09/01/2025 - MDE P5A

CSR REPORT

APRIL PROJECT

Thibaut DESAUTY
Léo GOUTARD
Antoine BETBEDER
Zi YANG
Théo POINET

1) Context & Measurement Considerations

The **A.P.Ri.L. (Analyse de la Perception des Risques Littoraux en Occitanie)** project focuses on analysing perceptions of coastal risks in the Occitanie region of southern France. It employs data collection and analysis methodologies, primarily leveraging web scraping and Natural Language Processing (NLP) techniques.

Given the project's design as a low-impact initiative, with brief and intermittent usage, we focus on measuring resource consumption during active computational tasks, rather than the production period and manufacturing process of components like CPUs or RAM. Specifically, we focus on the resources used by our Docker containers as the entirety of our services are running through two of them.

2.1 What we measure

To evaluate the project's resource consumption and environmental footprint, we track the following key metrics:

- **CPU usage** reflects processor utilization during computational tasks. This metric helps quantify the processing power required for tasks such as data extraction, NLP, and other analyses related to the project (we don't use GPU).

- **RAM usage** tracks the memory consumed during the execution of the project's processes.

- **Network usage** measures the data exchanged during operations like web scraping, where frequent interactions with external sources occur. This metric helps evaluate the project's impact on data transfer and related energy costs.

Resource monitoring is conducted across two Docker containers:

1. **Database Service Container** – Responsible for managing data storage and retrieval.
2. **Backend and Frontend Container** – Handles the processing logic and user interface for the project.

It is important to note that, for the A.P.Ri.L. project, we focus on the runtime consumption of resources during active data collection or analysis/processing phases, rather than the manufacturing process of hardware or the continuous daily operation of servers. This is because the project involves very occasional, short bursts of activity with relatively few users (1 to 4). The sporadic nature of these operations means we are not concerned with long-term hardware usage or heavy daily loads, which would require accounting for manufacturing energy or continuous operational resources like CPU and RAM usage over long periods.

2.2 Why These Metrics Matter

Although the computational tasks are performed intermittently, these short bursts of activity still contribute to the overall environmental impact. While resource consumption is often secondary in research projects, measuring CPU, RAM, and network usage is important for the A.P.Ri.L. project to ensure transparency and align the project with sustainable research practices, even for what might be considered a "low-impact" initiative.

Additionally, monitoring resource usage can reveal inefficiencies or bottlenecks in the data processing workflow, which may lead to unnecessary energy consumption. Although optimizing resource use isn't the project’s main focus, addressing these issues can help cut down on waste and improve performance if the APRIL project is being reused or scaled up (which will induce more consumption from CPU, RAM, and network).

The power consumption data of our system will enable us to calculate the project's carbon footprint given France's specific energy mix, which includes a significant proportion of low-carbon energy sources.

2) Methodology

2.1 Small-Scale Testing Phase

Our methodology begins with a small-scale testing phase that involves scraping and analyzing 30 documents. This controlled test allows us to measure key resource consumption metrics—CPU, RAM, and network usage—during the Docker container runtime. Simulating the system's real-world behavior on a smaller scale provides accurate performance data, enabling us to identify potential bottlenecks and inefficiencies before scaling up operations.

This approach is also essential for minimizing our own carbon emissions during the assessment phase. Given the frequent evolution of the project's features, conducting measurements on all 700 documents each time would significantly increase our carbon footprint solely from the effort to measure and estimate resource usage. Starting small ensures that we gather meaningful insights without contributing unnecessarily to environmental impact.

2.2 Extrapolation to Full-Scale Scraping

The full-scale scraping task will involve processing approximately 700 documents. To estimate the resource consumption for this larger operation, we will extrapolate data collected from the small-scale test using two distinct methods:

- **CPU and RAM Usage (Time-Based Extrapolation):** Since scraping 700 documents will take roughly 23 times longer than scraping 31 (700/31 = ~23), we will multiply the duration of CPU and RAM usage by a factor of 23. This accounts for the extended runtime of these resources in the full-scale operation.
- **Network Usage (Volume-Based Extrapolation):** For data uploaded and downloaded, we will scale directly based on volume rather than time. The total amount of network data transferred during the small-scale test will be multiplied by 23 to estimate the full data transfer requirements.

By applying these extrapolation methods, we can accurately predict the environmental impact—including energy consumption and CO2 emissions—associated with processing the complete set of 700 documents.

2.3 Measurements

2.3.1 Monitoring using Docker stats

To accurately measure resource usage during the small-scale testing phase, we developed a custom Bash script that leverages the `` `docker stats` `` command to monitor system performance in real time. This script collects key metrics every 15 seconds, including memory usage (in MB and %), CPU utilization (%), and network activity (incoming and outgoing data).

After data collection, we analyze the results by calculating the average memory and CPU usage over the testing period to reflect typical resource consumption. For network activity, we compute the cumulative sum of data transferred (both incoming and outgoing) to determine the total network load. This method allows us to obtain reliable resource consumption metrics, which we then extrapolate to estimate the impact of full-scale operations.

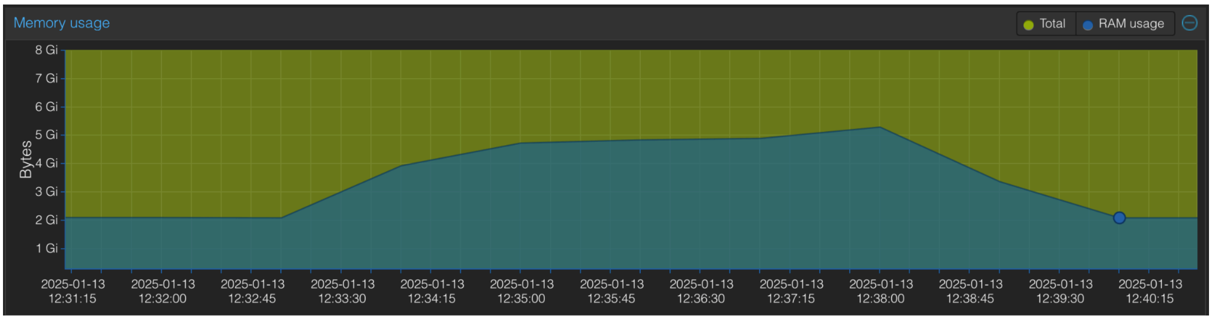
Container	CPU Usage	RAM Usage	Network Usage	Documents Scraped
Backend & frontend	19% across 2 CPU	2.89 GiB (37.4%)	1485.32 MB in 44.02 MB out	31
Database	0.32% across 2 CPU	86.40 MiB (1.08%)	11.65 MB in 30.41 MB out	31

The execution time was **3 minutes and 15 seconds (12:35:00 – 12:38:15)**.

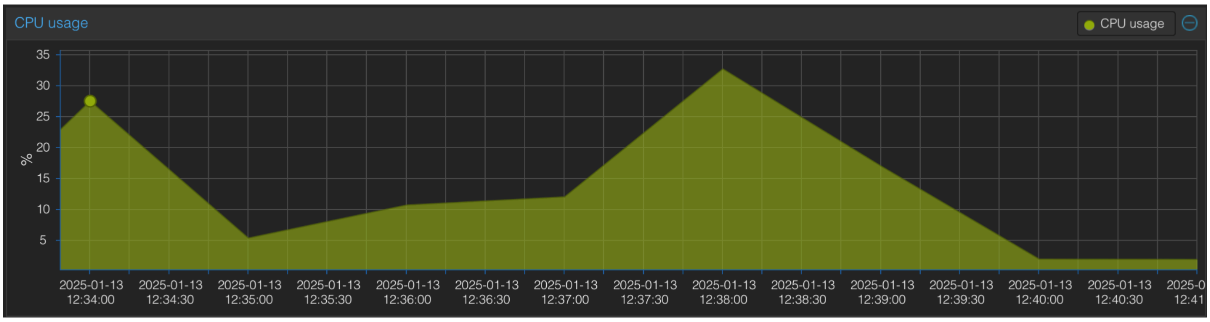
2.3.2 Virtual Machine Resource Monitoring with Proxmox

In addition to container-level monitoring, we evaluated the overall resource usage of our virtual machine (VM) using the Proxmox UI. Proxmox provided system-wide performance graphs for CPU, memory, and network activity.

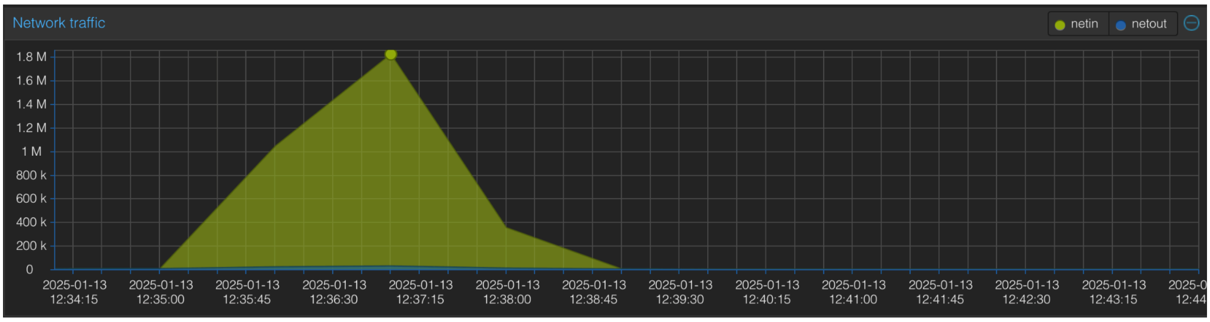
Proxmox UI - Memory usage graph



Proxmox UI - CPU usage graph



Proxmox UI - Network usage graph



Summary:

Memory_Usage (avg)	CPU_Usage (avg)	Network In and out (sum)
4.6Mb	22.4%	11.64Mb

The data collected from Proxmox closely aligns with the results from our Docker-based monitoring script, with slight discrepancies explained by the overhead of the VM itself:

- **CPU Usage:** Proxmox reported an average CPU usage of **22.4%**, compared to **19%** recorded by `` `docker stats` ``. This difference is attributed to the VM's own resource requirements for managing system operations beyond the containers.
- **Memory Usage:** Proxmox measured memory consumption at **5.9 MB**, while Docker reported **2.9 MB**. Similarly, the additional memory usage in Proxmox reflects the baseline memory needed to run the VM environment and some cache.

However, a significant difference was observed in **network usage**:

- By calculating the area under the Proxmox network usage graph (integral), we determined a total data transfer of approximately **11.64 MB**.
- In contrast, Docker reported around **1,500 MB** of network activity.

This discrepancy arises because `docker stats` captures **all internal and external data exchanges** within the containers, including communication between our two containers, which inflates the network usage figure. For our sustainability analysis, this internal traffic is irrelevant since we aim to assess external data transfers (between the VM and the internet) to accurately estimate the carbon footprint using network-based emission factors.

As a result, we will rely on **Proxmox network data** for evaluating network resource consumption. This provides a more accurate measurement of the actual data I/O with external networks, ensuring that our CO_2 impact calculations are both relevant and precise.

3) Calculation and Estimation of Energy Consumption

To estimate the environmental impact, we can use the Thermal Design Power (TDP) of the CPUs involved to estimate power consumption and translate it into energy use (in watt-hours). Similarly, we can estimate the energy consumption for network usage based on typical power consumption rates for network connections, factoring in the location and type of connection. Regarding the RAM, its energy consumption can be estimated by considering the average power draw per GB of memory in use. Combining these estimates for CPU, network, and RAM usage provides a comprehensive view of the project's energy footprint, which can then be translated into CO2 emissions based on the energy mix.

3.1 CPU Energy Consumption (TDP Approach)

Thermal Design Power (TDP) is a metric used to define the amount of heat generated by a CPU that the system's cooling solution must dissipate under maximum load. TDP can also be used to estimate power consumption, assuming the CPU is running at full capacity. Different CPUs have different TDP values, which can be used to estimate energy consumption during the runtime of the Docker containers.[1]

In our case:

- CPU model: Intel Xeon Silver 4210 @ 2.20 GHz, allocated 2 cores
- TDP of this CPU: **85 watts** [2]
- CPU Usage:
  - **Frontend/Backend Container: 19%**
  - **Database Service Container: 0.32%**

To estimate the energy consumption, we calculate the proportion of the TDP that is used based on CPU utilization:

$$CPU\ Power\ =\ TDP\ \ *\ CPU\ usage$$

For the Frontend/Backend Container with **19% CPU usage**:

$$CPU\ Power\ =\ 85\ *\ \frac{19}{100}\ \ =\ 16.15\ W$$

For the Database Service Container with **0.32% CPU usage**:

$$CPU\ Power\ =\ 85\ *\ \frac{0.32}{100}\ \ =\ 0.272\ W$$

Total CPU Consumption:

$$Total\ CPU\ Power\ =\ \ 16.15W\ +\ 0.272W\ \ =\ \ 16.422\ W$$

As the container runs for **0.0542 hour**s, the energy consumption in watt-hours (Wh) is:

$$Energy\ Consumption\ (Wh)\ \ =\ Total\ CPU\ Power\ \ *\ \ Time$$

$$Energy\ Consumption\ (small\ scale)\ \ =\ 16.422\ W\ \ *\ \ 0.0542\ hours\ \ =\ \ 0.89\ Wh$$

<u>Extrapolation to a full launch:</u>

For a full-scale task involving 700 documents, if CPU usage remains consistent, the energy consumption would be extrapolated based on the time involved, thus we get **1.22 hours**.

$$Energy\ Consumption\ (Full\ launch)\ \ =\ 16.422\ W\ \ *\ \ 1.22\ hours \approx 20.035\ Wh$$

3.2 RAM Energy Consumption

To estimate the energy consumption of RAM during the scraping and analysis tasks, we use average power consumption figures for an 8 GiB DDR4 module. Typically, an idle module consumes between 2 to 4 watts, while under heavy usage, the consumption rises to approximately 4 to 7 watts. [3, 4, 5] We assume intensive usage (**5.5 watts**) as scraping and analysis tasks are generally demanding operations for the system, particularly for the RAM.

The energy consumption per GiB of memory can therefore be estimated based on the average power usage during the process. By taking the average power consumption for both containers and multiplying it by the runtime and memory usage, we can calculate the total energy consumption for the RAM.

$$RAM\ Power\ =\ RAM\ used(MiB)\ *\ \frac{RAM\ Consumption\ (W)}{1024\ (MiB)}$$

For the Frontend/Backend Container with **37.4% RAM usage**:

$$RAM\ Power\ =\ 2.89\ GiB\ *\ 1024\ *\ \frac{5.5\ W}{1024\ (MiB)}\ =\ 15.895\ W$$

For the Database Service Container with **1.08% RAM usage**:

$$RAM\ Power\ =\ 86.40MiB\ *\ \frac{5.5\ W}{1024\ (MiB)}\ =\ 0.464\ W$$

Total RAM Consumption:

$$Total\ RAM\ Power\ =\ 15.895\ W\ +\ 0.464\ W\ =\ 16.359\ W$$

As the container runs for **0.0542 hours**, the energy consumption in watt-hours (Wh) is:

$$Energy\ Consumption\ (Wh)\ =\ Total\ RAM\ Consumption\ *\ Time$$

$$Energy\ Consumption\ (small\ scale)\ =\ 16.359\ W\ *\ 0.0542\ hours\ =\ 0.887\ Wh$$

<u>Extrapolation to a full launch:</u>

Following the same procedure as for the CPU, we obtain:

$$Energy\ Consumption\ (Full\ launch)\ =\ 16.359\ *\ 1.22\ hours\ =\ 19.958\ Wh$$

3.3 Network Energy Consumption

To estimate the energy consumption associated with network usage, we can use the average power consumption for Ethernet connections, which applies both to our system and that of our client. A typical Ethernet connection consumes between **0.05 and 0.1 kWh** [6] per gigabyte, depending on the network node infrastructure in use, as reported by the International Energy Agency (IEA).

For the task of downloading 31 documents, we get **11.64 MB** of data transferred which allows us to calculate the related energy consumption.

$$Network\ Power\ Consumption\ =\ Ethernet\ connection\ consumption\ *\ Data\ load$$

$$Network\ Energy\ Consumption\ (small\ scale)\ =\ 0.1kWh/GB\ *\frac{11.64}{1000}\ GB \approx 1.164\ Wh$$

Extrapolation to a full launch:

The network power consumption is assumed to be constant during the entire scraping period. To extrapolate to the full 700-document scrape, we can assume that there will be ~23 times more data download/upload:

$$Network\ Power\ Consumption\ (Full\ \ launch)\ =\ 0.1kWh/GB\ *\frac{262.839}{1000}\ GB \approx 26.284\ Wh$$

3.4 Total Energy Consumption Estimation

Now, let's combine the energy consumption from CPU, RAM and network of full-scale scraping:

- CPU Energy Consumption (full launch): $20.035\ Wh$
- RAM Energy Consumption (full launch): $19.958\ Wh$
- Network Energy Consumption (full launch): $26.284\ Wh$

Total Consumption:

$$Total\ Energy\ Consumption\ =\ 20.035\ Wh\ +\ 19.958\ Wh\ +\ 26.284\ Wh\ \approx\ 66.277\ Wh$$

4) CO2 Emissions Estimation

To estimate the CO2 emissions associated with the energy consumption, we can use the French energy mix, which is composed of approximately 70% low-carbon energy (nuclear and renewables) and 30% fossil fuels. The average CO2 emission factor for electricity in France is roughly 0.027 kg CO2 per kWh in 2024. [7]

To calculate the CO2 emissions:

$$CO2\ Emissions\ =\ Energy\ Consumption(kWh)\ \ *\ CO2\ Emissions\ Factor$$

$$CO2\ Emissions\ (Full\ launch)\ =\ \frac{66.277}{1000} kWh\ *\ 0,027\ kg\ CO2\ /kWh\ =\ 0.001789\ kg\ CO2$$

Thus, the **estimated CO2 emissions** for the full-scale scraping of 700 documents would be approximately **1.789 grams.**

Conclusion

Summary of the results	CPU Energy Consumption	RAM Energy Consumption	Network Energy Consumption	Total Energy Consumption	CO2 Emissions
Small-Scale Test (50 Docs)	$0.89\ Wh$	$0.887\ Wh$	$1.164\ Wh$	$2.941\ Wh$	$0.0794\ g\ CO2$
Full-Scale Test (700 Docs)	$20.035\ Wh$	$19.958\ Wh$	$26.284\ Wh$	$66.277\ Wh$	$1.789\ g\ CO2$

The energy consumption and CO2 emissions observed during the small-scale and full-scale tests of our system highlight important insights into its environmental impact. In the small-scale test (31 documents), the system consumed a total of 2.941 Wh of energy, resulting in 0.0794 grams of CO_2 emissions. By contrast, the full-scale test (700 documents) revealed a significantly higher total energy consumption of 66.277 Wh, leading to an emission of 1.789 grams of CO_2 per entire launch. These results clearly show how energy use and carbon footprint scale with the size of the task.

In terms of sustainability, the carbon footprint associated with this project, calculated using France's energy mix, helps contextualize the environmental impact. For example, the emissions of 1.789 grams of CO_2 from the full-scale test are roughly equivalent to cycling around 0.03 to 0.11 kilometers [8], or driving a petrol-fueled car for about 12 meters [9]. Alternatively, the emissions could be offset by brewing a fraction of a cup of coffee, considering that a standard espresso has a carbon footprint of about 60 grams of CO_2 equivalent [10].

By calculating these emissions, we ensure that the project aligns with sustainability objectives, promoting responsible research practices. This approach not only facilitates the responsible use of resources but also contributes to advancing scientific understanding of coastal risk perception in the Occitanie region, all while maintaining a commitment to reducing environmental impact.

References

[1] – **Processor TDP Overview**: Processor Base Power: Which CPU TDP For Performance, Efficiency & Reliability? | Things Embedded UK

[2] – **Intel TDP**: Intel® Xeon® Silver 4210 Processor

[3] – **RAM Energy Consumption**: How much power does memory use? | Crucial.com

[4] – **DDR4 RAM Energy Usage**: DDR4 SDRAM | Micron Technology Inc.

[5] – **DRAM Power Calculators**: DRAM power calculators | Micron Technology Inc.

[6] – **Network Energy Consumption**: The carbon footprint of streaming video: fact-checking the headlines – Analysis - IEA

[7] – **CO2 Emissions in France**: CO2 emissions per kWh in France - Nowtricity

[8] – **Bike CO2 equivalent:** Which form of transport has the smallest carbon footprint? - Our World in Data

[9] – **Car CO2 equivalent:** Average CO2 emissions from new cars and new vans increased again in 2019

[10] – **Coffee CO2 equivalent:** Analysis: here's the carbon cost of your daily coffee – and how to make it climate-friendly | UCL News