

Auroral Documentation

None

None

Copyright © 2023 bAvenir

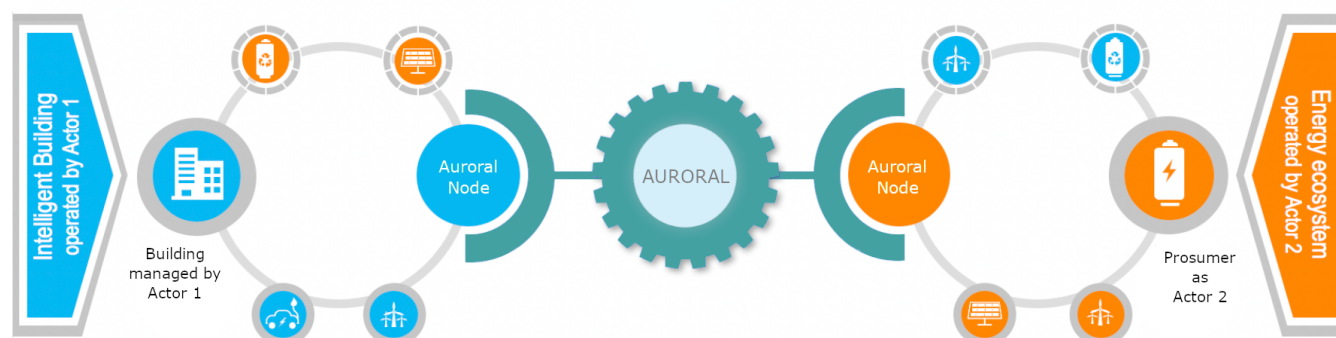
Table of contents

1. Home	3
1.1 What is AURORAL?	3
1.2 Why using AURORAL	5
1.3 Architecture	6
1.4 Limitations	12
2. Getting Started	13
2.1 Setup the organisation	13
2.2 Install a Node	19
2.3 Start using the Auroral	27
3. Key Features	50
3.1 Organisations	50
3.2 Users	0
3.3 Nodes	0
3.4 Items	0
3.5 Governance	0
4. Adapters	0
4.1 Adapters in AURORAL	0
4.2 Dummy Adapter	0
4.3 Node-red Adapter	0
4.4 Helio adapter	0
4.5 Custom adapter	0
5. Applications (New!)	0
5.1 AgentUI	0
5.2 TD Editor	0
6. Developers	0
6.1 Advanced node usage	0
6.2 Semantic interoperability	0
6.3 AURORAL Open API	0
7. Downloads	0

1. Home

1.1 What is AURORAL?

AURORAL is a **data interoperability ecosystem** that allows the **integration of heterogenous infrastructures** owned by **different stakeholders** that eventually can **interact and exchange data** among them.



- **Data interoperability ecosystem**, by using same data models and standards to interact among AURORAL participants
- **Integration of heterogenous infrastructures**, by supporting AURORAL users to integrate different types of technologies with adapters, and also help generating **RDF** mappings for their data
- **Different stakeholders**, by having a constellation of nodes owned by different users that can interact and create relationships
- **Interact and exchange data**, by enabling the transparent discovery and exchange of data using common APIs and data models, among users that agreed to do so with partnerships or data sharing agreements

What are the working principles?

AURORAL platform inherits parts of the architecture and components from the VICINITY H2020 project platform. VICINITY platform was oriented to work with IoT devices and based the control and exchange of data on a **XMPP** network. AURORAL platform plans to go beyond that, giving support to other data transfer protocols and providing functionality to other types of data or new paradigms as the data spaces.

AURORAL is composed by two main parts:

- The control plane or AURORAL cloud, which is managed through the **Neighbourhood Manager** website and allows management and configuration of the **XMPP** network.
- The distributed nodes, which provide functionality to integrate data and interact with other nodes in the platform.

Some other benefits derived from using **XMPP** technology are that AURORAL gives support to infrastructures that do not necessarily have a public DNS, and also, **XMPP** is a good technology for federation, thus opening the door to have different AURORAL based platforms able to interact among them.

AURORAL security relies on PKI and JWT for controlling identities and allowing interactions, while also uses channel encryption and benefits from built-in security features of the **XMPP** server. But one of the biggest privacy and security features is that the user data is actually living in the user infrastructure (behind the AURORAL node). Therefore, AURORAL is not holding any data, not even context data, limiting potential data loses.

In regards to the platform governance, AURORAL has a democratic system where all organisations are at the same level and have absolute control of their assets. There are several ways to share **metadata** and/or data that will be described in future sections. However, AURORAL is learning from the interaction with real users and extending the options to collaborate based on the feedback received.

In order to be up-to-data in a fast changing environment, AURORAL aims to grow in the and enhance its interoperability capabilities so options like extending support to integrate other ontologies or IoT platforms, or extension of the IDM and

authentication capabilities, where DID and Verifiable Credentials are on the table. A complete roadmap with new features plan should be made available shortly.

So what can I do with it?

- It is possible to join the collaborative neighbourhood and configure identities and relationships with other organisations there.
- [Developers Neighbourhood Manager](#)
- [Production Neighbourhood Manager](#)
- Then you can join your infrastructure or service to AURORAL using the Node
- [Node installer](#)
- There are adapters that can be installed with the Node to integrate various technologies or data sources, or you can develop your own.
- Node-RED adapter
- Semantic adapter - Helio
- Custom adapters
- Once you are integrated in the ecosystem, you can start joining communities or establishing partnerships in the neighbourhood, an eventually create agreements to share data.
- Now you are ready to use the AURORAL Open API locally with your Node, to discover and access data from your infrastructure or remote nodes.
- [Online API specification](#)

1.2 Why using AURORAL

Semantic interoperability

Creation of the AURORAL ontology reusing other well known ontologies as SAREF, OWL, FOAF..

Description of AURORAL Things interfaces and metadata using W3C WoT Thing Descriptions

Exchanging data using RDF in JSON-LD format

Decentralized

The nodes are the distributed component of AURORAL

Nodes are installed by the user and act as gateway with AURORAL

The nodes provide all the functionality to interact with other nodes, but also to work with your local data

AURORAL nodes have a series of capabilities that allow them to independently work with RDF data and to use SPARQL

Governance

In AURORAL we believe in democratic governance, so all organisations registered are at the same level

Different levels of granularity both for allowing discovery of metadata and access to actual data

Possibility to set up different user roles within an organisation

Security and privacy by design

The user data is never stored in the platform and remains managed by the owner

The context data is also stored at the nodes, and it is only visible to other nodes if collaboration has been agreed

All data exchanged is signed to avoid tampering and the transactions happen over encrypted channels

Opensource

Code available in GitHub

Possibility to have AURORAL adapters or plugins created by the community

Reuse of other opensource projects like [Openfire](#) or [Node-RED](#)

Following standards

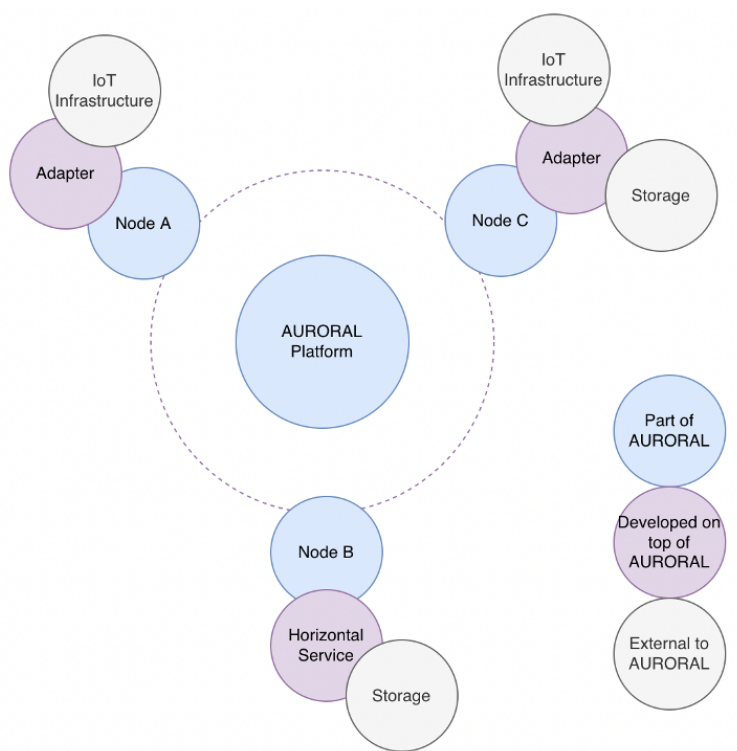
Reusing and extending W3C semantic web standards

Building communication channels using battle proofed protocols like HTTP or XMPP

1.3 Architecture

1.3.1 Ecosystem

AURORAL ecosystem can be described in one sentence as a network of nodes that are coordinated by a control plane.



THE DECENTRALIZED NETWORK

The AURORAL network is composed by nodes that can interact among them. Each node belongs to an organisation and has two main purposes:

- Data integration and standarization
- Gateway into AURORAL (Discovery and data Sharing)

In the node detail section below we will dig in the inner side of the node.

THE CONTROL PLANE

AURORAL platform uses a XMPP server and a web interface to configure the control plane. By creating relationships and data sharing agreements, the users of the platform are configuring the communication channels and the ACL rules that afterwards will be enforced by the nodes.

The web interface that allows to:

- Configure visibility rules of the metadata (Who can discover me?)
- Configure accessibility rules to the data (Who can access my data?)

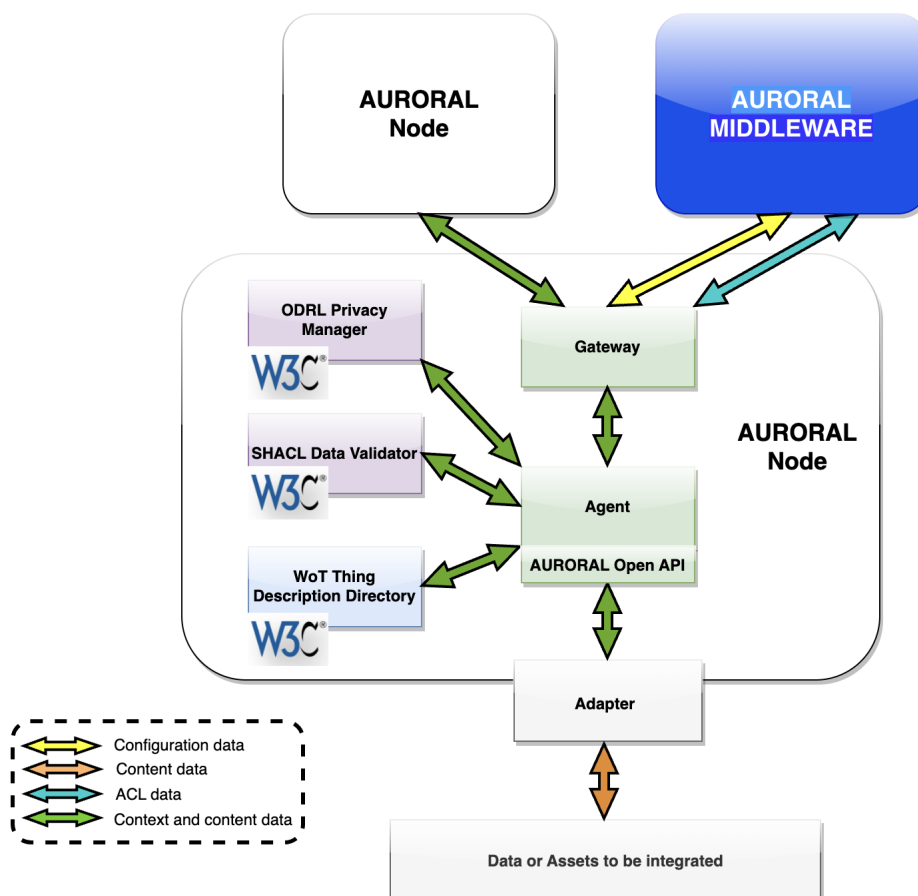
The web interface, otherwise known as Neighbourhood Manager, is used also for creating the virtual entities of the nodes, which among other things, will hold the node credentials. In general, the Neighbourhood manager can be seen as the AURORAL 'social network', users can configure their accounts, see their assets and find other users with which they might want to collaborate.

THE AURORAL NODE IN DETAIL

The node is a client application that can be installed in most computers and servers (supporting AMD64, ARM64 and ARMv7 architectures). [Download the Node here!](#)

As we can see in the picture, an AURORAL Node can communicate with the data assets or real world, by means of using an adapter, on the other hand, the Node is the gateway to communicate with other AURORAL Nodes and access configuration and authentication services from the Middleware.

Now, we will describe each part of the Node and their utility:



AGENT

The AURORAL Agent coordinates the various tasks that the Node can do. Its main functionalities are:

- Coordination of the AURORAL Node modules
- Expose API to interact with the communication and semantic interoperability modules. Thus, acting as an abstraction layer of the rest of the Node components. The agent API will be known as the AURORAL OpenAPI.
- Coordinate registration process between semantic modules and gateway.
- Store credentials and other information relevant to the registered objects.
- Process, validate against ACL rules, and redirect data requests to the adapter.

GATEWAY

The AURORAL Gateway shall provide the following functionality:

- Act as communications module of the AURORAL Node
- Relay consumption requests to AURORAL Node Agent
- Expose API to allow resource consumption of remote items
- Expose API for registration and discovery
- Access Neighbourhood Manager to obtain updates and configuration rules
- Access the ACL rules over DLT Network through the interface with the DLT Client, or alternatively, it is possible to choose using XMPP network.

WOT DIRECTORY

The WoT Thing Description Directory allows to discover Thing Descriptions (TD) from one AURORAL Node to another by point-to-point and discovery a set of Nodes that are discoverable to the former according to the AURORAL access control and privacy policies by point-to-cloud. The WoT Thing Description Directory provides the following functionalities and services:

- Register TD objects in the directory.
- Retrieve TD objects from the directory.
- Update TD, modifying totally or partially and existing TD.
- Remove a TD object from the directory.
- Provides a publication/subscription mechanism allowing to follow the creation, updates and delete operations done over the TD.
- Retrieve all the TD objects stored in a directory or a subset of TD.
- Validate TD objects before registering.
- Provide mechanism that enrich the validation already afforded.

To persist the triples the default storage is Jena-Fuseki.

ODRL SERVICE

The ODRL Privacy Manager allows applying privacy policies over the TD registered in the WoT Thing Description Directory and check if the request is allowed to discover or access the resources associated to a privacy policy. Its main functionalities are:

- Applying context privacy policies.
- Applying content privacy policies.
- Applying context compliance checking.
- Applying content compliance checking.

This privacy policies allows to, for example, discover resources only if the requester is inside a certain location or discover resources only in certain moments of the day.

SHACL VALIDATOR

The SHACL Data Validator apply context and content data validation over the AURORAL Node. For that, the main functionalities are:

- Ensures that all data across the Node is always aligned to the AURORAL ontology.
- Apply some restriction over the data.
- Perform CRUD operations over SHACL shapes.
- Validate data express in RDF or another format stored in the Node.

ADAPTER

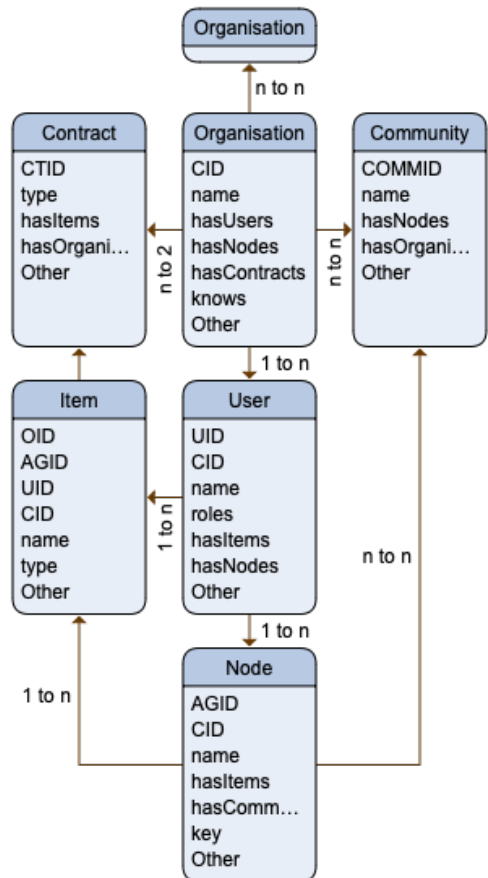
The adapter allows to perform operations or enrich the data stored in the AURORAL Node from the connected data sources. For that it should:

- Apply context enrichment to the TD stored in the AURORAL Node.
- Perform operations across the AURORAL Node.
- Apply a semantic upgrade to the TD stored in the AURORAL Node.
- Translate heterogeneous formats (i.e., JSON or CSV) to W3C standards (i.e., RDF or JSON-LD).

It is possible to run a Node without adapter, for this end, a service or application that wishes to use AURORAL can communicate directly with the AURORAL OpenAPI that is exposed by the Agent component.

1.3.2 Conceptual model

In AURORAL the different participating entities are organized as in the schema below:



HIERARCHY

Everything starts with a user registering an organization, that then can invite other members to it.

Organisations in AURORAL have users and nodes

- The users interact over the web
- The nodes integrate items

The items in AURORAL represent any connected data source

- An item can be private, 'shareable' with friends, or public
- And item can be discovered (metadata) or accessed (data, i.e., sensor measurements)

Organisations can create relationships:

- Contracts, data sharing
- Communities, discovery

1.3.3 Collaboration

This section covers the basic modes of collaboration between different organisations in AURORAL.

The first two collaboration options create the possibility to access metadata or context information from remote nodes of other organisations.

[Read more detailed info about the collaboration in AURORAL here](#)

COMMUNITIES

These are open to everybody groups, that can be used to share context data about same domains or topics of interest. These groups are free to join, but the discovery queries will still be restricted based on the visibility levels of the items under each node. As an example, if organisation A queries all nodes in a community, the result will be all the context info of the public items and of the 'for friend' items that belong to partners of organisation A.

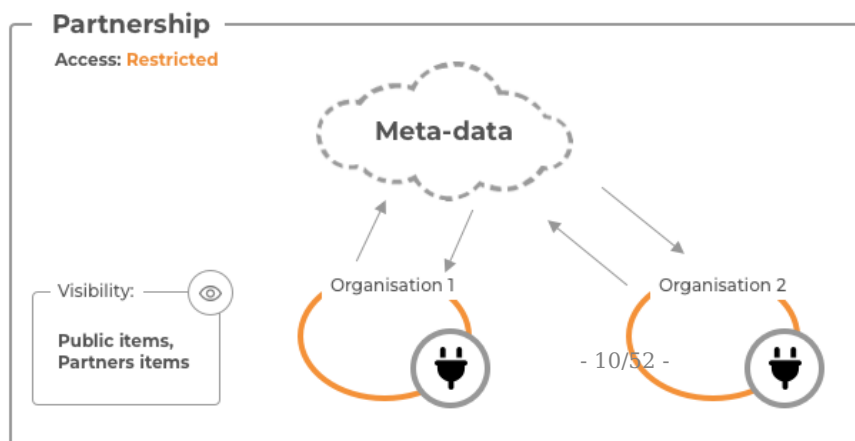
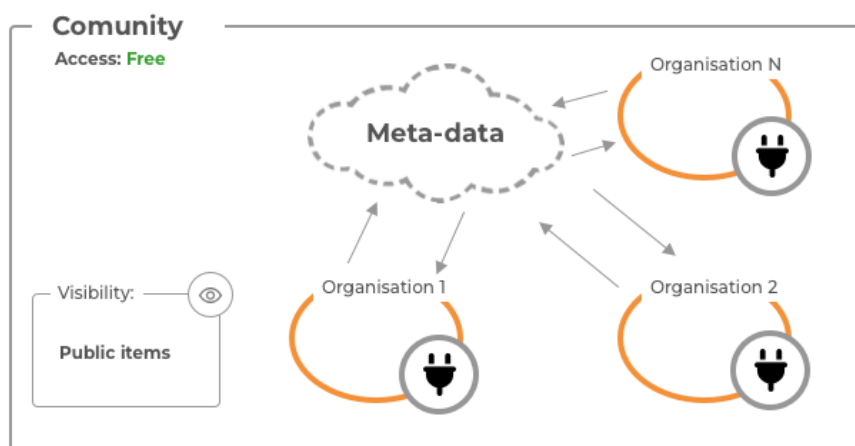
PARTNERSHIPS

These are 1 to 1 deals, two organisations agree to share context data of all their nodes added to this group. Again visibility rules apply, so private items metadata will not be accessible by the partner.

The last collaboration option allows access to data, and has as a prerequisite that both organisations are partners.

CONTRACTS OR DATA SHARING AGREEMENTS

After a contract is agreed, both organisations can add items to it. These items data will be accessible from the partner nodes & items that are included in the deal.



1.3.4 Interoperability

TBD

1.4 Limitations

This section reflects use cases that were not in scope during development and are not recommended applications for AURORAL.

Under development

The project is still in evaluation and testing phase so the list might be extended.

- AURORAL is not optimised for media streaming or transfer of large files, however it is possible to use AURORAL to describe those data sources and make their metadata discoverable.

2. Getting Started

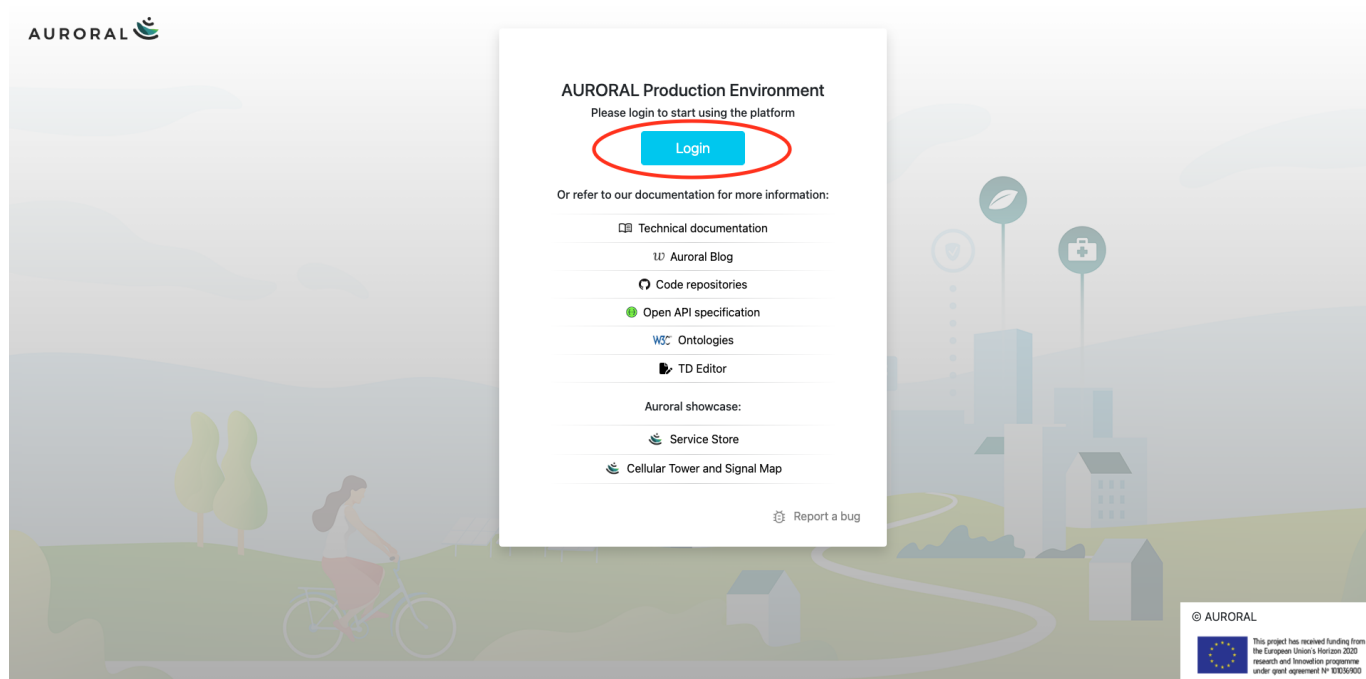
2.1 Setup the organisation

Before you start using a platform, the first thing to do is sign up for an organization account. This means creating a profile, choosing an `administrator` for your organization, and providing information like your organization's name and contact details. This process helps set up your company on the platform and gives you access to all the cool features.

By registering an organization account, you are taking the first step towards utilizing the `platform` to its full potential and benefiting from the resources it provides.

2.1.1 Create the organisation

To get started, head to the [Auroral Neighborhood Manager](#) entrypoint. To set up a new organization, you'll need to visit an authorization page. Just click the `Login` button to continue.



Here you can login using your `Email` and `Password` to access an existing organisation or create brand new organisation. To do so, please click on `Register new company` link located on the bottom of the form.



About project

Log in to start your session

 Show password Remember Me

Log In

Send magic link

[Register new company](#)

[Forgot my password](#)

review process takes time

All new companies have to be reviewed by our team. This is done manually so it may take some time to finish. We want to kindly ask you to please be patient with us until review process is done. If you know some company that is using Auroral platform and is already authorized, you can skip the next steps and ask them to send you an invitation. For more information please refer to [Organisation invitation](#) section.

Next you will have to fill out basic information about your organisation and your company administrator. For the administrator you will have to provide an `Email` and create a `Password` which has to meet a security criteria.



About project

Registration form

Company

Name*

Location*

Administrator

Email*

Password*

Repeat password*

✓ Passwords match and are complex

Name*

Surname*

Occupation*

[Back to log in](#)

[Terms and Conditions](#)

Administrator

Every organisation has to have at least one administrator account to manage users, roles and settings. When you create a new company, this account is created for you automatically. For more information about the user roles please refer to [Users](#) section.

How to use existing Auroral User as the administrator?

If you want to use existing Auroral User as administrator for the new company, this user will have to receive an invitation from the existing organisation. For more information please refer to [Organisation invitation](#) section.

After you submit registration request, your company will be manually reviewed by our team. During this process you may be contacted by our employees requesting further information about your company. After the review process has been `successful` you will receive a confirmation email with a link. To finish autoauthorization please click on link in the email.

After your organisation account has been setup you can access it by logging in [here](#) using `administrator` account credentials you

have provided in previous step. When you successfully log in you should see following screen:

The screenshot shows the AURORAL user interface. The top header is green with the text 'AURORAL' on the left and a user profile 'John Doe' on the right. A dark sidebar on the left contains the 'ABC org' logo and a menu with the following items: Items, Marketplaces, Contracts, Organisations, and Access Points. The main content area is titled 'My Items' and features a filter section with three dropdown menus: 'Visibility and access rights' (set to 'My items'), 'Item type' (set to 'All'), and 'Domain'. Below the filter, it displays 'No items found for the current selection...'. At the bottom, there is a European Union funding notice, a copyright notice for bAvenir, s.r.o., and a version indicator 'v0.1' with a help icon.

2.1.2 Configure the organisation

There are several things that can be configured for your organisation. Some are just cosmetic changes like adding company logo, changing location and so on. And some are more advanced. For example changing `User roles`.

All of this settings can be changed by clicking on your company avatar in top left corner of website:

AURORAL

John Doe

My Items

Filter

Visibility and access rights: My items

Item type: All

Domain:

No items found for the current selection...

© 2015-2019 bAvenir, s.r.o. All rights reserved.

v0.1

When a new organisation is registered to the Auroral platform, default roles given to a first user are: `administrator`, `infrastructure operator`, `system integrator`. In order to proceed with setting up a connection to the Platform all these roles need to be present. To confirm this, please open `Role management` section. You should see the following roles assigned to your user:

AURORAL

John Doe

Company profile

ABC org

Partners: 0

User accounts: 1

About Me

Organisation: ABC org

Location: Berlin

Notes

Theme color: Change color to:

Items Partners History User accounts Invitations **Role management**

John Doe b6219d640d8d440ea131@mailspn

user administrator infrastructure operator system integrator

Change roles Delete User

User roles

All interactions with the platform such as: `manage users`, `add access points`, `remove services` and so on, are determined by the `roles` which have been assigned to a user by the `administrator`. For more information about the user roles please refer to [Users](#) section

When all necessary roles are present, the next step would be to install Auroral Node in your infrastructure. Node acts as a gateway to Auroral ecosystem and is crucial for exchanging data and metadata between other organisations in the Platform or even in your own organisation.

[Proceed to install Auroral Node](#) →



Secure your Node

The Node can run privately, but if you expose its APIs online, secure them with SSL and Basic Authentication. See [Setup SSL and Basic Auth](#) for more details.

2.2 Install a Node

The Auroral platform uses a client software known as the `Auroral Node` to integrate IoT Infrastructures. It serves as a gateway for sending and receiving data/metadata in your infrastructure.

2.2.1 Software requirements

-  Docker with docker-compose
[Download](#)
-  WSL2 (Windows users)
[Ubuntu WSL download](#) or
[Debian WSL download](#)

2.2.2 Install using CLI

If you only have CLI access to host machine you can install Node by running `node_cli.sh` script that comes with the Node's directory when you clone/download it from the [Auroral Node's repository](#).

Additional requirements:

-  Git (optional)
[Download](#)

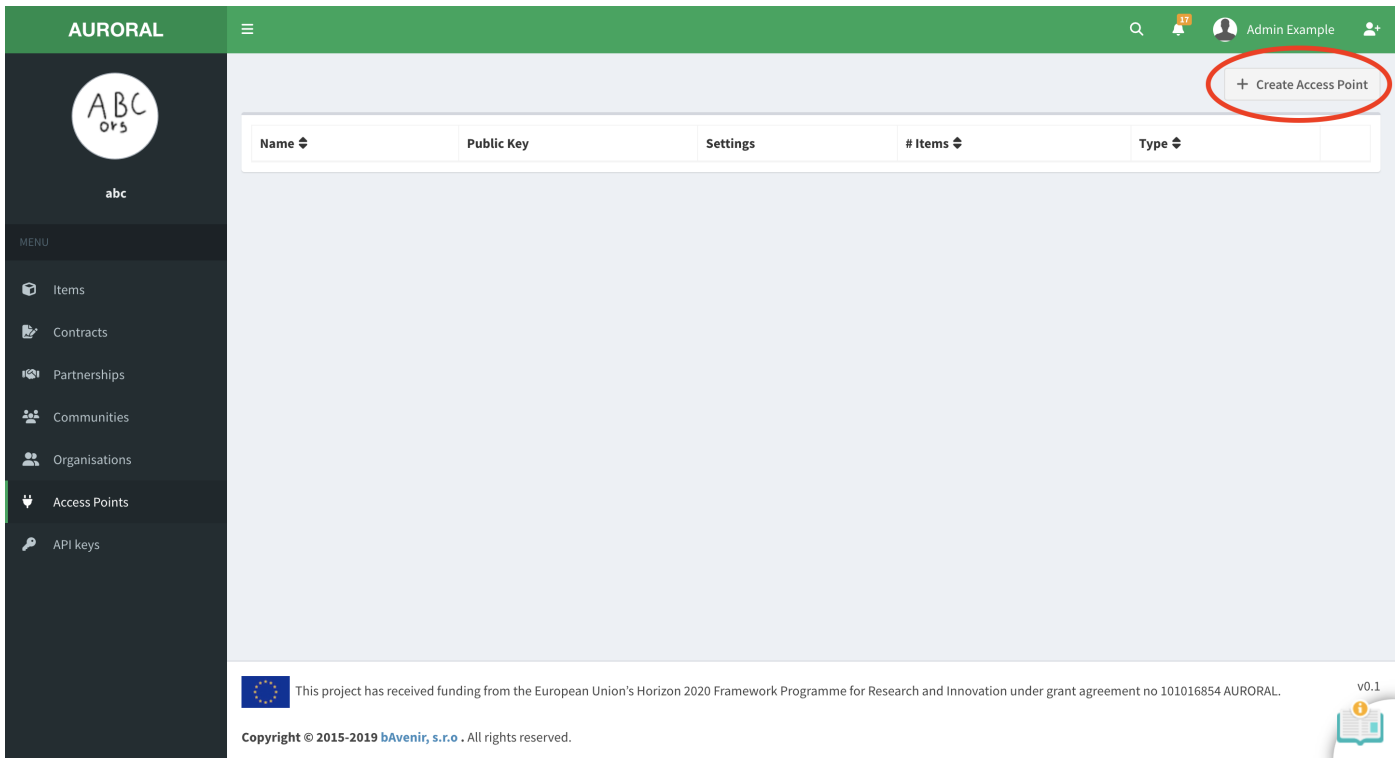
Tip: Using Git

Using Git to clone Auroral Node's repository is recommended approach. It will let you easily update Node to the latest version by simply running `git pull` to get the latest updates from the repository.

Create an Access Point

First you will need to generate the Access Point to identify your Node in the Platform and to secure your communication with the Auroral Neighbourhood Manager. To generate the Access Point please:

1. login to your account
2. open Access Point page
3. click on + Create Access Point in top right corner:



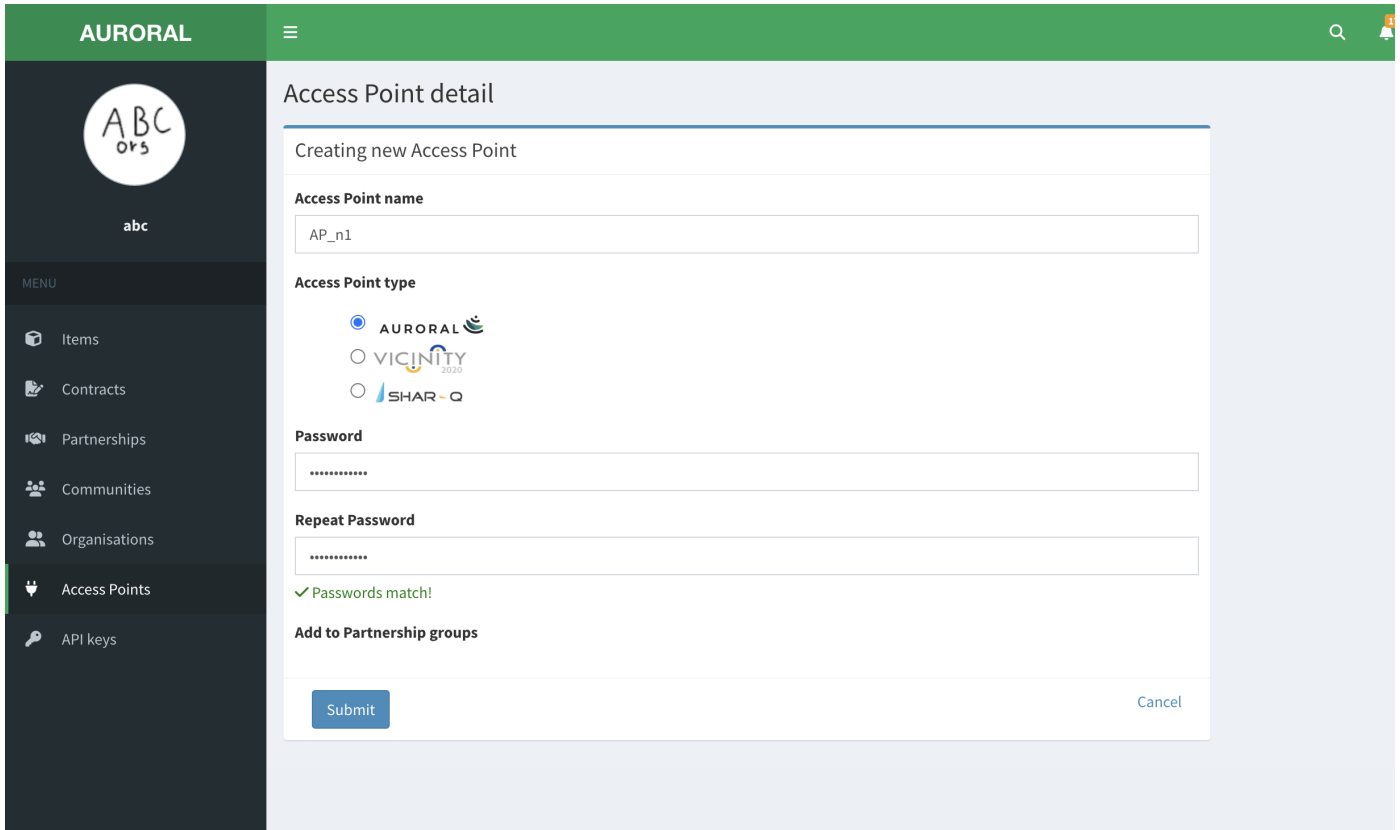
The screenshot shows the Auroral platform interface. The top navigation bar is green and contains the 'AURORAL' logo, a search icon, a notification bell with '17', a user profile for 'Admin Example', and a '+ Create Access Point' button circled in red. Below the navigation bar is a table with columns: Name, Public Key, Settings, # Items, and Type. The table is currently empty. On the left side, there is a dark sidebar menu with the 'Access Points' option highlighted. At the bottom of the page, there is a footer with a European Union logo, a copyright notice for bAvenir, s.r.o., and a version number 'v0.1'.

Don't have an account?

To generate the Access Point you will need organisation account with a System integrator role. If you do not have the account, please refer to [Setup the account](#) section.

Before you create Access Point, you will be prompted to provide some information:

1. type the `name` of your new Access Point
2. select `Auroral` as a type
3. create a `password` :



The screenshot shows the Auroral web interface. The header is green with 'AURORAL' on the left and a search icon on the right. A dark sidebar on the left contains a logo with 'ABC' and 'ors', the name 'abc', and a menu with items: Items, Contracts, Partnerships, Communities, Organisations, Access Points (highlighted), and API keys. The main content area is titled 'Access Point detail' and contains a form for 'Creating new Access Point'. The form has the following fields: 'Access Point name' with the value 'AP_n1'; 'Access Point type' with radio buttons for 'AURORAL' (selected), 'VICINITY 2020', and 'SHAR-Q'; 'Password' and 'Repeat Password' fields, both containing masked text. A green checkmark and the text 'Passwords match!' are displayed below the password fields. At the bottom of the form are 'Submit' and 'Cancel' buttons.

Tip: Remember the password!

Remember the password. It will be used in node installation procedure later on.

After the Access Point has been created, please save the `AGID`, a unique identifier, some place safe. You can leave the browser open for now. You will have to add a `public_key` to your new Access Point after installing the Node in your infrastructure later on:

The screenshot displays the Auroral web application interface. The top navigation bar is green and contains the 'AURORAL' logo, a search icon, a notification bell with '17', a user profile for 'Admin Example', and a '+ Create Access Point' button. The left sidebar is dark grey and lists navigation options: 'Items', 'Contracts', 'Partnerships', 'Communities', 'Organisations', 'Access Points', and 'API keys'. The main content area features a table with the following data:

Name	Public Key	Settings	# Items	Type
AP_n1 agid: ce62b701-5dd3-4f98-ba5c-82fdda5ee36c	Public key missing	Visible: Yes Auto-enable: Off Device: Off Service: Off	0	AURORAL

At the bottom of the page, there is a European Union funding notice: 'This project has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under grant agreement no 101016854 AURORAL.' and a copyright notice: 'Copyright © 2015-2019 bAvenir, s.r.o. All rights reserved.' A version indicator 'v0.1' is also present.

Now that you have registered new Access Point into Auroral platform, you can use it to setup a connection between your Node and the Platform. You have everything ready to install your first Auroral Node.

Install a Node

First you will need to clone or download latest Node repository from [GitHub](#) by running:

Cloning **Downloading**

```
git clone https://github.com/AuroralH2020/auroral-node.git

curl https://github.com/AuroralH2020/auroral-node/archive/refs/heads/master.zip -L -o auroral-node.zip &&
unzip auroral-node.zip &&
mv auroral-node-master auroral-node &&
rm auroral-node.zip
```

After you have Node on your target machine, you need to `cd` into the directory to run `node_cli.sh` installation script:

```
cd ./auroral-node &&
./node_cli.sh
```

You will be asked with series of questions to help you setup the Node. If you are unclear what type of `extensions` you want to be running, you can just use the `dummy` mode for now to test the Platform. To do so please type the **following** answers to run a node with no extensions :

Question	Answers
1. Run in production mode?	1 - Yes 2 - No
2. Use default external port 81?	1 - Yes 2 - No
3. Do you want to install an extension?	1 - No, just the Node 2 - Node-red adapter 3 - Helio adapter
4. Please select adapter mode	1 - dummy 2 - proxy 3 - semantic
5. Please insert generated AGID:	Paste the AGID copied when generating Access Point
6. Please insert Node password:	Type the password created when generating Access Point

Tip: Port managment

For your first Node, you should use the default port 81. If you plan to install more, any new one should be running in a different port (i.e. 82, 83, ...)

Extensions and adapters

The extensions are Auroral developed adapters and plugins that might be added when installing the Node. Some popular extensions are the `Node-Red` and `semantic` adapters. If you choose one of the `extensions` when installing the Node, the adapter mode will be configured for you according to what is best for the selected extension. New extensions will be included with next Node releases. For more information about the adapters please refer to [Adapters](#) section

Your encryption keys are now generated and you can copy the `public_key` shown in the terminal. If everything goes well you should see following result:

```

System running on Mac
Run in PRODUCTION mode?
1) Yes
2) No
#? 2
Enable caching adapter values?
1) Yes
2) No
#? 1
1
Use default external port? (81)
1) Yes
2) No
#? 1
Do you want to install an extension?
1) No, just the Node
2) Node-red adapter
3) Helio adapter
#? 1
Please select adapter mode
1) dummy
2) proxy
3) semantic
#? 1
Now please register new Node in AURORAL website: https://auroral.dev.bavenir.eu/nm/#!/myNodes, in section 'Access points'
Please insert generated AGID:54cc1327-db51-4d0f-9dab-11ccb7e6aecf
[+] Running 4/1ode password:****
  :: Network auroral_node_default          Crea...                0.0s
  :: Volume "auroral_node_aur_triplestore" Created              0.0s
  :: Volume "auroral_node_aur_gateway"     Created                0.0s
  :: Volume "auroral_node_aur_redis"       C...                   0.0s
Generating certificates
Please copy this public key to Access Point settings in AURORAL website:
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA4eko3P05m6YYhZ5KHgil
al+TAJWMRC+Au+FuiPqRPjMgwGZKn2GSUy3mEv/xjpmBTH7veGeJB6xM1RptVph1
xdtjY1DQDIVUp5Av8PVvPzfUpqR811QxVPg1W9Qy53vDVZRYTrm3vH94ixvZD5UG
OiKlpGQvMj6xQ9atjr4E5AmnHVLgOWnbEy+E5C1iNK0gCpgxzczq1df9t7u06t24a
YGpbSe4c330zFhknrn14zPJZtgJVjC45aqn0Kys8p8HCFHw7+e7cvQ66mph4+Kj4
go3BggfJbdkK5VkJZ8XzqIcCu4vL1mSmLn0y3YkopLRHhQMlyKEVvcdy/6/AeM03U
rQIDAQAB
-----END PUBLIC KEY-----
Hit enter after done

```

How to copy the public_key?

When copying the `public_key`, you need to include

```

-----BEGIN PUBLIC KEY----- opening tag and
-----END PUBLIC KEY----- closing tag.

```

Last thing you need to do before you can start using the Node is to pass the `public_key` to your Access Point. Do do so you you have to:


1. open the Auroral NM Access Point page
2. click the `+` add public key button next to your Access Point
3. paste the `public_key` in the dialog and press `Save key` button:

The screenshot shows the Auroral web interface. On the left is a dark sidebar with the 'ABC ORS' logo and a menu. The main area displays a table of access points. One entry, 'AP_n1', has a 'Public Key' column with a red 'X' and the text 'Public key missing'. A red circle highlights a '+' icon in the 'Add public key' button. A modal dialog titled 'Public key' is open, showing a text area with the following content:

```
-----BEGIN RSA PUBLIC KEY-----
MEgCQQCo9+BpMRyQ/dL3DS2CyJxRF+j
6ctbT3/Qp8N1Qa8fezVAADkF9SusUWkQ
UblPo0w7cyziZRVSYkN1Qa8fezVAADkF9
SusUWkQUblPo0w7cyziZRVSYkN1Qa8fe
zVAADkF9SusUWkQUblPo0w7cyziZRVSY
k4+KeFhni7NT7fELiKUSnx
S30WAvQCCo2yU1orfgr41mM70MBAg
MBAAE=
-----END RSA PUBLIC KEY-----
```

At the bottom of the dialog are 'Remove key' and 'Save key' buttons. The footer of the interface includes a European Union logo, funding information, and copyright text: 'Copyright © 2015-2019 bAvenir, s.r.o. All rights reserved.' and 'v0.1'.

Start the Node

To start the Node you need to use  Docker to run Auroral Node Container. To do so run this command in Node's directory where the `docker-compose.yml` file is located:

Latest docker version Older docker version

```
docker compose up
```

```
docker-compose up
```



Docker

In order to use Docker the `docker` daemon needs to be running. If you need more information on how to start the daemon, please refer to [official Docker documentation](#).

Test the Node

In order to test if your Node is running, you can try the following command:

```
curl -s -o /dev/null -w "%{http_code}\n" http://localhost:81/api/agent/info
```

if everything is OK you should see the 200 in the response.

DONE you are now ready to start using the Auroral Node.

Secure your Node

The Node can run privately, but if you expose its APIs online, secure them with SSL and Basic Authentication. See [Setup SSL and Basic Auth](#) for more details.

[Start using the Auroral Node](#) →

2.3 Start using the Auroral

2.3.1 Start using the Auroral

With the Auroral Node running, you are now ready to start using all [features](#) Auroral Platform has to offer. In this section you will find some of a most common usecases together with the step-by-step implementation guidelines on how to achieve them.

These are the steps to be done before you can start using AURORAL platform:



Use-cases

For all the scenarios bellow, we will assume that you already have an account and know the AURORAL basics.

I have some IoT sensors or devices that produce data, how do I integrate them?

In this example we will focus on the case that there are some smart devices with the capability to produce data, and I want to connect them to the AURORAL ecosystem.

[Learn how to integrate the data](#) →

I would like to build a service based on some data I have, how do I make it available with AURORAL?

In this example we will focus on the following scenarios:

1. I have a dataset and I would like to offer access via AURORAL open API
2. I would like to build a service that returns a response based on an input value

[Learn how to start offering the service](#) →

I would like to use AURORAL data to build a service, how can I connect with relevant data providers?

In this example we will show how to discover context data about other AURORAL users, and establish collaboration links to access that data.

[Learn how to start using data from AURORAL](#) →

More use-cases

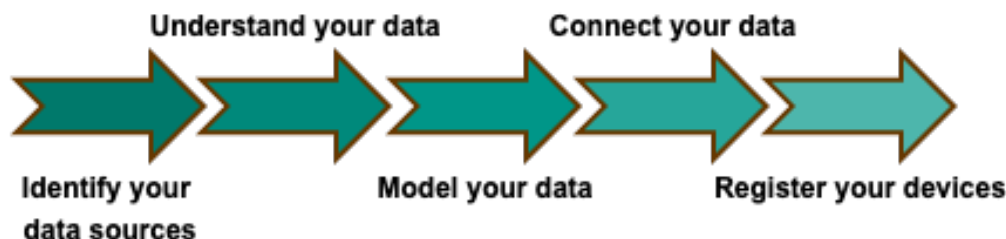
AURORAL project is still exploring new usecases and ways to satisfy the needs of its users. For that reason new usecases will be added to this section regularly.

2.3.2 Integrating data

Data provider

To depict this usecase we will imagine that we want to integrate data produced for some sensors and make it available in AURORAL. Therefore, this example assumes that you have some devices that are producing data constantly.

This are the steps to be followed:



1. **Identify your data sources:** Find which devices you would like to integrate. Then make sure that you are able to access to the data they produce.

[Learn how to identify your data source](#) →

2. **Understand your data:** Have a look at a snapshot of your data. Decide if you want to use it all or part of it and try to identify relevant information about it such as type (number/string), units of measurement, properties (temperature, humidity,...), etc.

[Learn how to understand your data](#) →

3. **Model your data:** Here AURORAL can help you. In order to make your data interoperable you need to model it in [RDF](#). **Wait, what??** AURORAL offers a series of ontologies that can be used to model and add context to various types of data. [Here](#) you can find the list of AURORAL ontologies.

[Learn how to model your data](#) →

4. **Connect your data:** Now we have the data and we know how to model it, how can I connect it to AURORAL. For that we have the concept of the adapter. There are some readily available to use, however you can also build your own. See the adapters section for more info. The general purpose of an adapter is to bridge your data and AURORAL, so it will have to (1) be able to access your data, (2) map the data to the selected ontology and (3) be accessible to the AURORAL Node.

[Learn how to connect your data](#) →

5. **Register your devices:** Finally, you need to tell AURORAL where to find your data. For this purpose you will have to register your devices using the AURORAL Node. The registration is perform with a type of document called [Thing Description](#), in it you describe your device main and info and the URIs for accessing the data that your adapter is making available.

[Learn how to register your device](#) →

Example

For demonstration purposes we will use the following example. We have a small office with a sensor that is measuring several parameters. We want to make this data available in AURORAL so that we can use it in our applications.

IDENTIFY YOUR DATA SOURCES

Our demonstrative office is equipped with a sensor from Netatmo called Smart Indoor Air Quality Monitor. This device is able to measure temperature, humidity, CO2, noise and air quality index. The device is connected to the internet and it is possible to access the data it produces through a [REST API](#) from Netatmo cloud.

Netatmo API

The Netatmo cloud has several APIs that are well described in their documentation. In this example we are retrieving data from their [Aircare API](#).

UNDERSTAND YOUR DATA

After reading Netatmo documentation and generating some tokens, we are able to retrieve the data from the device. The data is provided in [JSON](#) format and it looks like this:

```
{
  "body": {
    "devices": [
      {
        "dashboard_data": {
          "AbsolutePressure": 1016.5,
          "CO2": 760,
          "Humidity": 51,
          "Noise": 42,
          "Pressure": 1016.5,
          "Temperature": 22.5,
          "time_utc": 1600000000
        },
        "data_type": [
          "Temperature",
          "CO2",
          "Humidity",
          "Noise",
          "Pressure",
          "AbsolutePressure"
        ],
        "date_setup": 1600000000,
        "firmware": 1600000000,
        "last_status_store": 1600000000,
        "last_upgrade": 1600000000,
        "module_name": "Indoor",
        "place": {
          "altitude": 0,
          "city": "Madrid",
          "country": "ES",
          "timezone": "Europe/Madrid",
          "location": [
            30.89600807058707,
            29.94281464724796
          ]
        },
        "station_name": "Office",
        "type": "NAMain",
        "wifi_status": 0
      }
    ]
  },
  "status": "ok",
  "time_exec": 0.00000000
}
```

The amount of data is not big, but it is enough to demonstrate how to integrate it to AURORAL. We can see that the data is organized in a [JSON](#) object with several fields. For our usecase we are interested in the following fields: **temperature**, **humidity**, **co2**, **noise** and **pressure**. Here is a table with the data types and units of measurement extracted from the documentation:

Field	Type	Unit
temperature	float	°C
CO2	float	ppm
humidity	float	%
noise	float	dB
pressure	float	mbar

And here are some other information we can extract from the data when we look at the response:

- all the properties above are **sensor** data
- Netatmo device measuring the temperature is located **indoor**
- measuring is done in a intervals and we can extract the **timestemp** when the measuring occurred

MODEL YOUR DATA

Now that we know what data we have, we need to model it in RDF. For this purpose we will check the [AURORAL ontologies](#) and start looking for best match for our properties: Temperature, CO2, Humidity, Noise, Pressure.

In this case the best match is the [AURORAL Adapters ontology](#). It contains best match for all our properties.

What if one or more of my properties are missing from the ontologies?

Sometimes there is no matching ontology for some of the properties you are trying to describe. This properties then need to be **excluded** from the final response since they can **not** be described in RDF format.

Let's start modeling. First we will have to `import` the ontology. We do this using `@context` keyword which is an array. This will import all the keywords from [AURORAL Adapters ontology](#):

```
{
  "@context": [
    "https://auroralh2020.github.io/auroral-ontology-contexts/adapters/context.json"
  ]
}
```

Importing multiple ontologies

First import is `default` so you do not need to use any `unique_key` to reference the keywords from that ontology. But for any other import it is mandatory to distinguish the imports. For example if we want to import [Units of measure](#) ontology made by ©eFoodLab you do it like this:

```
{
  "@context": [
    "https://auroralh2020.github.io/auroral-ontology-contexts/adapters/context.json", // default
    {
      "om": "http://www.ontology-of-units-of-measure.org/resource/om-2/"
    }
  ]
}
```

and then when you want to use a keywords from that ontology later in your RDF you reference it using the `om:` like this:

```
{
  "@context": [...]
  ...,
  {
    ...
    "property": "AmbientTemperature", // <- keyword from default ontology
    "isMeasuredIn": "om:degree_Celsius", // <- keyword from non-default ontology
    ...
  },
  ...
}
```

Now that we have keywords for our properties we also want to have a keywords for the units of measurement: `°C`, `%`, `ppm`, `dB`, `mbar`.

Good ontology for units is [Units of measure](#) made by ©eFoodLab. This will import all keywords from that ontology:

```
{
  "@context": [
    "https://auroralh2020.github.io/auroral-ontology-contexts/adapters/context.json",
    {
      "om": "http://www.ontology-of-units-of-measure.org/resource/om-2/"
    }
  ]
}
```

Perfect. Now to wrap up our imports we still need one thing. In the Auroral platform items communicate using special identifiers which are:

- **object identifier:** `oid`
- **interaction identifier:** `iid`

We need to define those in our [RDF](#) so we can connect the device to the platform but they are not part of Auroral ontologies. We can however define them using [W3](#) ontology. This will allow us to use `label` keyword which is what we can use to define the identifiers.

To define `oid` and `iid` in our [RDF](#) we will first import the `rdfs` ontology and then add them to our `@context` :

```
{
  "@context": [
    "https://auroralh2020.github.io/auroral-ontology-contexts/adapters/context.json",
    ...,
    // import W3 ontology:
    {
      "rdfs": "http://www.w3.org/2000/01/rdf-schema#"
    },
    // define oid and iid keywords:
    {
      "oid": {
        "@id": "rdfs:label"
      },
      "iid": {
        "@id": "rdfs:label"
      }
    }
  ]
}
```

We are finally ready to start defining our `properties`. Let's start with `Temperature`. Based on the understanding of our data and the imported ontologies we can model the property like this:

```
{
  "property": "IndoorAmbientTemperature",
  "value": 22.5, // <- value provided by the sensor
  "isMeasuredIn": "om:degree_Celsius",
  "timestamp": "2023-02-28T09:18:09.610Z" // <- timestamp provided by the sensor
}
```

Now we need to put everything together to get the final result.

The `oid` will be generated later in the next step by the Node, when you register the device. As for the `iid` you will have to come up with some **unique** string. This string will be used in the next step as a `key` for your property in [Thing Description](#). For this property we will use a string `temperature` :

```
// Temperature:
{
  "@context": [
    "https://auroralh2020.github.io/auroral-ontology-contexts/adapters/context.json",
    {
      "om": "http://www.ontology-of-units-of-measure.org/resource/om-2/"
    },
    {
      "oid": {
        "@id": "rdfs:label"
      },
      "iid": {
        "@id": "rdfs:label"
      }
    }
  ],
  "@type": "Sensor",
  "oid": "", // <- generated in the next step (registration)
  "iid": "temperature", // <- chosen by you (unique!)
  "measurement": [
    {
      "property": "IndoorAmbientTemperature",
      "value": 22.5,
      "isMeasuredIn": "om:degree_Celsius",
      "timestamp": "2023-02-28T09:18:09.610Z"
    }
  ]
}
```

Now we need to model the response for the rest of the properties. Process is the same like modeling the `Temperature`. You just need to choose the right property from the ontology and add the value and the unit of measurement:

```
// CO2:
{
  "@context": [
    "https://auroralh2020.github.io/auroral-ontology-contexts/adapters/context.json",
    {
      "om": "http://www.ontology-of-units-of-measure.org/resource/om-2/"
    },
    {
      "oid": {
        "@id": "rdfs:label"
      },
      "iid": {
        "@id": "rdfs:label"
      }
    }
  ],
  "@type": "Sensor",
  "oid": "",
  "iid": "co2",
  "measurement": [
    {
      "property": "CO2Concentration",
      "value": 760,
      "isMeasuredIn": "om:ppm",
      "timestamp": "2023-02-28T09:18:09.610Z"
    }
  ]
}
```

```
// Humidity:
{
  "@context": [
    "https://auroralh2020.github.io/auroral-ontology-contexts/adapters/context.json",
    {
      "om": "http://www.ontology-of-units-of-measure.org/resource/om-2/"
    },
    {
      "oid": {
        "@id": "rdfs:label"
      },
      "iid": {
        "@id": "rdfs:label"
      }
    }
  ],
  "@type": "Sensor",
  "oid": "",
  "iid": "humidity",
  "measurement": [
    {
      "property": "RelativeHumidity",
      "value": 51,
      "isMeasuredIn": "om:percentage",
      "timestamp": "2023-02-28T09:18:09.610Z"
    }
  ]
}
```

```
// Noise:
{
  "@context": [
    "https://auroralh2020.github.io/auroral-ontology-contexts/adapters/context.json",
    {
      "om": "http://www.ontology-of-units-of-measure.org/resource/om-2/"
    },
    {
      "oid": {
        "@id": "rdfs:label"
      },
      "iid": {
        "@id": "rdfs:label"
      }
    }
  ],
  "@type": "Sensor",
  "oid": "",
  "iid": "noise",
  "measurement": [
    {
      "property": "Noise",
      "value": 42,
      "isMeasuredIn": "http://www.ontology-of-units-of-measure.org/resource/om-2/decibel",
      "timestamp": "2023-02-28T09:18:09.610Z"
    }
  ]
}
```



```
// Pressure:
{
  "@context": [
    "https://auroralh2020.github.io/auroral-ontology-contexts/adapters/context.json",
    {
      "om": "http://www.ontology-of-units-of-measure.org/resource/om-2/"
    },
    {
      "oid": {
        "@id": "rdfs:label"
      },
      "iid": {
        "@id": "rdfs:label"
      }
    }
  ],
  "@type": "Sensor",
  "oid": "",
  "iid": "pressure",
  "measurement": [
    {
      "property": "AtmosphericPressure",
      "value": 1016.5,
      "isMeasuredIn": "om:mbar",
      "timestamp": "2023-02-28T09:18:09.610Z"
    }
  ]
}
}
```

Tip: Using node-red adapter to model your data

Auroral NODE in node-red adapter mode can help you to model your data. It will automatically generate RDF model to data requests based on item's TD. This feature supports only basic data types, but it can be a good starting point for your data modeling. This feature can be disabled in the [configuration file](#) by setting `use_mapping` to `false`.

CONNECTING YOUR DATA AND REGISTERING DEVICES

To connect your data to AURORAL you will have to make the response, modeled in the previous step, accessible and tell the Node where to `GET` it. This will ensure that every time someone wants to `access` or `discover` your data in the AURORAL, the platform knows where to look for it and understands the `context` of the data.

To do so you create a `module` that will take care of generating the response and populating the proper fields with the data. In AURORAL this modules are called the `Adapters`. There are multiple existing generic `Adapters` that can be used to automate this process. For this example we will be using one of them called the `Node-RED` adapter.

Adapters

Generic adapters like `Node-red` are good way to simplify the process of connecting and registering the item. Even though they can be used to register almost any item, the disadvantage of using the generic adapters is that they usually put an overhead `cost` for your `system resources`. To avoid this you can build your custom adapter that will handle the RDF modeling and data populating. To find out more about the `Adapters` please refer to [Adapters](#) section.

If your modeled data is part of `Auroral Adapters ontology` you can use `Node red` adapter. It can take care of modeling the data and make it **available** and **accessible** for you. When using this adapter the steps connecting data and registering devices are tight together.

Node red adapter only works with `Auroral Adapters ontology`!

To use this adapter you will have to provide the Thing Description of the item you want to register and it will handle the rest.

Thing description

Thing description is an JSON-LD description of the item that the AURORAL platform understands. For more information about the Thing description please refer to [Item - Thing description](#) section.

Below is the Thing description for our Netatmo data source. To find out [step-by-step tutorial](#) how to generate this example please visit [here](#):

```
{
  "@context": [
    "https://www.w3.org/2019/wot/td/v1",
    {
      "adp": "https://auroral.iot.linkeddata.es/def/adapters#",
      "om": "http://www.ontology-of-units-of-measure.org/resource/om-2/",
      "geo": "http://www.w3.org/2003/01/geo/wgs84_pos#"
    }
  ],
  "security": [
    "nosec_sc"
  ],
  "securityDefinitions": {
    "nosec_sc": {
      "scheme": "nosec"
    }
  },
  "geo:location": {
    "geo:lat": "30.89600807058707",
    "geo:long": "29.94281464724796"
  },
  "title": "OfficeSensor",
  "adapterId": "my-office-sensor1",
  "@type": "adp:Sensor",
  "description": "Netatmo indoor sensor",
  "properties": {
    "temperature": {
      "title": "temperature",
      "description": "Temperature in the room",
      "@type": "adp:AmbientTemperature",
      "unit": "om:degree_Celsius",
      "readOnly": true,
      "type": "number",
      "forms": [
        {
          "op": "readproperty",
          "href": "" // <-- Will be populated by the adapter
        }
      ]
    },
    "humidity": {
      "title": "humidity",
      "description": "Humidity in the room",
      "@type": "adp:RelativeHumidity",
      "unit": "om:degree_Celsius",
      "readOnly": true,
      "type": "number",
      "forms": [
        {
          "op": "readproperty",
          "href": "" // <-- Will be populated by the adapter
        }
      ]
    },
    "co2": {
      "title": "co2",
      "description": "CO2 in the room",
      "@type": "adp:CO2",
      "unit": "om:percentage",
      "readOnly": true,
      "type": "number",
      "forms": [
        {
          "op": "readproperty",
          "href": "" // <-- Will be populated by the adapter
        }
      ]
    },
    "noise": {
      "title": "noise",
      "description": "Noise in the room",
      "@type": "adp:Noise",
      "unit": "http://www.ontology-of-units-of-measure.org/resource/om-2/decibel",
      "readOnly": true,
      "type": "number",
      "forms": [
        {
          "op": "readproperty",
          "href": "" // <-- Will be populated by the adapter
        }
      ]
    },
    "pressure": {
      "title": "pressure",
      "description": "Pressure in the room",
      "@type": "adp:AtmosphericPressure",
      "unit": "om:mbar",
      "readOnly": true,
      "type": "number",
    }
  }
}
```

```

"forms": [
  {
    "op": "readproperty",
    "href": "" // <-- Will be populated by the adapter
  }
]
}
}
}

```

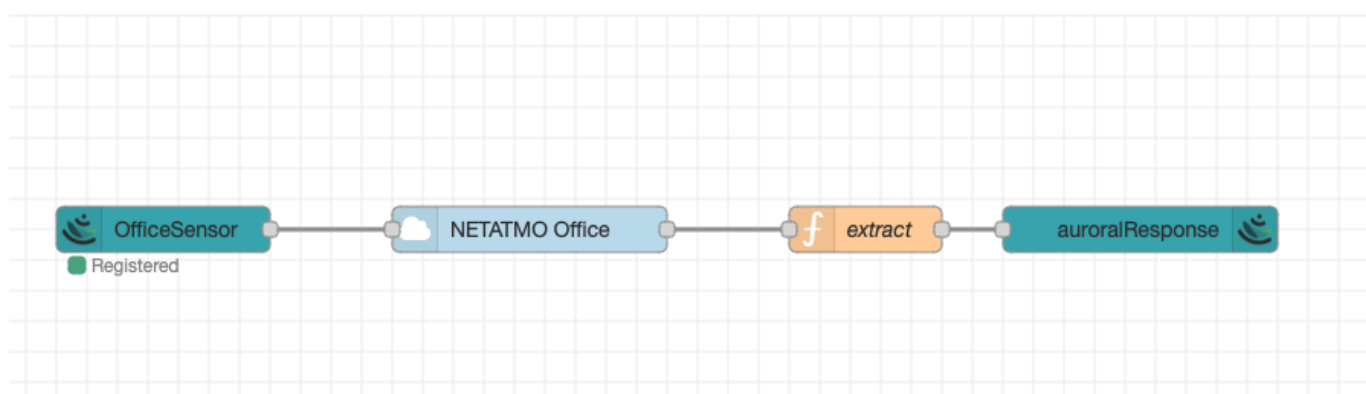
Note that besides the properties, we also added some metadata to the Thing description, such as `location` and `adapterId`. As for the `forms` array field, this is where all the endpoints (in our case only one endpoint) for accessing the data should go. Since we are using the `Node-red` adapter, we can leave the `href` field empty because the adapter will take care of populating this field.

Now when we have our data source described in a Thing description, we can register it in AURORAL. To do this we expect your Node is installed with `Node-red` extension and is running.

How to install the Node-red extension?

When you are installing new Node, you have an option to install `Node-red` extension. To find out how to do this please refer to our [Getting started - install a Node](#) section.

Our `Node-red` flow is available in [Node-red-examples repository](#), where are also described all the requirements and instructions how to run it. Below is the example how the flow for our Netatmo use-case should look like in `Node-red`:



Because this adapter is doing registration of devices automatically, we don't need to do anything else. After a while we can see that our device is registered and after enabling it can be used in AURORAL.

Do not see the device?

Maybe you forgot to enable the device. If you don't enable device, it will not be visible in AURORAL. More about enabling devices can be found [here](#).

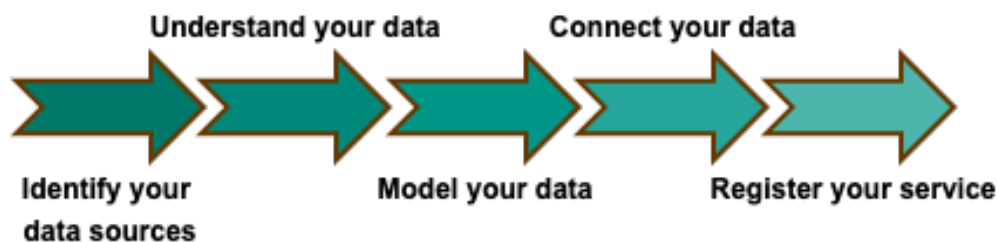
2.3.3 Offering a service

Service provider

In this scenario we want to showcase how to return data or a response to a query by using a service. There are two basic examples:

- Using a service to expose a static dataset. Using an API to access the data through AURORAL.
- Using a service that expects an input, process the input and return a response.

These are the steps to be followed:



1. **Create a service:** You need to have a service ready to provide some value to AURORAL.
2. **Understand your data:** Have a look at a snapshot of your data. Decide if you want to use it all or part of it and try to identify relevant information about it such as type (number/string), units of measurement, properties (range, longitude, latitude...), etc.
[Learn how to understand your data](#) →
3. **Model the response:** The response of your service needs to be modelled in RDF. AURORAL offers a series of ontologies that can be used to model and add context to various types of data. [Here](#) you can find the list of AURORAL ontologies.
[Learn how to model your data](#) →
4. **Connect your data:** I have a service, how can I connect it to AURORAL. In this case we do not need an adapter, though it could also be used. As services are software, your application can be updated with a module to interact with AURORAL. The functionalities of this new module are two, (1) map the data to the selected ontology and (2) make the service accessible to the AURORAL Node.
[Learn how to connect your data](#) →
5. **Register your service:** Finally, you need to tell AURORAL where to find your data. For this purpose you will have to register your service using the AURORAL Node. The registration is performed with a type of document called Thing Description, in it you describe your service main and info and the URIs for accessing the data that your adapter is making available.
[Learn how to register your service](#) →

Example

For demonstration purposes we will use the following example. We have an REST API service with an endpoint which returns closest cell towers in the vicinity of a user's location. We want to make this endpoint available in AURORAL so that we can use it in our applications.

CREATE A SERVICE

For this example we have created a microservice and we are running it on the `localhost:8000`. This service is built on top of the database which contains the data about the cell towers. Endpoint to retrieve **5 closest** cell towers is `/api/cell-towers/closest` with the following OpenAPI specification:

GET /api/cell-towers/closest Get Closest Towers

Parameters Try it out

Name	Description
lat * required number (query)	lat
lon * required number (query)	lon

Responses

Code	Description	Links
200	Successful Response	No links

Media type: application/json

Controls Accept header.

Example Value | Schema

```
[
  {
    "cellid": 0,
    "range": 0,
    "lon": 0,
    "lat": 0,
    "radio": "string",
    "operator": "string",
    "operatorcode": 0,
    "countryname": "string",
    "countrycode": "string"
  }
]
```

UNDERSTAND YOUR DATA

As we can see the endpoint is expecting a `lat` and `lon`, which in our case is a location of a user requesting the information. The data is then returned in `JSON` format and it looks like this:

```
[
  {
    "cellid": 255742873,
    "range": 1977,
    "lon": 17.1405464,
    "lat": 48.1501398,
    "radio": "UMTS",
    "operator": "02 Slovakia",
    "operatorcode": 6,
    "countryname": "Slovakia",
    "countrycode": "SK"
  },
  {
    "cellid": 77158166,
    "range": 1000,
    "lon": 17.14112,
    "lat": 48.151193,
    "radio": "LTE",
    "operator": "02 Slovakia",
    "operatorcode": 6,
    "countryname": "Slovakia",
    "countrycode": "SK"
  },
  {
    "cellid": 2608949,
    "range": 1000,
    "lon": 17.139924,
    "lat": 48.149887,
    "radio": "LTE",
    "operator": "Orange Slovensko",
    "operatorcode": 1,
    "countryname": "Slovakia",
    "countrycode": "SK"
  },
  {
    "cellid": 125492,
    "range": 1000,
    "lon": 17.140036,
    "lat": 48.149654,
    "radio": "LTE",
    "operator": "Slovak Telekom",

```

```

    "operatorcode": 2,
    "countryname": "Slovakia",
    "countrycode": "SK"
  },
  {
    "cellid": 2608947,
    "range": 1000,
    "lon": 17.142115,
    "lat": 48.15131,
    "radio": "LTE",
    "operator": "Orange Slovensko",
    "operatorcode": 1,
    "countryname": "Slovakia",
    "countrycode": "SK"
  }
]

```

The amount of data is not big, but it is enough to demonstrate how to integrate it to AURORAL. We can see that the data is organized in a [JSON](#) object with several fields. Here is a table with the data types and description:

Field	Type	Description
cellid	int	unique id of the cell tower
range	int	range of tower (meters)
lon	float	location of tower (longitude)
lat	float	location of tower (latitude)
radio	string	name of the technology
operator	string	name of the operator
operatorcode	int	code of the operator (MNC)
countryname	string	name of the country
countrycode	string	code of the country (ISO)

MODEL YOUR DATA

Now that we know what data we have, we need to model it in [RDF](#). For this purpose we will check the [AURORAL ontologies](#) and start looking for best match for our properties: `cellid`, `range`, `lon`, `lat`, `radio`, `operator`, `operatorcode`, `countryname`, `countrycode`.

In this case the best match is the [AURORAL CellTower ontology](#).

What if one or more of my properties are missing from the ontologies?

Sometimes there is no matching ontology for some of the properties you are trying to describe. This properties then need to be **excluded** from the final response since they can **not** be described in [RDF](#) format.

Let's start modeling. First we will have to `import` the ontology. We do this using `@context` keyword which is an array. This will import all the keywords from [AURORAL CellTower ontology](#):

```

{
  "@context": [
    "https://auroralh2020.github.io/auroral-ontology-contexts/cellTowers/context.json"
  ]
}

```

We are finally ready to start defining our `properties`. Based on the understanding of our data and the imported ontologies we can model the `cell tower` properties like this:

```

// FIRST CELL TOWER:
{
  "cellId": "461548", // <- cellid (cast: int -> string)
  "hasRange": {
    "range": "2052" // <- range (cast: int -> string)
  },
}

```

```

"hasOperator": {
  "operatorId": "1", // <- operatorcode (cast: int -> string)
  "operatorName": "Orange Slovensko" // <- operator
},
"country": {
  "code": "SK", // <- countrycode
  "name": "Slovakia" // <- countryname
},
"location": {
  "lat": 48.148721, // <- lat
  "long": 17.154939 // <- lon
},
"providesNetwork": {
  "@type": "UMTS" // <- radio
}
}

```

Now we need to do the same for the rest of the cell towers and put everything together with the `@context` to get the final result:

```

[
  // FIRST CELL TOWER:
  {
    "@context": "https://auroralh2020.github.io/auroral-ontology-contexts/cellTowers/context.json",
    "cellId": "461548",
    "hasRange": {
      "range": "2052"
    },
    "hasOperator": {
      "operatorId": "1",
      "operatorName": "Orange Slovensko"
    },
    "country": {
      "code": "SK",
      "name": "Slovakia"
    },
    "location": {
      "lat": 48.148721,
      "long": 17.154939
    },
    "providesNetwork": {
      "@type": "UMTS"
    }
  },
  // SECOND CELL TOWER:
  {
    "@context": "https://auroralh2020.github.io/auroral-ontology-contexts/cellTowers/context.json",
    "cellId": "77175573",
    "hasRange": {
      "range": "2816"
    },
    "hasOperator": {
      "operatorId": "6",
      "operatorName": "02 Slovakia"
    },
    "country": {
      "code": "SK",
      "name": "Slovakia"
    },
    "location": {
      "lat": 48.148685,
      "long": 17.1562553
    },
    "providesNetwork": {
      "@type": "LTE"
    }
  },
  // THIRD CELL TOWER:
  {
    "@context": "https://auroralh2020.github.io/auroral-ontology-contexts/cellTowers/context.json",
    "cellId": "255732702",
    "hasRange": {
      "range": "2457"
    },
    "hasOperator": {
      "operatorId": "6",
      "operatorName": "02 Slovakia"
    },
    "country": {
      "code": "SK",
      "name": "Slovakia"
    },
    "location": {
      "lat": 48.1493839,
      "long": 17.1556414
    },
    "providesNetwork": {
      "@type": "UMTS"
    }
  },
  // FOURTH CELL TOWER:
  {
    "@context": "https://auroralh2020.github.io/auroral-ontology-contexts/cellTowers/context.json",

```

```

"cellId": "255742702",
"hasRange": {
  "range": "858"
},
"hasOperator": {
  "operatorId": "6",
  "operatorName": "02 Slovakia"
},
"country": {
  "code": "SK",
  "name": "Slovakia"
},
"location": {
  "lat": 48.1489738,
  "long": 17.1571062
},
"providesNetwork": {
  "@type": "UMTS"
}
},
// FIFTH CELL TOWER:
{
  "@context": "https://auroralh2020.github.io/auroral-ontology-contexts/cellTowers/context.json",
  "cellId": "541989",
  "hasRange": {
    "range": "423"
  },
  "hasOperator": {
    "operatorId": "2",
    "operatorName": "Slovak Telekom"
  },
  "country": {
    "code": "SK",
    "name": "Slovakia"
  },
  "location": {
    "lat": 48.1494822,
    "long": 17.1549103
  },
  "providesNetwork": {
    "@type": "UMTS"
  }
}
]

```

CONNECTING YOUR DATA

To connect your data to AURORAL you will have to make the response, modeled in the previous step, accessible and tell the Node where to GET it. This will ensure that every time someone wants to access or discover your data in the AURORAL, the platform knows where to look for it and understands the context of the data.

To do so you create a module that will take care of generating the response and populating the proper fields with the data. In AURORAL this modules are called the `Adapters`. There are multiple existing generic `Adapters` that can be used to automate this process but for this example we will be creating a custom adapter.

Adapters

Generic adapters like `Node-red` are good way to simplify the process of connecting and registering the item. Even though they can be used to register almost any item, the disadvantage of using the generic adapters is that they usually put an overhead cost for your system resources. To avoid this you can build your custom adapter that will handle the `RDF` modeling and data populating. To find out more about the `Adapters` please refer to [Adapters](#) section.

Since we are the owners of the service we can just extend the API of our service with the new endpoint called `/api/cell-towers/closest/rdf-mapping` with the following `OpenAPI` specification:

GET /api/cell-towers/closest/rdf-mapping Get Closest Towers Rdf Mapping

Parameters Try it out

Name	Description
lat * required number (query)	<input type="text" value="lat"/>
lon * required number (query)	<input type="text" value="lon"/>

Responses

Code	Description	Links
200	Successful Response	No links

Media type:

Controls Accept header.

Example Value | Schema

🔗

This new endpoint will return the same response as `/api/cell-towers/closest`, only now enriched with RDF based on our modeled response from previous step:

```
http://localhost:8000/api/cell-towers/closest/rdf-mapping?lat=48.151135386739156&lon=17.13893244921691
```

```
[
- {
  @context: "https://auroralh2020.github.io/auroral-ontology-contexts/cellTowers/context.json",
  cellId: "255732812",
  - hasRange: {
    range: "1000"
  },
  - hasOperator: {
    operatorId: "6",
    operatorName: "O2 Slovakia"
  },
  - country: {
    code: "SK",
    name: "Slovakia"
  },
  - location: {
    lat: 48.151128,
    long: 17.138952
  },
  - providesNetwork: {
    @type: "UMTS"
  }
},
- {
  @context: "https://auroralh2020.github.io/auroral-ontology-contexts/cellTowers/context.json",
  cellId: "125493",
  - hasRange: {
    range: "1000"
  },
  - hasOperator: {
    operatorId: "2",
    operatorName: "Slovak Telekom"
  },
  - country: {
    code: "SK",
    name: "Slovakia"
  },
  - location: {
    lat: 48.151229,
    long: 17.138948
  },
  - providesNetwork: {
    @type: "LTE"
  }
},
- {
```

Adapter needs to be accessible!

Please don't forget that Node needs to be able to access the adapter that is taking care of the mapping. For example if your adapter is running on localhost of the machine, please make sure the Node is also running on the same machine or can reach this machine over the DNS.

REGISTER YOUR SERVICE

To register the service you will have to provide the Thing Description of the service to the Node.

Below is the Thing description for our service. To find out step-by-step tutorial how to generate this example please visit [here](#):

```
{
  "@context": [
    "https://www.w3.org/2019/wot/td/v1",
    "https://auroralh2020.github.io/auroral-ontology-contexts/core/services.json"
  ]
}
```

```

},
"@type": "Service",
"adapterId": "bavenir-celltower-service",
"title": "Closest Cell Towers Service",
"description":
  "Cell tower data service"
,
"securityDefinitions": {
  "nosec_sc": {
    "scheme": "nosec"
  }
},
"security": "nosec_sc",
"language": "eng",
"place": "Slovakia",
"serviceFree": [ true ],
"applicableGeographicalArea": "Europe",
"hasDomain": "Mobility",
"hasSubDomain": "Coverage",
"hasFunctionality": "Only read",
"provider": "bAvenir",
"currentStatus": "Available",
"properties": {
  "gettowers": {
    "type": "array",
    "title": "Get towers",
    "readOnly": true,
    "description": "Returns 5 closest cell towers based on user's location [lat,lon]",
    "forms": [
      {
        "href": "http://localhost:8000/api/cell-towers/closest/rdf-mapping"
      }
    ],
    "items": {
      "type": "object",
      "@type": "https://auroral.iot.linkeddata.es/def/cell#CellTowers"
    }
  }
}
}
}

```

As for the `forms` array field, this is where all the endpoints (in our case only one endpoint) for accessing the data should go. Since we are using the custom adapter, we have to provide the `url` where the Node can access our enriched data using the `href` field.

Service store

Every `public` service you register in the AURORAL platform is automatically available in [Service store](#). If you want to know more about how the Service store works, please refer to our short [tutorial video](#).

Now when we have our data source described in a Thing description, we can register it in AURORAL. To do this we expect your Node is installed with `Custom adapter` extension and is running.

How to install the Custom adapter extension?

When you are installing new `Node`, you have an option to install `Custom adapter` extension. To find out how to do this please refer to our [Getting started - install a Node](#) section.

To register the service you need to use the Node's API. Navigate to `url` where the Node is running. If you have been following the tutorials we provide your node should be running on `localhost:81`.

The endpoint you need to use is `/api/registration`. Just replace the `td: {}` in body with [Thing Description](#) from previous step and press `Execute` :

Registry Registration of new devices or services into AURORAL

POST /api/registration Register an item

Register an object in the AURORAL platform.

For the registration AURORAL uses W3C Thing Descriptions, and optional parameters to send additional information to the platform.

- `id`: [Mandatory] Thing Description Object.
- `avatar`: [Optional] Base64 image representation for the object. It will be displayed in the Neighbourhood Manager.

For creating a Thing Description please refer [here](#), note that besides the standard, other fields are accepted. Currently the `adapterId` and the `@type`.

- `adapterId`: It supports adding an additional ID that would help the integrator map the object to the local infrastructure. If it is not included it will be equal to the AURORAL object ID (`oid`).
- `@type`: Can be added at global level and also for each interaction. Describes the type of object or interaction respect to the AURORAL ontology. For list of available values refer to the ontology.

Or use the online [editor](#).

Example TDs can be found in [repository](#).

Note: An AURORAL object ID or OID will be automatically added during the registration under the field `id`. This id will be the same as the OID in the AURORAL platform.

Parameters
Cancel
Reset

No parameters

Request body required
application/json

Registration info

```

{
  "id": {
    "@context": [
      "https://www.w3.org/2019/wot/td/v1",
      "https://auroralh2020.github.io/auroral-ontology-contexts/core/services.json"
    ],
    "@type": "Service",
    "adapterId": "havenir-calltower-service",
    "title": "Closest Cell Towers Service",
    "description": "Cell tower data service"
  },
  "securityDefinitions": {
    "nosec_sc": {
      "scheme": "nosec"
    }
  },
  "security": "nosec_sc",
  "language": "eng",
  "..."
}

```

Execute

avatar: "string"

This field is not mandatory to provide when you are registering the service but if you are not providing any value you should remove it.

After a while we can see that our device is registered and after enabling it can be used in AURORAL.

? Do not see the device?

Maybe you forgot to enable the device. If you don't enable device, it will not be visible in AURORAL. More about enabling devices can be found [here](#).

2.3.4 Using data from AURORAL

Data consumer

In this scenario we want to use a service to collect data from other AURORAL Nodes. Then this data can be used to be processed and offered back into AURORAL, to be stored or to be displayed in a UI for instance.

This are the steps to be followed:



- 1. Create a service:** You need to have a service ready to provide some value to AURORAL.
- 2. Register your service:** Finally, you need to tell AURORAL where to find your data. For this purpose you will have to register your service using the AURORAL Node. The registration is perform with a type of document called Thing Description, in it you describe your service main and info and the URIs for accessing the data that your adapter is making available.
[Learn more about registering your service](#) →
- 3. Connect your service:** I have a service, how can I connect it to AURORAL. In this case we do not need an adapter, though it could also be used. As services are software, your application can be updated with a module to interact with AURORAL. In case when you just want to use the data and you are not exposing any data to AURORAL, you just need to call the API for requesting the data from your service.
[Learn more about connecting your service](#) →
- 4. Discover data in AURORAL:** Now that you have your service registered, you can discover the data that is available in AURORAL. You can use the API calls or the Neighbourhood Manager to find the data that you are interested in.
[Learn more about discovering data](#) →
- 5. Create a contract:** As all the data in AURORAL is private by default, you need to have a contract with the owner of the data. This contract will allow you to access the data and use it in your service.
[Learn more about creating a contract](#) →
- 6. Consume data:** Now that you have a contract, you can access the data. You can use the API calls to access the data
[Learn more about consuming data](#) →

Example

CREATE A SERVICE

In this example we will use tool [service-data-connector](#) developed by bAvenir. This tool can be used to collect data from different sources defined by the user. The data is then stored in a choosen database - InfluxDB in our case.

REGISTER YOUR SERVICE

Requirements:

- Auroral node
- InfluxDB running

We register our service with following [TD](#) to AURORAL node:

```
{
  "@context": [
    "https://www.w3.org/2019/wot/td/v1",
    "https://auroralh2020.github.io/auroral-ontology-contexts/core/services.json"
  ],
  "@type": "Service",
  "title": "toInfluxService",
  "description": "Service - Influx data collector",
  "provider": "bAvenir",
  "currentStatus": "Available",
  "hasDomain": "Environment",
  "hasSubDomain": "Indoor",
  "hasFunctionality": "Only read",
  "serviceFree": true,
  "numberOfDownload": 1,
  "versionOfService": "1.4",
  "language": "en",
  "place": "Bratislava",
  "securityDefinitions": {
    "nosec_sc": {
      "scheme": "nosec"
    }
  },
  "security": "nosec_sc ",
  "properties": {}
}
```

We can confirm registration by checking the list of registered items in [Neighbourhood Manager](#).

CONNECT YOUR SERVICE

Your service can access data and [metadata](#) from AURORAL by using the Node's [API](#). For actual data access, we will use API call `/api/properties/{id}/{oid}/{pid}`. - `id` is the id of my service - `oid` is the id of the owner of the data - `pid` is the id of the property that I want to access

Our tool is already prepared to interact with AURORAL devices, so we just need to specify urls for concrete devices and properties. Let's move this step when we have a contract with the owner of the data and we know the ids of the devices and properties.

DISCOVERING DATA IN AURORAL

In this example we found desired data in one of partner's devices. We can retrieve [TD](#) of the device to check what data are available.

```
{
  "@context": [
    "https://www.w3.org/2019/wot/td/v1",
    {
      "adp": "https://auroral.iot.linkeddata.es/def/adapters#",
      "om": "http://www.ontology-of-units-of-measure.org/resource/om-2/",
      "geo": "http://www.w3.org/2003/01/geo/wgs84_pos#"
    }
  ],
  "security": [
    "nosec_sc"
  ],
  "securityDefinitions": {
    "nosec_sc": {
      "scheme": "nosec"
    }
  },
  "geo:location": {
    "geo:lat": "48.17422",
    "geo:long": "17.18163"
  },
  "title": "TemperatureSensor1",
  "@type": "adp:Thermometer",
  "description": "Room temperature sensor",
  "properties": {
    "room_temperature": {
      "title": "room_temperature",
      "description": "temperature in the room",
      "@type": "adp:AmbientTemperature",
      "unit": "om:degree_Celsius",
      "readOnly": true,
      "type": "number",
      "forms": [
        {
          "op": "readproperty",
          "href": "http://node-red:1250/api/property/f8041311-d45f-4b35-a046-e6e823a4b13c/room_temperature"
        }
      ]
    }
  }
}
```

```

},
"adapterId": "t43freas",
"id": "f8041311-d45f-4b35-a046-e6e823a4b13c"
}

```

CREATE A CONTRACT

So we now that we want to get **room_temperature** property from **TemperatureSensor1** and as it matches our requirements, we can create a contract with the owner of the device.

Process of creating a contract is described in [Relationships](#) section. In next steps we assume that we have a contract with the owner of the device.

Accessing partner's data

During data access, we need to have also access to metadata of the device. Result of that is, that bot involved nodes needs to be shared in partnership or community (if items are not public). More information about relationships [here](#)

CONSUME DATA

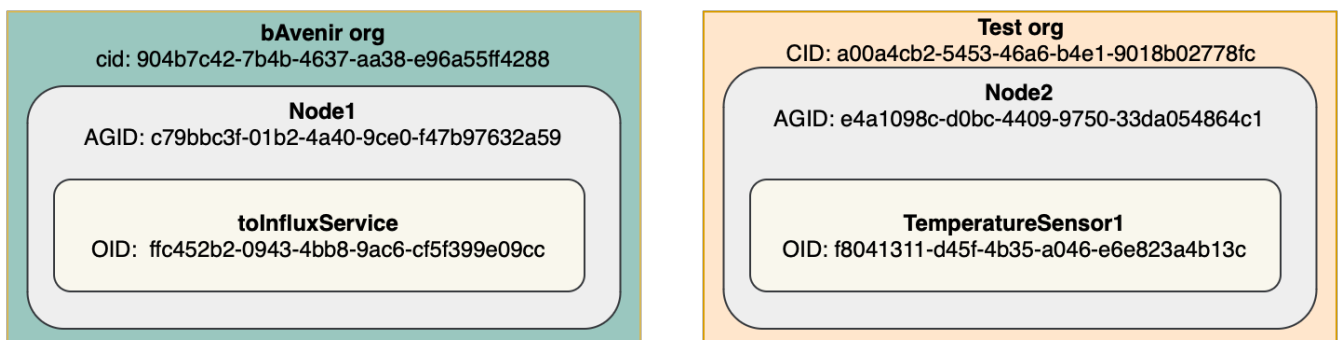
This is format of the data that we get from the device:

```

{
  "@context": [
    "https://auroralh2020.github.io/auroral-ontology-contexts/adapters/context.json",
    {
      "om": "http://www.ontology-of-units-of-measure.org/resource/om-2/"
    }
  ],
  {
    "oid": {
      "@id": "rdfs:label"
    },
    "iid": {
      "@id": "rdfs:label"
    }
  }
],
"@type": "adp:Thermometer",
"oid": "f8041311-d45f-4b35-a046-e6e823a4b13c",
"iid": "temperature",
"measurement": [
  {
    "property": "adp:AmbientTemperature",
    "value": 22.28,
    "isMeasuredIn": "om:degree_Celsius",
    "timestamp": "2023-02-28T09:18:09.610Z"
  }
]
}

```

There will be lot of ids in next steps, so to make it easier, we will use the following drawing to explain them:



Now we can configure and start service-data-connector.

```

SERVICE_ENV=development
SERVICE_IP="0.0.0.0"
DS_FILE="ds.json"
DATA_CONNECTOR_PORT="1444"
DB_TYPE=influx
INFLUXDB_USERNAME="admin"
INFLUXDB_URL="http://myhost:8086"

```

```
INFLUXDB_PASSWORD="secretPassword"
INFLUXDB_ORG="myorg"
INFLUXDB_BUCKET="bucket0"
INFLUXDB_TOKEN="my_generated_token"
```

According to ids in the drawing, we created this configuration file for service-data-connector:

```
{
  "ds": [
    {
      "enabled": true,
      "dsid": "myId123",
      "oid": "f8041311-d45f-4b35-a046-e6e823a4b13c",
      "agid": "90467c42-7b4b-4637-aa38-e96a554288",
      "cid": "904b7c42-7b4b-4637-aa38-e96a55ff4288",
      "iid": "room_temperature",
      "type": "read",
      "service": "ffc452b2-0943-4bb8-9ac6-cf5f399e09cc",
      "requestUrl": "http://myhost:81/api/properties/ffc452b2-0943-4bb8-9ac6-cf5f399e09cc/f8041311-d45f-4b35-a046-e6e823a4b13c/room_temperature",
      "monitors": "temperature",
      "frequency": 6000,
      "queryParams": {},
      "body": {}
    }
  ]
}
```

Most of the ids are self-explanatory, but there are some that need to be explained.

- requestUrl is the url of the property that we want to read:
- http://myhost:81 - this is the url of my node
- /api/properties - API route for reading properties
- fc452b2-0943-4bb8-9ac6-cf5f399e09cc - this is the id of the service that we want to use to read the data
- f8041311-d45f-4bb8-9ac6-cf5f399e09cc - this is the id of the device that we want to read the data from
- room_temperature - this is the id of the property that we want to read
- dsid - unique id of the data source - generated by user
- frequency - how often the data should be read from the device (in milliseconds)

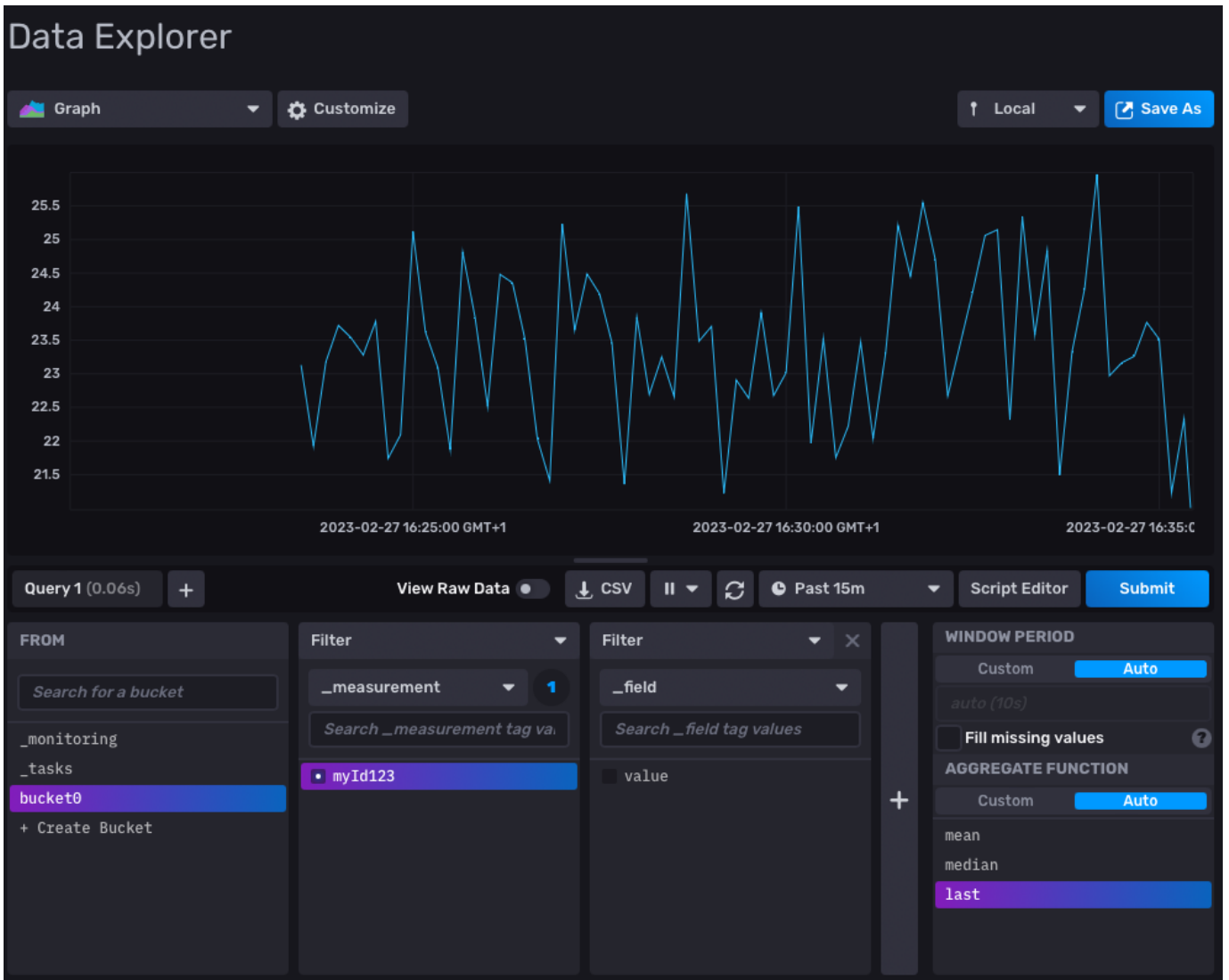
After the configuration is done, we can start the service-data-connector.

```
docker run -d --name service-data-connector -p 1444:1444 -v $(pwd)/ds.json:/service/ds.json -v $(pwd)/.env:/service/.env service-data-connector
```

If everything is configured correctly, we should see the following log in the service-data-connector container

```
2023-02-27T15:23:22.443Z info [undefined]: #####
2023-02-27T15:23:22.444Z info [undefined]: #####
2023-02-27T15:23:22.444Z info [undefined]: Starting SERVICE!!
2023-02-27T15:23:22.444Z info [undefined]: Influx DB
2023-02-27T15:23:22.445Z info [undefined]: Loaded data streams: 1
2023-02-27T15:23:22.446Z info [undefined]: #####
2023-02-27T15:23:22.446Z info [undefined]: #####
2023-02-27T15:23:28.449Z debug [undefined]: Collecting data from http://localhost:81/api/properties/ffc452b2-0943-4bb8-9ac6-cf5f399e09cc/f8041311-d45f-4b35-a046-e6e823a4b13c/room_temperature
2023-02-27T15:23:34.448Z debug [undefined]: Collecting data from http://localhost:81/api/properties/ffc452b2-0943-4bb8-9ac6-cf5f399e09cc/f8041311-d45f-4b35-a046-e6e823a4b13c/room_temperature
```

And after some time, we should see the data in the influxdb database:



And that's it. We have successfully read data from the device and stored it in the database.

3. Key Features

3.1 Organisations

Organization can be thought of as an umbrella that groups together a set of users. This structure provides a way to manage and categorize users within the platform. The organization structure can help to define the relationships between users, as well as to enforce certain rules, policies, and permissions.

Overall, the organizational structure provides a convenient and effective way to manage large groups of users and to maintain control over the flow of information and resources within the platform.

There is no user without the organisation

In the AURORAL platform each user belongs to exactly one organization, there is no concept of a user existing outside of the organizational structure. In this system, the organization is considered the central and defining aspect of the user's identity, and all users must be associated with a specific organization.

3.1.1 Review process

All new organizations must go through a review process conducted by our colleagues. This process is designed to ensure that the organization meets certain standards and criteria, and are compliant with the AURORAL goals.

Cooperation with the AURORAL

To find out more about the companies that collaborate with the AURORAL please visit [Auroral website](#).

Inviting organisation to join AURORAL

When an organization is `invited` to join the platform by other organisation, it can often be an `advantage` for the new organization in terms of the time spend on onboarding process. Since organisation sending the invitation has been already reviewed by our colleagues, the review process can be `skipped` for the new organisation.

3.1.2 How to?

Here you will find an tutorials on how to `setup` your organisation account or `invite` other organisation to join the platform.

Create the organisation

To find out how to create the organisation account in AURORAL platform, please refer to our [Getting started](#) section.

Send invitation

Administrator role required

Only users with `administrator` role are allowed to send invitations to other companies.

To send an invitation for other company to join AURORAL, first you will have to `login` to your organisation account in [Neighbourhood Manager here](#) and then:

1. press `add` button located in a top right corner of the website
2. select `invite other company` option from the drop down menu:

Here you will have to provide the `name` of the organisation and `email` where the invitation will be sent to:

Invite new company to join AURORAL
✕

Name of the company: *

Email address: *

Send invitation

After you press `Send invitation` company should receive email with link to registration form. Below is an example how this form looks like.

 **Example: Organisation registration form**

AURORAL

Company

Name *	<input type="text" value="bca"/>
Location	<input type="text" value="Location of your organisation"/>

Administrator

Email *	<input type="text" value="company2@bavenir.eu"/>
Password *	<input type="password" value="Password"/>
Repeat password *	<input type="password" value="Same password again"/>
Name*	<input type="text" value="First name of administrator"/>
Surname *	<input type="text" value="Last name of administrator"/>
Occupation*	<input type="text" value="Occupation of administrator"/>

[Terms and Conditions](#)

This is an **example** of registration form sent to a company.