Event Driven Architectures

How and When Should You Use It?

Name	Sumith Kumar Puri
Experience	~17 Years
Designation	Principal Consultant
Company	Xebia IT Architects, Bangalore

Word Count: 999, Paragraph Count: 44, Line Count: 125

Introduction and Needs

Over the past four years, I've worked on various microservices projects, and we've always had the same primary concern, Data Management. Senior Architects have had to answer questions such as: How do we maintain data? Where is the single version of truth? And how can we achieve distributed transactions? How can we quickly process high velocity business events?

Great problems bring greater solutions... and new problems too! Here is the introduction to the star-studded solution, Event Driven Architecture (EDA). However, EDA has been in use since the advent of enterprise software systems. It was the need for loosely coupled systems to communicate via asynchronous mechanisms that led to their wider adoption. Event-driven systems play a greater role in the newer forms of Software Architecture, such as Microservices. To get a sane, transactional distributed system we must implement EDA or its alternatives correctly.

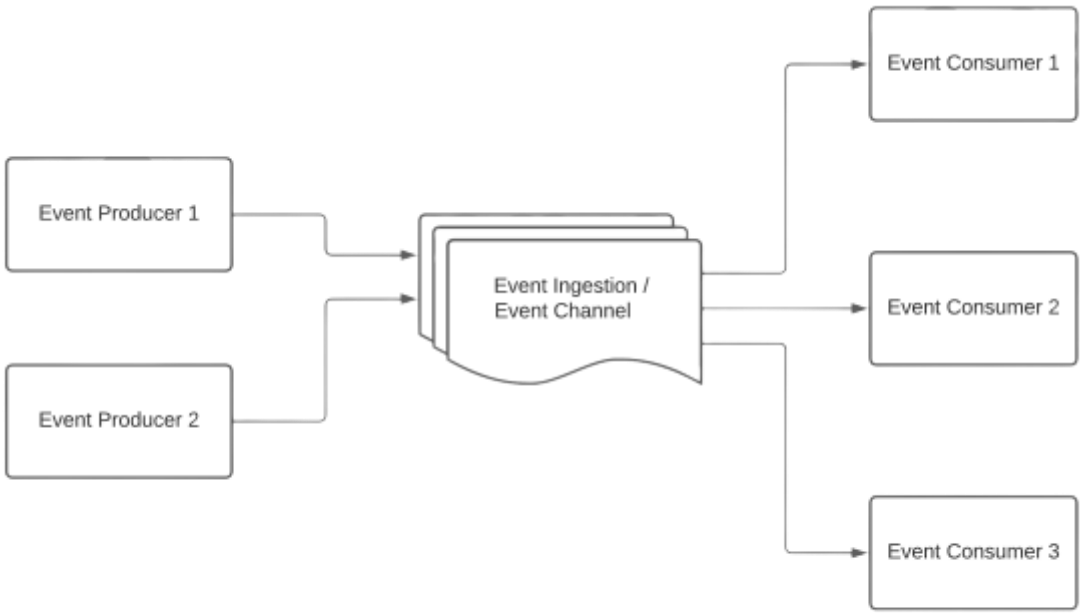
The growing market for EDA and its alternatives offers a wide range of products, tools, and solutions. The simplest form of EDA needs to be understood before we move forward.

How to Use EDA? (Components of EDA)

We can achieve the most basic form of EDA by providing implementation for each mentioned, abstract yet mandatory component.

- Event Producer - It is usually a software application or system that detects a new event or fact. For example, starting a database transaction, successful database transaction, failure of transaction, parameter change, etc. are all Events.

- Event Channel - If implemented, it would propagate information in a publish/subscribe manner (queues or topics) or by event streaming (data is available to read from wherever and whenever).



Event Driven Architecture (EDA) - Most Basic Form

- Event Consumer - The event consumer is, usually, the event processor. This includes the actual purpose of the event and may involve logic processing based on the specific event or stream of events. It may choose to process data in any of the following forms

- Simple Event Processing - A simple event is read from the event channel by an event consumer for the purpose of running some action or logic based on the event.

- Event Stream Processing - Uses a Data Ingestion Platform such as Apache Kafka, as a pipeline to ingest events and then process them using event stream processors.

- Complex Event Processing - Usually processes data as a series of events using platforms like Apache Storm. It tries to find event correlations among a continuous window or batch of events.

Example of an Event Driven System (Java, Drools Expert, Drools Fusion, Java Queue)

Budha, Complex Event Processing (By Sumith Puri)

When to Use EDA?

There are many uses of the Event Driven Architectures, as mentioned below:

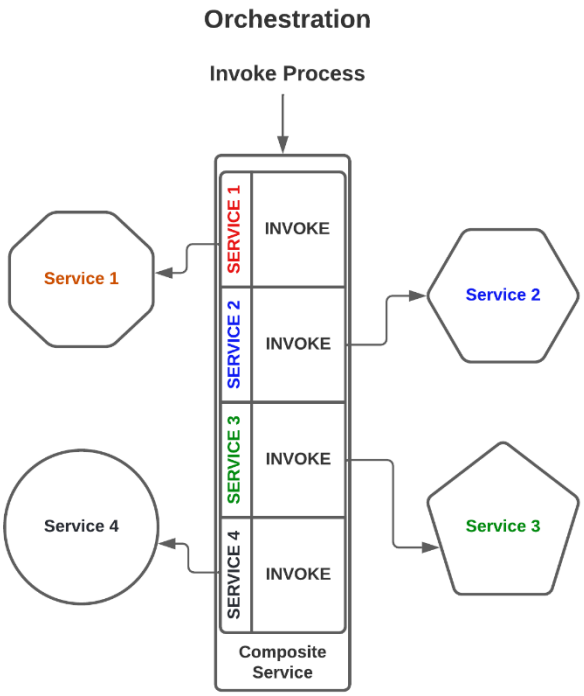
- Used when Multiple Subsystems Need to Receive Event Updates for Performing Some Activity
- Performing Event Correlation via Pre-Defined Rules on an Incoming Stream or Batch of Data
- Used to Perform a Real-Time Activity or Biz Logic when the Velocity of Incoming Data is High
- Perform Transaction or Data Management via Orchestration of Distributed Actions or Services

In older architectures, the primary use of EDA was for the purpose of ingesting and processing events/messages. It later included the processing of streams of events at a very high speed. Additionally, it involves analysing events by running rules that match events within a given sliding window or batch.

EDA, especially within microservices, is utilized for the purposes of data management or distributed transactions. It can also be used to orchestrate steps or services to accomplish a single logical business transaction.

Variants of Event Driven Architecture

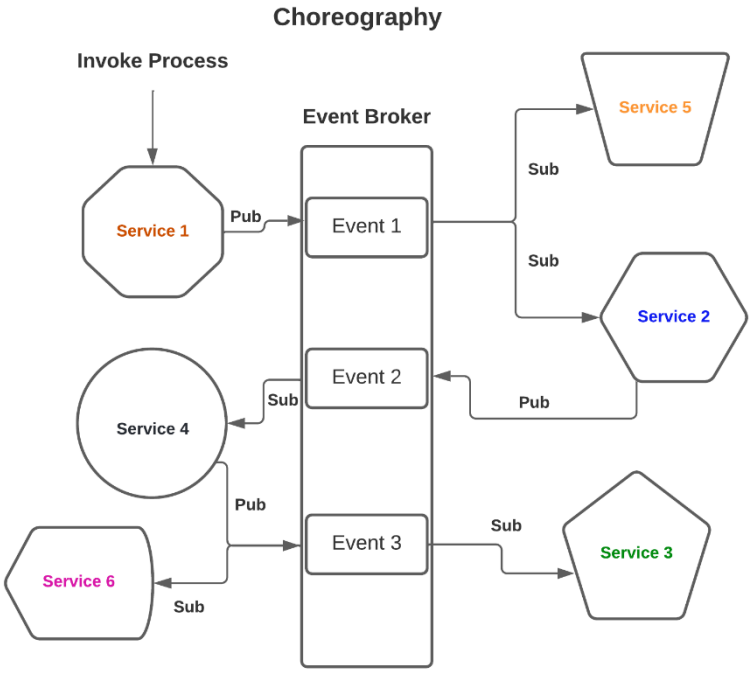
This discussion is primarily around the application of EDA to *Take an Action*, *Execute a Workflow*, *Synchronize Data*, *Apply Distributed and Compensating Transactions*. There are *alternatives* or *variants* or *newer* forms that are used to achieve EDA. Most of the ones in the Microservices world are related to Transaction Management. Orchestration is when an orchestrator instructs participants on the local transactions to execute. Choreography is achieved when each service publishes events that trigger local transactions in other services.



Event Driven Architecture

Saga (Camel, Axon)

Saga is the closest variant of EDA is based upon the idea that a series of one or more compensating transactions can be executed if a step fails in a distributed transaction. Saga supports both choreography and orchestration. Apache Camel Saga or Axon Saga are popular frameworks that allow its implementation.



Event Driven Architecture

Step Functions (AWS)

Step Functions are a series of steps modelled to achieve a distributed business transaction. It can be used to design a workflow with retry, catcher and compensatory steps. It can model either Saga, Orchestration or Choreography based transactional mechanisms. AWS Step Functions are the most popular implementation of step functions. Step Functions internally create state machines.

Workflow (Temporal.IO)

There are various adaptations of workflow modelers or state machines that allow definition of workflows for the purposes of business orchestration. It can be used to create any form of workflows, with or without compensating transactions. Temporal.IO is a good example of a Workflow Designer for Microservices orchestration or choreography.

Command Query Responsibility Segregation (CQRS)

Even though this existed quite before microservices become mainstay in enterprise development, CQRS is equally applicable for microservices. It simply emphasizes the need for one channel or set of services to write to the database and another set to read from the database, with both the databases kept in sync via external mechanisms.

Event Sourcing

The idea of writing a sequence of events to the database that can be read to recreate the actual final state of the activity or transaction is called as event sourcing. This is one popular alternative to Event Driven Architectures and can be used to run workflows or decide on transactional state.

Challenges of EDA

I will try and briefly cover the challenges or drawbacks of EDA. Again, these are easily understood without detailed explanation:

- Initial Curve of Understanding Problem at Hand
- Learning Curve of Newer Tools & Technologies
- Recovery from Total Failure in ED Architectures
- Reconciliation of [Partial Failures] in EDA Steps
- Needs Acceptance: Performance isn't Real-Time

References

- https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/event-driven
- https://microservices.io/patterns/data/event-driven-architecture.html
- https://microservices.io/patterns/data/saga.html

[End of Article / Blog Entry]

Real-World Experience

The experience gained as a (Platform & API Dev Lead) in a Neo Banking Project at an IT Consultancy and experience as a (Principal Architect) in leading the Microservices Adoption Strategy in a Software Product Company was the basis of writing the above article. Also, my experience was further enriched via links from Microsoft, Microservices.IO, IBM at the time of writing this article.

About the Author

Sumith Kumar Puri is a Principal Consultant at Xebia IT Architects, Bangalore. He has a total of 17y of Experience and is a Java, Java EE, and Microservices Expert. He completed his Xth, XIIth (Computer Science) from Naval Public School, Kochi, India. He graduated as a Bachelor of Engineering (Computer Science & Engineering*) from SRSIT, Bangalore, India. He also has completed his Executive Program (Data Mining & Analytics) from IIT, Roorkee and Executive Program (Entrepreneurship) from the IIM, Kashipur. He is highly recognized in the Developer, Technical, Engineering Communities under the DZone MVB/Core*, Java Code Geeks, Foojay.IO, Developer.com, jGuru.com, IEEE, ACM and CSI.

He is the Author of the Book – Microservices Architecture, The Decision Maker. You may buy a copy of this book from Amazon, Flipkart or Notion Press. Kindly subscribe to his technical blog, directly using your Google or Gmail Id.