Impact Factor:					
	ISRA (India) = 3.117	SIS (USA) = 0.912	ICV (Poland) = 6.630
	ISI (Dubai, UAE) = 0.829	РИНЦ (Russia) = 0.156	PIF (India) = 1.940
	GIF (Australia) = 0.564	ESJI (KZ) = 8.716	IBI (India) = 4.260
	JIF = 1.500	SJIF (Morocco) = 5.667	OAJI (USA) = 0.350

SOI: 1.1/TAS DOI: 10.15863/TAS

International Scientific Journal

Theoretical & Applied Science

p-ISSN: 2308-4944 (print) **e-ISSN:** 2409-0085 (online)

Year: 2019 **Issue:** 08 **Volume:** 76

Published: 05.08.2019 **http://T-Science.org**

QR – Issue QR – Article

Vadim Andreevich Kozhevnikov
Peter the Great St.Petersburg Polytechnic University
Senior Lecturer
vadim.kozhevnikov@gmail.com

Evgeniya Sergeevna Pankratova
Peter the Great St. Petersburg Polytechnic University
student
jane_koks@mail.ru

Ilgar Vugar Ogly Shahbazov
Peter the Great St. Petersburg Polytechnic University
student
ilgar.shahbazov@gmail.com

PUSH NOTIFICATION SERVICE

Abstract: *This article is about the development of web services. The article presents the main approaches to the development. Next, we are talking about different software for the project development. Besides, there are a few words about design and detailed description of the project. In addition, the article describes the manual testing of the project.*

Key words: *push, gRPC, Scala, notifications, Android, Amazon, Windows Phone, iOS.*

Language: *Russian*

Citation: Kozhevnikov, V. A., Pankratova, E. S., & Shahbazov, I. V. (2019). Push notification service. *ISJ Theoretical & Applied Science, 08 (76)*, 17-31.

Soi: http://s-o-i.org/1.1/TAS-08-76-4 ***Doi***: crossref https://dx.doi.org/10.15863/TAS.2019.08.76.4

Classifiers: *Computer science, computer engineering and automation.*

СЕРВИС PUSH-УВЕДОМЛЕНИЙ

Аннотация: *Данная статья относится к сфере разработки веб сервисов. В статье приводятся основные подходы разработки ПО и проводится анализ современных способов общения с клиентами. Далее речь идет о выборе программных средств для реализации проекта. Также рассматривается проектирование и детальное описание реализации проекта. Кроме этого, в статье рассказывается о ручном тестирование проекта.*

Ключевые слова: *push, gRPC, Scala, push-уведомления, Android, Amazon, Windows Phone, iOS.*

Введение

Основным интересом многих компаний является оценка лучшей коммуникационной технологии для расширения взаимодействия с клиентами. Как компаниям убедить клиентов пользоваться вашим продуктом в наиболее контекстуальном ключе? Самые популярные способы общения на данный момент – это электронная почта, push-уведомления и SMS - и каждая из этих технологий имеет плюсы и минусы в конкретном контексте.

Сузим деятельность компании – представим, что есть продукт, являющийся веб-сервисом, и для веб-сервиса написаны клиенты для различных мобильных платформ, а также есть веб-приложение. При уведомлениях по электронной почте возникает сразу несколько проблем: обязательная регистрация на сервисе, долгий отклик и пользователь (или его почтовый сервер)

может подумать, что это спам. Использованием SMS решает все проблемы, обозначенные выше, но нужно учитывать расходы при отправке большого количества сообщений.

Технология Push является самой современной и решает все вышеперечисленные проблемы, а также имеет ряд полезных функций, таких как быстрый переход в приложение, или использование картинок в уведомлениях. - Очевидно, что веб-приложений с клиентами на разных платформах не мало, а задача отправки push-уведомлений является общей.

На данный момент есть продукты, реализующие мульти-платформенную отправку push-уведомлений, но все они основаны на REST и имеют закрытый исходный код. У технологии REST есть ряд ограничений, и часть компаний используют только RPC [1] подход.

Исходя из этого, разработка одного общего веб-сервиса с открытым исходным кодом, основанного на RPC, который будет отправлять уведомления на указанные клиентом сервиса платформы, является очень актуальной.

Постановка задачи

Нашей целью является разработка веб-сервиса для push-уведомлений.

В соответствии с поставленной целью необходимо решить следующие задачи:

- исследование взаимодействия языка Scala с фреймворком gRPC.;
- изучение работы с серверами пуш уведомлений для iOS, Android, Windows 10 Mobile и Fire OS;
- проектирование и разработка сервиса пуш уведомлений;
- тестирование веб-сервиса, путём создания приложений на iOS, Android, Windows Phone и Fire OS.

Практическая ценность данного сервиса заключается в том, что после отгрузки в публичный репозиторий, им смогут пользоваться все команды разработчиков, у которых есть клиенты на смартфоны и веб, и т.к. сервис будет с открытым исходным кодом, его сможет изменить под свои нужды любой желающий.

Подходы к разработке ПО

Монолитная архитектура - «состоящая из одной части» - является унифицированной моделью для разработки программного обеспечения. Все компоненты в монолитной архитектуре взаимосвязаны и взаимозависимы, и для компиляции приложения на монолитной архитектуре все компоненты должны присутствовать. Кроме того, если какой-либо компонент должен быть обновлен, всё приложение должно быть переписано.

Многие стартапы начинаются в монолитной архитектуре: структура такого приложения хорошо поддаётся для небольших групп разработки и имеет малые операционные накладные расходы. Монолиты также имеют только одну большую базу кода – логика на стороне сервера, логика на стороне клиента, фоновые процессы – всё определено в одной и той же базе кода. Это значит, что, если разработчики хотят внести какие-либо изменения, даже небольшие, им придется собрать и развернуть весь проект сразу.

Альтернативой монолитной архитектуре является микросервисная (рис. 1).

Рис.1 Архитектуры ПО

Идея заключается в том, чтобы разделить целое приложение на небольшие сервисы, где каждый сервис выполняет какую-либо роль. Каждый сервис запускается отдельным процессом и взаимодействует с другими сервисами через HTTP-протокол. Говоря простым языком, мы переходим от большого монолита, который хранит внутри множество компонент, и разрабатывается, развертывается, выпускается как единое целое, к архитектуре, где каждый модуль

— это компонента, у которой своё адресное пространство, свой отдельный процесс, удаленный интерфейс. То есть, каждый микросервис – это независимый модуль с независимым жизненным циклом.

У такого решения есть ряд преимуществ:

- Лучшая организация: каждый микросервис имеет одну определенную цель и никак не связан с другими микросервисами.
- Независимость: независимые компоненты легко перенастроить, пересобрать, изменить всю архитектуру микросервиса, добавлять новые возможности, т.к. для внешнего мира важно только публичное API данного сервиса, а всё, что внутри - не интересно.
- Производительность: при правильных обстоятельствах, микросервисы могут быть очень производительными. Например, самые нагруженные микросервисы можно изолировать и масштабировать.

Микросервисная архитектура, безусловно, не является идеальной, и у неё есть свои недостатки – затраты на взаимодействие между микросервисами. Представим наиболее частое решение при построении такой архитектуры: REST API, который общается с помощью формата JSON через HTTP. В таком случае, для каждого запроса следует выполнить следующие действия:

- сериализовать данные в JSON и создать HTTP-запрос;
- отправить запрос;
- десериализовать данные;
- выполнить работу;
- сериализовать ответ в JSON и создать HTTP-ответ;
- отправить ответ;
- десериализовать полученный ответ.

Ведущий архитектор из компании Credera Джереми Лейзи провёл собственный эксперимент [2], выявляющий затраты на дополнительные действия, описанные выше. Эксперимент показал следующее: по мере увеличения количества передаваемых данных увеличивается время ввода/вывода и сериализации. Например, передаваемые данные типа double в Java весят 8 байт. В микросервисной архитектуре это число придется упаковать в JSON и переменная будет весить уже не 8 байт, а 19. Это означает, что для сериализации полной точности double из памяти в JSON вы значительно увеличиваете объем данных, необходимых для его представления (Рис. 2).

В связи с этим, компания Google придумала свой протокол Protobuf, который, по словам компании, является лучшей альтернативой протоколу JSON и микросервисную архитектуру лучше строить с помощью их технологии. Исследования коллеги Джереми Лейзи – Майка Энсора показали, что взаимодействие через Protobuf быстрее JSON почти в 2.4 раза [3].

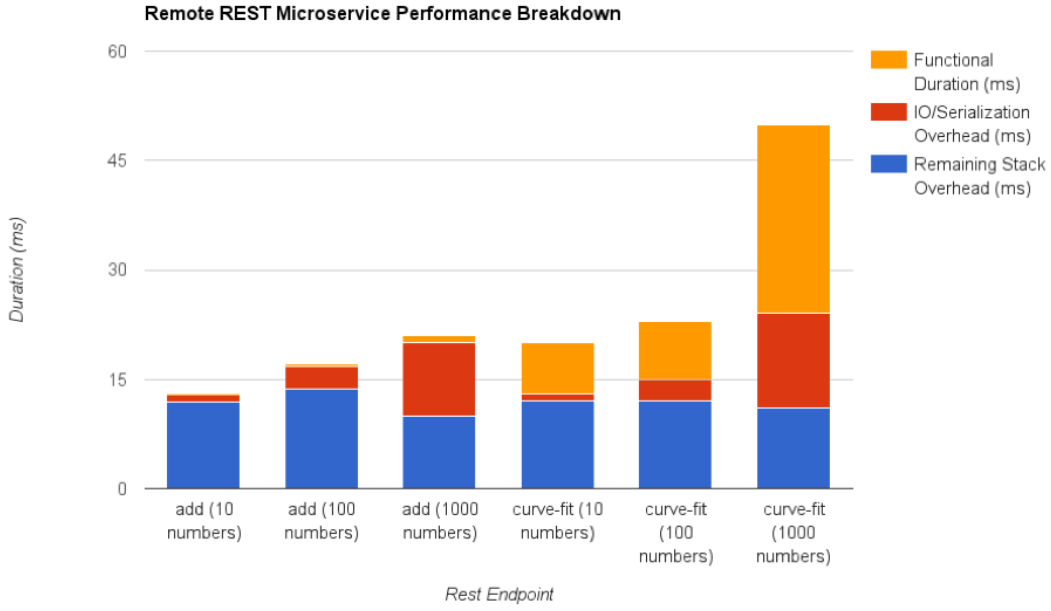

Рис 2. Увеличение затрат на дополнительные действия в микросервисе

Современные технологии общения с клиентами

Основной проблемой для многих компаний является оценка лучшего канала связи для увеличения взаимодействия с клиентами. Электронная почта, push-уведомления и SMS-сообщения – это самые популярные способы общения с клиентами. Чтобы клиенты возвращались к приложению, уведомления должны обеспечивать наибольшую актуальность в момент получения.

Итак, какие преимущества каждой среды? Исходя из некоторых исследований, примерно 90% SMS-сообщений читаются в течение первых 15 минут после доставки. Этот канал обмена должен использоваться для предоставления критически важной для бизнеса информации. Отправка SMS нужна, например, в случаях, когда курьер приехал, или купленный товар привезен в указанный магазин и готов для выдачи.

Сочетание высокой скорости взаимодействия и глобального охвата также делает технологию SMS привлекательным выбором для аутентификации пользователей. Такие сценарии, как проверка номера, используются в основном для SMS, потому что имеют высокую доступность. Google, Facebook, Apple и др. при двухфакторной авторизации отправляют автоматическое текстовое сообщение, содержащее PIN-код, который клиент может ввести через другой канал, чтобы подтвердить свою личность. SMS не стоит использовать для информации, которую пользователь должен будет указать позже, например, квитанции и т. д.

По сравнению с электронными письмами, SMS легко удалить и сложнее найти. Любая информация, отправляемая по SMS, должна быть легко доступна в короткие сроки. Также, максимальный объем SMS – 160 символов, и каждая отправка SMS стоит денег.

Электронная почта должна использоваться при передаче информации, которая не зависит от времени. Сообщение об успешной регистрации, подтверждения, заказы, квитанции – всё это случаи, при которых рекомендуется использовать электронную почту. Электронная почта не должна использоваться для срочных действий. Согласно некоторым исследованиям среднее время для просмотра сообщения электронной почты составляет 6.4 часа [4]. Исходя из этого, было бы нецелесообразно уведомлять людей по электронной почте о том, что их доставка еды или такси прибыли.

Наконец, push-уведомления – технология, являющаяся стандартным способом связи для приложений на смартфонах. Push является менее навязчивым, чем SMS, т. к. он может быть доставлен на экран пользователя без активности пользователя. Также, Push является менее затратным и имеет больше возможностей (например, фото в уведомлении).

Выбор программных средств

Scala – это типобезопасный JVM-язык, который включает в себя как объектно-ориентированное, так и функциональное программирование [5]. Это чрезвычайно сжатый, логичный и чрезвычайно мощный язык. Есть мнение, что Scala – сложный язык, и это правда. Некоторые из сложных функций языка (Tuples, Functions, Macros и т.д.) в конечном итоге облегчают разработчику писать лучший код и повышать производительность.

Scala – это не единственная попытка создать «улучшенную версию Java». Альтернативы, такие как Kotlin [6] и Ceylon [7], также пошли по этому пути. Главным отличием Scala от Kotlin и Ceylon в том, что последние оставили Java-подобный синтаксис, чтобы минимизировать сложность перехода с Java.

Это может показаться отличной идеей, но в итоге такой синтаксис в дальнейшем ограничивает и заставляет оставаться в рамках тех же самых парадигм Java, которые были причиной создания «лучшую Java» в первую очередь. В отличии от этого Scala была создана специально для того, чтобы стать лучшим языком, пропуская те самые ограничивающие аспекты. В результате между Scala и Java действительно существуют различия в коде, которые могут сделать изучение Scala немного сложнее.

Simple Build Tool (далее, sbt) – это инструмент для сборки, написанный на Scala. Проекты Scala обычно собирают с помощью sbt.

Инструмент имеет следующие преимущества:

- удобные средства для управления зависимостями;
- создание задач на языке Scala;
- инструменты для тестирования;
- запуск REPL в контексте проекта.

Remote Procedure Call (далее, RPC) – удаленный вызов процедур – это протокол, с помощью которого одна программа может использовать функции другой программы, расположенной на другом компьютере в сети. RPC использует клиент-серверную модель. Программа, которая хочет вызвать функцию другой программы – клиент, а программа, предоставляющая эту функцию – сервер.

gRPC Remote Procedure Calls – это среда с открытым исходным кодом, которая может запускаться, где угодно (клиент и сервер), она позволяет клиенту и серверу взаимодействовать и упрощать между собой связь [8].

Основные свойства gRPC:

- gRPC является двоичным, а не текстовым, как RPC с JSON или SOAP, поэтому является более компактным и эффективным.
- Он мультиплексирует запросы по одному TCP-соединению. Это позволяет одновременно отправлять множество сообщений и сокращает тем самым использование сетевых ресурсов.
- Используется сжатие заголовка для уменьшения размера запросов и ответов.

Данную технологию используют такие организации, как Google, Netflix, Docker, Cisco и т.д. Поддерживаются практически все основные языки – Java, C++, Objective-C, Python, Ruby, Go, C#, Node JS.

gRPC использует протокольные буферы Protobuf – метод, разработанный Google для сериализации структурированных данных в двоичный формат.

Система удаленных уведомлений для устройств Android

Firebase Cloud Messaging (FCM) [9] - это система удаленных уведомлений компании Google, созданная для разработчиков приложений, с помощью которого можно распространять информацию на устройства Android и Web.

Серверная часть FCM состоит из двух компонент:

- FCM-серверы, предоставляемые Google;
- сервер приложения.

Сервер приложений должен отвечать следующим критериям:

- возможность отправки запросов клиенту;
- возможность отправки запросов на сервера FCM;
- возможность обрабатывать запросы;
- возможность безопасного хранения ключей сервера и клиентских токенов.

Системы удаленных уведомлений для устройств Apple

Apple Push Notification service (далее, APNs) – это система удаленных уведомлений компании Apple, созданная для разработчиков приложений, с помощью которой можно распространять информацию на устройства iOS, macOS и tvOS (рис. 3) [10].

При первоначальном запуске приложения iOS на устройстве пользователя система автоматически устанавливает зашифрованное и постоянное IP-соединение между вашим приложением и APNs. Это соединение позволяет приложению выполнять настройку для получения уведомлений.

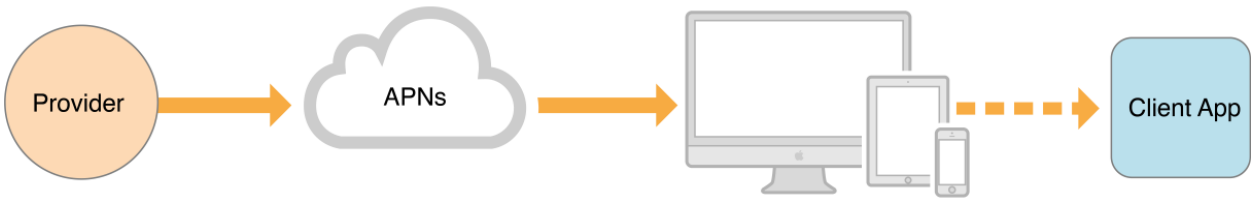

Рис. 3. Архитектура APNs

Provider – это сервер, который нужно развернуть разработчику и настроить для работы с APNs.

Первым шагом для отправки удаленного уведомления является установление связи с APNs серверами:

Development server: api.development.push.apple.com:443

Production server: api.push.apple.com:443

Также, сервер Provider должен поддерживать TLS 1.2 и выше при подключении к APNs. Есть несколько способов подключения к серверам APNs:

- Использовать SSL сертификат клиента Provider.
- Создать токен аутентификации Provider'а, подписанный с ключом, предоставленным через учетную запись разработчика. В таком случае необходимо обновлять токен – каждый токен аутентификации работает в течение одного часа.

Каждое уведомление, отправленное Provider-сервером в APNs включает payload. Payload – это JSON-словарь, который отправляется вместе с HTTP/2 запросом. Максимальный размер payload – 4 КБ.

Система удаленных уведомлений для устройств Windows

Windows Push Notification Services (WNS) [11] позволяют сторонним разработчикам отправлять пуш уведомления на устройства Windows, в том числе на Windows Phone 8.1 и Windows 10 Mobile.

Для отправки уведомлений используется платформа Universal Windows Platform (UWP) (рис. 4).

Рис. 4. Платформа UWP

Universal Windows Platform (UWP) – платформа, созданная компанией Microsoft и впервые представленная в Windows 10. Данная платформа предназначена для создания универсальных приложений, запускаемых как на Windows 10, так и на Windows 10 Mobile без изменения в коде. Имеется поддержка создания таких приложений на C++, C#, VB.NET и XAML. API реализован в C++ и поддерживается в C++, VB.NET, C#, F# и JavaScript.

Принцип работы:

- приложение запрашивает канал push-уведомлений от UWP;
- Windows просит WNS создать канал уведомлений. Этот канал возвращается вызывающему устройству в виде URI;
- URI канала уведомлений возвращается в приложение;
- приложение отправляет URI в сервер, откуда происходит отправка уведомлений;
- сервер сохраняет URI для дальнейших отправок уведомлений;
- когда сервер готов отправить, он уведомляет WNS, используя URI;
- взаимодействие происходит по HTTP POST через SSL;
- WNS получает запрос и направляет уведомление на соответствующее устройства.

Система удаленных уведомлений для устройств Amazon

Amazon Device Messaging (ADM) позволяет отправлять уведомления на устройства Amazon [12].

При отправке сообщения через ADM используются четыре компонента, два из которых контролирует разработчик, отправляющий сообщение.

Опишем функции каждой из компонент.

Сервера разработчика:

- аутентификация на ADM серверах с помощью токена доступа;
- отправка сообщений на сервера ADM для дальнейшей доставки в приложения на Amazon устройствах.

Сервера ADM:

- используют токен доступа для валидации серверов разработчика;
- доставляют сообщения с серверов разработчика до ADM клиентов на устройства.

Клиенты ADM:

- обрабатывают регистрацию приложения с ADM серверами;
- получают сообщения из ADM серверов и передают их приложению.

Приложение:

- регистрируется с клиентом ADM для получения сообщений из серверов разработчика;
- получает сообщение от клиента ADM и обрабатывает его.

Разработчик контролирует компоненты Приложение и Сервер разработчика.

На высоком уровне процесс доставки сообщения, отправляемого ADM, выглядит следующим образом (рис. 5):

- сервер разработчика отправляет на серверы ADM сообщение, содержащее данные в формате JSON;
- серверы ADM отправляют сообщение клиенту ADM на устройства, на котором установлено приложение;
- клиент ADM получает сообщение и передает в приложение, которое приложение обрабатывает и выводит.

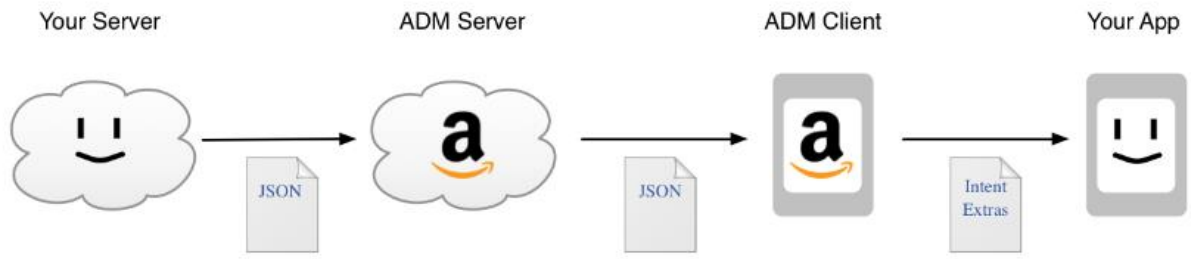

Рис. 5. Принцип взаимодействия компонент Amazon

Функциональная схема сервиса

Проектирование веб сервисов является одним из важных шагов перед разработкой. Уделяя время грамотному проектированию архитектуры, можно сэкономить в будущем много времени и сил при дальнейших улучшениях сервиса.

Есть сервис, предоставляющий пользователю различные возможности. Пользователь может иметь несколько каналов взаимодействия с сервисом – по HTTP протоколу, по RPC протоколу и т. п. В данной статье будет рассматриваться взаимодействие по RPC протоколу, но должна быть возможность гибкого подключения других протоколов для клиент-серверного взаимодействия (рис. 6).

Рис. 6. Взаимодействие сервиса с внешними системами

Следовательно, надо декомпозировать функциональность сервиса на маленькие компоненты.

Требования к функциональности:

- возможность отправки уведомлений на Android;
- возможность отправки уведомлений на iOS;
- возможность отправки уведомлений на Windows Phone;
- возможность отправки уведомлений на Kindle Fire (>2nd generation).

Разделим функциональность сервиса по требованиям к функциональности и получим 4 компоненты по взаимодействию с различными системами удаленных уведомлений (рис. 7).

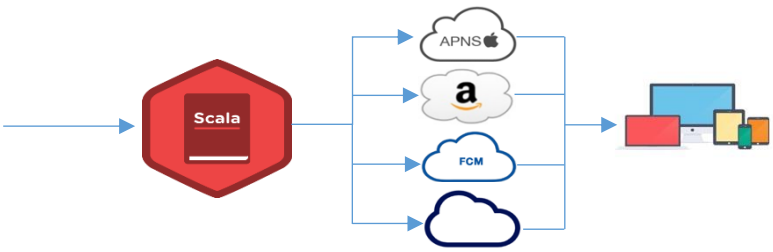

Рис. 7. Взаимодействие сервиса с системами удаленных уведомлений

- Компонент взаимодействия с Android устройствами:

Для отправки сообщений на платформу Android, необходимо иметь API-ключ, который хранится на серверах Firebase, и идентификатор зарегистрированного устройства (deviceRegistrationId). На рис. 8 представлен программный код, предоставляющий возможность отправки push-уведомлений на устройства Android.

```
val fcmApiKey: String = ???
val deviceRegistrationId: FCMToken = FCMToken("registration_id ")
val client = new FCMClient(fcmApiKey)
val message = FCMMessage(Map("key" -> "value"))
val response: Future[MappedFCMResponse] = client.push(deviceRegistrationId, message)
```

Рис. 8. Код для отправки уведомления на Android

- Компонент взаимодействия с iOS устройствами:

Как было выяснено ранее, для отправки уведомлений на устройства iOS, необходимо получить сертификат для своего приложения, либо иметь актуальный токен аутентификации приложения. Наш компонент для отправки уведомлений на платформу iOS поддерживает два вида аутентификации.

При использовании сертификата приложения код показан на рис .9.

```
val certificateKeyStore: KeyStore = ???
val certificatePassword: String = ???
val topic = APNSTopic("ru.company.App")
val deviceToken: APNSToken = APNSToken.build(“hex_device_token”).get
val message = APNSMessage.simple("Scala push!")
val request = APNSRequest.withTopic(topic, message)
val client = APNSHttpClient(certificateKeyStore, certificatePassword, isSandbox = true)
val result: Future[Either[ErrorReason, APNSIdentifier]] = client.push(deviceToken,
request)
```

Рис. 9. Код для отправки уведомления на iOS с использованием сертификата на языке Scala

На рис. 10 показана отправка уведомления при использовании токенов.

```
val conf = APNSTokenConf(
  Paths.get("priv-key.p8"),
  KeyId("key_id "),
  TeamId("team_id ")
)
val client = APNSTokenClient(conf, isSandbox = true)
val topic = APNSTopic("ru.company.App ")
val deviceToken: APNSToken = APNSToken.build("hex_device_token").get
val message = APNSMessage.simple("Scala push!")
val request = APNSRequest.withTopic(topic, message)
val result: Future[Either[APNSError, APNSIdentifier]] = client.push(deviceToken, request)
```

Рис. 10. Код для отправки уведомления на iOS с использованием токенов на языке Scala

- Компонент взаимодействия с Windows Phone устройствами:

Для отправки уведомления на WNS, необходимо иметь следующие параметры:

- packageSid и clientSecret – выдается при регистрации Windows-приложения в UWP;
- uri – канал для взаимодействия с WNS.

Имея эти параметры, можно отправить push-уведомление, как показано на рис. 11.

```
val packageSid: String = ???
val clientSecret: String = ???
val uri: String = ???
val credentials = WNSCredentials(packageSid, clientSecret)
val client = new WNSClient(credentials)
val payload = ToastElement.text("Scala push!")
val message = WNSMessage(payload)
val token = WNSToken.build(uri).get
val response: Future[WNSResponse] = client.push(token, message)
```

Рис. 11. Код для отправки уведомления на Windows Phone

- Компонент взаимодействия с Amazon Device Messaging:

Для отправки уведомления на устройства Amazon, необходимы параметры:

- clientId и clientSecret – получаем при добавлении приложения на Amazon App Distribution Portal;
- token – токен доступа, который необходимо получить из ADM серверов.

Программный код для отправки уведомления на устройство Amazon иллюстрирован на рис. 12.

```
val clientId: String = ???
val clientSecret: String = ???
val token: String = ???
val deviceID: ADMToken = ADMToken(token)
val client = new ADMClient(clientId, clientSecret)
val message = AndroidMessage(Map("key" -> "value"), expiresAfter = 20.seconds)
val response: Future[HttpResponse] = client.push(deviceID, message)
```

Рис. 12. Код для отправки уведомления на устройство Amazon

Реализация gRPC интерфейса

Для того, чтобы микросервисы в gRPC могли взаимодействовать, необходимо описание RPC взаимодействия на языке Protobuf.

Описание сообщения для запроса на ADM сервер приведено на рис. 13.

```
message AdmRequest {
    string clientId = 1;
    string clientSecret = 2;
    string admToken = 3;
    map<string, string> message = 4;
}
```

Рис. 13. Описание сообщения на ADM

Параметры сообщения:

- clientId – идентификатор пользователя на ADM, находится на https://developer.amazon.com/home.html;
- clientSecret – ключ пользователя на ADM, находится на https://developer.amazon.com/home.html;
- admToken – токен приложения, берется из приложения;

- message – данные сообщения.

Следующим в коде сообщением является запрос к APNs (см. рис. 14).

```
message ApnsRequest {
    string privateKeyPath = 1;
    string keyId = 2;
    string teamId = 3;
    string certKeyStore = 4;
    string certPass = 5;
    string topic = 6;
    string deviceToken = 7;
    string message = 8;

}
```

Рис.14. Описание сообщения на APNs

Параметры сообщения:

- privateKeyPath – путь до приватного ключа;
- keyId – идентификатор приватного ключа;
- teamId – идентификатор аккаунта разработчика Apple;
- certKeyStore – SSL сертификат, который генерируется на developer.apple.com;
- certPass – пароль SSL сертификата;
- topic – заголовок;
- deviceToken – токен устройства, берется из приложения;
- message – данные сообщения.

Рассмотрим сообщение к серверу FCM (см. рис. 15).

```
message FcmRequest {
    string fcmApiKey = 1;
    string registrationId = 2;
    map<string, string> message = 3;
    map<string, string> data = 4;
}
```

Рис. 15. Описание сообщения на FCM

Параметры сообщения:

- fcmApiKey – ключ API для веб-приложения;
- registrationId – токен;
- message – данные сообщения.

Рассмотрим сообщение на WNS сервера (см. рис. 16).

```
message WnsRequest {
    string packageSid = 1;
    string clientSecret = 2;
    string text = 3;
    string uri = 4;
}
```

Рис. 16. Описание сообщения на WNS

Параметры сообщения:

- packageSid – идентификатор пакета безопасности;
- clientSecret – ключ;
- text – текст уведомления;
- uri – канал для взаимодействия с WNS.

Ответом на все запросы к нашему сервису будет являться сообщение, которое содержит в

себе одно единственное поле success, которое обозначает успешность выполнения операции.

Для использования сообщений, описанных выше, необходимо объявить сервис на языке Protobuf и описать методы. В данном случае, сервисом является ScalaPusher, внутри которого есть четыре метода (рис. 17).

```
service ScalaPusher {
    rpc AdmSend (AdmRequest) returns (NotificationResponse) {}
    rpc ApnsSend (ApnsRequest) returns (NotificationResponse) {}
    rpc FcmSend (FcmRequest) returns (NotificationResponse) {}
    rpc WnsSend (WnsRequest) returns (NotificationResponse) {}
}
```

Рис. 17. Описание методов для удаленного вызова

Сборка сервиса

Есть несколько способов развертки сервиса для различных платформ:

- сборка исходников в универсальные архивы форматов zip, tar.gz, xz;
- deb и rpm пакеты для систем на базе Debian/RHEL;
- dmg для OSX;
- msi для Windows;
- docker-образы.

Для Simple Built Tool существует плагин, который поддерживает выше перечисленные сборки - SBT Native Packager (рис. 18).

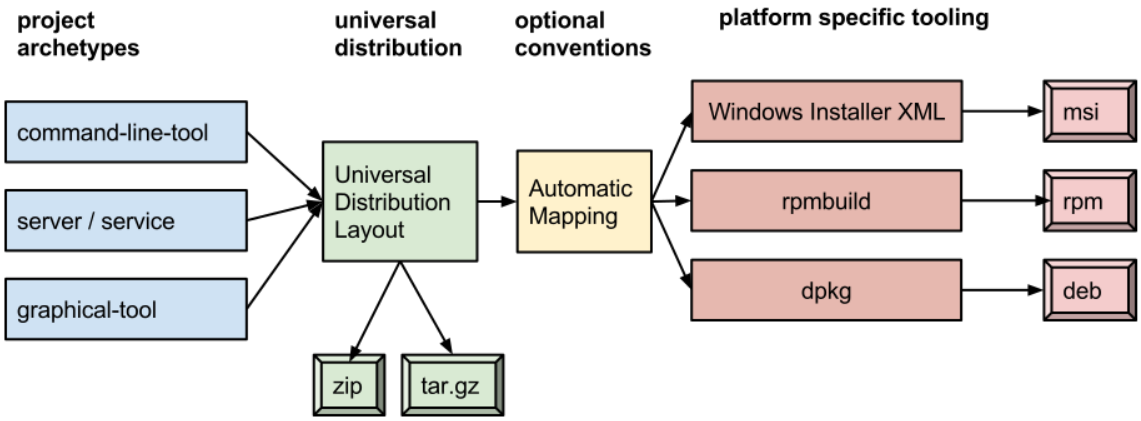

Рис. 18. Архитектура SBT Native Packager

Для установки плагина в SBT, достаточно просто установить плагин. И потом активировать плагин, который предоставит скрипты запуска для Linux, OSX и Windows.

Для создания пакета имеются несколько команд:

- universal:packageBin – генерирует универсальный zip файл;
- universal:packageZipTarball - генерирует универсальный tgz файл;
- debian:packageBin – генерирует deb пакет;
- docker:publishLocal – строит докер образ с использование локального Docker сервера;
- rpm:packageBin – генерирует rpm пакет;
- universal:packageOsxDmg – генерирует dmg файл с тем же контентом внутри, что и в zip/tgz архивах;
- windows:packageBin – генерирует msi.

Требования к сервису для тестирования

Цели тестирования:

- Оценка работы базовых функций сервиса:
  - отправка push уведомлений на устройства Android;
  - отправка push уведомлений на устройства iOS;
  - отправка push уведомлений на устройства Amazon;

- отправка push уведомлений на устройства Windows.

- Оценка корректной работы системы gRPC в рамках сервиса.

Мы выбрали ручной способ тестирования.

Предусловия для тестирования:

- сервис – наш севрис push-уведомлений;
- клиент для gRPC – отправляет запросы в сервис push-уведомлений;
- мобильные приложения на Android, iOS, Fire OS и Windows Phone;
- устройство или эмулятор;
- доступ в интернет с сервиса – для того, чтобы сервис мог послать запрос к серверам push-уведомлений той или иной платформы;
- доступ устройства/эмулятора в интернет – для получения push-уведомления.

Клиент для gRPC

Система для удаленного вызова процедур gRPC поддерживает ряд языков программирования: C/C++, C#, Dart, Go, Java, Node.js, PHP, Python, Ruby.

Данный список содержит в себе только те языки программирования, которые официально поддерживаются. Также, есть ряд других языков, которые поддерживаются сообществом того или иного языка, как, например, в случае со Scala. Для наглядности работы gRPC мы возьмем отличный от Scala язык программирования. Для тестирования подойдет Python, т. к. имеет упрощенный синтаксис и простую настройку окружения.

Для начала следует скомпилировать описание сервиса на Protobuf в модули на языке Python. Выполним следующие шаги для достижения данной цели:

- поставим Python на операционную систему;
- выполним команды в терминале:
  pip install grpcio;
  pip install grpcio-tools;
- выполним команду, которая скомпилирует proto-файл в модуль на языке Python:

python –m grpc_tools.protoc –I. –python_out=. –grpc_python_out=. Scalapusher.proto.

В папке, из которой выполнялись команды, появились модули scalapusher_pb2_grpc.py и scalapusher_pb2.py, которые будут использованы для дальнейшего тестирования сервиса.

Тестирование для Android

В качестве мобильного приложения возьмем приложение с официального репозитория firebase – Firebase Cloud Messaging Quickstart. Главная функция данного приложения выводить токен устройства, который нужен будет для отправки уведомления. Также, для отправки уведомления нужен ключ самого приложения на firebase.google.com.

В среде разработки Android Stuido имеется инструмент для эмуляции устройств – Android Virtual Device. Запустим эмулятор на последней версии Android Oreo с приложением и выведем токен на экран.

Запустим сервис и запустим следующий клиентской код на языке Python.

Устройство получает push-уведомление и выводит сообщение (рис. 19). Из переменной response получаем поле success – значение success равно True.

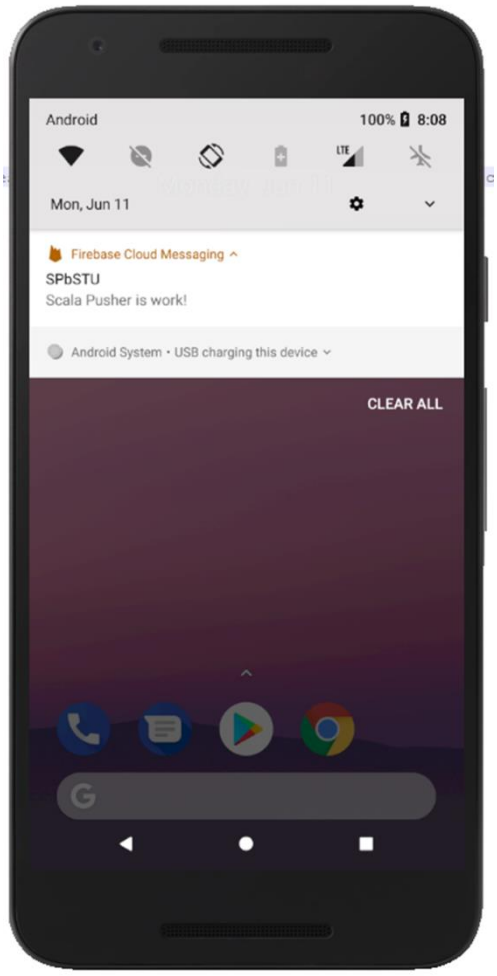

Рис. 19. Успешное получение push-уведомления на Android

Тестирование для iOS

Для тестирования устройств на базе iOS необходим аккаунт разработчика Apple, который стоит $99, поэтому было проведено тестирование без мобильного приложения.

Отправим с помощью языка Python запрос без токенов для аутентификации и получаем код ответа 401, который означает, что запрос неавторизованный.

Тестирование для Amazon

Для устройств Amazon на базе Android и Fire OS отсутствует поддержка тестирования push-уведомлений без физического устройства. Отправим push-уведомление без полей для аутентификации – возвращается ответ с кодом 401 – неавторизованный запрос.

Тестирование для Windows

В качестве мобильного приложения возьмем приложение с официального сайта code.msdn.microsoft.com – Push and periodic notifications client-side sample [13]. Для запуска необходима среда разработки Visual Studio, которая поддерживает возможность эмуляции мобильных операционных систем и запуска на них приложений.

Запустим наше приложение и отправим с помощью скрипта на Python запрос – уведомление успешно получено и выведено устройством (рис. 20).

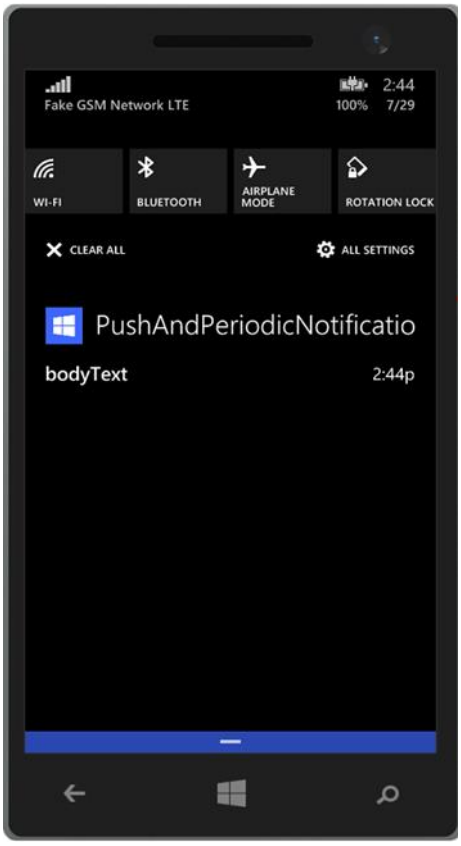

Рис. 20. Успешное получение уведомления на Windows Phone

Заключение

В ходе работы были проведены исследования и сравнительный анализ подходов разработки программного обеспечения и были рассмотрены способы взаимодействия микросервисов в сети. Мы проанализировали технологии, использовавшиеся при разработке сервиса, и рассмотрели взаимодействие gRPC и Scala. Для отправки уведомлений использовались официальные документации по API серверов APNs, FCM, ADM и WNS для платформ iOS, Android, Amazon и Windows Phone соответственно.

Была спроектирована архитектура сервиса и разработаны компоненты для работы с серверами FCM, APNs, ADM и WNS. Реализована поддержка gRPC для отправки уведомлений. Добавлена гибкая сборка проекта с помощью плагинов сборщика SBT.

Было проведено функциональное тестирование сервиса с использованием эмуляторов мобильных устройств и мобильных приложений, на которые приходят уведомления.

References:

1. (n.d.). *RPC: Remote Procedure Call Protocol Spetification* Version 2. Retrieved July 30, 2019, from: https://tools.ietf.org/html/rfc5531
2. (n.d.). *A Common Pitfall With Microservice Architectures.* Retrieved July 30, 2019, from https://www.credera.com/blog/technology-insights/open-source-technology-insights/a-common-pitfall-with-microservice-architectures/
3. (n.d.). *Protobuf Alternative to REST for Microservices.* Retrieved July 30, 2019, from https://dzone.com/articles/protobuf-alternative-to-rest-for-microservices

4. (n.d.). *Email and SMS marketing.* Retrieved July 30, 2019, from: http://www.zipstripe.com/text-vs-email-sms-marketing/
5. (n.d.). *The Scala Programming Language*. Retrieved July 30, 2019, from https://www.scala-lang.org
6. (n.d.). *Kotlin Programming Language*. Retrieved July 30, 2019, from https://kotlinlang.org
7. (n.d.). *Eclipse Ceylon: Welcome to Ceylon.* Retrieved July 30, 2019, from: https://ceylon-lang.org
8. (n.d.). *gRPC*. Retrieved July 30, 2019, from https://grpc.io/
9. (n.d.). *Firebase*. Retrieved July 30, 2019, from https://firebase.google.com/docs/cloud-messaging/
10. (n.d.). *Local and Remote Notification Programming Guide*. Retrieved July 30, 2019, from: https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/APNSOverview.html
11. (n.d.). *Windows Push Notification Services (WNS) overview*. Retrieved July 30, 2019, from https://docs.microsoft.com/en-us/windows/uwp/design/shell/tiles-and-notifications/windows-push-notification-services--wns—overview
12. (n.d.). *Set up Amazon Device Messaging*. Retrieved July 30, 2019 from https://developer.amazon.com/docs/adm/set-up.html
13. (n.d.). *Push and periodic notifications client-side sample.* Retrieved 30.07.2019 from: https://code.msdn.microsoft.com/windowsapps/push-and-periodic-de225603