FAQ

By Matt Gaunt

Matt is a contributor to Web**Fundamentals**

Why doesn't push work when the browser is closed?

This question crops up quite a bit, largely because there are a few scenarios that make it difficult to reason with and understand.

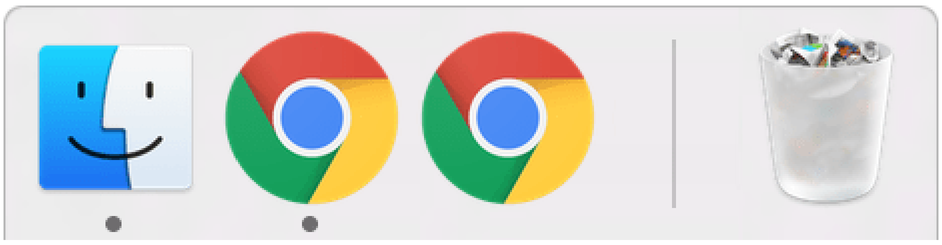
Let's start with Android. The Android OS is designed to listen for push messages and upon receiving one, wake up the appropriate Android app to handle the push message, regardless of whether the app is closed or not.

This is exactly the same with any browser on Android, the browser will be woken up when a push message is received and the browser will then wake up your service worker and dispatch the push event.

On desktop OS's, it's more nuanced and it's easiest to explain on Mac OS X because there is a visual indicator to help explain the different scenarios.

On Mac OS X, you can tell if a program is running or not by a marking under the app icon in the dock.

If you compare the two Chrome icons in the following dock, the one on the left is running, illustrated by the marking under the icon, whereas the Chrome on the right is **not running**, hence the lack of the marking underneath.



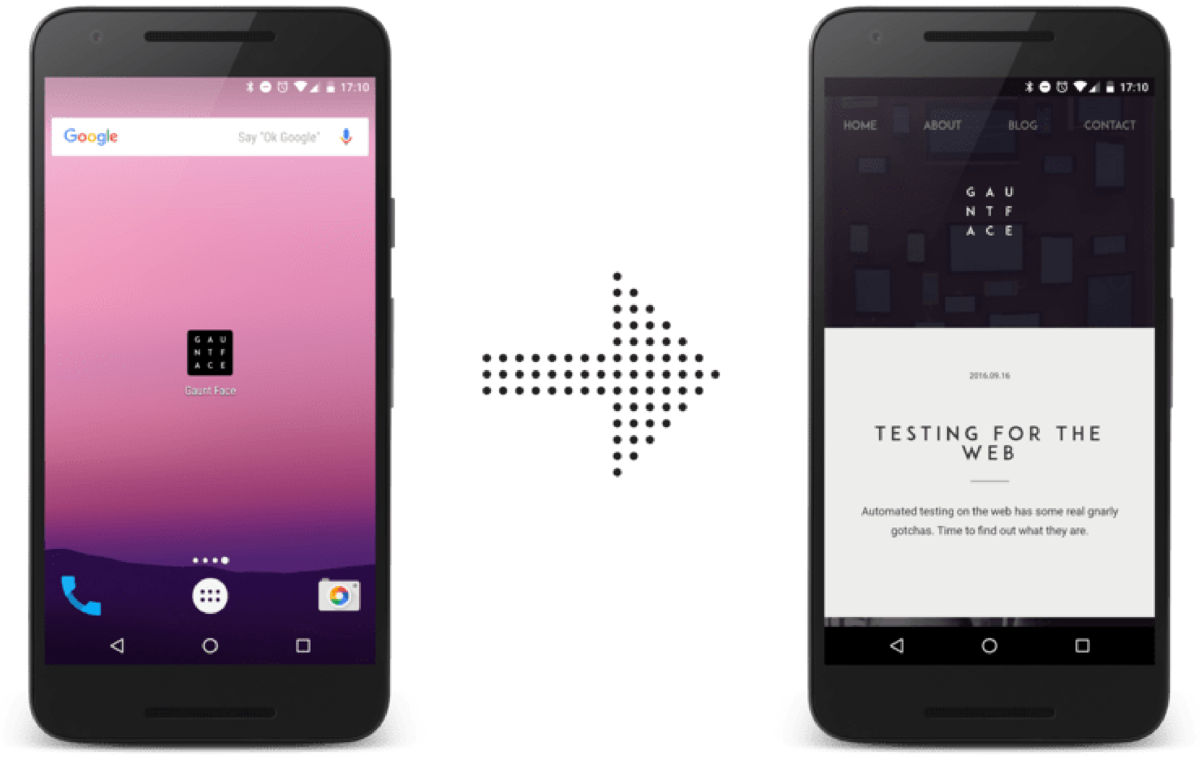
In the context of receiving push messages on desktop, you will receive messages when the browser is running, i.e. has the marking underneath the icon.

This means the browser can have no windows open, and you'll still receive the push message in your service worker, because the browser in running in the background.

The only time a push won't be received is if the browser is completely closed, i.e. not running at all (no marking). The same applies for Windows, although it's a little trickier to determine whether or not Chrome is running in the background.

How do I make my home screen web app open fullscreen from a push?

On Chrome for Android, a web app can be added to the home screen and when the web app is opened from the home screen, it can launch in fullscreen mode without the URL bar, as shown below.



To keep this experience consistent, developers want their clicked notifications to open their web app in fullscreen as well.

Chrome "sort of" implemented this, although you may find it unreliable and difficult to reason with. The relevant implementation details are:

> Sites which have been added to homescreen on Android should be allowed to open in standalone mode in response to push notifications. As Chromium cannot detect what sites are on the homescreen after they have been added, the heuristic is sites which have

been launched from homescreen within the last ten days will be opened in standalone from a tap on a notification. –Chrome Issue

What this means is that unless your user is visiting your site through the home screen icon fairly regularly, your notifications will open in the normal browser UI.

This issue will be worked on further.

Note: This is just the behavior of Chrome, though other browsers may do different things as well. Feel free to raise an issue if you have anything to add to this discussion.

Why is this any better than web sockets?

A service worker can be brought to life when the browser window is closed. A web socket will only live as long as the browser and web page is kept open.

What is the deal with GCM, FCM, Web Push and Chrome?

This question has a number of facets to it and the easiest way to explain is to step through the history of web push and Chrome. (Don't worry, it's short.)

December 2014

When Chrome first implemented web push, Chrome used Google Cloud Messaging (GCM) to power the sending of push messages from the server to the browser.

This **was not web push**. There are a few reasons this early set-up of Chrome and GCM wasn't "real" web push.

- GCM requires developers to set up an account on the Google Developers Console.
- Chrome and GCM needed a special sender ID to be shared by a web app to be able to set up messaging correctly.
- GCM's servers accepted a custom API request that wasn't a web standard.

July 2016

In July a new feature in web push landed - Application Server Keys (or VAPID, as the spec is known). When Chrome added support for this new API, it used Firebase Cloud Messaging (also known as FCM) instead of GCM as a messaging service. This is important for a few reasons:

- Chrome and Application Sever Keys **do not** need any kind of project to be set up with Google or Firebase. It'll just work.
- FCM supports the *web push protocol*, which is the API that all web push services will support. This means that regardless of what push service a browser uses, you just make the same kind of request and it'll send the message.

Why is it confusing today?

There is a large amount of confusion now that content has been written on the topic of web push, much of which references GCM or FCM. If content references GCM, you should probably treat it as a sign that it's either old content OR it's focusing too much on Chrome. (I'm guilty of doing this in a number of old posts.)

Instead, think of web push as consisting of a browser, which uses a push service to manage sending and receiving message, where the push service will accept a "web push protocol" request. If you think in these terms, you can ignore which browser and which push service it's using and get to work.

This book has been written to focus on the standards approach of web push and purposefully ignores anything else.

Firebase has a JavaScript SDK. What and Why?

For those of you who have found the Firebase web SDK and noticed is has a messaging API for JavaScript, you may be wondering how it differs from web push.

The messaging SDK (known as Firebase Cloud Messaging JS SDK) does a few tricks behind the scenes to make it easier to implement web push.

- Instead of worrying about a `PushSubscription` and its various fields, you only need to worry about an FCM Token (a string).
- Using the tokens for each user, you can use the proprietary FCM API to trigger push messages. This API doesn't require encrypting payloads. You can send a plain test payload in a POST request body.
- FCM's proprietary API supports custom features, for example FCM Topics (It works on the web too, though it's poorly documented).

- Finally FCM supports Android, iOS and web, so for some teams it is easier to work with in existing projects.

This uses web push behind the scenes, but its goal is to abstract it away.

Like I said in the previous question, if you consider web push as just a browser and push service, then you can consider the Messaging SDK in Firebase as a library to simplify implementing web push.

← Previous
Common Notification Patterns

Next
Common Issues and Reporting Bugs →

Last updated July 2, 2018.