Seetharamugn Follow 8 min read · Aug 29, 2023

The Complete Guide to Event-Driven Architecture

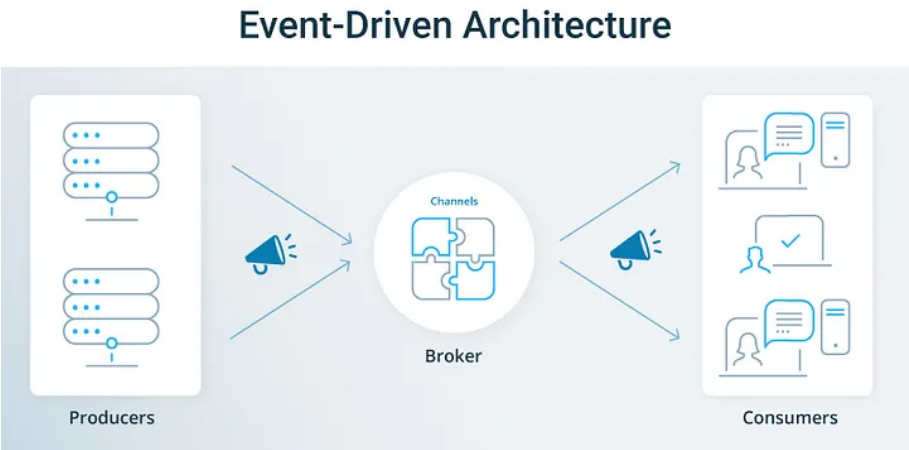


Fig 1. Event-Driven Architecture

What is Event-Driven Architecture?

Seetharamugn highlighted

Event-driven architecture is a software design pattern that allows decoupled applications to asynchronously publish and subscribe to events through an event broker (modern messaging-oriented middleware).

Event-driven architecture is a method of developing enterprise IT systems that allows information to flow in real time between applications, microservices, and connected devices as events occur throughout the business.

The event-driven architecture enables loose coupling of applications by introducing a middleman known as an event broker. This means that applications and devices do not need to know where they are sending information or where the information they are consuming comes from.

Seetharamugn highlighted

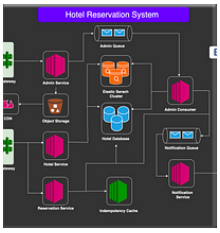
What does the term "event" mean in EDA?

A change in state is called an event. In an event-driven architecture, everything that happens within and to your enterprise is an event — customer requests, inventory updates, sensor readings, and so on.

Designing a Scalable and Resilient Hotel Reservation System Architecture

fig. Hotel Reservation System.

medium.com



Why should you use an event-driven architecture?

The value of knowing about a given event and being able to react to it degrades over time. The more quickly you can get information about events where they need to be, the more effectively your business can react to opportunities to delight a customer, shift production, and reallocate resources.

That's why event-driven architecture, which pushes information as events happen, is a better architectural approach than waiting for systems to periodically poll for updates, as is the case with the API-led approach most companies take today.

Event-driven architecture ensures that when an event occurs, information about that event is sent to all of the systems and people that need it. It's a simple concept, but these events have had quite the journey. They need to efficiently move through a multitude of applications written in many different languages, using different APIs, leveraging different protocols, and arriving at many endpoints such as applications, analytics engines, and user interfaces.

If they don't get there? Connected "things" can't connect, the applications and systems you rely on fail, and people across your company can't react to situations that need their attention.

The Advantages of Event-Driven Architecture

The main advantages of event-driven architecture are improved responsiveness, scalability, and agility in business. The ability to respond to real-time information and quickly and easily add new services and analytics significantly improves business processes and the customer experience. Despite the fact that modernizing IT infrastructure can be costly,

With event-driven architecture, you can do the following:

- Everything happens as soon as possible, and nothing is waiting on anything else.
- You don't have to consider what's happening downstream, so you can add service instances to scale.
- Topic routing and filtering can divide up services quickly and easily, as in command query responsibility segregation.
- To add another service, you can just have it subscribe to an event and have it generate new events of its own; there's no impact on existing

services.

Event-Driven Architecture Use Cases

The benefits of event-driven architecture covered above are especially relevant in use cases where a single change can have huge consequences, rippling all the way down the chain. One of the most common questions people have is, “When should you use event-driven architecture?” The answer lies in what you are trying to accomplish with your data.

Businesses looking to take advantage of real-time data in their daily activities are turning to event-driven architecture as the backbone for use cases that can benefit the most. According to a 2021 survey, the top 4 use cases for event-driven architecture were:

- Integrating applications
- Sharing and democratizing data across applications
- Connecting IoT devices for data ingestion and analytics
- Event-enabling microservices

Examples of Event-Driven Architecture

The value of event-driven architecture transcends industries and can be applied to small businesses as well as large multinational corporations. Retailers and banks can use it to aggregate data from point-of-sale systems and across distribution networks to execute promotions, optimize inventory, and offer excellent customer service.

Retail and eCommerce: An Example of Event-Driven Architecture

This is an example of event-driven architecture from a retail perspective. Notice that no systems (inventory, finance, or customer support) are polling to ask if there are any new events; they are simply filtered and routed in real-time to the services and applications that have registered their interest.

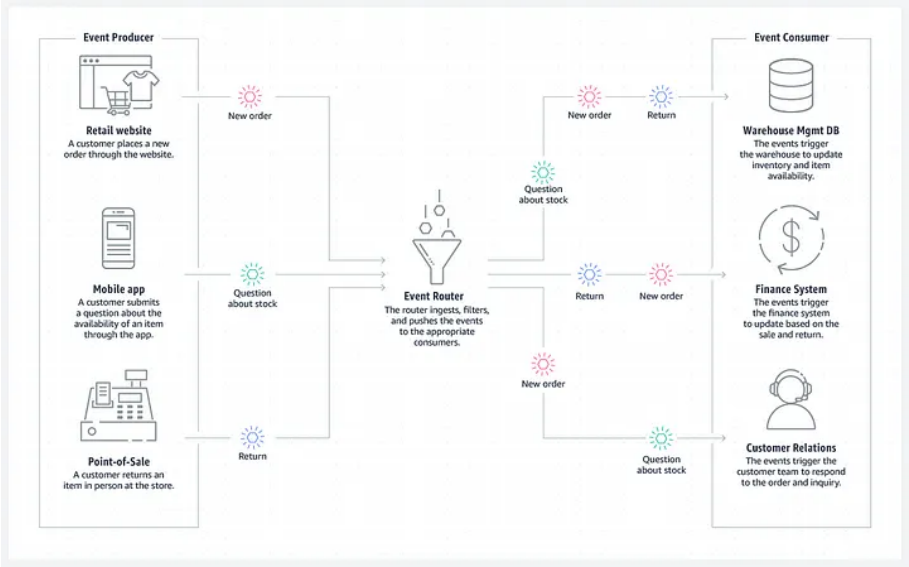


Fig 2. eCommerce Event-Driven Architecture.

Common Event-Driven Architecture Concepts That You Should Know

Top highlight

There are eight key architectural concepts that need to be understood for event-driven architecture to be successful:

1. Event broker
2. Event portal
3. Topics
4. Event mesh
5. Deferred execution
6. Eventual consistency
7. Choreography
8. Command Query: Responsibility Segregation

1. Event Broker

An **event Broker** is a middleware (which can be software, an appliance, or SaaS) that routes events between systems using the publish-subscribe messaging pattern. All applications connect to the event broker, which is responsible for accepting events from senders and delivering them to all systems subscribed to receive them.

It takes good system design and governance to ensure that events end up where they are needed and effective communication between those sending events and those who need to respond. This is where tooling—such as an **event portal**—can help capture, communicate, document, and govern event-driven architecture.

2. Event Portal

As organizations look to adopt an event-driven architecture, many are finding it difficult to document the design process and understand the impacts of changes to the system. **Event portals** let people design, create, discover, catalog, share, visualize, secure, and manage events and event-driven applications. Event portals serve three primary audiences:

1. Architects use an event portal to define, discuss, and review events, data definitions, and application relationships.
2. Developers use an event portal to discover, understand, and reuse events across applications, lines of business, and between external organizations.
3. Data scientists use an event portal to understand event-driven data and discover new insights by combining events.

3. Topics

Events are tagged with metadata that describes the event, called a "topic." A topic is a hierarchical text string that describes what's in the event. Publishers just need to know what topic to send an event to, and the event broker takes care of delivery to systems that need it. Application users can register their interest in events related to a given topic by *subscribing* to that topic. They can use wildcards to subscribe to a group of topics that have similar topic strings. By using the correct topic taxonomy and subscriptions, you can fulfill two rules of event-driven architecture:

1. A subscriber should subscribe only to the events it needs. The subscription should do the filtering, not the business logic.
2. A publisher should only send an event once, to one topic, and the event broker should distribute it to any number of recipients.

4. Event Mesh

An event mesh is created and enabled through a network of interconnected event brokers. It's a configurable and dynamic infrastructure layer for distributing events among decoupled applications, cloud services, and devices by dynamically routing events to any application, no matter where these applications are deployed in the world, in any cloud, on-premises, or IoT environment. Technically speaking, an event mesh is a network of interconnected event brokers that share consumer topic subscription information and route messages amongst themselves so they can be passed along to subscribers.

5. Deferred Execution

If you're used to REST-based APIs, the concept of deferred execution can be tricky to comprehend. The essence of event-driven architecture is that when you publish an event, you don't wait for a response. The event broker "holds" (persists) the event until all interested consumers accept or receive it, which may be sometime later. Acting on the original event may then cause other

events to be emitted that are similarly persistent.

So event-driven architecture leads to cascades of events that are temporally and functionally independent of each other but occur in a sequence. All we know is that **event A** will at some point cause something to happen. The execution of the logic-consuming **event A** isn't necessarily instant; its execution is deferred.

6. Eventual Consistency

Following on from this idea of deferred execution, where you expect something to happen later but don't wait for it, is the idea of **eventual consistency**. Since you don't know when an event will be consumed and you're not waiting for confirmation, you can't say with certainty that a given database has fully caught up with everything that needs to happen to it and doesn't know when that *will* be the case. If you have multiple stateful entities (database, MDM, ERP), you can't say they will have exactly the same state; you can't assume they are consistent. However, for a given object, we know that it will become consistent *eventually*.

7. Choreography

Deferred execution and eventual consistency lead us to the concept of choreography. To coordinate a sequence of actions being taken by different services, you could choose to introduce a master service dedicated to keeping track of all the other services and taking action if there's an error. This approach, called *orchestration,* offers a single point of reference when tracing a problem, but also a single point of failure and a bottleneck.

With event-driven architecture, services are relied upon to understand what to do with an incoming event, frequently generating new events. This leads to a "dance" of individual services doing their own things but, when combined, producing an implicitly coordinated response, hence the term choreography.

8. CQRS: Command Query Responsibility Segregation

A common way of scaling microservices is to separate the service responsible for doing something (a command) from the service responsible for answering queries. Typically, you have to answer many more queries than for an update or insert, so separating responsibilities this way makes scaling the query service easier.

Using event-driven architecture makes this easy since the topic should contain the verb, so you simply create more instances of the query service and have it listen to the topics with the query verb.

The 6 Principles of Event-Driven Architecture

There we have it, some main principles of event-driven architecture:

1. Use a network of event brokers to make sure the right “things” get the right events.
2. Use topics to make sure you only send once and only receive what you need.
3. Use an event portal to design, document, and govern event-driven architecture across internal and external teams.
4. Use event broker persistence to allow consumers to process events when they’re ready (deferred execution).
5. Remember, this means not everything is up-to-date (eventual consistency).
6. Use topics again to separate out different parts of a service (command query responsibility segregation).

Seetharamugn highlighted

Conclusion

Event-driven architectures are a way of designing systems that make your applications more flexible and able to grow without problems across your business. Although this method can bring some new difficulties and complexities, it’s a good way to create complicated applications by allowing different groups to work on their own. Some cloud services even provide managed tools that can help your organization build these kinds of architectures. This guide explains many concepts about event-driven architectures, suggests some best practices, and mentions useful services to assist your development team in making these architectures. Remember, choosing an event-driven architecture isn’t just about technology. To truly get the benefits, you need to change how you think about development and how your team works. Developers will need a lot of freedom to choose the right technology and design for their specific part of the application.

References

1. https://solace.com/
2. https://aws.amazon.com/event-driven-architecture/#:~:text=An%20event%2Ddriven%20architecture%20uses on

Event Driven Architecture | Messaging | Kafka | Rabbitmq | Event Driven Design

Written by Seetharamugn

322 followers · 3 following

Tach Lead , senior software developer

Responses (3)

Ishu

What are your thoughts?

Ayush Mittal he/him
Jun 25, 2024

very well defined and written, PERFCT !!

2 1 reply Reply

Alex Pliutau
Jun 14, 2024

Great write-up. We also recently published an article on how to bridge Backend and Data Engineering teams using Event Driven Architecture - https://packagemain.tech/p/bridging-backend-and-data-engineering

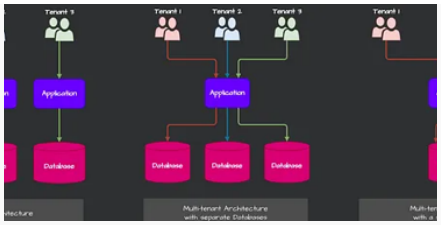
1 Reply

octree
Jan 5, 2024

Nice article, but I question EDA making CQRS easier.

1 Reply

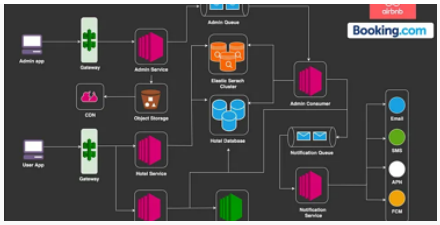
More from Seetharamugn



Seetharamugn

Complete Guide to Multi-Tenant Architecture

Oct 2, 2024 206 1



Seetharamugn

Complete Guidance of Hotel Reservation System Architecture

fig. Hotel Reservation System.

Nov 13, 2024 150 2

Seetharamugn

When to use RabbitMQ or Apache Kafka

Apr 20, 2023 117

Seetharamugn

JWT OAuth2.0 With Go Gin Gorm MySQL

Apr 5, 2023 2 2

See all from Seetharamugn

Recommended from Medium

Concurrent Mind

The Architecture Pattern That Scaled Netflix, Amazon, and Uber...

I remember the day my team's dashboard bled red.

Jul 27 9

In ITNEXT by Animesh Gaitonde

Scaling Distributed Counters: Designing a View Count System f...

Architecture, challenges and bottlenecks in counting views at scale

Jul 19 650 13

Subodh Shetty

Designing Reliable Distributed Systems: Transactional Outbox +...

If you're not a Medium member, you can still enjoy the full article for free. Click here to rea...

May 15

Ankit Kumar Srivastava

Design Uber Backend

System requirements

May 25 2

Anh Trần Tuấn

Event Sourcing in Microservices with Axon Framework

May 6 2

Agam Kakkar

Achieving Data Consistency in Microservices: A Practical Guide

One of the biggest challenges when designing microservices is ensuring data...

Aug 25 1

See more recommendations