Understanding the capabilities of today's web communication technologies

목차

1. HTTP
2. REST
3. HTTP Polling
4. HTTP Streaming
5. SSE (Server Sent Events / EventSource)
6. HTTP/2 Sever Push
7. WebSockets
8. REST vs Websockets - Perf Test
9. Webhooks (for communication between servers)

1. HTTP

HTTP는 WWW (World Wide Web) 의 기본 통신 프로토콜이다.

HTTP는 클라이언트-서버 컴퓨팅 모델에서 요청-응답 프로토콜로 작동한다.

HTTP의 초기 버전과 비교하여 최신버전은 지속적 및 파이프 라인 연결, 청크 전송, 요청/응답 본문의 새 해더 필드 등과 같은 중요한 성능 최적화 및 기능 향상을 구현할 수 있다.

Keep-Alive 호스트 간의 오래 지속되는 통신 정책을 설정하기 위한 헤더 (연결 당 처리 할 시간 초과 기간 및 최대 요청 수)

Upgrade 헤더는 HTTP / 2.0 (같이 향상된 프로토콜 모드로 접속을 전환 h2, h2c() 또는 WebSockets를 websocket)

2. REST

아키텍처 스타일 인 REST (Representational State Transfer) 는 요청에 대한 웹 API를 구성하는 가장 표준화 된 방법이다.

WWW과 같은 분산 하이퍼미디어 시스템을 위한 소프트웨어 아키텍처의 한 형식이다.

REST는 순전히 몇 가지 원칙을 기반으로 한 architectural 스타일이다.

REST 원칙을 준수하는 API를 RESTful API라고 한다.

REST API는 서버의 모든 메시지가 클라이언트의 메시지에 대한 응답인 요청 / 응답 모델을 사용한다.

일반적으로 RESTful API는 HTTP를 전송 프로토콜로 사용한다.

이러한 경우 조회는 GET 요청을 사용해야 한다.

PUT, POST 및 DELETE 요청은 각각 mutation, creation(생성) 및 deletion(삭제)에 사용되어야 한다. (정보를 업데이트하는 것에 대해서 GET 요청을 사용하는 것은 피해야 한다)

3. HTTP Polling

HTTP Polling 에서 클라이언트는 여러 매커니즘 중 하나를 준수하여 새 정보를 요청하는 서버를 폴링한다.
폴링은 오늘날 대부분의 애플리케이션에서 사용되면 대부분 RESTful 프랙티스와 함께 사용된다.
실제로 HTTP Short Polling 은 거의 사용되지 않으며 HTTP Long Polling 또는 Periodic Polling은 항상 선택 된다.

HTTP Short Polling

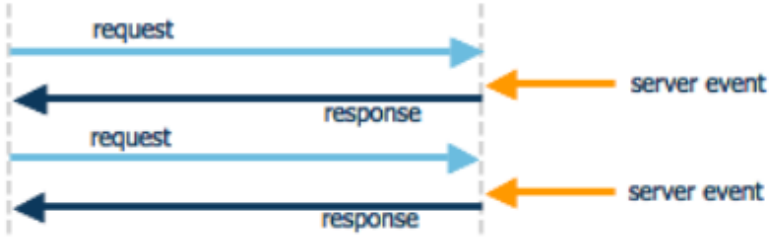
간단한 접근 방식이다.
많은 요청은 서버에 들어올 때 처리되어 많은 트래픽을 생성한다. (리소스를 사용하지만 응답이 전송되는 즉시 해제)
각 연결은 단시간 동안만 열리므로 많은 연결을 시간 다중화 할 수 있다.

```
00:00:00 C-> Is the cake ready?
00:00:01 S-> No, wait.
00:00:01 C-> Is the cake ready?
00:00:02 S-> No, wait.
00:00:02 C-> Is the cake ready?
00:00:03 S-> Yeah. Have some lad.
00:00:03 C-> Is the other cake ready?
```

3. HTTP Polling

HTTP Long Polling

하나의 요청이 서버로 이동하고 클라이언트가 응답이 오기를 기다린다.
서버는 새 데이터를 사용할 수 있을 때까지 요청을 열린 상태로 유지한다. (해결되지 않고 리소스가 차단됨)
서버 이벤트가 발생할 때 지연없이 통지된다.
더 복잡하고 더 많은 서버 리소스가 사용된다.



HTTP Long Polling — Response is held until server process data (Image from shyamapadabatabyal.wordpress.com)

```
12:00 00:00:00 C-> Is the
cake ready?
12:00 00:00:03 S-> Yeah.
Have some lad.
12:00 00:00:03 C-> Is the
other cake ready?
```

3. HTTP Polling

HTTP Periodic Polling

두 요청 사이에 사전 정의 된 시간 간격이 있다.

이것은 폴링의 개선 / 관리 버전이다.

두 요청 사이의 시간 간격을 늘려 서버 소비를 줄일 수 있다.

그러나 서버 이벤트가 발생할 때 지연없이 알림을 받아야하는 경우에는 이 옵션은 좋지 않다.


```
00:00:00 C-> Is the cake
ready?
00:00:01 S-> No, wait.
00:00:03 C-> Is the cake
ready?
00:00:04 S-> Yeah. Have
some lad.
00:00:06 C-> Is the other
cake ready?
```

4. HTTP Streaming

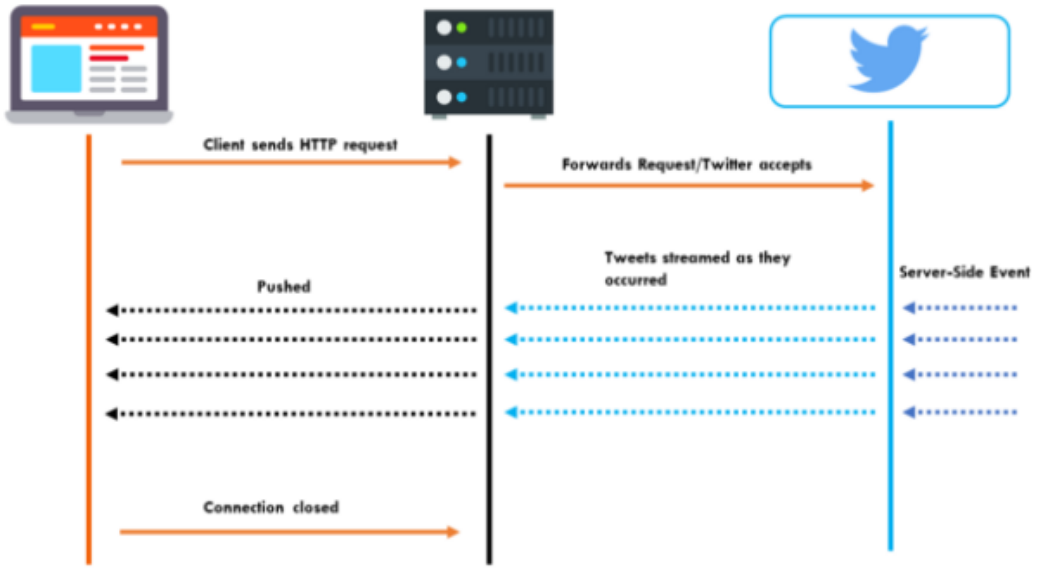
클라이언트는 HTTP 요청을 하고 서버는 무한 길이의 응답을 제한한다.

HTTP 스트리밍은 성능이 뛰어나고 사용하기 쉬우며 WebSockets의 대안이 될 수 있다.

- 문제점 :

  중개자가 연결을 중단할 수 있다. (예를 들어, 시간 초과나 다른 요청을 라운드 로빈 방식으로 제공하는 중개자)

  이러한 경우 완전한 실시간 성을 보장 할 수 없다.



HTTP Streaming — provides a long-lived connection for instant and continuous data push (Image from realtimeapi.io)

```
00:00:00 CLIENT-> I need cakes
00:00:01 SERVER-> Wait for a moment.
00:00:01 SERVER-> Cake-1 is in process.
00:00:02 SERVER-> Have cake-1.
00:00:02 SERVER-> Wait for cake-2.
00:00:03 SERVER-> Cake-2 is in process.
00:00:03 SERVER-> You must be enjoying cake-1.
00:00:04 SERVER-> Have cake-2.
00:00:04 SERVER-> Wait for cake-3.
00:00:05 CLIENT-> Enough, I'm full.
```

5. SSE (Server Sent Events / EventSource)

SSE 연결은 데이터를 브라우저로만 푸시 할 수 있다.

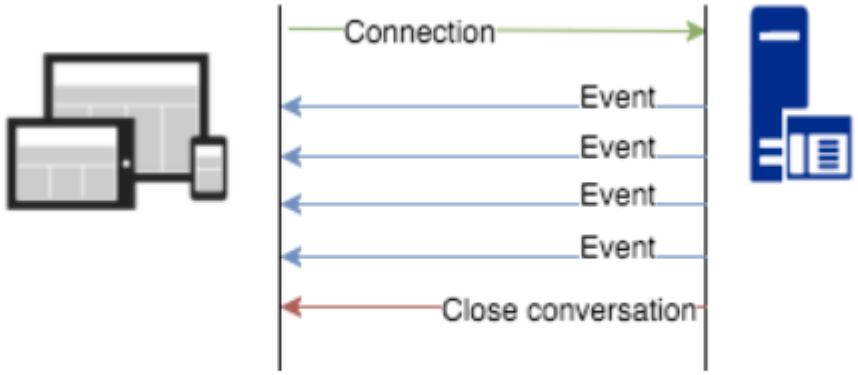
통신은 서버에서 브라우저로만 수행되며 브라우저는 서버에서 시작된 데이터 업데이트에만 가입할 수 있지만 서버로 데이터를 보낼 수는 없다.

Sample applications : Twitter update, 주가, 브라우저 알림

문제 1 : 일부 브라우저는 SSE를 지원하지 않는다.

문제 2 : 최대 열린 연결 수는 HTTP 1.1에서 6또는 8로 제한한다.

HTTP 2에서는 하나의 단일 TCP 연결로 모든 요청에 충분하기 때문에 문제가 없다. (HTTP 2의 다중화 된 지원이 있다)



SSE — events can be broadcast to multiple clients (Image from javaee.ch)

```
00:00:00 CLIENT-> I need cakes
00:00:02 SERVER-> Have cake-1.
00:00:04 SERVER-> Have cake-2.
00:00:05 CLIENT-> Enough, I'm full.
```

6. HTTP / 2 Server Push

서버가 사전에 자산(스타일 시트, 스크립트, 미디어)을 클라이언트 캐시에 미리 푸시하는 매커니즘

Sample applications : 소셜 미디어 피드, 단일 페이지 웹

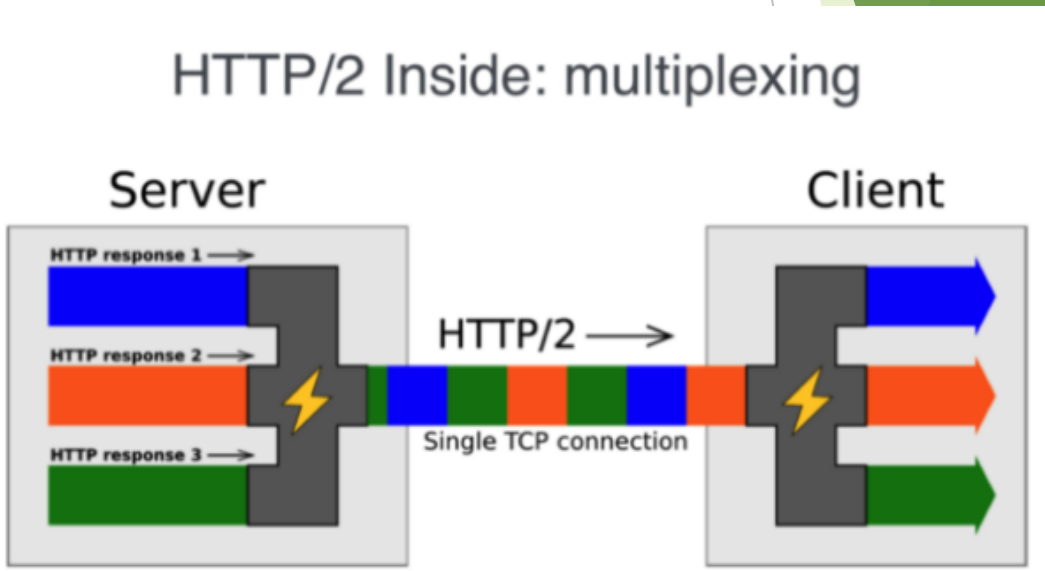
문제 1 : 중개자(프록시, 라우터, 호스트)가 원래 서버가 의도한대로 정보를 클라이언트에 올바르게 푸지하지 않도록 선택할 수 있다.

문제 2 : 연결이 무기한 연린 상태로 유지되지 않는다.

컨텐츠 푸시 프로세스가 발생하더라도 언제든지 연결을 닫을 수 있다.

일단 닫았다가 다시 열면 연결은 왼쪽에서 계속 진행할 수 없다.

문제 3 : 일부 브라우저 / 중개자는 서버 푸시를 지원하지 않는다.



HTTP/2 is an efficient transport layer based on multiplexed streams (Image from SessionStack.com) — According to IETF, a "stream" is an independent, bidirectional sequence of frames exchanged between the client and server within an HTTP/2 connection. One of its main characteristics is that a single HTTP/2 connection can contain multiple concurrently open streams, with either endpoint interleaving frames from multiple streams.

7. Web Socket

WebSockets 을 사용하면 서버와 클라이언트 모두 이전 요청과 관계없이 언제든지 메시지를 푸시 할 수 있다.

WebSockets를 사용하면 눈에 띄는 장점 중 하나는 거의 모든 브라우저가 WebSocket을 지원한다는 것이다.

WebSocket은 HTTP와 관련된 몇 가지 문제를 해결한다 :

- 양방향 프로토콜 - 클라이언트 / 서버가 상대방에게 메시지를 보낼 수 있다.
  (HTTP에서는 요청이 항상 클라이언트에 의해 시작되고 응답이 서버에 의해 처리된다. - HTTP를 단방향 프로토콜로 만든다.)
- 전이중 통신 - 클라이언트와 서버가 동시에 독립적으로 서로 통신 할 수 있다.
- 단일 TCP 연결 - 처음에 HTTP 연결을 업그레이드 한 후 클라이언트와 서버는 WebSocket 연결 수명주기 동안 동일한 TCP 연결을 통해 통신한다.

Sample application : IM / Chat 앱, 게임, 관리자 프론트 엔드

```
00:00:00 CLIENT-> I need cakes
00:00:01 SERVER-> Wait for a moment.
00:00:01 CLIENT-> Okay, cool.
00:00:02 SERVER-> Have cake-1.
00:00:02 SERVER-> Wait for cake-2.
00:00:03 CLIENT-> What is this flavor?
00:00:03 SERVER-> Don't you like it?
00:00:04 SERVER-> Have cake-2.
00:00:04 CLIENT-> I like it.
00:00:05 CLIENT-> But this is enough.
```

7. Web Socket

WebSocket은 모든 브라우저에서 지원된다고 하지만 중개자에게는 예외가 있을 수 있다.

- 중개자의 예기치 않은 동작 : WebSocket 연결이 프록시/ 방화벽을 통과하는 경우 이러한 연결이 항상 실패하는 것을 알 수 있다. 이러한 장애를 대폭 줄이려면 항상 보안 웹 소켓(WSS)을 사용하면 된다.
- WebSocket을 지원하지 않는 중개자 : 어떤 이유로 WebSocket 프로토콜을 사용할 수 없는 경우, 연결이 적절한 긴 폴링 옵션으로 자동 대체되는지 확인한다.

8. REST vs WebSockets - Perf Test

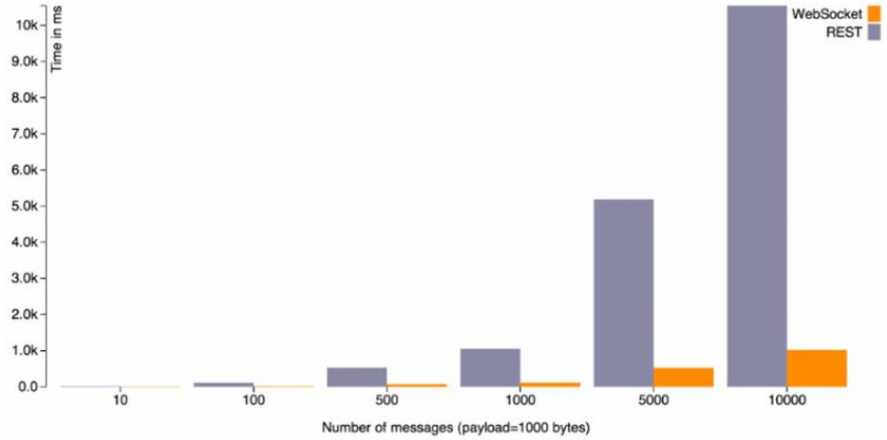
REST 및 WebSocket에 대한 성능 테스트를 수행한 결과, 로드가 많을 때 WebSocket이 더 우수하다는 것을 알 수 있다.

하지만 반드시 REST가 비효율적임을 의미하지는 않다.

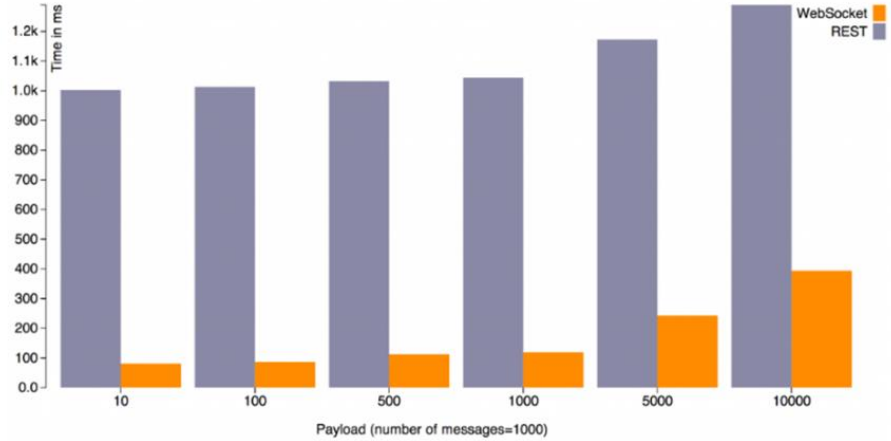
결론적으로 REST와 WebSocket을 비교하는 것은 바람직하지 않다.

이 두 기능은 서로 다른 두 가지 문제를 해결하며 간단한 성능 테스트로는 비교할 수 없다.

The first graph shows the time (in milliseconds) taken to process N messages for a constant payload size.



The second graph shows the time taken to process a fixed number of messages by varying the payload size.



WebSockets는 장기적인 양방향 데이터 스트리밍을 거의 실시간으로 처리하는 데 훌륭한 선택이 될 수 있다.

REST는 가끔 통신을 할 때 적합하다.

WebSockets을 사용하는 것은 상당한 투자이므로 가끔 연결하는데 너무 많은 비용이 든다.

9. Webhooks (for communication between servers)

데이터 변경시 API에서 데이터를 얻으려면 폴링을 가장 먼저 선택해야 한다.

그러나 서버 간 통신의 경우 폴링의 비 효율성으로 인해 많은 비용이 발생한다. (평균적으로 98.5%의 폴링이 낭비 됨)

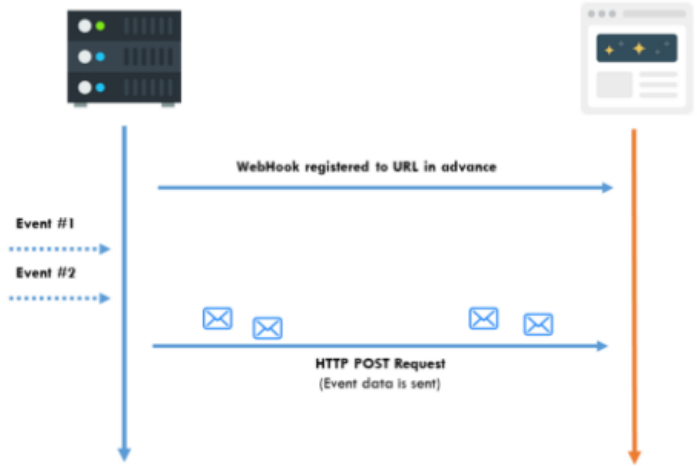
Webgook가 이 문제를 해결해 주는데, 여기서 통신은 일반적으로 서버간에 발생한다.

먼저, 발신자 노드는 수신자 노드에 콜백 URL을 미리 등록한다.

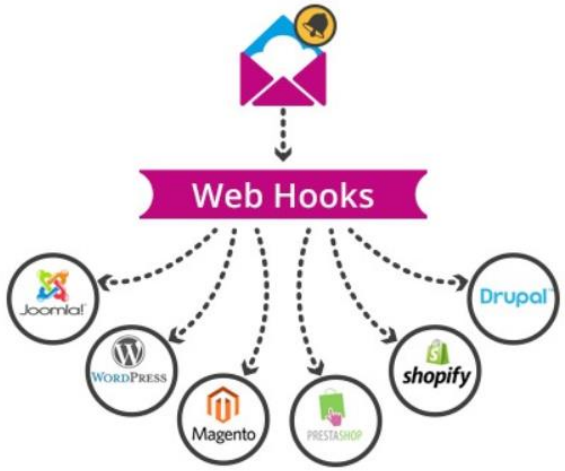
발신자 측에서 이벤트가 발생하면 WebHook가 트리거되고 각각에 등록 된 콜백 URL을 사용하여 HTTP POST 요청으로 새 데이터가 포함 된 이벤트 객체를 수신자 노드에 보낸다.

WebHook를 사용하여 발신자와 수신자 노드 모두에 대한 서버로드를 대폭 줄일 수 있다.

개발자가 폴링을 낭비하지 않고 의미 있는 일에 서비스 엔드 포인트를 활용 할 수 있다면 사용자는 더 좋은 성능을 이용할 수 있다.



Webhooks — simple way for sending data between servers with no long-lived polling connections (Image from realtimeapi.io)

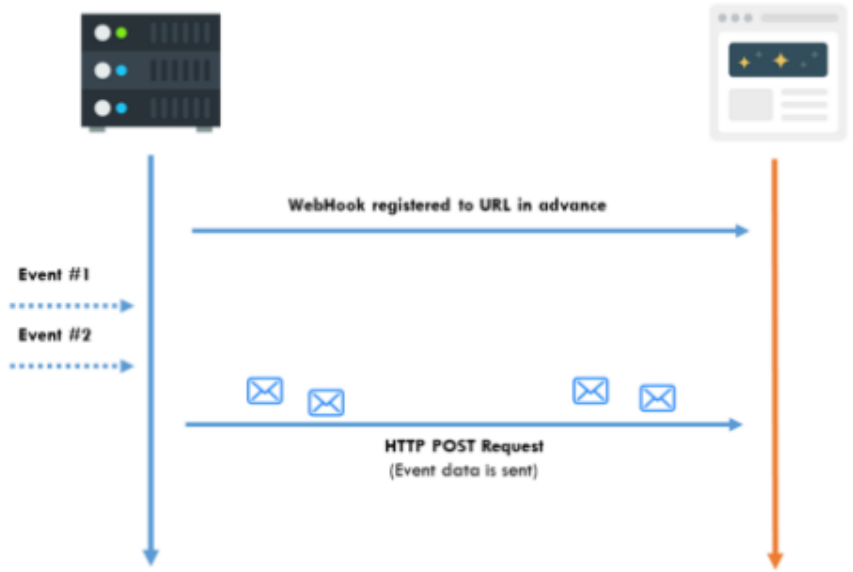


웹 후크는 일반적으로 이벤트가 발생할 때 서버간에 알림 및 상태 변경을 보내는 데 사용됩니다. 예를 들어, 사용자가 이메일에서 버튼 클릭을 통해 구독을 취소하면 서버에 도착하고 사용자 수신 거

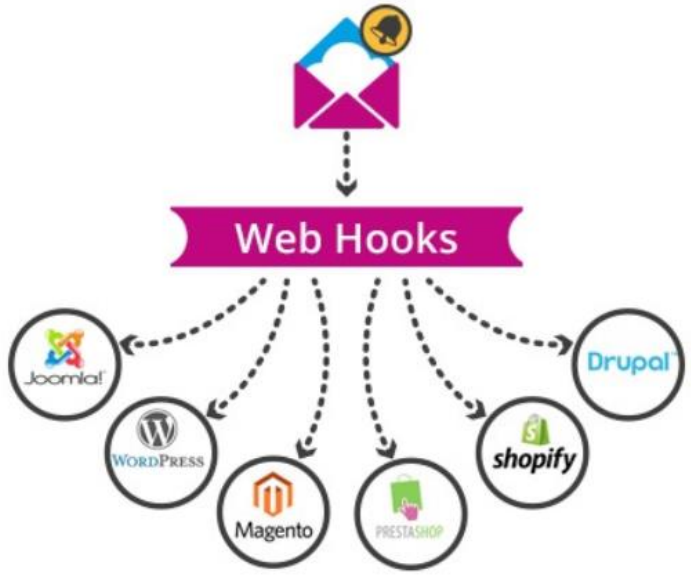
9. Webhooks (for communication between servers)

Sample application : 새로운 사용자가 등록하거나 현재 사용자가 기존 프로필 설정을 업데이트 할 때 알림

문제 1 : 개발자는 WebHook를 설정하고 HTTP 서비스를 확장하기가 어렵다



Webhooks — simple way for sending data between servers with no long-lived polling connections (Image from realtimeapi.io)



웹 후크는 일반적으로 이벤트가 발생할 때 서버간에 알림 및 상태 변경을 보내는 데 사용됩니다. 예를 들어, 사용자가 이메일에서 버튼 클릭을 통해 구독을 취소하면 서버에 도착하고 사용자 수신 거부 이벤트가 발생하면이 이벤트는 해당 웹 후크를 트리거하고 모든 서버 / 서비스에 사용자가 현재 구독을 취소했음을 알립니다 서비스 (kloudymail.com의 이미지)