

# A new distributed data analysis framework for better scientific collaborations

**Philipp S. Sommer**  
Institut für Küstensysteme –  
Analyse und Modellierung

EGU21-1614, 28.04.2021



Helmholtz-Zentrum  
**hereon**

# The problem of analyzing distributed data

Philipp S. Sommer, Viktoria Wichert, Daniel Eggert, Tilman Dinter, Klaus Getzlaff, Andreas Lehmann, Christian Werner, Brenner Silva, Lennart Schmidt, and Angela Schäfer

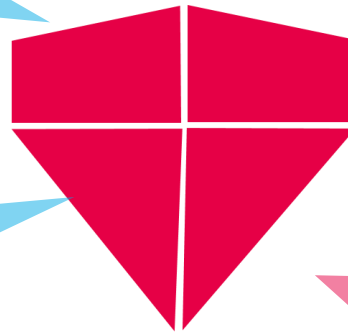
**Let's do some research together!**

**Awesome! Let me run some tests on your model data.**

**Alright! But that's too much data and ...**

**I can't give you access to my super computer!**

**No worries, I do my analysis with python, so let's just use the de-messaging-python framework!**



# Presentation material and Code

<https://github.com/Chilipp/de-messaging-python-presentation-20210428>



# AK Datenanalyse

Distributed data analysis Working Group within Datahub

## Contributors

- **HZG:** Philipp S. Sommer, Viktoria Wichert
- **GFZ:** Daniel Eggert (Digital Earth)
- **AWI:** Tilman Dinter, Brenner Silva, Angela Schäfer
- **Geomar:** Klaus Getzlaff, Andreas Lehmann
- **KIT:** Christian Werner
- **UFZ:** Lennart Schmidt



# What is distributed Data analysis

## Examples

### Ship campaign

- Sonne (Geomar) and Ludwig Prandtl (HZG) measure real-time-data in a campaign.
- Sonne sends to internal area of Geomar, Ludwig Prandtl to HZG.
- How can people from HZG access and analyze the data at Geomar?

### Model simulations

- Compare a COSMO-CLM-Simulation (HZG) with output of the Baltic Sea Model (Geomar)
- And with ship measurements
- How to share terra-bytes of data?
- How to get the latest version?

# It's about *analyzing* distributed data

## The ideal world

- We all have one single big cloud
  - Run model simulations in the cloud
  - Store NRT data in the cloud
- Post processing and data analysis runs in the cloud
- Someone from HZG needs access to data from Geomar? *Just grant it.*

## The real world

- We have many different clusters.
  - Every center (or even every scientist) has different requirements
  - We are behind VPNs
  - Each center has his own cluster for processing, storage, etc.
- Someone from HZG needs access to data from Geomar? *Ok, I upload it to Dropbox.*

# Can we do it without the cloud?

## What we need:

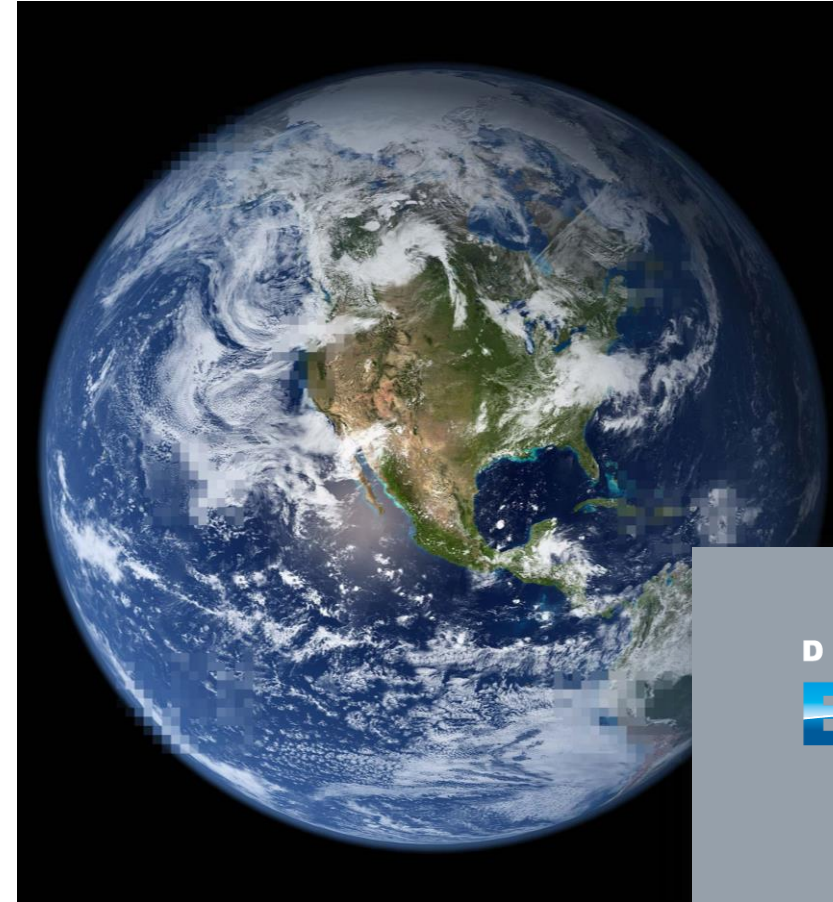
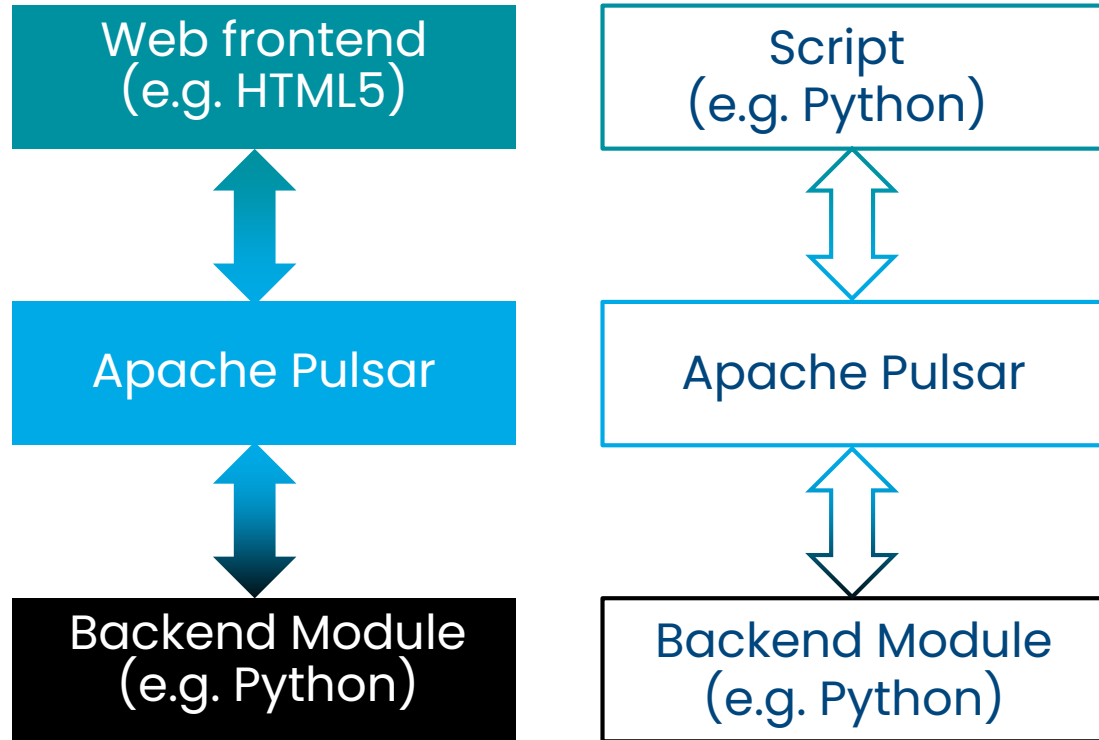
- Access to data in another research center
- Access to computing power in another research center

## And:

- It must be safe
- It must be easy

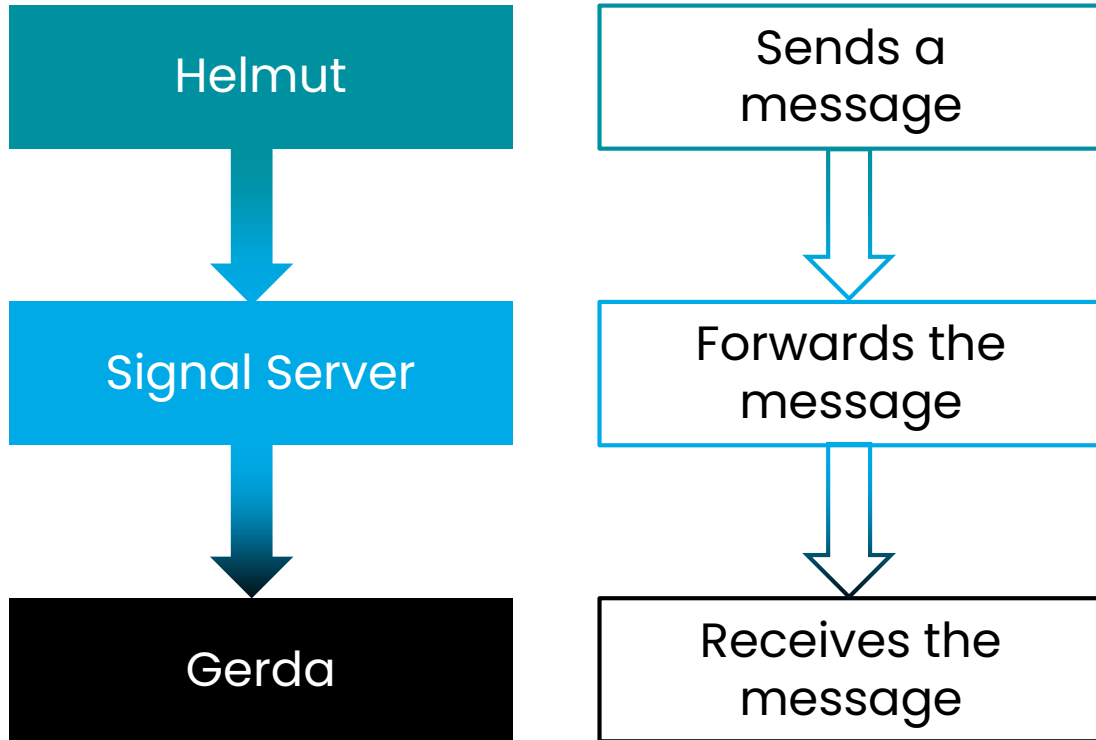
# We are not the first

with this idea

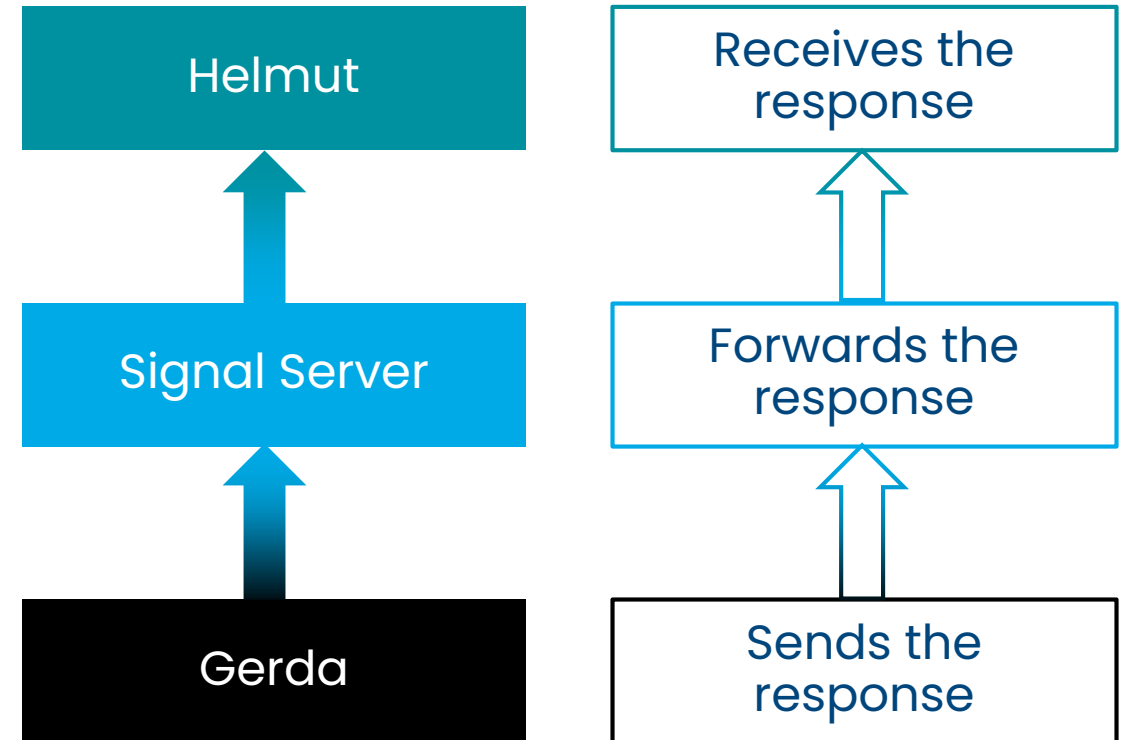


# Just like ~~WhatsApp~~Signal

## Request

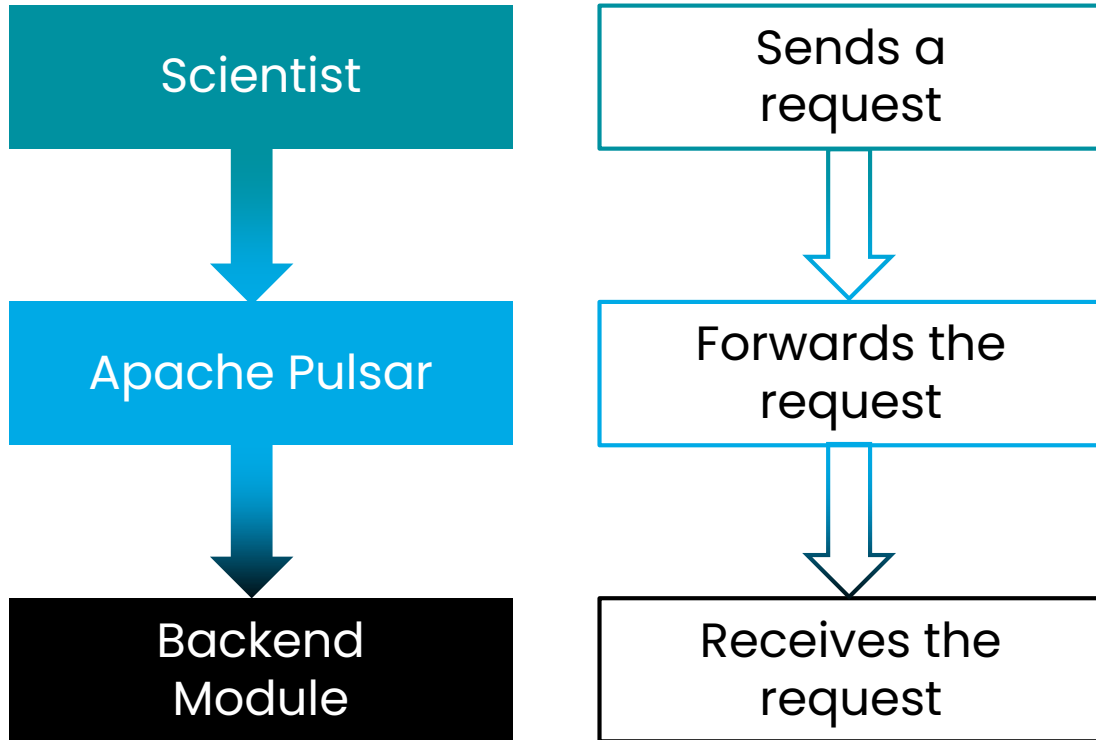


## Response

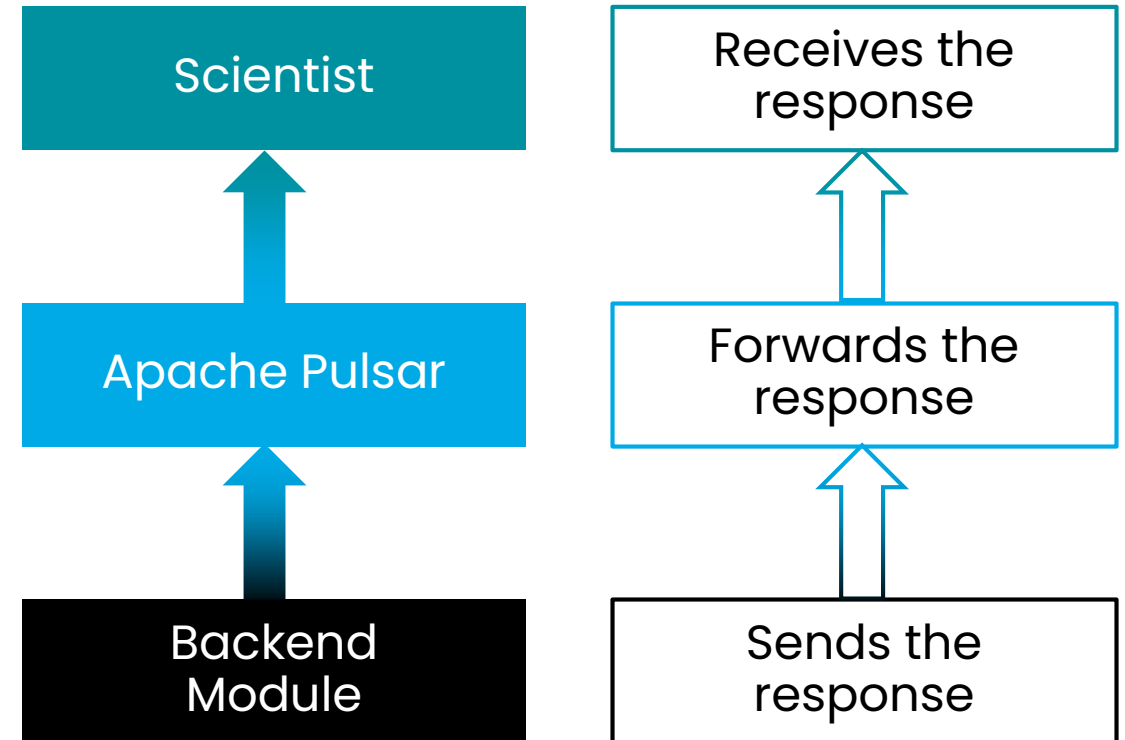


# Just like ~~WhatsApp~~Signal

## Request



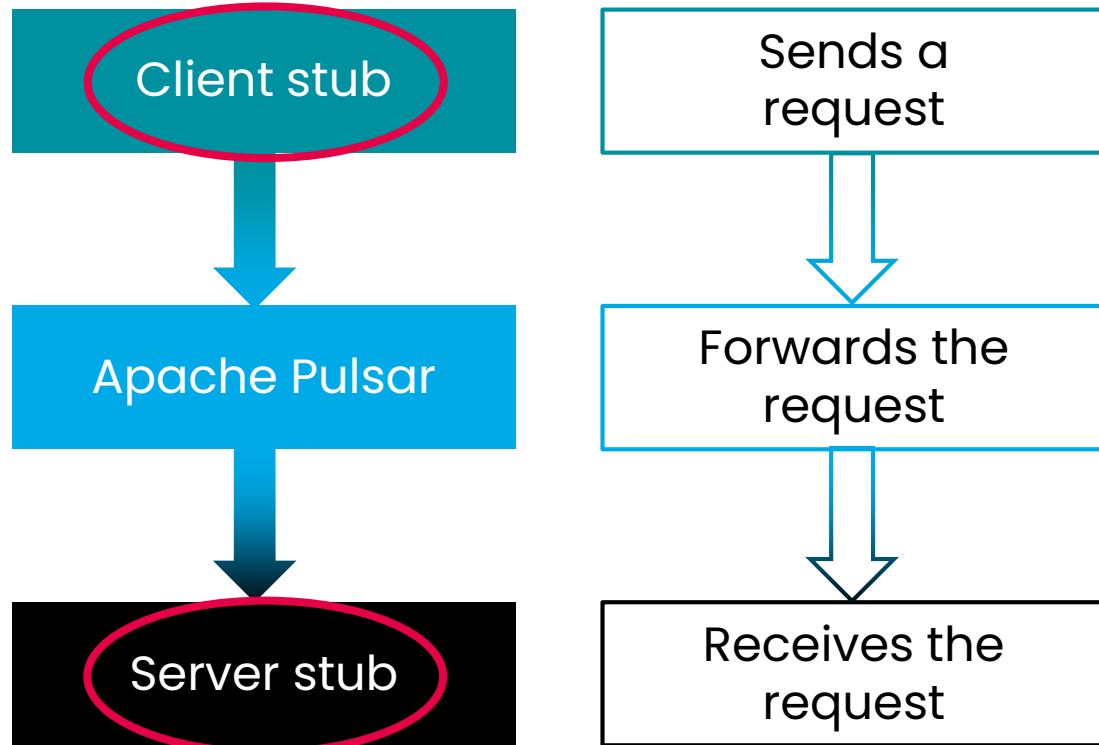
## Response



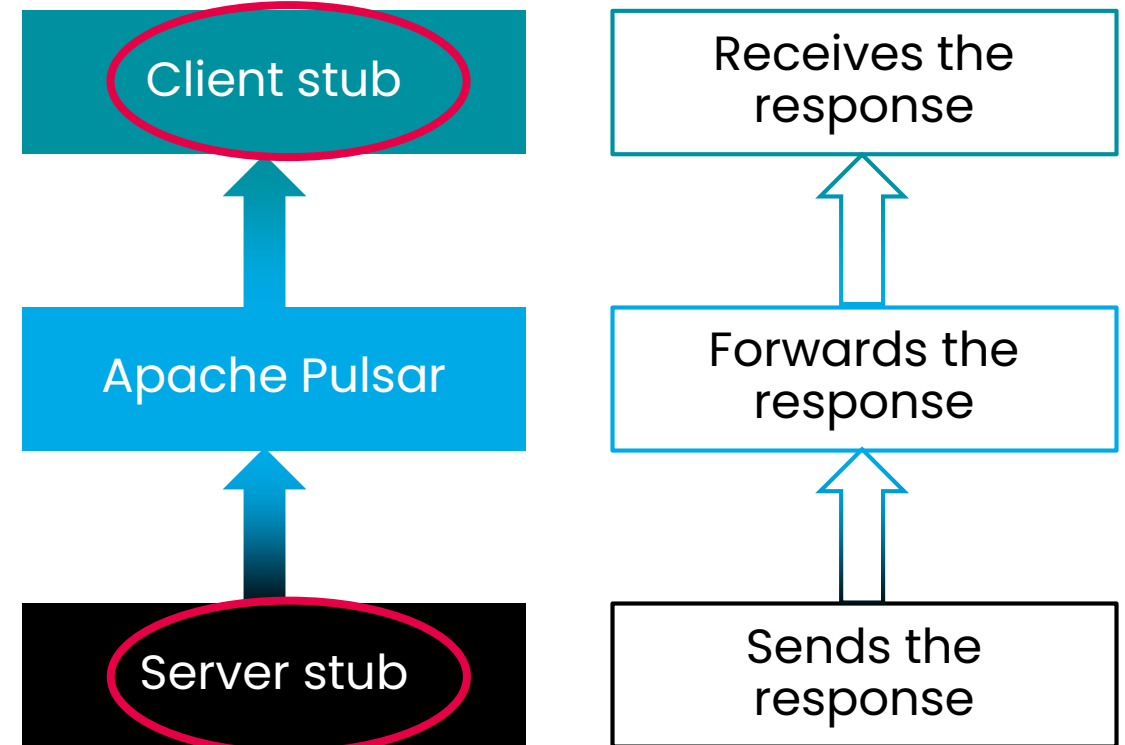
# Just like ~~WhatsApp~~Signal

A Remote Procedure Call (RPC)

## Request



## Response



# Pros and Cons

## Advantages

- Scientist can simply send a request and retrieve the response on any other machine
- Backend Module can run everywhere, not necessarily on a dedicated web server (e.g. on the cluster)

## Disadvantages

- Scientists are not familiar with web requests (nor are the backend module developers)
- Request needs serialization (transformation to JSON)
- Potential vulnerability for internal computing resources
- Scientists do have better stuff to do

# Be nice

and do not add more work

## Use the scientists methods

- abstract standard python functions and classes into web requests
- everything's basic python, (almost) no need for special stuff
- Client stub is automatically generated
- Requests are abstracted and standardized (JSONschema)

```
from demessaging import main

def compute_sum(
    da: demessaging.types.xarray.DataArray,
) -> demessaging.types.xarray.DataArray:
    """
    Compute the sum over a data array.

    Parameters
    -----
    da : DataArray
        The input data array

    Returns
    -----
    DataArray
        The sum of the data array
    """
    request = {
        "member": {
            "func_name": "compute_sum",
            "da": da,
        }
    }

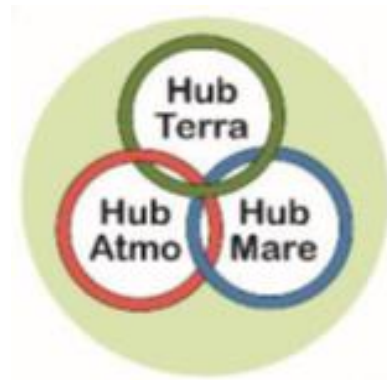
    model = BackendModule.parse_obj(request)
    model.compute()

    return model.member.func_returns # type: ignore
```

# de-messaging-python

## Summary

- Remote Procedure Call
- High-level API to easily create server and client stubs
- Very close to scientists common workflows



## Outlook

- More effort into security
  - User management for backends
  - End-to-End encryption
- How to handle large amounts of data
- We are looking for use cases and project that may use our framework!

# Thank you!

# Vielen Dank.

[www.hereon.de](http://www.hereon.de)



# basic

April 28, 2021

## 1 Basic example for de-messaging-python

*Philipp S. Sommer*

Supplementary notebook to EGU21-1614, April, 28th 2021.

### 1.1 A very basic backend module

Take a netCDF variable and computes the sum.

```
[1]: cat > backend.py << EOF

from demessaging import main
from demessaging.types.xarray import DataArray

def compute_sum(da: DataArray) -> DataArray:
    """Compute the sum over a data array.

    Parameters
    -----
    da : DataArray
        The input data array

    Returns
    -----
    DataArray
        The sum of the data array
    """
    return da.sum()

if __name__ == "__main__":
    main(topic="hello_world", members=["compute_sum"])

EOF
```

## 1.2 Command line utility

The main function we used above gives us a command line utility

```
[2]: python backend.py --help
```

```
usage: backend.py [-h] [-t TOPIC] [-m MODULE_NAME] [-d DOC] [-H HOST]
                  [-p PORT] [--persistent PERSISTENT] [--tenant TENANT]
                  [--namespace NAMESPACE] [--members member [member ...]]
                  {test-connect,listen,schema,send-request,shell,generate} ...
```

optional arguments:

```
-h, --help            show this help message and exit
-t TOPIC, --topic TOPIC
                        The topic identifier under which to register at the
                        pulsar. Default: hello_world
-m MODULE_NAME, --module MODULE_NAME
                        Name of the backend module. Default: __main__
-d DOC, --description DOC
                        The documentation of the object. If empty, this will
                        be taken from the corresponding ``__doc__`` attribute.
-H HOST, --host HOST  The remote host of the pulsar Default: rz-vm154.gfz-
                        potsdam.de
-p PORT, --port PORT  The port of the pulsar at the given :attr:`host`
                        Default: 8082
--persistent PERSISTENT
                        Default: non-persistent
--tenant TENANT        Default: public
--namespace NAMESPACE
                        Default: digital-earth
--members member [member ...]
                        List of members for this module
```

Commands:

```
{test-connect,listen,schema,send-request,shell,generate}
test-connect          Connect the backend module to the pulsar message
                        handler.
listen                Connect the backend module to the pulsar message
                        handler.
schema                Print the schema for the backend module.
send-request          Test a request via the pulsar messaging system.
shell                 Start an IPython shell
generate              Generate an API module
```

## 1.3 Command line utility

We can use it to connect to the Apache pulsar

```
[3]: python backend.py listen &
```

[1] 43487

and to generate the client stub

```
[4]: python backend.py generate
```

```
connection to ws://rz-vm154.gfz-potsdam.de:8082/ws/v2/consumer/non-
persistent/public/digital-earth/hello_world/backend-module-2021-04-28T12:30:51
established
waiting for incoming request
"""
```

```
"""
```

```
from typing import Callable
```

```
import demessaging.types.xarray
```

```
from demessaging import BackendModule as _BackendModule
```

```
from demessaging import main
```

```
from demessaging.config import ModuleConfig
```

```
__all__ = ["compute_sum"]
```

```
def compute_sum(
```

```
    da: demessaging.types.xarray.DataArray,
```

```
) -> demessaging.types.xarray.DataArray:
```

```
    """
```

```
    Compute the sum over a data array.
```

```
    Parameters
```

```
    -----
```

```
    da : DataArray
```

```
        The input data array
```

```
    Returns
```

```
    -----
```

```
    DataArray
```

```
        The sum of the data array
```

```
    """
```

```
    request = {"member": {"func_name": "compute_sum", "da": da}}
```

```
    model = BackendModule.parse_obj(request)
```

```
    model.compute()
```

```
    return model.member.func_returns # type: ignore
```

```
backend_config = ModuleConfig.parse_raw(
```

```

"""
{
    "host": "rz-vm154.gfz-potsdam.de",
    "port": "8082",
    "persistent": "non-persistent",
    "tenant": "public",
    "namespace": "digital-earth",
    "topic": "hello_world"
}
"""
)

_creator: Callable
if __name__ == "__main__":
    _creator = main
else:
    _creator = _BackendModule.create_model

BackendModule = _creator(
    __name__,
    config=backend_config,
    class_name="BackendModule",
    members=[compute_sum],
)

```

## 1.4 Command line utility

If we pipe this into a file, we generate a module that we can import.

```
[5]: python backend.py generate > api.py
```

## 1.5 Let's try it

We defined `compute_sum`, to take xarray `DataArray`.

```
def compute_sum(da: DataArray) -> DataArray:
    """Compute the sum over a data array.
```

*Parameters*

-----

*da : DataArray*

*The input data array*

*Returns*

-----

*DataArray*

*The sum of the data array*

```

"""
return da.sum()

```

```

[6]: python -- << EOF

import xarray as xr
from api import compute_sum

summed = compute_sum(xr.DataArray([1, 2, 3]))

print('-----')
print(summed)

EOF

```

```

connection to ws://rz-vm154.gfz-potsdam.de:8082/ws/v2/producer/non-
persistent/public/digital-earth/hello_world established
connection to ws://rz-vm154.gfz-potsdam.de:8082/ws/v2/consumer/non-
persistent/public/digital-earth/hello_world_MoSd0TV0/python-
backend-2021-04-28T12:30:54.556100 established
[2021-04-28 12:30:54.619954] processing request
connection to ws://rz-vm154.gfz-potsdam.de:8082/ws/v2/producer/non-
persistent/public/digital-earth/hello_world_MoSd0TV0 established
request successful
-----
<xarray.DataArray ()>
array(6)

```

## 1.6 Let's try it

We can also load netCDF files from the disk and send them via the web

```

[7]: python -- << EOF

import xarray as xr
from api import compute_sum

ds = xr.open_dataset('demo.nc')
summed = compute_sum(ds.t2m[0])

print('-----')
print(summed)

EOF

```

```

connection to ws://rz-vm154.gfz-potsdam.de:8082/ws/v2/producer/non-
persistent/public/digital-earth/hello_world established
connection to ws://rz-vm154.gfz-potsdam.de:8082/ws/v2/consumer/non-

```

```
persistent/public/digital-earth/hello_world_pLoNEXbx/python-  
backend-2021-04-28T12:30:56.299857 established  
[2021-04-28 12:30:56.629695] processing request  
connection to ws://rz-vm154.gfz-potsdam.de:8082/ws/v2/producer/non-  
persistent/public/digital-earth/hello_world_pLoNEXbx established  
request successful  
-----  
<xarray.DataArray 't2m' ()>  
array(5142918.25942993)  
Coordinates:  
    lev      float64 1e+05  
    time      <U19 '1979-01-31T18:00:00'
```