Home task - Trace reconstruction from logs

Introduction

Each application in a microservice environment outputs some log describing the boundaries of an HTTP request, with the following format:

[start-timestamp] [end-timestamp] [trace] [service-name] [caller-span]->[span]

The trace ID is a random string that is passed along every service interaction. The first service (called from outside) generates the string and passes it to every other service it calls during the execution of the request. The called services take the trace (let's say, from an HTTP header) and also pass it to the services the call themselves.
The span ID is generated for every request. When a service calls another, it passes its own span ID to the callee. The callee will generate its own span ID, and using the span passed by the caller log the last part of the line, that allows to connect the requests.
So, a trace could look like this:

2016-10-20T12:43:34.000Z 2016-10-20T12:43:35.000Z trace1 back-end-3 ac->ad
2016-10-20T12:43:33.000Z 2016-10-20T12:43:36.000Z trace1 back-end-1 aa->ac
2016-10-20T12:43:38.000Z 2016-10-20T12:43:40.000Z trace1 back-end-2 aa->ab
2016-10-20T12:43:32.000Z 2016-10-20T12:43:42.000Z trace1 front-end null->aa

Meaning that the "front-end" received a call from the outside ("null") and assigned the span "aa" to the request. Then it called back-end-1, who assigned the span "ac", who in turn called service "back-end-3", who assigned span "ad". Then, "front-end" called "back-end-2".

The entries are logged when the request finishes (as they contain the finishing time), so they are not in calling order, but in finishing order. Logs can be mixed up a bit (just because enforcing FIFO semantics is hard in distributed setups), but it is expected that the vast majority are only off for a few milliseconds.

Timestamps are in UTC.

This execution trace can then be represented as:


```
{
“id”: “trace1”,
“root”: {
“service”: “front-end”,
“start”: “ 2016-10-20T12:43:32.000Z”,
“end”: “2016-10-20T12:43:42.000Z”,
“calls”: [
{
“service”: “back-end-1”,
“start”: “ 2016-10-20T12:43:33.000Z”,
“end”: “2016-10-20T12:43:36.000Z”,
“calls”: [
{
“service”: “back-end-3”,
“start”: “ 2016-10-20T12:43:34.000Z”,
“end”: “2016-10-20T12:43:35.000Z”,
“calls”: [],
“span”: “ad”
}
],
“span”: “ac”
},
{
“service”: “back-end-2”,
“start”: “ 2016-10-20T12:43:38.000Z”,
“end”: “2016-10-20T12:43:40.000Z”,
“calls”: [],
“span”: “ab”
}
],
“span”: “aa”
}
}
```


The task is to produce these JSON trees. That is, given a sequence of log entries, output a JSON for each trace. We should imagine that this application could be deployed as part of some pipeline that starts at the source of the data and ends in some other monitoring application, that presents a stream of recent traces.