# HomeApp Backend Server

The HomeApp server stores the pollution measurement events, as well as future data such as accounts, authorizations, devices etc.

The backend supports remote querying of data, in particular finding the nearby level of pollution.

The backend interaction with the user device is via industry standard REST API.

For reasons of accessibility, cost and ease of development, all of the backend components are deployed on Amazon Web Services (AWS). We used service components so we don’t have to deal with server management at all.

**AWS**

**Lambda**

**Database**

# Backend components

**Amazon API Gateway**

## Database server

The database is Mysql 5.5.54, running as RDS (Relational Database Service). This means that AWS are running the server and operating system, while we deal only with data definitions and data management.

The database server runs in Dublin, Ireland, on a t2.micro instance.

## AWS Lambda function

The code to deal with the REST requests and responses is written in Python 2.7.

It runs as a Lambda serverless event driven function, so the code is loaded and executed only when required by an API event.

## AWS API Gateway

# Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. With a few clicks in the AWS Management Console, we can create an API that acts as a “front door” for applications to access data, business logic, or functionality from our back-end services, such as workloads running on Amazon Elastic Compute Cloud (Amazon EC2), code running on AWS Lambda, or any Web application. Amazon API Gateway handles all the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls, including traffic management, authorization and access control, monitoring, and API version management.

# Future developments

* Reconsideration of the database engine to allow for scalability
* User and device registration
* Authentication and authorization of users and devices
* API authentication

# Code

## SQL data model

CREATE TABLE IF NOT EXISTS substances (

id int(10) NOT NULL AUTO\_INCREMENT,

code varchar(10) NOT NULL,

description varchar(256) NOT NULL ,

threshold dec(20,5) NOT NULL ,

PRIMARY KEY (id)

) ;

CREATE TABLE IF NOT EXISTS devices (

id int(10) NOT NULL AUTO\_INCREMENT,

deviceid varchar(256) NOT NULL ,

userid int(10) NOT NULL,

appversion varchar(32) NOT NULL,

arduinoversion varchar(32) NOT NULL,

make varchar(32) NOT NULL,

model varchar(32) NOT NULL,

os varchar(32) NOT NULL,

osversion varchar(32) NOT NULL,

registrationdate timestamp,

PRIMARY KEY (id)

) ;

CREATE TABLE IF NOT EXISTS events (

id int(10) NOT NULL AUTO\_INCREMENT,

eventdate timestamp ,

deviceid varchar(256) NOT NULL ,

userid int(10) NOT NULL,

longitude dec(11,9) NOT NULL,

latitude dec(11,9) NOT NULL,

substanceid int(10) not null,

measurement dec(18,9) not null,

PRIMARY KEY (id)

) ;

## Python code

import json

import os

import datetime

import random

import urlparse

# external dependencies

import pymysql

# private dependencies

import db\_config

"""

https://www.scribd.com/presentation/2569355/Geo-Distance-Search-with-MySQL

https://www.scribd.com/document/24744656/Distance-Calculation-Between-2-Points-on-Earth

Simplified approximate query - assume near euclidan geometry.

111.3278 = 1 degree in km

57.29 = 180/PI conversion degrees to radians

SELECT longitude, latitude , sqrt(

POW( 111.3278 \* ( latitude - @@mylat ) , 2 ) + POW( 111.3278 \* ( 34.6 - @@mylon ) \* COS( @@mylat / 57.29 ) , 2 )

) AS distance\_sq

FROM events

HAVING distance\_sq <10

ORDER BY distance\_sq

"""

# Main function

def lambda\_handler(event, context):

# write event to database, and then fetch the current results

assert event["context"]["http-method"] in ["POST","GET"] ,

"Method %r is not supported" % event["context"]["http-method"]

# return event

print event["context"]["http-method"]

connection=connect\_database()

if event["context"]["http-method"]=="POST"

and event["params"]["header"]["Content-type"] == "application/json":

response = write\_event(event, connection)

longitude=event["body-json"]["longitude"]

latitude=event["body-json"]["latitude"]

elif event["context"]["http-method"]=="POST"

and event["params"]["header"]["Content-type"] == "application/x-www-form-urlencoded":

body=dict(urlparse.parse\_qsl(event["body-json"]))

return body

longitude=event["body-json"]["longitude"]

latitude=event["body-json"]["latitude"]

elif event["context"]["http-method"]== "GET":

longitude=event["params"]["querystring"]["longitude"]

latitude=event["params"]["querystring"]["latitude"]

else:

return {}

response = read\_stats(connection, longitude, latitude)

return response

def connect\_database():

server\_address = os.getenv('DATABASE', db\_config.db\_host)

# Connect to database

connection = pymysql.connect( server\_address,

user=db\_config.db\_username,

passwd=db\_config.db\_password,

db=db\_config.db\_name,

connect\_timeout=25,

autocommit=True)

return connection

#-------------------------------------------------------------------------------------------

def write\_event( event, connection):

deviceid=event["body-json"]["device"]["deviceid"]

eventdate=event["body-json"]["timestamp"]

userid=event["body-json"]["userid"]

longitude=event["body-json"]["longitude"]

latitude=event["body-json"]["latitude"]

measurements=event["body-json"]["measurements"]

for point in measurements:

substanceid=point["substance"]

measurement=point["value"]

with connection.cursor() as cur:

statement= """INSERT INTO events(eventdate, deviceid, userid,

longitude, latitude, substanceid, measurement) VALUES (%s,%s,%s,%s,%s,%s,%s)"""

result = cur.execute(statement,

(eventdate, deviceid, userid, longitude, latitude, substanceid, measurement )

)

return result

#-------------------------------------------------------------------------------------------

def read\_stats( connection, longitude, latitude):

statement = """

select longitude, latitude, distance\_sq, substances.code , measurement

from

(SELECT longitude as longitude, latitude ,

sqrt( POW( 111.3278 \* ( latitude - %r ) , 2 ) +

POW( 111.3278 \* ( 34.6 - %r ) \* COS( latitude / 57.29 ) , 2 )

) AS distance\_sq,

substanceid , avg(measurement) as measurement

FROM events

group by longitude, latitude, distance\_sq, events.substanceid

HAVING distance\_sq <%r ) as events

JOIN substances ON events.substanceid = substances.id

WHERE events.measurement > substances.threshold

ORDER BY distance\_sq;

""" % (latitude,longitude, 10)

with connection.cursor(pymysql.cursors.DictCursor) as cur:

result = cur.execute( statement )

result = cur.fetchall()

result\_wip={}

for item in result:

longitude=float(item["longitude"])

latitude=float(item["latitude"])

tuple=(longitude, latitude)

print tuple

if not bool(result\_wip.get(tuple)):

result\_wip.update( { tuple : {

"longitude": longitude,

"latitude": latitude,

"data":[]

}

}

)

result\_wip[tuple]["data"].append(

{"code":item["code"], "measurement": float(item["measurement"])}

)

final\_list = [ value for key, value in result\_wip.iteritems()]

return final\_list