Table of Contents

Push Notification Service for Pivotal Cloud Foundry

This is documentation for the Push Notification Service for Pivotal Cloud Foundry (PCF).

Product Snapshot

Current Push Notification Service for PCF Details

- **Version**: v1.7.1
- **Release Date**: January 2017
- **SDK Versions**: Android v1.7.0, iOS v1.7.0
- **Compatible Ops Manager Version(s)**: v1.7.8 and later
- **Compatible Elastic Runtime Version(s)**: v1.7.x, v1.8.x, v1.9.x
- **vSphere support?** Yes
- **AWS support?** Yes
- **GCP support?** Yes

Upgrading to the Latest Version

Please upgrade to Push v1.4.5+ before upgrading to PCF v1.7.1.

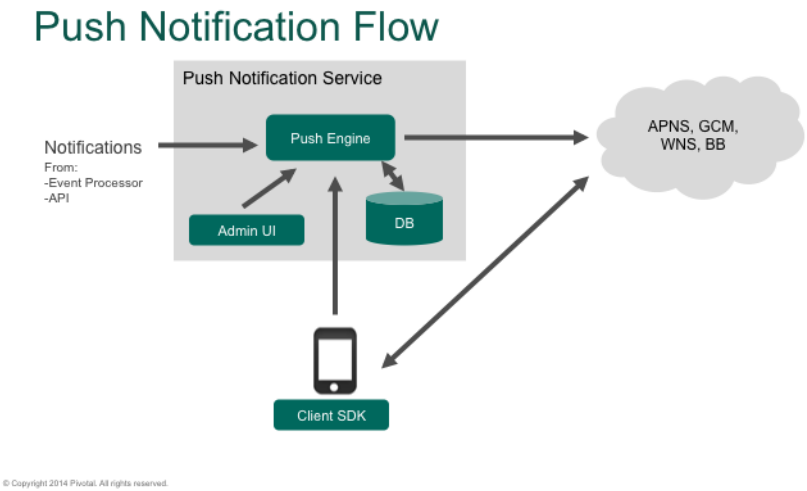
Consider the following compatibility information before upgrading the Push Notification Service for Pivotal Cloud Foundry.

Ops Manager Version	Supported Upgrades from Imported Push Installation	
	From	To
v1.3.x	v1.1.0	v1.1.1
	v1.1.1	v1.2.0
	v1.2.0	v1.2.1
v1.4.x	v1.2.0	v1.2.1
	v1.3.0	v1.3.1
v1.5.2+	v1.2.0	v1.2.1
	v1.2.1	v1.3.0
	v1.3.0	v1.3.1
	v1.3.x	v1.3.2 – v1.3.5
v1.6.x	v1.3.5	v1.4.5+
v1.7.8+	v1.4.5+	v1.7.1
v1.8.x	v1.4.5+	v1.7.1

About

The Pivotal Push Notification Service for Pivotal Cloud Foundry allows developers to create a mobile backend that can be used to send push notifications to mobile apps. The service connects and manages the interface to Apple Push Notification Service, Google Cloud Messaging, Windows Push Notification Services, and BlackBerry Push Service.

Each mobile app communicates with the service for registrations and notification preferences by using the corresponding client SDK. Back-end business logic servers send push notifications to all users, users by platform, users by geolocation, or users with specified tags by sending the request to the appropriate Push Notification Service API endpoint.



For installation, a PCF administrator initially imports the Pivotal Push Notifications package into PCF Operations Manager and configures it via the Dashboard at which point the service becomes available to send notifications. The Dashboard provides the ability to configure apps, environments, and device-specific service parameters. Client SDKs for iOS and Android provide a simplified way to integrate with the Push Notifications service. Windows Phone 8, Windows 8, and Blackberry 10 apps can also use the service.

The Push Notifications service requires:

- Pivotal RabbitMQ
- Redis database (Pivotal Redis or user provided)
- MySQL database (Pivotal MySQL or user provided)

Installation

This document describes how to install the Pivotal Cloud Foundry (PCF) Push Notification Service.

The PCF Push Notification Service installs as a suite of five CF apps deployed in the `system` org under the `push-notifications` space.

- API
- Dashboard
- Service Broker
- Scheduler
- Analytics

A default installation deploys five Application Instances (AIs), one for each app show above. For **production deployments**, Pivotal recommends deploying a minimum of two instances for each push app, 10 AIs total, per PCF environment. Additional API application instances may be required depending on the peak load required, with peak load defined as the maximum number of notifications sent per second.

Dependencies

The Push Notification service depends on MySQL (optionally MySQL for Pivotal CF), RabbitMQ for Pivotal CF, and Redis for Pivotal CF being successfully installed on Pivotal Cloud Foundry.

Download the Product

Download the Push Notification software from Pivotal Network

Adding the Product

To get started with Push, you need to add the product with Pivotal Ops Manager.

Before you can complete the installation you must provide some configuration.

Set Encryption Key

From Ops Manager click on the Pivotal Push Notification Service tile and go to the “Security Settings” section. Generate an encryption key by running the following command in terminal (you should set your own password here):

```
openssl enc -aes-128-cbc -k samplepassword -P -md sha1
```

This produces a salt, key, and initialization vector. Copy the key into the “Encryption Key” field on Ops Manager and click “Save”. This key is used for symmetric encryption of push certificates and API keys.

Configure MySQL

From Ops Manager click on the Push Notification Service tile and go to the “MySQL Settings” section. Select MySQL Service to use MySQL for PCF. See the [Installation] section of the MySQL for PCF documentation for more information.

When using the MySQL for PCF service for Push Notifications, you must provide a MySQL for PCF service plan name. Pivotal recommends creating a custom MySQL for PCF service plan called “Push”. You can find instructions for creating a custom service plan in the MySQL for PCF Service Plans documentation. After you identify the appropriate service plan, enter its name in the text field, such as “Push”.

To use an external (user provided) MySQL server select “External” and fill in the required fields.

After you have completed this configuration click “Save”.

Configure Redis for Analytics and Logs

From Ops Manager click on the Push Notification Service tile and go to the “Analytics Redis Settings” section. Select the Redis service to use Pivotal Redis service. If you select this option you must install the Pivotal Redis service as well. Select from the drop-down the type of service plan to use. See more information about the Pivotal Redis service

To use an external (user provided) Redis server select “External” and fill in the required fields. - NOTE: This release does not supportRedis Cluster if you are using external redis. - If you are using redis behind a tcp proxy, make sure to use Session Persistence.

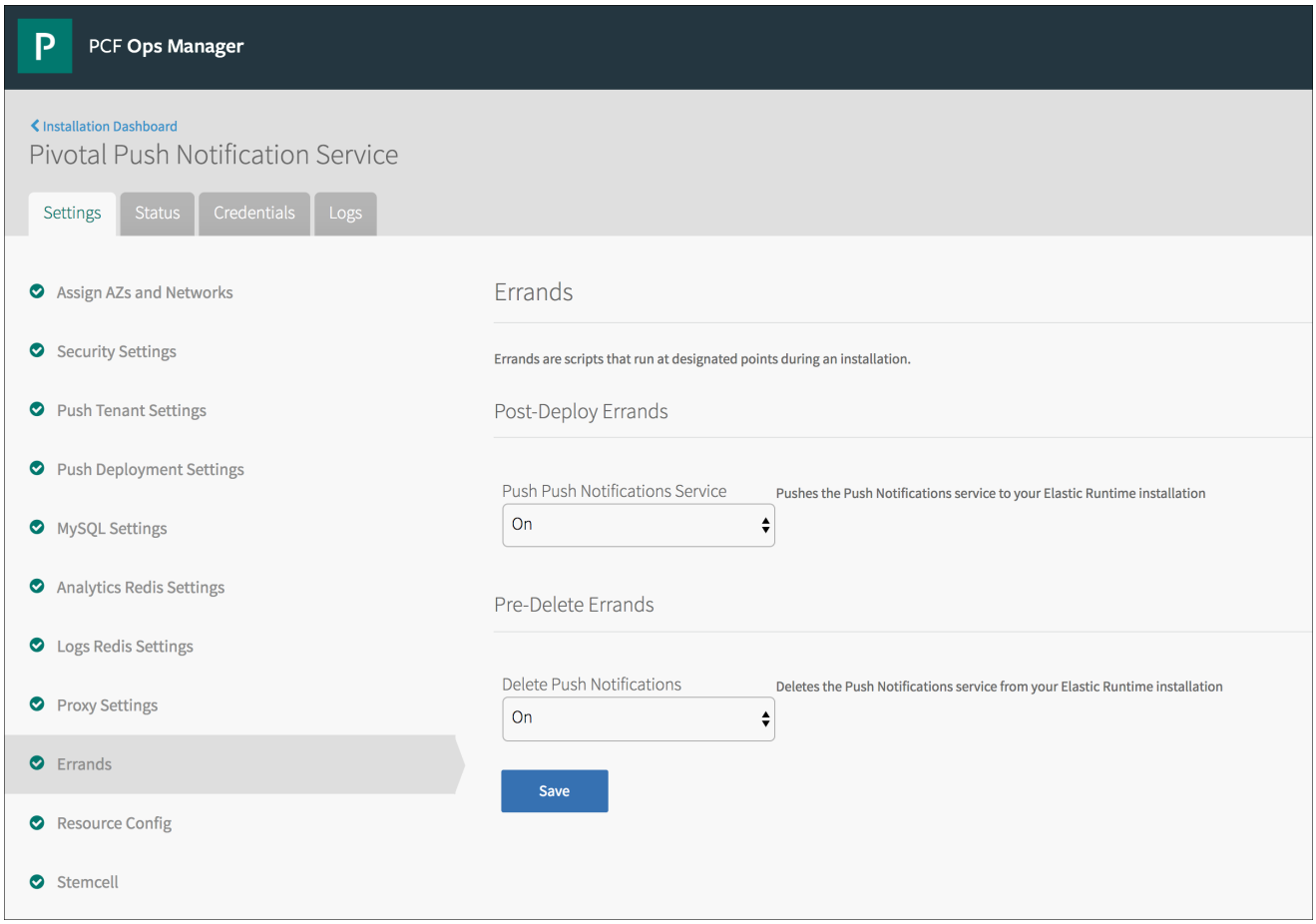
The same steps apply to set the “Logs Redis Settings” section as above.

After you have completed these configurations click “Save”.

Default Errand Behavior

As of v1.10, Ops Manager skips all unnecessary BOSH errands when performing updates to PCF services. For more information about this behavior, see the Ops Manager documentation, Managing Errands in Ops Manager.

For PCF Push Notification services, Pivotal **strongly** recommends that operators set the default Errand execution behavior to **On**, through the **Errands Form** in the Push Notifications tile settings in Ops Manager.



Upload Stemcell

Ops Manager versions greater than v1.5 require that you upload the stemcell that the Push Notification Service uses. You can acquire this stemcell from the Bosh Stemcell Directory. After you have the stemcell, upload it to Ops Manager via the “Stemcell” tab in the Push Notification Services configuration page.

Apply Changes

After the security settings and MySQL configuration are complete you can click “Installation Dashboard” to return to the Ops Manager dashboard and then click “Apply Changes” to complete the installation.

Creating a Tenant

Since v1.4, the PCF Push Notification Service supports multiple tenants. Each tenant in the PCF Push Notification Service can have its own set of applications. In order to set up a new tenant, you need to create a new space in your PCF Apps Manager. You can use any org that is appropriate for your needs.

The applications for the Push Notification Service itself are in the “push-notifications” space in the “system” org. Don’t use this space for your own tenant. Create a new space instead.

After you have selected your space you can create your Push service instance by clicking the “Add Service” button. Select the “PCF Push Notification Service” service from the Marketplace. Select the default (free) plan. Give the service a name and add it to your space.

Only create one instance of the Push Notification Service per space.

After the service instance is created you can click the “Manage” link on the service instance to show the Dashboard for the Push Notification Service.

You can control access to the Push Dashboard by using the using Cloud Controller. Any users with access to see the space also have access to use the Push Notification Dashboard. You need to be logged in to the Apps Manager before you can access the Push Dashboard.

Dashboard setup

After the service has been added, verify the successful installation by viewing the dashboard.

Note:

The Push Notification service is a CF Service that is installed in the “System” org and “push-notifications” space. You see it in the Marketplace. Each instance of the Push Notifications Service has its own dashboard URL.

Login as “admin” to the CF console and go to that org and space. To access the Push Dashboard, click on the “Manage” link for the “push-service-instance” service.

Installation Verification

There are two different ways to manually verify the installation was successful.

The first way is to use the CF CLI to view the installed apps and services. Instructions to log in are included on the CF CLI page.

The organization is “System” and the space is “push-notifications”, both are needed to view the apps and services using the CF CLI.

After setting the api and logging in to the CF CLI, type in `cf a` to see a listing of all the apps currently under the push-notifications space, with a quick overview of their current status.

```
$ cf apps
Getting apps in org system / space push-notifications as admin...
OK

name                requested state   instances   memory   disk   urls
push-service-broker   started           1/1         512M     1G     push-service-broker.cove.cf-app.com
push-analytics        started           1/1         1G       1G     push-analytics.cove.cf-app.com
push                  started           1/1         512M     1G     push.cove.cf-app.com
push-scheduler        started           1/1         512M     1G     push-scheduler.cove.cf-app.com
push-api              started           1/1         2G       1G     push-api.cove.cf-app.com, push-notifications.cove.cf-app.com
```

The apps that should appear are as follows:

- Dashboard (push)
- Backend (push-api)
- Scheduler (push-scheduler)
- Analytics (push-analytics)

- Service Broker (push-service-broker)

And they should all have their own unique urls.

For the services, typing in `cf s` gives a list of the services plus the apps which they are bound to.

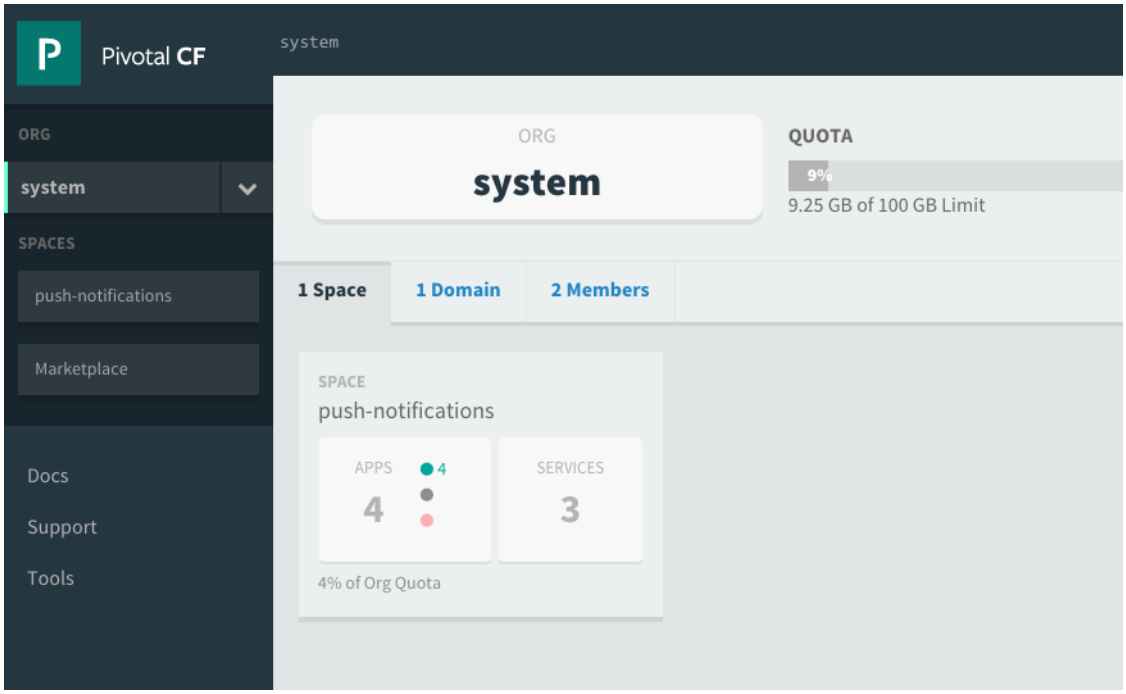
```
$ cf services
Getting services in org system / space push-notifications as admin...
OK

name                              service           plan          bound apps                          last operation
push-notifications-logs-redis     p-redis           dedicated-vm  push-analytics                      create succeeded
push-notifications-rabbitmq       p-rabbitmq        standard      push-analytics, push-api            create succeeded
push-notifications-mysql          p-mysql           100mb-dev     push, push-api                      create succeeded
push-notifications-analytics-redis p-redis          dedicated-vm  push-analytics, push-scheduler      create succeeded
push-service-instance             pcf-push-service  default                                           create succeeded
```

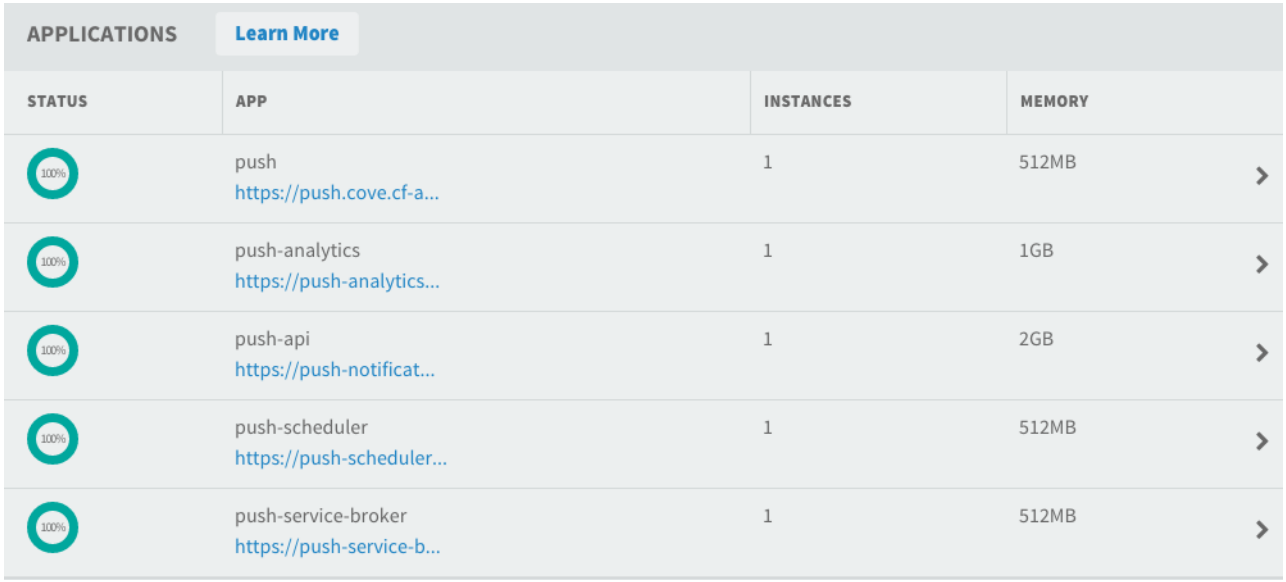
The services that should appear are as follows:

- MySQL (push-notifications-mysql)
- RabbitMQ (push-notifications-rabbitmq)
- Redis for Analytics (push-notifications-analytics-redis)
- Redis for Logs (push-notifications-logs-redis)
- Push Service Broker (push-service-broker)

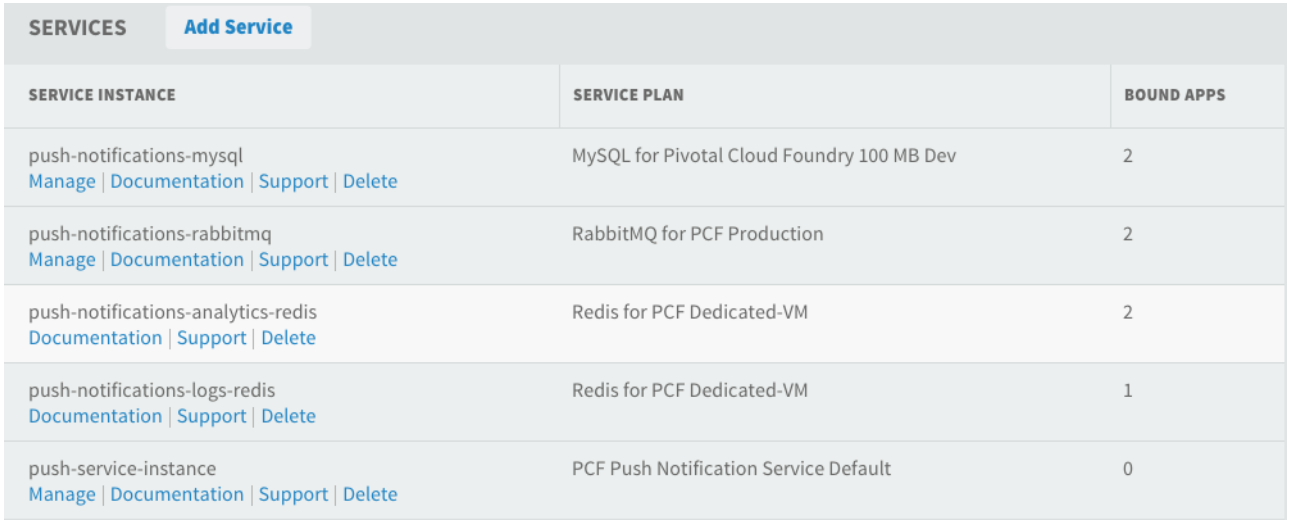
The second way is to use the developer console. After logging in, select the System organization from the dropdown box. Selecting the organization shows all of the spaces which are nested within.



Click on the push-notifications space, which then shows the apps and services running under that space.



The listing of applications show the status, the name, the url to access the app, how many instances of that app is running, and how much memory that app is using. Verify that each apps status is 100%, which means it is running as expected.



The listing of services show the name, the plan, and how many apps are bound to it. Some services have extra options, such as managing the service, or looking up documentation on the service.

Notes

There is no automated upgrade path from v1.2.x to v1.3.0. Steps are available to backup and restore data between these versions.

For v1.0 - v1.2, the default location for the dashboard `http://push-notifications-dashboard.cf.example.com`

For v1.0 - v1.1, the default installation installs the applications to the “Pivotal” org and “push-notifications” space.

DevOps

Uninstalling

IMPORTANT

Push is a stateful service!

It is advised that you do **NOT UNINSTALL** the Push tile in order to solve problems with binding or communicating with other services. The Push team will provide instructions on how to manually restore these connections.

Deleting the tile will cause all of the Push user data stored in the MySQL, Redis, and RabbitMQ services to be **DELETED** as well.

If you need to delete the Push tile or delete any of its connections to the above services then you will need to **BACKUP** and **RESTORE** all of the Push user data in these services.

Instructions for backing up and restore the user data is provided below.

Troubleshooting Common Problems

For solutions to common problems, please see our troubleshooting guide.

Configurable Environment Variables

Push Api

```
push_security_trustAllCerts (Boolean, default: inherited from cf runtime)
```

When the `push_security_trustAllCerts` environment variable is set to `true` the Push API will skip SSL validation on calls to RabbitMQ and the Push Scheduler. This variable is necessary in environments that use self-signed certificates. The default value is `false` unless the CF Runtime is configured to trust self-signed certificates.

Certificates generated in Elastic Runtime are signed by the Operations Manager Certificate Authority. They are not technically self-signed, but they are referred to as 'Self-Signed Certificates' in the Ops Manager GUI and throughout this documentation.

```
push_scheduler_sendImmediatelyWithin (Integer, default: 60)
```

The `push_scheduler_sendImmediatelyWithin` environment variable pertains to scheduled push notifications. It is a threshold (in seconds) within which the push server will skip scheduling a push and simply send it right away. The default value is 60 seconds. If a push is scheduled within 60 seconds of the current time it will not be scheduled but simply be sent right away. You can modify that threshold by modifying this environment variable.

```
push_apns_sendReceipt (Boolean, default: true)
```

The `push_apns_sendReceipt` environment variable is a flag that enables passing a receipt to the device as part of the push payload. The receipt is a unique id for each message that can be used for analytics. This flag enables sending receipts for iOS/APNS.

```
push_apns_logDeviceTokens (Boolean, default: true)
```

The `push_apns_logDeviceTokens` environment variable controls the log verbosity of the APNS push handler. When set to `true` the device token for every recipient of a push will be logged as the push is sent. Note that this extra logging will reduce push throughput.

```
push_gcm_sendReceipt (Boolean, default: true)
```

The `push_gcm_sendReceipt` environment variable is a flag that enables passing a receipt to the device as part of the push payload. The receipt is a unique id for each message that can be used for analytics. This enables sending receipts for Android/GCM.

```
push_gcm_logDeviceTokens (Boolean, default: true)
```

The `push_gcm_logDeviceTokens` environment variable controls the log verbosity of the Android push handler. When set to `true` the device token for every recipient of a push will be logged as the push is sent. Note that this extra logging will reduce push throughput.

Installing the push server behind a proxy (available in v1.3.2+)

Starting in Push version 1.3.2 you can route communication with push providers (APNS, Google Cloud Messaging) through a proxy server. GCM pushes can use either a HTTP or socks proxy. APNS pushes can only use a socks proxy. Use the following environment variables to specify proxies.

```
push_gcm_httpProxyHost (String, default: [empty])
push_gcm_httpProxyPort (Integer, default: [empty])
```

The `push_gcm_httpProxyHost` and `push_gcm_httpProxyPort` environment variables allow you to specify an HTTP proxy server through which to route Google API requests (for Android pushes).

```
push_gcm_socksProxyHost (String, default: [empty])
push_gcm_socksProxyPort (String, default: [empty])
```

The `push_gcm_socksProxyHost` and `push_gcm_socksProxyPort` environment variables allow you to specify a SOCKS proxy through which to route Google API requests.

Note: If both HTTP and SOCKS proxies are defined for GCM, SOCKS will be used.

```
push_apns_socksProxyHost (String, default: [empty])
push_apns_socksProxyPort (String, default: [empty])
```

The `push_apns_socksProxyHost` and `push_apns_socksProxyPort` environment variables allow you to specify a SOCKS proxy through which to route APNS push requests.

Backup And Restore

Backup MySQL data

It is highly recommended that you enable automatic backups with your MySQL Tile (Requires an Amazon s3 Bucket). Additionally, you should always backup your MySQL tile if you are planning on removing Push Notification Service or MySQL. You can perform a manual backup by following the directions found here: MySQL Manual Backup

Follow these instructions to backup *solely* the Push Notification database.

- In the Apps Manager console in the “system” org go to the “push-notifications” space and the “push-analytics” app.
- Go to the “Services” tab.
- Click “▸ Show credentials” for the MySQL service.
- Get “username”, “password” and “database name”.
- SSH into the proxy for your Pivotal CF environment.
- From the proxy run (using the credentials above):

```
mysqldump -h hostname -p -u username database_name > push_db.sql
```

Backup encryption key

- In the Apps Manager console go to the “push-api” app and go to the “Env Variables” tab.
- Get and record the value for `crypto_applicationKey`. You will need this key during the installation.
  - The `crypto_applicationKey` environment variable contains the key which will be used to encrypt sensitive information used by the push server

(i.e.: iOS push certificates, Google API keys). This value is set at install time and ***should not be modified***. You will however need to record this value in order save and restore the push notification service database.

Restore MySQL data

- From the Apps Manager console in the “push notifications” space through the “system” org, stop the “push” and “push-api” applications.
- Go to “Services”.
- Click “▸ Show credentials” for MySQL.
- Get “username”, “password” and “database name”.
- SSH into the proxy for your Pivotal CF environment.
- Delete data from Push installation (this should just be empty data) by running the following command from the proxy (using the above credentials):

```
mysql -h hostname -p -u username name -e "drop database database_name; create database database_name;"
```

- Import data from old install by running the following command from the proxy (using the above credentials):

```
mysql -h hostname -p -u username database_name < push_db.sql
```

- Enable migrations:
  - In the Apps Manager console, find the “push-api” application and go to the “Env Variables” tab.
  - Edit `liquibase_runMitgations` and set it to `true`.
- Start the “push-api” and “push” applications.
- Disable migrations:
  - In the Apps Manager console, find the “push-api” application and go to the “Env Variables” tab.
  - Edit `liquibase_runMigrations` and set it to 'false'.
- Restart the “push-api” and “push” applications.

Backup Redis Data

- See redis backup instructions

Configuring Heartbeat Monitor for iOS

This topic describes how Pivotal Cloud Foundry (PCF) operators can configure the Push Notification Heartbeat Monitor app for iOS.

Heartbeat Monitor is a Cloud Foundry app deployed by the PCF Push Notification service to help you ensure the service runs correctly end-to-end. It does this by sending a push, or *heartbeat*, every minute to the devices registered with the app. You can also select the app in the Push dashboard to view its historical data.

Follow the instructions below to configure Heartbeat Monitor and run the companion iOS app on your device.

Prerequisites

To configure the Heartbeat Monitor app for iOS, you must have the following:

- An iOS 8+ device
- The latest Xcode that supports Swift 2.2 installed on your workstation
- An Apple Developer account

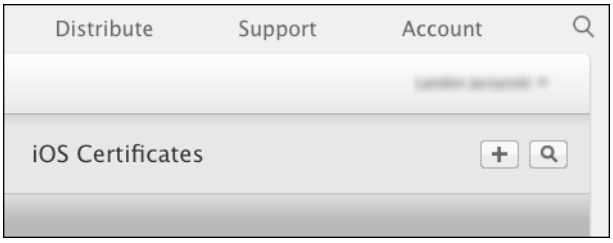
Request an iOS Development Certificate

Follow these steps to obtain an iOS development certificate:

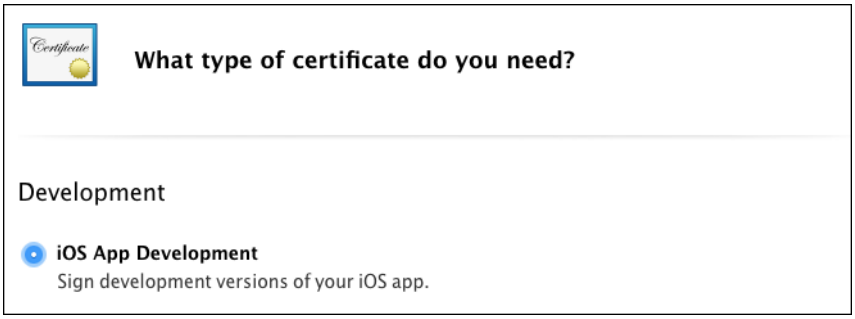
1. Navigate to the **Certificates, Identifiers & Profiles** section of the Apple Developer Portal.

2. In the side navigation, select **Certificates** > **All**.

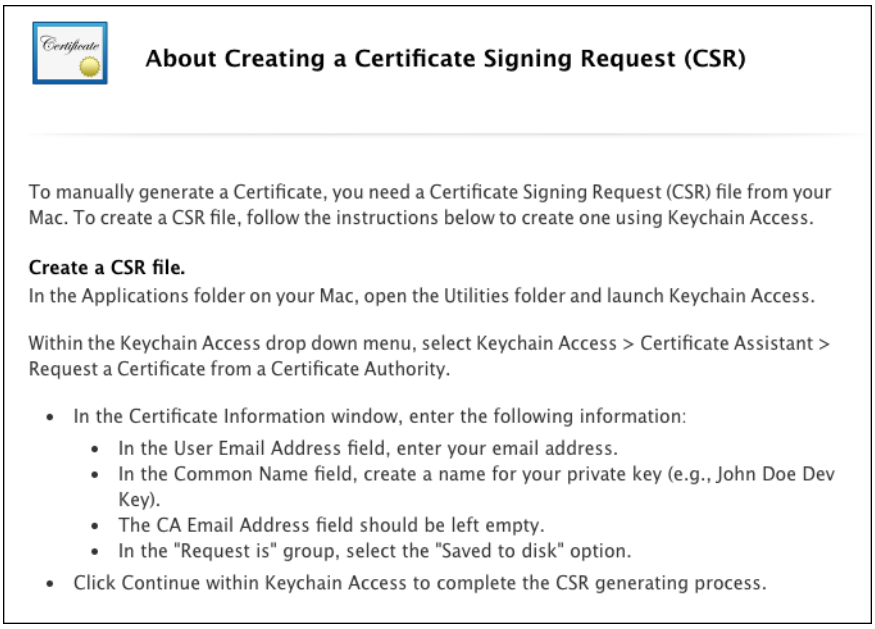
3. Click the **+** button in the top right to add a new certificate.



4. Select **iOS App Development** and click **Continue**.



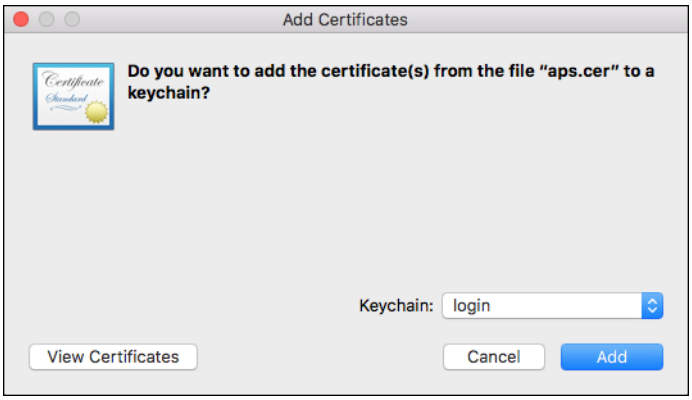
5. Follow the on-screen instructions to **Create a CSR file** and click **Continue**.



6. **Upload** the `.csr` file you created and click **Continue** to generate the new certificate.

7. Click **Download**.

8. Open your certificate and import it to the Keychain Access app when prompted.

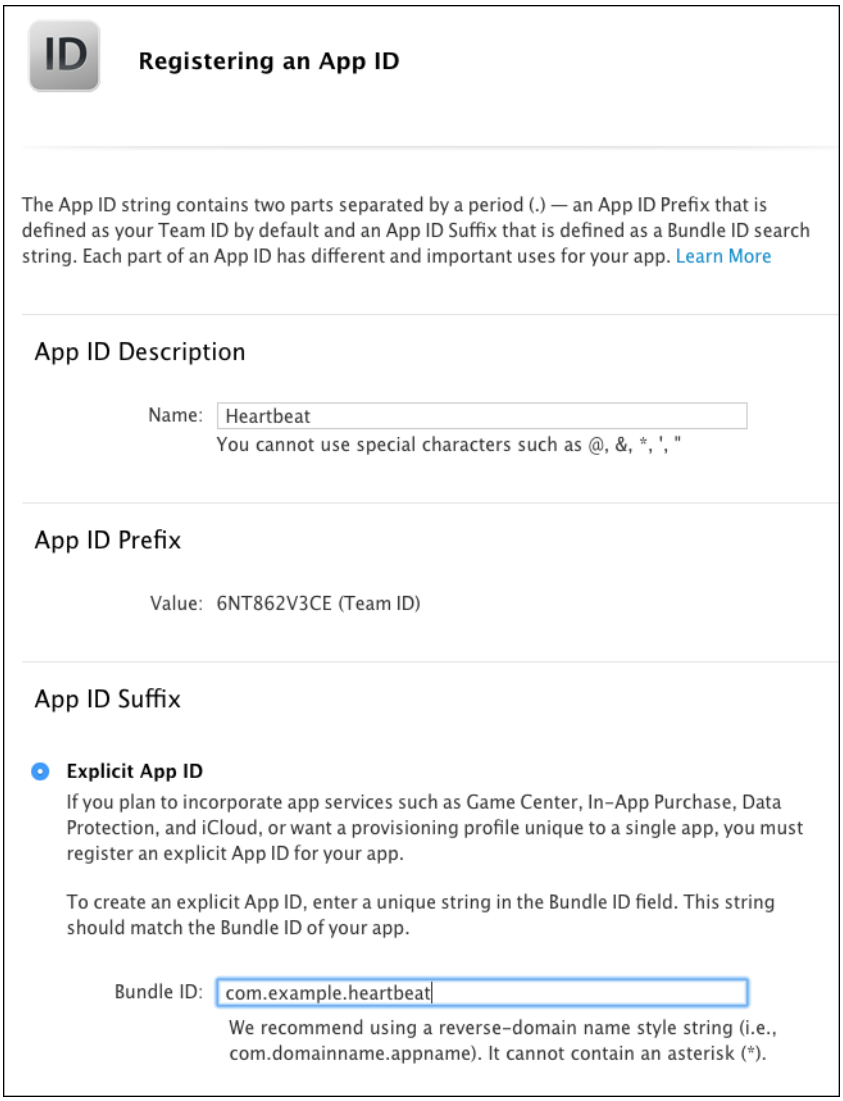


Request an APNS Certificate

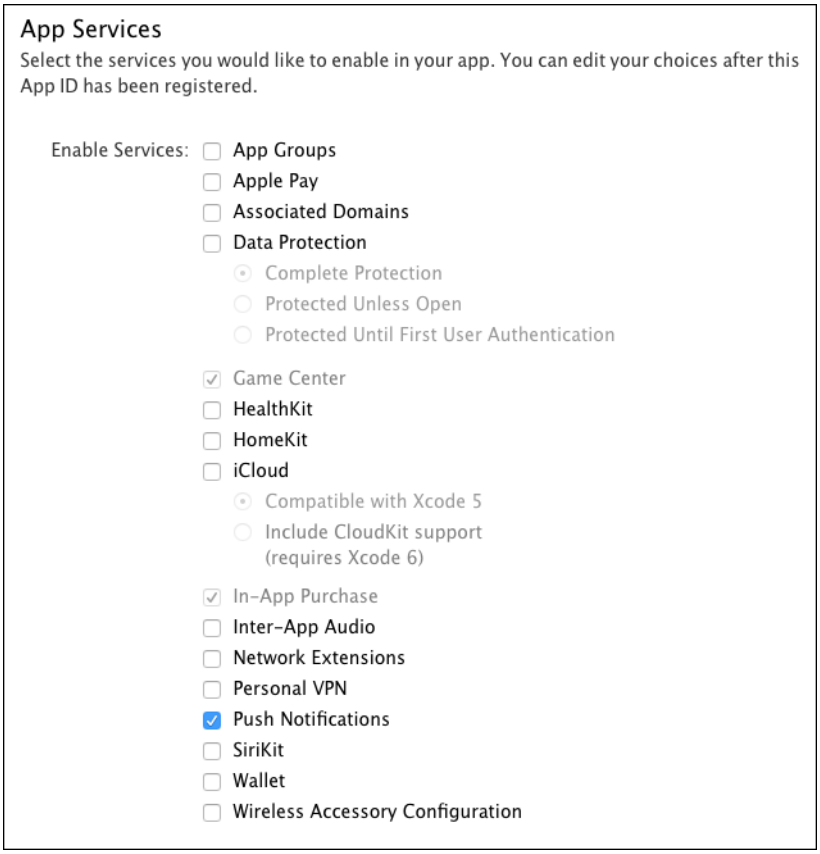
Follow these steps to enable your app to receive push notifications:

Create an App ID

1. Navigate to the **Certificates, Identifiers & Profiles** section of the Apple Developer Portal.
2. In the side navigation, select **Identifiers** > **App IDs**.
3. Click the **+** button to create an App ID.
4. Enter an **Name** and a **Bundle ID**.



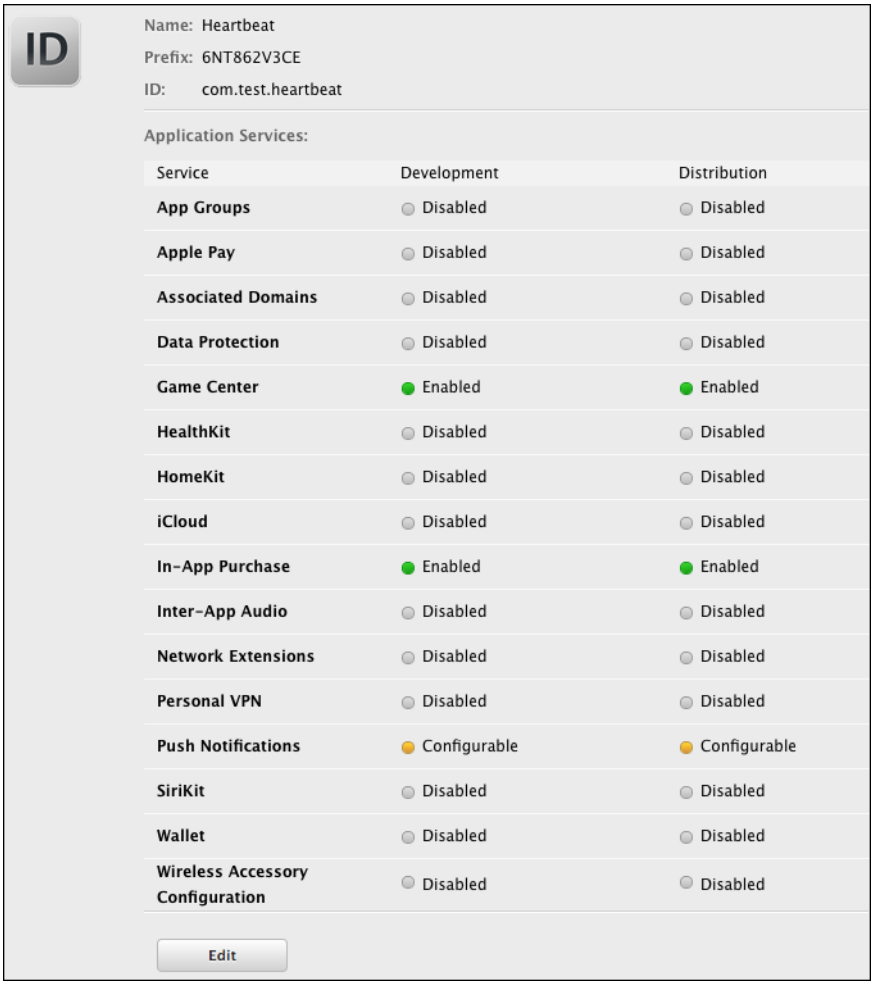
5. Select the **Push Notifications** checkbox and click **Continue**.



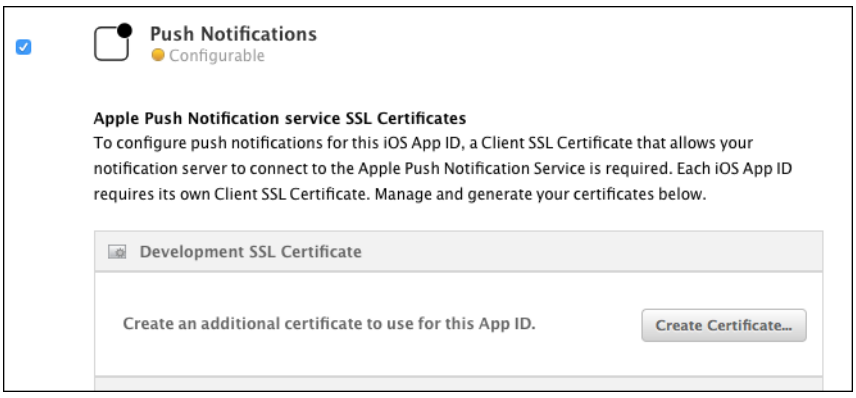
6. Click **Register**.

Create an APNS Certificate

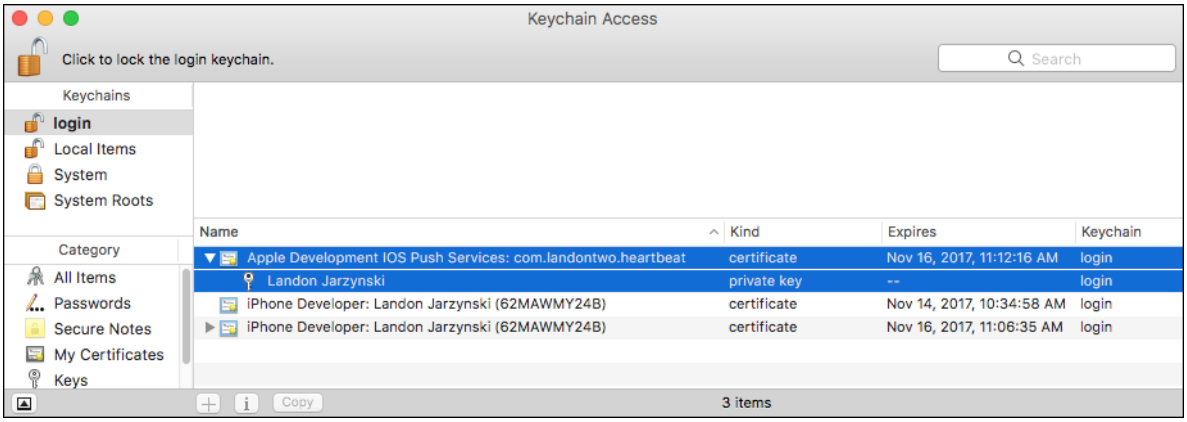
1. In the **App IDs** list, select the App ID you registered and click **Edit**.



2. Under the **Push Notifications** section, choose **Development SSL Certificate** and click the corresponding **Create Certificate** button.

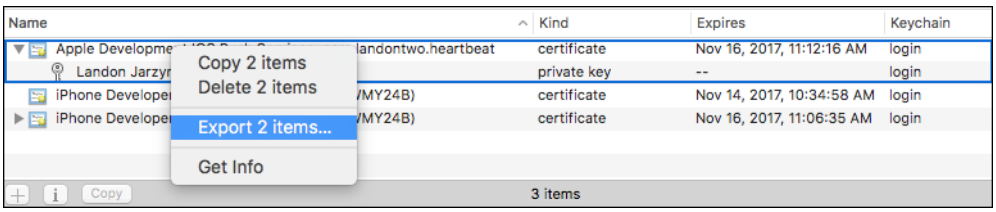


3. Follow the on-screen instructions to create a new CSR.
4. **Upload** the .csr file you created and click **Continue** to generate the new certificate.
5. Click **Download**.
6. Open your certificate and import it to Keychain Access when prompted.
7. In **Keychain Access**, select both your **Apple Development iOS Push Services** certificate and the private key it was signed with.



8. Right click your selection and choose **Export 2 Items...**

9. Save the **.p12** file for uploading to the Push dashboard in a later step.



Create a Provisioning Profile

Follow these steps to create a Provisioning Profile that you specify when building the Heartbeat Monitor iOS app in Xcode:

Register a Device

Note: You need an iOS 8+ Device to use Heartbeat Monitor. If you already have an iOS 8+ Device registered, skip to the next step.

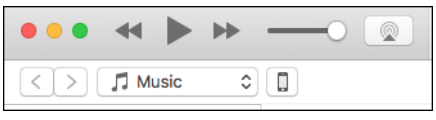
1. Navigate to the **Certificates, Identifiers & Profiles** section of the Apple Developer Portal.

2. In the side navigation, select **Devices** > **All**.

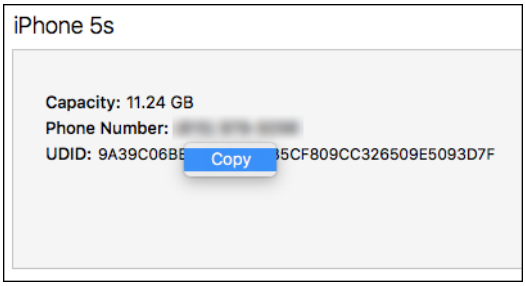
3. Click on the **+** button in the top right to add a device.

4. Retrieve the **UDID** of your device:

    a. Connect your device to your computer.
    b. Open **iTunes**.
    c. Select the device tab.



5. Click the **Serial Number** of the device to reveal the **UDID** and right click the field to copy it.



6. In the Apple Developer Portal, enter a **Name** for your device and paste the **UDID** into its field.

7. Click **Register**.

Create a Profile

1. Navigate to the **Certificates, Identifiers & Profiles** section of the Apple Developer Portal.
2. In the side navigation, select **Provisioning Profiles** > **All**.
3. Click on the **+** button in the top right to create a Provisioning Profile.
4. Choose the **iOS App Development** type and click **Continue**.
5. From the **App ID** dropdown, select the App ID you created earlier and click **Continue**.
6. Select the **Developer Certificate(s)** that you want to use and click **Continue**.
7. Select the **Devices** that you registered and click **Continue**.
8. Provide a descriptive **Name** for the Provisioning Profile and click **Continue**.
9. Click **Download** and open your **Provisioning Profile**.

> **Note**: When you open your Provisioning Profile, Xcode installs it without providing any confirmation. In a later step, you configure your Xcode app project to use this Provisioning Profile.

Configure your Push Dashboard

Follow these steps to navigate to the Push dashboard and then configure the service to talk to your device.

You can navigate to the Push dashboard using either Apps Manager or the Cloud Foundry Command Line Interface (cf CLI). Use the cf CLI instructions if you did not enable the **Push Apps Manager** errand when deploying Elastic Runtime.

Navigate to Push Dashboard using Apps Manager

1. In a browser, navigate to `apps.YOUR-SYSTEM-DOMAIN`.
2. Select the **system** org and the **push-notifications** space.
3. Click the **Services** tab.
4. Select the **PCF Push Notification Service** row and click the **Manage** link.

Navigate to Push Dashboard using cf CLI

1. Open a terminal window and log in:

```
$ cf login -a https://api.YOUR-SYSTEM-DOMAIN -u USERNAME -p PASSWORD
```

2. Target the correct org and space:

```
$ cf target -o system -s push-notifications
```

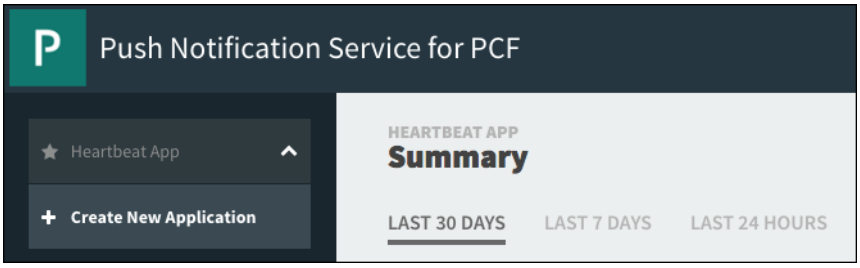
3. Run the following command:

```
$ cf service push-service-instance
```

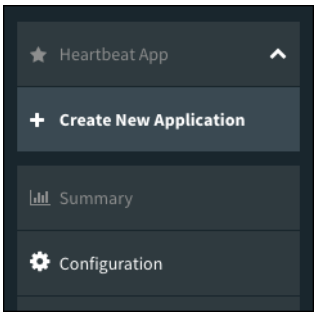
4. Copy the URL from the **Dashboard** field and paste it into your browser.

Configure the Push Notification Service

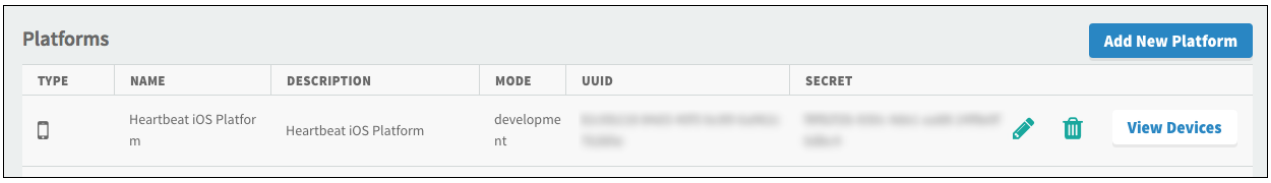
1. Select the **Heartbeat App** from the list of applications.



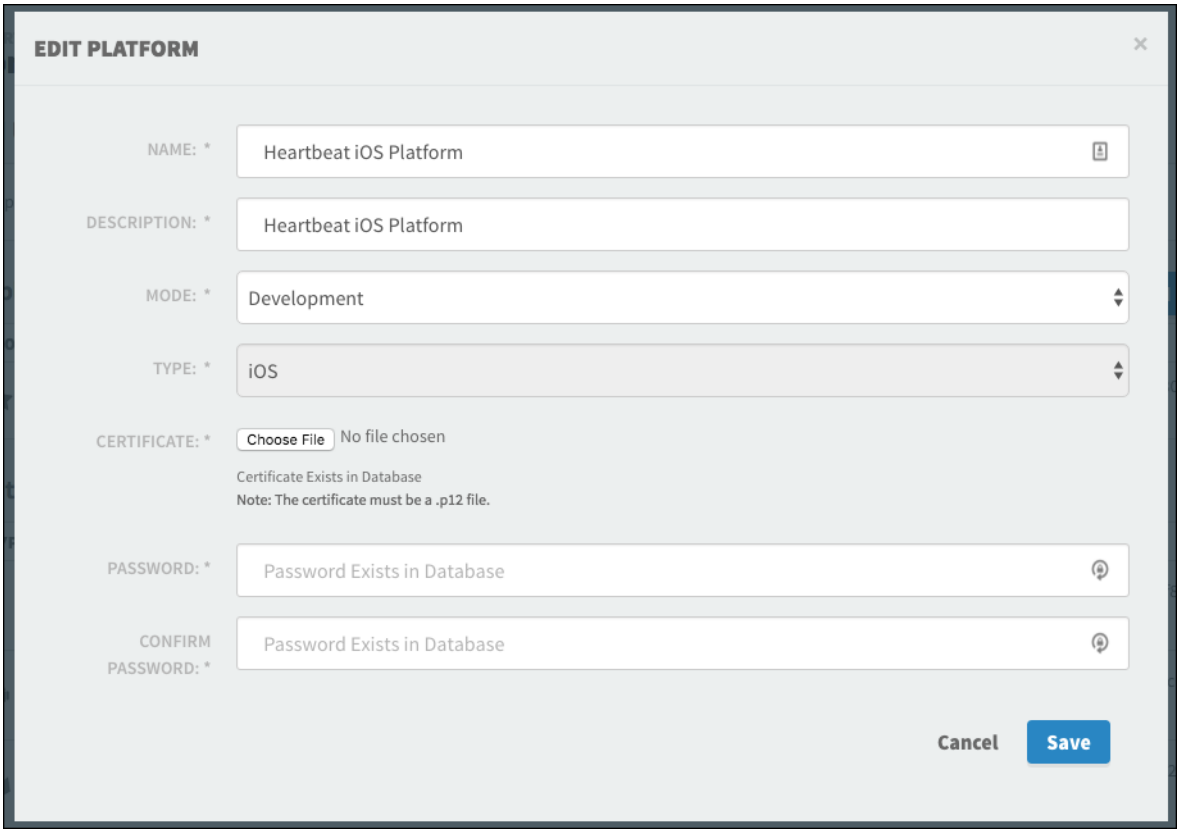
2. Select the **Configuration** pane.



3. Under the **Platforms** section, in the **Heartbeat iOS Platform** row, click the pencil icon to edit the record.



4. Complete the following fields:
   - **MODE**: Select `Development` from the dropdown menu.
   - **CERTIFICATE**: Click **Choose File** and upload the APNS certificate you created.
   - **PASSWORD**: Enter the password you used when creating your APNS certificate.



5. Click **Save**.

Run the App on Your Device

Follow these steps to open the project for the Heartbeat Monitor iOS app in Xcode and run the app on your device:

Download the App Repo

1. Clone the Push iOS Heartbeat Monitor repository:

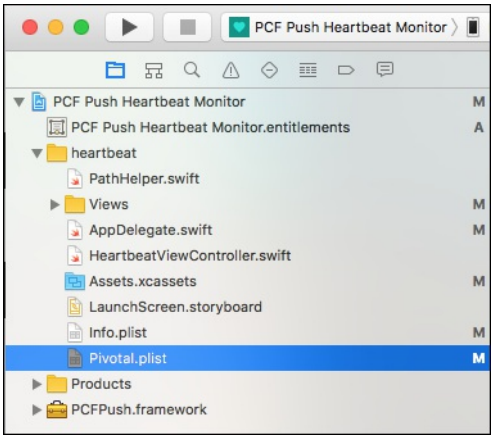
```
$ git clone git@github.com:cfmobile/push-ios-heartbeatmonitor.git
```

2. Run the following command to open the Xcode project:

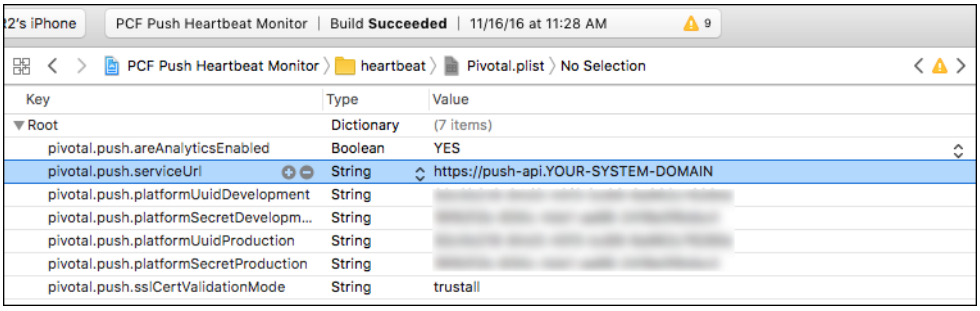
```
$ open PCF\ Push\ Heartbeat\ Monitor.xcodeproj/
```

Configure the App Project

1. In the Project Navigator, select the `Pivotal.plist` file.



2. In the editor, change the value for `pivotal.push.serviceUrl` to the Push Notification API endpoint for your environment: `https://push-api.YOUR-SYSTEM-DOMAIN`.

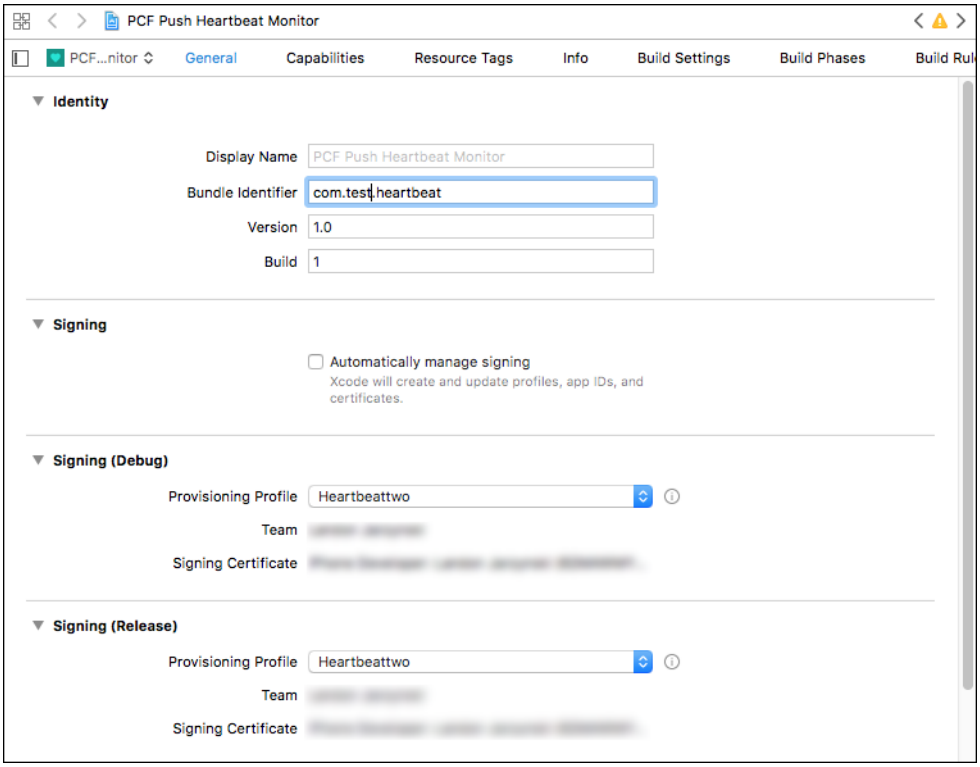


3. Ensure that the values for the following **Root** fields in the editor match the corresponding values in the Push dashboard under the **Heartbeat iOS Platform** record:

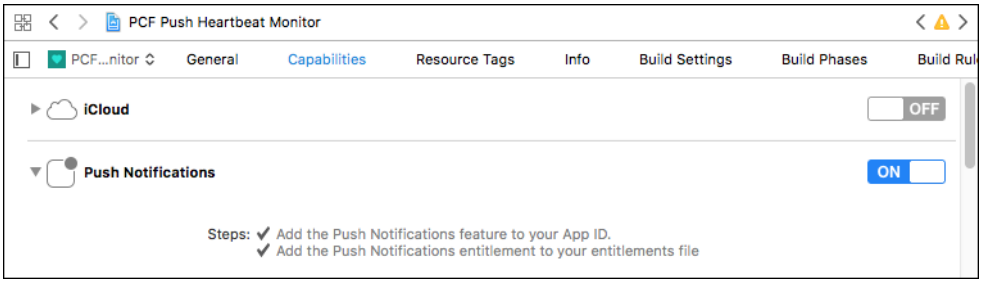
Root Field in Editor	Platform Field in Push Dashboard
`pivotal.push.platformUuidDevelopment`	**Platform UUID**
`pivotal.push.platformSecretDevelopment`	**Platform Secret**

4. Under the **General** tab, set the **Provisioning Profile** dropdowns to the profile you created earlier.

> **Note**: Do not select the checkbox to automatically manage signing.



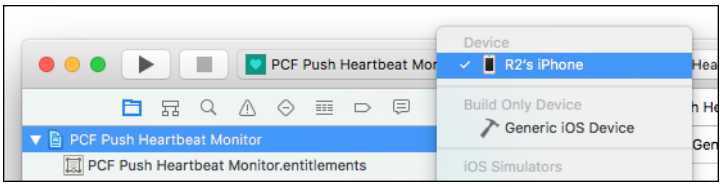
5. Under the **Capabilities** tab, ensure that both **Steps** are enabled for **Push Notifications**.



6. If your PCF deployment does not use an SSL certificate signed by a Certificate Authority (CA), add an exception domain to the `info.plist` file by selecting **App Transport Security Settings** > **Exception Domains** and entering `push-api.YOUR-SYSTEM-DOMAIN`.

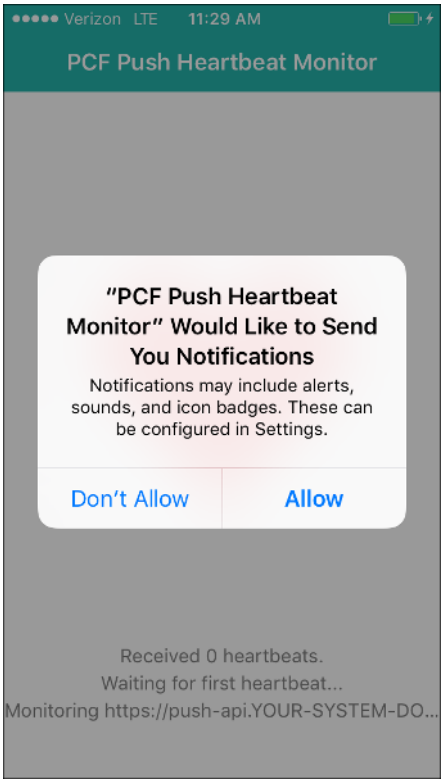
Build and Run the App

1. At the top of the Xcode window, select the device icon and choose your device.



2. Click the play button to build and run the app on your device.

3. Select **Allow** when the app asks if it can send you notifications.



The screen updates with a new heartbeat count every minute as it receives pushes from your environment.

Note: If you send a test push to your device from the Push dashboard, ensure the app is not open on your device. You cannot see the test push while the app is open.

Configuring Heartbeat Monitor for Android

This topic describes how Pivotal Cloud Foundry (PCF) operators can configure the Push Notification Heartbeat Monitor app for Android.

Heartbeat Monitor is a Cloud Foundry app deployed by the PCF Push Notification service to help you ensure the service runs correctly end-to-end. It does this by sending a push, or *heartbeat*, every minute to the devices registered with the app. You can also select the app in the Push dashboard to view its historical data.

Follow the instructions below to configure Heartbeat Monitor and run the companion Android app on your device.

Prerequisites

The procedures in this document require the following:

- You must have access to a PCF environment with the Push Notification Service installed.
- You must have Android Studio 2.2 or later installed on your machine.
- You must have the Google Repository from the Android SDK Manager.
- You must have the Push Android SDK 1.7 or later from Github.
- The devices that you want to send push notifications to must run Android 2.3 (Gingerbread) or later.
- The devices that you want to send push notifications to must have Google Play Services 9.8.0 or later.

Prepare an FCM Project

Follow these steps to prepare an FCM project for your app.

1. Navigate to the Firebase Console and create an account if you do not have one already.
2. Once logged in, **Create** a project for the Heartbeat Monitor.
    a. When prompted, click **Add Firebase to your Android app**.
    b. For **Package name**, enter `io.pivotal.android.push.heartbeatmonitor`.
    c. Ensure the **Debug signing certificate SHA-1** matches the SHA-1 from your debug signing certificate. For instructions on how to get this fingerprint, refer to Authenticating Your Client in the Google APIs for Android documentation.
    d. After you finish creating or importing your project, a `google-services.json` file downloads. Keep track of this file for later use.
3. Click your project.
4. Click the settings icon next to your project name and select **Project Settings**.
5. Select the **Cloud Messaging** tab.
6. Record the **Server key** for later use.

Configure Your Push Dashboard

Follow these steps to navigate to the Push dashboard and then configure the service to talk to your device.

You can navigate to the Push dashboard using either Apps Manager or the Cloud Foundry Command Line Interface (cf CLI). Use the cf CLI instructions if you did not enable the **Push Apps Manager** errand when deploying Elastic Runtime.

Navigate to Push Dashboard using Apps Manager

1. In a browser, navigate to `apps.YOUR-SYSTEM-DOMAIN`.
2. Select the **system** org and the **push-notifications** space.
3. Click the **Services** tab.

4. Select the **PCF Push Notification Service** row and click the **Manage** link.

Navigate to Push Dashboard using cf CLI

1. Open a terminal window and log in:

```
$ cf login -a https://api.YOUR-SYSTEM-DOMAIN -u USERNAME -p PASSWORD
```

2. Target the correct org and space:

```
$ cf target -o system -s push-notifications
```

3. Run the following command:

```
$ cf service push-service-instance
```

4. Copy the URL from the **Dashboard** field and paste it into your browser.

Configure the Push Notification Service

1. Select the **Heartbeat App** from the list of applications.
2. Select the **Configuration** pane.
3. Under the **Platforms** section, in the **Heartbeat Android Platform over FCM** row, click the pencil icon to edit the record.
4. In the **Google Key** field, paste the server key that you recorded earlier.

Run the App on Your Device

Follow these steps to compile and run the app on your Android device.

1. Navigate to the Push Android Heartbeat Monitor repository.
2. Clone the repository to your workspace.
3. Checkout the `release-v1.7.0` branch, or the branch of a later version.
4. Copy the `google-services.json` file from earlier into the `app` directory of the Heartbeat Monitor project.
5. Open a project in Android Studio using the repo you cloned.
6. Update `pivotal.properties` file located in `app/src/main/res/raw`:
   - `pivotal.push.platformUuid`: This value must match the platform **UUID** of the **Android FCM Heartbeat Platform** in the Push dashboard.
   - `pivotal.push.platformSecret`: This value must match the platform **SECRET** of the **Android Heartbeat FCM Platform** in the Push dashboard.
   - `pivotal.push.serviceUrl`: Enter the server address to your push backend API in the form of `https://push-api.YOUR-SYSTEM-DOMAIN`.
7. Compile and deploy the application to your Android device.

> **Note**: To verify that your device registered, see the **Devices** tab in the Push dashboard. The device **Type** field displays a Firebase logo.

8. Open the app on your device and select **Allow** when the app asks if it can send you notifications. The screen updates with a new heartbeat count every minute as it receives pushes from your environment.

> **Note**: If you send a test push to your device from the Push dashboard, ensure the app is not open on your device. You cannot see the test push while the app is open.

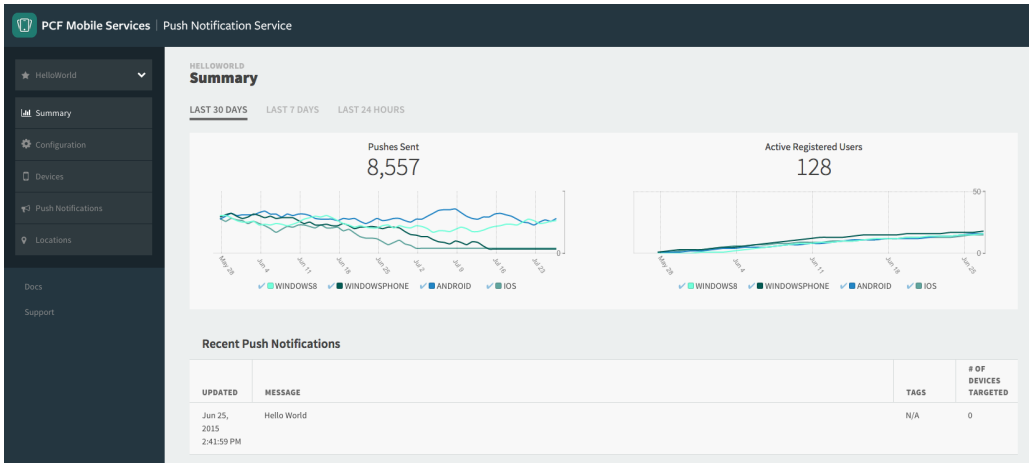
Using the Dashboard V1.7.0

Dashboard guide for other versions

Version List

Applications

An application in the Push Dashboard represents a mobile application from the perspective of the application author, including all supported platforms. Applications are listed in the dropdown at the top of the sidebar.

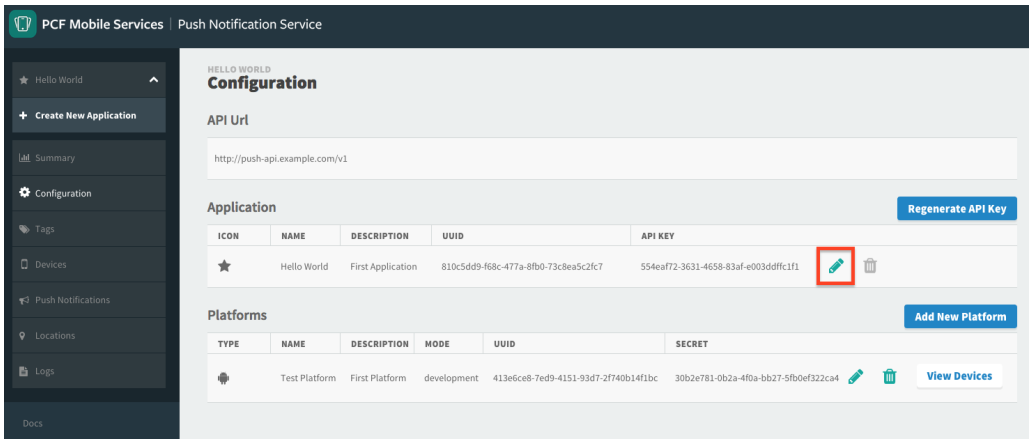


Adding an application

Click on `Create New Application` in left sidebar. Fill in the form and click `Save` to create the application or optionally click 'Add Platform' to add a platform for the application.

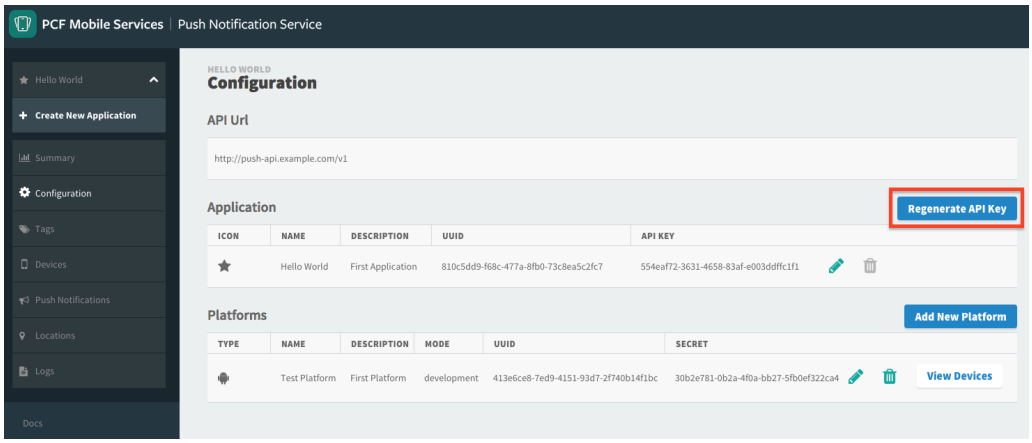
Editing an application

Click on the `Configuration` link in the sidebar menu to bring up the information about the application. Click on the pencil icon under the `Actions` column to edit the application. Edit the fields and click `Save` to update the application. The **UUID** is immutable.



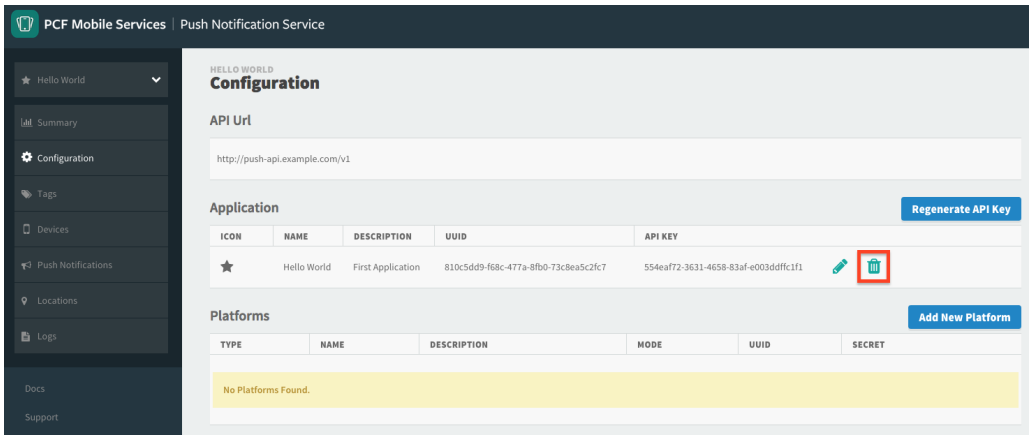
Regenerating an API key

Click on the `Regenerate API Key` button. A new API key will be generated. NOTE: you will no longer be able to send pushes using the previous API key.



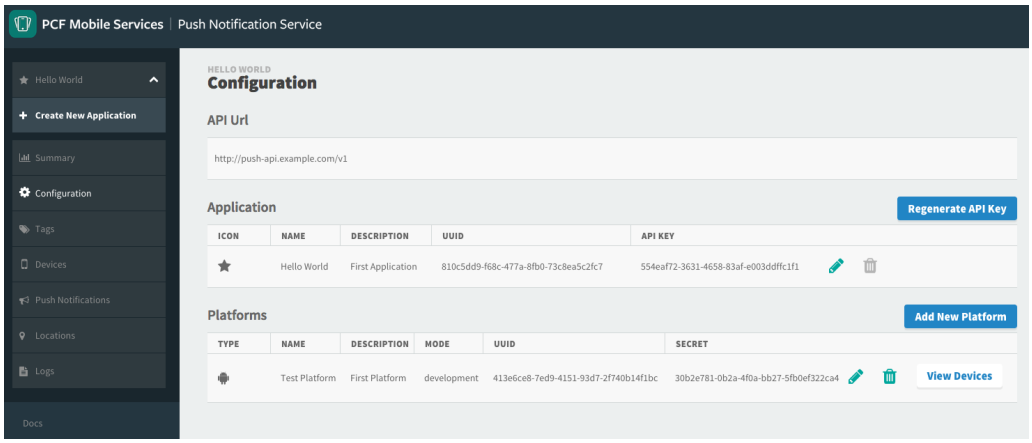
Deleting an application

To delete an application, click on the `Configuration` link in the sidebar menu to bring up information about the application. Click on the delete icon under the `Actions` column to delete the application. NOTE: This icon will be disabled if the application has one or more platforms.



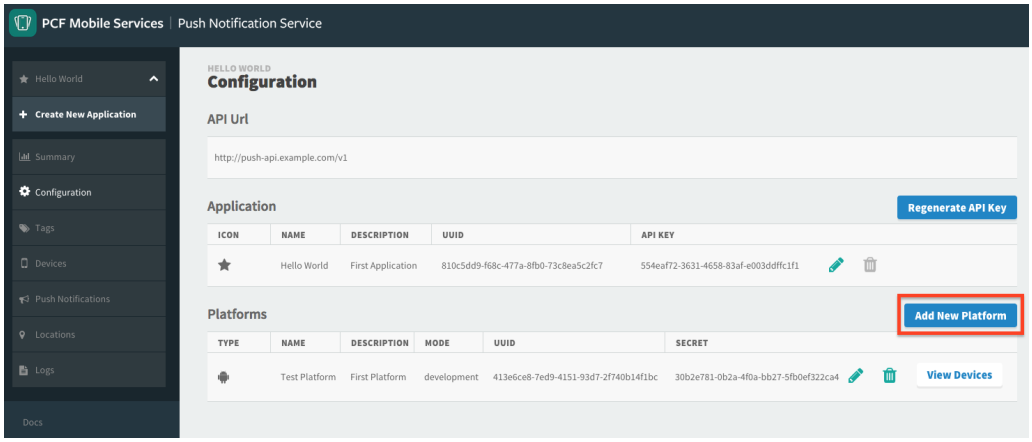
Platforms

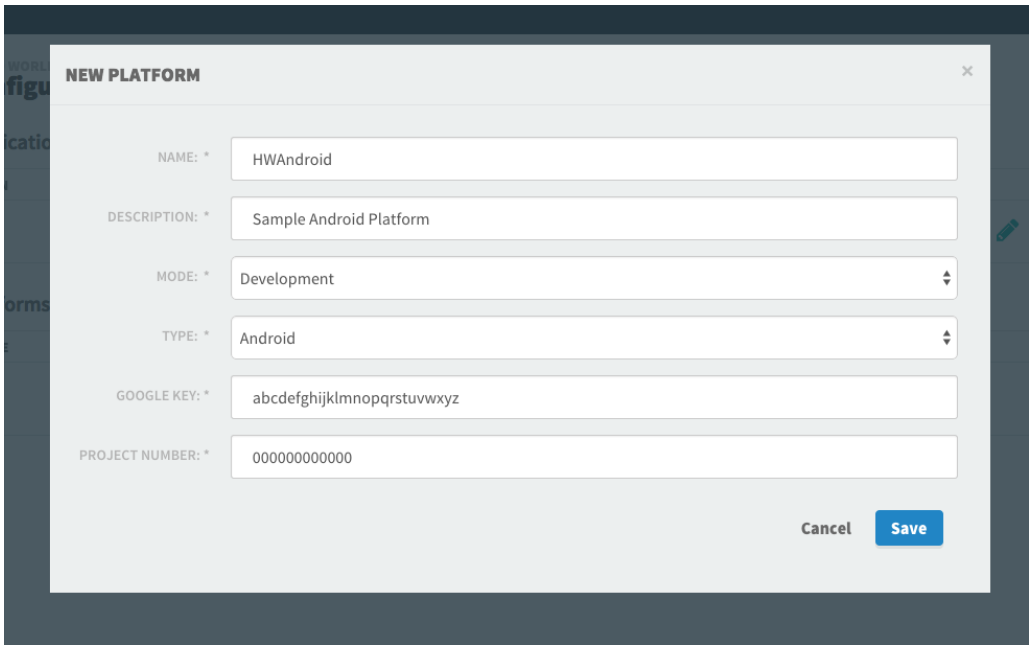
A platform configures platform specific attributes to send push messages. For example, this would include a certificate necessary to send messages to Apple's APNS, or a token necessary to send messages to Google's GCN. A platform has many devices.



Adding a platform

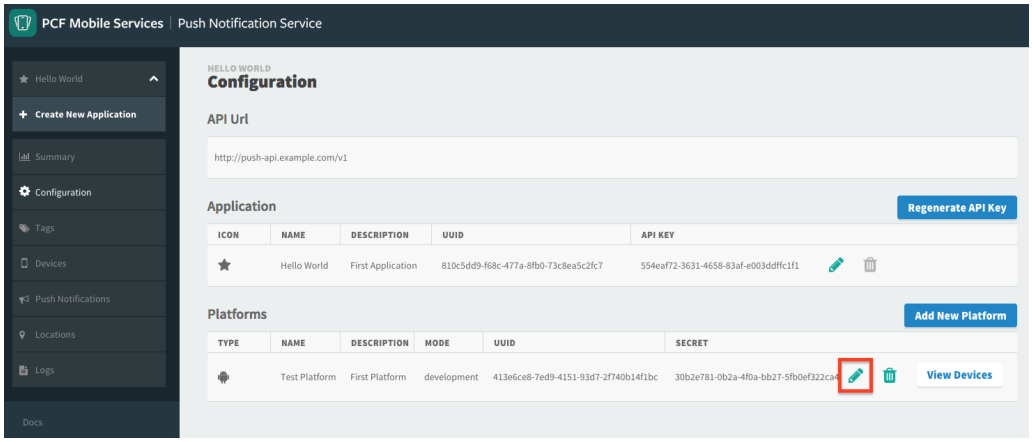
On the Configuration page, click on the `Add New Platform` button . Fill in the form and click `Save` to create the platform.

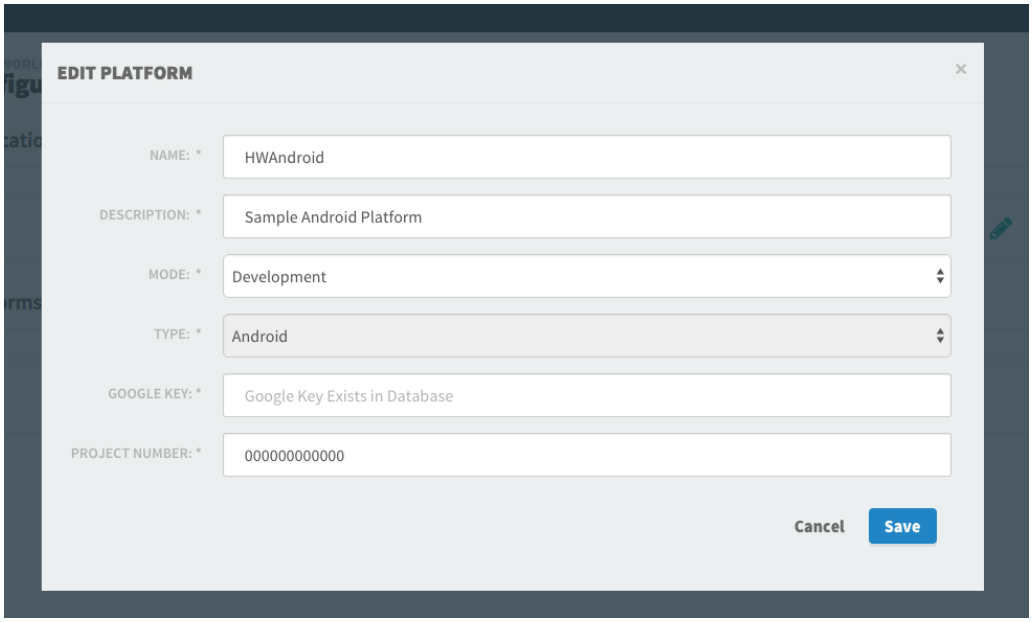




Editing a platform

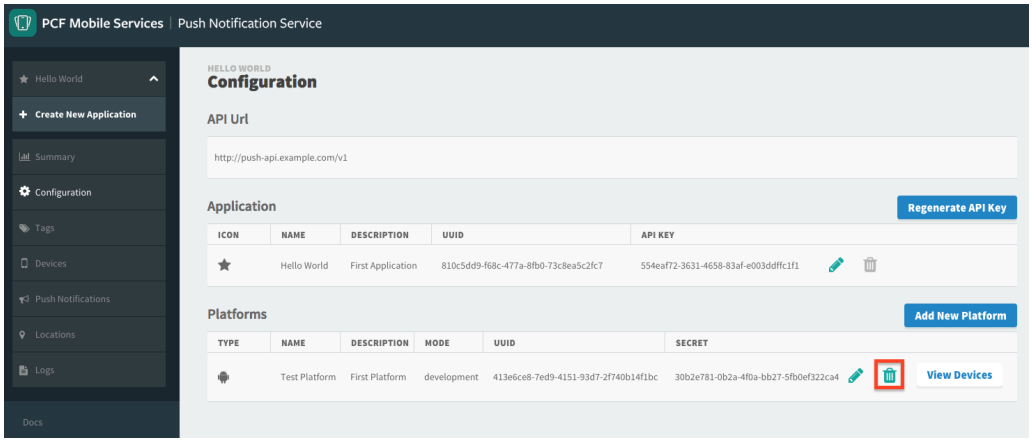
On the Configuration page, click on the pencil icon link next to the platform you want to edit. Edit the fields and click `Save` to update the platform. The **Type** field cannot be changed once set.

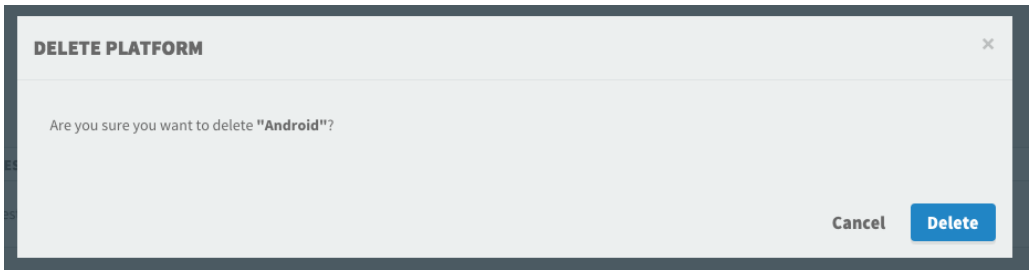




Deleting a platform

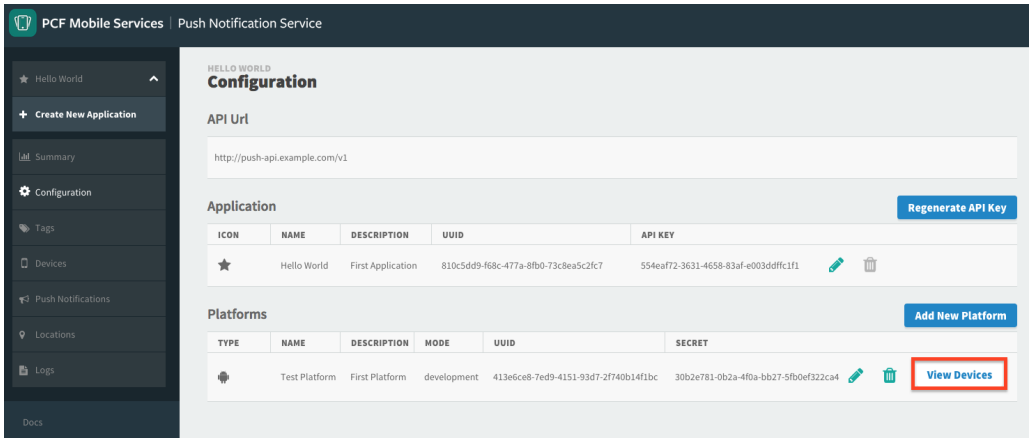
On the Configuration page, click on the trash icon link next to the platform you want to delete. NOTE: You cannot delete a platform that has devices. In order to remove devices you must unregister from the device.





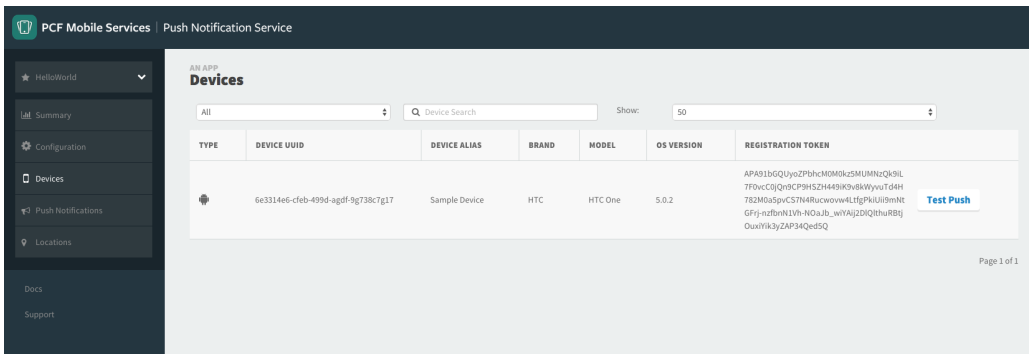
Devices

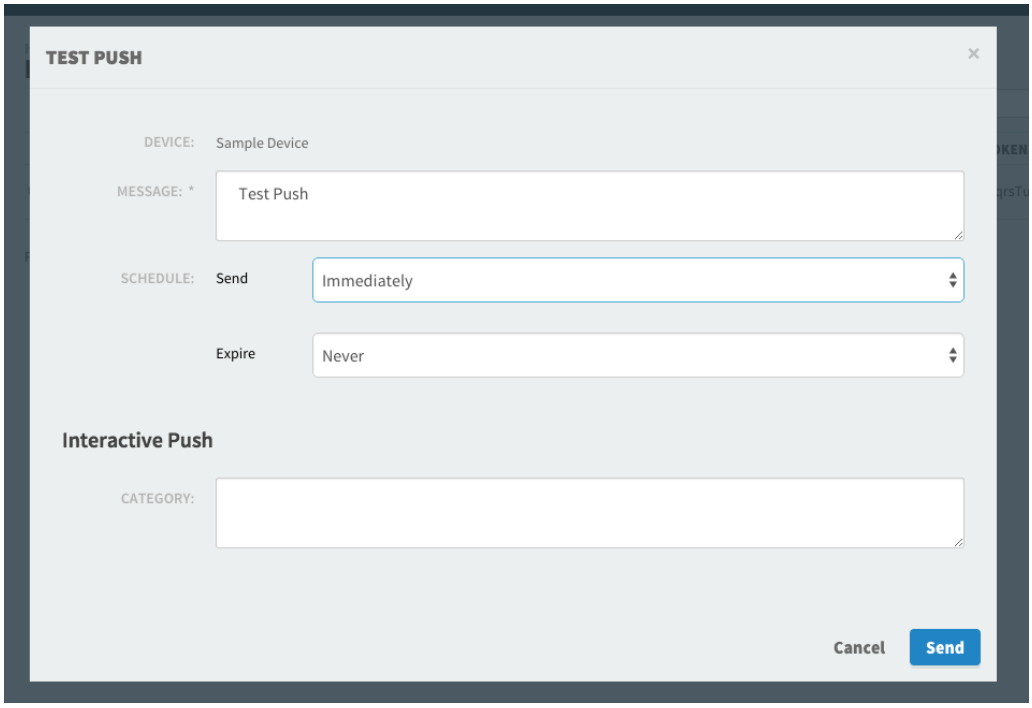
A device is given a unique identifier which represents a user opting in to receive push notifications. This identifier is not necessarily unique to a device since it might change if the user reinstalls the mobile application, or unsubscribes and resubscribes.



Send a test push notification to a device

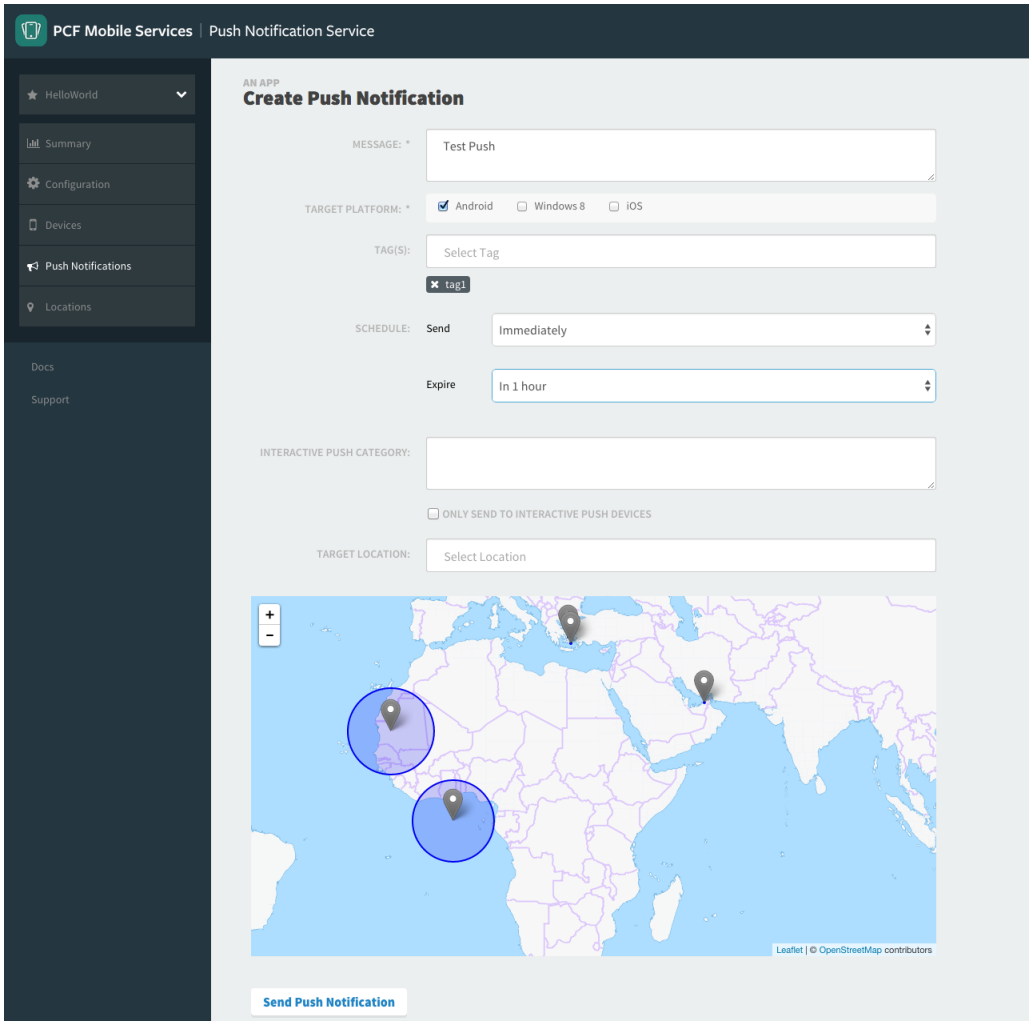
Click on 'Devices' in the sidebar menu. Click on the Test Push button next to the device. Fill out the push form. See “Sending a push message” for details on the form fields.





Sending a Push Message

Click on `Push Notifications` in the sidebar menu, and click the button `Create Push Notification`.



On the Create Push Message page, fill in the form and click `Send Push Notification`.

- Push Message: The alert body for the message
- Target Platform: Send the push to all devices belonging to a specific platform (eg. iOS, Android, etc)
- Tag(s): Send the push to all devices subscribed to one or more tags
- Schedule
  - Send: Schedule the push to be sent immediately or at a later time. Defaults to “Immediately”
  - Expire: Prevent delivery of the message after a specified time, if delivery is delayed for some reason (eg. no connectivity on user device). Default is “Never”
- Interactive Push Category
  - **iOS Only** - Set the category for a push (required for interactive pushes)
- ONLY SEND TO INTERACTIVE PUSH DEVICES: Filter targetted devices for **only** devices that support interactive push
- Target Location: Pick a location to setup a geofence
- Trigger Type: If a location is selected, trigger type determines when a geofence is activated

A Note About Targeting

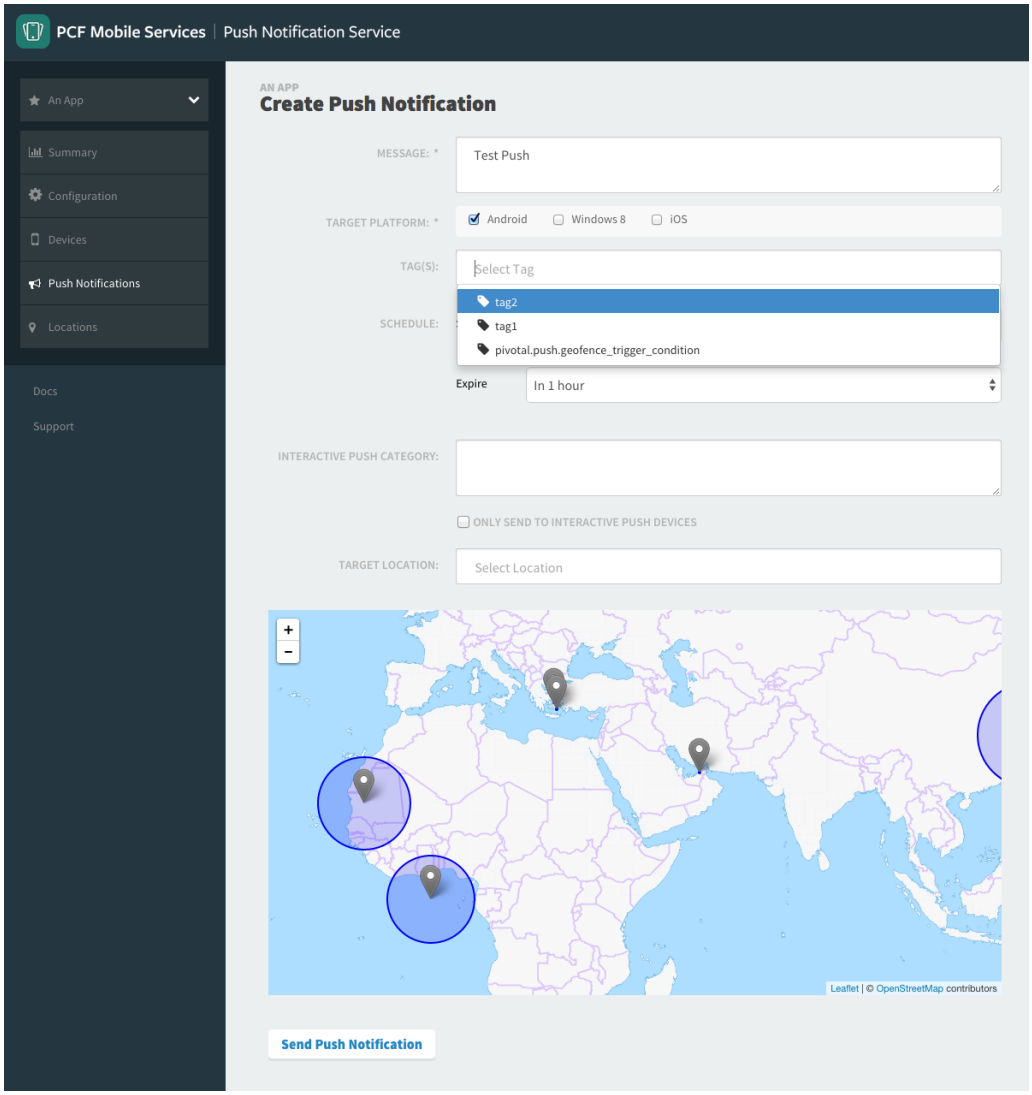
`Target Platform` will target all devices of the selected platform. Adding tags to the `Tag(s)` field will refine the target list down, adding only those devices subscribed to one of the listed tags.

A Note About Sending Push With Invalid Certificate

iOS Only Sending a push to a device using an **invalid .p12 certificate** set up in the device's corresponding platform results in the device getting removed from the platform.

Tags

A tag allows push notifications to be sent to all devices that have explicitly subscribed to it as opposed to all users that have the application installed. This allows an application to send targeted push notifications to a subset of devices. Devices can subscribe to tags via the registrations api. Available tags are listed in the targetting section of the Create Notification form.

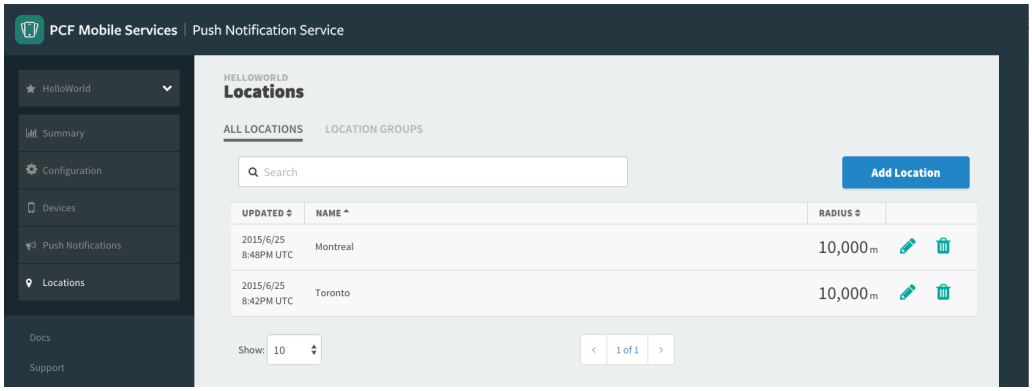


Locations

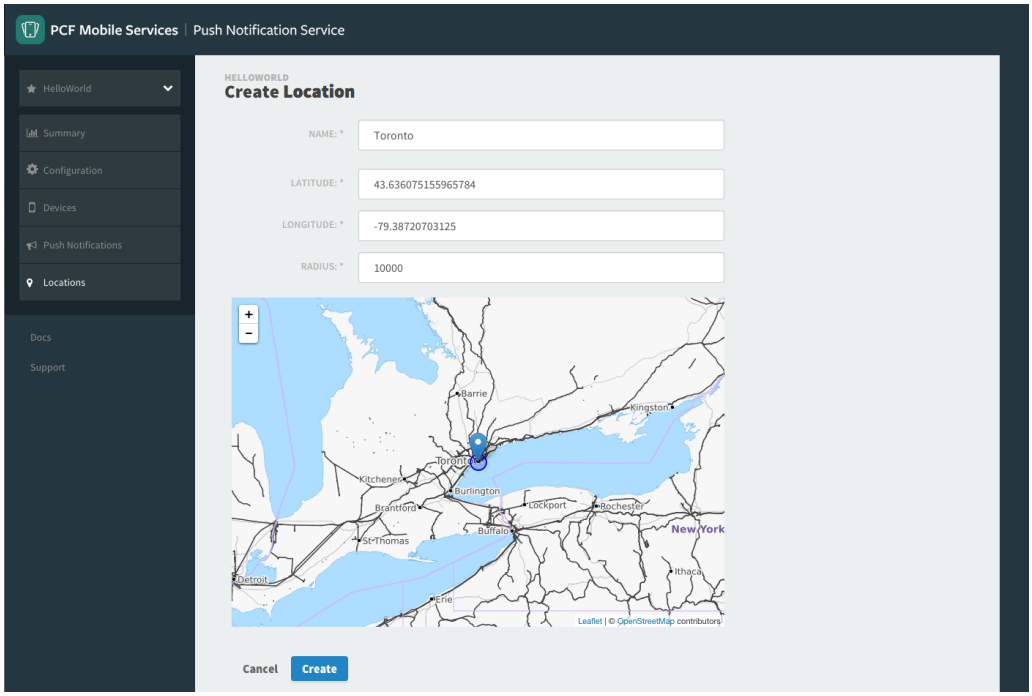
Locations allow you to send push notifications to a subset of users who are within (or enter) the radius of a specified area.

Adding a Location Group

Click `Locations` on the left sidebar and then click the `Add Location` button.

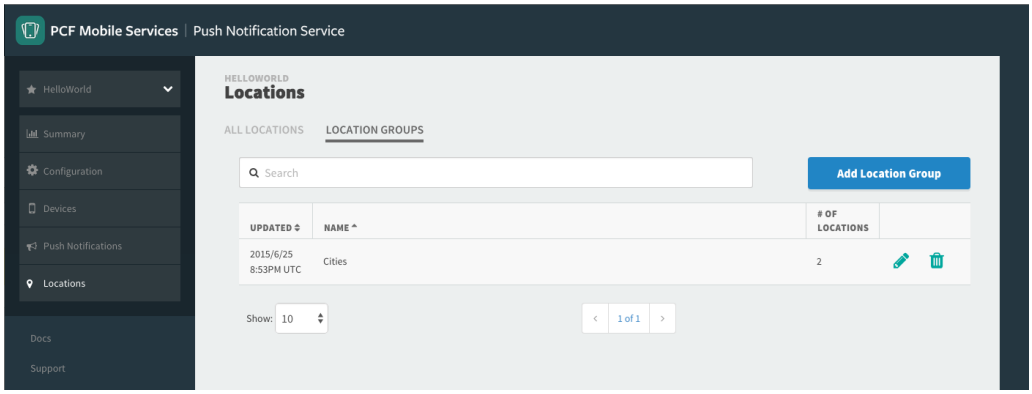


Fill in the Name of the location. You may type in a Latitude and Longitude pair, or simply click on the map. Select a radius that suits the location. Once all the details are set, click the `Create` button.

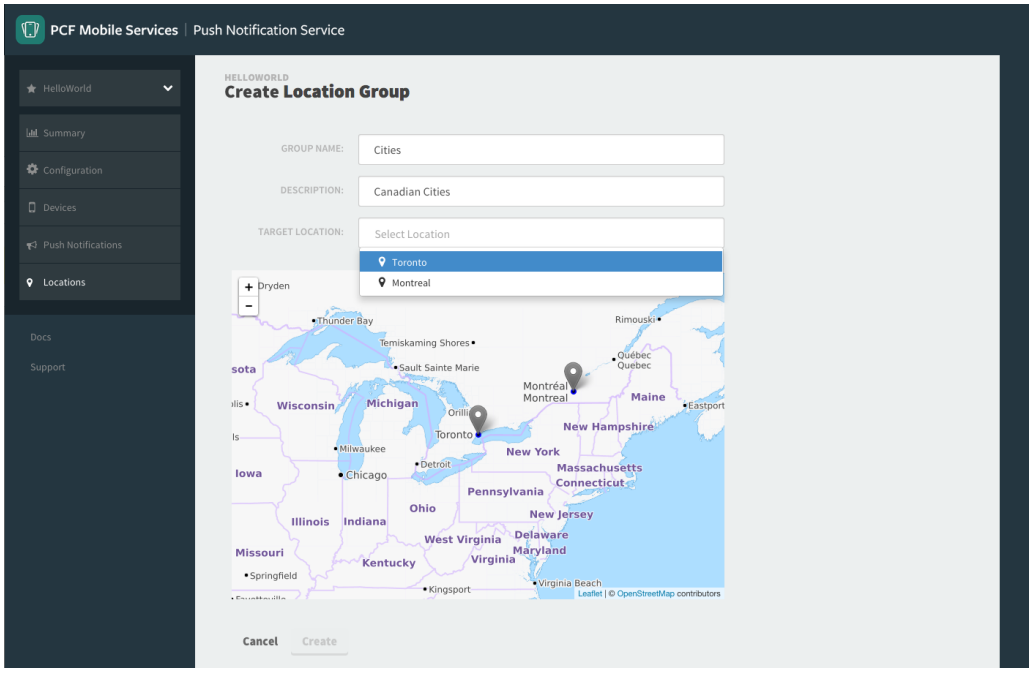


Adding a Location Group

Click on the `Location Group` tab, and then on the `Add Location Group` button.

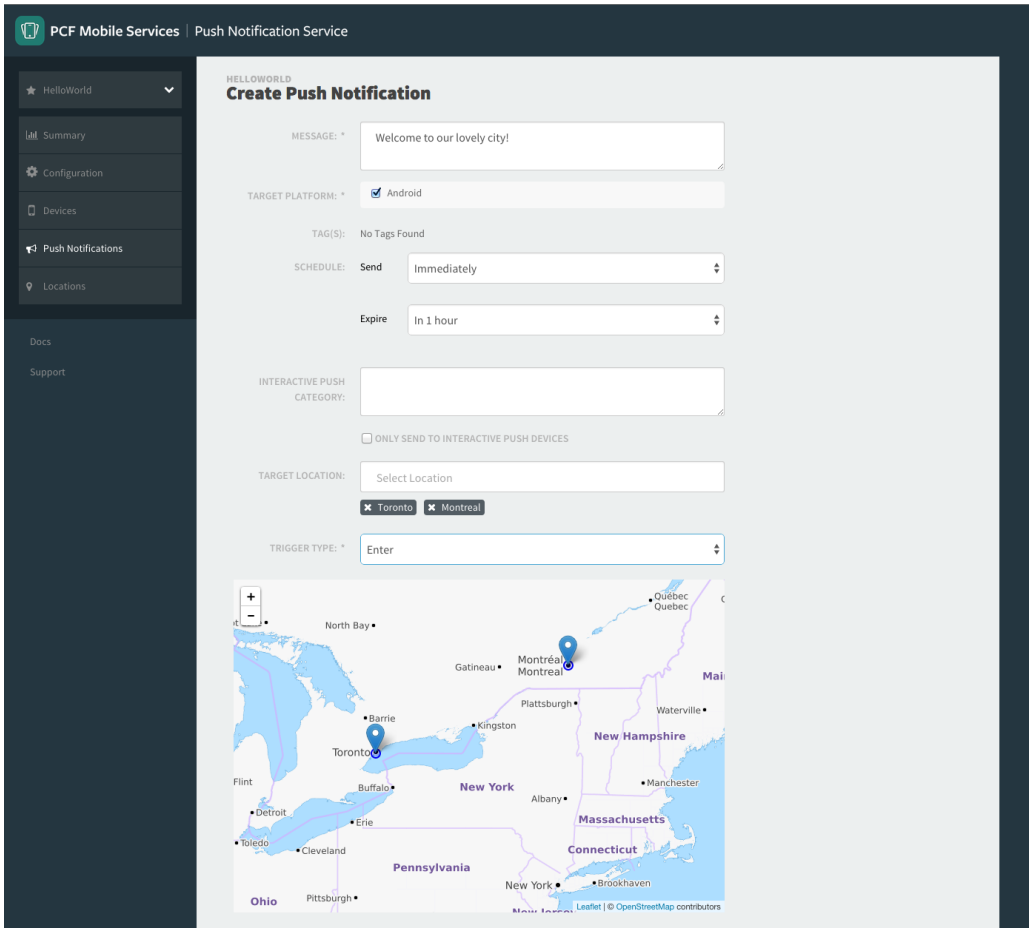


Fill in the Name and Description of the Location Group. In the Target Location field, select a location from the drop-down or click on one of the markers on the map. Once all the details are set, click the `Create` button.



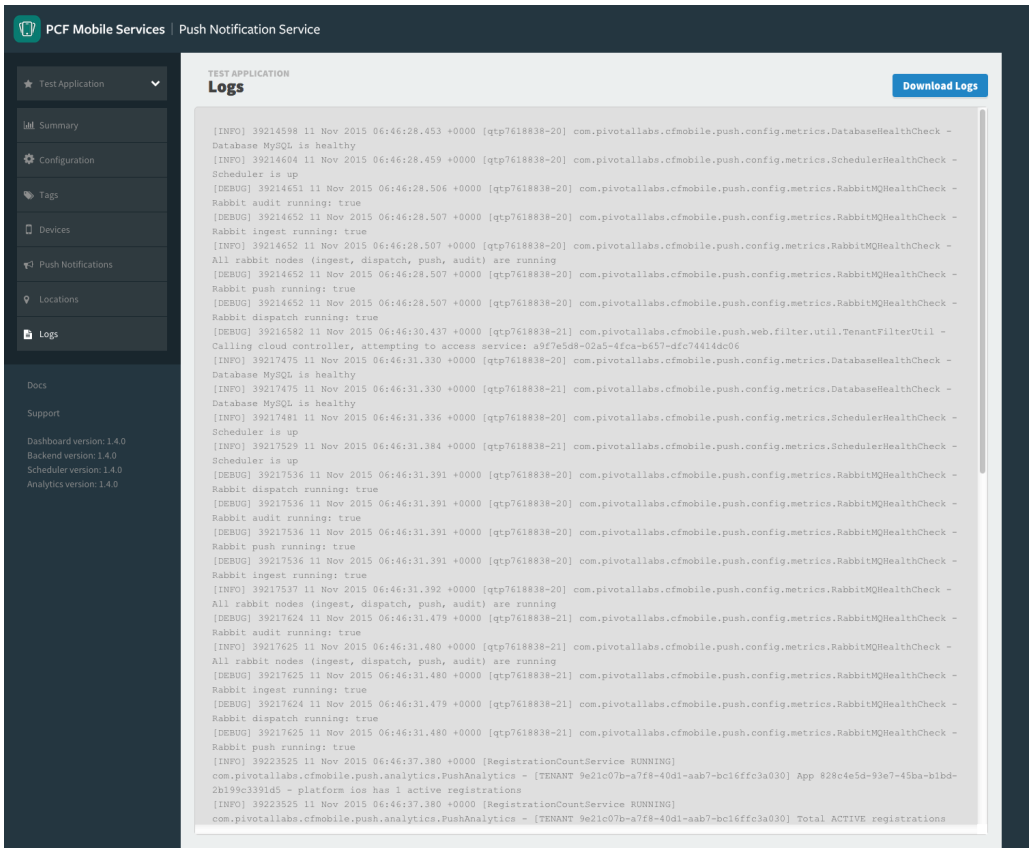
Geofence Push Notifications

Fill in the details of the Push Notification, such as Message, Platform, and Schedule. Select from the `Target Location` drop-down either a Location or a Location Group. Trigger Type field will appear upon the addition of Location/Location Group. Select either Enter or Exit, depending on how you want the Geofence to activate. Once all the details are set, click the `Send Push Notification` button.



Logs

The Logs page displays any logged events that occur while the Logs page is open. Clicking the “Download Logs” button will copy the logs displayed into a text file onto your local machine.



Push Notifications ASG Installation

Application Security Groups

To allow this service to have network access you will need to createApplication Security Groups (ASGs).

> **Note**: Without Application Security Groups the service will not be usable.

Pre-Installation Requirements

Push Notification Service depends on MySQL, RabbitMQ, and Redis. Please refer to their corresponding ASG documentation to ensure their required ASGs are in place.

Push Service Network Connections

This service is deployed as a suite of applications to the `push-notifications` space in the `system` org, and requires the following outbound network connections:

Destination	Ports	Protocol	Reason
17.0.0.0/8	5223, 2195, 2196	tcp	This is Apple's IP address which is used to access APNS
`GOOGLE_IP_RANGE`	5228, 5229, 5230, 443	tcp	This is Google's url for sending GCM Messages
`LOAD_BALANCER_IP`	80, 443	tcp	This service will access the load balancer and CAPI
`ASSIGNED_NETWORK`	3306, 5672, 6379	tcp	This service requires access to p-mysql, p-rabbitmq, p-redis, or external services. `ASSIGNED_NETWORK` is the CIDR of the network assigned to this service.

APNS

Apple exposes the entire 17.0.0.0/8 block and uses ports 2195, 2196, and 5223. Create a file apns.json as follows:

```
[
  {
    "protocol": "tcp",
    "destination": "17.0.0.0/8",
    "ports": "2195, 2196, 5223"
  }
]
```

Create a security group called apns: `cf create-security-group apns apns.json`

GCM

Google unfortunately has a very large range of IP addresses that it can use for GCM.

> **Note**: Google's ASN is 15169. You can search for "ASN 15169" to find the most up to date list of their IP addresses.

Create a file gcm.json as follows:

```
[
  {
    "protocol": "tcp",
    "destination": "8.8.4.0/24",
    "ports": "5228,5229,5230,443"
  }, {
    "protocol": "tcp",
    "destination": "8.8.8.0/24",
    "ports": "5228,5229,5230,443"
  },

  ...rest of Google IPs elided...
]
```

Create a security group called gcm: `cf create-security-group gcm gcm.json`

Load Balancer

If the built-in HAProxy is being used as the load balancer. The IP addresses can be found in Pivotal Elastic Runtime Tile → Settings Tab → Networking under HAProxy IPs, (e.g., 10.68.196.250). Create a file load-balancer-https.json as follows:

```
[
  {
    "protocol": "tcp",
    "destination": "10.68.196.250",
    "ports": "80,443"
  }
]
```

Create a security group called load-balancer-https: `cf create-security-group load-balancer-https load-balancer-https.json`

Assigned Network

Note: If you decide to use external services, the IP addresses, ports, and protocols will be dependent on what you use.

Log into Ops Manager and click on the Pivotal Elastic Runtime Tile → Settings Tab → AZ and Network Assignments. Note the name of the network selected in the drop-down (e.g., “first-network”). Then click on the Ops Manager Director tile → Settings Tab → Create Networks → “first-network” and note the CIDR in the subnets section (e.g., 10.68.0.0/20). This should allow the space to access `p-mysql` , `p-rabbitmq` , and `p-redis` Then create a file assigned-network.json as follows:

```
[
  {
    "protocol": "tcp",
    "destination": "10.68.0.0/20",
    "ports": "3306,5672,6379"
  }
]
```

Create a security group called assigned-network: `cf create-security-group assigned-network assigned-network.json`

Pre-installation ASG binding

Log in as an administrator and create the above ASGs. Afterwards, create the space `push-notifications` in the `system` org and bind each of them to the it :

```
cf target -o system
cf create-space push-notifications
cf bind-security-group apns system push-notifications
cf bind-security-group gcm system push-notifications
cf bind-security-group load-balancer-https system push-notifications
cf bind-security-group assigned-network system push-notifications
```

Network Setup Guide

APNS / iOS Push

Server and Device Settings

The push-api backend needs to have persistent sockets open to the Apple APNs servers.

Information from the Apple Support site

To use Apple Push Notification service (APNs) you need a direct and persistent connection to Apple's servers. Your device connects to APNs using cellular data if it's available. If there's no viable cellular connection the device switches to Wi-Fi.

If you use Wi-Fi behind a firewall or a private Access Point Name (APN) for cellular data then you'll need a direct unproxied connection to the APNs servers on these ports:

- TCP port 5223: For communicating with Apple Push Notification services (APNs).
- TCP port 2195: For sending notifications to APNs.
- TCP port 2196: For the APNs feedback service.
- TCP port 443: For a fallback on Wi-Fi only when devices can't reach APNs on port 5223.

The APNs servers use load balancing so your devices won't always connect to the same public IP address for notifications. It's best to allow access to these ports on the entire 17.0.0.0/8 address block which is assigned to Apple.

GCM / Android Push

Server and Device Settings

The push-api backend needs to send requests to "https://gcm-http.googleapis.com/gcm/send" (port 443).

Devices will need direct unproxied connections to Google servers on port 5228. Android 4.3 and up have fallback capabilities to use port 443.

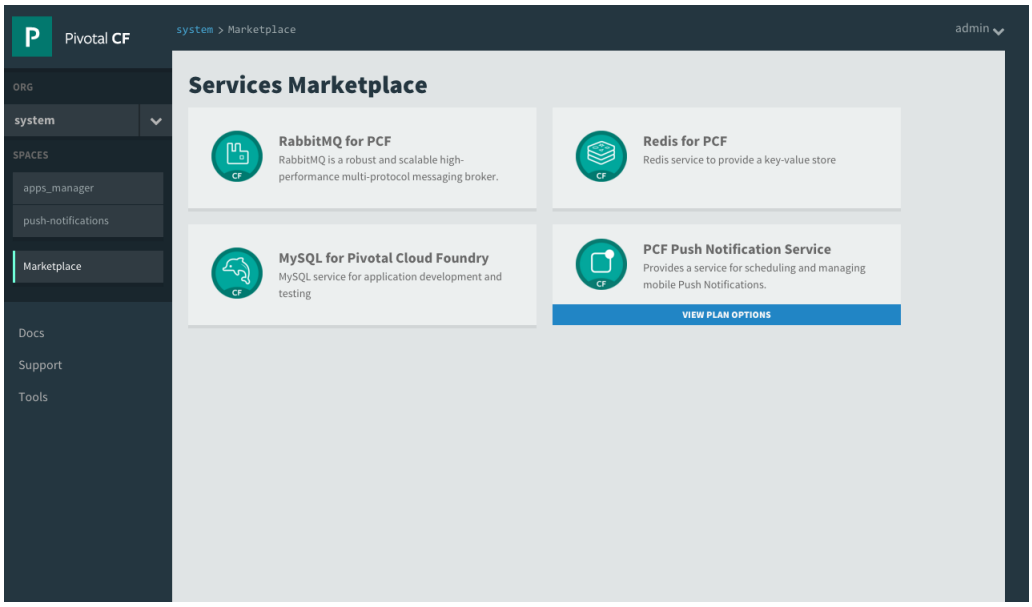
Development Guide

- First Push Walkthrough
- First Geofence Walkthrough
- iOS
  - Sample App
- Android
  - Sample App
- Windows Phone 8.1
  - Sample App

First Push Walkthrough

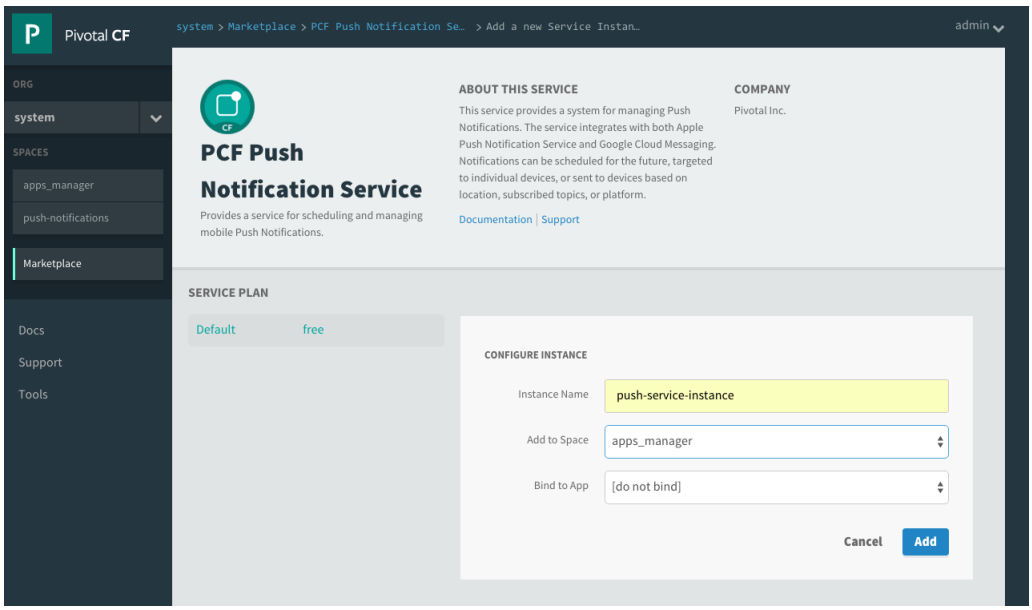
Step 1

In the Cloud Foundry App Manager, click on the “Marketplace” link. Select “Push Notification Service” from the list of available services.



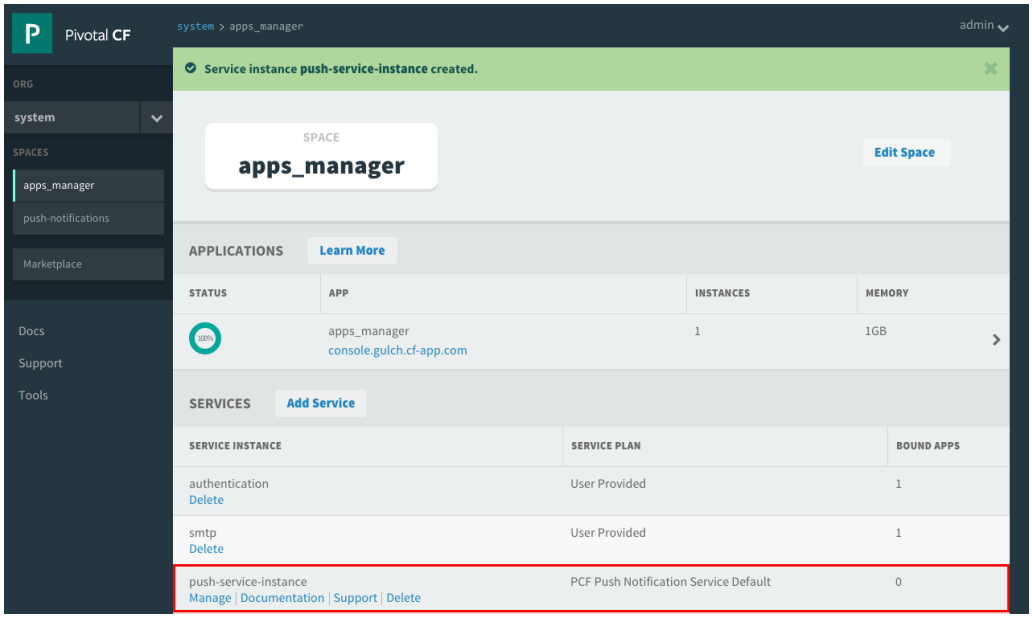
Step 2

Select the “Default” service plan. Give the service instance a name and make sure to select the correct Space for the service to be created in before clicking the “Add” button.

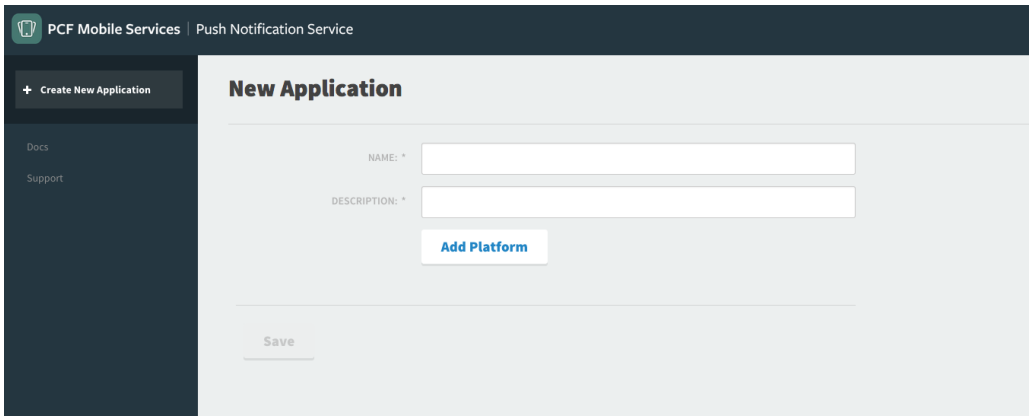


Step 3

You can now click on the “Manage” link for the Push Notifications Service instance you’ve created. This will open the Push Dashboard.

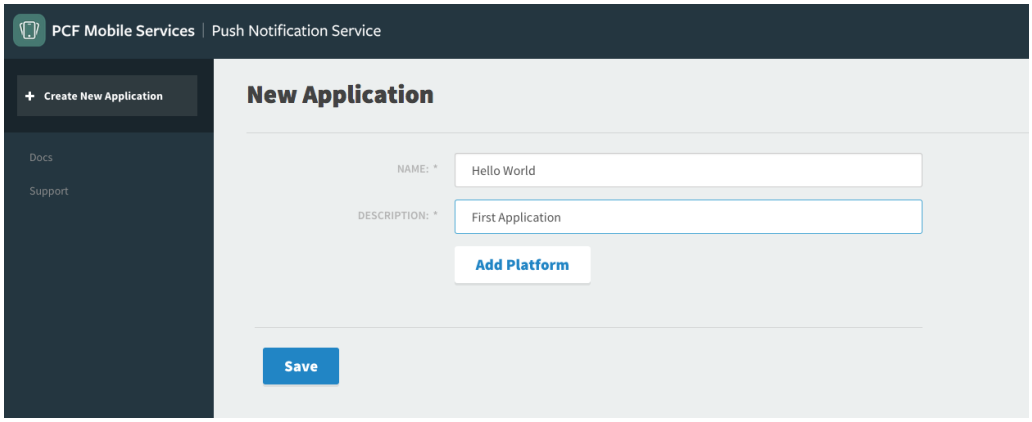


Add an application by filling in the form that appears when first navigating to the dashboard. If applications already exist, you can access the add application screen by clicking on “Create New Application” on the left hand sidebar dropdown.



Step 4

Fill in fields on the new application screen. There are two fields: name and description. These fields are purely for keeping track of which application is which.



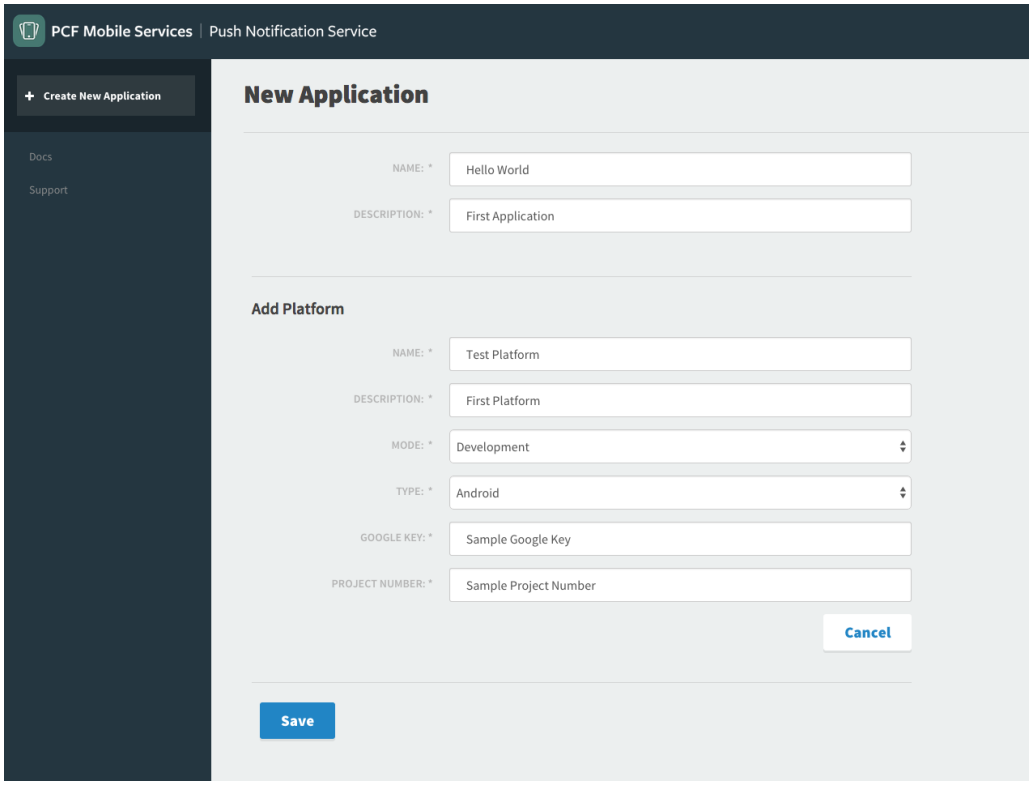
Step 5

Create a new platform by clicking on the ‘Add Platform’ button and filling out the proper fields depending on the platform type.

For Android platforms you will need to provide **Project Number** and **Google Key** values. The **Project Number** is the numeric value found at the top middle of a project on the Google Developers Console. Do not use the 'Project ID'. The **Google Key** is a Server API key, created on the “Credentials”

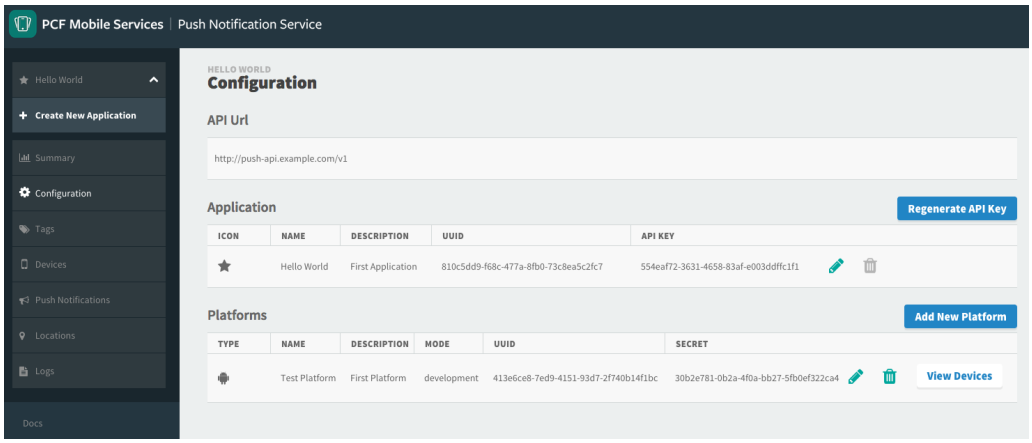
screen of the Google Developers Console.

For iOS platforms you will need to create a **APNS Development Certificate** and **APNS Production Certificate** using the Apple Developer Website. These files, along with their associated private keys, need to be exported from your **Keychain Access** program into a password protected **P12** file. You will upload this P12 file and provide its password when you create your platform on the PCF Push Notification Service dashboard.



Step 6

After saving, click on 'Configuration' on the left sidebar, this is where the UUID and secret will be found. These values are used to register devices and eventually send pushes.

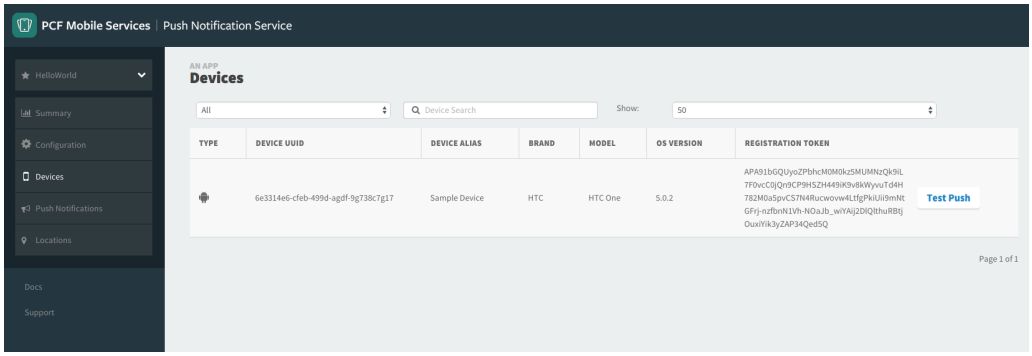


Step 7

Now you will have to integrate the sdk with your app. See the getting started section of the SDK documentation.

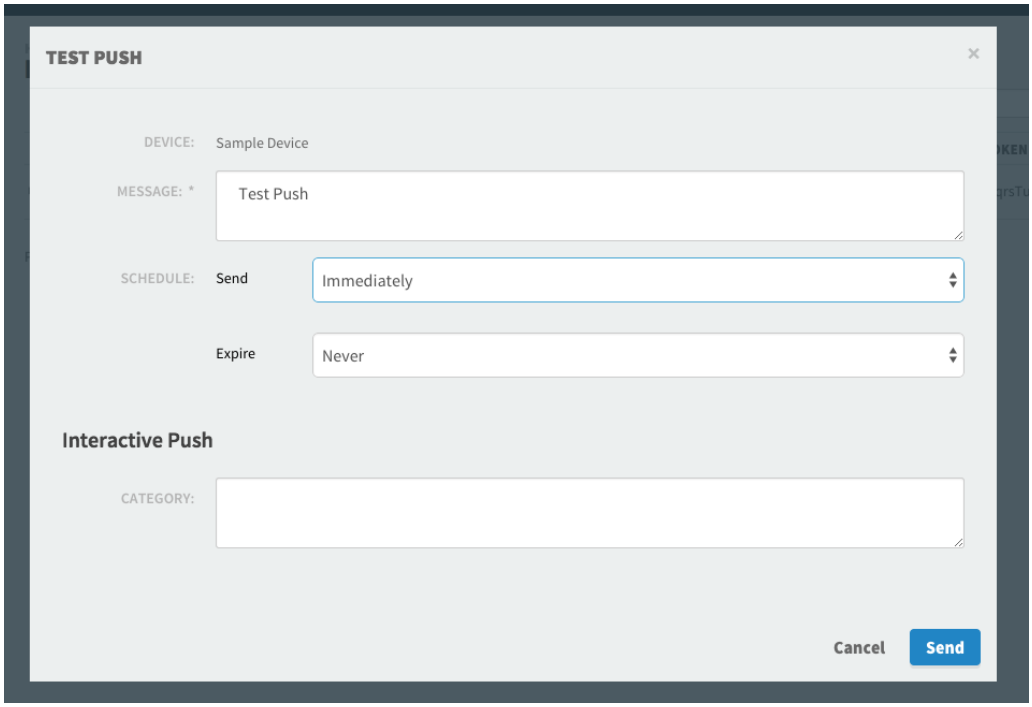
Step 8

Click on the 'Devices' link on the left sidebar to see registered devices, and click on the 'Test Push' button for the device you wish to send a push.



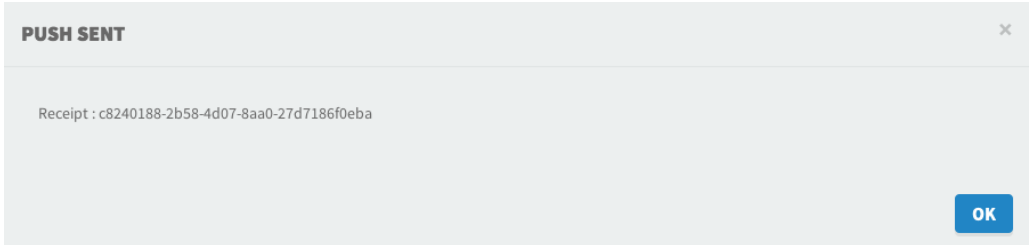
Step 9

Fill in a message and press send to send a test message.



Step 10

If the server accepts this push for delivery, a receipt will be shown on screen. This does not guarantee delivery to the device (device could be off, notifications could be disabled, etc).



Geofence Walkthrough

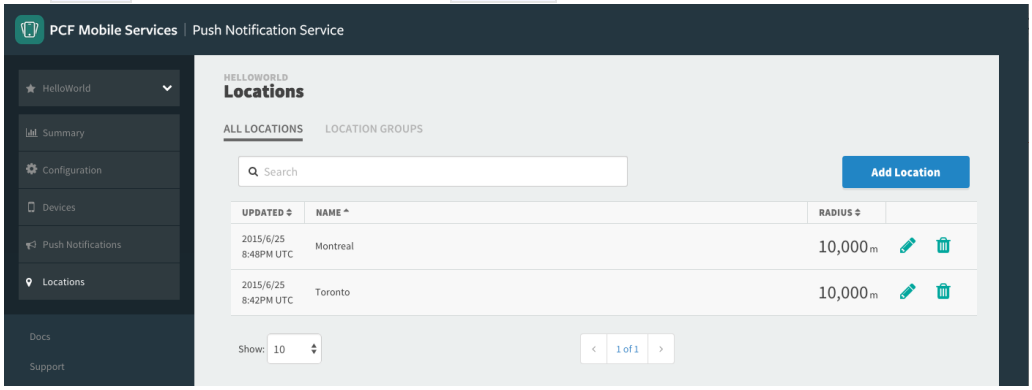
Modern mobile devices can track numerous geofences, each of which are defined by a lat/long pair and a radius. Whenever the device enters or exits the boundries of a geofence, a notification can be triggered. The triggering of a notification is not dependant on the device having an Internet/Data connection.

Step 1

Complete the steps from the First Push Walkthrough guide. (Setup an application, platform(s) and devices).

Step 2

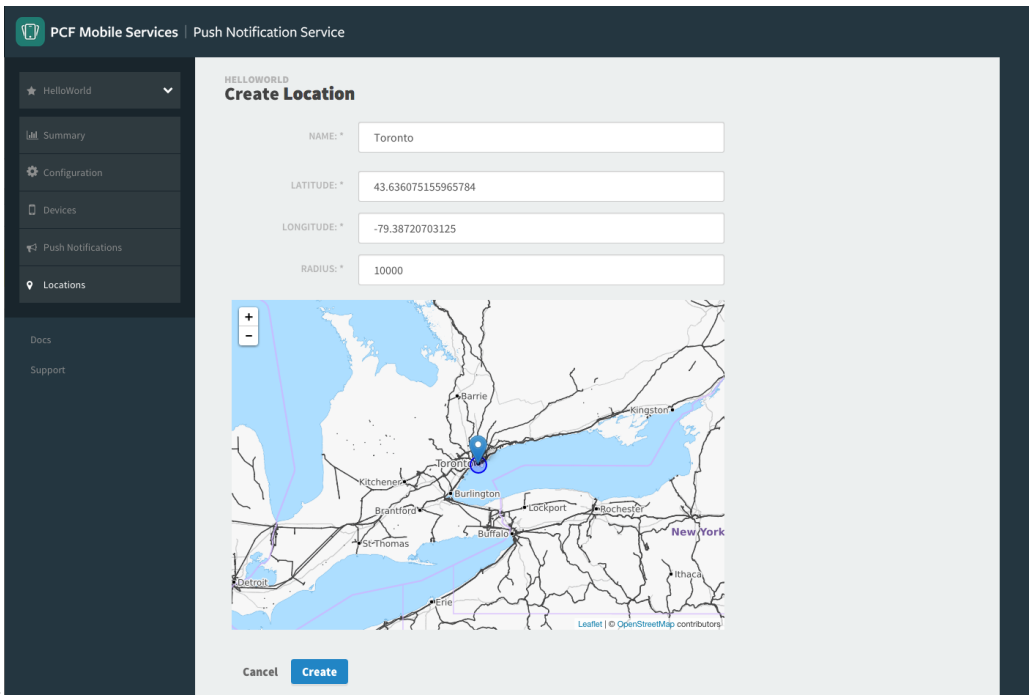
Click `Locations` on the left sidebar and then click the `Add Location` button.



Step 3

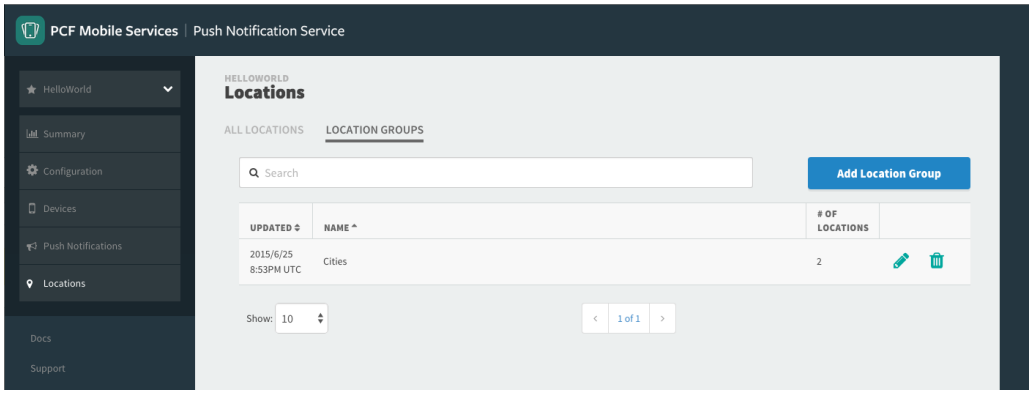
Fill in the Name of the location. You may type in a Latitude and Longitude pair, or simply click on the map. Select a radius that suits the location. Once all the details are set, click the `Create` button.

Create a few more locations.



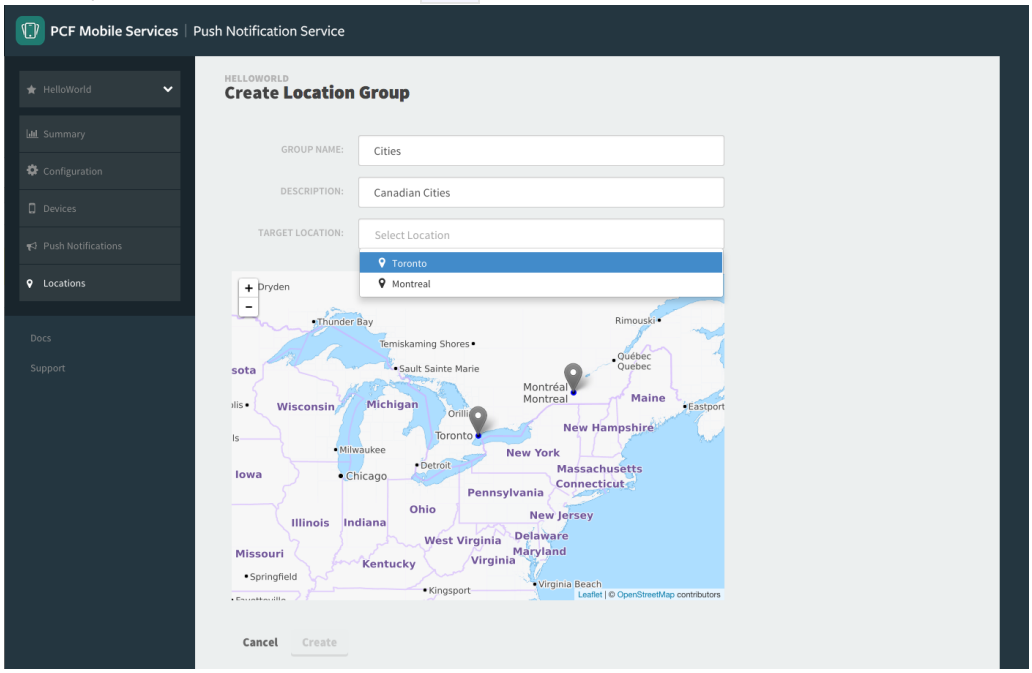
Step 4

Click on the `Location Group` tab, and then on the `Add Location Group` button.



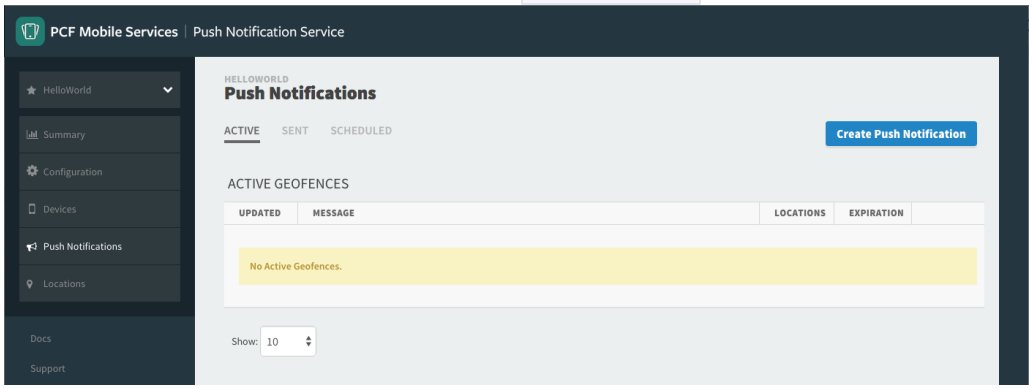
Step 5

Fill in the Name and Description of the Location Group. In the Target Location field, select a location from the drop-down or click on one of the markers on the map. Once all the details are set, click the `Create` button.



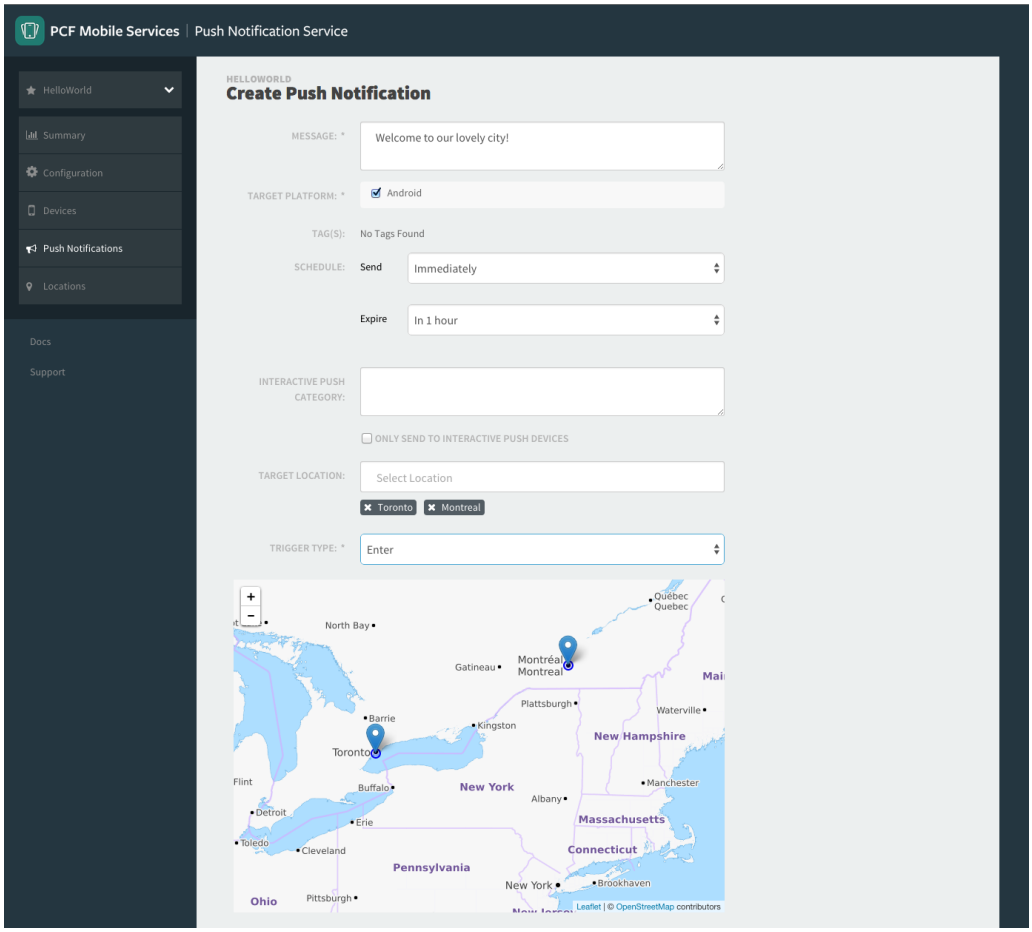
Step 6

Click on `Push Notifications` on the left sidebar, and then on the `Create Push Notification` button.



Step 7

Fill in the details of the Push Notification, such as Message, Platform, and Schedule. Select from the `Target Location` drop-down either a Location or a Location Group. Trigger Type field will appear upon the addition of Location/Location Group. Select either Enter or Exit, depending on how you want the Geofence to activate. Once all the details are set, click the `Send Push Notification` button.


PCF Mobile Services | Push Notification Service
HelloWorld
Summary
Configuration
Devices
Push Notifications
Locations
Docs
Support
HELLOWORLD
Create Push Notification
MESSAGE: *
Welcome to our lovely city!
TARGET PLATFORM: *
Android
TAG(S):
No Tags Found
SCHEDULE:
Send
Immediately
Expire
In 1 hour
INTERACTIVE PUSH CATEGORY:
ONLY SEND TO INTERACTIVE PUSH DEVICES
TARGET LOCATION:
Select Location
Toronto
Montreal
TRIGGER TYPE: *
Enter
Québec
Québec
North Bay
Gatineau
Montréal
Montreal
Plattsburgh
Waterville
Barrie
Kingston
Toronto
New Hampshire
Flint
Buffalo
New York
Albany
Manchester
Detroit
Erie
Massachusetts
Toledo
Cleveland
Connecticut
Pennsylvania
New York
Brookhaven
Ohio
Pittsburgh
Leaflet | © OpenStreetMap contributors

iOS Push Client SDK

Sample Apps

You can find the newest version of the iOS Sample Appon github

Version

This document covers the Pivotal Cloud Foundry Push Notification Service iOS Push Client SDK v1.6.0.

Old documentation:

- Version 1.4.0
- Version 1.3.3
- Version 1.3.2
- Version 1.3.1
- Version 1.3.0
- Version 1.0.4

Please note that there was no release of the Push iOS SDK for 1.5.0.

Features

The PCF Push Notification Service Push Client SDK is a light-weight library that will help your application register with the PCF Mobile Services Push Notifications service.

The SDK does not provide any code for registering with APNS or for handling remote push notifications.

Device Requirements

The Push SDK requires iOS 7.0 or greater. The Push SDK supports iOS 9.0 as of version 1.4.0.

Required Setup

Getting Started

In order to receive push messages from the Push Server in your iOS application, you will need to follow these steps:

Configure iOS Push Notifications on Apple Developer

If you are not familiar with the steps to set up an application on Apple Developer Member Center and set it up for push notifications, see theinstructions below.

You will need to create an **Explicit App Id with Push Notifications enabled**

Note that you can NOT use a Wildcard App ID in an application with push notifications.

Configure iOS Push Notifications on the Push Dashboard

Create your application and platforms on the PCF Mobile Services Push Dashboard. You will need two platforms – one for**development** mode and one

for **production**. Each of these two platforms will need their own Apple Push Notification Service (APNS) SSL certificates; the development platform needs a **sandbox** SSL certificate and the production platform needs a **production** SSL certificate. You will need to export both of these certificates and their associated private signing keys as **P12 files** using the **Keychain Access** program on our Mac OS machine. This task is beyond the scope of this document (see the documentation for the Push Notification Service Dashboard). After setting up your platforms in the administration console make sure to note the **Platform UUID** and **Platform Secret** parameters have been defined under Configuration for both platforms. You will need them below.

You can find steps on how to create your application and platforms on PCF Mobile Services Push Dashboard notes:Push Dashboard Document

Link to the Framework

1. Download the project framework from Pivotal Network and add it to your project in Xcode. You can drag and drop the .framework file into your project in the Project Navigator view. Make sure to enable **Copy items if needed**.

2. Go to the Build Settings in Xcode. Go to the General tab. Remove PCFPush.framework from the **Linked Frameworks And Libraries**. Add PCFPush.framework to the list of **Embedded Binaries**.

3. Go to Build Settings in Xcode, then navigate down to the **Linking** section and add **-ObjC** to **Other Linker Flags**.

NOTE: if you are targeting iOS 7.0 then you will have to compile and link the SDK from source. iOS 7.0 does not support iOS 8.0 frameworks.

Set up your Pivotal.plist file

Create a **Pivotal.plist** file in your project's root directory. The following keys are required:

Key	Type	Required?	Description
pivotal.push.serviceUrl	String	YES	The URL of the PCF Push Notification Service API Server.
pivotal.push.platformUuidDevelopment	String	YES	The platform UUID of your push development platform.
pivotal.push.platformSecretDevelopment	String	YES	The platform secret of your push development platform.
pivotal.push.platformUuidProduction	String	YES	The platform UUID of your push production platform.
pivotal.push.platformSecretProduction	String	YES	The platform secret of your push production platform.
pivotal.push.sslCertValidationMode	String	NO	Can be set to `default`, `trustall`, `pinned`, or `callback`. More details below in the SSL Authentication section.
pivotal.push.pinnedSslCertificateNames	Array	NO	A list of SSL certificates in the `DER` format stored in the application bundle that are used during pinned SSL authentication.
pivotal.push.areAnalyticsEnabled	Boolean	NO	Set to `NO` in order to disable the collection of push analytics at runtime. If this parameter is omitted then analytics are assumed to be enabled.

- None of the above values may be `nil`. None of the above values may be empty.
- The `pivotal.push.platformUuidDevelopment` and `pivotal.push.platformSecretDevelopment` parameters should be the **development platform UUID** and **secret** values from the Push Dashboard. The Push Client SDK uses this platform if it detects that the APNS Sandbox environment is being used at runtime. These values may not be empty or `nil`.
- The `pivotal.push.platformUuidProduction` and `pivotal.push.platformSecretProduction` parameters should be the **production platform UUID** and **secret** values from the Push Dashboard. Note that if you are just trying the Push Client SDK out and don't have an actual production environment set up then you can put dummy data in these fields. These values may not be empty or `nil`.
- For instructions on converting your `PEM` certificate files to `DER`, see the OpenSSL documentation.
- Note that the `pivotal.push.trustAllSslCertificates` property was removed in PCF Push Client SDK 1.3.3.

Register for Push Notifications with APNS

You will need to register your app for push notifications with APNS. Add the following code to your `application:didFinishLaunchingWithOptions:` method in your application delegate.

```
- (BOOL) application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // Register for push notifications with the Apple Push Notification Service (APNS).
    //
    // On iOS 8.0+ you need to provide your user notification settings by calling
    // [UIApplication.sharedDelegate registerUserNotificationSettings:] and then
    // [UIApplication.sharedDelegate registerForRemoteNotifications];
    //
    // On < iOS 8.0 you need to provide your remote notification settings by calling
    // [UIApplication.sharedDelegate registerForRemoteNotificationTypes:].  There are no
    // user notification settings on < iOS 8.0.
    //
    // If this line gives you a compiler error then you need to make sure you have updated
    // your Xcode to at least Xcode 6.0:
    //
    if ([application respondsToSelector:@selector(registerUserNotificationSettings:)]) {

        // iOS 8.0 +
        UIUserNotificationType notificationTypes = UIUserNotificationTypeAlert | UIUserNotificationTypeBadge | UIUserNotificationTypeSound; // Provide different notification types if you need them
        UIUserNotificationSettings *settings = [UIUserNotificationSettings settingsForTypes:notificationTypes categories:nil]; // Provide custom categories if you need them
        [application registerUserNotificationSettings:settings];
        [application registerForRemoteNotifications];

    } else {

        // < iOS 8.0
        UIRemoteNotificationType notificationTypes = UIRemoteNotificationTypeAlert | UIRemoteNotificationTypeBadge | UIRemoteNotificationTypeSound; // Provide different notification types if you need th
        [application registerForRemoteNotificationTypes:notificationTypes];
    }

    return YES;
}
```

- If using geofences you will also need to request authorization for location services here (i.e.: `[self.locationManager requestAlwaysAuthorization]`). Please see the Geofences section below.
- The notification types for < iOS 8.0 are described in the UIApplication Class Reference.
- Note that the OS will display a dialog box on the screen at runtime to confirm the requested notification types to the user when the app attempts to register for push notifications the first time.

Register for Push Notifications with Pivotal CF

Include the following header in your application delegate class:

```
#import <PCFPush/PCFPush.h>
```

In your application delegate's `application:didRegisterforRemoteNotifications:` method put the following code:

```
// This method is called when APNS registration succeeds.
- (void) application:(UIApplication *)app didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
    NSLog(@"APNS registration succeeded!");

    // APNS registration has succeeded and provided the APNS device token.  Start registration with PCF Push
    // Notification Service and pass it the APNS device token.
    //
    // Required: Create a file in your project called "Pivotal.plist" in order to provide parameters for registering with
    // PCF Push Notification Service
    //
    // Optional: You can provide a custom user ID to associate your device with its user.
    //
    // Optional: You can also provide a set of tags to subscribe to.
    //
    // Optional: You can also provide a device alias.  The use of this device alias is application-specific.
    // We recommend that you use the user's device name to populate this field.
    //
    // Optional: You can pass blocks to get callbacks after registration succeeds or fails.
    //
    [PCFPush registerForPCFPushNotificationsWithDeviceToken:deviceToken
                                                       tags:YOUR_TAGS
                                                deviceAlias:YOUR_DEVICE_ALIAS
                                               customUserId:YOUR_CUSTOM_USER_ID
                                         areGeofencesEnabled:ARE_GEOFENCES_ENABLED
      success:^{
        NSLog(@"CF registration succeeded!");
    } failure:^(NSError *error) {
        NSLog(@"CF registration failed: %@", error);
    }];
}
```

- The `YOUR_TAGS` parameter is a parameter that provides a set of the tags that you'd like the application to subscribe to. This parameter should be an `NSSet` object containing a set of `NSString` objects. If you pass in tags via this register method then you need to provide ALL tags that the user has subscribed to each time registration is called. To manage your tags you can also call the `[PCFPush subscribeToTags:success:failure:]` method (described below).
- The `YOUR_DEVICE_ALIAS` parameter is a custom parameter that you can use to identify a user's device (eg: a user may have multiple devices) - this is for future use. We recommend that you use the user's device name to populate this field (e.g.: `UIDevice.currentDevice.name`).
- The `YOUR_CUSTOM_USER_ID` parameter is another custom parameter that you can use to associate this device with the user. It is possible to target push notifications to custom user IDs. If you don't want to use the custom user ID then you can set this argument to `nil` or an empty string. Custom user IDs are treated as case-sensitive. For more information, see Registering with a Custom User ID.
- The `ARE_GEOFENCES_ENABLED` is a `BOOL` value that turns the geofences feature on and off (described below).
- All of the `deviceAlias`, `tags`, `success`, and `failure` parameters are optional and may be set to `nil`.
- You can call the `[PCFPush registerForPCFPushNotificationsWithDeviceToken:tags:deviceAlias:customUserId:areGeofencesEnabled:success:failure:]` method whenever your parameterization changes during runtime (e.g.: when you want to update the device alias). It is not harmful to call this method several times during the lifetime of a process.

Registration Examples

Example 1: Registering for Push Notifications with no options, tags, and without geofences.

```
- (void) application:(UIApplication *)app didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
    [PCFPush registerForPCFPushNotificationsWithDeviceToken:deviceToken
                                                       tags:nil
                                                deviceAlias:nil
                                               customUserId:nil
                                         areGeofencesEnabled:NO
                                                    success:^{ NSLog(@"CF registration succeeded!"); }
                                                    failure:^(NSError *error) { NSLog(@"CF registration failed: %@", error); }];
}
```

Example 2: Registering for Push Notifications with a customer user ID using the user's account name (for example).

```
- (void) application:(UIApplication *)app didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
    [PCFPush registerForPCFPushNotificationsWithDeviceToken:deviceToken
                                                       tags:nil
                                                deviceAlias:nil
                                               customUserId:@"test@example.net" // User's account name
                                        areGeofencesEnabled:NO
                                                    success:^{ NSLog(@"CF registration succeeded!"); }
                                                    failure:^(NSError *error) { NSLog(@"CF registration failed: %@", error); }];
}
```

Example 3: Removing the registration for the custom user ID (which will prevent the user from being targeted by their custom user ID).

```
- (void) application:(UIApplication *)app didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
    [PCFPush registerForPCFPushNotificationsWithDeviceToken:deviceToken
                                                       tags:nil
                                                deviceAlias:nil
                                               customUserId:@"" // Remove the user's account name. Can use nil or empty string.
                                        areGeofencesEnabled:NO
                                                    success:^{ NSLog(@"CF registration succeeded!"); }
                                                    failure:^(NSError *error) { NSLog(@"CF registration failed: %@", error); }];
}
```

Example 4: Subscribing to several topics on a news service.

```
- (void) application:(UIApplication *)app didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
    [PCFPush registerForPCFPushNotificationsWithDeviceToken:deviceToken
                                                       tags:[NSSet setWithArray:@[@"breaking_news", @"local_news"]]
                                                deviceAlias:nil
                                               customUserId:nil
                                        areGeofencesEnabled:NO
                                                    success:^{ NSLog(@"CF registration succeeded!"); }
                                                    failure:^(NSError *error) { NSLog(@"CF registration failed: %@", error); }];
}
```

Example 5: Unsubscribing from the “breaking_news” tag while remaining subscribed to the “local_news” tag.

```
- (void) application:(UIApplication *)app didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
{
    [PCFPush registerForPCFPushNotificationsWithDeviceToken:deviceToken
                                                       tags:[NSSet setWithObject:@"local_news"]
                                                deviceAlias:nil
                                               customUserId:nil
                                        areGeofencesEnabled:NO
                                                    success:^{ NSLog(@"CF registration succeeded!"); }
                                                    failure:^(NSError *error) { NSLog(@"CF registration failed: %@", error); }];
}
```

Receiving Push Notifications

To receive push notifications you can implement the following code in your application delegate class.

- **VERY IMPORTANT:** You must call the `[PCFPush didReceiveRemoteNotification:completionHandler:]` method in your application delegate `application:didReceiveRemoteNotification:fetchCompletionHandler` method, as demonstrated below.

```
// This method is called when APNS sends a push notification to the application.
- (void) application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userInfo
{
    [self handleRemoteNotification:userInfo];
}

// This method is called when APNS sends a push notification to the application when the application is
// not running (e.g.: in the background).  Requires the application to have the Remote Notification Background Mode Capability.
- (void)application:(UIApplication *)application didReceiveRemoteNotification:(NSDictionary *)userInfo fetchCompletionHandler:(void (^)(UIBackgroundFetchResult))completionHandler
{
    [self handleRemoteNotification:userInfo];

    // IMPORTANT: Inform PCF Push Notification Service that this message has been received.
    [PCFPush didReceiveRemoteNotification:userInfo completionHandler:^(BOOL wasIgnored, UIBackgroundFetchResult fetchResult, NSError *error) {

        if (completionHandler) {
            completionHandler(fetchResult);
        }
    }];
}

// This method is called when the user touches one of the actions in a notification when the application is
// not running (e.g.: in the background).  iOS 8.0+ only.
- (void)application:(UIApplication *)application handleActionWithIdentifier:(NSString *)identifier forRemoteNotification:(NSDictionary *)userInfo completionHandler:(void(^)())completionHandler
{
    NSLog(@"Handling action %@ for message %@", identifier, userInfo);
    if (completionHandler) {
        completionHandler();
    }
}

- (void) handleRemoteNotification:(NSDictionary*) userInfo
{
    if (userInfo) {
        NSLog(@"Received push message: %@", userInfo);
    } else {
        NSLog(@"Received push message (no userInfo).");
    }
}
```

If you do not call `[PCFPush didReceiveRemoteNotification:completionHandler:]` then the SDK will not be able to fetch geofence updates nor will it be able to capture push analytics data.

Optional Items

Enable or disable push analytics

Version 1.3.3 of the PCF Push Client SDK supports the collection of some simple push analytics data:

- Receiving push notifications
- Opening push notifications
- Triggering geofences

Analytics are enabled by default. You can disable it by setting the `pivotal.push.areAnalyticsEnabled` BOOLEAN parameter in your `pivotal.plist` file to `NO`.
Ensure that you have an up-to-date version of the PCF Push API server and that it is generating `receiptId` data in the remote notifications that it generates.

In order for the SDK to capture push analytics data you will need to make sure to call the `[PCFPush didReceiveRemoteNotification ...]` method in your `application:didReceiveRemoteNotification:` handler, as described in the Receiving Push Notifications section above.

Ensure your that the **remote notifications** background mode has been set for your project target configuration in order to capture analytics data when push notifications are received by the device when your application is in the background.

NOTE: If a remote notification does not have the `"content-available":1` field in its payload and if the user does not touch the notification then there will be no analytics event logged for receiving the notification when the application is in the background (since iOS does not call the application for the remote notifications in the background without `"content-available":1`).

Subscribing to Tags

The `[PCFPush subscribeToTags:success:failure:]` method allows you to manage your tags after registration has completed. If you call this method before registration is complete then an error will occur. This parameter should be an `NSSet` object containing a set of `NSString` objects.

In general, an application should keep track of all of the tags it is currently subscribed to. Whenever you call `[PCFPush registerForPCFPushNotificationsWithDeviceToken:tags:deviceAlias:customUserId:areGeofencesEnabled:success:failure:]` or `[PCFPush subscribeToTags:success:failure:]` you need to pass **ALL** of the tags that the application is currently subscribed to. If you want to add new tags you must provide them alongside the tags you are currently subscribed to. If you omit some tags then the SDK will think that you want to **unsubscribe** from those tags.

Unregistering from Pivotal Cloud Foundry Push Notification Service

The `[PCFPush unregisterFromPCFPushNotificationsWithSuccess:failure:]` method allows you to unregister from push notifications from PCF. After unregistering PCF will stop sending the device any notifications.

Reading the Device UUID

In order to target individual devices for remote notifications using the PCF Push Notification Service you will need to target the**Device UUID** assigned to each device by the service. You can read the Device UUID at run time any time after a successful registration with the service by calling the `[PCFPush deviceUuid]` method. This method will return `nil` if the device is not currently registered with the PCF Push Notification Service.

Example:

```
[PCFPush registerForPCFPushNotificationsWithDeviceToken:deviceToken
                                                   tags:nil
                                            deviceAlias:UIDevice.currentDevice.name
                                           customUserId:nil
                                  areGeofencesEnabled:YES
                                                success:
^{
    PCFPushLog(@"The Device UUID is \"%@\".", [PCFPush deviceUuid]);

    // Note: add code to transmit the deviceUuid to your middleware server.

} failure:^(NSError *error) {
    PCFPushLog(@"CF registration failed: %@", error);
}];
```

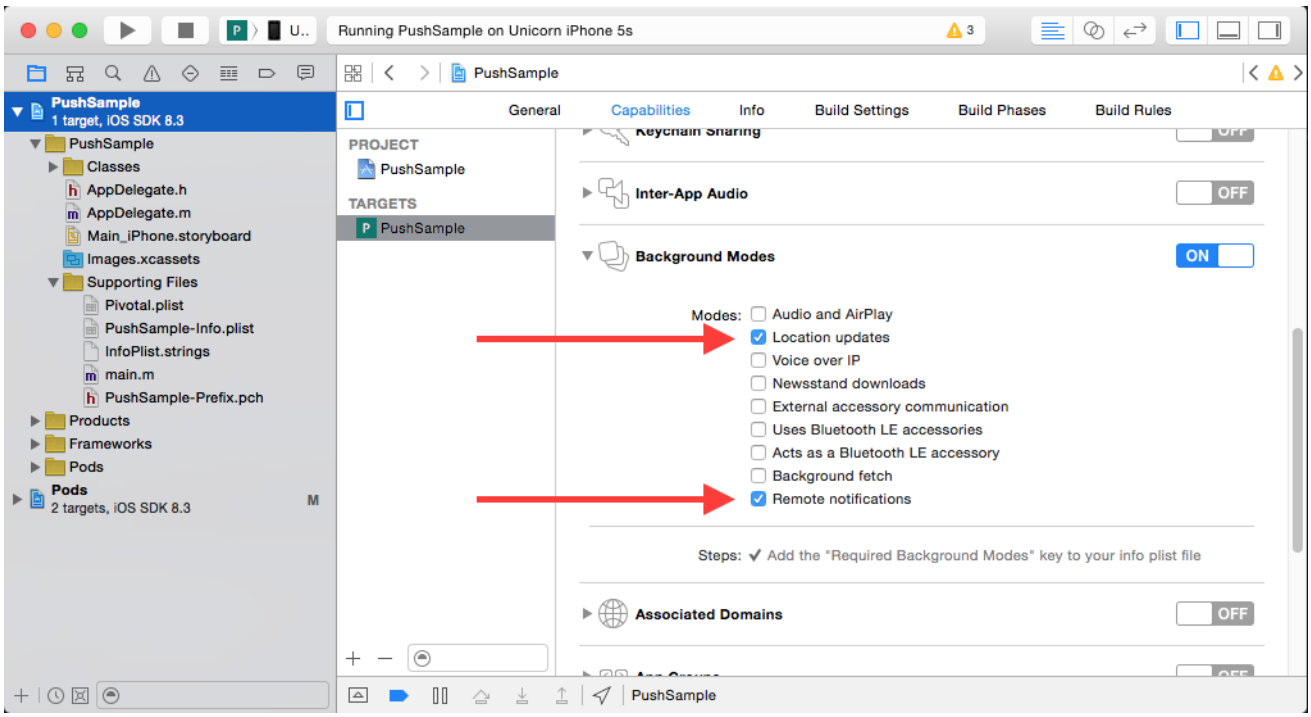
Geofences

Geofences are newly supported in version 1.3.0 of the Push Notification Service. Using this service you will be able to register push notifications that your app users will see when they enter or exit certain geographic regions that you define on the Push Notification Service Dashboard.

In order to set up your app to receive geofence notifications, follow these steps.

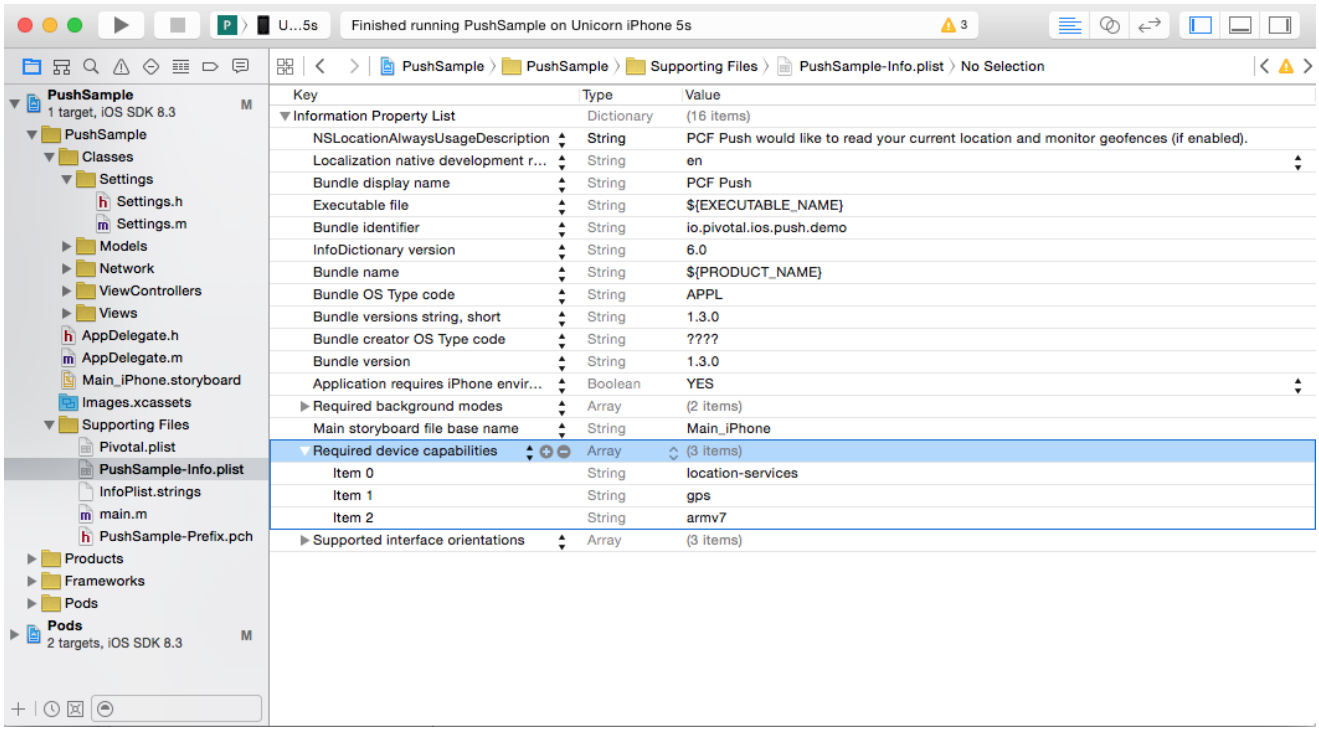
Step 1 - Set your background modes

Ensure your **location updates** and **remote notifications** background modes have been set for your project target capabilities. Both of these modes are required for your application to fetch and monitor geofence updates from the server.



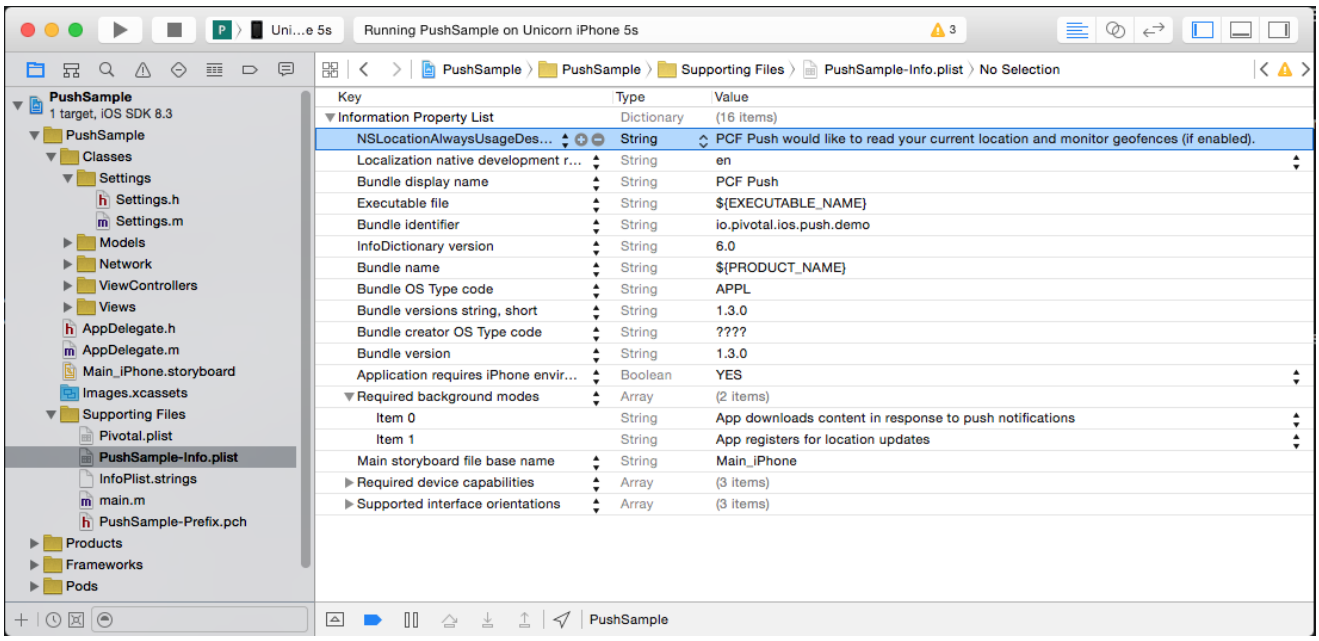
Step 2 - Set required device capabilities

Add **location-services** and **gps** to your application Info.plist file under “Required device capabilities”.



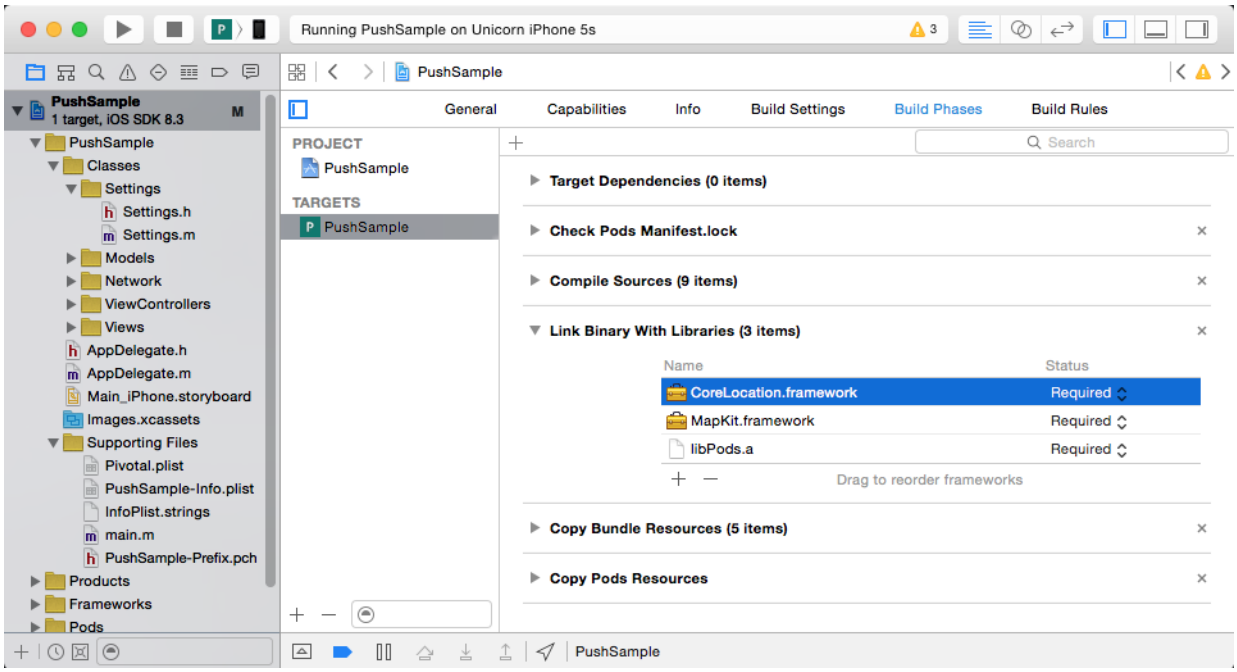
Step 3 - Set your location usage description

If this is the first time that your app is using any location services then you will need to set the text that is displayed on iOS 8.0+ when the app first requests the permission to read your current device location. You can set this text by setting the `NSLocationAlwaysUsageDescription` key in your app's **Info.plist** file (contained in Supporting Files folder by default). e.g.: “Your App Name would like to read your current location and monitor geofences (if enabled).”



Step 4 - Link to Core Location

Ensure that your app is linked to the Core Location framework. In Xcode, go to your app targets build phases screen and add `CoreLocation.framework` to the **Link Binary With Libraries** build phase.



Step 5 - Enable geofences

In order to enable geofences at runtime you will need to pass `YES` to the `areGeofencesEnabled` argument when you call the `[PCFPush registerForPCFPushNotificationsWithDeviceToken ...` method in your application delegate. If this parameter is set to `NO` then no geofences features will be available at runtime. Any geofences that may have been monitored before will be cleared and will no longer be monitored.

Step 6 - Authorize location services

If using geofences on iOS 8.0+ devices you will need to add the method call to request permission from the user to read the current device location. A good place for that is in your application delegate `application:didFinishLaunchingWithOptions` method. This call will show an alert dialog box to the user that shows

the `NSLocationAlwaysUsageDescription` text in your PLIST file.

```
- (BOOL) application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // Register for push notifications with the Apple Push Notification Service (APNS).
    //
    // On iOS 8.0+ you need to provide your user notification settings by calling
    // [UIApplication.sharedDelegate registerUserNotificationSettings:] and then
    // [UIApplication.sharedDelegate registerForRemoteNotifications];
    //
    // On < iOS 8.0 you need to provide your remote notification settings by calling
    // [UIApplication.sharedDelegate registerForRemoteNotificationTypes:].  There are no
    // user notification settings on < iOS 8.0.
    //
    // If this line gives you a compiler error then you need to make sure you have updated
    // your Xcode to at least Xcode 6.0:
    //
    if ([application respondsToSelector:@selector(registerUserNotificationSettings:)]) {

        // iOS 8.0 +
        UIUserNotificationType notificationTypes = UIUserNotificationTypeAlert | UIUserNotificationTypeBadge | UIUserNotificationTypeSound;
        UIUserNotificationSettings *settings = [UIUserNotificationSettings settingsForTypes:notificationTypes categories:nil];
        [application registerUserNotificationSettings:settings];
        [application registerForRemoteNotifications];

        // NOTE: add this block to enable location services for geofences
        if ([application respondsToSelector:@selector(registerUserNotificationSettings:)]) {
          self.locationManager = [[CLLocationManager alloc] init];
          [self.locationManager requestAlwaysAuthorization]; // iOS 8.0+ only
        }

    } else {

        // < iOS 8.0
        UIRemoteNotificationType notificationTypes = UIRemoteNotificationTypeAlert | UIRemoteNotificationTypeBadge | UIRemoteNotificationTypeSound;
        [application registerForRemoteNotificationTypes:notificationTypes];
    }

    return YES;
}
```

Step 7 - Add property to application delegate

Required only if you are using geofences: add a property to your application delegate class (AppDelegate.h) as follows:

```
@property (strong, nonatomic) CLLocationManager *locationManager;
```

You will also need to include the following header to the same file:

```
#import <CoreLocation/CoreLocation.h>
```

Step 8 - Receiving Local Notifications

If you follow the above steps then your application will be able to show geofences when they are triggered. Geofences are delivered as local notifications to your app. Similar to remote notifications, local notifications will be automatically displayed when your application is in the background but you will need to add your own code in order to display them when your app is in the foreground.

If you need to know if the geofence was triggered via an ‘enter’ or 'exit' condition then look at the `pivotal.push.geofence_trigger_condition` key in the userInfo dictionary provided with the location notification. You can also use this userInfo field to distinguish geofence local notifications from other kinds of local notifications.

As an example, if you want to print a log message when a local notification is received:

```
- (void) application:(UIApplication *)application didReceiveLocalNotification:(UILocalNotification *)notification
{
    NSLog(@"Received %@ local notification '%@'", notification.userInfo[@"pivotal.push.geofence_trigger_condition"], notification.alertBody);
}
```

Step 9 - Receive Geofence Status Updates

The PCF Push Notification Service server will push updated geofences to user devices via push notifications. You don't need to do any more work to process these updates or monitor these geofences. You can read the geofence status object to find out if any problems occur during these background updates. These errors can be reported directly to your application if you add an observer to the `PCF_PUSH_GEOFENCE_STATUS_UPDATE_NOTIFICATION` notification in `NSNotificationCenter`.

Example:

You can subscribe to the geofence update notification with the following code in your program. You could put it in your one of your view controllers or your application delegate, as you see fit.

```
[[NSNotificationCenter defaultCenter] addObserver:self selector:@selector(geofenceStatusChanged:) name:PCF_PUSH_GEOFENCE_STATUS_UPDATE_NOTIFICATION object:nil];
```

The above method call will cause the `geofenceStatusChanged` method to be called. You will need to define this method yourself in the same class (or in whatever object instance you passed to `NSNotificationCenter` above:

```
- (void) geofenceStatusChanged:(NSNotification*)notification
{
   PCFPushGeofenceStatus *status = [PCFPush geofenceStatus];
   NSLog(@"%@", status);
}
```

SSL Authentication

The property `pivotal.push.sslCertValidationMode` allows the application to accept the following supported SSL Authentication modes:

1. **default** : When the service URL is not HTTPS or when using a server trusted certificate this mode should be set.
2. **trustall** : When using a development environment there is the ability to trust all certificates while using a HTTPS service URL. This mode replaces the previous property (prior to v1.3.3) `pivotal.push.trustAllSslCertificates`.
3. **pinned** : To ensure no man in the middle attacks this mode should be set. The server certificate will be verified with the local copy of the certificate referred to as Certificate Pinning authentication. When this mode is set the local copy of the certificate(s) should be provided with the `pivotal.push.pinnedSslCertificateNames` array property. All certificates provided will be stored in the assets folder of the application in a **DER** format.
4. **callback** : When a custom SSL authentication schema is required this mode can be set whereby the specific authentication logic would be added inside the application as a callback to the SDK. The callback must be a block that receives the arguments `(NSURLConnection *, NSURLAuthenticationChallenge *)` and will be called when attempting to make an HTTPS network request.

In order for this method to take effect you will need to call it both *before* `[PCFPush registerForPCFPushNotificationsWithDeviceToken:...]` and also *before* `[PCFPush didReceiveRemoteNotification:...]`.

example:

```
@implementation AppDelegate

  ...

  - (PCFPushAuthenticationCallback) getAuthenticationCallback
  {
     return ^(NSURLConnection *connection, NSURLAuthenticationChallenge *challenge) {
        // Handle the SSL challenge here!
     };
  }

  - (void) application:(UIApplication *)app didRegisterForRemoteNotificationsWithDeviceToken:(NSData *)deviceToken
  {
     [PCFPush setAuthenticationCallback:[self getAuthenticationCallback]];
     ...
     [PCFPush registerForPCFPushNotificationsWithDeviceToken:deviceToken ... ];
  }


  - (void)application:(UIApplication *)app didReceiveRemoteNotification:(NSDictionary *)userInfo fetchCompletionHandler:(void (^)(UIBackgroundFetchResult))completionHandler
  {
     [PCFPush setAuthenticationCallback:[self getAuthenticationCallback]];
     ...
     [PCFPush didReceiveRemoteNotification:userInfo completionHandler: ... ];
  }

  ...

@end
```

Please see Apple's documentation for the NSURLConnectionDelegate connection:willSendRequestForAuthenticationChallenge method for more information on how to handle the callback.

Setting custom HTTP request headers

In order to inject custom headers into any HTTP requests made by the Push SDK you should call the `[PCFPush setRequestHeaders:]` method with a dictionary of the required HTTP header values. All values should be pairs of (NSString, NSString) values. Note that you can not provide any 'Authorization' or 'Content-Type' headers via this method; they will be ignored by the Push SDK.

In order for this method to take effect you will need to call it *before* `registerForPCFPushNotificationsWithDeviceToken`.

example:

```
[PCFPush setRequestHeaders:@{ @"Cookie:"@"MY_SESSION_COOKIE", @"My-Special-Custom-Header":@"My-Special-Custom-Value" }];
...
[PCFPush registerForPCFPushNotificationsWithDeviceToken:@"My-Device-Token" ... ...];
```

Appendix

iOS 9.0+ Notes - App Transport Security

Apple introduced App Transport Security (ATS) in iOS 9.0. ATS will, by default block all HTTP connections. If you want to use HTTP in iOS 9.0 apps then you will have to set up an ATS exception in your `Info.plist` file and enable `NSExceptionAllowsInsecureHTTPLoads` for your desired subdomain. Apple does not recommend HTTP and recommends using ATS as soon as possible.

If you are using HTTPS and need to use any of the "trustall", "pinned", or "callback" sslCertValidationModes then you will also need to enable `NSExceptionAllowsInsecureHTTPLoads` for your desired subdomain. Enabling insure HTTP loads will allow the custom SSL validation in the PCF Push SDK.

Example info.plist:

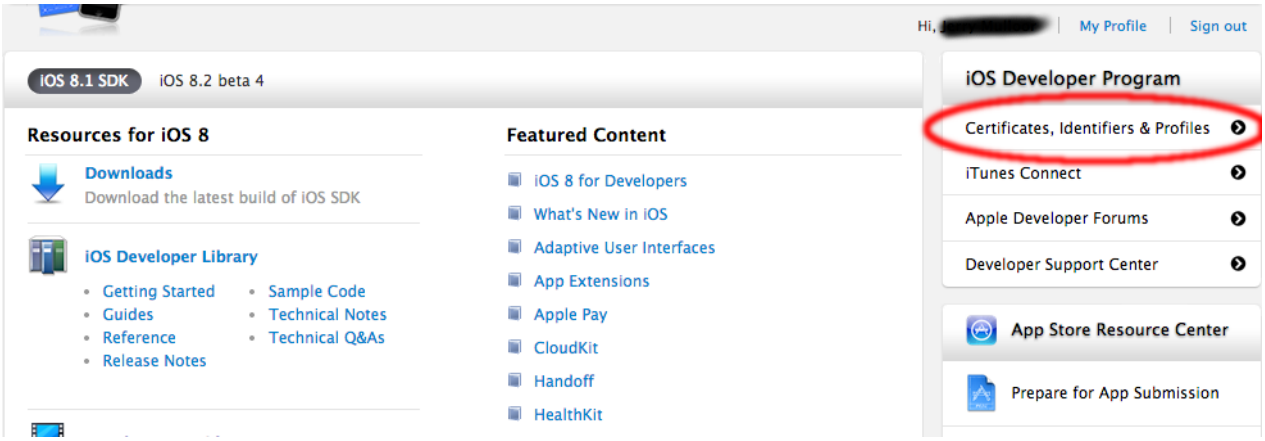
```
<key>NSAppTransportSecurity</key>
<dict>
 <key>NSExceptionDomains</key>
 <dict>
  <key>yourserver.com</key>
  <dict>
   <!--Include to allow subdomains-->
   <key>NSIncludesSubdomains</key>
   <true/>
   <!--Include to allow HTTP request and custom SSL validation -->
   <key>NSExceptionAllowsInsecureHTTPLoads</key>
   <true/>
  </dict>
 </dict>
</dict>
```

Setting up your app on Apple Developer Member Center

If you are not familiar with how to create an application on the Apple Developer Member Center, follow the steps below. This information is subject to change and you may find more up-to-date information at App Distribution Guide.

Generating an App ID

1. Log into your Apple Developer Account.
2. Click the `Certificates, Identifiers & Profiles` link on the right side of the page.



3. On the `iOS Apps` section on the left side of the page click the `Identifiers` link.
4. You should now be on the `iOS App IDs` page. Click the `+` button on the top right to create your AppID.
5. Fill in your `App ID Description` and `Bundle ID` under `App ID Suffix` → `Explicit App ID`. This `Bundle ID` is the same `Bundle Identifier` that was generated when you create your application in Xcode.
6. Scroll down to the `App Services` Section and under `Enable Services` check `Push Notifications`. Once `Push Notifications` are enabled click the `Continue` button.
7. Look over the settings on the next page and click `Submit` when you've verified your settings.
8. You should now see your `App ID` in the list on the `iOS App IDs` page.

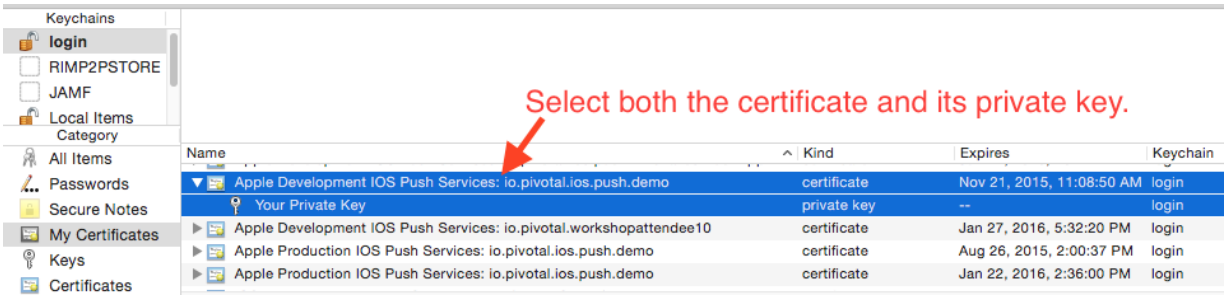
Push Sandbox SSL Certificate

1. Click on your newly created `App ID` and click the `Edit` button.
2. Scroll down to the `Push Notifications` section. We will now generate a `Development SSL Certificate`. Navigate to the `Development SSL Certificate` section and then click on the `Create Certificate` button.
3. Follow the instructions on the `About Creating a Certificate Signing Request (CSR)` page:
   - Open `Keychain Access`.

- Within the `Keychain Access drop down menu` select `Certificate Assistant` → `Request a Certificate from a Certificate Authority`.
- Type in your email address.
- Ensure `Saved to disk` is checked.
- Click the `Continue` button.
- Save the certificate to disk and Reveal in Finder.

4. Go back to your web browser to the `About Creating a Certificate Signing Request (CSR)` page and click `Continue`. Choose the certificate signing request that you just saved to disk and click `Generate`. You will need to download this file and open it. `Keychain Access` should open this file. If prompted, add it to the `login` keychain. You should be able to see this certificate if you navigate to the `My Certificates` section in `Keychain Access`.

5. Export your certificate as a `p12` file with a password.
   - Navigate to your `My Certificates` section in `Keychain Access`
   - Expand your certificate and select both items.



   - Right click on the certificate and select `Export 2 items...`
   - Name this certificate with your `Bundle ID` and append `Sandbox` to the end, and ensure that the File Format is `Personal Information Exchange (.p12)`
   - Select a password to protect this certificate with, you will need this password when you setup the PCF Push server though the PCF Push Dashboard. Save this `.p12` file in a location you will remember.

Generate your provisioning profile

1. Go to the `Provisioning Profiles` on the left and click the `Development` link.
2. Click the `+` at the top right of the page by `iOS Provisioning Profiles`
3. Go to the `Development` section and select `iOS App Development`. Click the `Continue` button to proceed.
4. Select the `AppID` that you created above. Click the `Continue` button to proceed.
5. Select your signing certificate. Click the `Continue` button to proceed.
6. Select your desired test devices.
7. Click the `Generate` button to generate your provisioning profile.
8. Click the `Download` button to download your provisioning profile. Open this file and go back to Xcode.
9. In Xcode, make sure you are on the `Build Settings` tab and navigate down to `Provisioning Profile`. Select the provisioning profile that you just created. This profile will only show up if you opened the file from the previous step.

Troubleshooting

Please see our troubleshooting guide

Android Push Client SDK

Sample App

You can find the Android Sample App on Github.

Version

This document covers the Android Push Client SDK v1.6.0.

Old documentation:

- v1.4.0
- v1.3.3
- v1.3.2
- v1.3.1
- v1.3.0
- v1.0.4

There was no release of the Push Android SDK for v1.5.0.

Features

The Android Push Client SDK is a light-weight library that helps your app:

1. Register for push notifications with Google Cloud Messaging (GCM) and an instance of the PCF Push Notification Service.
2. Receive push messages sent via the same frameworks.
3. Monitor geofences that have been configured from a central server.

Device Requirements

The Push SDK requires **Android API level 16** or greater. Support for Android 14 and 15 was dropped as of Push SDK v1.4.0.

The **Google Play Services** app must be installed on the device before you can register your device or receive push messages. Typically, the user needs to be logged into a Google account as well. Most devices already have this app installed, but some odd ones may not. You should be able to receive push notifications on a Android emulated device if it has the **Google APIs** installed.

Required Setup

Getting Started

To receive push messages from the PCF Push Notification Service in your Android app, you need to create a project within the Google Developers Console. See Google Developers Console below.

Set up your app and an **Android Platform** on the PCF Push Notification Service Dashboard. This task is beyond the scope of this document, but note that you need the **API Key** parameter from Google Cloud Console above. After setting up your Android platform in PCF Mobile Services, note down the **Platform UUID** and **Platform Secret** parameters. You need them below. At this time, the Android Push software makes no distinction between developer and production modes.

For information on how to create your app and platforms, see Using the Dashboard.

Link to PCF Push SDK

Download the PCF Push Client SDK for Android from Pivotal Network. The Client SDK is delivered as an Android Library (i.e.: an "AAR" file). Copy the AAR file into the `libs` directory of your project and ensure that the following line line is in the `dependencies` section of your module-level `build.gradle` file:

```
repositories {
   mavenCentral()
   flatDir {
      dirs 'libs'
   }
}
```

Additionally, add the following dependency to the `dependencies` section of your module-level `build.gradle` file:

```
dependencies {
   compile(name:'PCFPush-1.6.0', ext:'aar')
   compile 'com.google.code.gson:gson:2.4'
   compile 'com.google.android.gms:play-services-location:8.4.0'
   compile 'com.google.android.gms:play-services-gcm:8.4.0'
   compile 'com.android.support:support-annotations:23.3.0'
   compile 'com.android.support:appcompat-v7:23.3.0'
}
```

You need to define and use the following `permission` element in the `manifest` element of your app's `AndroidManifest.xml` file. Ensure that the base of the **permission name** is your app's **package name**:

```
<permission
   android:name="[YOUR.PACKAGE.NAME].permission.C2D_MESSAGE"
   android:protectionLevel="signature" />

<uses-permission android:name="[YOUR.PACKAGE.NAME].permission.C2D_MESSAGE" />
```

You need to add the following `receiver` to the `application` element of your app's `AndroidManifest.xml` file. Ensure that you set the **category name** to your app's **package name**:

```
<receiver
   android:name="io.pivotal.android.push.receiver.GcmBroadcastReceiver" android:permission="com.google.android.c2dm.permission.SEND">
   <intent-filter>
      <action android:name="com.google.android.c2dm.intent.RECEIVE"/>
      <category android:name="[YOUR.PACKAGE.NAME]"/>
   </intent-filter>
</receiver>
```

Configuration: Set Up Your pivotal.properties File

Create a **pivotal.properties** file in your project's `src/main/assets` or `src/main/res/raw` directory. The following properties are required:

Property	Required	Description
pivotal.push.serviceUrl	Yes	The URL of the PCF Push Server.
pivotal.push.platformUuid	Yes	The platform UUID of your push platform on the PCF Push server.
pivotal.push.platformSecret	Yes	The platform secret of your push platform on the PCF Push server.
pivotal.push.gcmSenderId	Yes	The project number assigned by Google Cloud Console.
pivotal.push.sslCertValidationMode	No	Can be set to `default`, `trustall`, `pinned`, or `callback`. More details below in the SSL Authentication section.
pivotal.push.pinnedSslCertificateNames	No	If using `pinned` SSL validation mode then this property should be a list of SSL certificates in the `DER` format stored in the assets directory. The list is space separated.
pivotal.push.areAnalyticsEnabled	No	Set to `false` to disable the capture of push analytics data. Defaults to `true`.

- None of the above values may be `null`. None of the above values may be empty.
- The `pivotal.push.platformUuid` and `pivotal.push.platformSecret` parameters are the **platform UUID** and **secret** values from the Push Dashboard. If you use the SDK v1.6, then use UUID and secret of platform type `Android`. If you use the SDK v1.7, then use the type `Android-FCM`.
- For instructions on how to convert your `PEM` certificate files to `DER`, see the OpenSSL documentation.
- Note that the `pivotal.push.trustAllSslCertificates` property was removed in PCF Push Client SDK v1.3.3.

Registration

It is recommended that you initialize the Push Client SDK in your app's primary `Activity` subclass' `onCreate` method.

Add the following lines of code to the initialization section of your app. You need a `Context` object to pass to the `getInstance` method, so you should try to add this code to your `Activity` class. In the example below the `Context` is the `this` object passed to the `getInstance` method (assuming that we're in an Activity):

```
try {
   // RegistrationListener is optional and may be `null`.
   Push.getInstance(this).startRegistration(DEVICE_ALIAS, CUSTOM_USER_ID, TAGS, ARE_GEOFENCES_ENABLED, new RegistrationListener() {

      @Override
      public void onRegistrationComplete() {
         Log.i("MyLogTag", "Registration with PCF Push successful.");
      }

      @Override
      public void onRegistrationFailed(String reason) {
         Log.e("MyLogTag", "Registration with PCF Push failed: " + reason);
      }
   });
} catch (Exception e) {
   Log.e("MyLogTag", "Registration with PCF Push failed: " + e);
}
```

The **DEVICE_ALIAS** is a custom field that you can use to differentiate this device from others and is intended for future use. If you don't want to use the device alias then you can set this argument to `null` or an empty string. At this time you can not use the device alias for targeting push notifications. We recommend that you use the user's device name to populate this field.

The **CUSTOM_USER_ID** is another custom field that you can use to associate this device with the user. It is possible to target push notifications to custom user IDs. If you don't want to use the custom user ID then you can set this argument to `null` or an empty string. Custom user IDs are treated as case-sensitive.

The **TAGS** parameter is a `Set<String>` of tags that your app would like to subscribe to. There are many possible uses of tags but they are dependent on your particular use cases. Always ensure that you provide all of the tags that you'd like to be subscribed to; if you omit tags in future calls to the register method then the SDK thinks that you are trying to unsubscribe from those tags. If there are no tags that you want to register to then you can set this argument to `null`. Tags are treated as case-insensitive.

The **ARE_GEOFENCES_ENABLED** is a `boolean` value that turns the geofences feature on and off (described below). If you want to use geofences in your app, then request permission to read the device location. If you want to support Android Marshmallow, you must write extra code to request the device location. This extra code is described in the geofences section below.

You should only have to call `startRegistration` once in the lifetime of your process – but calling it more times is not harmful. The `startRegistration` method is asynchronous and will return before registration is complete. If you need to know when registration is complete (or if it fails), then provide a `RegistrationListener` as the second argument.

Registration Examples

Example 1: Registering for Push Notifications with no options, tags, without geofences and with no callback.

```
Push.getInstance(this).startRegistration(null, null, null, false, null);
```

Example 2: Registering for Push Notifications with a customer user ID using the user's account name (for example).

```
final String customUserId = "test@example.net"; // Your user's account name
Push.getInstance(this).startRegistration(null, customUserId, null, false, null);
```

Example 3: Removing the registration for the custom user ID (which prevents the user from being targeted by their custom user ID).

```
final String customUserId = ""; // Can use null or empty string to remove the custom user ID
Push.getInstance(this).startRegistration(null, customUserId, null, false, null);
```

Example 4: Subscribing to several topics on a news service.

```
final Set<String> tags = new HashSet<>();
tags.add("breaking_news");
tags.add("local_news");
Push.getInstance(this).startRegistration(null, null, tags, false, null);
```

Example 5: Unsubscribing from the “breaking_news” tag while remaining subscribed to the “local_news” tag.

```
final Set<String> tags = new HashSet<>();
tags.add("local_news");
Push.getInstance(this).startRegistration(null, null, tags, false, null);
```

Receiving Push Notifications

To receive push notifications in your app, you need to add a custom `Service` to your app that extends the `GcmService` provided in the SDK. The intent that GCM sends is passed to your service’s `onReceiveMessage` method. Here is a simple example:

```
public class MyPushService extends GcmService {

   @Override
   public void onReceiveMessage(Bundle payload) {
      if (payload.containsKey("message")) {
         final String message = payload.getString("message");
         handleMessage(message);
      }
   }

   private void handleMessage(String msg) {
      // Your code here. Display the message
      // on the device's bar as a notification.
   }
}
```

Finally, you need to declare your service in your **AndroidManifest.xml** file.

```
<service android:name=".MyPushService" android:exported="false" />
```

Optional Items

Push Analytics

Version 1.3.3 of the PCF Push Client SDK supports the collection of some simple push analytics data:

- Receiving push notifications
- Opening push notifications
- Triggering geofences

Analytics are enabled by default. You can disable it by setting the `pivotal.push.areAnalyticsEnabled` parameter in your `pivotal.properties` file to `false`. Ensure that you have an up-to-date version of the PCF Push API server and that it is generating `receiptId` data in the remote notifications that it generates (which is activated by default).

Since the notification capabilities on Android are very diverse the SDK doesn’t do any work to help apps display them. It relies on your app to decide how to display and handle all push notifications. As such, there is no way for the SDK to know when the user touches a notification and opens your app. If you want to collect metrics about how many users are opening the notifications in your app then the SDK relies on your app to inform it. You need to call the `logOpenedNotification` method in the `Push` class with the same `Bundle` that was delivered in the push notification.

The capturing push analytics data requires v1.3.2 of the Push API server. The SDK checks the server version before capturing any analytics data. If the server version is too old, then no analytics data is recorded. The SDK checks the server version once every 24 hours in release builds and every 5 minutes in debug builds.

e.g.:

Let's say that you use this code to display a push notification in your subclass of `GcmService`:

```
@Override
public void onReceiveMessage(Bundle payload) {
   final String message = payload.getString("message");

   final NotificationManager notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);

   final Intent intent = new Intent(this, MyAppsMainActivity.class);
   intent.setAction("YOUR_CUSTOM_NOTIFICATION_ACTION_NAME");
   intent.putExtras(payload);
   final PendingIntent contentIntent = PendingIntent.getActivity(this, 0, intent, 0);

   final NotificationCompat.Builder builder = new NotificationCompat.Builder(this)
      .setSmallIcon(R.drawable.ic_your_app_logo)
      .setContentTitle(getString(R.string.app_name))
      .setContentIntent(contentIntent)
      .setContentText(msg);

   notificationManager.notify(NOTIFICATION_ID, builder.build());
}
```

Then you can use the following code in the opened activity to report that the notification has been opened:

```
public class MyAppsMainActivity extends Activity {
...

@Override
protected void onResume() {
   super.onResume();

   final Intent i = getIntent();
   if (i.getAction().equals("YOUR_CUSTOM_NOTIFICATION_ACTION_NAME")) {
      Push.getInstance(this).logOpenedNotification(i.getExtras());
   }
}
```

Note that it is important to pass the entire remote notification payload `Bundle` into the `logOpenedNotification` method. This example accomplishes this requirement by saving the payload `Bundle` in the `Intent` `Extras` in the `PendingIntent` passed to the notification.

Tags

If any of your tags change during the lifetime of your process (e.g.: your app wants to change the list of tags that it has subscribed to) then call `subscribeToTags` with your new set of parameters. Example:

```
// The SubscribeToTagsListener is optional and may be `null`.
Push.getInstance(this).subscribeToTags(TAGS, new SubscribeToTagsListener() {
  @Override
  public void onSubscribeToTagsComplete() {
     Log.i("MyLogTag", "Successfully subscribed to tags with PCF Push.");
  }

  @Override
  public void onSubscribeToTagsFailed(String reason) {
     Log.e("MyLogTag", "Failed to subscribe to tags with PCF Push:" + reason);
  }
});
```

Unregistration

If you want to unregister from push notifications then you can call the `startUnregistration` method:

```
// The UnregistrationListener is optional and may be `null`.
Push.getInstance(this).startUnregistration(new UnregistrationListener() {
   @Override
   public void onUnregistrationComplete() {
      Log.i("MyLogTag", "Successfully unregistered from PCF Push.");
   }

   @Override
   public void onUnregistrationFailed(String reason) {
      Log.e("MyLogTag", "Failed to unregister from PCF Push: " + reason);
   }
});
```

Reading the Device UUID

In order to target individual devices for remote notifications using the PCF Push Notification Service, you need to target the Device UUID assigned to each device by the service. You can read the Device UUID at run time any time after a successful registration with the service by calling the `getDeviceUuid` method. This method returns `null` if the device is not currently registered with the PCF Push Notification Service.

Example:

```
Push.getInstance(this).startRegistration(deviceAlias, subscribedTags, areGeofencesEnabled, new RegistrationListener() {

   @Override
   public void onRegistrationComplete() {
      Log.i("MyLogTag", "Device Uuid: " + Push.getInstance(this).getDeviceUuid());
   }

   @Override
   public void onRegistrationFailed(String reason) {
      Log.e("MyLogTag", "Failed to unregister from PCF Push: " + reason);
   }
});
```

SSL Authentication

The property `pivotal.push.sslCertValidationMode` allows the app to accept the following supported SSL Authentication modes:

1. **default** : When the service URL is not HTTPS or when using a server trusted certificate this mode should be set.

2. **trustall** : When using a development environment there is the ability to trust all certificates while using a HTTPS service URL. This mode replaces the previous property (prior to v1.3.3) `pivotal.push.trustAllSslCertificates` .

3. **pinned** : To ensure no man in the middle attacks this mode should be set. The server certificate is verified with the local copy of the certificate referred to as Certificate Pinning authentication. When this mode is set the local copy of the certificate(s) should be provided with a space-separated list in the `pivotal.push.pinnedSslCertificateNames` property. All certificates provided are stored in the assets folder of the app in a **DER** format.

4. **callback** : When a custom SSL authentication schema is required this mode can be set whereby the specific authentication logic would be added inside the app as a callback to the SDK. You need to create your own implementation of a class extending the `CustomSslProvider` interface and declare it in your manifest file in a `<meta-data>` element in your `<application>` element. The name of the meta-data is “io.pivotal.android.push.CustomSslProvider” and the value of the meta-data should be the name of your custom SSL provider class (with its full package name). This class must have a default (empty) constructor and is instantiated at runtime when network requests are made to HTTPS service endpoints.

example CustomSslProvider implementation:

```
public class MyCustomSslProvider implements CustomSslProvider {

  public MyCustomSslProvider() { /* default constructor is required */ }

  @Override
  public SSLSocketFactory getSSLSocketFactory() throws NoSuchAlgorithmException, KeyManagementException {

    TrustManager[] trustAllCerts = new TrustManager[] { FILL ME IN };

    SSLContext context = SSLContext.getInstance("TLS"); // or "SSL" - please look at the Java documentation
    context.init(null, trustAllCerts, null);

    return context.getSocketFactory();
  }

  @Override
  public HostnameVerifier getHostnameVerifier() {
    return new HostnameVerifier() {
      public boolean verify(String hostname, SSLSession session) { FILL ME IN }
    };
  }
}
```

example AndroidManifest.xml:

```
<application>

  ...

  <meta-data
    android:name="io.pivotal.android.push.CustomSslProvider"
    android:value="YOUR PACKAGE NAME.MyCustomSslProvider"/>

  ...

</application>
```

Setting Custom HTTP Request Headers

In order to inject custom headers into any HTTP requests made by the Push SDK you should call the `setRequestHeaders` method in the `Push` class with a `Map<String, String>` of the required HTTP header values. Note that you can not provide any 'Authorization' or 'Content-Type' headers via this method; they are ignored by the Push SDK.

In order for this method to take effect you need to call it *before* `startRegistration`, `subscribeToTags`, or any other methods that make network requests.

Geofences

Geofences are newly supported in v1.3.0 of the Push Notification Service. Using this service, you can register push notifications that your app users see when they enter or exit certain geographic regions that you define on the Push Notification Service Dashboard.

To set up your app to receive geofence notifications, perform the following steps.

Step 1: Set Up Your AndroidManifest.xml File

Add these two permissions to the `application` element of your **AndroidManifest.xml** file.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
```

Step 2: Set Up Your Push Service

You need to override the following two methods in your app custom `Service` (see Step 7 above).

```
@Override
public void onGeofenceEnter(Bundle payload) {
   Log.i("My Log Tag", "Entered geofence " + payload.getString("message"));
      // Process geofence enter event
}

@Override
public void onGeofenceExit(Bundle payload) {
   Log.i("My Log Tag", "Exited geofence " + payload.getString("message"));
      // Process geofence exit event
}
```

Step 3: (Optional) Receive Geofence Status Updates

The PCF Push Notification Service server pushes updated geofences to user devices via push notifications. You don't need to do any more work to process these updates or monitor these geofences. You can read the geofence status object to find out if any problems occur during these background updates. These errors can be reported directly to your app if you create a `BroadcastReceiver` that listens to `io.pivotal.android.push.geofence.UPDATE` intents.

Example:

Create a class called `MyGeofenceUpdateBroadcastReceiver`:

```
public class MyGeofenceUpdateBroadcastReceiver extends BroadcastReceiver {

   @Override
   public void onReceive(Context context, Intent intent) {
      final GeofenceStatus status = Push.getInstance(context).getGeofenceStatus(); // Read geofence status
      if (status != null) {
         if (status.isError()) {
            Toast.makeText(context, status.getErrorReason(), Toast.LENGTH_LONG).show();
         }
         Toast.makeText(context, "Number of currently monitoring geofences: " + status.getNumberCurrentlyMonitoringGeofences(), Toast.LENGTH_LONG).show();
      }
   }
}
```

You can configure your `BroadcastReceiver` class to listen to geofence updates by adding the following element in your `AndroidManifest.xml`:

```
<receiver
   android:name=".MyGeofenceUpdateBroadcastReceiver" android:exported="false" >
   <intent-filter>
      <action android:name="io.pivotal.android.push.geofence.UPDATE"/>
   </intent-filter>
</receiver>
```

Step 4: Request device location permission (Android v6.0 Marshmallow and up)

Android v6.0 Marshmallow introduced a new system for obtaining user permission for "dangerous" operations. If you want to use geofences in your app then you need to request the permission to read the device location at runtime. Before Android v6.0 Marshmallow it was sufficient to simply add a `uses-permission` element to your `AndroidManifest.xml` file in order to request permission as described in Step 1 above. In Android v6.0 Marshmallow you must still add the `uses-permission` element to your `AndroidManifest.xml` file but you must also request permission from the user directly at runtime. We've added a helper method to the Push SDK to help you with this task but you still need to do some of the work yourself in your app.

In one of your app's primary `Activity` classes, you need to add the following code to your `onCreate` method BEFORE you initialize the Push SDK. The dialog box must contain a message that explains to your user why your app needs to read the device location. You may style or theme this dialog box any way that you would like to. You only need to give the dialog box one button: "OK".

```
if (ARE_GEOFENCES_ENABLED) {

    // If you want to use geofences and are targetting Android Marshmallow or greater, then you must specifically
    // ask the user for permission to read the device location.  The following Dialog class is used to explain
    // to the user why your app is requesting permission to read the device location.

    final Dialog dialog = new AlertDialog.Builder(this)
            .setMessage("This application needs permission to read the device location in order to send you notifications when you enter certain locations.")
            .setPositiveButton("OK", null)
            .create();

    final boolean werePermissionsAlreadyGranted = Push.getInstance(this).requestPermissions(this, REQUEST_PERMISSION_FOR_GEOFENCES_RESPONSE_CODE, dialog);

    if (werePermissionsAlreadyGranted) {

        // If Push.requestPermissions returns true then ACCESS_FINE_LOCATION permission has already been granted
        // and we can immediately begin push registration.

        startPushRegistrationWithGeofencesEnabled(true);
    }

} else {
    startPushRegistrationWithGeofencesEnabled(false);
}
```

If the permission to read the device location has not yet been granted, then Google shows a system dialog box to request permission. It may also show your user-defined dialog box. After the user presses “Allow” or “Deny” then Google calls the `onRequestPermissionsResult` callback in the same activity:

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {

    // This callback is invoked by Android after the user decides to allow or deny permission for ACCESS_FINE_LOCATION.
    // If Push.requestPermissions returns false then you need to wait for this callback before attempting
    // to register for pushes.

    if (requestCode == REQUEST_PERMISSION_FOR_GEOFENCES_RESPONSE_CODE && permissions[0].equals(android.Manifest.permission.ACCESS_FINE_LOCATION)) {
        if (grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            startPushRegistrationWithGeofencesEnabled(true);
        } else {
            startPushRegistrationWithGeofencesEnabled(false);
        }
    }
}
```

The `REQUEST_PERMISSION_FOR_GEOFENCES_RESPONSE_CODE` value is a unique integer that is echoed back to the `onRequestPermissionsResult` method after the user allows or denies the permission. You can select any integer that you would like.

```
// Request code when requesting permission to use geofences.
private static final int REQUEST_PERMISSION_FOR_GEOFENCES_RESPONSE_CODE = 27; // Your favourite integer
```

Step 5: Enable geofences

In order to enable geofences at runtime you need to pass `true` to the `areGeofencesEnabled` argument when you call the `startRegistration` method in your app main activity. If this parameter is set to `false` then no geofences features are available at runtime. Any geofences that may have been monitored before are cleared and are no longer monitored.

The `startPushRegistrationWithGeofencesEnabled` method in the above example will finally initialize the Push SDK. If the device location permission was not granted then you should disable geofences. Note that the user is able to allow or revoke this permission at any other time in the future. It is important to request this permission EVERY TIME you initialize your Push SDK:

```
private void startPushRegistrationWithGeofencesEnabled(boolean areGeofencesEnabled) {

    Push.getInstance(this).startRegistration(DEVICE_ALIAS, TAGS, areGeofencesEnabled, new RegistrationListener() {

        @Override
        public void onRegistrationComplete() {
            printMessage("Registration successful.");
        }

        @Override
        public void onRegistrationFailed(String reason) {
            printMessage("Registration failed. Reason: " + reason);
        }
    });
}
```

Appendix

Google Developers Console

1. Log into Google Developers Console. You need a Google account.
2. Click **Create Project**.
3. Enter a **Project Name** and leave the auto-generated **Project ID** field untouched. Click **Create**.
4. Wait until the project is completed, this might take a couple of minutes. After this, you are on the project page.
5. Note at the top your **Project Number**. This value should be in light gray text. Make note of this value because you need it later. Make sure you use the numeric project number. Do not use the project ID with the words.
6. On the left, in the **APIs & Auth** section, click **APIs**.
7. In the **Browse APIs** field, enter **Google Cloud Messaging** and ensure that **Google Cloud Messaging for Android** is enabled by clicking **Enable API**.
8. On the left click the **Credentials** link which is directly below the **APIs** link.
9. Find **Public API Access** on the page and click the **Create new Key** button below. Click **Server key** when the dialog pops up.
10. In the text field inside the dialog box enter `0.0.0.0/0` and click the **Create** button.
11. Make note of the **API KEY** value because you need it later.

Troubleshooting

See Troubleshooting.

Setting up Push Notifications with FCM

This document describes how developers can set up the Pivotal Cloud Foundry (PCF) Push Notification Service with the Firebase Cloud Messaging (FCM) platform so their apps can send push notifications to Android devices.

Prerequisites

The procedures in this document require the following:

- You must have access to a PCF environment with the Push Notification Service installed.
- You must have Android Studio 2.2 or later installed on your machine.
- You must have the Google Repository from the Android SDK Manager.
- You must have the Push Android SDK 1.7 or later from Github.
- The devices that you want to send push notifications to must run Android 2.3 (Gingerbread) or later.
- The devices that you want to send push notifications to must have Google Play Services 9.8.0 or later.

Prepare an FCM Project

Follow these steps to prepare an FCM project for your app.

1. Navigate to the Firebase Console and create an account if you do not have one already.
2. Once logged in, **Create** or **Import** a project you want to use with FCM.
    a. When prompted, click **Add Firebase to your Android app**.
    b. Enter a **Package name** that matches the ID of your app:
        - For the push-sample app, the ID is `io.pivotal.android.push.sample`.
        - For the push-demo app, the ID is `io.pivotal.android.push.demo`.
    c. Ensure the **Debug signing certificate SHA-1** matches the SHA-1 from your debug signing certificate. For instructions on how to get this fingerprint, refer to Authenticating Your Client in the Google APIs for Android documentation.
    d. After you finish creating or importing your project, a `google-services.json` file downloads. Keep track of this file for later use.
3. Click your project.
4. Click the settings icon next to your project name and select **Project Settings**.
5. Select the **Cloud Messaging** tab.
6. Record the **Server key** for later use.

Configure Your Push Dashboard

Follow the steps below to navigate to the Push dashboard and configure the Push Notification service.

You can navigate to the Push dashboard using either Apps Manager or the Cloud Foundry Command Line Interface (cf CLI). Use the cf CLI instructions if you did not enable the **Push Apps Manager** errand when deploying Elastic Runtime.

Navigate to Push Dashboard using Apps Manager

1. In a browser, navigate to `apps.YOUR-SYSTEM-DOMAIN`.
2. Select the **system** org and the **push-notifications** space.
3. Click the **Services** tab.
4. Select the **PCF Push Notification Service** row and click the **Manage** link.

Navigate to Push Dashboard using cf CLI

1. Open a terminal window and log in:

```
$ cf login -a https://api.YOUR-SYSTEM-DOMAIN -u USERNAME -p PASSWORD
```

2. Target the correct org and space:

```
$ cf target -o system -s push-notifications
```

3. Run the following command:

```
$ cf service push-service-instance
```

4. Copy the URL from the **Dashboard** field and paste it into your browser.

Configure the Push Notification Service

Follow these steps to configure the Push backend by creating a new platform for the sample app.

1. In the Push dashboard, select the **+** icon from the left to create a new app to send push notifications to, either the push sample app or push demo app.
   - Enter a **Name** and **Description**.
2. Once you create an app, select the **Configuration** tab for that app.
3. Click **Add New Platform**.
4. Enter a **Name** and **Description**, and choose a **Mode**.
5. For **Type**, select `Android-FCM`.
6. Once created, click the pencil icon to edit the platform.
7. In the **Google Key** field, paste the server key that you recorded earlier.

Note: You can add multiple FCM Platforms with server keys from different FCM projects, depending on how your FCM applications and projects are organized. There is no requirement that all FCM Platforms use the same server key in the Push backend.

Run the App on Your Device

Follow these steps to compile and deploy the app on your Android device.

1. Navigate to the Push Android Samples repository.
2. Clone the repository to your workspace.
3. Checkout the `release-v1.7.0` branch, or the branch of a later version.
4. Copy the `google-services.json` file from earlier into your app project:
   - If you want to compile the sample app, copy the `json` file to the `push-sample` subdirectory of the app project.
   - If you want to compile the demo app, copy the `json` file to the `push-demo` subdirectory.
5. Open a project in Android Studio using the repo you cloned.
6. Open the `pivotal.properties` file.
   - For the sample app, you can find this file in `push-sample/src/main/res/raw/`.
   - For the demo app, you can find this file in `push-demo/src/main/assets/`.
7. Update the file using the following guidance:
   - `pivotal.push.platformUuid`: This value must match the platform **UUID** of the FCM Platform you created in the previous section.

- `pivotal.push.platformSecret`: This value must match the platform **SECRET** of the FCM Platform you created in the previous step.
- `pivotal.push.serviceUrl`: Enter the server address to your push backend API in the form of `https://push-api.YOUR-SYSTEM-DOMAIN`.

8. Compile and deploy the application to your Android device.

Once the application registers with the Push backend, it can receive push notifications. To verify that your device registered, see the **Devices** tab in the Push dashboard. The device **Type** field displays a Firebase logo.

You can also send test pushes to the device from the Push dashboard.

> **Note**: If you send a test push to your device from the Push dashboard, ensure the app is not open on your device. You cannot see the test push while the app is open.

Windows Phone 8.1 Push Client SDK

Sample App

You can find the Windows Phone Sample App on github

Features

The Pivotal Cloud Foundry Mobile Services Push Client SDK is a light-weight library that will help your application register with Windows Notification Service (WNS) and an instance of the PCF Mobile Services Push Notification server.

The SDK does not provide any code for handling remote push notification.

Device Requirements

The Push SDK requires WP8.1.

Getting Started

In order to receive push messages from the Push Server in your WP8.1 application, you will need to follow these steps:

1. You must have a Windows Store account, and your app must be registered with the windows store, and must you have a certificate for receiving pushes. Follow the instructions here:

http://msdn.microsoft.com/en-us/library/windows/apps/hh465407.aspx

Finally, you need to associate your app with the windows store. Right click on your project in Visual Studio. Select Store. Select 'Associate App with the Store…' Follow Microsoft's on-screen instructions to complete app-to-store association.

1. Set up your application, environment, and a variant on the PCF Mobile Services administration console. This task is beyond the scope of this document, but please note you will need the TLS certificate above. After setting up your variant in PCF Mobile Services, make sure to note the Variant UUID and Variant Secret parameters. You will need them below.

2. Add a reference to the SDK NuGet package to your project. If you don't have access to the NuGet, you could simply obtain the compiled DLL files. Please contact the PCF Mobile Services team for help.

3. Add the following lines of code to the initialization section of your application.

```
MSSParameters parameters = new MSSParameters(variant_uuid, variant_secret, base_server_url, server_name, device_alias, tags);
await MSSPush.SharedInstance.RegisterForPushAsync(parameters);
```

The `variant_uuid`, `variant_secret`, and `server_name` are described above. The `base_server_url` parameter is the base url of your push server. The `device_alias` parameter is a custom field that you can use to differentiate this device from others in your own push messaging campaigns; can be null. The `tags` parameter is a custom field that you can use to differentiate registrations in your own push messaging campaigns; can be null.

You should only have to call `RegisterForPushAsync` once in the lifetime of your process - but calling it more times is not harmful. The `RegisterForPushAsync` method is asynchronous and will return before registration is complete. To know when registration is complete (or if it fails) and obtain the `PushNotificationChannel`, provide an `Action<PushNotificationChannel>` as the second argument.

The PCF Mobile Services Push SDK takes care of the following tasks for you:

- Registering with WNS.
- Sending your push notification channel URI to the back-end (i.e.: the PCF Mobile Services).
- Re-registering after the push notification channel URI or any other registration parameters are updated.

APIs

You can use the following APIs for the Push Notification Service:

- Push API
- Registration API
- Registrations API
- Topics API
- Custom User IDs API
- Schedules API
- Geofences API

Push

Push a message

POST /v1/push

Push a message out to a list of devices or devices targeted by platform.

Authentication: HTTP Basic application_uuid:api_key

Query Parameters: None

Request Body:

There are many possible options for the request body. All of the options are listed in the JSON text example below. Note that most of the individual JSON fields are optional. The options you need to use are described below. Several examples are illustrated below.

The **message → body** field in the JSON request body is the *default* message that is supplied in notifications to remote devices. It will be overridden by any platform-specific **custom** message body data.

In particular, iOS devices will receive the **message → body** field as their alert message unless the **custom → ios → alert → body** field is populated. Android devices will receive the **message → body** field in their payload **message** field unless the **custom → android → message** field is populated.

Response Data, status: 200 (OK)

The fields returned by the /v1/push POST API depend on the type of push notification that was requested:*scheduled* or *immediate*.

If the push is given **scheduleAt** or **scheduleIn** fields then the push is *scheduled* to be delivered in the future. These pushes will return a **schedule_id** field in the response data. These **schedule_id** values can be used in the /v1/schedule APIs to update or cancel the scheduled push before it is delivered.

Otherwise the push will be queued to be sent *immediately*. In this case, the response will also contain a **receipt_id** field that can be used to follow the push notification delivery status in the audit logs.

Response Data

```
{
 "schedule_id": "",  # only returned if the push notification is a scheduled push
 "receipt_id": ""    # only returned if the push is being delivered immediately.
}
```

Scheduled pushes are described further below.

Targeting / Audience Selection

You can target your push notifications in many ways:

- By platform(s). e.g.: “ios”, “android”.
- By device UUID(s):
  - Devices UUIDs are determined by:
    - the device registration process in the PCF Push Client SDKs. See the documentation for theiOS or Android SDKs.
    - or by the /v1/registration POST/PUT APIs if you are registering your devices without using the Client SDKs.
- By topic(s):

- Topics are created:
  - implicitly when devices subscribed to them. See the/v1/registration POST/PUT APIs.
  - or when the /v1/topics POST API is used.

- By Custom User ID:
  - Custom User IDs are created via the Register a device API/v1/registration POST APIs.
  - Sending a push to a Custom User ID:
    - the push will be sent simultaneously to all devices registered with this Custom User ID.

- By Custom User ID and Topic(s):
  - Sending a push to a Custom User ID and Topic(s):
    - the push will be sent simultaneously to all devices registered with this Custom User ID and all devices registered with any of the provided Topic(s).

Key	Description
devices	A list of up to 4096 device UUIDs to target. These device UUIDs are the same ones that are returned by the PCF Push Client SDKs after registration or by the **/v1/registration** HTTP POST call if you are registering your devices without using the Client SDKs.
topics	A list of up 1024 topics (formerly tags) to which devices may be subscribed. Only devices subscribed to one of more of the listed topics will be targeted. Devices select which topics to subscribe to by calling the appropriate **subscribeToTopics** methods in the client SDKs or by calling the /v1/registration HTTP POST or PUT.
platforms	A list of platforms to be targeted. Available platforms are 'ios', 'android', 'windows8', 'windowsPhone', 'bb10' (if enabled).
platform	DEPRECATED. Possible values are 'all', 'ios', 'android', 'windows8', 'windowsPhone', 'bb10' (if enabled). If 'platforms' is also populated the platform(s) selected here will be added to list of platforms.
interactive-only	If set to **true** then only those devices that can accept interactive pushes are targetted. At this time only iOS 8+ or Android 4.1+ devices are considered to support interactive pushes.
custom_user_ids	A list of IDs for devices that is meaningful to your system, such as their login. The same Custom User ID can be used to refer to multiple devices.

LIMITS

Pushing to multiple targets is bounded by the following limits per request:

- Devices: 4096
- Custom User Ids: 4096
- Topics: 1024

NOTES

- At least one of **devices**, **topics**, **platforms**, or **platform** is required.
- **devices** will override any other targeting type. Any **topics**, **platforms**, or **platform** targetting key will be ignored if there is a **devices** key.
- **topics** and **platforms** can be used in a complementary way to push a message to just a subset of users (See example below).
- Devices only need to be subscribed to at least one of the topics in the targetting data in order to receive the message (See example below). There is no way using the Push API to send a message to a device that is subscribed to *all* of the topics in a list.

Target Examples

- Sending push messages to three specific devices (specified by their device UUIDs):

```
{
 ...
  "target": {
   "devices": [ "device_uuid1", "device_uuid2", "device_uuid3" ]
  }
 ...
}
```

- Sending pushes to all devices (regardless of platform) subscribed to one of two specific topics (a device only needs to be subscribed to one of the topics in the list of topics in order to receive the message).

```
{
 ...
 "target": {
  "topics": [ "exciting_topic", "pedantic_topic" ]
 }
 ...
}
```

- Sending pushes to all iOS devices:

```
{
 ...
 "target": {
  "platforms": [ "ios" ]
 }
 ...
}
```

- Sending pushes to all “Android” devices subscribed to one specific topic:

```
{
 ...
 "target": {
  "platforms": [ "android" ],
  "topics": [ "best_topic_ever" ]
 }
 ...
}
```

- Sending pushes to interactive only devices:

```
{
 ...
 "target": {
  "platforms": [ "android", "ios" ],
  "interactive-only": true
 }
 ...
}
```

- Sending pushes to devices registered with a Custom User ID:

```
{
 ...
 "target": {
  "custom_user_ids": [ "some_customer_user_id", "some_other_custom_user_id" ]
 }
 ...
}
```

- Sending pushes to devices registered with a Custom User ID and devices registered with Topic(s):

```
{
 ...
 "target": {
  "custom_user_ids": [ "some_customer_user_id" ],
  "topics": [ "exciting_topic", "pedantic_topic" ]
 }
 ...
}
```

- Sending pushes to devices registered with Custom User IDs and devices registered with Topic(s):

```
{
 ...
 "target": {
  "custom_user_ids": [ "some_customer_user_id1", "some_customer_user_id2" ],
  "topics": [ "exciting_topic", "pedantic_topic" ]
 }
 ...
}
```

Setting Expiration Time on Pushes

The "**expiryTime**" field can be used to specify a time after which a push should not be displayed. It should be an **Epoch** timestamp integer in milliseconds (i.e.: the number of milliseconds since midnight January 1, 1970). If **expiryTime** is not set the behavior will be the platform default. For iOS, Android and BB10 pushes will be queued for delivery if the target device is unreachable at the time of the push and delivered as soon as it is reachable. If **expiryTime** is set and the the device becomes reachable AFTER the expiry time, the push will not be delivered.

Windows 8 behavior is similar for tile and badge notifications but only one tile and one badge notification will be queued. See the Push notification service request and response headers documentation for more information.

IMPORTANT NOTE:

- If omitted, the default expiry time used for Apple devices is `Integer.MAX_VALUE` seconds (i.e.: sometime in the year 2038).
- If omitted, the default expiry time used on GCM is 4 weeks (2,419,200 seconds). The maximum time-to-live for messages delivered on GCM is also 4 weeks.
- Windows 8 toast notifications are not queued and cannot have an expiry time. They will only be delivered at the time the push is sent if the target device is connected.
- Windows Phone 7 has no expiry option. Instead pushes fall into one of three categories: immediate, up to 450 seconds, or up to 900 seconds. Depending on the time set in the "**expiryTime**" field, a push to Windows Phone 7 will be slotted into one of those categories. See the Sending push notifications for Windows Phone documentation.

Scheduled Pushes

Pushes can be scheduled to be sent at a later time. Use the **scheduleAt** field to specify the time when the push should be sent. As with **expiryTime** this should be an **Epoch** timestamp integer in milliseconds. Alternatively you can use the **scheduleIn** field to specify the scheduled time as the number of seconds from the time the server receives the push request. NOTE: You cannot set both the **scheduleAt** and **scheduleIn** fields at the same time as doing this would result in an error message from the server.

If the scheduled time is less than a preconfigured time in the future, the push will not be scheduled and will be sent immediately. By default this amount is 60 seconds.

Scheduled Pushes Examples

- Scheduling a push message to be delivered for February 2, 2016 at 8 AM (UTC):

```
{
 ...
 "scheduleAt": 1454313600000
 ...
}
```

- Scheduling a push message to be delivered two hours from now:

```
{
 ...
 "scheduleIn": 7200000
 ...
}
```

Custom Fields for Platform specific Pushes

Custom Fields for iOS Pushes

The fields available in the custom block for iOS are described here:

```
{
 "ios": {
  "alert": {
   "body": "iOS only message body",
   "action-loc-key": "actionKey",
   "loc-key": "localizedStringKey",
   "loc-args": [
    ""
   ],
   "title": "Title",
   "title-loc-key": "titleKey",
   "title-loc-args": [
    "arg1",
    "arg2"
   ],
   "launch-image": "Default.png"
  },
  "category": "SAMPLE_CATEGORY",
  "badge": 1,
  "sound": "default",
  "content-available": true,          # Note - the Push API expects this field to be a boolean. (see below)
  "extra": { "freeform custom data" : "freeform custom data", ... }
 }
}
```

- **extra**
  type: dictionary or null
  This property can be used to pass free-form arbitrary payload data to the receiving iOS device. This data will be passed in the `userInfo` dictionary in the `application:didReceiveRemoteNotification` callback in the application's app delegate class. It is up to the application to use this data as it needs.
- **alert**
  type: string or dictionary
  If this property is included, the system displays a standard alert. You may specify a string as the value of alert or a dictionary as its value. If you specify a string, it becomes the message text of an alert with two buttons: Close and View. If the user taps View, the app is launched. Alternatively, you can specify a dictionary as the value of alert. See Table 3-2 at the Apple Documentation for descriptions of the keys of this dictionary.
- **badge**
  type: number
  The number to display as the badge of the app icon. If this property is absent the badge is not changed. To remove the badge, set the value of this property to 0.
- **sound**
  type: string
  The name of a sound file in the app bundle. The sound in this file is played as an alert. If the sound file doesn't exist or default is specified as the value, the default alert sound is played. The audio must be in one of the audio data formats that are compatible with system sounds; see Preparing Custom Alert Sounds for details.
- **content-available**
  type: boolean
  Provide this key with a value of `true` to indicate that new content is available. Including this key and value means that when your app is launched in the background or resumed, `application:didReceiveRemoteNotification:fetchCompletionHandler:` is called. (Newsstand apps are guaranteed to be able to receive at least one push with this key per 24-hour window). The Push API will translate the value of this field to `1` or `0` before sending it to APNS.
- **title**
  type: string
  A short string describing the purpose of the notification. This field was introduced on Apple Watch but is also displayed on iOS devices as of iOS version 10.0. This key was added in iOS 8.2.
- **body**
  type: string
  The text of the alert message.
- **title-loc-key**
  type: string or null
  The key to a title string in the "Localizable.strings" file for the current localization. The key string can be formatted with `%@` and `%n$@` specifiers to take the variables specified in the **title-loc-args** array. See Localized Formatted Strings for more information. This key was added in iOS 8.2.
- **title-loc-args**
  type: array of strings or null
  Variable string values to appear in place of the format specifiers in title-loc-key. See Localized Formatted Strings for more information. This key was added in iOS 8.2.
- **action-loc-key**
  type: string or null
  If a string is specified, the system displays an alert that includes the Close and View buttons. The string is used as a key to get a localized string in the current localization to use for the right button's title instead of "View". See Localized Formatted Strings for more information.

- **loc-key**
  type: string
  A key to an alert-message string in a “Localizable.strings” file for the current localization (which is set by the user’s language preference). The key string can be formatted with `%@` and `%n$@` specifiers to take the variables specified in the loc-args array. SeeLocalized Formatted Strings for more information.
- **loc-args**
  type: array of strings
  Variable string values to appear in place of the format specifiers in **loc-key**. See Localized Formatted Strings for more information.
- **launch-image**
  type: string
  The filename of an image file in the app bundle; it may include the extension or omit it. The image is used as the launch image when users tap the action button or move the action slider. If this property is not specified then the system either uses the previous snapshot or uses the image identified by the **UILaunchImageFile** key in the app’s “Info.plist” file, or falls back to “Default.png”. This property was added in iOS 4.0.

Please check the Apple documentation for more detailed information.

Custom Fields for Android Pushes

The custom fields for android are a dictionary that can contain any fields required by your application. You can also specify a**collapse_key** in the custom fields for Android. A message with a **collapse_key** that has not yet been delivered may be replaced by a newer message with the same collapse key. Please see the Google documentation on collapsable messages.

Otherwise, you can specify any arbitrary freeform payload data to deliver to the receiving Android device. All of the fields in this in the **android** element in the push request will be supplied to the receiving Android device in the `Bundle` provided to `onReceiveMessage` method in the Android application’s subclass of `GcmService`. In general the push message data would be provided in a `message` JSON field but it is up to your application to use the message payload as it needs.

Note that the “message” → “body” field in the Push request body, if present, will be delivered in the “message” field of the GCM push notification payload.

Custom Fields for Other Platforms

BB10

The **bb10** custom field consists of a string which will override the message body on BlackBerry 10. Note that PCF Push Notification Service support for Windows 8 is basic. Pivotal does not provide any up-to-date client SDKs for BlackBerry 10 at this time.

NOTE: The PCF Push Notification Service does not ship with BB10 support enabled due to restrictions on redistribution of BlackBerry’s push libraries. If you need to enable BB10 pushes please contact Pivotal Support.

Windows 8 (WNS)

See this reference for a description of Windows 8 push options. Note that PCF Push Notification Service support for Windows 8 is basic. Pivotal does not provide any up-to-date client SDKs for Windows 8 at this time.

Windows Phone (MPNS)

See this reference for a description of Windows Phone push options. Note that PCF Push Notification Service support for Windows 8 is basic. Pivotal does not provide any up-to-date client SDKs for Windows Phone at this time.

Complete Examples

Unlike the above examples, these examples will show the complete Push request body.

- Send a message to all users subscribed to the “local_seminars” topics to alert them to an important community meeting. This message expires on the morning of Friday April 1, 2016.

```
{
 "message": {
  "body": "Town Hall This Thursday: Forging, Cheese, And You"
 },
 "target": {
  "topics": [ "local_seminars" ]
 },
 "expiryTime": 1459468800000
}
```

- Send a push to all iOS and Android devices that are subscribed to one (or more) of the “breaking_news”, “local”, or “dairy” topics. Provide some custom fields that apps can use to deep link to article data. This message is scheduled to be delivered in two hours.

```
{
 "message": {
  "custom": {
   "ios": {
    "alert": {
     "body": "Breaking News: World's Biggest Cheese Forged At Local Bakery"
    },
    "content-available": true,
    "extra": { "story_url": "https://my_server/article/123456789" }
   },
   "android": {
    "message": "Breaking News: World's Biggest Cheese Forged At Local Bakery",
    "story_url": "https://my_server/article/123456789"
   }
   }
 },
 "target": {
  "topics": [ "breaking_news", "local", "dairy" ],
  "platforms": [ "ios", "android" ]
 },
 "scheduleIn": 7200
}
```

- Send a push to one particular device informating the user that they have one new email notification. The badge on the app icon will be set to “1” and a sound will be played. Some of the email metadata is provided in the message extras so that the application can show a preview of the message. The message is given the “new_email” category so that iOS 8.0+ devices can provide appropriate action buttons for the user.

```
{
 "message": {
  "custom": {
   "ios": {
    "alert": {
     "body": "You've got mail!"
    },
    "category": "new_email",
    "badge": 1,
    "sound": "new_email",
    "content-available": true,
    "extra": {
     "from": "Your Local Bakery",
     "to": "You",
     "subject": "Special Deal on Cheese",
     "message_body": "Please come to your local bakery before Friday to sample a piece of the world's biggest cheese."
    }
   }
  }
 },
 "target": {
  "devices": [ "111-222-333444" ]
 }
}
```

- All options in request body for pushing a message out to a list of devices or devices targeted by platform.

```
{
 "message": {
  "body": "Message body",                # The text of the push message
  "custom": {
   "ios": {
    "alert": {
     "body": "iOS only message body",   # The body of the push message
     "action-loc-key": "actionKey",     # (overrides body defined above)
     "loc-key": "localizedStringKey",
     "loc-args": [ "arg1", "arg2", ... ],
     "title": "Title",
     "title-loc-key": "titleKey",
     "title-loc-args": [ "arg1", "arg2", ... ],
     "launch-image": "Default.png"
    },
    "category": "SAMPLE_CATEGORY",
    "badge": 1,
    "sound": "default",
    "content-available": true,          # Note - the Push API expects this field to be a boolean. (see below)
    "extra": {}
   },
   "android": {
    "collapse_key": "collapseKey"
   },
   "windows8": {
    "template_name": "TileSquareBlock",
    "template_fields": {
     "textField1":"text1",
     "textField2":"text2"
    },
    "options": {},
    "type": "tile"
   },
   "windowsPhone": {
    "template_fields": {
     "subtitle":"text",
     "parameter":"text"
    },
    "template": "toast"
   },
   "bb10": "BB10 only message body"
  }
 },
 "target": {
  "topics": [ "topic1", "topic2", ... ],
  "platforms": [ "platform1", "platform2", ... ],
  "devices": [ "device_uuid1", "device_uuid2", ... ],
  "interactive-only": false,     # Either true or false
  "platform": "all"              # One of the following options
 },                              # (ios, android, windows8, windowsPhone, bb10)

 "scheduleAt": 1345852800000,    # Epoch timestamp in milliseconds.
 "scheduleIn": 0,                # Integer (time delta in seconds)
 "expiryTime": null              # Epoch timestamp in milliseconds.
}
```

Registration

GET /v1/registration/:deviceUuid

Retrieves a device's registration for a specific platform.

Authentication: HTTP Basic platform_uuid:platform_secret

Query Parameters: None

Response Data, status: 200 (OK)

```
{
 "os": "",                  # one of [ios|android|windowsPhone|windows8]
 "device_model": "",          # device model identifier
 "device_manufacturer": "",   # device manufacturer identifier
 "device_alias": "",          # application specific device/user identifier
 "device_uuid": "",           # unique device identifier
 "registration_token": "",    # token provided by APNS (ios), GCM (android), MPNS (windowsPhone), or WNS (windows8)
 "tags": [                  # tags the device/user is subscribed to, this will overwrite any existing tags the device/user was previously subscribed to
  {
   "text": ""
  }
 ],
 "active": "",              # can the device be targeted for pushes
 "os_version": ""           # device version string
}
```

GET /v1/registration/count/

Returns the total number of device registrations that have been stored for one platform.

Authentication: HTTP Basic platform_uuid:platform_secret

Query Parameters: None

Response Data, status: 200 (OK)

Returns an integer.

POST /v1/registration/

Register a device to an app release. The response will include `device_uuid`. You should save this identifier, as other registration endpoints will require it (ex. `DELETE`).

When the environment variable `push_security_verifyCustomUserId` is set to true (which is default), creating a registration with a custom_user_id, it is required that the custom_user_id is encrypted with a unique HMAC using the device shared secret as the cryptigraphic key.

For more information see Registering with a Custom User ID.

Authentication: HTTP Basic platform_uuid:platform_secret

Query Parameters: None

Request Body:

```
{
 "device_alias": "string",             # application specific device/user identifier. We recommend that you use the user's device name as device alias
 "device_model": "string",             # device model identifier
 "device_manufacturer": "string",      # device manufacturer identifier
 "os": "string",                       # device os, one of [ios|android|windowsPhone|windows8]
 "os_version": "string",               # device version string
 "registration_token": "string",       # token provided by APNS (ios), GCM (android), MPNS (windowsPhone), or WNS (windows8)
 "tags": [ "tag1", "tag2" ],           # tags the device/user is subscribed to, this will overwrite any existing tags the device/user was previously subscribed to
 "custom_user_id": "string"            # allows you to register a device under an ID that is meaningful to your system such as their login
}
```

Response Data, status: 200 (OK)

```
{
 "os_version": "",          # os version string
 "tags": [                  # tags that the device has subscribed to
   {
    "text": "tag1"
   },
   {
    "text": "tag2"
   }
 ],
 "os": "",                  # one of [ios|android|windowsPhone|windows8]
 "device_model": "",        # device model identifier
 "device_manufacturer": "", # device manufacturer identifier
 "device_alias": "",        # application specific device/user identifier
 "device_uuid": "",         # the unique identifier assigned to the device by Push Notifications
 "registration_token": "",  # token provided by APNS (ios), GCM (android), MPNS (windowsPhone), or WPN (windows8)
 "active": "",              # can the device be targeted for pushes
 "custom_user_id": ""       # device registered with custom user id
}
```

LIMITS

Registering a device is bounded by the following limits per request:

- Devices: Auto-Generated
- Custom User Ids: 1
- Tags: 1024

Examples:

- Register a device:

```
{
  "device_alias": "John's iPhone",
  "device_model": "iPhone 6",
  "device_manufacturer": "Apple",
  "os": "ios",
  "os_version": "9.0",
  "registration_token": "b50edac575bfba07dd019b28b2af7189a3ddda17c806ef14a9abbfd00533f67e",
  "tags": [ "beta", "gamma", "alpha" ],
  "custom_user_id": "jsmith"
}
```

PUT /v1/registration/:device_uuid

Update a registration. Requires that the device_uuid returned when you registered is sent as a url parameter.

When the environment variable `push_security_verifyCustomUserId` is set to true (which is default), updating a registration with a custom_user_id, it is required that the custom_user_id is encrypted with a unique HMAC using the device shared secret as the cryptigraphic key.

Authentication: HTTP Basic platform_uuid:platform_secret

Query Parameters: None

Request Body:

```
{
 "device_alias": "string",           # application specific device/user identifier. We recommend that you use the user's device name as device alias.
 "device_manufacturer": "string",    # device manufacturer identifier
 "device_model": "string",           # device model identifier
 "os_version": "string",             # os version string
 "registration_token": "string",     # token provided by APNS (ios), GCM (android), MPNS (windowsPhone), or WPN (windows8)
 "tags": {
  "subscribe": ["tag1","tag2"],     # add new tags subscriptions to the device/user
  "unsubscribe": ["tag3","tag4"]    # remove tags that the device/user is subscribed to
 },
 "custom_user_id": "string"          # allows you to register a device under an ID that is meaningful to your system such as their login
}
```

Examples:

- Update device registration:

```
 {
  "device_alias": "John Smith's iPhone",
  "device_model": "iPhone 6",
  "device_manufacturer": "Apple",
  "os": "ios",
  "os_version": "9.0",
  "registration_token": "b50edac575bfba07dd019b28b2af7189a3ddda17c806ef14a9abbfd00533f67e",
  "tags": [ "beta", "gamma", "alpha", "delta" ],
  "custom_user_id": "john.smith"
 }
```

DELETE /v1/registration/:device_uuid

Delete a registration. Requires that the device_uuid returned when you registered is sent as a url parameter

Authentication: HTTP Basic platform_uuid:platform_secret

Query Parameters: None

Request Body:

None.

Response Data, status: 204 (NO CONTENT)

Registrations

API calls for the `v1/registration/` endpoint can be found here.

GET /v2/registrations/

Retrieves all device registrations.

Authentication: HTTP Basic app_uuid:api_key

Query Parameters:

Parameters	Description
size	Controls the maximum number of registrations to be returned. This value defaults to 20 if not provided. Values in the range 0-50 are accepted.
page	Controls which page of results will be returned with an offset of size page. This value defaults to 1 if not provided.
q	Returns only the registrations results containing the query string provided in either the deviceUuid, the registration token, the custom user id, or the device alias.
platform	Returns only the registrations results registered to the given platform. Valid inputs are all, ios, android, windows8, and bb10.
platformUuid	Returns only the registrations results registered to the given platformUuid.
topic	Returns only the registrations results registered to the given topic name.

Response Data, status: 200 (OK)

```
{
 "registrations":
 [
   {
    "os": "",                  # one of [ios|android|windowsPhone|windows8]
    "os_version": "",          # device version string
    "device_model": "",        # device model identifier
    "device_manufacturer": "",  # device manufacturer identifier
    "device_alias": "",        # application specific device/user identifier
    "device_uuid": "",         # unique device identifier
    "registration_token": "",   # token provided by APNS (ios), GCM (android), MPNS (windowsPhone), or WNS (windows8)
    "active": ""               # can the device be targeted for pushes
   }
 ]
 "totalRegistrations": 1        # the total number of device registrations for all pages
 "totalPages": 1                # the number of pages of device registrations
 "page": 1                      # the page of results requested
 "size": 1                      # the size of the page of results requested
}
```

Topics

A Topic (formerly a Tag) is a keyword that users can subscribe to in order to receive pushes sent to the same topic.
The topics themselves are free-form, that is, your app defines them as needed and they can be any text that your app needs.

GET /v2/topics

Returns all non-expired topics.

Authentication: HTTP Basic app_uuid:api_key

Query Parameters:

Parameter	Description
q: string	Optional — Match all topics that contain the string. Default match all non-expired topics.
size: integer	Optional — Maximum number of topics to return. Range between 1 and 50. Default set to 20.
page: integer	Optional — Page number to return set of topics. Default set to 1.
hasExpiry: boolean	Optional — If set to true, filter results to topics that have an expiry. If false, filter results to topics with no expiry. If missing, no filtering is done, all resulting topics are returned. Default returns all resulting topics.

Response Data, status: 200 (OK)

Returns a json list of topics.

For example:

```
{
   "topics": list,             // List of topic objects that match the request
   "totalTopics": integer,     // Total number of topics that match the request
   "totalPages": integer,      // Total number of pages of topic results
   "page": integer,            // Current page returned. Same as page in request
   "size": integer,            // Current size of page. Same as size in request
}

// Topic Object

{
   "id": integer,          // Unique ID of the topic
   "name": string,         // Topic name
   "expireAt": long        // Optional - Epoch time, in ms, of when topic will expire. If missing, topic will not expire.
}
```

POST /v2/topics/

Creates a topic, if not already created, with an optional expiry time.

Authentication: HTTP Basic app_uuid:api_key

Query Parameters: None

Request Body:

```
{
   "name": string,         // Name of the topic to create
   "expireAt": long,       // Optional - Expiry time of the topic, in Unix epoch time in ms
   "timeToLive":long       // Optional - Duration, in seconds, before expiring the topic. Must be at least 60 seconds.
}
```

Note: Either expireAt or timeToLove may be present, not both. If both expireAt and timeToLive are missing, then the topic will never expire.

Response: status: 201 (CREATED)

```
{
   "id": integer,          // Unique ID of the topic
   "name": string,         // Topic name
   "expireAt": long,       // Optional - Epoch time, in ms, of when topic will expire. If missing, topic will not expire.
}
```

DELETE /v2/topics/:topicId

Deletes a non-expired topic, defined by its topic ID.

Authentication: HTTP Basic app_uuid:api_key

Query Parameters: None

Request Body:

None.

Response Data, status: 204 (NO CONTENT)

POST /v2/topics/batch/

Creates multiple topics in one batch.

Authentication: HTTP Basic app_uuid:api_key

Query Parameters: None.

Request Body:

```
{
   "topics": list,             // List of topic objects to create. Maximum size is 1024
   "returnTopics": boolean     // Optional - If true, the response will return the list of created topics. If false, only the count will be returned. Defaults to false.
}

//Topic object

{
   "name": string,         // Name of the topic to create
   "expireAt": long,       // Optional - Expiry time of the topic, in Unix epoch time in ms
   "timeToLive":long       // Optional - Duration, in seconds, before expiring the topic. Must be at least 60 seconds.
}
```

Note: Either expireAt or timeToLive may be present, not both. If both expireAt and timeToLive are missing, then the topic will never expire.

Response: status: 201 (CREATED)

```
{
   "numTopicsCreated": integer,      // Number of newly created topics
   "numTopicsExisted": integer,      // Number of topics that already existed from requests.
   "topics": list                    // List of topics added. Not present if "returnTopics" in the request is false.
}

// Topic Object

{
   "id": integer,          // Unique ID of the topic
   "name": string,         // Topic name
   "expireAt": long        // Optional - Epoch time, in ms, of when topic will expire. If missing, topic will not expire.
}
```

DELETE /v2/topics/batch

Delete multiple topics in one batch.

Authentication: HTTP Basic app_uuid:api_key

Query Parameters: None.

Request Body:

```
{
   "topicIds": list       // List of topic ids (integer)
}
```

Response: status: 200 (OK)

```
{
   "numTopicsDeleted": integer     // Number of topics deleted
}
```

Custom User IDs

The Custom User ID feature allows you to register a device under an ID that is meaningful to your system such as their login. In addition, the same Custom User ID can be used to refer to multiple devices. This means that a push sent to the Custom User ID will be sent simultaneously to all devices registered with this Custom User ID.

Note: The Custom User ID field is case sensitive for device registrations.

Custom User ID and Topics

Custom User ID works in combination with topics so that you can target a set of Custom User IDs as well as topics and the Push Notification Service will ensure that all devices receive only 1 copy of the notification.

GET /v2/custom_user_ids

Get a list of Custom User IDs

Authentication: HTTP Basic app_uuid:api_key

Query Parameters: None

Response Body:

```
{
  "custom_user_ids": [
    "string 1",
    "string 2"
  ]
}
```

Examples:

- Retrive a list of all Custom User IDs:

```
{
  "custom_user_ids": [
    "custom-user-id1"
  ]
}
```

GET /v2/custom_user_ids?q={query}

Get Custom User IDs by Query Parameter

Authentication: HTTP Basic app_uuid:api_key

Query Parameters:

Parameter	Description
q	Returns only the Custom User IDs results containing the query string provided.

Response Body:

```
{
 "custom_user_ids": [
   "string 1",
   "string 2"
 ]
}
```

Examples:

- Retrive Custom User IDs by query parameter (i.e. query parameter is ‘id1’):

```
{
 "custom_user_ids": [
   "custom-user-id1"
 ]
}
```

Note: In order to use the Custom User IDs feature, you will have to register a device using the POST method on `/v1/registration` endpoint, with a `custom_user_id` field populated as described in Register section of the Registration API.

```
{
 ...
 "custom_user_id": "custom-user-id1"
 ...
}
```

For additional information regarding Registration, please consult the Registration API section of our API.

Schedule

This document decscibes the endpoints for managing scheduled pushes.

Pushes can be scheduled for delivery in the future by providing the schedule information in the /v1/push POST API. These pushes return a **schedule_id** field that can be used as the identify for the /v1/schedule APIs that are descibed below.

GET /v1/schedules

Get all scheduled pushes for an application.

Authentication: HTTP Basic app_uuid:api_key

Query Parameters: None

Request Body:

None.

Response Data, status: 200 (OK)

```
[
  {
    "schedule_id": "fc226fbc1443ebfe",
    "scheduled_for": 1423513994000,    # Epoch Timestamp in milliseconds
    "push": {
      "scheduleAt": 1423513994000,     # Epoch Timestamp in milliseconds
      "scheduleIn": 0,
      "expiryTime": null,
      "message": {
        "custom": {
          "windows8": {
            "options": "object",
            "template_name": "",
            "template_fields": "object",
            "type": ""
          },
          "windowsPhone": {
            "template_fields": "object",
            "template": ""
          },
          "ios": {
            "alert": {
                "body": "",          # The body of the push message (overrides body defined above)
                "action-loc-key": "",
                "loc-key": "",
                "loc-args": [ "arg1", "arg2", ... ],
                "title": "",
                "title-loc-key": "",
                "title-loc-args": [ "arg1", "arg2", ... ],
                "launch-image": ""
            },
            "category": "",
            "badge": 0,
            "sound": "",
            "content-available": false,
            "extra": {}
          },
          "bb10": "",
          "android": "object"
        },
        "body": ""
      },
      "target": {
        "topics": [ "topic1", "topics2", ... ],
        "platforms": [ "platform1", "platform2", ... ],
        "devices": [ "device_uuid1", "device_uuid2", ... ],
        "interactive-only": false,
        "platform": "",
      }
    }
  }
]
```

GET /v1/schedules/:schedule_id

Get a single scheduled push for an application.

Authentication: HTTP Basic app_uuid:api_key

Query Parameters: None

Request Body:

None.

Response Data, status: 200 (OK)

```
{
 "schedule_id": "fc226fbc1443ebfe",
 "scheduled_for": 1423513994000,                  # Epoch Timestamp in milliseconds
 "push": {
  "scheduleAt": 1423513994000,                  # Epoch Timestamp in milliseconds
  "scheduleIn": 0,
  "expiryTime": null,
  "message": {
   "custom": {
    "windows8": {
     "options": "object",
     "template_name": "",
     "template_fields": "object",
     "type": ""
    },
    "windowsPhone": {
     "template_fields": "object",
     "template": ""
    },
    "ios": {
     "extra": "object",
     "category": "",
     "badge": 0,
     "sound": "",
     "content-available": false,
     "alert": {
      "body": "",
      "loc-key": "",
      "action-loc-key": "",
      "loc-args": [ "arg1", "arg2", ... ],
      "launch-image": ""
     }
    },
    "bb10": "",
    "android": "object"
   },
   "body": ""
  },
  "target": {
   "interactive-only": false,
   "platform": "",
   "topics": [ "topic1", "topic2", ... ],
   "platforms": [ "platform1", "platform2", ... ],
   "devices": [ "device_uuid1", "device_uuid2", ... ]
  }
 }
}
```

PUT /v1/schedules/:schedule_id

Update a scheduled push for an application.

Authentication: HTTP Basic app_uuid:api_key

Query Parameters: None

Request Body:

```
{
 "scheduleAt": 1345852800000,     # Epoch timestamp in milliseconds.
 "message": {
  "custom": {
   "android": "object"
  },
  "body": ""
 },
 "target": {
  "interactive-only": false,
  "platform": "",
  "platforms": [ "platform1", "platform2", ... ],
  "topics": [ "topic1", "topic2", ... ],
  "devices": [ "device_uuid1", "device_uuid2", ... ]
 }
}
```

Response Data, status: 200 (OK)

```
{
 "schedule_id": "fc226fbc1443ebfe",
 "scheduled_for": 1345852800000,    # Epoch Timestamp in milliseconds
 "push": {
  "scheduleAt": 1345852800000,
  "scheduleIn": 0,
  "expiryTime": null,
  "message": {
   "custom": {
    "windows8": {
     "options": "object",
     "template_name": "",
     "template_fields": "object",
     "type": ""
    },
    "windowsPhone": {
     "template_fields": "object",
     "template": ""
    },
    "ios": {
     "extra": "object",
     "category": "",
     "badge": 0,
     "sound": "",
     "content-available": false,
     "alert": {
      "body": "",
      "loc-key": "",
      "action-loc-key": "",
      "loc-args": [ "arg1", "arg2", ... ],
      "launch-image": ""
     }
    },
    "bb10": "",
    "android": "object"
   },
   "body": ""
  },
  "target": {
   "interactive-only": false,
   "platform": "",
   "platforms": [ "platform1", "platform2", ... ],
   "topics": [ "topic1", "topic2", ... ],
   "devices": [ "device_uuid1", "device_uuid2", ... ]
  }
 }
}
```

DELETE /v1/schedules/:schedule_id

Cancel a scheduled push for an application.

Authentication: HTTP Basic app_uuid:api_key

Query Parameters: None

Request Body:

None.

Response Data, status: 204 (NO CONTENT)

Geofences

Endpoints for Managing Geofences

Create Geofence

POST /v1/geofence

Create a geofence for an app

Authentication: HTTP basic application_uuid:api_key

Query Parameters: None

Request Body:

```
{
 "tags": [
   "tag1",
   "tag2"
 ],
 "locations": [
   "1",
   "2"
 ],
 "trigger_type": "enter",
 "start_time": 0,
 "expiry_time": 1424443201000,
 "platform": "",
 "data": {
   "ios": {
    "alertBody": "",
    "category": "",
    "alertAction": "",
    "alertTitle": "",
    "alertLaunchImage": "",
    "hasAction": false,
    "applicationBadgeNumber": 0,
    "soundName": "",
    "userInfo": "object"
   },
   "android": "object"
 }
}
```

Geofence Fields

- tags

  type: array of strings

  required: no

  This is a list of tags to target. If not empty it will limit the audience for the geofence to only users that have subcribed to one or more of the listed tags

- locations

  required: yes

  type: array of numbers

  List of location ids for the locations that should be included in the geofence

- trigger_type

required: yes

type: “string”; possible values are “enter”, “exit”

When trigger_type is set to “enter” the notification will be displayed when a user enter the geofence. When it is set to “exit” the notification will not be displayed until the user exits the geofence.

- start_time

  required: yes

  type: millisecond timestamp (integer)

  Geofences are only active for a fixed period of time. “start_time” determines when the geofence should become active. Set this to “0” to activate the geofence upon creation

- expiry_time

  required: yes

  type: millisecond timestamp (integer)

  Sets the time when the geofence should become inactive.

- platform

  required: yes

  type: string; possible values are “android”, “ios”, “all”

  Target the geofence to devices of a specific platform

- data

  required: yes

  type: object

  The data object contains platform specific fields for constructing the notification to be displayed. These are slightly different than fields used in the push api because geofence notifications are actually local notifications. Custom User IDs are not a supported way to target registered devices for geofences.

iOS Geofence Data Fields

For Apple’s reference on local notifications see https://developer.apple.com/library/ios/documentation/iPhone/Reference/UILocalNotification_Class/index.html#//apple_ref/occ/instp/UILocalNotification/alertBody

All fields here are optional.

- alertBody

  type: string

  A string or localized-string key to use as the notification alert message. If nil or empty there no alert will be shown. Printf style escape characters are stripped from the string prior to display; to include a percent symbol (%) in the message, use two percent symbols (%%).

- category type: string

  The value of this property is the category name associated with a registered UIUserNotificationSettings object. When the alert for the local notification is displayed, the system uses the string you specify to look up the group and retrieve its actions. It then adds a button to the alert for each action defined by the group. When the user taps one of those buttons, the app is woken up (or launched) and given a chance to perform the designated action. If the specified category name does not belong to a registered group of actions, the alert does not display any additional action buttons.

- alertAction

  type: string

A string or localized-string key to use as the title of the right button of the alert or the value of the unlock slider, where the value replaces “unlock” in “slide to unlock”. If you specify nil, and alertBody is non-nil, “View” (localized to the preferred language) is used as the default value.

- alertTitle

  type: string

  A short description of the reason for the alert. Apple Watch displays the title string as part of the short look notification interface, which has limited space.

- alertLaunchImage

  type: string

  Identifies the image used as the launch image when the user taps (or slides) the action button (or slider).

- hasAction type: boolean Determines whether or not to show an alert action.
- applicationBadgeNumber

  type: number

  The number to display as the app icon’s badge. Default value is 0 which will simply not display a badge.

- soundName

  type: string

  The name of the file containing the sound to play when an alert is displayed.

- userInfo

  type: dictionary A dictionary for passing custom information to the notified app.

Response:


```
{
 "id": 0,
 "tags": [
  ""
 ],
 "expiry_time": 0,
 "trigger_type": "",
 "locations": [
  {
   "name": "",
   "id": 0,
   "long": "",
   "rad": 0,
   "lat": "",
   "created_at": 0,
   "updated_at": 0
  }
 ],
 "platform": "",
 "created_at": 0,
 "updated_at": 0,
 "data": {
  "ios": {
   "alertBody": "",
   "category": "",
   "alertAction": "",
   "alertTitle": "",
   "alertLaunchImage": "",
   "hasAction": false,
   "applicationBadgeNumber": 0,
   "soundName": "",
   "userInfo": "object"
  },
  "android": "object"
 },
 "start_time": 0
}
```

Get Geofences

GET /v1/geofence

Get all geofences for an app

Authentication: HTTP basic application_uuid:api_key

Query Parameters:

Parameters	Description
page: integer	result page to display
size: integer	number of results per page
timestamp: long	timestamp in milliseconds

Request Body: None

Response:


```
{
 "size": 25,
 "totalGeofences": 1,
 "totalPages": 1,
 "page": 1,
 "geofences": [
  {
   "id": 1,
   "expiry_time": 1424443201000,
   "trigger_type": "enter",
   "updated_at": 1423513994000,
   "created_at": 1423513994000,
   "data": {"object":{"key":"value"}},
   "tags": ["tag1"],
   "locations": [
    {
     "name": "sample",
     "id": 1,
     "lat": "0.0",
     "long": "0.0",
     "rad": 100,
     "updated_at": 1423513994000,
     "created_at": 1423513994000
    }
   ]
  }
 ]
}
```


Get Geofence Updates

GET /v1/geofences

Get updated geofences since a timestamp. This endpoint is used by devices to fetch an updated list of geofences to monitor.

Authentication: HTTP basic platform_uuid:platform_secret

Query Parameters:

Parameters	Description
timestamp: long	timestamp in milliseconds

Request Body:

None.

Response:


```
{
 "num": 3,
 "deleted_geofence_ids": [1,2],
  "geofences": [
   {
    "id": 5,
    "expiry_time": 1424443201000,
    "trigger_type": "enter",
    "updated_at": 1423513994000,
    "created_at": 1423513994000,
    "data": {"object":{"key":"value"}},
    "tags": ["tag1"],
    "locations": [
     {
      "name": "sample",
      "id": 1,
      "lat": "0.0",
      "long": "0.0",
      "rad": 100,
      "updated_at": 1423513994000,
      "created_at": 1423513994000
     }
    ],
  "last_modified": 1423513994000
}
```


- deleted_geofence_ids

  type: array of numbers

  List of ids for geofences that have been deleted since the requested timestamp.
- geofences

  type: array of geofence objects

  List of geofences that have been added since the requested timestamp.

Get One Geofence

GET /v1/geofence/:geofence_id

Authentication: HTTP basic application_uuid:api_key

Query Parameters: None

Request Body:

None.

Response:

```
{
 "id": 1,
 "expiry_time": 1424443201000,
 "trigger_type": "enter",
 "updated_at": 1423513994000,
 "created_at": 1423513994000,
 "data": {"object":{"key":"value"}},
 "tags": ["tag1"],
 "locations": [
  {
   "name": "sample",
   "id": 1,
   "lat": "0.0",
   "long": "0.0",
   "rad": 100,
   "updated_at": 1423513994000,
   "created_at": 1423513994000
  }
 ]
}
```


Update a Geofence

PUT /v1/geofence/:geofence_id

Authentication: HTTP basic application_uuid:api_key

Query Parameters: None

Request Body:


```
{
 "trigger_type": "enter",
 "expiry_time": 1424443201000,
 "data": {"object":{"key":"value"}},
 "tags": [
  "tag1"
 ],
 "locations": [1]
}
```


Response:


```
{
 "id": 1,
 "expiry_time": 1424443201000,
 "trigger_type": "enter",
 "updated_at": 1423513994000,
 "created_at": 1423513994000,
 "data": {"object":{"key":"value"}},
 "tags": [
  "tag1"
 ],
  "locations": [
  {
   "name": "sample",
   "id": 1,
   "lat": "0.0",
   "long": "0.0",
   "rad": 100,
   "updated_at": 1423513994000,
   "created_at": 1423513994000
  }
 ]
}
```

Delete a Geofence

DELETE /v1/geofence/:geofence_id

Authentication: HTTP basic application_uuid:api_key

Query Parameters: None

Request Body:

None.

Response: 204 (NO CONTENT)

Locations

Endpoints for managing geofence locations.

Get All Locations

GET /v1/locations

Get all geofence locations for an app

Authentication: HTTP basic application_uuid:api_key

Query Parameters:

Parameters	Description
page: integer	result page to display
size: integer	number of results per page
timestamp: long	timestamp in milliseconds
q: string	keyword to search for

Request Body:

None.

Response:

```
{
 "size": 25,
 "locations": [
   {
    "name": "sample",
    "id": 1,
    "long": "0.0",
    "rad": 100,
    "lat": "0.0",
    "created_at": 1423513994000,
    "updated_at": 1423513994000
   }
 ],
 "totalLocations": 1,
 "totalPages": 1,
 "page": 1
}
```

Get One Location

GET /v1/locations/:location_id

Authentication: HTTP basic application_uuid:api_key

Query Parameters: None

Request Body:

None.

Response:


```
{
 "name": "sample",
 "id": 1,
 "lat": "0.0",
 "long": "0.0",
 "rad": 100,
 "updated_at": 1423513994000,
 "created_at": 1423513994000
}
```


Create a New Location

POST /v1/locations

Authentication: HTTP basic application_uuid:api_key

Query Parameters: None

Request Body:


```
{
 "name": "sample",
 "lat": "0.0",
 "long": "0.0",
 "rad": 100
}
```

- name: a name for the location
- lat: latitude in degrees
- long: longitude in degrees
- rad: radius in meters

Response:

```
{
 "name": "sample",
 "id": 1,
 "lat": "0.0",
 "long": "0.0",
 "rad": 100,
 "updated_at": 14235139940000,
 "created_at": 1423513994000
}
```

Update a Location

PUT /v1/locations/:location_id

Authentication: HTTP basic application_uuid:api_key

Query Parameters: None

Request Body:

```
{
 "name": "sample",
 "lat": "0.0",
 "long": "0.0",
 "rad": 100
}
```

Response:

```
{
 "name": "sample",
 "id": 1,
 "lat": "0.0",
 "long": "0.0",
 "rad": 100,
 "updated_at": 1423513994000,
 "created_at": 1423513994000
}
```

Delete a Location

DELETE /v1/locations/:location_id

Authentication: HTTP basic application_uuid:api_key

Query Parameters: None

Request Body:

None.

Response: 204 (NO CONTENT)

Location Groups

Endpoints for managing geofence locations.

Get All Location Groups

GET /v1/location_groups

Get all location groups for an app

Authentication: HTTP basic application_uuid:api_key

Query Parameters:

Parameters	Description
page: integer	result page to display
size: integer	number of results per page
timestamp: long	timestamp in milliseconds
q: string	keyword to search for

Request Body:

None.

Response:


```
{
  "size": 25,
  "location_groups": [
    {
      "name": "sample group",
      "id": 1,
      "description": "sample location group",
      "locations": [
        {
          "name": "sample",
          "id": 1,
          "long": "0.0",
          "rad": 100,
          "lat": "0.0",
          "createdAt": 1423513994000,
          "updatedAt": 1423513994000
        }
      ],
      "created_at": 1423513994000,
      "updated_at": 1423513994000
    }
  ],
  "totalLocationGroups": 1,
  "totalPages": 1,
  "page": 1
}
```

Get One Location Group

GET /v1/location_groups/:location_group_id

Authentication: HTTP basic application_uuid:api_key

Query Parameters: None

Request Body:

None.

Response:

```
{
  "name": "sample group",
  "id": 1,
  "description": "sample location group",
  "locations": [
    {
      "name": "sample location",
      "id": 1,
      "long": "0.0",
      "lat": "0.0",
      "rad": 100
      "createdAt": 1423513994000,
      "updatedAt": 1423513994000
    }
  ],
  "created_at": 1423513994000,
  "updated_at": 1423513994000
}
```

Create a Location Group

POST /v1/location_groups

Authentication: HTTP basic application_uuid:api_key

Query Parameters: None

Request Body:

```
{
  "name": "sample group",
  "location_ids": [
    1
  ],
  "description": "a sample location group"
}
```

- name: name for the location group
- location_ids: list of ids for locations to include in the group
- description: a short description of the group

Response:

```
{
 "name": "sample group",
 "id": 1,
 "description": "",
 "locations": [
  {
   "name": "sample",
   "id": 1,
   "long": "0.0",
   "rad": 100,
   "lat": "0.0",
   "createdAt": 1423513994000,
   "updatedAt": 1423513994000
  }
 ],
 "created_at": 1423513994000,
 "updated_at": 1423513994000
}
```


Update a Location Group

PUT /v1/location_groups/:location_group_id

Authentication: HTTP basic application_uuid:api_key

Query Parameters: None

Request Body:


```
{
 "name": "sample group",
 "location_ids": [
  1
 ],
 "description": "a sample location group"
}
```


Response:


```
{
 "name": "sample group",
 "id": 1,
 "description": "",
 "locations": [
  {
   "name": "sample",
   "id": 1,
   "long": "0.0",
   "rad": 100,
   "lat": "0.0",
   "createdAt": 1423513994000,
   "updatedAt": 1423513994000
  }
 ],
 "created_at": 1423513994000,
 "updated_at": 1423513994000
}
```


Delete a Location Group

DELETE /v1/location_groups/:location_group_id

Authentication: HTTP basic application_uuid:api_key

Query Parameters: None

Request Body:

None.

Response: 204 (NO CONTENT)

Push Notification Service Release Notes

v1.7.1

Release Date: January 2017

Bug fixes

Fix the ability to edit scheduled pushes

Known issues

Push Scheduler and Push Service Broker are not compatible with any PCF that includes ruby-offline-buildpack version greater than 1.6.39. Refer to Pivotal Elastic Runtime Release Notes to see which release contains the appropriate ruby-offline-buildpack.

v1.7.0

Release Date: December 2016

Features

Support for Android FCM push notifications

Bug fixes

Dashboard session timeout error

Known issues

Once a scheduled push is created, it cannot be edited (will address this bug in a later release)

v1.6.3

Release Date: September 2016

Updated stemcell to v3263 to address kernel vulnerabilities (includes 4.4 kernel)

v1.6.2

Release Date: September 2016

Note: Update to Push Notification Service v1.6.2 prior to upgrading to Pivotal Cloud Foundry v1.8.

Updated stemcell to v3263 to address kernel vulnerabilities (includes 4.4 kernel)

Fixes:

- Fixed dashboard issue found when upgrading from PCF v1.7 to PCF v1.8

Known Issues

If you installed Push v1.6.2+ after upgrading to PCF v1.8, then remove the app named `push-notifications-analytics` with the following command:

```
$ cf delete push-notifications-analytics
```

v1.6.1

Release Date: August 2016

Features:

- Proxy support in Push Tile: Users can now add a proxy in the Push Tile (via Ops Mgr console)
- Installation logs now available in Ops Mgr console upon installation failure fixes

Fixes:

- Fixed issue with multiple tenants being provisioned in system org in push notifications space
- Fixed scaling issue with push api instances due to lack of database connections

Known Issues:

Upgrading to PCF v1.8 exposes a bug in versions of Push v1.6.1 and older. The impact is that the dashboard won't be able to display analytics (a message will appear stating "Analytics Data is not available at the moment"). Analytics data is still collected on the backend, the bug prevents it from being displayed.

The recommended solution is to upgrade to push v1.6.2 prior to upgrading to PCF v1.8 (this is now a pre-requisite for PCF v1.8)

If installing push v1.6.1 or earlier on PCF v1.8, follow the instructions below

1. To confirm this is the problem you are experiencing, you can check to see if there is a CF app running in the `system` org and `push-notifications` space called `push-notifications-analytics`.
2. Replace `push-analytics` with `push-notifications-analytics` and add a matching route as per the commands shown below

```
cf delete push-analytics
cf rename push-notifications-analytics push-analytics
cf map-route push-analytics $ENV_URL --hostname push-analytics
```

where `$ENV_URL` is the value of the domain name used for your PCF environment

v1.6.0

Release Date: July 2016

- Devices can be grouped under Custom User IDs which can be targeted for pushes
- Tags have been replaced by Topics
- Topics can be created with expiry dates

1.5.7

Release Date: December 2016

Security release for CVE as detailed in USN-3156-1

1.5.6

Release Date: December 2016

Security release for CVE as detailed in USN-3151-2

v1.5.3

Release Date: June 2016

- Bug fix for Service broker bug with HTTPS

v1.5.0

Release Date: June 2016

- New Heartbeat Application is deployed with the Push Notifications Service
- Heartbeat Monitor App available on iOS and Android

v1.4.27

Release Date October 2016

- Bump to stemcell v3151.3 for CVE as detailed in USN-3106-2: https://www.ubuntu.com/usn/usn-3106-2/

v1.4.25

Release Date October 2016

- Bump Ubuntu stemcell for USN-3099-2: Linux kernel (Xenial HWE) vulnerabilities

v1.4.24

Release Date: October 2016

- Updated Ubuntu stemcell for USN-3087-2: OpenSSL regression

v1.4.12

Release Date: June 2016

- Updated BOSH stemcell to v3262.2
- Bug fix for cf CLI

v1.4.10

Release Date: June 2016

- Security release requiring stemcell v3232.8

v1.4.9

Release Date: June 2016

- Security release requiring stemcell v3232.6
- Bug fix for Service broker bug with HTTPS

v1.4.7

Release Date: May 2016 - Security release requiring stemcell v3232.2

v1.4.5

Release Date: May 2016 - PCF v1.7 compatibility. - Update to this version of push *before* updating to PCF v1.7.0

v1.4.3

Release Date: March 2016 - Security release requiring stemcell v3146.10.

v1.4.2

Release Date: February 2016 - Security release requiring stemcell v3146.8.

v1.4.0

Release Date: November 2015

- The Push Notifications Service now supports multiple tenants.
    - Push Notifications is now a service that can be provisioned from the CF Marketplace.
    - The dashboard now requires a Tenant Id.
- The dashboard now displays logs related to push activities.
- The analytics system now configures a second Redis to behave as a cache for storing logs.
- Update to the Push SDK supports iOS 9 and includes a Swift sample app.
- The Push SDK for Android now supports Android 6.0 Marshmallow, including the new permissions system.
    - See the Push Sample app for an example of Android 6.0 Marshmallow permissions.

v1.3.5

Release Date: October 2015

- Support for PCF v1.6 and Diego.
- SOCKS proxy bug fix.

v1.3.4

Release Date: October 2015

- Bug fixes for smoke tests.

v1.3.3

Release Date: September 2015

- Bug fixes for certain scenarios regarding expiry time.

v1.3.3 iOS and Android Client SDK

- Push app analytics.
- Custom HTTP request headers.

- Custom SSL authentication.

v1.3.2

Release Date: August 2015

- Deprecated lucid64 stack in favour of the new Trusty/cflinuxfs2 stack
- Proxy Support for iOS push notifications. Supports SOCKS proxies.
- Proxy Support for Android push notifications. Supports HTTP and SOCKS proxies.

v1.3.2 iOS and Android Client SDK

- Enable and disable geofences at runtime.
- Added a method to read the device UUID at runtime.

v1.3.1

Release Date: August 2015

- Support for RabbitMQ Service versions v1.4.0 and later
- Tag management added to dashboard
- Ability to regenerate push api keys
- Minor improvements to installation
- Allow certificate checks to be disabled in cf environments that use self signed certificates

v1.3.1 iOS and Android Client SDK

- SSL Certificate pinning.
- Any geofences with tags will be monitored only if the user is subscribed to that tag.

v1.3.0

Release Date: June 2015

- Location based notifications
- Android and iOS support (SDKs)
- Dashboard support
    - Maps
    - Saved locations and groups of locations
    - Active geofences view

Upgrading from version v1.2.x to v1.3.0

v1.2.1

Release Date: April 2015

- Offline installation support

v1.2.0

Release Date: March 2015

- Scheduled push notifications
- Notifications with expiry time
- Updated UI/UX for dashboard (sending scheduled push with expiry time)

v1.1.0 - January 2015

v1.0.1 - November 2014

v1.0.0 — July 2014