DRAFT - IT Project

On Event Stream Architecture

Version:

0.1

## Purpose

The purpose of this document is to outline the objectives, scope, and foundational rationale behind adopting an event stream architecture for our IT project. It aims to provide stakeholders with a clear understanding of the anticipated benefits, guiding principles, and the strategic vision driving this architectural choice.

## Synopsis

This document provides an overview of the event stream architecture proposed for the IT project. It describes the core concepts of event-driven systems and explores the practical motivations for their adoption. The synopsis addresses and compares event streaming differences with more traditional models, including considerations of system responsiveness, scalability, and integration. Key aspects, such as data flow, interoperability, and operational challenges, are introduced to frame the scope and direction of the detailed sections that follow.

## Contents

[Purpose 1](#_Toc195791704)

[Synopsis 1](#_Toc195791705)

[Contents 2](#_Toc195791706)

[Introduction 3](#_Toc195791707)

[Background of this work 3](#_Toc195791708)

[Context 3](#_Toc195791709)

[Background of issue 3](#_Toc195791710)

[Problem Statement 3](#_Toc195791711)

[H3 4](#_Toc195791712)

[H4 4](#_Toc195791713)

[Appendices 5](#_Toc195791714)

[Appendix A - Document Information 5](#_Toc195791715)

[Versions 5](#_Toc195791716)

[Images 5](#_Toc195791717)

[Tables 5](#_Toc195791718)

[References 5](#_Toc195791719)

[Review Distribution 5](#_Toc195791720)

[Audience 5](#_Toc195791721)

[Structure 5](#_Toc195791722)

[Diagrams 6](#_Toc195791723)

[Acronyms 6](#_Toc195791724)

[Terms 6](#_Toc195791725)

# Context

# Comparison

## Default Web Service Architecture

Traditional web-based transactional service architecture is fundamentally built around the concept of representing and persisting the current state of data within a system. At its core, this approach relies on database records where each entity is represented by its latest snapshot, facilitating transactional consistency and straightforward retrieval of the most up-to-date information. Applications interact with this data via CRUD (Create, Read, Update, Delete) operations, manipulating the system’s state in direct response to user or system requests.

Traditional web-based service architecture is fundamentally built around the concept of representing and persisting the current state of data within a system. At its core, this approach relies on database records where each entity is represented by its latest snapshot, facilitating transactional consistency and straightforward retrieval of the most up-to-date information. Applications interact with this data via CRUD (Create, Read, Update, Delete) operations, manipulating the system’s state in direct response to user or system requests.

Audit mechanisms within this architecture are typically implemented by introducing shadow tables or history tables, designed to capture previous versions of records whenever changes occur. While this approach preserves a linear progression of value changes—offering a level of traceability—it primarily surfaces the ‘what’ of data evolution, rather than the ‘how’ or ‘why’ behind each transition. This can present limitations in terms of forensic analysis or reconstructing the logic leading to a particular state.

Furthermore, traditional architectures often emphasize the integrity and security of audit data, protecting these tables from deletion or unauthorized modification. Such protections are critical to maintaining trust in the audit trail, ensuring that records of change remain reliable for compliance or investigative purposes.

Despite its strengths, including maturity, abundant resources, and a well-established ecosystem, this architecture can introduce friction when deeper event analysis is required. Tracing the full context of a change—particularly when attempting to reconstruct the sequence of decisions or actions that produced a given outcome—can involve navigating across multiple tables and piecing together disparate records. Additionally, while effective for transaction recording, this model is less adept at simulating alternative scenarios or projecting outcomes under hypothetical rulesets, which are increasingly valuable in dynamic or highly regulated environments.

## Event Driven Architecture

In contrast to traditional web-based transactional service architectures, Event Driven Architecture (EDA) is centered around the generation, capture, and propagation of discrete events that represent meaningful changes or occurrences within the system.

Rather than focusing solely on the current state of data, EDA treats the flow of events themselves as primary artifacts, capturing not only the ‘what’ of change but also the ‘how’ and ‘why’—the full narrative of system evolution.

The fundamental difference between EDA and traditional approaches lies in the perspective on data and its lifecycle.

Where traditional systems persist only the latest state (with optional auditing layers), EDA embraces the persistence of every significant event as it happens.

Each event is an immutable record that describes a domain-specific fact, and together, these events form an event log or stream.

The current state of any entity can always be reconstructed by replaying its sequence of events from the event log, providing a detailed and auditable history of all system activity.

This paradigm shift offers several distinctive advantages. First, EDA enables richer forensic and analytical capabilities, as every step in the evolution of data is inherently traceable.

In a traditional audited transactional web app, audit mechanisms—while robust in capturing the progression of state through history or shadow tables—primarily document the “what” of each change. They can sometimes record associated metadata (such as user ID or timestamp), but typically do not capture the full context, intent, or causal sequence leading to a change. This means that, although you can reconstruct a timeline of changes, discerning the precise reasoning or chain of business logic behind each transition is often difficult and fragmented across various tables.

EDA fundamentally differs because it treats each event as a first-class, immutable artifact. Every event is time-stamped, causally linked, and domain-specific, often encoding not just the “what,” but also elements of the “how” and “why.”

By persisting all events, EDA allows for a much more granular and reconstructible system history, supporting detailed forensic analysis, sophisticated auditing, and the ability to project state under alternative scenarios simply by replaying or filtering the event stream.

While both approaches can meet compliance and audit requirements, EDA offers deeper transparency, traceability, and the potential for richer analytical insights.

However, it does so at the cost of increased complexity in terms of system design, consistency management, and operational monitoring.

Thus, the suitability of EDA versus a conventional audited web application hinges on the specific demands for traceability, scale, and adaptability within a given context.

Developers and auditors can reconstruct exact sequences of actions, analyze causal relationships, and even simulate alternative scenarios by replaying or branching from specific points in the event stream.

This approach is particularly well-suited for systems requiring high transparency, regulatory compliance, or advanced analytics.

EDA also naturally supports scalability and flexibility. Because components communicate asynchronously by producing and consuming events, systems are decoupled and can evolve independently. This facilitates horizontal scaling, integration with external systems, and resilience in the face of partial failures—since events can be queued and processed as resources re/become available.

However, EDA is not without its disadvantages. The complexity of managing event streams, ensuring consistency, and handling eventual consistency scenarios can be significant.

Designing for idempotency, managing distributed transactions, and guaranteeing data integrity across services require new patterns and careful implementation.

Debugging and reasoning about system behavior can become more challenging, as system state is distributed across potentially vast collections of immutable events rather than centralized tables.

Additionally, the learning curve and ecosystem maturity may be less favorable compared to the well-established CRUD-based approaches. Tooling, operational monitoring, and developer expertise for EDA are evolving, but may not be as extensive as those for traditional architectures.

In essence, Event Driven Architecture offers a paradigm that excels in transparency, flexibility, and auditability, with the trade-off of increased complexity and operational considerations. Its suitability depends on the specific requirements, scale, and regulatory context of the system in question.

## Hybrid Architecture

While EDA and conventional audited web applications often appear as distinct, opposing paradigms, there are hybrid approaches that seek to balance their respective strengths and limitations. Some architectures adopt a layered model, combining traditional CRUD operations with event-sourcing selectively for domains where traceability and auditability are essential, while retaining simpler, synchronous workflows elsewhere. Others leverage change data capture or append-only logs integrated into otherwise conventional systems, providing a degree of audit trail and replayability without the full commitment to event-driven paradigms. These intermediary solutions can offer a pragmatic compromise—enhancing transparency and flexibility within critical domains, while minimizing overall system complexity and leveraging established tooling where appropriate. Such blended strategies enable organizations to tailor their architecture according to varying regulatory demands, performance considerations, and team expertise, rather than making an all-or-nothing choice.

Let’s make an example to better demonstrate how EDA works.   
Personally, I am currently on the wrong track, trying to boil it down to pub sub mostly.   
However, yesterday, I was shown all kinds of .NET Core ode I have never seen similar before – and using tech I didn’t know purpose of -- for example MartenDB? And the business entities had two methods on it: Apply, and transform (could be by other names). Some thing I didn’t understand was the Apply sent a package of the object’s current and changed values, and Transform seemed to apply that package to the object’s properties. The properties could not be set from outside (no public set)  
  
Assume: an interface that permits uploading a form. I suspect that creates an event.

Appendices

Appendix A - Document Information

Authors & Collaborators

* Sky Sigal, Solution Architect

### Versions

* 1. Initial Draft

### Images

[Figure 1: TODO Image 2](#_Toc144995112)

### Tables

[Table 1: TODO Table 3](#_Toc145048484)

[Table 2: TODO Table 2 3](#_Toc145048485)

### References

**There are no sources in the current document.**

### Review Distribution

The document was distributed for review as below:

|  |  |
| --- | --- |
| Identity | Notes |
|  |  |
|  |  |
|  |  |

### Audience

The document is technical in nature, but parts are expected to be read and/or validated by a non-technical audience.

### Structure

Where possible, the document structure is guided by either ISO-\* standards or best practice.

### Diagrams

Diagrams are developed for a wide audience. Unless specifically for a technical audience, where the use of industry standard diagram types (ArchiMate, UML, C4), is appropriate, diagrams are developed as simple “box & line” monochrome diagrams.

### Acronyms

API

: [Application Programming Interface](#Term_ApplicationProgrammingInterface).

DDD

: Domain Driven Design

GUI

: [Graphical User Interface](#Term_ApplicationProgrammingInterface). A form of [UI](#Acronym_UI).

ICT

: acronym for Information & Communication Technology, the domain of defining Information elements and using technology to automate their communication between entities. [IT](#Acronym_IT) is a subset of ICT.

IT

: acronym for Information, using Technology to automate and facilitate its management.

UI

: User Interface. Contrast with [API](#Acronym_API).

### Terms

Refer to the project’s Glossary.

Application Programming Interface

: an Interface provided for other systems to invoke (as opposed to User Interfaces).

Capability

: a capability is what an organisation or system must be able to achieve to meet its goals. Each capability belongs to a domain and is realised through one or more functions that, together, deliver the intended outcome within that area of concern.

Domain

: a domain is a defined area of knowledge, responsibility, or activity within an organisation or system. It groups related capabilities, entities, and functions that collectively serve a common purpose. Each capability belongs to a domain, and each function operates within one.

Entity

: an entity is a core object of interest within a domain, usually representing a person, place, thing, or event that holds information and can change over time, such as a Student, School, or Enrolment.

Function

: a function is a specific task or operation performed by a system, process, or person. Functions work together to enable a capability to be carried out. Each function operates within a domain and supports the delivery of one or more capabilities.

Person

: a physical person, who has one or more Personas. Not necessarily a system User.

Persona

: a facet that a Person presents to a Group of some kind.

Quality

: a quality is a measurable or observable attribute of a system or outcome that indicates how well it meets expectations. Examples include reliability, usability, and performance. Refer to the ISO-25000 SQuaRE series of standards.

User

: a human user of a system via its UIs.

User Interface

: a system interface intended for use by system users. Most computer system UIs are Graphics User Interfaces ([GUI](#Acronym_GUI)) or Text/Console User Interfaces (TUI).