From Monolith To Microservices

05-02-2024

Presented By Benmoussa Younes

Table of content

Introduction

By Default any application should be monolith

Understanding the Goal

- What are you hoping to achieve?
- Have you considered alternatives to using microservices?
- How will you know if the transition is working?

1. What Are Microservices?

Microservices are independently deployable services modeled around a business domain.

They are a type of service-oriented architecture (SOA)

They communicate with each other via networks, and as an architecture choice offer many options for solving the problems we may face.

It follows that a microservice architecture is based on multiple collaborating microservices.

1.2 Independent Deployability

Independent deployability is the idea that we can make a change to a microservice and deploy it into a production environment without having to utilize any other services.

We need to be able to change one service without having to change anything else.

This means we need explicit, well-defined, and stable contracts between services.

1.3 Modeled Around a Business Domain

Making a change across a process boundary is expensive.

We want to find ways of ensuring we make cross-service changes as infrequently as possible

If you need to make a change to two services to roll out a feature, that takes more work than making the same change inside a single service

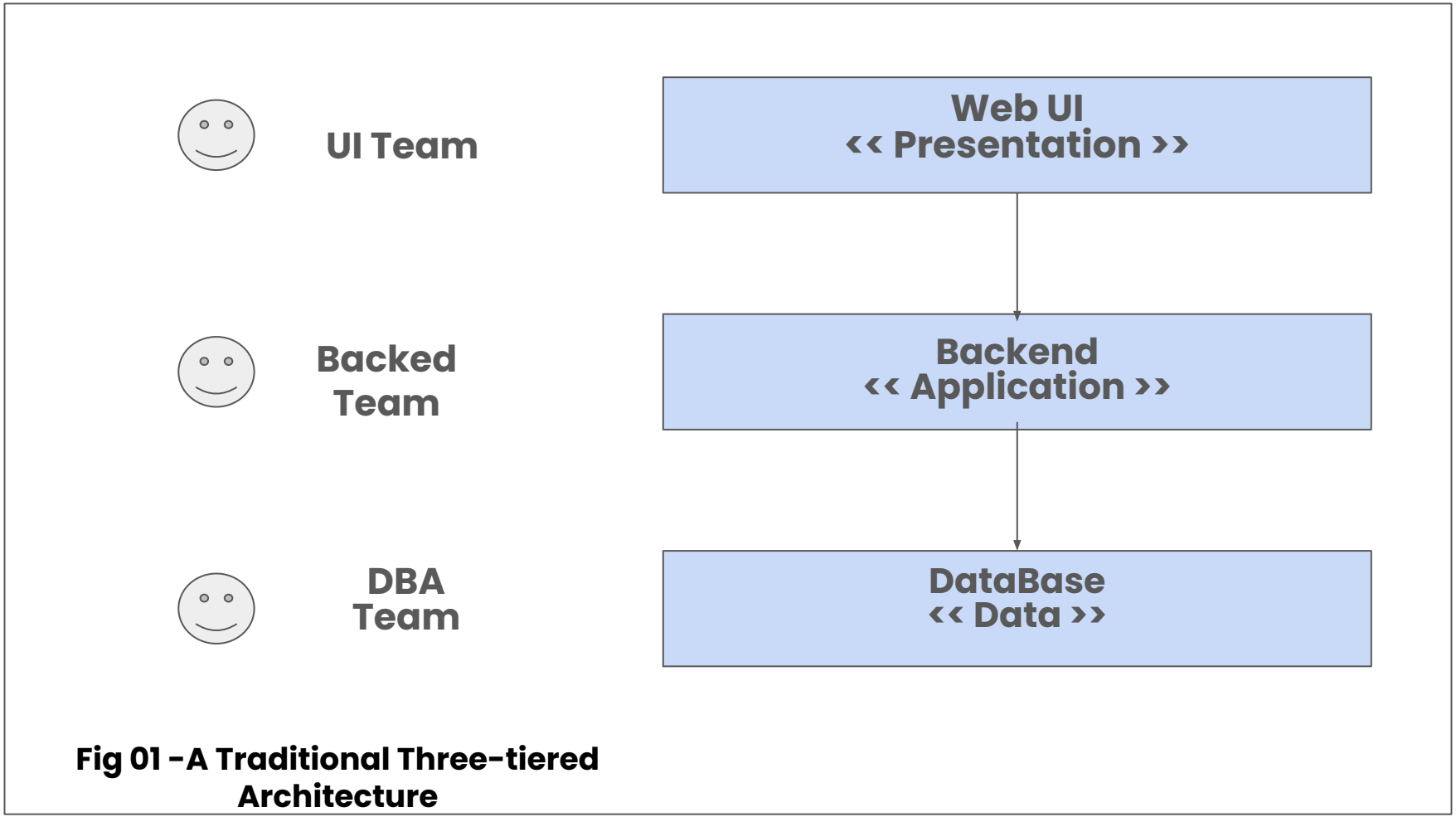


Fig 01 -A Traditional Three-tiered Architecture

The company in question is Music Corp despite it focusing almost entirely on selling CDs.

We want to allow our customers to specify their favorite genre of music

This change requires us to change the user interface, the backend service and the database to accept this change

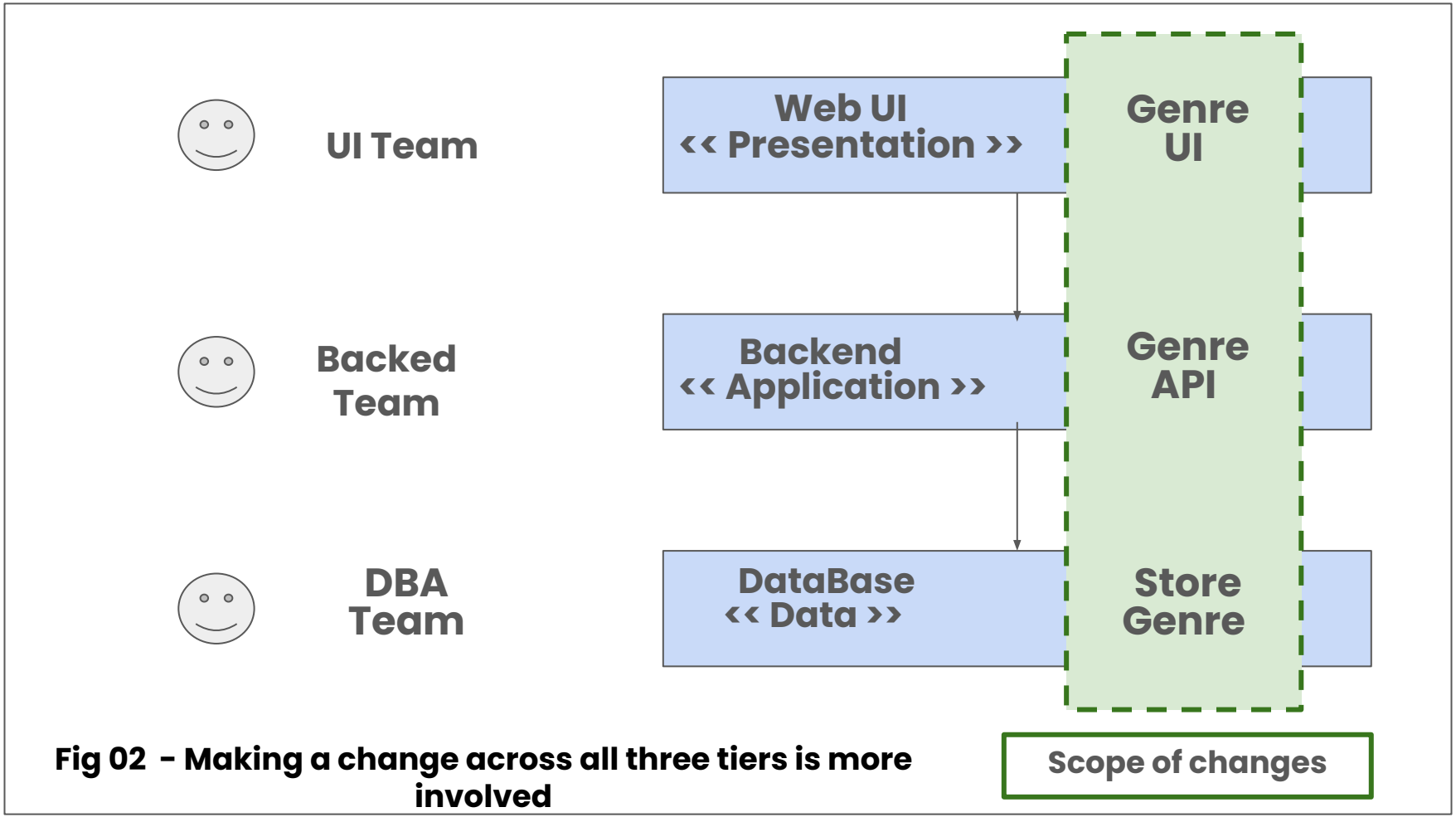


Fig 02 - Making a change across all three tiers is more involved

Any organization that designs a system…will inevitably produce a design whose structure is a copy of the organization's communication structure.

— **Melvin Conway, How Do Committees Invent?**

The three tiered architecture is a good example of this in action.

In the past, the primary way IT organizations grouped people was in terms of their core competency

This architecture is so common. It’s not bad, it’s just optimized around one set of forces

Our aspirations around our software have changed. We now group people in poly skilled teams.

We want to ship software much more quickly than ever before.

Changes in functionality are primarily about changes in business functionality.

In figure 02 , this is an architecture in which we have high cohesion of related technology, but low cohesion of business functionality increasing the chance that a change in functionality will cross layers.

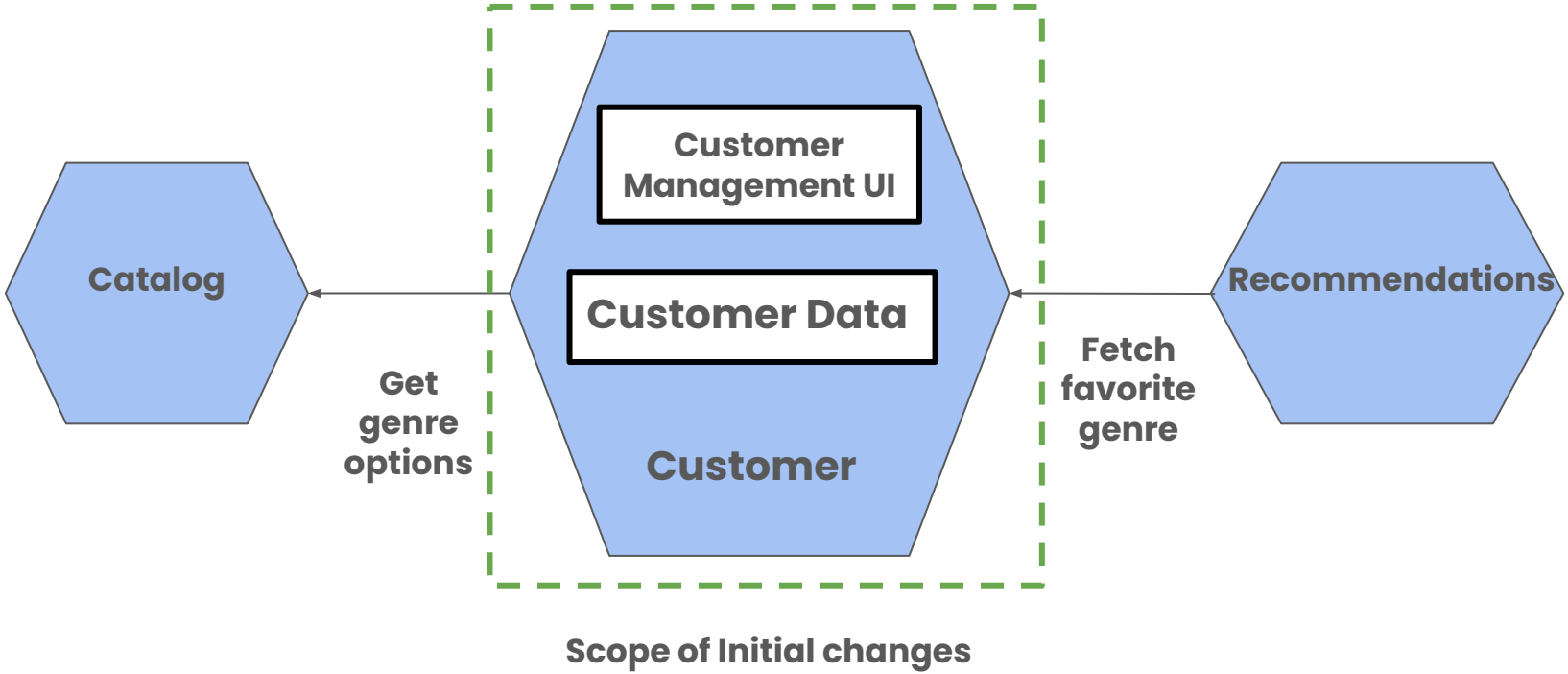


Fig 03 -A dedicated Customer service may make it much easier to record the favorite musical genre of a customer

In such a situation, our Customer service encapsulates a thin slice of each of the three tiers

it has a bit of UI, a bit of application logic, and a bit of data storage—but these layers are all encapsulated in the single service.

Our business domain becomes the primary force driving our system architecture, hopefully making it easier to make changes, and making it easier for us to organize our teams around our business domain.

1.4 Own Their Own Data

Microservices should not share databases !

If one service wants to access data held by another service, then it should go and ask that service for the data it needs.

This gives the service the ability to decide what is shared and what is hidden.

It also allows the service to map from internal implementation details, to a more stable public contract, ensuring stable service interfaces.

1.5 Advantages of Microservices

Independent deployment open up new models for improving robustness of systems, and allows you to mix and match technology.

As services can be worked on in parallel, you can bring more developers to bear on a problem without them getting in each other's way

It can also be easier for those developers to understand their part of the system

Our business domain becomes the primary force driving our system architecture, hopefully making it easier to make changes, and making it easier for us to organize our teams around our business domain.

However, it’s important to note that none of these advantages come for free !

There are many ways you can approach system decomposition, and fundamentally what you are trying to achieve will drive this decomposition in different directions

1.6 What Problems Do They Create?

The main challenges in all of this, is the way in which these computers talk to each other: networks.

Communication between computers over networks is not instantaneous (this apparently has something to do with physics).

which can make system behavior unpredictable

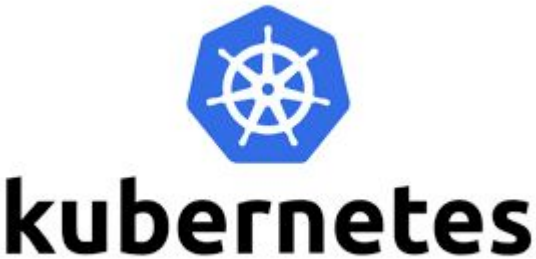
These challenges make activities that are relatively simple with a single process monolith, like transactions, much more difficult.

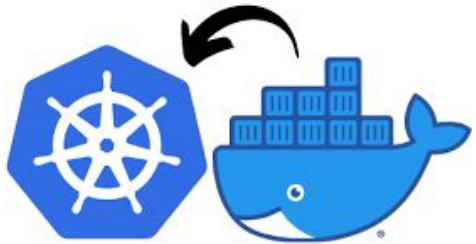
So difficult, in fact, that as your system grows in complexity

The services you might be talking to could go offline for whatever reason or otherwise start behaving oddly.

And then, of course, we have a huge wealth of new microservice-friendly technology to take into account

New technology that, if used badly, can help you make mistakes much faster and in more interesting, **expensive** ways.





Microservices buy you options.

— James Lewis

1.7 User Interfaces

UI as a monolithic blob can be a big mistake

If we want an architecture that makes it easier for us to more rapidly deploy new features, we should consider breaking apart our user interfaces too

1.8 Technology

It can be all too tempting to grab a whole load of new technology to go along with your shiny new microservice architecture

As long as your services can communicate with each other via a network, everything else is up for grabs.

Making use of a technology stack you are familiar with, and then consider whether changing your existing technology may help address problems as you find them.

1.9 Size

How big should a microservice be ?

How do you measure size ? Lines of code?

When you are first starting out, it’s much more important that you focus on two key things

how many microservices can you handle? (As you have more services, the complexity of your system will increase)

how do you define microservice boundaries to get the most out of them, without everything becoming a horribly coupled mess ?

1.10 Ownership

With microservices modeled around a business domain, we see alignment between our IT artifacts and our business domain.

This idea resonates well when we consider the shift toward technology companies breaking down the divides between “The Business” and “IT.” In traditional IT organizations

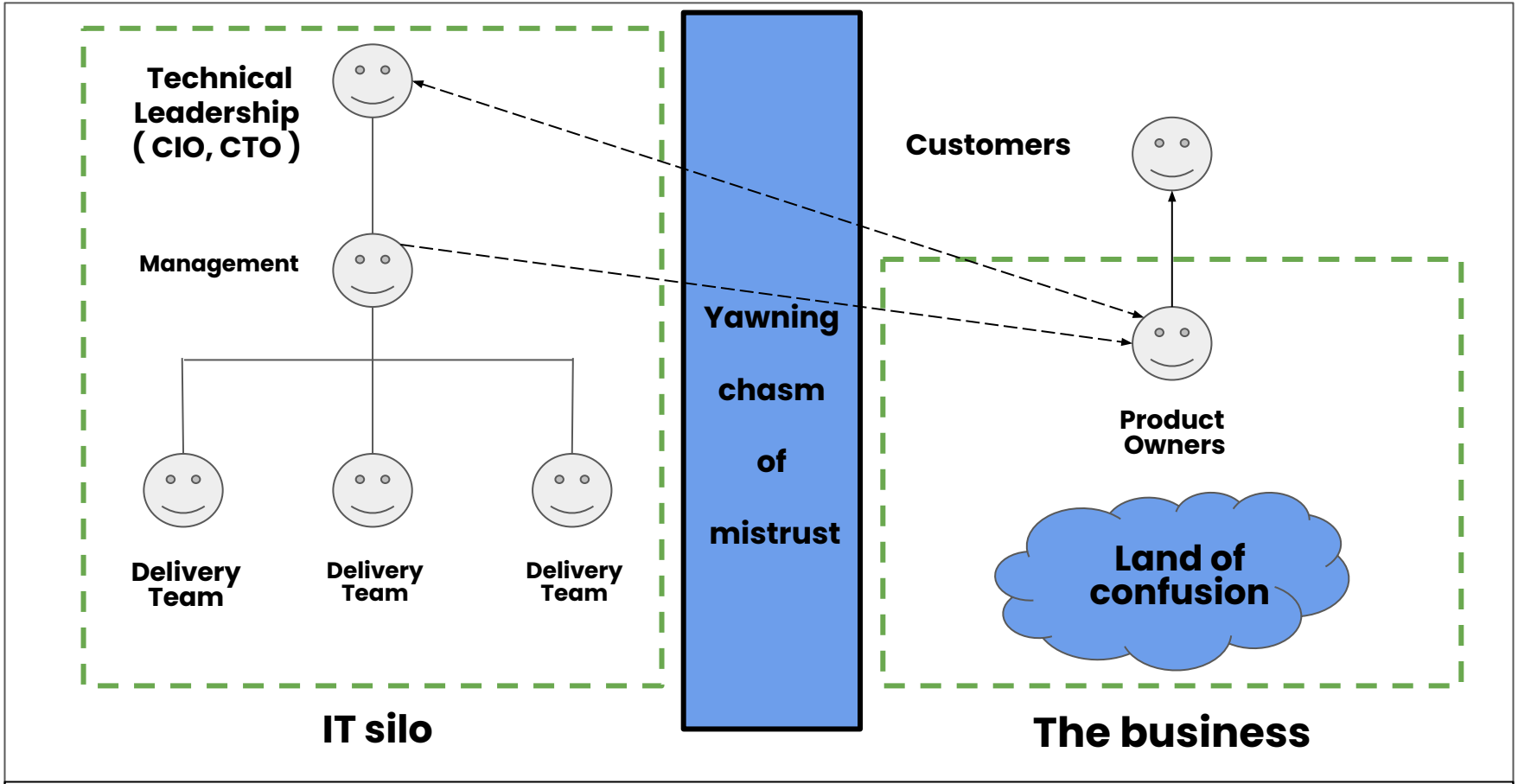


Fig 04 -An organizational view of the traditional IT/business divide

Instead, we're seeing true technology organizations totally combine these previous disparate organizational silos, as we see in Figure 1-5.

Product owners now work directly as part delivery teams, with these teams being aligned around customer facing product lines, rather than around arbitrary technical groupings.

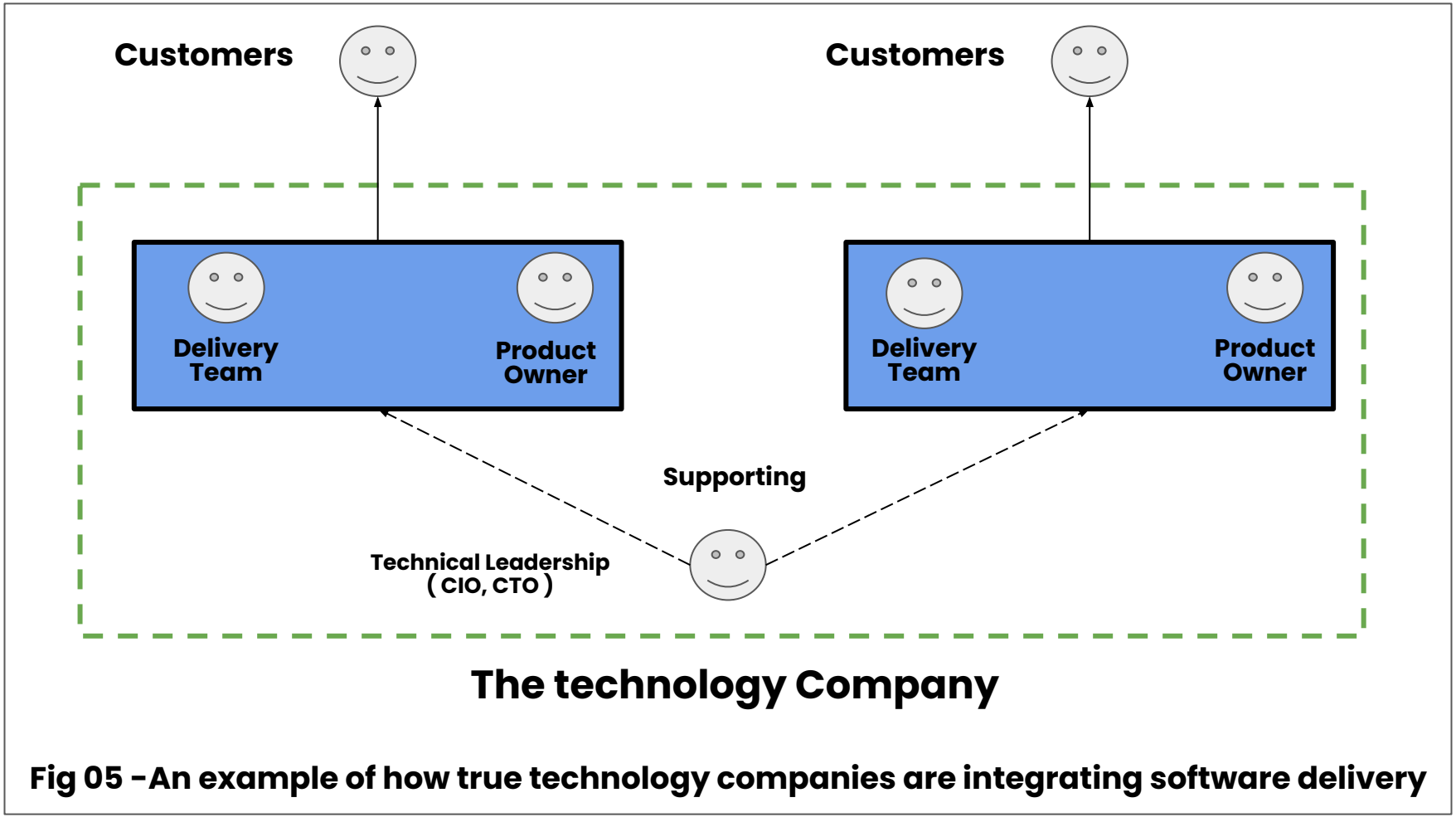


Fig 05 -An example of how true technology companies are integrating software delivery

While not all organizations have made this shift, microservice architectures make this change much easier.

It becomes easier to clearly assign ownership to these product-oriented delivery teams.

Reducing services that are shared across multiple teams is key to minimizing delivery contention

2. The Monolith

Unit Of Deployment

When all functionality in a system had to be deployed together, we consider it a monolith.

There are at least three types of monolithic systems that fit the bill :

- **Single-process system**
- **The distributed monolith**
- **Third-party black-box systems**

2.1 The Single Process Monolith

A system in which all of the code is deployed as a single process as in **Figure 6**

these single-process systems can be simple distributed systems in their own right, as they nearly always end up reading data from or storing data into a database.

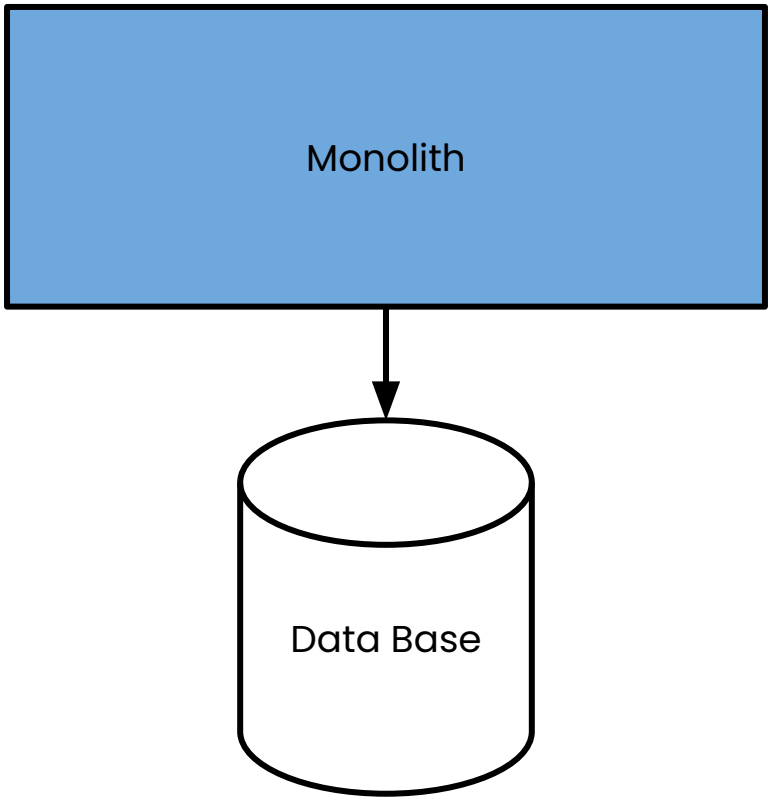


Fig 06 - A single-process monolith: all code is packaged into a single process

2.1.1 The modular monolith

A subset of the single process monolith

The modular monolith is a variation: the single process consists of separate modules, each of which **can be worked on independently**, but which still need to be combined for deployment, as shown in **Figure 7**

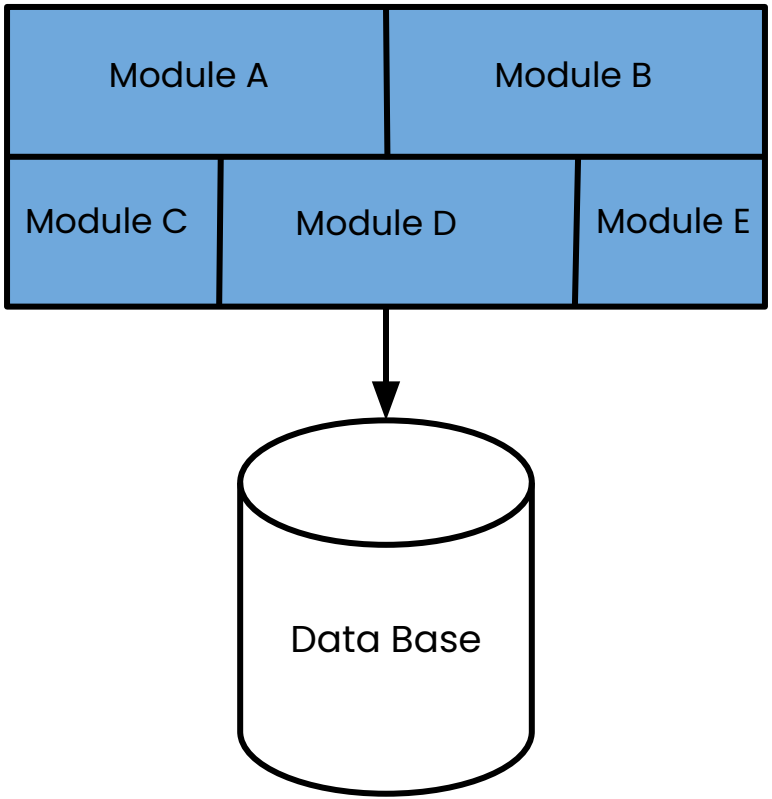


Fig 07 -A modular monolith : the code inside the process is broken down into modules

For many organizations, the modular monolith can be an excellent choice.

If the module boundaries are well defined, it can allow for a high degree of parallel working

Shopify is a great example of an organization that has used this technique as an alternative to microservice decomposition, and it seems to work really well for that company.

! One of the challenges of a modular monolith is that the database tends to lack the decomposition we find in the code level .

! Many teams attempt to push the idea of the modular monolith further, Having the database decomposed along the same lines as the modules, as shown in Figure 8

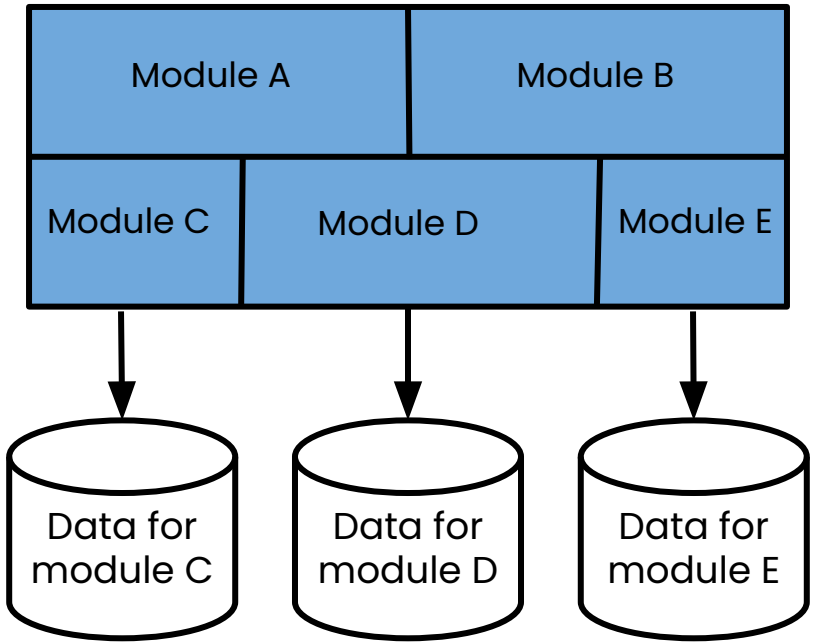


Fig 08 -A modular monolith with a decomposed database

2.2 The Distributed Monolith

A distributed system is one in which the failure of a computer you didn’t even know existed can render your own computer unusable.

— James Lewis

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

! **Distributed monoliths have all the disadvantages of a distributed system, and the disadvantages of a single-process monolith, without having enough upsides of either**

2.3 Third-Party Black-Box Systems

We can also consider some third-party software as monoliths that we may want to “decompose” as part of a migration effort.

These might include things like payroll systems, CRM systems, and HR systems. (software developed by other people)

2.4 Challenges of Monoliths

The monolith, be it a single-process monolith or a distributed monolith, is often more vulnerable to the perils of coupling

As you have more and more people working in the same place, they get in each other's way (delivery contention) .

2.5 Advantages of Monoliths

Simpler deployment topology can avoid many of the pitfalls associated with distributed systems.

It can result in much simpler developer workflows; and monitoring, troubleshooting, and activities like end-to-end testing can be greatly simplified as well.

Monoliths can also simplify code reuse within the monolith itself

Conclusion

Thanks

Do you have any questions ?

Part 2