# Event-based Architecture

## Definition

- Event based architecture is an architecture that controls a software's behaviour through the production and listening of events. This architecture is made up of the following components: events, an event bus, event producers, event consumers.
- In event based architecture, event producers generate events when a change in the state of the system is detected. The events are then sent to the event bus, which is known by every event consumer in the system. The event consumers then continuously check the event bus until it detects an event that it is looking for. Once it detects such an event, the event consumer will respond to the event by taking some action within the system.
- **Vocabulary for components and connectors:**
    - Event: a notification of a specific change in the current state of the system.
    - Event bus: a software or hardware component that events are sent to when they are generated.
    - Event producer: a software or hardware component that can generate events.
    - Event consumer: a software component that monitors the event bus for the appearance of certain types of events.

## Topological constraints

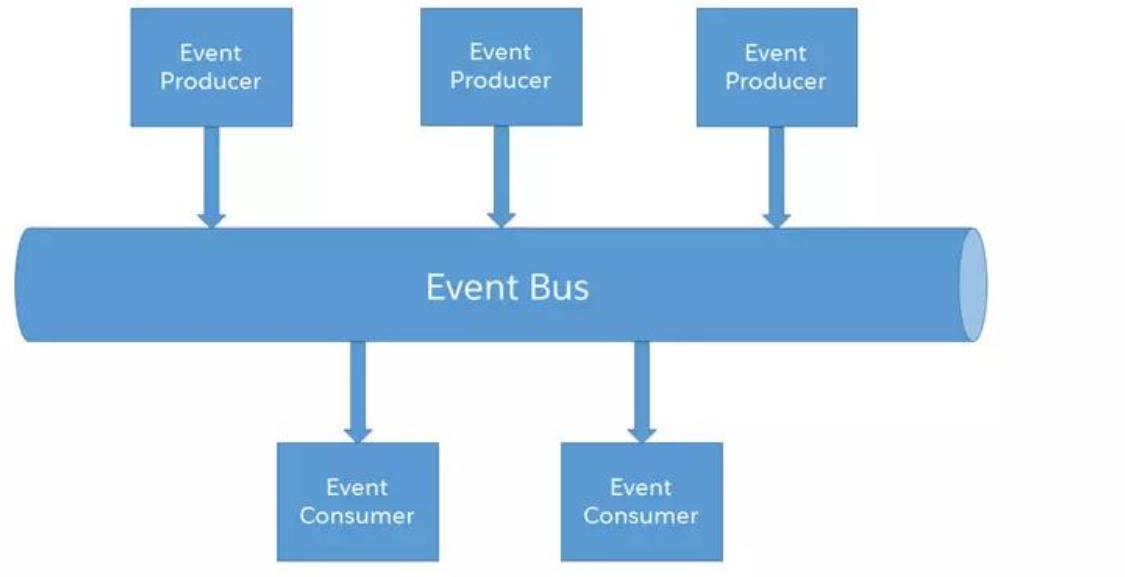


fig. [1]

[1] Source: https://trailhead.salesforce.com/en/content/learn/modules/platform_events_basics

- The topological constraints of an event based architecture are largely in the mediator between components.
- The topology requires at least one event bus.
- No component should be allowed to talk directly to each other.
    - Components do not know of each other's existence and can only communicate to each other the event bus.
    - However, it is possible that a component both produces and consumes events to the same event bus.

# Applicable problems

- Multiple subsystems or services need to process the same events
    - Same events can be sent to multiple components allowing them to gain access to the same data.
- Low latency and high volume of data
    - Event consumers are notified and can process events in near real time since event buses are read asynchronously.
- Distributed systems
    - The architecture is referentially decoupled and well distributed because it allows events and components to exist anywhere without knowing the existence of each other.
- User Interface Programming
    - System can continue its execution and be notified when the user generates an input.

# Resilience to change

- Highly distributed components
    - In event-based architecture, any component within the system can be an event generator, event consumer or both.
- Loose coupling amongst components
    - Event generators and event consumers do not directly communicate with one another. They communicate information to each other via the event bus. Hence, event generators and event consumers can be added or removed with ease.
- Architecture is tightly coupled to event schema
    - Any changes to event type or event name requires a change to every generator/consumer that produces/handles the specific event.

# Negative behaviours

- Inefficiency in event processing:
    - Event bus acts as an extra abstraction layer and thus, instead of quick and direct communication between event generators and event consumers, all events must pass through the event bus.
- Single point of failure:
    - Event bus implementation errors can result in the entire system crashing or not producing expected results.
    - Event bus is a bottleneck when multiple producers are generating and sending more events to the event bus, than what it is capable of handling.
- Lack of a feedback mechanism:
    - Absence of feedback mechanism makes it hard to ensure whether a certain event has been received and processed by a consumer.
    - No acknowledgement of events by event consumers.
- Absence of a logical control flow:
    - There is not a direct link between generators, consumers and asynchronous calls happening simultaneously, making it difficult to trace code paths.
    - The architecture uses implicit function calls, making debugging and refactoring difficult.
- Tight semantic coupling:
    - If components change information about the event produced, changes to event component and connector must be made such that the event can be handled accordingly.

# Supported NFPs

- Scalability:
    - It is very easy to add new event generators and consumers to the system. It has independent and decoupled event consumers and generators. Changes to components are isolated and can be quickly made without affecting other components.
- Performance:
    - It has high performance because of its asynchronous capabilities. It performs decoupled parallel operations. This is faster than the process of queuing and dequeuing messages. Event consumers can immediately respond to events as they arrive and they do not have to wait on each other since these components are not directly related to each other.
- Heterogeneity:
    - Event based architecture is formed of many loosely coupled components. As a result, it is true to say that such an architecture is composed of disparate parts. It is also

possible to separate out only the components that are related to each other without much change to any of the components involved, if any.
    - The generators and consumers are separate from one another and do not directly interact with one another. They are cohesive but have isolated functionalities

# Inhibited NFPs

- Performance:
    - For the processing of events, it is slower in performance, due to the fact that there is an extra layer that the data has to go through. So, when an event has to be relayed from the generator to the consumer, it must go through the event bus, which makes this aspect of the system slower.
- Security:
    - If the integrity of the event bus is compromised and is no longer secure, then the integrity of the data flowing between the event generators and consumers is also compromised. Also, if events are generated or modified without authorization, then consumers will receive these events not knowing that it was done so unauthorized, since they do not have direct communication with the generators.
- Dependability:
    - Problems in the event bus result in events not being processed since components are highly dependent on the connector.

# Comparison with other architectures

- Event-based architecture is not subscribing to a generator, rather to a bus.
- In the publish-subscribe architecture, subscribers express interest in one or more classes and only receive messages that are of interest.
- In peer-to-peer architecture, the information is broadcast one set of components at a time.
    - Further, there is no single point of failure like in event-based and publish-subscribe architectures.

# Credit:

Students:

Best Cat Hustlers (Steven Yang, Xavier Chong, Jiaxi Zhang, Alexander Kim)
clarityapp (Fazha Fareez, Disha Hasmukhbhai Patel, Tran My Han Phung)

Instructor and TAs:

Mei Nagappan, Achyudh Ram Keshav Ram, Aswin Vayiravan, Wenhan Zhu