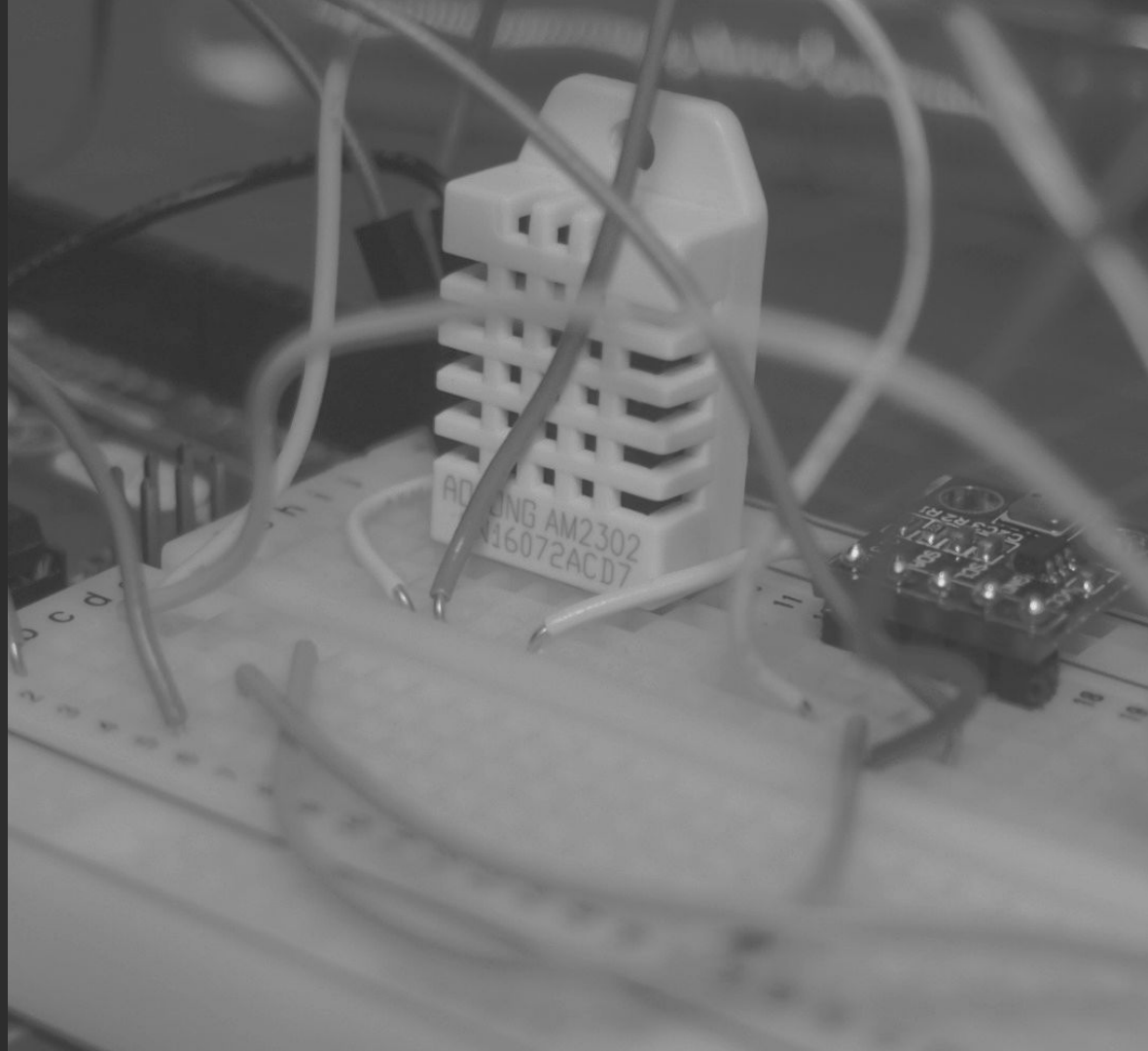
Local Weather Visualisation with IoT, Kafka and CrateDB

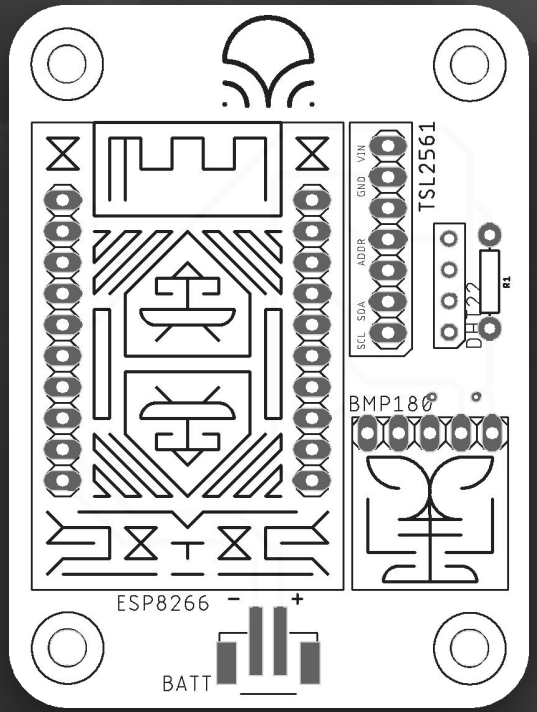


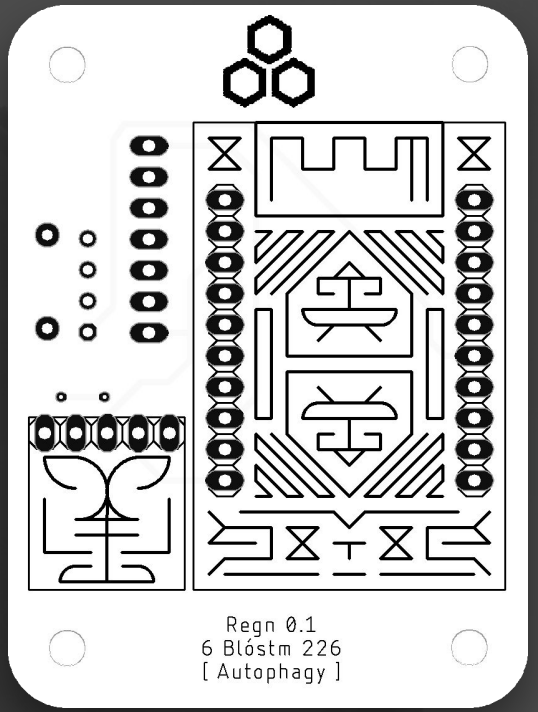
Mika Naylor
mika@crate.io
@autophagyDev

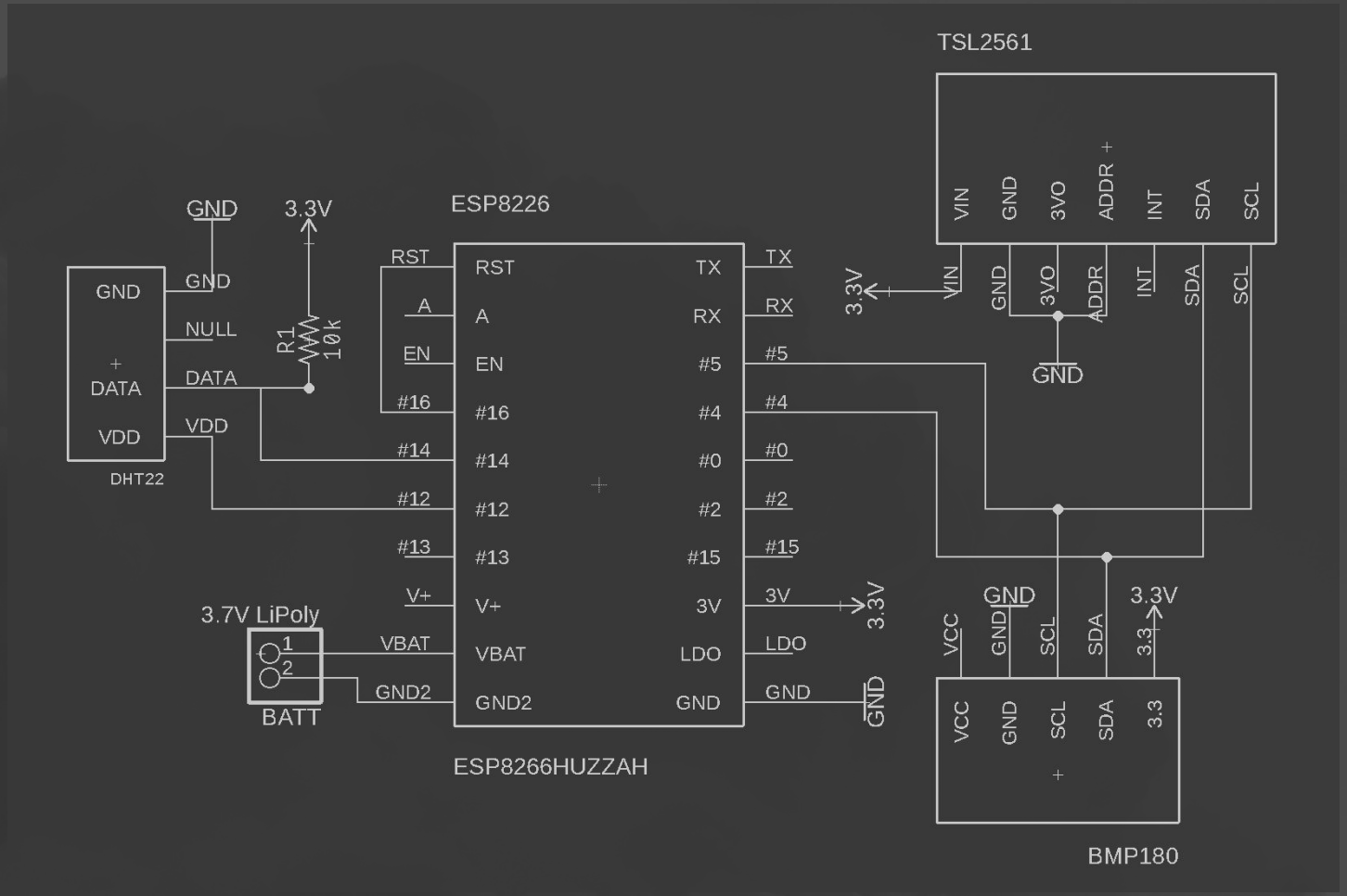
INTRODUCTION

WHY BUILD A WEATHER STATION?

Temperature/Humidity	DHT22
Barometric Pressure	BMP180
Luminosity	TSL2561
Microcontroller	ESP8266


TSL2561
SCL SDA ADDR GND VIN
DHT22
R1
BMP180
ESP8266
BATT


Regn 0.1
6 Blóstm 226
[Autophagy]


TSL2561
VIN
GND
3VO
ADDR
INT
SDA
SCL
3.3V
GND
ESP8226
RST
A
EN
#16
#14
#12
#13
V+
VBAT
GND2
TX
RX
#5
#4
#0
#2
#15
3V
LDO
GND
3.3V
GND
GND
NULL
DATA
VDD
DHT22
R1
10k
3.7V LiPoly
1
2
BATT
ESP8266HUZZAH
VCC
GND
SCL
SDA
3.3
BMP180

ACT 1

A SIMPLE PROTOTYPE

```
// Libraries
#include <DHT.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_TSL2561_U.h>
#include <Adafruit_BMP085_U.h>
#include <ESP8266WiFi.h>
#include <ArduinoJson.h>


// Constants
#define DHTPIN 12 // DHT22 connected to pin 12
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
Adafruit_TSL2561_Unified tsl = Adafruit_TSL2561_Unified(TSL2561_ADDR_LOW, 12345);
Adafruit_BMP085_Unified bmp = Adafruit_BMP085_Unified(10085);



Setup
```

```
dht.begin();
tsl.enableAutoRange(true);
tsl.setIntegrationTime(TSL2561_INTEGRATIONTIME_13MS);
bmp.begin();

WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
}
```

Setup

```
temperature = dht.readTemperature();

humidity = dht.readHumidity();

sensors_event_t bmp_event;
bmp.getEvent(&bmp_event);
pressure = bmp_event.pressure;

sensors_event_t tsl_event;
tsl.getEvent(&tsl_event);
lux = tsl_event.light;
```

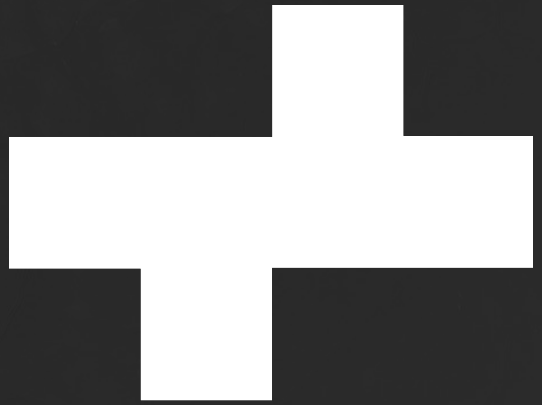
Loop

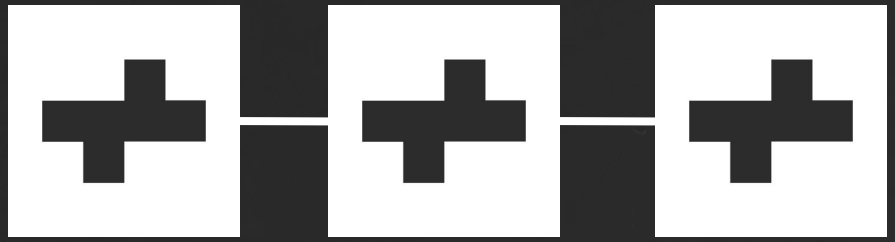
```
StaticJsonBuffer<256> JSONbuffer;
JsonObject& JSONencoder = JSONbuffer.createObject();
JSONencoder["temperature"] = temperature;
JSONencoder["humidity"] = humidity;
JSONencoder["pressure"] = pressure;
JSONencoder["luminosity"] = lux;
char JSONmessageBuffer[256];
JSONencoder.prettyPrintTo(JSONmessageBuffer, sizeof(JSONmessageBuffer));
```

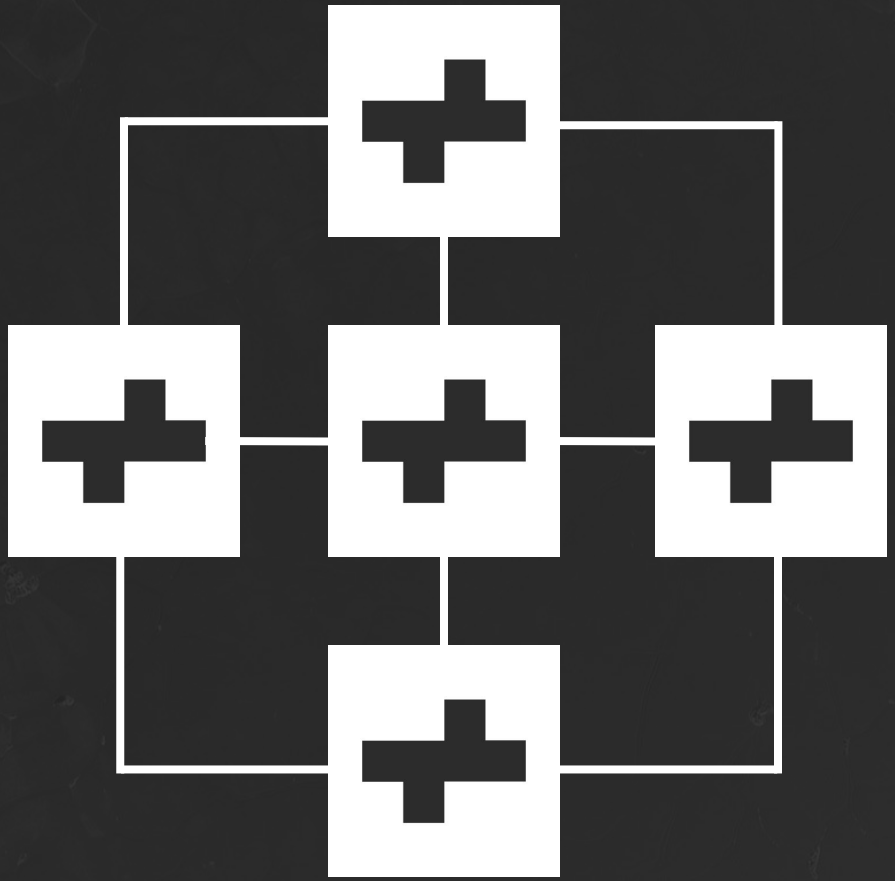
Loop

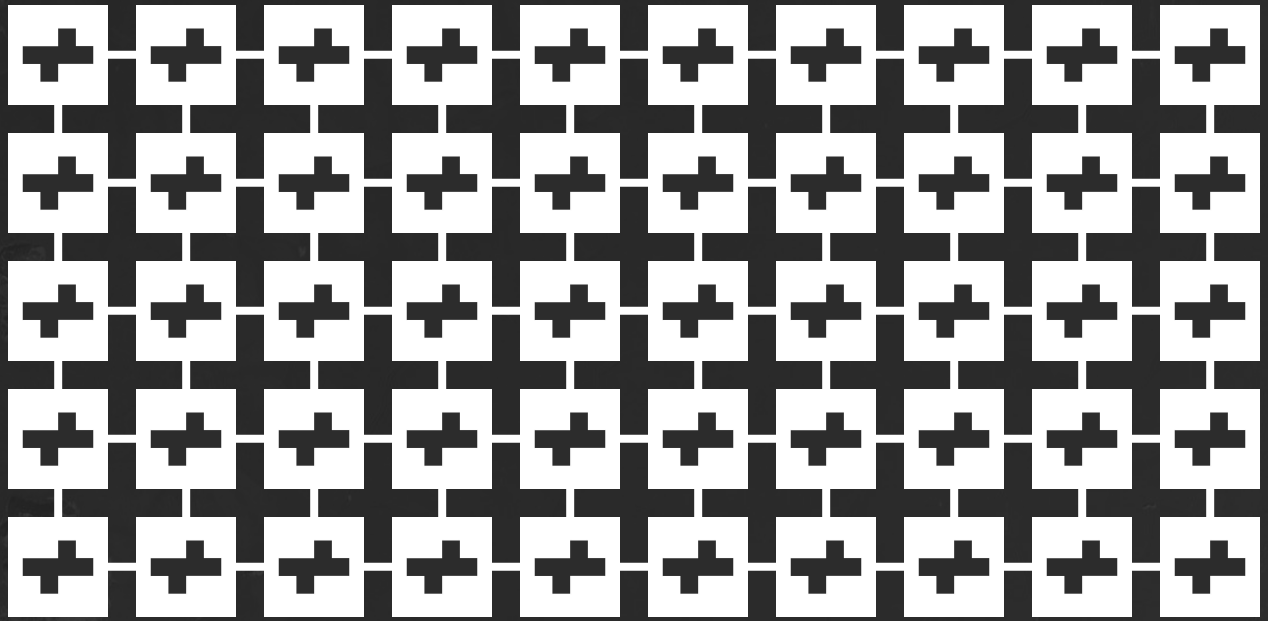
```
int code = -1;
HTTPClient http;
while(code != 201) {
  http.begin(host, fingerprint);
  http.addHeader("Content-Type", "application/json");
  code = http.POST(JSONmessageBuffer);
  http.end();
  delay(1000);
}
delay(sensor_interval);
```

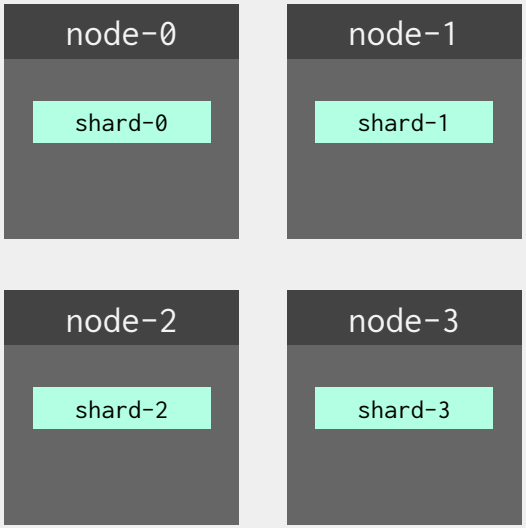
Loop



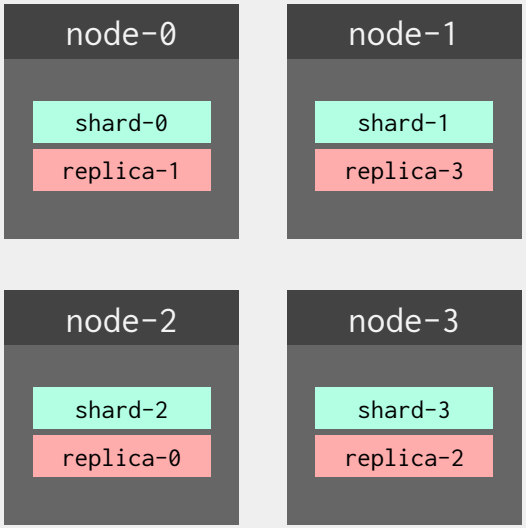




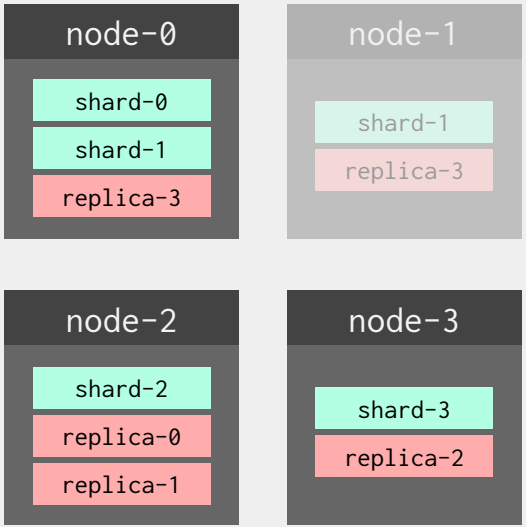





```
CREATE TABLE IF NOT EXISTS
"doc"."sensordata" (
   "timestamp" TIMESTAMP,
   "value" FLOAT
)
CLUSTERED INTO 4 SHARDS
```




```
CREATE TABLE IF NOT EXISTS
"doc"."sensordata" (
   "timestamp" TIMESTAMP,
   "value" FLOAT
)
CLUSTERED INTO 4 SHARDS
WITH (number_of_replicas = 1)
```




```
CREATE TABLE IF NOT EXISTS
"doc"."sensordata" (
   "timestamp" TIMESTAMP,
   "value" FLOAT
)
CLUSTERED INTO 4 SHARDS
WITH (number_of_replicas = 1)
```

value	timestamp
12.4	1532080800000
68.2	1532080850000
107	1532080900000
12.2	1526810400000
69.0	1526810423000
110	1526810434000
11.9	1524219010000
69.1	1524219140000
109	1524219230000

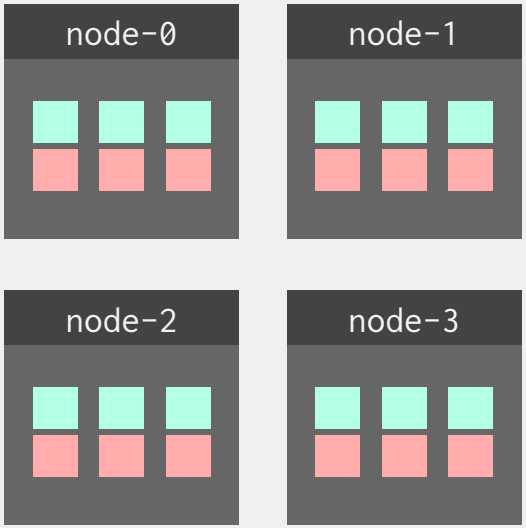
```
CREATE TABLE IF NOT EXISTS
"doc"."sensordata" (
   "timestamp" TIMESTAMP,
   "value" FLOAT
)
CLUSTERED INTO 4 SHARDS
WITH (number_of_replicas = 1)
```

value	timestamp	month
12.4	1532080800000	1532080800000
68.2	1532080850000	1532080800000
107	1532080900000	1532080800000
12.2	1526810400000	1526810400000
69.0	1526810423000	1526810400000
110	1526810434000	1526810400000
11.9	1524219010000	1524218400000
69.1	1524219140000	1524218400000
109	1524219230000	1524218400000

```
CREATE TABLE IF NOT EXISTS
"doc"."sensordata" (
   "timestamp" TIMESTAMP,
   "value" FLOAT,
   "month" TIMESTAMP GENERATED ALWAYS
   AS date_trunc('month', "timestamp")
)
CLUSTERED INTO 4 SHARDS
WITH (number_of_replicas = 1)
```

value	timestamp	month
12.4	1532080800000	1532080800000
68.2	1532080850000	1532080800000
107	1532080900000	1532080800000
12.2	1526810400000	1526810400000
69.0	1526810423000	1526810400000
110	1526810434000	1526810400000
11.9	1524219010000	1524218400000
69.1	1524219140000	1524218400000
109	1524219230000	1524218400000

```
CREATE TABLE IF NOT EXISTS
"doc"."sensordata" (
   "timestamp" TIMESTAMP,
   "value" FLOAT,
   "month" TIMESTAMP GENERATED ALWAYS
   AS date_trunc('hour', "timestamp")
)
CLUSTERED INTO 4 SHARDS
PARTITIONED BY ("month")
WITH (number_of_replicas = 1)
```




```
CREATE TABLE IF NOT EXISTS
"doc"."sensordata" (
   "timestamp" TIMESTAMP,
   "value" FLOAT,
   "month" TIMESTAMP GENERATED ALWAYS
   AS date_trunc('hour', "timestamp")
)
CLUSTERED INTO 4 SHARDS
PARTITIONED BY ("month")
WITH (number_of_replicas = 1)
```

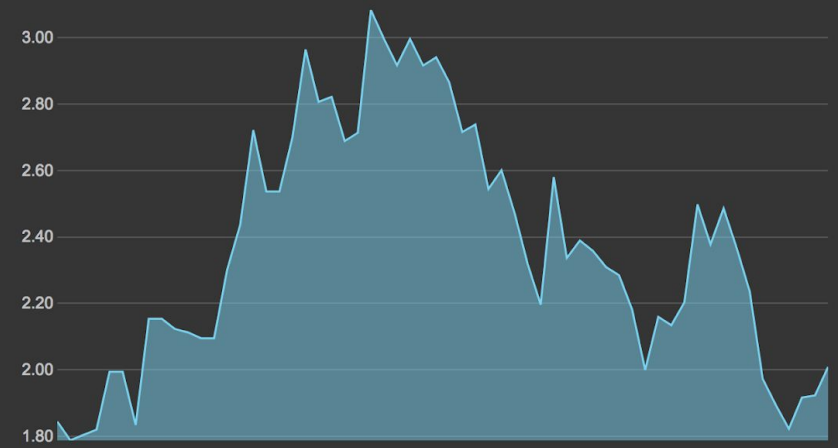
Cluster: crate

Health	Replicated Data	Available Data	Total Records	Underrepl. Records
good	100.0%	100.0%	0	0

Unavail. Records: 0

Cluster Load

Load 1 | Load 5 | Load 15



Cluster Checks

INFO
CrateDB Enterprise features are active. Please request a license before deploying in production or deactivate the Enterprise features. https://crate.io/enterprise

Flask

```python
@main.route("/api/insert", methods=["POST"])
@validate_schema(insert_sensor_schema)
def insert():
    data = json.loads(request.data)
    timestamp = datetime.utcnow()
    db_data = SensorData(
        timestamp=timestamp,
        temperature=data["temperature"],
        humidity=data["humidity"],
        pressure=data["pressure"],
        luminosity=data["luminosity"],
    )
    db.session.add(db_data)
    db.session.commit()
    return "", 201
```

```python
@main.route("/api/latest", methods=["GET"])
def latest():
    latest_results = COORDINATES
    latest = db.session.query(SensorData).order_by(desc("timestamp")).first()
    if (latest is not None):
        latest_results.update(latest.dict())
    return jsonify(latest_results)
```

```
mika@autophagy > curl localhost:8080/api/latest

{
  "latitude": 52.489,
  "longitude": 13.354,
  “temperature”: 11.2,
  “humidity”: 91,
  “pressure”: 1012,
  “lux”: 107
}
```

```
@main.route("/api/<string:sensor>/since/<int:ts>", methods=["GET"])
def sensorReadingsSinceTimestamp(sensor, ts):
    if sensor not in VALID_SENSOR_TYPES:
        abort(404)
    dt = datetime.utcfromtimestamp(ts / 1000)
    values = db.session.query(getattr(SensorData, sensor), SensorData.timestamp).filter(
        SensorData.timestamp > dt
    ).order_by(
        asc("timestamp")
    ).all()
return jsonify(values)
```

```
mika@autophagy > curl localhost:8080/api/temperature/since/1531919792197

[
  { "timestamp": 1531920179000, "value": 12.0 },
  { “timestamp": 1531920180000, "value": 12.4 },
  { "timestamp": 1531920181000, "value": 12.2 },
  ...
]
```

52.489
Latitude

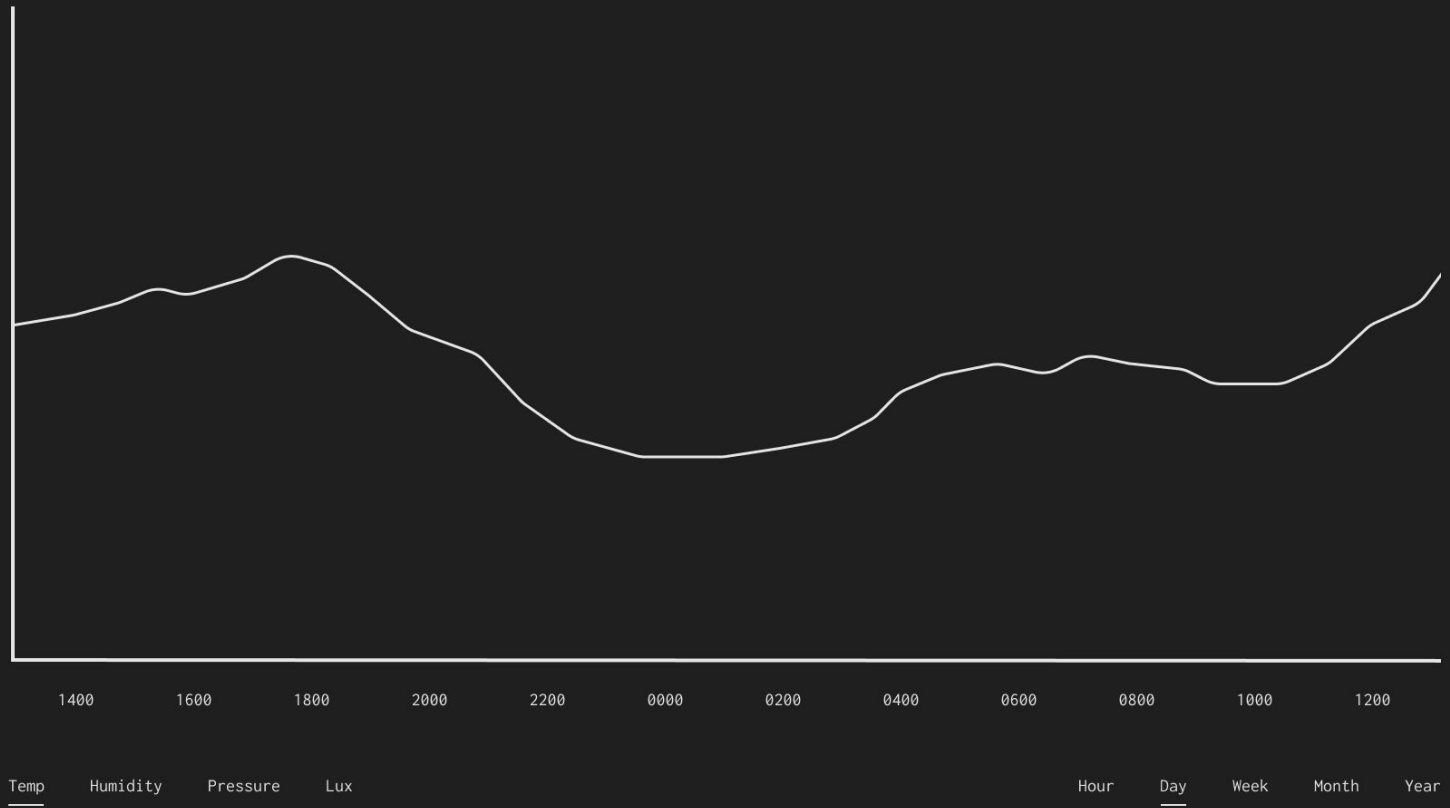
13.354
Longitude

11.2°C
Temp

91%
Humidity

1012 hPa
Pressure

107
Lux



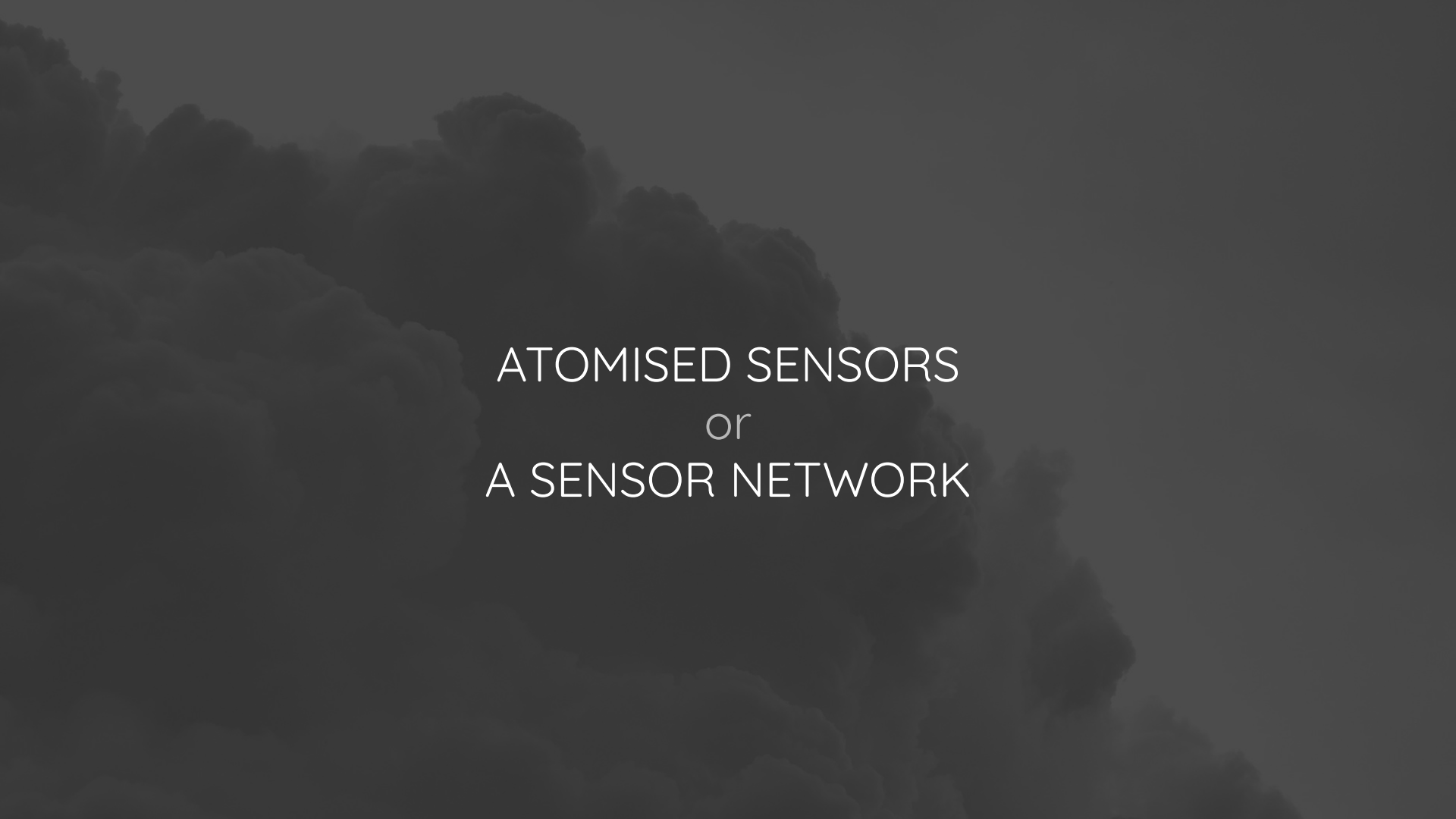
Temp Humidity Pressure Lux

Hour Day Week Month Year

INTERLUDE

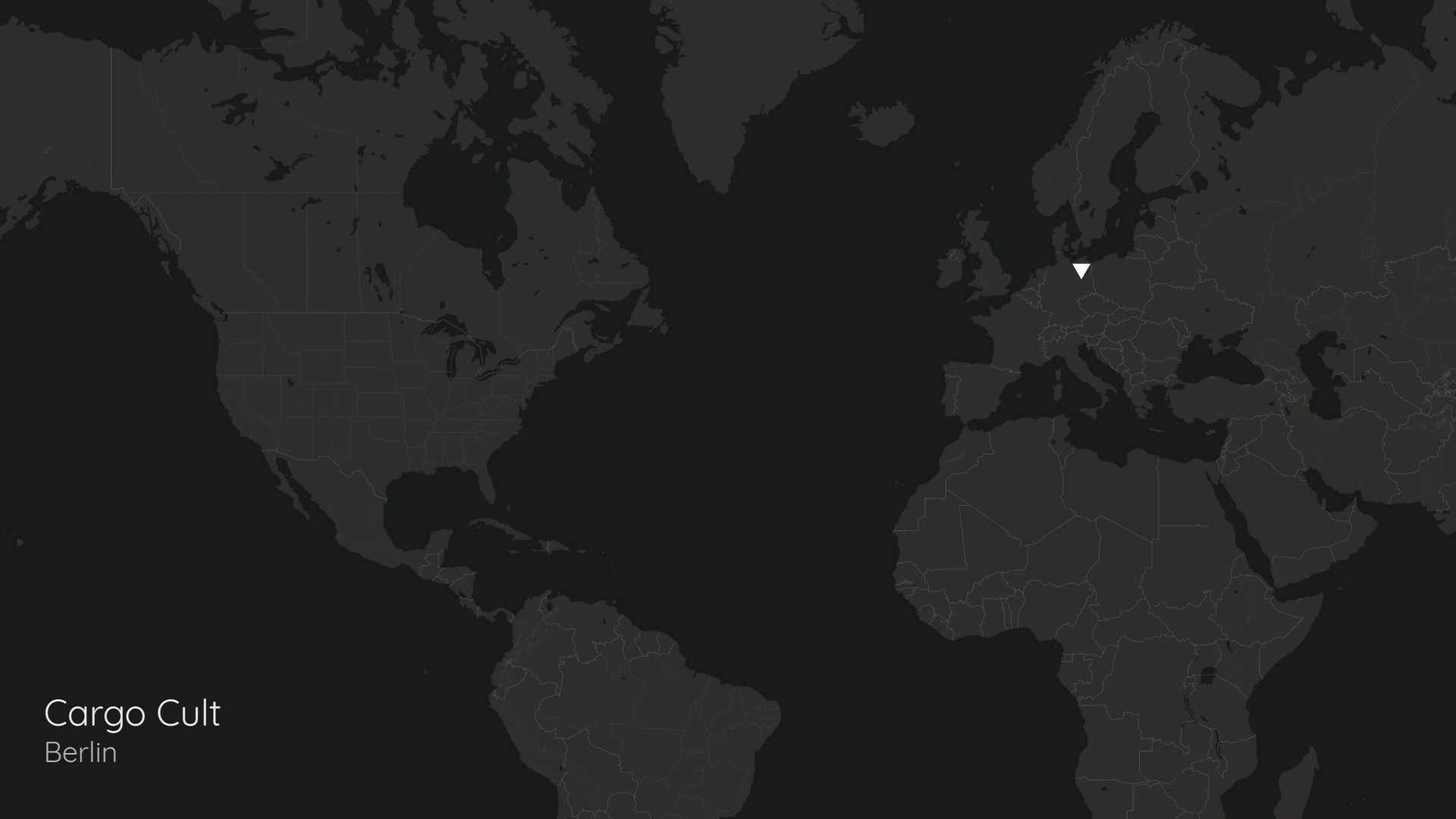

SCALABILITY

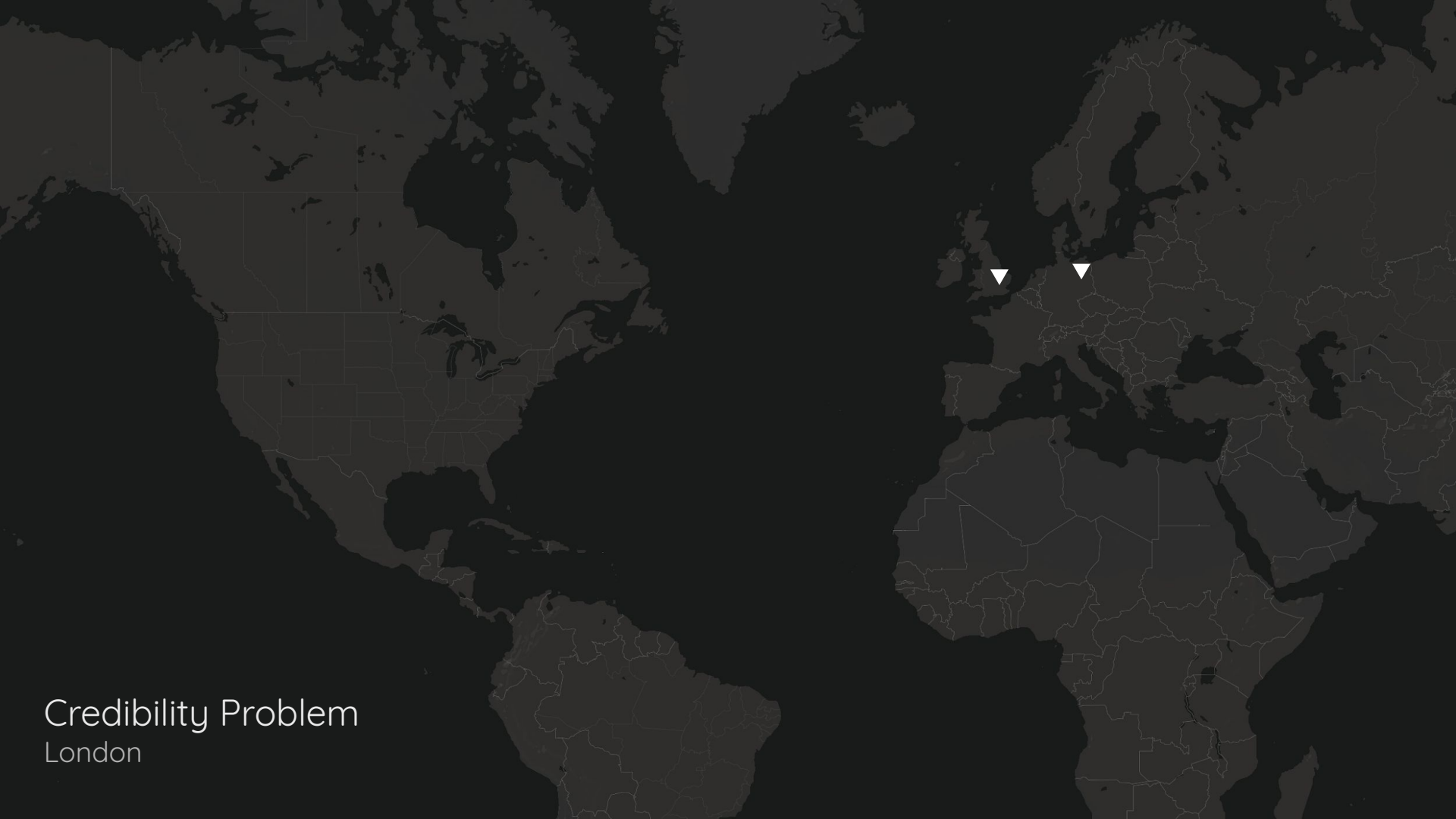
FRIENDS WANTED A REGN
STATION FOR THEIR
OWN PURPOSES

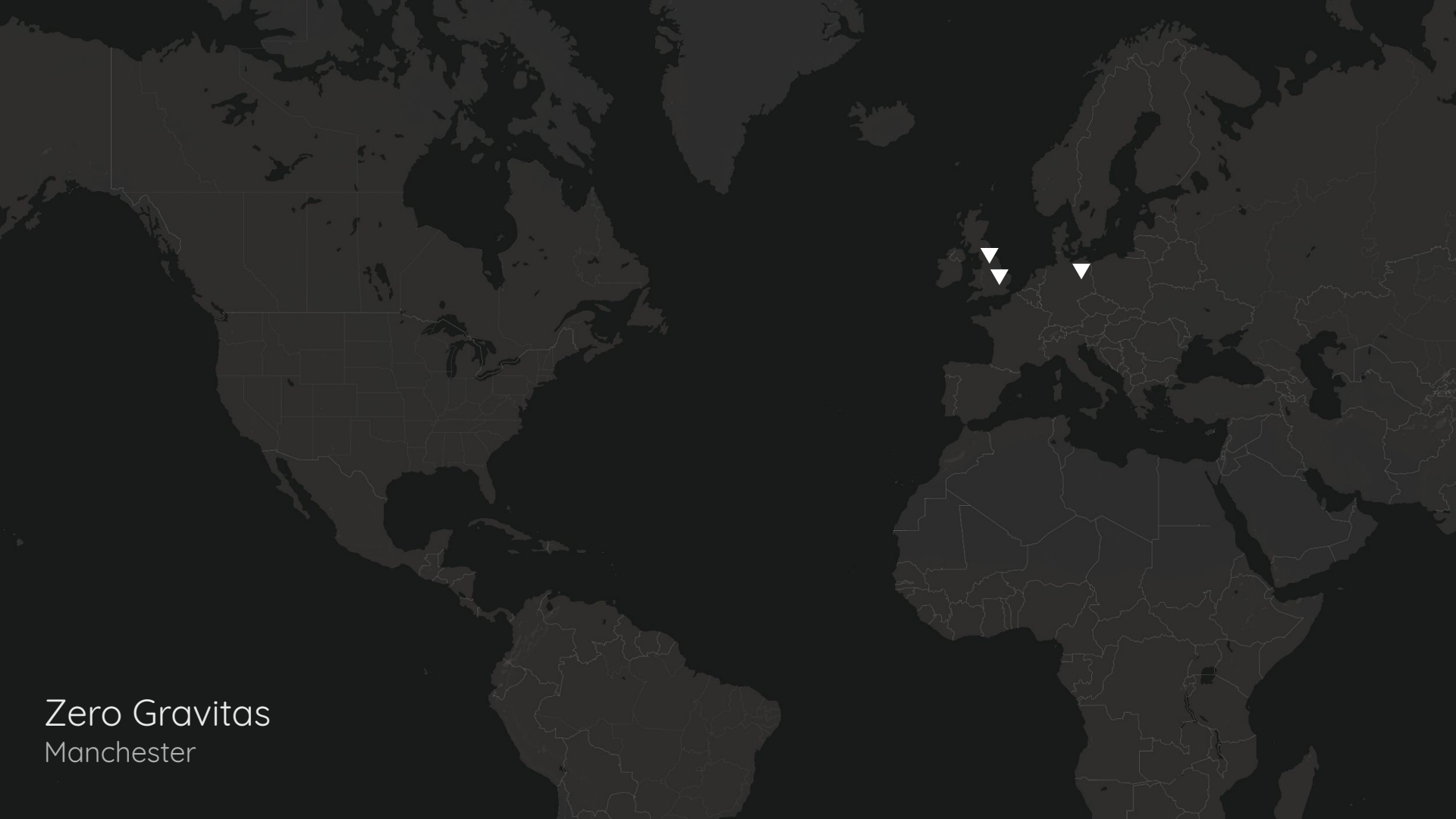

ATOMISED SENSORS
or
A SENSOR NETWORK

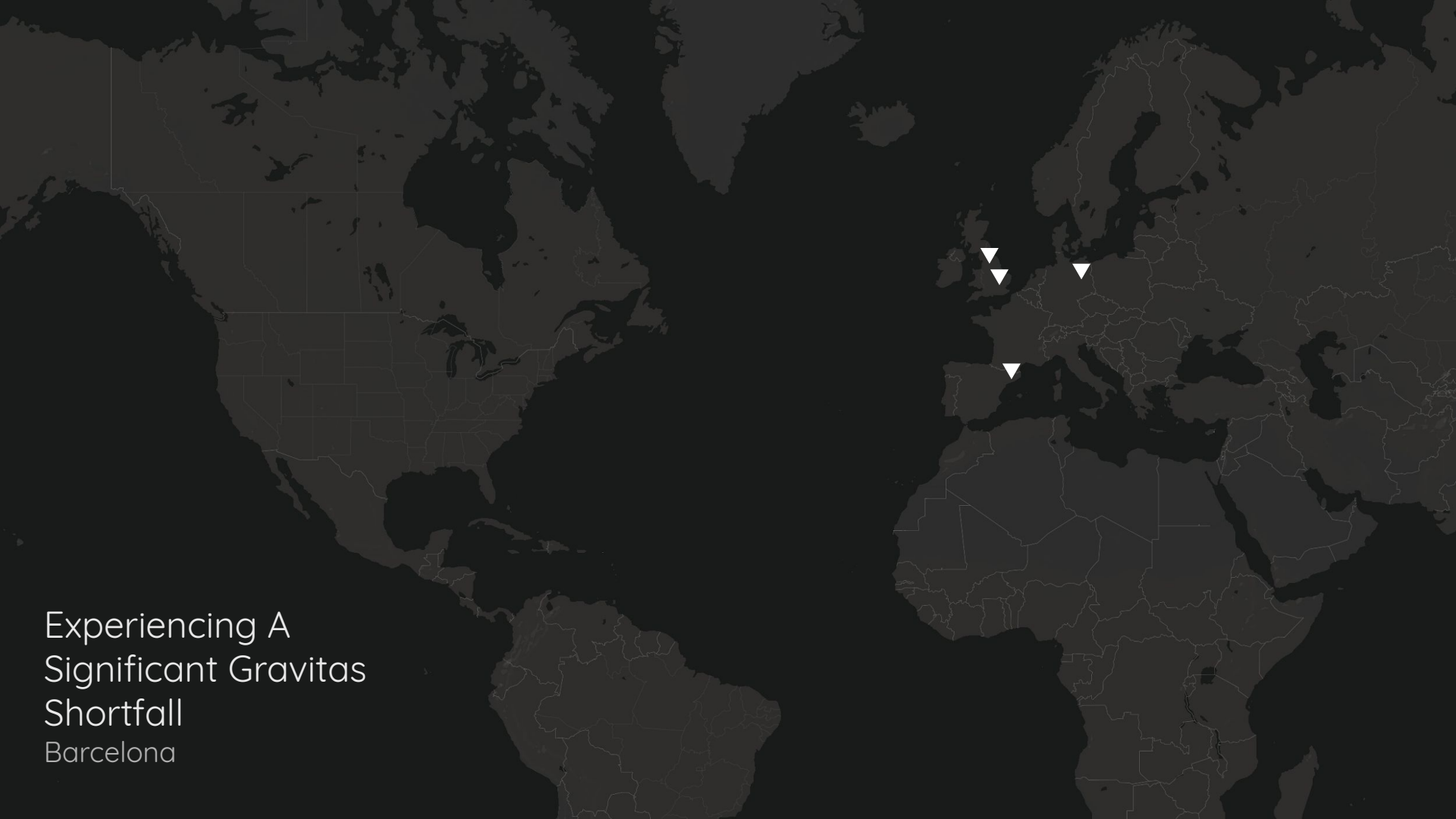
ACT 2

THE GANG MAKES A WEATHER STATION NETWORK


Cargo Cult
Berlin

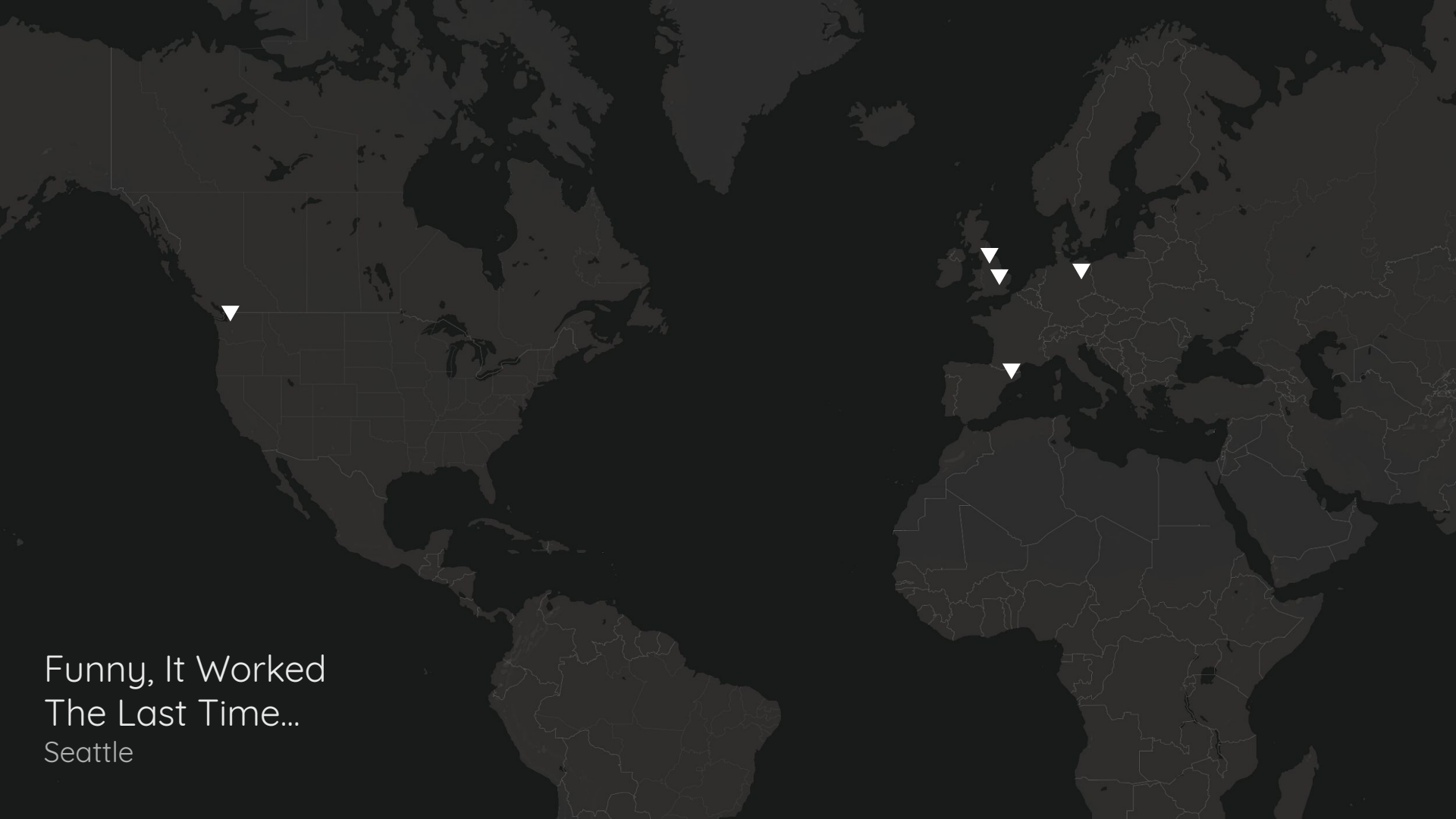

Credibility Problem
London


Zero Gravitas
Manchester

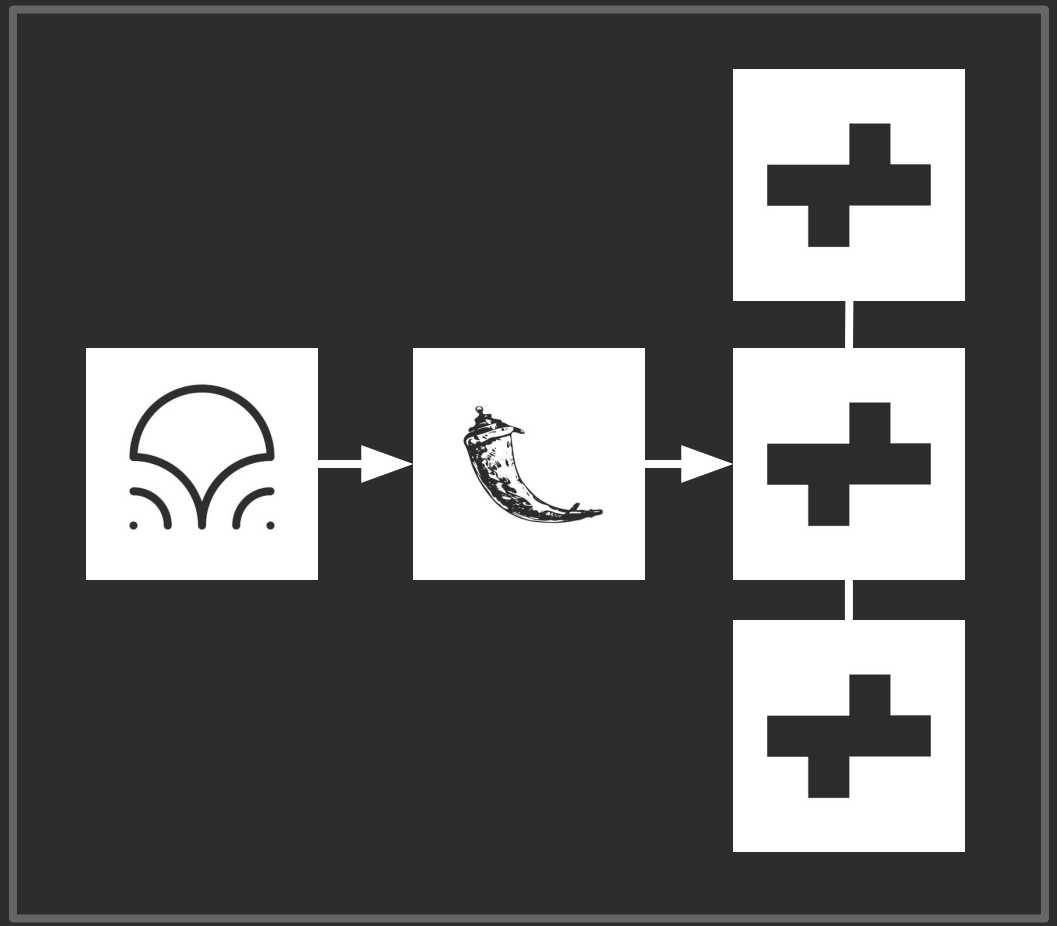


Experiencing A Significant Gravitas Shortfall

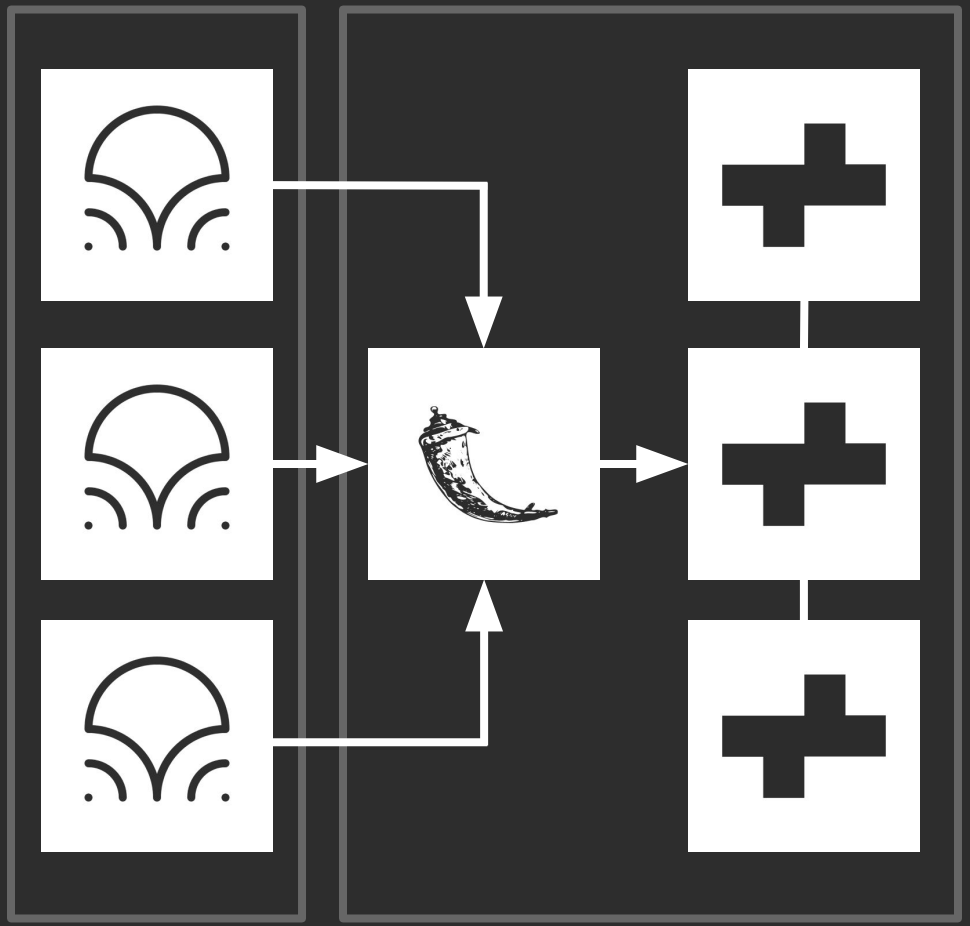
Barcelona


Funny, It Worked
The Last Time...
Seattle

My Stuff

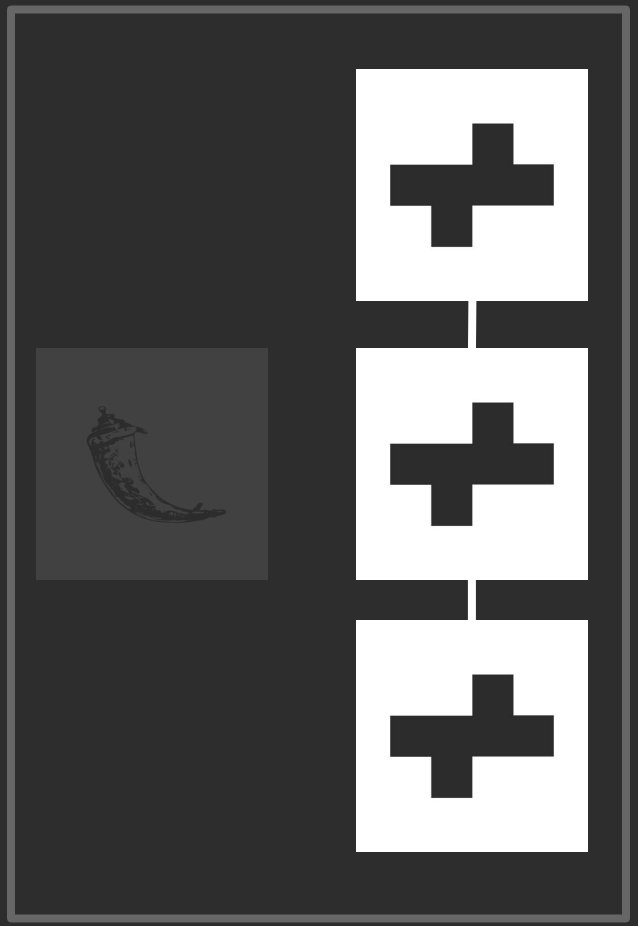


Not My Stuff

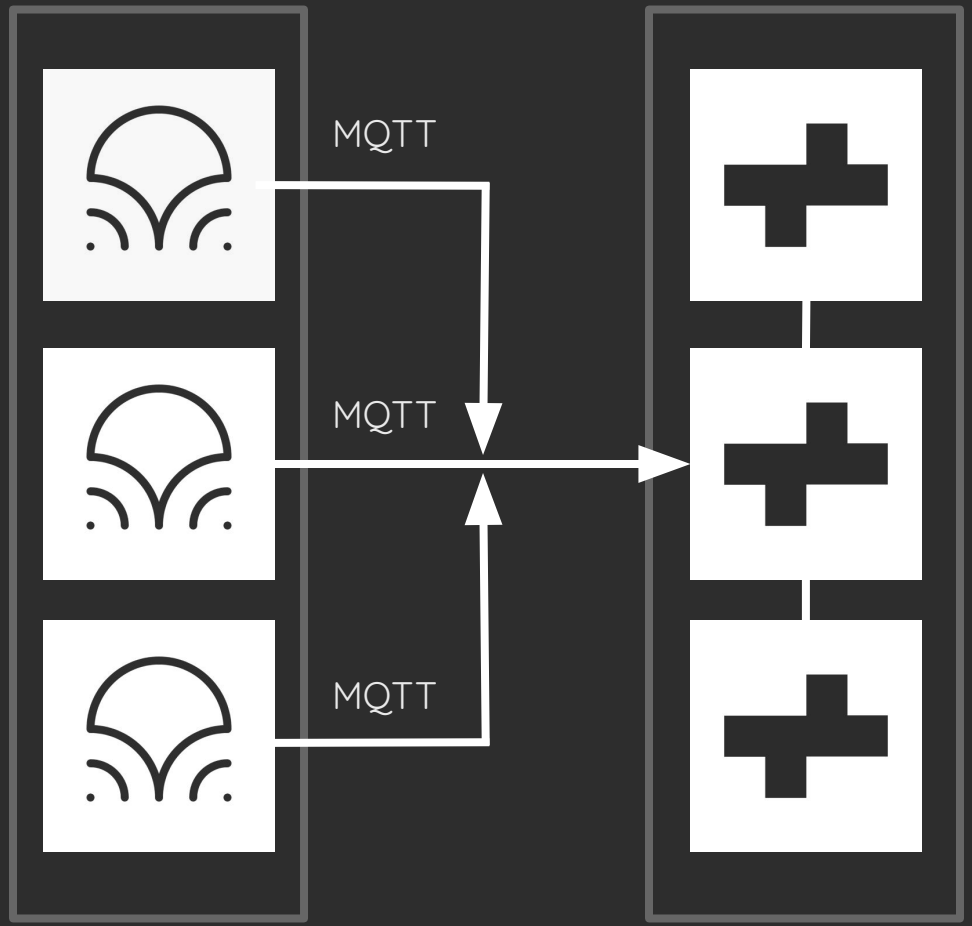


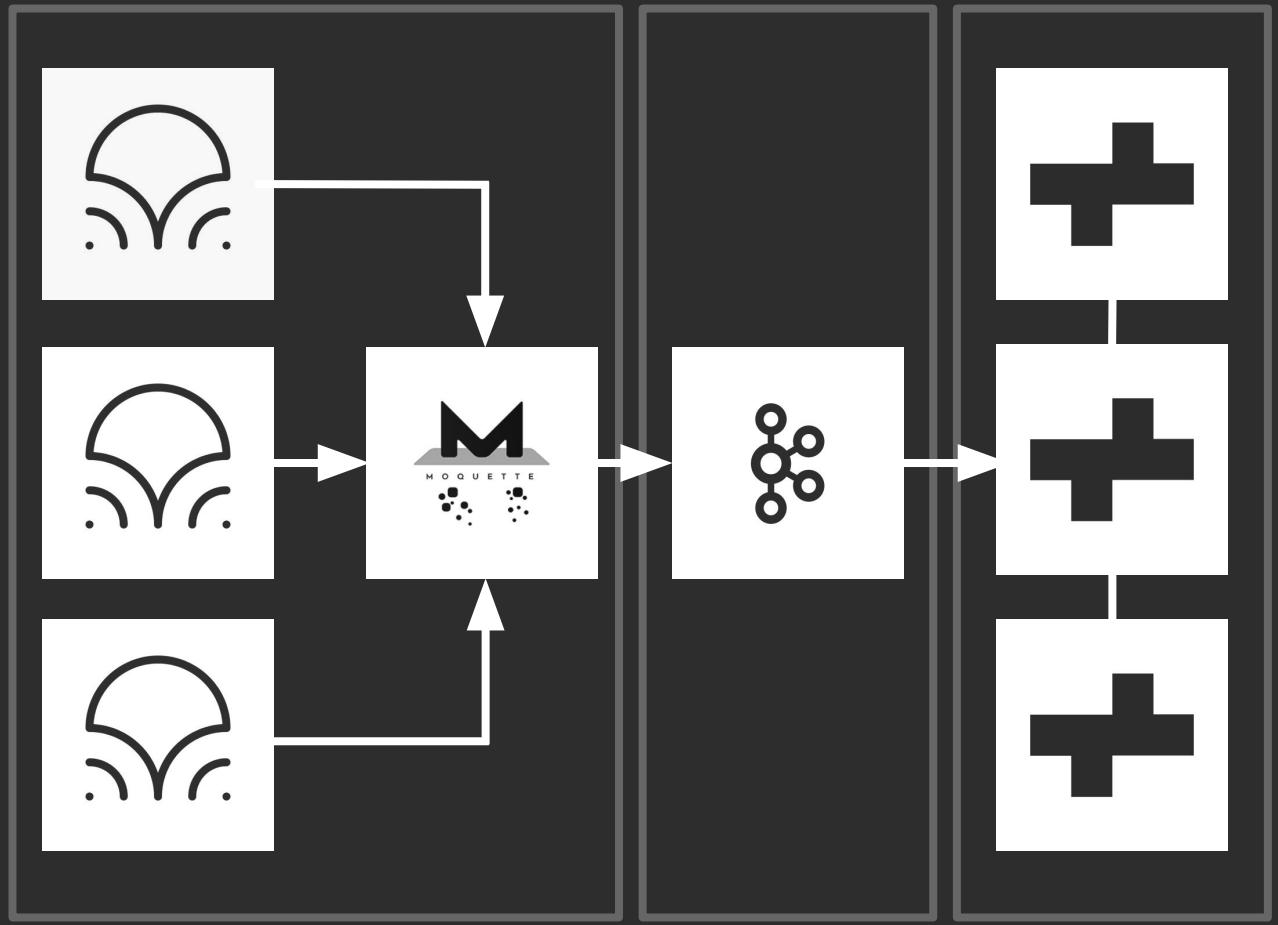
My Stuff?
Our Stuff?

Not My Stuff



My Stuff?
Our Stuff?


MQTT
MQTT
MQTT


MOQUETTE

```
// Libraries
#define MQTT_MAX_PACKET_SIZE 256
#include <PubSubClient.h>


// Constants
WiFiClient espClient;
PubSubClient client(espClient);
```

Setup

```
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Received Message :: ");
  Serial.print(topic);
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}
```

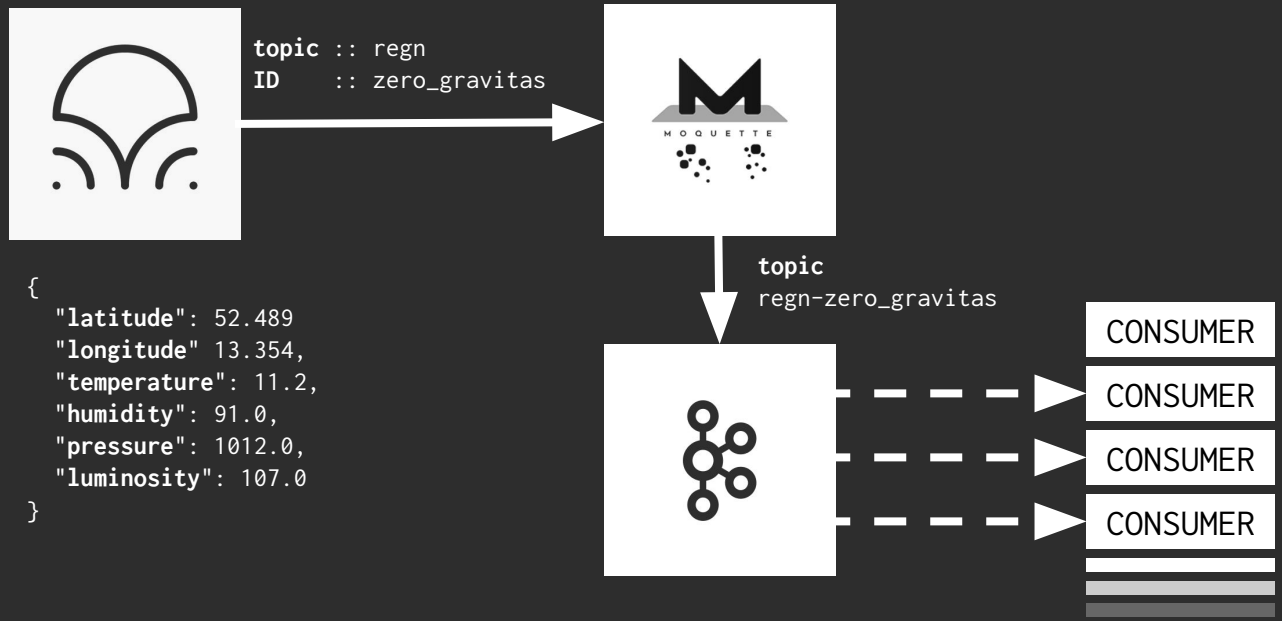
Setup

```
client.setServer(mqtt_server, 1883);
client.setCallback(callback);

if (client.connect("zero_gravitas")) {
  client.subscribe("regn/zero_gravitas");
}
```

Setup

```
StaticJsonBuffer<256> JSONbuffer;
JsonObject& JSONencoder = JSONbuffer.createObject();
JSONencoder["latitude"] = sensor_latitude;
JSONencoder["longitude"] = sensor_longitude;
JSONencoder["temperature"] = temperature;
JSONencoder["humidity"] = humidity;
JSONencoder["pressure"] = pressure;
JSONencoder["luminosity"] = lux;
char JSONmessageBuffer[256];
JSONencoder.prettyPrintTo(JSONmessageBuffer, sizeof(JSONmessageBuffer));

client.publish("regn", data);
delay(sensor_interval);
```

Loop




```
{
  "latitude": 52.489
  "longitude" 13.354,
  "temperature": 11.2,
  "humidity": 91.0,
  "pressure": 1012.0,
  "luminosity": 107.0
}
```

CONCLUSION
LESSONS // FUTURE WORK

ESP8266 IS A GREAT INTRO
TO IOT PROJECTS

TIGHTLY COUPLED IOT DATA PIPELINES ARE KINDA BAD


CRATEDB & KAFKA ARE
PRETTY AWESOME

FUTURE

LEVERAGE MQTT MORE

BROADEN CRATEDB INGESTION

EXPERIMENT WITH OUR DATA!

Twitter	@autophagyDev
Github	autophagy
Email	mika@crate.io