Jupyter Notebooks based Reporting

How to use Jupyter Notebooks to deliver fancy html reports and give freedom to the user

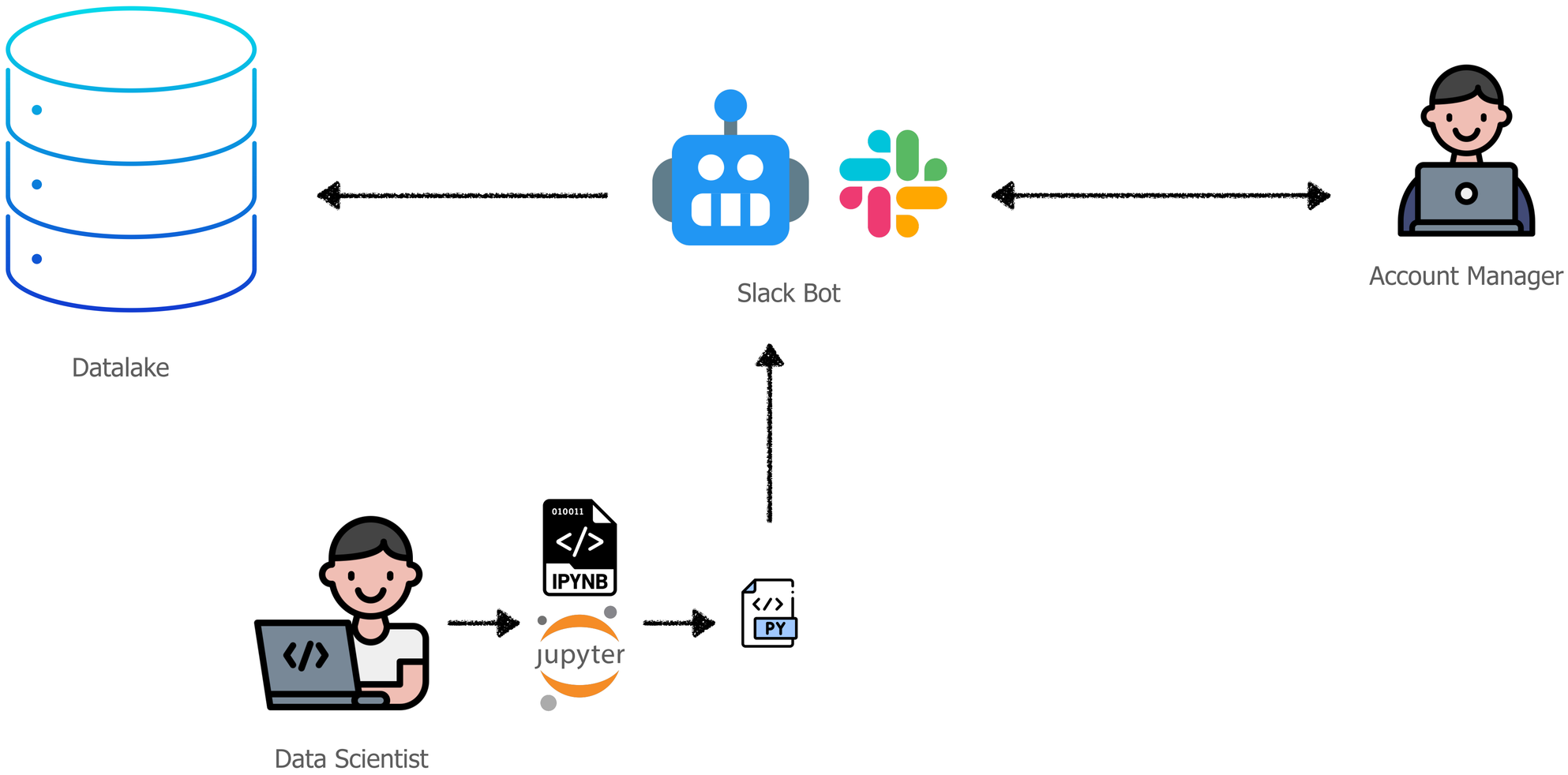
Python Meetup BCN - Xavier Orduña - Feb 2025

About me

- I learned to code in the last century (HTML, Visual Basic …)
- Computer Science at UPC
- Founded DEXMA (2007 - 2015)
- Data engineering freelance (TESCO, ABI, Moonpay, …) (2015 - 2021)
- Team lead at Circutor (2021 - present)

- Obsessed about automation, product and platform!

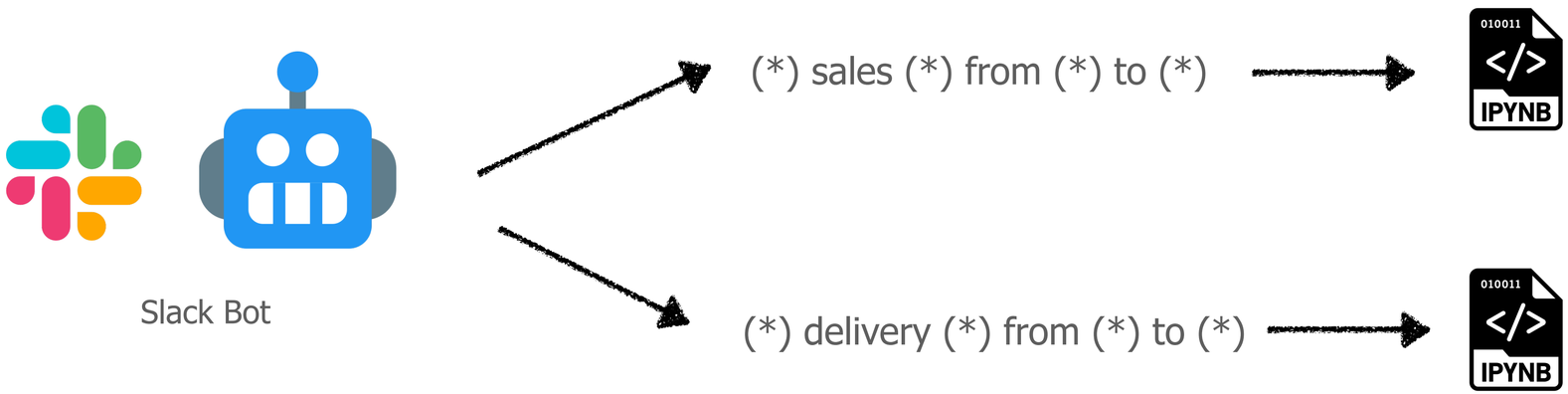
Get back 8 years



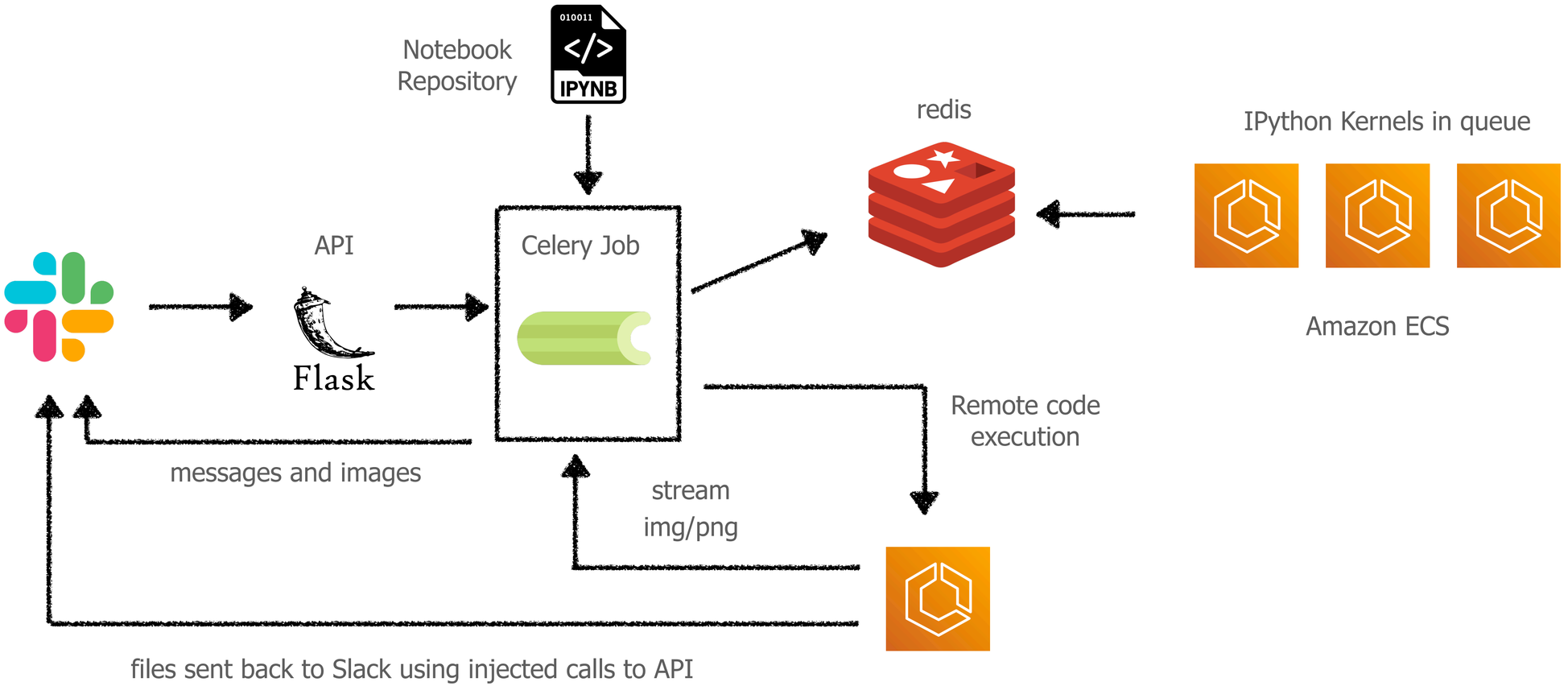
Problems

- Data scientist used to create reports (xls) and charts (png) using Jupyter Notebook
- Development cycle was very slow
- We needed a way to run isolated “semi trusted code”

Solution

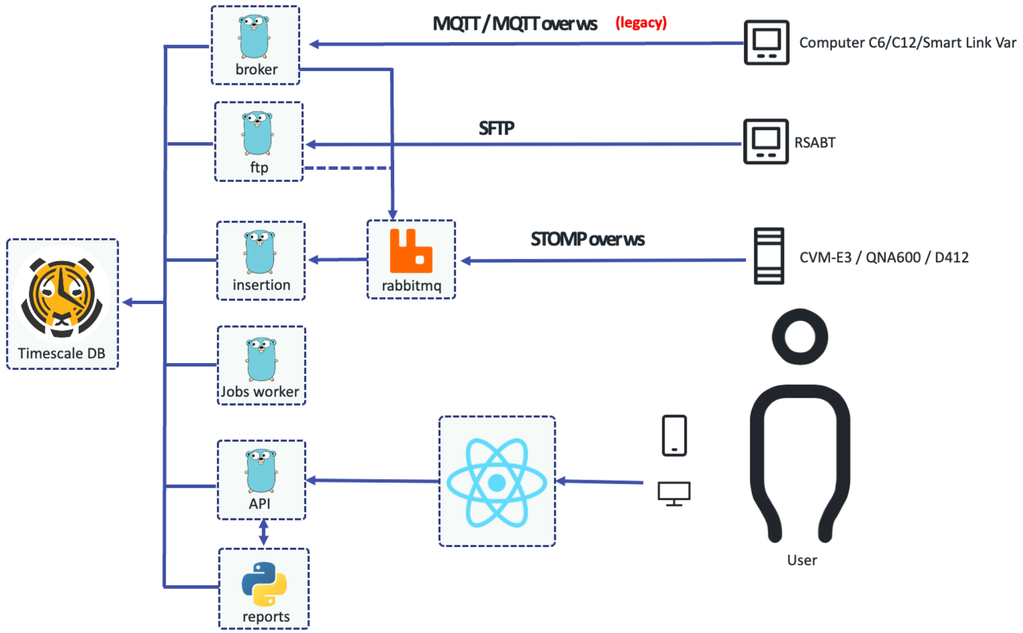

Slack Bot
(*) sales (*) from (*) to (*)
(*) delivery (*) from (*) to (*)
010011
</>
IPYNB
010011
</>
IPYNB

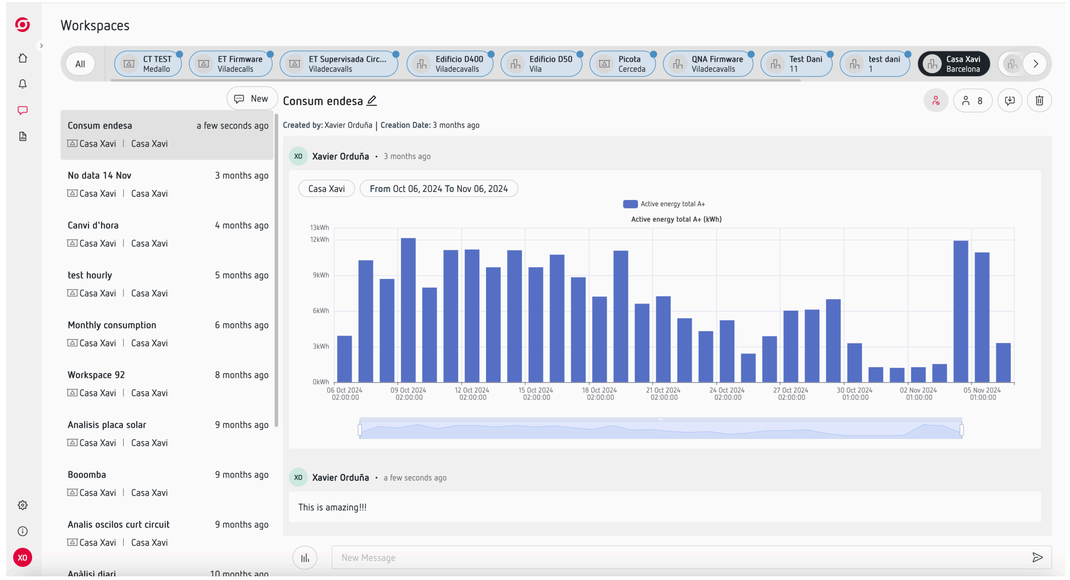
Technical solution



MISSING PYTHON NOTEBOOK

Circutor Architecture





IOT Platform for a variety of Hardware for enegy efficency






Circutor Needs

- SaaS IOT Platform
- How to customize SaaS for industrial customers?
- Development lifecycle
- Developer independency
- Self contained
- Need to run “untrusted” code

Ingredients

- Python Notebook
- Jinja2 HTML Template

```jinja2
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>This is a title</title>
</head>
<body style="font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, 'Helvetica Neue', Arial, sans-serif; line-height: 1.6;
    <div style="max-width: 800px; margin: 0 auto; background-color: white; padding: 30px; border-radius: 8px; box-shadow: 0 2px 4px rgb

        <h1 style="color: #0d6efd; margin-bottom: 20px; font-weight: 500;">Asset Report</h1>

        <div style="background-color: #e9ecef; padding: 15px; border-radius: 6px; margin-bottom: 20px;">
            <p style="margin: 0 0 10px 0;">My String: <span style="font-weight: 500;">{{ my_string }}</span></p>
            <p style="margin: 0 0 10px 0;">Asset ID: <span style="font-weight: 500;">{{ asset_id }}</span></p>
            <p style="margin: 0 0 10px 0;">Time Range: <span style="font-weight: 500;">{{ start_ts }} to {{ end_ts }}</span></p>
            <p style="margin: 0;">Date Range: <span style="font-weight: 500;">{{ from_day }} to {{ to_day }}</span></p>
        </div>

        <h2 style="color: #495057; margin: 30px 0 20px 0; font-size: 1.5rem;">{{ _('Environment Variables') }}</h2>

        <div style="overflow-x: auto;">
            <table style="width: 100%; border-collapse: collapse; margin-bottom: 1rem; background-color: transparent;">
                <thead>
                    <tr style="background-color: #0d6efd; color: white;">
                        <th style="padding: 12px 15px; text-align: left; border-top: 1px solid #dee2e6;">Key</th>
                        <th style="padding: 12px 15px; text-align: left; border-top: 1px solid #dee2e6;">Value</th>
                    </tr>
                </thead>
                <tbody>
                    {% for key, value in env.items() %}
                    <tr style="border-bottom: 1px solid #dee2e6;">
                        <td style="padding: 12px 15px;">{{ key }}</td>
                        <td style="padding: 12px 15px;">{{ value }}</td>
                    </tr>
                    {% endfor %}
                </tbody>
            </table>
        </div>
    </div>
</body>
</html>
```

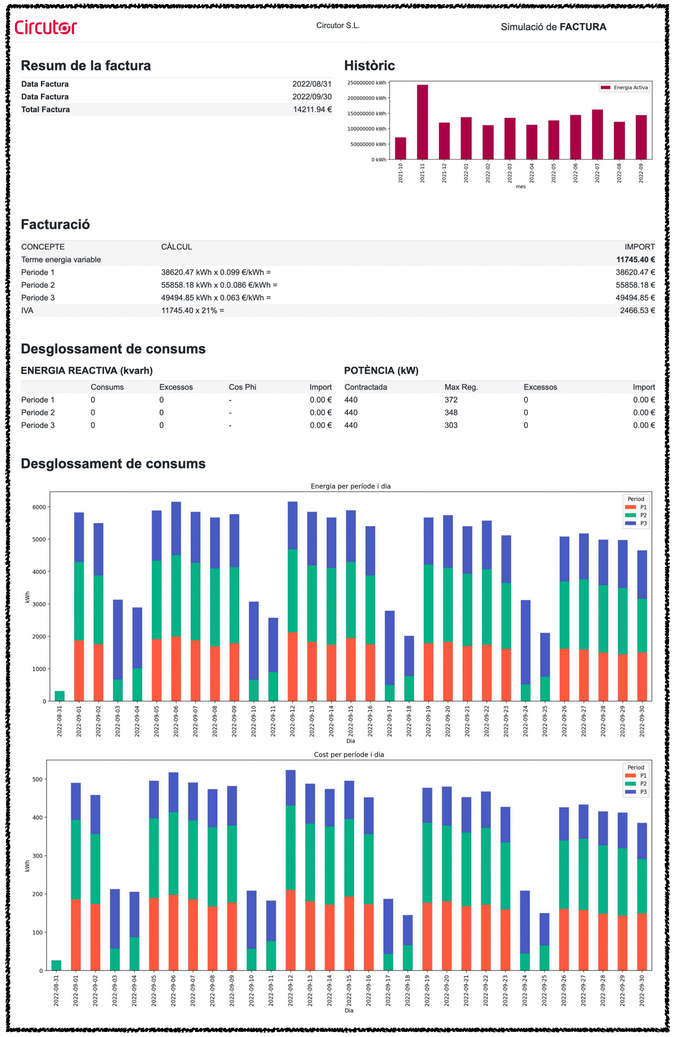
```
In _  1  import json, os, requests
```

```
In _  1  #Environment
      2  asset_id = "$ASSET_ID"
      3  from_day = "$FROM_DAY"
      4  to_day = "$TO_DAY"
      5  language = "$LANGUAGE"
      6  token = "$TOKEN"
```

```
In _  1  # Get some data
      2  url = "https://mydata.com/{asset_id}?from={from_day}&to={to_day}".format(
      3      asset_id=asset_id,
      4      from_day=from_day,
      5      to_day=to_day
      6  )
      7  data = requests.get(url, headers=token)
      8  # Do some processing
      9  title = "This is a report for {asset} from {from_day} to {to_day}".format(
     10      asset=asset_id, from_day=from_day, to_day=to_day
     11  )
```

```
In _  1  output = {
      2      'title': title,
      3      'asset_id': asset_id,
      4      'from_day': from_day,
      5      'to_day': to_day,
      6      'env': dict(os.environ)
      7  }
      8  with open("output.json", "w") as f:
      9      f.write(json.dumps(output))
```

Examples



Circutor

Circutor S.L. Simulació de **FACTURA**

Resum de la factura

Data Factura	2022/08/31
Data Factura	2022/09/30
Total Factura	14211.94 €

Històric

Facturació

CONCEPTE	CÀLCUL	IMPORT
Terme energia variable		**11745.40 €**
Periode 1	38620.47 kWh x 0.099 €/kWh =	38620.47 €
Periode 2	55858.18 kWh x 0.086 €/kWh =	55858.18 €
Periode 3	49494.85 kWh x 0.063 €/kWh =	49494.85 €
IVA	11745.40 x 21% =	2466.53 €

Desglossament de consums

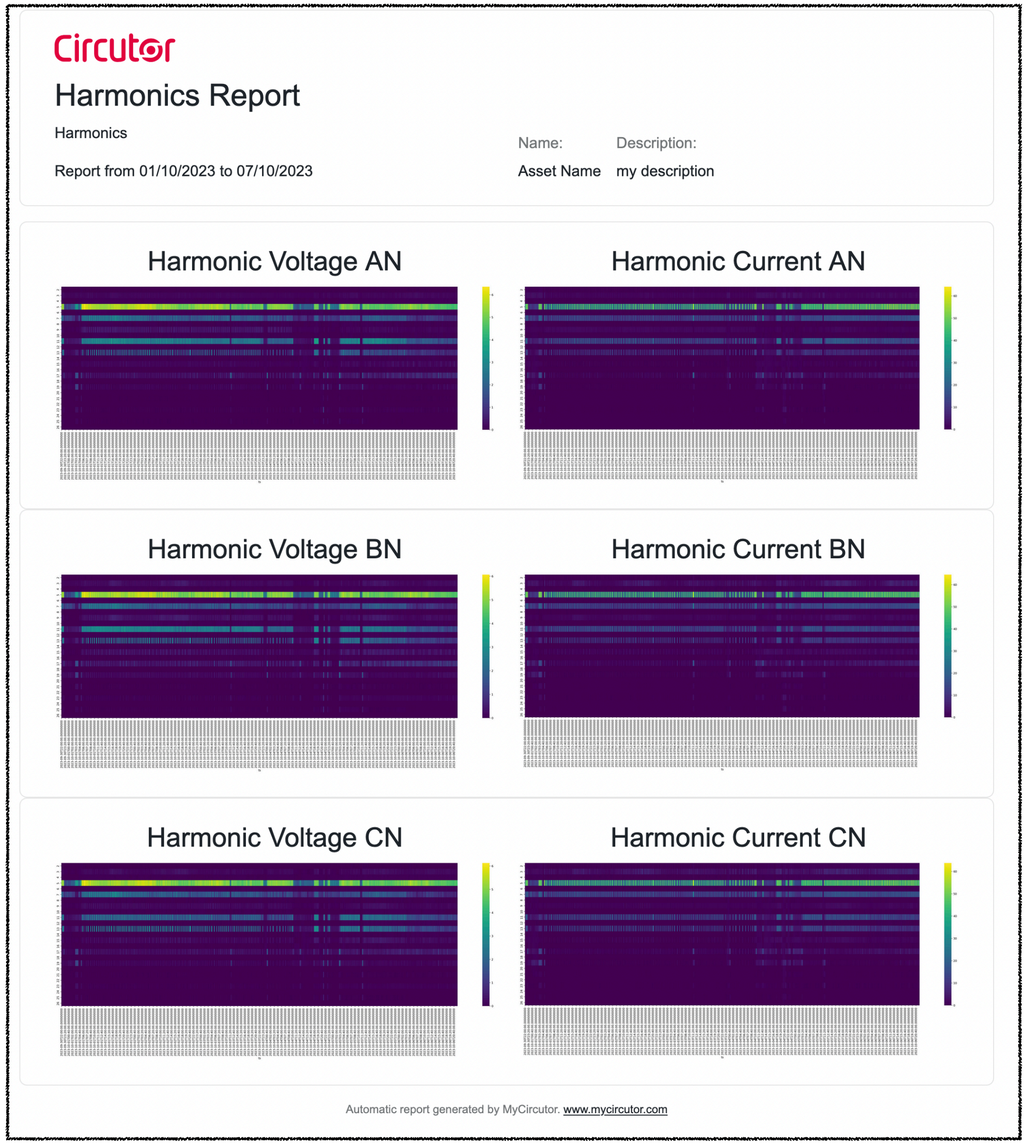
ENERGIA REACTIVA (kvarh)

	Consums	Excessos	Cos Phi	Import
Periode 1	0	0	-	0.00 €
Periode 2	0	0	-	0.00 €
Periode 3	0	0	-	0.00 €

POTÈNCIA (kW)

	Contractada	Max Reg.	Excessos	Import
Periode 1	440	372	0	0.00 €
Periode 2	440	348	0	0.00 €
Periode 3	440	303	0	0.00 €

Desglossament de consums



Circutor

Harmonics Report

Harmonics

Report from 01/10/2023 to 07/10/2023

Name: Asset Name

Description: my description

Harmonic Voltage AN

Harmonic Current AN

Harmonic Voltage BN

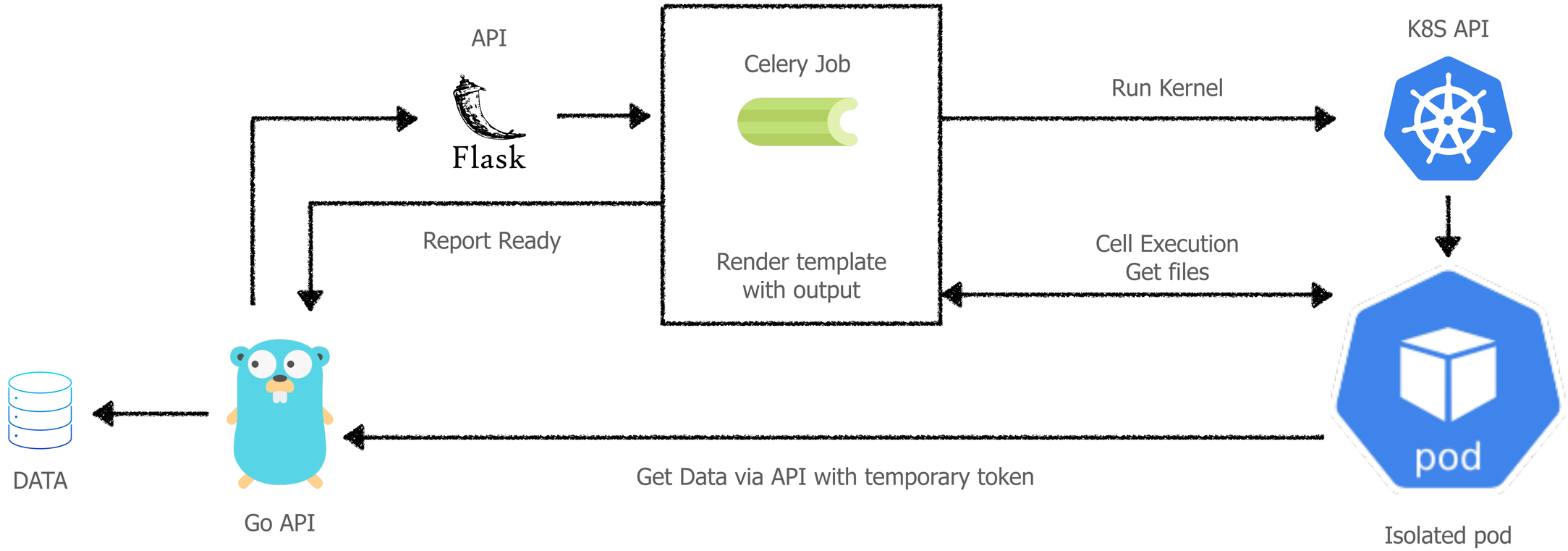
Harmonic Current BN

Harmonic Voltage CN

Harmonic Current CN

Automatic report generated by MyCircutor. www.mycircutor.com

Technical solution



Steps

- **Create report file**
- Get Notebook
- Start Pod
- Get Kernel file
- Connect to pod
- Execute code
- Get output
- Render template
- Push report

- Create temporal Token for each report file
- Set parameters:
  - language
  - asset
  - from (day)
  - to (day)

Steps

- Create report file
- **Get Notebook**
- Start Pod
- Get Kernel file
- Connect to pod
- Execute code
- Get output
- Render template
- Push report

```
- languages
  - en/LC_MESSAGES/messages.mo
  - ca/LC_MESSAGES/messages.mo
  - messages.pot
- report.ipynb
- template.html
- config.json
```

Steps

- Create report
- Get Notebook
- **Start Pod**
- Get Kernel file
- Connect to pod
- Execute code
- Get output
- Render template
- Push report

Kernel Image

```
FROM python:3.12-slim

COPY kernel-requirements.txt /

#RUN apt-get update && apt-get install -y gcc python3-dev -y
RUN pip install --no-cache-dir -r kernel-requirements.txt

ARG BUILD_DATE
ENV PYTHONUNBUFFERED=1

WORKDIR /run
```

```
ipykernel==6.29.5
ipython==8.32.0
Jinja2==3.1.5
matplotlib==3.10.0
numpy==2.2.2
pandas==2.2.3
requests==2.32.3
```

```
apiVersion: v1
kind: Pod
metadata:
  name: python-kernel-<random_string>
spec:
  containers:
    - name: app
      image: kernel-runner:latest
      env:
        - name: foo
          value: bar
      command:
        - python -m ipykernel_launcher --ip=0.0.0.0 --shell=8888 --iopub=8889 --control=8890 \
        - --stdin=8891 --hb=8892 -f /kernel_config.json --Session.key=my-random-key --log-level=DEBUG
      ports:
        - containerPort: 8888
        - containerPort: 8889
        - containerPort: 8890
        - containerPort: 8891
        - containerPort: 8892
```

Steps

- Create report
- Get Notebook
- Start Pod
- **Get Kernel file**
- Connect to pod
- Execute code
- Get output
- Render template
- Push report

Kernel File (secret, ports, etc ..)

Steps

- Create report
- Get Notebook
- Start Pod
- Get Kernel file
- **Connect to pod**
- Execute code
- Get output
- Render template
- Push report

ZeroMQ protocol

Comunication with kernel is done via 5 channels and JSON based messages through ZeroMQ

Channel	Type	Purpose
Shell	request-reply	To run code
Control	request-reply	Like shell, but for control orders like stop the kernel
IOPub	Publish-Subscribe	To publish results, status messages and code output
Stdin	request-reply	To control user input
Heartbeat	ping-reply	To check that the kernel is alive

```
{
  "header": {
    "msg_id": "12345",
    "msg_type": "execute_request"
  },
  "parent_header": {},
  "metadata": {},
  "content": {
    "code": "print('hello meetup!')",
    "silent": false
  }
}
```

Steps

- Create report
- Get Notebook
- Start Pod
- Get Kernel file
- Connect to pod
- **Execute code**
- Get output
- Render template
- Push report

```python
def run_notebook(self, notebook_file: str, parameters: dict, timeout=120):
    """..."""
    logger.info("Running notebook file")

    # Check exist notebook file.
    try:
        with open(notebook_file, "r") as file:
            self.notebook = nbformat.read(file, nbformat.NO_CONVERT)
    except OSError:
        raise NotebookExecutionError(type_error="ExecutionNotebook", message="Not found file notebook")

    try:
        self.iteration_cell_notebook(parameters=parameters, timeout=timeout)
    except NotebookExecutionError as custom_error:
        self.stop_kernel()
        raise custom_error

    self.stop_kernel()

def iteration_cell_notebook(self, parameters: dict, timeout: float):
    """..."""
    for index, cell in enumerate(self.notebook.cells):
        logger.info("reportengine --> execution cell {} ".format(index))
        if cell.cell_type == "code":
            self.kernel_client.execute(command=cell.source, timeout=timeout, index=index)

def stop_kernel(self):
    self.kernel_client.stop_kernel()
```

Steps

- Create report
- Get Notebook
- Start Pod
- Get Kernel file
- Connect to pod
- **Execute code**
- Get output
- Render template
- Push report

Before executing any cell, we look for special “TAGS” or comments

#NOEXECUTE

This cell is skipped (used for local development)

#ENVIRONMENT

Looks for `from_day = “$FROM_DAY”` and replaces `$FROM_DAY` for the value defined for this particular execution

Steps

- Create report
- Get Notebook
- Start Pod
- Get Kernel file
- Connect to pod
- Execute code
- **Get output**
- Render template
- Push report

Getting output

- last cell of any report should be a code to write a file named output.json
- this file should contain all data that needs to be rendered in HTML template
- Images are passed as base64 strings

```python
import io
import matplotlib.pyplot as plt
import base64

plt.figure(figsize=(10, 6))
plt.plot(...)
plt.title('Chart title')

buffer = io.BytesIO()
plt.savefig(buffer, format='png')
buffer.seek(0)

image_hex = base64.b64encode(buffer.read()).decode('utf-8')

plt.close()
```

Steps

- Create report
- Get Notebook
- Start Pod
- Get Kernel file
- Connect to pod
- Execute code
- **Get output**
- Render template
- Push report


```python
import select
from kubernetes.stream.ws_client import STDOUT_CHANNEL, STDERR_CHANNEL
from websocket import ABNF


class WSFileManager:

    def __init__(self, ws_client):
        self.ws_client = ws_client

    def read_bytes(self, timeout=0):
        stdout_bytes = None
        stderr_bytes = None

        if self.ws_client.is_open():
            if not self.ws_client.sock.connected:
                self.ws_client._connected = False
            else:
                r, _, _ = select.select(
                    (self.ws_client.sock.sock, ), (), (), timeout)
                if r:
                    op_code, frame = self.ws_client.sock.recv_data_frame(True)
                    if op_code == ABNF.OPCODE_CLOSE:
                        self.ws_client._connected = False
                    elif op_code == ABNF.OPCODE_BINARY or op_code == ABNF.OPCODE_TEXT:
                        data = frame.data
                        if len(data) > 1:
                            channel = data[0]
                            data = data[1:]
                            if data:
                                if channel == STDOUT_CHANNEL:
                                    stdout_bytes = data
                                elif channel == STDERR_CHANNEL:
                                    stderr_bytes = data
        return stdout_bytes, stderr_bytes, not self.ws_client._connected
```

```python
from tempfile import TemporaryFile
import tarfile
from logging import getLogger
from kubernetes.stream import stream
from wsfilemanager import WSFileManager

logger = getLogger(__name__)


def stream_copy_from_pod(self, pod_name, name_space, source_path, destination_path):
    command_copy = ['tar', 'cf', '-', source_path]
    with TemporaryFile() as tar_buffer:
        exec_stream = stream(self.coreClient.connect_get_namespaced_pod_exec, pod_name, name_space,
                             command=command_copy, stderr=True, stdin=True, stdout=True, tty=False,
                             _preload_content=False)
        # Copy file to stream
        try:
            reader = WSFileManager(exec_stream)
            while True:
                out, err, closed = reader.read_bytes()
                if out:
                    tar_buffer.write(out)
                elif err:
                    logger.debug("Error copying file {0}".format(err.decode("utf-8", "replace")))
                if closed:
                    break
            exec_stream.close()
            tar_buffer.flush()
            tar_buffer.seek(0)
            with tarfile.open(fileobj=tar_buffer, mode='r:') as tar:
                member = tar.getmember(source_path)
                tar.makefile(member, destination_path)
                return True
        except Exception as e:
            raise e
```

Steps

- Create report
- Get Notebook
- Start Pod
- Get Kernel file
- Connect to pod
- Execute code
- Get output
- **Render template**
- Push report

Render Template

- template is opened
- template is rendered with loaded output.json

Translations

- language is set in report file (lang)
- HTML has support for gettext translations:
  - `{{ _(`My Comment’) }}`

Steps

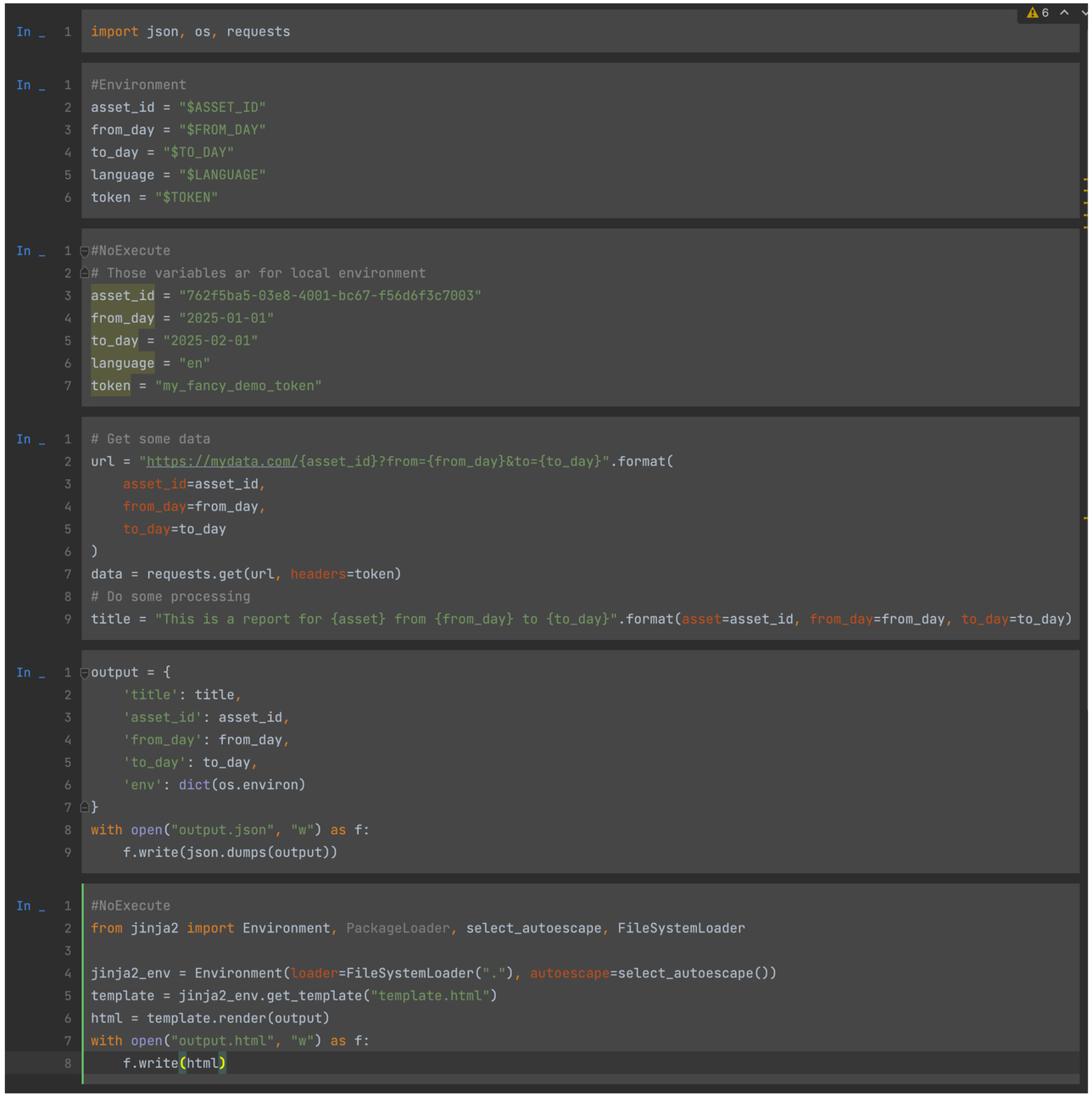
- Create report
- Get Notebook
- Start Pod
- Get Kernel file
- Connect to pod
- Execute code
- Get output
- Render template
- **Push report**

Final steps

- Push report to S3
- Set report file status as “completed” or “error”
- In case of error, add Trace
- Kill kernel and ensure pod is deleted

Local Development

- Set static data with #NoExecute
- Render template example with #NoExecute
- Use regular Jupyter Notebook as any other analysis



```
In _  1  import json, os, requests

In _  1  #Environment
      2  asset_id = "$ASSET_ID"
      3  from_day = "$FROM_DAY"
      4  to_day = "$TO_DAY"
      5  language = "$LANGUAGE"
      6  token = "$TOKEN"

In _  1  #NoExecute
      2  # Those variables ar for local environment
      3  asset_id = "762f5ba5-03e8-4001-bc67-f56d6f3c7003"
      4  from_day = "2025-01-01"
      5  to_day = "2025-02-01"
      6  language = "en"
      7  token = "my_fancy_demo_token"

In _  1  # Get some data
      2  url = "https://mydata.com/{asset_id}?from={from_day}&to={to_day}".format(
      3      asset_id=asset_id,
      4      from_day=from_day,
      5      to_day=to_day
      6  )
      7  data = requests.get(url, headers=token)
      8  # Do some processing
      9  title = "This is a report for {asset} from {from_day} to {to_day}".format(asset=asset_id, from_day=from_day, to_day=to_day)

In _  1  output = {
      2      'title': title,
      3      'asset_id': asset_id,
      4      'from_day': from_day,
      5      'to_day': to_day,
      6      'env': dict(os.environ)
      7  }
      8  with open("output.json", "w") as f:
      9      f.write(json.dumps(output))

In _  1  #NoExecute
      2  from jinja2 import Environment, PackageLoader, select_autoescape, FileSystemLoader
      3
      4  jinja2_env = Environment(loader=FileSystemLoader("."), autoescape=select_autoescape())
      5  template = jinja2_env.get_template("template.html")
      6  html = template.render(output)
      7  with open("output.html", "w") as f:
      8      f.write(html)
```

Tips and tricks

- Timeout is very important
- Set limits for memory and CPU
- Capture exception trace from kernel and give it back to user
- Add as much log as possible
- Make easy for customer to debug

Examples

Circutor

Daily Steps Report

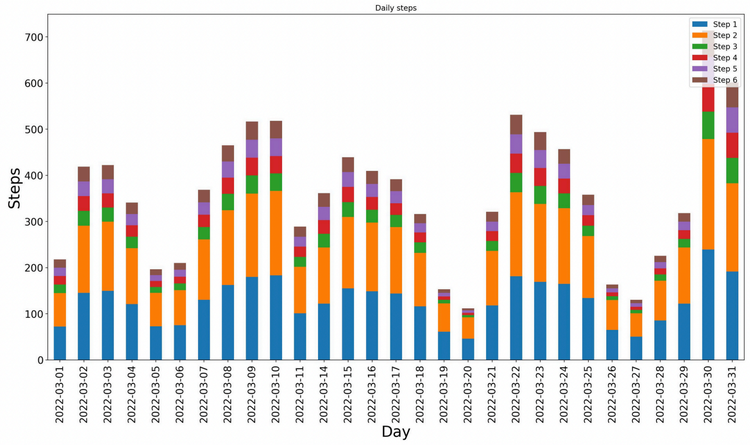
Capacitor Bank Info

Report from 28/02/2022 to 31/03/2022

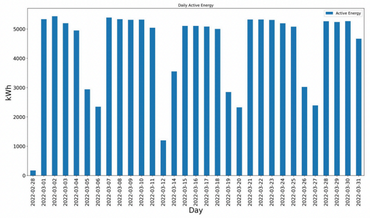
Name:	Description:	Company:	Location:	Power:
CT Circutor	Batería ET	CIRCUTOR	Viladecavalls	125

Voltage:	Number of Steps:	Rejection Filters:
400	6	No

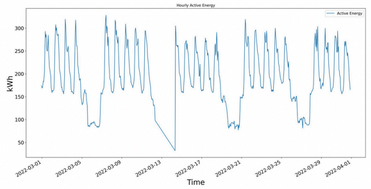
Daily Steps



Daily Energy



Hourly Energy



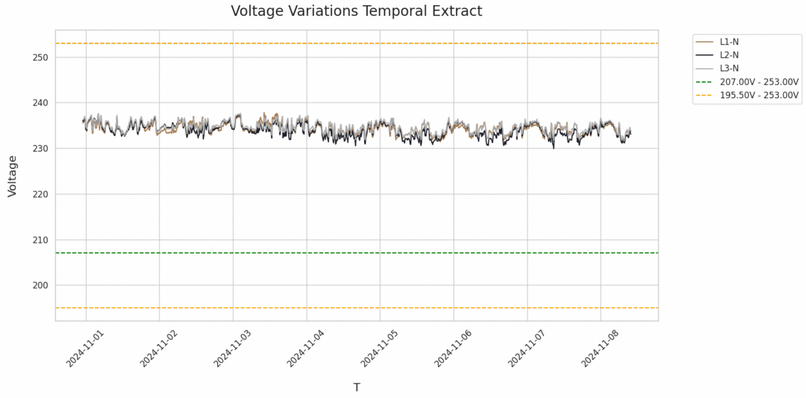
Automatic report generated by MyCircutor. www.mycircutor.com

2. Supply Voltage Variations

2.1. L-N Voltage Variations

Nominal Voltage:	230.0V
Parameter definition:	Supply voltage RMS values averaged over 10 min
Limitation:	For networks coupled by synchronous connections to an interconnected system Interruption intervals excluded

EN 50160 Requirement	% time needed for compliance	L1 Voltage (% values)	L2 Voltage (% values)	L3 Voltage (% values)	Result
207.00V - 253.00V	95.0%	100.0%	100.0%	100.0%	PASS
195.50V - 253.00V	100%	100.0%	100.0%	100.0%	PASS



Any question?

Thank you :)

xavier.orduna@gmail.com
www.maulabs.cat