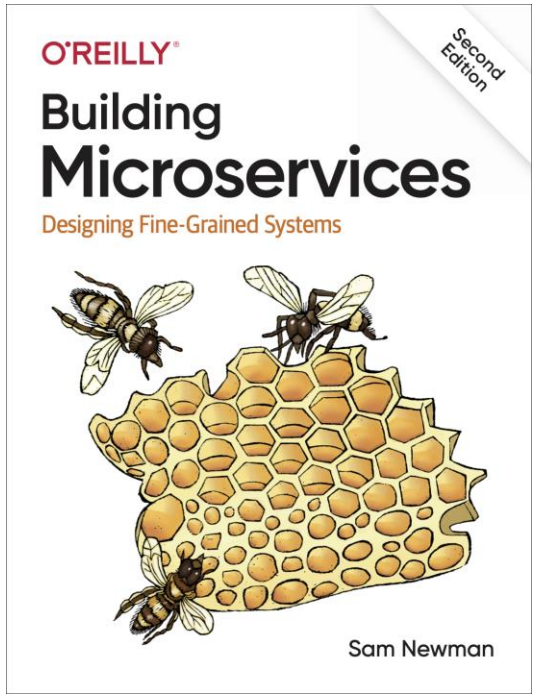
Microservices

IF3110 – Web-based Application Development
School of Electical Engineering and Informatics
Institut Teknologi Bandung

Reference



Sam Newman - Building Microservices_ Designing Fine-Grained Systems-O'Reilly Media (2021)

Part 1 Foundation
Chapter 1 What Are Microservices?

What is Microservices?

Microservices are an architecture choice for building our systems.

Microservices are *independently releasable services that are modeled around a business domain*.

A service encapsulates functionality and makes it accessible to other services via networks—you construct a more complex system from these building blocks.

For example:
One microservice might represent inventory, another order management, and yet another shipping, but together they might constitute an entire ecommerce system.

What is Microservices?

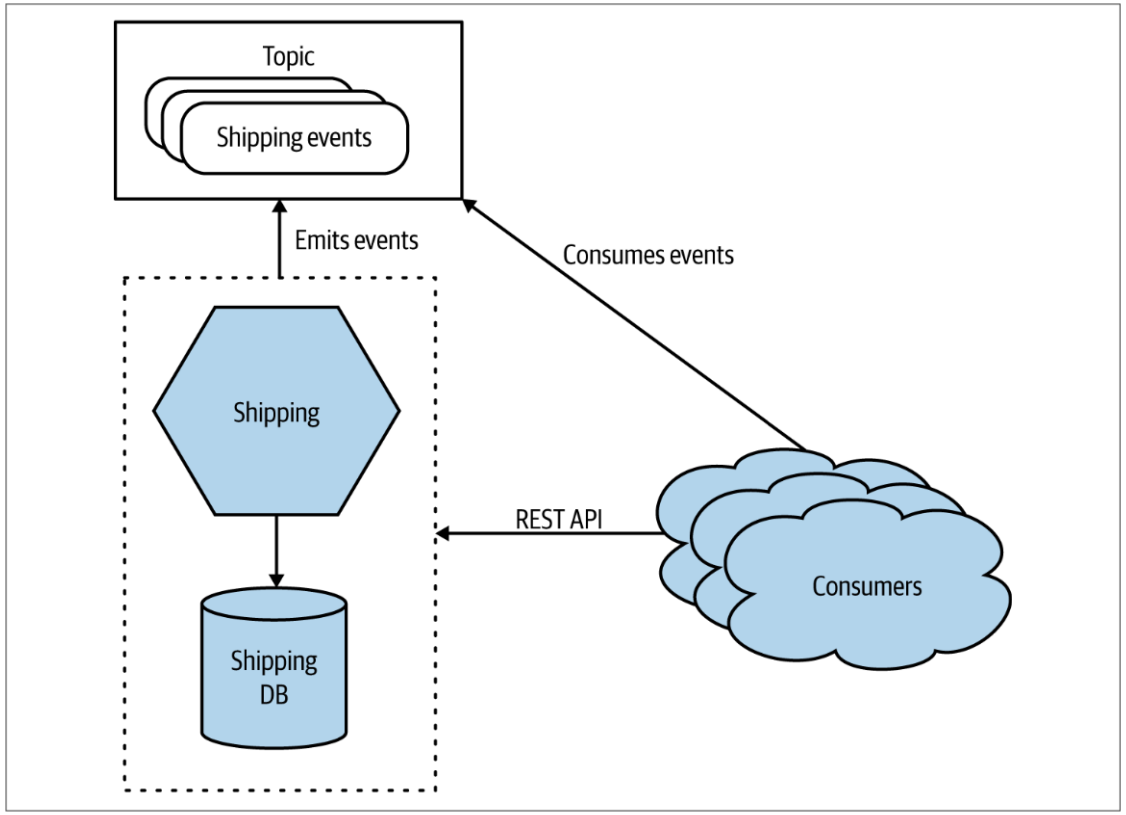


Figure 1-1. A microservice exposing its functionality over a REST API and a topic

Microservices embrace the concept of information hiding.

Information hiding means hiding as much information as possible inside a component and exposing as little as possible via external interfaces.

From the outside, a single microservice is treated as a black box.

Service-Oriented Architecture and Microservices

Service-oriented architecture (SOA) is a design approach in which multiple services collaborate to provide a certain end set of capabilities.

You should think of microservices as being a specific approach for SOA in the same way that Extreme Programming (XP) or Scrum is a specific approach for Agile software development.

Key Concepts of Microservices

Independent Deployability

Modeled Around a Business Domain

Owning Their Own State

Size: How big should a microservice be?

Flexibility

Independent Deployability

Independent deployability is the idea that we can make a change to a microservice, deploy it, and release that change to our users, without having to deploy any other microservices.

To ensure independent deployability, we need to make sure our microservices are *loosely coupled*: we must be able to change one service without having to change any- thing else.

Modeled Around a Business Domain

By modeling services around business domains, we can make it easier to roll out new functionality and to recombine microservices in different ways to deliver new functionality to our users.

By making our services end-to-end slices of business functionality, we ensure that our architecture is arranged to make changes to business functionality as efficient as possible.

Arguably, with microservices we have made a decision to prioritize high cohesion of business functionality over high cohesion of technical functionality.

Owning Their Own State

The idea that microservices should avoid the use of shared databases.

If a microservice wants to access data held by another microservice, it should go and ask that second microservice for the data.

This gives the microservices the ability to decide what is shared and what is hid-den, which allows us to clearly separate functionality that can change freely (our internal implementation) from the functionality that we want to change infrequently (the external contract that the consumers use).

Size

“How big should a microservice be?”

Considering the word “micro” is right there in the name.

How do you measure size? By counting lines of code?

“a microservice should be as big as my head.” James Lewis

The rationale behind this statement is that a microservice should be kept to the size at which it can be easily understood.

Flexibility

“microservices buy you options.” – James Lewis

They have a cost, and you must decide whether the cost is worth the options you want to take up.

The resulting flexibility on a number of axes—organizational, technical, scale, robustness—can be incredibly appealing.

Alignment of Architecture and Organization

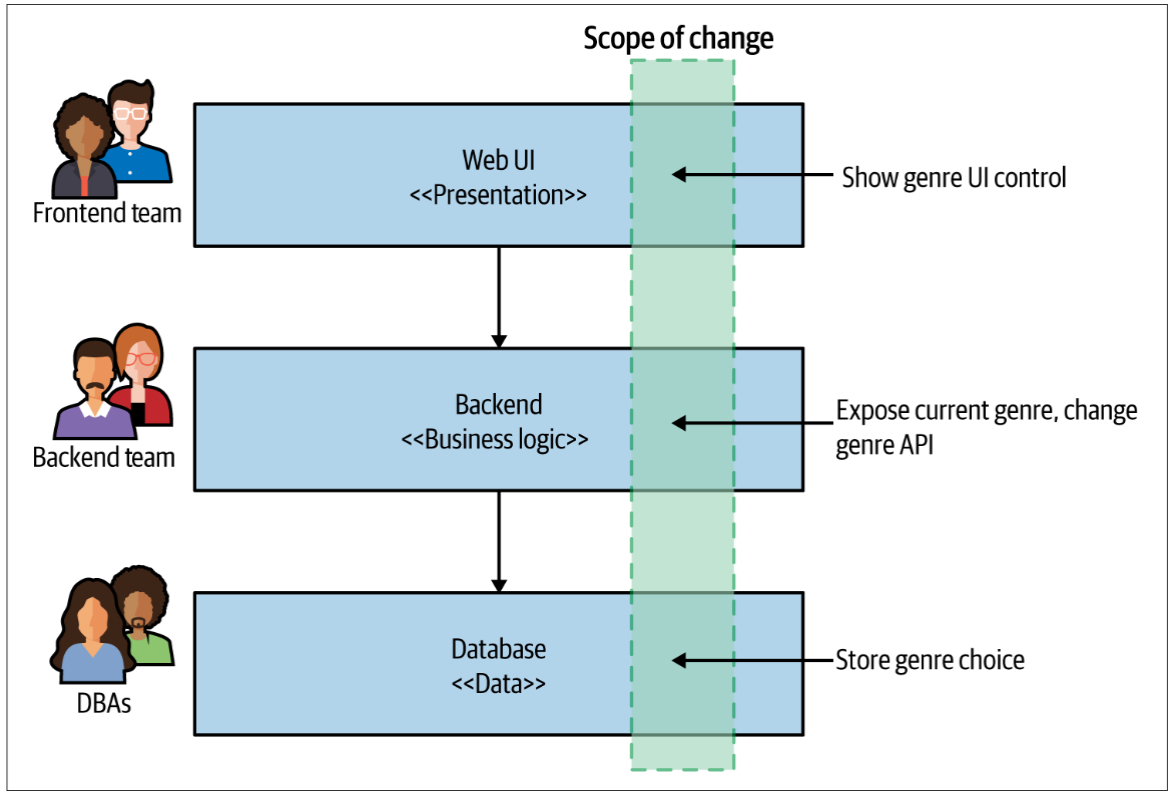


Figure 1-3. Making a change across all three tiers is more involved

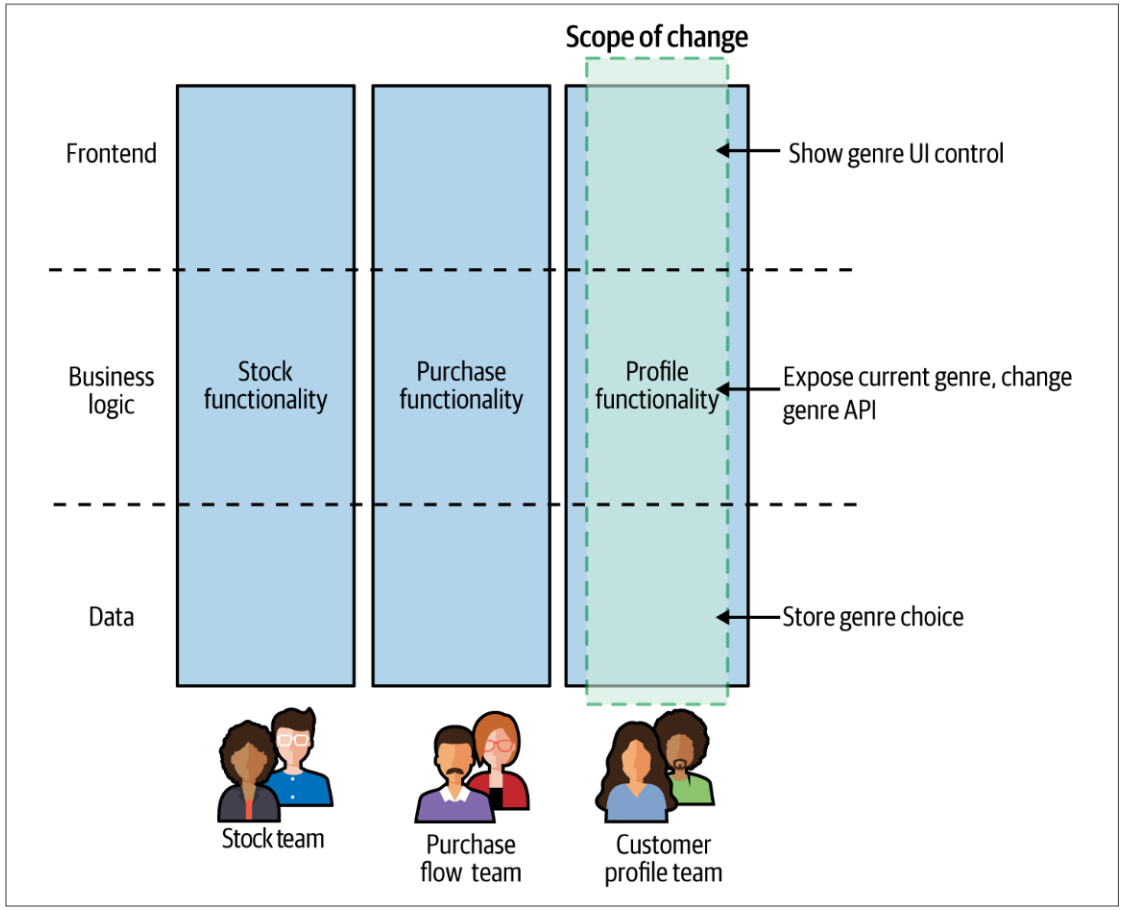


Figure 1-4. The UI is broken apart and is owned by a team that also manages the server-side functionality that supports the UI

The Monolith

Microservices are most often discussed as an architectural approach that is an alternative to monolithic architecture.

A system in which all functionality must be deployed together is a monolith.

The term "monolith" often refers to the concept of a single, unified unit of deployment.

The Single-Process Monolith

The most common example that comes to mind when discussing monoliths is a system in which all of the code is deployed as a *single process*

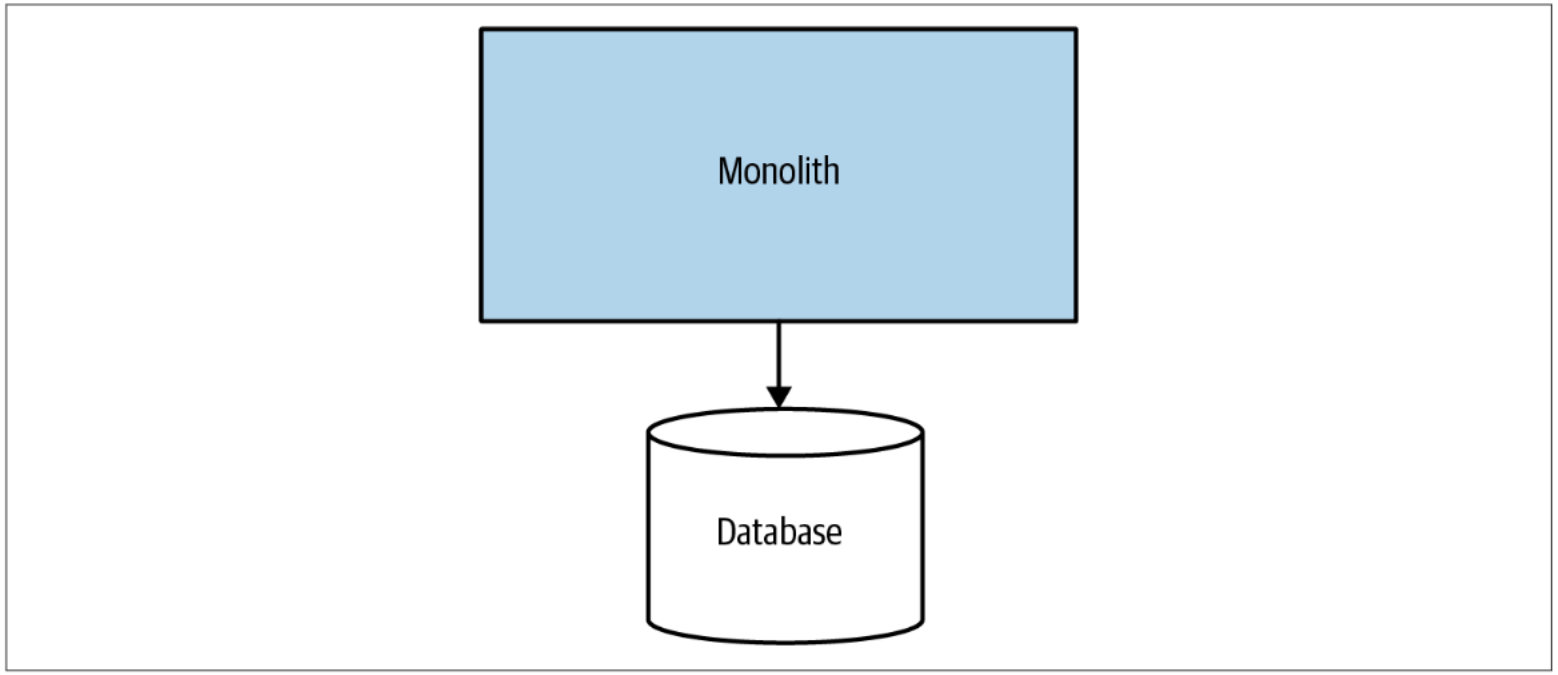


Figure 1-6. In a single-process monolith, all code is packaged into a single process

The Modular Monolith

As a subset of the single-process monolith, the modular monolith is a variation in which the single process consists of separate modules. Each module can be worked on independently, but all still need to be combined together for deployment.

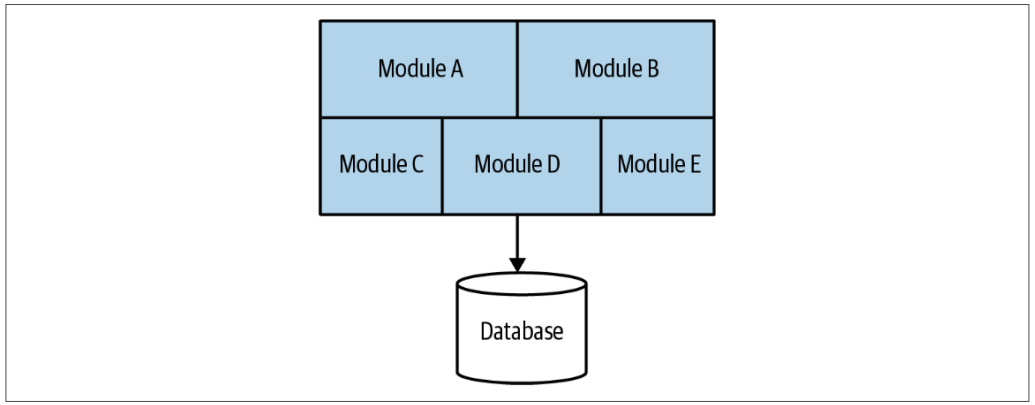


Figure 1-7. In a modular monolith, the code inside the process is divided into modules

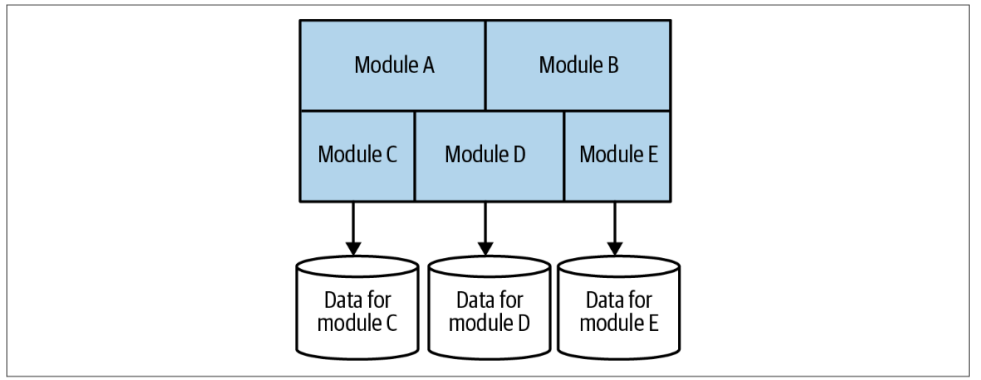


Figure 1-8. A modular monolith with a decomposed database

The Distributed Monolith

A distributed monolith is a system that consists of multiple services, but for whatever reason, the entire system must be deployed together.

A distributed monolith might well meet the definition of an SOA, but all too often, it fails to deliver on the promises of SOA.

The Distributed Monolith

A distributed monolith is a system that consists of multiple services, but for whatever reason, the entire system must be deployed together.

A distributed monolith might well meet the definition of an SOA, but all too often, it fails to deliver on the promises of SOA.

Monoliths and Delivery Contention

As more and more people work in the same place, they get in one another's way—for example, different developers wanting to change the same piece of code, different teams wanting to push functionality live at different times (or to delay deployments), and confusion around who owns what and who makes decisions.

Having a monolith doesn't mean you will definitely face the challenges of delivery contention any more than having a microservice architecture means that you won't ever face the problem.

But a microservice architecture does give you more concrete boundaries around which ownership lines can be drawn in a system, giving you much more flexibility when it comes to reducing this problem.

Advantages of Monolith

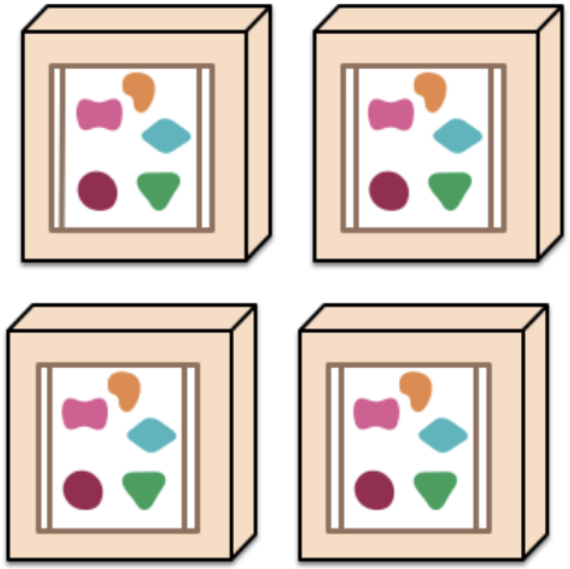
Their much simpler deployment topology can avoid many of the pitfalls associated with distributed systems. This can result in much simpler devel- oper workflows, and monitoring, troubleshooting, and activities like end-to-end test- ing can be greatly simplified as well.

Monoliths can also simplify code reuse within the monolith itself. If we want to reuse code within a distributed system, we need to decide whether we want to copy code, break out libraries, or push the shared functionality into a service. With a monolith, our choices are much simpler, and many people like that simplicity—all the code is there; just use it!

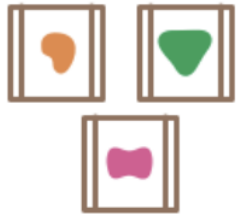
Distributed Monolith → Microservice

A monolithic application puts all its functionality into a single process...

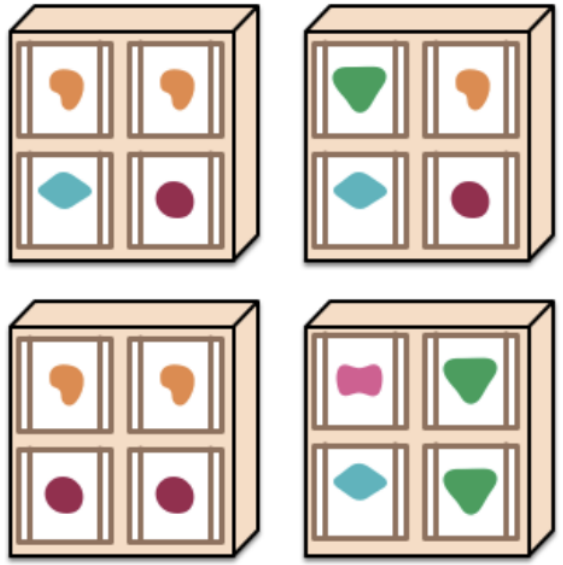
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



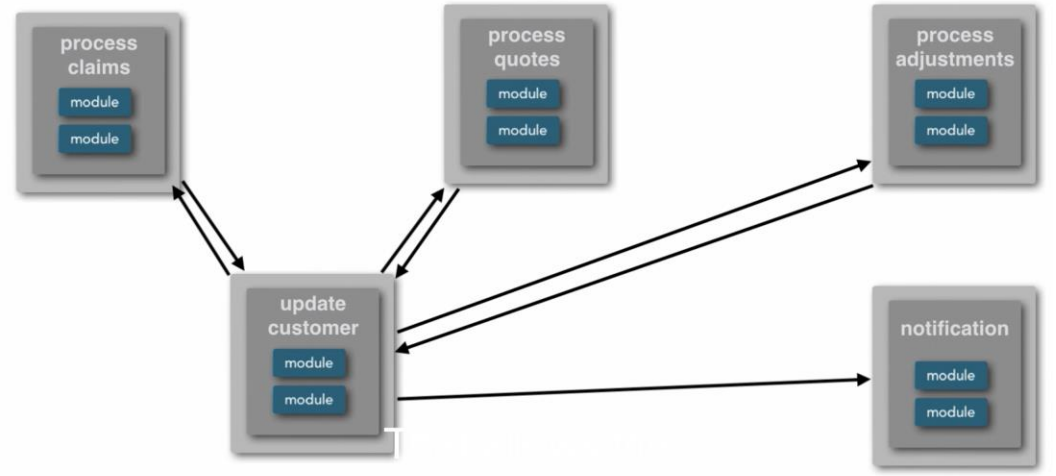
... and scales by distributing these services across servers, replicating as needed.



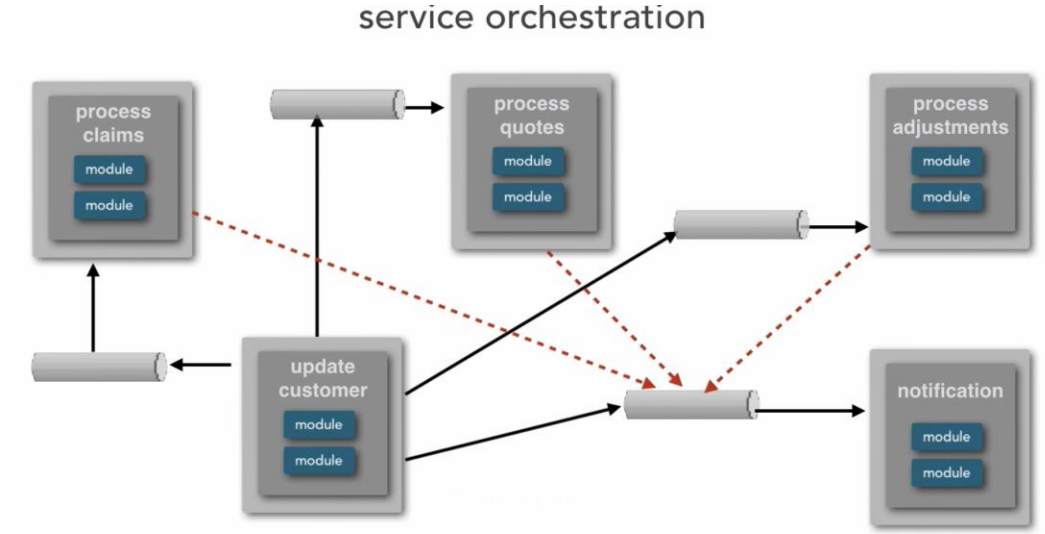
Choreography of series Services

Service Orchestration

Front Orchestration (Orchestration generally is not preferred in microservices although)



Choreography With Message Queue



Advantages of Microservices [1]

Technology Heterogeneity

With a system composed of multiple, collaborating microservices, we can decide to use different technologies inside each one. This allows us to pick the right tool for each job

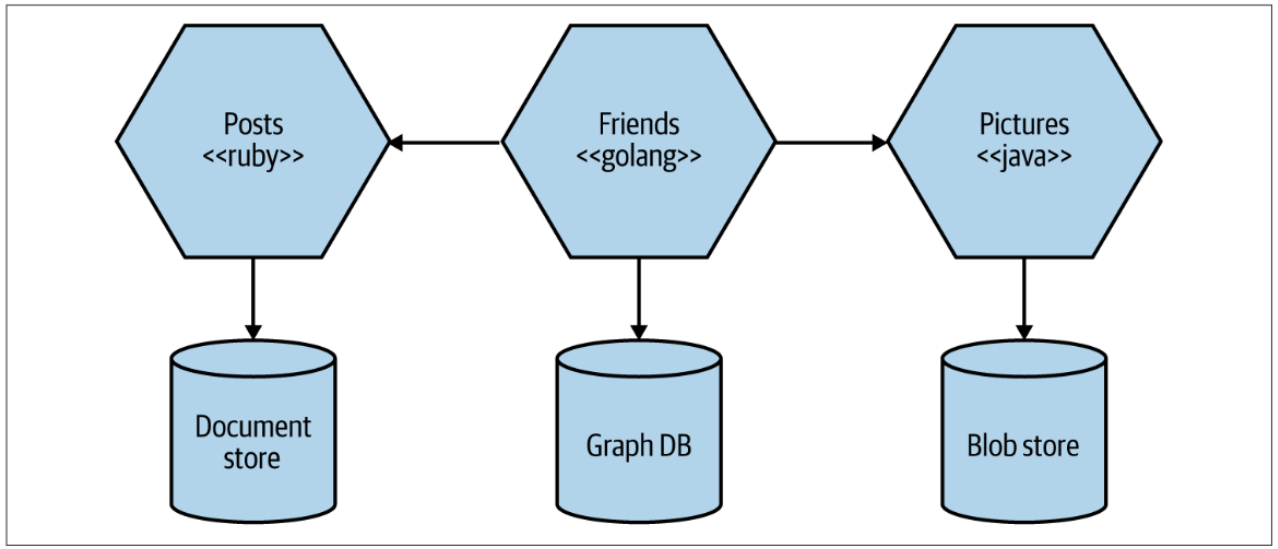


Figure 1-10. Microservices can allow you to more easily embrace different technologies

Advantages of Microservices [2]

Scaling

With smaller services, we can scale just those services that need scaling, allow- ing us to run other parts of the system on smaller, less powerful hardware

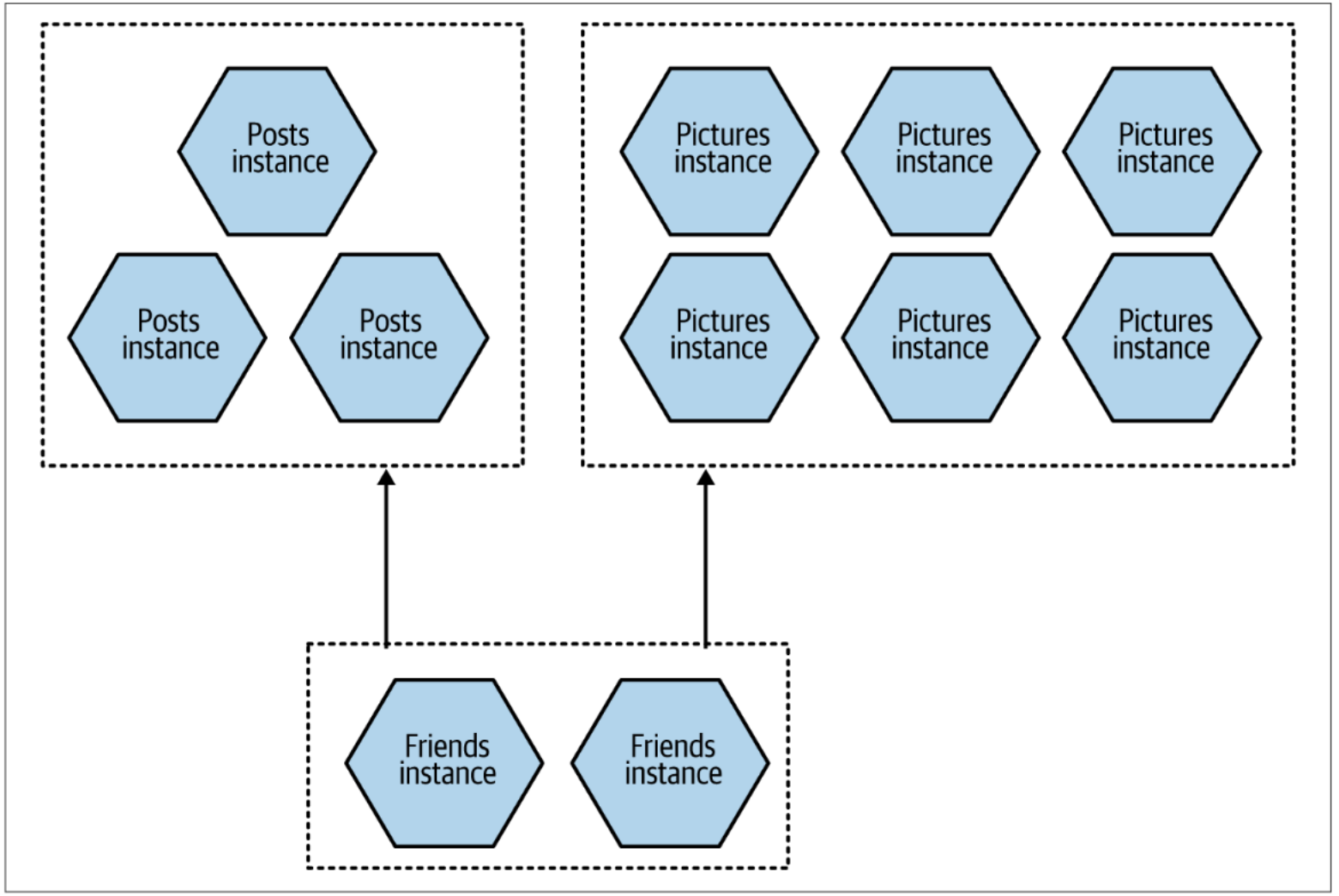


Figure 1-11. You can target scaling at just the microservices that need it

Advantages of Microservices [3]

Ease of Deployment

A one-line change to a million-line monolithic application requires the entire appli- cation to be deployed in order to release the change. That could be a large-impact, high-risk deployment.

With microservices, we can make a change to a single service and deploy it independ- ently of the rest of the system.

Organizational Alignment

Many of us have experienced the problems associated with large teams and large codebases. These problems can be exacerbated when the team is distributed. We also know that smaller teams working on smaller codebases tend to be more productive.
Microservices allow us to better align our architecture to our organization, helping us minimize the number of people working on any one codebase to hit the sweet spot of team size and productivity.

Advantages of Microservices [4]

Composability

One of the key promises of distributed systems and service-oriented architectures is that we open up opportunities for reuse of functionality.

With microservices, we allow for our functionality to be consumed in different ways for different purposes. This can be especially important when we think about how our consumers use our software.

Microservices Pain Points [1]

Developer Experience

As you have more and more services, the developer experience can begin to suffer.
For example: More resource-intensive runtimes like the JVM can limit the number of microservi- ces that can be run on a single developer machine.

Technology Overload

The sheer weight of new technology that has sprung up to enable the adoption of microservice architectures can be overwhelming.
There is a danger, though, that this wealth of new toys can lead to a form of technology fetishism.
You have to carefully balance the breadth and complexity of the technology you use against the costs that a diverse array of technology can bring.

Microservices Pain Points [2]

Cost

Firstly, you'll likely need to run more things—more processes, more computers, more network, more storage, and more supporting software (which will incur additional license fees).

Secondly, any change you introduce into a team or an organization will slow you down in the short term. It takes time to learn new ideas, and to work out how to use them effectively.

Reporting

With a microservice architecture, we have broken up this monolithic schema. That doesn't mean that the need for reporting across all our data has gone away; we've just made it much more difficult, because now our data is scattered across multiple logi- cally isolated schemas.

Microservices Pain Points [3]

Monitoring and Troubleshooting

With a microservice architecture, do we understand the impact if just a single instance of a service goes down?

Security

You might need to direct more care to protecting data in transit and to ensuring that your microservice endpoints are protected so that only authorized parties are able to make use of them

Testing

With a microservice architecture, the scope of our end-to-end tests becomes very large. We would now need to run tests across multiple processes, all of which need to be deployed and appropriately configured for the test scenarios.

Microservices Pain Points [4]

Latency

With a microservice architecture, processing that might previously have been done locally on one processor can now end up being split across multiple separate micro- services. Information that previously flowed within only a single process now needs to be serialized, transmitted, and deserialized over networks that you might be exerising more than ever before.

Data Consistency

Shifting from a monolithic system, in which data is stored and managed in a single database, to a much more distributed system, in which multiple processes manage state in different databases, causes potential challenges with respect to consistency of data.

Should I Use Microservices?

You need to assess your own problem space, skills, and technology landscape and understand what you are trying to achieve before deciding whether microservices are right for you. They are an architectural approach, not the architectural approach.

Microservices are very often a solution to an organizational problem rather than a technical one. That organizational problem is often about enabling large numbers of people to work on the system effectively, in parallel. It can also be about developer experience, which is important in speeding up delivery of value—because developers don’t have to fight the process and the tooling—but also in recruiting and retaining people.

Sticking with a Monolithic Architecture

Unless you have compelling reasons to adopt microservices *and* can get the conditions in place to be successful, be wary.

There are many advantages of building a monolith, or at least of building a monolith first. For example, they are a lot simpler to operate and to understand. And you can invest in making changes to the monolith to tackle those big challenges that tend to make people look toward microservices.

Specifically, you can address the challenges of scaling your organization and with developer experience by building a modular monolith, one where the code is structured into domains with clearly defined boundaries.