PRACTICE

The Hidden Dividends of Microservices

By Tom Killalea

Communications of the ACM, Vol. 59 No. 8, Pages 42-45
10.1145/2948985
Comments

ACM RESOURCES

Microservices are an approach to building distributed systems in which services are exposed only through hardened APIs; the services themselves have a high degree of internal cohesion around a specific and well-bounded context or area of responsibility, and the coupling between them is loose. Such services are typically simple, yet they can be composed into very rich and elaborate applications. The effort required to adopt a microservices-based approach is considerable, particularly in cases that involve migration from more monolithic architectures. The explicit benefits of microservices are well known and numerous, however, and can include increased agility, resilience, scalability, and developer productivity. This article identifies some of the hidden dividends of microservices that implementers should make a conscious effort to reap.

The most fundamental of the benefits driving the momentum behind microservices is the clear separation of concerns, focusing the attention of each service upon some well-defined aspect of the overall application. These services can be composed in novel ways with loose coupling between the services, and they can be deployed independently. Many implementers are drawn by the allure of being able to make changes more frequently and with less risk of negative impact. Robert C. Martin described the *single responsibility principle*: "Gather together those things that change for the same reason. Separate those things that change for different reasons."[5] The clear separation of concerns, minimal coupling across domains of concern, and the potential for a higher rate of change lead to increased business agility and engineering velocity.

Martin Fowler argues the adoption of continuous delivery and the treatment of infrastructure as code are more important than moving to microservices, and some implementers adopt these practices on the way to implementing microservices, with positive effects on resilience, agility, and productivity. An additional key benefit of microservices is they can enable owners of different parts of an overall architecture to make very different decisions with respect to the hard problems of building large-scale distributed systems in the areas of persistence mechanism choices, consistency, and concurrency. This gives service owners greater autonomy, can lead to faster adoption of new technologies, and can allow them to pursue custom approaches that might be optimal for only a few or even for just one service.

Back to Top

The Dividends

While difficult to implement, a microservices-based approach can pay dividends to the organization that takes the trouble, though some of the benefits are not always obvious. What follows is a description of a few of the less obvious ones that may make the adoption of microservices worth the effort.

Back to Top

Dividend #1: Permissionless Innovation

Permissionless innovation is about "the ability of others to create new things on top of the communications constructs that we create,"[1] as put forth by Jari Arkko, chair of the Internet Engineering Task Force (IETF). When enabled, it can lead to innovations by consumers of a set of interfaces that the designers of those interfaces might find surprising and even bewildering. It contrasts with approaches where *gatekeepers* (a euphemism for *blockers*) have to be consulted before an integration can be considered.

To determine whether permissionless innovation has been unleashed to the degree possible, a simple test is to look at the prevalence of meetings *between* teams (as distinct from *within* teams). Cross-team meetings suggest coordination, coupling, and problems with the granularity or functionality of service interfaces. Engineers do not seek out meetings if they can avoid them; such meetings could mean that a service's APIs are not all that is needed to integrate. An organization that has embraced permissionless innovation should have a high rate of experimentation and a low rate of cross-team meetings.

Back to Top

Dividend #2: Enable Failure

It should come as no surprise to hear that in computer science, we still don't know how to build complex systems that work reliably,[6] and the unreliability of systems increases with size and complexity. While opinions differ as to whether microservices allow a reduction in overall complexity, it's worth embracing the notion that microservices will typically increase the number of failures. Further, failures across service boundaries will be more difficult to troubleshoot since external call stacks are inherently more fragile than internal ones, and the debugging task is limited by poorer tooling and by more challenging ad hoc analysis characteristics. This tweet by @Honest_Update can sometimes feel uncomfortably accurate: "We replaced our monolith with micro services so that every outage could be more like a murder mystery."[4]

Designing for the inevitability and indeed the routineness of failure can lead to healthy conversations about state persistence, resilience, dependency management, shared fate, and graceful degradation. Such conversations should lead to a reduction of the blast radius of any given failure by leveraging techniques such as caching, metering, traffic engineering, throttling, load shedding, and backoff. In a mature microservices-based architecture, failure of individual services should be expected, whereas the cascading failure of all services should be impossible.

Back to Top

Dividend #3: Disrupt Trust

In small companies or in small code bases, some engineers may have a strong sense of trust in what is being deployed because they look over every shoulder and review every commit. As team size and aggregate velocity increase, "Dunbar's number" takes effect, leading to such trust becoming strained. As defined by British anthropologist Robin Dunbar, this is the maximum number of individuals with whom one can maintain social relationships by personal contact.

A move to microservices can force this expectation of trust to surface and be confronted. The boundary between one service and another becomes a set of APIs. The consumer gives up influence over the design of what lies behind those APIs, how that design evolves, and how its data persists, in return for a set of SLAs (service-level agreements) governing the stability of the APIs and their runtime characteristics. Trust can be replaced with a combination of autonomy and accountability.

Microservices encourage the "you build it, you own it" model.

As stated by Melvin Conway, who defined what is now known as Conway's law: "Any organization that designs a system will inevitably produce a design whose structure is a copy of the organization's communication structure."[2]

Microservices can provide an effective model for evolving organizations that scale far beyond the limits of personal contact.

Back to Top

Dividend #4: You Build It, You Own It

Microservices encourage the "you build it, you own it" model. Amazon CTO Werner Vogels described this model in a 2006 conversation with Jim Gray that appeared in *ACM Queue*: "Each service has a team associated with it, and that team is completely responsible for the service—from scoping out the functionality, to architecting it, to building it, and operating it. You build it, you run it. This brings developers into contact with the day-to-day operation of their software. It also brings them into day-to-day contact with the customer. The customer feedback loop is essential for improving the quality of the service."[3]

In the decade since that conversation, as more software engineers have followed this model and taken on responsibility for the operation as well as the development of microservices, they have driven broad adoption of a number of practices that enable greater automation and that lower operational overhead. Among these are continuous deployment, virtualized or containerized capacity, automated elasticity, and a variety of self-healing techniques.

Back to Top

Dividend #5: Accelerate Deprecations

In a monolith, it's difficult to deprecate anything safely. With microservices, it's easy to get a clear view of a service's call volume, to stand up different and potentially competing versions of a service, or to build a new service that shares nothing with the old service other than backward compatibility with those interfaces that consumers care about the most.

In a world of permissionless innovation, services can and should routinely come and go. It's worth investing some effort to make it easier to deprecate services that have not meaningfully caught on. One approach to doing this is to have a sufficiently high degree of competition for resources so that any resource-constrained team that is responsible for a languishing service is drawn to spending most of their time on other services that matter more to customers. As this occurs, responsibility for the unsuccessful service should be transferred to the consumer who cares about it the most. This team may rightfully consider themselves to have been left "holding the can," although the deprecation decision also passes into their hands. Other teams that wish not to be left holding the can have an added incentive to migrate or terminate their dependencies. This may sound brutal, but it's an important part of "failing fast."

Back to Top

Dividend #6: End Centralized Metadata

In Amazon's early years, a small number of relational databases were used for all of the company's critical transactional data. In the interest of data integrity and performance, any proposed schema change had to be reviewed and approved by the DB Cabal, a gatekeeping group of well-meaning enterprise modelers, database administrators, and software engineers. With microservices, consumers should not know or care about how data persists behind a set of APIs on which they depend, and indeed it should be possible to swap out one persistence mechanism for another without consumers noticing or needing to be notified.

Back to Top

Dividend #7: Concentrate The Pain

A move to microservices should enable an organization to take on very different approaches to the governance expectations that it has of different services. This will start with a consistent companywide model for data classification and with the classification of the criticality of the integrity of different business processes. This will typically lead to threat modeling for the services that handle the most important data and processes, and the implementation of the controls necessary to serve the company's security and compliance needs. As microservices proliferate, it can be possible to ensure the most severe burden of compliance is concentrated in a very small number of services, releasing the remaining services to have a higher rate of innovation, comparatively unburdened by such concerns.

Back to Top

Dividend #8: Test Differently

Engineering teams often view the move to microservices as an opportunity to think differently about testing. Frequently, they will start thinking about how to test earlier in the design phase, before they start to build their service. A clearer definition of ownership and scope can provide an incentive to achieve greater coverage. As stated by Yelp in setting forth its service principles, "Your interface is the most vital component to test. Your interface tests will tell you what your client actually sees, while your remaining tests will inform you on how to ensure your clients see those results."[7]

The adoption of practices such as continuous deployment, smoke tests, and phased deployment can lead to tests with higher fidelity and lower time-to-repair when a problem is discovered in production. The effectiveness of a set of tests can be measured less by their rate of problem detection and more by the rate of change that they enable.

Back to Top

Warning Signs

The following indicators are helpful in determining that the journey to microservices is incomplete. You are probably not doing microservices if:

- Different services do coordinated deployments.
- You ship client libraries.
- A change in one service has unexpected consequences or requires a change in other services.
- Services share a persistence store.
- You cannot change your service's persistence tier without anyone caring.
- Engineers need intimate knowledge of the designs and schemas of other teams' services.
- You have compliance controls that apply uniformly to all services.
- Your infrastructure isn't programmable.
- You can't do one-click deployments and rollbacks.

Back to Top

Conclusion

Microservices aren't for every company, and the journey isn't easy. At times the discussion about their adoption has been effusive, focusing on autonomy, agility, resilience, and developer productivity. The benefits don't end there, however, and to make the journey worthwhile, it's important to reap the additional dividends.

Related articles
on queue.acm.org

A Conversation with Werner Vogels
http://queue.acm.org/detail.cfm?id=1142065

The Verification of a Distributed System
Caitie McCaffrey
http://queue.acm.org/detail.cfm?id=2889274

There's Just No Getting around It: You're Building a Distributed System
Mark Cavage
http://queue.acm.org/detail.cfm?id=2482856

Back to Top

References

1. Arkko, J. Permissionless innovation. IETF; https://www.ietf.org/blog/2013/05/permissionless-innovation/.

2. Conway, M.E. How do committees invent? *Datamation Magazine* (1968); http://www.melconway.com/Home/Committees_Paper.html.

3. Gray. J. A conversation with Werner Vogels. *ACM Queue 4*, 4 (2006); http://queue.acm.org/detail.cfm?id=1142065.

4. Honest Status Page. @honest_update, 2015; https://twitter.com/honest_update/status/651897353889259520.

5. Martin, R.C. The single responsibility principle; http://blog.8thlight.com/uncle-bob/2014/05/08/SingleReponsibilityPrinciple.html.

6. Perera, D. The crypto warrior. Politico; http://www.politico.com/agenda/story/2015/12/crypto-war-cybersecurity-encryption-000334.

7. Yelp service principles; https://github.com/Yelp/service-principles.

Back to Top

Author

Tom Killalea was with Amazon for 16 years and now consults and sits on several company boards, including those of Capital One, ORRECO, and MongoDB.

No entries found

Comment on this article

Signed comments submitted to this site are moderated and will appear if they are relevant to the topic and not abusive. Your comment will appear with your username if published. View our policy on comments

☐ Notify me via email when subsequent user comments are published with this article.

SUBMIT FOR REVIEW