Project 1 – EC2 REST Service: Pounds to Kilograms

Braden Gant
CS454 – Cloud Computing

Due: September 24, 2025

Contents

1 Overview

This project demonstrates the provisioning and configuration of an AWS EC2 instance to host a minimal REST web service. The service converts pounds (lbs) into kilograms (kg) and was designed with emphasis on security, reliability, and cost hygiene.

The implementation uses Node.js with the Express framework to handle HTTP requests, and Morgan middleware to provide request logging. Reliability is achieved through `systemd`, ensuring the service automatically restarts if it fails and persists across reboots. Security was enforced through restrictive AWS Security Group rules following the principle of least privilege.

The required API endpoint is:

```
GET /convert?lbs=<number>
```

Expected JSON response:

```
{
  "lbs": 150,
  "kg": 68.039,
  "formula": "kg = lbs * 0.45359237"
}
```

Error handling was built into the service:

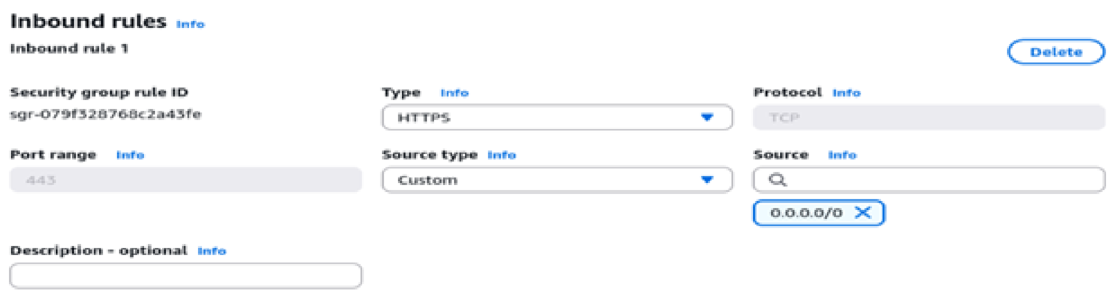
- **400 Bad Request** – if the parameter is missing, non-numeric, or NaN.
- **422 Unprocessable Entity** – if the parameter is negative or non-finite.

2 EC2 Provisioning

Provisioning began by launching an **Amazon Linux 2023 (t2.micro)** instance, which fits within the AWS free tier and provides a stable environment for Node.js. A new key pair was created for secure SSH access.

A Security Group was configured with strict rules:

- SSH (22) was restricted to only my personal IP address.
- Port 8080 was opened only to my IP to allow API testing.
- Port 443 (HTTPS) was initially present but removed since TLS was not required.



Security group rule ID: sgr-079f328768c2a43fe	Type: HTTPS	Protocol: TCP
Port range: 443	Source type: Custom	Source: 0.0.0.0/0
Description - optional:		

Figure 1: Inbound rule configuration showing least-privilege setup.

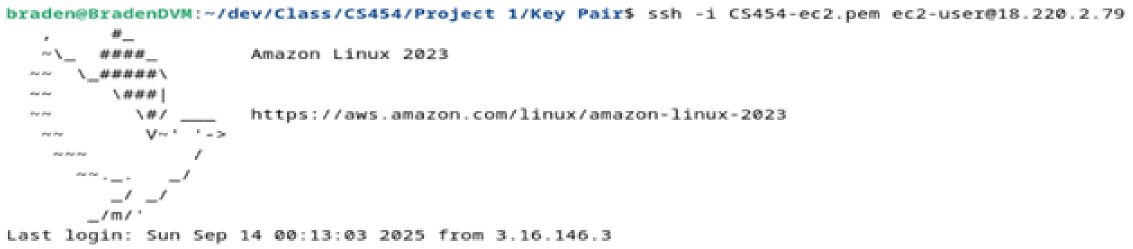
These decisions ensured that the instance was reachable for administration but not exposed broadly to the internet, aligning with cloud security best practices.

3 Setup Instructions

After launching the instance, the following setup steps were performed.

3.1 Connect to EC2

```
chmod 400 MyKeyPair.pem
ssh -i MyKeyPair.pem ec2-user@<PUBLIC_IP>
```



```
braden@BradenDVM:~/dev/Class/CS454/Project 1/Key Pair$ ssh -i CS454-ec2.pem ec2-user@18.220.2.79
   ,     #_
   ~\_  ####_        Amazon Linux 2023
  ~~  \_#####\
  ~~     \###|
  ~~       \#/ ___   https://aws.amazon.com/linux/amazon-linux-2023
   ~~       V~' '->
    ~~~         /
      ~~._.   _/
         _/ _/
       _/m/'
Last login: Sun Sep 14 00:13:03 2025 from 3.16.146.3
```

Figure 2: Successful SSH login to EC2 instance using generated key pair.

3.2 Install Runtime

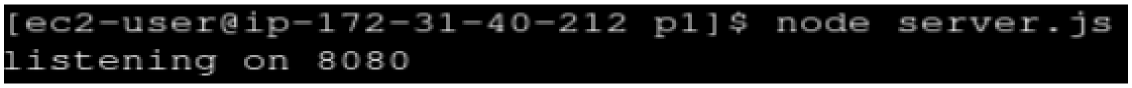
Node.js and Git were installed:

```
sudo yum update -y
sudo yum install -y nodejs npm git
```

3.3 Deploy Service

The project repository was cloned and dependencies installed:

```
git clone https://github.com/Braden-7455/project1-ec2-rest-converter.git
    p1
cd p1
npm install
node server.js   # manual run
```



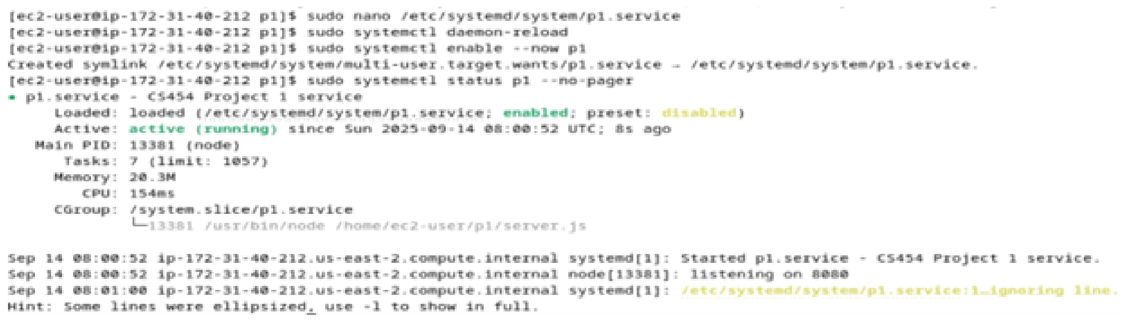
```
[ec2-user@ip-172-31-40-212 p1]$ node server.js
listening on 8080
```

Figure 3: Server running manually and listening on port 8080.

3.4 Run as a Service (systemd)

To ensure reliability, a `systemd` service was created:

```
sudo bash -c 'cat␣>/etc/systemd/system/p1.service␣<<"UNIT"
[Unit]
Description=CS454␣Project␣1␣service
After=network.target

[Service]
User=ec2-user
WorkingDirectory=/home/ec2-user/p1
ExecStart=/usr/bin/node␣/home/ec2-user/p1/server.js
Restart=always
Environment=PORT=8080

[Install]
WantedBy=multi-user.target
UNIT'

sudo systemctl daemon-reload
sudo systemctl enable --now p1
sudo systemctl status p1 --no-pager
```



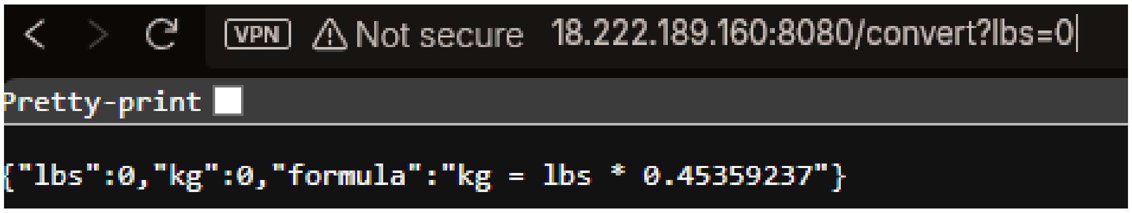
```
[ec2-user@ip-172-31-40-212 p1]$ sudo nano /etc/systemd/system/p1.service
[ec2-user@ip-172-31-40-212 p1]$ sudo systemctl daemon-reload
[ec2-user@ip-172-31-40-212 p1]$ sudo systemctl enable --now p1
Created symlink /etc/systemd/system/multi-user.target.wants/p1.service → /etc/systemd/system/p1.service.
[ec2-user@ip-172-31-40-212 p1]$ sudo systemctl status p1 --no-pager
● p1.service - CS454 Project 1 service
     Loaded: loaded (/etc/systemd/system/p1.service; enabled; preset: disabled)
     Active: active (running) since Sun 2025-09-14 08:00:52 UTC; 8s ago
   Main PID: 13381 (node)
      Tasks: 7 (limit: 1057)
     Memory: 20.3M
        CPU: 154ms
     CGroup: /system.slice/p1.service
             └─13381 /usr/bin/node /home/ec2-user/p1/server.js

Sep 14 08:00:52 ip-172-31-40-212.us-east-2.compute.internal systemd[1]: Started p1.service - CS454 Project 1 service.
Sep 14 08:00:52 ip-172-31-40-212.us-east-2.compute.internal node[13381]: listening on 8080
Sep 14 08:01:00 ip-172-31-40-212.us-east-2.compute.internal systemd[1]: /etc/systemd/system/p1.service:1…ignoring line.
Hint: Some lines were ellipsized, use -l to show in full.
```

Figure 4: Systemctl status confirming service active and managed by systemd.

4 Testing

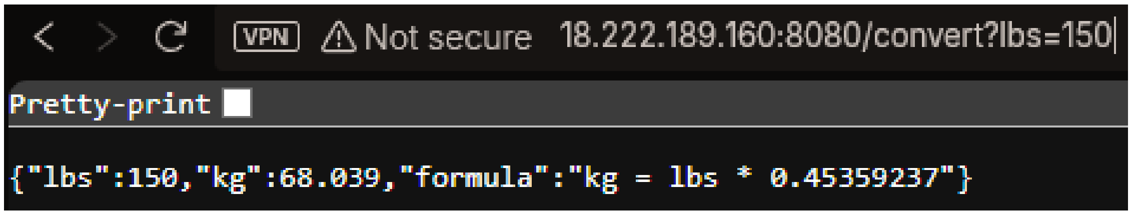
Six test cases were executed using both `curl` and browser for testing.

Case 1: Zero Input



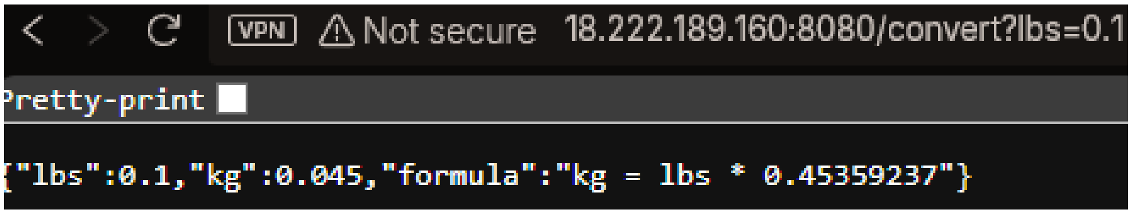
```
18.222.189.160:8080/convert?lbs=0
{"lbs":0,"kg":0,"formula":"kg = lbs * 0.45359237"}
```

Case 2: Normal Input (150 lbs)



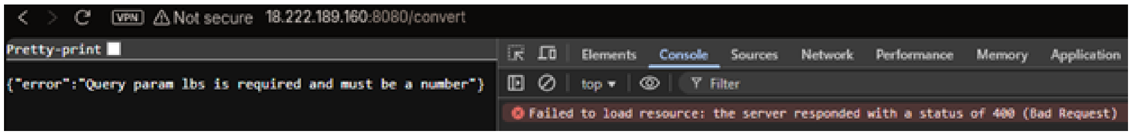
```
18.222.189.160:8080/convert?lbs=150
{"lbs":150,"kg":68.039,"formula":"kg = lbs * 0.45359237"}
```

Case 3: Decimal Input (0.1 lbs)



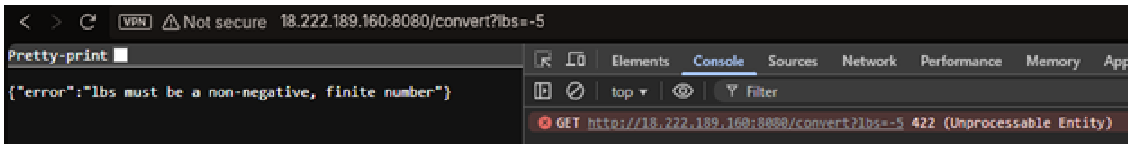
```
18.222.189.160:8080/convert?lbs=0.1
{"lbs":0.1,"kg":0.045,"formula":"kg = lbs * 0.45359237"}
```

Case 4: Missing Parameter



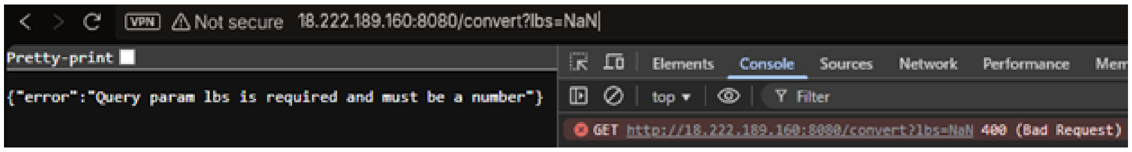
```
18.222.189.160:8080/convert
{"error":"Query param lbs is required and must be a number"}
Failed to load resource: the server responded with a status of 400 (Bad Request)
```

Case 5: Negative Input (-5 lbs)

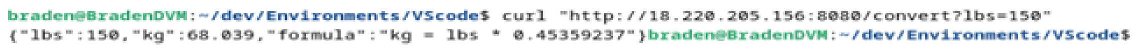


```
18.222.189.160:8080/convert?lbs=-5
{"error":"lbs must be a non-negative, finite number"}
GET http://18.222.189.160:8080/convert?lbs=-5 422 (Unprocessable Entity)
```

Case 6: Not-a-Number (NaN)



```
18.222.189.160:8080/convert?lbs=NaN
{"error":"Query param lbs is required and must be a number"}
GET http://18.222.189.160:8080/convert?lbs=NaN 400 (Bad Request)
```



```
braden@BradenDVM:~/dev/Environments/VScode$ curl "http://18.220.205.156:8080/convert?lbs=150"
{"lbs":150,"kg":68.039,"formula":"kg = lbs * 0.45359237"}braden@BradenDVM:~/dev/Environments/VScode$
```

Figure 5: Example curl request validating API output from local machine.

5 Security Group Summary

The final inbound rules were:

- SSH (22): restricted to my IP only.
- API (8080): restricted to my IP only.
- HTTPS (443): removed entirely.

This configuration strictly enforced least-privilege access and reduced the instance's attack surface.

6 Cleanup and Cost Hygiene

AWS best practices were followed to avoid unnecessary charges:

- Instances were terminated after use.
- Orphaned EBS volumes (left behind after termination) were deleted.
- Old and unused key pairs were removed to reduce clutter and security risk.
- Security Groups created for the project were deleted after submission.

7 Conclusion

This project was implemented with the following:

- Correct API implementation with proper error handling.
- Reliable operation through `systemd`.
- Secure configuration using least-privilege Security Groups.
- Comprehensive testing with screenshots as evidence.
- Cloud cost hygiene through responsible cleanup.

The result is a secure, reliable, and minimal REST API running on AWS EC2 that demonstrates practical skills in cloud provisioning, service management, and DevOps practices.