**Practical Microservices: Correlation Tokens**

Yesterday I read a few resources about microservices ([http://microservices.io](http://microservices.io/), [Nginx blog](https://www.nginx.com/?post_type=post&s=microservices)) with very compact and useful description of common patterns and practices.

As a result, I still have a few practical things, that I can add to this topic. Let’s call this **Correlation Token Pattern**.

**Where it comes from?**

Doing microservices, we are creating a lot of interservice calls and events. It means that debugging and tracing whole the flow is not an elementary task anymore. Each call from client side can allocate tens of calls inside microservices infrastructure (in fact, even if we are trying to keep them not so chatty). During the integration tests and production usage, understanding of the flow of calls can be the very time-consuming task.

Even more, nowadays most of the calls are async and such things like a time ordering will not help to work with it. That is a clear drawback of low coupling and async flows.

**How can we resolve this unavoidable complexity?**

I will promote technique of correlation tokens as the way of consolidation requests and events into a single group with a single initiator. Each new flow (client’s call, scheduler job task) can be marked with a single token, which we can spread across all the downstream calls. And given each event of the system, we can find a full path from an initiator to floors of the call only by this token.

**What do I mean?**

I think that the easiest way to represent the idea — is in the practical example. The example has two services:

1. API, with only the single endpoint

*GET*[*http://localhost:3000/products*](http://localhost:3000/products)

2) Backend, with another endpoint

*GET*[*http://localhost:3001/products*](http://localhost:3000/products)

Client side call will look like this

*curl*[*localhost:3000/products*](http://localhost:3000/products)

Middleware example in *api.js* will assign *app\_token* to each request. In this example *app\_token*consists of the several fields:

1. req.method + req.path (for unique endpoint identification)
2. userId (for unique user representation, very useful for debugging and digging strange cases with users)
3. timestamp (to compare a sequence of the several flows in time order)

This *app\_token*will be injected in all the downstream calls following this client call. I find it useful to make this parameter required for any call inside your system, it will help a lot later.

I hope the usage of the token is clear now. One last step — is include this token to your log consolidation infrastructure (like ELK stack).

After it, you can just find any interesting call in Kibana and filter all the flow by the correlation token field.

Hope it will help and reduce troubleshooting time for you.