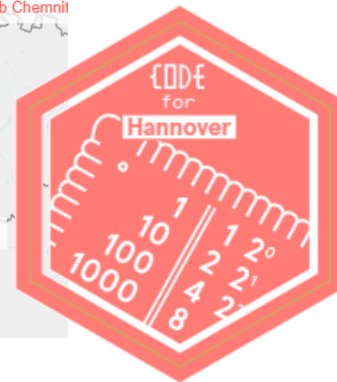
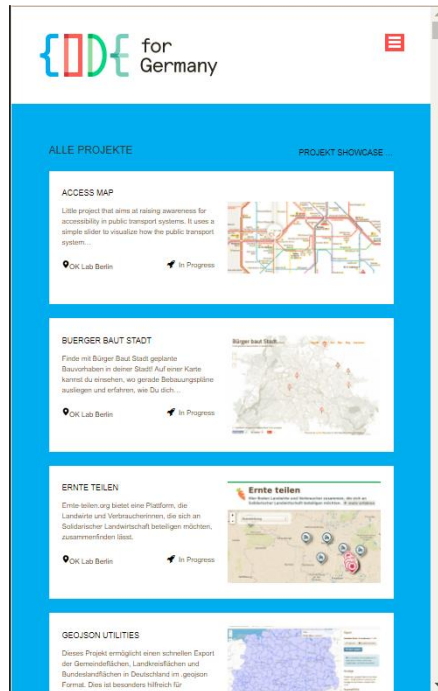
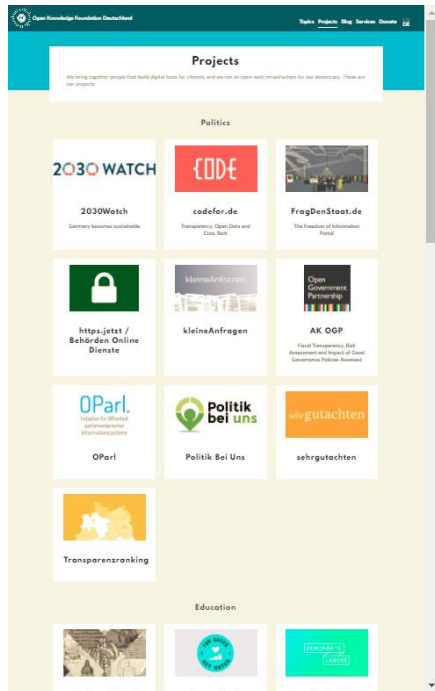
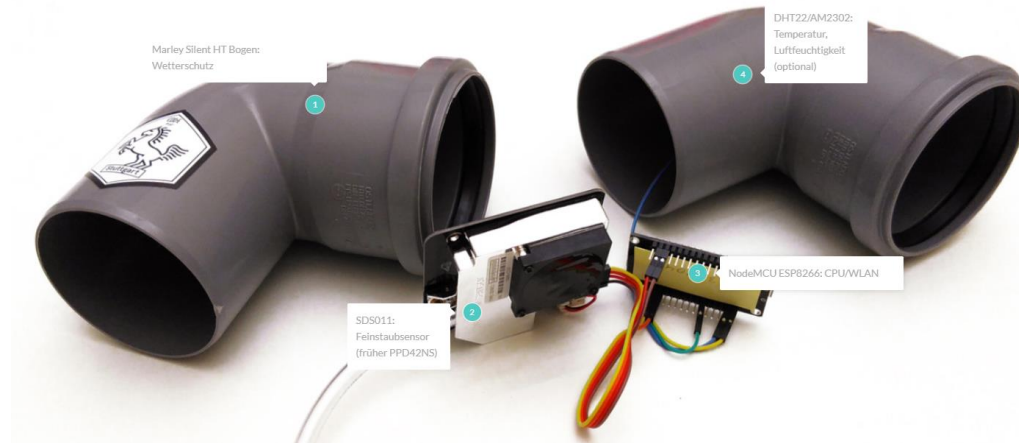


sensorplane.io

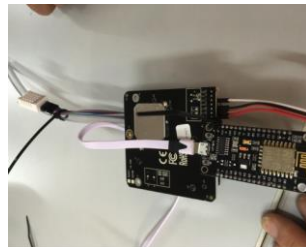
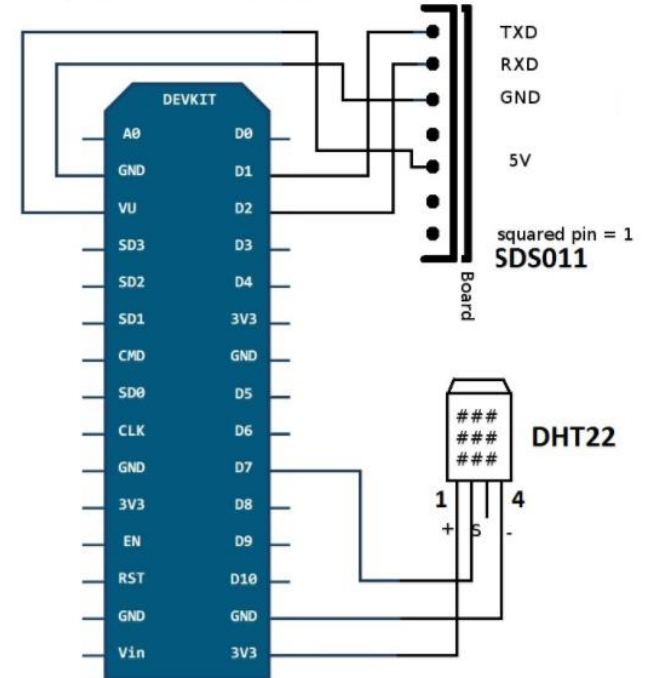
OK Labs



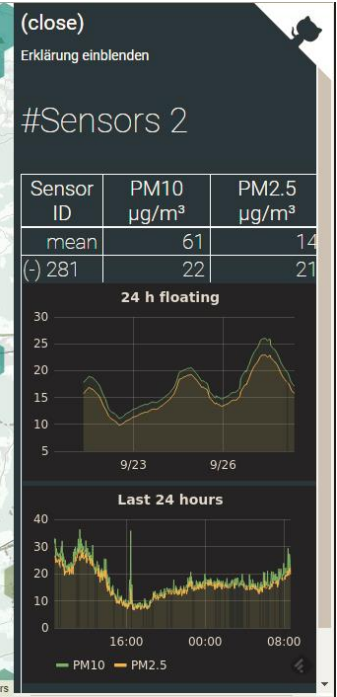
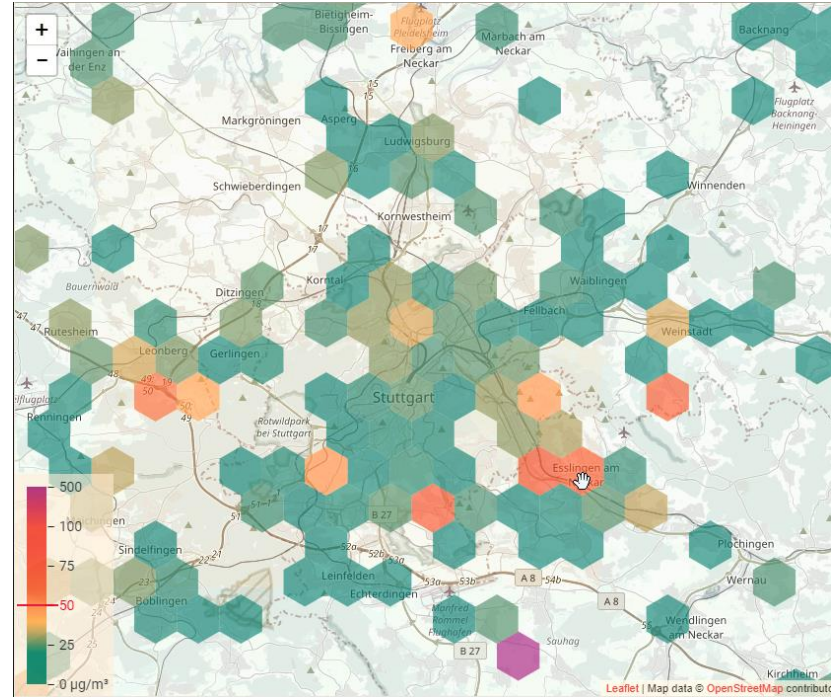
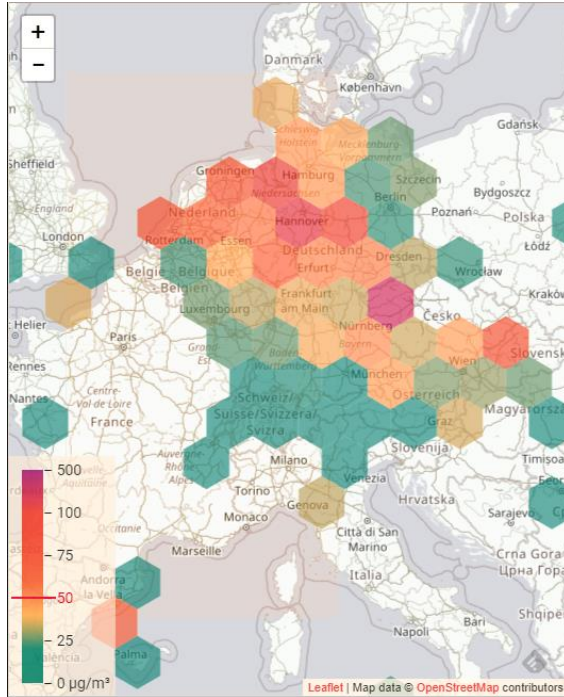
OKLab Stuttgart: DIY Feinstaubsensor



NodeMCU+SDS011+DHT22



OK Lab Stuttgart: Data Visualization



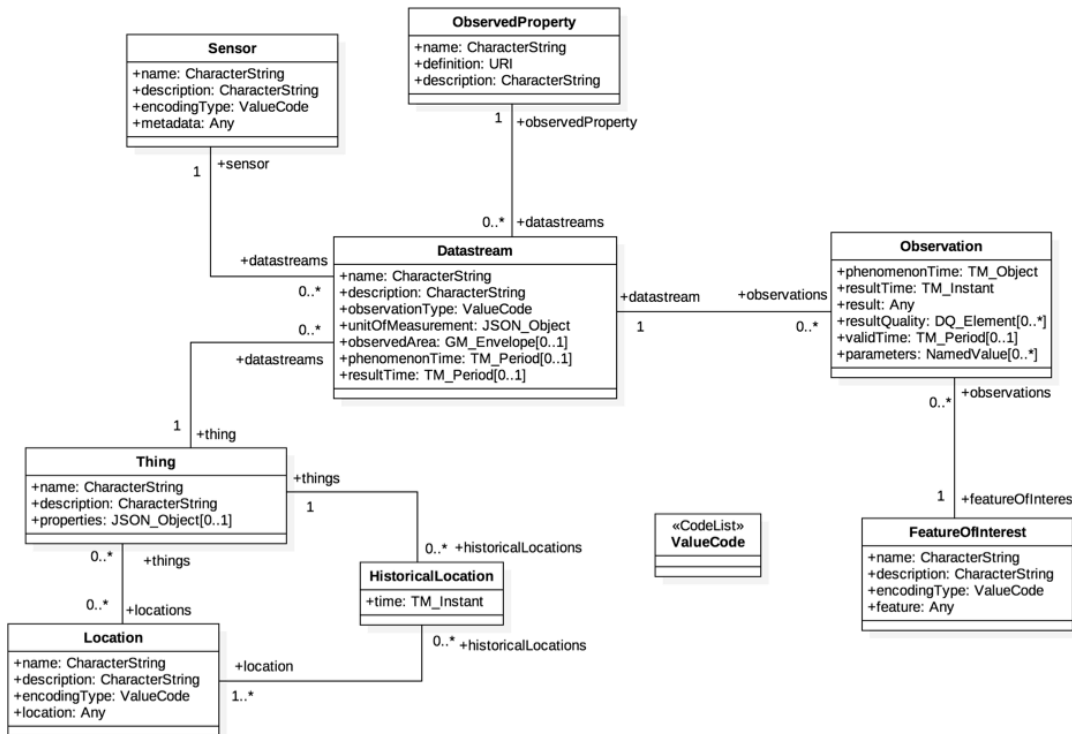
While the existing Backend API is effective it comes with a couple of downsides:

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

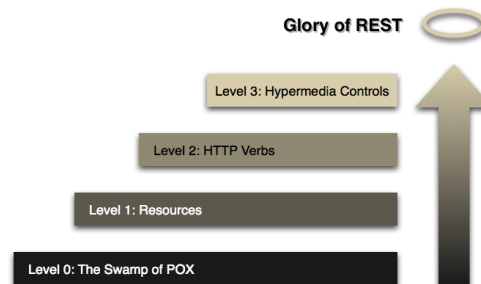
PaaS services like AWS are potentially a very good fit for a project like this:

- 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



Glory of REST



```

GET /v1.0/Things(14)?expand=Datastreams

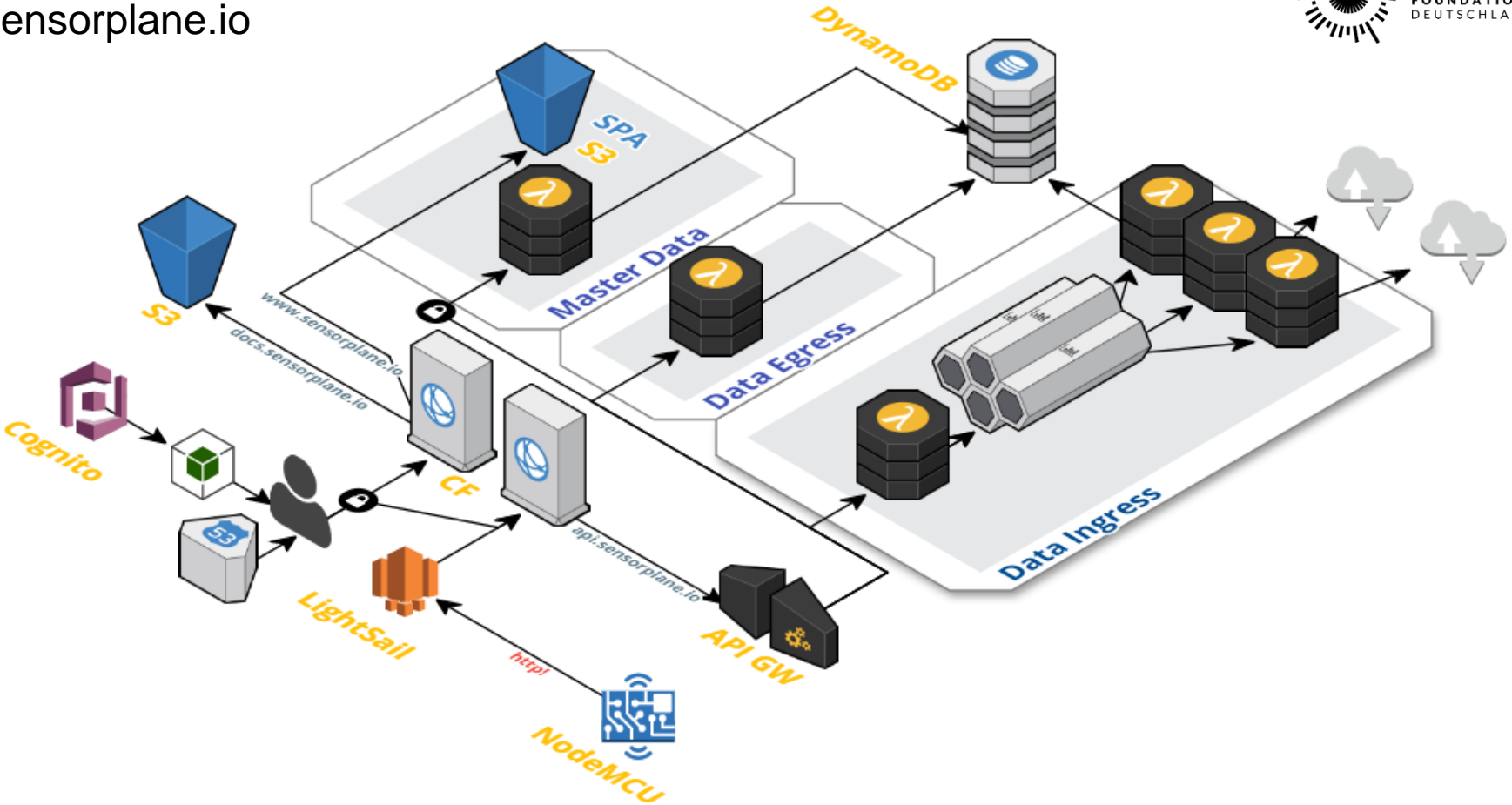
id: 5 expand: Datastreams

Request
http://toronto-bike-snapshot.sensorup.com/v1.0/Things(5)?expand=Datastreams

Send Request

{
  @id: 5,
  @iot.selfLink: http://toronto-bike-snapshot.sensorup.com/v1.0/Things(5),
  description: "Ft. York / Capreol Crt. Toronto bike share station with data of available bikes and available docks",
  name: "7800/Ft. York / Capreol Crt.",
  properties: { },
  Datastreams: [
    {
      @id: 5,
      @iot.selfLink: http://toronto-bike-snapshot.sensorup.com/v1.0/Datastreams(5),
      description: "The datastream of available docks count for the Toronto bike share station Ft. York / Capreol Crt.",
      name: "7800/Ft. York / Capreol Crt. available docks",
      observationType: http://www.opengis.net/def/observationType/OGC-0N/2.0/CountObservation,
      unitOfMeasurement: {
        symbol: "COUNT",
        name: "dock count",
        definition: http://unitsformeasure.org/ucum.html#para-50
      },
      Observations@iot.navigationLink: http://toronto-bike-snapshot.sensorup.com/v1.0/Datastreams(5)/Observations,
      ObservedProperty@iot.navigationLink: http://toronto-bike-snapshot.sensorup.com/v1.0/Datastreams(5)/ObservedProperty,
      Sensor@iot.navigationLink: http://toronto-bike-snapshot.sensorup.com/v1.0/Datastreams(5)/Sensor,
    }
  ]
}

```

- 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) {
5 }
6
7 Datastream.prototype.data = {}
8
9 Datastream.prototype.renderAPIOutput = function(baseUrl) {
10 }
11
12 Datastream.prototype.save = function(author) {
13 };
14
15 Datastream.prototype.expandObservedProperty = function() {
16 }
17
18 Datastream.list = function (top, start) {
19 }
20
21 Datastream.findById = function (datastream_id) {
22 }
23
24 Datastream.guid = function () {
25 }
26
27 module.exports = Datastream;
28
29 var ObservedProperty = require('model/ObservedProperty');
30 var Sensor = require('model/Sensor');
31 var Thing = require('model/Thing');
```

```
1 'use strict';
2 var AWS = require("aws-sdk");
3
4 var Sensor = function (data) {
5 }
6
7 Sensor.prototype.data = {}
8
9 Sensor.prototype.renderAPIOutput = function(baseUrl) {
10 }
11
12 Sensor.prototype.save = function(author) {
13 };
14
15 Sensor.list = function (top, start) {
16 }
17
18 Sensor.findById = function (sensor_id) {
19 }
20
21 module.exports = Sensor;
```


sensorplane.io

```
1 'use strict';
2 var AWS = require("aws-sdk");
3 var jsonpath = require('jsonpath');
4 var sensorthing = require('sensorthing');
5
6 var Sensor = require('model/Sensor');
7 var ObservedProperty = require('model/ObservedProperty');
8 var Datastream = require('model/Datastream');
9 var Thing = require('model/Thing');
10 var Location = require('model/Location');
11 var Sensor = require('model/Sensor');
12
185 module.exports.post_datastream = (event, context, callback) => {
186
187     console.log("Received event:", event);
188
189     var eventBody = event.body;
190
191     var datastream = new Datastream(event.body);
192     var datapromise = datastream.save(event.cognitoPoolClaims.sub);
193
194     datapromise
195     .then(function(datastream) {
196         callback(null, datastream.renderAPIOutput(getBaseUrl(event)));
197     })
198     .catch(function(error) {
199         console.log(error);
200     });
201
202 };
203
```

```
40 Datastream.prototype.save = function(author) {
41
42     var self = this;
43     console.log("save Datastream ID", this.data.id);
44
45     this.data.authors = [author];
46
47     var docClient = new AWS.DynamoDB.DocumentClient();
48     var dynamopromise = docClient.put(
49         {
50             "TableName": "sensorplane_datastreams",
51             "Item": this.data
52         })
53     .promise()
54     .then(function() {
55         console.log("put into DynamoDB: ", self.data);
56         var sensor = Sensor.findById(self.data.sensor_id);
57         var property = ObservedProperty.findById(self.data.observedproperty_id);
58         var thing = Thing.findById(self.data.thing_id);
59         return Promise.all([sensor, property, thing]);
60     })
61     .then(function(data) {
62         console.log("found dependent entries: ", data);
63         var sensor = data[0];
64         var property = data[1];
65         var thing = data[2];
66
67         if (!sensor.data.datastream_ids.some(function(entry) {return entry == self.data.id;})) {
68             sensor.data.datastream_ids.push(self.data.id);
69         }
70         var sensorpromise = sensor.save();
71
72         if (!property.data.datastream_ids.some(function(entry) {return entry == self.data.id;})) {
73             property.data.datastream_ids.push(self.data.id);
74         }
75         var propertypromise = property.save();
76
77         if (!thing.data.datastream_ids.some(function(entry) {return entry == self.data.id;})) {
78             thing.data.datastream_ids.push(self.data.id);
79         }
80         var thingpromise = thing.save();
81
82         return Promise.all([sensorpromise, propertypromise, thingpromise]);
83     })
84     .then(function(data) {
85         return self;
86     })
87     .catch(function(error) {
88         console.log("Error in Datastream.save(): ", error);
89     });
90
182 module.exports = Datastream;
183
184 var ObservedProperty = require('model/ObservedProperty');
185 var Sensor = require('model/Sensor');
186 var Thing = require('model/Thing');
```

- 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}
```

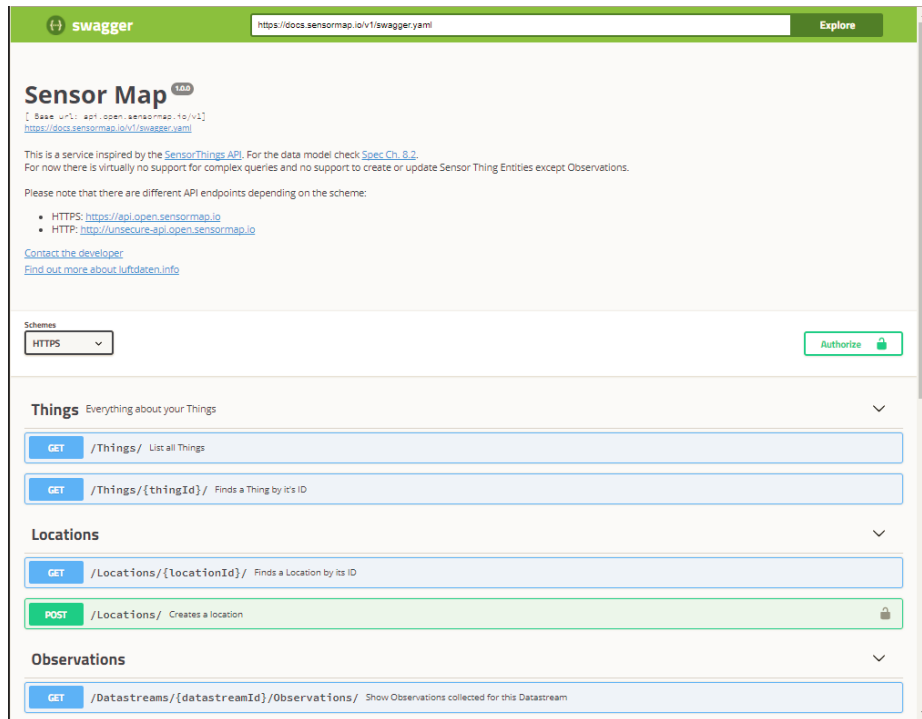
```
httpgetdatastreamroot:
  handler: egress.get_datastream
  events:
    - http:
        path: Datastreams/{id}
        method: get
        integration: lambda
        cors: true
        request:
          parameters:
            paths:
              id: true
  environment:
    stage: ${opt:stage, self:provider.stage}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

```
httppostdatastreamroot:
  handler: ingest.post_datastream
  events:
    - http:
        path: Datastreams
        method: post
        integration: lambda
        cors: true
        authorizer:
          arn: arn:aws:cognito-idp:us-east-1:0
          claims:
            - email
  environment:
    stage: ${opt:stage, self:provider.stage}
```

```

1  swagger: "2.0"
2  info:
3    description: "This is a service inspired by the <a href='http://docs.opengeospatia
4    version: "1.0.0"
5    title: "Sensor Map"
6    contact:
7      email: "nikolaus.pohle@sensorplane.io"
8    host: "api.open.sensorplane.io"
9    basePath: "/v1"
10   tags:
11     - name: "Things"
12       description: "Everything about your Things"
13     - name: "Observations"
14       description: "Access to collected observations"
15   schemes:
16     - "https"
17
18   securityDefinitions:
19     ApiKeyAuth:
20       type: apiKey
21       in: header
22       name: Authorization
23
24   paths:
25     /Things/:
26       get:
27         tags:
28           - "Things"
29         summary: "List all Things"
30         description: "No support for filters and selectors yet"
31         operationId: "things"
32         produces:
33           - "application/json"
34         responses:
35           200:
36             description: "successful operation"
37             schema:
38               type: "object"
39               properties:
40                 "@iot.count":
41                   type: "integer"
42                 "@iot.nextLink":
43                   type: "string"
44                 value:
45                   type: "array"
46                   items:
47                     $ref: "#/definitions/Thing"
48           400:
49             description: "Invalid status value"
50     /Things/{thingId}/:

```



The image shows the Swagger UI for the Sensor Map API. The top bar is green with the Swagger logo and the URL `https://docs.sensorplane.io/v1/swagger.yaml`. The main header is **Sensor Map** with a version tag `1.0.0`. Below the header, there is a description: "This is a service inspired by the [SensorThings API](#). For the data model check [Spec Ch. 8.2](#). For now there is virtually no support for complex queries and no support to create or update Sensor Thing Entities except Observations." It also notes that there are different API endpoints depending on the scheme:

- HTTPS: <https://api.open.sensorplane.io>
- HTTP: <http://unsecure-api.open.sensorplane.io>

 There are links to "Contact the developer" and "Find out more about luftdaten.info". A "Schemes" dropdown is set to "HTTPS", and there is an "Authorize" button. The main content area lists the API endpoints:

- Things** (Everything about your Things):
 - GET `/Things/` List all Things
 - GET `/Things/{thingId}/` Finds a Thing by its ID
- Locations**:
 - GET `/Locations/{locationId}/` Finds a Location by its ID
 - POST `/Locations/` Creates a location
- Observations**:
 - GET `/Datastreams/{datastreamId}/Observations/` Show Observations collected for this Datastream