Whitepaper

Event Mesh: Event-driven Architecture (EDA) for the Real-time Enterprise

Jason English

Principal Analyst, Intellyx

November 2019

Are events a blind spot, or are they driving your business?

Life is asynchronous. As conscious beings, we can perceive life as a series of observed ***events***, which we subscribe to by giving them attention.

The same goes for ***event-driven architecture (EDA)*** in enterprise IT. Your systems are already soaking in events – as each transaction, compute process or data transfer could be thought of as another event, kicked off by stimuli at a given point in time. The movement toward cloud-native microservices and the proliferation of millions of connected IoT device interactions with enterprise systems only raises the number of events exponentially.

EDAs have existed for years, handling several of the most high-volume asynchronous integration scenarios such as banking transaction networks, stock trading, travel reservation systems, and government agency communications and control.

This paper will discuss how an ***event mesh*** can enable EDA and co-exist with commonly used point-to-point integration and service mesh approaches, providing the enterprise with reduced integration effort, better scalability, and the ability to operate and meet customer demands in real-time.

Why become event-driven anyway?

The ability to asynchronously align services and data through EDA should be at the heart of any secure, agile, distributed, and cloud-native application's future. Why? Because, like oil, the value of data increases exponentially when it's put in motion, and the time value of data is often diminishing. Real-time, event-driven data movement unlocks much more potential value than point-to-point movement.

But adopting EDA is not easy today, because enterprise IT teams are more distributed than ever, the volume of events being created by modern technologies increases daily, and we lack the tools, patterns and culture that would make adopting EDA less daunting.

Event mesh plays a critical role as an IT architecture layer that routes events from where they are produced to where they need to be consumed – regardless of the system, cloud, or protocols involved. If you're familiar with service mesh, event mesh is a similar concept, but for the event-driven world.

Loosely coupled by nature, this layer connects event brokers (modern messaging middleware) within and across different environments. Applications in one environment can receive event notifications created by applications in any other environment, as long as both applications are connected to a local event broker.

Challenging EDA misconceptions

Event-driven architectures will commonly follow a Publish-Subscribe (or, pub-sub) messaging pattern.

In pub-sub, the message providers (or Publishers) do not send messages to specific consumers (or, Subscribers) but rather, categorize their data as it is published, by type or content. Subscriber services can then, in turn, filter and consume the messages they are looking for as events, rather than explicitly requesting a response from the publisher.

This pub-sub pattern has existed for about as long as computing itself. For instance, your favorite RSS blog or podcast feed connects publishers and subscribers based on the event of a new post or show becoming available.

Pub-Sub messaging, and therefore the EDAs built on them, represent an ideal model for today's distributed, decoupled service architectures. Given that, what are some misconceptions that are holding enterprises back from embarking on an EDA approach?

- **High-capacity, high-security applications.** Companies have very high expectations of their systems to securely handle transactions at volume. They may assume a more direct form of service integration would scale best, even though EDAs have been

proven to scale quite fast and offer fewer threat vectors such as open ports to cyber attackers.

- **Point-to-point architectural thinking.** In designing distributed computing systems, it may seem easier to build more direct point-to-point, request-response message interactions that are synchronized between known services, where the looser coupling of the EDA approach seems to require additional architectural forethought (this isn't the case, as we'll discuss).

- **REST-based integration is currently far more popular.** We owe much of the growth of service-based computing to the development community's support of REST (Representational State Transfer) integration tools and techniques for connecting services. This doesn't mean REST vs. EDA is an either/or choice, as both approaches will co-exist now and in the future.

- **Industry pressure for a service mesh.** No business system is an island, and the need to continually bring forward legacy technology into an increasingly distributed computing environment, with containers and microservices strewn across hybrid clouds, has caused a spaghetti mess of protocols, message queues, APIs and data pipelines.

  Thus, heavy commercial and open source efforts are underway to define solutions for a ***service mesh***, providing a common layer to orchestrate synchronous, RESTful service-to-service communications. The hottest service mesh out right now is Istio for orchestrating Kubernetes instances. But where does this innovation leave event-driven developers?

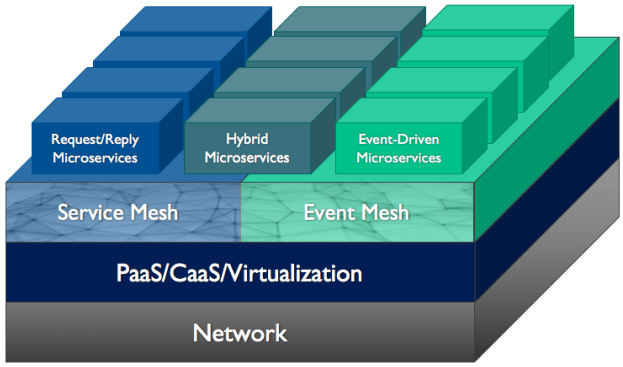
While a service mesh is well-suited for synchronous RESTful service orchestration and request-reply interaction among cloud-based applications, it cannot support asynchronous, EDA-style, any-to-any integration of heterogeneous services based on events.

Solution: Event Mesh and Service Mesh Together

By comparison, the event-driven integration world has its own lightweight orchestration approach called an ***event mesh***, based on a network of interconnected event brokers such as the PubSub+ solution offered by Solace.

Both an event mesh and a service mesh can provide a communication and data routing layer between cloud-based microservices and the network. But service meshes are not as useful when non-cloud legacy applications, third-party data feeds, IoT devices and edge

computing enter the picture with asynchronous inputs. These situations call for an event mesh, which includes an intelligent ***event broker*** to accept and route messages between all these discontinuous services on the fly.



Example of Event Mesh and Service Mesh orchestrating microservices workloads atop a common network. (Source: Solace)

Think of it this way: If our extended enterprise systems were the human body, the service mesh may provide a circulatory function with a synchronized heartbeat moving data between applications, while the event mesh would behave more like the nervous system, sensing any asynchronous events, with the event broker as the brain, quickly interpreting and routing messages wherever they need to go.

As the world becomes more event-driven, with legacy application functionality being called on to interact with modern stateless microservices, running in ephemeral cloud-native infrastructure, informed by IoT device signals and orders occurring anywhere in the world, you'll see event mesh taking its rightful place alongside service mesh.

Where to start with event mesh + service mesh?

Anywhere you depend on integrating a mix of cloud-native containers in any private or public cloud provider with any other services, devices and/or legacy systems, you have a good reason to leverage the best of both worlds.

So while you may use the service mesh to discover, coordinate and balance your RESTful microservices across Kubernetes instances on AWS or Azure, anywhere your workflow is transmitting messages outside that particular cloud region -- through JSON, or JMS, or even an old MQ ESB – you need an event mesh and its event broker ready to go.

An event broker should enable almost any system, service or device, in any region, to quickly publish or subscribe to another service, whether its events are in a secured, permissioned channel, or publicly available.

Three steps to getting started:

1. **Open your mind toward an event-driven approach.**
   Select your next application design or improvement project based on asynchronous events moving through a pub-sub architecture, rather than the more direct request-reply approach indicative of a RESTful workflow.

   Changing your mind and skillset is easier said than done, of course, but remember: the EDA pattern itself has a well-established history, its own modern open source APIs and tools, and an active development practitioner community with educational resources. Developers and architects can engage and learn more at the AsyncAPI open source project and when the EDA community launches Event Academy, an educational hub to help architects, developers and IT executives understand and embrace event-driven architecture.

2. **Don't just govern applications and data as assets. Govern your events as assets.**
   So much attention has already been spent on managing the lifecycle of applications, and treating data as the primary asset: connecting to it, cleansing and filtering it, ingesting and transforming data, pushing it through pipelines to its ultimate destination, with security at every point.

   What if you instead focused on governing the events as assets, given the real-time nature of so many business applications, and the increasingly heterogeneous underpinnings of our systems?

   An event broker that securely interprets and handles event data-in-motion can reduce data management costs and system-wide resource needs, while making the resulting applications more responsive to the content of the data itself.

3. **Start building an event mesh.**
   Fortunately, you don't need to start from scratch here. If well planned, deploying an event mesh of brokers can be done with little interruption to live services, and at much lower cost impact than you typically expect of an architectural paradigm shift. An event mesh can be turned on as an overlay to your existing service mesh, as well as any other integration methods.

The event mesh and its associated tooling should naturally be able to discover, publish and subscribe to any and all services in your hybrid IT environment, or the eventual migration to EDA could get stalled by a dependency.

An event broker like Solace's PubSub+ includes ***protocol mediation***, which allows events to be published or subscribed to across multiple messaging frameworks and associated protocols.

Whether protocols and APIs are proprietary to service mesh vendors, cloud-native platforms like Istio and Kubernetes, Kafka topics and Spring messages, or any of the hundreds of other forms of APIs and messaging frameworks of legacy apps, the business logic of event processing carries on, unlimited by the integration differences of underlying systems.

Event-driven examples in the real world

Transforming a Bank:
This bank's innovation team designed a new cloud-based micropayments service for customers, using microservices in Kubernetes pods, orchestrated with an Istio service mesh. However, the new app still needed a way to incorporate the real events – balance checks, orders and settlement confirmations – from all of their existing legacy banking and third-party transaction systems to complete the loop.

An event mesh allowed them to quickly add the ability to accept and communicate critical events with systems outside the scope of the cloud-native development area.

Transforming Government:
Take the FAA (Federal Aviation Administration) or any similar government agency with a huge transactional and data footprint.

As their control centers must each process millions of daily flight notifications and events, they've been on an EDA approach all along. But attempting to make their existing Kafka clusters extend across the enterprise to share more real-time data across control centers would require serious point-to-point integration work.

By employing an event mesh, the agency can immediately incorporate the Kafka services as pub-sub providers, while expanding their integration with legacy air traffic and weather systems, and making their own events available to commercial travel applications.

Transforming Commerce:

Customer-facing applications demand the highest levels of performance and reliability, and to maintain critical service levels, many e-commerce firms design their application workloads to run across multiple public cloud and private cloud data centers.

Travel apps are among the most demanding e-commerce workflows to integrate, as they must take in a huge number of customer and service requests, interact with financial systems, and gather up-to-the-minute availability data from a constellation of carriers and travel providers.

Enter the event mesh, with event brokers that can learn to route and optimize events for processing using the lowest-cost or most responsive compute resources, ensuring better performance-to-cost ratios on each travel request. Since many of the travel partners constantly publish availability data for potential travelers, the event mesh lets the firm consume this data where it resides as events, rather than moving and processing the data in much more costly and less time-efficient ways.

The Intellyx Take

We live and work in an event-driven world.

Tightly coupled integrations, and conventional data transfer and processing techniques won't be able to keep up with the accelerating pace of events in the new economy. Customer demand and cost pressures on IT will require enterprises to change the way they think about integration and messaging.

In your next design or change exercise, rather than calling for more point-to-point integrations – start finding ways to incorporate pub-sub calls that support an event-driven approach.

Seek out an intelligent event mesh to reduce the configuration burden, and allow consumers and producers of events to communicate in real-time. When done right, an event-driven architecture is responsive and resilient, with a smaller data footprint and cost profile.

An event mesh doesn't mean all the integration goodness of REST-based request-response models will go away. True cloud-native architectures will incorporate the inherent strengths of both approaches, considering a service mesh for RESTful orchestration of Kubernetes clusters, and overlaying it with an event mesh for pub-sub capabilities and integration with both cloud and non-cloud-based legacy systems.

Will we see one platform to rule them all? No, we will continue to roll into a heterogeneous, hybrid IT future, of which an event mesh is an essential element.

Only one question remains: Are you ready to drive events?

About the Author

Jason "JE" English is Principal Analyst and CMO at Intellyx. He is focused on covering how agile collaboration between customers, partners and employees accelerates innovation. He led marketing efforts for the development, testing and virtualization software company ITKO, from its bootstrap startup days, through a successful acquisition by CA in 2011. JE co-authored the book Service Virtualization: Reality is Overrated to capture the then-novel practice of test environment simulation for Agile development, and more than 60 thousand copies are in circulation today.