Mejora. Plataforma PUSH

1. Resumen ejecutivo

1.1 Plataforma corporativa de envío PUSH

Se trata de una solución integral para el envío masivo de notificaciones PUSH a dispositivos móviles de múltiples plataformas, permitiendo el envío a usuarios concretos, a grupos o a todos los usuarios de una APP. Esta solución puede ser desplegada en el entorno del cliente para su uso exclusivo como un servicio de alta demanda, aunque también puede utilizarse como servicio en la nube.

1.2 Integración con servicios de la organización

Este servicio ha sido diseñado para un uso genérico del servicio de notificaciones, facilitando la integración con cualquier aplicativo gracias a su API de envío de notificaciones y registro/des-registro de dispositivos.

Concebido como un sistema basado en servicios, hace posible la conexión con servicios muy heterogéneos tecnológicamente.

1.3 Tipos de notificaciones

Esta plataforma abarca todas las características estándar proporcionadas por los distintos proveedores (Apple, Google…): uso de notificaciones sonoras, texto, botones o el indicador numérico. Adicionalmente, se ha completado con funciones más avanzadas de envío de metadatos, notificaciones programadas y envío de notificaciones a grupos.

1.4 Seguridad

Todas las comunicaciones realizadas desde que se solicita el envío de la notificación a este servicio hasta que es entregada al dispositivo viajan a través de conexiones seguras SSL. Adicionalmente, las funciones de envío de notificaciones están protegidas mediante API key para impedir suplantación de identidades dentro de la propia organización.

1.5 Arquitectura y escalabilidad

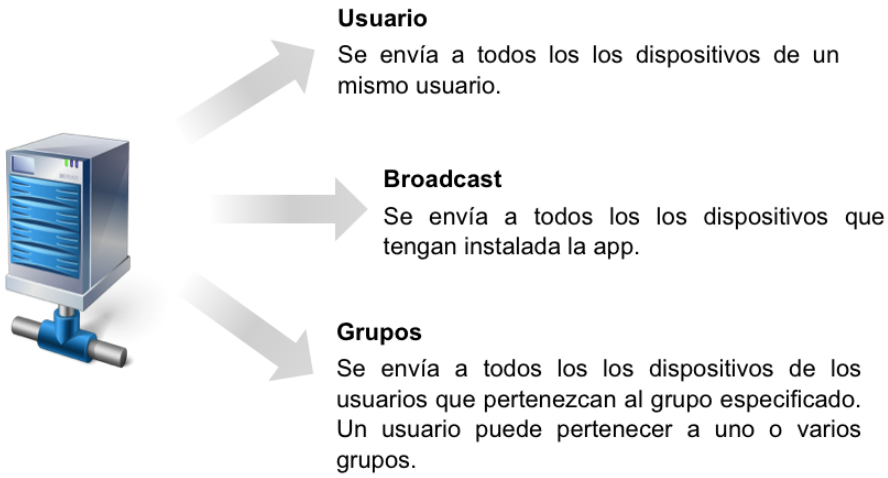
La plataforma consta de diferentes partes desarrolladas cada una en la tecnología más adecuada dependiendo de la función que desempeña.

- La API y el panel de gestión han sido desarrollados sobre Codeigniter, el framework PHP que mejor rendimiento presenta.
- La gestión de la cola de notificaciones se desarrolla sobre RabbitMQ, una implementación del estándar AMQP, lo que permite una gran escalabilidad y rendimiento en entornos de alta demanda.
- Los procesos que consumen las notificaciones de la cola y las envían a los distintos proveedores están implementados en NodeJS, ya que está orientado a eventos y permite paralelizar el envío con una sobrecarga mínima. Esta solución permite minimizar considerablemente los tiempos de envío de grandes cantidades de notificaciones y una mejor escalabilidad de toda la plataforma.

2. Características funcionales de nuestra solución

2.1. Modos de envío

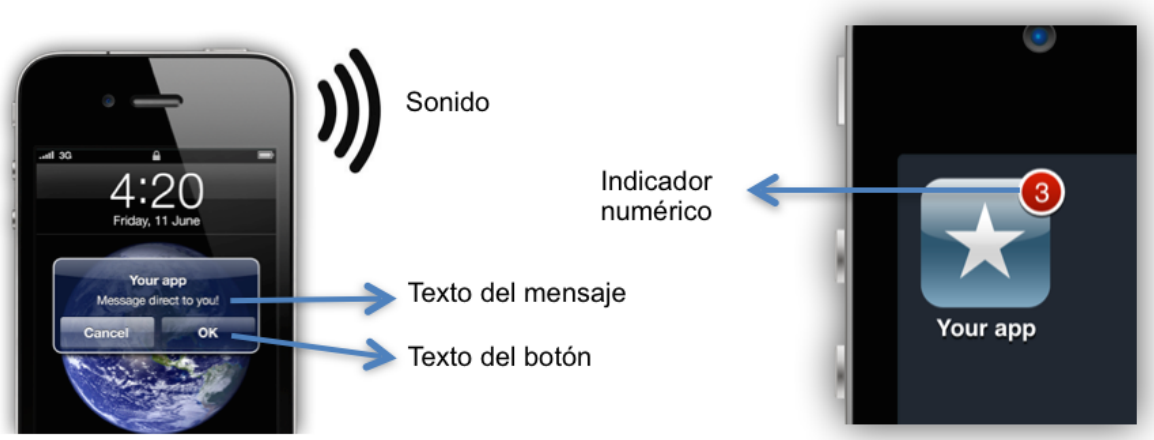
El sistema es capaz de enviar notificaciones a uno o varios dispositivos mediante distintos modos de envío:



2.2. Personalización de las notificaciones

Además del mensaje, las notificaciones pueden contener varios campos opcionales que permiten especificar ciertas características de las mismas:

- Sonido que se reproduce en la recepción
- Texto del botón que abre la notificación
- Indicador numérico en el icono (badge), aunque sólo está disponible para dispositivos iOS.



2.3. Metadatos

Permite incorporar ciertos datos adicionales en una notificación. Estos datos son invisibles para el usuario, pero permiten ajustar el comportamiento de la app en función de los datos asociados a la notificación.

Paquete enviado

```
Alert: "Hay una oferta cerca
de ti."
Sound: "bing"
Button: "Ver oferta"
Metadata:
{
   "id_categoria" : "moda",
   "id_oferta" : 138462,
   "id_comercio" : 128,
   "descuento" : "30%"
}
```

Acción en la app

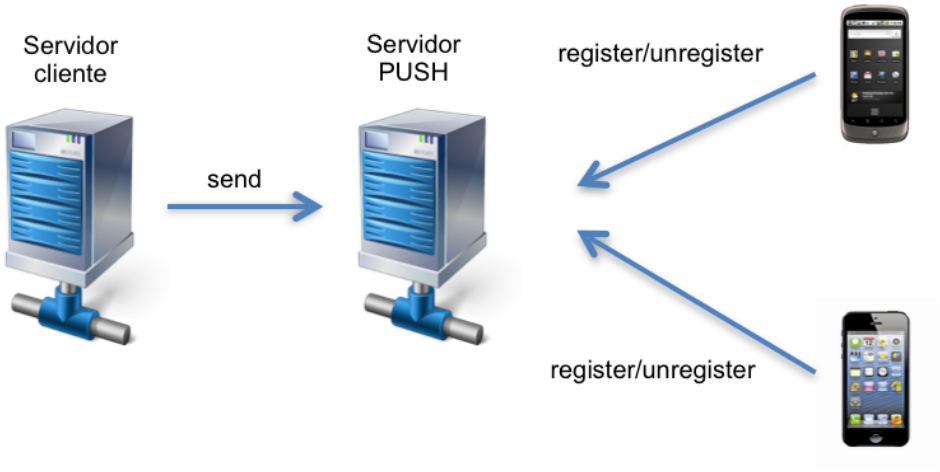
2.4. Notificaciones programadas

También resulta posible establecer la fecha y hora en la que será enviada la notificación, de manera que podemos generar las notificaciones antes de su entrega en el momento que más nos interese.

2.5. API

La integración con las plataformas cliente se realiza mediante el uso de una API de envío de notificaciones, de modo que puede integrarse con sistemas basados en tecnologías muy diversas.

Por otra parte, el registro y des-registro de los dispositivos en el servidor push, también se realiza a través de métodos de dicha API



En un entorno con usuarios diferenciados y notificaciones específicas para cada usuario, lo primero que debe realizarse es el acceso o login de dicho usuario en la aplicación asociada a estas notificaciones. De esta manera, obtenemos un identificador de usuario y un nombre de aplicación que utilizaremos como dirección de envío.

En segundo lugar, para que un dispositivo móvil pueda recibir notificaciones, la aplicación móvil los da de alta en sus respectivos servidores[1], dependiendo del tipo de dispositivo:

- iOS: Se registra en el servidor APNS de Apple, obteniendo un token que lo identifica para el envío de mensajes.
- Android: Se registra en el servidor GCM de Google, obteniendo un token que lo identifica para el envío de mensajes.

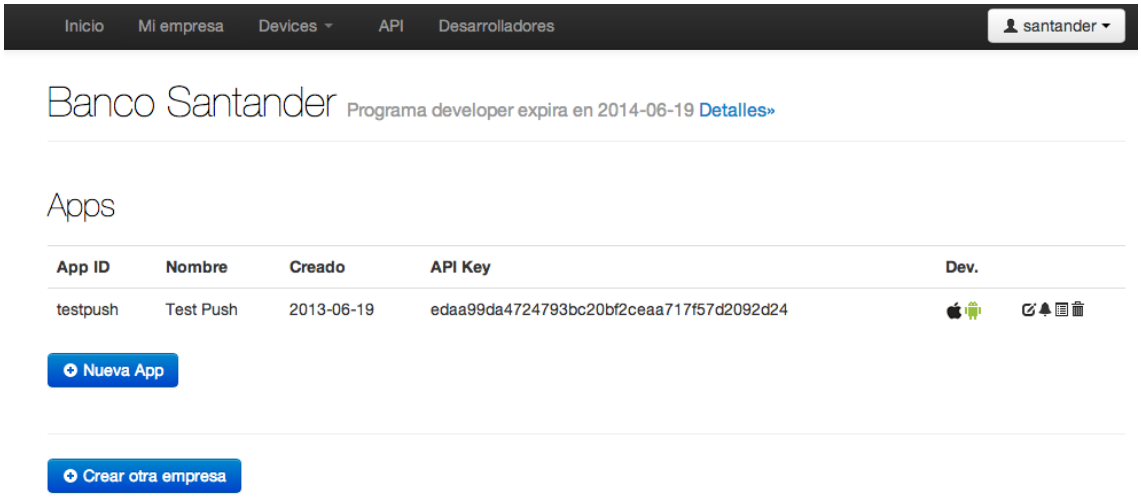
Una vez que tenemos el token que identifica a un dispositivo, éste se asocia al identificador de usuario y nombre de aplicación. De esta manera, conseguimos que el envío de mensajes posteriormente resulte mucho más cómodo para el resto de aplicaciones que lo utilicen, ya que permite abstraerse de la gestión de los tokens y resulta mucho más natural. El envío se realiza simplemente indicando el usuario, la aplicación y el mensaje que queremos enviar. Además, el token puede cambiar, caso que también considera el servidor push, que actualiza el token asociado manteniendo siempre el par usuario-aplicación. Este mecanismo permite incluso el envío de notificaciones a varios dispositivos de un mismo usuario.

En el caso en el que se envíe un mensaje a un identificador de usuario y una aplicación, el servidor push comprobará qué dispositivos tiene registrados con esos datos y enviará el mensaje a cada uno de los token asociados. Dependiendo del tipo de dispositivo, enviará el mensaje a un servidor APNS si es iOS, o bien un servidor GCM si es Android. Estos servidores serán los que se encarguen finalmente de enviar los mensajes a los dispositivos.

2.6. Gestión

La plataforma cuenta con un front-end de gestión, que permite administrar las aplicaciones que recibirán las notificaciones, sus correspondientes certificados, monitorizar los dispositivos registrados, registros del estado de los mensajes en entornos de depuración, envío manual de notificaciones, etc.

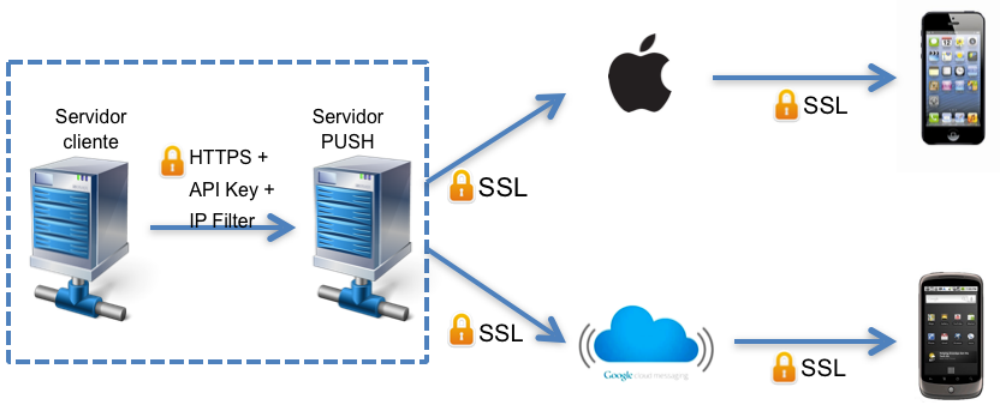
[1] Este registro lo realiza la aplicación móvil sin intervención del usuario, resulta totalmente transparente.



3. Seguridad

En entornos donde las notificaciones contienen datos sensibles, es preciso asegurar que no puede ser interceptada por usuarios a los que no va dirigida y que sólo el usuario al que va dirigida será capaz de visualizarla correctamente. Para ello se requiere que todas las conexiones que se realizan desde que la notificación se genera en el servidor cliente, hasta que los distintos proveedores envían la notificación a los dispositivos correspondientes, se realicen de forma segura para evitar ataques del tipo "Man in the middle". Este requerimiento queda cubierto debido a que todas las conexiones que se realicen entre cada uno de los nodos de la arquitectura se hace mediante conexiones SSL.

Adicionalmente, en la comunicación entre el servidor (o servidores) cliente, estará protegida mediante un API key y un filtrado de IP. Esto garantiza que dentro del entorno en el que se encuentran estos sistemas un sistema no puede suplantar a otro en el envío de notificaciones.



4. Arquitectura del proyecto

Podemos distinguir 3 bloques generales en la arquitectura del proyecto:

- **B1: Administrador web y controlador de API REST**

Es la aplicación web que permite gestionar las distintas aplicaciones, generar nuevas API keys para utilizar el servicio, administrar la plataforma y revisar los logs o bitácoras de seguimiento para cada una de las aplicaciones.

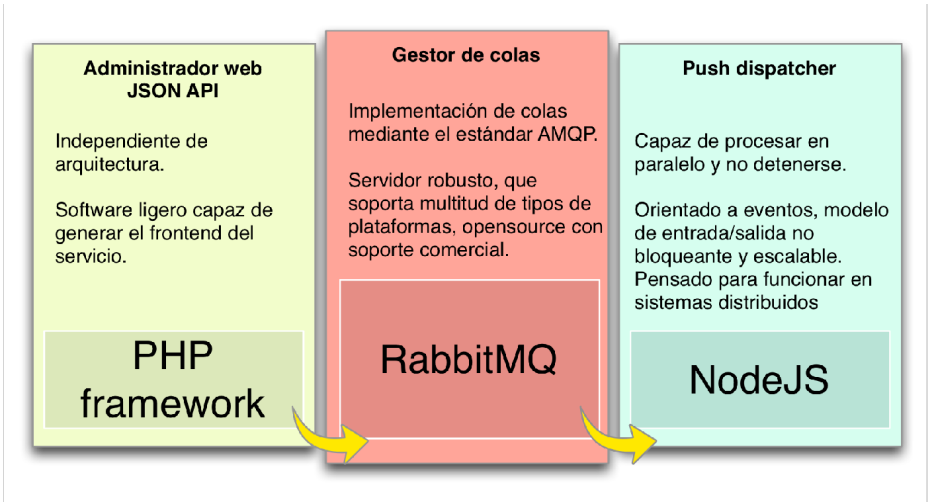
Además, unido a esto se encuentra el controlador de API JSON que provee de la comunicación con los clientes externos que quieran utilizar este servicio así como el método universal para registrar los dispositivos en el servidor de envío push.

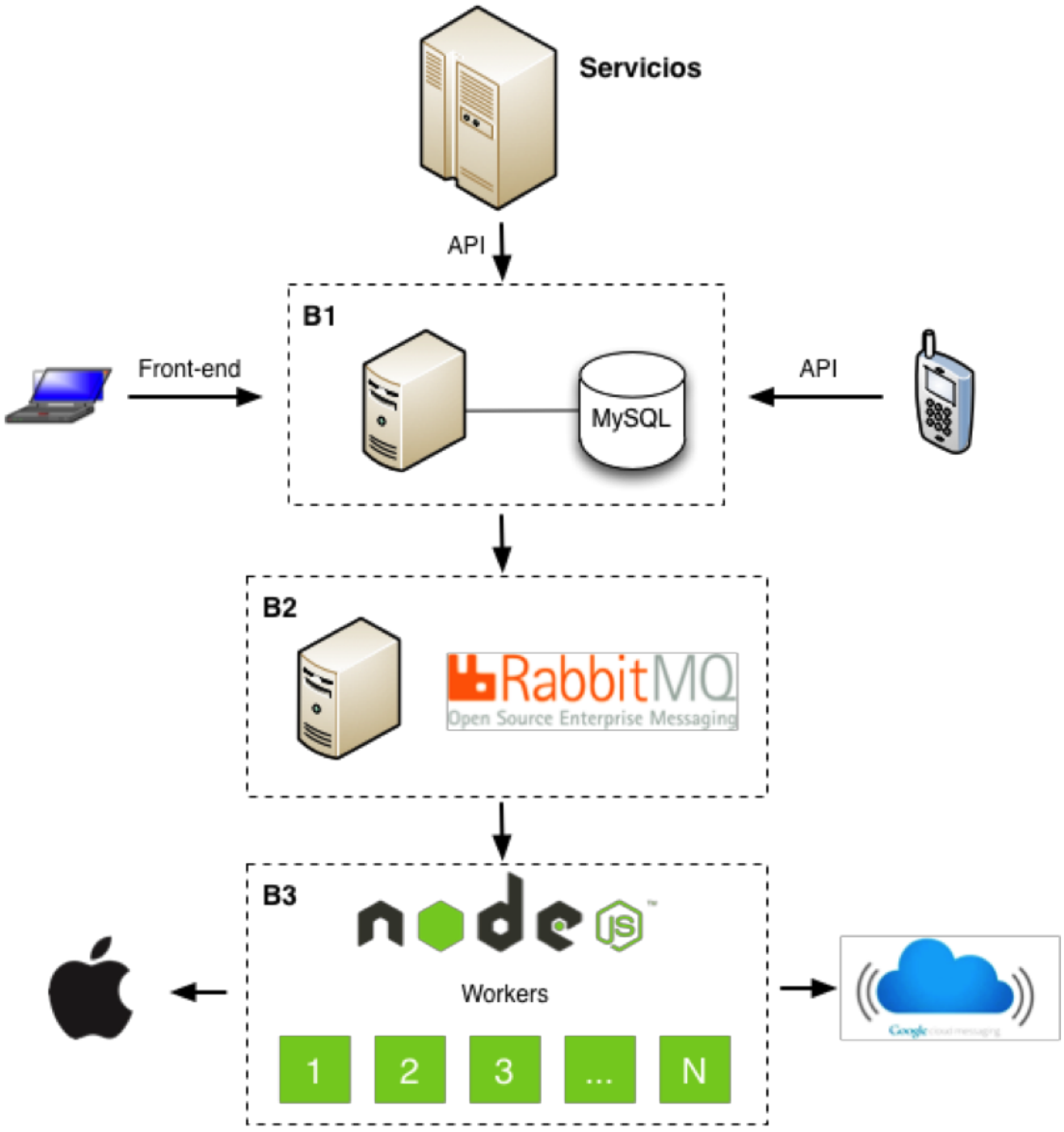
- **B2: Gestor de colas de proceso**

Es el proceso que trata los mensajes que van llegando para dejarlos en la cola de envío para ser procesados por uno o más procesos dispatcher.

- **B3: Dispatcher o procesador de envío de PUSH**

Estos procesos permiten recoger un mensaje de la cola y encargarse de enviarlo a Google o a Apple para ser recibido por un dispositivo móvil. Pueden existir uno o más procesos dependiendo de la configuración y la capacidad de paralelización de procesos que tenga disponible el servidor (número de CPU’s).


Administrador web
JSON API
Independiente de arquitectura.
Software ligero capaz de generar el frontend del servicio.
PHP framework
Gestor de colas
Implementación de colas mediante el estándar AMQP.
Servidor robusto, que soporta multitud de tipos de plataformas, opensource con soporte comercial.
RabbitMQ
Push dispatcher
Capaz de procesar en paralelo y no detenerse.
Orientado a eventos, modelo de entrada/salida no bloqueante y escalable. Pensado para funcionar en sistemas distribuidos
NodeJS



A continuación describiremos la arquitectura de cada bloque y sus características.

B1: Administrador web y controlador para JSON API

Las características deseables para este bloque son las de un framework web ligero, de escaso impacto en los recursos del sistema, de fácil mantenimiento, fácilmente extensible y modular.

A continuación se muestra una tabla comparativa de los lenguajes de programación de reconocida experiencia en aplicaciones web de alta demanda que se han considerado para el desarrollo:

Características	JAVA Frameworks	PHP Frameworks	Ruby On Rails	PERL
Bajo impacto en los recursos del sistema	✗	✓	✓	✓
Agilidad en la programación	✗	✓	✓	✗
Fácil mantenimiento	✓	✓	✓	✗
Escalable	✓	✓	✓	✓
Modular	✓	✓	✓	✗
Ampliamente usado y probado	✓	✓	✗	✓

Se ha seleccionado el lenguaje PHP por disponer de todas las características deseables para la implementación de administrador web así como de la construcción del API JSON, aunque si obviamos algunas de las características de la tabla, este bloque podría ser adaptado a cualquier otro entorno de programación web que se requiera.

Existen importantes proyectos web realizados sobre el lenguaje PHP que utilizan diferentes herramientas para acelerar los tiempos de respuesta y la alta disponibilidad. Entre estos proyectos web podemos reconocer a:

- Facebook
- Wikipedia
- Yahoo
- Photobucket
- Digg

Para favorecer el mantenimiento y la modularidad se ha utilizado un framework PHP reconocido y de gran uso en la comunidad PHP, CodeIgniter es un framework web MVC, rápido, modular y de escaso consumo de memoria.

Esto permite un desarrollo rápido y un mantenimiento ágil de las características del administrador web que permite la configuración de las aplicaciones y del sistema, operaciones con los certificados para el envío de mensajes y ampliar el API JSON cuando sea necesario.

También se ha seleccionado un API en formato JSON ya que la generación e interpretación de las respuestas se realiza más rápidamente además de optimizar las transmisiones al ocupar menos.

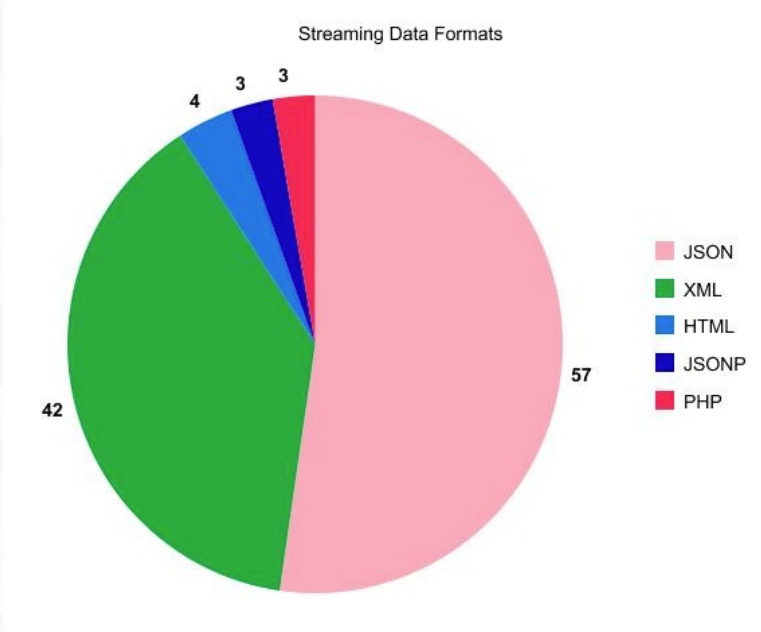


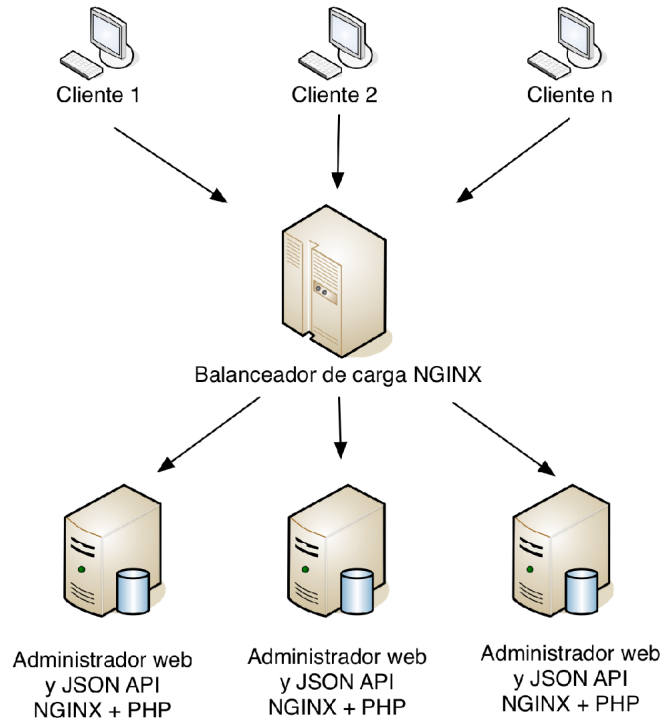
Gráfico de uso de formatos de transmisión en servicios web actuales

Independencia del bloque y escalabilidad

Este bloque de la arquitectura se comunica con el resto a través del gestor de colas utilizando el protocolo AMQP, permitiendo además independizar esta parte. El framework web utilizado permite utilizar Apache y NGINX aunque se puede usar cualquier otro servidor HTTP con soporte PHP a través de FastCGI.

Balanceador de carga

Es posible configurar NGINX como proxy inverso que, a través de la directiva *upstream* podría balancear la carga de peticiones a uno o varios servidores con distintos pesos y configuración ante fallos.



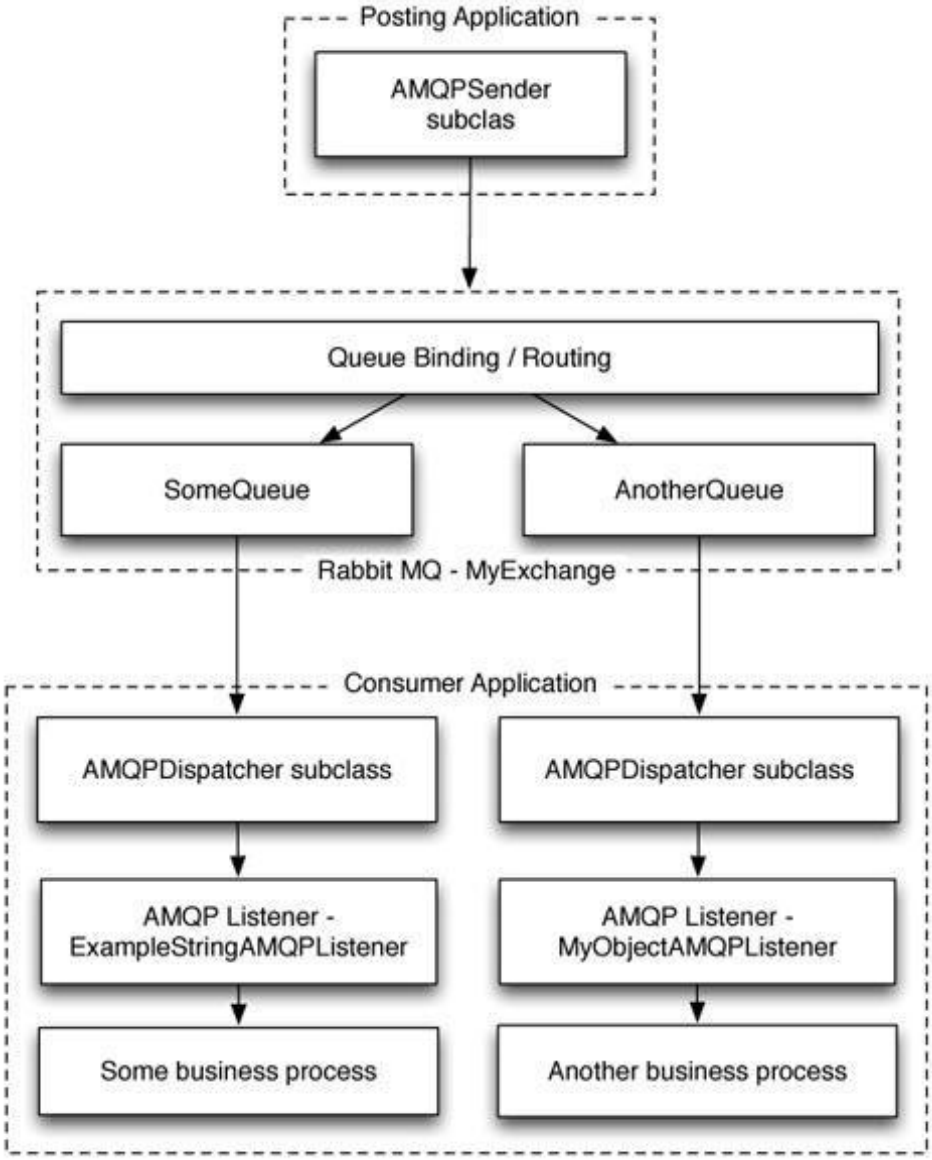
B2. Gestor de colas de proceso

Este bloque se encarga de gestionar el sistema de colas para el envío de mensajes Push. Como una llamada al API permite enviar a múltiples dispositivos de una aplicación o a varios usuarios, una sola petición se puede convertir en miles de mensajes que deben ser procesados para enviar a los servidores Push de Google y Apple en el mínimo tiempo.

Es crítico por este mismo motivo disponer de un gestor que controle una cola de procesado de mensajes a dispositivos de la forma más robusta, rápida y óptima posible.

El gestor de colas está construido sobre el protocolo estándar AMQP que modeliza el paso de mensajes entre distintos procesos a través de una cola, permite además mantener procesos suscritos a una cola de forma que empiezan a consumir mensajes en cuanto la cola tiene elementos en ella. Existen diferentes implementaciones de este protocolo en distintos lenguajes y plataformas (Apache QPid, SwiftMQ, JORAM, StormMQ,...). De forma que los bloques B1 y B3 únicamente necesitan de mecanismos para comunicarse mediante este protocolo, independientemente del software utilizado como gestor de colas.

El software utilizado es RabbitMQ. Se trata de un software que implementa el protocolo AMQP 0.9.1 utilizado en sistemas en la nube como CloudFoundry o Amazon EC2 así como en otros proyectos que requieren el uso de colas de alta demanda.



B3. Push dispatcher o procesos de envío de mensajes Push

Son los procesos que finalmente envían el mensaje a las plataformas Google y Apple. El proceso de envío adquiere mucha latencia en servidores de terceros por lo que se hace necesario un sistema que sea capaz de recoger los mensajes de la cola y generar nuevos envíos en paralelo mientras los primeros mensajes están a la espera de la confirmación de recepción del mensaje en las plataformas Google o Apple de envío de mensajes Push.

Se busca por tanto un sistema capaz de seguir funcionando aunque existan operaciones en curso sin terminar y que pueda atender las respuestas de las plataformas de mensajes Push cuando lleguen después de una latencia importante.

Para estos procesos se usa NodeJS, una plataforma que usa el motor V8 de ejecución de JavaScript creado por Google para crear rápida y fácilmente escalable. Proporciona un sistema orientado a eventos con un modelo de Entrada/Salida no bloqueante que a la vez es muy ligero y eficiente, usándose en la actualidad para aplicaciones en tiempo real que funcionan en sistemas distribuidos.

Los procesos de envío de mensajes se ejecutan como procesos independientes que consumen la cola de mensajes del gestor de cola. Además, al no ser bloqueantes, un único proceso (worker) que consume datos de la cola permite coger un cierto número de elementos y enviar a los servidores de Google y Apple sin tener que esperar la respuesta de estos, optimizando los tiempos de proceso sin tener que detener la ejecución de los siguientes envíos mientras se espera una respuesta.

Una vez se recibe una respuesta de un envío, los procesos (workers) guardan en otra cola para almacenar los registros de error en caso de tener notificación de alguno y no detener sus procesos de envío. El número de procesos (workers) que se pueden ejecutar en paralelo suele depender de la cantidad de procesadores que disponga un servidor, cada uno de los procesos además permite configurar la cantidad de envíos en paralelo que queremos realizar al servidor de Push de Google o Apple.

5. Envíos masivos y clusterización

La arquitectura de la plataforma permite el envío masivo de mensajes atendiendo a las siguientes características.

A) Envío del mismo mensaje a todos los destinatarios (caso más favorable)
6000 mensajes / segundo
Utilizando un Servidor físico, procesador de 4 núcleos Intel(R) Xeon(R) CPU E3-1245 V2 @ 3.40GHz y 8 gb de RAM, para cada uno de los 3 procesos que intervienen.

B) Envío de mensaje distinto a cada usuario, con consulta a BBDD (caso más desfavorable)
1900 mensajes / segundo
Utilizando un Servidor físico, procesador de 4 núcleos Intel(R) Xeon(R) CPU E3-1245 V2 @ 3.40GHz y 8 gb de RAM, para cada uno de los 3 procesos que intervienen.

Como vemos, mediante una solución basada en clusters podríamos escalar el envío e incrementar los ratios de envío en caso de ser necesario.