SnapSoft AWS Cloud Homework

Our client, the cybersecurity company, The DNS Detectives, is on the trail of a new virus that is changing DNS settings to redirect users to malicious websites. They need a way for their cyber agents to quickly and easily report suspicious sites, so their internal team could investigate them and take action.
They asked us to provide them a small service which can receive and store such reports and resolve the ip address right away.
One of their engineers already started working on this, but due to other commitments, they had to abandon the project. They managed to finish the business-logic lambda function, and started a few snippets to create the infrastructure. See attached snippets below, and the lambda source and zip in *Appendix 1*.

Your task is to deliver this project.

Create the following infrastructure in terraform (terragrunt is allowed):

- All resource must use the eu-central-1 region
- A db.t3.micro database in RDS with postgres engine.
  The terraform fragment creating the database (change as needed):

```
resource "aws_db_instance" "sample_db" {
  allocated_storage     = 10
  db_name               = "mydb"
  engine                = "postgres"
  engine_version        = "15"
  instance_class        = "db.t3.micro"
  username              = "master"
  password              = "securepass"
  parameter_group_name  = "default.postgres15"
  skip_final_snapshot   = true
  publicly_accessible   = true
}
```

- A lambda function that accepts the following JSON object, resolves the target's ip and stores the *message*, the call's *timestamp*, the *target*, and the *ip* in the database.

```
{
   "message": "secret message",
   "target": "google.com"
}
```

  The lambda function takes care of setting up the database.

  See the lambda source code, and package URL in appendix 1.

The terraform fragment creating the lambda function (change as needed):

```
resource "aws_lambda_function" "lambda_function" {
  function_name     = "mylambda"
  handler           = "lambda.handler"
  s3_key            = "lambda.zip"
  s3_bucket         = "snapsoft-cloud-hw-lambda-bucket"
  memory_size       = 256
  runtime           = "nodejs16.x"
  timeout           = 5
  package_type      = "Zip"
  environment {
    variables = {
      DB_HOST = "localhost"
      DB_PASSWORD = "securepass"
      DB_USERNAME = "master"
    }
  }
}
```

- An API Gateway, which must be the only public interface for this service. Its only route should be directed to the lambda function.

Make sure the solution is as **secure as possible**, and follows AWS best practices. The client may have other workloads on the same account, which may be vulnerable to malicious attacks.
Include a readme file with all the necessary steps to deploy it to the client's super secret infrastructure, and how can they test and use it. The less manual work needed, the better.

The task should be done in around 4 hours.

Please provide your solution as a zip file either through email or a google drive file upload.
Please do not upload your code to public repositories.

Appendix 1 - Lambda source code

The packaged sources can be accessed on S3 with any valid AWS credentials:

```
s3://snapsoft-cloud-hw-lambda-bucket/lambda.zip
```

Lambda source

```
const { Client } = require("pg");
const dns = require("dns")

const DB_USERNAME = process.env.DB_USERNAME || ""
const DB_PASS = process.env.DB_PASSWORD || ""
const DB_HOST = process.env.DB_HOST || ""
const DB_PORT = process.env.DB_PORT || "5432"
const DB_DATABASE = process.env.DB_DATABASE || "postgres"

const dbInitScript = `CREATE TABLE IF NOT EXISTS requests(
  id serial primary key,
  message text,
  target text,
  ip text,
  timestamp TIMESTAMP
);`

exports.handler = async (event) => {
    try{
        validateEnv("DB_USERNAME", DB_USERNAME)
        validateEnv("DB_PASSWORD", DB_PASS)
        validateEnv("DB_HOST", DB_HOST)
        validateEnv("DB_PORT", DB_PORT)
        validateEnv("DB_DATABASE", DB_DATABASE)
    }catch (e){
        console.error(e)
        return {
            statusCode: 500,
            body: e,
        }
    }
    console.log("Lambda input: ",event);
    let message = "";
    let target = "";
    if(event.body){
        let body = JSON.parse(event.body);
        message = body.message;
        target = body.target;
    }else{
        message = event.message
        target = event.target
    }
    const client = new Client(buildDbConfig());
    try {
        console.log("Establishing database connection ...")
        await client.connect();
        console.log("Database client connected")
        await ensureDbSchema(client);
        console.log(`Resolving domain ${target} to ip address ...`)
        const ip = await getDomainIP(target)
        console.log("Domain to ip resolved: "+ip)
        await insertRow(client, message, target,ip,new Date())
    } catch(e){
        console.error("Error ",e)
        return {
            statusCode: 200,
            body: "An error has happened"+e,
        };
    } finally {
        await client.end();
    }
    return {
        statusCode: 200,
        body: "Record inserted! Thank you for your report!",
    };
};

async function insertRow(client, message, target, ip, timestamp){
    const res = await client.query(
        "insert into requests (message, target, ip, timestamp) values ($1, $2, $3, $4)",
        [message, target, ip, timestamp]
    );
    console.log("Insert result: ",res);
    return res

}
async function getDomainIP(domain) {
    try {
        const ipAddresses = await dns.promises.resolve(domain);
        return ipAddresses[0];
    } catch (error) {
        return "failed";
    }
}

async function ensureDbSchema(client){
    return client.query(dbInitScript)
}

function buildDbConfig(){
    return  {
        user: DB_USERNAME,
        password: DB_PASS,
        host: DB_HOST,
        port: parseInt(DB_PORT),
        database: DB_DATABASE,
        ssl: { rejectUnauthorized: false },
    };
}

function validateEnv(envName, envValue){
    if(!envValue){
        console.error(`No ${envName} configured!`);
        throw `No ${envName} configured!`;
    }
}
```