Editor: **Robert Blumen**
Symphony Commerce
robert@robertblumen.com

Microservices

Johannes Thönes

ONE GOAL of the Software Engineering Radio podcast is to be a source of information about the latest architectural trends. Trends emerge from practice and take a while to show up in written form. The first book on microservices isn't due until spring 2015. For the professional software engineer, conferences, online talks, and podcasts are often the best sources for the newest information.

In this month's podcast (episode 213), Johannes Thönes talks with James Lewis about microservices. This podcast is the third one in the fall schedule to address this topic. In episode 210, Stefan Tilkov discusses architecture and microservices; in episode 216, Netflix architect Adrian Cockroft discusses the cloud-based platform. The upcoming episode 217 on the Docker container covers a popular piece in the deployment of these systems (see the sidebar).

The following excerpt contains only a fraction of the show. Space didn't permit us to include discussions covering the relationship between microservices and Conway's law, CQRS (Command Query Responsibility Segregation), REST (representational state transfer), operational complexity and the impact on development operations, "isn't this just SOA?," agile development, testing, and monitoring.

I hope you'll download the entire show and listen. —Robert Blumen

What's a microservice?

A microservice, in my mind, is a small application that can be deployed independently, scaled independently, and tested independently and that has a single responsibility. It is a single responsibility in the original sense that it's got a single reason to change and/or a single reason to be replaced. But the other axis is a single responsibility in the sense that it does only one thing and one thing alone and can be easily understood.

What would such a single thing be?

An example of a single thing might be a single responsibility in terms of a functional requirement, or it might be in terms of a nonfunctional requirement or, as we've started talking about them, cross-functional requirements.

An example might be a queue processor—something that's reading a message from a queue, performing a small piece of business logic, and then passing it on. Or it might be something that's cross-functional, or nonfunctional, or it might be something that has the responsibility for serving a particular resource or resource representation.

Like a user.

Like a user or, say, an article, or it might be a risk in insurance or something like this, but something that's very focused and very small and that performs a single task on its own.

I have the impression that microservices have become quite popular. Why do you think that is?

Continued on p. 113

0740-7459/15/$31.00 © 2015 IEEE

Continued from p. 116

We've got this big application. It's been growing for two-and-a-half years or five years or 10 years, but we can't maintain it anymore. It's just too difficult to actually make any functional changes to it. We need to deploy this application into the cloud. We need software as a service, but at the moment that is impossible.

As a result of that, the ideas evolved of starting to splitting applications into smaller cooperating components that are running out of process and talking to one another, which can be maintained separately, scaled separately, or thrown away if needed.

A number of different communities have grown over time that have demonstrated that this approach to building software is viable for production. When you look at companies on the scale of Netflix, then it's almost a necessity as they grow income. Adrian Cockroft has said that they work this way because they want to build systems and make changes as fast as possible.

To answer your question, 'Why is it so popular now?' a lot of organizations have built up technical debt over the last number of years. They have realized that to scale more, to be more effective at delivering software into production, and to take advantage of things like continuous delivery, they need an approach that allows them to do scale along different axes independently of things like continuous delivery.

I think it's about the right time for an idea like microservices to take off because a lot of companies are facing the same problems.

You said something interesting about how people with a large monolithic application are splitting it into microservices. Is there a typical form of introducing microservices?

That's a great question and one I've actually been struggling with. It goes right to the heart of the question: do you start with microservices, or do you refract to them later?

Empirically, most of the organizations have actually started with something big and have split that big thing up. That's the case for most organizations that are building a microservices-style implementation. For example, Netflix. The canonical example is Amazon. Amazon started with a big database and then moved to a service-oriented architecture.

Let's talk a little bit more about how you technically build a microservice. When I build a microservice for user authentication, what languages would I use? What standards do I build on, and what do I need to do to make it happen?

One of the guiding principles behind this is that you get the freedom to choose a lot of your tooling on a case-by-case basis. Rather than it being a particular language or particular back-end data store for your entire product stack, you get the flexibility to make informed decisions based on the right tooling for the situation at hand.

There are no right or wrong choices. If you're talking about a user service, it is easily implemented in C#, Java, or any other modern programing language. Pretty much any programming language is going to be suitable.

The key thing is to make the stack lightweight. Rather than using the traditional heavy stacks and deploying them into big application containers (like JBoss and Tomcat), you can use lightweight alternatives, such as embedded Jetty, embedded Tomcat, SimpleWeb, or WebIt.

.NET-land is an interesting place at the moment because traditionally it has deployed into IIS. We've deployed all of our applications into this managed environment. But even in the .NET world, there's been a movement to bring in some of their learnings from the Unix and Java communities around using embedded services. For example, we're seeing more projects using a non-CFX alternative to some of their web APIs or MVC frameworks, and then using things like Owen. It's about recognizing the centralization of the model that requires you to put all of your logic in one place. That place is the ESB, which provides all of the routing and data transformation required to get your applications talking to each other.

Is the "smart endpoint and dumb network" a reference to the Unix model?

It could be read like that. The reason we chose that name was more around the enterprise service bus (ESB) model. Inside Thoughtworks, for as long as I can remember, there's been a tendency to distrust heavy iron when it comes to integration.

Big ESB products make a lot of promises about solving all your problems. I have seen a lot of implementations of "service-oriented architecture" with everything hanging off a big central ESB. I have never seen one of those succeed. It's about recognizing the centralization of the model that requires you to put all of your logic in one place. That place is the ESB, which provides all of the routing and data transformation required to get your applications talking to each other.

SOFTWARE ENGINEERING RADIO

Visit www.se-radio.net to listen to these and other insightful hour-long podcasts.

RECENT EPISODES

- 213—James Lewis sits down with Johannes Thönes to explain the microservices architectural pattern, the forces driving it, the costs and benefits, and the organizational impact.
- 214—Grant Ingersoll, founder and CTO of LucidWorks, talks with Tobias Kaatz about his book *Taming Text: How to Find, Organize, and Manipulate It*.
- 215—The three living authors of *Design Patterns: Elements of Reusable Object-Oriented Software*, with Johannes Thönes, offer a 20-year retrospective on the writing of the book and its subsequent impact on design in software engineering.

UPCOMING EPISODES

- 216—Former Netflix architect Adrian Cockroft talks with Stefan Tilkov about the modern cloud-based platform, Netflix's move to the cloud, and microservices.
- 217—James Turnbull talks with Charles Anderson about the open source container Docker, containers versus virtual machines, and the implications for system administrators.

Relying on one of these things to solve all your problems is, in my mind, not the right approach. There's a great talk by Jim Weber and Martin Fowler called "Does My Bus Look Big in This?," which they did as a keynote at QCon some years ago. Jim talked about the idea of the spaghetti box: the ESB as the panacea for all your ills. His line on that is it makes your diagrams look nice. You look at your enterprise architecture [diagrams], and they've got all these crossing ugly lines It's really tempting to put the ESB box in the middle because suddenly all your lines are straight. That's a great thing if you're an architect.

But of course all the lines are still there. They're just in the middle of a spaghetti box. It still looks like a spaghetti box.

But when all the routing isn't done by the ESB, who does the routing? Do I need to do the routing?

You certainly need to understand more about how your applications communicate with one another. If you're building more services you end up with more integration problems. In the past, you might have been unlucky to talk to three external systems. Now you have to be cognizant of integration problems when you talk to your own systems. And there are ways to do that. Event-driven applications (with either publish-and-subscribe messaging, or HTTP and resource representation) allow you to decouple compared to using point-to-point RPC the whole time.

Isn't that a bit like moving the complexity from the monolith into the networking layer?

The short answer to that is yes. Actually, when I originally talked to people about this, one of the great comments I got back, from Martin Fowler, was that we're shifting the accidental complexity (in the sense that Fred Brooks used the term) from inside our application in glue code in our components and modules within our application out into infrastructure.

This is one of the reasons that now is a good time for this because we have many more ways to manage that complexity: programmable infrastructure, infrastructure automation, the movement to the cloud, the cloud being ubiquitous. Those sorts of problems, the problems of understanding how many applications we have, how they're talking to one another—we have better tools to address those things now.

You mentioned domain-driven design in the beginning. Is microservices domain-driven design with a "service" label?

Microservices is the coming together of a bunch of better practices from a number of different communities. It is a combination of great stuff from the domain-driven-design community around strategic design, bounded context, subdomains, how to separate out your domains, and how to partition a very big problem domain into smaller domains so that you can manage them. It's also taking a bunch of the better practices from operational automation and programmable infrastructure, de-

velopment operations communities, cloud communities, and the integration communities.

You've been working hard to make people aware they can solve integration problems using just the tooling available for free that drives the Web, without having to invest in big iron.

From the domain-driven-design community, the way you do "architecture" has to be driven from the business in the business context. You have to understand what the business problems are, what the business landscape looks like, and what the business processes are, and then drive a software product underneath that. For me, that's the heart of domain-driven design.

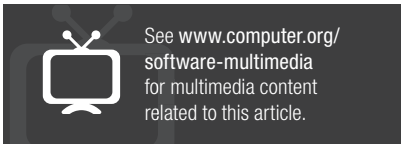
One of my colleagues uses the great phrase "business and architecture isomorphism." This is the idea that your business and the design of your systems should be very similar. When you look at your business, you should see your IT systems and look at your architecture and see your business. If you're a technologist or business person, there should be recognition both ways that this is going on.

How big are these services?

That's something we've been talking about internally for quite a while. I've seen them ranging from a couple of hundred lines of code up to a couple of thousand lines of code. The guidance I've been giving people is it does one thing and one thing only. It's difficult to imagine a million lines of code doing one thing and one thing only. The guidance is you should be able to understand them. They should have a single reason to change, and they probably shouldn't be more than a couple thousand lines of code.

When you get to that point, the number becomes important. It's probably more important to think about how many of them you're capable of supporting operationally than it is to think about how small they actually are because it's better to have slightly bigger ones and fewer of them if you don't have fully automated deployment into production.

JOHANNES THÖNES is a developer and consultant for ThoughtWorks. Contact him at johannes.thoenes@gmail.com.



IEEE Software

NEXT ISSUE:

March/April 2015

Release Engineering

IEEE Software (ISSN 0740-7459) is published bimonthly by the IEEE Computer Society. IEEE headquarters: Three Park Ave., 17th Floor, New York, NY 10016-5997. IEEE Computer Society Publications Office: 10662 Los Vaqueros Cir., Los Alamitos, CA 90720; +1 714 821 8380; fax +1 714 821 4010. IEEE Computer Society headquarters: 2001 L St., Ste. 700, Washington, DC 20036. Subscribe to *IEEE Software* by visiting www.computer.org/software.

Postmaster: Send undelivered copies and address changes to *IEEE Software*, Membership Processing Dept., IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854-4141. Periodicals Postage Paid at New York, NY, and at additional mailing offices. Canadian GST #125634188. Canada Post Publications Mail Agreement Number 40013885. Return undeliverable Canadian addresses to PO Box 122, Niagara Falls, ON L2E 6S8, Canada. Printed in the USA.

Reuse Rights and Reprint Permissions: Educational or personal use of this material is permitted without fee, provided such use: 1) is not made for profit; 2) includes this notice and a full citation to the original work on the first page of the copy; and 3) does not imply IEEE endorsement of any third-party products or services. Authors and their companies are permitted to post the accepted version of IEEE-copyrighted material on their own webservers without permission, provided that the IEEE copyright notice and a full citation to the original work appear on the first screen of the posted copy. An accepted manuscript is a version which has been revised by the author to incorporate review suggestions, but not the published version with copyediting, proofreading, and formatting added by IEEE. For more information, please go to: http://www.ieee.org/publications_standards/publications/rights/paperversionpolicy.html. Permission to reprint/republish this material for commercial, advertising, or promotional purposes or for creating new collective works for resale or redistribution must be obtained from IEEE by writing to the IEEE Intellectual Property Rights Office, 445 Hoes Lane, Piscataway, NJ 08854-4141 or pubs-permissions@ieee.org. Copyright © 2015 IEEE. All rights reserved.

Abstracting and Library Use: Abstracting is permitted with credit to the source. Libraries are permitted to photocopy for private use of patrons, provided the per-copy fee indicated in the code at the bottom of the first page is paid through the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923.