COLLECTiEF

REPORT ON COLLECTIEF CLUSTER NODE

Project acronym: COLLECTiEF

Project title: Collective Intelligence for Energy Flexibility

Call: H2020-LC-SC3-2018-2019-2020

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

Disclaimer

Project no.	101033683
Project acronym:	COLLECTiEF
Project title:	Collective Intelligence for Energy Flexibility
Call:	H2020-LC-SC3-2018-2019-2020
Start date of project:	1 June 2021
Duration:	48 months
Deliverable title:	Report on COLLECTiEF Cluster Node
Deliverable No.:	D3.4
Document Version:	2.1
Due date of deliverable:	31.05.2023
Actual date of submission:	31.05.2023
Deliverable Lead Partner:	Partner No. 7, NODAIS AB (NODA)
Work Package:	3
No of Pages:	26
Keywords:	architecture, brig, broker, client, cluster node, container, containerized, database, decentralized, distributed, docker, docker-compose, edge node, eventual consistency, message passing, mqtt, multi-agent system, mysql, postgresql, publish-subscribe, server

Name	Organization
Jens Brage	Partner No. 7, NODA

Dissemination level	
PU	Public

History

Version	Date	Reason	Revised by
1.0	20.05.2023	First version	Jens Brage, NODA
1.1	24.05.2023	Internal review	Kavan Javanroodi, ULUND
2.0	25.05.2023	Final version	Jens Brage, NODA
2.1	31.05.2023	Final approval	Mohammadreza Aghaei, NTNU

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

Executive Summary

This deliverable reports on Task 3.3, Implementation of the Cluster Node. It details the architecture of the cluster node, motivates technical decisions, and explains how the architecture supports the COLLECTiEF solution. Since the report is public, it avoids possibly confidential algorithms. Instead, it focuses on architecture and how to provide a general framework sufficient for the NODA solution as well as the novel demand side management (DSM) solution (called nDSM in this report) but abstracts over the details of the latter.

In addition to the above, the work exemplifies the benefits of frameworks for eventual consistency, preferably with support for automatically deriving the corresponding message passing from application-specific use of persistent memory. And while the report does not provide such a framework, it provides a light-weight substitute and details how it can be used to implement parts of the COLLECTiEF solution.

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

Table of Contents

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

List of Acronyms

API	Application Programming Interface
BRIG	Borted Router + iGateway (Edge Node)
CI	Collective Intelligence
D	Deliverable
DSM	Demand Side Management
ECMWF	European Centre for Medium-Range Weather Forecasts
JSON	JavaScript Object Notation
JSONB	JavaScript Object Notation, Binary
MPC	Model Predictive Control
MQTT(S)	Message Queuing Telemetry Transport (Secure)
nDSM	novel Demand Side Management
RL	Reinforcement Learning
UUID	Universally Unique IDentifier
WP	Work Package

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

List of Figures

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

List of Tables

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

1. Introduction

This deliverable reports on Task 3.3, Implementation of the Cluster Node. It details the architecture of the cluster node, motivates technical decisions, and explains how the architecture supports the COLLECTiEF solution. Since the report is public, it avoids possibly confidential algorithms. Instead, it focuses on architecture and how to provide a general framework sufficient for the NODA solution as well as the demand side management (nDSM) solution but abstracts over the details of the latter.

The Cluster Node is not an isolated component but depends on its environment for weather forecasts and communication with the Edge Node. Moreover, beyond the algorithms implemented by the Cluster Node and Edge Node, it is necessary to make sure that the solutions for distributed persistent memory and communication can be configured, tested, and evaluated in a controlled manner in preparation for deployment. The situation is complicated by the asynchronous and decentralized nature of the system, but by providing mock-ups of the environment as well as the Edge Node and its interactions with sensors and actuators, progress can be made. To this end, this report does not only concern itself with the Cluster Node, but also provides architecture for an Edge Node running the NODA solution.

The approach permits the NODA and nDSM solutions to be run concurrently on the Cluster Node and to delay the combination of the corresponding energy cost and energy mode until on the Edge Node, where the energy mode is just another name for the flexibility signal by the nDSM solution. This can presumably be done in several ways, but this report only considers the following mechanism:

- For `energy_mode == 0`, or the absence of a recent energy mode, the NODA solution controls the edge.
- For `energy_mode >= 1`, the nDSM solution controls the edge.

In short, the report covers the Cluster Node as well as parts of the Edge Node but abstracts over sensors and actuators. And while it remains to align on some details of the Edge Node pertaining to sensors and actuators, the remaining work is expected to be manageable.

The report consists of two parts, Section 2, which describes a further development of the collectief-members/minimas framework of D2.1, and Sections 3-4, which describe how the framework can be used to implement the desired functionality of the Cluster Node and more. The code is currently being migrated to the GitHub repository collectief-members/cluster-node. The repository name reflects the project plan but is otherwise a misnomer as the content cuts across several different domains such as the Cluster Node, the Edge Node, algorithms, and the communication layer.

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

2. Architecture, Software

Decentralized systems and more generally, distributed systems, are well known to be challenging to design, develop and maintain, and how to overcome these challenges constitute an active area of research. The approach presented here is inspired by the two recent and related developments of conflict-free replicated data types (CFRD) and eventual consistency. The idea is to structure distributed state in such a way that the various parts can be merged into a global state through the exchange of patches in a way that largely does not depend on the order of the exchange. And while this is still an active area of research, it is possible to achieve some of the benefits by adhering to the following principles:

- Every part of the system should use the same table structure for its database.
- The table structure should permit different databases of the same system to be merged into one database without risk of confusion of what entries belong together.
- While every part of the system can in principle be allowed to inspect the entire global state, it is only allowed to update its local state.
- There is a way to communicate those updates to the global state that occur in practice.

Adhering to the above principles, it is possible to use a publish-subscribe messaging patten to synchronize the global state across the local instances. However, to uphold privacy, is necessary to keep track of who is allowed to inspect what part of the global state and to restrict the messages accordingly. Ideally, the rights management should be expressed in code and the synchronization derived from the rights management. However, a general solution is beyond the scoop of the project, and the message exchange will be implemented by hand.

Another and related challenge with decentralized and distributed systems is how to avoid introducing errors as part of the work of configuring such a system. And while a configuration can be equated to a message expressing an update to the global state, unless the message format is simple enough to rule out errors in the first place, the work of configuring the system is likely to require greater effort than desired. For example, consider the need to relate different parts of the system by mentioning a corresponding key in more than one place. Although simple in the small, it should be avoided in the large.

2.1. Framework

The challenge of relating different parts of the system without mentioning a corresponding key in more than one place can be avoided by restricting attention to tree-shaped structures. Fortunately, this applies to the part of the COLLECTiEF solution under consideration, and it turns out that following PostgreSQL table structure suffices:

Table 1 Database Table Structure (s, sindex, t, tindex)

~~~
```postgresql
DROP TABLE IF EXISTS t;

DROP TABLE IF EXISTS s;

CREATE TABLE s(
   coindex SERIAL8 PRIMARY KEY,
   covalue INT8 REFERENCES s(coindex) ON DELETE CASCADE,
   index TEXT,
   value JSONB,
~~~

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

```
    UNIQUE (covalue, index)
); -- spatial

CREATE INDEX sindex ON s(covalue, index);

CREATE TABLE t(
    coindex SERIAL8 PRIMARY KEY,
    covalue INT8 REFERENCES s(coindex) ON DELETE CASCADE,
    index TIMESTAMP,
    value FLOAT8,
    UNIQUE (covalue, index)
); -- temporal

CREATE INDEX tindex ON t(covalue, index);
```
```

The table structure expresses a forest of trees where every s-node has a locally unique index: TEXT, a value: JSONB, and some associated time series data. The content can be serialized and deserialized using dataclasses and dataclasses_json, and by permitting None in some places, the corresponding messages can also be used to delete content.

Table 2 Data Model Data Classes (TDumps, SDumps)

```
```python
@dataclass_json
@dataclass
class TDumps:
    index: datetime
    value: float


@dataclass_json
@dataclass
class SDumps:
    index: str
    value: Any # no direct counterpart for JSONB
    tdumps_list: list['TDumps']
    sdumps_list: list['SDumps']
```
```

The uniqueness constraints are necessary for the intended interpretation of patch: list['SDumps'].

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683
```

Given the uniqueness constraints, the table structure can in principle be serialized and deserialized using dictionaries. However, without the uniqueness constraints, associative lists better reflect the semantics. Consequently, the corresponding classes (TIndex, T, SIndex, S) uses associative lists.[1]

**Table 3 Data Model (TIndex, T, SIndex, S)**

~~~
```python
class TIndex(object):

    def __init__(self, cursor: Cursor, covalue: int | None = None) -> None:
        ...

    def __getitem__(self, index: datetime | None) -> list['T']:
        ...

    def __setitem__(self, index: datetime | None, dumps_list: list['TDumps']) -> None:
        ...

    def dumps(self) -> list['TDumps']:
        ...


class T(object):

    def __init__(self, cursor: Cursor, covalue: int | None = None, index: datetime | None = None, value: float | None =
None, coindex: int | None = None) -> None:
        ...

    @property
    def covalue(self) -> int | None:
        ...

    @covalue.setter
    def covalue(self, covalue: int | None) -> None:
        ...

    @property
    def index(self) -> datetime:
        ...

    @index.setter
    def index(self, index: datetime) -> None:
~~~

[1] If I, the author, where to rework this solution, then I would start with a structure of nested dictionaries, guaranteeing local uniqueness by design, and then worry about how to implement a corresponding ORM.

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

```
        ...

    @property
    def value(self) -> float:
        ...

    @value.setter
    def value(self, value: float) -> None:
        ...

    def dumps(self) -> 'TDumps':
        ...

class SIndex(object):

    def __init__(self, cursor: Cursor, covalue: int | None = None) -> None:
        ...

    def __getitem__(self, index: str | None) -> list['S']:
        ...

    def __setitem__(self, index: str | None, dumps_list: list['SDumps']) -> None:
        ...

    def dumps(self) -> list['SDumps']:
        ...


class S(object):

    def __init__(self, cursor: Cursor, covalue: int | None = None, index: str | None = None, value: Any | None = None,
coindex: int | None = None) -> None:
        ...

    @property
    def covalue(self) -> int | None:
        ...

    @covalue.setter
    def index(self, covalue: int | None) -> None:
        ...

    @property
    def index(self) -> str:
        ...

    @index.setter
```

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

```
    def index(self, index: str) -> None:
        ...

    @property
    def value(self) -> Any:
        ...

    @value.setter
    def value(self, value: Any) -> None:
        ...

    @property
    def sindex(self) -> 'SIndex':
        ...

    @sindex.setter
    def sindex(self, dumps_list: list['SDumps']) -> None:
        ...

    @property
    def tindex(self) -> 'TIndex':
        ...

    @tindex.setter
    def tindex(self, dumps_list: list['TDumps']) -> None:
        ...

    def dumps(self) -> 'SDumps':
        ...

    def search(self, predicates: list[Callable[[Any], bool]]) -> list['S']:
        if predicates:
            return [
                s
                for s in self.sindex[None] if predicates[0](s.value)
                for s in s.search(predicates[1:])
            ]
        else:
            return [self]
```
```

To fully understand how the above constructions handle None and NULL, it is necessary to study the corresponding SQL queries. However, here it suffices to note that with these constructions, the method search in particular, it becomes unproblematic to implement a MQTT client for publishing and/or subscribing to such data. While for the cluster node, it suffices to characterize nodes according to their tree depth in the s-table, the search method enables addressing finer structures with ease.

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683
```

Note that, to serve the intended purpose, to update a time series with some other time series data, it is necessary to first delete the data in the time series from the earliest index: datetime | None in the time series data and then insert the time series data, and that this is already handled by the API method TIndex.__setitem__. And for this method as well as for the other API methods, None functions as wildcard.[2]

This leaves the publish-subscribe messaging solution. With the above constructions, this becomes trivial, and it suffices to publish some patch: list['SDumps'] under some topic and use it to update other local databases.

Table 4 Message Payload Data Class (Payload)

~~~~
```python
@dataclass_json
@dataclass
class Payload:
    patch: list['SDumps']
```
~~~~

Consequently, it suffices to implement one MQTT client for publishing and subscribing to such topics, parametrized over the choice of broker, the choice of topics and the content of the patches.

For COLLECTiEF, it suffices to communicate time series data since an offset relative to the current time and, in addition, either publish such data or subscribe to and validate such data. And to keep things simple, the ability to configure the MQTT client has been limited accordingly.

Table 5 Default Client (DefaultClient)

~~~~
```python
class DefaultClient(Client):

    def __init__(
            self,
            connection: Connection,
            topic: str,
            patch: list[tuple[list[str | None], timedelta | None]],
            callback: Callable[[None], None]
    ) -> None:
        self._connection = connection
        self._topic = topic
        self._patch = patch
        self._callback = callback
        super().__init__()
        self.on_connect = self._on_connect
        self.on_message = self._on_message
~~~~

[2] This seemingly special treatment of index: datetime | None for time series data can be understood as part of a more general pattern, where the indices are subject to a partial order with a least element None.

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

```
    def _on_connect(self, client: Client, userdata: Any, flags: dict[str, int], rc: int) -> None:
        ...

    def _on_message(self, client: Client, userdata: Any, message: MQTTMessage) -> None:
        ...

    @overload
    def publish(self, index: datetime) -> MQTTMessageInfo:
        ...
```
```

The client is limited to one topic: str for which it either publish or subscribe to and validate time series data since an offset relative to the current time according to patch: list[tuple[list[str | None], timedelta | None]]. To facilitate testing, the client is also parametrized by callback that is triggered after the client has processed a message. This mechanism makes it possible to run the asynchronous system in a synchronous way and in simulated time, and thus test complex scenarios as fast as the computational resources permit.

The framework is contained in the two modules minimas.measure (data) and minimas.control (publish-subscribe), which improves on the functionality of the `minimas` framework of D2.1.

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683
```

# 3. Architecture, System

Python is widely used in research and development due to its extensive collection of third-party modules and robust ecosystem. It is also recognized for its package management and environment solutions, although they tend to receive mixed reviews.[3]

In the past decade, containers have emerged as a practical solution to handle the complexities associated with software development. They provide developers with the ability to write software on one computer and confidently run it on another. However, using containers introduces additional complexity and a more intricate development environment. Nevertheless, the benefits of containers generally outweigh the drawbacks, resulting in overall lower effort.

During the same period, microservice architectures experienced a rise and fall in popularity. This approach involves breaking down large computer systems into smaller, independent solutions that can be modified and redeployed separately. While microservice architectures have their advantages in certain scenarios, they are not considered a universal solution. The increased bureaucracy and complexity associated with managing internal communication often outweigh the benefits.

However, there are specific cases where microservice architectures prove to be a sensible choice. For instance, they are suitable for developing decentralized or distributed software and enabling collaborative deployment of software by multiple organizations, such as in the case of COLLECTiEF.

Docker and Docker Compose offer solutions for packaging software applications into containers and managing their dependencies, configurations, and deployment. Docker allows developers to create portable and isolated environments called containers, ensuring consistent behavior across different systems. Docker Compose simplifies the management of multiple containers, enabling developers to define and orchestrate their relationships.

Docker and Docker Compose are often preferred for their development-friendly features, which is also why they are used here. Other alternatives, like Kubernetes, instead focus on container orchestration at scale. Kubernetes provides advanced features for automating deployment, scaling, and management of containers across clusters of machines.

## 3.1. Example Configurations

Building on the details of Section [Architecture, Software] and the above introduction to containers, it is possible to accurately depict an example configuration of the intended system to serve as a point of reference for further discussions, see Figure 1, where `node1` corresponds to the Cluster Node and `node2` corresponds to the Edge Node. The figure is subject to the following graphical notation,

- Arrow, dashed: Control flow, mixing event-driven subscribers and scheduled modelers and publishers.
- Arrow, solid: Data flow.
- Box: Container labeled by name for Docker Compose (PostgreSQL, MQTT, Python).

[3] https://xkcd.com/1987/

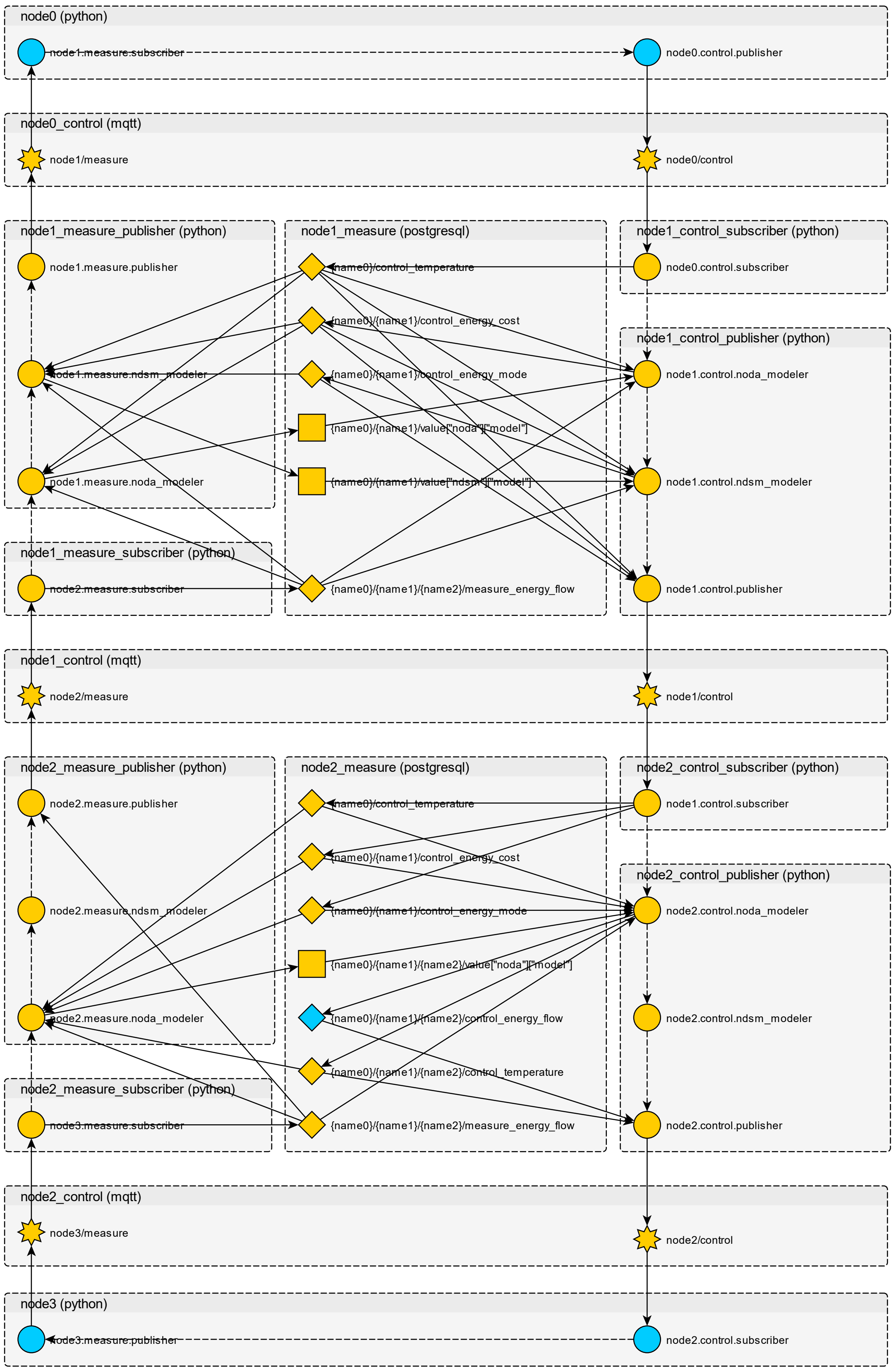
This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

- Circle: Module labeled by fully qualified name.
- Diamond: Time-series labeled by fully qualified name.
- Square: JSONB data labeled by fully qualified name.
- Star: MQTT message labeled by topic.

Take note of the usage of curly braces, which are used to denote placeholders for locally unique names.

The graphics has been produces using yEd, which provides a solution for automatic layout, that is useful for sorting out this kind of diagrams.

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683



node0 (python)
node1.measure.subscriber
node0.control.publisher
node0_control (mqtt)
node1/measure
node0/control
node1_measure_publisher (python)
node1.measure.publisher
node1.measure.ndsm_modeler
node1.measure.noda_modeler
node1_measure (postgresql)
{name0}/control_temperature
{name0}/{name1}/control_energy_cost
{name0}/{name1}/control_energy_mode
{name0}/{name1}/value["noda"]["model"]
{name0}/{name1}/value["ndsm"]["model"]
{name0}/{name1}/{name2}/measure_energy_flow
node1_control_subscriber (python)
node0.control.subscriber
node1_control_publisher (python)
node1.control.noda_modeler
node1.control.ndsm_modeler
node1.control.publisher
node1_measure_subscriber (python)
node2.measure.subscriber
node1_control (mqtt)
node2/measure
node1/control
node2_measure_publisher (python)
node2.measure.publisher
node2.measure.ndsm_modeler
node2.measure.noda_modeler
node2_measure (postgresql)
{name0}/control_temperature
{name0}/{name1}/control_energy_cost
{name0}/{name1}/control_energy_mode
{name0}/{name1}/{name2}/value["noda"]["model"]
{name0}/{name1}/{name2}/control_energy_flow
{name0}/{name1}/{name2}/control_temperature
{name0}/{name1}/{name2}/measure_energy_flow
node2_control_subscriber (python)
node1.control.subscriber
node2_control_publisher (python)
node2.control.noda_modeler
node2.control.ndsm_modeler
node2.control.publisher
node2_measure_subscriber (python)
node3.measure.subscriber
node2_control (mqtt)
node3/measure
node2/control
node3 (python)
node3.measure.publisher
node2.control.subscriber

**Figure 1 Example Configuration, where node1 = Cluster Node and node2 = Edge Node**

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

The example configuration of Figure 1 is close to the intended production configuration with the following caveats,

- To facilitate offline development, it has a mock `node0` corresponding to the environment.
- To facilitate offline development, it has a mock `node3` corresponding to sensors and actuators.
- To work around the combined lack of grid data and sporadic access to smart meter data, the `node3` sensors and actuators are instead expected to report some `measure_energy_flow` to use as a proxy.
- To facilitate offline development, it includes an additional time series (blue diamond) to allow `node3` to return the `control_energy_flow` in place of the `measure_energy_flow`.
- It does not aggregate energy data. This is instead addressed in Section 3.1.2 and Section 4.2.
- It is necessary to use different `node0` and `node3` counterparts depending on whether the purpose is development, DIMOSIM or production. This is done by means of Docker Compose.
- While `node1` (the Cluster Node) encompasses details pertaining to the nDSM solution, the corresponding details are absent from `node2` (the Edge Node) as they fall outside the concern of the Cluster Node.
- The example has one broker per dialogue between a parent node and its children, but the solution permits different brokers to be merged with preserved semantics.

As mentioned above, in this report, the Edge Node only covers the NODA solution. That said, the two solutions are easily coordinated by means of the mechanism of Section 1.

### 3.1.1. Containers

`node0`: Mock-up of the environment and responsible for weather forecasts.

`node0_control`: Top-level MQTT broker.

`node1_control`: Cluster Node MQTT broker, likely to be superseded by `node0_control`.

`node1_control_publisher`: Modelers responsible for control, and MQTT publisher; scheduled.

`node1_control_subscriber`: MQTT client; event driven.

`node1_measure`: PostgreSQL database.

`node1_measure_publisher`: Modelers, and MQTT client; scheduled.

`node1_measure_subscriber`: MQTT client; event driven.

`node2_control`: Edge Node MQTT broker, likely to remain internal to the Edge Node.

`node2_control_publisher`: Modelers responsible for control, and MQTT client; scheduled.

`node2_control_subscriber`: MQTT client; event driven.

`node2_measure`: PostgreSQL or MySQL database depending on the development outlined in Section 4.2.

`node2_measure_publisher`: Modelers, and MQTT client; scheduled.

`node2_measure_subscriber`: MQTT client; event driven.

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

`node3`: Mock-up of the BRIG and responsible for returning patches under the `node3/measure` topic.

Note that the container `node2_control_subscriber` contains the module `node1.control.subscriber`, that the container `node1_measures_subscriber` contains the module `node2.measure.subscriber`, etc. This adheres to the patten of locating the code for subscribing to a topic together with the code responsible for producing and publishing the corresponding messages in the first place.

The contains are here used to organize the content in event driven and scheduled processes, but the content can be organized differently. For example, the MQTT library `paho.mqtt` permits the MQTT client to run in the background, so it is possible to run subscribers together with modelers and publishers. Conversely, it is also possible to run modelers on separate threads or move them to separate containers.

The container names have been chosen to reflect the longest common namespace or somehow most indicative function of the corresponding content but are otherwise arbitrary.

### 3.1.2. Data

`{name0}/control_temperature`: Forecasted outdoor temperature [K].

`{name0}/{name1}/control_energy_cost`: The synthetic price signal from D2.1 [·].

`{name0}/{name1}/control_energy_mode`: The nDSM solution energy mode with weighted signals [·].

`{name0}/{name1}/value["ndsm"]["model"]`: Implementation specific serialization.

`{name0}/{name1}/value["noda"]["model"]`: Implementation specific serialization.

`{name0}/{name1}/{name2}/control_energy_flow`: Expected energy flow [W] for testing and evaluation.

`{name0}/{name1}/{name2}/control_temperature`: Setpoint temperature [K].

`{name0}/{name1}/{name2}/measure_energy_flow`: Aggregated energy flow [W].

`{name0}/{name1}/{name2}/value["noda"]["model"]`: Implementation specific serialization.

The absence of a `{name0}/{name1}/{name2}/value["ndsm"]["model"]` is due to the authors' lack of insight into the preferred organization of the nDSM solution on the Edge Node. That said, it is trivial to add new s-nodes for models and time series, and it can be done by without excessive coordination.

The Cluster Node and NODA Edge Node solutions will also use the following data, which were omitted from the example configuration in Figure 1 for pedagogical reasons; see Section 4.2 for further details.

`{node0}/{node1}/control_energy_flow`: Expected energy flow [W] for testing and evaluation.

`{node0}/{node1}/measure_energy_flow`: Aggregated energy flow [W].

`{node0}/{node1}/{node2}/{node3}/measure_energy_flow`: Measured energy flow [W].

### 3.1.3. Topics

`node0/control`: Weather forecast, see Section 4.1.

`node1/control`: Weather forecast, energy cost and energy mode time series data.

`node1/measure`: Currently unused.

`node2/control`: Energy flow and setpoint temperature time series data.

`node2/measure`: Energy flow time series data.

`node3/measure`: Energy flow time series data.

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

As already mentioned in Section 2.1, the solution is highly flexible with respect to the choice of topics, and their function is more a matter of avoiding unnecessary work and less a matter of managing control flow. However, having the topics reflect who published something works well in both respects, hence the above topic structure. If useful, the topic structure can also be extended with a prefix. Also, the labels `node0`, `node1`, `node2` and `node3` do not matter as long as they are distinct.

### 3.1.4. Modules

`node0.control.publihser`: Factory for `DefaultClient`.

`node0.control.subscriber`: Factory for `DefaultClient`; used by `node1_control_subscriber`.

`node1.control.ndsm_modeller`: Functions for using …`/{name1}/value["ndsm"]["model"]`.

`node1.control.noda_modeller`: Functions for using …`/{name1}/value["noda"]["model"]`.

`node1.control.publisher`: Factory for `DefaultClient`.

`node1.control.subscriber`: Factory for `DefaultClient`; used by `node2_control_subscriber`.

`node1.measure.ndsm_modeller`: Functions for estimating …`/{name1}/value["ndsm"]["model"]`.

`node1.measure.noda_modeller`: Functions for estimating …`/{name1}/value["noda"]["model"]`.

`node1.measure.publisher`: Factory for `DefaultClient`.

`node1.measure.subscriber`: Factory for `DefaultClient`; used by `node0`.

`node2.control.ndsm_modeller`: Functions for using …`/{name2}/value["ndsm"]["model"]`.

`node2.control.noda_modeller`: Functions for using …`/{name2}/value["noda"]["model"]`.

`node2.control.publisher`: Factory for `DefaultClient`.

`node2.control.subscriber`: Factory for `DefaultClient`; used by `node3`.

`node2.measure.ndsm_modeller`: Functions for estimating …`/{name2}/value["ndsm"]["model"]`.

`node2.measure.noda_modeller`: Functions for estimating …`/{name2}/value["noda"]["model"]`.

`node2.measure.publisher`: Factory for `DefaultClient`.

`node2.measure.subscriber`: Factory for `DefaultClient`; used by `node1_measure_subscriber`.

`node3.measure.publisher`: Factory for `DefaultClient`.

`node3.measure.subscriber`: Factory for `DefaultClient`; used by `node2_measure_subscriber`.

Note the pattern of locating the code for subscribing to a topic together with the code responsible for producing and publishing the corresponding messages in the first place, even though it will be used by a different node.

The modelers are responsible for the algorithmic work. Everything else is configurations, MQTT clients, and scheduling, with https://github.com/dbader/schedule a convenient solution for the latter. And while manageable, the large number of parts are inconvenient to work with, and the situation is made worse by the need to test different configurations against different endpoints.

The modules contain code as well as container files. While not ideal, the approach simplifies development.

## 3.2. Alternative Configurations

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

There are plenty of other configurations. Fortunately, several relevant configurations can be understood as the result of a few independent choices.

- The example configuration subscribes to a broker for weather forecasts. However, the `node0.control.subscriber` can be replaced by a `node.control.weatherforecaster`, scheduled to retrieve weather forecasts through some REST API. The container `node0` and its content as well as `node1.meassure.publisher` can then be discarded. For testing, it makes sense to secure a way to synchronize between `node1.measure.nDSM_modeller` and the weather forecaster, for example, by placing them in the same container and using the callback mechanism of DefaultClient.
- For *x* in {`node1`, `node2`} and *y* in {`control`, `measure`}, put *x*.*y*.`subscriber`, *x*.*y*.`noda_modeller`, *x*.*y*.`nDSM_modeller` and *x*.*y*.`publisher` in the same container and synchronize the control flow between *x*.*y*.`subscriber` and *x*.*y*.`noda_modeller` using the callback mechanism. Combined with a mechanism for keeping track of the expected number of `node3/measure` and `node2/measure` messages, this makes it possible to run the entire system in a synchronous way. This configuration is useful for testing.
- The COLLECTiEF approach promotes edge computing. However, until the necessary equipment becomes more affordable to deploy, cloud computing is likely to remain economically advantageous. For such a setup, discard
    - `node1_control`, `node1_measure_subscriber`, `node1.control.publisher`,
    - `node2_measure`, `node2_control_subscriber`, `node2.measure.publisher`,

  and containerize
    - `node2.measure.noda_modeler`, `node2.measure.nDSM_modeler`,
    - `node1.measure.noda_modeler`, `node1.measure.nDSM_modeler`,
    - `node1.measure.publisher`,
    - `node1.control.noda_modeler`, `node1.control.nDSM_modeler`,
    - `node2.control.noda_modeler`, `node2.control.nDSM_modeler`,
    - `node2.control.publisher`

  into one or two containers connected to the `node1_measure` database.
- The example configuration includes `{name0}/{name1}/{name2}/control_energy_flow` to facilitate offline development. For online development, replace `node3` with a container for integrating with DIMOSIM, some equivalent solution, or the sensors and actuators of some test facility. Note that it makes sense to retain something like `control_energy_flow` for evaluation.

And to just exercise the algorithms, consider discarding the containers as well as everything related to communication, setup a virtual environment, and execute the algorithms against a local database populated with test data. While testing this kind of control solutions remains complex, this approach makes it easier to manage.

While some alternatives like the suggested `node0.control.weatherforecaster` fits well into the current module structure, is nevertheless necessary to keep track of auxiliary code for combining it with the components of the container `node1_control_publisher` into a new container. This need is even more

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

pronounced for the other alternatives, and of particular importance when integrating with an external system conforming to someone else's design.

To avoid a mess, consider introducing new top-level modules `myalternative0`, `myalternative1`, `myalternative2` and `myalternative3` for code and container files, and a corresponding top-level file `myalternative.docker-compose.yml` for composing the solution. The top-level modules can then import form `node0`, `node1`, `node2` and `node3` and other alternative modules while retaining the overall organization into subscribers, modelers, and publishers. For example, to integrate with DIMOSIM, create a module `mydimosim3` for the code and a `mydimosim.docker-compose.yml` to compose the solution along the lines of some other configuration, replacing the counterpart of container `node3` with a container `mydimosim3`.

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

# 4. Integration with other Components

The Cluster Node is intended to be used together with some service providing weather forecasts and some sensors and actuators augmenting the Edge Node such as the BRIG. To integrate with these, or with DIMOISM, it is necessary to replace `node0_control_subscriber`, `node2.control.publisher` and `node3_measure_subscriber` with solutions providing the desired functionality. This section is mainly concerned with the details of such work. For the organization of such work, consider following the approach outlined at the end of Section 3.2, placing the new code and the new container files in new top-level modules such as myopenweathermap0 and mydimosim3.

## 4.1. Weather Forecasts

The COLLECTiEF consortium is leaning towards OpenWeatherMap for weather forecasts, see D2.1. Most API methods serve JSON, and the data is organized in a what that is easy to understand. For the COLLECTiEF solution, the most relevant variables seem to be

- Cloudiness
- Humidity
- Temperature (actual, felt, min, max)
- Wind (speed, direction)

Among these, the Cluster Node solution and the NODA Edge Node solution are currently only using the actual temperature in the form of `node0/control_temperature`, but future development of the Cluster Node may benefit from also utilizing cloudiness, wind speed and humidity in that order. The impact of wind direction in an urban setting seems too complicated to be worth the effort. Future development of the Edge Node may instead emphasize humidity and its impact on the perceived indoor climate.

OpenWeatherMap weather forecasts are being published to the project main MQTT broker. It remains to implement a module `openweather0` with a subscriber for updating the database with the corresponding time series data.

## 4.2. BRIG

The BRIG does, among other things, collect energy data from sources the corresponding building and periodically packages the data into a JSON array, which it publishes to the project main MQTT broker under a topic containing the BRIG UUID, see D2.1.

This energy data falls outside Section 3.1.2. It remains to implement a module `brig3` for updating the database with the corresponding `{name0}/{name1}/{name2}/{name3}/measure_energy_flow` time series data, and to refine module `node2` with support for aggregating energy data. In fact, module `node1` should also be refined with support for aggregating energy data from …`/{name2}/measure_energy_flow` into …`/{name1}/measure_energy_flow` time.

Note that these constructions are essential but were omitted from the example configuration in Figure 1 for pedagogical purposes because they cluttered the graphics and made the big picture difficult to convey.

To aggregate energy data, it is necessary to first interpolate any gaps in the data. This can be challenging, and for the current setup, this challenge occurs for the Cluster Node and Edge Node alike. This is also the reason NODA recommends the use of grid energy data and smart meter energy data rather than in-house solutions. How well the latter will work in practice remains to be seen.

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683

The BRIG uses MySQL rather than PostgreSQL and a different table structure than the one in Table 1, which constitute an obstacle to the NODA Edge Node solution. The current version of minimas.measure factors over a typing.Protocol in a way that abstracts over whether it is connected to a MySQL or a PostgreSQL database, but it remains to take the different table structures into account. However, it is feasible to keep the changes local and keep the minimas.measure interface for the rest of the codebase.

## 4.3. DIMOSIM

While desirable, there is currently no support for translating between the data model of `minimas.measure` and the data model of DIMOSIM. Consequently, to integrate with DIMOSIM, it is necessary to provide a scenario-specific mapping between DIMOSIM parameters and variables, and `minimas.measure` control and measure time series.

To integrate with DIMOSIM, consider packaging the general parts of the work in a module `dimosim3` and packaging the scenario-specific parts of the work in a module `myscenario3` and a corresponding file `myscenario.docker-compose.yml` to compose the solution.

This project has received funding from the European Union's H2020 research and innovation programme under Grant Agreement No 101033683