



# The Monolith Strikes Back: Why Istio Migrated From Microservices to a Monolithic Architecture

Nabor C. Mendonça, Craig Box, Costin Manolache, and Louis Ryan

## From the Editors

To break or not to break the monolith? This question of modularity is often answered positively by microservice architects. In this issue's "Insights" department, authors argue why the latest version of the popular Istio middleware—which provides the fabric of many contemporary microservice architectures—has not been released as a set of independently deployable components, but as a single unit that is tested, deployed, and scaled as a whole. We learn that modularity affects the entire software lifecycle: it is not only a development and maintenance concern but also a deployment-time decision. The operator experience matters, just like user experience and developer experience.—*Cesare Pautasso and Olaf Zimmermann*

**MICROSERVICES HAVE BECOME** an essential architectural enabler in cloud-based software development and delivery.<sup>1</sup> Many organizations, from large tech companies (e.g., Google, Amazon, Netflix) to small start-ups, have adopted microservices as a best practice.<sup>2</sup> This has served them well in some cases and, as you will see here, not so well in others.

Briefly, microservices constitute a service-oriented architectural style in

which server-side applications are constructed by combining many single-purpose, low-footprint distributed services.<sup>3</sup> This significantly impacts the application's agility because each microservice becomes an independent unit of development, deployment, versioning, scaling, and management.<sup>1</sup> The other touted benefits of microservices include reduced testing effort, better functional composition, environmental isolation, and development team autonomy.<sup>4</sup> The opposite is a more monolithic architecture, where several discrete functions are composed into a

single unit that is tested, deployed, and scaled as a whole.

Despite their benefits, the adoption of microservices poses several technical and organizational challenges, for example, high operational complexity, increased technological diversity, and the need for better coordination among teams.<sup>1,3,4</sup> Early evidence on the practical gains and pains of microservices have started to emerge in academic publications<sup>5–7</sup> and industry forums.<sup>8</sup> However, there are still relatively few industrial reports on microservice projects in which the pains are assessed to outweigh the

Digital Object Identifier 10.1109/MS.2021.3080335  
Date of current version: 20 August 2021

gains.<sup>9,10</sup> In these situations, project developers may have to face the nontrivial decision of abandoning microservices in favor of a monolithic architecture.

This article reports on the design decisions, tradeoffs, and lessons learned from one of those projects—the Istio open source service mesh.<sup>11</sup> Istio adopted a microservice architecture in its control plane early on. However, less than three years after Istio was first released, the control plane microservices were consolidated into a monolith. You can find more information on Istio’s microservices-to-monolith journey in the project’s online blog.<sup>12</sup>

## Istio

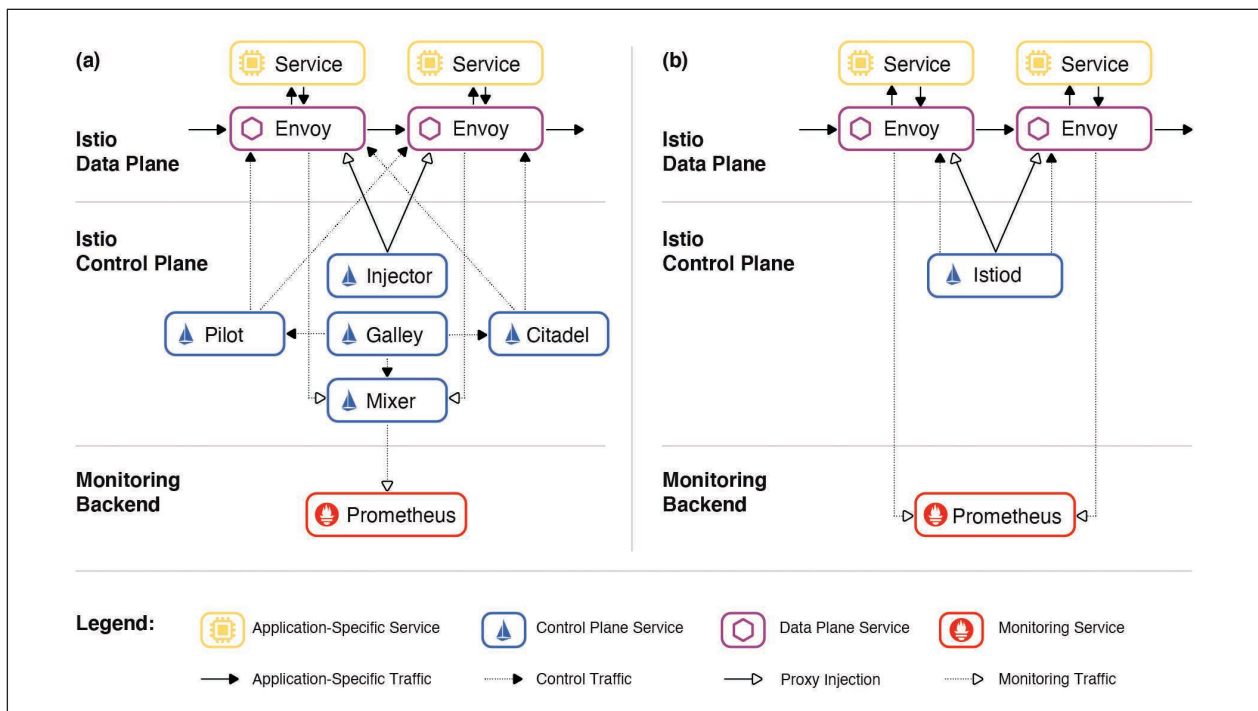
Istio was released to the public in 2017 as a collaboration among Google, IBM, and Lyft. The system provides a uniform way to connect, secure, manage, and monitor cloud-native applications

deployed in Kubernetes, an open source container orchestration platform that has become the de facto standard for managing containerized applications.<sup>13</sup> Istio is logically split into a data plane and a control plane from an architectural perspective, similar to other service mesh solutions.<sup>14</sup> The data plane is composed of a set of Envoy proxies<sup>15</sup> deployed as application service sidecars. These proxies mediate and control all network communication between application containers. They also collect and report telemetry on all mesh traffic. The control plane, in turn, manages and configures the data plane proxies to monitor and route application traffic according to user-provided traffic management rules. A monitoring back end running systems such as Prometheus,<sup>16</sup> a metrics server, and Jaeger,<sup>17</sup> a distributed tracer, is also typically deployed alongside the control plane to collect and store a variety of mesh- and

application-specific metrics (e.g., latency and error rate per service).

Istio’s original control plane architecture was composed of five independently deployable microservices [see Figure 1(a)]:

- Pilot—the core data plane configuration service, which provides service discovery for the Envoy proxies, traffic management capabilities for intelligent routing (e.g., A/B tests, canary rollouts, etc.), and resiliency (timeouts, retries, circuit breakers, etc.)
- Citadel—service responsible for enforcing strong service-to-service and end-user authentication with built-in identity and credential management
- Galley—service responsible for configuration validation, ingestion, processing, and distribution of configuration to the other services



**FIGURE 1.** The architectural evolution of Istio’s control plane: (a) microservices and (b) monolith.

- Injector—service responsible for auto-injecting the data plane proxies in the application containers and setting up bootstrap
- Mixer—service responsible for enforcing access control and usage policies across the service mesh; it also collects telemetry data from the Envoy proxies and other services and reports this data to the monitoring back end.

### Microservice Burdens

The Istio team thought microservices were the right architectural approach from the very beginning, as many of the authors had developed and operated a similar system within Google and ported what they knew. Thus, they initially built the control plane as a set of microservices, like a modern, cloud-native application. As Istio adoption increased and the team got feedback from customers, they realized that many of the benefits typically associated with microservices, that is, independent rollout, independent scale, and security isolation, did not apply to the Istio control plane. In most Istio installations, all control plane components are installed and operated by a single team or individual. This means that most Istio components are not delivered and managed independently from one another. Consequently, Istio operators were paying the price of the greater operational complexity inherent to a microservice architecture without benefiting from it. While the teams operating Istio may have the skills to offset these challenges, they did not perceive any value in performing that work.

### From Microservices to a Monolith

Once the team established that many of the expected benefits of microservices did not apply to the Istio control plane, they decided to consolidate

them into a single binary: `istiod` [see Figure 1(b)]. This unified service combines the use cases of Pilot, Citadel, Galley, and Injector. The Mixer component was removed in a concurrent project. The Envoy proxies now directly enforce policy and report telemetry to the monitoring back end, which had previously been performed using the Mixer as a central intermediary. The motivation for this change was similar to that of `istiod` in that a design pattern identified as necessary at massive operational scale was not justified against the maintenance and performance overheads for typical users.

Compared to Istio's previous microservice architecture, this new monolithic control plane offers several benefits. Here are some examples. First, installation is simplified, as fewer Kubernetes deployments and associated configurations are required. Second, configuration becomes more straightforward, as many of the configuration options that Istio had before were ways to orchestrate the control plane services and therefore are no longer needed. Third, debugging becomes less of a burden, as having fewer services means less cross-service environmental debugging. Scalability is also simplified, as there is now only one service to scale. Finally, the time to start, upgrade, and remove Istio goes down, as these no longer require a complicated dance of version dependencies and start-up orders.

### Monolith Tradeoffs

Generally, when a team adopts microservices and their inherent complexity, they look for improvements in other areas to justify the tradeoffs. After looking at the Istio control plane through that lens, the team concluded that the value of microservices was not greater than the cost of having to orchestrate them during setup and operation. For instance, at the

time of writing, computer resource costs in the Istio control plane are dominated by a single feature: serving the Envoy dynamic resource discovery (i.e., `xDS`) APIs that program the data plane. Every other feature has a marginal cost. This means there is very little value to having those features in separately scalable microservices. Also, full security isolation is not attainable in the control plane as well. This is because multiple control plane services hold nearly equivalent roles in securing behavior of the proxy and are installed by default into the same Kubernetes namespace. Thus, exploiting any of these services would cause near equal damage. Therefore, moving the Istio control plane to a monolithic architecture turned out to be the right architectural decision.

We should note that internally Istio still maintains the logical separation between some of its original control plane components and that each capability is exposed as a discrete API. This still enables functions to be swapped and combined with other implementations. This feature can be particularly useful in some advanced use cases, such as in multicloud deployments, where `istiod` can be deployed as a single-purpose service such as “injection,” “certification provider,” or “validation.” This design decision maintains many of the benefits of microservices to more seasoned Istio operators without the downsides to those only interested in Istio's most common use cases.

### Lessons Learned

According to Sam Newman, author of a recent book on microservice migration patterns,<sup>18</sup> developers should contemplate these three questions before adopting a microservice architecture: What are you hoping to achieve? Have you considered alternatives to using microservices? How will you know if the transition is working? Here we

report on the main lessons learned from the Istio team’s decision to consolidate their microservices from the perspectives of these three questions. We also discuss whether this decision aligns with industry best practices.

**What Was the Team Hoping to Achieve With Microservices?**

Initially, the Istio team expected to benefit from microservices’ well-known advantages, such as independent rollout and independent scale. However, as they regularly talked to customers and teams running Istio in the real world, they were told that none of these were the case for the Istio control plane. The first lesson learned then is that good teams should look back upon their design choices and, with the benefit of hindsight, revisit them.<sup>19</sup>

**Had the Team Considered Alternatives to Using Microservices?**

In retrospect, the team admits that they favored microservices because they confused their operational experience for that of the end-user. For Istio, they are not the operator. Thus, the need for the control plane services to communicate securely and be observable provided opportunities for Istio to “eat its own dog food.” To put it another way, because Istio was targeted explicitly at securing and observing individual microservices, the team thought implementing the Istio control plane as secure and observable microservices was a natural choice. As we have seen, this wasn’t necessarily the case. In summary, the team believes honesty and courage are required to undo a previous design, which is

facilitated when done in an open and constructive “error culture.”

**How Did the Team Know Microservices Were Not Working?**

At first, the Istio team did not pay much attention to the burden of managing Istio’s control plane as independently deployable microservices. However, after the team started receiving feedback from other Istio users, they soon realized that microservices were not as beneficial as they initially had thought. The main reason was that all control plane services were being deployed and used together and shared the same administrative and security domains. Thus, moving the Istio control plane to a monolithic architecture was a welcome decision, as it greatly reduced Istio’s operational complexity. While this decision may seem like a significant change, the Istio team is confident that it has paid off and verifiably made the lives of Istio users better. Thus, the team thinks this change shows a willingness to change based on user feedback and a continued focus on simplification for all users.

**How Well Does the Decision to Consolidate the Control Plane Microservices Align With Industry Best Practices?**


To answer this question, we again resort to Sam Newman’s microservice recommendations, in particular, to his list of four situations in which microservices might be a bad idea.<sup>16</sup> Table 1 briefly describes those four situations and whether they apply to Istio. As you can see, all of them apply to Istio, either partially or in full. Notably, using microservices in customer-installed and -managed software was the clear reason that microservices were a bad

**Table 1. Sam Newman’s recommendations on when not to use microservices and whether they apply to Istio.**

Situation	Why Microservices Are Bad	Does It Apply to Istio?
Unclear domain	Getting service boundaries wrong can be expensive.	In part. From the perspective of fault and security isolation, it can be argued that splitting the Istio control plane into multiple independent services was an unnecessary decision.
Start-ups	A start-up needs to focus all its attention on finding the right fit for its product. Microservices primarily solve the sorts of problems start-ups have once they’ve found that fit with their customer base.	In part. While Istio was initially designed by mature organizations, it was run like a start-up and did indeed need to focus on finding the right fit. As it turned out, microservices were solving a problem Istio didn’t actually have.
Customer-installed and managed software	Microservices push a lot of complexity into the operational domain. Coping with this complexity isn’t something you can typically expect of your end customers.	Yes. Negative user feedback on the complexity of deploying and managing Istio was the main reason for the team’s decision to consolidate the control plane microservices into a single binary.
Not having a good reason!	Do not adopt microservices if you don’t have a clear idea of what exactly it is that you’re trying to achieve.	In part. Although the Istio team had a clear view of the benefits and cost of microservices, they didn’t realize right from the start that, in their case, the costs would outweigh the benefits.

idea in Istio's case. The others are partial and might not be very convincing, but this one has clear guidance relevant to readers: if you ship customer-installed software, be wary of shipping a bunch of microservices. One final lesson learned then is that although the Istio team was aware of Sam Newman's microservice recommendations, they didn't give appropriate weight to the guidance around operations. This shows the importance of sharing real-world microservice experiences with a broader audience, especially those that did not turn out as well as expected.

**A**s microservices become increasingly popular, they are more likely to be used in situations where the costs far outweigh the benefits. Istio's recent decision to migrate its control plane from microservices to a monolithic architecture is a real-world example of some of these situations. While the Istio team recognizes that microservices can work well in some systems and use cases, their unmet expectations with microservices are a timely reminder

that microservices are not, and never will be, the right solution in all cases. Microservices are a tool in a toolbox, and they work best when reflected in the organizational reality. 

### Acknowledgment

Nabor C. Mendonça was partly supported by Brazil's National Council for Scientific and Technological Development under grants 424160/2018-8 and 311344/2020-8.

### References

1. P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The journey so far and challenges ahead," *IEEE Softw.*, vol. 35, no. 3, pp. 24–35, May/June 2018. doi: 10.1109/MS.2018.2141039.
2. C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis, "Microservices in practice, part 1: Reality check and service design," *IEEE Softw.*, vol. 34, no. 1, pp. 91–98, Jan./Feb. 2017. doi: 10.1109/MS.2017.24.
3. J. Lewis and M. Fowler. "Microservices: A definition of this new architectural term." martinFowler, Mar. 2014. <https://martinfowler.com/articles/microservices.html> (accessed May 28, 2021).
4. O. Zimmermann, "Microservices tenets: Agile approach to service development and deployment," *Comput. Sci.—Res. Develop.*, vol. 32, nos. 3–4, pp. 301–310, 2017. doi: 10.1007/s00450-016-0337-0.
5. J. Soldani, D. A. Tamburri, and W. J. Van Den Heuvel, "The pains and gains of microservices: A systematic grey literature review," *J. Syst. Softw.*, vol. 146, pp. 215–223, Dec. 2018. doi: 10.1016/j.jss.2018.09.082.
6. H. Zhang, S. Li, Z. Jia, C. Zhong and C. Zhang, "Microservice architecture in reality: An industrial inquiry," *Proc. IEEE Int. Conf. Softw. Architecture (ICSA 19)*, 2019, pp. 51–60. doi: 10.1109/ICSA.2019.00014.
7. H. Knoche and W. Hasselbring, "Drivers and barriers for microservice adoption—A survey among professionals in Germany," *Enterprise Model. Inform. Syst. Architectures*, vol. 14, no. 1, pp. 1–35, 2019. doi: 10.18417/emisa.14.1.

## ABOUT THE AUTHORS



**NABOR C. MENDONÇA** is a full professor at the University of Fortaleza, Fortaleza, Ceará, CEP 60811-905, Brazil. Contact him at [nabor@unifor.br](mailto:nabor@unifor.br).



**CRAIG BOX** is a developer advocate at Google. Contact him at [craigbox@google.com](mailto:craigbox@google.com).



**COSTIN MANOLACHE** is a software engineer at Google. Contact him at [costin@google.com](mailto:costin@google.com).



**LOUIS RYAN** is a principal engineer at Google. Contact him at [lryan@google.com](mailto:lryan@google.com).



8. “Panel: Microservices—Are they still worth it?” InfoQ, Apr. 2020. <https://www.infoq.com/presentations/microservices-panel-value/> (accessed May 28, 2021).
9. A. Noonan. “Goodbye microservices: From 100s of problem children to 1 superstar.” Segment, July 2018. <https://segment.com/blog/goodbye-microservices/> (accessed May 28, 2021).
10. C. Posta. “Istio as an example of when not to do microservices,” Jan. 2020. <https://blog.christianposta.com/microservices/istio-as-an-example-of-when-not-to-do-microservices/> (accessed May 28, 2021).
11. “Istio—Connect, secure, control, and observe services.” Istio.io. <https://istio.io/> (accessed May 28, 2021).
12. C. Box. “Introducing istiod: Simplifying the control plane.” Istio.io, Mar. 2020. <https://istio.io/v1.5/blog/2020/istiod/> (accessed May 28, 2021).
13. “Kubernetes—Production-grade container orchestration.” Kubernetes.io. <https://kubernetes.io/> (accessed May 28, 2021).
14. L. Calcote, *The Enterprise Path to Service Mesh Architectures*. O’Reilly, 2018. (accessed May 28, 2021).
15. “EnvoyProxy.io.” Envoy. <https://www.envoyproxy.io/> (accessed May 28, 2021).
16. “Prometheus—From metrics to insight.” Prometheus.io. <https://prometheus.io/> (accessed May 28, 2021).
17. “Jaeger: Open source, end-to-end distributed tracing.” JaegerTracing.io. <https://www.jaegertracing.io/> (accessed May 28, 2021).
18. S. Newman, *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*. Sebastopol, CA: O’Reilly, 2020.
19. O. Zimmermann. “A definition of done for architectural decisions.” Medium, May 2020. <https://medium.com/olzzio/a-definition-of-done-for-architectural-decisions-426cf5a952b9> (accessed May 28, 2021).


 SUBMIT  
TODAY

 IEEE TRANSACTIONS ON  
**BIG DATA**

**SUBSCRIBE AND SUBMIT**

For more information on paper submission, featured articles, calls for papers, and subscription links visit: [www.computer.org/tbd](http://www.computer.org/tbd)

TBD is financially cosponsored by IEEE Computer Society, IEEE Communications Society, IEEE Computational Intelligence Society, IEEE Sensors Council, IEEE Consumer Electronics Society, IEEE Signal Processing Society, IEEE Systems, Man & Cybernetics Society, IEEE Systems Council, and IEEE Vehicular Technology Society

TBD is technically cosponsored by IEEE Control Systems Society, IEEE Photonics Society, IEEE Engineering in Medicine & Biology Society, IEEE Power & Energy Society, and IEEE Biometrics Council


 75 YEARS  
 IEEE COMPUTER SOCIETY


 IEEE