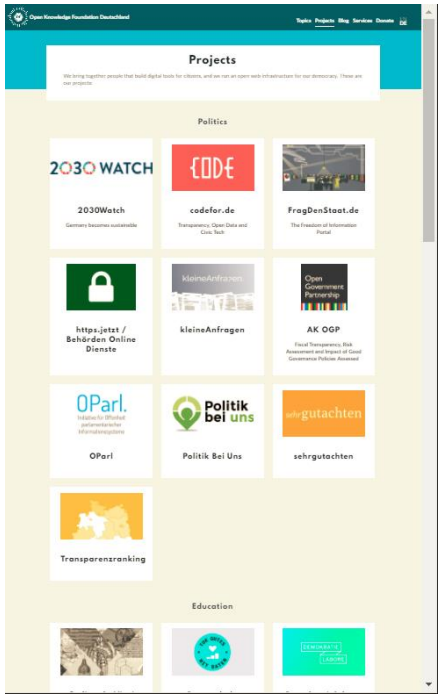
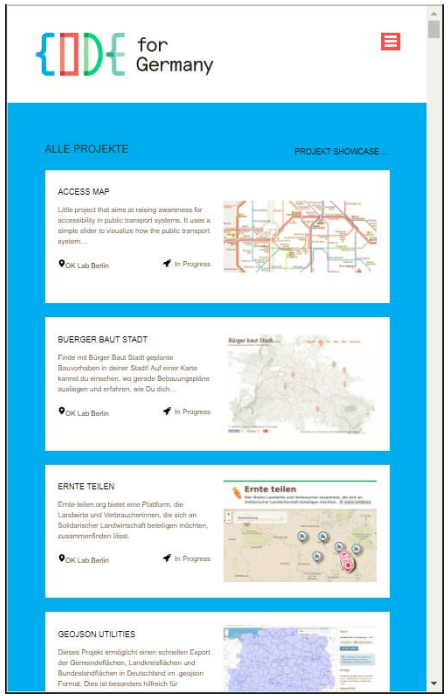
sensorplane.io

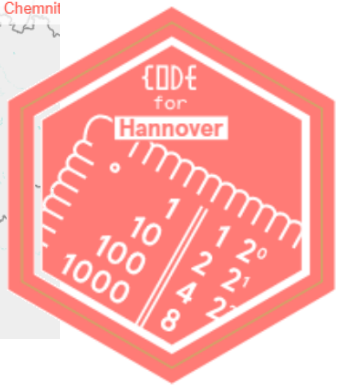
Nikolaus Pohle. OK Lab Hannover. Sep 28, 2017

OK Labs

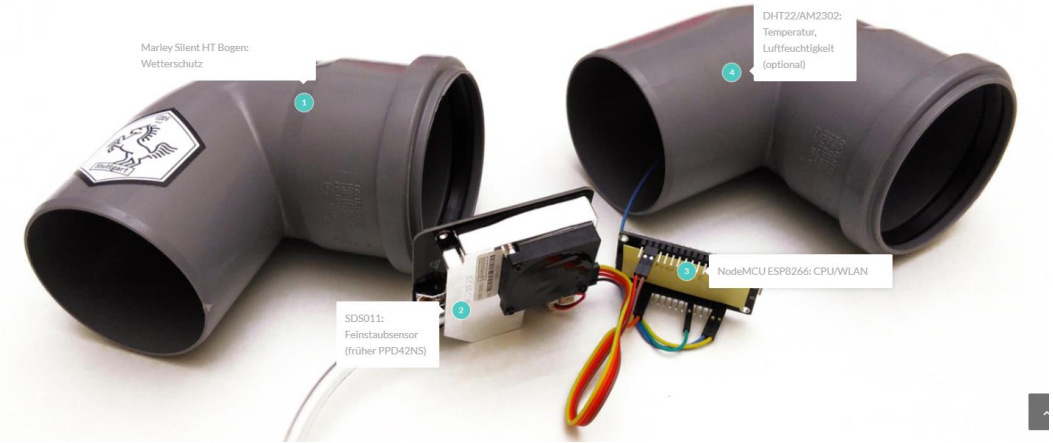

Projects
Politics
2030Watch
codefor.de
FragDenStaat.de
https.jetzt / Behörden Online Dienste
kleineAnfragen
AK OGP
OParl
Politik Bei Uns
sehgutachten
Transparenzranking
Education

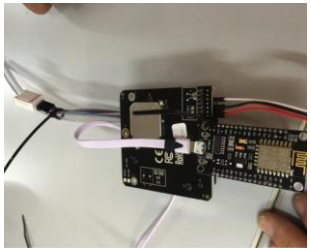

for Germany
ALLE PROJEKTE
PROJEKT SHOWCASE
ACCESS MAP
BUERGER BAUT STADT
ERNTE TEILEN
GEOJSON UTILITIES

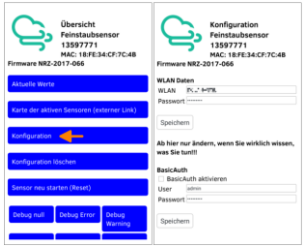

OK Lab Hamburg
OK Lab Hannover
OK Lab Potsdam
OK Lab Münster
OK Lab Wuppertal
OK Lab Leipzig
OK Lab Jena
OK Lab Gießen
OK Lab Frankfurt
OK Lab Heidelberg
OK Lab Karlsruhe
OK Lab Heilbronn
OK Lab Stuttgart
OK Lab Ulm
OK Lab München

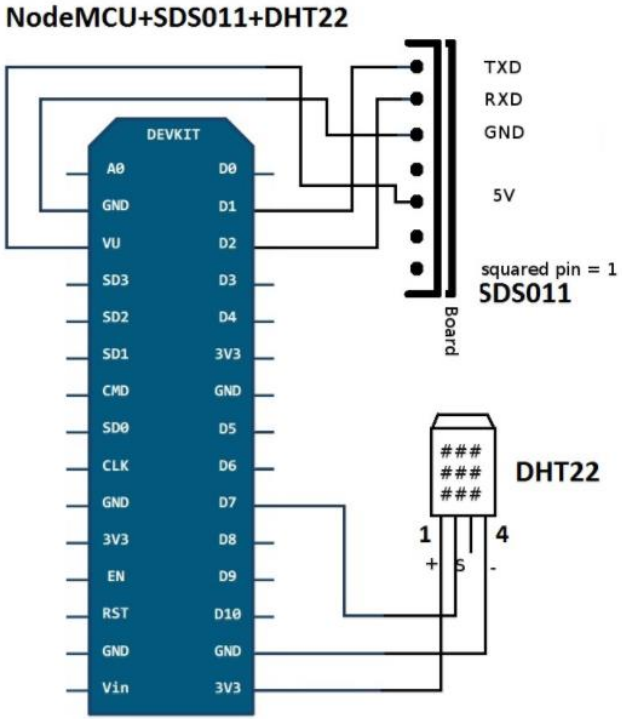

for
Hannover

OKLab Stuttgart: DIY Feinstaubsensor



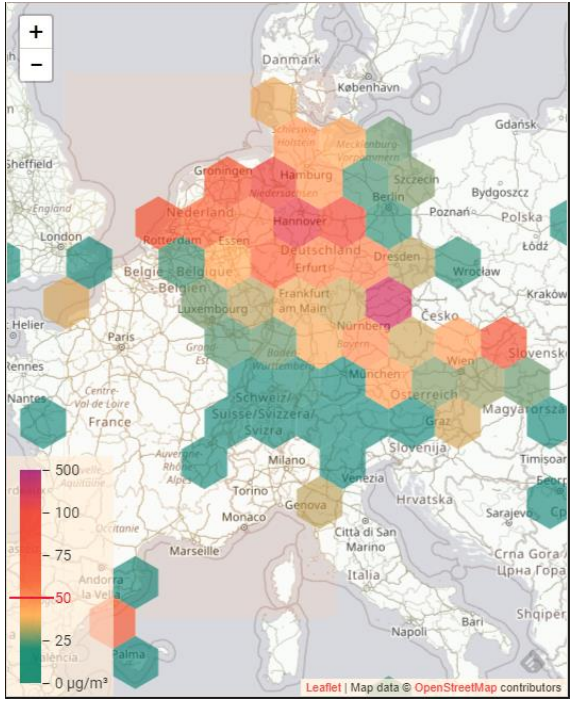


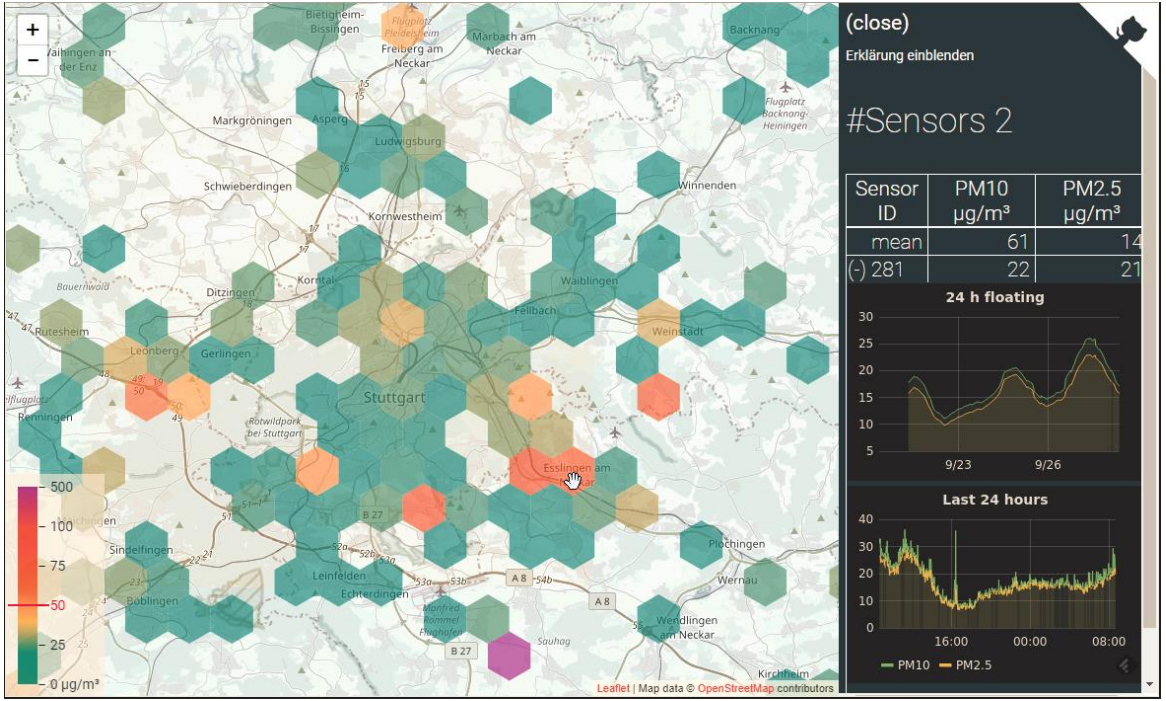




Source: http://luftdaten.info/

OK Lab Stuttgart: Data Visualization





Source: http://luftdaten.info/

sensorplane.io

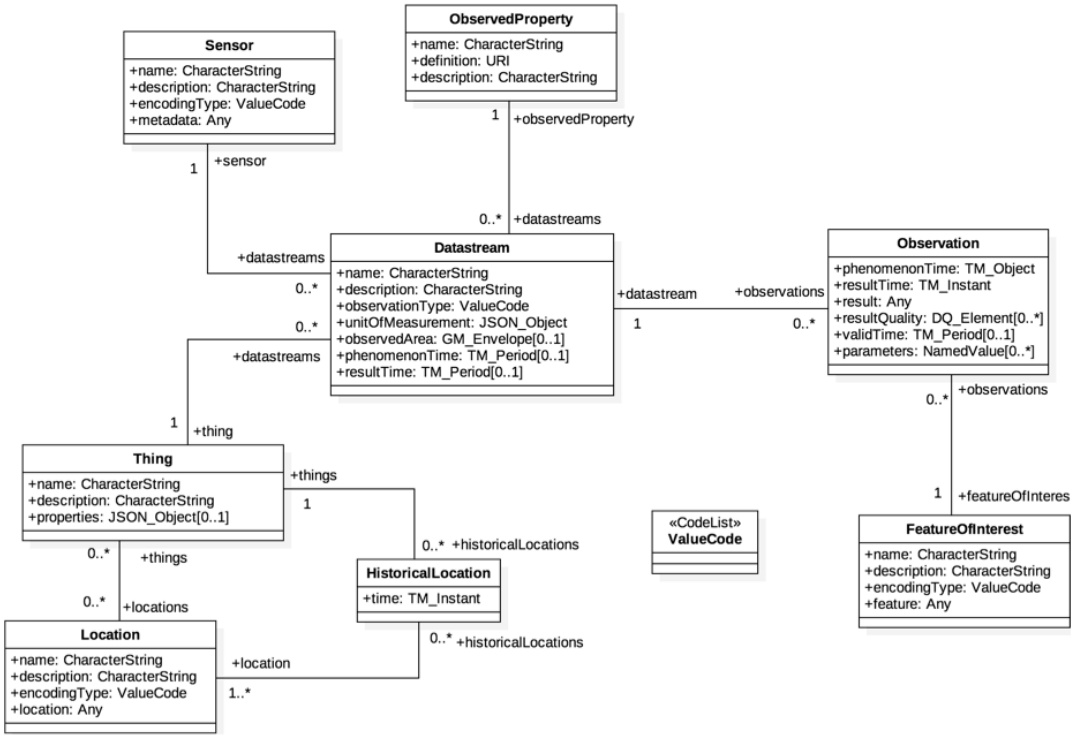
While the existing Backend API is effective it comes with a couple of downsides:

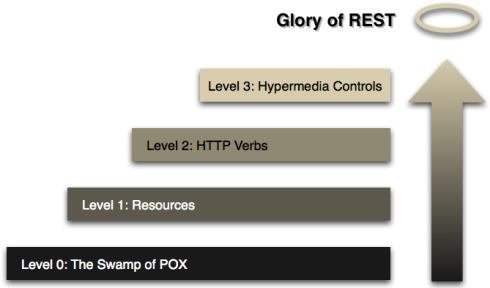
- not flexible: fixed number of measurement dimensions.
- not rich: no support for metadata.
- not elastic: architecture not cloud native.

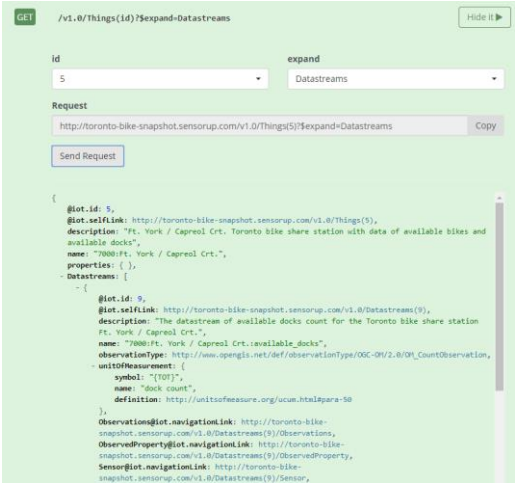
PaaS services like AWS are potentially a very good fit for a project like this:

- Fully elastic with costs close to zero during ramp up phase. Super easy to scale out and in.
- Security and availability are covered out of the box. No patching, no Restores, no DDOS
- Multiple runtimes available (Node, Java, PHP, etc.)
- Availability of deployment frameworks like serverless.com

SensorThing API



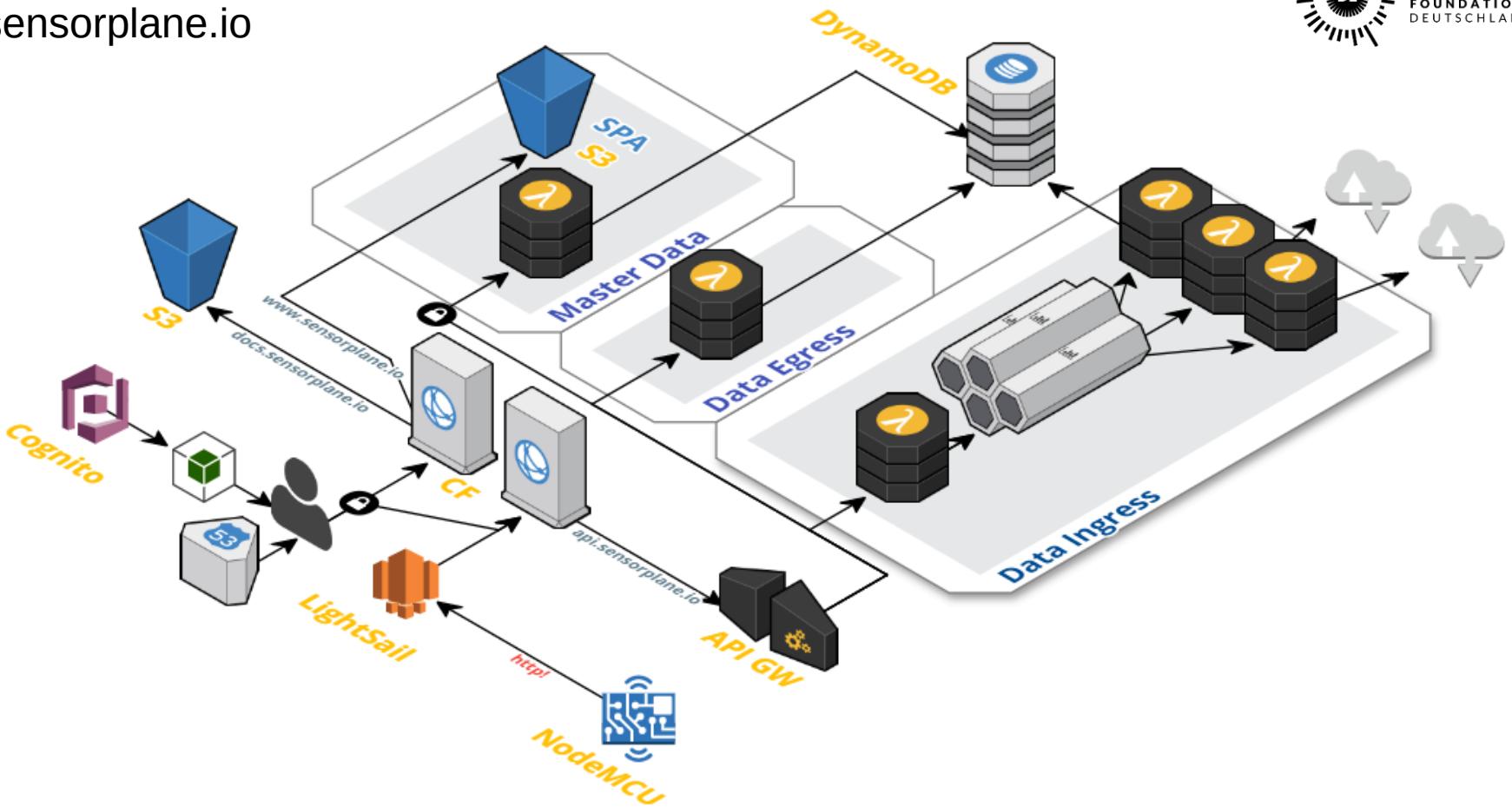




https://github.com/opengeospatial/sensorthings

https://martinfowler.com/articles/richardsonMaturityModel.html

sensorplane.io



sensorplane.io

- Entities are represented as POJOs with some business logic.
- List, Find, Save, Render JSON output (recursively)

```
1   'use strict';
2   var AWS = require("aws-sdk");
3
4   var Datastream = function (data) {
7   }
8
9   Datastream.prototype.data = {}
10
11  Datastream.prototype.renderAPIOutput = function(baseurl) {
37
38  }
39
40  Datastream.prototype.save = function(author) {
92
93  };
94
95  Datastream.prototype.expandObservedProperty = function() {
104 }
105
106 Datastream.list = function (top, start) {
133
134 }
135
136 Datastream.findById = function (datastream_id) {
169
170 }
171
172 Datastream.guid = function () {
180 }
181
182 module.exports = Datastream;
183
184 var ObservedProperty = require('model/ObservedProperty');
185 var Sensor = require('model/Sensor');
186 var Thing = require('model/Thing');
```

```
1   'use strict';
2   var AWS = require("aws-sdk");
3
4   var Sensor = function (data) {
7   }
8
9   Sensor.prototype.data = {}
10
11  Sensor.prototype.renderAPIOutput = function(baseurl) {
23
24  }
25
26  Sensor.prototype.save = function(author) {
45
46  };
47
48  Sensor.list = function (top, start) {
71
72  }
73
74  Sensor.findById = function (sensor_id) {
96
97  }
98
99  module.exports = Sensor;
100
```

sensorplane.io

```
'use strict';
var AWS = require("aws-sdk");
var jsonpath = require('jsonpath');
var sensorthing = require('sensorthing');

var Sensor = require('model/Sensor');
var ObservedProperty = require('model/ObservedProperty');
var Datastream = require('model/Datastream');
var Thing = require('model/Thing');
var Location = require('model/Location');
var Sensor = require('model/Sensor');
```

```
module.exports.post_datastream = (event, context, callback) => {

  console.log("Received event:", event);

  var eventBody = event.body;

  var datastream = new Datastream(event.body);
  var datapromise = datastream.save(event.cognitoPoolClaims.sub);

  datapromise
    .then(function(datastream) {
      callback(null, datastream.renderAPIOutput(getBaseUrl(event)));
    })
    .catch(function(error) {
      console.log(error);
    });

};
```

```
Datastream.prototype.save = function(author) {

  var self = this;
  console.log("save Datastream ID", this.data.id);

  this.data.authors = [author];

  var docClient = new AWS.DynamoDB.DocumentClient();
  var dynamopromise = docClient.put(
        {
        "TableName": "sensorplane_datastreams",
        "Item" : this.data
        })
      .promise()
      .then(function() {
        console.log("put into DynamoDB: ", self.data);
        var sensor = Sensor.findById(self.data.sensor_id);
        var property = ObservedProperty.findById(self.data.observedproperty_id);
        var thing = Thing.findById(self.data.thing_id);
        return Promise.all([sensor, property, thing]);
      })
      .then(function(data) {
        console.log("found dependent entries: ", data);
        var sensor = data[0];
        var property = data[1];
        var thing = data[2];

        if (!sensor.data.datastream_ids.some(function(entry) {return entry == self.data.id;})) {
          sensor.data.datastream_ids.push(self.data.id);
        }
        var sensorpromise = sensor.save();

        if (!property.data.datastream_ids.some(function(entry) {return entry == self.data.id;})) {
          property.data.datastream_ids.push(self.data.id);
        }
        var propertypromise = property.save();

        if (!thing.data.datastream_ids.some(function(entry) {return entry == self.data.id;})) {
          thing.data.datastream_ids.push(self.data.id);
        }
        var thingpromise = thing.save();

        return Promise.all([sensorpromise, propertypromise, thingpromise]);
      })
      .then(function(data) {
        return self;
      })
      .catch(function(error) {
        console.log("Error in Datastream.save(): ", error);
      });

module.exports = Datastream;

var ObservedProperty = require('model/ObservedProperty');
var Sensor = require('model/Sensor');
var Thing = require('model/Thing');
```

sensorplane.io

- serverless.yaml maps events to JS functions.
- Events can be https requests or Kinesis messages

```
httpgetdatastreamlist:
   handler: egress.get_datastreamlist
   events:
     - http:
         path: Datastreams/
         method: get
         integration: lambda
         cors: true
   environment:
     stage: ${opt:stage, self:provider.stage}
```

```
 httpgetdatastreamroot:
   handler: egress.get_datastream
   events:
     - http:
         path: Datastreams/{id}
         method: get
         integration: lambda
         cors: true
         request:
           parameters:
             paths:
               id: true
   environment:
     stage: ${opt:stage, self:provider.stage}
```

```
 httppostdatastreamroot:
    handler: ingest.post_datastream
    events:
      - http:
          path: Datastreams
          method: post
          integration: lambda
          cors: true
          authorizer:
            arn: arn:aws:cognito-idp:us-east-1:06
            claims:
              - email
    environment:
      stage: ${opt:stage, self:provider.stage}
```

sensorplane.io

```yaml
swagger: "2.0"
info:
  description: "This is a service inspired by the <a href='http://docs.opengeospatia
  version: "1.0.0"
  title: "Sensor Map"
  contact:
    email: "nikolaus.pohle@sensormap.io"
host: "api.open.sensormap.io"
basePath: "/v1"
tags:
- name: "Things"
  description: "Everything about your Things"
- name: "Oberservations"
  description: "Access to collected observations"
schemes:
- "https"

securityDefinitions:
  ApiKeyAuth:
    type: apiKey
    in: header
    name: Authorization

paths:
  /Things/:
    get:
      tags:
      - "Things"
      summary: "List all Things"
      description: "No support for filters and selectors yet"
      operationId: "things"
      produces:
      - "application/json"
      responses:
        200:
          description: "successful operation"
          schema:
            type: "object"
            properties:
              "@iot.count":
                type: "integer"
              "@iot.nextLink":
                type: "string"
              value:
                type: "array"
                items:
                  $ref: "#/definitions/Thing"
        400:
          description: "Invalid status value"
  /Things/{thingId}/:
```