SafelyYou Coding Challenge: Monitoring the Fleet

Background

At SafelyYou, we’re responsible for the **uptime** and **performance** of our equipment installed at customer facilities. This includes but is not limited to:

- Cameras/Sensors
- Servers
- Switches

Because we do AI detection at the edge, network availability is one of the most important metrics of service availability and performance. For each device, we generally need to know **when and how long it is connected to our services and running correctly** and **what is the upload speed of a device to our cloud services**.

For reporting purposes we are interested in:

- What the avg video upload time of a device is.
- Whether a device is operating normally (via a “heartbeat” that communicates with our cloud services).

Goals

You will be provided with a list of devices, and a device simulator that will send telemetry data to two endpoints. Your task is to implement the endpoints’ contracts and provide calculations of per device statistics to determine a device’s uptime and its upload performance.

Input

Device Simulator

The table below provides links to the device-simulator for various platforms and architectures. You will run the simulator against your API and it will compare its responses to what is expected and output results.

Platform	Architecture	Link
Linux	AMD64	Download
Linux	ARM64	Download
Mac OS X	AMD64	Download
Mac OS X	ARM64	Download
Windows	AMD64	Download
Windows	ARM64	Download

Device Definitions

A CSV file with device definitions is provided. You will use this to create devices in your API that the device simulator will send metrics for. You can download it here.

OpenAPI Specification

There are three endpoints that are required of your API described briefly below, **but the full contract you need to implement is in `openapi.json` that you can download here.**

devices.csv

Field	Type	Description
device_id	string	The ID of the device

POST /devices/{device_id}/heartbeat

Request:

Field	Type
sent_at	date-time

POST /devices/{device_id}/stats

Request:

Field	Type
sent_at	date-time
upload_time	integer

GET /devices/{device_id}/stats

Response:

Field	Type
uptime	Float/Double
avg_upload_time	string

Calculating Uptime

Each device sends a "heartbeat" every minute. A device is considered "OFFLINE" if there is no heartbeat for that minute. Your uptime calculation will look like:

```
uptime = (sumHeartbeats / numMinutesBetweenFirstAndLastHeartbeat) *
100
```

Calculating Average Upload Time

```
timeDuration = avg(arrayOfUploadTimeDurations)
```

Deliverables

Your code should read the `devices.csv` on startup and begin listening on a port. The device simulator will accept the port number of your service and begin sending data. At the end it will query the stats endpoint for each device and output the results to `results.txt` and the console. You can use the results to determine correctness of your code (**Note:** your results precision may differ slightly from that expected by the device simulator due to your runtime environment. This will not count against you).

- Please submit your code in Github, Gitlab or a zip file containing your local git repo of your code. Github and Gitlab are preferred as many email providers will flag email attachments containing code as malicious.
- Include instructions for how to run it, and the `results.txt` in your submission.
- Include a short write up of your solution, answering the following questions to the best of your ability:
    - How long did you spend working on the problem? What did you find to be the most difficult part?
    - How would you modify your data model or code to account for more kinds of metrics?
    - Discuss your solution's runtime complexity

Evaluation

Your program will be evaluated based on the following criteria, in decreasing order of importance:

1. Clarity – Is your code well-organized and easy to read?
2. Correctness - Does your program implement the API contract? Are the outputs in line with the expected results?
3. Performant - Code should be safe to run in a production environment. Shortcuts or exceptions to this should be identified and explained.
4. Extensibility – Is your solution architected in a manner that makes it easy to collaborate with multiple engineers, and allow them to add and modify features in a consistent, testable way? This includes but is not limited to: unit tests, documentation.

Final Notes and Suggestions **Important! Please read carefully:**

- We strongly suggest first implementing a simple, well-written, correct, and (reasonably) performant program before attempting to optimize it further.
- Your solution **must** be written in Go, unless prior arrangements have been made.
- If you are in a time pinch, it should take you at most 2-3 hours to write a minimal working program and some thoughtful analysis. If you have more time and are confident in your working solution, we encourage you to elaborate on your solution to showcase your architecture and code organization skills, possibly implementing some of your ideas from the write-up questions. If you do expand on your solution after completing the basic requirements, consider using git to commit working checkpoints, in order for us to see your progress and give credit for past versions if the final version does not pan out.

More Tips:

- Unlike in many other coding challenges, your program’s clarity and efficiency is more important than its optimality.

- You may also use any external libraries that you wish, but you really shouldn’t need to use anything fancy (e.g. databases).