Setting Up AWS Infrastructure for a Scalable Web Application

This documentation provides comprehensive, step-by-step instructions to set up a custom AWS environment for hosting a scalable web application. The environment includes:

1. **Custom VPC Creation:**
    - Configure a Virtual Private Cloud (VPC) with subnets, gateways, and route tables to securely manage the application's network traffic.
2. **EC2 Instances Setup:**
    - Create two EC2 instances:
        - One instance for hosting the application's API.
        - Another instance for deploying the Node.js code.
    - Both EC2 instances are placed in private subnets, secured within the VPC.
3. **Bastion Host Configuration:**
    - Set up a bastion EC2 instance in a public subnet, providing secure SSH access to the private EC2 instances. This prevents direct public access while allowing authorized users to connect.
4. **S3 Bucket for Static Website Hosting:**
    - Create and configure an S3 bucket to host static website files. The bucket can be configured for public access to serve web pages.
5. **Auto Scaling Group (ASG) and Application Load Balancer (ALB):**
    - Implement Auto Scaling Groups (ASG) to ensure the application can scale based on demand.
    - Use an Application Load Balancer (ALB) to distribute traffic across multiple instances, improving performance and fault tolerance.

For ease of use, this process will use the AWS Management Console as much as possible.

Building and Configuring a Custom VPC

This section creates and configures a custom VPC with 9 subnets, an Internet Gateway, a NAT Gateway, and the necessary route tables to support a scalable infrastructure.

Step 1: Create the VPC

1. **Open the VPC Dashboard**:
    - Login to the AWS Management Console as an IAM User *(not Root User).*
    - Search for "VPC" in the search bar.
    - Select **VPC** from the dropdown menu.
2. **Create a New VPC**:
    - In the VPC Dashboard, click the yellow **Create VPC** button.
    - Leave most of the settings as default, except:
        - Set the **IPv4 CIDR block** to `10.0.0.0/16`.
        - Set the **Number of Availability Zones** to `3`.
    - Click **Create VPC**.

Step 2: Create Subnets (9 Total: 3 Public, 3 Server, 3 Database)

1. **Open the Subnets Menu**:
    - Back in the VPC Dashboard, click **Subnets** from the left sidebar.
    - Click **Create Subnet**.
2. **Assign Subnets to the Custom VPC**:
    - Choose the recently created VPC from the dropdown menu.
    - For each Availability Zone, create the following subnets:
        - **Public Subnets**:
          Name as `public-subnet-1`, `public-subnet-2`, and `public-subnet-3`.
        - **Server Subnets**:
          Name as `server-subnet-1`, `server-subnet-2`, and `server-subnet-3`.
        - **Database Subnets**:
          Name as `database-subnet-1`, `database-subnet-2`, and `database-subnet-3`.
3. **Assign Availability Zones and CIDR Blocks**:
    - For each **subnet-1** (public, server, database), assign **us-east-1a** as the Availability Zone.
    - For each **subnet-2** (public, server, database), assign **us-east-1b** as the Availability Zone.
    - For each **subnet-3** (public, server, database), assign **us-east-1c** as the Availability Zone.
    - Use the following pattern for assigning the **CIDR blocks**:
        - Start with `10.0.0.0/28` for the first subnet.
        - For each subsequent subnet, increase by 16 (e.g., `10.0.0.16/28`, `10.0.0.32/28`, etc).
    - Click **Create Subnet**.
4. **Repeat** Step 2 until 9 total subnets are created.

Step 3: Create the Internet Gateway

1. **Open the Internet Gateways Menu**:
    - Back in the VPC Dashboard, click **Internet Gateways** from the left sidebar.
    - Click **Create Internet Gateway**.
2. **Attach the Internet Gateway to the VPC**:
    - After naming the Internet Gateway, click **Create Internet Gateway**.
    - In the Internet Gateway Dashboard, select the newly created gateway.
    - From the **Actions** dropdown, select **Attach to VPC** and choose the custom VPC.

Step 4: Create the NAT Gateway

1. **Open the NAT Gateways Menu**:
    - In the VPC Dashboard, click **NAT Gateways** from the left sidebar.
    - Click **Create NAT Gateway**.
2. **Configure the NAT Gateway**:
    - Select one of the **public subnets** from the dropdown menu.
    - **Connectivity Type** should be **public**.
    - Choose **Allocate Elastic IP** for the **Elastic IP Allocation ID**.
    - Click **Create NAT Gateway**.

Step 5: Create Route Tables (3 Total: Public, Server, Database)

1. **Create the Route Tables**:
    - In the VPC Dashboard, click **Route Tables**.
    - Click **Create Route Table**.

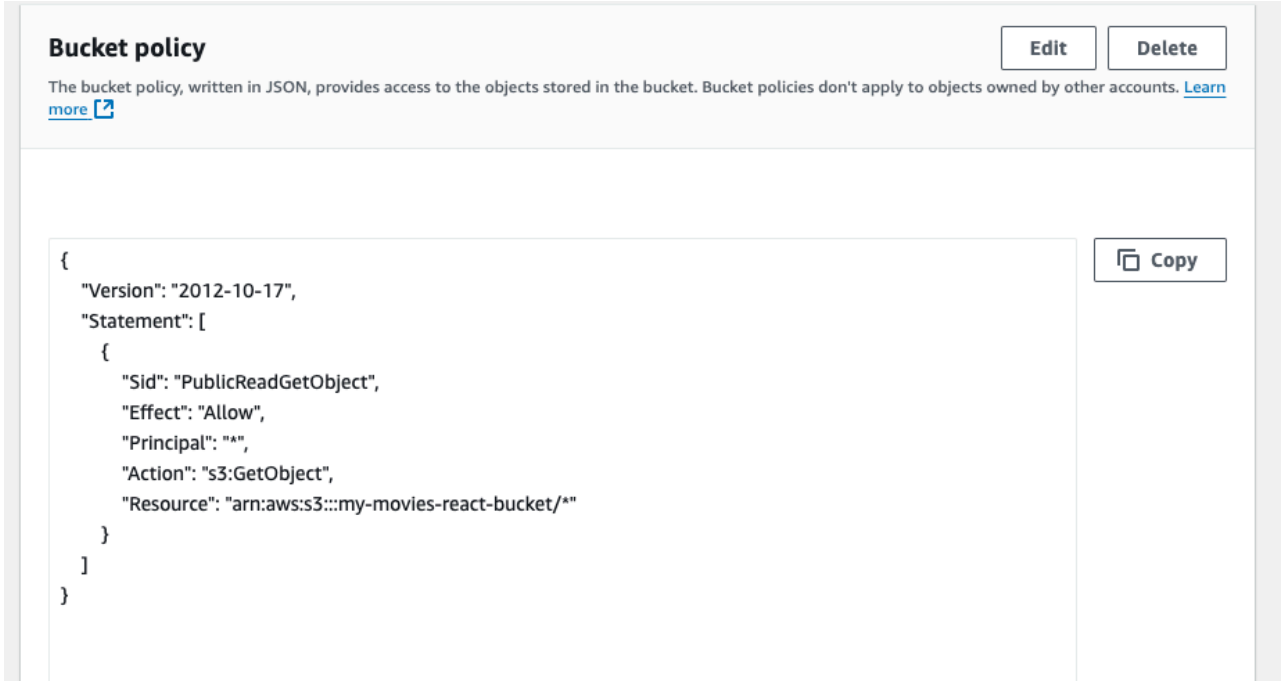
    - Name each route table for its purpose:
        - One for the **Internet Gateway** (public traffic).
        - One for the **NAT Gateway** (private server traffic).
        - One for **local traffic** (database traffic).
2. **Configure Routes for the Public Subnets**:
    - Select the route table for the Internet Gateway.
    - From the **Actions** dropdown, click **Edit Routes**, then **Add Route**.
    - Enter `0.0.0.0/0` as the **Destination** and select the **Internet Gateway** as the **Target**.
    - Click **Save Changes**.
3. **Configure Routes for the Server Subnets**:
    - Select the route table for the NAT Gateway.
    - Follow the same process, using `0.0.0.0/0` as the **Destination** and selecting the **NAT Gateway** as the **Target**.
    - Save changes.
4. **Assign Route Tables to all Subnets**:
    - In the **Subnets** dashboard, select a subnet
    - From the **Actions** dropdown, click **Edit Route Table Association**, then assign:
        - **Public Subnets** to the **Internet Gateway** route table.
        - **Server Subnets** to the **NAT Gateway** route table.
        - **Database Subnets** to the **local** route table.
    - Repeat until all 9 subnets are assigned to the appropriate route table.
5. **Congratulations**, this custom VPC is now fully configured!

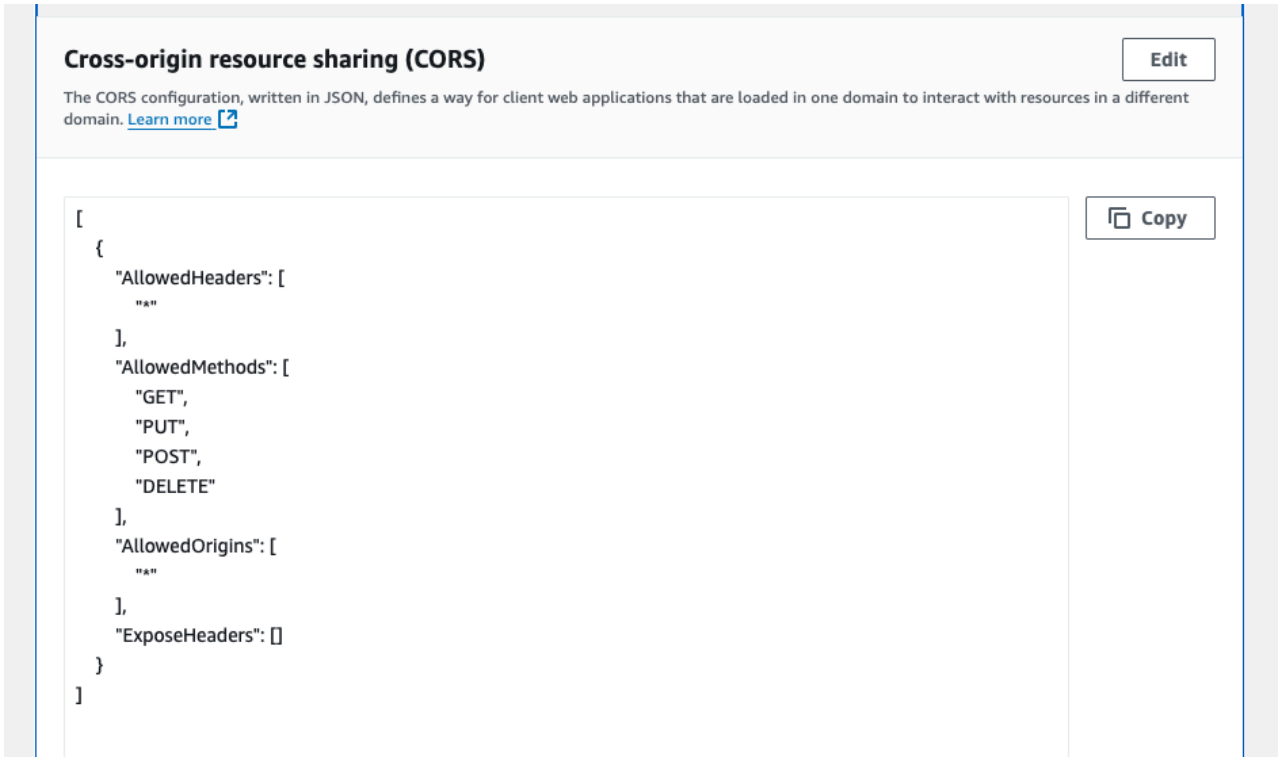
Creating the S3 Bucket for Static Web Hosting

This section creates an S3 Bucket, configures it, and adds the appropriate files so as to statically host the frontend React app.

Step 1: Create the S3 Bucket

1. **Open the S3 Menu**:
    - In the AWS Management Console, find the **Services** dropdown menu in the upper left corner.
    - Select **S3**.
2. **Create a new S3 Bucket**:
    - Click the yellow **Create Bucket** button.
    - Leave most of the settings as default except:
        i. **Disable** (uncheck) the **Block *all* Public Access**.
3. **Configure the S3 Bucket**:
    - In the S3 Dashboard, select the newly created S3 Bucket.
    - Under the **Properties** tab, select **Edit** in the **Static Website Hosting** section.
        i. Select **Enable**.
    - Under the **Permissions** tab, edit the **Bucket Policy** and **CORS** as follows:

- 

- 

4. **Upload files**:
    - Under the **Objects** tab, upload the individual files from the React repository's "dist" folder.

Creating and Configuring the EC2 Instances

This section creates and configures 3 EC2 instances - Bastion (Public), Database, and Server. Each instance has a unique role and purpose within the VPC; once set up, secure access is established between these 3 instances and configurations are made for MongoDB, Node.js, and Nginx.

Step 1: Create the EC2 Instances (3 total: Public (Bastion), Database, Server)

1. **Open the EC2 Menu**:
    - In the AWS Management Console, find the **Services** dropdown menu in the upper left corner.
    - Select **EC2**.
2. **Create a new EC2 Instances**:
    - Click the yellow **Launch an Instance** button.
    - Name each instance based on its purpose: Bastion, Server, Database.
    - Select **Ubuntu** from the **Application and OS (AMI)** section.
    - Leave most of the settings as default except:
        - **Key Pair Settings**:
            - Choose **Create New Key Pair** and note its local location.
            - Use this same key pair for all 3 EC2 instances.
        - **Network Settings**:
            - Click **Edit**.
            - Select the recently created custom VPC.
            - Assign subnets for each instance from the **Subnet** dropdown menu:
                - Public (Bastion) EC2: `public-subnet-1`
                - Database EC2: `database-subnet-1`
                - Server EC2: `server-subnet-1`
            - After naming the **Security Group**, click **Launch Instance**.

Step 2: Configure the Public (Bastion) EC2 Instance

1. **Set the Security Group Rules**:
    - Back in the EC2 Dashboard, select the Server EC2
    - In the **Security** tab, click the **Security Groups** link
    - **Edit Inbound Rules**:
        - SSH (Port 22) from `0.0.0.0/0`
        - Custom TCP (Port 8080) from `0.0.0.0/0`
        - Click **Save Rules**.
    - **Edit Outbound Rules**:
        - SSH (Port 22) to the `<Server EC2 Security Group>`
        - SSH (Port 22) to the `<Database EC2 Security Group>`
        - All (Port all) to `0.0.0.0/0`
            - *This allows communication between the EC2 instances.*
        - Custom TCP (Port 8080) to `0.0.0.0/0`
        - Click **Save Changes**
2. **Allocate the Elastic IP**:
    - In the EC2 Dashboard, select **Elastic IPs** from the left side bar, under **Network & Security**.
    - Select the Elastic IP auto-assigned to this EC2
    - From the **Actions** dropdown menu, select **Associate Elastic IP Address**
    - Select this EC2 Instance, then click **Associate**
    - *This ensures the IP address does not change during build or production.*
3. **Add Public Key in the Terminal**:
    - In the terminal, copy the .pem file (from the key pair created for this project):
        - `scp -i /path/to/your-bastion-key.pem /path/to/your-key.pem ubuntu@<Bastion-EIP>:/home/ubuntu/`

- Set the permissions:
  - `chmod /path/to/your-bastion-key.pem`
- **SSH into the Bastion EC2** Instance:
  - `ssh -i /path/to/your-bastion-key.pem ubuntu@<Bastion-EIP>`
  - *It is now possible to securely access the private EC2's from the bastion instance.*
    - From the Bastion EC2, **SSH into the Database EC2**:
      a. `ssh -i /home/ubuntu/<your-key-pair>.pem ubuntu@<Database-private-IPv4>`
    - To exit back to the Bastion EC2: `exit`
    - From here, **SSH into the Server EC2**:
      a. `ssh -i /home/ubuntu/<your-key-pair>.pem ubuntu@<Server-private-IPv4>`

Step 3: Configure the Database EC2 Instance

1. **Set the Security Group Rules**:
   - Back in the EC2 Dashboard, select the Database EC2
   - In the **Security** tab, click the **Security Groups** link
   - **Edit Inbound Rules**:
     - **Type**: SSH (Port 22) from `<Bastion EC2 Security Group>`
       - *Ensures that this instance is only accessible via the bastion instance, not public internet.*
     - Custom TCP (Port 27017) from `<Server EC2 private IPv4>`
       - *Port 27017 serves MongoDB.*
     - Custom TCP (Port 27017) to `<Bastion EC2 private IPv4>`
     - Click **Save Rules**.
   - **Edit Outbound Rules**:
     - HTTPS (Port 443) to `0.0.0.0/0`
     - HTTP (Port 80) to `0.0.0.0/0`
     - Custom TCP (Port 27107) to `<Server EC2 private IPv4>`
     - Custom TCP (Port 27107) to `<Bastion EC2 private IPv4>`
2. **Securely SSH into the Database EC2 from the Bastion Instance**:
   - In the terminal, SSH into the Bastion instance:
     - `ssh -i /path/to/your-bastion-key.pem ubuntu@<Bastion-EIP>`
   - Then, securely SSH into the Database EC2:
     - `ssh -i /home/ubuntu/<your-key-pair>.pem ubuntu@<Database-private-IPv4>`
3. **Install Necessary Software and Configure**:
   - Update APT: `sudo apt-get update`
   - Install Git: `sudo apt-get install git`
   - Install MongoDB and required packages:
     - `wget -qO - https://www.mongodb.org/static/pgp/server-6.0.asc | sudo tee /usr/share/keyrings/mongodb-archive-keyring.gpg > /dev/null`
     - `echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/6.0`

```
multiverse" | sudo tee
/etc/apt/sources.list.d/mongodb-org-6.0.list
```
        - `sudo apt-get update`
        - `wget http://archive.ubuntu.com/ubuntu/pool/main/o/openssl/libssl1.1_1.1.1f-1ubuntu2_amd64.deb`
        - `sudo dpkg -i libssl1.1_1.1.1f-1ubuntu2_amd64.deb`
        - `sudo apt-get install -y mongodb-org`
    - To configure MongoDB, open the config file:
        - `sudo nano /etc/mongod.conf`
        - Leave most of the file as default except:
            - Change the **bindIP** to `127.0.0.1,<Database EC2 private IPv4>`
            - `fork: true`
            - To exit: “ctrl” + “x”, then “y”, then “enter”
    - Enable MongoDB: `sudo systemctl enable mongod`
    - Start MongoDB: `sudo systemctl start mongod`
    - Check status of MongoDB: `sudo systemctl status mongod`
    - Verify all installations:
        - `git –version`
        - `mongod –version`
        - `sudo dpkg -l | grep mongodb`
    - To exit back to the Bastion EC2: `exit`

Step 4: Configure the Server EC2 Instance

1. **Set the Security Group Rules**:
    - Back in the EC2 Dashboard, select the Server EC2
    - In the **Security** tab, click the **Security Groups** link
    - **Edit Inbound Rules**:
        - SSH (Port: 22) from `<Bastion EC2 Security Group>`
            - *Ensures that this instance is only accessible via the bastion instance, not public internet.*
        - SSH (Port 22) from `0.0.0.0/0`
        - Custom TCP (Port 8080) from `0.0.0.0/0`
        - HTTPS (Port 443) from `0.0.0.0/0`
        - HTTP (Port 80) from `0.0.0.0/0`
        - Click **Save Rules**.
    - **Edit Outbound Rules**:
        - All (Port: all) to `0.0.0.0/0`
            - *This allows communication between the EC2 instances.*
        - Custom TCP (Port 27017) to `<Database EC2 private IPv4>`
        - Click **Save Changes**
2. **Allocate the Elastic IP**:
    - In the EC2 Dashboard, select **Elastic IPs** from the left side bar, under **Network & Security**.
    - Select the Elastic IP auto-assigned to this EC2
    - From the **Actions** dropdown menu, select **Associate Elastic IP Address**

- Select this EC2 Instance, then click **Associate**
- *This ensures the IP address does not change during build or production.*

3. **Securely SSH into the Server EC2 from the Bastion Instance**:
   - In the terminal, SSH into the Bastion instance:
     - `ssh -i /path/to/your-bastion-key.pem ubuntu@<Bastion-EIP>`
   - Then, securely SSH into the Server EC2:
     - `ssh -i /home/ubuntu/<your-key-pair>.pem ubuntu@<Server-private-IPv4>`
4. **Install Necessary Software**:
   - Update APT: `sudo apt-get update`
   - Install Nginx: `sudo apt-get install nginx`
   - Start Nginx service: `sudo systemctl start nginx`
   - Install Git: `sudo apt-get install git`
   - Clone the desired Node.js repository: `git clone <git-repo-url>`
   - Checkout the branch for local use: `git checkout <branch-name>`
   - Prepare NodeJS: `curl -fsSL https://deb.nodesource.com/setup_22.x | sudo -E bash -`
   - Install NodeJS and NPM: `sudo apt-get install -y nodejs`
   - Install PM2: `sudo npm install -g pm2`
   - Verify all installations:
     - `git –version`
     - `node -v`
     - `npm -v`
     - `nginx -v`
5. **Configure Nginx and Setup Proxy**:
   - Begin: `cd <Node.js-repo-name>`
   - Install dependencies and packages: `npm install`
   - Start the app using PM2:
     - `pm2 start <entry-file>`
     - `pm2 startup`
     - `pm2 save`
   - Pass credentials from the MongoDB database server
     - Create .env file: `nano .env`
     - Check that the CONNECTION_URI is the same here as in original .env file
     - Add this .env to .gitignore
   - Also pass these credentials in the nano terminal:
     - `touch .env`
     - `nano .env`
   - Add the same exact configuration as in the original .env file
   - Then:
     - `touch .gitignore`
     - `nano .gitignore`
   - Open the configuration file:
     - `sudo nano /etc/nginx/sites-available/<Node.js-repo-name>`
   - Add this server block to the configuration file in the nano terminal:

```
server {
    listen: 80;
    server_name <ALB-address>;

    location / {
        proxy_pass <S3-bucket-address>;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

        - Save changes by pressing “ctrl” + “o”, then “enter”
        - Exit by pressing “ctrl” + “x”
    - Link these two:
        - `sudo ln -s /etc/nginx/sites-available/MyMovies-Server /etc/nginx/sites-enabled/`
    - Edit nginx.conf: `sudo nano /etc/nginx/nginx.conf`
        - Leave most of the file as default except:
            - `server_names_hash_bucket_size 128;`
    - Test the configuration: `sudo nginx -t`
    - Reload Nginx: `sudo systemctl reload nginx`
    - Update the index.js file: `node index.js`
        - Add `api/` to every CRUD operation
        - `pm2 <start-file>`
    - To exit back to the Bastion EC2: `exit`
6. **Update the Node.js Repository**:
    - Change the index.js connection code to:

```
  .connect(
    process.env.CONNECTION_URI,)
```

    - Check that the CORS logic allows S3 IPc4.
    - Deploy:
        - `git add .`
        - `git -m”<your-commit-message>”`
        - `git push`
    - To check that the backend is running, open the Server EC2 public IPv4 link.
7. **Enable static website hosting**:
    - `aws s3 website s3://<your-S3-bucket-name>/ --index-document index.html --error-document error.html`

Attaching this Infrastructure to an ALB and ASG

This section attaches the current infrastructure to an ALB and sets up an ASG for backend scaling. Both the ALB and ASG are created with security groups, the ALB is configured to handle increased traffic, and the ASG automatically scales EC2 instances as needed. Finally, the React app is updated to interact with the backend in this scalable environment.

Step 1: Create New Security Groups (2 total: 1 for ALB, 1 for ASG)

1. Back in the EC2 Dashboard, click on **Security Groups** from the left side bar.
2. Click **Create Security Group**, be sure to create within the custom VPC.
3. In the **ALB Security Group**,
    - **Edit Inbound Rules**:
        - HTTPS (Port 443) from `0.0.0.0/0`
        - HTTP (Port 80) from `0.0.0.0/0`
    - **Edit Outbound Rules**:
        - All (all) to all
        - HTTP (Port 80) to `<ASG Security Group>`
4. In the **ASG Security Group**,
    - **Edit Inbound Rules**:
        - HTTPS (Port 443) from `<ALB Security Group>`
        - HTTP (Port 80) from `<ALB Security Group>`
        - Custom TCP (Port 8080) from `<ALB Security Group>`
    - **Edit Outbound Rules**:
        - All (all) to all

Step 2: Create ALB (Application Load Balancer)

1. In the EC2 Dashboard, click on **Load Balancers** from the left side bar
2. Click **Create Load Balancer**, select **Application Load Balancer**, click **Create**
3. Leave most of the settings as default except:
    - **Scheme**: **Internet-facing**
    - In **Network Mapping**:
        - Select the custom VPC from the dropdown
        - Select 3 **Public Subnets** from **Availability Zones**
    - In **Security Groups**, choose recently created **ALB Security Group**
    - In **Listeners and Routing**, select **Create Target Group**
    - Click **Create Load Balancer**

Step 3: Create a Launch Template and ASG (Auto Scaling Group)

1. In EC2 Dashboard, select **Launch Templates** from the left side bar.
2. Click **Create Launch Template**
3. Leave most of the settings as default except:
    - Select **Ubuntu** from **Application and OS (AMI)** section.
    - Select same key-pair as used thus far.
    - Select **Don't include in launch template** under **Subnet** dropdown menu.
    - Select **ASG Security Group**, then **Create Launch Template**.
4. Returning to the EC2 Dashboard, select **Auto Scaling Groups** from the left side bar.

5. Click **Create Auto Scaling Group**.
6. From the **Launch Template** dropdown, select the recently created Launch Template.
7. In **Network**:
    - Select the custom VPC from the dropdown menu.
    - Select `server-subnet-1` from the dropdown menu.
8. In **Load Balancing**, select **Attach to an Existing Load Balancer**, select recently created **Target Group**.
9. In **Scaling**, change **Min/Max Desired Capacities** to 3.
10. Leave all other setting as default, then **Create Auto Scaling Group**.

Step 4: Update the React Repository:

1. Change API calls to use ALB DNS

```
fetch(`http://<ALB-address>/users`, {
```

2. Deploy:
    - `git add .`
    - `git -m”<your-commit-message>”`
    - `git push`