Project 1 – EC2 REST Service: Pounds → Kilograms

CS 454/554 – Cloud Computing (UAH)

In this project you will provision an AWS EC2 instance and deploy a small REST web service that converts pounds (lbs) to kilograms (kg). You may implement the service in Node.js/Express (recommended) or any comparable stack (Python/Flask, Go, Java/Spring, etc.).

Learning Objectives

1. Provision and secure a VM in the public cloud (AWS EC2).
2. Expose a minimal REST API over HTTP/HTTPS.
3. Use system services and/or a reverse proxy to run a web service reliably.
4. Apply basic DevOps practices: logging, README, reproducible setup instructions.
5. Practice cloud cost hygiene and cleanup.

Deliverables (What you submit)

1. Git repository (GitHub/GitLab) with source code, README, and a short DESIGN.md (1–2 pages).
2. README with setup steps (install, run, test), and curl examples.
3. Screenshot(s) of a successful request using curl/Postman and your service logs.
4. Public endpoint & IP/DNS (include Security Group rule summary) – you may restrict by my IP if desired.
5. Optional: a short video (≤2 min) demo (unlisted link).

Required API Specification

Implement an HTTP GET endpoint that converts pounds to kilograms:

```
GET /convert?lbs=<number>
Response: 200 OK, application/json
{
  "lbs": <number>,
  "kg":  <number rounded to 3 decimals>,
  "formula": "kg = lbs * 0.45359237"
}

Errors:
- 400 Bad Request if query param missing or not a number.
- 422 Unprocessable Entity if value is negative or non-finite (NaN/Inf).
```

Grading Rubric (100 points)

Category	Points	Details
EC2 Provisioning & Security	20	AMI choice, key pair, Security Group (SSH limited; HTTP/HTTPS as need
REST API Correctness	25	Endpoint behavior, JSON shape, numeric handling, error codes.
Reliability (Service mgmt)	15	Runs as a service (systemd/pm2) and survives reboot; logs visible.
Docs & Repo Hygiene	20	README clarity, curl examples, DESIGN.md rationale, .gitignore, structur
Testing Evidence	10	curl/Postman screenshots; sample inputs incl. edge cases.
Cleanup & Cost	10	Terminate/stop resources; explain cleanup steps in README.

Part A — EC2 Provisioning (Console or CLI)

Console (summary)

1. Launch an Amazon Linux 2 (or Ubuntu LTS) t2.micro in your default VPC.
2. Create or choose a key pair for SSH.
3. Security Group: allow SSH (22) from your IP; allow HTTP (80) for testing. HTTPS (443) optional.
4. Launch, wait to reach *running*; copy the Public IPv4 address.

CLI (example)

```
# Replace IDs with ones valid in your region
AMI_ID=ami-xxxxxxxxxxxxxxxxx   # Amazon Linux 2 or Ubuntu
SUBNET_ID=subnet-xxxxxxxxxxxxx  # any public subnet
SG_ID=sg-xxxxxxxxxxxxxxxxx      # must allow 22 and 80
KEY_NAME=MyKeyPair

aws ec2 run-instances \
  --image-id $AMI_ID \
  --count 1 --instance-type t2.micro \
  --key-name $KEY_NAME \
  --security-group-ids $SG_ID \
  --subnet-id $SUBNET_ID \
  --tag-specifications 'ResourceType=instance,Tags=[{Key=Name,Value=cs454-p1}]'

# Get the public IP
aws ec2 describe-instances --filters 'Name=tag:Name,Values=cs454-p1' \
  --query 'Reservations[0].Instances[0].PublicIpAddress' --output text
```

Part B — Connect & Install Runtime

```
# Linux/Mac
chmod 400 MyKeyPair.pem
ssh -i MyKeyPair.pem ec2-user@<PUBLIC_IP>        # Amazon Linux
# or: ssh -i MyKeyPair.pem ubuntu@<PUBLIC_IP>    # Ubuntu

# Update & install Node.js (choose one approach)
sudo yum update -y && sudo yum install -y nodejs npm   # Amazon Linux extras may be needed
# Ubuntu example:
sudo apt update && sudo apt install -y nodejs npm
```

Part C — Implement the Service (Node/Express example)

```
mkdir -p ~/p1 && cd ~/p1
npm init -y
npm install express morgan

cat > server.js <<'EOF'
const express = require('express');
const morgan = require('morgan');
const app = express();
app.use(morgan('combined'));
app.get('/convert', (req, res) => {
  const lbs = Number(req.query.lbs);
  if (req.query.lbs === undefined || Number.isNaN(lbs)) {
    return res.status(400).json({ error: 'Query param lbs is required and must be a number' });
  }
  if (!Number.isFinite(lbs) || lbs < 0) {
    return res.status(422).json({ error: 'lbs must be a non-negative, finite number' });
  }
  const kg = Math.round(lbs * 0.45359237 * 1000) / 1000;
  return res.json({ lbs, kg, formula: 'kg = lbs * 0.45359237' });
});
const port = process.env.PORT || 8080;
app.listen(port, () => console.log(`listening on ${port}`));
EOF

node server.js
```

Open your browser to *http://<PUBLIC_IP>:8080/convert?lbs=150* or test with curl:

```
curl 'http://<PUBLIC_IP>:8080/convert?lbs=150'
```

Part D — Run as a Service (systemd) & Optional Reverse Proxy

```
# Create a systemd unit
sudo bash -c 'cat >/etc/systemd/system/p1.service <<"UNIT"\n[Unit]\nDescription=CS454 Project 1 servic
sudo systemctl daemon-reload
sudo systemctl enable --now p1
sudo systemctl status p1 --no-pager

# (Optional) NGINX reverse proxy on :80 → :8080
sudo yum install -y nginx  # or: sudo apt install -y nginx
sudo sed -i 's|try_files.*|proxy_pass http://127.0.0.1:8080;|' /etc/nginx/conf.d/default.conf || true
sudo systemctl enable --now nginx
```

If you enable NGINX, adjust your Security Group to allow inbound TCP/80. You can later add TLS with Certbot/Let’s Encrypt (extra credit).

Part E — Test Cases (use in README)

1 Happy path: /convert?lbs=0 → 0.000 kg
2 Typical: /convert?lbs=150 → 68.039 kg
3 Edge: /convert?lbs=0.1 → 0.045 kg
4 Error: /convert (missing param) → 400
5 Error: /convert?lbs=-5 → 422
6 Error: /convert?lbs=NaN → 400

Alternative Stacks (if not using Node)

1 Python/Flask: return jsonify as above; run with gunicorn, systemd; optional NGINX.
2 Go: net/http; build a single binary; run as systemd service.
3 Java/Spring Boot: simple controller; package with Maven/Gradle; run as systemd.

Security & Cost Hygiene

1 Limit SSH to your IP in the Security Group; consider changing the key after the project.
2 Don’t run as root; keep service under a non-privileged user.
3 Rotate logs or cap size (e.g., logrotate or pm2 if used).
4 Stop or terminate the instance when finished; delete orphaned EBS volumes and Key Pairs.

Submission Checklist

1. Repo URL with code + README + DESIGN.md.
2. Public URL/IP (or instructions if restricted), and SG rule summary.
3. Screenshots: curl success + error cases; systemctl status; security group.
4. Cleanup note: what you terminated/stopped.