WattVault

Cloud Computing Project

SAPIENZA
UNIVERSITÀ DI ROMA

Speakers:
Gaia Landolfo
Loris Lindozzi

Roadmap

01 Introduction & Architecture

Why WattVault
and what it is

02 Implementation

Detail of the
implementation

WattVault

03 Testing

Environment setup &
Scaling test

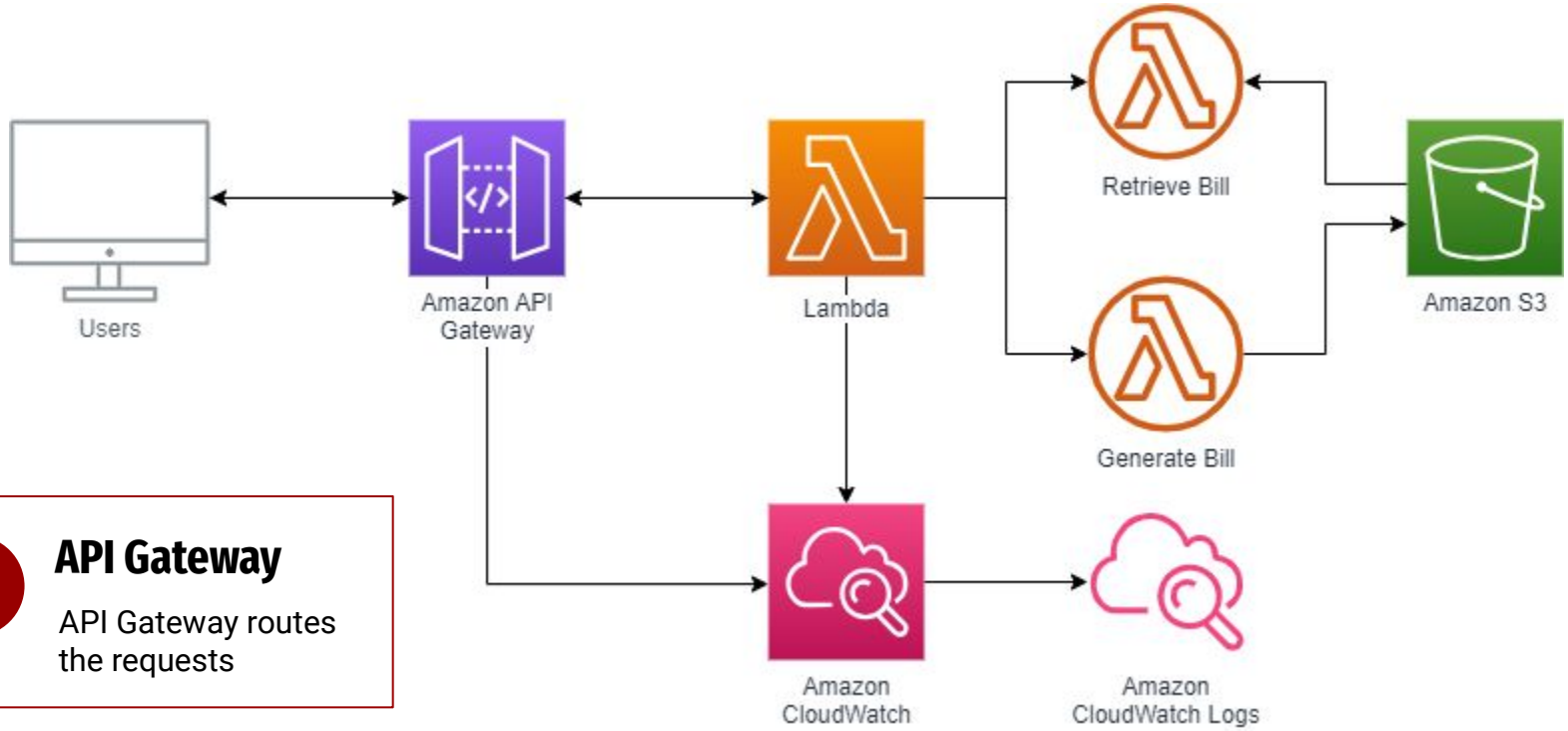
04 Conclusions

Further works &
Conclusions

Introduction

WattVault is a serverless cloud application designed for the calculation and storage of electricity bills.
It allows users to send their data to the application to generate a PDF report for monthly expenses, which will then be archived for later retrieval at any time.

Architecture



1

API Gateway

API Gateway routes the requests

2

Lambda

It is the backend, with 2 independent microservices

3

S3 Bucket

The bucket is used to store the bills

4

CloudWatch

Cloudwatch is used for collecting statistics and monitoring

Implementation

Creation of Lambda functions

- GeneratePDF: The PDFgenerator Lambda function is designed to process electricity bill data, generate a PDF report, and store it in an Amazon S3 bucket

- GetPDF: The getPDF Lambda function is designed to retrieve a PDF report of an electricity bill from an Amazon S3 bucket based on the user's request. Then constructs an HTTP response with the PDF content.

lambda_function × Environment Vari ×

```python

s3_client = boto3.client('s3')
BUCKET_NAME = 'pdf-bill-store2'  # replace with your S3 bucket name

RATE_PER_UNIT = 0.35 #Define your rate per unit of consumption


def generate_pdf(data):
    buffer = BytesIO()
    pdf = canvas.Canvas(buffer, pagesize=letter)
    pdf.drawString(100, 750, "Electricity Bill Report")
    pdf.drawString(100, 730, f"id: {data['id']}")
    pdf.drawString(100, 710, f"Name: {data['name']}")
    pdf.drawString(100, 690, f"Address: {data['address']}")
    pdf.drawString(100, 670, f"Billing Month: {data['billing_month']}")
    pdf.drawString(100, 650, f"Total Consumption: {data['total_consumption']} kWh")
    total_amount = float(data['total_consumption']) * RATE_PER_UNIT
    pdf.drawString(100, 630, f"Total Amount: ${total_amount:.2f}")
    pdf.save()
    buffer.seek(0)
    return buffer

def lambda_handler(event, context):
    try:
        # Check if 'body' is present in the event
        if 'body' not in event:
            return {
                "statusCode": 400,
                "body": json.dumps({
                    "message": "'body' field is missing in the event"
                })
            }

        # Parse the JSON request
        body = json.loads(event['body'])

        # Generate the PDF
        pdf_buffer = generate_pdf(body)
```

Implementation

2 Creation of the API Gateway

- /generate-bill
  - POST — AWS Lambda
- /retrieve-bill
  - /{user_id}
    - /{billing_month}
      - GET — AWS Lambda

3 Creation of the S3 bucket

Nome	Tipo	Ultima modifica
April_2024.pdf	pdf	31 May 2024 03:44:37 PM CEST
August_2024.pdf	pdf	05 Jun 2024 06:24:35 PM CEST
June_2024.pdf	pdf	31 May 2024 03:44:43 PM CEST

Final Result

An example of electricity bill report

GET https://5sdkcfhy1e.execute-api.us-east-1.amazonaws.com/retrieve-bill/1234/August_2024 Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

Key	Value	Description	••• Bulk Edit
Key	Value	Description	

Body Cookies Headers (6) Test Results 200 OK 925 ms 1.78 KB Save as example •••

Electricity Bill Report
id: 1234
Name: Loris Lindozzi
Address: 123 Main St, Anytown, USA
Billing Month: August 2024
Total Consumption: 532 kWh
Total Amount: $186.20

Testing Environment Setup

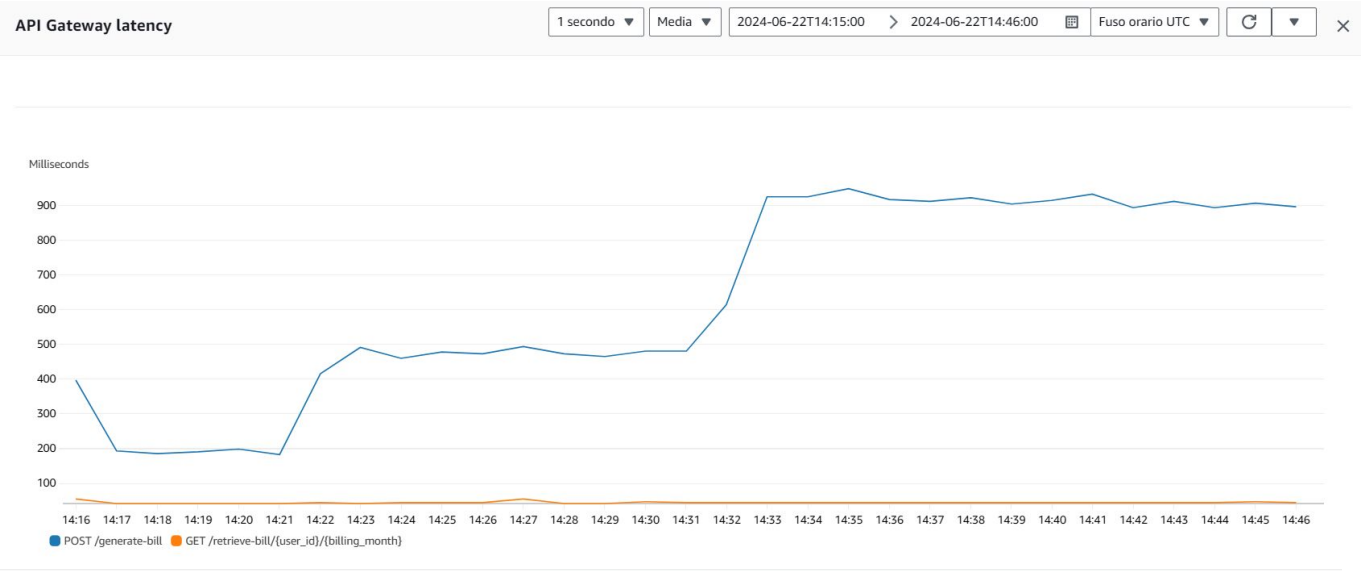
We focused on testing the performance of the infrastructure by conducting stress tests. For this purpose, we used:

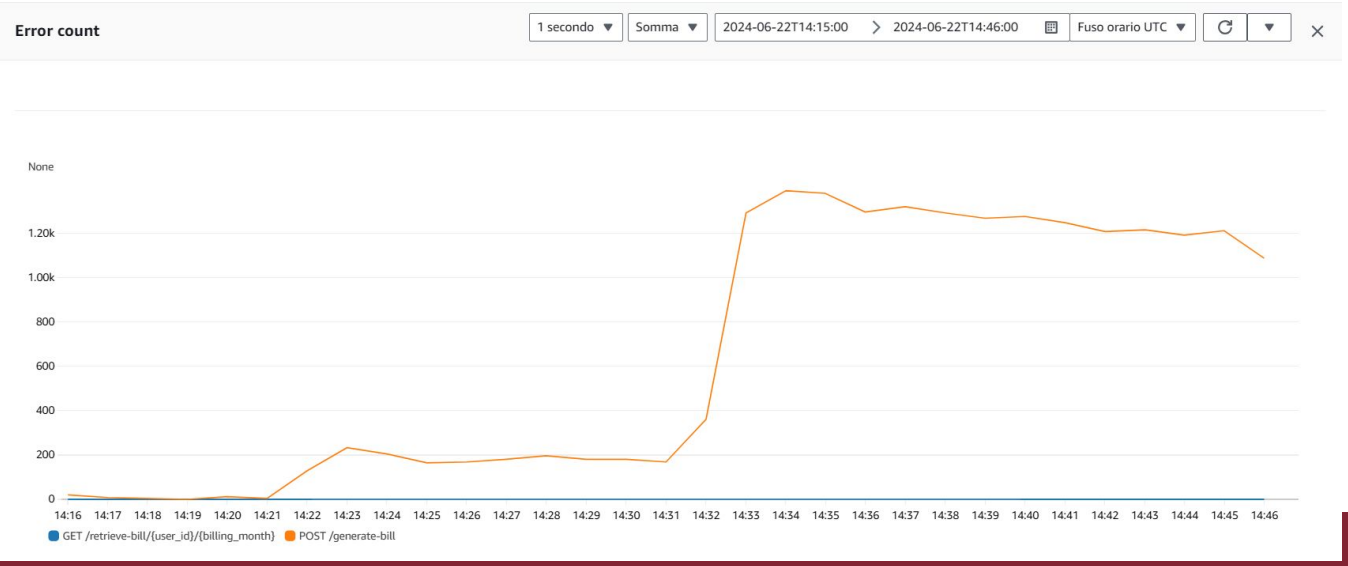
- AWS CloudWatch as a monitoring tool;
- Apache JMeter as a performance testing toolkit.

We have created a Test Plan with 3 different phases:

- Low phase: 5 minutes simulating 10 users (theads)
- Medium phase: 10 minutes simulating 50 users (threds)
- High phase: 15 minutes simulating 200 users (threads)

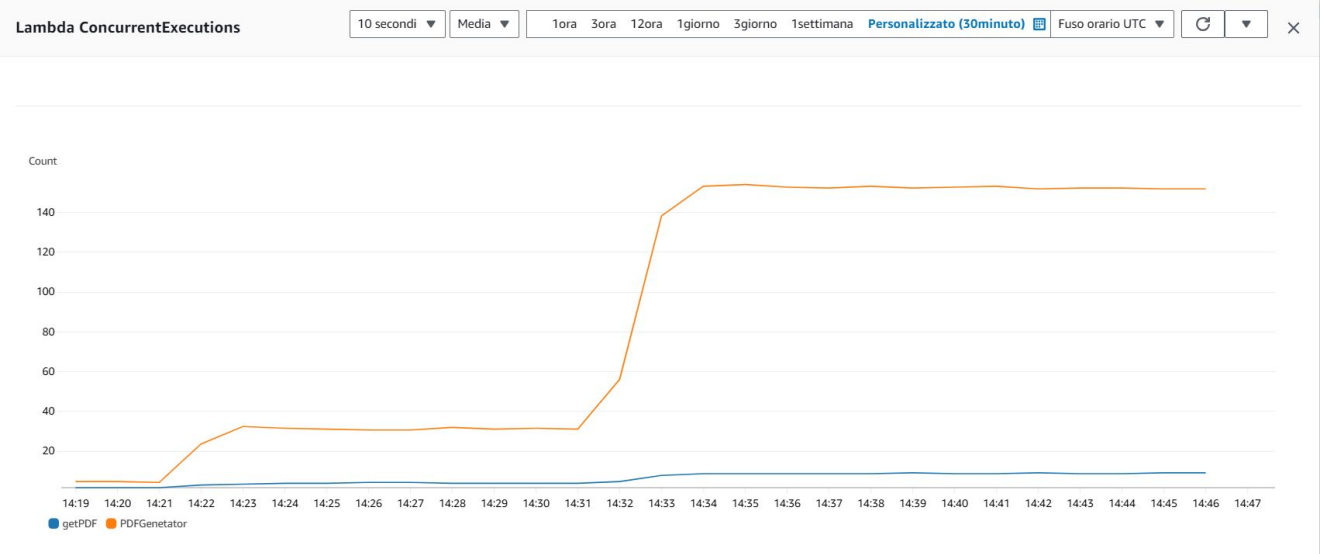
Testing - API Gateway Performance

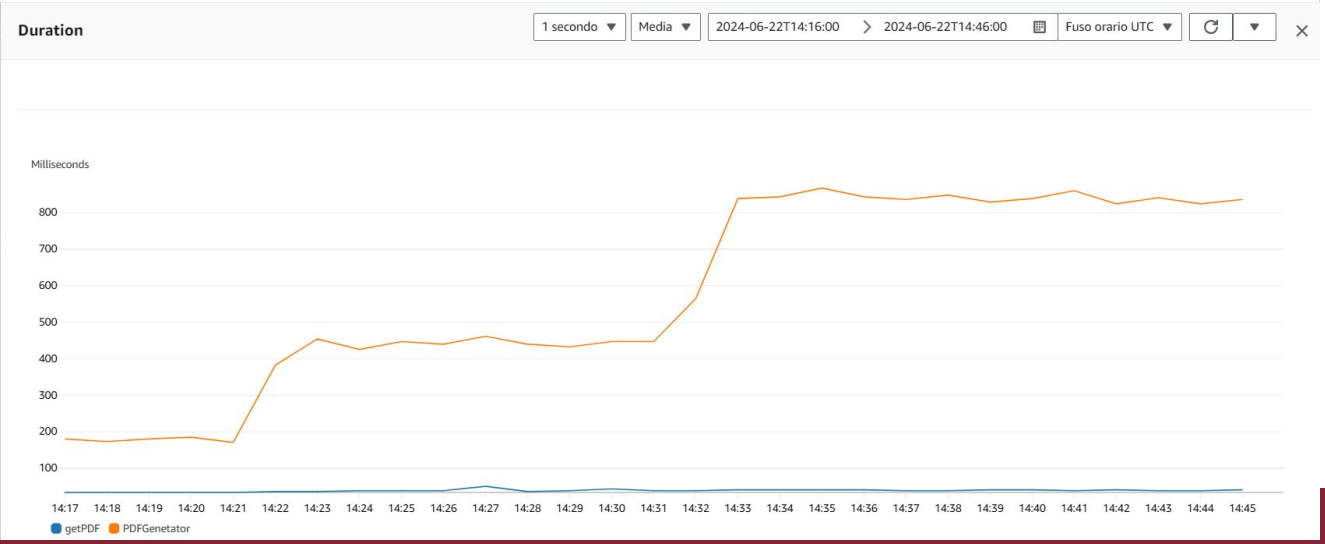




We can notice how the latency and error count for the ”generate-bill” method increase with the numbers of requests, instead the ”retrieve-bill” method remains stable.
This highlights how the first function is more computationally expensive than the other.

Testing - Lambda Function Performance



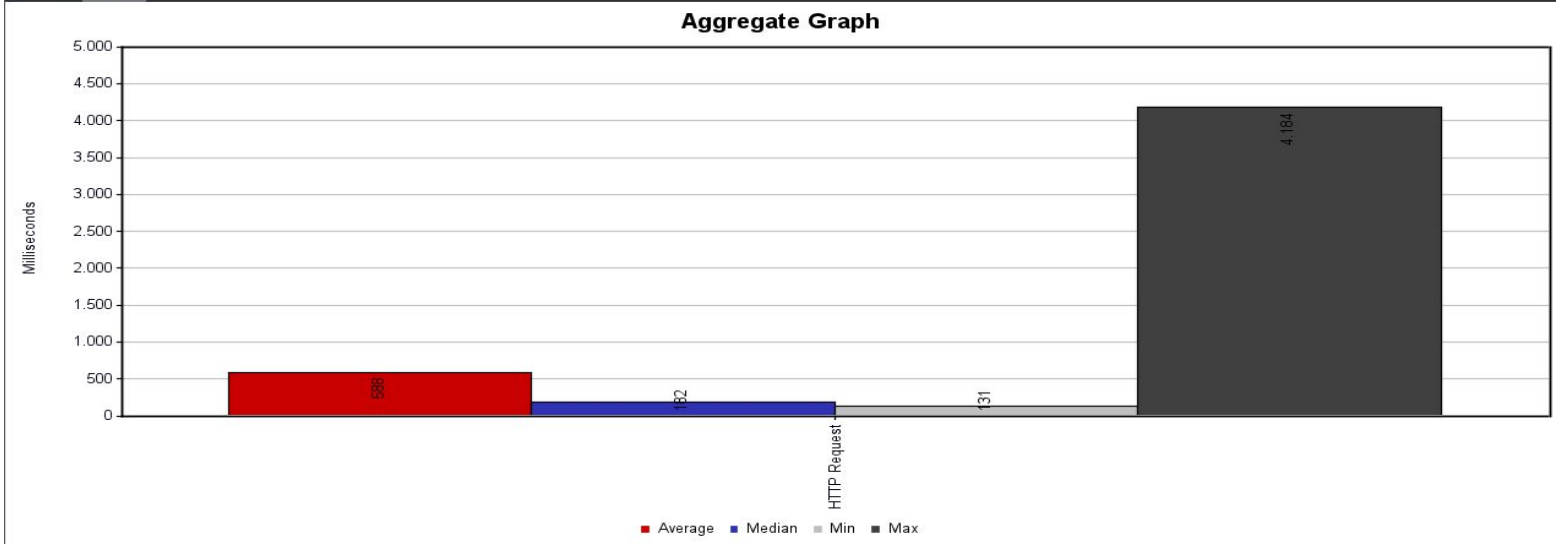


The ”generate-bill” function trends reflect the 3 phases of the test, with a peak of 150 concurrent executions during the third phase. Instead the ”retrieve-bil” has a much lower increment with a pick of 10 concurrent execution.

Testing - JMeter Report

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Maximum	Error %	Throughput
HTTP Request	287741	588	182	2005	3141	3540	131	4184	6,27%	318,6/sec
TOTAL	287741	588	182	2005	3141	3540	131	4184	6,27%	318,6/sec

Settings Graph



Although the average request duration ranges from 220 milliseconds to 600 milliseconds depending on the test, the maximum peak is almost stable, ranging from 3300 to 4100 milliseconds.

Thank you

Questions?

WattVault