Outbound messaging was the preferred less code option to send updates to records within Salesforce to an external system. It has certain characteristics like automatic retries, full configuration (on SF side at least) and conditional logic(via workflows meeting predefined criteria)
However the external system had to implement a Soap based web service which the outbound messaging endpoint would hit to send the message. This is pretty anachronistic with today's time and was perhaps fine years ago.
Enter Platform events which have been available for quite some now and have matured a lot on Salesforce platform. Platform events are based on Kafka style event based messaging and hence adhere to the current style and norms of integration. They have various use cases from being used for streaming api , performing generic updates to carrying out change data capture.
This post shows us how we can use them in lieu of the traditional outbound messaging. Note that there is no comparison between them at a technical level but at a functional level the following similarities exist.

Feature	Outbound messaging	Platform events
Configuration	No code needed on Salesforce side	No code needed on Salesforce side if event publishing done via process builder
Retry capability	Automatic retires based on exponential back-off	Consumer web service can use replay-id to access events from an arbitrary point in the past
Record criterion match	Based on workflow rule	As part of the process builder

Thus whenever you think of using outbound messaging think of replacing them with Platform events
In addition to replacing outbound messaging the following 2 features also make their usage appealing
High volume events - As per the Salesforce roadmap they plan to support up to 100 millions events per day in the near future which effectively makes their usage overcome event api governor limits for most use cases.
Async triggers - In many projects the business logic is written within triggers and their helpers in such a way that there are always two parts to it. The first are immediate, real time operations (like validations) and the second are the non real time operations (like notifications) which are written in @future or in 'Queueable' manner like shown below

```
trigger AccountTrigger on Account (before insert) {

        //Some real time code like validations

        //Some async code like parent record update, notifications, web service calls

}
```

With platform events the same code can be now written on two different triggers.The real time stuff can be written as before on the normal trigger while the async code can be written on the *<object>changeEvent* trigger. This trigger is guaranteed to be invoked after a real commit is done to the database. This paradigm helps in better code decoupling as compared to the traditional approach

```
trigger AccountTrigger on Account (before insert) {

        //only real time code like validations

                |

}


trigger AccountEventTrigger on AccountChangeEvent  (after  insert) {

        //only async code like parent record update, notifications, web service calls


}
```

Overall platform events usage should be done more actively and it should be at the forefront of the enterprise strategy.