Carbon aware scheduling of serverless workflows

About the project

The objective of this work is the implementation of a carbon aware scheduling of Lambda functions that distributes the load among the AWS Regions, considering the carbon footprint data.

Motivation

The large use of cloud computing services is causing an increase in carbon emissions. In particular, the data center industry contributes around 0,3% to overall carbon emissions, and the ICT ecosystem accounts for more than 2% of global emissions.

User Services

Fibonacci

The user provides an integer to calculate the number of Fibonacci

Matrix Inversion

The user provides a square matrix of which he wants to calculate the inverse matrix

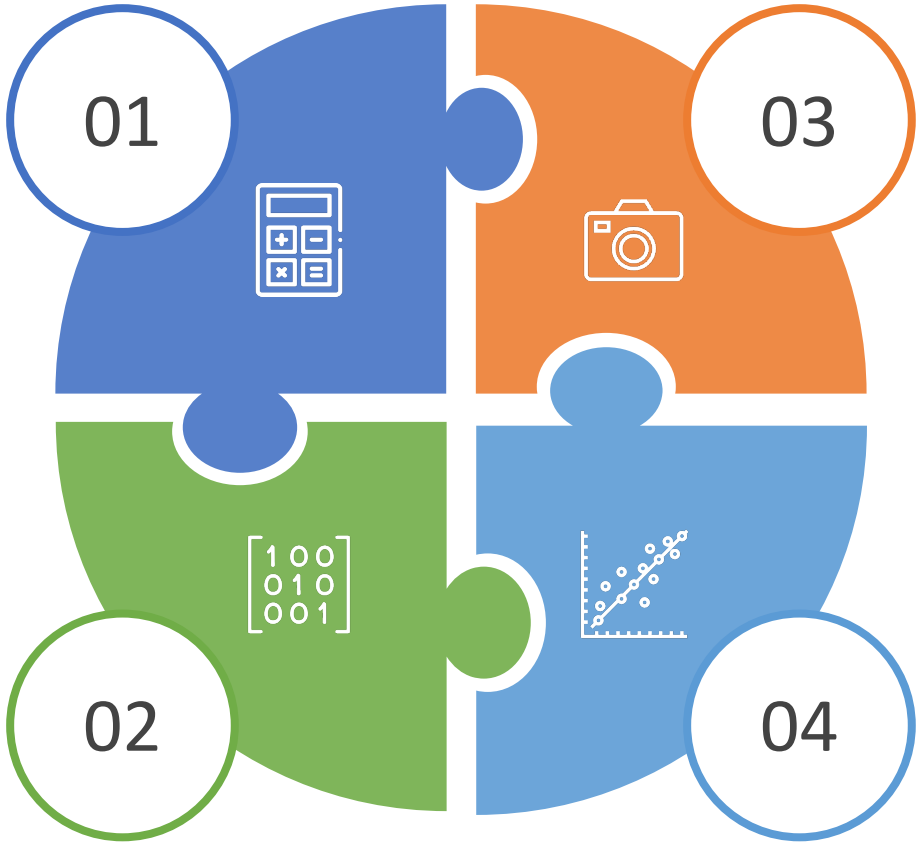


Image Resizing

The user provides the path of an image that he wants to resize

Linear Regression

Provides the X matrix and the y vector to calculate the coefficients of the linear regression

Amazon Web Services (AWS)

Amazon Lambda Function

Amazon EventBridge

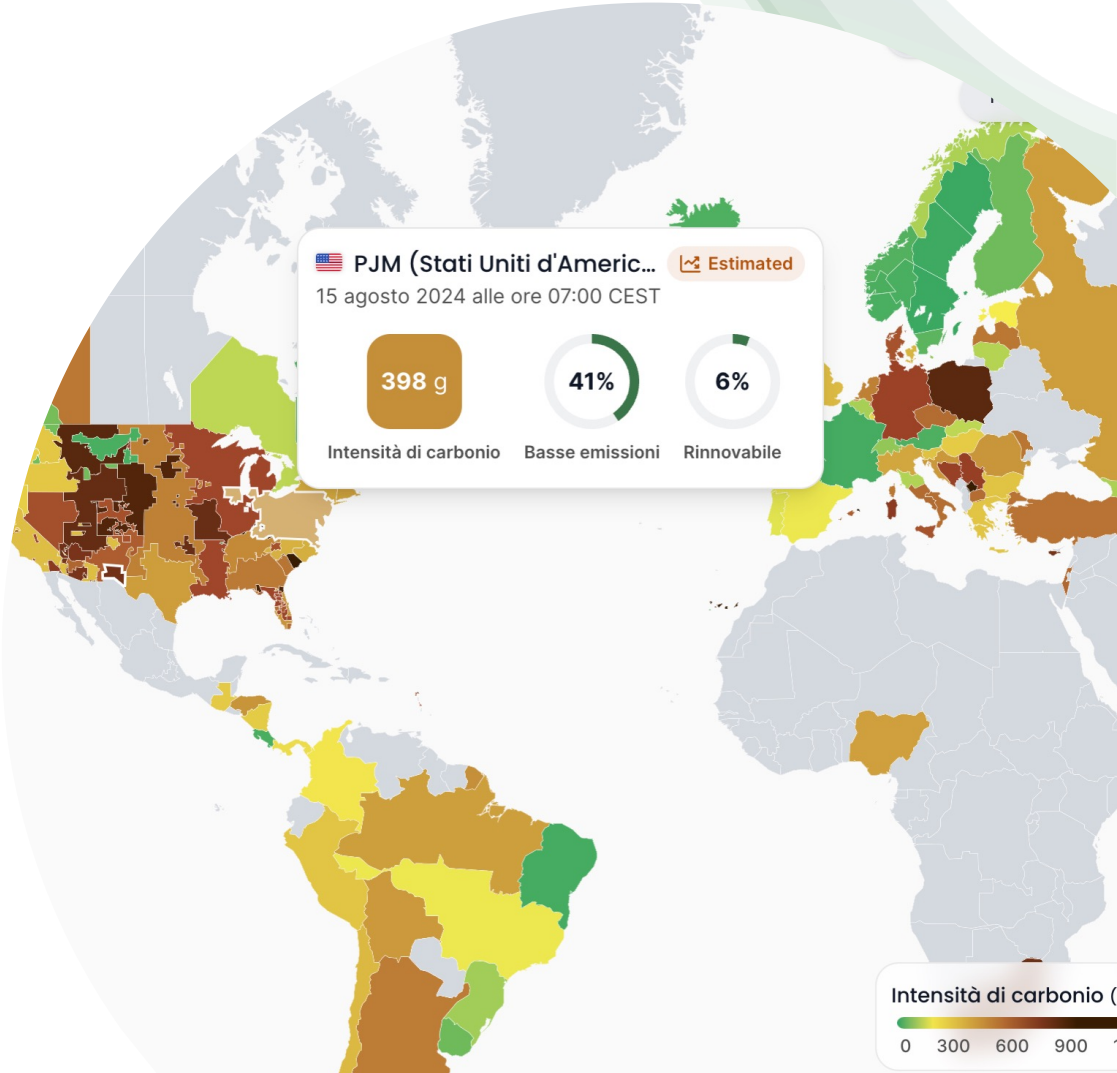
Amazon Simple Storage Service (S3)

Amazon CloudWatch

Amazon DynamoDB

Electricity Maps

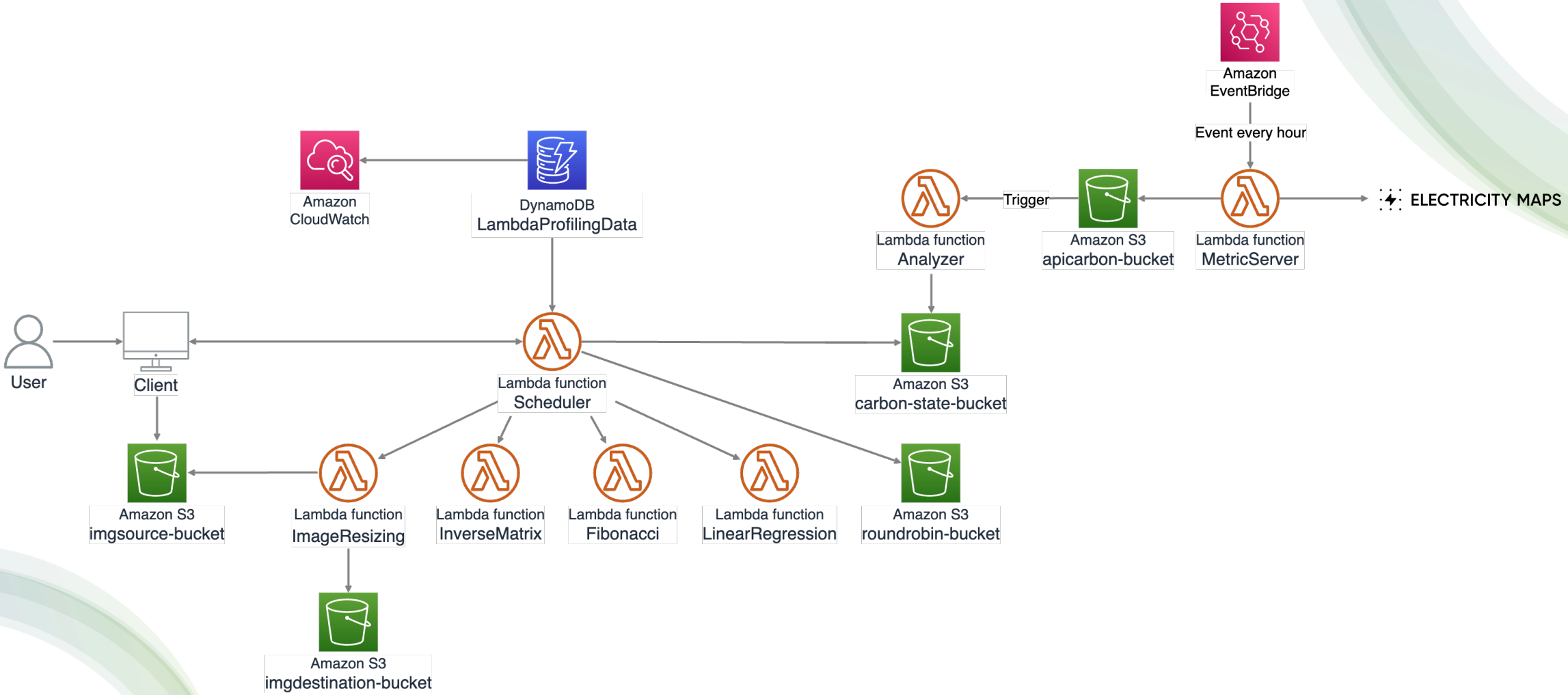
We collected carbon footprint data with Electricity Maps, a platform with access to carbon emissions and electricity sources. The API delivers data with hourly granularity, for more than 230 regions, but we focused only in the areas concerned, mapping the zones provided by Electricity Maps with the AWS regions.



https://api.electricitymap.org/v3/carbon-intensity/latest?zone=US-MIDA-PJM

https://api.electricitymap.org/v3/power-breakdown/latest?zone=US-MIDA-PJM

Architecture



EcoEfficiency Index (EEI)

The EcoEfficiency Index (EEI) *(1)* is the metric we considered to define the most substainable regions. The higher the EEI value, the more eco-friendly the region is. Then, we applied a Min-Max Normalization to the EEI values *(2)*.

(1)

$$EEI = \frac{RenewablePercentage}{CarbonIntensity} * 100$$

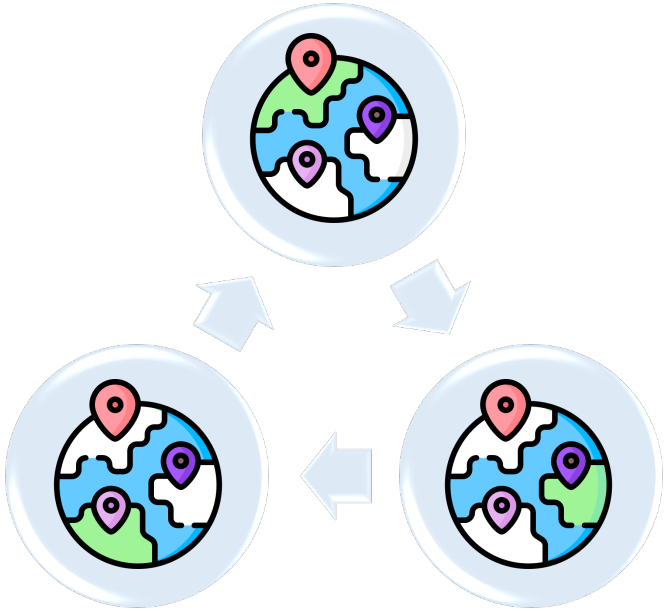
(2)

$$EEI_{normalized} = \frac{EEI - EEI_{min}}{EEI_{max} - EEI_{min}} * 100$$

Round Robin

A round robin scheduling allows to distribute the load beetween the regions. Each invocation selects a region in a circular way within a group. The state of round robin consists of an index per group, indicating the position of the next region to choose.

Since the regions are sorted by decreasing values of the EEI metric, it would be more sustainable to invoke all Lambda functions on the first region. However, the best region would be overloaded, making it more convenient to distribute the workload also at the expense of energy efficiency.



Monitoring

We invoked the Lambda functions by running tests created on the AWS console, with different inputs, a significant number of times to be able to capture the performance trend.

We used CloudWatch Logs Insights to search and analyze our log data and retrieve average memory and execution time of the Lambda functions through queries.

Time

```
fields @timestamp, @message
| filter @message like /REPORT/
| parse @message /Duration: (?<duration>\d+\.\d+) ms/
| display @timestamp, duration
| stats avg(@duration) as avgDuration
```

Memory

```
fields @timestamp, @message
| filter @message like /REPORT/
| parse @message /Max Memory Used:(?<maxMemory>\d+\.\d+) MB/
| display @timestamp, maxMemory
| stats avg(maxMemory) as avgMaxMemory
```

Results 1

The table shows, for each function invocation, the average execution time (ms) and memory usage (MB), and maximum and minimum input dimensions allowed.

Name	Execution Time	Memory Used	Min Input	Max Input
Fibonacci	8,4982	31,6951	0	10.000
LinearRegression	7,9916	75	1	1.000
ImageResizing	1496,6932	160,3684	1	64.000.000
InverseMatrix	15,1337	77,7408	1	50

Function Classification and Intensity Index

The function is directly classified as high intensity if the average execution time or memory used exceed the values of 100 ms and 100 MB.

Otherwise, we calculate the Intensity Index, which also takes into account the size of the input.

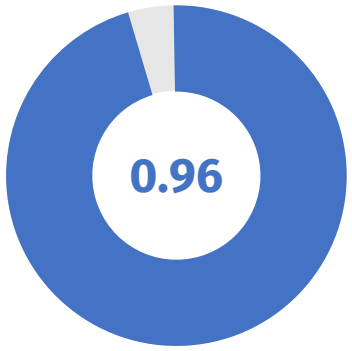
$$intensityIndex = 0.4 * input_{norm} + 0.3 * time_{norm} + 0.3 * memory_{norm}$$

≥ 0.5 → The function is high intensity

< 0.5 → The function is low intensity

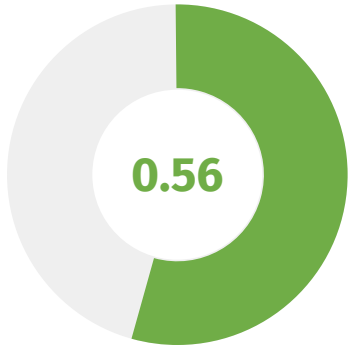
Results 2

The chart shows for each function the minimum input dimension that enables to classify the function as high intensity. Fibonacci is the least computationally expensive Lambda function.



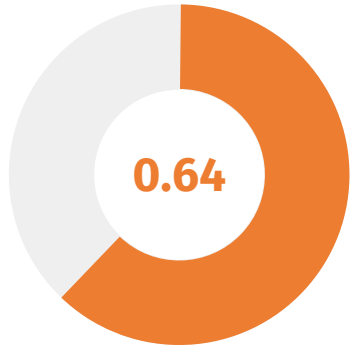
Fibonacci

InverseMatrix



ImageResizing

LinearRegression



Thanks for the attention