TigaseDoc

Release 0.1

Tigase, Inc.

Dec 16, 2022

TIGASE PUSH ADMINISTRATION GUIDE

CHAPTER

ONE

TIGASE PUSH COMPONENT

Tigase Push component is a Push notifications component implementing XEP-0357: Push Notifications. It is a gateway between Push Notification services and XMPP servers. It is configured by default to run under name of `push`.

Note: Tigase Push component requires at the minimum version 8.0.0 of Tigase XMPP Server.

Push notifications enable messages and pertinent information for clients, even if they are offline as long as they are registered with the push service. Tigase Messenger for iOS and Tigase Messenger for Android both have support for this feature.

1.1 Workflow

The workflow for enabling and using push notifications works as follows:

1.1.1 Enabling notifications

In order to receieve notifications, clients will require registration with a push service. Although this process is mainly invisible to the user, the steps in registration are listed here:

- The client registers and bootstraps too it's assicoated push service. This is done automatically.
- The client registers itself with the push service server which then will dedicate a node for the device.
- Node information is passed back to the client and is shared with necessary components.

1.1.2 Receiving notifications

Notifications sent from the server are recieved the following (simplified) way:

- A message is published on the XMPP node which is then sent to the push service on the same server.
- The push service will then inform the user agent (an application on the device running in the background) that a push notification has been sent.
- The user agent will then publish the notification to the client for a user to see, waking up or turning on the client if is not running or suspended.

CHAPTER

TWO

TIGASE PUSH RELEASE NOTES

Welcome to Tigase Push 1.2.0! This is a feature release for with a number of fixes and updates.

2.1 Tigase Push 1.2.0 Release Notes

2.1.1 Major Changes

- Added support for sending VoIP push notifications using PushKit
- Support for storing APNS certificates in repository instead of filesystem for easier cluster deployments
- Add priority detection for push notifications to avoid excessive pushes to devices
- Inclusion of APNS certificate validity task that notifies if it's about to expire

2.1.2 All Changes

- #push-29 Added support for sending VoIP push notifications using PushKit
- #push-30 Added REST API handler for quick unregistration of a device
- #push-32 Fixed issue with APNS certificate validation
- #push-33 Added statistics gathering
- #push-35 Added support for APNS certificate in PEM
- #push-36 Improved priority detection of push notifications
- #push-37 Enable APNS certificates to be stored in UserRepository - management is done via ad-hoc command;
- #push-39 Changes to improve error handling
- #push-41 Fixed issue with `ApnsService` exceptions not being thown logged
- #push-42 Fixed error type reported back on `tooManyRequestsForDeviceToken`
- #push-47 Added task to periodically validate SSL certificates for Push notifications
- #push-48 Fixed handling events by APNsBinaryApiProvider
- #push-49 Added enforcement to use HTTP/2 protocol (with use of ALPN)

CHAPTER THREE

CONFIGURATION

3.1 Enabling component

Push notifications may be sent by Tigase XMPP Server with or without use of Push component. Push Component is only required if you have your own application for mobile devices for which you want to send push notifications.

This component is not loaded and enabled by default as it requires implementations of Push notifications providers and additional configuration (including credentials required to authorize to push services). Following entries will activate component:

```
push () {
}
```

Note: You need to enable and configure push providers implementations before it will be possible to send push notifications. For more details about this process, please check documentations of push service provider projects.

CHAPTER

FOUR

USAGE

4.1 Sending notifications

When you will register a device for a Push Notifications, you will receive name of the PubSub node where you should publish items. Publishing items to this node, as specified in XEP-0357: Push Notifications will result in push notifications being delivered to the registered device.

4.2 Registering device

To register a device you need to execute the adhoc command `register-device` available at Push Notification component. This command will return a form which needs to be filled.

Form consists of following fields:

provider
    ID of a provider for which you want to register a device. It contains a list of available providers and you need to select a proper one.

device-token
    Unique token which your application retrieved from a device or client library and which should be used to identify device you want to register for push notifications.

When you submit this form, it will be processed and will respond with a `result` type form. Within this form you will find a `node` field which will contain a PubSub node name created by the Push Notifications component, to which you should publish notification items. This returned node with jid of the Push Notifications Component should be passed to your XMPP server as the address of the XMPP Push Service.

4.3 Unregistering device

To unregister a device, you need to execute the adhoc command `unregister-device` available within the Push Notification component. This command will return a form which needs to be filled out.

This form consists of the following fields:

provider
    ID of a provider for which your devices was registered.

device-token
    Unique token which your application retrieved from a device or client library and was registered at this push notifications component.

When you submit this form, it will be processed and will respond with a `result` form to notify you that device was successfully unregistered from the push notifications component.

4.4 Unregistering device via HTTPS

There is REST API handler (in form of `UnregisterDeviceHandler.groovy` script) which placed in `/scripts/rest/push/` directory in Tigase XMPP Server installation directory will enable endpoint (documented in Development Guide) allowing client to disable their push notifications even without authentication to their XMPP server.

Note: It is recommended to not expose this endpoint using HTTP but only with HTTPS.

CHAPTER

FIVE

PROVIDERS

Providers availability depends on the deployed binaries, by default Tigase includes following providers:

5.1 Tigase Push Component - FCM provider

5.1.1 Overview

Tigase Push Component - FCM provider is an implementation of FCM provider for Tigase Push Component. It allows Tigase Push Component to connect to Firebase Cloud Messaging and send notifications using this service.

5.1.2 Configuration

Enabling provider

To enable this provider, you need to enable fcm-xmpp-api bean within push component configuration scope.

Example.

```
push () {
    'fcm-xmpp-api' () {
        # FCM configuration here
    }
}
```

Note: You need to pass FCM configuration parameters to make it work, see below.

Setting FCM credentials

FCM XMPP API provider will not work properly without API key and project id as this values are required for authorization by FCM. You need to get information from FCM account.

When you have this data, you need to pass sender id as sender-id property and server key as server-key property.

Example.

```
push () {
    'fcm-xmpp-api' () {
        'sender-id' = 'your-sender-id'
        'server-key' = 'your-server-key'
    }
}
```

Connection pool

By default this provider uses single client to server connection to FCM for sending notifications. If in your use case it is to small (as you need better performance), you should adjust value of pool-size configuration property. Setting it to 5 will open five connections to FCM for better performance.

Example.

```
push () {
    'fcm-xmpp-api' () {
        'pool-size' = 5
    }
}
```

5.2 Tigase Push Component - APNs provider

5.2.1 Overview

Tigase Push Component - APNs provider is an implementation of APNs provider for Tigase Push Component. It allows Tigase Push Component to connect to Apple Push Notification service and send notifications using this service.

5.2.2 Configuration

Enabling provider

To enable this provider, you need to enable apns-binary-api bean within push component configuration scope.

Example.

```
push () {
    'apns-binary-api' () {
        # APNs configuration here
    }
}
```

Note: You need to pass APNs configuration parameters to make it work, see below.

Setting APNs credentials

APNs binary API provider will not work properly without certificate file required for authorization by APNs and password to decrypt this certificate file. You need to get certificate using Apple Developer Account.

When you have this certificate, you need to pass path to certificate file as cert-file property, password as cert-password and APNS topic (bundle id) as apns-topic.

Example for /etc/apns-cert.p12, Pa$$word and com.bundle.id.

```
push () {
    'apns-binary-api' () {
        'cert-file' = '/etc/apns-cert.p12'
        'cert-password' = 'Pa$$w0rd'
        'apns-topic' = 'com.bundle.id'
    }
}
```

Alternatively, certificate can be stored in the database and in that case the TDSL configuration file should only contain `'apns-topic'` entry and the certificate and the password should be updated via ad-hoc command (Service discovery → Push component → Set APNS certificate). In the ad-hoc you should select the APNS provider from the list and include base64 encoded certificate obtained from Apple (`.p12` file), for example:

```
base64 -w 0 PushCertificate.p12
```

CHAPTER

SIX

DEVELOPMENT GUIDE

Warning: THIS IS FOR INTERNAL USE ONLY!!

6.1 Push Notification format

6.1.1 FcmXmppApiProvider

This provider sends APNs notifications using following fields and values.

Field	Value	Description
me ssage_id	“m-” + UUID	Random id for the notification
priority	high	Priority of the notification
data	Description of fields used within “data” object	

F ie ld	V al ue	Descr iption
account	user@example.com	Bare JID of user a ccount
unread-messages	2	Number of unread me ssages w ait-ing
sender	send@example/tes t	Full JID of a sender of the last m essage
body	” Text of a message body”	Text of a m es-sage body l imited to 500 chars with el lipsis added if m essage body e xceeds 512 chars

Note: Any of fields described above may be not passed if notification sent to PUSH provider does not contain values for a particular fields.

6.1.2 APNsBinaryApiProvider

This provider sends APNs notifications using following fields and values.

Field	Value	Description
content -available	1	Mark notification as there is a content waiting for download (ie. XMPP message to retrieve)
account	user@example.com	Bare JID of user account
unread- messages	2	Number of unread messages waiting
sender	sender@example.com/test	Full JID of a sender of the last message
body	“Text of a message body”	Text of a message body limited to 500 chars with ellipsis added if message body exceeds 512 chars

Note: Any of fields described above may be not passed if notification sent to PUSH provider does not contain values for a particular fields.

6.2 Disabling Push while not connect to the XMPP account

In some cases users may want/need to disable push notifications for their device while not being able to connect to the XMPP server (ie. password was changed, server is offline, user is blocked, etc.). To allow that we have created REST API call available at: `/rest/push/unregister-device/{push-component-jid}` where `push-component-jid` is a jid of a push component. This endpoint answers POST calls with JSON payload in the following format:

```
{
  "account": "user@example.com",
  "provider": "provider-id",
  "device-token": "XXXXXXX"
}
```

where:

- `account` is a bare jid of users account for which we want to disable push notifications
- `provider` is a provider id for which client registered (provider is responsible for delivering pushes to proper push service, ie. APNs or FCM)
- `device-token` is a device id or token used to register for push notifications

On the request handler with answer with:

On success.

```
{
    "result": "success"
}
```

On failure.

```
{
  "result": "failure"
}
```

Warning: If push component is not a local component, or a payload is incorrect (fields are missing), then endpoing will respond with HTTP error code 404. This is intentional as this API is exposed publicly and we do not want easy for anyone else except client authors to be able to execute those calls.