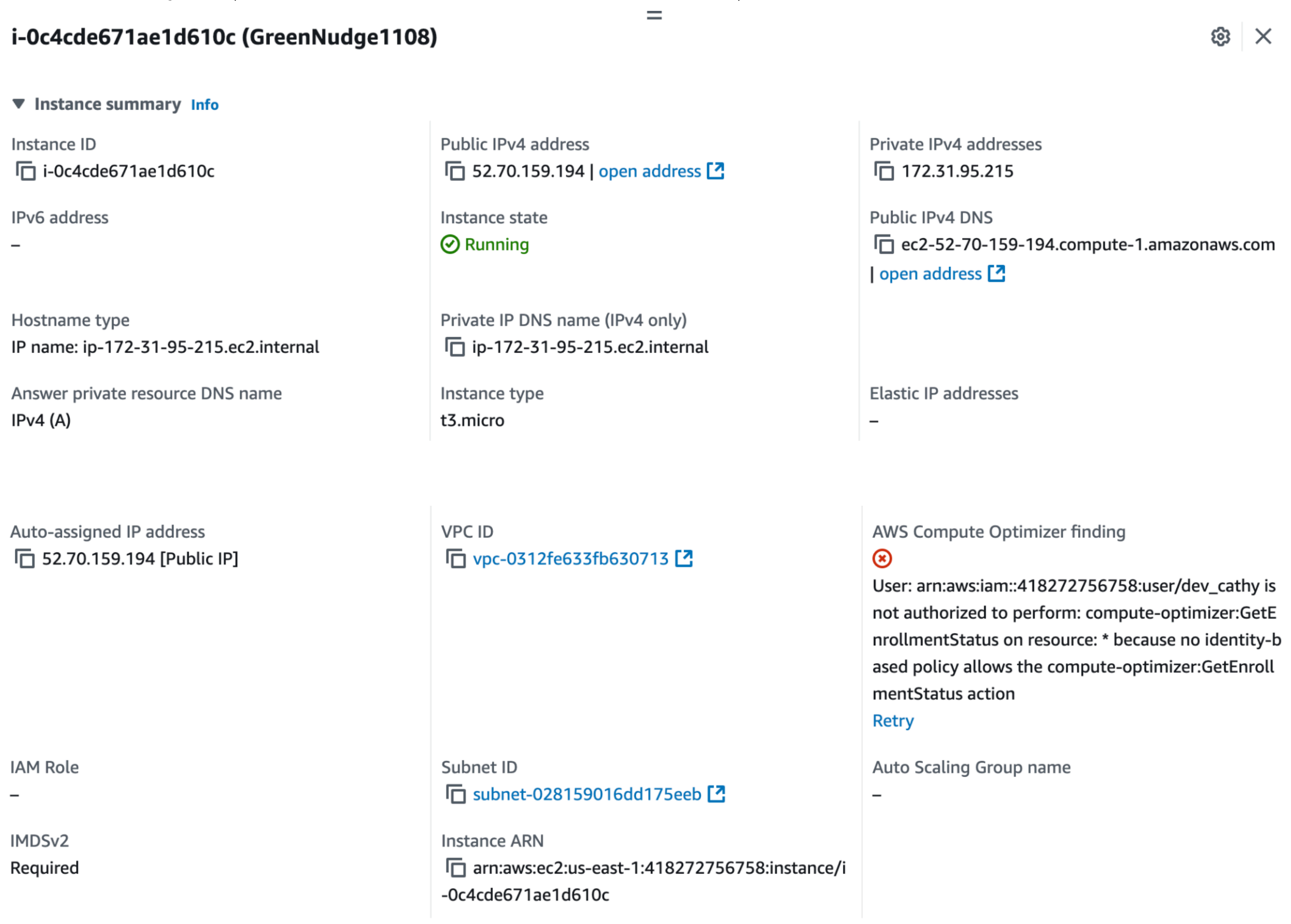
Our EC2's public DNS is: **ec2-52-70-159-194.compute-1.amazonaws.com**

This is also the url of our website

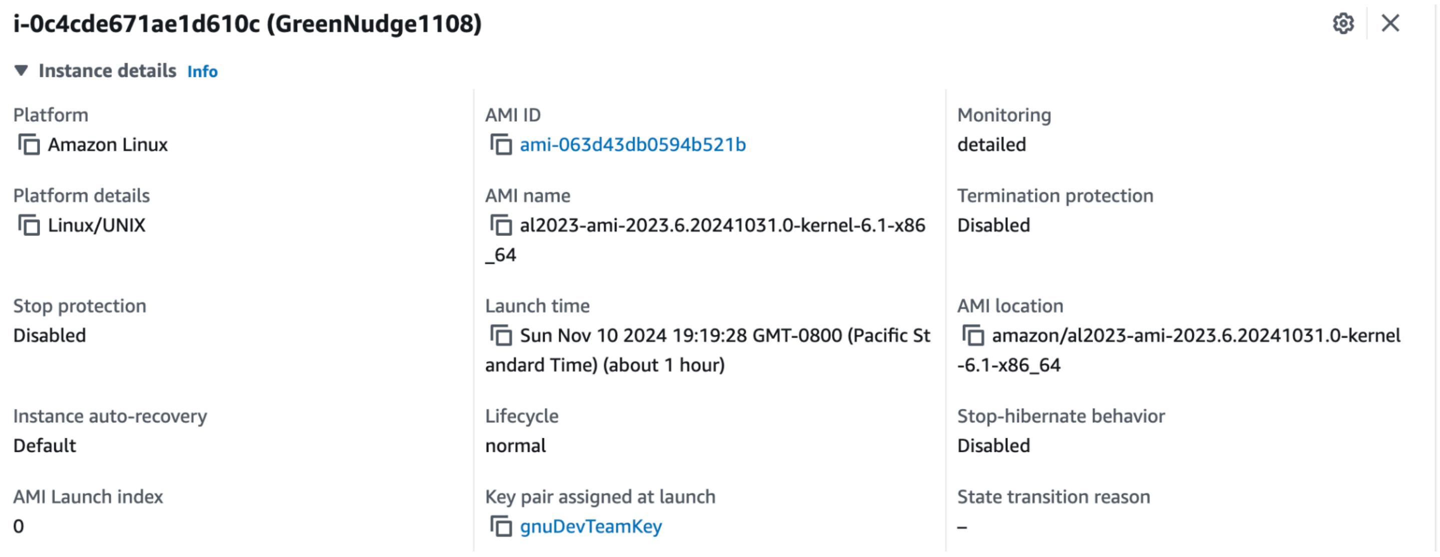
Its summary is (find more info in our ec2 console!):



i-0c4cde671ae1d610c (GreenNudge1108)

▼ **Instance summary** Info

Instance ID i-0c4cde671ae1d610c	Public IPv4 address 52.70.159.194 | open address	Private IPv4 addresses 172.31.95.215
IPv6 address –	Instance state Running	Public IPv4 DNS ec2-52-70-159-194.compute-1.amazonaws.com | open address
Hostname type IP name: ip-172-31-95-215.ec2.internal	Private IP DNS name (IPv4 only) ip-172-31-95-215.ec2.internal	
Answer private resource DNS name IPv4 (A)	Instance type t3.micro	Elastic IP addresses –
Auto-assigned IP address 52.70.159.194 [Public IP]	VPC ID vpc-0312fe633fb630713	AWS Compute Optimizer finding User: arn:aws:iam::418272756758:user/dev_cathy is not authorized to perform: compute-optimizer:GetEnrollmentStatus on resource: * because no identity-based policy allows the compute-optimizer:GetEnrollmentStatus action Retry
IAM Role –	Subnet ID subnet-028159016dd175eeb	Auto Scaling Group name –
IMDSv2 Required	Instance ARN arn:aws:ec2:us-east-1:418272756758:instance/i-0c4cde671ae1d610c	



i-0c4cde671ae1d610c (GreenNudge1108)

▼ **Instance details** Info

Platform Amazon Linux	AMI ID ami-063d43db0594b521b	Monitoring detailed
Platform details Linux/UNIX	AMI name al2023-ami-2023.6.20241031.0-kernel-6.1-x86_64	Termination protection Disabled
Stop protection Disabled	Launch time Sun Nov 10 2024 19:19:28 GMT-0800 (Pacific Standard Time) (about 1 hour)	AMI location amazon/al2023-ami-2023.6.20241031.0-kernel-6.1-x86_64
Instance auto-recovery Default	Lifecycle normal	Stop-hibernate behavior Disabled
AMI Launch index 0	Key pair assigned at launch gnuDevTeamKey	State transition reason –

1. Connect to Your EC2 Instance

First, you'll need to connect to your EC2 instance using SSH. Open your terminal and run:

(the .pem file came from one message from Sneha, in our teams chat)

```
ssh -i /path/to/your-key-pair.pem ec2-user@your-ec2-instance-public-dns
```

2. Update the System

Once connected, update the package lists and install any available updates:

```
sudo yum update -y
```

Copy Only the Build Directory

If you prefer a simpler approach and only need to host the static files, you can just copy the `build` directory to your EC2 instance. This method is straightforward and avoids the need to install npm or other development tools on the server.

1. Build Locally:

```
npm run build
```

2. Copy Build Directory to EC2 **(remember to locate the .pem file and the build folder)**

```
scp -i /path/to/your-key.pem -r build/*
ec2-user@your-ec2-instance-public-dns:/var/www/html/
```

About why we need Nginx:

Hosting your React project on EC2 with Nginx offers **several advantages** over using EC2 alone:

1. Reverse Proxy: Nginx acts as a reverse proxy, which means it can handle incoming requests and distribute them to your application. This helps in managing load and improving performance.
2. Static File Serving: Nginx is highly efficient at serving static files (like your React app's HTML, CSS, and JavaScript). It can do this faster than a Node.js server, which is often used to run React apps.
3. Security: Nginx can add an extra layer of security by handling SSL/TLS termination, which means it can manage HTTPS connections and offload this work from your application server.

4. Caching: Nginx can cache responses, which can significantly reduce the load on your application server and improve response times for your users.
5. Load Balancing: If you scale your application to multiple instances, Nginx can distribute incoming traffic across these instances, ensuring no single server becomes a bottleneck.
6. Configuration Flexibility: Nginx offers extensive configuration options, allowing you to fine-tune how requests are handled, which can be crucial for optimizing performance and security.

Using EC2 alone means you would need to handle all these aspects within your application, which can be less efficient and more complex to manage. Nginx simplifies and optimizes these tasks, making your deployment more robust and scalable.

Another reason is : when EC2 directly handles our project, after I run “npm install” it will freeze , and the CPU utility displays 100%(check CPU monitoring around Nov 11). It means that EC2 the virtual machine is not enough to run our project(possibly due to its memory and CPU of the free tier). So it is best to generate a production build and use a middleware to host the website.

1. Install Nginx

First, you need to install Nginx on your EC2 instance.

For Amazon Linux:

```
sudo yum install nginx -y
```

2. Configure Nginx

Next, configure Nginx to serve your static files. Open the Nginx configuration file in a text editor.

For Amazon Linux:

```
sudo nano /etc/nginx/nginx.conf
```

Add the following configuration to serve your React app:

```
server {
    listen 80;
    server_name your-ec2-instance-public-dns;

    location / {
        root /var/www/html;
        index index.html index.htm;
        try_files $uri $uri/ /index.html;
    }
}
```

Replace `your-ec2-instance-public-dns` with your actual EC2 instance public DNS.

3. Restart Nginx

After updating the configuration, restart Nginx to apply the changes.

```
sudo systemctl restart nginx
```

4. Open HTTP Port in Security Group

Ensure that your EC2 instance's security group allows inbound traffic on port 80 (HTTP).

1. Go to the **Amazon EC2 console**.
2. Select **Security Groups** from the left-hand menu.
3. Find the security group associated with your instance.
4. Edit the inbound rules to allow HTTP traffic on port 80.

5. Access Your App

Open a web browser and navigate to your EC2 instance's public DNS or IP address. You should see your React app running.

Yes, if you copy your new build to the same directory, it will replace the old files with the new ones. This is a common approach and should work fine. Here's how you can do it:

1. Build Your Project:
   - Run `npm run build` to generate the new production build.
2. Transfer the New Build to EC2:
   - Use `scp` to copy the new build to your EC2 instance:

```
scp -i path/to/your-key.pem -r build/ ubuntu@your-ec2-ip:/path/to/your/project
```

   -
3. Overwrite the Existing Files:
   - When you copy the new build to the same directory, it will overwrite the existing files. This ensures that your website serves the latest version.
4. Restart Nginx:
   - Restart Nginx to apply the changes:

```
sudo systemctl restart nginx
```

By following these steps, your new build will replace the old one, and your website will be updated with the latest changes.

To switch your Nginx configuration to use HTTPS, you'll need to:

1. Obtain an SSL Certificate: You can get a free SSL certificate from Let's Encrypt.
2. Install Certbot: Certbot is a tool to automate the process of obtaining and renewing SSL certificates from Let's Encrypt.
3. Update Nginx Configuration: Modify your Nginx configuration to listen on port 443 (HTTPS) and use the SSL certificate.

Since you're using **Amazon Linux 2023**, you can install Certbot using `dnf` and `pip`. Here are the steps:

Step 1: Update Your System

First, ensure your system is up to date:

sudo yum update

Step 2: Install Required Packages

Install the necessary packages for Certbot:

sudo dnf install python3 augeas-libs

Step 3: Set Up a Virtual Environment for Certbot

Create a virtual environment for Certbot:

sudo python3 -m venv /opt/certbot/

Step 4: Install Certbot

Activate the virtual environment and install Certbot:

sudo /opt/certbot/bin/pip install --upgrade pip

```
sudo /opt/certbot/bin/pip install certbot certbot-nginx
```

Step 5: Obtain an SSL Certificate

Run Certbot to obtain and install the SSL certificate:

```
sudo /opt/certbot/bin/certbot --nginx -d your-ec2-instance-public-dns
```

We need to custom url instead of our ec2's public dns address to get a SSL !!!

Follow the prompts to complete the process.

Step 6: Update Nginx Configuration

Modify your Nginx configuration to include the SSL settings. It should look something like this:

```
server {
    listen 80;
    server_name your-ec2-instance-public-dns;

    location / {
        root /var/www/html;
        index index.html index.htm;
        try_files $uri $uri/ /index.html;
    }

    # Redirect HTTP to HTTPS
    if ($scheme != "https") {
        return 301 https://$host$request_uri;
    }
}

server {
    listen 443 ssl;
    server_name your-ec2-instance-public-dns;

    ssl_certificate /etc/letsencrypt/live/your-ec2-instance-public-dns/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/your-ec2-instance-public-dns/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    location / {
        root /var/www/html;
        index index.html index.htm;
        try_files $uri $uri/ /index.html;
    }
}
```

Step 7: Restart Nginx

Finally, restart Nginx to apply the changes:

```
sudo systemctl restart nginx
```