ok
eb_migrate: Lock file created
eb_migrate: Create Procfile to control NodeJS startup
eb_migrate: Create .ebextensions folder
eb_migrate: Create disable-npm.config to prevent 'npm rebuild'
eb_migrate: Zip server files for deployment
 .
........
...........
.
```

```
.
.

eb_migrate: Upload deployment package to S3
eb_migrate: Removing lock files
-----------------------------------
 Elastic Beanstalk Setup Finished
-----------------------------------
[root@ip-10-11-13-137 ts gallery]#
```



So now we have setup a serverless database cluster, migrated our data into the new cluster and updated the server to use the new cluster.