Innovation Insight for Event Brokers

Published: 31 July 2018 **ID:** G00362911

Analyst(s): Yefim V. Natis, Keith Guttridge, W. Roy Schulte, Nick Heudecker, Paul Vincent

Application leaders engaged in digital transformation must include event-driven computing in their portfolio of skills and technology, including the key related responsibilities for event brokering and event stream analytics.

Key Findings

- Event brokers are middleware products that are used to facilitate, mediate and enrich the interactions of sources and handlers in event-driven computing.
- Event-driven architecture (EDA) is inherently intermediated. Therefore, the intermediating role of event brokering is a definitional part of EDA. All implementations of event-driven applications must use some technology in the role of an event broker.
- Many middleware products, starting with message-oriented middleware (MOM), can play the role of an event broker, but most are designed for objectives other than event processing.
- The strategic role of event-driven computing in digital business, that strives for continuous awareness, intelligence and agility, creates a demand for event broker technology that is designed specifically for event-driven use cases.
- A hybrid integration platform (HIP) plays a central role in the strategic design of a digital business technology platform. A mature HIP includes event broker capability, along with traditional integration and API management, to support the multiarchitecture design of digital business.

Recommendations

Application leaders engaged in modernizing their application architecture and infrastructure should:

- Build skills and technology for event-driven design to enable application designers to freely draw on traditional request-driven SOA or EDA capabilities as the business requires.
- Choose advanced event brokers for digital business innovation initiatives because of their specialized support for event processing, even if the previous-generation alternative pub-sub technology, such as MOMs or ESBs, is already used elsewhere in your organization.

- Include advanced event brokers when assembling your hybrid integration platform suite to support strategic event-driven solutions in your multiarchitecture application environment.
- Adopt event brokers with the understanding that the increasing maturity and standardization of technology over time will likely lead you to make changes to technology and its use patterns.

Table of Contents

List of Figures

Analysis

Just as the use of request-driven service-oriented architecture (SOA) has created an essential requirement for API management, event-driven applications are driving increasing demand for centralized control of event-driven communications. As digital business increases the complexity of application portfolios, API managers (APIM), event brokers and multifunctional integration platforms emerge as key components of a well-functioning HIP, and as strategic enablers of digital business optimization and transformation.

Definition

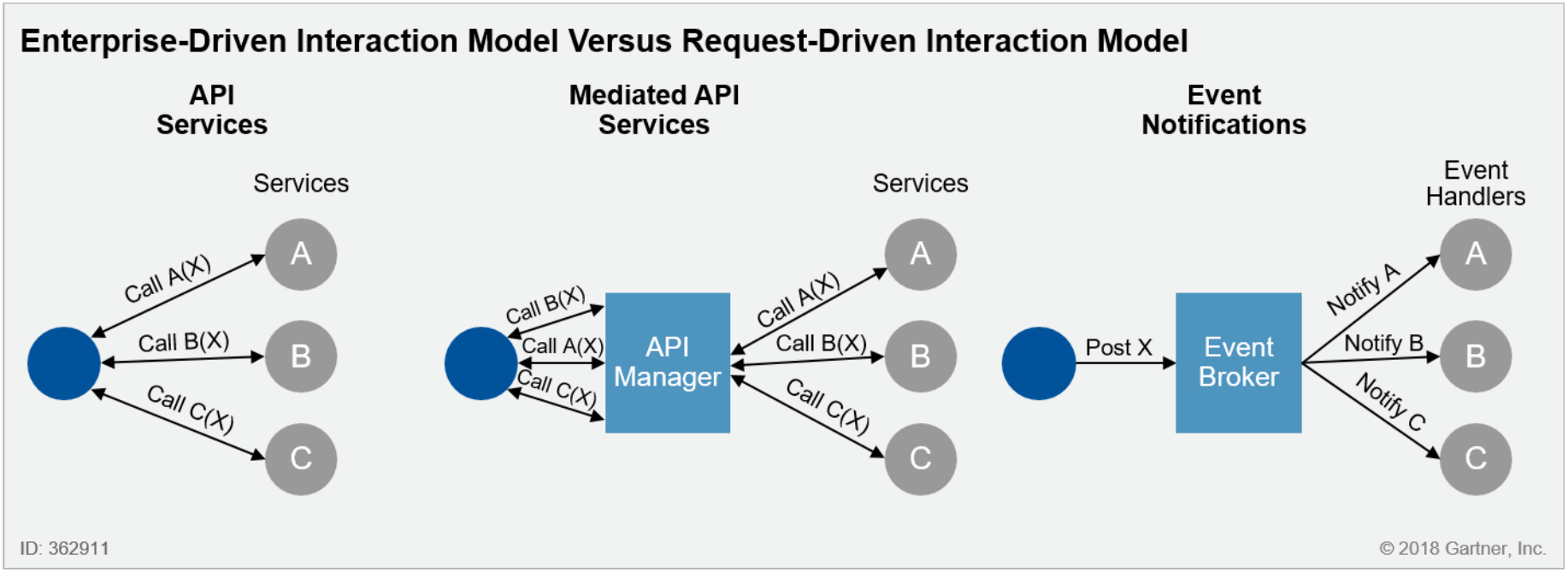
Event brokering is a role played by middleware in facilitating event-driven application architecture. The minimum capability required to play the role of event broker is pub-sub messaging. All middleware products, including MOMs and ESBs, supporting pub-sub can play the role of an event broker and can be referred to as basic "event brokers" when so deployed. Middleware products that additionally offer special support for event-centric use cases (for example, a persistent event ledger for analysis and event sourcing, or programmable extensibility for custom filtering and analysis) are "advanced" event brokers.

Description

Event-Driven Architecture in Context

Event-driven architecture (EDA) is an application design model based on indirect event notifications. It complements the more-traditional request/reply service invocations via direct calls to service APIs. The indirect nature of interactions in EDA makes an intermediary an essential part of the architecture. (Compare this to the common SOA practices that implement direct access to service APIs; and to the use of an APIM, which acts as an optional SOA intermediary [see Figure 1].) Note that "notification" may carry the semantic content of the event object by value, or it may only deliver a reference to an external location where the event data can be found.

Figure 1. A Comparison of Event-Driven Versus Request-Driven Interaction Models



Source: Gartner (July 2018)

The key distinction of EDA and the key responsibility of an event broker is the managed asynchronous autonomy of application components (sources and handlers of event notifications). For example, a "taxi arrived" event notification issued by a taxi tracking application is posted to the broker instead of directly invoking the passenger app. In this approach, the passenger, the taxi dispatcher and the city agencies all can be notified without the taxi tracking application engaging with any of them, and without the overall operation depending on simultaneous availability of all services.

The key development benefits of such isolation (and of the use of an event broker) are the low-friction extensibility and increased agility of the application's life cycle. These characteristics are a natural fit for the continuous product (versus scheduled project) model of application delivery. Autonomous application components can be delivered, replaced and changed with minimal impact on any other parts of the application. That, in turn, reduces costs and friction when continuously updating and extending the application product.

The key business benefit is the potential access to continuous intelligence and real-time business awareness and responsiveness. Dispatching events to multiple endpoints, some for immediate analysis and decision making, others for filing and processing later, provides the organization with both the real time and the later, consolidated analyses to deliver rich, continuous intelligence opportunities. An event broker serving as the dispatcher of the event stream data is the key enabler for this business benefit.

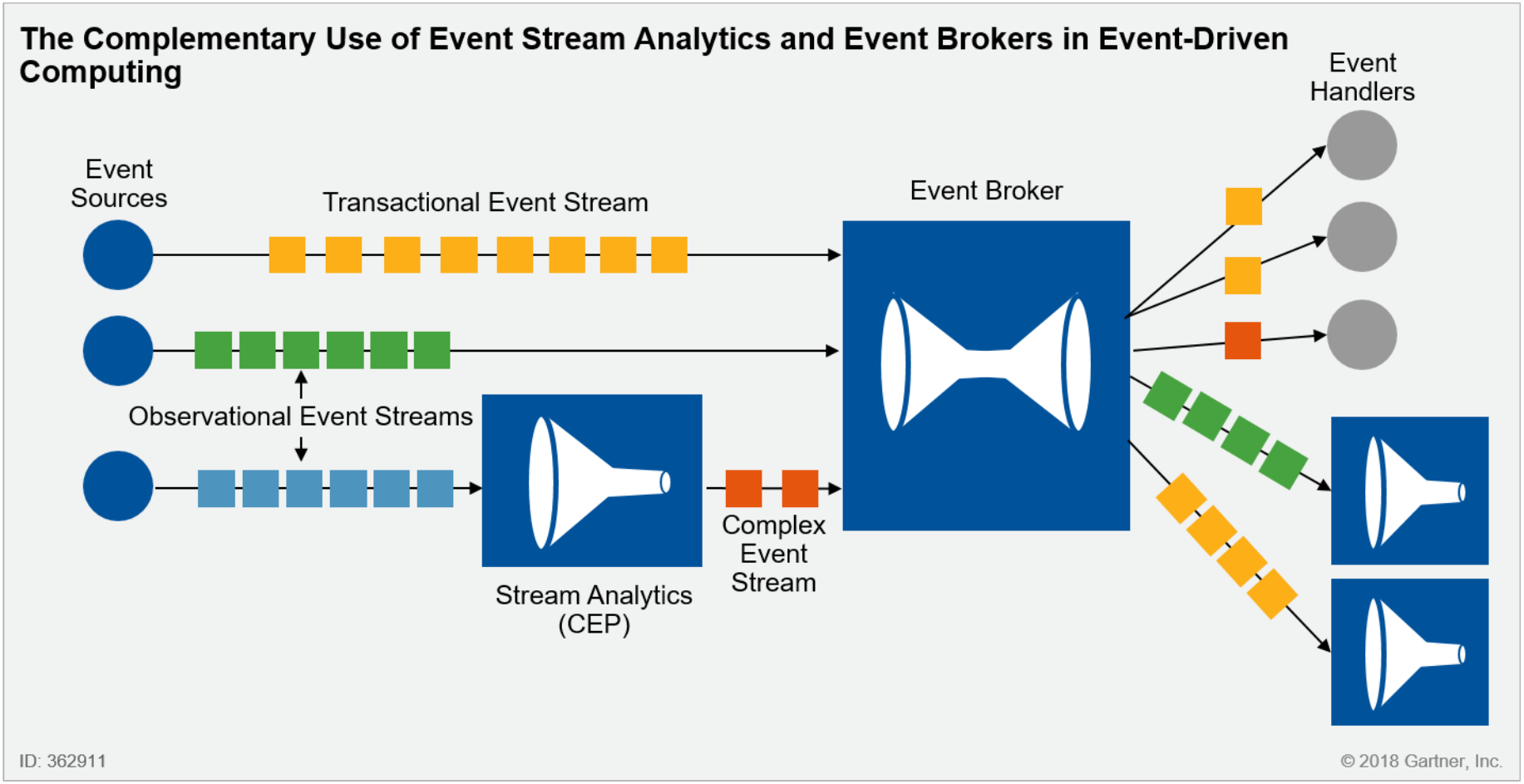
The cost of these benefits is the diffusion of control over business processes and increased governance complexity. Advanced event brokers are designed to mitigate both of these challenges, though the current state of technology is still limited and leaves much of the governance responsibility to the developers (see the Risk section for more detail).

Event Brokers in Context

Many middleware products can play the role of an event broker. ESBs, MOMs, business process management (BPM) suites and Java EE application servers (via the embedded JMS) all typically have pub-sub capability and, therefore, can perform at least the minimum function of facilitating EDA interactions. However, few of these products were designed for event processing as their primary objective. ESBs center on integration and MOMs on delivering messages, while BPM suites manage workflows and application servers are a general-purpose application platform.

With the increasing emphasis on event-driven computing in digital business, new middleware products are emerging with the primary purpose of supporting event processing. Two main categories of technology, offered together or separately, are part of this market: the event brokers and the stream analytics engines. (Stream analytics is a modern evolution of complex event processing [CEP] technology and the two names sometimes are used interchangeably.) Stream analytics may act as either a source or a handler of event notifications (see Figure 2).

Figure 2. The Complementary Use of Event Stream Analytics and Event Brokers in Event-Driven Computing



Source: Gartner (July 2018)

All Event Notifications Arrive as Streams, Though Only Some Are Subject to Stream Analytics

- Some streamed events are directed to event brokers to be sent to subscribing event handlers one at a time (for example, transaction notifications like "purchase order submitted").
- Other streamed events are directed straight to stream analytics engines to be processed in sets (for example, time-series observations like temperature measurements inside a refrigerated shipping container).
- Stream analytics engines produce complex events, most of which are to be processed one at a time. Complex events may be aggregations that summarize the information (for example, sum, average, count etc.) or they may signify detection of a pattern (temperature in the refrigerated container is over the safe threshold). When a notable pattern (anomaly) is detected, a notification is sent to event handlers via an event broker.
- Some event streams are processed both ways: individually (by an event handler, such as "credit card payment submitted") and also as time series by a stream analytics engine (such as "detect and signal suspicious patterns in credit requests over a given period").
- Event brokering and stream analytics may be supported in the same product or cloud service (for example, Apache Kafka and Kafka KSQL; Amazon Kinesis and Kinesis Data Analytics) or offered separately. In both cases, they have distinctly separate roles and are implemented by separate and different technology stacks.

Event brokers (from basic MOMs like RabbitMQ to advanced brokers like Microsoft Azure Event Grid) receive event notifications, may apply prearranged governance and mediation policies to each notification, then notify the subscribers (event handlers). This is a "one-at-a-time" processing of event notifications. (Note that the event handler may be a stream analytics engine. In that case, the events are handled as a continuous stream.)

The prearranged policies and procedures that advanced event brokers apply to incoming event notifications can vary from simple, to elaborate, to custom-programmed algorithms. All event brokers must support the basic pub-sub logistics, but those advanced ones that are designed to support event processing in the modern digital business environment offer significant additions, including some or all of the following:

- **Integrity** by ensuring guaranteed "exactly once," "at most once," or "at least once" delivery and ordering.
- **Event ledger** by way of a permanent, append-only, immutable log of events. It is accessible as a data store for postevent analysis, machine learning, replay or event sourcing. In some cases, the event ledger may be used as the application's record of truth.
- **Security** at various levels of authentication, authorization and topic/channel access control.
- **Filtering** of the data content of the event object schema to apply control, distribution compliance and other policies.
- **Transformation** of the notification data to fit the schema of the channel.

- **Operational analytics** to improve efficiency of operations.
- **Optimization and scaling** to support the varying traffic density of certain event types through intelligent variable clustering.
- **Tracking and billing** for monetizing public access to event notifications.
- **Resilience** by supporting high-availability and disaster recovery.
- **Contextualization** by including historic and other relevant data in directing the notifications.
- **Custom processing** by supporting programmable logic add-ons.

ESP (stream analytics) engines include Apache Spark Streaming, Azure Stream Analytics, IBM Streams, TIBCO StreamBase, Software AG Apama, Kafka KSQL and others. They receive an ordered stream of event notifications from event brokers or directly from event sources, apply analysis on a set of notifications and, if a certain pattern is identified, form an event notification that is directed to an event broker. This is then distributed to appropriate event handlers. Stream analytics technology and architecture are reviewed in detail in these Gartner publications: "Technology Insight for Event Streaming Processing" and "Market Guide for Event Stream Processing."

Benefits and Uses

The role of an event broker is central to facilitating event-driven computing in digital business. In that sense, event brokers carry the same use cases and benefits as does event-driven architecture and event-driven application design.

The key use cases and their benefits include support of:

- **IoT applications —** IoT applications are driven by the interaction of real-world devices (things) and the organization's IT. The device sensors communicate with the IT environment by generating a continuous stream of events. Event brokers help organize and manage the processing of the IoT event streams and, therefore, facilitate and accelerate operations of IoT applications.
- **Data Integration and reconciliation —** Often, synchronization of redundant or dependent data held in multiple data stores by different applications is time-sensitive and cannot wait for a delayed batch update. Event-based notifications of change in one application's data store, which are published to dependent applications, allows cross-application data synchronization in near real time.
- **Application-as-product extensibility —** This is a highly valued characteristic in digital business. This is especially true in the transition from the project to product model of the application life cycle. Event brokers facilitate the decoupling of application components and allow for the addition, replacement or removal of certain functionality with minimal impact on the rest of the running application. Thus, new capabilities can be added to a running application gradually, without disturbing the existing use, as required in the application-as-product model.

- **Continuous intelligence —** Continuous intelligence is a value directly experienced by business consumers of IT services. Continuously refreshed awareness of the state of the organization's business, its ecosystem, its customers and the open world, typically depends on monitoring event streams from various sources, then applying predictive and prescriptive analytics to help the business with its proactive decision making.
- **Serverless functions** are principally designed to be triggered by events and state changes in data, apps, applications and other resources. The increasingly demanding use of serverless functions requires an elevated level of management, and event broker technology is where central management oversight is implemented.
- **Stream analytics** does not require the use of an event broker, However, a combination of event stream processing and an event broker elevates the use cases and design options for both. An event stream can be directed by an event broker to multiple subscribing analytics engines. Once the analytics work identifies a notable complex event, a notification is sent to an event broker to target the subscribing event handlers for all the necessary response work.

Adoption Rate

The role of an event broker has been played by message-oriented and other middleware for decades, though the majority of the mainstream use of messaging middleware was for message passing or temporary state storing, not event processing. Most products that are specifically dedicated to event processing started appearing with complex event processing (CEP) software like Esper and TIBCO StreamBase. Today, vendors like Confluent, Push Technology, Streamlio, Solace and VANTIQ are offering features designed specifically for event brokering, such as persistent ledgers; topic schema control; time, ordering and other metadata; event filtering and topic access control.

Adoption of event-driven design is on the rise, driven by the demands of digital business, such as IoT, continuous intelligence, situation awareness and global scale. Adoption of event brokers will continue to track the adoption of event-driven computing. Industry verticals such as ground, air and sea transportation, retail, marketing and advertisement, manufacturing, construction and insurance lead the way.

Tracking the adoption of event brokers by revenue is complicated by the fact that most on-premises event brokering technology today is largely open-source (RabbitMQ, Apache Kafka, Apache Pulsar). In the cloud, of course, every use of event brokers generates subscription revenue. Therefore, the revenue numbers for cloud event broker services are likely to better reflect the rates of adoption of event brokers than the same on-premises.

Risks

The choice of basic pub-sub versus advanced event broker carries a risk on both sides. The mature pub-sub technologies used for event brokering, such as IBM MQ, TIBCO FTL, RabbitMQ, Software AG Universal Messaging or implementations of JMS, are well-established and proven in mission-critical computing. However, the more recent event brokering technologies, such as Confluent and Streamlio, are a better fit for the digital business use cases, but are new, less mature

and less proven. Customers face the dilemma of choosing proven technology that lacks some modern capabilities or a modern alternative that may lack implementation maturity. Also, enterprises may be familiar with best practices for existing pub-sub middleware, but those lessons don't entirely translate to newer, more-advanced event brokers. In many cases, enterprises are starting almost from scratch when it comes to scaling and managing this new generation of technology.

Multiple risks flow from the larger decision to adopt an event-driven design model.

The risk of unintended consequences. Because the sources and the event handlers in EDA are unaware of each other, the source cannot control — or even know with complete certainty — what consequences will follow the notification. For a simple source-handler pair, this challenge is mitigated by event brokers' management tools. But the nature of event-driven design is such that an event handler can turn around and be a source to a different handler and that can repeat the pattern again. Tracking the third, fourth and beyond ripple effect of the consequence of a single notification becomes exceedingly difficult without advanced pattern tracking tools.

The risk of misdirection. As said earlier, a key characteristic of event-driven computing facilitated by event brokers is the diffusion of transaction control. In traditional request-driven SOA design, the caller directs its request directly to the service by name and retains control until the service completes or fails. In EDA, the source passes control to the event broker before the action is even initiated. Most of the event-driven designs are applied to manage observational or some transactional activity (queries are typically implemented via request/reply SOA). Observations do not pose a transaction control challenge, but commands do. When a command is expressed as a notification, it carries an expectation for a certain action (a deposit event would anticipate an update to an account record). Choosing an event notification to convey a command delegates to the event broker the responsibility to preserve the integrity of the application. The broker may redirect the action by design, which is a high-value, advanced flexibility characteristic of event-driven computing, but it may also fail to operate properly and misdirect notifications because of developer error or malicious interference. This vulnerability can endanger the integrity of the transaction and the application.

The risk of eventual consistency. Because the source and the handler of the event notification are not directly connected (and, potentially, are not even active at the same time), the transactional content of the two cannot be synchronized, thus delivering, at best, the optimistic assumption of eventual consistency. For example, the notification of a book order received in an e-commerce app will be conveyed by the event broker to the event handler, which will then take the ordered book out of the inventory. If the latter task fails (no available inventory), the acceptance of the order cannot be rolled back. The expected eventual consistency fails, the data in the two applications is out of sync and additional processes must be engaged to mitigate the conflict.

The protocol risk. Unlike the ubiquitous use of REST for request/reply SOA, there is no standard protocol for communicating or encoding event notifications.

A variety of protocols, from AMQP 0-9-1 through MQTT, JMS, STOMP, HTTP Long Polling, CometD/Bayeux, SignalR, HTTP Streaming, HTML 5 Server-Sent Events, WebSockets and Kafka's client protocol (binary over TCP), is an incomplete list of available options. Excessive lock-in with one

selected protocol may require a redesign in the future if the chosen protocol fails to advance; on the other hand, supporting multiple protocols increases the complexity and cost of operations.

Encoding of the topic/event notification schema has also not been standardized, although some attempts at standardization are underway (AsyncAPI, CloudEvents). Most custom designs for defining the data and metadata of an event notification will need to be redesigned once a standard emerges. Meanwhile, all communicating sides must agree on one shared encoding policy — a burden that reduces the decoupling effect of event-driven design.

The risk of a defensive posture. Some organizations lack the skills and technology know-how for event-driven design. The integrity, complexity and maturity risks can also compel organizations to delay adoption of event-driven design. However, avoiding the immediate, tactical risks associated with new learning itself represents a larger, strategic risk of falling behind in digital business transformation and, with that, limiting an organization's ability to meet the demands for innovation from its business and customers.

Event Broker Alternatives

Event processing without EDA. Real-world events can be reported by sensors, applications or people. Reactions to the events can be custom programmed, invoked directly via a request/reply API, or the event information can be simply saved in a data store for later processing. This event processing does not require the use of an event broker because it does not follow the event-driven architecture (EDA) principles of isolating the sources and the handlers of event notifications.

Event processing in APIM. The primary alternative to EDA is the traditional request/reply SOA, in which case, the role of an intermediary may be played by an APIM, such as Apigee or MuleSoft. Future APIM offerings may incorporate some of the event brokering responsibilities as well, but today such integrated support is rare.

Spectrum of pub-sub technologies. To implement EDA, designers can choose any form of pub-sub middleware. ESBs like MuleSoft or TIBCO, BPM suites like Pega and Appian, MOMs like RabbitMQ and IBM MQ, and Java EE application servers like Oracle WebLogic and IBM WebSphere, all include pub-sub capability and can be used to facilitate EDA-style applications. Most organizations have already deployed many of these alternative technologies and, therefore, may choose one or several of them for tactical event-driven initiatives. However, as said earlier, none of these product categories are focused specifically and directly on event processing. They support the basic needs of event brokering, but lack dedicated tools for designing and monitoring event semantics, especially considering the scale and functional demands of event-driven computing in digital business.

Recommendations

Application leaders engaged in modernizing their application architecture and infrastructure should:

- Drive organization to become competent in the skills and technology required for event-driven design to enable application designers to freely draw on traditional request-driven SOA or EDA capabilities as the business requires.
- Choose advanced event brokers for digital business innovation initiatives, because of their specialized support of event processing, even if alternative pub-sub technology, such as MOMs or ESBs, is already used elsewhere for some isolated application scenarios.
- Include advanced event brokers in the suite of offerings that constitutes your HIP to support strategic event-driven solutions in your multiarchitecture application environment.
- Adopt event brokers with the understanding that the increasing maturity and standardization of technology over time will likely lead you to make changes to technology and its use patterns.

Representative Providers

Advanced:

- Oracle Event Hub Cloud Service
- Microsoft Azure Event Grid
- IBM Event Hub
- Confluent Cloud
- Solace PubSub+
- Vantiq Application Platform
- Serverless' Event Gateway
- SAP Project Kyma
- AWS Simple Notification Service (SNS)
- RabbitMQ
- Apache ActiveMQ
- IBM MQ
- TIBCO FTL

Additional research by Gary Olliffe and Kevin Matheny

Gartner Recommended Reading

Some documents may not be available as part of your current Gartner subscription.

"Innovation Insight for Event Thinking"

"Gartner on Event Processing in Digital Business: Recent Research"

"CTO Alert: Master Event-Driven IT to Master Digital Business"

"CIO Challenge: Adopt Event-Centric IT for Digital Business Success"

"Follow the Leaders: Digital Business Innovation Is Event-Driven"

"Articulating the Business Value of Event-Driven Architecture"

"You're Already Capturing Event-Driven Digital Business Opportunities"

"Assessing Event-Driven Middleware Technology for Modern Application Architecture"

GARTNER HEADQUARTERS

Corporate Headquarters
56 Top Gallant Road
Stamford, CT 06902-7700
USA
+1 203 964 0096

Regional Headquarters
AUSTRALIA
BRAZIL
JAPAN
UNITED KINGDOM

For a complete list of worldwide locations,
visit http://www.gartner.com/technology/about.jsp