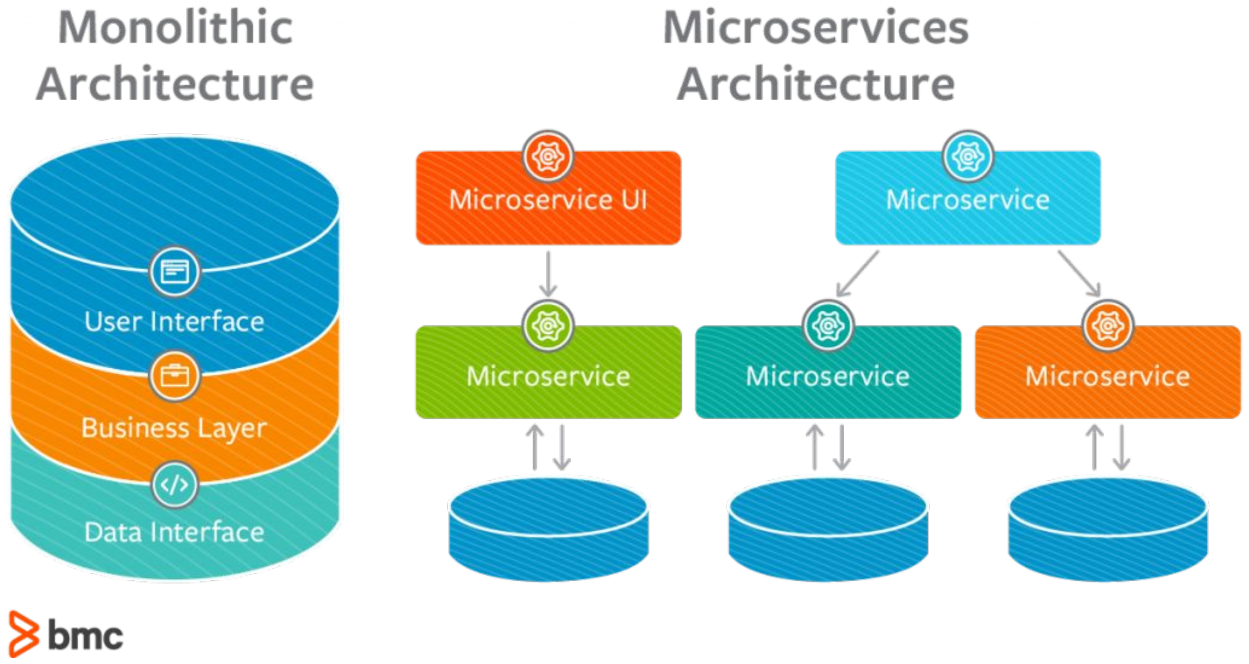
Scalable Web Systems

Introduction to Microservices

What Are Microservices?

Microservices are small, autonomous services that work together.



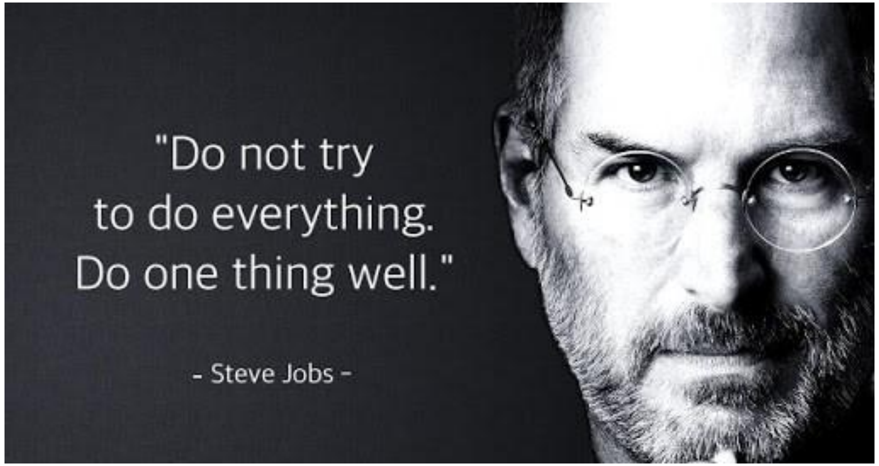
Small, and Focused on Doing One Thing Well

Codebases grow as we write code to add new features.

Over time, it can be difficult to know where a change needs to be made because the codebase is so large.

Despite a drive for clear, modular monolithic codebases, all too often these arbitrary in-process boundaries break down.

Code related to similar functions starts to become spread all over, making fixing bugs or implementations more difficult.

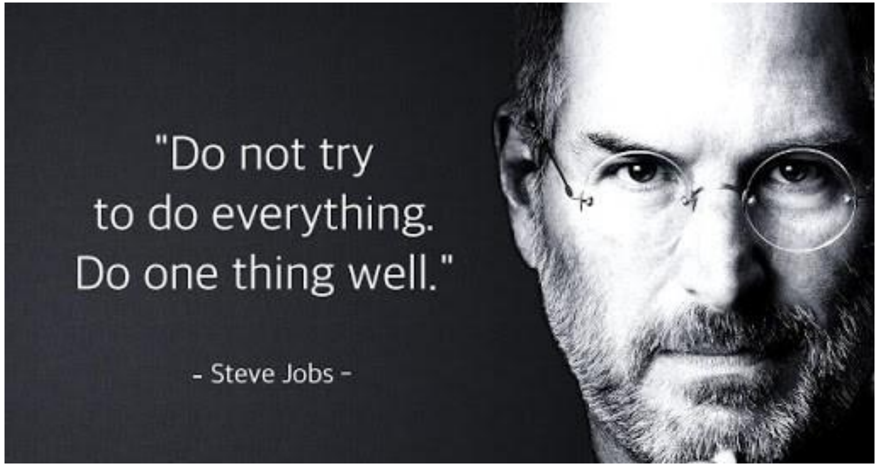


Monolithic Systems

Within a monolithic system, we fight against these forces by trying to ensure our code is more cohesive, often by creating abstractions or modules.

Cohesion, the drive to have related code grouped together, is an important concept when we think about microservices.

This is reinforced by Robert C. Martin's definition of the Single Responsibility Principle, which states "Gather together those things that change for the same reason, and separate those things that change for different reasons."



Interesting side note: five of Martin's principles have become known collectively as the "SOLID principles". Though he invented most of the principles he promotes, the Liskov substitution principle was devised by Barbara Liskov.

What is interesting about this is that Barbara was the PhD advisor of Elliot Moss (CICS) and Elliot was my PhD advisor. So, the world is a small place and clearly the influence of interest continues...

Autonomous

A microservice is a separate entity, running on the same machine or distributed across many machines.

All communication between the services themselves are via network calls, to enforce separation between the services and avoid the perils of tight coupling.

These services need to be able to change independently of each other, and be deployed by themselves without requiring consumers to change.

We need to think about what our services should expose, and what they should allow to be hidden.

If there is too much sharing, our consuming services become coupled to our internal representations. This decreases our autonomy, as it requires additional coordination with consumers when making changes.

Microservices are independent and autonomous communicating over a well defined API.

Key Benefits

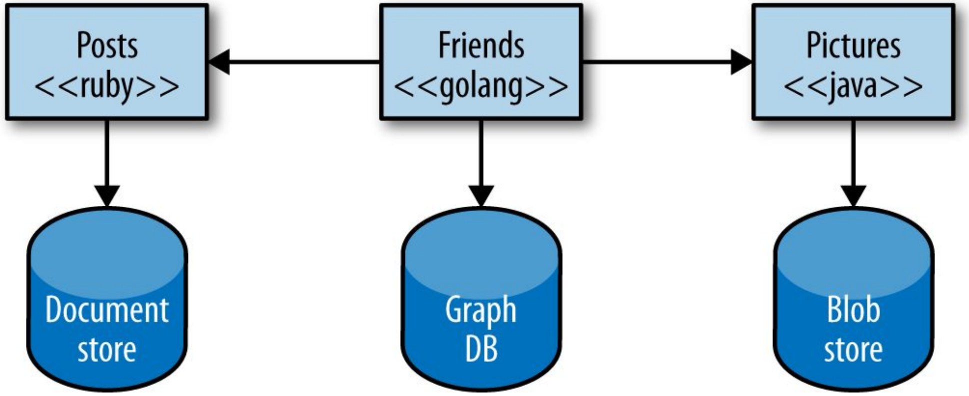
The benefits of microservices are many and varied.

Many of these benefits can be laid at the door of any distributed system.

Microservices, however, tend to achieve these benefits to a greater degree primarily due to how far they take the concepts behind distributed systems and service-oriented architectures.

Key Benefits - Technology Heterogeneity

With a system composed of multiple, collaborating services, we can decide to use different technologies inside each one. This allows us to pick the right tool for each job, rather than having to select a more standardized, one-size-fits-all approach that often ends up being the lowest common denominator.



Key Benefits - Resilience

A key concept in resilience engineering is the bulkhead.

If one component of a system fails, but that failure doesn’t cascade, you can isolate the problem and the rest of the system can carry on working.

Service boundaries become your obvious bulkheads.

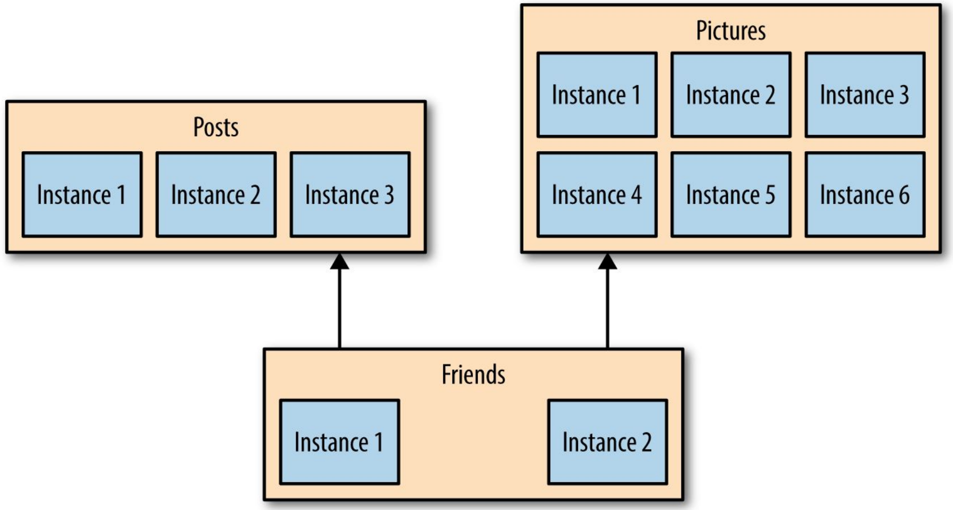
In a monolithic service, if the service fails, everything stops working. With a monolithic system, we can run on multiple machines to reduce our chance of failure, but with microservices, we can build systems that handle the total failure of services and degrade functionality accordingly.

Key Benefits - Scaling

With a large, monolithic service, we have to scale everything together.

One small part of our overall system is constrained in performance, but if that behavior is locked up in a giant monolithic application, we have to handle scaling everything as a piece.

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



Key Benefits - Ease of Deployment

A one-line change to a million-line-long monolithic application requires the whole application to be deployed in order to release the change.

That could be a large-impact, high-risk deployment. In practice, large-impact, high-risk deployments end up happening infrequently due to understandable fear.

Unfortunately, this means that our changes build up and build up between releases, until the new version of our application hitting production has masses of changes.

And the bigger the delta between releases, the higher the risk that we’ll get something wrong!

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

Key Benefits - Organizational Alignment

It is common to experience problems associated with large teams and large codebases.

<u>These problems can be exacerbated when the team is distributed.</u>

Smaller teams working on smaller codebases tend to be more productive.

Microservices allow us to **better align an architecture** to an organization, helping us minimize the number of people working on any one codebase to hit the sweet spot of team size and productivity.

Key Benefits - 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.

Gone is the time when we could think narrowly about either our desktop website or mobile application. Now we need to think of the myriad ways that we might want to weave together capabilities for the Web, native application, mobile web, tablet app, or wearable device.

Key Benefits - Optimizing for Replaceability

In medium-size or bigger organizations, chances are there are some big, nasty legacy system sitting in the corner. **The one no one wants to touch!**

The one that is vital to how the company runs, but that happens to be written in some odd Fortran variant and runs only on hardware that reached end of life 25 years ago. Why hasn't it been replaced?

It's too big and risky a job!

Microservices make it easier to replace or update parts of a larger system.