



## Microservices: Avoiding Dumb Pipes

One of the hottest new terms in the world of enterprise computing is the *microservice*. Starting with the [seminal 2014 article by James Lewis and Martin Fowler](#) of [ThoughtWorks](#), microservices have taken on a life of their own – and as with any other overhyped term, they have generated their fair share of confusion as well.

Lewis and Fowler specifically focus on the *microservice architectural style*, which they define as “an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.”

However, there are two limitations to this definition: first, the definition expressly focuses on single, discrete applications, rather than the complex, interconnected web of functionality characteristic of modern architectures. And second, they leave the word “service” woefully undefined, leaving it up to the reader to figure out what they mean by a service – even though there is still rampant confusion on this point as well.



Helping to clear up this confusion is [Janakiram MSV's January 2015 article for ComputerWeekly](#). In this article, Janakiram defines microservices as “fine-grained units of execution.” He continues:

*They are designed to do one thing very well. Each microservice has exactly one well-known entry point. While this may sound like an attribute of a component, the difference is in the way they are packaged.*

*Microservices are not just code modules or libraries – they contain everything from the operating system, platform, framework, runtime and dependencies, packaged as one unit of execution.*

*Each microservice is an independent, autonomous process with no dependency on other microservices. It doesn't even know or acknowledge the existence of other microservices.*

*Microservices communicate with each other through language and platform-agnostic application programming interfaces (APIs). These APIs are typically exposed as Rest endpoints or can be invoked via lightweight messaging protocols such as RabbitMQ. They are loosely coupled with each other avoiding synchronous and blocking-calls whenever possible.*

What I like most about Janakiram's definition of microservices is first, it's concrete, and second, it's not circular. In other words, he defines a microservice in terms of its physical components, and he doesn't use the word "service" in his definition.

### Smart Endpoints, Dumb Pipes

The focus of this article, however, is on how microservices communicate with each other and with everything else. As Janakiram points out, such communication should be lightweight and platform-agnostic, what Lewis and Fowler refer to as *dumb pipes*.

The contrast that the ThoughtWorks fellows are trying to make is between the lightweight protocols Janakiram is referring to and the heavyweight, middleware-centric enterprise service buses (ESBs) that drove many of the SOA implementations in the 2000s.

Lewis and Fowler clearly have a bone to pick with this first-generation approach to SOA. "In particular we have seen so many botched implementations of service orientation, from the tendency to hide complexity away in ESB's, to failed multi-year initiatives that cost millions and deliver no value, to centralised governance models that actively inhibit change," Lewis and Fowler pontificate, "that it is sometimes difficult to see past these problems."


Lewis and Fowler are making an important point, of course. Many SOA initiatives failed – or at least, took far longer and cost far more than they should have – because large middleware vendors hoodwinked enterprise IT shops into believing that buying and deploying their heavyweight ESBs was the best way to implement SOA.

In response, Lewis and Fowler recommend "applications built from microservices aim to be as decoupled and as cohesive as possible," following RESTful protocols instead of centralized orchestration on an ESB.

For those situations where more than simple HTTP-based interactions are necessary, they recommend a "dumb" message bus – "dumb as it acts as a message router only" – simply to provide a reliable asynchronous fabric, but "the smarts still live in the end points," in other words, in the microservices themselves.

### From "Dumb" to "Web Scale"

Unfortunately, Lewis and Fowler's diatribe against first-generation SOA – no matter how justified – has muddied the water over the appropriate role for microservice integration technologies.



MANY SOA INITIATIVES  
FAILED BECAUSE LARGE  
MIDDLEWARE VENDORS  
HOODWINKED ENTERPRISE IT  
SHOPS INTO BELIEVING THAT  
BUYING AND DEPLOYING  
THEIR HEAVYWEIGHT ESBs  
WAS THE BEST WAY TO  
IMPLEMENT SOA.

Clearly, no one aspires to be dumb, so using such a pejorative term doesn't advance the discussion properly.

If we take a step back and compare notes between the ThoughtWorks piece and Janakiram's, however, a more useful set of architectural principles emerge: horizontal scalability. Stateless communications. Event-driven, reactive interactions. Asynchronous interactions that favor eventually consistent data. In other words, *web scale*.

While it's true that we don't want centralized, stateful orchestration of services, we also don't want to limit our microservice interactions to the extent that they are "dumb." Rather, we simply have a new way of thinking about "smart" pipes – microservice integration that is architected following web scale, cloud-centric principles.

Rethinking the smart endpoints/dumb pipes dichotomy as smart endpoints/web scale pipes sends a better message about the proper approach to integration in a microservices architecture. Furthermore, this way of thinking about microservice integration opens up an appropriate consideration of a broad class of web scale integration technologies, including products from companies like [SnapLogic](#) and [Fiorano Software](#) as well as a rapidly expanding category of integration-platform-as-a-service (iPaaS) providers.

In fact, both SnapLogic and Fiorano have implemented their integration technology as microservices, and furthermore, SnapLogic customers can leverage their Ultra Pipelines to build microservices that serve as "web scale pipes."

This approach to integration bears only a passing resemblance to the traditional heavyweight ESBs of old, and are far more deserving of the "smart" moniker than those centralized middleware behemoths.

In the final analysis, microservices are one aspect of modern distributed computing best practice. Today's customers require massive scale and blisteringly fast performance from the technology that enterprises (and everyone else) delivers. Microservices are a part of this story, but so are the web scale principles of microservice integration.

*SnapLogic and Fiorano Software are [Intellyx](#) clients. At the time of writing, no other organizations mentioned in this article are Intellyx clients. Intellyx retains full editorial control over the content of this article. Image credit: [Seeweb](#) and [Ben Jones](#).*



WE HAVE A NEW WAY OF THINKING ABOUT "SMART" PIPES – MICROSERVICE INTEGRATION THAT IS ARCHITECTED FOLLOWING WEB SCALE, CLOUD-CENTRIC PRINCIPLES.