Richtlinie zum „Software-Sprint“

AndroidOpenPush – Implementierung eines Open-Source Push services für Android Apps

Schlussbericht

Zuwendungsempfänger:

Marcus Hoffmann

Das diesem Bericht zugrundeliegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 01IS17S48 gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt beim Autor.

Kurze Darstellung der Aufgabenstellung und Motivation

Was war Deine Motivation? Welches Problem wolltest Du mit Deinem Projekt lösen? Wie war die geplante Vorgehensweise zur Problemlösung (auch Angabe der wichtigsten Meilensteine)?

Ich bin Kern-Entwickler und Maintainer beim F-Droid Projekt, dessen Ziel es ist, einen vollständig freien App-Store für Android bereitzustellen. Viele Entwickler von Open-Soure Anwendungen sind gerne bereit ihre App neben dem Google Play Store auch über F-Droid zur Verfügung zu stellen. Da F-Droid nur freie Software erlaubt, sind gelegentlich Änderungen am Quelltext der App notwendig um sie in F-Droid zu integrieren. Die einzige Komponente, die bei der F-Droid Version dabei meist keinen geeigneten Ersatz findet sind Push-Benachrichtigungen. Diese sind meist ausschließlich über Googles closed-source Softwarebibliotheken (Firebase Cloud Messaging) realisierbar.

Push-Benachrichtigungen funktionieren, indem der Client (das Smartphone) eine langlebige Verbindung zu einem Server aufbaut und diese durch das gelegentliche senden von 'Keepalive' Paketen aufrecht erhält. Idealerweise sind diese Keepalives nur im Bereich von mehreren 10 Minuten notwendig um den Akku des mobilen Geräts zu schonen.

Der Push-Server ordnet einzelne aktive Verbindungen den Clients zu und schickt bei Bedarf über die offene Verbindung eine Benachrichtigung an den Client. Das Gerät wird durch die eingehende Datenpakete aus dem Sleep-Mode geweckt.

Im Rahmen des Projekts wurde eine Android Client-App zusammen mit einer Serverkomponente entwickelt.

Die Android App kümmert sich im das Aufrechterhalten der Verbindung und Registrierung des Clients. Die Serverkomponente stellt eine Schnittstelle für Webservices bereit um registrierten mobilen Endgeräten Push Nachrichten senden zu können.

Als Protokoll zwischen Client und Push Server sollten COAP und MQTT evaluiert werden.

Die Entwicklung Schnittstellen (Client-Lib und Push-Server API) sollte in Anlehnung an Google's FCM Dienste erfolgen.

Als Meilensteine dienten die Evaluiering eines Protokolls, die Festlegung der Pushserver APIs, die Implementierung des Pushservers und die Implementierung der Client-App.

Als optionaler Meilenstein wurde die Integration in eine bestehende App definiert.

Beitrag des Projektes zu den Zielen der Förderinitative „Software-Sprint“

Wer ist die Zielgruppe für Deine Lösung? Wie profitiert sie von den Ergebnissen Deines Projekts? Welche Bezüge gibt es zu den Themenfeldern und Zielen des Software Sprints?

Die Zielgruppe sind freie Softwareprojekte, die auf mobilen Plattformen Push-Benachrichtigungen benötigen. Beispiele solcher Projekte sind RocketChat, Matrix und Nextcloud.

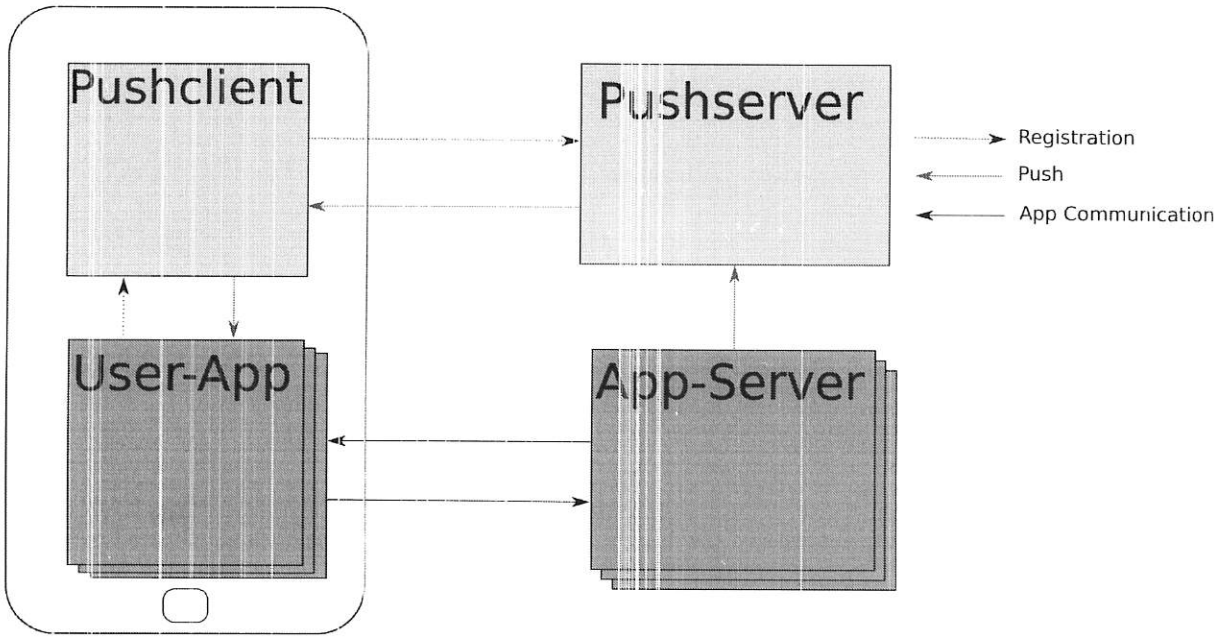
Bisher ist die einzige Möglichkeit auf einfache Weise Push-Benachrichtigungen für Mobile Apps zu senden die Nutzung von entweder Google's oder Apples Push Diensten. Diese erfodern einerseits die Nutzung von proprietären Systemdiensten und Bibliotheken und andererseits erlauben sie Google (oder Apple) das Sammeln von allen Metadaten für alle Push Nachrichten. Selbst wenn die App den Inhalt der gesendeten Nachricht verschlüsselt kann man über die Analyse der Metadaten Rückschlüsse auf das Kommunikationsverhalten von Nutzern ziehen. Außerdem macht es alle Apps abhängig von Googles Infrastruktur und Terms of Services.

Mein Projekt ist somit wohl dem Förderbereich Infrastruktur als auch Datensicherheit zuzuordnen, da es das Selbsthosten und somit das Vermeiden der (Meta-)Datenweitergabe an Dritte ermöglicht.

Ausführliche Darstellung der Ergebnisse

Welche konkreten Ergebnisse hast Du erzielt? Konnten alle Meilensteine erreicht werden? Welche zusätzlichen Erkenntnisse hast Du aus der Projektarbeit gewonnen, auch im Hinblick auf die Begleitung durch die Open Knowledge Foundation?

Ein grober Überblick über das Push-System sieht wie folgt aus:



Die oberen beiden Komponenten (Pushclient und Pushserver) sind dabei im Rahmen dieses Projektes entwickelt wurden.

Es wurde eine Protokollspezifikation für die Pushserver/Pushclient Interaktion geschrieben und auf Server und Clientseite implementiert. Verschiedene Protokolle für die Pushverbindung wurden evaluiert und sich für die erste Implementierung für Server-Sent-Events (SSE) entschieden. SSE's sind ein sehr einfaches Protokoll, und haben sich in ersten Tests als ausreichend für diese Anwendung erwiesen. Gleichzeitig ist das Protokoll des Pushkanals bewusst unabhängig von der Architektur des Gesamtsystems gehalten. Dies lässt Potential für spätere Optimierungen, ohne, dass das Auswirkungen auf Benuzter des OpenPush Projektes hat. Außerdem bietet es die Möglichkeit Pushbenachrichtigungen in Zukunft über einen schon bestehenden Kanal, wie z.B. IMAP Idle oder eine XMPP Verbindung zu senden. Dafür muss dann serverseitig (im IMAP Server oder XMPP Server) die OpenPush `/message` API implementiert werden.

Als initiale Herangehensweise an das Projekt war die Anlehnung der API und Architektur an Googles FCM geplant. Im Laufe des Projektes stellte sich heraus, dass ein kompletter Nachbau der API nicht sinnvoll ist. Somit war es Möglich ein teilweise einfacheres Design zu wählen und manche Self-Hosting betreffende Use-cases besser zu unterstützten (siehe Zielgruppe/Nutzen Abschnitt).

Die Protokollspezifikation erfolgte als OpenAPI (früher Swagger) Spezifikation. Diese lässt sich als Maschinenlesbare und ausführbare Dokumentation auffassen.

Die Implementierung der Pushserver Komponente ist im Python Flask Webframework erfolgt. Dabei wird mittels des Connexion (https://github.com/zalando/connexion) Frameworks die OpenAPI Spezifikation direkt geladen und eingehende API-Anfragen anhand dieser verifiziert.

Die Client-Implementierung der API wird mittels des openapi-generator (https://github.com/OpenAPITools/openapi-generator) Projektes aus derselben Spezifikation als Java Code generiert.

Dieser Java API-Client Code wird in das Pushclient Android Projekt eingebunden.

Dies ermöglicht das Pflegen der API Dokumentation an einer einzigen Stelle. Auf Server-Seite werden Änderungen durch Connexion sehr leicht umsetzbar, auf Client Seite können neue oder geänderte API Funktionen sofort benutzt werden.

Der Client-App Meilenstein wurde nicht komplett erreicht. Fertiggestellt wurde eine Client-App für das Android Gerät, welche sich bei einem vom User gewählten Pushserver registrieren kann. Apps, die den Pushservice benuzten wollen registrieren sich bei dieser App und erhalten dabei die Pushserver URL und einen App-Token. Um die App-Registrierung zu erleichtern sollte die Client-Library erstellt werden, korrespondierend zur FCM Client-Library. Die Registrierung muss im Moment noch manuell erfolgen, bis diese Library fertiggestellt ist. Das Zustellen von Nachrichten von der Pushclient-App and die registrierte User-App wurde ist noch nicht implementiert.

Die optional definierten Meilensteine, die die Integration der OpenPush Lösung in bestehende System vorsah wurden nicht erreicht.

Zielgruppe, Nutzen und mögliche Weiterentwicklungen

Welcher Nutzen ergibt sich für die Zielgruppe aus den Ergebnissen Deines Projekts? Welche weitergehenden Effekte ergeben sich aus der Open-Source-Stellung der Ergebnisse? Gibt es Ideen für die Weiterentwicklung Deiner Lösung und Pläne zu deren Umsetzung?

Ein Nutzen für oben genannten Softwareprojekte ist, dass es eine Lösung für eine vollständig freie Android Version ihrer Client-Apps mit Push-Benachrichtigungen geben kann. Somit können diese Versionen auch über den F-Droid App-Store verbreitet werden, welcher nur vollständig freie Software akzeptiert.

Ein weiterer großer Nutzen für Softwareprojekte, die Selfhosting als erklärtes Ziel haben, ist es keine Abhängigkeit auf externe Webservices zu haben. Im Google/Firebase Ökosystem gibt es üblicherweise eine gestaffelte Abhängigkeit zum einen zu Googles Firebase Cloud Messaging Servern, zum anderen aber auch zu einem Push-Proxy oder Gateway Server (z.b. https://github.com/matrix-org/sygnal). Dieser Gateway Server muss vom Distributor der bei Google Play verbreiteten App gehosted werden, da nur der Ersteller der App den Token hat um der App Push-Nachrichten zu senden. Dies macht z.B. die Entwickler der Riot-Android (matrix client) App verantwortlich für die Zustelluung sämtlicher Mobilen Pushbenachrichtigungen, auch wenn der Nutzer seine eigene Serverinfrastruktur hostet. Da dieser „Entwickler-Token" keine Sicherheitsrelevante Rolle spielt, sondern auf Googles Seite nur Accounting von Push-Anfragen macht, ist im OpenPush System kein so ein Token notwendig und das Hosten von Push-Proxies oder Neukompilieren von eignen App-versionen für jeden undabhängigen Hoster entfällt.

Der Plan für das weitere Vorgehen ist vorerst ist die Fertigstellung der clientseitigen Implementierung um das Projekt in der Praxis demonstrieren zu können. Als nächster Schritt steht die Integration in bestehende Projekte an, hierbei hoffe ich auf finanzielle und technische Unterstüztung von den

Entwicklern der jeweiligen Platformen zu bekommen, da eine offene, selbsthostbare Pushlösung viele Vorteile für diese Platformen mit sich bringt (s.O.).

Hat die Arbeit in dem Projekt Dich in Deiner persönlichen, fachlichen Weiterentwicklung unterstützt?

Das Förderprojekt gab den Anlass als selbstständiger Softwareenwickler zu Arbeiten, was ich nach dem Projektende weiter fortsetzen werde.

Ich habe viel dadurch gelernt, eine eigene API Schnittstelle zu designen, spezifizieren und zu implementieren. Dabei war es sehr hilfreich die 6 Monate nutzen zu können, um die Spezifikation der Schnittstelle zu verbessern, ohne dass es externe Konsumenten von dieser gab. So konnten initiale Denkfehler effizient vor einer ersten Veröffentlichung behoben werden. Die Fokussierung auf eine OpenAPI Spezifiaktion als Input für Server und Clientimplementierung hat initial etwas mehr Zeit gekostet, hat sich aber sehr bewährt um schnelle Iterationen der API vornehmen zu können, ohne den gleichen Boilerplate Code ständig anpassen zu müssen.

Kurze Darstellung der Arbeiten, die zu keiner Lösung geführt haben

Gab es Arbeiten bzw. Lösungsansätze, die nicht weiter verfolgt wurden? Was waren die Hintergründe, und wie bist Du alternativ vorgegangen?

Es wurde versucht auf die bestehenden freien Push-Notification Lösung „Gotify" aufzubauen. Dieser Ansatz hat sich aber als nicht zielführend erwiesen. Gotify adressiert Benachrichtigungen an User, wohingegen Push-Benachrichtigungen (im Sinne von FCM/APN) an einzelne Apps und per Gerät adressiert sind. Ohne komplette Änderungen der bestehenden Gotify Architektur lässt sich das nach Angaben der Autoren auch nicht ändern.
Als Alternative wurde ein eingener Pushserver in Python und ein Android Pushclient entwickelt.

Eine Integration vom Gotify-Pushservice in die Nextcloud Server-Software wurde als funktionstüchtiger Prototyp entwickelt. Diese Arbeit wurde auch nicht weiterverfolgt, die daraus gewonnenen Erkenntnisse können aber für die folgende Integration des eigenen OpenPush-Services wiederverwendet werden.

Ein Einbau der Push-Client Funktionalität in microg wurde begonnen, aber aufgrund von Zeitmangel vorerst verschoben. Es war schneller und einfacher eine alleinstehende Pushclient App zu entwickeln. Eine Integration in microg ist nach wie vor zu einem späteren Zeitpunkt geplant.

Kurze Angabe von Präsentationsmöglichkeiten für mögliche Nutzer

Wo können sich Interessenten detailliert über Deine Projektergebnisse informieren (z.B. Webseite, GHiCHub, Veröffentlichungen)?

Projektwebseite: https://bubu1.eu/openpush

Pushserver-API: https://gitlab.com/Bubu/pushserver/blob/master/openapi.yml

Serverimplementierung: https://gitlab.com/Bubu/pushserver

Clientimplementierung: https://gitlab.com/Bubu/pushclient

Kurze Erläuterung zur Einhaltung der Arbeits- und Kostenplanung

Gab es im Projektverlauf Ereignisse, die eine Anpassung der Planung erforderlich machten – z.B. Mehr- oder Minderaufwand bei der Bearbeitung von Teilaufgaben?

Hauptsächlich hat die Implementierung der Android Clientseite deutlich mehr Zeit gekostet als geplant. Dies lag zum einen an der zuerst versuchten direkten Integration in das microg Projekt, was sich als zu inflexibel für die am Anfang benötigten schnelleren Iterationszyklen herausstellte. Zum anderen wurde die Komplexität der Android Entwicklung allgemein unterschätzt was zu dem nicht vollständig erreichten Android Meilenstein geführt hat.

Kurze Darstellung von etwaigen Ergebnissen bei anderen Stellen

Gab es Entwicklungen anderer Personen oder Institutionen, die Einfluss auf Deine Arbeiten und die Zielsetzung hatten? Wenn ja, worin bestand dieser und wie bist Du damit umgegangen?

Soweit ich das mitbekommen habe gab es während des Förderzeitraums keine Weiterentwicklung oder Veröffentlichung einer vergleichbaren oder ähnlichen Lösung.