HTTPS: Make the Application Speak HTTPS

We'll cover the following

Objective

- Migrate our endpoint from HTTP to HTTPS.

Steps

- Make the application speak HTTPS.
- Remove the HTTP endpoint.

Making our application speak HTTPS

Let’s update our application to listen to HTTPS requests on port 8443. We’re going to make the application use the self-signed certificate that the EC2 instance generated in the `UserData` script when it came online.

```
const { hostname } = require('os');
const http = require('http');
const https = require('https');
const fs = require('fs');

const STACK_NAME = process.env.STACK_NAME || "Unknown Stack";
const port = 8080;
const httpsPort = 8443;
const httpsKey = '../keys/key.pem'
const httpsCert = '../keys/cert.pem'

if (fs.existsSync(httpsKey) && fs.existsSync(httpsCert)) {
  console.log('Starting https server')
  const message = `Hello HTTPS World from ${hostname()} in ${STACK_NAME}\n`;
  const options = { key: fs.readFileSync(httpsKey), cert: fs.readFileSync(httpsCert) };
  const server = https.createServer(options, (req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end(message);
  });

  server.listen(httpsPort, hostname, () => {
    console.log(`Server running at http://${hostname()}:${httpsPort}/`);
  });
}

console.log('Starting http server')
const message = `Hello HTTP World from ${hostname()} in ${STACK_NAME}\n`;
const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end(message);
});
server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname()}:${port}/`);
});
```

server.js

Line #3: Import the `https` node library.

Line #8: We'll use 8443 as our local HTTPS port number.

Line #9 and #10: Path to the key and certificate generated in the instance launch script.

Line #12: For a graceful roll-out, we'll check for the key and certificate before launching the HTTPS server.

Line #16: We launch an HTTPS server if we have a key and certificate.

Line #28: We continue to launch our HTTP server until all traffic has been migrated.

And let's push this change to GitHub.

```
git add server.js
git commit -m "Add HTTPS listener to the application"
git push
```

terminal

> **Note:** All the code has been already added and we are pushing it on our repository as well.

This code requires the following API keys to execute:

This code requires the following API keys to execute:

username	Not Specified...
AWS_ACCESS_KE...	Not Specified...
AWS_SECRET_AC...	Not Specified...
AWS_REGION	us-east-1
Github_Token	Not Specified...

```
{
  "name": "aws-bootstrap",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "start": "node ./node_modules/pm2/bin/pm2 start ./server.js --name hello_aws --log ../logs/app
    "stop": "node ./node_modules/pm2/bin/pm2 stop hello_aws",
    "build": "echo 'Building...'"
  },
  "dependencies": {
    "pm2": "^4.2.0"
  }
}
```

Now let's wait for the deployment to go through. Our HTTP and HTTPS endpoints should then start working.

```
for run in {1..20}; do curl -s http://staging.the-good-parts.com; done | sort | uniq -c
10 Hello HTTP World from ip-10-0-12-230.ec2.internal in awsbootstrap-Staging-10LP6MF0TQC9Y
10 Hello HTTP World from ip-10-0-85-169.ec2.internal in awsbootstrap-Staging-10LP6MF0TQC9Y
```

terminal

```
for run in {1..20}; do curl -s https://staging.the-good-parts.com; done | sort | uniq -c
10 Hello HTTPS World from ip-10-0-12-230.ec2.internal in awsbootstrap-Staging-10LP6MF0TQC9Y
10 Hello HTTPS World from ip-10-0-85-169.ec2.internal in awsbootstrap-Staging-10LP6MF0TQC9Y
```

terminal

```
for run in {1..20}; do curl -s http://prod.the-good-parts.com; done | sort | uniq -c
9 Hello HTTP World from ip-10-0-116-176.ec2.internal in awsbootstrap-Prod-1PT61TNHUQWTE
11 Hello HTTP World from ip-10-0-30-144.ec2.internal in awsbootstrap-Prod-1PT61TNHUQWTE
```

terminal

```
for run in {1..20}; do curl -s https://prod.the-good-parts.com; done | sort | uniq -c
10 Hello HTTPS World from ip-10-0-116-176.ec2.internal in awsbootstrap-Prod-1PT61TNHUQWTE
10 Hello HTTPS World from ip-10-0-30-144.ec2.internal in awsbootstrap-Prod-1PT61TNHUQWTE
```

terminal

Removing the HTTP endpoint

If this were a production migration, we would first wait for our users to migrate from the HTTP endpoint to HTTPS. Once that happens, we would tear down the HTTP infrastructure.

Let’s start by removing all the resources and outputs for the HTTP traffic and endpoints. We have to remove the endpoints before we modify the application code. To do otherwise would lead to failed health checks.

Let’s remove the following from `stage.yml`:

- from `Resources`
    - from `SecurityGroup`
        - from `SecurityGroupIngress`
            - the section for traffic on port `8080`
            - the section for traffic on port `80`
    - from `ScalingGroup`
        - the `TargetGroupARNs` entry for `LoadBalancerTargetGroup`
    - the entire `LoadBalancerListener` resource
    - the entire `LoadBalancerTargetGroup` resource
- from `Outputs`
    - the `LBEndpoint` output

And let’s remove the following from `main.yml`:

- from `Outputs`
    - the `StagingLBEndpoint` output
    - the `ProdLBEndpoint` output

We can now deploy the updates.

```
./deploy-infra.sh


=========== Deploying setup.yml ===========
Waiting for changeset to be created..

No changes to deploy. Stack awsbootstrap-setup is up to date



=========== Packaging main.yml ===========



=========== Deploying main.yml ===========

Waiting for changeset to be created..
Waiting for stack create/update to complete
Successfully created/updated stack - awsbootstrap
[
    "https://prod.the-good-parts.com",
    "https://staging.the-good-parts.com"
]
```

terminal

```
for run in {1..20}; do curl -s https://staging.the-good-parts.com; done | sort | uniq -c
9 Hello HTTPS World from ip-10-0-35-100.ec2.internal in awsbootstrap-Staging-10LP6MF0TQC9Y
11 Hello HTTPS World from ip-10-0-64-92.ec2.internal in awsbootstrap-Staging-10LP6MF0TQC9Y
```

terminal

```
for run in {1..20}; do curl -s https://prod.the-good-parts.com; done | sort | uniq -c
9 Hello HTTPS World from ip-10-0-21-46.ec2.internal in awsbootstrap-Prod-1PT61TNHUQWTE
11 Hello HTTPS World from ip-10-0-68-190.ec2.internal in awsbootstrap-Prod-1PT61TNHUQWTE
```

terminal

Now, our HTTP endpoints don’t exist anymore, and hitting them will result in a time out.

```
curl -v http://prod.the-good-parts.com
* Rebuilt URL to: http://prod.the-good-parts.com/
*   Trying 35.153.128.232...
* TCP_NODELAY set
* Connection failed
* connect to 35.153.128.232 port 80 failed: Operation timed out
*   Trying 54.147.46.5...
* TCP_NODELAY set
* Connection failed
* connect to 54.147.46.5 port 80 failed: Operation timed out
* Failed to connect to prod.the-good-parts.com port 80: Operation timed out
* Closing connection 0
curl: (7) Failed to connect to prod.the-good-parts.com port 80: Operation timed out
```

terminal

Let’s commit our infrastructure changes.

```
git add main.yml stage.yml
git commit -m "Remove HTTP endpoint"
git push
```

terminal

> **Note:** All the code has been already added and we are pushing it on our repository as well.

This code requires the following API keys to execute:

username	Not Specified...
AWS_ACCESS_KE...	Not Specified...

```
{
  "name": "aws-bootstrap",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "start": "node ./node_modules/pm2/bin/pm2 start ./server.js --name hello_aws --log ../logs/app
    "stop": "node ./node_modules/pm2/bin/pm2 stop hello_aws",
    "build": "echo 'Building...'"
  },
  "dependencies": {
    "pm2": "^4.2.0"
  }
}
```

And now, all that remains is to update our application to stop listening for HTTP requests.

```
const { hostname } = require('os');
const https = require('https');
const fs = require('fs');

const STACK_NAME = process.env.STACK_NAME || "Unknown Stack";
const httpsPort = 8443;
const httpsKey = '../keys/key.pem'
const httpsCert = '../keys/cert.pem'

if (fs.existsSync(httpsKey) && fs.existsSync(httpsCert)) {
  console.log('Starting https server')
  const message = `Hello HTTPS World from ${hostname()} in ${STACK_NAME}\n`;
  const options = { key: fs.readFileSync(httpsKey), cert: fs.readFileSync(httpsCert) };
  const server = https.createServer(options, (req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end(message);
  });
  server.listen(httpsPort, hostname, () => {
    console.log(`Server running at http://${hostname()}:${httpsPort}/`);
  });
} else {
  console.log('Could not find certificate/key');
}
```

server.js

Let's push our application change to GitHub and wait for it to go through the pipeline.

```
git add server.js
git commit -m "Remove http listener from the application"
git push
```

terminal

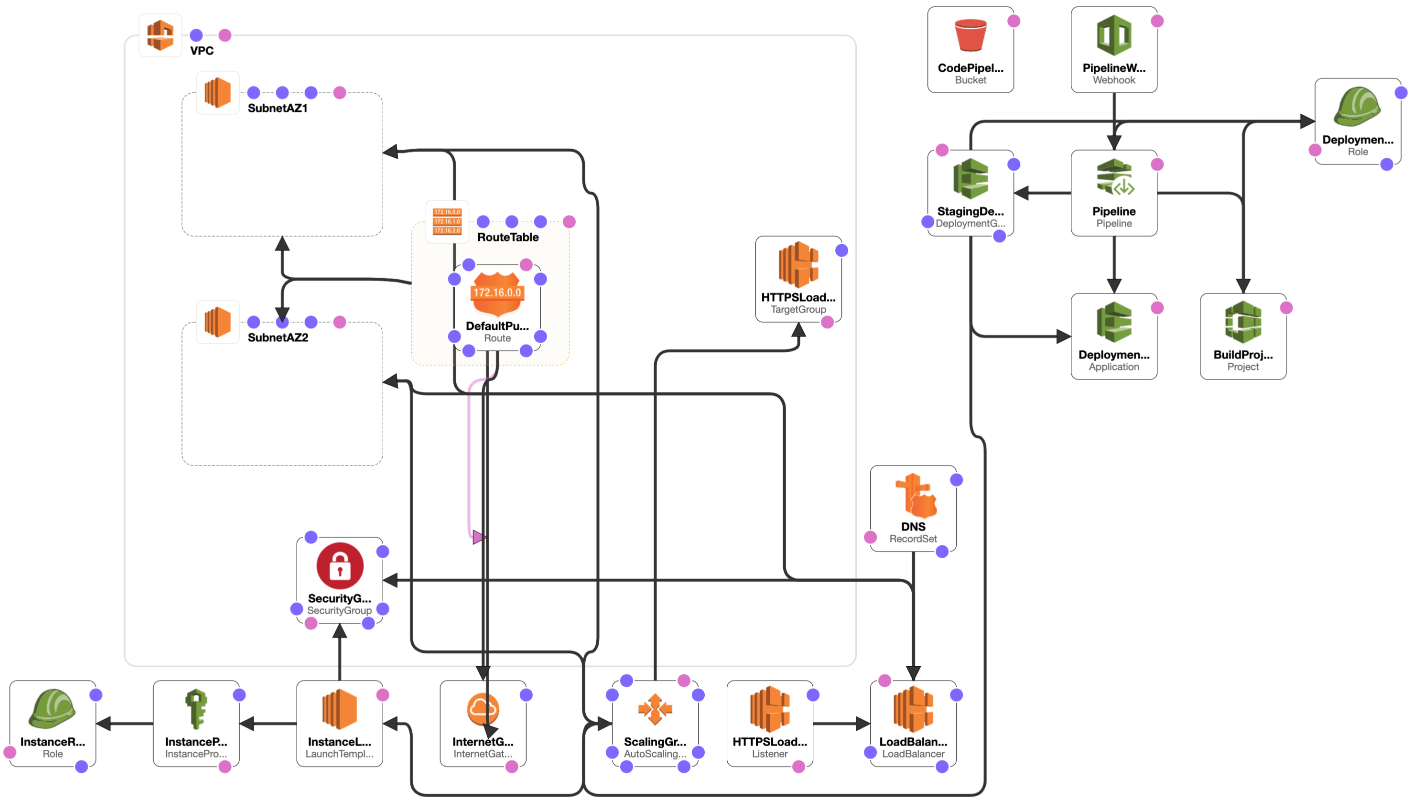
This code requires the following API keys to execute:

username	Not Specified...
AWS_ACCESS_KE...	Not Specified...
AWS_SECRET_AC...	Not Specified...
AWS_REGION	us-east-1
Github_Token	Not Specified...

```
{
  "name": "aws-bootstrap",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "start": "node ./node_modules/pm2/bin/pm2 start ./server.js --name hello_aws --log ../logs/app
    "stop": "node ./node_modules/pm2/bin/pm2 stop hello_aws",
    "build": "echo 'Building...'"
  },
  "dependencies": {
    "pm2": "^4.2.0"
  }
}
```

We've successfully migrated our application from HTTP only to HTTPS only. We did this without affecting users, by being thoughtful about the phases of our migration.

In order to get a pictorial view of our developed cloudformation stack so far, below is the design view which shows the resources we created and their relationships.



HTTPS

In the next lesson, we will improve the network security of our stack, and make our instances inaccessible from the internet.