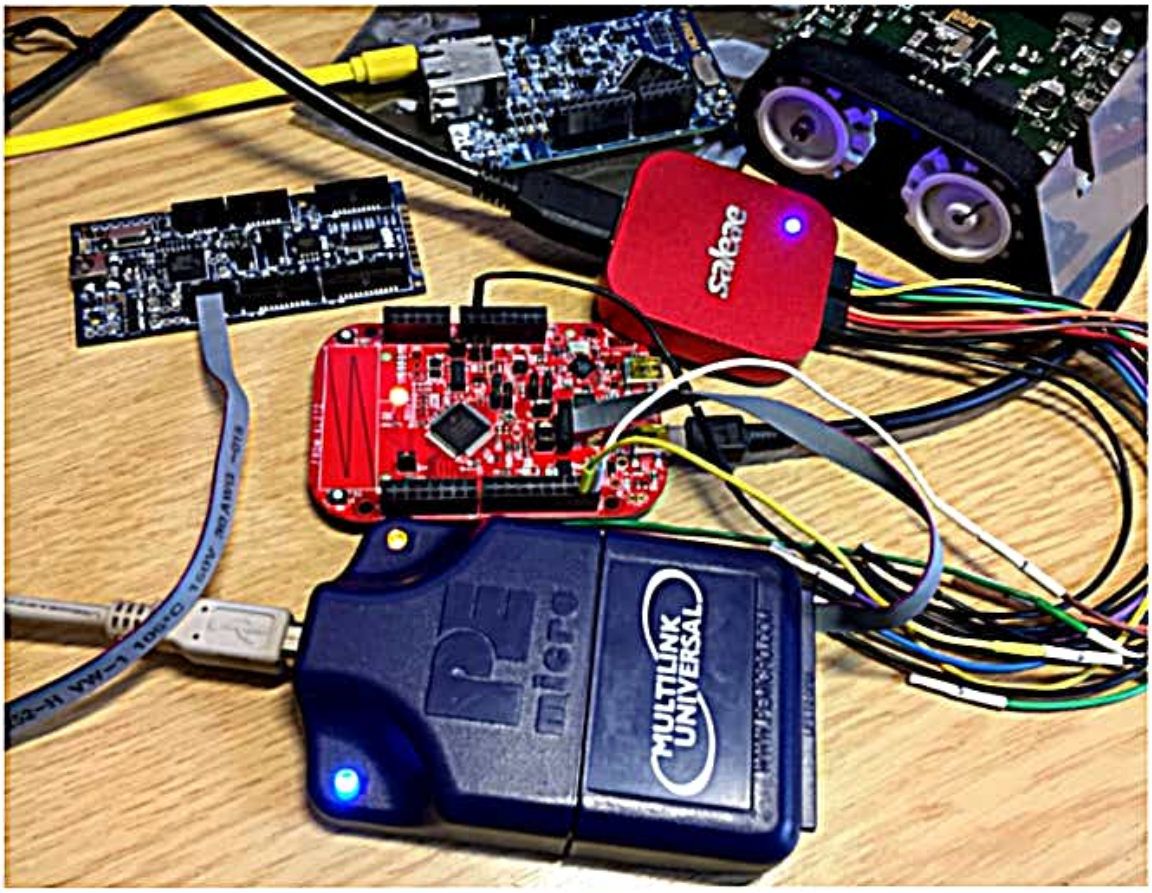
Questions from students or readers of my articles are a great source for all kind of articles. And here is the 'question of this week': "What is real-time debugging"?

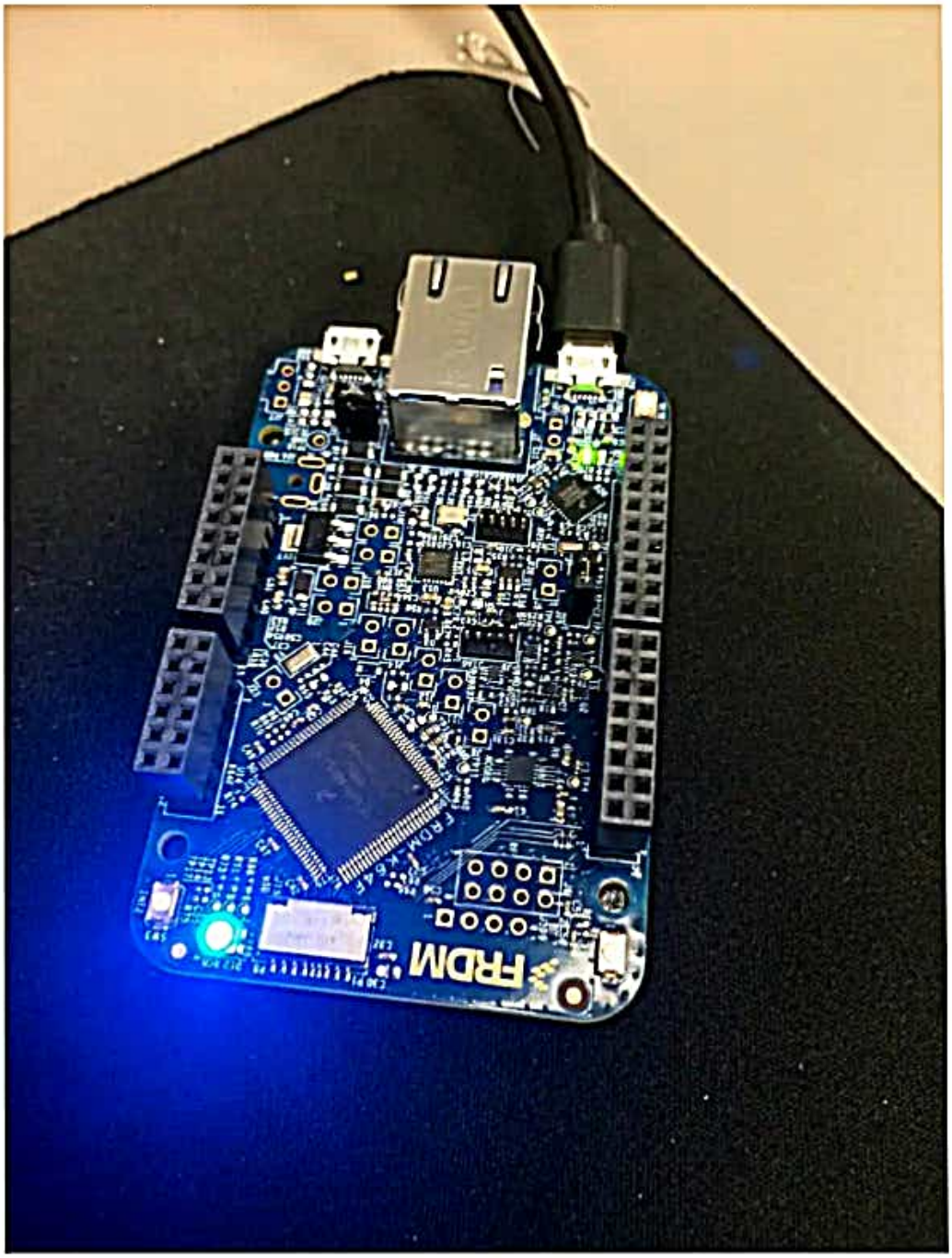
It's a good question because the topic of 'real-time' and 'debugging' was a topic in the lectures this week. So this question gives me the opportunity to combine the two things of 'real-time' and 'debugging' — I love it!



Debugging setup

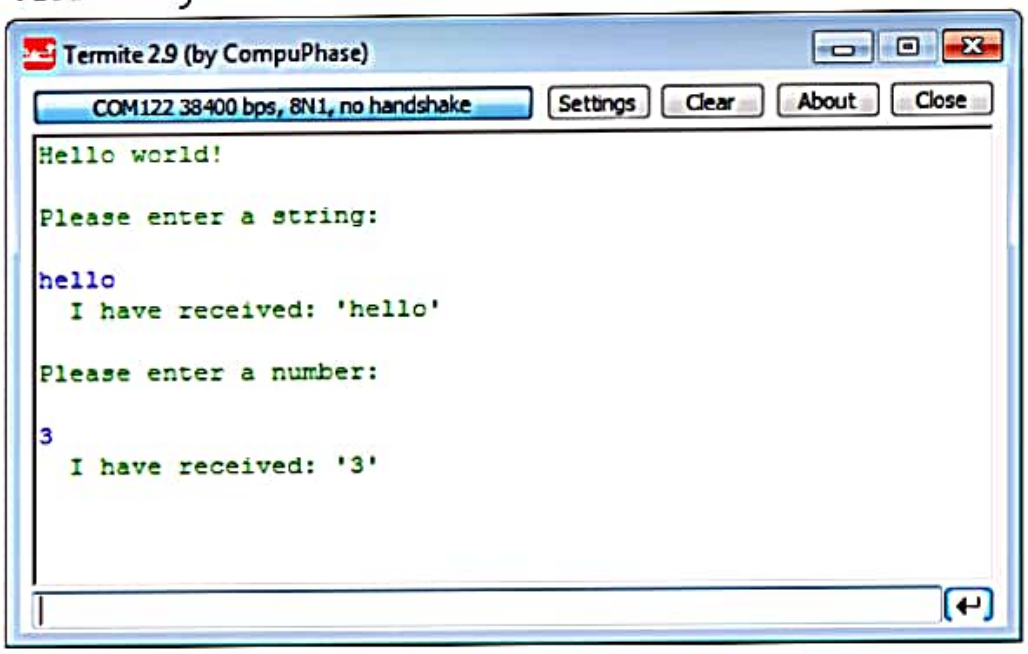
Debugging is probably the part in the development where the most time is spent, for many reasons. It is the part of the development process that usually can be planned the least. If you are looking for an overview about different debugging techniques, or if you are wondering what 'Real-time Debugging' means, then I hope this article is useful for you.

1. **LED blinking**: I could use an LED and watch the blinking to see if the system is operating correctly. I could use that LED in my code to show the status of the system, and based on the LED blinking, I would gain knowledge about what is going on. That certainly works for very simple and slow computer systems.



2. **Pin toggling**: For faster systems, I could record the LED or microcontroller pin with an oscilloscope or logic analyzer. That allows me to inspect faster signals, and I can pack more information into the signal by, say, sending

3. **Printf debugging**: I can issue more information by using a communication channel from the system, say sending and receiving text using a UART or USB connection. This is what I refer as 'printf debugging': By using, sending, and receiving text, I can inspect the target system and get visibility into its behavior:



4. **Stop-mode debugging**: The above approaches inspect a system that is constantly running. Another approach to halt the target so I can inspect it. That way, I can inspect multiple things, and when I'm finished, I let the system run again. This is what I refer as 'stop-mode debugging.' For this, the hardware/microcontroller needs special hardware built-in that allows us to halt the system for inspection. A commonly used technology for this is JTAG (or SWD on ARM cores, which is similar, but with fewer pin numbers). Typical debug probes in the ARM Cortex world are the P&E Multilink or the Segger J-Link. This usually needs some special

What Is Distributed Tracing?

Traces complement logs. While logs provide information about what happened inside the service, distributed tracing tells you what happened between services/components and their relationships. This is extremely important for microservices, where many issues are caused due to the failed integration between components.

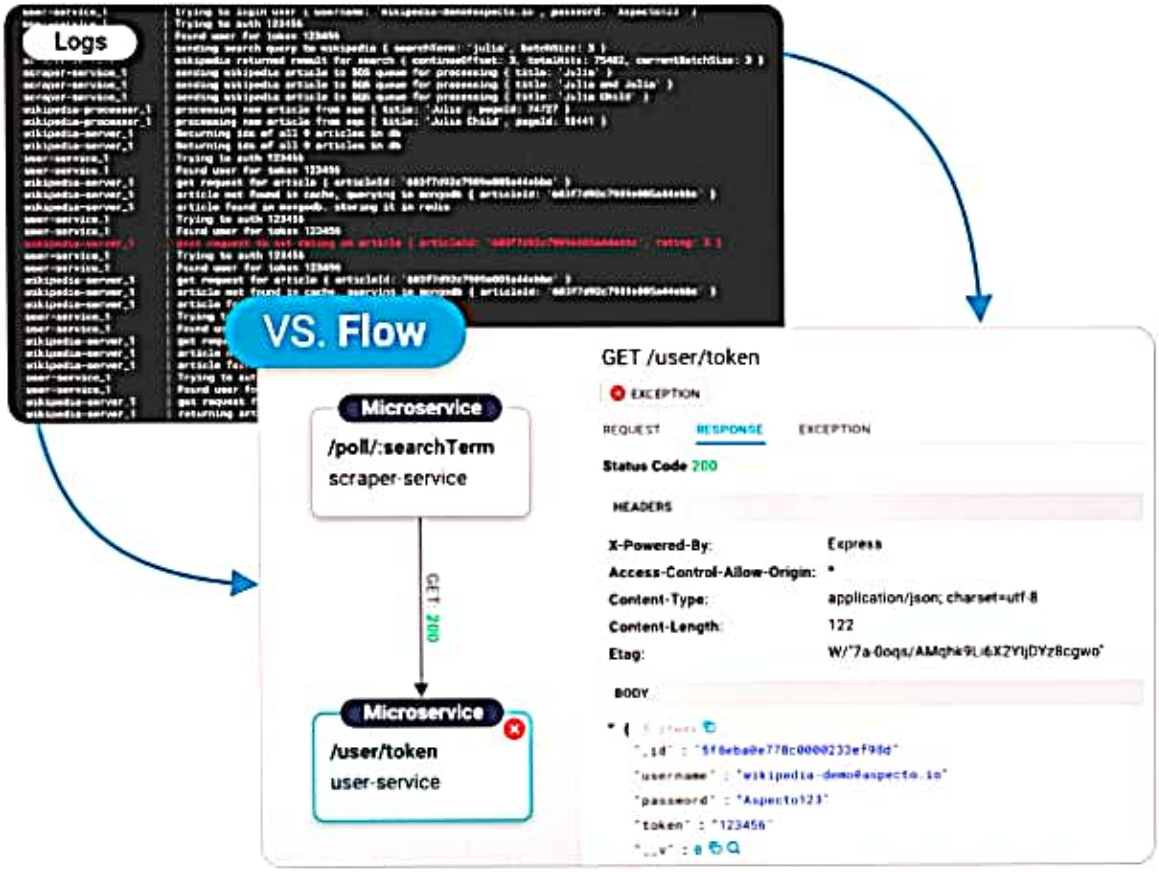
Also, logs are a manual developer tool and can be used for any level of activity – a specific low-level detail, or a high-level action. This

Distributed Tracing Advantages

Where logging is bounded, distributed tracing thrives. Let's see how distributed tracing answers logging limitations when it comes to debugging microservices.

1. Visualization

Traces are visual instrumentation. As opposed to text logs, with traces, developers don't have to imagine the communication flows and make up an image in their minds. Instead, they can see it right before their eyes. This makes



2. Automation

Unlike logs, traces are automatic. Developers don't have to make the manual effort of adding logs to get the complete picture. Instead, they automatically get a visualization of what happened. This also solved the standardization problem. With automated traces, the standardization is hard-coded in.

3. Accelerate Time-to-Market

Distributed tracing provides observability and a clear picture of the services. This improves productivity because it enables developers to spend less time trying to locate errors and debugging them, as the answers are more clearly presented to them. As a result, productivity is increased, and developers can spend more time developing features, (or taking a break), while you accelerate time-to-market.

4. Tracking Requests Across Services

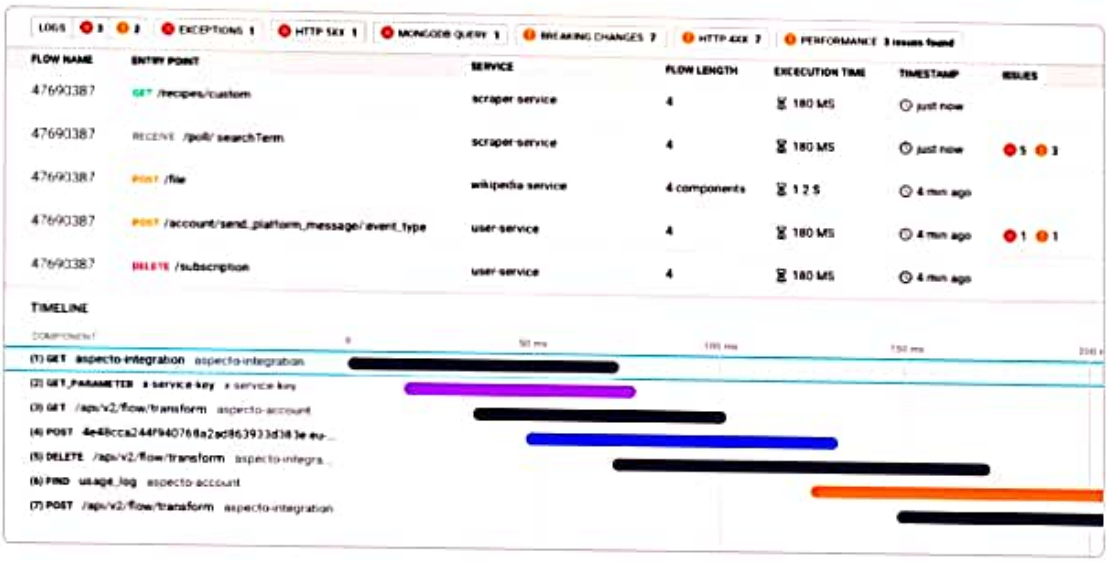
Microservices interactions span multiple services. Distributed tracing enables understanding the system and the relationships between components. This is done

5. Easy to Use and Implement

With the right setup, developers can work with multiple applications and across different programming languages. This is unique for distributed tracing and saves your team a lot of time and headaches, by not restricting you to one language or certain apps.

6. Insightful

Distributed tracing provides the developer with a lot of insightful information. This includes request time, information about components, latency, application health, and more. All this info can be useful when debugging and during root cause analysis, for improving code quality and resolving customer issues quickly.



When Should We Use Distributed Tracing?

Great question! Here are the three main use cases in which distributed tracing can be helpful for you and your team.

1. For a Distributed-Application Architecture

If your department is using a distributed infrastructure, we highly recommend implementing distributed tracing. As you can see, this is the best method for tracking requests across services,