Hosting Dynamic Web by *AWS*

0_ Scheme

➔ Traffic **SENDING**: Using **Gateway** (NAT&IGW) & **RECEIVING**: **Load Balancer**
➔ To access the private instance using **Bastion Host Server** on Public Subnet by SSH
➔ Private Instance: 1**Web** Server, 2**Database** Server (**RDS**)
➔ Temporary USE: Database **Migration Server**, which is used for migration from S3 to **RDS**

1_ Create IAM Role

(1) Root user: MFA (Multi-Factor Authentication) is registered when accessing the AWS resources.
(2) **Roles**: Creating the **policy** [e.g. AmazonS3FullAccess] to attach the IAM role of **EC2**s.
Tag: data-access-in-s3

2_ Create VPC and essential configuration

(1) Create **VPC** (Tag: console-cloud)
(2) Assigned **Subnets** (Considering **High Availability, Faulty Tolerance**):
- AZ1: Public Subnet, Private Subnet [eu-west-1a]
- AZ2: Public Subnet, Private Subnet [eu-west-1b]

192.168.56.0/27 -> tag: cc-**public**-main // AZ: eu-west-1a

192.168.56.32/28 -> tag: cc-public-sub2 // AZ: eu-west-1b

==========

192.168.56.112/27 -> tag: cc-**private**-main // AZ: eu-west-1a

192.168.56.144/28 -> tag: cc-private-sub2 // AZ: eu-west-1b

192.168.56.160/28 -> tag: cc-private-sub3 // AZ: eu-west-1c

192.168.56.192/27 -> tag: cc-**private-db1** // AZ: eu-west-1a

192.168.56.224/28 -> tag: cc-private-db2 // AZ: eu-west-1b

(3) 3 **Route Tables** Connect with **Gateway** [1 IGW, 2 NATs]

- All Public Route tables can share the IGW attached to VPC.
- All Private subnets per AZ need 1 Route table and 1 NAT gateway.
- **NAT** *gateway* is created per Availability Zone with **EIP.**
- 1 **Public** Route Table (**VPC**) & 2 **Private** Route Table (**AZ1, AZ2**)

(4) **5 Security groups**

➔ General Traffic CASEs:

- Inbound: HTTP / HTTPS (WWW) // SSH (Remote)
- Outbound: All traffic [0.0.0.0/0]

1)	cc-bastion-ssh ➔	IN	SSH	OUT	[0.0.0.0/0]
2)	cc-load-balancer ➔	IN	HTTP, HTTPS	OUT	[0.0.0.0/0]
3)	cc-shopwise-web ➔	IN	HTTP, HTTPS – [cc-load-balancer] SSH – [bastion-ssh]		
4)	cc-rds-instance ➔	IN	MYSQL – 2 [web-server/ data-migration] SSH – [bastion-ssh]		
5)	cc-db-migration ➔	IN	SSH – [bastion-ssh]		

(5) **Network ACL**s: using the *default* Network ACLs this time.

(6) Issuing the **Key Pairs**: type: RSA / OpenSSH

3_ Create *Two* EC2 Servers (Bastion Host / Data Migration Server)

1) Tag: cc-bastion-server // subnet: cc-public-main // cc-bastion-ssh(security)
2) Tag: cc-migration // subnet: cc-private-db1 // cc-db-migration
   ** IAM instance profile: db-access-s3
3) Using *SSH Agent* to Access the EC2 instances:
   - eval "$(ssh-agent -s)" → ssh-add ~/.ssh/mykey.pem
   - ssh -A ec2-user@bastionHost_ip & ssh ec2-user@db-migrationEC2_ip

4_ Create RDS instance (AZ1, AZ2):

1) Create **DB subnet Group** (Deciding **where**[subnets] to place the **RDS**!)
   Tag: shopwisedb-subnet-groups // AZ: *eu-west-1a, 1b* //
   subnets: cc-private-db1 / cc-private-db2
2) Create **Database**
   - Engine: MySQL //
   - DB instance **Identifier**[system] & **initial** DB **name**: *shopwisedb*
     Master user/pwd: *dbadmin* / *data7006*!
   - **Connectivity**: shopwisedb-**subnet-groups** BY RDS subnet group
   - VPC Security: cc-rds-instance // AZ: eu-west-1[main]

5_ Migrate sql file from S3 to RDS: Flyway

1) Create **S3** Bucket: Tag: cc-migration-bucket
2) **Upload** the **sql** file(V1___shopwise.sql) *in S3* [cc-migration-bucket]

---

3) **Migrate** the sql file by *Flyway* (access globally)
   - ➔ **Access & Configuration**: Bastion EC2 > On Data Migration EC2
   - ➔ **Install** the **Flyway** on the Data Migration Server and Create **Symbolic** Link.

     - sudo dnf update -y
     - **wget** -qO- https://download.red-gate.com/maven/release/com/redgate/flyway/flyway-commandline/10.13.0/flyway-commandline-10.13.0-linux-x64.tar.gz | tar -xvz
     - sudo **ln -s** `pwd`/flyway-10.13.0/flyway **/usr/local/bin**

   - ➔ Input the **variables** on *Data Migration Server* >>

```
S3_URI= // RDS_ENDPOINT= // RDS_DB_NAME= // RDS_DB_USERNAME= // RDS_DB_PASSWORD=
```

   - ➔ Create a **new directory** to save the sql file & **Copy** the sql file from S3 bucket
     - sudo mkdir migrate
     - sudo aws s3 cp "$S3_URI" migrate/
   - ➔ Command RDS Info configuration & **Migrate** Process >>

```
flyway -url=jdbc:mysql://"$RDS_ENDPOINT":3306/"$RDS_DB_NAME" \
  -user="$RDS_DB_USERNAME" \
 -password="$RDS_DB_PASSWORD" \
  -locations=filesystem:migrate \
  Migrate
```

4) After migration, the migration server should be **Terminated**.

6_ LAMP Server Configuration and Web files Deployment: phpMyAdmin

- **Deployment** of **Web files** on Web Server (Private) through Bastion Server (Public)

1) Create **S3** Bucket and **Upload** the **Web zip file to S3**
2) Prepare **LAMP server**:
- **Install Apache server, MariaDB, and PHP** packages
- *Enable & start Apache & MariaDB*

** File, Directory **Authorisation** Configuration

- **Usermod**: Add ec2-user to apache group
- **Chown**: Assign the ownership(access) of /var/www(recursive)
- **Chmod**: Grant the appropriate permissions to **Directory(2775)** and **Files(0664)**
- Preparation Complete -> **exit** to check the changes

3) Test the LAMP server & Secure the DB
- On Web Server (EC2) , Check the mariadb105-server, php-mysqlnd,
- sudo mysql_secure_installation

4) Install **phpMyAdmin**
- Install phpMyAdmin and place it at **/var/www/html/phpMyAdmin**
- *Start mariadb*

5) **Enable Overrides** for certain Configuration
- /etc/httpd/conf/**httpd.conf** → *AllowOverride All*

6) **Synchronisation** of Web zip file from **S3** and **Update authorisation**
- sudo aws s3 **sync** s3://"$S3_BUCKET_NAME" /var/www/html
- **Unzip** the web files and **locate** the unzipped directory under **/var/www/html/**
- /var/www/html: Grant file, directory authorisation [0664/2775]
- /var/www/html/storage: Grant directory authorisation 2775

Storage: Used for storing various files such as cache, logs, uploaded and session files.

7) Configure **Environment Variables**: /var/www/html/**.env**
- Change **APP_URL** to Load Balancer / **APP_ENV** to production
- **Update RDS** configuration to connect
(DB_HOST: RDS's Endpoint / DB_DATABASE: DB name / RDS Username, Password)

8) Update and **Restart Apache**: sudo service httpd restart

7_ Create Load Balancer (to EC2) and Listener Configuration

1) Issuing **ACM** (AWS **Certificate** Manager): To apply **TLS**(SSL) for ALB's HTTPS Listener
   - Request Public certificate Registering MY DNS (root, subdomain)
   - Create records (CNAME) on my certificate in Route 53

2) Creating a **Target Group** (for Load Balancer)
   ➔ To decide **where to send Traffic** with **Health Check**:
   - Type: Instances / Name: *cc-alb-target-group* / Protocol Port: HTTP
   - Success Codes: 200, 301, 302

3) Creating **Web Server** (EC2)
   - Name*: cc-shopwise-web* // Subnet: cc-private-main
   - IAM instance profile: data-access-in-s3 // Security Group: web-server-security
   - **Registering** Web Server (EC2) in **Target Group**
4) Creating *Application* **Load Balancer**
   - Type: Application // Name: *cc-alb-web* // Internet-facing // Security Group: cc-load-balancer
   - Mappings: **2 Public Subnets** in AZ1, AZ2

- Listeners: HTTP:80 → Forward to: Target Group [*cc-alb-target-group*]

---------- After creation, Add listener(HTTPS) & Changing Routing actions ----------

- **HTTP:80** → *Redirect* to URL(**HTTPS**)
- **HTTPS:443** with **TLS** certificate for secure routing → *Forward* to **Target Group** (**web**-ec2)

8_ Creating A record for Final Destination with HTTPS

1) Changing and Checking *the environment variable*:

➔ Route 53 > Hosted zones > my domain >> Create **A-alias Record** to apply APP_URL's path.

➔ In Web Server(EC2), cd /var/www/html >> **sudo vi.env** >>

- **APP_URL**=https://*A-alias Record*
- **APP_ENV**=production

2) **Forcing** the URL to **HTTPS**:

➔ *cd /var/www/html/app/Providers/* >> **sudo vi AppServiceProvider.php**

- Class AppServiceProvider extends ServiceProvider{ } >> **boot() function** >>
- Adding the command which forces all URL to redirect to HTTPS:
  - if (env('APP_ENV') === 'production')
    {\Illuminate\Support\Facades**URL::forceScheme('https');**}

3) **Reflect all changes** the Web Server Configuration & Final **Check** the Web **connection**:

➔ *sudo service httpd restart*

➔ In web browser → mydogtomsky.com and check the **HTTPS protocol changed**.

9_ Auto-Scaling Preparation

1) Create AMI: To launch the AMI for auto-scaling

➔ Web Server (EC2) >> Actions >> Image and templates > **Create Image >>**
Tag and Snapshots together > Name: cc-shopwise-web-image

➔ Check: Images > **AMIs** & Elastic Block Store > **Snapshots**

2) Create the launch **Template**

- Web Server(EC2): *Terminated*
- Instances >> **Launch Templates** > name: cc-launch-shopwise-web
  Auto Scaling guidance > **enable** (when the purpose is for auto-scaling)
  Applications and OS images > **My AMIs** > Owned by me > *cc-shopwise-web-image*
  Subnets are assigned **automatically** according to **AMI'S original Subnets**.
  Security Group should be **selected**: *cc-shopwise-web*

10_Create the Auto-scaling Group

1) **Choose the Template** and Set up the **Configuration**

- EC2 > Auto Scaling > Creating Auto Scaling Groups > choose the **Template** to *LAUNCH*
- **Subnets**: AZ1, AZ2 // **Load Balancing**: **Choosing** Application Load Balancer & Target Groups
- **Health Checks > enable ^** Turn on Elastic Load Balancing health checks
- **Group size**: 2 // *Scaling*: 1 to 3
- Add Notification: **SNS topic** [recipients: youngandtom2@gmail.com, event types: all]

2) **Check 3 steps**: the Auto-scaling & Health Check
- Check the number of EC2 as same as the group size
- When creating the **EC2** instance, **the target group** of the load balancer is verified *automatically* through **Health Checks**.
- **Routing** the https://**A-alias record**