How Push Works

By Matt Gaunt

Matt is a contributor to Web**Fundamentals**

Before getting into the API, let's look at push from a highlevel, start to finish. Then as we step through individual topics or API's later on, you'll have an idea of how and why it's important.

The three key steps to implementing push are:

1. Adding the client side logic to subscribe a user to push (i.e. the JavaScript and UI in your web app that registers a user to push messages).
2. The API call from your back-end / application that triggers a push message to a user's device.
3. The service worker JavaScript file that will receive a "push event" when the push arrives on the device. It's in this JavaScript that you'll be able to show a notification.

Let's look at what each of these steps entails in a little more detail.

Step 1: Client Side

The first step is to "subscribe" a user to push messaging.

Subscribing a user requires two things. First, getting **permission** from the user to send them push messages. Second, getting a `PushSubscription` from the browser.

A `PushSubscription` contains all the information we need to send a push message to that user. You can "kind of" think of this as an ID for that user's device.

This is all done in JavaScript with the Push API.

Before subscribing a user you'll need to generate a set of "application server keys", which we'll cover later on.

The application server keys, also known as VAPID keys, are unique to your server. They allow a push service to know which application server subscribed a user and ensure that it's the same server triggering the push messages to that user.

Once you've subscribed the user and have a `PushSubscription`, you'll need to send the `PushSubscription` details to your backend / server. On your server, you'll save this subscription to a database and use it to send a push message to that user.

1. Get Permission to Send Push Messages

2. Get PushSubscription

3. Send PushSubscription to Your Server

Step 2: Send a Push Message

When you want to send a push message to your users you need to make an API call to a push service. This API call would include what data to send, who to send the message to and any criteria about how to send the message. Normally this API call is done from your server.

Some questions you might be asking yourself:

- Who and what is the push service?
- What does the API look like? Is it JSON, XML, something else?
- What can the API do?

Who and What is the Push Service?

A push service receives a network request, validates it and delivers a push message to the appropriate browser. If the browser is offline, the message is queued until the the browser comes online.

Each browser can use any push service they want, it's something developers have no control over. This isn't a problem because every push service expects the **same** API call. Meaning you don't have to care who the push service is. You just need to make sure that your API call is valid.

To get the appropriate URL to trigger a push message (i.e. the URL for the push service) you just need to look at the `endpoint` value in a `PushSubscription`.

Below is an example of the values you'll get from a **PushSubscription**:

```
{
  "endpoint": "https://random-push-service.com/some-kind-of-unique-id-1234/v2/",
  "keys": {
    "p256dh" :
"BNcRdreALRFXTkOOUHK1EtK2wtaz5Ry4YfYCA_0QTpQtUbVlUls0VJXg7A8u-Ts1XbjhazAkj7I99e8Q
    "auth"   : "tBHItJI5svbpez7KI4CCXg=="
  }
}
```

The **endpoint** in this case is *https://random-push-service.com/some-kind-of-unique-id-1234/v2/*. The push service would be 'random-push-service.com' and each endpoint is unique to a user, indicated with 'some-kind-of-unique-id-1234'. As you start working with push you'll notice this pattern.

The **keys** in the subscription will be covered later on.

What does the API look like?

I mentioned that every web push service expects the same API call. That API is the **Web Push Protocol**. It's an IETF standard that defines how you make an API call to a push service.

The API call requires certain headers to be set and the data to be a stream of bytes. We'll look at libraries that can perform this API call for us as well as how to do it ourselves.

What can the API do?

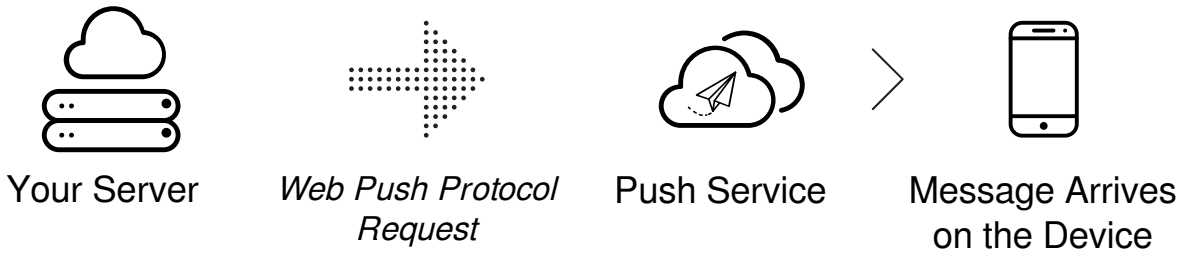
The API provides a way to send a message to a user, with / without data, and provides instructions of *how* to send the message.

The data you send with a push message must be encrypted. The reason for this is that it prevents push services, who could be anyone, from being able to view the data sent with the push message. This is important given that it's the browser who decides which push service to use, which could open the door to browsers using a push service that isn't safe or secure.

When you trigger a push message, the push service will receive the API call and queue the message. This message will remain queued until the user's device comes online and the push service can deliver the messages. The instructions you can give to the push service define how the push message is queued.

The instructions include details like:

- The time-to-live for a push message. This defines how long a message should be queued before it's removed and not delivered.
- Define the urgency of the message. This is useful in case the push service is preserving the users battery life by only delivering high priority messages.
- Give a push message a "topic" name which will replace any pending message with this new message.



Step 3: Push Event on the User's Device

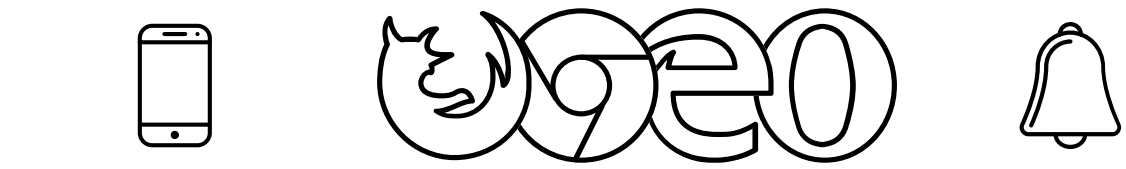
Once we've sent a push message, the push service will keep your message on its server until one of following events occurs:

1. The device comes online and the push service delivers the message.
2. The message expires. If this occurs the push service removes the message from its queue and it'll never be delivered.

When the push service does deliver a message, the browser will receive the message, decrypt any data and dispatch a **push** event in your service worker.

A service worker is a "special" JavaScript file. The browser can execute this JavaScript without your page being open. It can even execute this JavaScript when the browser is closed. A service worker also has API's, like push, that aren't available in the web page (i.e. API's that aren't available out of a service worker script).

It's inside the service worker's 'push' event that you can perform any background tasks. You can make analytics calls, cache pages offline and show notifications.



1. Message Arrives on Device

2. Browser Wakes Up Service Worker

3. Push Event is Dispatched

That's the whole flow for push messaging. Lets go through each step in more detail.

Previous
← Overview

Next
Subscribing a User →

Last updated July 2, 2018.