APPDYNAMICS

How to Build and Scale With Microservices

Contents

Chapter 1: Introduction to Microservices

Chapter 2: The Business Case for Microservices

Chapter 3: The Microservices Model

Chapter 4: The Challenges You'll Face Migrating to Microservices

Chapter 5: Starting the Migration Process

Chapter 6: Ongoing Management of Microservices

CHAPTER 1

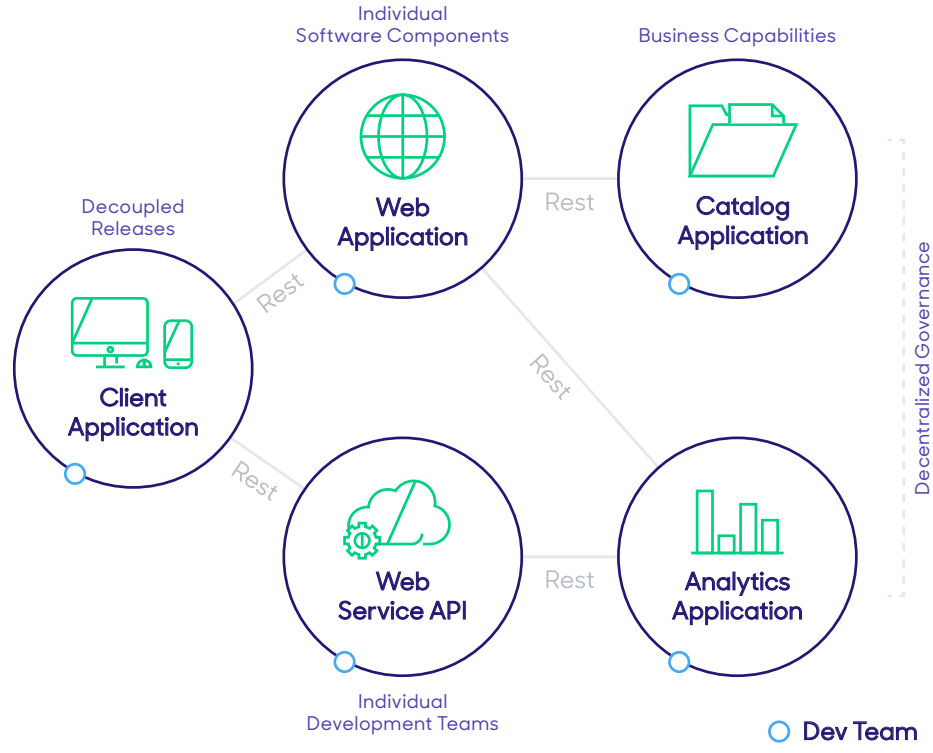
Introduction to Microservices

This short guide discusses the benefits of implementing a microservice-based software architecture. Microservices are revolutionizing the way software applications are designed, simplifying code and enabling rapid implementation of change.

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.

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 communicate with each other and external services through remote procedure calls and 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 collective stakeholder responsibility for a complete stack of functions such as UX and project management.
- **Dumb pipes, smart endpoints:** Microservice applications contain their own logic. Resources that are often used are cached easily.
- **Products rather than projects:** Instead of focusing on a software project that is delivered following completion, microservices teams treat applications as products of which they take ownership. Teams establish an ongoing dialogue with stakeholders, with a goal of continually matching the app to the business function.
- **Decentralized governance:** Tools are built and shared to handle similar problems on other teams.



What Problems Do Microservices Help Solve?

Larger organizations run into problems when monolithic architectures cannot be scaled, upgraded, or maintained as they grow over time. Monolithic implementations frequently try to serve many masters, which means that over time their complexity grows and modification becomes difficult.

To serve clients efficiently, a monolith might end up being stateful — that is, it maintains information about the clients it is serving over time, which means that it also consumes more resources. In an attempt to make monoliths easier to maintain, an architecture group may try to enforce standards over what technologies are used and what design principles are followed. This means that as a monolith adds functionality, the new implementation may not be as efficient as desired because the standards are not tuned to accommodate that particular use case's needs.

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. Rather than accept the enforced standards of architecture groups in large organizations, microservices promote engagement with open formats like HTTP, ATOM, and others, which provides enough consistency in how communication occurs to allow great interoperability.

As applications get bigger, intricate dependencies and connections grow. Whether you're talking about a monolithic architecture or smaller units, microservices let you separate out existing and new application functionalities into manageable and reusable components, allowing software teams to become more agile.

Microservices: Concept Rather Than Technology

It's important to understand that microservices architecture is a conceptual approach and not tied to one or more development technologies. This allows developers to use the languages they are most familiar with or that are most appropriate to solve a particular problem.

All of the following technologies and more are being used to implement microservices:

- Java
- PHP
- Python
- Node.js
- .NET

CHAPTER 2

The Business Case for 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 also separate from it. This decoupling from legacy systems provides the freedom to experiment with new approaches and rapidly iterate changes.

Below are some of the business cases for microservices:

Adaptability and Innovation

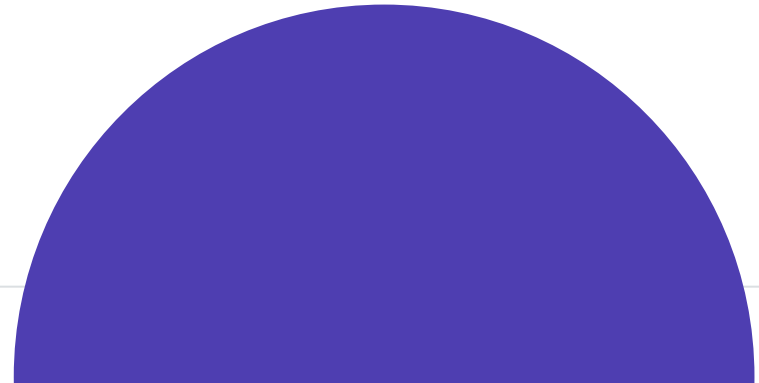
Microservices are also making an enormous impact on how organizations think through their business processes, what products they bring to market, and how they are going to support their products with customers in the marketplace.

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

Enterprises have always been able to build large applications that manage the complexity of their business. But that doesn't mean they've been able to deliver or modify these applications quickly to adjust to business requirements and customer demands. Many are therefore adapting their legacy systems to utilize microservices architecture to get rid of dependencies and be able to test and deploy code changes quickly, making them more adaptable to customer needs.

Companies that have adopted fast development methodologies like agile software are constantly looking for greater simplicity and the ability to make changes rapidly without going through numerous committees. A microservices approach is often the answer. The code is small, and every software engineer makes production changes on an ongoing basis.



Transforming Software Development

The beauty of microservices is that it frees developers from having to use "company standard" languages or frameworks. Components can be created with Ruby, Java, or JavaScript/Node.js, depending on the preference of the programmer and requirements of the particular business problem the microservice addresses. Programmers can either stick with languages and development tools they know, like, and trust, or choose to adopt something new – which boosts productivity and satisfaction on the job.

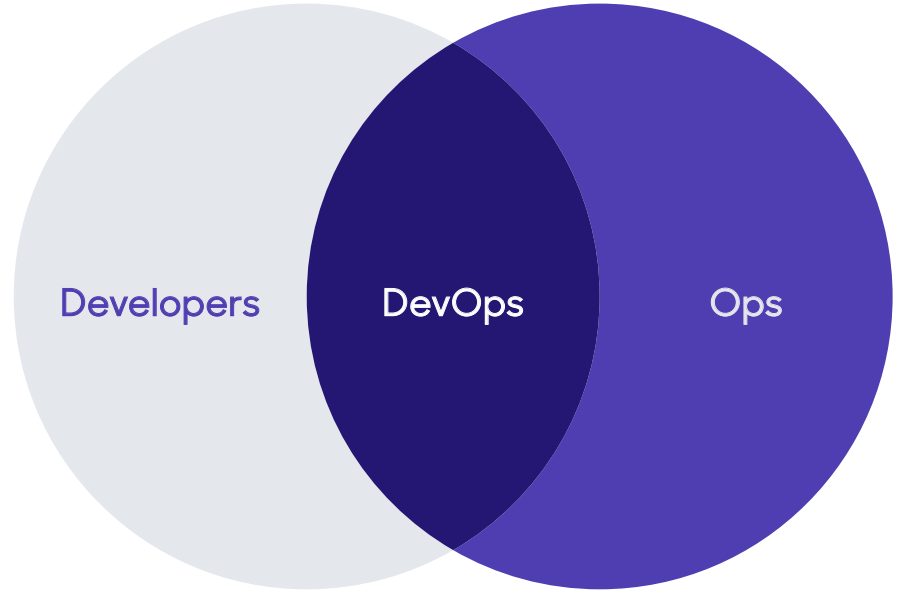
This is a transformation in how traditional software development takes place. The speed at which code changes in mobile apps and modern websites is way too fast for the legacy software development system. Constantly evolving apps require a new approach, like microservices.

Breaking Down Barriers

The rise of microservices is also changing an IT culture that is deeply ingrained: a division between software development and operations. 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 (Ops) to deploy and maintain. Today, developers and Ops are merging into small, focused teams responsible for building, deploying, and monitoring application components. This practice and philosophy is now being referred to as DevOps.

As a result, microservices help break down barriers between the development of software and its operation – something that's critical during a time when the pace of change is dizzying, and the need for speed in application development is greater than ever before.



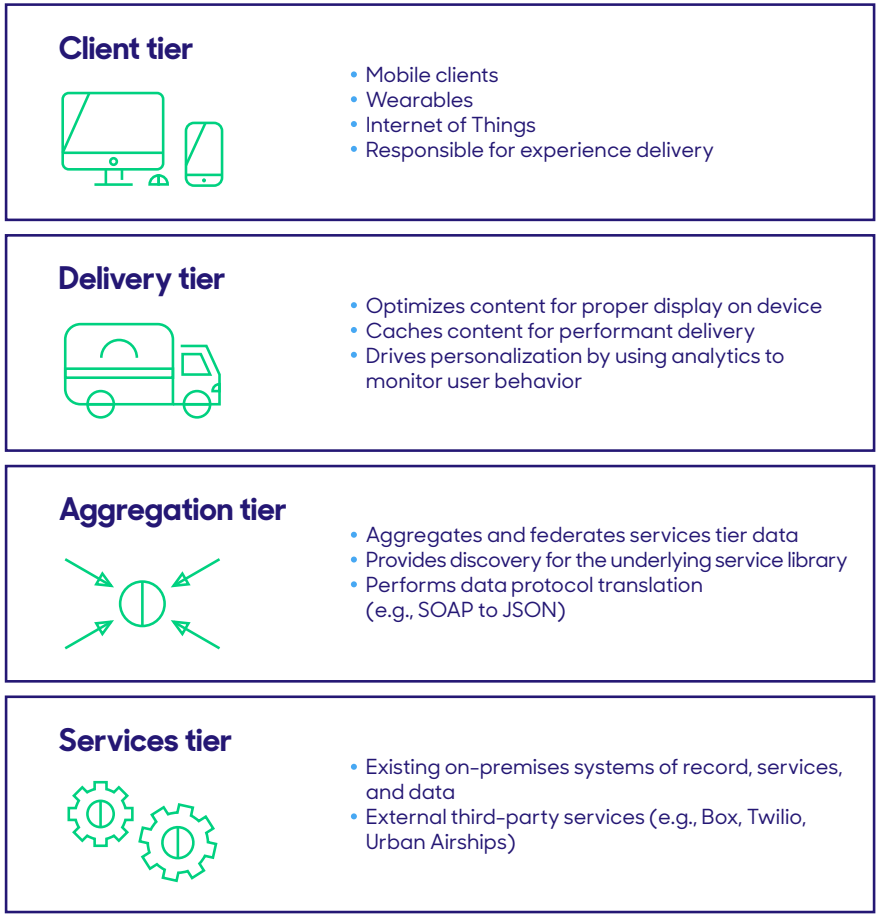
CHAPTER 3

The Microservices Model

We're 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 the popularity of microservices was driven by these market influences. In fact, **Forrester Research** notes that a new architectural approach has emerged that offers agility, flexibility, and scalability:

The Four-Tier Engagement Platform.



As you can see, this 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.

The services tier is where microservices fits in, with **Forrester** noting:

> "A services tier spans internally and externally provisioned data and functionality. This final architectural element dynamically composes data and business processes through a set of continuously deployable services. This tier provides data to the layers above without concern for how that data is consumed; the other layers can exist behind the corporate firewall or externally — or both! This allows for the ultimate flexibility in the consumption and dynamic composition of services, whether leveraged by apps or by the evolving partner ecosystem."

The shift to microservices is clear. The adoption is driven by the confluence of mobile computing, inexpensive hardware, cloud computing, and low-cost storage.

Data Coupling

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

- Limited scope and focused intelligence built-in
- Intelligence set apart from the messaging function
- Tolerance for a wide variety of modifications of microservices with similar functions

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.

Microservices and Containers

When discussing microservices, it's hard not to mention containers. As you're probably aware, containers allow application components (including microservices) to be packaged as small, self-contained, highly scalable modules. Containers have many similarities to virtual machine instances. However, they are much simpler to manage and do not require a hypervisor layer to function.

Major container service providers include **Docker** and **Kubernetes**, who offer frameworks and orchestration to help build, manage, and deploy your applications. Containers fit in well with the microservice model of breaking down application functionality into small, manageable components.

Multiple containers can be deployed in clusters and managed using a range of tools. Many containers will be prebuilt components that can be layered together to build up application images. This makes it easy to update individual containers while an application is still running – reducing the need for downtime and any impact on business continuity. This is particularly beneficial for microservice-based applications where frequent deployments of code changes to production are the norm.

CHAPTER 4

The Challenges of Migrating to Microservices

A microservices architecture significantly enhances the agility and accelerates the velocity of continuous integration and delivery of enterprise applications.

However, this approach can also result in an exponentially larger number of microservices that are loosely coupled and communicate primarily via asynchronous mechanisms, creating increased complexity and a significant management challenge.

DevOps Guidance

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.

Below are some challenges to keep in mind:

Operations and infrastructure

With microservices, things can spin out of control due to the multitude of operations going on at once. As a result, the development group has to work more closely with operations than ever before.

Monitoring

When you add additional new services, your ability to maintain and configure monitoring for them becomes a challenge. You'll have to lean on automation to make sure monitoring can keep up with the changes in the scale of services.

Legacy considerations

Consider a legacy system coded in C and running on multiple mainframes. It has been running successfully for years 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. Also consider modularization of the legacy architecture so you can still share code and deployments, but move modules into microservices independently if needed.

Security

The proliferation of services in this architecture creates more soft targets for hackers 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 isn't vulnerable.

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.

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.

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.

Fault tolerance

The watchword with microservices is "interdependence." Services have to be able to withstand outright failures and inexplicable timeouts when communicating with each other. 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.

Human resourcing

When it comes to microservices, managing staffing issues is as complex and challenging — perhaps more so — than code and technology obstacles. You can easily have so many team members working on microservices that you may not have enough staff available to review changes.

As you can see, while there are certainly benefits to microservices, there are also challenges that come with it. However, the benefits of creating loosely coupled components by independent teams far outweigh the disadvantages. In our current computing environment, speed and flexibility are the keys to success, and microservices deliver both.

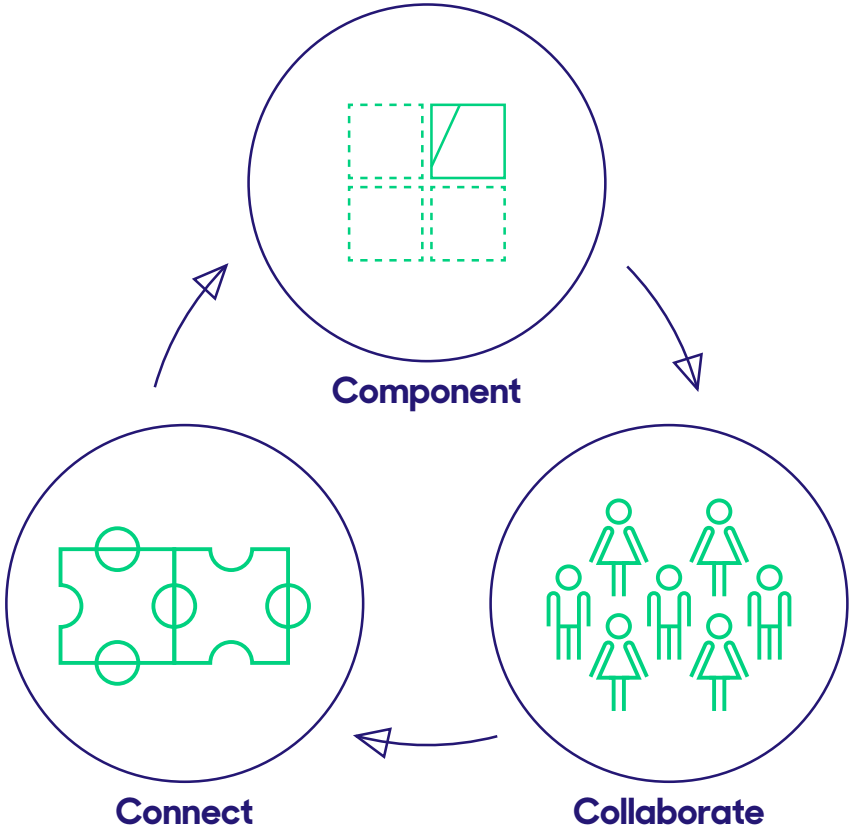
CHAPTER 5

Starting the Migration Process

So how do you handle a migration to microservices?

Owen Garrett wrote an essay for **InfoWorld** outlining three different stages of microservices deployment:

Componentize, Collaborate, and Connect.



1. COMPONENTIZE

To start, choose a pilot project you want to work on. The best approach may be to select a section of an existing monolithic app you believe can be moved to a microservice without much difficulty. You want to create a microservice that you can develop, test, and deploy.

Next, decide on the application functionality this microservice will expose and then create an API using tools familiar to your engineers. Remember that microservices are intended to be simple to design, code, and consume. It is far better to create multiple microservices than try to make a single service too complex. In this way, you'll promote service reuse and begin to develop a system of continuous delivery you can tweak and modify.

2. COLLABORATION

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.

What's more, 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.

Implementing microservices also provides an opportunity to review your approach to performance engineering. The simple consumer endpoint model lends itself well to early performance testing, enabling Dev to trend service endpoint performance in-sprint, across builds, and against performance SLAs.

3. CONNECTION

Finally, building the individual components of a microservice is only the beginning. The functionality provided by the service endpoints must now be integrated into your application stack. In other words, they must be connected, and the results published to promote reuse across other Dev teams.

In this way, microservices can be shown to be more flexible and adaptable than previous approaches such as service-oriented architecture (SOA).

CHAPTER 6

Ongoing Management of Microservices

While microservices can certainly improve the agility and velocity of enterprise application delivery, there are increased complexity and management challenges which must be addressed to ensure that your microservices architecture is functioning efficiently.

For example, as you add more microservices, you have to ensure they can scale together. More granularity means more moving parts, which increases complexity.

This requires enterprise-wide visibility, and AppDynamics' end-to-end monitoring solution offers the following microservice-focused support:

TRACK MICROSERVICES DEPLOYED IN AN ELASTIC INFRASTRUCTURE

- Efficiently track microservices deployed in an elastic infrastructure such as containers or cloud where nodes scale up and down very rapidly
- Retain historical data about the microservice and infrastructure nodes and correlate it with past and future instances of the microservice

CORRELATE YOUR APPLICATION, CONTAINER, AND UNDERLYING HOST METRICS

- Automatic discovery of entry and exit points of your microservice as service endpoints for focused microservices monitoring
- Track the key performance indicators (KPIs) of your microservice as distinct from transaction-based monitoring
- Drill down and isolate the root cause of microservice performance issues

CHECK AVAILABILITY OF MICROSERVICES DEPLOYED WITHIN YOUR NETWORK OR EXTERNALLY

- Check availability and basic performance metrics for HTTP-based microservices that are not monitored by an AppDynamics agent

Conclusion

In summary, AppDynamics helps to simplify your microservice implementation and ongoing management by giving you real-time visibility into service performance and availability, in addition to comprehensive application monitoring.

WANT TO LEARN MORE ABOUT APPDYNAMICS?

Take a Guided Tour or Schedule a Demo Today

GUIDED TOUR **SCHEDULE A DEMO**

APPDYNAMICS

AppDynamics is now part of Cisco. CISCO