

How to set up a domain for your React app, an AWS EC2 instance, SSL cert, HTTP to HTTPS redirect and React client-side routing(React acts different once deployed).

This video, by Cloud Guru, is perfect for creating/adding a SSL certificate, making your site use HTTPS: <https://www.youtube.com/watch?v=YYleHdvCUv8&t=416s>

Domain Name: For me, purchasing the Domain name from AWS made the process faster and less confusing, as used in the video above.

Creating Load Balancers:

The video touches very little on setting up a load balancer, but it was pretty self explanatory once you create one for your application. This is what mine looked like after created and linked to my EC2 instance:

The screenshot displays the AWS Management Console interface for a Load Balancer. At the top, there's a 'Create Load Balancer' button and an 'Actions' dropdown. Below this is a search bar and a table listing the load balancers. The table has columns for Name, DNS name, State, VPC ID, Availability Zones, Type, Created At, and Monitoring. One load balancer is listed: JTaylorEducationDCAPBalancer, which is in an 'active' state, associated with VPC ID vpc-b4b36cdf, and located in us-east-2a and us-east-2c availability zones. Below the table, the console shows the details for the selected load balancer, JTaylorEducationDCAPBalancer. The 'Description' tab is active, showing the 'Basic Configuration' section. The configuration details are as follows:

Property	Value
Name	JTaylorEducationDCAPBalancer
ARN	arn:aws:elasticloadbalancing:us-east-2:860430903381:loadbalancer/app/JTaylorEducationDCAPBalancer/id359185c84557cb7
DNS name	JTaylorEducationDCAPBalancer-204209007.us-east-2.elb.amazonaws.com (A Record)
State	active
Type	application
Scheme	internet-facing
IP address type	ipv4

EC2(Front End Only):

Setup:

```
cd {{Project Name}}
echo node_modules/ > .gitignore
cd client
npm run build
```

```
rmdir /s .git
del /f .gitignore
```

```
cd ..
git init
git add .
git commit -m "initial commit"
git remote add {{ Your Repo }}
git push -u origin master
```

AWS:

Create an AWS EC2 instance using Ubuntu 18.04 LTS (Free)

Step 6 is the only one to modify:

SSH = MyIP

HTTP = Anywhere

HTTPS = Anywhere

Once instance is created, press connect, you will then connect using this command:

```
ssh -i "keyname.pem" ubuntu@ec2-XXX-XXX-XXX-XXX.compute-1.amazonaws.com
```

```
sudo apt update
```

```
sudo apt install nodejs npm nginx git -y
```

```
nodejs -v
```

```
# this should print out version 8.10.0
```

```
curl -sL https://deb.nodesource.com/setup_10.x -o nodesource_setup.sh
```

```
sudo bash nodesource_setup.sh
```

```
sudo apt install nodejs
```

```
nodejs -v
```

```
# this should now print out version 10.19.0
```

```
sudo apt install build-essential
```

```
git clone {{ Your Repo link }}
cd ~/$repoName/client
sudo rm -rf /var/www/html
sudo mv build /var/www/html
sudo service nginx restart
sudo grep -rl localhost /var/www/html | xargs sed -i 's/http:\\/\\/localhost:8000//g'
```

Front end is now ready on the EC2 instance link.

Solution for HTTP to HTTPS redirecting:

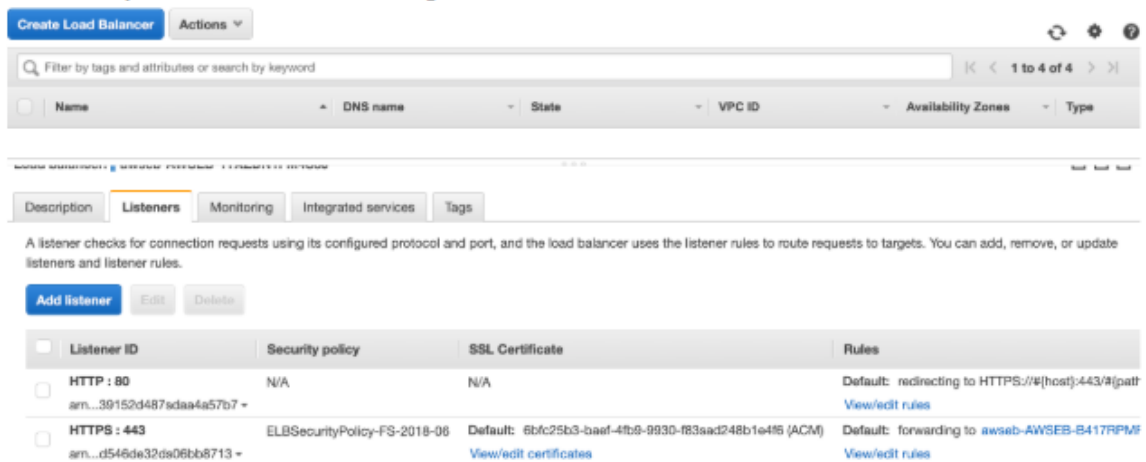
This solution only work with application load balancer, not classic load balancer.

On the console go to **EC2 > Load balancers > your load balancer > listeners**

Here you should have 2 rules: HTTP : 80 and HTTPS : 443

You just have to:

- edit the HTTP : 80 rule
- remove the forward rule and add a redirect rule to port 443
- save and you should see something like:



The screenshot shows the AWS Management Console interface for an Application Load Balancer. The 'Listeners' tab is selected, showing a table of listeners. The 'HTTP : 80' listener is highlighted, and its 'Rules' column shows a default rule redirecting to the 'HTTPS : 443' listener. The 'HTTPS : 443' listener is also shown with its own rule forwarding to an Amazon S3 bucket.

Listener ID	Security policy	SSL Certificate	Rules
HTTP : 80 arn:...39152d487cdaa4a57b7	N/A	N/A	Default: redirecting to HTTPS://#[host]:443/#[path] View/edit rules
HTTPS : 443 arn...d546de32ds06bb8713	ELBSecurityPolicy-FS-2018-06	Default: 6bfc25b3-baef-4fb9-9930-f83aad248b1e486 (ACM) View/edit certificates	Default: forwarding to awsseb-AWSEB-B417RPMF View/edit rules

REACT Routing with deployment

React router acts differently once deployed and should look something like this instead of using @reach/router, using HashRouter as Router, Route, and Switch from 'react-router-dom':

App.js

```
import React from 'react';
import Main from './views/Main';
import Register from './views/Register';
import {HashRouter as Router, Route, Switch} from 'react-router-dom';

function App() {
  return (
    <div className='App'>
      <Router>
        <div>
          <Switch>
            <Route path="/register">
              <Register />
            </Route>
            <Route path="/">
              <Main />
            </Route>
          </Switch>
        </div>
      </Router>
    </div>
  );
}
export default App;
```

This isn't the only way, nor the best way for it to work, but it's the easiest way (for me it was). To navigate between views, I used:

```
import { useHistory } from 'react-router-dom';

let history = useHistory();

history.push("/some-route")
```

This method does add a hashtag in between your route names in the url.