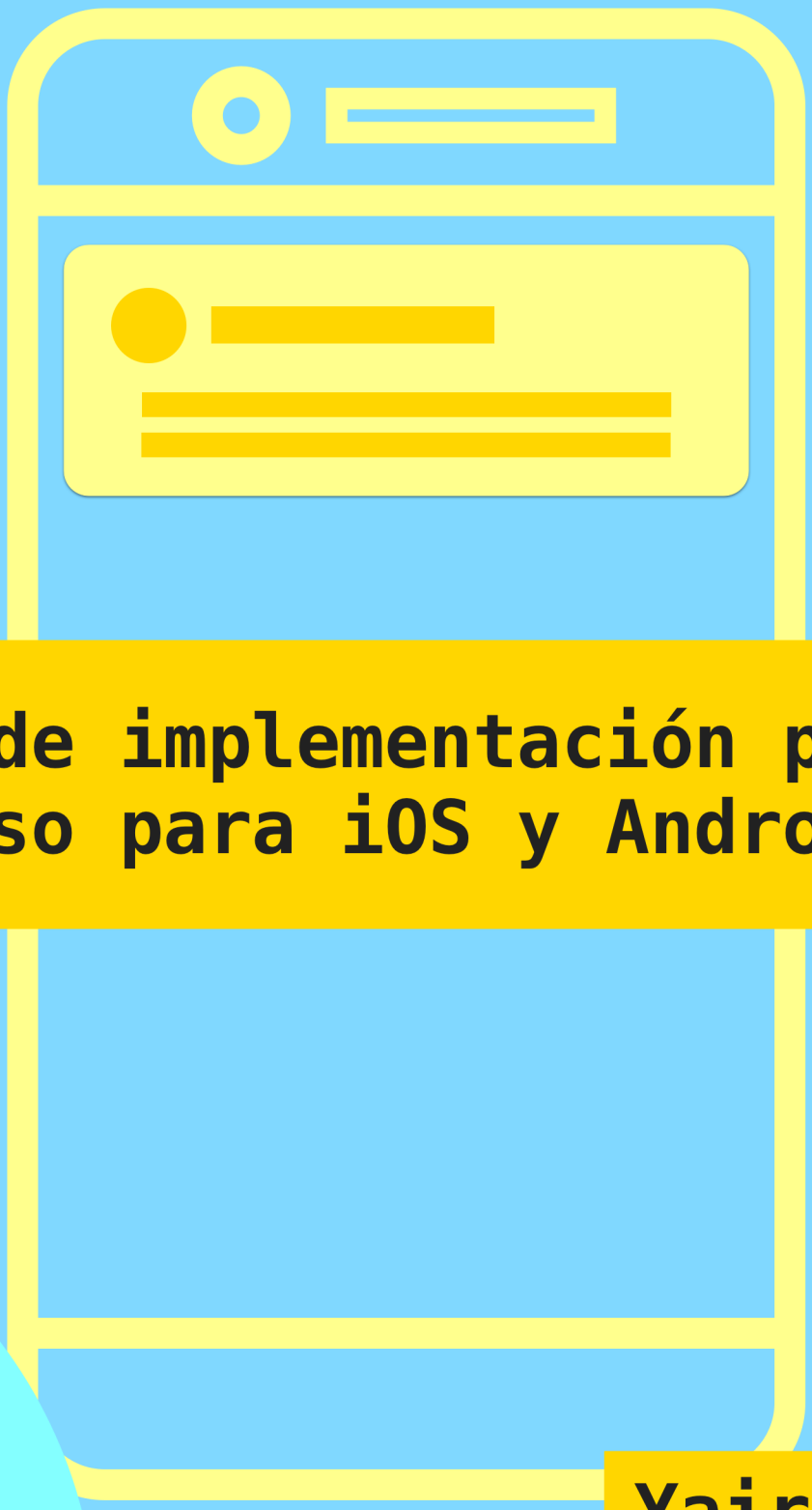
Push Notifications HandBook



Guía de implementación paso a paso para iOS y Android

2nd Edition

Actualizado para Swift, Java y Kotlin.

Yair Carreno

Manual de Push Notifications para iOS y Android

Yair Carreno

Este libro está a la venta en http://leanpub.com/push-notifications-ios-android-spanish

Esta versión se publicó en 2021-10-13

Índice general

Prefacio

Acerca del libro

Estimado lector. Es grato comunicarme contigo y emprender esta aventura por el mundo de las **Push Notifications**.

Quizás muchos estén de acuerdo conmigo al considerar *push notifications* como un mecanismo *cool* para entregar mensajes al usuario. Considero que es una de las capacidades más versátiles que tienen las aplicaciones móviles. Pero seamos honestos, a veces durante el proceso de implementación de un *feature* que incluye esta capacidad se puede complicar más de lo debido, ya sea porque la notificación no se muestra, o que de repente dejaron de mostrarse, o que reportan un *bug* muy extraño y difícil de detectar.

Pensando en ello, tengo una buena noticia para ustedes amigos lectores. Al finalizar este libro estarán de acuerdo conmigo cuando afirmo que las *Push Notifications* no solo son un excelente mecanismo de llegarle al usuario, sino que además no es tan difícil después de todo entender cómo funcionan y se implementan.

En muchas ocasiones, la solución o las respuestas están a la vista en la documentación oficial, pero nos perdemos a la hora de unir todas las piezas.

Después de muchas sesiones de trabajo con este tópico en las aplicaciones móviles, decidí compartir mis conocimientos a través este trabajo en donde se relacionan los dos sistemas operativos más importantes en aplicaciones móviles, **iOS** y **Android**. Después de todo *¿Qué aplicación móvil hoy en día no usa Push Notifications en alguna de sus funcionalidades?* Seguramente habrá un porcentaje considerable que aún no use estas capacidades, pero no cabe duda de que es cuestión de tiempo para que requieran hacerlo al notar las múltiples ventajas que éstas incorporan a una aplicación.

Con base a información recopilada desde Firebase Cloud Messaging, Apple, Blogs, eBooks e implementaciones en proyectos productivos, he organizado cada contenido para que de forma progresiva el lector vaya adquiriendo los conceptos fundamentales para posteriormente presentar el desarrollo e implementación de cada uno de dichos conceptos.

Si logro que el lector incluya este manual dentro de sus documentos de referencia, sentiría que he cumplido con mi labor con respecto a este tema.

También, animo al lector a que constantemente esté verificando las actualizaciones y mejoras en este campo que al igual que los otros campos constantemente evolucionan.

Espero amigo lector que sea de su agrado este trabajo y no siendo más, **"Let's start!"**.

Acerca del código fuente

Para desarrollar el contenido de este libro, he creado múltiples repositorios públicos con el código fuente para iOS y Android. En la **primera edición** del libro, el proyecto iOS fué implementado con *Swift* y el proyecto

Android tiene la implementación en *Java*. Para esta **segunda edición** del libro, se agrega la implementación del proyecto para Android usando *Kotlin*.

En el capítulo Implementación de Push Notifications se detalla las estrategias implementadas y el lector podrá encontrar los siguientes repositorios el código fuente de las APPs:

- Push Notifications Handbook iOS[1].
- Push Notifications Handbook Android-Java[2].
- Push Notifications Handbook Android-Kotlin[3].

Desarrollo del contenido

Se ha dividido el contenido en tres bloques principales de temas. El primer bloque es de conceptos, el segundo bloque es dedicado a configuraciones y el tercer bloque es usado para mostrar las implementaciones en las APPs.

El **primer bloque** de conceptos contiene el capítulo de Generalidades en donde se presenta la información básica para entender en su totalidad los mecanismos de envío y recepción de *push notifications*. Conceptos tales como los tipos de mensajes, el estado de la aplicación al momento de recibir el mensaje, los tipos de manejo que se le pueden dar a una notificación, patrones de diseños y estrategias, entre otros, son presentados en este capítulo de generalidades.

De este bloque también hace parte el capítulo Implementación de Push Notifications. Este capítulo presenta los pasos necesarios para implementar los patrones de diseños que se presentan en el capítulo de generalidades. Se describe cada una de las estrategias con sus pros y sus contras para que el lector en su buen criterio escoja el que considere más adecuado para sus soluciones.

Este bloque es la base para desarrollar los temas de los siguientes dos bloques.

El **segundo bloque** es dedicado a configuraciones, es decir, las actividades que el lector deberá realizar con respecto a herramientas y consolas para permitir el envío de mensajes de forma exitosa.

El primer capítulo de este bloque es Configuración del proyecto en Firebase en donde se proporcionan las instrucciones para crear un proyecto en Firebase para los casos de distribución de mensajes con FCM.

Posteriormente se presenta el capítulo Configuración de APP iOS en Apple Developer Portal. Allí se describe la información requerida y diligenciada para dar soporte de *push notifications* a los clientes iOS.

Y este bloque finaliza con el capítulo Integración de APP a Firebase que no es más que integrar las APPs de iOS y Android a las plataformas de distribución de mensajes de Firebase.

El **tercer bloque** lo conforman los capítulos de netamente implementación. Con base a los conceptos básicos y teniendo las configuraciones finalizadas, se procede a mostrar el código fuente en las aplicaciones móviles para recibir y administrar los mensajes provenientes de las plataformas de distribución de mensajes FCM o APNs. Esto se muestra en el capítulo Configuración de APP para recibir Push Notifications.

[1] https://github.com/yaircarreno/Push-Notifications-Handbook-iOS
[2] https://github.com/yaircarreno/Push-Notifications-Handbook-Android
[3] https://github.com/yaircarreno/Push-Notifications-Handbook-Android-Kotlin

Moviéndonos para el lado del *backend*, en el capítulo Creación del servidor de Push Notifications se presenta la implementación de microservicios responsables de realizar el envío de los mensajes a través de FCM o APNs. Estos servicios se construyen con un diseño *Serverless* y a través de Cloud Services en *GCP*.

Cerrando el bloque, se presenta el capítulo Herramientas de Testing para Push Notifications clave a la hora de verificar la correcta implementación del sistema *push notifications* en APPs y detectar posibles fallas, malas configuraciones o posibles *bugs* en el manejo de los mensajes.

Adicional a estos tres bloques, he querido dejarle también al lector un capítulo extra de recomendaciones, *tips* y técnicas que podrían ser de utilidad y complemento al contenido. Este capítulo es llamado Bono extra.

Glosario

Durante el desarrollo de los temas en el libro, el lector podrá encontrar algunos términos los cuales son descritos a continuación para dar mayor claridad.

FCM: Firebase Cloud Messaging, servicio de Firebase para la distribución de mensajes a clientes móviles y web.

APNS: Apple Push Notifications Service, servicio de Apple para la distribución de mensajes a clientes iOS.

Notificaciones locales: Notificaciones generadas desde un ámbito de *frontend*, es decir desde la misma aplicación cliente.

Notificaciones remotas: Notificaciones generadas desde un ámbito de *backend*, es decir desde un servidor remoto.

Audiencia

Este libro lo recomiendo para todo actor que participa en procesos de diseño, implementación y validación de componentes de software para clientes móviles en sistemas operativos iOS y/o Android. Si bien *Push Notifications* es solo uno de los tópicos en el mundo de desarrollo de soluciones móviles, es importante conocer a detalle estos mecanismo para usarlos en implementaciones *in-house*, optimizaciones o mejoras en procesos de fidelización de usuarios, entre otros usos y aplicaciones basadas en este tipo de mensajes.

Consultas y/o contacto

Este trabajo pretender guiar al lector en la búsqueda de buenas prácticas de diseño en soluciones que involucran *push notifications*. Ha sido desarrollado desde mi punto de vista y teniendo en cuenta las recomendaciones de fuentes expertas citadas en la bibliografía.

Muy seguramente el lector podría encontrar alternativas a las técnicas expuestas en este libro, que pueden variar un poco a las aquí consignadas, y esa, es precisamente la idea del presente trabajo.

Si el lector encuentra en el libro algún aspecto que merezca ser revisado son bienvenidos los comentarios y la retroalimentación que se dé al respecto. Para ello y cualquier duda o inquietud se encuentra disponible los siguientes canales:

- Email: yaircarreno@gmail.com
- Twitter: @yaircarreno[4]
- Blog: yaircarreno.com[5]

Idioma de textos en las imágenes

Por estándar, los textos y la descripción de las imágenes presentadas a lo largo del presente libro contienen textos en inglés.

Idioma en Code Snippets

Por estándar, los ejemplos de código que acompañan el presente libro se encuentran en inglés. También algunos textos principales se encuentran en inglés para mantener consistencia con la documentación oficial.

Versiones de IDEs y tecnologías usadas

Android Environment

- **IDE**: Android Studio Arctic Fox 2020.3.1
- **Deployment SDK**: API 30 - Android 11.
- **Minimum SDK**: API 21 - Android 5.0 (Lollipop)
- **Languaje**: Java and Kotlin
- **Interface**: Activities

iOS Environment

- **IDE**: Xcode 12.5
- **iOS Deployment Target**: 14.5
- **Language**: Swift 5
- **Interface**: Storyboard

Change log

Para mantener al lector informado sobre las actualizaciones y cambios en el presente libro, se proporciona el capítulo Changelog.

Futuras ediciones

Teniendo en cuenta el advenimiento de las nuevas tecnologías con UI declarativas, se proyecta una próxima edición que incluya los proyectos con SwitfUI, Jetpack Compose y Flutter. Así que, estén atentos amigos lectores.

[4] https://twitter.com/yaircarreno
[5] https://www.yaircarreno.com/

Generalidades

Introducción

Una de las capacidades en las aplicaciones móviles que más me gusta es la de permitir enviarle al usuario un mensaje sin necesidad que la aplicación se encuentre ejecutándose en primer plano. Ya sea para propósitos de alerta, de carácter informativo, para publicidad de ofertas, de marketing o para simplemente llamar la atención del usuario con respecto a un mensaje que se quiere comunicar.

Push Notifications son mensajes en tiempo real, esto le agrega un valor de inmediatez clave para ciertas funcionalidades en una aplicación.

Notificaciones remotas Vs Notificaciones locales

Son *notificaciones remotas* aquellos mensajes que son enviados desde un servidor externo, es decir en un contexto externo a la aplicación. Son *notificaciones locales* aquellos mensajes que se generan al interior de la aplicación y que son administrados por la API de sistema de eventos del sistema operativo del dispositivo.

Antes de entrar en el detalle de la implementación de *Push Notifications* tanto en clientes *iOS* como *Android*, se presenta en las siguientes secciones aquellos conceptos que considero son claves para entender el funcionamiento completo desde que se envía una notificación hasta que se recibe por parte de la aplicación para ser presentada al usuario o administrada por la aplicación.

Tener estos conceptos claros es importante tanto para la fase de diseño como para la fase de implementación, conceptos tales como el estado de la aplicación en el momento del arribo de la notificación, si la notificación es silenciosa o es de tipo alerta, si el servidor de notificaciones envía el mensaje como *notifications*, como *data*, como ambas. Estas consideraciones son relevantes a la hora de implementar *Push Notifications* y evitar posibles errores difíciles de detectar.

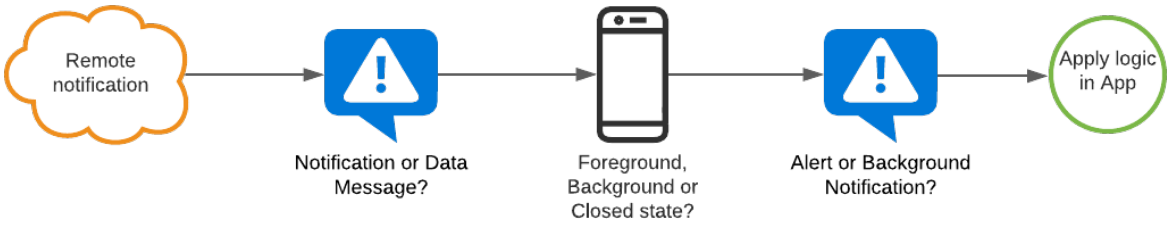


Figure 1.1 Flow receiving notification

Tipos de mensajes

El mensaje de notificación que recibe la aplicación móvil puede provenir a través de dos plataformas, **FCM** (Firebase Cloud Messaging) o de **APNS** (Apple Push Notification Service). Una de las diferencias entre uno u otro caso es el mensaje **JSON** entregado por estas plataformas.

De acuerdo a la estrategia seleccionada, la **APP** podría recibir mensajes enviados desde el servidor de notificaciones a través de FCM, APNS o ambos. Estos patrones y estrategias para envío de notificaciones es analizado en la sección de patrones que se encuentra más adelante.

Es importante conocer la estructura del mensaje enviado por el servidor de notificaciones a través de **FCM** o **APNS**, ya que dependiendo del tipo de mensaje que se reciba y el estado de aplicación, se determinará el comportamiento de la notificación. Estas conclusiones serán demostradas más adelante en las secciones Conclusiones de las pruebas sobre notificaciones en iOS y Conclusiones de las pruebas sobre notificaciones en Android.

Analicemos a continuación el mensaje entregado por cada una de estas plataformas:

Mensaje desde FCM

En **Firebase Cloud Messaging** el mensaje puede contener únicamente el objeto *Notification*, únicamente el objeto *Data* o puede contener ambos así como se muestra en el siguiente código:

Example message FCM

```
{
  "message":{
    "token":"bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
    "notification":{
      "title":"Argentina vs. Brasil",
      "body":"Great match!"
    },
    "data" : {
      "Nick" : "Mario",
      "Room" : "ArgentinaVSBrasil"
    }
  }
}
```

A los objetos *notification* y *data* también se les suele conocer como *notification payload* y *data payload* respectivamente. Es importante que el lector tenga en mente lo siguiente:

- Se envía el objeto *notification* cuando se quiere que **FCM** administre de forma automática la notificación en nombre de la aplicación.
- Se envía el objeto *data* cuando se quiere que la **aplicación** administre y tome el control sobre la notificación.

Estas reglas las resume FCM de la siguiente forma:

	Use scenario	How to send
Notification message	FCM automatically displays the message to end-user devices on behalf of the client app. Notification messages have a predefined set of user-visible keys and an optional data payload of custom key-value pairs.	1. In a trusted environment such as Cloud Functions or your app server, use the Admin SDK or the FCM Server Protocols: Set the **`notification`** key. May have optional data payload. Always collapsible. See some examples of display notifications and send request payloads. 2. Use the Notifications composer: Enter the Message Text, Title, etc., and send. Add optional data payload by providing Custom data.
Data message	Client app is responsible for processing data messages. Data messages have only custom key-value pairs with no reserved key names (see below).	In a trusted environment such as Cloud Functions or your app server, use the Admin SDK or the FCM Server Protocols: Set the **`data`** key only.

Table 1.1 General rules in FCM: Image taken from Firebase Documentation

El mensaje enviado por FCM contiene campos que son comunes y pueden ser interpretados por igual en *iOS*, *Android* o *Web*. Estos campos son precisamente los de los objetos *notification* y *data*.

- message.notification.title
- message.notification.body
- message.data

Esto quiere decir que por ejemplo el siguiente mensaje enviado desde un servidor con **Node** sería interpretado de la misma manera tanto en iOS, Android o Web:

Example message in Node sending to FCM

```
const message = {
  notification: {
    title: 'Hi notification',
    body: 'Testing my push notification.'
  }
};
```

Es posible también enviar información en el mensaje que vaya destinado únicamente a una plataforma específica. Por ejemplo, el siguiente mensaje envía información para las plataformas Android, iOS y Web a través de los objetos **android**, **apns**, **webpush**:

Example message in Node sending to FCM with specific information

```
const message = {
  notification: {
    title: 'Sparky says hello!'
  },
  android: {
    notification: {
      imageUrl: 'https://foo.bar.pizza-monster.png'
    }
  },
  apns: {
    payload: {
      aps: {
        'mutable-content': 1
      }
    },
    fcm_options: {
      image: 'https://foo.bar.pizza-monster.png'
    }
  },
  webpush: {
    headers: {
      image: 'https://foo.bar.pizza-monster.png'
    }
  },
  topic: topicName,
};
```

En el ejemplo anterior, el campo *title* es igual para todas las plataformas, pero la presentación de la imagen es particular para cada plataforma. El resultado de la configuración podría ser similar al mostrado en la siguiente imagen:

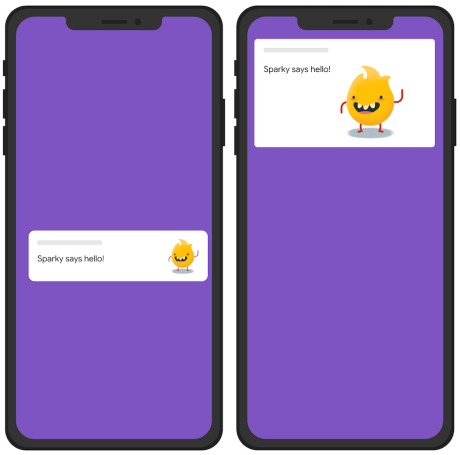


Figure 1.2 Example setting image notification. Image taken from Firebase documentation

En la siguiente documentación de Firebase Cloud Messaging se pueden encontrar más ejemplos de este tipo de configuraciones: *Build send requests*[^1].

Adicional a estos dos objetos principales *notification* y *data*, el mensaje también podría incluir más información acerca de la forma en que será distribuido. Esto se hace a través de los campos excluyentes *token*, *topic* o *condition*.

En el siguiente ejemplo, se muestra la configuración de un mensaje de tipo *notification* que será distribuido a **múltiples** dispositivos a través de sus tokens:

Example message in Node sending to FCM by tokens

```
const message = {
  notification: {
    title: 'Sparky says hello!'
  },
  tokens: ['ABCDFE...', 'OPQRSTUV...'],
};
```

En el siguiente ejemplo, se muestra la configuración de un mensaje de tipo *notification* que será distribuido a un dispositivo **específico** a través de su token:

Example message in Node sending to FCM by only one token

```
const message = {
  notification: {
    title: 'Sparky says hello!'
  },
  token: 'ABCDFE...',
};
```

En el siguiente ejemplo, se muestra la configuración de un mensaje de tipo *notification* que será distribuido a subscriptores a través de un **topic**:

Example message in Node sending to FCM by topic

```
const message = {
  notification: {
    title: 'Sparky says hello!'
  },
  topic: 'all-team',
};
```

El servidor de notificaciones usa **Firebase Admin SDK** y el **v1 HTTP protocol**[^2] para la implementación y envío de *Push Notifications*. Este servidor puede ser diseñado con microservicios escritos en *Node*, *Java*, *Phyton*, *C#* o *Go*. En la sección Creación del servidor de Push Notifications se entregará mayor detalle con códigos de ejemplo sobre estas implementaciones.

Interpretación del mensaje de FCM en Android

Mencionaba anteriormente que es importante conocer el tipo de mensaje que se envía desde el servidor de notificaciones a través de FCM. Esto es debido a que dependiendo de los tipos de objetos enviados en el mensaje, es decir *notification*, *data* o ambos, se determinará qué tipo de comportamiento tendrá la notificación remota teniendo en cuenta también el estado de la aplicación. Los estados de la aplicación es el tema tratado en la siguiente sección.

En **Android**, se cumplen las siguientes condiciones:

Caso de uso 1: Si la notificación remota requiere ser presentada únicamente cuando la aplicación se encuentra en estado *Foreground*, se recomienda usar un mensaje como el siguiente:

Example message FCM for Foreground State Android

```
{
  "message":{
    "token":"bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
    "notification":{
      "title":"Argentina vs. Brasil",
      "body":"Great match!"
    }
}
```

En este caso solamente se envía el objeto *notification* y la notificación será administrada de forma automática por FCM mostrando el *title* y el *body* con un componente UI estándar de notificación.

Sin embargo, este caso es poco común en las aplicaciones, ya que generalmente el objetivo es presentar la notificación independientemente de si la aplicación se encuentra en *Foreground*, *Background* o *Closed*. Esto nos lleva al segundo caso mostrado a continuación.

Caso de uso 2: Si la notificación remota requiere ser administrada y procesar información enviada en el mensaje cuando la aplicación se encuentre en los estados *Foreground*, *Background* o *Closed*, se recomienda usar un mensaje como:

Example message FCM for Foreground, Background or Closed State Android

```
{
  "message":{
    "token":"bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
    "data" : {
      "title":"Argentina vs. Brasil",
      "body":"Great match!"
    }
  }
}
```

En este caso solamente se incluye el objeto *data*. No puede ser incluido el *notification*, ya que de lo contrario el mensaje solo sería administrado para un estado en *Foreground*.

En resumen y teniendo en cuenta la documentación de FCM, las siguientes son las condiciones que se cumplen para **Android**:

App state	Notification	Data	Both
Foreground	• Doesn't show a notification. • **onMessageReceived**: called.	• Doesn't show a notification. • **onMessageReceived**: called.	• Doesn't show a notification. • **onMessageReceived**: called.
Background	• Show notification. • **onCreate**: called if the user clicks on the message.	• Doesn't show a notification. • **onMessageReceived**: called.	• Show notification. • **onCreate**: called if the user clicks on the message.
Closed	• Show notification. • **onCreate**: called if the user clicks on the message.	• Doesn't show a notification. • **onMessageReceived**: called.	• Show notification. • **onCreate**: called if the user clicks on the message.

Table 1.2 FCM rules for Android

Estas conclusiones y casos, serán verificados a través de un conjunto de pruebas sobre *push notifications* en la sección de implementación.

¿Cuándo se ejecuta onMessageReceived?

De la tabla anterior 1.2, se puede deducir que en Android la operación *onMessageReceived* será ejecutada cuando la aplicación se encuentra en *Background* si y solo si el mensaje recibido es de tipo *Data*. Esto es importante tenerlo en cuenta para las implementaciones en Android. En las secciones posteriores se hablará más del tema.

Interpretación del mensaje de FCM en iOS

En el caso de iOS, todos los mensajes entregados por FCM se hacen a través de APNS. Esta es una de las razones por las cuales a pesar de usar FCM también se debe configurar el envío de la notificación en APNS.

¿Es posible usar únicamente FCM para distribuir mensajes a clientes iOS sin usar APNS?

No. La distribución de mensajes a clientes iOS requerirá siempre involucrar la integración de APNS.

En la sección anterior de Android se hacía énfasis en el concepto de tipos de mensajes (*notification* y/o *data*) y cómo dependiendo de estos tipos de mensajes se podría influir en el comportamiento de la notificación cuando es recibida en la APP.

En cambio en iOS tiene mayor relevancia el concepto de tipo de notificación, esto es, *Alert push notification* y *Background push notifications*. Estos conceptos serán tema en una próxima sección llamada Tipos de notificaciones.

Sin embargo, en iOS estos conceptos se interpretan de forma similar. Un mensaje tipo *notification* o *notification + data* será interpretado como una *Alert push notification* y un mensaje tipo *data* será interpretado como una *Background push notifications*. Esto será demostrado más adelante.

Por lo pronto, se mostrarán los ejemplos de estos dos tipos de mensajes en FCM:

Example message in Node sending to FCM for Alert notification

```
const registrationToken = 'bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...';

const message = {
  notification: {
    title: 'Urgent action needed!',
    body: 'Urgent action is needed to prevent your account from being disabled!'
  },
  data: {
      key1: "some_value",
      key2: "another_value",
      key3: "one_more"
  },
  token: registrationToken
};

admin.messaging().send(message)
  .then((response) => {
    console.log('Successfully sent message:', response);
  })
  .catch((error) => {
    console.log('Error sending message:', error);
  });
```

Example message in Node sending to FCM for Background notification

```
const registrationToken = 'bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...';

const message = {
  notification: {
    title: 'Urgent action needed!',
    body: 'Urgent action is needed to prevent your account from being disabled!'
  },
  data: {
      key1: "some_value",
      key2: "another_value",
      key3: "one_more"
  }
};

const options = {
  priority: 'high',
  contentAvailable: true
};

```

```
admin.messaging().sendToDevice(registrationToken, message, options)
  .then((response) => {
    console.log('Successfully sent message:', response);
  })
  .catch((error) => {
    console.log('Error sending message:', error);
  });
```

Limitaciones de los mensajes FCM

- Para los mensajes de tipo *notification* se deben usar los valores llave-valor predefinidos en el estándar. El cambio de estos nombres podrían generar errores.
- Para los mensajes de tipo *data*, los valores llave-valor pueden ser personalizados y deben ser tenidos en cuenta a la hora de procesar la notificación en la aplicación cliente.
- El mensaje de tipo *notification* puede contener de forma opcional un *data payload*. No confundir este *payload* con el otro tipo de mensaje que hemos llamado *data*.
- Máximo es permitido un tamaño de 4KB por mensaje, ya sea que se envíe solo el *notification*, *data* o ambos.
- Para testing y envío de mensajes a través de la consola Firebase, es permitido hasta 1024 caracteres.

Mensaje desde APNS

Si la plataforma escogida para la distribución de mensajes a los usuarios iOS ha sido APNS, entonces la APP deberá prepararse para recibir mensajes con los siguientes tipos de estructuras[^3]:

Este es un mensaje de tipo Alert Push Notifications:

Example message APNS for Alert notification

```
{
   "aps" : {
      "alert" : {
         "title" : "Game Request",
         "subtitle" : "Five Card Draw",
         "body" : "Bob wants to play poker"
      },
      "category" : "GAME_INVITATION"
   },
   "gameID" : "12345678"
}
```

Este es un mensaje de tipo Background Push Notifications también conocidos como mensajes silenciosos:

Example message APNS for Background notification

```
{
   "aps" : {
      "content-available" : 1
   },
   "acme1" : "bar",
   "acme2" : 42
}
```

Este es un mensaje de tipo *Sound notification*:

Example message APNS for playing sound notification

```
{
   "aps" : {
      "badge" : 9,
      "sound" : "bingbong.aiff"
   },
   "messageID" : "ABCDEFGHIJ"
}
```

Este es un mensaje de tipo *With Localized Content*:

Example message APNS for notification with localized content

```
{
   "aps" : {
      "alert" : {
         "loc-key" : "GAME_PLAY_REQUEST_FORMAT",
         "loc-args" : [ "Shelly", "Rick"]
      }
   }
}
```

Estados de la aplicación

Como se mencionó en secciones anteriores, dentro de las estrategias usadas por la APP para la administración de las *push notifications* es clave conocer el estado en el cual se encuentra la aplicación cuando el mensaje es recibido. Dichos estados son descritos a continuación.

Foreground State

Se dice que una aplicación se encuentra en estado *Foreground* cuando al menos una de sus pantallas o servicios se encuentran visible al usuario, es decir se encuentra en ejecución en primer plano.

Background State

Se dice que la aplicación se encuentra en estado *Background* cuando ninguna de sus vistas se encuentra visible al usuario, es decir cuando la aplicación ejecuta todos sus servicios en segundo plano.

Este estado tiene ciertas restricciones para operar. Por ejemplo tener acceso limitado a la memoria o estar condicionado al nivel de batería del dispositivo. También por políticas de privacidad y transparencia al usuario algunas operaciones son restringidas en el estado background como por ejemplo acceder a la localización.

Se recomienda evitar usar estos estados para ejecutar tareas extensas y prolongadas, en su lugar puede ser usado otras alternativas como procesos con *jobs* que trabajen en segundo plano y no interfieran en la interacción con el usuario.

Closed State

Si la aplicación no se encuentra en ejecución en primer plano ni tampoco en ejecución en segundo plano, se dice entonces que es una aplicación cerrada con todos sus procesos apagados.

Determinar el estado de la aplicación

Una de las funciones que cumplen las plataformas de distribución de mensaje (FCM o APNS), es determinar el estado de la APP cuando hace la entrega del mensaje y de acuerdo a ello decide el comportamiento de la notificación entregada. Sin embargo, el lector podría agregar validaciones adicionales sobre el estado de la aplicación cuando se recibe el mensaje. Esto con el objetivo de hacer cumplir un flujo particular o un caso de uso específico.

A continuación se describe la forma programática con la cual se podría verificar el estado de la aplicación tanto en iOS como en Android.

En iOS

Para determinar los estados *foreground* y *background* de la APP en iOS, se puede emplear la siguiente función en Swift:

Checking App status in iOS

```swift
func isForeground() -> Bool {
    return UIApplication.shared.applicationState == .active
}
```

Importante que la clase delegada para la implementación de esta verificación cuente con la dependenccia:

Dependency included

```swift
import UIKit
```

En Android

Para Android, usaremos la recomendación de Google[^4] de emplear la clase *ProcessLifecycleOwner*[^5] para determinar los estados *foreground* y *background* de la APP así:

Checking App status in Android

```
private boolean inForeground() {
    return ProcessLifecycleOwner.get().getLifecycle().getCurrentState()
            .isAtLeast(Lifecycle.State.STARTED);
}
```

Es necesario instalar las siguientes dependencias[^6] en la aplicación:

Dependencies included

```
def lifecycle_version = "2.3.1"

// Lifecycles only (without ViewModel or LiveData)
implementation "androidx.lifecycle:lifecycle-runtime:$lifecycle_version"

// alternately - if using Java8, use the following instead of lifecycle-compiler
implementation "androidx.lifecycle:lifecycle-common-java8:$lifecycle_version"

// optional - ProcessLifecycleOwner provides a lifecycle for the whole application process
implementation "androidx.lifecycle:lifecycle-process:$lifecycle_version"
```

¿Qué hay acerca del estado closed/kill?

En el ámbito de la aplicación tendrá sentido determinar si el estado es *foreground* o *background*. Para los casos en los que la aplicación se encuentre cerrada totalmente el manejo de los mensajes será administrados por el sistema operativo así:

Para el caso de Android, ver la tabla anterior *Table 1.2 FCM rules for Android*. Y para el caso de iOS ver la siguiente tabla:

App state	Alert push	Background push
Foreground	• Show notification. • **willPresent**: called. • **didReceive**: called if the user clicks on the message	• Doesn't show a notification • **didReceiveRemoteNotification**: called.
Background	• Show notification. • **didReceive**: called if the user clicks on the message	• Doesn't show a notification • **didReceiveRemoteNotification**: called.
Closed	• Show notification. • **didFinishLaunchingWithOptions**, **didReceive**: called if the user clicks on the message	• Doesn't show a notification. • **didFinishLaunchingWithOptions**, **didReceiveRemoteNotification**: called if the user clicks on the message

Table 1.3 APNs rules for iOS

En el capítulo Implementando en iOS se mostrará el conjunto de verificaciones que concluyen los datos presentados en la tabla anterior.

Tipos de notificaciones

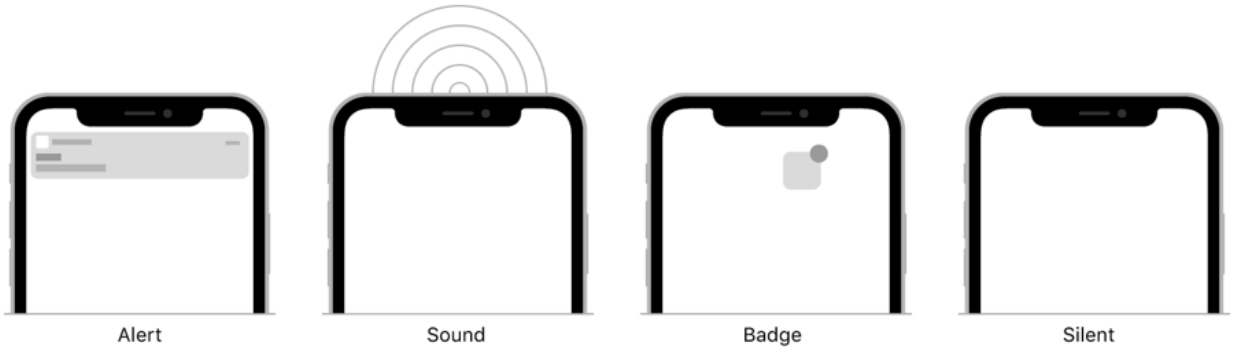


Figure 1.3 Notifications type: Image taken from Apple Documentation

Alert Push Notifications

Alert Push Notifications son mensajes visibles tanto al usuario como a la aplicación. Permiten que el usuario interactúe con la notificación y ejecute determinadas acciones según lo defina la experiencia de usuario. Este tipo de notificaciones pueden tener un diseño UI estándar o personalizado.

Son notificaciones que pueden ser ejecutadas cuando la aplicación se encuentra en estado Foreground, Background o Closed.

Background Push Notifications

También son conocidas como *notificaciones silenciosas*. Son mensajes visibles únicamente a la aplicación. El usuario no es notificado sobre estos mensajes por cuál son notificaciones silenciosas que solo son conocidas por la aplicación en un segundo plano.

Este tipo de notificaciones pueden ser usadas para activar la ejecución de tareas en segundo plano que no requieran ser informadas al usuario, como por ejemplo la actualización de un componente, la consulta en caché para optimización, entre otros casos de usos.

Al igual que Alert Push Notifications, son notificaciones que pueden ser ejecutadas cuando la aplicación se encuentra en estado *Foreground*, *Background* o *Closed*, sin embargo existen una serie de restricciones que podrían interrumpir o evitar su normal funcionamiento como por ejemplo un nivel de batería bajo o una configuración por parte del usuario que impida procesos en background. Estas limitantes deben ser tenidas en cuenta al momento de diseñar funcionalidades con este tipo de notificaciones.

Si bien el concepto de *alert push notifications* y *background push notifications* es documentado para iOS, es un concepto que se puede encontrar también en Android y ser aplicado de forma similar. En el capítulo Configuración de APP para recibir Push Notifications se presentará el detalle de implementación de este tipo de notificaciones tanto en iOS como en Android.

Patrones de implementación de Push Notifications

Existen múltiples estrategias y patrones que se pueden emplear para la implementación de un sistema de *push notifications*. En este capítulo se presentan los patrones más comunes, los cuales podrían tener variaciones dependiendo de las necesidades de la solución.

Después de analizar estos patrones, se presenta el detalle de implementación de dichos patrones expuestos en este capítulo, estos son: *Https triggers with FCM and APNS approach*, *Https triggers with APNS approach (only iOS)* y *Https triggers with FCM approach (only Android)*. La implementación de dichas estrategias pueden ser tomadas como referencia para la implementación de los otros aprovechamientos expuestos.

HTTPS Triggers with FCM and APNS Approach

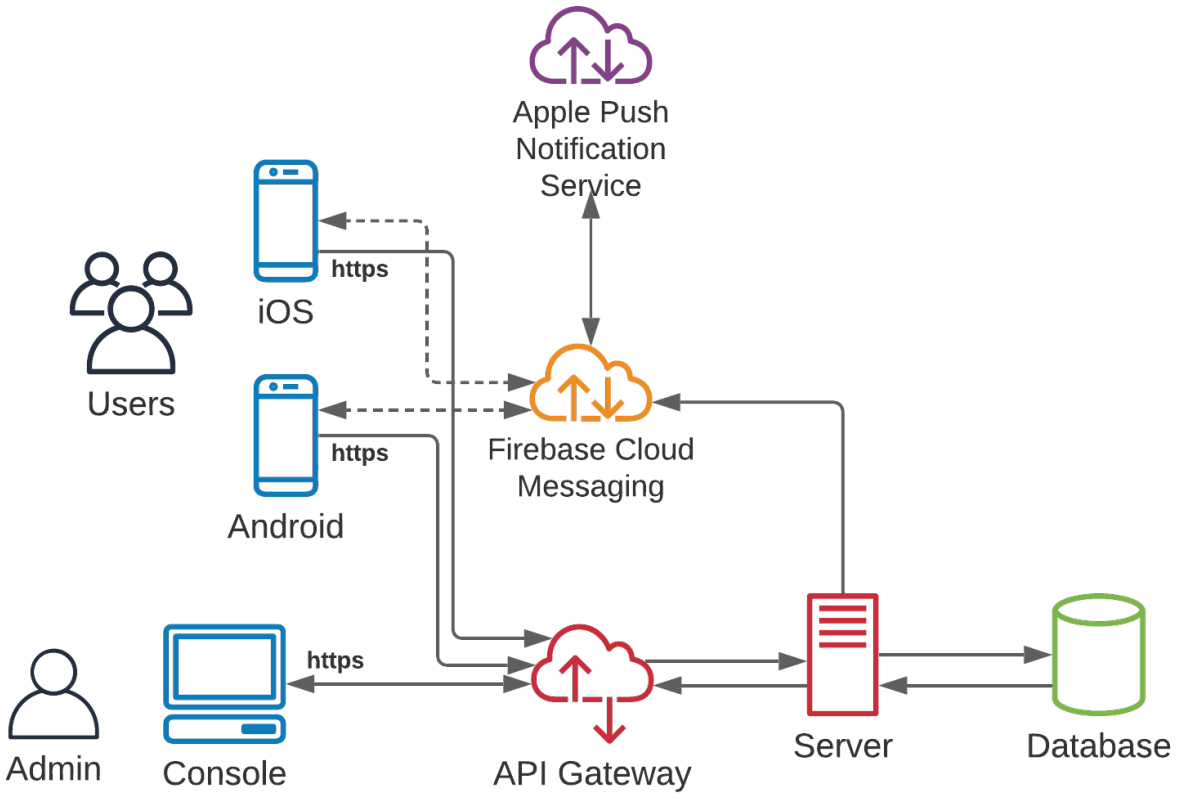


Figure 1.4 HTTPS Triggers with FCM and APNS Approach

Este patrón utiliza la plataforma **FCM** (Firebase Cloud Messaging) para la distribución de los mensajes hacia los clientes *iOS*, *Android* y *Web*.

Adicionalmente FCM se comunica con **APNS** (Apple Push Notification Service) para garantizar la entrega de las notificaciones a los clientes *iOS* de forma segura y autorizada por *Apple*.

El servidor de notificaciones **Server**, provee a través de *Microservicios* y *API Gateway* la integración para que los clientes registren sus dispositivos para recibir notificaciones, actualizar el token de uno o varios dispositivos, generar solicitudes de envío de notificaciones, configuración de las notificaciones y proveer otras funcionalidades que la solución requiera a nivel de *Push Notifications* a través por ejemplo, de una consola.

El servidor también accederá al mecanismo de persistencia **Database** que almacenará los tokens de los dispositivos registrados para ser notificados con los mensajes.

Database también podría almacenar configuraciones particulares para la notificación y requeridas por **Server**.

HTTPS Triggers with Independent FCM and APNS Approach

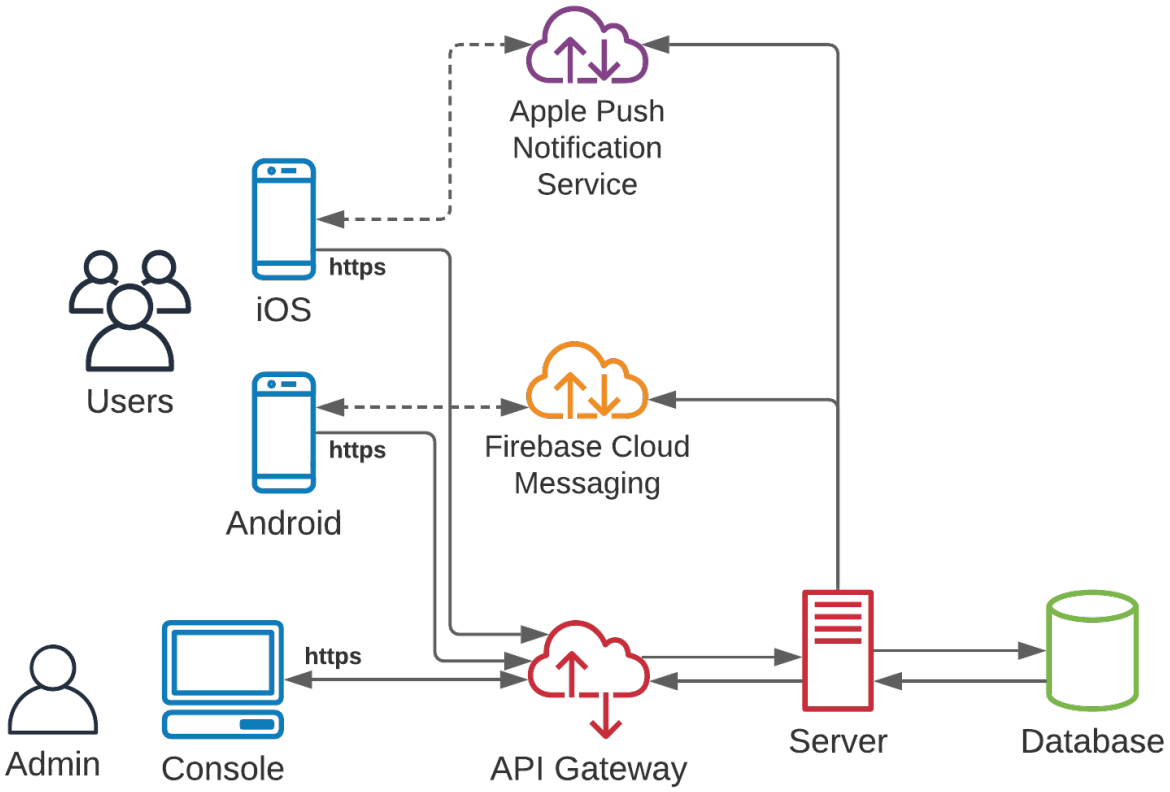


Figure 1.5 HTTPS Triggers with Independent FCM and APNS Approach

Este patrón utiliza la plataforma **FCM** para distribuir los mensajes a los clientes *Android* y utiliza la plataforma **APNS** para distribuir los mensajes a los clientes *iOS*. Se diseña la distribución de *Push Notifications* usando de forma independiente ambas plataformas a diferencia del patrón anteriormente expuesto.

En este patrón los componentes *API Gateway*, *Server* y *Database* cumplen las mismas funciones descritas en el patrón anterior, con la diferencia que en esta estrategia *Database* almacenará tanto los tokens generados por FCM como los tokens generados por *APNS*.

HTTPS Triggers with APNS Approach

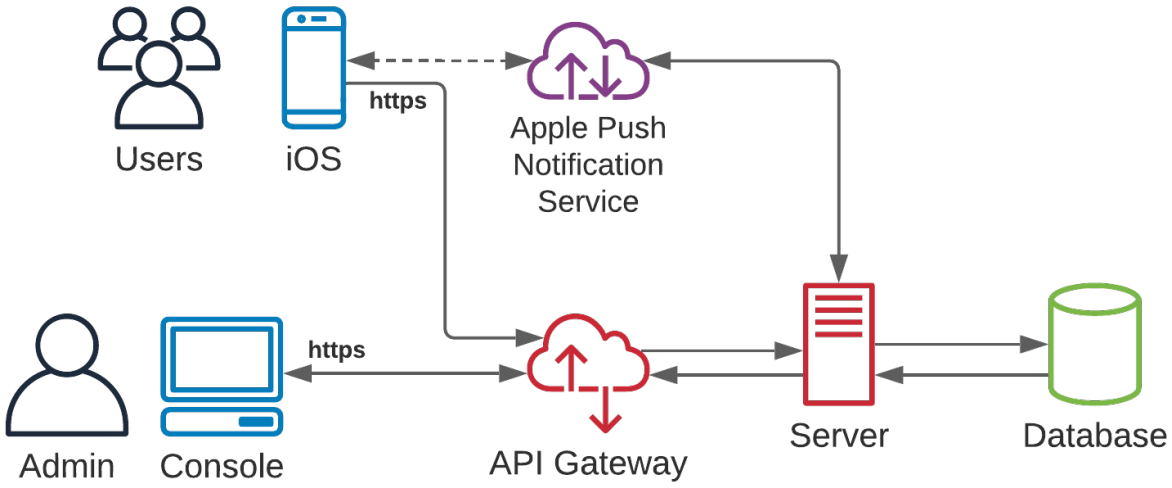


Figure 1.6 HTTPS Triggers with APNS Approach

Este patrón es oportuno cuando se requiere proveer de notificaciones únicamente a los usuarios *iOS*. En este caso únicamente es usado **APNS** como plataforma para gestionar y distribuir los mensajes.

Para recordar

Recordemos que también a través de *FCM* se puede proveer de notificaciones a los clientes *iOS*. Sin embargo si no se tiene dentro de los planes a corto y largo plazo atender a usuarios *Android* un aprovechamiento más acertado es emplear *APNS*.

HTTPS Triggers with FCM Approach

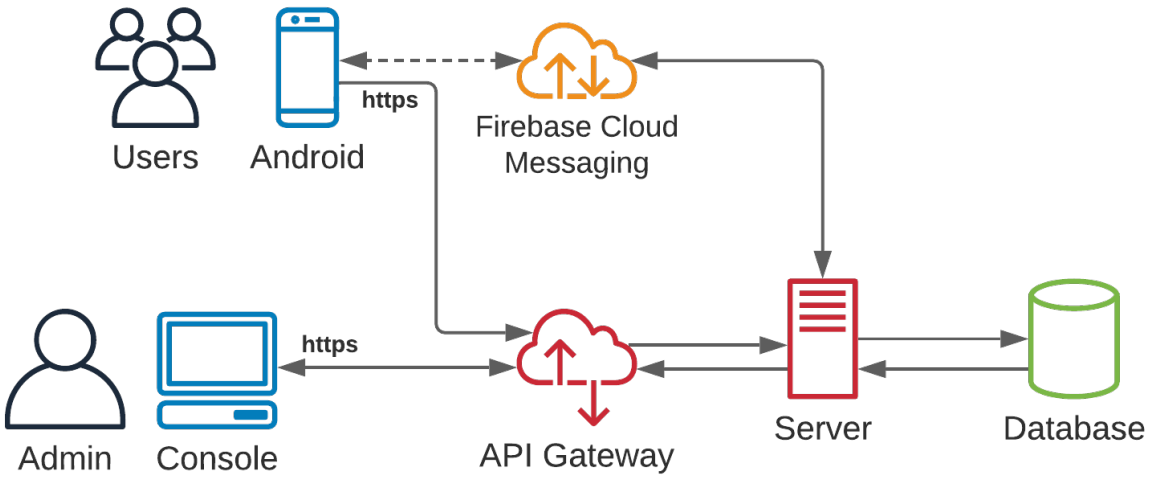


Figure 1.7 HTTPS Triggers with FCM Approach

Este patrón es oportuno cuando se requiere proveer de notificaciones únicamente a los usuarios *Android*. En este caso únicamente es usado **FCM** como plataforma para gestionar y distribuir los mensajes.

Event-Driven Approach

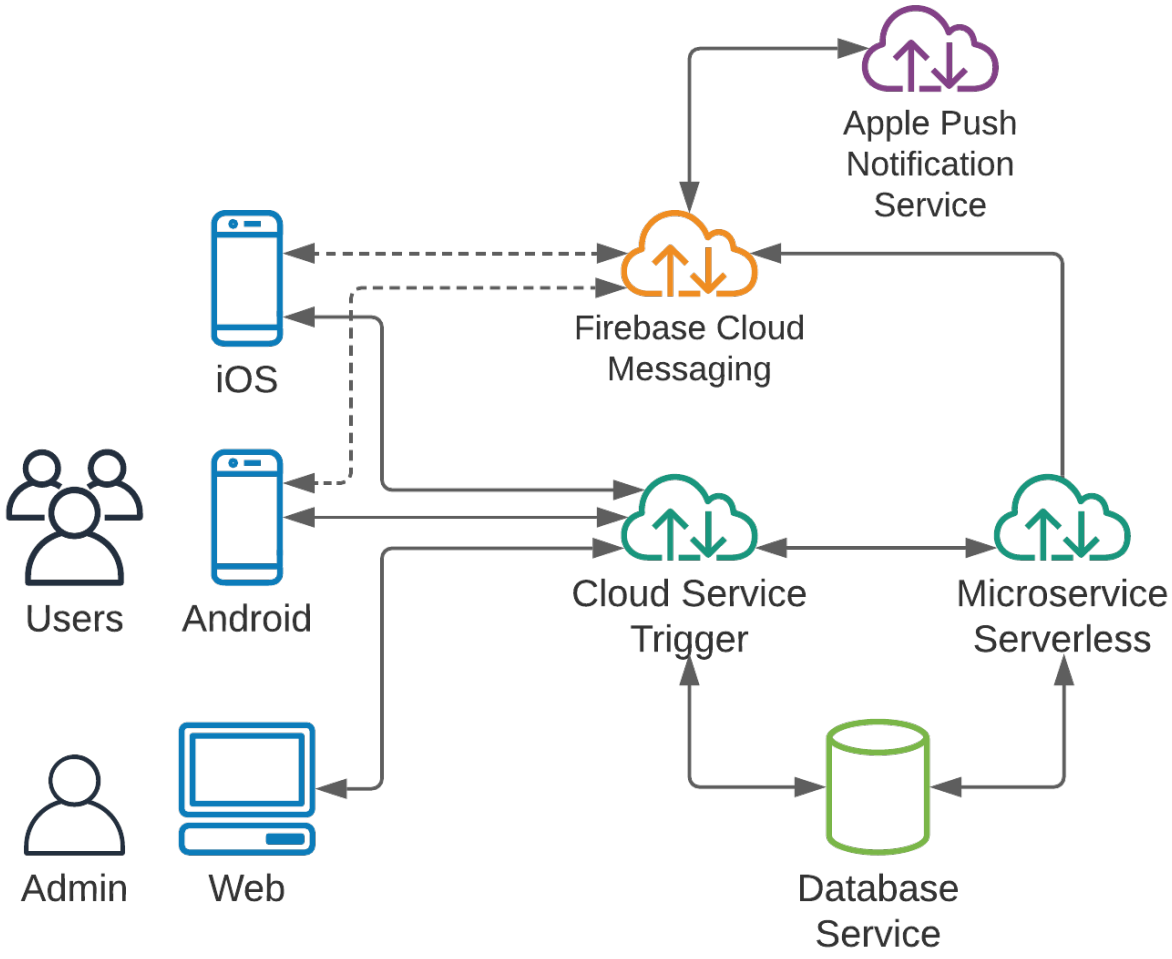


Figure 1.8 Event-Driven Approach

Este patrón es un diseño con aprovechamiento *Serverless* en donde las acciones de ejecución no solo provienen desde *http triggers*, sino además de *triggers* originados desde las bases de datos, funciones serverless (*Cloud Functions* or *Lambdas*), o procesos *batch* programados y automatizados.

En este patrón es usado **Database Service** para almacenar los tokens y las configuraciones del sistema de notificaciones requeridas. Estos servicios podrían ser bases de datos tales como *Firestore*, *DynamoDB*, *Realtime Database*, *RDS*, *Cloud SQL*, entre otros.

Como **Microservice Serverless** podrían emplearse servicios disponibles con *FaaS* tales como *Cloud Functions* o *Lambdas* o contenedores serverless tales como *Cloud Run* o *Fargate*.