AWS Deployment Guide

---------- AWS ----------

Setting up your new server on AWS

1) Create new VPC (don't forget to check region)
    a. Manual checklist
        i. VPC
        ii. Route Table
        iii. Internet Gateway
        iv. Subnet -> explicit association
2) Create 2 Security Groups
    a. Web
        i. HTTP = 80, HTTPS = 443
        ii. Source -> 0.0.0.0/0
    b. SSH
        i. SSH = 22
        ii. Source -> My IP
3) Create new EC2 instance (Ubuntu 22.04)
    a. Use the ED25519 encryption for your key (RSA still works but is older)
    b. Save your Key this is the only time you will be able to download it! Make multiple copies for the same reason.
    c. On MacOS or Brave Browser it might change the .pem key file into a .cer file automatically. Just rename the extension back to .pem
    d. Make sure auto-assign IP is enabled under VPC.
4) Connect manually (ssh -i . . .)
    a. chmod 400 example.pem
        i. Tab to autocomplete is your friend, especially for whitespaces.
    b. ssh -i "example.pem" ubuntu@1.2.3.4
        i. ssh needs to be run in the same directory as your key file
        ii. -i is the flag for IdentityFile (key file)
        iii. The key filename must be in quotes, tab to autocomplete is a good idea
        iv. Username (left of the @) is different for each OS
            1) Ubuntu = ubuntu
            2) Amazon Linux = ec2-user
        v. Location of your instance is to the right of the @ symbol
            1) replace 1.2.3.4 with the dns name or public ip of your instance
5) *Caution* Only continue with the next steps once you are ready to have your instance deployed on your domain name. In other words once you are going to have your instance run 24x7, as Elastic IP accrues charges when NOT in use.
6) Associate an Elastic IP to your instance and verify you can still connect to your instance
7) Setup Route 53
    a. If a Hosted Zone was not created for you when you register create a new Hosted Zone
    b. Verify the Name Servers (ns-XXX.awsdns-XX.com/.net/.org/.co.uk) from your Hosted Zone match the ones in Registered Domains. If not, change the Register Domain's name servers to match the Hosted Zone name servers. DO NOT change the Hosted Zone's Name Servers
    c. Create new A Record for yourdomain.com (Simple Routing) using your new Elastic IP as the value
    d. Then create another new A Record for www.yourdomain.com (Simple Routing) and either
        i. use the Elastic IP for the value, this preserves www.yourdomain.com as a url.
        ii. or use the alias toggle and select the yourdomain.com this forwards www.yourdomain.com to

yourdomain.com.
    1) This works in reverse to.
8) Once connected successfully, create an SSH shortcut using the Elastic IP or DNS

---------- Operating System ----------

Configuring Ubuntu 22.04

1) sudo apt update && sudo apt upgrade -y
2) sudo apt install nginx -y
3) sudo systemctl enable nginx
    a. This is so nginx runs on instance start
4) Install Node Version Manager from https://github.com/nvm-sh/nvm
    a. should look like " curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.XX.X/install.sh | bash "
    b. After installation, exit the terminal and ssh back into the instance to reload the terminal to enable nvm commands.
5) Install your node
    a. nvm install --lts (for the latest LTS version)
    b. nvm install node (for the latest stable release)
    c. nvm install <node version> (to specify a specific node version)
    d. nvm use and the version(or --lts / node) to switch node versions

---

---------- Dynamic Site ----------

Configuring NGINX - Dynamic Site

*Default site html is located at /var/www/html/index.nginx-debian.html

1) sudo nano /etc/nginx/sites-available/<desired website configuration file name here>
        i. Example: sudo nano /etc/nginx/sites-available/yourdomain_com
    a. frontend code:

```
----------< code >----------
        server {
            listen 80;
            server_name yourdomain.com www.yourdomain.com;
            location / {
                proxy_pass http://127.0.0.1:8080;  ##<--change port to match your app##
                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;
                proxy_redirect off;
            }
        }
----------< /code >----------
```

    b. both frontend and backend:

```
----------< code >----------
        server {
```

```
        listen 80;
        server_name yourdomain.com www.yourdomain.com;
        location / {
           proxy_pass http://127.0.0.1:3000;  ##<-- change port to match your frontend app##
           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;
           proxy_redirect off;
         }
        location /api/ {   ##<-- should match the prefix path in your backend routes##
           proxy_pass http://127.0.0.1:3001;  ##<-- change port to match your backend app##
           }
      }
----------< /code >----------
```

2) symlink
   a. sudo ln -s /etc/nginx/sites-available/<website name here> /etc/nginx/sites-enabled/<same website name here>
   b. To check if successful, run "ls /etc/nginx/sites-enabled/" the new link should be cyan. If it is red there was an error, remove it, check your spelling / syntax and try again.
   c. use rm command to remove the symlink in sites-enabled to disable a website.
   d. Do not remove /etc/nginx/sites-enabled/default
3) sudo nginx -t (tests configuration for site configurations in /sites-enabled)
   a. Test OK! = success
   b. if there is feedback (errors) = error
      i. Check for punctuation (; {} . ,)
4) sudo systemctl reload nginx
   a. There should be no feedback to this command.
5) Check the URL, you should see a 502 bad gateway error, this means nginx is working but there is no running website to point to.

Editing your code for local development and remote deployment

1) Edit your project to either remove localhosts from connection URLs or setup turnery statements based on npm run type or define connection vars in npm run statements
   a. ///Axios: localhost:3001/api ----> /api

   b. ///Axios
      baseURL: process.env.REACT_APP_AXIOS === 'production' ? : '/api' : 'http://localhost:3001/api'

      ///package.json
      "start": "REACT_APP_AXIOS='production' react-scripts start",
      "dev": "REACT_APP_AXIOS='development' react-scripts start"

   c. /// Axios
      baseURL: *process.env.REACT_APP_AXIOS*

      /// package.json
      "start": "REACT_APP_AXIOS=/api react-scripts start",
      "dev": "REACT_APP_AXIOS=http://localhost:3001/api react-scripts start"
2) All your routes in the backend should start with /api to work with the above and NGINX's path routing.

3) You should also edit your backend so that CORS origin uses an environmental variable as well.
    a. app.use(cors(
        {
          origin: process.env.CORS_ORIGIN
        }
       ))

    b. and then add CORS_ORIGIN to your .env

Getting your Project onto your Instance - Dynamic Site

1) Git
    a. git clone
        i. *Optional*
        ii. You can use a ssh deploy key to authenticate instead of username/password, but you should not need to since you will only be pulling to the server not pushing to GitHub. If you want to be able to push as well, set up GitHub deploys keys
        iii. You can use CI/CD like AWS CodePipeline & CodeDeploy to automatically send changes to your instance when you push code to GitHub. (Your first CodePipeline is free with AWS Free Tier)
    b. npm install
    c. create .env(s)
        i. remember to update your .env on the server when you change it in the project as git will not
        ii. your .env can include PORT=<port> which will set the port of the app i.e. 8080
        iii. If you are using bcrypt don't forget your SECRET_KEY=mysupersecretkey
        iv. set CORS_ORIGIN to the current environment
            1) local = http://localhost:3000
            2) cloud = http://yourdomain.com
        v. See below for adding your database to the .env
    d. git push - localhost- Push updated code to git
    e. git pull - instance - to pull updated code to the instance
2) setup MongoDB
    a. *VPC Peering does not work with M0 - Free Tier, whitelist the elastic IP instead
    b. You do not to create a Security Group for MongoDB as it is outbound from the instance.
    c. Use the connect button to get your connection string. Connect You Application -> Node.js
        i. Password needs to be URL safe or URI encoded to be use in the connection string
            1) URI Encoder link: https://meyerweb.com/eric/tools/dencoder/
        ii. place in .env
            1) ex. MONGODB_URI=mongodb+srv://dbAdmin:password@clustertest-1wair.mongodb.net/myFirstDatabase?retryWrites=true&w=majority
            2) in the above /myFirstDatabase is the collection name

3) Check that your project works, you can use multiple terminal tabs/windows each logged in with ssh to your instance to monitor the frontend and backend.

Process Manager for when everything is working correctly - Dynamic Site

1) npm install pm2 -g
2) Process Manager runs commands as processes so you do not need to be logged in to run an app and while running it will attempt to keep it alive if there is a failure.
    a. Commands

pm2 start <what you want to start>	Starts an app - run in the same directory as package.json

pm2 start --name *appname* npm -- *start*	Runs npm with the script *start* and labels it as *appname*
pm2 stop *name* || pm2 stop *id*	stop the app with matching name || id number
pm2 restart *name* || pm2 stop *id*	restarts the app with matching name || id number
pm2 start *name* || pm2 stop *id*	starts the stopped app with matching name || id number
pm2 status	Displays current processes and their statuses

May use "all" for all processes i.e. "pm2 stop all"

b. Naming a Process
   i. use the name flag
      1) --name <Name>
   ii. to rename a current process:
      1) pm2 restart *CurrentName* --name *NewName*

3) How to run your app(s)
   a. pm2 start --name backend npm -- start
      i. ** note there is a space between "--" and "start" **
      ii. Need to run in the same directory as package.json in other words the same place you would do "npm start" from
      iii. The "start" at the end is the script name from package.json
   b. repeat for other apps i.e. frontend
4) PM2 Startup - Setting up PM2 to run your apps on boot, this is for when PM2 is setup the way you want.
   a. pm2 startup
      i. This will create a script that starts with "sudo env PATH=…"
   b. Copy and paste the command as instructed from "pm2 startup" in your terminal
   c. pm2 save
      i. After the command you pasted successfully completes, typing "pm2 save" will save the currently running apps. Now you can reboot and pm2 will run your apps on boot.
      ii. Anytime you make changes to pm2 run "pm2 save" to save those changes so they apply on next boot.
5) proceed to enable HTTPS

---------- Static Site ----------

Getting your Project onto your Instance - Static Site

1) Git
   a. Make sure your project is committed to your GitHub.
   b. git clone
      i. Go to your repo on GitHub and click the Code button, copy the string for cloning using the HTTPS method not SSH.
      ii. Git clone should be run in /var/www
         1) sudo git clone <HTTPS code string from GitHub>
         2) After cloning, the final path to your index file should look like:
            a) /var/www/repo-name/index.html
      iii. Optional
      iv. You can setup GitHub SSH deploy keys on your instance to authenticate for pushing to GitHub. However, you should not need to since you will only be pulling to the server not pushing to GitHub. If using SSH keys use the SSH option instead of HTTPS when cloning.
      v. You can use CI/CD like AWS CodePipeline & CodeDeploy to automatically send changes to your instance when you push code to GitHub. (Your first CodePipeline is free with AWS Free Tier)
   c. Use sudo git pull to update code on you instance

Configuring NGINX - Static Site

*Default site html is located at /var/www/html/index.nginx-debian.html

1) Use the terminal command "pwd" in the directory where index.html is to get the directory structure and copy the output.
   a. Example output: /var/www/repo-name/
2) sudo nano /etc/nginx/sites-available/<desired website configuration file name here>
   a. static frontend:

```
----------< code >----------
        server {
           listen 80;
           server_name yourdomain.com www.yourdomain.com;
           location / {
                   root /var/www/repo-name/;  ##<-- directory where index.html is located
                   index index.html;
           }
        }
----------< /code >----------
```

3) symlink
   a. sudo ln -s /etc/nginx/sites-available/<website name here> /etc/nginx/sites-enabled/<same website name here>
   b. To check if successful, run "ls /etc/nginx/sites-enabled/" the new link should be cyan. If it is red there was an error, remove it, check your spelling / syntax and try again.
   c. use rm command to remove the symlink in sites-enabled to disable a website.
   d. Do not remove /etc/nginx/sites-enabled/default
4) sudo nginx -t (tests configuration for site configurations in /sites-enabled)
   a. Test OK! = success
   b. if there is feedback (errors) = error
      i. Check for punctuation (; {} . ,)
5) sudo systemctl reload nginx
   a. There should be no feedback to this command.
6) Check the URL, you should see your website working.

---------- HTTPS ----------

Enable HTTPS

(https://www.digitalocean.com/community/tutorials/how-to-secure-nginx-with-let-s-encrypt-on-ubuntu-20-04)

1) cd ~
2) sudo apt update
3) sudo apt install certbot python3-certbot-nginx
   (# --- is for prompts/response)
   # Y
4) sudo ufw status
   # Status: inactive
   * if you want to enable the firewall*
   sudo ufw allow 'Nginx Full'
5) Make sure the domain is pointing to the server at this point. (both Route 53 and NGINX)
6) sudo certbot --nginx -d example.com -d www.example.com
   # Enter email --> email that is registered to ICANN / Route 53
   # Y to agree to TOS

Share email - Y/N - optional
-- Fails if no domain or cannot verify domain is yours

7) Site should now be reachable on HTTPS
8) Verify Renewal of Certification
    a.
        i. This makes sure the renewal service is running, look for "enabled"
    b. sudo certbot renew --dry-run
        i. This makes sure the service can connect to the renewal server
9) To add a new domain (new nginx config file)
    a. Make sure to have the Route 53 A record and new NGINX config file for the new domain.
    b. Run from step 6 again just change the domain(s)
    c. sudo certbot --nginx -d newdomain.com
10) To add to an existing domain, you will need to add the domain to Route 53 and the NGINX configuration file (server_name entry).
    a. Additionally at the bottom of the NGINX configuration file there is an if statement for redirecting http traffic to https. Copy it just below and change the existing domain to the additional domain.
    b. Finally rerun certbot with the existing domain and the additional domain
        i. sudo certbot --nginx -d currentdomain.com -d expanded.currentdomain.com

---

---------- Sub-Domain ----------

How to setup a Sub-Domain (subdomain.yourdomain.com)

1) Create a new A record in Route 53
    a. Record Name - input your desired subdomain (ex. "subdomain") *wildcards will work
    b. Value - input your Elastic IP address
2) SSH into your server and create a new nginx configuration file in /etc/nginx/sites-available
3) Follow the previous steps from **Configuring NGINX** except for server_name put in the subdomain url
    a. server_name subdomain.yourdomain.com
4) Do not forget to symlink, test and reload nginx
5) Once your site is working properly redo the **Enable HTTPS** section to enable https
    a. sudo certbot --nginx -d subdomain.yourdomain.com

---------- SSH Config ----------

Create SSH shortcuts

1) sudo nano ~/.ssh/config

----------< code >----------
Host <Name>
    User <OS user> *(ubuntu for Ubuntu AMIs || ec2-user for Amazon Linux AMIs)*
    Hostname <public IP or DNS> *(either use the public IP or the DNS address you have associated with the instance)*
    Port <port number> *(default port is 22, not necessary to include if using the default port but it is an option)*
    IdentityFile <ssh key file> *(absolute path to the key file (.pem) for the instance)*
----------< /code >----------

example:

----------< code >----------
Host MySite
    User ubuntu
    Hostname www.yourdomain.com
    IdentityFile ~/aws-keys/my-key.pem
----------< /code >----------

1) how to use:
    a. ssh <Host Name> (case sensitive)
    b. example:
        i. ssh MySite

Configuring SSH Timeout

To prevent a timeout from occuring when using SSH

1) sudo nano /etc/ssh/ssh_config
2) at the very bottom of the file add:
    a. ServerAliveInterval 60

---------- Terminal Commands ----------

Resource Monitoring

top	Show real time processes
htop	Show real time processes with ASCII graphics

Directory Operations

pwd	Show current directory
mkdir *dir*	Make directory *dir*
cd dir	Change directory to dir
cd ..	Go up a directory
ls	List files *Options* -a Show all (including hidden) -R Recursive list -r Reverse order -t Sort by last modified -S Sort by file size -l Long listing format -1 One file per line -m Comma-separated output -Q Quoted output

File Operations

touch file1	Create file1
cat file1 file2	Concatenate files and output
cp file1 file2	Copy file1 to file2 *Options* -r recursive - copy folders and their contents
mv file1 file2	Move file1 to file2 *Options* -r recursive - copy folders and their contents
rm file1	Delete file1 *Options* -r recursive -f remove folders

Service Management

sudo systemctl status *application*	shows the status of the service *application*
sudo systemctl start *application*	starts the service *application*
sudo systemctl stop *application*	stops the service *application*
sudo systemctl restart *application*	restarts the service *application*
sudo systemctl reload *application*	reloads the configuration files for the service *application*
sudo systemctl enable *application*	starts the service *application* at boot
sudo systemctl disable *application*	stops the service *application* from running at boot

---

---------- Situational Commands ----------

---- Port in Use Error ----

If the server is giving an error about a port already in use it means that you already have an app running on that port. Only one app per port is allowed.
First make sure you are not running any other apps in another window or pm2 that could be the culprit. If you are still getting the error do the following commands to kill the app.

1) To list the active apps that are using the port
    a. sudo lsof -t -i:<port>
        i. sudo lsof -t -i:3000
        ii. outputs a process id number (pid) that you will need for killing the app
2) Kill the app
    a. sudo kill -9 <pid>
        i. sudo kill -9 12345
    b. pid comes from the first command

---- Out of Memory ----

If you are getting out of memory issues like an npm install freezing, you may be running out of memory. We can alleviate this by adding a swap file. A swap file basically uses some storage as memory, since storage is much slower than memory this should used for temporary memory issues, like for install or startup issues.

1) Create the swap file
    a. sudo dd if=/dev/zero of=/swapfile bs=128M count=8
2) Update the read and write permissions for the swap file:
    a. sudo chmod 600 /swapfile
3) Set up a Linux swap area
    a. sudo mkswap /swapfile
4) Make the swap file available for immediate use by adding the swap file to swap space
    a. sudo swapon /swapfile
5) Verify that the procedure was successful
    a. sudo swapon -s
6) If you are just having problems with installing stop here and do your install. A reboot will then clear out the swap file.
7) *If you need to keep the swap file we can start the swap file at boot time by editing the /etc/fstab file.
    a. Open the file in the editor:
        i. sudo nano /etc/fstab
    b. Add the following new line at the end of the file, save the file, and then exit:
        i. /swapfile swap swap defaults 0 0

---------- Example NGINX configs ----------

---

single app only (i.e. frontend):

```
server {
   listen 80;
   server_name yourdomain.com www.yourdomain.com;
   location / {
      proxy_pass http://127.0.0.1:8080;
      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;
      proxy_redirect off;
    }
}
```

**the port (number after the colon in the ip address -- 8080 here) must match the port you are running the app on.*

frontend and backend:

```
server {
   listen 80;
   server_name yourdomain.com www.yourdomain.com;
   location / {
      proxy_pass http://127.0.0.1:3000;
      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;
        proxy_redirect off;
    }
  location /api/ {
        proxy_pass http://127.0.0.1:3001;
        }
}
```

**/api/ must be start all backend routes i.e.* www.example.com/api/users?=. . .

static website:

```
server {
   listen 80;
   server_name yourdomain.com www.yourdomain.com;
      location / {
            root /var/www/repo-name/;
            index index.html;
      }
}
```

OR

```
server_name yourdomain.com www.yourdomain.com;
     location / {
           root /var/www/repo-name/;
           try_files $uri /index.html;
     }
```

*root *points to the folder that contains the index.html of the project*
*index *points to the index file, default is index.html*
**Default site html is located at /var/www/html/index.nginx-debian.html*