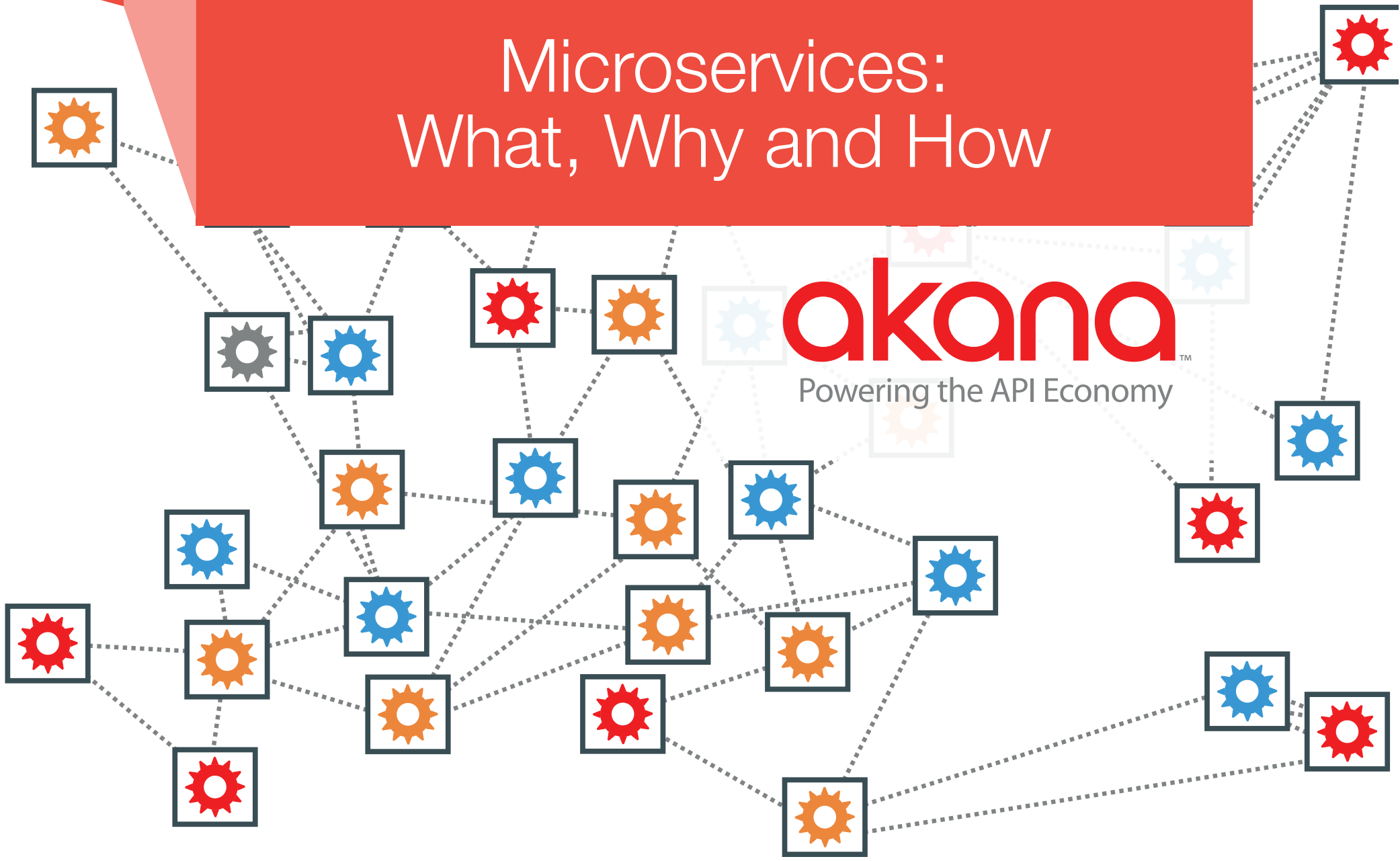
Microservices: What, Why and How



Abstract: Enterprise use of microservices is on the rise. By breaking large applications down into small, independently functioning services, microservices enable advances in agility and flexibility for developers, but are not exempt from challenges. We will explore the definitive, “What, Why and How?” overview of microservices. This paper will discuss how microservices work and will cite examples and suggested practices to help ensure the successful adoption of a microservices architecture. We will discuss the role of APIs in microservices architectures and look at how helpful an API management platform can be.

Introduction

Microservices is/are new and exciting enough to bring quite a bit of FUD into enterprises that are considering how to best use them. The challenge in explaining microservices is immediately apparent in the use of “is/are” in the last sentence. Microservices define both an abstract concept and a set of concrete objects. At a high level, the microservices pattern is a new way to build applications consisting of sets of independent microservices implementations. We will try to focus on describing the concept as the microservices pattern, and the concrete service implementations as microservices.

The microservices pattern represents a new approach to creating applications, combining the concepts of service-oriented architecture (SOA), containerization and DevOps. The pattern combines the principles of SOA with bounded contexts to create small, autonomous microservices, working together to efficiently scale overloaded systems. Microservices are most commonly built by small agile teams, using continuous integration and delivery, often leveraging containerization for rapid deployment. As a result, the microservices pattern enables scalability, speed, efficiency and agility.

Let’s get a clear sense of how microservices work best and/or the ways to implement them. When we look at the leading trends in enterprise architecture, the microservices pattern presents a bucket of opportunities with some concomitant challenges. Implementing a successful microservices architecture can mean rethinking a number of enterprise architectural concepts, thought processes and workflows. We will highlight the main concerns by offering you a rundown on the what, why and how of microservices.

Microservices: What?

Microservices are gaining traction, making headlines and stimulating new thinking about how to organize application architecture. Now, we will dive deeper into what exactly is the microservices pattern, and what exactly are microservices. The microservices pattern breaks a large application down into small, independent services that are not language-specific. It offers IT organizations tools for agility and cost reduction as long as the organization is sufficiently mature to be able to follow the core guiding principles.

The term "microservices" refers to a style of software architecture where complex applications can be composed of small independent services. The independent "services" (or processes) exchange data and procedural requests. Microservices implementations use application programming interfaces (APIs) or events that are most likely standards-based and language-agnostic. Regardless of the language used within your organization, you can implement a microservices architecture.

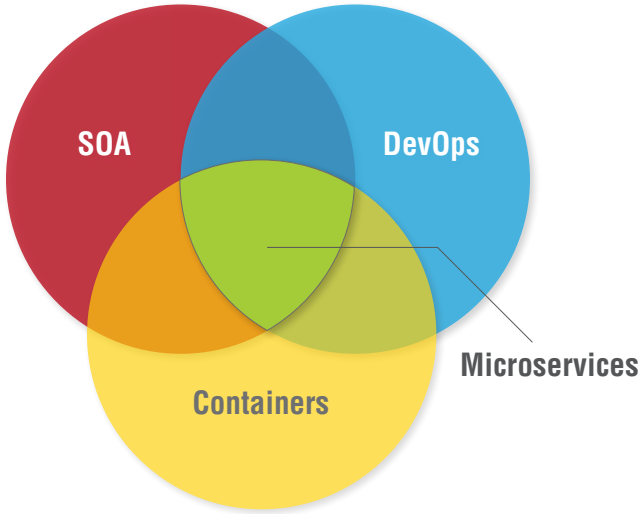


Figure 1 - Microservices, at the intersection of SOA, DevOps and containers.

Microservices use lightweight communications protocols and are highly focused on providing one capability. The "micro" refers to narrowness of focus, not size. They are usually built by small "Two Pizza" teams to ensure adequate focus on the capability.

Microservices vs. SOA

Microservices exist at the intersection of SOA, the mashup of agile development and, IT operations known as DevOps and containerization. Figure 1 shows this concept as a Venn diagram. However, microservices go beyond the actual architecture. When you combine fast-moving software development that leverages the principles of SOA and containers, those are microservices.

Microservices differ from SOA, though the two architectural styles share a common history and a number of traits. SOA was designed to be technology-agnostic, but over time SOA has become very standards- and vendor-driven. Vendors drove SOA down the path of SOAP and enterprise service buses (ESBs). While microservices reaffirm many SOA principles, they are almost a reaction against that traditional vendor-driven SOA. The heavy SOA architecture is the opposite of today's lightweight microservices. Microservices are decidedly anti-ESB.

Microservices vs. APIs

Some people in the API world might argue that microservices are just APIs, but they are focused on solving very different problems. APIs are concerned with how to manage digital transformation and support a large eco-system of developers and partners by making it easy for them to consume data or

applications. They provide connectivity and integration for easier and better consumption of services, often by web and mobile apps.

The difference between an API and a microservice is not based on technology. APIs and microservices are complementary to each other. Microservices provide agility and scale to applications that are increasingly surfaced via APIs. Essentially, the API is the result of building an application and an application could be built using microservices.

Fundamentally:

• SOA services are provider-centric, focusing on the re-use of business capabilities between applications.

• APIs are consumer-centric, focusing on enabling the consumption of business capabilities by external, partner, and internal developers.

• Microservices are application-centric, delivering specific capabilities for one application in a scalable, distributable manner.

Microservices: Why?

Microservices are catching on in the enterprise space because they enable IT organizations to be more agile and build more scalable applications.

Microservices translate into faster development and change cycles. As Figure 2 suggests, breaking down a large application into smaller services makes the development workflow move faster. Tasks can move more quickly through the microservice's smaller funnel.

As a small development effort, a microservice can be built faster than the typical large-scale software development project involving a monolithic application. Microservices are more fluid and move independently at their own pace. There are no waterfalls in the development process - the opposite of a monolithic application. Due to their independent nature, microservices can be developed using any programming language or constructs. They have better variability. With containerization, microservices can be quickly deployed on-demand as part of a continuous delivery process.

Microservices failures are also less catastrophic than breakdowns in bigger systems. A failure in one part of a stand-alone application can take down everything else. A system-wide diagnosis may lead to releasing another version of the application to fix that particular error. In a microservices architecture, the function provided by the microservice may not be essential and it is typically quicker and easier to fix problems. The agility of the design helps you to identify, isolate, and compensate for errors. If you have a microservices architecture, you can scale microservices to overcome a performance constraint. Another attribute is being able to quickly build a new version of the faulty microservices without dealing with the cumbersome operational cascade that usually exists within a monolithic application.

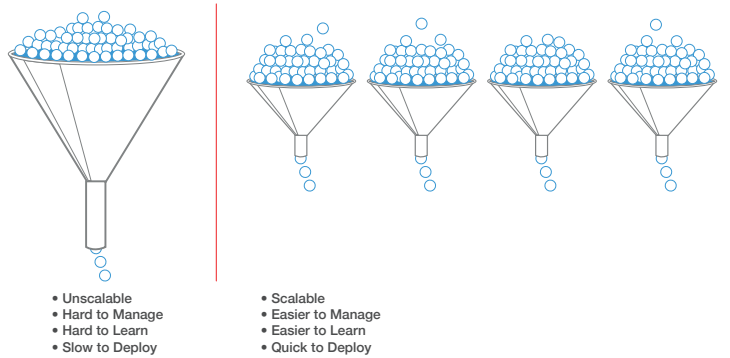


Figure 2 - Why microservices are popular: Compared to traditional applications, they are faster, more scalable and easier to manage.

akana™
Powering the API Economy

Microservices give you better reliability, range and flexibility. For example, you can scale up any one part of your application. If a login is suffering because the system is getting of a lot of new users, you can better scale that particular part of the application with a microservices architecture. Normally, you would have to stand up new app servers, which are running the big monolithic application, just because that one particular page or one particular part of a site is being used more than others.

Now, with microservices, if you have a spike in the number of logins, the microservices architecture provides a menu of more elastic solutions. In this case, containerization allows IT to quickly deploy as many new instances of the login microservices as needed and then load balance across them all.

Microservices: How?

Some architects believe that migrating old applications to microservices means breaking apart the existing applications into component parts and putting them back together in a new way, which would streamline the architecture. However, architects would do well to stop thinking of microservices as a way to build entire applications. The idea is not about taking monoliths and completely re-implementing them as a set of microservices. Rather, architects should look at isolating services that make sense to develop as microservices, and incrementally branch out from there.

The biggest impact of adopting a microservices pattern is organizational. When adopting microservices, the organization has to change structure and approach or they will run afoul of Conway's Law. Microservices require training and new skillsets. They affect design, development and testing. Thoughtful staffing of the development and testing teams and scoping out of parameters will help inform design and implementation of microservices, which work best when ops are mature, including properly functioning DevOps and competent adoption of containerization.

Microservices affects many familiar processes. The following table highlights key needs in the areas of development, data and testing.

Development	Data	Testing
• Development teams must know about other available interfaces, tooling, etc. • Development teams must know what libraries, strategies, patterns will provide for resiliency, health, bulkheads and alternatives. • Development teams will get to know a lot more about the SDLC before them and after full lifecycle responsibility.	• Distributed data management requires eventual consistency accommodation. • Each microservice should have its own dedicated datastore.	• Testing at the unit and integration levels still exist. • Testing tools will have to account for new paradigms for microservices - registry, health dashboard - and build in these scenarios. • Testing needs to occur inside the service - Delivery as testing. Source: martinfowler.com/articles/microservice-testing/

Martin Fowler and James Lewis, industry thought leaders on enterprise software, provide an outstanding rundown on new skills and practices that IT organizations must master in order to succeed with microservices. In their view, the following concepts offer a solid idea of what and how to choose and implement the right kind of microservice. As Fowler and Lewis note, "You can take a bad application and build a bunch of bad microservices

out of it." If you consider the following principles, it is possible to avoid a bad outcome:

• **Componentization via services** – Knowing how to create an interface that leverages the best technology for the job, e.g. REST.

• **Organize around business capabilities** – Microservices need to be organized around distinct business capabilities. The microservices developer creates products not projects. Given that the service will have its own complete lifecycle, it is really a matter of product management not software development.

• **Deliver smart end points and dumb pipes** - You will have to orchestrate or choreograph the different end points. Load balancing is built into the framework and the end points. The registry contains end points. The content has decentralized governance. The registry is built into the service itself. The data management will become decentralized, moving to a more functional domain-driven design context versus a traditional data model.

• **Automate the infrastructure and design for failure** - The architecture needs to be continually adjusting for and compensating for failure. Ideally, you build failure into the testing of the microservices infrastructure, so that while you're doing testing, services fail. Containers disappear or shut down. Things start running more slowly. The infrastructure has to scale automatically.

• **Design** - Traditional design and the oversights that you had are no longer relevant. Everyone will use the best tools to create the microservice. You will need to control the microservice definition and the interface, but you cannot control all the moving parts of the process.

• **Designing microservices the right way** - You need to facilitate design time review of available services and find a way to force people to request the service. This is why you need to have a registry that enables people to look for services. If you don't find them, then you have to request a new service or a new interface. At this point, somebody will say, "Yes. That looks like a good interface. Let's build that as a microservice." Of course, there will probably be some approvals to go through.

• **Adopt conventions** - You will need to adopt certain conventions around the interface that you are exposing. For example, you will need to have an end point. These conventions must be established around the design of your interfaces. Then the conventions have to integrate within your framework and infrastructure to enable making those lasting choices and decisions during runtime.

• **Design for robustness** - Microservices affect design and the design pattern. You might have a proxy end point, for instance, that enables you to do the routing and resiliency on the back end, with awareness of distributed data design and the main driven design around the data. Ultimately, data has to be separated into a microservice.

Microservices and API Management

The result of an application built using microservices is often an API. An API Management platform can ensure the reliability, security and performance of that API.

The Akana API Management platform provides the functionality to deliver applications as a series of microservices. The platform streamlines management, deployment, development and operation of APIs, enhancing security and regulatory compliance through authentication, authorization and audit capabilities.

It can also do a lot more. In many cases the API Management platform, specifically the API Gateway, can take the role of the application controller by distributing requests across a series of microservices.

akana™
Powering the API Economy

The platform lets users take advantage of lightweight container platforms for microservices deployment while still providing the core capabilities required by an enterprise services environment. This includes central definition and management of security, routing, orchestration, mediation, auditing, threat protection and other operational governance policies across multiple instances. This makes it possible to have microservices which comply with security policies – a common goal of many architects struggling to adopt microservices in security and compliance-intensive enterprises.

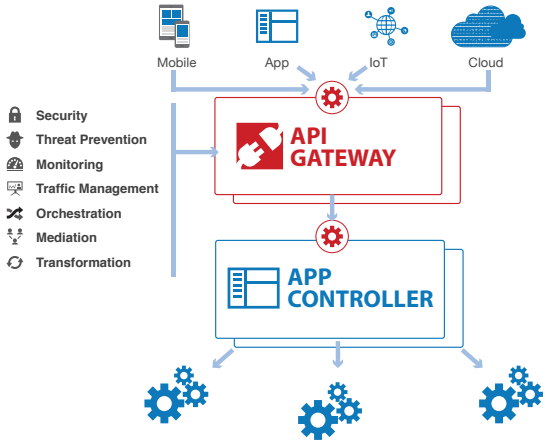


Figure 3 – Akana API Gateway working with an App controller.

Akana gives administrators the ability to scale microservices independently based on real-time usage needs. Akana's tools make it possible for IT to provide high-availability with load balancing over just enough redundant microservices instances to support load, should one instance fail. Users can create managed container instances for easy deployment of secure microservices, automating the registration and discovery of microservices instances with container configuration tasks.

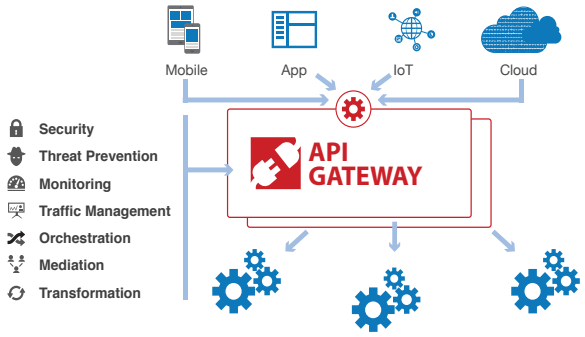


Figure 4 - Akana API Gateway.

Conclusion

Microservices are gaining in popularity because they give architects a way to build flexible, scalable web-scale applications. They do this by breaking a large application down into small independent services. This process enables greater agility because changes and development can be executed faster than those in a monolithic application, though microservices do present a host of challenges. They are best approached incrementally, eating the elephant one bite at a time. Transforming entire applications into microservices is not an optimal approach. Rather, given the technology's impact on developing, testing and deployment, it is best to be selective about which application components should be first made into microservices.

The API is the application. Applications built using microservices architectures will often result in an API. For this reason, the choice of an API management platform is critical to making a microservices architecture function as envisioned. An effective API management platform can act as the application controller while simultaneously monitoring performance and providing security and governance.

akana
Powering the API Economy

About Akana

Akana is a leading provider of API Security and Management products that help businesses plan, build, run and share APIs, through comprehensive cloud and on-premise solutions that encompass API lifecycle, security, management and developer engagement. The world's largest companies including Bank of America, Pfizer, and Verizon use Akana solutions to transform their business.

For more information, please visit www.**akana**.com

akana™

Powering the API Economy

12100 Wilshire Blvd, Suite 1800
Los Angeles, CA 90025

(866) 762-9876 | www.akana.com | info@akana.com