Event Bus Architecture

Group 14:
Andrew Dunetz, Austin Fett, Chenming Wang, Daniel Gaugler, Matthew Layne, Ryell Deruijter

What is an Event Bus Architecture?

- Notify Subscribers of an Event or Changed State
- Handle rule based routing to certain handlers.
- Optionally, ensure the Subscriber processes event.

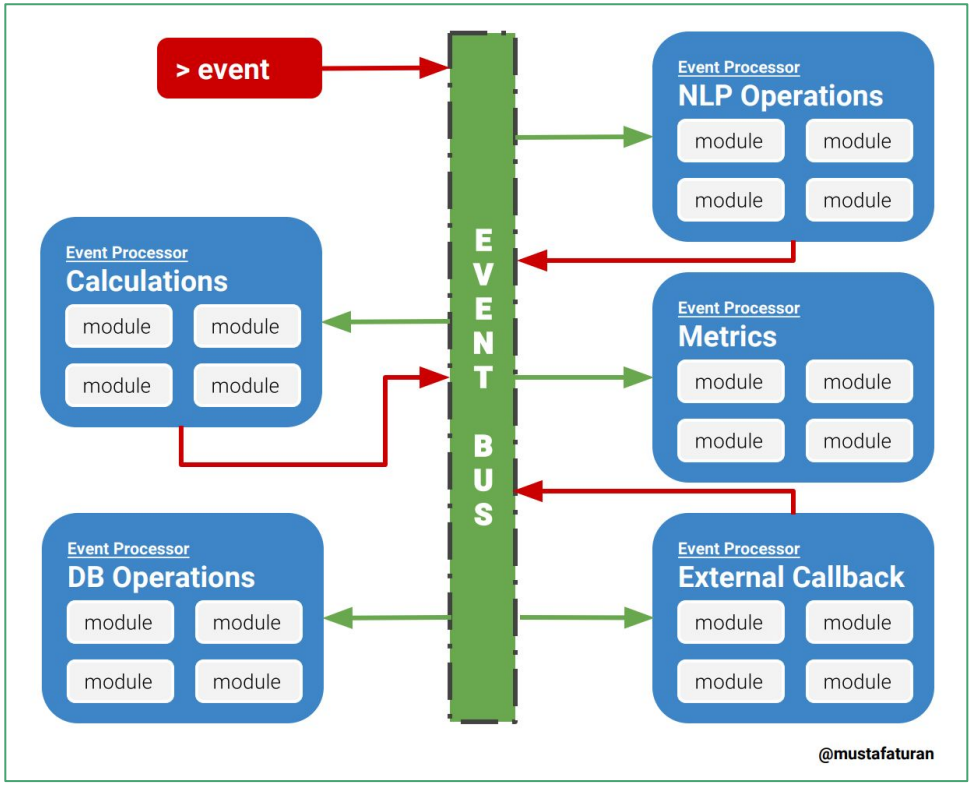

Figure 1: A Very Simple Event Bus Architecture [1]

What is the Purpose?

- Allow a centralized way to notify all subscribers of important changes.
- Provides easy scalability.
- Customizable:
  - Send update to all subscribers.
  - Store update and notify subscribers with pointer to update.
  - Can use *forced subscriber acknowledgement* or *no acknowledgement*.
  - Rule based event notification.
  - etc.

Real world examples that Follow this architecture:

- Retail store with online shopping (Belk, Target, Walmart, etc.) (See figure.)
- Online Catalog prices and details (Live-updating carts)
- Internet Routers
- Android framework

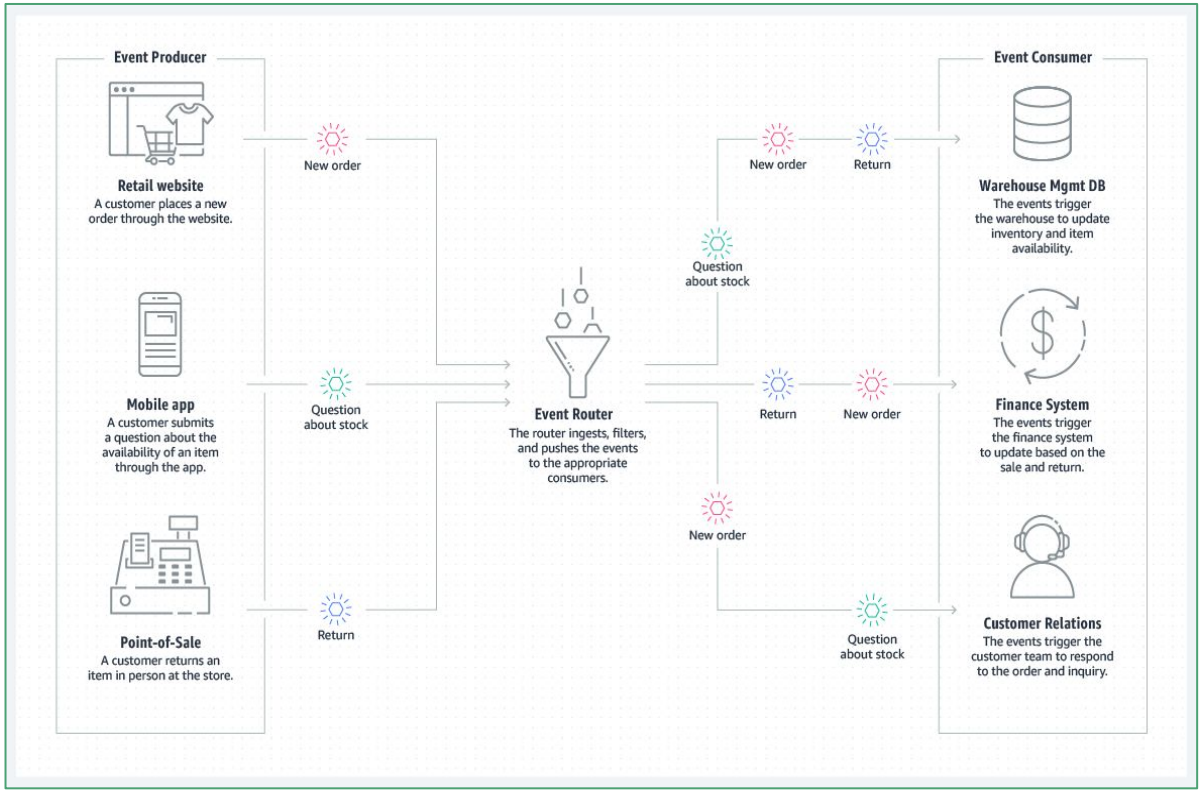

Figure 2: Retail Store Event Bus Example [3]

Advantages

- Creates a buffer between event and processors, acting as a centralizing location
  - Communication can be done through events instead of through direct object reference
- Allows for decoupling of services
  - When coupling becomes cumbersome
- Push-based
  - Reduces resource consumption

Disadvantages

- Not efficient for small scale communications (1 to 1 or 1 to few)
- Not ideal if there are many events to be monitored compared to small subscriber count
- Provides a potential centralized single point of failure (“Bottleneck”)
    - An error in the event bus means everything in the system will be affected

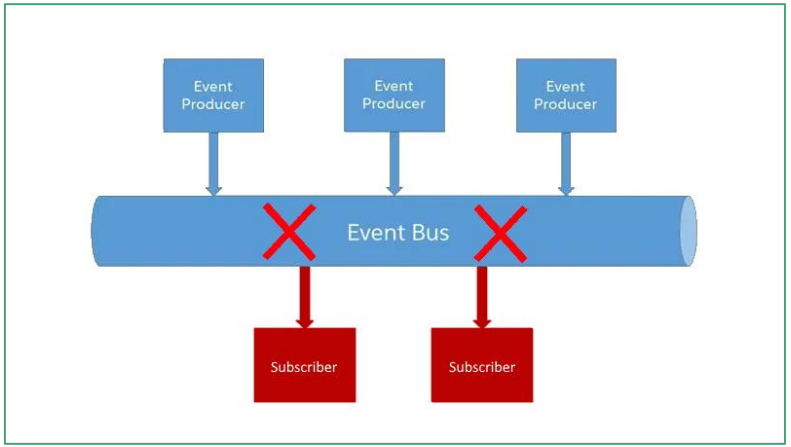

Figure 3: Single point of failure causing widespread outage.

Disadvantages Continued

- There can be significant overhead/memory consumption if there are many subscribers depending on how the event bus is implemented
  - Requires very good infrastructure on both the ends (producer and consumer)
- Update to the same event and duplication of an event makes the system more challenging to handle, making the system complex. This may also result in increased time for testing and debugging scenarios

Implementation

Basic Implementation Details:

- Subscribers
- Events-data (ex: javascript)
- Notify
- Subscribe
- Unsubscribe
- Event Publisher

```java
public class EventBusSubscriber {

    private final EventBus mEventBus;

    public EventBusSubscriber(EventBus eventBus) {
        mEventBus = eventBus;
    }

    public void subscribe(EventBus.Subscriber subscriber) {
        mEventBus.subscribe(subscriber);
    }

    public void unsubscribe(EventBus.Subscriber subscriber) {
        mEventBus.unsubscribe(subscriber);
    }
}

public class EventBusPublisher {

    private final EventBus mEventBus;

    public EventBusPublisher(EventBus eventBus) {
        mEventBus = eventBus;
    }


    public void publish(EventBus.Event event) {
        mEventBus.publish(event);
    }
}
```

Questions?

What are some real world examples of situations where this architecture would not work?

Resources:

1. https://medium.com/elixirlabs/event-bus-implementation-s-d2854a9fafd5
2. https://www.techyourchance.com/event-bus/
3. https://aws.amazon.com/event-driven-architecture/
4. https://ducmanhphan.github.io/2020-06-06-Event-Bus-pattern/