APPDYNAMICS

How to build (and scale) with microservices

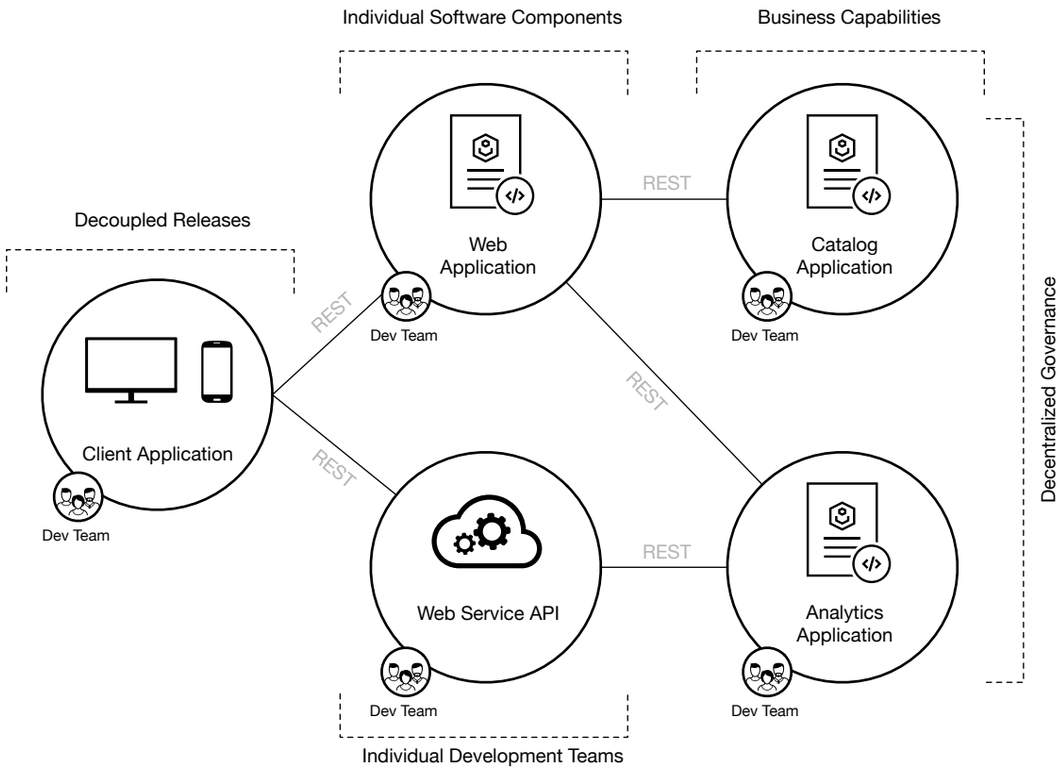
Table of contents

Chapter 1: Introduction to microservices

Microservices are a type of software architecture where large applications are made up of small, self-contained units working together through APIs that are not dependent on a specific language. Each service has a limited scope, concentrates on a particular task and is highly independent. This setup allows IT managers and developers to build systems in a modular way. In his book, "Building Microservices," Sam Newman said microservices are small, focused components built to do a single thing very well.

Martin Fowler's "Microservices - a Definition of This New Architectural Term" is one of the seminal publications on microservices. He describes some of the key characteristics of microservices as:

- **Componentization:** Microservices are independent units that are easily replaced or upgraded. The units use services to communicate with things like remote procedure or web service requests.
- **Business capabilities:** Legacy application development often splits teams into areas like the "server-side team" and the "database team." Microservices development is built around business capability, with responsibility for a complete stack of functions such as UX and project management.
- **Products rather than projects:** Instead of focusing on a software project that is delivered following completion, microservices treat applications as products of which they take ownership. They establish an ongoing dialogue with a goal of continually matching the app to the business function.
- **Dumb pipes, smart endpoints:** Microservice applications contain their own logic. Resources that are often used are cached easily.
- **Decentralized governance:** Tools are built and shared to handle similar problems on other teams.



History of microservices

The phrase "Micro-Web-Services" was first used at a cloud computing conference by Dr. Peter Rodgers in 2005, while the term "microservices" debuted at a conference of software architects in the spring of 2011. More recently, they have gained popularity because they can handle many of the changes in modern computing, such as:

- Mobile devices
- Web apps
- Containerization of operating systems
- Cheap RAM
- Server utilization
- Multi-core servers
- 10 Gigabit Ethernet

The concept of microservices is not new. Google, Facebook, and Amazon have employed this approach at some level for more than ten years. A simple Google search, for example, calls on more than 70 microservices before you get the results page.

Also, other architectures have been developed that address some of the same issues microservices handle. One is called Service-Oriented Architecture (SOA), which provides services to components over a network, with every service able to exchange data with any other service in the system. One of its drawbacks is the inability to handle asynchronous communication.

How microservices differ from service-oriented architecture

Service-oriented architecture (SOA) is a software design where components deliver services through a network protocol. This approach gained steam between 2005 and 2007 but has since lost momentum to microservices. As microservices began to move to the forefront a few years ago, a few engineers called it "fine-grained SOA." Still others said microservices do what SOA should have done in the first place.

SOA is a different way of thinking than microservices. SOA supports Web Services Definition Language (WSDL), which defines service end points rigidly and is strongly typed while microservices have dumb connections and smart end points. SOA is stateless; microservices are stateful and use object-oriented programming (OOP) structures that keep data and logic together.

Some of the difficulties with SOA include:

- SOA is heavyweight, complex and has multiple processes that can reduce speed.
- While SOA initially helped prevent vendor lock-in, it eventually wasn't able to move with the trend toward democratization of IT.
- Just as CORBA fell out of favor when early Internet innovations provided a better option to implement applications for the Web, SOA lost popularity when microservices offered a better way to incorporate web services.

Problems microservices solve

Larger organizations run into problems when monolithic architectures cannot be scaled, upgraded or maintained easily as they grow over time. Microservices architecture is an answer to that problem. It is a software architecture where complex tasks are broken down into small processes that operate independently and communicate through language-agnostic APIs.

Monolithic applications are made up of a user interface on the client, an application on the server, and a database. The application processes HTTP requests, gets information from the database, and sends it to the browser. Microservices handle HTTP request/response with APIs and messaging. They respond with JSON/XML or HTML sent to the presentation components. Microservices proponents rebel against enforced standards of architecture groups in large organizations but enthusiastically engage with open formats like HTTP, ATOM and others.

As applications get bigger, intricate dependencies and connections grow. Whether you are talking about monolithic architecture or smaller units, microservices let you split things up into components. This allows horizontal scaling, which makes it much easier to manage and maintain separate components.

The relationship of microservices to DevOps

Incorporating new technology is just part of the challenge. Perhaps a greater obstacle is developing a new culture that encourages risk-taking and taking responsibility for an entire project "from cradle to crypt." Developers used to legacy systems may experience culture shock when they are given more autonomy than ever before. Communicating clear expectations for accountability and performance of each team member is vital.

DevOps is critical in determining where and when microservices should be utilized. It is an important decision because trying to combine microservices with bloated, monolithic legacy systems may not always work. Changes cannot be made fast enough. With microservices, services are continually being developed and refined on-the-fly. DevOps must ensure updated components are put into production, working closely with internal stakeholders and suppliers to incorporate updates.

The move toward simpler applications

As DreamWorks' Doug Sherman said on a panel at the Appsphere 15 Conference, the film-production company tried an SOA approach several years ago but ultimately found it counterproductive. Sherman's view is that IT is moving toward simpler applications. At times, SOA seemed more complicated than it should be; microservices were seen as an easier solution than SOA, much like JSON was considered to be simpler than XML and people viewed REST as simpler than SOAP. We are moving toward systems that are easier to build, deploy and understand. While SOA was initially designed with that in mind, it ended up being more complex than needed.

Another panelist, Allan Naim, product manager at Google, agreed. He explained that SOA is geared for enterprise systems because you need a service registry, a service repository and other components that are expensive to purchase and maintain. They are also closed off from each other. Microservices handle problems that SOA attempted to solve more than a decade ago, yet they are much more open.

How microservices differ among different platforms

Microservices is a conceptual approach, and as such it is handled differently in each language. This is a strength of the architecture because developers can use the language they are most familiar with. Older languages can use microservices by using a structure unique to that platform. Here are some of the characteristics of microservices on different platforms:

Java

- Avoids using Web Archive or Enterprise Archive files
- Components are not auto-deployed. Instead, Docker containers or Amazon Machine Images are auto-deployed.
- Uses fat jars that can be run as a process

PHP

REST-style PHP microservices have been deployed for several years now because they are:

- Highly scalable at enterprise level
- Easy to test rapidly

Python

Easy to create a Python service that acts as a front-end web service for microservices in other languages such as ASP or PHP

Lots of good frameworks to choose from, including Flask and Django

Important to get the API right for fast prototyping

Can use Pypy, Cython, C++ or Golang if more speed or efficiency is required

Node.js

Node.js is a natural for microservices because it was made for modern web applications. Its benefits include:

- Takes advantage of JavaScript and Google's high-performance, open-source V8 engine
- Machine code is optimized dynamically during runtime
- HTTP server processes are lightweight
- Nonblocking, event-driven I/O
- High-quality package management
- Easy for developers to create packages
- Highly scalable with asynchronous I/O end-to-end

.NET

In the early 2000s, .NET was one of the first platforms to create applications as services using Simple Object Access Protocol (SOAP), a similar goal of modern microservices. Today, one of the strengths of .NET is a heavy presence in enterprise installations. Here are two examples of using .NET microservices:

- Building a .NET self-hosted web service using Open Web Interface for .NET (OWIN). You can then use it to incorporate microservices.
- Setting up a fictional Human Resources firm

Responding to a changing market

The shift to microservices is clear. The confluence of mobile computing, inexpensive hardware, cloud computing and low-cost storage is driving the rush to this exciting new approach. In fact, organizations do not have any choice. Matt Miller's article in The Wall Street Journal sounded the alarm; "Innovate or Die: The Rise of Microservices" explains that software has become the major differentiator among businesses in every industry across the board. The monolithic programs common to many companies cannot change fast enough to adapt to the new realities and demands of a competitive marketplace.

Service-oriented architecture attempted to address some of these challenges but eventually failed to achieve liftoff. Microservices arrived on the scene just as these influences were coming to a head; they are agile, resilient and efficient, qualities many legacy systems lack. Companies like Netflix, Paypal, Airbnb and Goldman Sachs have heeded the alarm and are moving forward with microservices at a rapid pace.

Chapter 2: Microservices as a business initiative

For microservices to work in an organization, there must be a business initiative attached to it. Questions arise among IT professionals on whether microservices are suited only for giant Web applications like Google and Facebook. However, scale is only one of the business benefits of microservices.

In today's computing environment, innovation and speed are critical. The movement toward microservices is generated by the need to create new software that can enhance and improve a monolithic system but is separate from it. This decoupling from the legacy system provides the freedom to experiment with new approaches and rapidly iterate changes and modifications.

Traditional systems cannot move at that speed, and that may leave companies disadvantaged. At the Appsphere 15 conference, Boris Scholl from Microsoft shared a situation they once had with a monolithic system. It had become so complex that when they added new code, the system would stop working, and it took two days for engineers to figure out why. It is too slow.

Companies are trying to decide where microservices fit in with their traditional systems. Developers used to worry simply about coding, but now with the modular approach to technology, they need to widen their view of all the technologies involved and how they work together. They now share responsibility and accountability for the project as a whole -- the micro view of their direct assignment, say coding the UX; and the macro view of the final product, a home banking app for example.

Code must be monitored the minute it is deployed. The feedback loop is instantaneous. DevOps may be monitoring 50 different microservices. The data is available right away, but that means IT teams must also continuously monitor, tweak and adjust on-the-fly. It is a challenge.

The business case for microservices

Allan Naim, Product Manager of Container Engine and Kubernetes at Google, told the audience during the panel discussion at Appsphere 15 that it is not easy for IT organizations to incorporate microservices, so they must have an associated business initiative. Often business objectives originate with CEO and Board of Directors. From there, the CMO or the CSO begin to implement them, and it forces the IT staff to start working with microservices. Naim said he sees a time in the not too distant future where every organization, no matter the industry or segment of the market, will ultimately become a software company. That is because the customer data is becoming as valuable as their product or service.

To leverage that asset, organizations must act quickly, changing their offerings based on a constantly evolving landscape. Legacy apps have a hard time adjusting to the new demands of the market such as mobility and the Internet of Things. Competition, especially in the form of aggressive startups that look to disrupt industries, is forcing organizations to integrate microservices architecture with their legacy systems, whether the data is in a relational database or not.

From highly specialized to highly adaptable

It comes down to the need to provide the highest-quality software to large amounts of customers as quickly as possible. Microservices are not only changing the way companies write code; they are changing the companies themselves. For example, in a monolithic system, the roles of each team member tended to be highly specialized.

In the world of microservices, that approach is highly devalued. Instead, it is better for each team member to be free to operate on different parts of the application without interruption. Rather than hand off development to the next stage, the application is constantly being monitored and modified as it is being developed.

Homegrown analytics and monitoring tools

Another development resulting from these market pressures is that IT teams have started building their own tools. Netflix created its own monitoring system. In fact, they custom made some non-unified tools, a very different approach than that taken by companies like Facebook and Google.

For example, they built their analytics software to process huge volumes of data. How much volume are we talking about? Consider this eye-opening statistic: Networking provider Sandvine reports that just over 30 percent of the traffic on the Web during prime time are Netflix customers streaming movies.

The development of microservices is changing more than software code itself. It is making an enormous impact on how organizations think through their business processes, what products they bring to market and how they are going to support their products with customers in the marketplace.

Because of the explosion of mobile devices and the always shifting wants and needs of consumers, IT professionals have to adapt just as quickly. Microservices architecture is the vehicle in which they are creating rapid change. It is changing not only the technology but also how organizations evaluate business opportunities. On another level, it is altering the organization of talent, encouraging a culture of innovation, expanding the scope of individual responsibility and empowering smart people to take chances.

Agility and speed are paramount

Large firms such as Condé Nast and Gilt have always been able to handle large volumes of customer data. However, they see the future and are adapting their legacy systems to utilize microservices architecture. They want to get rid of dependencies and be able to test and deploy code changes quickly. Similar changes across enterprises are helping them become more adaptable to customer needs. It is also pushing them to adopt greater use of the cloud to operate with more agility and speed.

Microservices architecture has a similar mindset as other fast development methodologies like agile software. Fast-moving Web properties like Netflix are constantly looking for greater simplicity and the ability to make changes rapidly without going through numerous committees. The code is small, and every software engineer makes production changes on an ongoing basis.

Sea change in software development

That is why microservices architecture is a natural fit for Web languages such as Node.js that work well in small components. You want to be able to move rapidly and integrate changes to applications quickly. Because microservices are self-contained, it is easy to make changes to a code base and replace or remove services. Instead of rewriting an entire module and trying to propagate across a massive legacy code base, you simply add on a microservice. Any other services that want to tap into the functionality of the additional service can do so without delay.

This is a sea change in how traditional software development takes place. The speed at which code changes in mobile apps and modern websites is way too fast for the legacy software development system. Constantly evolving apps require a new way of thinking.

Changes in organizations

Back in the 1980s, the role of IT departments began to change with the debut of the personal computer. Every year, PCs became more powerful, and technology staff not only supported individual business functions, but they also had to maintain complete processes. Technology and data were moving closer to the center of the business.

By the 1990s, the IT department had become a critical system in every major company. If the computer systems were down for any length of time, it created bottlenecks for every department of the company.

Data-driven design

With microservices, the data inherent to each microservice can only be tapped through its API. The data in the microservice is private. This allows them to be loosely coupled so they can operate and evolve on an independent basis. This creates two challenges: maintaining consistency across several services and implementing queries that grab information from multiple services. With data-driven design, you can experiment and create transactions that cover multiple services consistently.

Unfortunately, many companies still maintain the old software engineering model. However, today they are under pressure to shorten the time to bring new Web and mobile applications to consumers. Speed has become the “coin of the realm.”

Changing culture in traditional IT departments

The rise of microservices is changing a culture in IT that is deeply ingrained. There has always been a division between software development and operations. Now software development is integrated much more tightly with DevOps. Over many years, IT departments had established standards on which technologies they would run. Since these technologies represented serious investments in time and capital, they budgeted carefully for capacity, upgrades and security.

In the brave new world of microservices, department leaders must make significant changes in their organization, so developers play a bigger role in monitoring the software creation during its lifecycle, from development through to production. Interestingly, a similar development happened decades ago when data centers were so complex; Only a select few IT engineers could operate all of the disparate functions. In many cases, the staff maintaining applications were the same people that built them.

Breaking down barriers

In effect, microservices is breaking down barriers between the development of software and its operation. That means that any firm that is considering implementing microservices on any substantial level needs to evaluate if they are ready to operate with this new approach.

It does not mean that legacy systems are being disregarded for the new kid in town. In many cases, the traditional system is doing an excellent job for the organization, so changing it without a business case would be folly.

However, the larger trends of cloud computing, mobile device adoption, and low-cost bandwidth are forever changing the way consumers buy and interact with software applications. The pace of change is dizzying, and the need for speed in application development is greater than ever before.

Chapter 3: Migrating to microservices

We are rushing headlong into an always-on, digital-centric, mobile world. Organizations that fail to modify their approach to technology will be left by the wayside as others incorporate highly flexible and scalable architectures that adapt quickly and efficiently to the demands of the modern marketplace.

The rapid rise in popularity of microservices was driven by these market influences. In just a few short years, companies have implemented various configurations of technologies to offer the best user experience.

Microservices challenges

One of the primary challenges when considering migrating to microservices is that monolithic legacy systems cannot be changed overnight. DevOps and IT managers must decide where and when they can incorporate microservices into their existing applications. This means a whole new way of thinking must be embraced. In the "Four-Tier Engagement Platform" for Forrester Research, Ted Schadler, Michael Facemire, and John McCarthy say it is time to move the technology industry to a four-tier architecture.

In an article for Infoworld, Patrick Nommensen summarized the Four-Tier Architecture. As he explains, the dramatic changes in computing, including the incredible market penetration of mobile devices, means developers must take an entirely new approach to thinking about application development. The Four-Tier approach is broken down into different layers:

- **Client tier:** The delivery of customer experience through mobile clients and the Internet of Things.
- **Delivery tier:** Optimizes user experience by device while personalizing content by monitoring user actions.
- **Aggregation tier:** Aggregates data from the services tier while performing data protocol translation.
- **Services tier:** The portfolio of external services such as Twilio and Box, as well as existing data, services, and record systems already in-house.

Perhaps the biggest difference with this new approach is the separation of the client tier. With this model, the layers underneath are constantly changing based on real-time interaction with users. This method is already being used by high-profile applications such as Uber and Netflix.

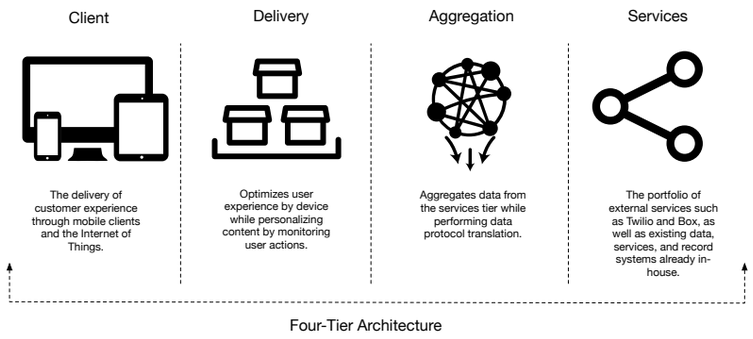


Figure 1: Four-tier architecture

Tools you need

All well and good, but what tools do you need to achieve what these sites have done? The first consideration is that you must decide on a microservices architecture. Figure out how the services will interact before trying to optimize their implementation. Next, while microservices architectures provide much speed, you have to continually optimize those speed gains. This means that you have to be flexible in the tools that you use to deploy the architecture.

Netflix grows their own

The beauty of microservices is they free developers from having to use internal languages or frameworks like Java/Spring or Ruby on Rails. Components can be created with Ruby, Java or JavaScript/Node. Programmers can stick with languages and development tools they know, like and trust, which boosts productivity and satisfaction on the job.

Some companies are building their own tools. It is hard to believe now, but Netflix used to be run on a single war file. They had to use a wide variety of tools, many of which they created when they designed their microservices set-up. It allowed them to be flexible and fast. Here are several of the tools they made:

- **Slalom:** An internal utility that lets a service figure out what the dependencies are -- both upstream and downstream, the level of contribution to service demand, and the health of the request.
- **Mogul:** This utility takes data from the company's Atlas monitoring framework, determines various correlations between metrics and chooses those that might be responsible for fluctuations in demand on specific microservices.
- **Vector:** An on-instance framework which monitors performance by exposing specific metrics to every developer's browser. Netflix added their user interface to an existing framework called Performance Copilot.

The reason the company developed many of these tools is that existing commercial tools did not meet the demand they place on their systems. This is another consideration you must examine when deploying microservices: Can existing commercial tools meet your needs, and/or will you have to develop your own tools?

A practical approach to migration

So how do you handle a migration to microservices? Owen Garret shared with InfoWorld a practical, three-step approach.

1. **Componentize:** Choose a component from your existing applications, and create a microservices implementation on a pilot basis.
2. **Collaborate:** Share the techniques and lessons learned from the pilot in Stage One with all stakeholders, programmers, and developers on the team. This gets them on board with new processes and initiatives.
3. **Connect:** Complete the application and connect to users in a real-world scenario.

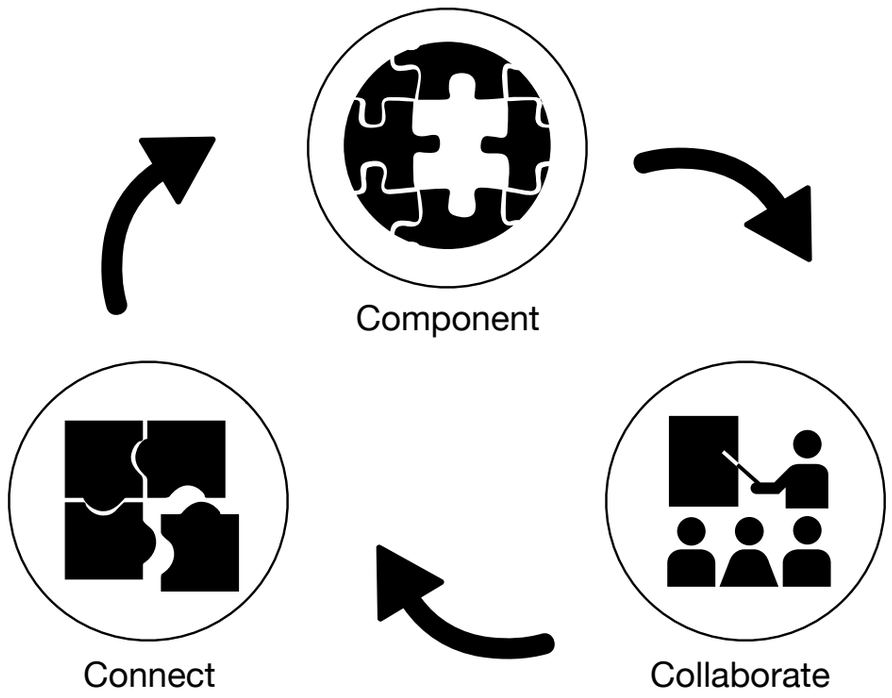


Figure 2: Three-step approach to migrating

Data coupling

Microservices architecture is loosely coupled with data often communicated through APIs. It is not unusual for one microservice to have less than a couple of hundred lines of code and manage a single task. Loose coupling relies on three things:

1. Limited scope and focused intelligence built-in.
2. Intelligence set apart from the messaging function.
3. Tolerance for a wide variety of modifications of microservices with similar functions -- changes are not forced or coordinated.

The APIs translate a specification that creates a contract which indicates what service is provided and how other programs are supposed to use it. Using APIs to decouple services creates a tremendous amount of freedom and flexibility.

New service platforms

Platforms for microservices are evolving rapidly. New platforms are emerging while more established platforms are modifying their approach. Examples include:

- Microsoft's Azure BizTalk Microservices lets clients using Azure build microservices applications in their choice of cloud. It is part of a greater effort to move Azure to a model of small components.
- Gilliam is a Platform as a Service (PaaS) custom made for creating, deploying and scaling microservices. It creates a Docker image of every code repository onsite.
- LSQ is a PaaS with pre-made templates, documentation editor, and NPM package manager. It includes a development environment, assembly and testing area, and cloud deployment.
- Pivotal is a native cloud platform that focuses on developing microservices for companies like Ford, Allstate, Mercedes Benz, GE, Humana, and Comcast.

Open source

As Allen Naim, Product Manager of Kubernetes, told a panel-discussion audience at AppSphere '15, many of the components for microservices are open source. That makes it much simpler to get them to work together, and the ROI is much better than service-oriented architecture (SOA). Chris Morgan, Product Management Director of the OpenShift Partner Ecosystem at Red Hat, agreed. Red Hat was built on open-source code from the beginning, and Morgan said he is excited that many microservices are open source. Old rivals that would never share ideas or code are now collaborating on projects every day. He said they are even friends with Microsoft now, something he could not say only a few years ago.

Examples

What are some examples of companies using microservices right now? They include:

- **Netflix:** As mentioned, Netflix moved rapidly from a single war file to full indoctrination of microservices.
- **The Guardian:** According to Martin Fowler in his seminal article, "Microservices - a definition of this new architectural term," the Guardian newspaper website is a prime example of a firm enhancing their monolithic architecture with microservices. They maintain the core monolith system but use microservices to add new components using the monolith API. This approach is especially well-suited to new pages that are temporary in nature, such as special events.
- **Gilt Groupe:** An online shopping site, Gilt uses a massive microservices architecture with multiple disparate teams working on different services at the same time. One significant change they made was to eliminate their architecture team in favor of engineers making architectural decisions in the individual teams.

Microservices to help legacy apps

Consider a legacy system coded in C and running on multiple mainframes. It has been running for years without any major hiccups and delivers the core competency of the business reliably. Should you attempt to rewrite the code to accommodate new features? A gradual approach is recommended because new microservices can be tested quickly without interrupting the reliability of the current monolithic structure. You can easily use microservices to create new features through the legacy API. Another approach is to modularize the monolithic architecture so that it can still share code and deployments, but move modules into microservices independently if needed.

Boris Scholl, Principal Program Manager for Azure Compute at Microsoft, told his panel discussion audience at Appsphere 15 that Microsoft uses two approaches. Azure is a completely open platform which runs almost everything, including rivals like Red Hat and Kubernetes. In addition, the Azure service fabric adds a layer on top that provides rapid management and orchestration, as well as reliable actors and collection.

People and processes

The deployment of microservices involves more than incorporating new technology. You must adopt new processes and team dynamics to make the transition effective over time. Often managers break applications down by technology, assigning responsibility to different teams. With microservices, applications are separated into services that are grouped by business capability. All software such as user experience, external connections, and storage are implemented within each business domain. Team members handle full application development from user interfaces down to databases.

This change in structure affect the people within it. Developers used to monolithic systems may have a difficult time switching from a world of one language to a multi-language land of new technologies. Microservices frees them up to be more autonomous and responsible for the “big picture.”

However, operating in this new found freedom can be overwhelming for programmers with years of experience in the old ways of doing things. Your must be constantly aware of your team’s ability to change. They may need time to adjust to new guidelines and procedures. Clear communication is the key. Detail their responsibilities in this new style of working and why they are important. Unless you have buy-in from your team members at the start, making adjustments later may be difficult at best and dead on arrival at worst.

New era of computing

We are in a new era of computing based on ultra-fast data processing. Events are monitored, analyzed and processed as they happen. We can make timely decisions based on this continually updated flow of data, resulting in better service for clients, improved operations and instant monitoring of business performance against predetermined targets.

Microservices are not a panacea. There are potential drawbacks such as code duplication, mismatch of interfaces, operations overhead and the challenge of continuous testing of multiple systems. However, the benefits of creating loosely coupled components by independent teams using a variety of languages and tools far outweigh the disadvantages. In our current computing environment, speed and flexibility are the keys to success -- microservices deliver both.

Chapter 4: Challenges in visibility and monitoring

Transitioning to microservices creates significant challenges for organizations. This article examines some of the obstacles you will face, how other companies handled them and the ultimate benefits of your efforts.

Microservices architecture

Microservices architecture is much more complex than legacy systems. In turn, the environment becomes more complicated because teams have to manage and support many moving parts. Some of the things you must be concerned about include:

- As you add more microservices, you have to be sure they can scale together. More granularity means more moving parts which increases complexity.
- When more services are interacting, you increase possible failure points. Smart developers stay one step ahead and plan for failure.
- Transitioning functions of a monolithic app to microservices creates many small components that constantly communicate. Tracing performance problems across tiers for a single business transaction can be difficult. This can be handled by correlating calls with a variety of methods including custom headers, tokens or IDs.
- Traditional logging is ineffective because microservices are stateless, distributed and independent -- you would produce too many logs to easily locate a problem. Logging must be able to correlate events across several platforms.

There are other considerations including:

1. **Operations and Infrastructure:** The development group has to work closer with operations more than ever before. Otherwise, things will spin out of control due to the multitude of operations going on at once.
2. **Support:** It is significantly harder to support and maintain a microservices setup than a monolithic app. Each one may be made from a wide variety of frameworks and languages. The infinite complexities of support influence decisions on adding services. If a team member wants to create a new service in an esoteric language, it impacts the whole team because they have to make sure it can work with the existing setup.
3. **Monitoring:** When you add additional new services, your ability to maintain and configure monitoring for them becomes a challenge. You will have to lean on automation to make sure monitoring can keep up with the changes in the scale of services.
4. **Security of Application:** The proliferation of services in this architecture creates more soft targets for hackers, crackers and criminals. With a variety of operating systems, frameworks and languages to keep track of, the security group has their hands full making sure the system is not vulnerable.
5. **Requests:** One way to send data between services is using request headers. Request headers can contain details like authentication that ultimately reduce the number of requests you need to make. However, when this is happening across a myriad of services, it can increase the need for coordination with members of other teams.
6. **Caching:** Caching helps reduce the number of requests you'll need to make. Caching requests that involve a multitude of services can grow complicated quickly, necessitating communication from different services and their development teams.
7. **Fault Tolerance:** The watchword with microservices is "interdependence." Services have to be able to withstand outright failures and inexplicable timeouts. Failures can multiply quickly, creating a cascading effect through some services, potentially spiking services needlessly. Fault tolerance in this environment is much more complicated than a monolithic system.

De Facto stewards of the monolith

When Phil Calçado was at SoundCloud, one of the challenges he ran into with migrating to microservices was that the code base of the monolithic system had grown so large and complex that nobody understood all of it. Different team members knew sections and became de facto stewards of their area of expertise. Others had to wait around until the resident experts were available to make changes. It was time-consuming and cumbersome.

They also had so many team members working on microservices that they did not have enough staff available to review changes. This shows why, when it comes to microservices, managing staffing issues is as complex and challenging - perhaps more so - than code and technology obstacles.

Componentize, collaborate and connect

Owen Garrett wrote an essay for Infoworld in June of 2015 where he stated there are three different stages of microservices deployment: componentize, collaborate and connect.

1. **Componentize:** Typically in this stage, engineers figure out what new technology pilot project they want to work on. The same approach works when implementing microservices. Working with a system of pilots allows you to figure out new ways of working and new technologies. You must set goals that fit your current needs while realizing you will run into problems. The benefit is that you will be able to use that negative feedback when you create new projects.

   Choose a section of an existing monolithic app you believe can be moved to a microservice without much difficulty. Create an API to tap into this service, and then build a microservice. Using the tools that are familiar to your engineers, you want to create a microservice that you can develop, test and deploy. You will begin to develop a system of continuous delivery you can tweak and modify.
2. **Collaboration:** Solving technical challenges is one factor. For example, traditional logging makes it hard to pinpoint performance issues. However, getting team members to collaborate is the larger goal of the transition process. The knowledge you gain when you develop the pilot project should be shared across the entire development staff. This gets them on board with the process and makes it much easier to gain their support when you expand your microservices development.

   Each team must have a complete skillset to create a service. This includes the data, presentation, and logic -- from beginning to end. Collaboration comes down to sharing technology standards and APIs.
3. **Connection:** Building the individual components of a microservice is only the beginning. They must be connected, and a wholly completed program must be presented to end users. In this way, microservices are more flexible and adaptable than previous approaches such as service-oriented architecture (SOA).

Devs in the spotlight

In old-school, traditional development, there was little integration of the separate functions within the operation of the IT department. DevOps is the evolution of collaboration where the operations, development and quality assurance teams collaborate and communicate throughout the software development process.

DevOps is not a separate role held by a single person or group of individuals. Rather, it conceptualizes the structure needed to help operations and development work closely together. With a microservices architecture, developers are responsible for creating a system to deliver the final product successfully.

Devs must evolve

Along with the continuing migration of large and small organizations to microservices, devs must also evolve. Because it is so easy to deploy microservices, developers are getting involved in code deployments and production monitoring. This contrasts with the traditional scenario where developers would write code and "throw it over the fence" for another team (DevOps) to deploy and maintain. Today, developers and DevOps are merging into smaller application teams responsible for three main components: building, deployment and monitoring.

Microservices are changing how teams are structured, allowing organizations to create teams centered on specific services and giving them autonomy and responsibility in a constrained area. This approach lets the company rapidly adjust in response to fluctuating business demand, without interrupting core activities. It also makes it easier to onboard new staff quickly.

Devs may have to handle some additional challenges including:

- A shortage of developers with JavaScript experience that know how to put together microservices architectures.
- Understanding and implementing services for the Internet of Things.
- The ability to help companies introduce technology into business planning and strategy.
- Teaching business leaders how open APIs can augment their current business lines and open new opportunities in the marketplace.
- How to simplify the development stack, choose the right technology and push back when vendors offer unproductive middleware.
- Learn from industry leaders like Netflix, and decide which implementations of microservices will best serve their organizations.
- Understand that many vendors have still not created a stable microservices platform.
- Be able to handle the pressure of managing and operating possibly hundreds of individual microservices at the same time.
- Manage an increasingly complex network of teams including operations, architects, coders, quality assurance and integrators that still may not completely understand the microservices approach.

Beginning the transition

Once you launch the transition process, you'll notice that new challenges emerge that you did not expect, including:

- How much of the workload should be moved to microservices?
- Should you allow code to be migrated to different services?
- How do you decide what the boundaries of each microservice will be while the operation is running?
- How do you monitor the performance of microservices?

APPDYNAMICS

appdynamics.com