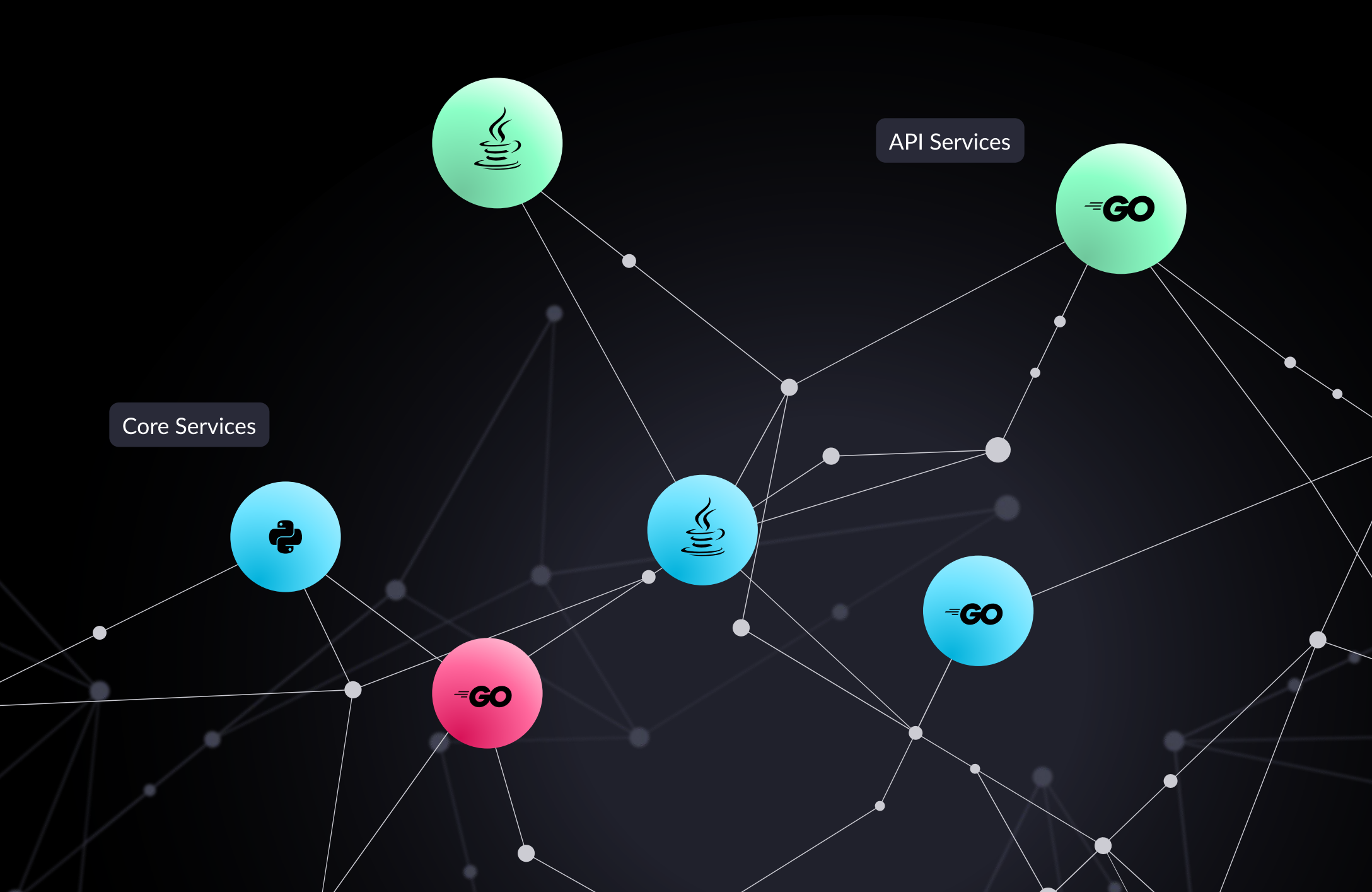
vFunction

GUIDE

Ending Microservices Chaos

How Architecture Governance Keeps Your Microservices on Track

By Alvin Lee, full-stack developer and founder, Out of the Box Development



Introduction

A microservices architecture is the gold standard for building scalable web applications. Gartner estimates that 74% of organizations use microservices for their web applications, with another 23% planning to use them soon.

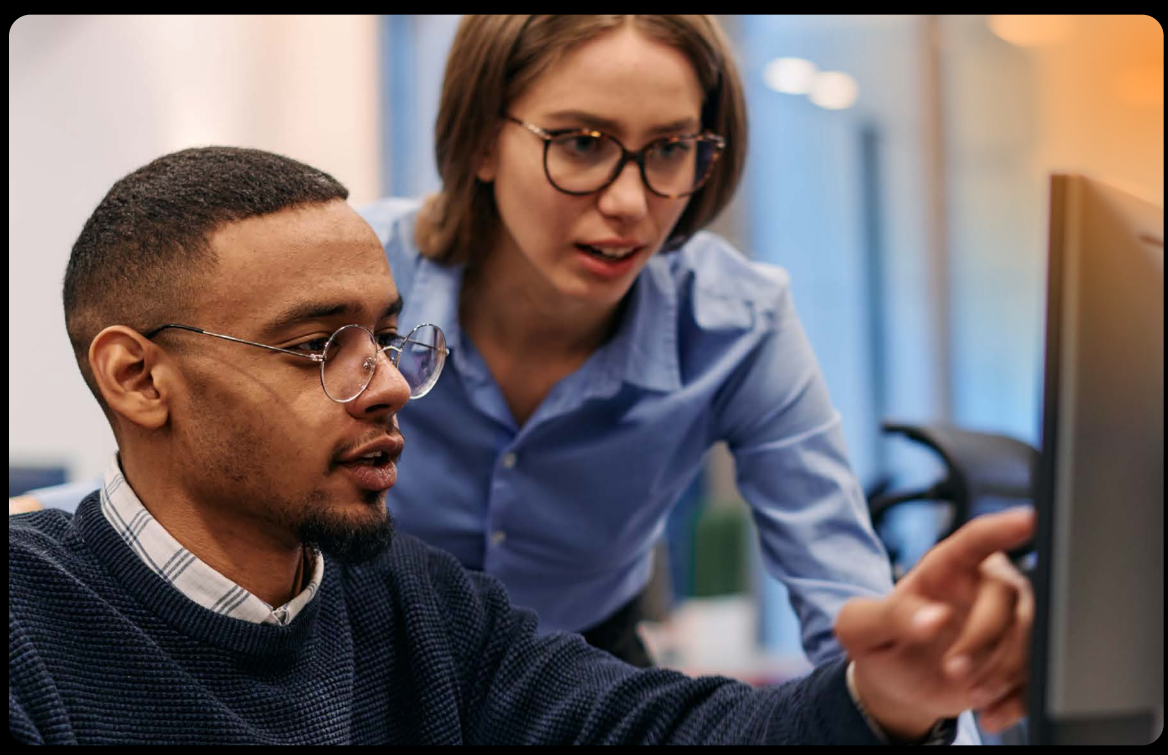
If you're an IT leader, architect, or developer, you might have experienced the faster deployments, better fault isolation, and easier scaling that come with microservices.

However, the old maxim, "There's no such thing as a free lunch," rings true here. Because microservices can also lead to significant operational overhead and complexity. Microservices sprawl, unclear component dependencies, service duplication, and anti-patterns like circular dependencies are all part of the chaos that comes hand-in-hand with microservices.

Perhaps you've seen some of the common symptoms of this microservices chaos:

- Engineering velocity drastically slows down.
- Onboarding new developers grows more difficult.
- Mean time to recovery (MTTR) for outages or performance issues increases.
- Overall system resiliency diminishes, putting business outcomes at risk.

While it's still best practice to modernize monolithic applications by breaking them into microservices, **it's also best practice to proactively guard against the issues that microservices introduce**.



74%

of organizations use microservices for their web applications

*Source: GARTNER, Microservices Architecture: Have Engineering Organizations Found Success? Jun 18 - Jul 13, 2023. GARTNER is a registered trademark and service mark of Gartner, Inc. and/or its affiliates in the U.S. and internationally and is used herein with permission. All rights reserved.

© vFunction 2024

Architecture Governance to Control the Chaos

The best way to proactively guard against these problems is by using **architecture governance** — a collection of rules, processes, and practices that manage and control your software architecture.

With proper software architecture governance, you can reduce microservices complexity, ramp up developers faster, reduce MTTR, and improve the resiliency of your system, all while building a culture of intentionality.

(Note that we're talking about software architecture governance in this article, not enterprise architecture governance which has been around for some time and is concerned with the policies, processes, and procedures of enterprise architecture strategies.)

But exactly how do you build architecture governance?

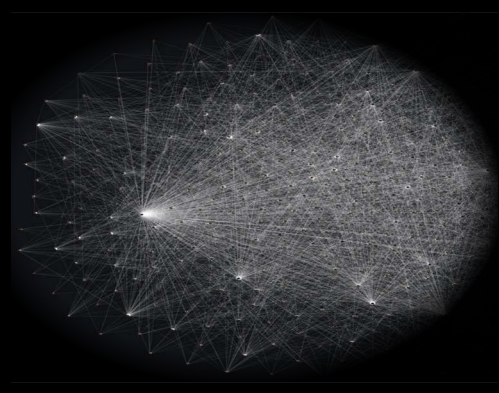
Governance, whether we're talking about access, data, or architecture, is often just establishing a set of rules. But for architecture governance, you need more than just rules. You must start with architectural observability.

Monolithic architecture complexity

In monolithic architectures, teams release updates less frequently, bundling changes into larger updates that slow the rate of new dependencies. However, when complexity does surface, it's deeply embedded and harder to identify or resolve due to the tightly coupled nature of monoliths.

→ Need to modernize

Dependency graph of a 12-year old monolith



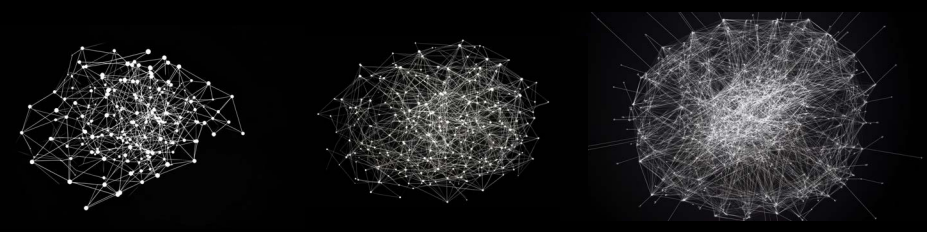
1.5m LOC

Distributed architecture complexity

In distributed architectures, frequent releases by multiple teams often create unintentional dependencies, leading to cascading failures, increased latency from inefficient communication, and reduced scalability. Without clear visibility into service interactions, anti-patterns emerge, jeopardizing reliability and resilience.

→ Need to control/govern

Dependency graph of a distributed architecture within months



100+ microservices → 300+ microservices → 500+ microservices

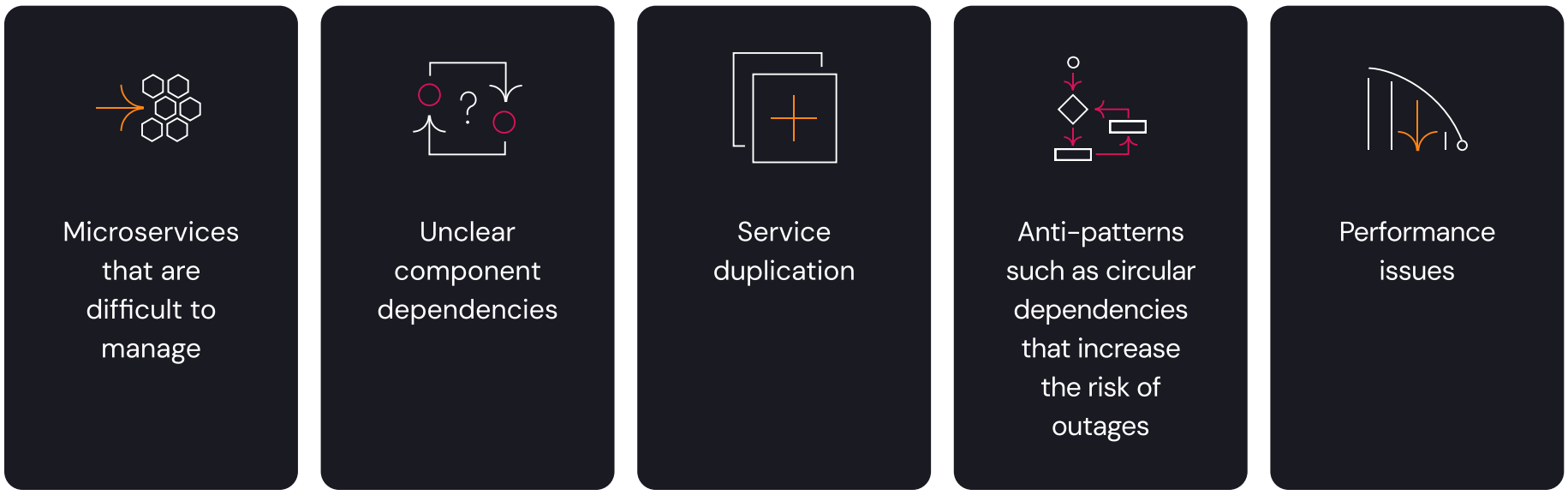
Architectural Observability: The Foundation of Good Governance

To implement effective governance, you first need **comprehensive visibility into your software architecture**.

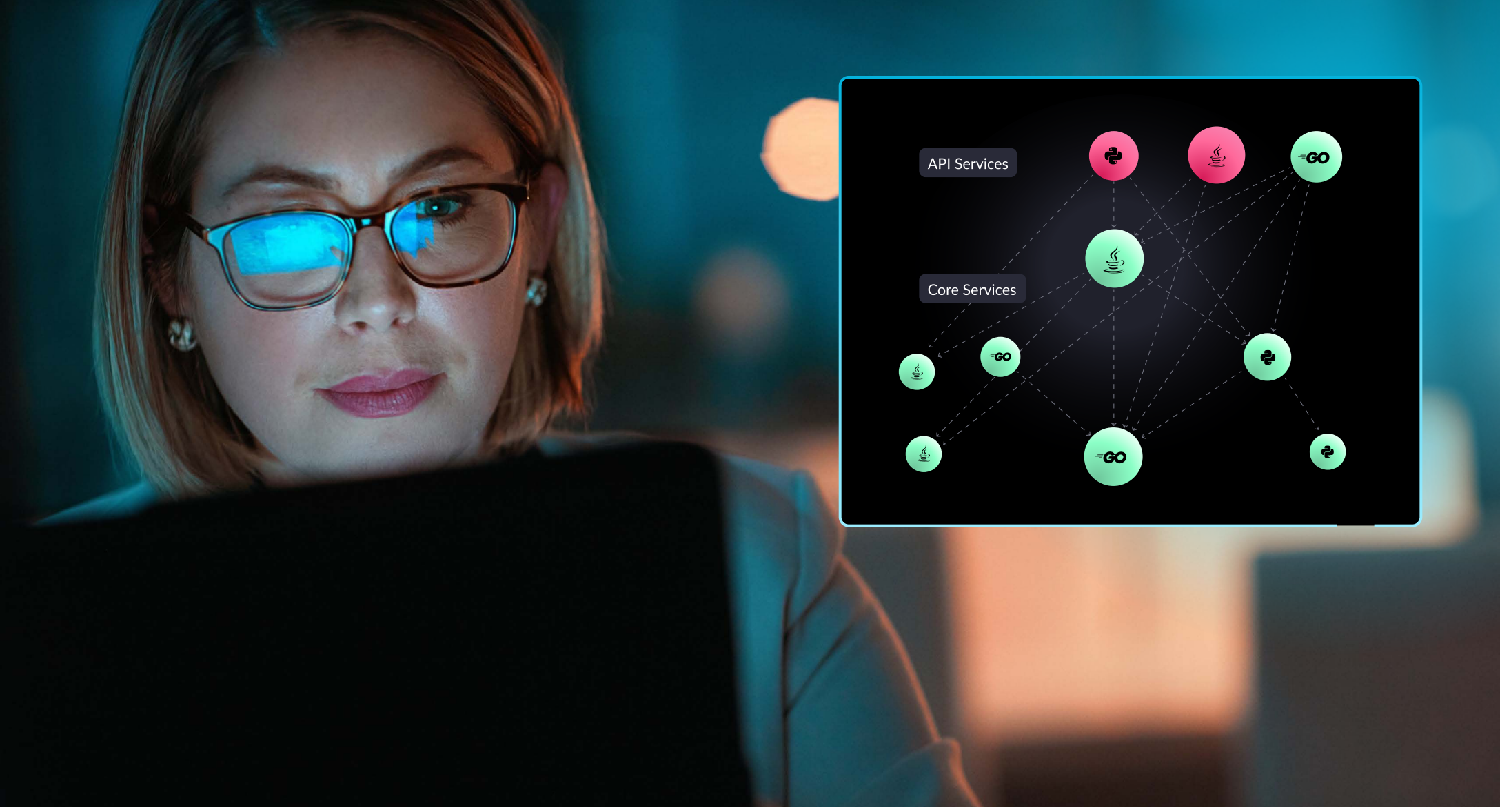
Architectural observability gives you a continuous, real-time view of your architecture. It provides your team with the necessary insights to manage the architecture and interconnectedness of distributed applications.

Architectural observability provides documentation from a working system, it shows you systems flows, dependencies, what's changing, and how your systems interact. It gives your team a thorough understanding of how the architecture works, how it drifts from release to release, and how its changes impact dependent services and resources.

Without this visibility, you're in trouble. If you've built a microservices project, you've probably encountered the results of a lack of visibility:



Your overall application resiliency, scalability, and engineering velocity are at high risk. And it only gets worse — the longer you let this lack of visibility continue, the more complex and unmanageable your architecture becomes.



Good Observability Equals Good Governance

But *with* good architectural observability, you can visualize your architecture layer in real-time, gaining an understanding of how your system works and monitoring changes as your architecture evolves.

With observability, your team can proactively address issues, such as service sprawl and dependency mismanagement — before they lead to technical debt or system failures.

And once you have that observability in place — once you know what's happening and what's changing— you can now create and enforce your rules of governance.

You can implement a governance program that sits over your architecture, monitors, and controls changes, and allows you to build and manage microservices effectively by proactively applying standards and rules to your architecture and codebase as it evolves.

Sounds great! But how do you actually build this? Let's look at an example of how to build this in the real world.

Implementing Architectural Observability and Governance

First, you need to implement your architectural observability.

The domain of architectural observability is new and rapidly evolving. For our example, we'll use one of the platforms at the forefront — vFunction, an AI-driven platform that provides visualizations of software architecture.

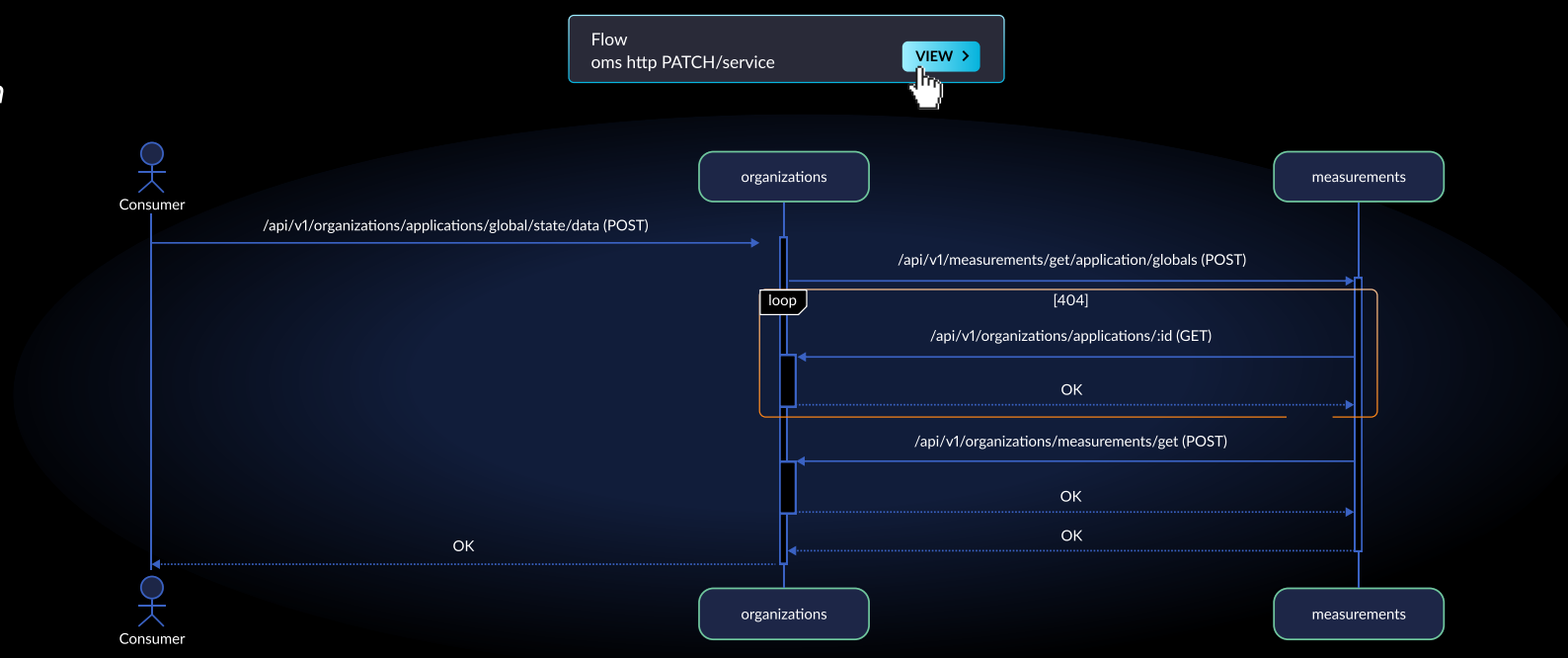
Platforms such as these leverage AI and OpenTelemetry tracing to enable teams to track every aspect of their architecture and monitor changes between releases. It can quickly identify complex flows, architectural technical debt, and architectural drift.

For example, you can better understand your distributed applications:

- Visualize the architecture and automatically create exportable sequence diagrams of all system and data flows.
- Identify drift, such as new services or dependencies.
- Identify resource exclusivity changes (such as multiple new services accessing the same database table) versus previous releases.
- Identify circular dependencies between services that could impact resiliency.
- Identify duplicate functionality/services that may be merged to remove complexity.
- Maintain an up-to-date, real-time view of the global microservices ecosystem.

vFunction uses the same data that APM tools use (so it's straightforward to install) but uses a different layer of intelligence to address the architectural root cause of issues. This is especially important for managing the sprawl and complexity that result from the rapid and frequent deployment of microservices.

Example of exportable sequence flow diagram

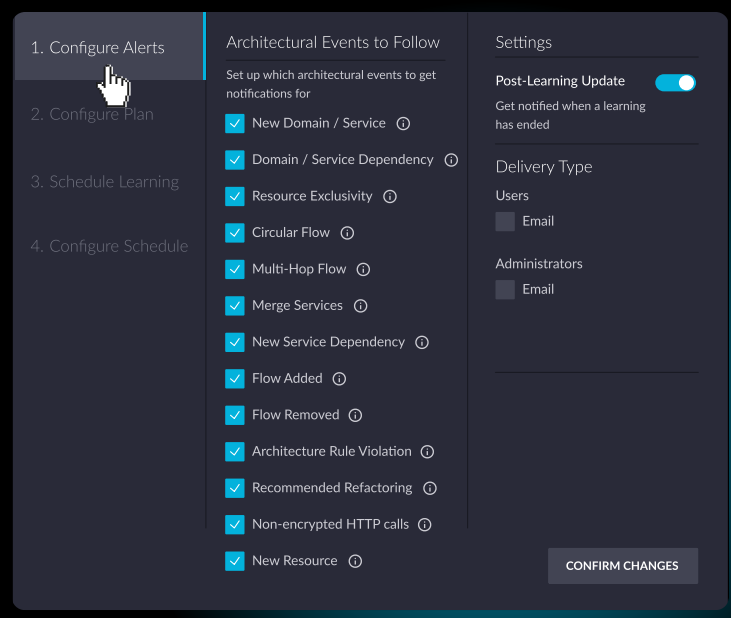


Implementing Rules to Maintain Standards

After implementing your observability, you're ready to create and enforce your rules of governance. With an architectural governance platform, you can create architecture rules and tags and then monitor your architecture in real-time against those rules, enforcing your critical governance policies. Any time a deployment or merge request breaks these rules, you can block that action.

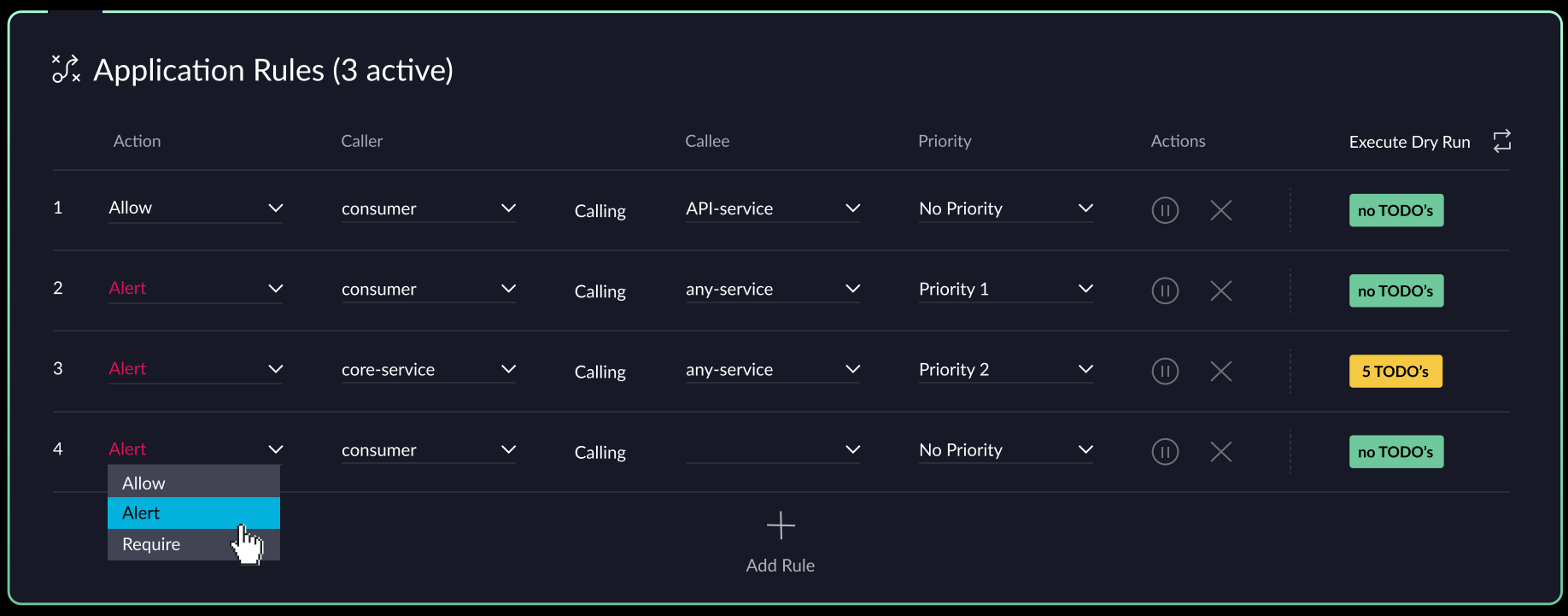
What type of rules might your governance enforce? You could enforce rules such as:

- Dependency requirements: For example, certain services should or should not communicate with other services.
- Restricted resources
- Multiple services should not depend on the same database table.
- Changes should not increase service interdependencies.
- Restrictions on new multi-hop flows that could impact performance and resiliency.



Then, your platform can watch and send alerts for all of the above.

By implementing a focused set of rules and tags, you can create adaptive governance that guides developers in maintaining an optimal, clean, and efficient architecture. You can build architecture governance into your code and your culture.



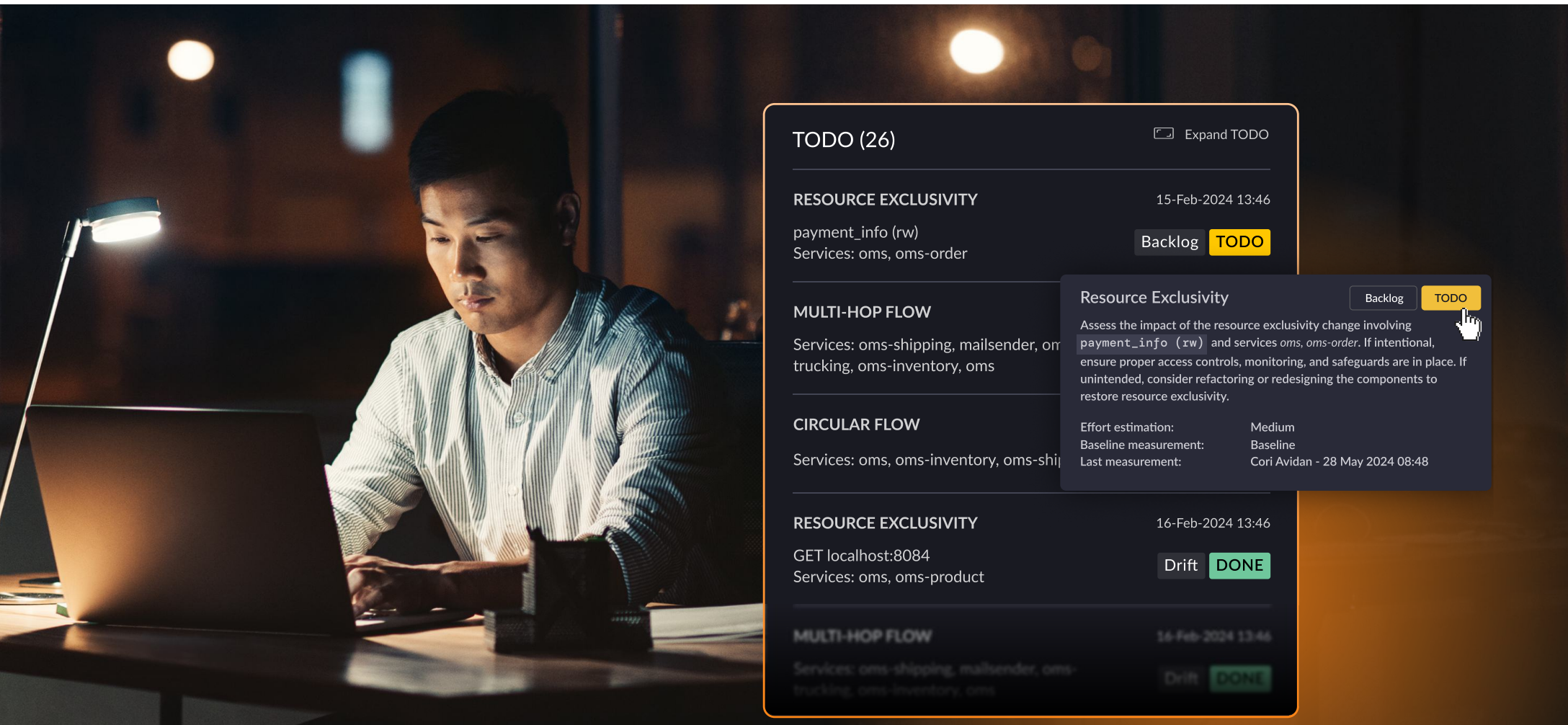
The Role of Microservices Security and Code Quality

In addition to controlling the chaos of microservices with governance and observability, maintaining a high standard of security and code quality is essential. When working with distributed systems, the complexity of microservices — if left unchecked — can lead to vulnerabilities and technical debt.

Qodo provides developers with intelligent insights into code logic, identifying bugs and vulnerabilities that may otherwise go unnoticed. It allows teams to address security concerns during development, reducing risks in production.

Tools from SonarSource — such as SonarLint or SonarQube — focus on continuous code quality and security. They help developers identify potential issues such as code smells, duplication, or even security risks like SQL injection. By integrating seamlessly with CI/CD pipelines, they ensure that every deployment follows strict security and code quality standards.

The connection between **code quality**, **application security**, and **architectural observability** is clear. Poor code quality and unresolved vulnerabilities can lead to a fragile architecture that is prone to outages and security incidents. By proactively managing your code quality and security using these tools, you reduce the risk of microservices complexity spiraling out of control. This, coupled with strong architectural observability, ensures you maintain a resilient and secure microservices ecosystem.



© vFunction 2024

Conclusion

Microservices chaos can threaten to derail development velocity, system resiliency, and operational efficiency. Architecture governance offers a way out of this chaos — a way to regain control over your architecture and steer it toward greater scalability and stability.

Whether your organization is just starting to scale with microservices or is already in the depths of microservices sprawl, it's the right time to introduce architecture governance to ensure that your architecture is efficient and resilient—and ready to support your next phase of growth.

About the Author

Alvin Lee is the founder at Out of the Box Development, LLC. He is a full-stack developer and technology consultant specializing in web architectures, microservices, and API integrations.

Article first appeared in DZone

About vFunction

vFunction, the pioneer of AI-driven architectural observability, delivers a platform that enables you to understand your application architecture, reduce technical debt and manage complexity. Whether you want to modernize monoliths, or add governance to your microservices architecture, vFunction provides the visibility, analysis, control and automation you need. Global system integrators and top cloud providers partner with vFunction to assist leading companies like Intesa Sanpaolo and Trend Micro in discovering their architecture and transforming applications to innovate faster and change their business trajectory. vFunction is headquartered in Menlo Park, CA, with offices in Israel, London, and Austin, TX.

To learn more, visit www.vfunction.com.

© vFunction 2024