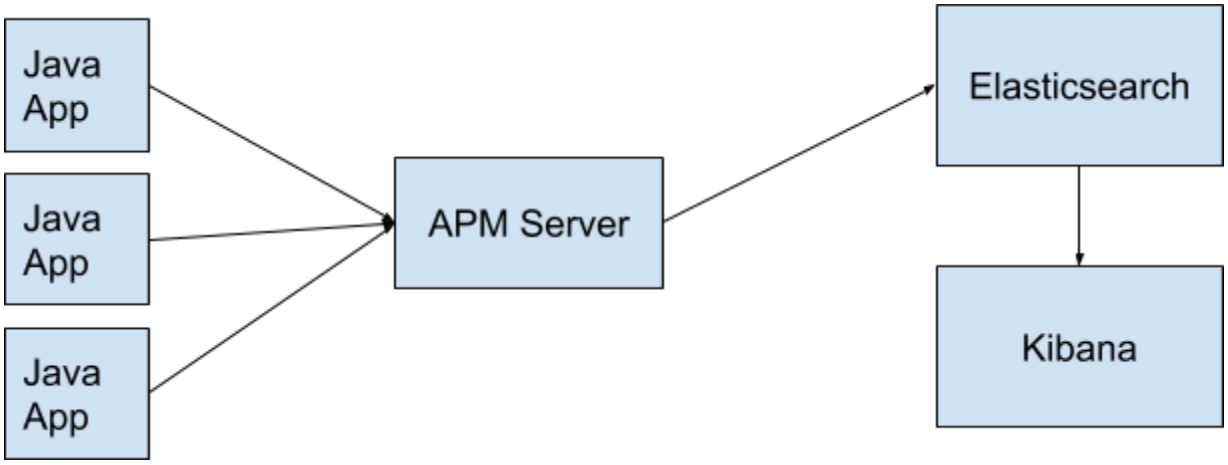
In this document, I will try to discuss some solutions for the FilesFromYou assignment.

Since the assignment's main assumption is that we are dealing with some client's platform that is suffering from very high CPU usage, I would argue against creating any extra API service for sending debug information to our backend server. We already have a problem, high CPU usage, so we should not make it worse.

Instead, I would use a service that runs alongside the client. A watchdog process that monitors the client and sends reports to a server.

One easy and quick solution can be to use the Elastic APM service. It has a java agent that will collect and send system and application metrics and logs to the APM server and then the APM server will transform these data and in the end, will send them to Elasticsearch. Once metrics and logs are stored in Elasticsearch, we can explore the application performance by using Kibana built-in APM dashboards. Elastic APM allows us to monitor applications in real-time and collect detailed performance metrics, system/application logs, and much other useful information.



Another approach can be creating a service that would work in a very similar manner. It can collect data on CPU usage caused by the client, establish a baseline for the system, and if the client application observes a deviation from "normal" then assemble an aggregated and depersonalized report and POST it as a JSON document to a simple REST endpoint. This application can save the data in some sort of buffer. Then an asynchronous, non-blocking function can consume this data and send it to our REST endpoint through HTTP. This service can retry sending data until it receives an acknowledgment (HTTP 200) from the backend server. In this way, we will not lose any of the matrices if our backend server is down for a while.

How often does the client send data:

Only if there's something interesting to send. Such as, when a client crashes or the CPU consumption is abnormally high. Otherwise, we're going to end up with a lot of data that can be hard to use. On the other hand, if we need to produce real-time metrics, we can piggyback CPU metrics (avg/top/etc) on any existing periodic "ping" or "keep-alive" packets or on some (or all) existing API calls.

What happens if the backend is unavailable:

The application keeps buffering/spooling data to disk until the connection is reestablished and buffered data can be sent.

What happens if the client fails to send data because it consumed all the CPU:

The watchdog should be running as a separate process and should be able to take care of that issue. We should be able to prioritize the number of resources that the client and watchdog can use. If the client is running on a platform that won't allow running an additional external process, then it probably doesn't matter what we do, the platform will likely "kill -9" the client anyway.

What is "low" or "high" CPU consumption

No simple answer, this depends on the client platform, the number of cores, the model of CPU. Preferably we don't just declare what's "low" or "high". We should analyze the data and report significant deviations from the baseline, i.e. typical CPU usage pattern for the specific computer.

How does API look like, which protocol it uses, and why

Which information does the client pass to the service

- Client thread dump at the time of the crash

- Client system info (CPU model, memory capacity, etc)

- Baseline CPU usage pattern(s) on the system where the client was running

- CPU usage for the last N seconds before the crash (N depends on how "active" the client is)

- If the client is logging anything, any log messages for the last N seconds

How to configure service deployment for maximum efficiency

We can run the backend system in some cloud, deploy in different geo zones to keep down latency. We can use Kubernetes.

How the service handles scaling, restarts, crashes, etc..

Kubernetes

How to design the service for most performance and least latency

The service can have an async, non-blocking IO and if we are going to collect real-time CPU metrics from thousands of clients, we should be fine to ignore the loss of some data points. Therefore, we can use UDP instead of TCP.

Which database to use (don't need to pick specific, the family is fine, or just general thoughts on what the database needs to support to run the solution)

What kind of schema would it have

If you need to store everything the client sends

I think we can extract many different kinds of information from the reports sent by this service. That is why I would not go with one database. The system metrics can be stored in a time-series database since it can be used in some analytical monitoring tools like Grafana. Then we can easily aggregate this data and group it on different tags and dimensions to filter out useful information and produce meaningful graphs.

For shallow key/value sets (like system info, thread dump, baseline CPU usage) we can use JSON documents in elasticsearch. We can use elastic/kibana for client logs or similar. Since we are storing raw metrics and time-series, we can also use some distributed NO-SQL columnar DB like Cassandra or HBase to process raw data with spark jobs or similar. Apache Druid also has very powerful querying and aggregation functionality which can be used to write complex multi-dimensions analytical queries.

How to query persisted data to generate reports (sample query or a draft of the query would be good here)

Druid has numerous query types for various use cases. We can perform aggregation queries on Druid data sources.

Druid queries should be POST like following:

```
curl -X POST '<queryable_host>:<port>/druid/v2/?pretty' -H
'Content-Type:application/json' -H 'Accept:application/json' -d
@<query_json_file>
```

And a query_json_file and be like below for Timeseries queries:

```
{
  "queryType": "timeseries",
  "dataSource": "sample_datasource",
  "granularity": "hour",
  "filter": {
    "type": "and",
    "fields": [
      { "type": "selector", "dimension": "dimension1", "value": "value1" }
    ]
  },
  "aggregations": [
    { "type": "longSum", "name": "name1", "fieldName": "fieldName1" }
  ],
  "intervals": [ "2020-07-01T00:00:00.000/2020-07-01T00:00:00.000" ]
}
```

How does the report look

A bunch of graphs/diagrams/histograms mostly. One important diagram could be a histogram of system metrics (or combinations) for which we have detected abnormally high CPU usage or crash. Below that one, I would present a bunch of diagrams that show top/bottom avg/max CPU usages for each field from system info, like CPU model, total memory available, type of hard-drive (SSD/HDD), operating system, etc.
Another set of diagrams showing breakdowns of CPU usage by named threads, top 5 functions/methods names (if possible), the top 5 last log-line before CPU-usage related crash
Histograms of the hours of the day when crashes occur.

Which information can we draw from the report at a glance

Most common system configurations where the crash occurs.

Which information in the report helps us answer the question "which clients are affected"?

Since the assumption is that only some users are experiencing this issue, we can look at the user's system information such as operating system, CPU model, total memory available, type of hard drive, etc. Sometimes finding a common factor between a few clients that have the same issue can be really helpful to find the root of the problem.