WHITEPAPER

A COMPREHENSIVE GUIDE TO EVENT-DRIVEN SYSTEMS

PubNub

Introduction

Delivering a true real-time digital experience is a must-have for modern app users. But with expectations around minimizing downtime and managing real-time "events" such as IoT device control, pricing/shipping updates, geolocation, and more, it can be challenging for organizations to avoid common pitfalls when building these types of event-driven systems.

Event-driven architecture (EDA) is a design pattern with loosely coupled components used to trigger and communicate events. It has become widely used in technology where real-time interactions are taking place because its benefits include:

- Reduced latency
- Effortless scaling
- Reliable operational efficiency

This white paper will help teams gain a better understanding of the EDA landscape and provide insight into how developers can leverage this architecture for deploying and scaling their own event-driven applications.

What is event-driven architecture?

Event-driven architecture (EDA) [1] is a software design pattern used in microservices or other decoupled services or apps. Each service in the ecosystem can asynchronously publish and subscribe to events via an event broker.

An **event** is any change in state or update that happens within the ecosystem. Examples of events include customer requests, inventory updates, sensor readings, and so on. The event broker acts as a middleman allowing a "loose coupling" between applications to transport events to and from event consumers.

NOTE: *An event is distinct from a message. A message is data transmitted to a specific address, whereas an event is data broadcasted as it happens and can be consumed by any service listening for it.*

EDA provides a flexible, scalable, and real-time approach to processing actions and responses quickly. It is ideal for managing high volumes and high-velocity of data in real time with a minimum time lag. It can handle complex event processing, such as pattern matching or aggregation over time windows.

Advantages of event-driven architecture [2]

Less Impact Failure

A failure at any one service will not impact other services since event-driven architecture uses decoupled applications. The event broker acts as the conduit between services, storing published events until a consumer calls for them. The event broker also manages any surges in published events.

Independently Build & Deploy Applications

The event broker speeds up your development process because both event producers and consumers don't need to work in tandem. The event broker stores the event and delivers it once the system comes back online. Since applications are loosely coupled, you don't need custom code to poll, filter, and route events. The event broker automatically filters and pushes events to consumers.

Audit With Ease

Auditing is easier and more reliable since every change in the system is stored as an event. The event broker creates an accurate record of every change, acting as your centralized command center deciding who can publish events, subscribe to the broker, and access your data.

Reduced Costs

EDA uses a push-based system so everything happens only when the event producer pushes an event to the event broker. This means the system is not polling periodically to check for an event, consuming less network bandwidth, less computing power, and less idle fleet capacity.

Disadvantages of event-driven architecture [3]

More Complexity

Event-driven architecture deals with a large ecosystem of event producers and event consumers transmitting events across different microservices. Managing this complex network can be a daunting task for any developer. They have to consider many factors like event lifecycles, event discovery, and ensuring the code works across systems, just to name a few.

Troubleshooting Challenges

Debugging and testing require more resources and time compared to traditional architecture. Tracing an event from its producer to its consumer can be a difficult task because of the distributed and decoupled nature of event-driven architecture.

Monitoring Difficulties

Event-driven architecture requires a proper schema, so developers can understand exactly how distributed, decoupled components interact with each other. This needs a comprehensive set of monitoring tools and alerting mechanism that provides developers with a complete view of the event flow.

To ensure success, consider the following when developing an event-driven architecture [1]:

- Event source durability: Your event source should be reliable. It must guarantee delivery if you need to process every single event.
- Performance control requirements: Your application should be able to handle the asynchronous nature of event routers.
- Event flow tracking: Focus on dynamic tracking via monitoring services, and avoid static tracking techniques using code analysis.
- Event source data: If you need to rebuild the state, your event source should be deduplicated and ordered.

Event-driven architecture vs. Traditional architecture

Traditional Architecture

Traditional architecture follows a request-response pattern where a server transmits a query and then waits for a response. This is great when working with static data but not convenient for real-time processing where data keeps changing.

Some key characteristics of traditional architecture:

- The service should be online when pull requests are made or the request will go unfulfilled.
- Making changes to the system is complicated and time-consuming.
- Any data changes erase previous information about state changes.
- It is unable to record changes from multiple sources simultaneously.
- It supports synchronized communication and recording of events, ensuring the information is consistent.

Event-Driven Architecture (EDA)

EDA functions in real time, so all event consumers subscribed to events receive them as they are produced by the producers. Some key characteristics of EDA:

- The system does not need to know consumers are online for events to be published.
- Changes can be made to a specific component without impacting the entire process flow.
- The system records and retains the history of every event.
- EDA can record data while executing processes, allowing continuous intelligence in a computer system.
- EDA does not follow a chronological alignment of timelines, making it difficult to synchronize information.

Event-driven architecture use cases

Multi-User Collaboration

Event-driven architecture can detect the online connectivity of a user or a device and update the state of the system accordingly. Using EDA, multiple users can interact simultaneously in digital workspaces, virtual classrooms, or team collaboration settings.

Some example scenarios include triggering actions like push notifications for when users are online, and gathering and sending real-time data such as quiz results and populating that to a leaderboard.

Marketplaces & Auctions [4]

Event-driven architecture gives marketplaces & auctions organizations the freedom to enlist different vendors for different services like order validation, stock management, and payment processing. Services produce and consume data for new events, which are then sent to an event broker. The event broker pushes each new event to the relevant services automatically.

Because it uses decoupled applications, event-driven architecture also gives organizations the flexibility to introduce changes to the system. For example, if the organization is introducing new suppliers or distributors, only that specific process will be changed. The remaining operation will continue to function, causing minimal or no interruption to the overall customer experience.

IoT Device Control [5]

Event-driven architecture can manage and process data from multiple sources in real time, making it an ideal system to run IoT devices. It can publish, receive, and process a wide range of IoT commands simultaneously, which allows the system to trigger the appropriate response in a timely manner without repeatedly asking for information through polls.

For example, if a sensor picks up a change in the state (like a change in temperature, location, or volume), the system can trigger the appropriate response. In this case, it would be notifying the appropriate personnel. Moreover, developers can modify existing code and introduce new processes without impacting the processes that are currently running.

Logistics and Transportation

Event-driven architecture is ideal for tracking shipments, monitoring inventory, and providing real-time updates to both the customer and logistics company. The logistics industry manages a large number of individual devices across a wide region. With EDA, developers can build an event mesh across all the services and platforms, allowing better synchronization between different parts of the organization to serve and collect data across multiple web and mobile apps.

Also, as the amount of data invariably increases, the organization can scale without losing previous information or disrupting the current workflows using a more agile approach to development.

Components of event-driven architecture

Event Producers

An event producer creates and publishes events. They are also known as event emitters or event agents. The event producer does not know the event consumer, and vice versa. Event producers do not need to belong to the same software ecosystem as the event consumer. An example of an event producer is the front end of a website, IoT devices, SaaS applications, etc.

Event Consumers

Event consumers, also known as event sinks, listen for specific events. When they receive a notification that the event has occurred, they can perform actions like starting a workflow, running an analysis, and updating a database. They are a downstream component that comes into play after the event broker receives the event from the event consumer.

Multiple consumers can use the same events. Applications can act as both event producers and consumers and do not need to belong to the same software ecosystem as the event producer. Examples of event consumers include service endpoints, message queues, databases, event processors, or another microservice.

Event Brokers

Event brokers act as the intermediary between event producers and event consumers. They enable asynchronous messaging between the two parties, i.e., allow producers and consumers to send messages to each other without waiting for a response. They also allow different systems to send messages to each other, improving interoperability between systems.

Event brokers support both push and pull consumption models. In the push consumption model, consumers subscribe to specific events and the event broker pushes the appropriate events to the consumer. In the pull consumption model, consumers request events from the event broker. Apache Kafka, RabbitMQ, or AWS EventBridge are examples of event brokers.

Event Schema [6]

An event schema shows the structure of an event and shares information about the event's data. For example, when you place an order on an e-commerce website, an "OrderCreated" event is generated with information like product ID, quantity, and shipping address. It can also include information like the data type — for example, "number of items" is an integer, "shopping address" is a text string, etc.

Event schemas provide consistency in the way events are produced and consumed, which is important to determine how events are routed and processed within the system. They are stored in a common location called the schema registry. When creating event schemas, you can create sparse events with little data about the event or an event with a full-state description like the example above.

Event Processing [7]

Event processing is the capture, enrichment, formatting, and release of events, the subsequent routing and processing of released events, and the consumption of the processed events. There are four common styles of event processing:

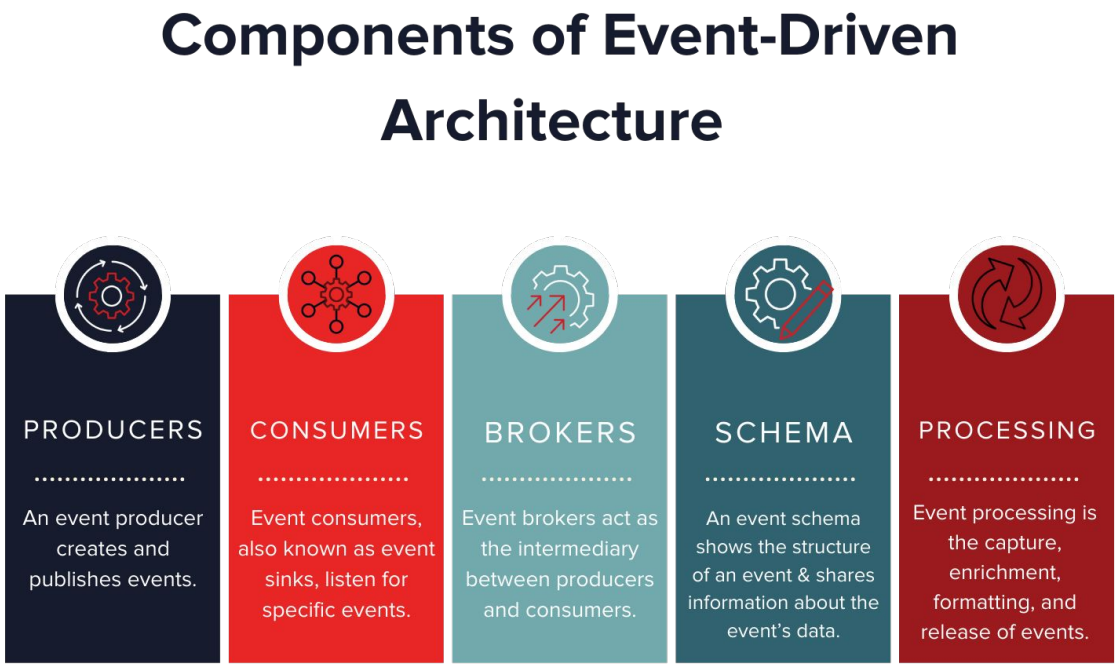
1. **Simple event processing (SEP):** Simple event processing is commonly used to coordinate real-time workflows when a notable state change occurs, significantly reducing latency and costs. Simple events occur separately from other events. The final event message may contain data required to initiate a response.

2. **Event stream processing (ESP):** This drives the real-time consumption of information around the enterprise by processing event data streams to identify relationships and patterns, and support timely decision-making. Stream event processing is ideal for IoT workloads where decisions need to be made instantly.

3. **Complex event processing (CEP):** This type of event processing is used when multiple events have to occur before any action takes place. No further action will be taken unless all the criteria are met. Complex event processing usually uses far more advanced language interpreters, software pattern definitions, and correlations. It is used mostly to identify business opportunities, threats, and anomalies.

4. **Publish/Subscribe processing (Pub/Sub):** When an event is published, it is sent to all event consumers who have subscribed to the event. Once the event is received, it cannot be replayed and no new subscriber can view the event.

Event-driven architecture relies heavily on event processes to facilitate communication and cooperation between different applications that may not be compatible otherwise. Event processes enable swift reactions to singular events, combinations of multiple events, and targeted observations of these events.



Implementing event-driven architecture

Choosing an Event Broker [8]

Finding the right event broker for your system is important since it defines the architecture, design, and operations of your applications. When choosing your event broker, first assess its functional and operational features to find the right one for your application.

Event broker capabilities and features can be divided into the following broad categories:

- **Client connectivity features** like protocol support, transaction support, and software development and integration features regulate the way event producers and consumers connect to the event broker and process event messages.
- **Message delivery features** establish relationships between event messages and event brokers. These features include message structure, topic organization, topic partitioning, metadata and message persistence semantics, routing, subscriptions, and quality of service.
- **Broker deployment features** are the features enabled to satisfy a number of non-functional tasks, like throughput, resilience, latency, and recoverability, while deploying and configuring event brokers.
- **Management and operations capabilities** are the features available for managing, monitoring, and securing the event broker system and event-driven applications like application programming interfaces (APIs), command-line interfaces (CLI), and user interfaces (UIs).

The categories are further subdivided into a large number of attributes and conditions that influence the application design and its resulting capabilities. When selecting the event broker, consider the organization's requirements, the capabilities of individual brokers, and where you're willing to meet halfway.

Event brokers are divided into three types based on their functions and capabilities:

- **Queue-oriented event brokers**, when you need multiprotocol client connectivity and flexible topic structure definition, without message retention and replay.
- **Log-oriented event brokers**, when you need longer message retention and replay and/or very high throughput and can tolerate a flat topic structure.
- **Subscription-oriented event brokers** are perfect for cloud-based event-driven applications hosted on a single cloud provider's environment. Note that this event broker only works if you are willing to design your application around the unordered, at-least-once delivery model.

As you can see, not all event brokers are the same. It is important to clearly understand their individual capabilities before selecting one. Here are a few things to consider:

- Keep the event broker configurations simple.
- Consider the main features when selecting the broker type.
- Don't be afraid to use multiple event brokers if required.
- Use well-defined schema management.

Defining the Event Schema

An event schema [8] ensures that certain events follow a predetermined behavior to allow for consistency throughout the system. To create an event schema:

1. **Identify the type of event** you want to define. They can be classified into different types like system events, business events, or user events.
2. **Define the event properties** that the event will contain. This includes both mandatory and optional properties. They should be specific and should capture all the important information that needs to be transmitted.
3. **Specify the data format** the event will be transmitted in, and ensure it is supported by all the components of the system. You can use various formats like JSON, XML, or binary formats.
4. **Define the event schema versioning strategy.** This helps maintain backward compatibility when any changes are made to the event schema. The strategy should cover how you will version the event schema, handle its evolution, and ensure different versions of the event schema are compatible.
5. **Share the event schema** with all the relevant components to ensure they understand the event structure and how to process it.
6. **Test the event schema** with various use cases, and confirm the event schema works for different scenarios. Verify that all components can process the event correctly.

Designing Event Producers & Consumers [10]

Event producers and consumers are designed to fulfill specific requirements:

- **Event producers** generate and publish events to the event broker.
- **Event consumers** subscribe to or receive events as they are published by the event producer.

System requirements like the scale, performance, and reliability of the event-driven architecture can also influence the design of event producers and event consumers.

Best practices when designing an event producer

- Choose a format that is efficient, easy to parse, and able to support the types of events being produced. Some possible candidates are JSON, XML, or Avro.
- Every event should contain all the pertinent information consumers require to process events independent of any external dependencies, including any metadata or contextual information.
- To improve reliability, design events in such a way that they can be processed multiple times without any problems.

Best practices when designing an event consumer

- Consider the specific requirements of your system when choosing an event processing model like the rate and volume of events, and the time required by the consumer to process them.
- Implement fail-safes like resending events so that your consumer can handle the workload if any component fails to perform its defined action.
- Ensure consumers can function simultaneously. This significantly improves the performance and scalability of the event-driven architecture.

Deploying & Scaling the Event-Driven System

The deployment process in an event-driven architecture should always remain consistent between services to ensure microservices continue to function seamlessly. Microservices testing and deployment should be carried out at the discretion of the teams.

Here are a few considerations to keep in mind during deployment:

- Reset event consumer offsets.
- Purge event stores.
- Review and update event schemas.
- Delete event streams.

There are tools available to perform the above functions, enabling further automation and supporting team autonomy.

Best practices for event-driven design patterns [11]

1. **Open communication between developers and architects**
   This is non-negotiable when creating an effective event-driven architecture. Both parties bring different perspectives about event opportunities and how to incorporate them into the process flow.

2. **Prepare for scaling the system**
   Keep scalability in mind when building the event architecture. This will enable reactive and flexible processes and rapid changes to how activities flow from one component to another.

3. **Apply API best practices like taxonomies, portfolios, and portals**
   Taxonomies create an event hierarchy subscribed to by one or more event consumers. Portfolios identify and organize events within business domains. An event portal creates a space for event producers and consumers to connect and collaborate on event usage policy and subscriptions to event streams.

4. **Design the event architecture to span across multiple clouds**
   Carefully consider which event technologies to use and how to connect them across different cloud infrastructures. In this way, the event architecture can span across multiple clouds and technologies.

5. **Define quality of service (QOS) and contextual guidelines**
   When implementing event producers or event consumers, define QOS guidelines (like where to have delivery guarantees, sequence preservation, and event persistence) and contextual guidelines (like which application acts as the event producer or event consumer).

Best practices for testing & monitoring event-driven architecture

1. **Event throughput**
Continuously monitor the number of events produced, processed, and transmitted through the system, and the time taken to perform these actions. This way, detecting and responding to any issues becomes easier.

2. **Event processing latency**
Monitor the processing time of events at specific stages of the event processing pipeline, or during the entire process, to ensure minimal latency and operational efficiency.

3. **Event error rate**
Monitor the error rate to ensure the event workflow is working reliably by tracking the number of events that fail to be processed or are rejected by event consumers.

4. **Log events and metadata**
Maintain a record of events as they are produced and processed, along with any metadata or contextual information associated with the events.

5. **Create alerts**
Set up alerts to detect and inform stakeholders about problems in the workflow like high error rates, high latency, or low throughput.

Best practices for performance optimization in event-driven architecture [12]

Use performance modeling

- Helps developers build scalable deployment models
- Makes architectural and design optimizations to reduce latency
- Design tests to validate performance and throughput

Leverage caching

- Caches create a faster data layer for data look-ups, reducing the cost of sending and receiving database queries
- The event stream data pipeline updates the cache almost immediately and allows event processors to access the data instead of querying the database or making service calls to systems of record every time

Collocate processing

- Event-driven architecture performs best when event producers, event consumers, and event brokers are collocated
- The only downside is the entire application will go down if the data center goes down. You can avoid this by setting up mirrors for the event architecture

IMPORTANT SECURITY CONSIDERATIONS FOR EDA

Similar to point-to-point architecture, restricting services to only produce or consume events related to specific queues helps protect important data in an event-driven architecture. Set these restrictions by providing very clear guidelines about which services can write and consume events.

Case studies and real world examples for event-driven architecture

Logistics [13]

An innovative supply chain application utilizes EDA for business-critical operations in the airline and airport fueling industry. Developers built a reliable, end-to-end solution that airports could trust to provide accurate, time-sensitive notifications and status updates about fueling, catering, and other operations that minimize turnaround time and verify that flights will depart on time worldwide.

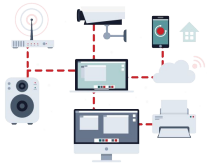
Since the transition, they send over 4 million mission-critical messages each day and leverage live tracking across airports globally to ensure planes are properly fueled and on time.

Transport & Delivery [14]

A leading delivery service in India connects restaurants, customers, and delivery partners throughout the order and delivery process.

With the real-time capabilities of this architecture, the platform handles service requests through a chatbot to agent handover, which has allowed them to automate up to 70% of their customer support tickets and reduce their ticket resolution times from five minutes to 30 seconds.

IT Software



An IT management software company uses EDA implementation in a large-scale system to monitor the online/offline status of 2.5 million devices.

By leveraging an event-driven infrastructure, their customers can quickly and securely receive immediate updates on state changes of devices and machines through triggers on the server-side events via Webhooks.

IoT (Internet of Things)

A smart lock company operating out of Japan uses event-driven systems for transferring commands to a device. The app allows users to control their keys from anywhere using a smartphone, providing an efficient and modern way to enhance home security.

The change to this architecture allowed for quicker implementation and saved them almost one year of development time.

Conclusion

Fundamentally, event-driven architecture allows developers to create real-time applications that are more scalable, flexible, and responsive. By following the best practices and staying up-to-date on the latest frameworks, developers can successfully integrate EDA into their workflow and create high-performing, modern applications.

At PubNub, we enable developers to build real-time interactivity for IoT devices, web apps, and mobile apps. The PubNub platform runs on our real-time edge messaging network, providing our customers with the industry's largest and most scalable global infrastructure for interactive applications.

With over 15 points-of-presence worldwide supporting 800 million monthly active users, and 99.999% reliability, you'll never have to worry about outages, concurrency limits, or any latency issues caused by traffic spikes.

Try for free!

SUPPORT WEBSITE LINKEDIN TALK TO OUR TEAM

Event-Driven Architecture Checklist

Integrating EDA requires careful planning and execution. For organizations that want to build real-time apps that are more scalable, have a high level of resilience and extremely low latency, you'll want to ensure that you're choosing the right provider who can support your build both now and in the future.

- [ ] Speeds up your development for quicker and easier deployment.
- [ ] Captures and reacts to critical events generated by users in real time.
- [ ] Maintains the security and privacy of event data.
- [ ] Ensures compatibility with existing software systems.
- [ ] Creates and executes business logic from language translation to moderation, IoT sensor data aggregation, geo-triggering, and more.
- [ ] Offers unlimited scalability to handle any number of concurrent users and devices from anywhere without financial penalties.
- [ ] Reduces operational costs because the system consumes less network bandwidth and less computing power.
- [ ] Runs on its own rock-solid architecture via worldwide data centers with 99.999% SLA, freeing you from reliability and latency worries.
- [ ] Enables teams to build the real-time capabilities users are demanding and leaves the door open for adding additional features and functionality in the future.

For more information about how PubNub can help you build and operate real-time interactivity, *sign up for a free trial* today.

References

1. https://aws.amazon.com/event-driven-architecture/
2. https://d1.awsstatic.com/psc-digital/2022/gc-300/event-driven-architectures-aws-guide/Event-Driven-Architectures-AWS-Guide.pdf
3. https://www.techtarget.com/searchapparchitecture/tip/Event-driven-architecture-pros-and-cons-Is-EDA-worth-it
4. https://www.emporix.com/blog/event-driven-architecture-for-ecommerce
5. https://aws.amazon.com/blogs/architecture/building-event-driven-architectures-with-iot-sensor-data
6. https://pandaquests.medium.com/defining-an-event-schema-in-an-event-driven-architecture-4fc2d011a201
7. https://complexevents.com/wp-content/uploads/2006/07/OMG-EDA-bda2-2-06cc.pdf
8. https://solace.com/blog/gartner-how-to-choose-an-event-broker/#gartner-source
9. https://levelup.gitconnected.com/designing-a-reliable-event-driven-architecture-for-handling-large-volumes-of-events-18e617ecd818
10. https://reprints2.forrester.com/#/assets/2/192/RES161456/report
11. https://link.springer.com/chapter/10.1007/978-0-387-35586-3_11
12. https://www.redhat.com/architect/apache-kafka-EDA-performance
13. https://www.confluent.io/events/kafka-summit-europe-2021/achieving-real-time-analytics-at-hermes/
14. https://www.slideshare.net/Pivotal/iot-scale-eventstream-processing-for-connected-fleet-at-penske