DEPLOY BACKEND ON AWS EC2

Step 1: CREATE EC2 INSTANCE

AWS Console → Search EC2 → Create Instance → Name (by choice) → Ubuntu OS → Instance Type: t2.micro (for free) → Create new Key Pair (for SSH) → Leave rest as Default → Create Instance

Step 2: ALLOW PORTS

Instance → Security → Security Group → Inbound Rules → Edit → Enable 443 IPV4 (Type: https), 443 IPV6 (Type: https), 80 IPV4 (Type: http), 80 IPV6 (Type: http), 3000 IPV4 (Type: Custom TCP), 3000 IPV6 (Type: Custom TCP).

NOTE:

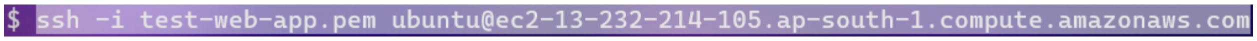
- By default all https goes to port 443. So https://www.google.com is same as https://www.google.com:443
- By default all http goes to port 80.
- 3000 port is being opened just for testing our app here.
- By default in the EC2 port 22 would have been added for SSH.
- Outbound Rules are what IPs can our machine access and inbound rules for what IPs can reach to our machine.

Step 3: CONNECT VIA SSH

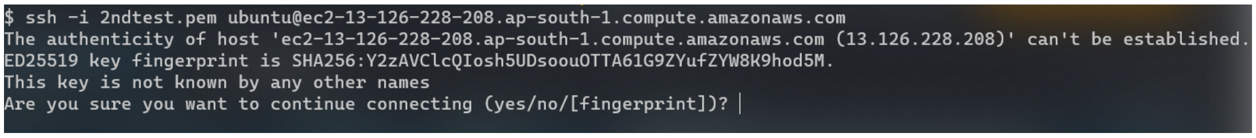
Open Terminal → Copy KEY-PAIR-FILE in same directory → Run command `ssh -i KEY-PAIR-FILE-NAME ubuntu@EC2-URL` → Type 'yes' (needed for 1st time) → Connected

NOTE :

- When we type 'yes' some credentials are appended in the system file '~/.ssh/known_hosts' so that next time it doesn't need to be verified.



```
$ ssh -i test-web-app.pem ubuntu@ec2-13-232-214-105.ap-south-1.compute.amazonaws.com
```

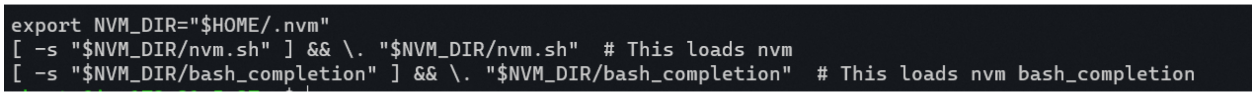


```
$ ssh -i 2ndtest.pem ubuntu@ec2-13-126-228-208.ap-south-1.compute.amazonaws.com
The authenticity of host 'ec2-13-126-228-208.ap-south-1.compute.amazonaws.com (13.126.228.208)' can't be established.
ED25519 key fingerprint is SHA256:Y2zAVClcQIosh5UDsoouOTTA61G9ZYufZYW8K9hod5M.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

Step 4: INSTALL NODE USING NVM

→ Open the website DigitalOcean
→ Option 3 Install Node Using NVM
→ Run command :
 'curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.3/install.sh | bash'
 (Command may change with time, look at the website).

→ Below command will appear at the end of screen. Copy and run it.
 Now NVM is ready to use.

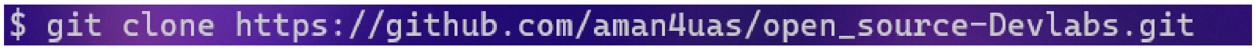


```
export NVM_DIR="$HOME/.nvm"
[ -s "$NVM_DIR/nvm.sh" ] && \. "$NVM_DIR/nvm.sh"  # This loads nvm
[ -s "$NVM_DIR/bash_completion" ] && \. "$NVM_DIR/bash_completion"  # This loads nvm bash_completion
```

→ Run command : 'nvm install node' to install node.

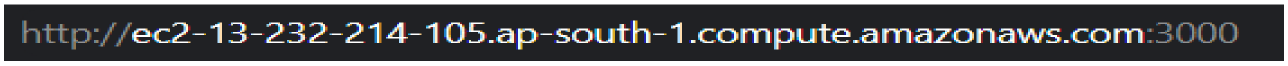
Step 5: RUN SERVER

→ Run command: 'git clone GITHUB-URL'.



```
$ git clone https://github.com/aman4uas/open_source-Devlabs.git
```

→ Run command: 'cd FOLDER-NAME'
→ Run command: 'npm install'
→ Run command: 'npm install -g pm2'
→ Run command: 'pm2 start index.js'
 Can explore more pm2 commands like 'pm2 list', 'pm2 kill'
→ Now our server starts running and can be visited on specified port.



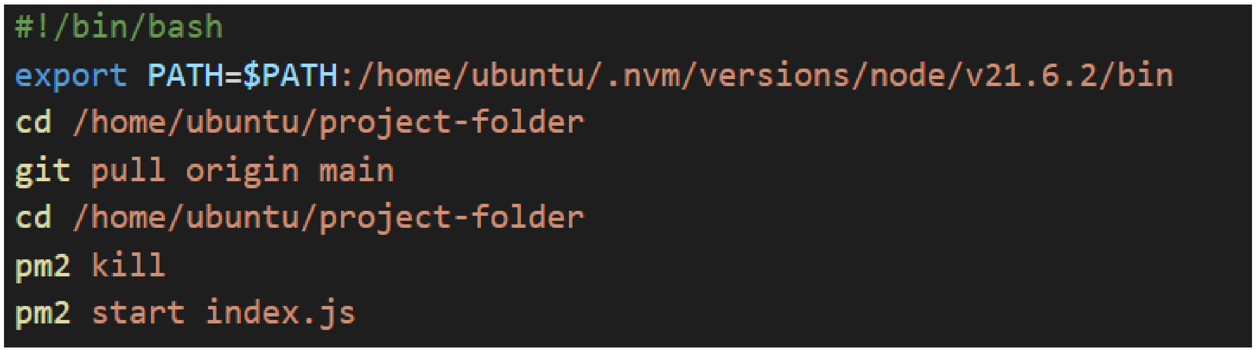
```
http://ec2-13-232-214-105.ap-south-1.compute.amazonaws.com:3000
```

Step 6: SHELL SCRIPT FOR PULLING LATEST CODE

→ Run command: 'cd ~' (Come out from project directory)

→ Create deploy.sh file : ‘touch deploy.sh’
→ Insert the below code, run command: ‘vim deploy.sh’
→ Press ‘i’ to switch to insert mode.
→ Write and copy the similar code as below:

```
#!/bin/bash
export PATH=$PATH:/home/ubuntu/.nvm/versions/node/v21.6.2/bin
cd /home/ubuntu/project-folder
git pull origin main
cd /home/ubuntu/project-folder
pm2 kill
pm2 start index.js
```



```
#!/bin/bash
export PATH=$PATH:/home/ubuntu/.nvm/versions/node/v21.6.2/bin
cd /home/ubuntu/project-folder
git pull origin main
cd /home/ubuntu/project-folder
pm2 kill
pm2 start index.js
```

NOTE:

- Check node version and edit in Line 2
- Check project folder name and edit in Line 3 and 5.

→ Now simple ‘bash deploy.sh’ command will do all lengthy processes at once.

Step 7: AUTO DEPLOY USING GITHUB CI/CD

→ Change directory to root folder of project in local machine.
→ Open terminal.
→ Run command: ‘ssh-keyscan ec2-url >> known_hosts’
This will create a new file named ‘known_hosts’.
It copies required credentials for not prompting yes/no questions while signing in using ssh.
→ Create new folder in the root directory : ‘.github/workflows/’.
→ Create a yaml file with any name e.g., : ‘ci.yaml’.
Run command: ‘touch ./.github/workflows/ci.yaml’

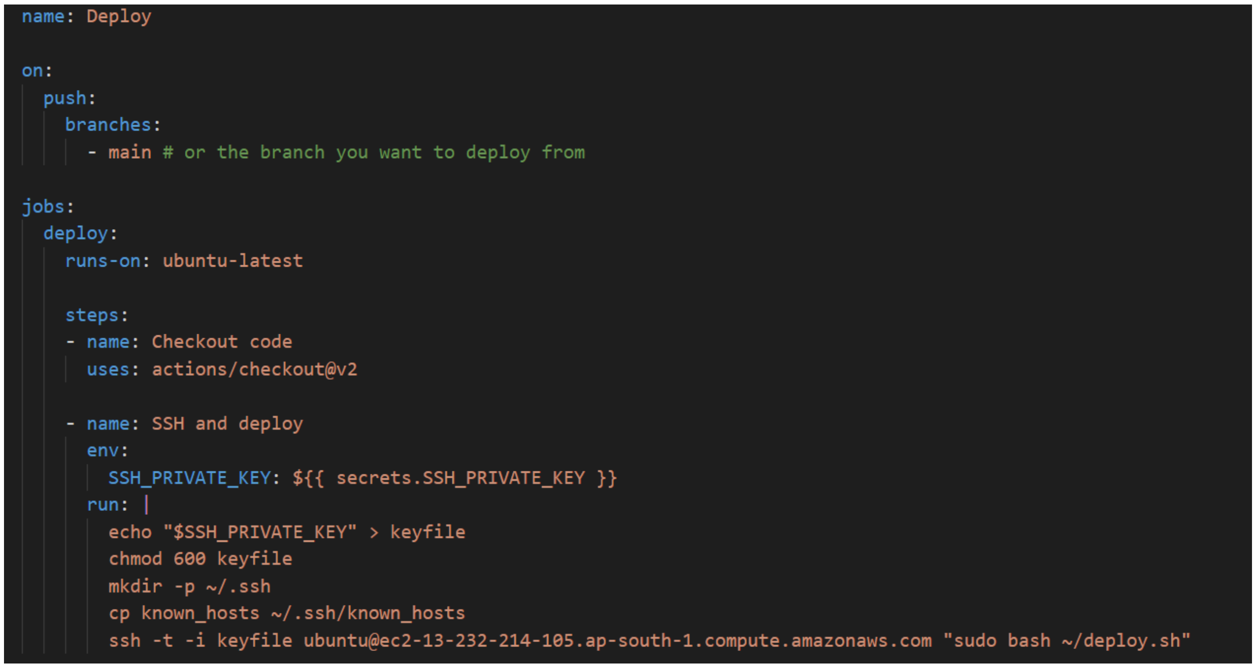
→ Copy the below code:

```
name: Deploy
on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: SSH and deploy
      env:
        SSH_PRIVATE_KEY: ${{ secrets.SSH_PRIVATE_KEY }}
      run: |
        echo "$SSH_PRIVATE_KEY" > keyfile
        chmod 600 keyfile
        mkdir -p ~/.ssh
        cp known_hosts ~/.ssh/known_hosts
        ssh -t -i keyfile ubuntu@EC2-URL "sudo bash ~/deploy.sh"
```



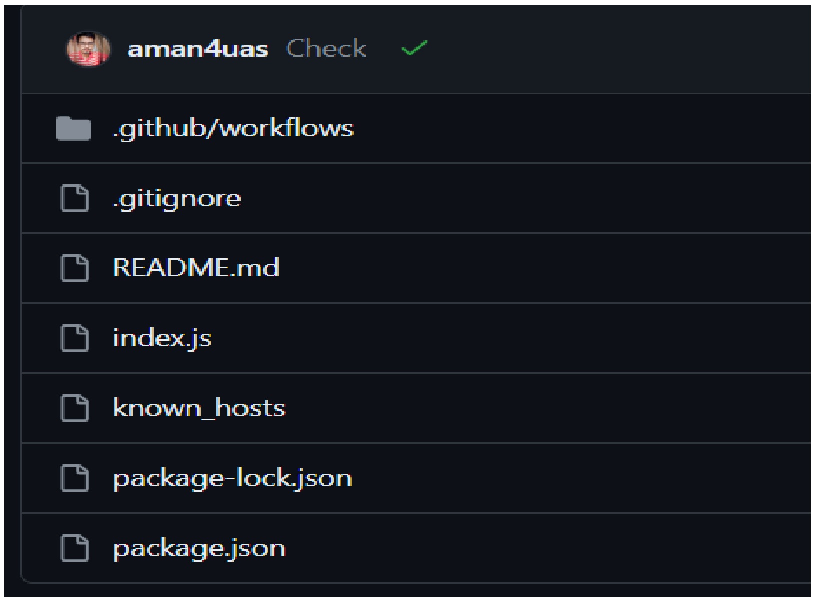
```
name: Deploy

on:
  push:
    branches:
      - main # or the branch you want to deploy from

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: SSH and deploy
      env:
        SSH_PRIVATE_KEY: ${{ secrets.SSH_PRIVATE_KEY }}
      run: |
        echo "$SSH_PRIVATE_KEY" > keyfile
        chmod 600 keyfile
        mkdir -p ~/.ssh
        cp known_hosts ~/.ssh/known_hosts
        ssh -t -i keyfile ubuntu@ec2-13-232-214-105.ap-south-1.compute.amazonaws.com "sudo bash ~/deploy.sh"
```


aman4uas Check ✓

.github/workflows
.gitignore
README.md
index.js
known_hosts
package-lock.json
package.json

NOTE:

- Change EC2-URL from your ec2-url as shown in figure.
- Now let's set SSH_PRIVATE_KEY in GitHub itself.

Step 8: SETTING SSH_PRIVATE_KEY
Project Repository → Settings → Security → Secrets and variables → Actions → Secrets → New Repository Secrets → Name: ‘SSH_PRIVATE_KEY’ → Secret: {Content From KEY_PAIR File} → Add Secret

Step 9: FINALIZING

→ Push all the local changes.
→ HURRAY !! Now auto deploy will be successfully running.

Step 10: POINTING DOMAIN TO SERVER

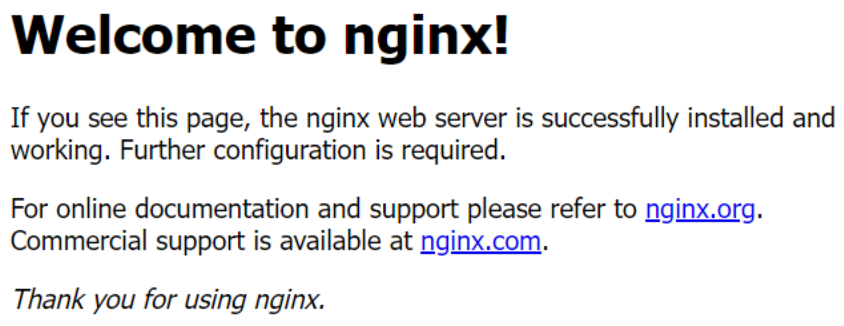
→ Buy domain (Google Domain, GoDaddy, etc.) → Set IP Address in DNS Setting

Step 11: SETUP REVERSE PROXY USING NGINX

NOTE:

- Generally we don't run our services in port 80/443 as we may run different services in a single machine so that our CPU is highly utilized. So we use reverse-proxy. Here we will be using nginx as our reverse proxy which will be running on port 80.
- Reverse vs forward proxy: Forward proxy are those who capture the outgoing requests whereas Reverse proxy capture the incoming requests.

→ Run command: 'sudo apt-get install nginx'
Now visiting the website/IP Address (don't use port) will show something like this.



Welcome to nginx!

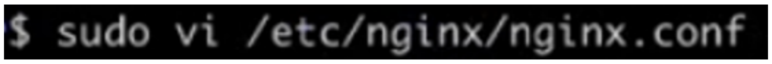
If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

→ Edit the file '/etc/nginx/nginx.conf' with the below code:

```
events {
   worker_connections 1024;
}
http {
   server {
      listen 80;
      server_name YOUR-DOMAIN-NAME;
      location / {
         proxy_pass http://localhost:3000;
         proxy_http_version 1.1;
         proxy_set_header Upgrade $http_upgrade;
         proxy_set_header Connection 'upgrade';
         proxy_set_header Host $host;
         proxy_cache_bypass $http_upgrade;
      }
   }
}
```



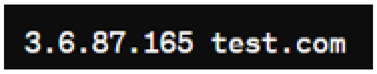
```
$ sudo vi /etc/nginx/nginx.conf
```

NOTE:

- Replace ‘YOUR-DOMAIN-NAME’ with your domain address.
- You can provide a DUMMY-DOMAIN-NAME if you don’t have one.
  e.g., test.com. For this to be working in your system, follow the below step.
  → Goto location: ‘C:\Windows\System32\drivers\etc\hosts’

  ```
  `C:\Windows\System32\drivers\etc\hosts`
  ```

  → Open this ‘hosts’ file in any editor
  → Add a new line: ‘[IP-ADDRESS-SERVER] [DUMMY-DOMAIN-NAME]’

  ```
  3.6.87.165 test.com
  ```

  → Save the file and exit
  This will resolve test.com in your machine to the given IP address.

→ Run command: ‘sudo nginx -s reload’. This will force nginx to reload.

```
$ sudo nginx -s reload
```

→ Hurray !! Now we don’t need to give a port number.

Step 12: CERTIFICATE MANAGEMENT - FOR HTTPS

Goto Cerbot website → Select software as ‘nginx’ and system as ‘Ubuntu 20’ → Follow the steps mentioned.

The steps are: SSH into Server → Don’t install ‘snapd’ as aws already install it → Run command: ‘sudo snap install --classic certbot’ to install Certbot → Run command: ‘sudo ln -s /snap/bin/certbot /usr/bin/certbot’ → Run command: ‘sudo certbot --nginx’ → Enter Mail ID → Enter the website domain number → Installed → Visit: ‘https://yourwebsite.com’ → Now https should work.

NOTE:

- This will edit the ‘nginx.conf’ file according to itself.
- Also, the process fails if the above mentioned ports are not open.
- Also, the process fails if Proper-Public-Domain-Name’ is not provided in the ‘nginx.conf’ file.
- Cert certificates are valid for 3 months only. So maybe we can write github actions for it.

DEPLOY FRONTEND

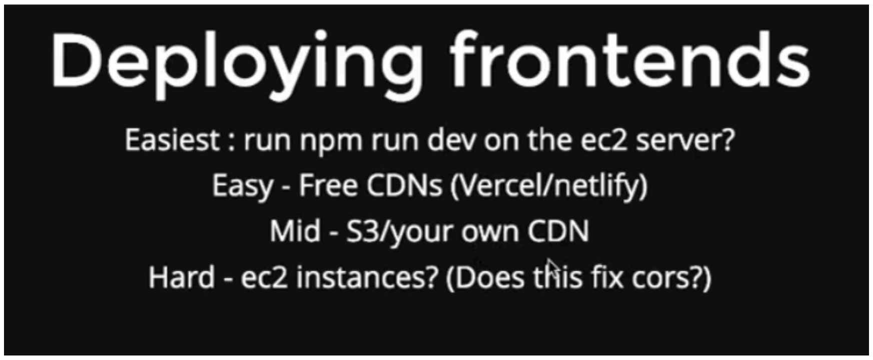
DIFFERENT WAYS OF DEPLOYING FRONTEND:

1. Run npm dev on the EC2 server: But this is really the worst way to deploy as the file produced is not optimized and is especially for development purpose only.

2. Free CDNs (Vercel/Netlify) - CDN serves the builded, static HTML, CSS and JS files.

3. Use S3 as storage (builded files in S3) and use our own CDN

4. Using EC2 instances - This fixes the cors error as the frontend and backend are deployed on the same server.



Deploying frontends

Easiest : run npm run dev on the ec2 server?
Easy - Free CDNs (Vercel/netlify)
Mid - S3/your own CDN
Hard - ec2 instances? (Does this fix cors?)

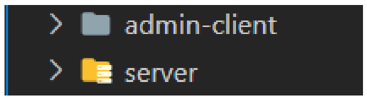
LET'S LEARN EASY STEPS TO SERVE REACT PROJECT:

Goto root directory of React Project → Run command: 'npm run build' → Creates a 'dist' folder in root directory having static HTML, CSS and JS files → Install 'serve' globally by running command: 'npm i -g serve' → goto 'dist' folder → Run command: 'serve' → this serves the static files

DEPLOY USING EC2 INSTANCE:

- Install node → Git clone the Project repo

- Let's suppose we have a project containing backend and frontend both.



- Inside the './server/package.json' we have start command as:
  '`cd ../admin-client && npm run build && rm -rf ../server/public && mkdir -p ../server/public && mv ./dist/* ../server/public && cd ../server && node index.js`'

  What does this command do?

  Goto admin-client folder → Build command to get static files in dist folder → Delete public folder in server directory (basically deletes older files if any) → make new public folder if not exist → move all files from dist folder to server/public folder → goto server → Runs command node index.js to run the server.

  This runs without cors.