Red Hat Managed Integration 2

Using the Push Notifications Service

For Red Hat Managed Integration 2

Last Updated: 2020-04-23

Red Hat Managed Integration 2 Using the Push Notifications Service

For Red Hat Managed Integration 2

Legal Notice

Abstract

This document provides instructions for using the Push Notifications service, Red Hat Managed Integration 2.

Table of Contents

PREFACE 3

CHAPTER 1. PUSH NOTIFICATIONS TERMINOLOGY 4

CHAPTER 2. SETTING UP THE PUSH NOTIFICATIONS MOBILE SERVICE 5
2.1. OBTAINING FIREBASE CLOUD MESSAGING CREDENTIALS 5
2.2. OBTAINING APPLE PUSH NOTIFICATION SERVICE CREDENTIALS 5
2.3. CREATING A VARIANT 5
2.4. SETTING UP THE PUSH NOTIFICATIONS SERVICE SDK 6

CHAPTER 3. REGISTERING DEVICE ON PUSH NOTIFICATIONS SERVICE 7

CHAPTER 4. SENDING A PUSH NOTIFICATION USING THE UNIFIED PUSH ADMIN UI 8

CHAPTER 5. HANDLING INCOMING PUSH NOTIFICATIONS 11

PREFACE

The Push Notifications service allows you to send native push notifications to different mobile operating systems.

- Code once and push notifications to iOS and Android.
- Push notifications to either iOS only or Android only.
- Push notifications to different variants of a mobile app.

The service supports:

- Apple Push Notification Service
- Firebase Cloud Messaging

The Push Notifications service offers a unified Notification Service API to the above mentioned Push Network Services. It can be seen as a *broker* that distributes push messages to different 3rd party Push Networks.

NOTE

- The payload of the push notification is delivered to 3rd party Push Network providers, Google or Apple. AeroGear recommends that users do not send any sensitive personal or confidential information (for example, a social security number, financial account or transactional information) as part of any Push Notification. Users of push notifications should not have an expectation of secure transmission.
- For analytic purposes, the service stores the content of the alert key sent to the UnifiedPush Server. The content of the alert key belongs to the metadata, which is deleted after 30 days, using a nightly job within the UnifiedPush Server.
- You are only able to send push notifications to a real device, sending push notifications to an emulator fails.
- Push is a signaling mechanism and is not suitable to be used as a data carrying system, for example, a chat application.

Additional resources

- Apple Push Notification Service
- Firebase Cloud Messaging

CHAPTER 1. PUSH NOTIFICATIONS TERMINOLOGY

This section describes terminology that is associated with Push Notifications.

Push Application

A logical construct that represents an Mobile App, for example, *Mobile HR*.

Push Notification Message

A simple message to be sent to a Push Application.

Sender Endpoint API

A RESTful API that receives Push Notification Message requests for a PushApplication or some of its different Variants. The Server translates this request into the platform specific details and delivers the payload to the 3rd party cloud providers, which eventually might deliver the message to the physical device.

Variant

A variant of the Push Application, representing a specific mobile platform, like iOS or Android, or even more fine-grained differentiation like iPad or iPhone. There can be multiple variants for a single Push Application (for example, *Mobile HR Android*, *Mobile HR iPad*, *Mobile HR iOS free* or _Mobile HR iOS premium). Each supported variant type contains some platform specific properties, such as a Google API key (Android) or passphrase and certificate (Apple).

APNs

Apple Push Notification service .

Installation

Represents an actual device, registered with the UnifiedPush Server. User1 running *HR Android* app, while User2 runs *HR iPhone premium* on his phone.

Administrative User Interface

(AUI) The Unified Push Admin UI Web UI that allows you manage Push Applications and Variants, view statistics and send Push Notifications to devices.

CHAPTER 2. SETTING UP THE PUSH NOTIFICATIONS MOBILE SERVICE

2.1. OBTAINING FIREBASE CLOUD MESSAGING CREDENTIALS

This procedure describes how to obtain Firebase Cloud Messaging Credentials.

Prerequisites

Before the Android application is able to receive the notifications, you must set up access to Firebase Cloud Messaging. The following credentials are necessary to set up Firebase Cloud Messaging for your app:

- **Server key**
- **Sender ID**
- **google-services.json** file containing the credentials required to connect your app to Firebase and Google services.

Procedure

1. From the *Project Settings* screen, switch to the *Cloud Messaging* tab, where you can find the **Server key** and **Sender ID** (known in GCM as **Project Number**). There is also a *Legacy server key* but it should not be used for new projects.
2. Download the **google-services.json** file as described in the Google Documentation.

2.2. OBTAINING APPLE PUSH NOTIFICATION SERVICE CREDENTIALS

This procedure describes how to enable Push Notifications for your iOS application and get the credentials required for push from Apple.

Procedure

1. Follow the official Apple guide to enable push notifications for your Xcode project.
2. Follow the official Apple guide to generate an APNs client TLS certificate and export the client TLS identity from your Mac.

NOTE

Make sure to protect the p12 file with a password.

2.3. CREATING A VARIANT

This procedure describes how to register your app on the UnifiedPush Server

Procedure

1. Log into the Solution Explorer.
2. Click **Open Console** beside **Push Notification Service**.

3. If push applications already exist, they are displayed, click "Create Application" to navigate to the Welcome page. If no push applications exist, the Welcome page is displayed.
4. Click **Start Here** to dismiss the Welcome Page and go to the **Create Application** screen
5. Enter a name for your application and click **Create App** to go to the Variant screen
6. Click **Add a Variant**
7. Enter a name for your variant and select the platform that you intend to target
8. Enter the Push Network details necessary for your push network and click **Create**
9. Copy each of the various code snippets into its corresponding file in your application
10. Build and deploy your app
11. Click the **Skip the wizard** link to finish.

2.4. SETTING UP THE PUSH NOTIFICATIONS SERVICE SDK

This section helps you to set up the Push Notifications service SDK in your App. It describes how to set up and initialize the Push Notifications service SDK.

Android

A Firebase account.

iOS

An APNs service.

Procedure

1. Import the libraries
2. Install **cordova-plugin-aerogear-push**:

```
$ cordova plugin add @aerogear/cordova-plugin-aerogear-push
```

3. Install the Unified Push Server package needed for device registration:

```
$ npm install --save @aerogear/push
```

CHAPTER 3. REGISTERING DEVICE ON PUSH NOTIFICATIONS SERVICE

To receive native push notifications from a Push Network, for example APNs or FCM, the mobile device is identified with a unique device-token assigned by that Push Network. This device-token is passed, by the operating system, to the mobile application. Refer to the operating system and Push Network documentation for further details.

Every time a user launches a mobile app, that app receives the **device-token**, from a *platform-specific* method (or callback). Since the Push Network may assign a new token to a device, AeroGear recommends that the app registers the **device-token** with the UnifiedPush Server each time.

The required metadata for an **Installation**:

- **deviceToken:** _Identifies the device/user-agent within its Push Network.
- **variantID:** The ID of the variant, where the client belongs to
- **variantSecret:** Password of the actual variant

The AeroGear UnifiedPush Server is able to store some user-specific metadata as well:

- **deviceType:** The device type of the device or the user agent.
- **operatingSystem:** The name of the underlying Operating System.
- **osVersion:** The version of the used Operating System.
- **alias:** Application specific alias to identify users with the system. For instance an **email address** or a **username**.
- **categories:** Used to apply one or more "tags".

The device-token needs to be registered with the UnifiedPush Server, to indicate there is a new Installation for a Variant. This registration is performed by calling an endpoint of the server.

```
import { PushRegistration } from "@aerogear/push";

new PushRegistration(app.config).register().then(() => {
    // Registration with UPS successful, you can now send push notifications from the UPS UI
}).catch(err => {
    // Error on device registration
});
```

NOTE

Optionally, you can pass the parameters below to the register method

```
{
  alias: 'some-alias',
  categories: ['one', 'or', 'more', 'categories'],
  timeout: 5000 // in milliseconds
}
```

CHAPTER 4. SENDING A PUSH NOTIFICATION USING THE UNIFIED PUSH ADMIN UI

The Unified Push Admin UI allows you to send Push Notifications.

Prerequisites

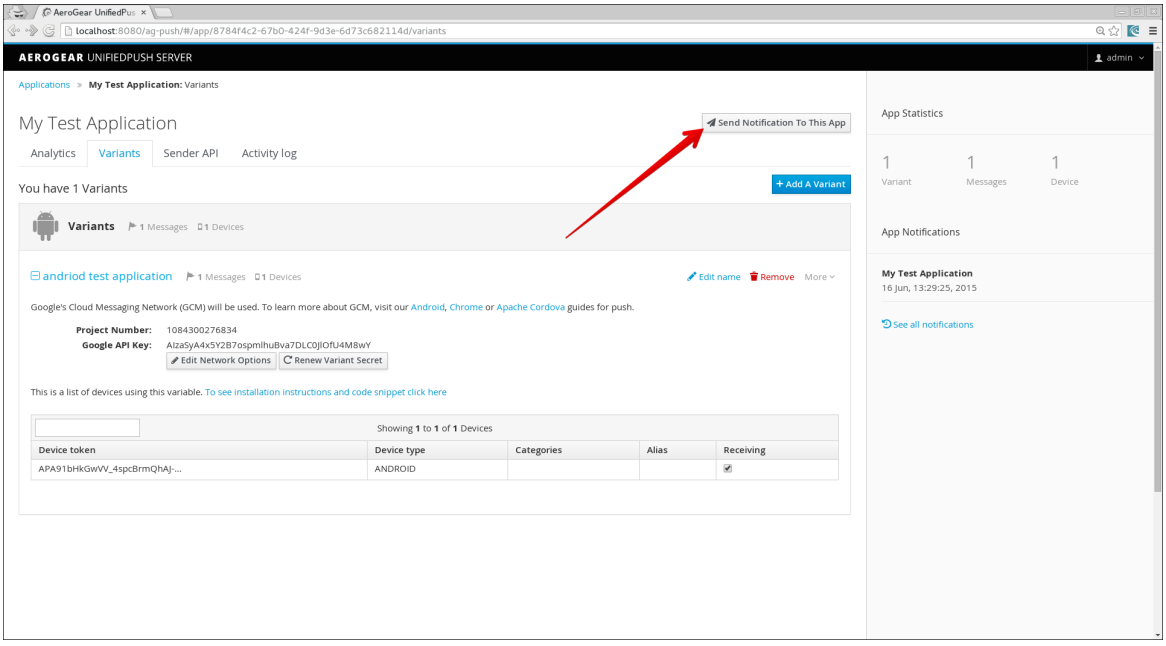
- Make sure the Push Notifications service is provisioned.
- Select a route in OpenShift.
- Login with your OpenShift credentials.

NOTE

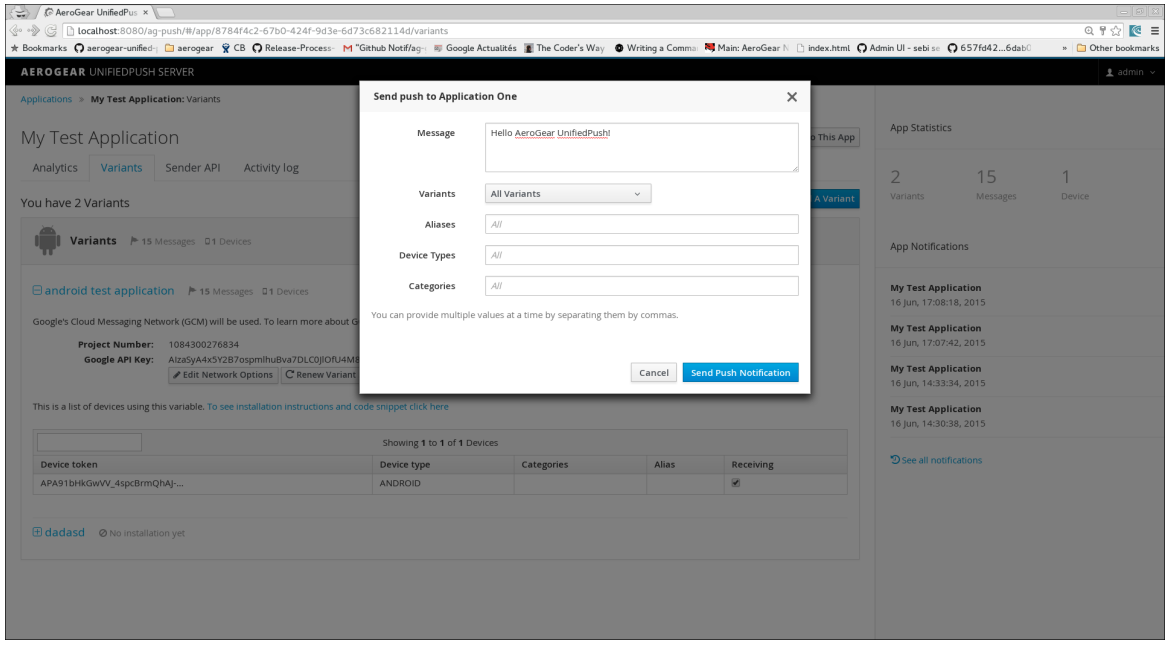
On first login you need to provide the OpenShift OAuth service permissions to read your user account.

Admin UI

1. Open the Unified Push Admin UI in a browser.
2. Select the target application from the home page and click **Send Notification To This App**



3. When the *Send Push* dialog displays, enter text in the **Message** form.



4. Click **Send Push Notification** to send the message to the target application.

Java API

1. Add unifiedpush-java-client as a dependency to your project.

```
<dependency>
  <groupId>org.jboss.aerogear</groupId>
  <artifactId>unifiedpush-java-client</artifactId>
  <version>[version]</version>
</dependency>
```

2. Send the message to the target application.

```
final PushSender sender = DefaultPushSender
        .withRootServerURL("<pushServerURL e.g http(s)//host:port/context>")
        .pushApplicationId("<pushApplicationId e.g. 1234456-234320>")
        .masterSecret("<masterSecret e.g. 1234456-234320>")
        .build();

final UnifiedMessage unifiedMessage = UnifiedMessage
        .withMessage()
        .alert("Hello from Java Sender API!")
        .build();

sender.send(unifiedMessage, () -> {
    //do cool stuff
});
```

Node.js API

1. Add unifiedpush-node-sender as a dependency to your project.

```
npm i unifiedpush-node-sender
```

2. Send the message to the target application.

```
const agSender = require('unifiedpush-node-sender');

const settings = {
  url: "<pushServerURL e.g http(s)//host:port/context>",
  applicationId: "<pushApplicationId e.g. 1234456-234320>",
  masterSecret: "<masterSecret e.g. 1234456-234320>"

};

const message = {
  alert: "Hello from the Node.js Sender API!"
};

const options = {
  config: {
    ttl: 3600
  }
};

agSender(settings).then((client) => {
  client.sender.send(message, options).then((response) => {
    console.log('success', response);
  }).catch((error) => {
    console.log('error', error);
  })
});
```

REST

1. Send the message to the target application.

```
curl -u "<pushApplicationId>:<masterSecret>"  \
   -v -H "Accept: application/json" -H "Content-type: application/json"  \
   -X POST  -d \
  '{
     "message": {
      "alert": "Hello from the curl HTTP Sender!",
      "sound": "default"
     }
   }'  \
   <pushServerURL>/rest/sender
```

NOTE

The 3rd party Push Network is responsible for delivering the Push Notification to the target application.

CHAPTER 5. HANDLING INCOMING PUSH NOTIFICATIONS

This section describes how to handle incoming push notifications in your foregrounded application.

NOTE

Push notifications that arrive when the application is in the background are always handled by the OS.

Procedure

1. Add the following code to your app:

```
import { PushRegistration } from "@aerogear/push";

PushRegistration.onMessageReceived((notification: any) => {
  console.log('Received a push notification', notification);
});
```

2. Build and run your app.