How to measure the Software Carbon Intensity (SCI) of your Cloud Software with OpenTelemetry and Cloud Carbon Footprint

Andreas Brunnert

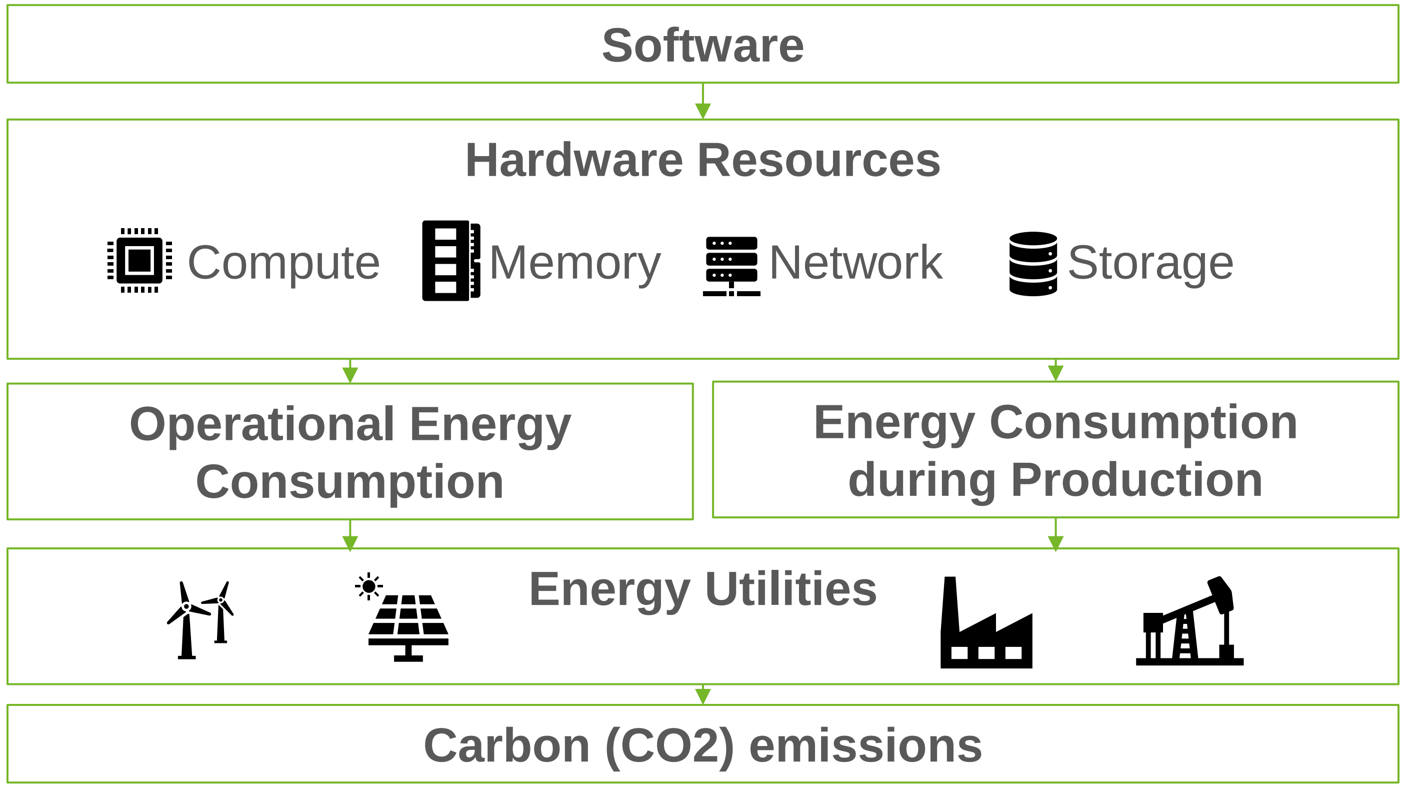
Professor @ University of Applied Sciences Munich HM
Founder @ RETIT GmbH

Agenda

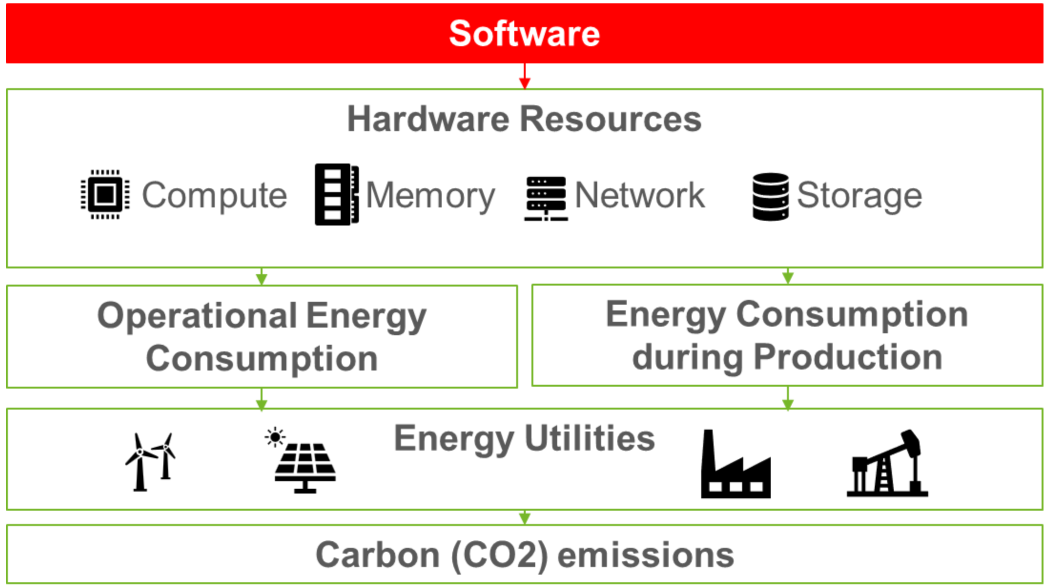
- How to measure the

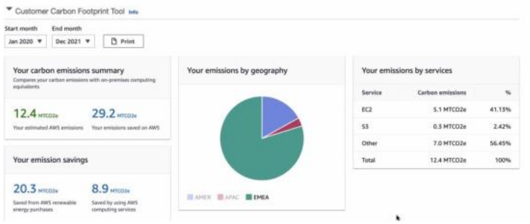
1. **Software Carbon Intensity (SCI)**
   - **…** of your cloud software with
2. **OpenTelemetry**
   - … and
3. **Cloud Carbon Footprint**

Software Carbon Emissions



Software Carbon Emissions




Customer Carbon Footprint Tool Info

Start month: Jan 2020 · End month: Dec 2021 · Print

Your carbon emissions summary
Compares your carbon emissions with on-premises computing equivalents

12.4 MTCO2e — Your estimated AWS emissions
29.2 MTCO2e — Your emissions saved on AWS

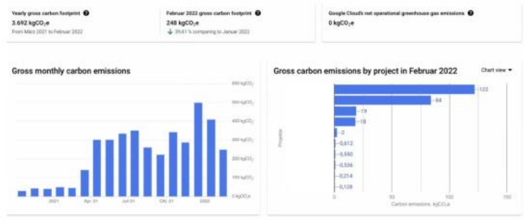
Your emission savings

20.3 MTCO2e — Saved from AWS renewable energy purchases
8.9 MTCO2e — Saved by using AWS computing services

Your emissions by geography

Your emissions by services

Service	Carbon emissions	%
EC2	5.1 MTCO2e	41.13%
S3	0.3 MTCO2e	2.42%
Other	7.0 MTCO2e	56.45%
Total	12.4 MTCO2e	100%

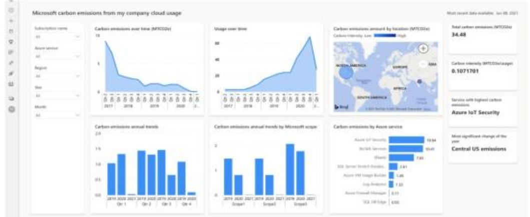

Yearly gross carbon footprint
3.692 kgCO₂e
From März 2021 to Februar 2022

Februar 2022 gross carbon footprint
248 kgCO₂e
39,41 % comparing to Januar 2022

Google Cloud's net operational greenhouse gas emissions
0 kgCO₂e

Gross monthly carbon emissions

Gross carbon emissions by project in Februar 2022


Microsoft carbon emissions from my company cloud usage

Total carbon emissions (MTCO2e): 34.48
Carbon intensity (MTCO2e/usage): 0.1071701
Service with highest carbon emissions: Azure IoT Security
Most significant change of the year: Central US emissions

Carbon emissions over time (MTCO2e)
Usage over time
Carbon emissions amount by location (MTCO2e)
Carbon emissions annual trends
Carbon emissions annual trends by Microsoft scope
Carbon emissions by Azure service
Azure IoT Security
Scope1
Scope2
Scope3

Software Carbon Intensity (SCI) Specification

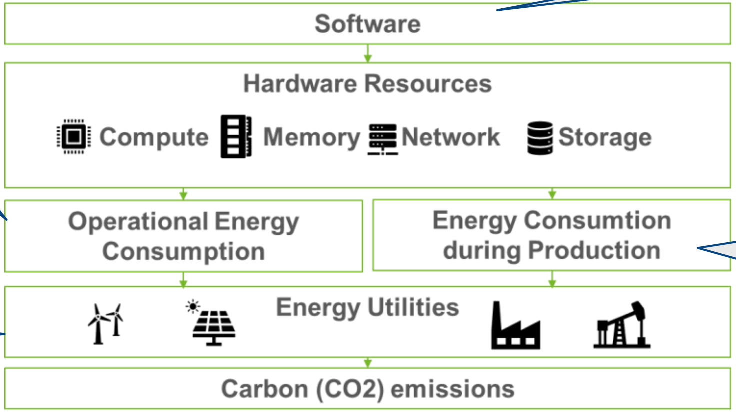
$$SCI = ((E * I) + M) \text{ per } R$$

E = This is the energy consumed by a software system for a functional unit of work.

R = Functional Unit; this is how software scales, for example per user or per device

I = Carbon emitted per kWh of energy, gCO2eq/kWh

M = Embodied carbon of the hardware that the software is running on



- See https://www.iso.org/standard/86612.html and https://sci.greensoftware.foundation/ for more details

Software Carbon Intensity (SCI) Specification

- ***SCI = ((E * I) + M) per R***

E = This is the energy consumed by a software system for a functional unit of work.

R = Functional Unit; this is how software scales, for example per user or per device

- *E needs to be measured **for a specific** software*
- *R needs to be measured **for a specific** software*
- *I needs to be collected from sources **outside** the software*
- *M needs to be collected from sources **outside** the software*

I = Carbon emitted per kWh of energy, gCO2eq/kWh

M = Embodied carbon of the hardware that the software is running on

Software Carbon Intensity (SCI) Specification

- ***SCI = ((E * I) + M) per R***
- *E needs to be measured **for a specific** software*
- *R needs to be measured **for a specific** software*
- *I needs to be collected from sources **outside** the software*
- *M needs to be collected from sources **outside** the software*

OpenTelemetry

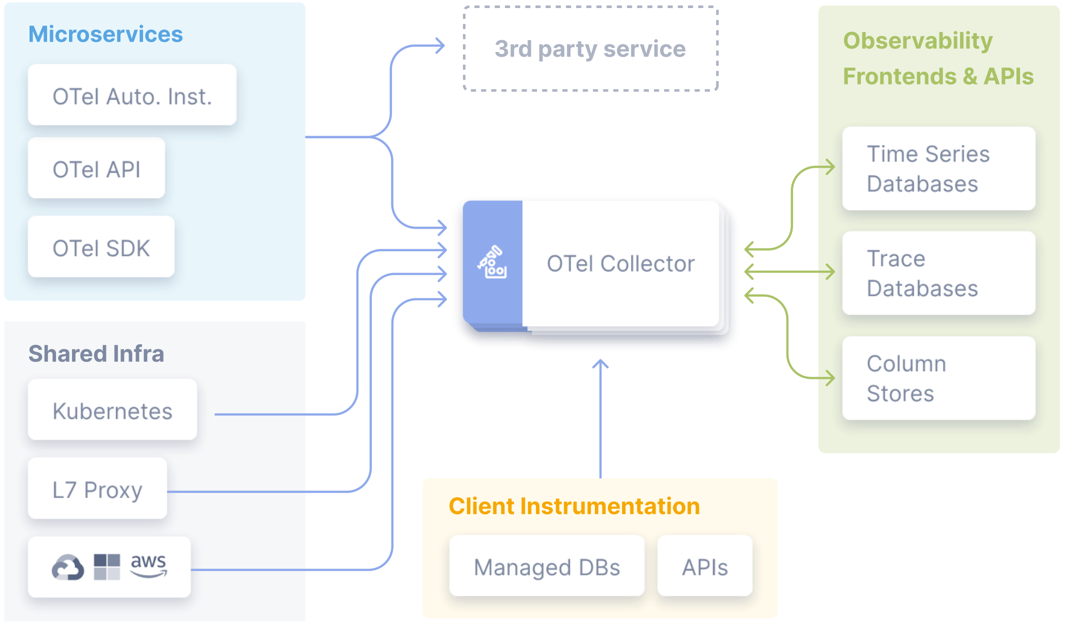
https://opentelemetry.io

Cloud Carbon Footprint

https://www.cloudcarbonfootprint.org

OpenTelemetry

SCI = *R*



https://opentelemetry.io/docs/

OpenTelemetry

- Our OpenTelemetry extension that also collects the resource demands for Java-based software systems (https://github.com/RETIT/opentelemetry-javaagent-extension)

```
YourService {
    yourAPI {
        $d_{before}$ = measureResourceDemandBefore()

        doBusinessWork(…)

        $d_{after}$ = measureResourceDemandAfter()

        $d = d_{after} - d_{before}$
    }
}
```

$$\Rightarrow \begin{pmatrix} d_{CPU} \\ d_{STO_r} \\ d_{STO_w} \\ d_{MEM} \\ d_{NET_i} \\ d_{NET_o} \end{pmatrix}$$

d	= Resource Demand
c	= carbon emissions
STO	= Storage
r	= read
w	= write
MEM	= Memory
NET	= Network
i	= Input
o	= Output

OpenTelemetry + Cloud Carbon Footprint Data

SCI = ((E) +) per R

E for CPU

$$P_{CPU} = P_{CPUmin} + (CPU_{util} * (P_{CPUmax} - P_{CPUmin}))$$

$$P_{CPUinclPUE} = P_{CPU} * PUE$$

$$CPU_{utilT} = (\sum_{t_0}^{t_n} d_{CPU}) \div (CPU_{util} * (t_n - t_0))$$

$$P_{TCPU} = P_{CPUinclPUE} * CPU_{utilT}$$

$$E_{CPU} = (P_{TCPU} * (t_n - t_0))/1000$$

$P_{CPUmin/max}$ values are different values derived by SPECpower benchmarks depending on CPU type (can be found on the CCF website)

PUE (Power Usage Effectiveness) values are different constants depending on cloud provider (can be found on the CCF website)

E for MEM / STO / NET

$$E_{GB} = (P_{GB} * PUE * (t_n - t_0))/1000$$

$$\sum_{t_0}^{t_n} d_{MEM} * 10^{-9} * E_{GB}$$

$$\sum_{t_0}^{t_n} d_{NET} * 10^{-9} * E_{GB}$$

$$\sum_{t_0}^{t_n} d_{STO} * 10^{-9} * E_{GB}$$

P_{GB} values are different constants depending on MEM, STO and NET type (can be found on the CCF website)

Cloud Carbon Footprint

https://www.cloudcarbonfootprint.org

Quelle: Brunnert / Gutzy (2024): Extending the OpenTelemetry Java Auto-Instrumentation Agent to Publish Green Software Metrics (Symposium on Software-Performance / Software-Technik-Trends)

OpenTelemetry + Cloud Carbon Footprint Data

$$SCI = ((E * I) + \quad) \text{ per } R$$

$$\begin{pmatrix} E_{CPU} \\ E_{STO_r} \\ E_{STO_w} \\ E_{MEM} \\ E_{NET_i} \\ E_{NET_o} \end{pmatrix} * \begin{matrix} \textbf{Grid} \\ \textbf{Emission} \\ \textbf{Factor} \\ \textbf{(GEF)*} \end{matrix} = \begin{pmatrix} c_{CPU} \\ c_{STO_r} \\ c_{STO_w} \\ c_{MEM} \\ c_{NET_i} \\ c_{NET_o} \end{pmatrix} \Rightarrow c_{CPU} + c_{STO_r} + c_{STO_w} + c_{MEM} + c_{NET_i} + c_{NET_o} = c_T$$

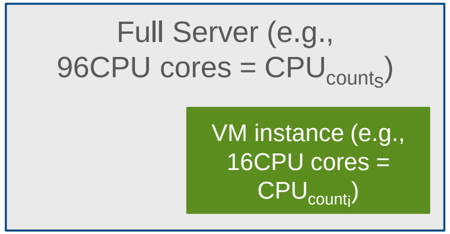
*CCF publishes Grid Emission Factors (GEF) for the different regions in which the cloud providers operate (https://www.cloudcarbonfootprint.org/docs/methodology/#carbon-estimates-co2e)

OpenTelemetry + Cloud Carbon Footprint Data

SCI = ((E * I) + M) per R

- The information about the total embodied emissions (TEE) is also available from Cloud Carbon Footprint (CCF) for most of the instance types of the main cloud providers (https://www.cloudcarbonfootprint.org/docs/methodology/#embodied-emissions)
  - The TEE value first need to be scaled down to the time the server used and the fraction of the resources used by the current instance type:

$$M = TEE * 0.0289^{[1]} * (t_n - t_0) * (CPU_{count_i} / CPU_{count_s})$$


Full Server (e.g., 96CPU cores = CPU_{counts})
VM instance (e.g., 16CPU cores = CPU_{counti})

  - Furthermore, we cannot allocate the full embodied carbon (***M***) on each transaction type, therefore we reduce it to the fraction used by the current transaction

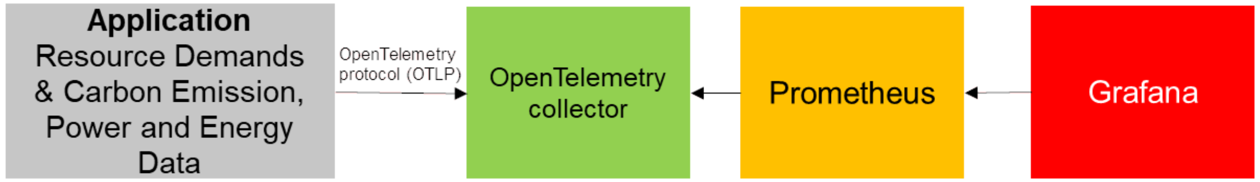
$$M_T = M * CPU_{utilT}$$

[1] 0.0289 = 1000 (kg to g) ÷ 4 (years of server usage) ÷ 12 (months per year) ÷ 30 (days per month) ÷ 24 (hours per day

OpenTelemetry + Cloud Carbon Footprint Data

$SCI = ((E * I) + M) \text{ per } R = c_T + M_T$

We can omit the R as our data is already scoped to individual transactions (T)

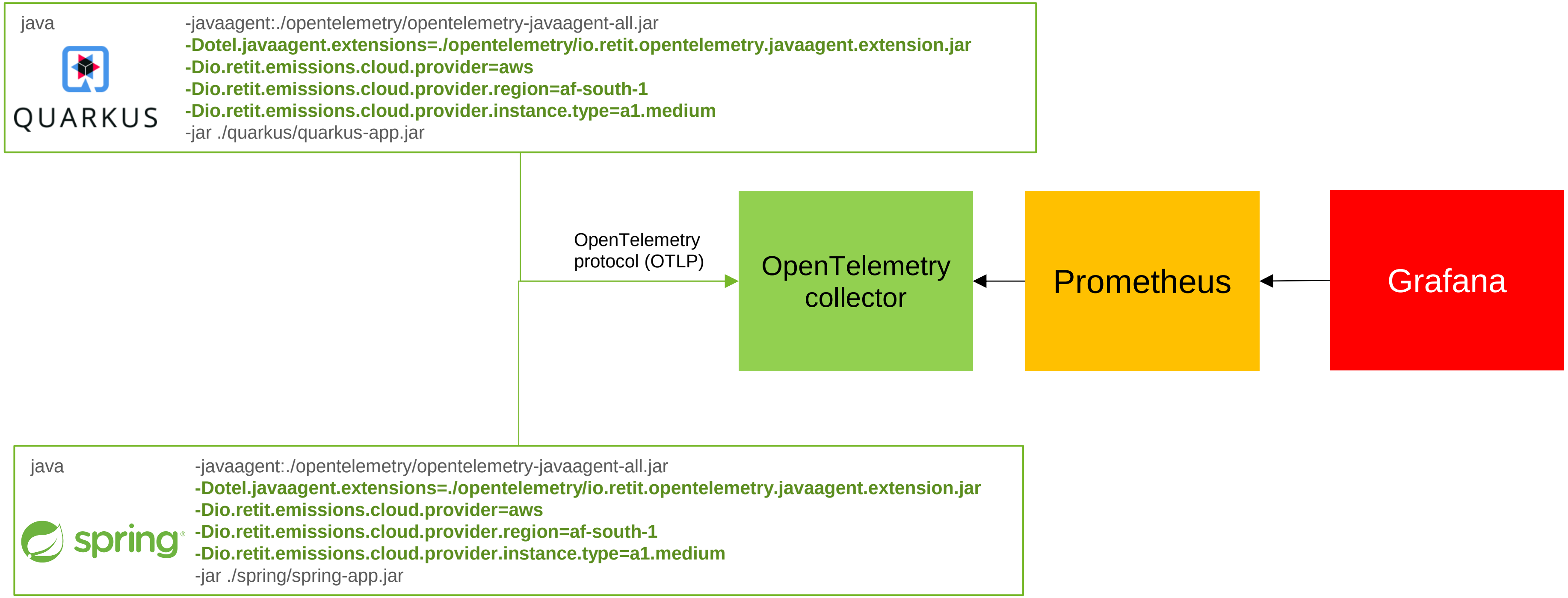

Application
Resource Demands & Carbon Emission, Power and Energy Data
OpenTelemetry protocol (OTLP)
OpenTelemetry collector
Prometheus
Grafana

Extension to the OpenTelemetry Java Agent

- Publishes the resource demands per Span and as Metrics
- Metrics only for top level transactions

https://github.com/RETIT/opentelemetry-javaagent-extension

Demo of the OpenTelemetry Extension!

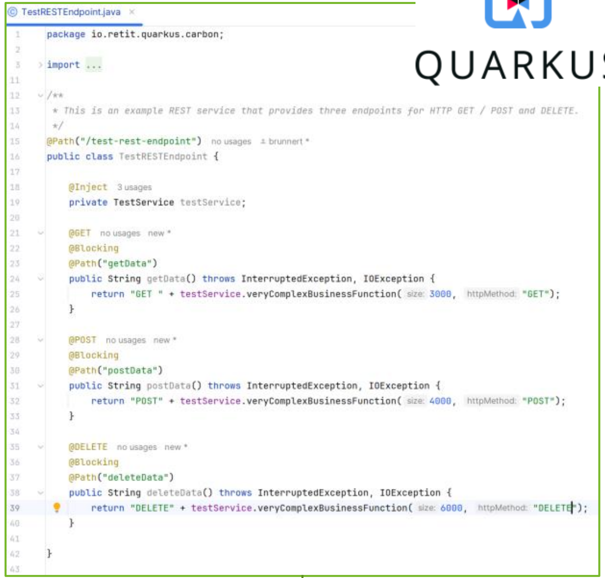


```
java    -javaagent:./opentelemetry/opentelemetry-javaagent-all.jar
        -Dotel.javaagent.extensions=./opentelemetry/io.retit.opentelemetry.javaagent.extension.jar
        -Dio.retit.emissions.cloud.provider=aws
        -Dio.retit.emissions.cloud.provider.region=af-south-1
        -Dio.retit.emissions.cloud.provider.instance.type=a1.medium
        -jar ./quarkus/quarkus-app.jar
```

```
java    -javaagent:./opentelemetry/opentelemetry-javaagent-all.jar
        -Dotel.javaagent.extensions=./opentelemetry/io.retit.opentelemetry.javaagent.extension.jar
        -Dio.retit.emissions.cloud.provider=aws
        -Dio.retit.emissions.cloud.provider.region=af-south-1
        -Dio.retit.emissions.cloud.provider.instance.type=a1.medium
        -jar ./spring/spring-app.jar
```

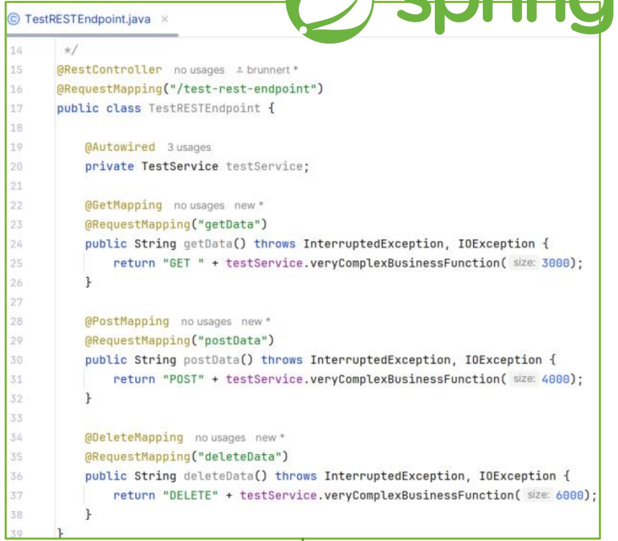
Note: The instance type is required for embodied emissions and to know the specific processor type

Demo of the OpenTelemetry Extension!



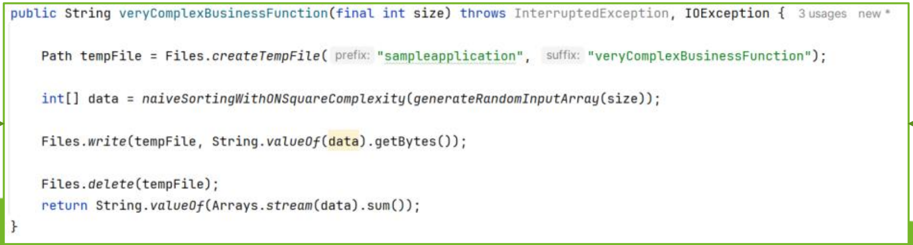
TestRESTEndpoint.java

```java
package io.retit.quarkus.carbon;

import ...

/**
 * This is an example REST service that provides three endpoints for HTTP GET / POST and DELETE.
 */
@Path("/test-rest-endpoint")  no usages  brunnert *
public class TestRESTEndpoint {

    @Inject  3 usages
    private TestService testService;

    @GET  no usages  new *
    @Blocking
    @Path("getData")
    public String getData() throws InterruptedException, IOException {
        return "GET " + testService.veryComplexBusinessFunction( size: 3000,  httpMethod: "GET");
    }

    @POST  no usages  new *
    @Blocking
    @Path("postData")
    public String postData() throws InterruptedException, IOException {
        return "POST" + testService.veryComplexBusinessFunction( size: 4000,  httpMethod: "POST");
    }

    @DELETE  no usages  new *
    @Blocking
    @Path("deleteData")
    public String deleteData() throws InterruptedException, IOException {
        return "DELETE" + testService.veryComplexBusinessFunction( size: 6000,  httpMethod: "DELETE");
    }

}
```



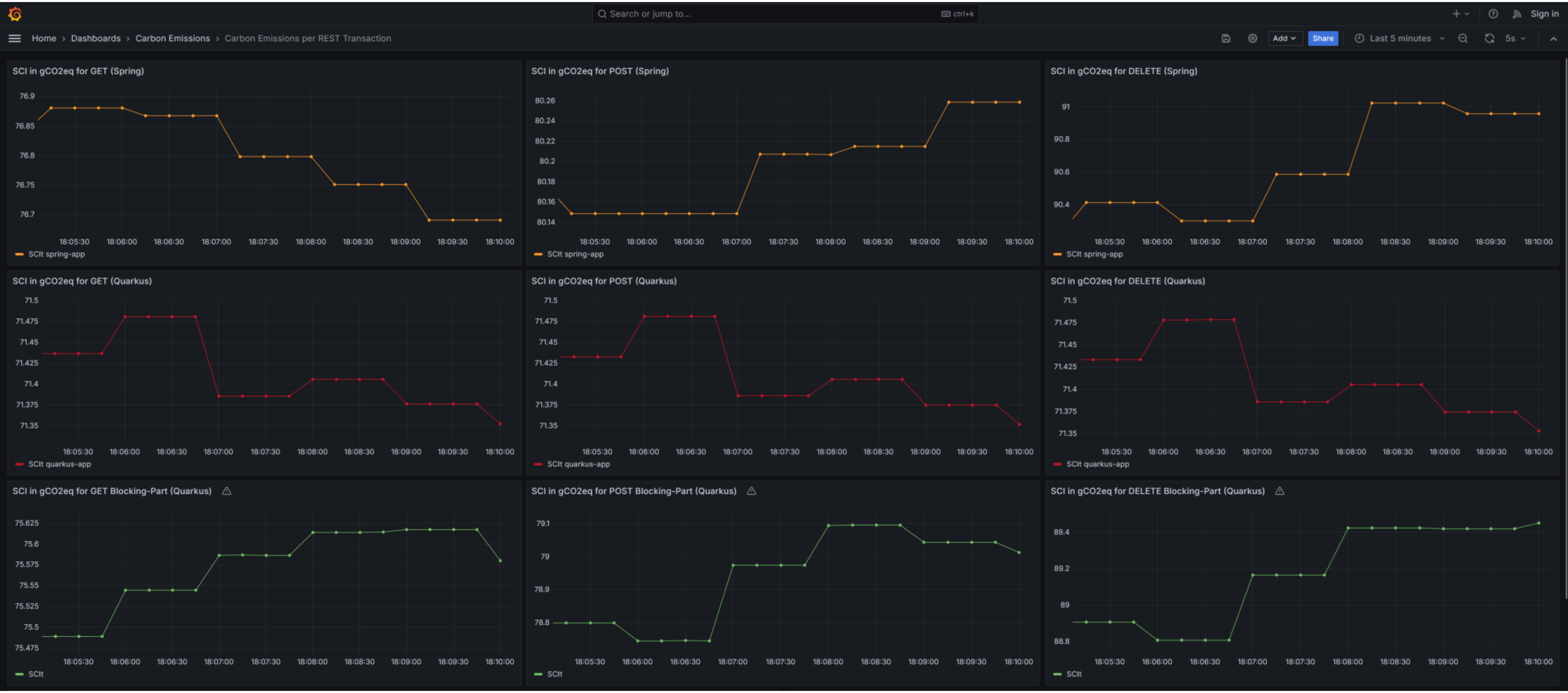
TestRESTEndpoint.java

```java
 */
@RestController  no usages  brunnert *
@RequestMapping("/test-rest-endpoint")
public class TestRESTEndpoint {

    @Autowired  3 usages
    private TestService testService;

    @GetMapping  no usages  new *
    @RequestMapping("getData")
    public String getData() throws InterruptedException, IOException {
        return "GET " + testService.veryComplexBusinessFunction( size: 3000);
    }

    @PostMapping  no usages  new *
    @RequestMapping("postData")
    public String postData() throws InterruptedException, IOException {
        return "POST" + testService.veryComplexBusinessFunction( size: 4000);
    }

    @DeleteMapping  no usages  new *
    @RequestMapping("deleteData")
    public String deleteData() throws InterruptedException, IOException {
        return "DELETE" + testService.veryComplexBusinessFunction( size: 6000);
    }
}
```



```java
public String veryComplexBusinessFunction(final int size) throws InterruptedException, IOException {  3 usages  new *

    Path tempFile = Files.createTempFile( prefix: "sampleapplication",  suffix: "veryComplexBusinessFunction");

    int[] data = naiveSortingWithONSquareComplexity(generateRandomInputArray(size));

    Files.write(tempFile, String.valueOf(data).getBytes());

    Files.delete(tempFile);
    return String.valueOf(Arrays.stream(data).sum());
}
```

Demo of the OpenTelemetry Extension!



References

- OpenTelemetry Java Agent:
  - https://github.com/open-telemetry/opentelemetry-java-instrumentation

- OpenTelemetry Java-Agent Extension:
  - https://github.com/RETIT/opentelemetry-javaagent-extension

- Paper about the OpenTelemetry Extension:
  - https://fb-swt.gi.de/fileadmin/FB/SWT/Softwaretechnik-Trends/Verzeichnis/Band_44_Heft_4/SSP24_16_camera-ready_5255.pdf

- Cloud Carbon Footprint – Methodology:
  - https://www.cloudcarbonfootprint.org/docs/methodology

- Cloud Carbon Footprint – Coefficients:
  - https://github.com/cloud-carbon-footprint/ccf-coefficients

Thanks a lot for your attention.

Questions?

Andreas Brunnert

brunnert@retit.de

Resource Efficient Technologies & IT Systems