Tame microservices with an Istio service mesh

Challenge

As modern applications move toward microservices-based architectures, development teams struggle to properly build, debug, and connect these distributed services. Application operations teams face additional challenges related to deploying and scaling these applications, as well as understanding performance and troubleshooting failures. Containers and container orchestration systems are not enough.

A service mesh addresses these challenges by adding a proxy for each microservice to handle the necessary traffic management, monitoring, and security. The open source Istio service mesh adds intelligence and reliability to applications, augmenting the vision of Red Hat® OpenShift® Container Platform and Kubernetes container orchestration—technologies at the center of digital transformation for many companies.

Microservices are a software development technique—a variant of the service-oriented architecture (SOA) structural style— that arranges an application as a collection of loosely coupled services. With this granular, modular architecture, development teams can rapidly build or modify app components to meet changing needs. Containerized microservices are the foundation for cloud-native applications.

The move to microservices

Public sector IT organizations must be able to react quickly. They do not have the luxury of taking their time to deliver a digital response to wildfires, utility outages, disease outbreaks, or legislative mandates.

Organizations are rethinking the model of monolith applications developed through a "waterfall" approach—a plodding linear sequence from specs to coding to testing and deployment over the course of months. They recognize the benefits of developing software in a way that takes advantage of the agility and pay-per-use nature of cloud technologies.

That is where microservices can help. Microservices divide an application into a collection of single-purpose, independent components. Each microservice performs a function, and together they provide the full functionality for an application.

Microservices offer a number of advantages:

- **Agility.** Microservices support DevOps methods and make continuous integration/continuous delivery (CI/CD) more seamless and achievable. Microservices can be owned and managed by a small team, and development can be done simultaneously by multiple teams. Applications are developed, updated, and delivered faster.
- **Flexibility.** Unlike monolithic applications, microservices can grow where and when they are needed. They can scale horizontally and independently to support temporary traffic spikes, run batch processes, or meet other unexpected situations.
- **Fit to purpose.** Thanks to language-agnostic application programming interfaces (APIs), development teams can work with the languages and frameworks best suited for the needs at hand.
- **Resilience.** Service issues, such as memory leaks or open database connections, only affect a single service. If one piece goes down, it does not bring down the entire application.

By adding more intelligence and reliability to applications on the platform, Istio reinforces the vision of Kubernetes and Red Hat OpenShift, which are at the center of digital transformation for many companies.

Technology solutions and advantages

Microservices challenges

Development teams have to address service communications needs such as load balancing, fault tolerance, end-to-end monitoring, dynamic routing, compliance, and security. You could cobble together libraries, scripts, and Stack Overflow snippets to provide these functions, but the result would be inconsistent and opaque, with potential security concerns.

Initially, libraries built within microservices might be able to handle service-to-service communication with minimal disruption to operations. But if you are fulfilling the potential of microservices by increasing scale and features, that will not be true for long. Scaling can bring problems over time as services get overloaded with requests and developers spend more and more time coding request logic for each service.

For all the merits, there is inherent complexity in adopting microservices. A microservices app is now a distributed system. What was a function call in the code of a monolithic application now has to traverse a network. Networks bring their own issues with latency, reliability, and security.

Each microservice must be coded with logic to govern service-to-service communication. Developers then have less time to focus on developing features, and they spend more time maintaining code for non-business logic.

Embedding communication logic into the service might not pose problems for simple applications, but it can quickly become unwieldy. As a microservices-based app gains size and complexity, it can become harder to understand and manage. When the service communication logic is hidden within each service, communication failures are much harder to diagnose – almost impossible in a highly complex app.

The app may also require such capabilities as discovery, load balancing, failure recovery, metrics, and monitoring. And a microservices-based application often calls for more complex operational capabilities, such as A/B testing, phased canary rollouts, rate limiting, access control, and end-to-end authentication.

Developers should not be encumbered with the operational aspects of traffic management, observability, and security within each microservice. You do not want microservices to have to run business logic as well as a host of other capabilities unrelated to their core functions. A service mesh can help.

Introducing the service mesh

A service mesh provides the necessary service-to-service capabilities – traffic monitoring, access control, discovery, security, resiliency, metrics, and more – without requiring changes to the code of any of the app's microservices.

In a service mesh, requests are routed between microservices through proxies in their own infrastructure layer. These proxies are sometimes called "sidecars," since they run alongside each service, rather than within them. Taken together, these sidecar proxies – decoupled from each service – form a mesh network.

All the traffic meant for a service goes to the proxy, which uses policies to decide how, when, or if that traffic should go on to the service. This visible infrastructure layer can document how well (or not) different parts of an app interact, so it becomes easier to optimize communication and avoid downtime as an app grows.

A service mesh does not introduce new functionality to an app's runtime environment – apps in any architecture have always needed rules to specify how requests get from point A to point B. A service mesh is different in that it takes the logic governing service-to-service communication out of individual services and abstracts it to a layer of infrastructure.

The infrastructure of a service mesh includes a data plane and a control plane. Microservices and their sidecar proxies make up the data plane, which includes data management and request processing and response. The control plane manages and configures proxies to route traffic. It also configures components to enforce policies and collect telemetry.

The service instance and sidecar proxy reside in containers managed by a container orchestration tool such as Kubernetes.

Just as microservices help to decouple feature teams from each other, a service mesh helps to decouple operators from application feature development and release processes.

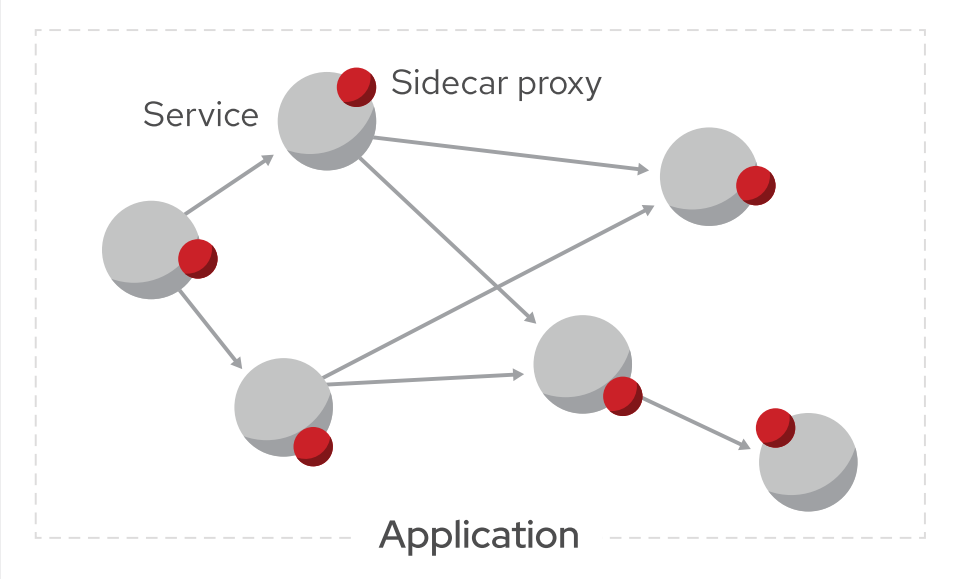

Figure 1. Microservices and sidecars

Red Hat provides a production-ready platform for adopting microservices, coupled with support, consulting, and essential middleware for building modern applications.

A sidecar proxy sits alongside a microservice and routes requests to other proxies. Together with a control plane, they form a service mesh that provides service-to-service communication, visibility, and control.

Introducing Istio

There are several options for a service mesh architecture. Red Hat has chosen Istio, an open source project from Google, IBM, and Lyft. Google and IBM have extensive experience with large-scale microservices in their own applications and with enterprise customers in sensitive, regulated environments. Lyft developed the Envoy proxy for its internal use, to manage more than 100 services spanning 10,000 virtual machines processing 2 million requests a second.

Istio provides behavioral insights and operational control over the service mesh and the microservices it supports:

- **Connect.** Easy rules configuration and traffic routing let you control the flow of traffic and API calls between services. Istio simplifies configuration of service-level properties such as circuit breakers, timeouts, and retries, and simplifies the process of setting up tasks such as A/B testing,

Create powerful and resilient cloud-native applications. Improve traffic control, service resiliency, testing, observability, and security of microservices.

canary rollouts, and staged rollouts with percentage-based traffic splits.
With better visibility into traffic and out-of-the-box failure recovery features, you can catch issues before they cause problems. Calls are more reliable and the network becomes more robust.

- **Provide security.** Istio automatically provides security services at scale through managed authentication, authorization, and encryption of communications between services. While Istio is platform-independent, its auth/authz capabilities and Kubernetes network policies can work together to offer more secure communication at both the network and application layers.
- **Control.** Configure custom policies for an application and enforce them at runtime, such as:
  - Rate limiting to dynamically limit the traffic to a service.
  - Denials, whitelists, and blacklists to restrict access to services.
  - Header rewrites and redirects.
- **Observe.** Istio provides a unified interface to monitor and manage service communication. See what is happening with automatic tracing, monitoring, and logging of all services. See how service activity affects performance upstream and downstream. Custom dashboards provide visibility into the performance of all services.

With Istio, you can more effectively set, monitor, and enforce service-level objectives (SLOs) and quickly detect and fix issues. For example, if a service fails, Istio can collect data on how long it took before a retry succeeded. Aggregate data on failure times for a given service can help define the optimal wait time before retrying that service, so the system is not overburdened by unnecessary retries.

You do not have to use every Istio feature when you start building your service mesh. For instance, you might first use the service mesh to provide a practical migration path to mutual transport layer security (mTLS) for securing internal service communication. Start with the most pressing use case and take advantage of other capabilities as needed.

An Istio service mesh reduces the complexity of deployments—and the burden on your development teams.

Red Hat and Istio

As organizations seek to be more agile, release new capabilities faster, and deliver more for less, the value of a service mesh becomes clear. Red Hat OpenShift Service Mesh is based on Istio and is available at no cost with Red Hat OpenShift Container Platform.

Red Hat helps you get started faster because OpenShift Service Mesh is engineered to be ready for production. Developers can implement communications policies without changing application code or integrating language-specific libraries. OpenShift Service Mesh also eases the path for operations because it installs easily on Red Hat OpenShift, has been tested with other Red Hat products, and comes with access to award-winning support.[1]

1 *Red Hat Customer Portal, "Awards & recognition." Accessed April 2020.*

Istio is the natural next step for developing and operating microservices. It moves language-specific, low-level infrastructure concerns out of microservices and into a common service mesh with Kubernetes container orchestration.

With an Istio service mesh:

- Developers can focus on adding business value instead of connecting, monitoring, and securing services.
- Problems are easier to recognize and diagnose, thanks to automatic tracing and deep visualization.
- Apps are more resilient to downtime, since requests can be routed away from failed services.
- Performance metrics reveal ways to optimize communications in the runtime environment.
- Service communication uses managed authentication, authorization, and encryption to provide security.

Any organization that has or will develop microservices-based apps can benefit from a service mesh. It particularly shines where resilience and zero-trust security are prime considerations. Red Hat OpenShift Service Mesh built on Istio provides a uniform way to connect, manage, and observe those applications.

Learn more

Read more about service mesh and watch the Introduction to service mesh and Istio webinar.

North America
1 888 REDHAT1
www.redhat.com

Europe, Middle East, and Africa
00800 7334 2835
europe@redhat.com

Asia Pacific
+65 6490 4200
apac@redhat.com

Latin America
+54 11 4329 7300
info-latam@redhat.com

facebook.com/redhatinc
@RedHat
linkedin.com/company/red-hat

redhat.com
#F23287_0420