ARRAY OF THINGS

API Primer

What is the Array of Things?

What if a light pole told you to watch out for an icy patch of sidewalk ahead? What if an app told you the most populated route for a late-night walk to the El station by yourself? What if you could get weather and air quality information block-by-block, instead of city-by-city?

The Array of Things, AoT, is an urban sensing network of programmable, modular nodes that is being installed around cities to collect real-time data on the cities’ environment, infrastructure, and activity for research and public use. AoT will essentially serve as a “fitness tracker” for the communities where nodes have been deployed, measuring factors that impact livability in cities such as climate, air quality, and noise.

What types of data are collected?

There are several general categories of environmental data measured by the nodes: environmental, air quality, and image processing.

Environmental Data

There is a wide range of environmental data points that are collected by the nodes. These observations are present in almost all nodes, and they are considered stable. The observations that are captured in this group include:

- Temperature, ambient and inside the node
- Humidity, ambient and inside the node
- Barometric pressure
- Ambient, ultraviolet and infrared light
- Sound pressure
- Vibration
- Magnetic field

Air Quality Data

Air quality data is concerned with measuring gas and particulate matter concentrations in the air. Some of the observations have been thoroughly calibrated and are mostly stable, while some others in this category are still under active development. The observations that are captured in this group include:

- Carbon monoxide
- Ozone
- Sulfur dioxide
- Nitrogen dioxide
- Hydrogen sulfide
- Total reducing gases
- Total oxidizing gases

- PM 2.5
- PM 10

Image Processing

In addition to the sensors onboard the nodes, there are two cameras on each node. The cameras are used exclusively by software that runs on a special board within the node, the edge processor. The software use the image data to make aggregate and statistical computations: the raw images are never sent to any non-ephemeral storage and never leave the node. The information gathered by the cameras include:

- Vehicle count
- Pedestrian count
- Flooding detection

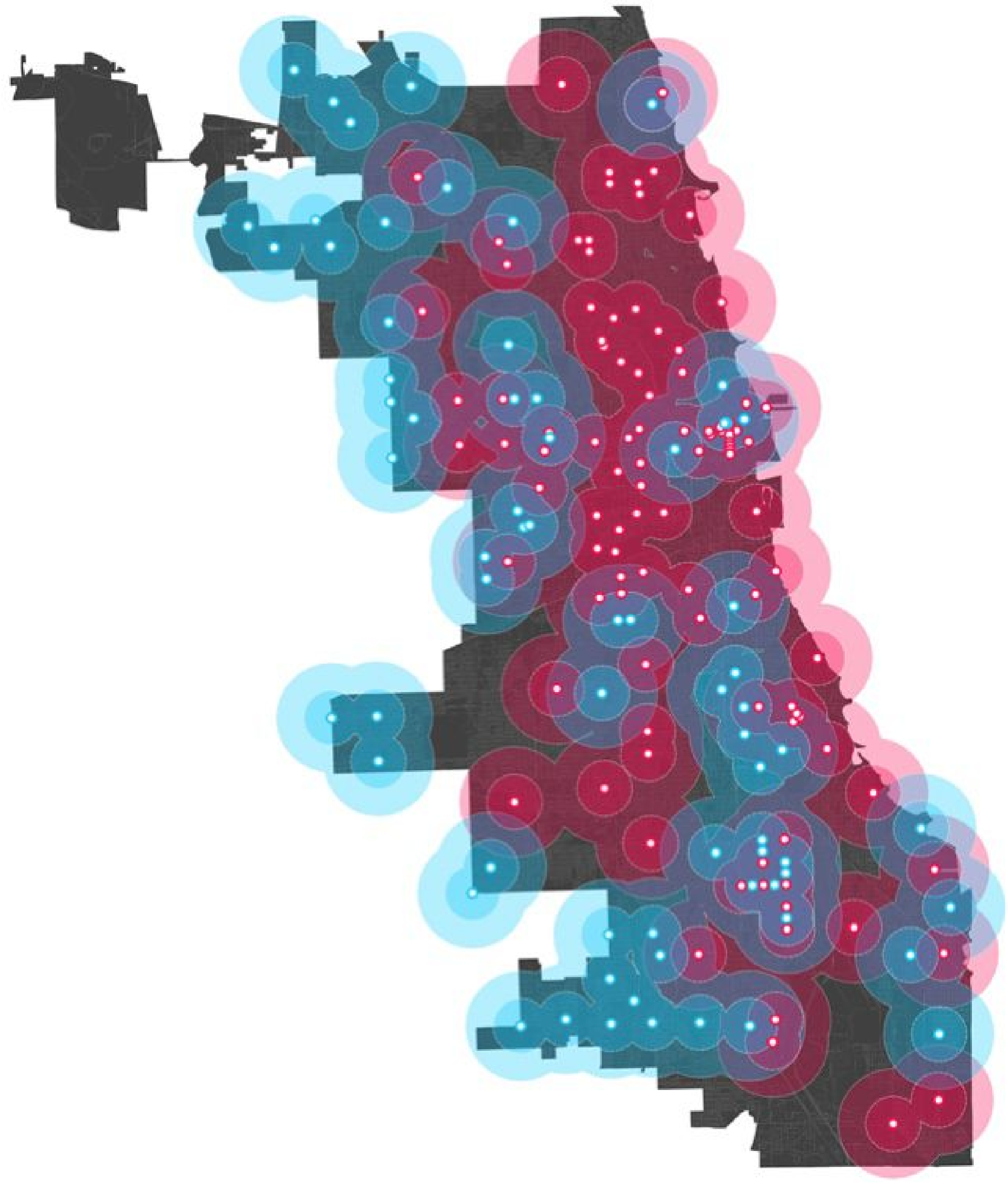
Who is behind Array of Things?

Array of Things is led by researchers from the Urban Center for Computation and Data at the Computation Institute, a joint initiative of Argonne National Laboratory and the University of Chicago under the direction of Argonne computer scientist Charlie Catlett. Efforts to develop the Waggle platform— the software and hardware that powers the Array of Things Project— have been led by Argonne Institute of Science and Engineering Co-Director Pete Beckman. Catlett, Beckman, and their associates are the minds behind the technical aspects of the project.

Array of Things has been executed in partnership with the City of Chicago, which has been installing nodes onto light posts across the city via its Department of Transportation. The City's lead agency on the project is the Department of Innovation and Technology (DoIT). Civic organization Smart Chicago Collaborative, another key player, has been managing community engagement and public outreach.

Where are the nodes in Chicago?

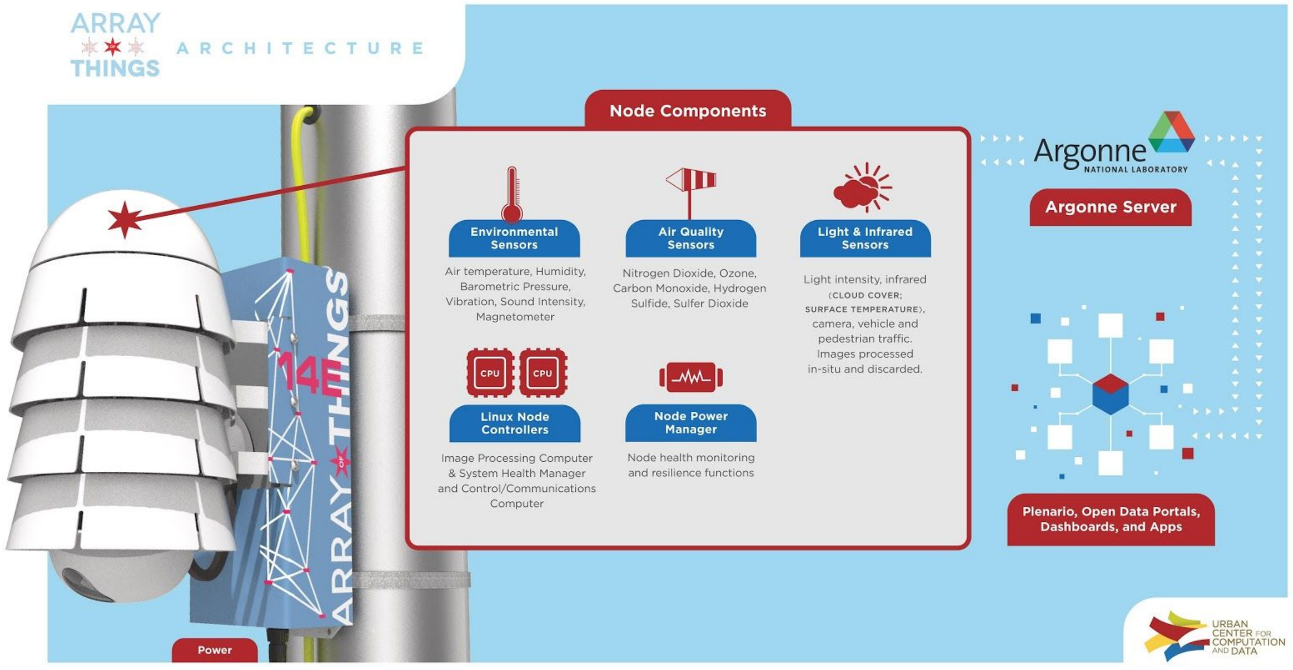
The following diagram shows the locations of nodes currently deployed in Chicago, as well as the next hundred planned for deployment in 2019.



System Design

Nodes

In order to understand the design of the API, it is important to have a basic understanding of the hardware embedded in the nodes.


ARRAY OF THINGS
ARCHITECTURE
Node Components
Environmental Sensors
Air temperature, Humidity, Barometric Pressure, Vibration, Sound Intensity, Magnetometer
Air Quality Sensors
Nitrogen Dioxide, Ozone, Carbon Monoxide, Hydrogen Sulfide, Sulfer Dioxide
Light & Infrared Sensors
Light intensity, infrared (CLOUD COVER; SURFACE TEMPERATURE), camera, vehicle and pedestrian traffic. Images processed in-situ and discarded.
CPU
CPU
Linux Node Controllers
Image Processing Computer & System Health Manager and Control/Communications Computer
Node Power Manager
Node health monitoring and resilience functions
Argonne NATIONAL LABORATORY
Argonne Server
Plenario, Open Data Portals, Dashboards, and Apps
Power
14E
URBAN CENTER FOR COMPUTATION AND DATA

The node is the finished unit: it's the big, white, plastic, pine cone looking thing you see on the light poles. Inside the node there are multiple pieces of hardware, but for our concerns we will focus on the sensors.

Sensors are the physical components that collect data. They have a hierarchical naming system: the *subsystem* is the highest level of the naming scheme and it roughly organizes the observations by a theme (like meteorological data); the next level is named (somewhat confusingly) *sensor* and that is the physical device on the boards that records that data; and finally there is the *parameter* value which helps define the measurement type.

For example, we have a sensor board named *metsense* that has an array of sensors that record meteorological data. On the metsense boards, there is a sensor named *BMP180* and it’s a multi-use sensor -- it records more than one type of data. Its parameters are *temperature* and *humidity*.

Another board on some of the nodes is the *chemsense* board, and it contains sensors that record air quality data related to gases. Each of its sensors are named for the gases they observe and their parameter values are *concentration*.

As you can see, we need a flexible naming system for sensors. When taken as a full path they make more sense: *metsense.bmp180.temperature*, *metsense.bmp180.humidity*, *chemsense.co.concentration*, *chemsense.o3.concentration*, etc.

API Schema

Array of Things is deployed across several geographic areas. These deployment regions are referred to as Projects. For example, our largest deployment is in the city of Chicago so we have a project named Chicago.
The physical devices being deployed are Nodes. Networks are comprised of Nodes. Nodes are identified by their VSN (a unique serial number we assign to each).

Onboard the Nodes are Sensors. Sensors observe various facets of the environment, such as temperature, humidity, light intensity, particulate matter, and other topics. It is important to note that there can be, and in some cases almost always is, redundancy in the types of information sensors record. It is important to read the sensor data sheets provided by the Waggle team to determine differences.

The information recorded by Sensors are Observations. Observations are snapshots of measurements at a given time on a given Node. It is important to note that some Sensors are still not fully tuned or are in early stages of experimentation, and by virtue of that their Observations may not have fully accurate data. To help disambiguate these situations, the API also provides Raw Observations with both the raw value of the observation and the clean or human readable value.

So, you can think of the hierarchy of system entities in the following sentence:

Observations are *recorded by* **Sensors** that are *on board* **Nodes** that *organized within* **Projects**.

Project Schema

Attribute	Type	Details	Description
name	Text	Unique	The name of the project
slug	Text	Unique, Primary Key	The downcased, URI compliant version of the name: e.g. Chicago becomes chicago; New York becomes new-york
first_observation	Timestamp		The first recorded observation within the project -- essentially its birthday
latest_observation	Timestamp		The latest recorded observation within the project
bbox	Geometry		A polygon that gives the south-western-most and north-eastern-most coordinates of nodes within the project
hull	Geometry		A polygon that gives the exact convex hull (outline) of the nodes within the project

Node Schema

Attribute	Type	Details	Description
vsn	Text	Unique, Primary Key	The unique identifier of the node
location	Geometry		The geographic point of the nodes (longitude, latitude)
description	Text		General information about the node
human_address	Text		A street address for the node -- something recognizable by humans
commissioned_on	Timestamp		The burn-in date of the node's firmware -- essentially its birthday
decommissioned_on	Timestamp		The timestamp the node was taken offline

Sensor Schema

Attribute	Type	Details	Description
path	Text	Unique, Primary Key	The unique identifier of the sensor -- the dotted concatenation of the subsystem, sensor and path
subsystem	Text		The physical or logical board of sensors on which this sensor is placed
sensor	Text		The name of the specific sensor
parameter	Text		What the sensor is measuring; e.g. temperature, humidity, gas concentration, etc.
ontology	Text		A hierarchical category for the sensor
uom	Text		The unit of measurement; e.g. C for celcius
min	Float		The minimum good value for an observation made by this sensor
max	Float		The maximum good value for an observation made by this sensor
data_sheet	Text		A link to a detailed spec sheet for the sensor

Observation Schema

Attribute	Type	Details	Description
node_vsn	Text	Foreign Key	The vsn of the node that made the observation
sensor_path	Text	Foreign Key	The path of the sensor that made the observation
timestamp	Timestamp		The timestamp of when the observation was recorded
value	Float		The parsed value of the observation -- essentially a heuristically verified good value

Raw Observation Schema

Attribute	Type	Details	Description
node_vsn	Text	Foreign Key	The vsn of the node that made the observation
sensor_path	Text	Foreign Key	The path of the sensor that made the observation
timestamp	Timestamp		The timestamp of when the observation was recorded
hrf	Float		The parsed value of the observation -- essentially a heuristically verified good value
raw	Float		The raw value of the observation -- essentially an unverified analog reading from a sensor

Many to Many Relationships

There are obvious one to many relationships between nodes and raw/observations, and sensors and raw/observations. But there are also many to many relationships between projects, nodes and sensors:

- Projects and Nodes share a relationship (it's how we calculate the bbox and hull for the project)
- Projects and Sensors share a relationship (sensors are onboard nodes and can be linked back)
- Nodes and Sensors share a relationship (via observations)

The obvious nature of the M2Ms between nodes and sensors and projects and sensors is intuitive. However, we cannot guarantee that a node cannot be included in multiple projects. Let's say for example that we had a special deployment of nodes to observe some small region within the city of Chicago (where we already have an established project). We may decide that we want a logical separation for this cluster of nodes in its own project, but by virtue of being in the same region as the Chicago project it would make sense to include those there. Hence the many to many relationship.

HTTP API

The API is accessible at https://api.arrayofthings.org/api , and the most up to date API documentation can be found at https://arrayofthings.docs.apiary.io/ .

If you're interested in viewing the source, and you either write Elixir or are interested in learning a new and awesome programming language, the entire application is open source and available on GitHub: https://github.com/UrbanCCD-UChicago/api_of_things .

The following sections will give you a super-duper high level overview of the API and its capabilities. Please read the online documentation for more details.

Listing available projects

Array of Things nodes have been deployed in many locations. These deployments are categorized into *projects*. To get a list of available projects:

```
GET https://api.arrayofthings.org/api/projects
```

This will return a JSON document whose *data* property will be a list of project metadata.

You can also format the response data as GeoJSON (for easy drop in uses for mapping applications) by adding the *format=geojson* parameter to the request:

```
GET https://api.arrayofthings.org/api/projects?format=geojson
```

Listing the nodes for a given project

Node information can be found at `/api/nodes`, but that will be for *all* nodes. Usually when exploring node metadata you will only want to see a subset of nodes.

To filter nodes, use the *slug* value of the project you want to explore. For example, to get the nodes in the Chicago project:

```
GET
https://api.arrayofthings.org/api/nodes?within_project=chicago
```

Just as with the projects endpoint, you can get the response data as GeoJSON by appending the format parameter:

```
GET
https://api.arrayofthings.org/api/nodes?within_project=chicago&f
ormat=geojson
```

Paging through data

As a brief interlude before we get to observation data, I want to go over how pagination works in this API.

By default, response data is chunked into pages of 200 elements. The absolute bounds for page size are 1 and 5,000. The size is controlled via the `size` parameter.

The page number is controlled via the `page` parameter. For initial queries, it defaults to 1. If you request a page that extends beyond the total number of resources available, you will receive and empty data array in the response.

There is an additional `meta` property of the response that contains metadata for the response itself. The meta object contains paging links for easier navigation.

Getting the observation data for a specific node

Typically, you'll be interested in getting the observations for either a specific node or a collection of nodes. To do this, you can use the `from_node` and `from_nodes` parameters respectively. The values to these parameters are the *vsn* attributes of the nodes.

For a single node (with VSN *02A* for example), you'd write your request as:

```
GET https://api.arrayofthings.org/api/observations?for_node=02A
```

If you wanted observations for, as an example, all the nodes on Milwaukee Ave in Chicago, and had a list of their VSNs, your query would be:

```
GET
https://api.arrayofthings.org/api/observations?for_nodes[]=001&f
or_nodes[]=002&...
```

The responses to observations queries are ordered in descending order by timestamp, by default.

There are tons of optional parameters and transformations that can be applied to querying observation data. Please review the online documentation for for details.

Client Libraries

In order to facilitate user adoption, I’ve created several client libraries. The Python and Javascript clients are the most popular and are the best tested and written. Full disclosure: I am **not** an R developer, and after having stumbled my way through writing this library I don’t want to be one :) Y’all are crazy.

Python Client

https://github.com/UrbanCCD-UChicago/aot-client-py

The Python library assumes you’re using Python 3, and preferably 3.6 or better. I will not support Python 2, and frankly I’ll mail you a box of bees if you open a ticket with issues related to Python 2. The future arrived 10 years ago...

Javascript Client

https://github.com/UrbanCCD-UChicago/aot-client-js

There are no special restrictions or warnings about this library.

R Client

https://github.com/UrbanCCD-UChicago/aot-client-r

Note that even Tom Schenk gave up on trying to help me here. If you can fix this, please do.