# DEPLOYING AND EXPOSING FIRST APPLICATION TO AWS

---

## OVERVIEW

In this tutorial, we will:

1. Clone a Node.js application from GitHub
2. Run it locally to test
3. Set up AWS IAM user (security best practice)
4. Create EC2 instance (virtual server)
5. Configure the server
6. Deploy the application
7. Expose it to the internet

---

## STEP 1: CLONE THE REPOSITORY

**Command:**

```
git clone https://github.com/verma-kunal/AWS-Session.git
```

**Explanation:**

- Downloads the application code from GitHub to your local machine
- Creates a folder named "AWS-Session"

**What happens:**

```
Cloning into 'AWS-Session'...

remote: Enumerating objects: 50, done.

remote: Counting objects: 100% (50/50), done.

remote: Compressing objects: 100% (35/35), done.

remote: Total 50 (delta 10), reused 45 (delta 8)

Receiving objects: 100% (50/50), done.

Resolving deltas: 100% (10/10), done.
```

---

**STEP 2: NAVIGATE TO PROJECT FOLDER**

**Command:**

```
cd AWS-Session
```

**Explanation:**

- cd: Change directory
- Enters the AWS-Session folder

---

**STEP 3: CHECK FOLDER CONTENTS**

**Command:**

```
ls
```

**Output:**

```
Dockerfile

LICENSE

README.md

client/

package.json

server.js
```

**Understanding the files:**

**Dockerfile:**

- Instructions to build Docker image
- Used for containerization

**LICENSE:**

- Software license information

**README.md:**

- Project documentation
- Instructions on how to use the application

**client/ (folder):**

- Frontend code
- HTML, CSS, JavaScript files

**package.json:**

- Node.js dependencies
- Project metadata
- Scripts to run the application

**server.js:**

- Backend code
- Main server file
- Entry point of the application

---

**STEP 4: OPEN PROJECT IN VS CODE**

**Command:**

```
code .
```

**OR**

**Manually:**

1. Open VS Code
2. File menu
3. Open Folder
4. Select AWS-Session folder

**Why use VS Code?**

- Easy to edit files
- Good for viewing project structure
- Built-in terminal
- Code highlighting

---

**STEP 5: SET UP ENVIRONMENT VARIABLES**

**What are environment variables?** Variables that store sensitive information like:

- API keys
- Database passwords
- Configuration settings

**Why use .env file?**

- Keep secrets separate from code
- Don't commit secrets to Git
- Easy to change settings

---

**Creating .env file:**

**Step 1: Create .env file in root directory**

**In VS Code:**

1. Right-click in file explorer
2. New File
3. Name it: .env

**OR via terminal:**

```
touch .env
```

**Step 2: Add environment variables**

**Open .env file and add:**

```
PORT=3000

DB_HOST=localhost

DB_USER=myuser

DB_PASSWORD=mypassword

API_KEY=your-api-key-here
```

**Note:** These are example variables. Use the actual variables required by your application.

**Step 3: Save the file**

**Important:** Never commit .env file to Git! Add it to .gitignore.

---

## STEP 6: RUN APPLICATION LOCALLY

**Why run locally first?**

- Test if application works
- Debug any issues
- Ensure everything is configured correctly
- Avoid deploying broken code

**Commands:**

**Install dependencies:**

```
npm install
```

**This installs all packages listed in package.json**

**Start the application:**

```
npm start
```

**OR**

```
node server.js
```

**Expected output:**

```
Server is running on port 3000

Connected to database

Application ready!
```

**Test the application:**

**Open browser and go to:**

http://localhost:3000

**If you see the application, it's working!**

---

## STEP 7: PREPARE FOR AWS DEPLOYMENT

**Goal:** Deploy this application to AWS so anyone on the internet can access it.

---

## STEP 8: SIGN IN TO AWS CONSOLE

**Step 1: Go to AWS**

https://aws.amazon.com

**Step 2: Click "Sign in to Console"**

**Step 3: Enter credentials**

- Account ID or alias
- Username (root user)
- Password

**Step 4: You're in AWS Management Console**

---

**STEP 9: UNDERSTANDING IAM (IMPORTANT SECURITY CONCEPT)**

**Problem with Root Account:**

**Root account has:**

- Full access to everything
- Can delete anything
- Can create massive bills
- No restrictions

**Security risk:** If root credentials are leaked, attacker has complete control!

**In an organization:**

- You don't want each member to have root access
- Different people need different permissions
- Need to track who did what
- Need to restrict access

**Solution: AWS IAM (Identity and Access Management)**

**IAM allows you to:**

- Create users with limited permissions
- Assign specific roles
- Restrict access to certain services
- Follow principle of least privilege
- Audit who accessed what

---

## STEP 10: CREATE IAM USER

**Step 1: Navigate to IAM**

**In AWS Console:**

- Search bar (top)
- Type "IAM"
- Click "IAM"

**You'll see IAM Dashboard**

---

**Step 2: Create New User**

**On IAM Dashboard:**

1. Click "Users" (left sidebar)
2. Click "Create user" button (top right)

---

**Step 3: Set User Details**

**User name:** Enter a username (e.g., "developer-john")

**Access type:**

- Check "Provide user access to the AWS Management Console"
- This allows the user to log in via web interface

**Console password:**

- Select "Custom password"
- Enter a strong password (e.g., "MyPassword123!")
- Optional: Check "Users must create a new password at next sign-in" (recommended for security)

**Click "Next"**

---

**Step 4: Set Permissions**

**Three options:**

**Option 1: Add user to group**

- Create groups with specific permissions
- Add user to that group
- Good for organizations with many users

**Option 2: Copy permissions**

- Copy permissions from existing user
- Quick way to replicate access

**Option 3: Attach policies directly**

- Select specific permission policies
- Most straightforward for single user

**We will select Option 3: Attach policies directly**

---

**Step 5: Select Permissions**

**Search for permissions:** Type "AdministratorAccess" in search box

**Check the box next to:**

- AdministratorAccess

**What is AdministratorAccess?**

- Full access to all AWS services
- Same as root user (almost)
- Use only for trusted administrators
- For regular developers, use more restricted policies

**Click "Next"**

---

**Step 6: Review and Create**

**Review page shows:**

- User name
- Console access: Enabled
- Permissions: AdministratorAccess

**Click "Create user"**

**Success message appears!**

---

**Step 7: Save Login Credentials**

**Important page appears with:**

**Console sign-in URL:**

https://your-account-id.signin.aws.amazon.com/console

**User name:**

developer-john

**Console password:**

MyPassword123!

**IMPORTANT:**

- Copy this sign-in URL
- Save it somewhere safe
- You'll use this to log in as IAM user

**Optional:**

- Download .csv file with credentials
- Send credentials to user via secure method

**Click "Return to users list"**

---

**STEP 11: VERIFY IAM USER CREATED**

**On IAM Console:**

- Click "Users" (left sidebar)
- You should see your newly created user in the list

**User details show:**

- User name
- ARN (Amazon Resource Name)
- Creation date

- Last activity

---

**STEP 12: SIGN IN AS IAM USER**

**Step 1: Copy the sign-in URL**

From earlier:

https://your-account-id.signin.aws.amazon.com/console

**Step 2: Open new browser tab/window**

**Why new tab?**

- Keep root session separate
- Best practice to use IAM user for daily work
- Can compare permissions

**Step 3: Paste sign-in URL**

Paste the URL in browser address bar and press Enter

**Step 4: Enter IAM credentials**

**IAM user name:**

developer-john

**Password:**

MyPassword123!

**Click "Sign in"**

**If prompted to change password, create a new one**

**Success! You're now logged in as IAM user**

---

**STEP 13: SEARCH FOR EC2**

**What is EC2?**

**EC2 = Elastic Compute Cloud**

**What it does:**

- Creates virtual servers in the cloud
- You can run applications on these servers

- Similar to having a remote computer
- Pay only for what you use

**In simple terms:**

- It's like renting a computer in AWS data center
- You control it remotely
- Can install anything you want
- Available 24/7

---

**Finding EC2:**

**In AWS Console (top search bar):**

1. Type "EC2"
2. Click "EC2" from results

**You're now in EC2 Dashboard**

---

**STEP 14: LAUNCH EC2 INSTANCE**

**What is an instance?** An instance is a virtual server (virtual machine) running in AWS.

**On EC2 Dashboard:** Click orange button: "Launch instance"

---

**STEP 15: CONFIGURE INSTANCE - NAME**

**Name and tags:**

**Name:** Enter a name for your instance

MyFirstAppServer

**Why name it?**

- Easy to identify among multiple instances
- Helps with organization
- Good practice in production

---

**STEP 16: CHOOSE OPERATING SYSTEM (AMI)**

**AMI = Amazon Machine Image**

**What is AMI?**

- Pre-configured operating system
- Includes basic software
- Starting point for your server

**Available options:**

- Amazon Linux
- Ubuntu
- Windows Server
- Red Hat
- And more...

**We will select: Ubuntu**

**Why Ubuntu?**

- Popular Linux distribution
- Free and open source
- Good documentation
- Easy to use
- Widely used in production

**Select:**

- Ubuntu Server 22.04 LTS (or latest LTS version)
- Free tier eligible (if available)

---

**STEP 17: CHOOSE INSTANCE TYPE**

**What is instance type?** Determines the hardware resources:

- CPU (processing power)
- RAM (memory)
- Network performance
- Storage

**We will select: t2.micro**

**Specifications:**

- 1 vCPU (virtual CPU core)
- 1 GB RAM
- Low to moderate network performance
- Eligible for AWS Free Tier

**Why t2.micro?**

- Free for first 12 months (750 hours/month)
- Good for learning and testing
- Sufficient for small applications
- Can upgrade later if needed

**Free Tier:** AWS gives you 750 hours/month of t2.micro for free for 12 months. 750 hours = 24 hours × 31 days = Always available for free!

---

**STEP 18: KEY PAIR (CRITICAL SECURITY STEP)**

**What is a key pair?**

- Public and private key for SSH authentication
- Used to securely connect to your server
- Like a password, but more secure

**Why needed?** Without key pair, you cannot log in to your EC2 instance!

---

**Creating Key Pair:**

**Click "Create new key pair"**

**Key pair name:**

my-app-server-key

**Key pair type:**

- Select "RSA"

**Private key file format:**

- Select ".pem" (for Linux/Mac/MobaXterm)
- Select ".ppk" if using PuTTY on Windows

**We select: .pem**

**Click "Create key pair"**

---

**What happens:**

**File downloads to your computer:**

my-app-server-key.pem

**CRITICAL:**

- Save this file securely
- You cannot download it again
- If lost, you cannot access your instance
- Never share this file
- Never commit to Git

**Move to safe location:**

**Recommended location:**

~/.ssh/my-app-server-key.pem

**On Windows:**

C:\Users\YourName\.ssh\my-app-server-key.pem

---

**STEP 19: NETWORK SETTINGS (DEFAULT FOR NOW)**

**Keep defaults:**

- Create security group
- Allow SSH traffic from anywhere (0.0.0.0/0)

**We'll configure security group later**

---

**STEP 20: STORAGE (DEFAULT)**

**Keep defaults:**

- 8 GB root volume
- General Purpose SSD (gp2)

**Sufficient for our application**

---

**STEP 21: LAUNCH INSTANCE**

**Review all settings:**

- Name: MyFirstAppServer
- OS: Ubuntu
- Instance type: t2.micro
- Key pair: my-app-server-key

**Click orange button: "Launch instance"**

---

**STEP 22: INSTANCE LAUNCHING**

**Success message appears:**

Successfully initiated launch of instance i-1234567890abcdef0

**Click: "View all instances"**

---

**STEP 23: VERIFY INSTANCE RUNNING**

**On Instances page:**

**You'll see your instance with:**

- Instance ID: i-1234567890abcdef0
- Instance state: Running (green dot)
- Instance type: t2.micro
- Public IPv4 address: (e.g., 54.123.45.67)

**Wait until:**

- Instance state: Running
- Status checks: 2/2 checks passed

**This takes 1-2 minutes**

---

**STEP 24: CONNECT TO INSTANCE**

**Method: SSH (Secure Shell)**

**What is SSH?**

- Protocol to securely connect to remote servers
- Encrypted connection
- Command-line access

---

**Getting Connection Instructions:**

**Step 1: Select your instance**

- Check the box next to your instance

**Step 2: Click "Actions" dropdown (top)**

**Step 3: Select "Connect"**

**Step 4: Choose "SSH client" tab**

**You'll see connection instructions**

---

**STEP 25: OPEN TERMINAL**

**Where is .pem file?** Navigate to where you saved my-app-server-key.pem

**Example:**

```
cd ~/.ssh
```

**OR on Windows:**

```
cd C:\Users\YourName\.ssh
```

---

**STEP 26: SET CORRECT PERMISSIONS (IMPORTANT)**

**Why needed?** SSH requires private key file to have restricted permissions for security.

**Command (Linux/Mac):**

```
chmod 400 my-app-server-key.pem
```

**Explanation:**

- chmod 400: Only owner can read
- Prevents accidental modifications
- SSH requirement

**On Windows (using Git Bash or WSL):** Same command works

---

**STEP 27: CONNECT VIA SSH**

**Command format:**

ssh -i pemfile.pem ubuntu@ip-address

**Actual command (replace with your IP):**

ssh -i my-app-server-key.pem ubuntu@54.123.45.67

**Breakdown:**

- ssh: Secure shell command
- -i: Identity file (key pair)
- my-app-server-key.pem: Your private key
- ubuntu: Username (default for Ubuntu AMI)
- 54.123.45.67: Your instance's public IP address

**Press Enter**

---

**STEP 28: FIRST CONNECTION WARNING**

**You'll see:**

The authenticity of host '54.123.45.67 (54.123.45.67)' can't be established.

ECDSA key fingerprint is SHA256:abcd1234efgh5678ijkl9012mnop3456qrst7890.

Are you sure you want to continue connecting (yes/no/[fingerprint])?

**Type:** yes

**Press Enter**

**What this does:**

- Adds IP address to known hosts file
- Located at ~/.ssh/known_hosts

- Won't ask again for this server
- Security measure to prevent man-in-the-middle attacks

---

**STEP 29: SUCCESSFULLY CONNECTED**

**You'll see:**

```
Welcome to Ubuntu 22.04.1 LTS (GNU/Linux 5.15.0-1028-aws x86_64)


Last login: Thu Nov 7 10:00:00 2024 from 203.0.113.1

ubuntu@ip-172-31-22-45:~$
```

**Success! You're now inside your EC2 instance!**

**Your terminal prompt changed:**

```
ubuntu@ip-172-31-22-45:~$
```

**Meaning:**

- ubuntu: Current user
- ip-172-31-22-45: Server hostname (private IP)
- ~: Current directory (home directory)
- $: Regular user prompt

---

**STEP 30: UPDATE SYSTEM PACKAGES**

**Why update?**

- Get latest security patches
- Update package lists
- Ensure system is up-to-date
- Best practice before installing software

**Command:**

```
sudo apt update
```

**Breakdown:**

- sudo: Execute as superuser (administrator)

- apt: Package manager for Ubuntu
- update: Update package lists from repositories

**Output:**

```
Hit:1 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy InRelease

Get:2 http://us-east-1.ec2.archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]

...

Fetched 15.2 MB in 3s (5,041 kB/s)

Reading package lists... Done

Building dependency tree... Done

All packages are up to date.
```

**This takes 30-60 seconds**

---

**STEP 31: INSTALL GIT**

**Why Git?**

- Need to clone our application code
- Version control tool
- Not installed by default on Ubuntu

**Command:**

```
sudo apt install git -y
```

**Breakdown:**

- sudo: Superuser permission
- apt install: Install package
- git: Package name
- -y: Automatically say "yes" to prompts

**Output:**

```
Reading package lists... Done

Building dependency tree... Done
```

```
Reading state information... Done
The following NEW packages will be installed:
  git
...
Setting up git (1:2.34.1-1ubuntu1.4) ...
```

**Verify installation:**

```
git --version
```

**Output:**

```
git version 2.34.1
```

---

**STEP 32: INSTALL NODE.JS**

**Why Node.js?**

- Our application is built with Node.js
- Need it to run server.js
- JavaScript runtime

**Command:**

```
sudo apt install nodejs -y
```

**Output:**

```
Reading package lists... Done
...
Setting up nodejs (12.22.9~dfsg-1ubuntu3) ...
```

**Verify installation:**

```
node --version
```

**Output:**

```
v12.22.9
```

**OR**

```
v18.12.0
```

(Version depends on Ubuntu repository)

---

**STEP 33: INSTALL NPM**

**What is NPM?**

- NPM = Node Package Manager
- Manages Node.js packages/dependencies
- Required to run npm install

**Command:**

```
sudo apt install npm -y
```

**Output:**

```
Reading package lists... Done

...

Setting up npm (8.5.1~ds-1) ...
```

**Verify installation:**

```
npm --version
```

**Output:**

```
8.5.1
```

---

**STEP 34: CLONE APPLICATION ON SERVER**

**Now clone the same repository on EC2:**

**Command:**

```
git clone https://github.com/verma-kunal/AWS-Session.git
```

**Output:**

```
Cloning into 'AWS-Session'...

remote: Enumerating objects: 50, done.

remote: Counting objects: 100% (50/50), done.

remote: Compressing objects: 100% (35/35), done.

remote: Total 50 (delta 10), reused 45 (delta 8)

Receiving objects: 100% (50/50), done.
```

```
Resolving deltas: 100% (10/10), done.
```

**Verify:**

```
ls
```

**Output:**

```
AWS-Session
```

**Navigate to folder:**

```
cd AWS-Session
```

---

**STEP 35: CREATE .ENV FILE ON SERVER**

**Create file:**

```
touch .env
```

**Verify it's created (including hidden files):**

```
ls -a
```

**Output:**

```
. .. .env .git Dockerfile LICENSE README.md client package.json server.js
```

---

**STEP 36: EDIT .ENV FILE**

**Open in vim editor:**

```
vim .env
```

**Vim editor opens (looks different than VS Code)**

**Enter insert mode:** Press i key

**Now you can type. Add your environment variables:**

```
PORT=3000

DB_HOST=localhost

DB_USER=myuser

DB_PASSWORD=mypassword

API_KEY=your-api-key
```

**Save and exit:**

1. Press Esc key (exit insert mode)
2. Type :wq! (write and quit, force)
3. Press Enter

**Verify content:**

```
cat .env
```

**Output:**

```
PORT=3000

DB_HOST=localhost

...
```

---

**STEP 37: INSTALL DEPENDENCIES**

**Command:**

```
npm install
```

**What this does:**

- Reads package.json
- Downloads all required packages
- Installs them in node_modules folder

**Output:**

```
npm WARN deprecated package1@1.0.0: This package is deprecated

...

added 150 packages, and audited 151 packages in 15s


12 packages are looking for funding

  run `npm fund` for details


found 0 vulnerabilities
```

**This takes 1-2 minutes**

---

**STEP 38: START APPLICATION**

**Command:**

```
npm run start
```

**OR**

```
npm start
```

**Output:**

```
> aws-session@1.0.0 start

> node server.js


Server is running on port 3000

Connected to database

Application ready!
```

**Great! Application is running!**

**BUT...**

**Problem:** You still cannot access it from internet!

**Why?**

- Application is running on port 3000
- EC2 security group blocks port 3000
- Only SSH (port 22) is allowed by default
- Need to open port 3000

---

**STEP 39: EXPOSE PORT 3000**

**What is a security group?**

- Virtual firewall for EC2 instance
- Controls inbound (incoming) and outbound (outgoing) traffic
- By default, blocks all ports except SSH (22)

**We need to:** Allow inbound traffic on port 3000

---

**Opening port 3000:**

**Step 1: Keep application running**

Don't close the terminal! Application needs to keep running.

**Step 2: Open new browser tab**

Go back to AWS Console

**Step 3: Go to your EC2 instance**

EC2 Dashboard → Instances → Select your instance

**Step 4: Go to Security tab**

Click "Security" tab (bottom panel)

**You'll see:**

- Security groups: sg-abc123def456

**Step 5: Click on security group link**

Click the security group ID (e.g., sg-abc123def456)

---

**STEP 40: EDIT INBOUND RULES**

**On Security Group page:**

**Step 1: Click "Inbound rules" tab**

**Step 2: Click "Edit inbound rules" button**

**Current rules:**

| Type | Protocol | Port Range | Source |
|---|---|---|---|
| SSH | TCP | 22 | 0.0.0.0/0 |

**Step 3: Click "Add rule"**

**Step 4: Configure new rule:**

**Type:** Custom TCP

**Protocol:** TCP (auto-filled)

**Port range:** 3000

**Source:**

- Select "Anywhere-IPv4" from dropdown

- Shows as: 0.0.0.0/0

**OR**

**Source:**

- Select "My IP" (more secure, only you can access)
- Shows your current IP address

**We'll use "Anywhere-IPv4" for demo purposes**

**Description (optional):** Node.js application port

**Step 5: Click "Save rules"**

---

**STEP 41: VERIFY RULES SAVED**

**Inbound rules now show:**

| Type | Protocol | Port Range | Source | Description |
|---|---|---|---|---|
| SSH | TCP | 22 | 0.0.0.0/0 | |
| Custom TCP | TCP | 3000 | 0.0.0.0/0 | Node.js application port |

**Success! Port 3000 is now open!**

---

**STEP 42: ACCESS APPLICATION FROM INTERNET**

**Step 1: Get Public IPv4 address**

Go back to EC2 Instances page

**Find your instance:**

- Public IPv4 address: 54.123.45.67 (example)

**Copy this IP address**

---

**Step 2: Open browser**

**In address bar, type:**

http://54.123.45.67:3000

**Format:**

http://PUBLIC-IP:PORT

**Press Enter**

---

**STEP 43: APPLICATION IS LIVE!**

**Success! Your application is now accessible from anywhere in the world!**

**What you see:**

- Your application's home page
- All features working
- Can be accessed by anyone with the URL

**Congratulations! You've successfully:**

1. Created AWS IAM user (security)
2. Launched EC2 instance (virtual server)
3. Configured the server
4. Deployed Node.js application
5. Exposed it to the internet
6. Made it publicly accessible

---

**IMPORTANT NOTES**

**Application will stop if:**

- You close terminal
- You press Ctrl+C
- EC2 instance stops

**To keep application running permanently:**

**Option 1: Use PM2 (Process Manager)**

```
sudo npm install -g pm2

pm2 start server.js

pm2 save

pm2 startup
```

**Option 2: Use nohup**

```
nohup npm start &
```

**Option 3: Use screen**

```
screen -S myapp
npm start
```

(Press Ctrl+A then D to detach)

---

**SECURITY CONSIDERATIONS**

**Current setup is for learning/testing only!**

**For production:**

**1. Don't allow 0.0.0.0/0 on port 3000**

- Use load balancer
- Use NGINX as reverse proxy
- Only expose port 80 (HTTP) or 443 (HTTPS)

**2. Don't use AdministratorAccess**

- Create specific permissions
- Follow principle of least privilege

**3. Use HTTPS, not HTTP**

- Get SSL certificate
- Use AWS Certificate Manager
- Configure HTTPS

**4. Set up monitoring**

- Use CloudWatch
- Set up alarms
- Monitor costs

**5. Regular updates**

```
sudo apt update
sudo apt upgrade
```

**6. Backup your data**

- Create AMI snapshots
- Backup databases
- Version control everything

---

**COST CONSIDERATIONS**

**Free Tier (first 12 months):**

- 750 hours/month of t2.micro
- 5 GB of storage
- 15 GB data transfer out

**After Free Tier:**

- t2.micro: approximately $0.0116/hour = $8.50/month
- Storage: $0.10 per GB-month
- Data transfer: varies

**To avoid unexpected charges:**

- Stop instances when not in use
- Delete unused resources
- Set up billing alarms
- Monitor AWS Cost Explorer

---

**STOPPING INSTANCE TO AVOID CHARGES**

**When you're done testing:**

**Step 1: Go to EC2 Instances**

**Step 2: Select your instance**

**Step 3: Instance State → Stop instance**

**Or to delete permanently:** Instance State → Terminate instance

---

**TROUBLESHOOTING COMMON ISSUES**

**Issue 1: Cannot connect via SSH**

**Check:**

- Correct IP address
- Correct key file
- Key file permissions (chmod 400)
- Security group allows SSH from your IP

**Issue 2: npm install fails**

**Solution:**

```
sudo apt update

sudo apt upgrade

npm cache clean --force

npm install
```

**Issue 3: Application not accessible**

**Check:**

- Application is running (npm start)
- Port 3000 open in security group
- Correct IP address:port
- No firewall on your computer blocking

**Issue 4: Port already in use**

**Solution:**

```
sudo lsof -i :3000

sudo kill -9 <PID>
```

**Issue 5: Permission denied**

**Solution:**

```
sudo chown -R ubuntu:ubuntu /home/ubuntu/AWS-Session
```

---

## NEXT STEPS FOR PRODUCTION

**1. Set up domain name**

- Buy domain from Route 53 or other registrar

- Point to EC2 IP address
- Access via yourapp.com instead of IP

**2. Set up HTTPS**

- Get SSL certificate (free from Let's Encrypt)
- Or use AWS Certificate Manager
- Configure NGINX

**3. Set up CI/CD**

- Use GitHub Actions
- Automatic deployment on git push
- Run tests before deployment

**4. Use load balancer**

- Handle more traffic
- Distribute load across multiple instances
- Better availability

**5. Use managed database**

- RDS for relational databases
- DynamoDB for NoSQL
- Don't run database on same server

**6. Implement logging and monitoring**

- CloudWatch Logs
- Error tracking (Sentry)
- Performance monitoring

---

## SUMMARY OF ENTIRE PROCESS

**Local Setup:**

1. Clone repository from GitHub
2. Create .env file with environment variables
3. Test application locally (npm install, npm start)

**AWS Setup:** 4. Create IAM user (security best practice) 5. Sign in as IAM user 6. Launch EC2 instance (virtual server) 7. Download key pair (.pem file)

**Server Configuration:** 8. Connect to EC2 via SSH 9. Update system packages (sudo apt update) 10. Install Git, Node.js, NPM 11. Clone application on server 12. Create .env file on server 13. Install dependencies (npm install) 14. Start application (npm start)

**Expose to Internet:** 15. Edit security group inbound rules 16. Add rule for port 3000 17. Access via http://PUBLIC-IP:3000

**Result:** Application is live and accessible worldwide!

---

This is your complete guide to deploying a Node.js application on AWS EC2!