

Exhibit B - U.S. Patent No. 9,198,117 (“’117 Patent”)

Accused Devices: Samsung’s mobile electronic devices (e.g., Galaxy phones and tablets, as well as Samsung devices which include Samsung Knox functionality), and Samsung Tizen devices (e.g., TVs and wearables), and all versions and variations thereof since the issuance of the asserted patent.

Claim 1

Issued Claim(s)	Public Documentation
[Ipre]. A network system comprising:	Samsung’s devices and push messaging servers comprise a “network system.” <i>See, e.g.,</i>  https://www.samsung.com/us/smartphones/galaxy-s22/;

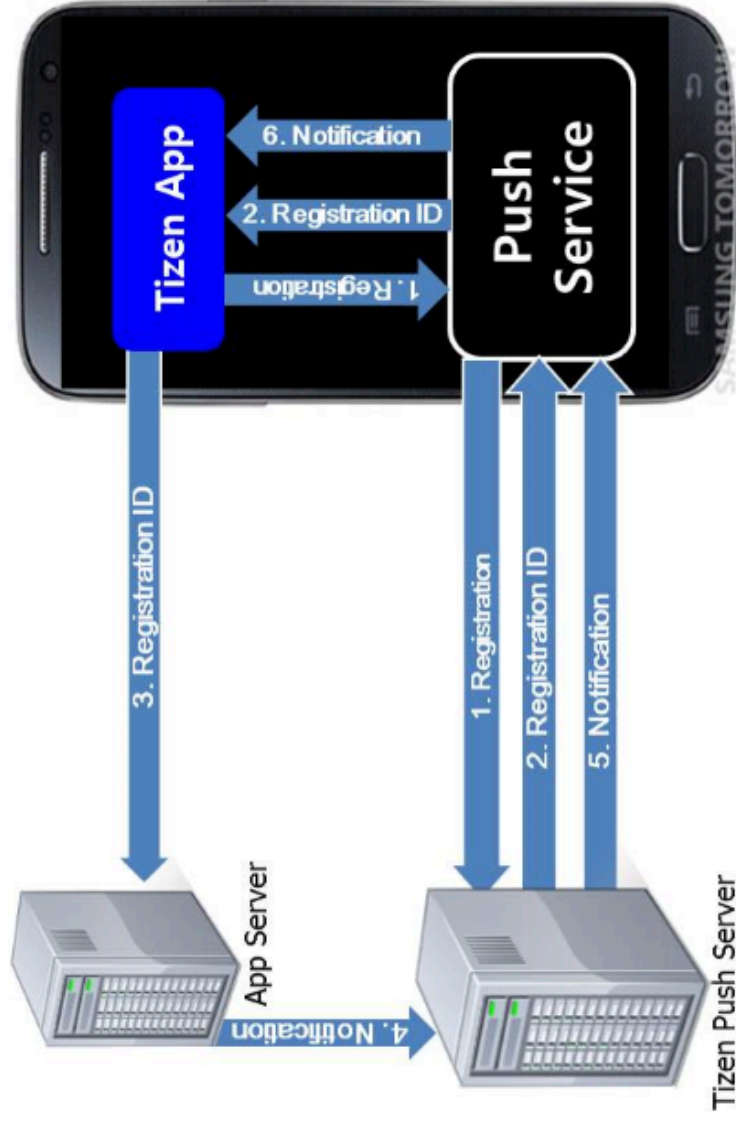


[https://www.samsung.com/us/televisions-home-theater/tvs/the-frame/55-class-the-frame-qled-4k-smart-tv-2022-qn55ls03bafxza/;](https://www.samsung.com/us/televisions-home-theater/tvs/the-frame/55-class-the-frame-qled-4k-smart-tv-2022-qn55ls03bafxza/)

Architecture

The architecture of the Tizen Push service is described in detail in the [mobile native Push guide](#).

Figure: Service architecture

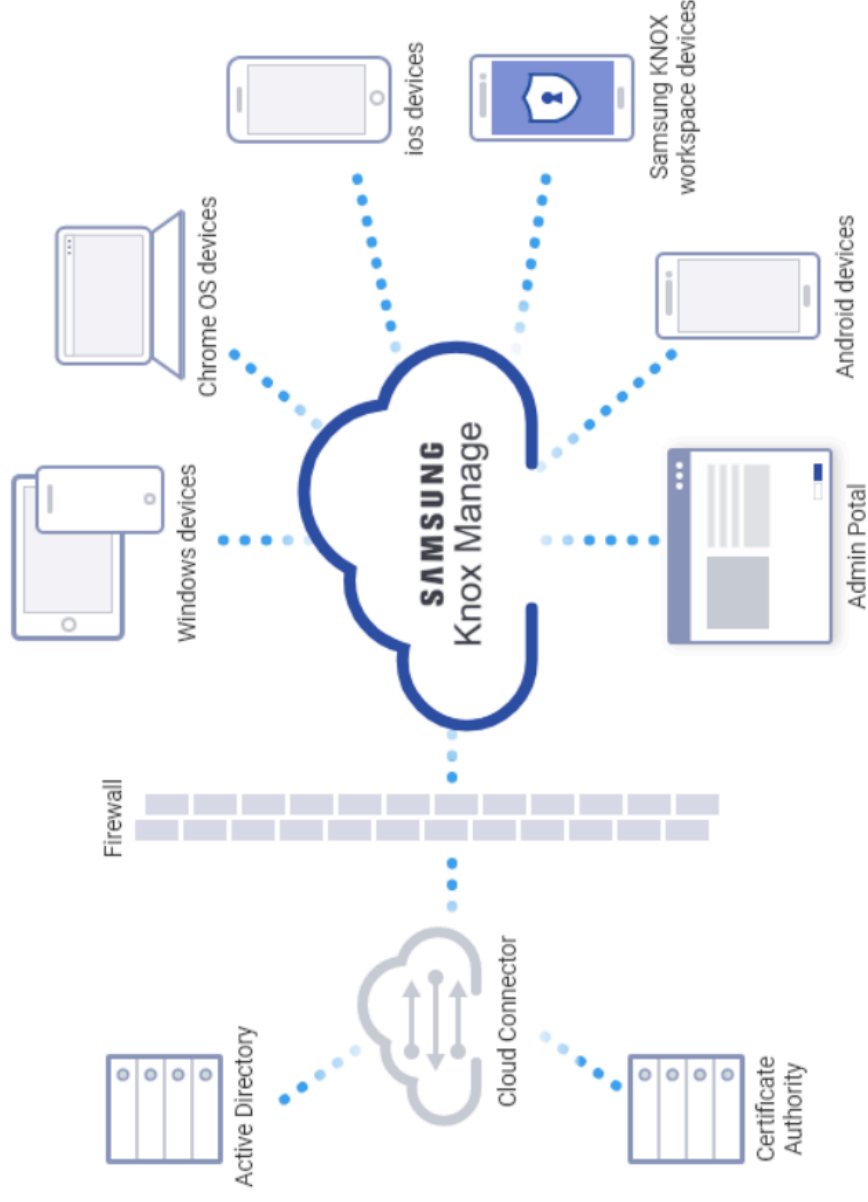


To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.

<https://docs.tizen.org/application/web/guides/messaging/push/>.

As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, comprise a "network system."

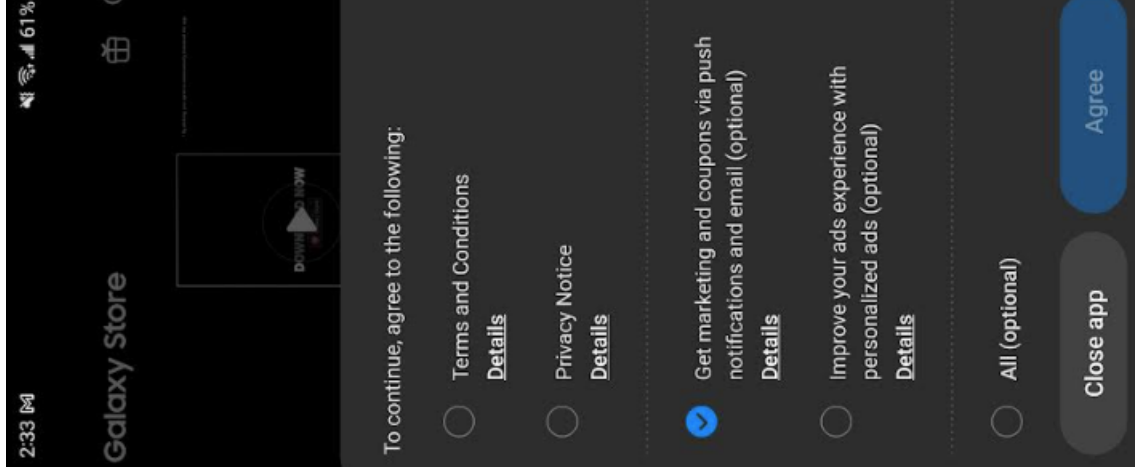


<https://docs.samsungknox.com/admin/knox-manage/welcome.htm>

On information and belief, Samsung owns and operates its own push messaging servers, including servers for communicating with the Samsung Push Service software on its end user devices. *See, e.g.,*

https://play.google.com/store/apps/details?id=com.sec.spp.push&hl=en_US&gl=US&pli=1;

<https://www.lifewire.com/samsung-push-service-4165507>. On information and belief, Samsung includes applications in its user devices, such as the “Shop Samsung” and “Galaxy Store” applications, which cause notifications (including marketing notifications) to be sent to user devices via Samsung’s push messaging servers. *See, e.g.:*



[1a] a plurality of device messaging agents, each executable on a respective one of a plurality of mobile end-user devices configured to exchange Internet data via a data connection to a wireless network; and	Samsung devices and push messaging servers comprise “a plurality of device messaging agents, each executable on a respective one of a plurality of mobile end-user devices configured to exchange Internet data via a data connection to a wireless network.” For example, Samsung Galaxy phones and tablets comprise multiple device agents which have an identifier. <i>See e.g.</i>, Was this helpful?  Firebase > Documentation > FCM > Engage Set up a Firebase Cloud Messaging client app on Android  ▾ Send feedback FCM clients require devices running Android 4.4 or higher that also have the Google Play Store app installed, or an emulator running Android 4.4 with Google APIs. Note that you are not limited to deploying your Android apps through Google Play Store. Set up the SDK This section covers tasks you may have completed if you have already enabled other Firebase features for your app. If you haven't already, add Firebase to your Android project
---	---

Edit your app manifest

Add the following to your app's manifest:

- A service that extends `FirebaseMessagingService`. This is required if you want to do any message handling beyond receiving notifications on apps in the background. To receive notifications in foregrounded apps, to receive data payload, to send upstream messages, and so on, you must extend this service.

```
<service
    android:name=".java.MyFirebaseMessagingService"
    android:exported="false">
    <intent-filter>
        <action android:name="com.google.firebase.messaging.MESSAGEING_EVENT" />
    </intent-filter>
</service>
```

   [AndroidManifest.xml](#)

Access the device registration token

On initial startup of your app, the FCM SDK generates a registration token for the client app instance. If you want to target single devices or create device groups, you'll need to access this token by extending `FirebaseMessagingService` and overriding `onNewToken`.

This section describes how to retrieve the token and how to monitor changes to the token. Because the token could be rotated after initial startup, you are strongly recommended to retrieve the latest updated registration token.

The registration token may change when:

- The app is restored on a new device
- The user uninstalls/reinstall the app
- The user clears app data.

<https://firebase.google.com/docs/cloud-messaging/android/client>;

Receive messages in an Android app

[Send feedback](#)

Firebase notifications behave differently depending on the foreground/background state of the receiving app. If you want foregrounded apps to receive notification messages or data messages, you'll need to write code to handle the `onMessageReceived` callback. For an explanation of the difference between notification and data messages, see [Message types](#).

Handling messages

To receive messages, use a service that extends [FirebaseMessagingService](#). Your service should override the `onMessageReceived` and `onDeletedMessages` callbacks. It should handle any message within 20 seconds of receipt (10 seconds on Android Marshmallow). The time window may be shorter depending on OS delays incurred ahead of calling `onMessageReceived`. After that time, various OS behaviors such as Android O's [background execution limits](#) may interfere with your ability to complete your work. For more information see our overview on [message priority](#).

`onMessageReceived` is provided for most message types, with the following exceptions:

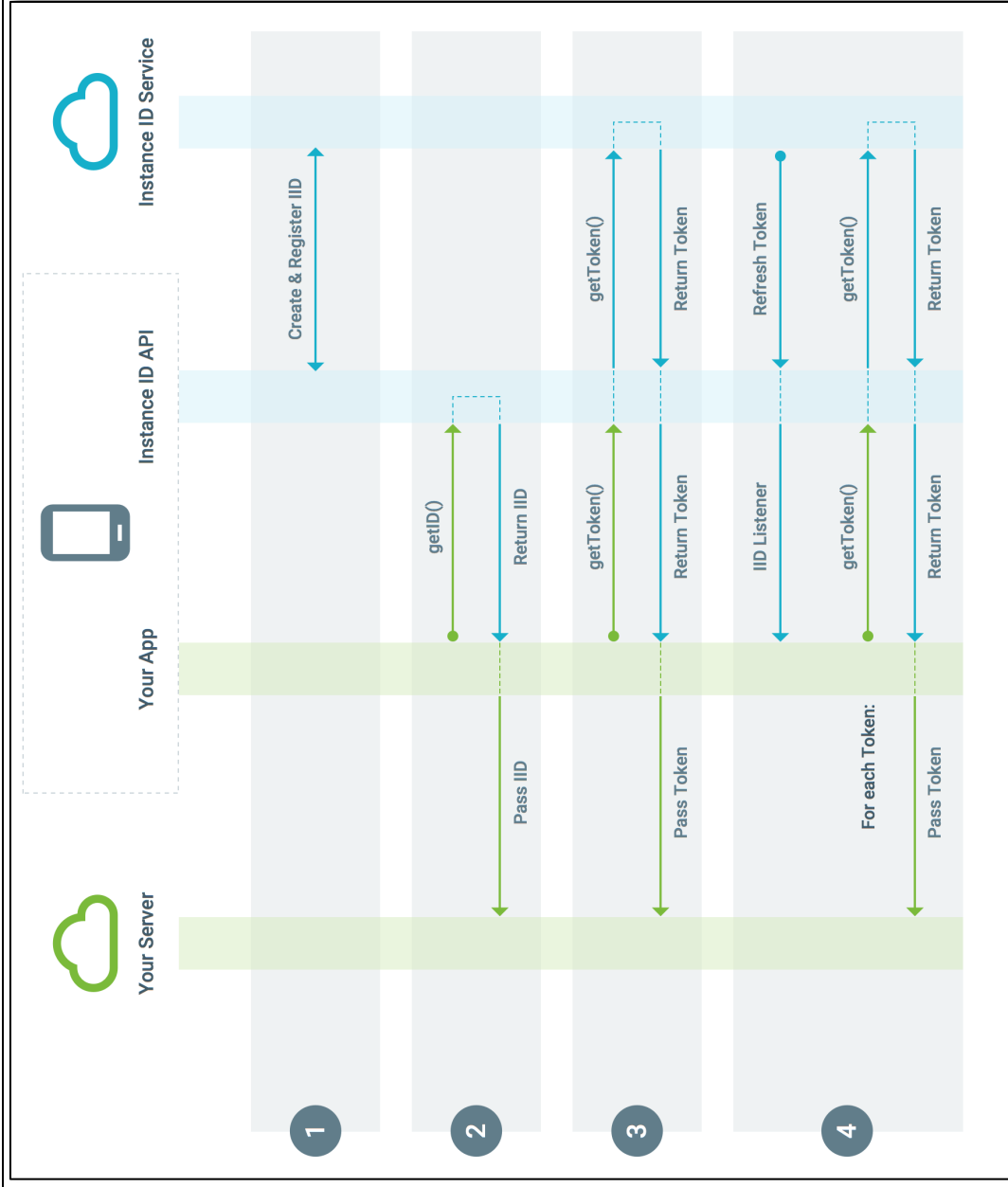
- Notification messages delivered when your app is in the background.** In this case, the notification is delivered to the device's system tray. A user tap on a notification opens the app launcher by default.
- Messages with both notification and data payload, when received in the background.** In this case, the notification is delivered to the device's system tray, and the data payload is delivered in the extras of the intent of your launcher Activity.

<https://firebase.google.com/docs/cloud-messaging/android/receive;>

What is Instance ID?

[Send feedback](#)

Instance ID provides a unique ID per instance of your apps. You can implement Instance ID for [Android](#) and [iOS](#) apps as well as [Chrome apps/extensions](#).



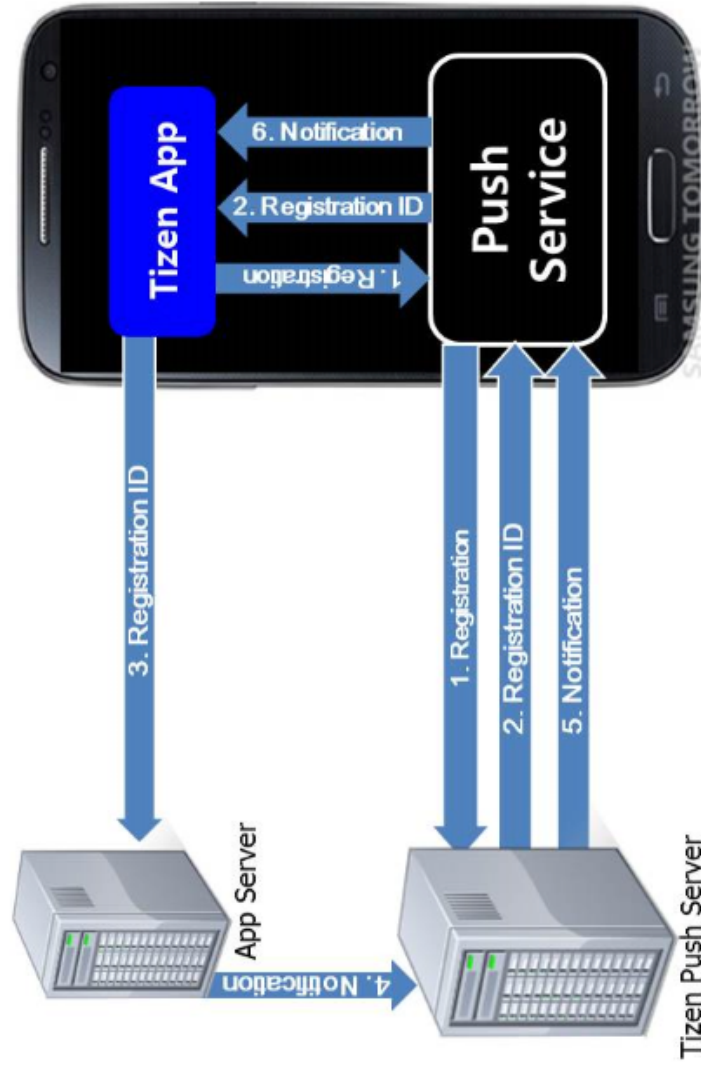
<https://developers.google.com/instance-id>.

For further example, Samsung's Tizen based devices comprise multiple device agents which have an identifier. See e.g.,

Architecture

The architecture of the Tizen Push service is described in detail in the [mobile native Push guide](#).

Figure: Service architecture



To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.

Push Notification

You can receive notifications from a push server. The push service is a client daemon that maintains a permanent connection between the device and the push server. Push enables you to push events from an application server to your application on a Tizen device. Connection with the push service is used to deliver push notifications to the application, and process the registration and deregistration requests.

The Push API is optional for Tizen Mobile, Wearable, and TV profiles, which means that it may not be supported on all mobile, wearable, and TV devices. The Push API is supported on all Tizen emulators.

Push notification helps your application server send data to your application on a device over a network, even if the application is not running. Using the push service can reduce battery consumption and data transfer.

If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a **LAUNCH** option, the push service forcibly launches the application and hands over the message to the application.

The main features of the Push API include:

- Registering to the push service

You can [register to the push service](#) and get the registration identifier.

Registering to the Push Service

To receive push notifications, you must learn how to register your application to the push service:

- Up to Tizen 2.4:

1. Define event handlers for the registration results:

```
/* Define the data to be used when this process
   is launched by the notification service
*/
var service = new tizen.ApplicationControl('http://tizen.org/appcontrol/operation/push_test');

/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}
```

2. Register the application for the service with the `register()` method. This operation has to be done only once.

```
/* Request application registration */
tizen.push.registerService(service, registerSuccessCallback, errorCallback);
```

- Since Tizen 3.0:

Before registering, you must connect to the push service:

1. Define event handlers:

```
/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}

/* Define the state change callback */
function stateChangeCallback(state) {
    console.log('The state is changed to: ' + state);
}

if (state == 'UNREGISTERED') {
    /* Request application registration */
    tizen.push.register(registerSuccessCallback, errorCallback);
}

/* Define the notification callback */
function notificationCallback(notification) {
    console.log('A notification arrives.');
```

2. Connect to the push service with the `connect()` method. The `register()` method is called in the `stateChangeCallback()` callback. This operation has to be done only once.

```
/* Connect to push service */
tizen.push.connect(stateChangeCallback, notificationCallback, errorCallback);
```

If the registration is successful, the `registerSuccessCallback()` callback is called, and the registration ID is passed as a parameter. Any time after a successful registration, you can get the registration ID using the `getRegistrationId()` method:

```
var registrationId = tizen.push.getRegistrationId();
if (registrationId != null) {
    console.log('The registration id: ' + registrationId);
}
```

<https://docs.tizen.org/application/web/guides/messaging/push/>;

Push Notification

You can receive notifications from a push server. The push service is a client daemon that maintains a permanent connection between the device and the push server. Push enables you to push events from an application server to your application on a Tizen device. Connection with the push service is used to deliver push notifications to the application, and process the registration and deregistration requests.

<https://developer.tizen.org/development/guides/web-application/messaging/push-notification;>

Push API

The Push API provides functionality for receiving push notifications from the Tizen push server. The push service is a client daemon that maintains a permanent connection between your device and the Tizen push server. Connection with push server is used to deliver push notifications to the application, and process the registration and deregistration requests.

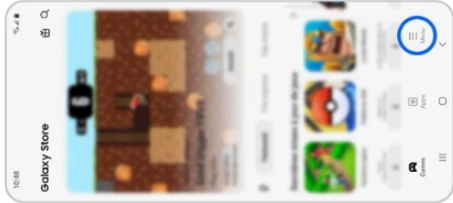
To receive push notifications, follow the steps below:

- Connecting to the push service
- Registering your application, if the application has not been registered yet
- Getting notification data

<https://developer.samsung.com/smarttv/develop/api-references/tizen-web-device-api-references/push-api.html>;

How to set up push notifications

If you want to set up push notifications, follow these steps.



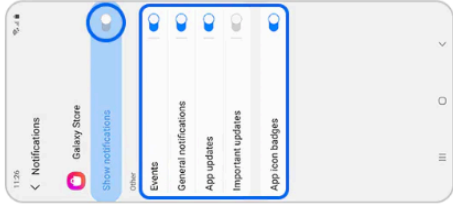
Step 1. Log in to Galaxy Store and tap the "Menu" icon from the main screen.



Step 2. Tap the settings icon located at the top right of the Galaxy Store menu.



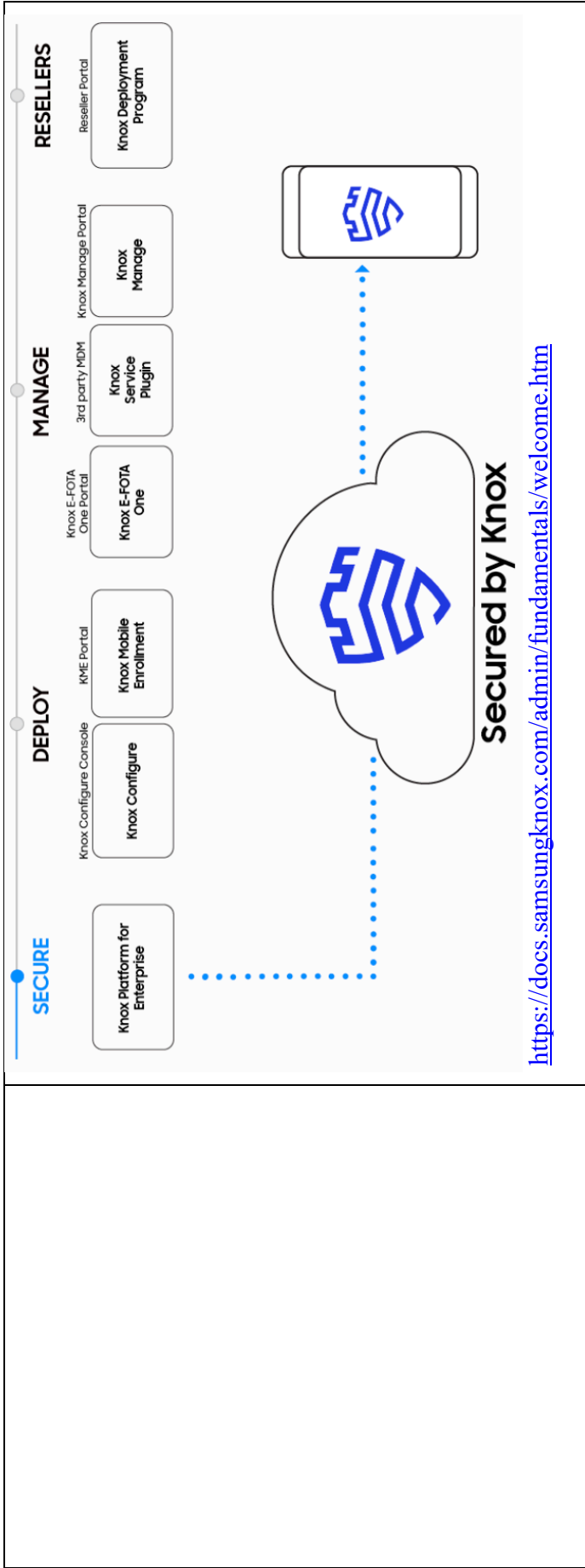
Step 3. Enable "Get news and special offers" and tap "Notifications."

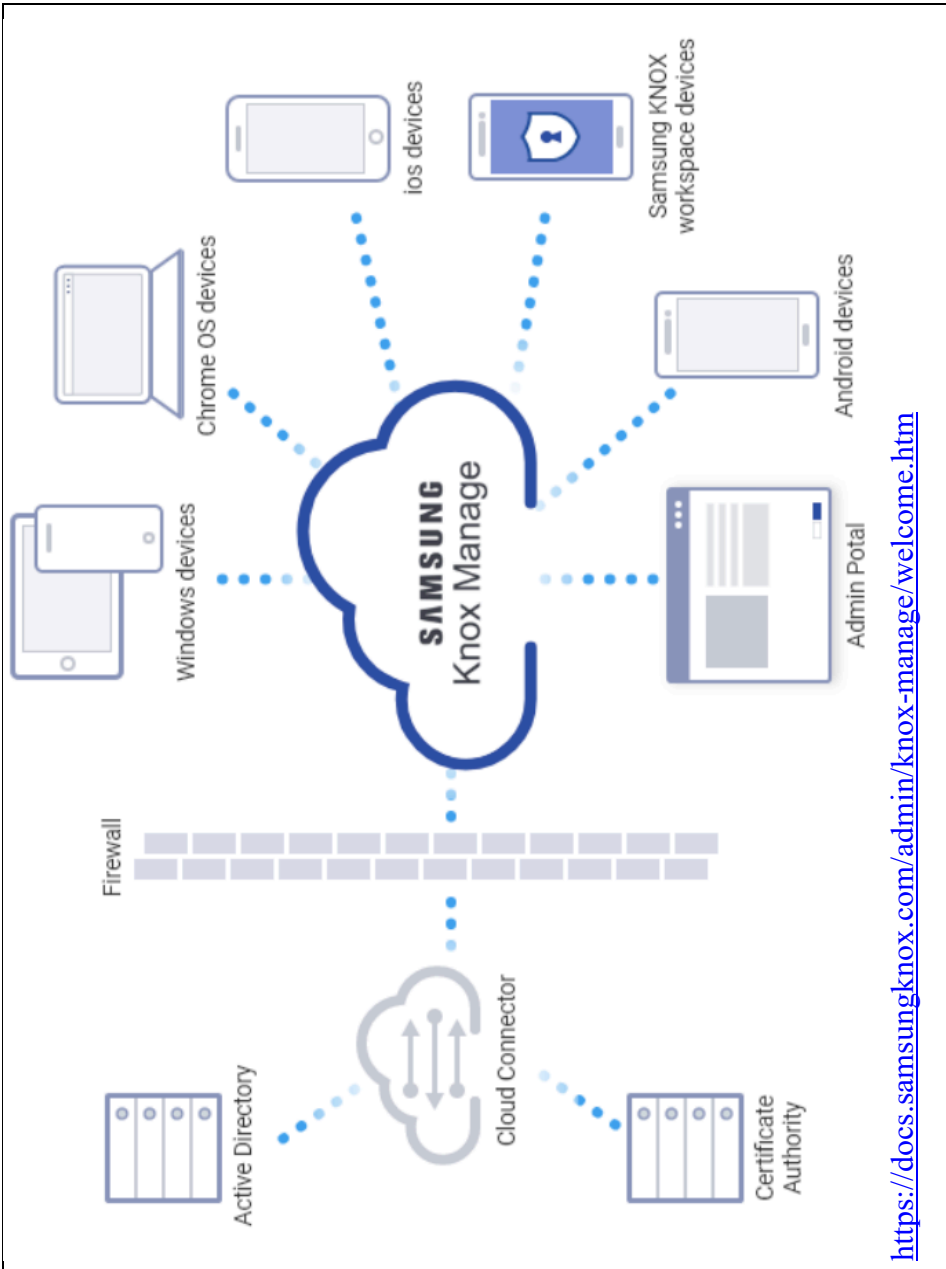


Step 4. Enable "Show notifications" and then enable the types of notifications you want to receive.

https://www.samsung.com/hk_en/support/apps-services/how-do-i-enable-push-notifications-from-galaxy-store/.

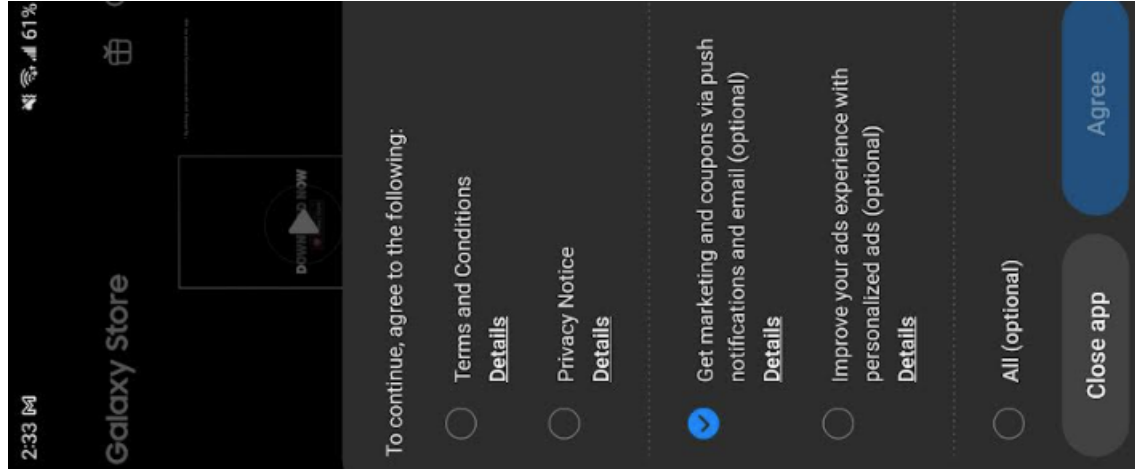
As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, comprise multiple device agents which have an identifier.





	https://docs.samsungknox.com/admin/knox-manage/km-features.htm On information and belief, Samsung owns and operates its own push messaging servers, including servers for communicating with the Samsung Push Service software on its end user devices. <i>See, e.g.,</i> https://play.google.com/store/apps/details?id=com.sec.spp.push&hl=en_US&gl=US&pli=1; https://www.lifewire.com/samsung-push-service-4165507. On information and belief, Samsung includes applications in its user devices, such as the “Shop Samsung” and “Galaxy Store”
--	--

applications, which cause notifications (including marketing notifications) to be sent to user devices via Samsung's push messaging servers. See, e.g.:

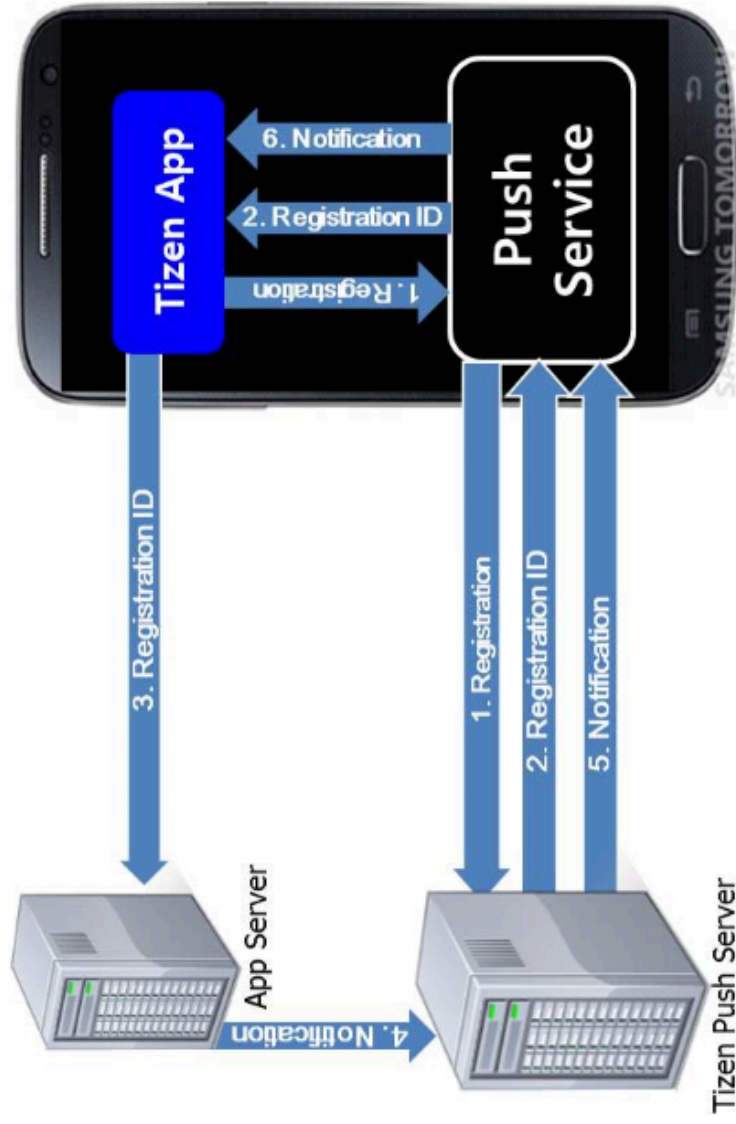


[1b] a network message server	Samsung devices and push messaging servers comprise “a network message server.” For example, Samsung’s push messaging servers are network servers. <i>See e.g.,</i>
-------------------------------	---

Architecture

The architecture of the Tizen Push service is described in detail in the [mobile native Push guide](#).

Figure: Service architecture



To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.

<https://docs.tizen.org/application/web/guides/messaging/push/>;

Push Server

You can push events from an application server to your application on a Tizen device. If the message sending fails for any reason, an error code identifying the failure reason is returned. You can use the [error code](#) to determine how to handle the failure.

The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices.

The main features of the Push API for the server developers include:

- Sending push notifications

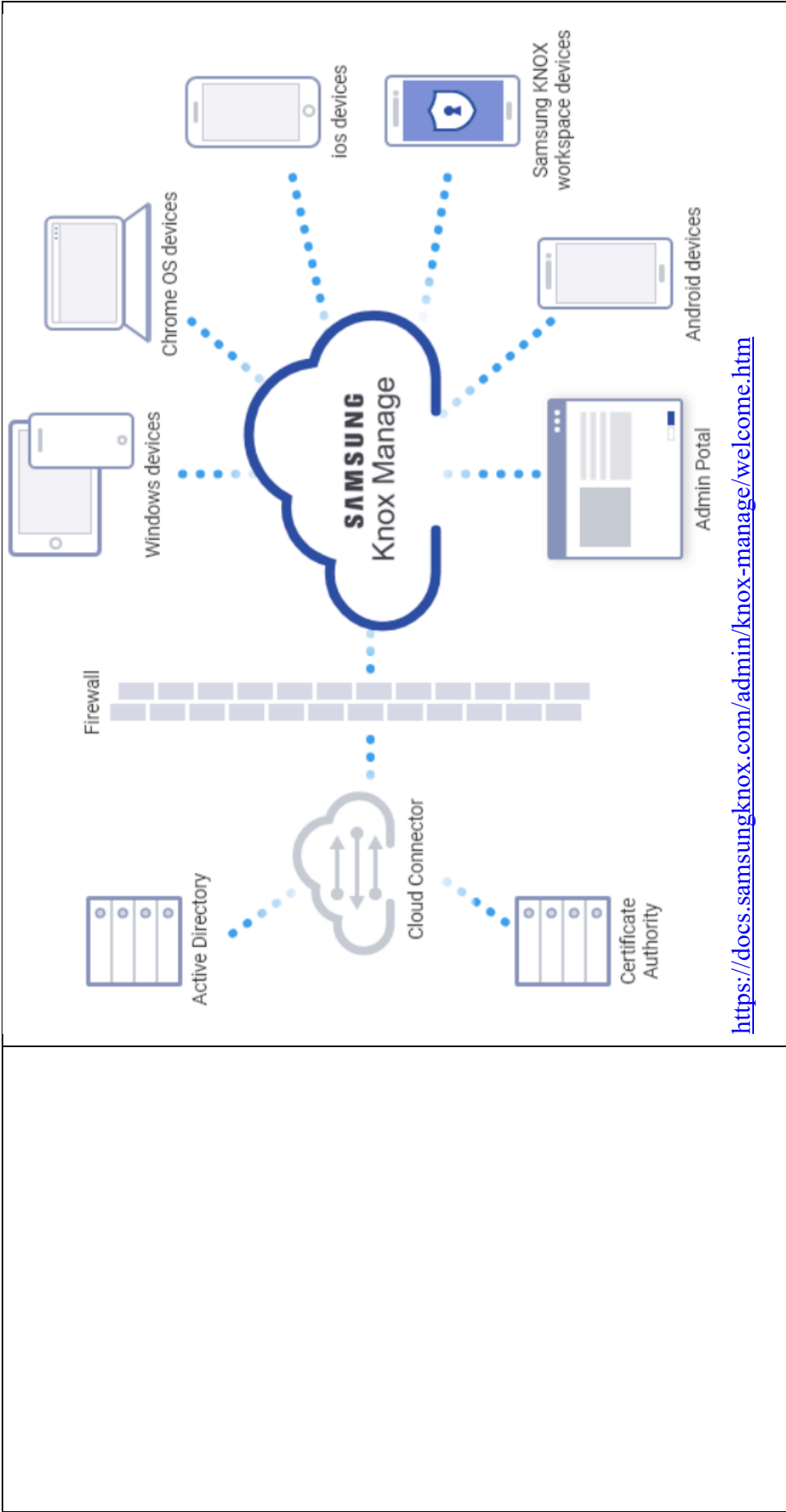
You can [send push notifications](#) from the application server to an application.

- Decorating push notifications

You can [add decorations](#) to the push notifications in the quick panel.

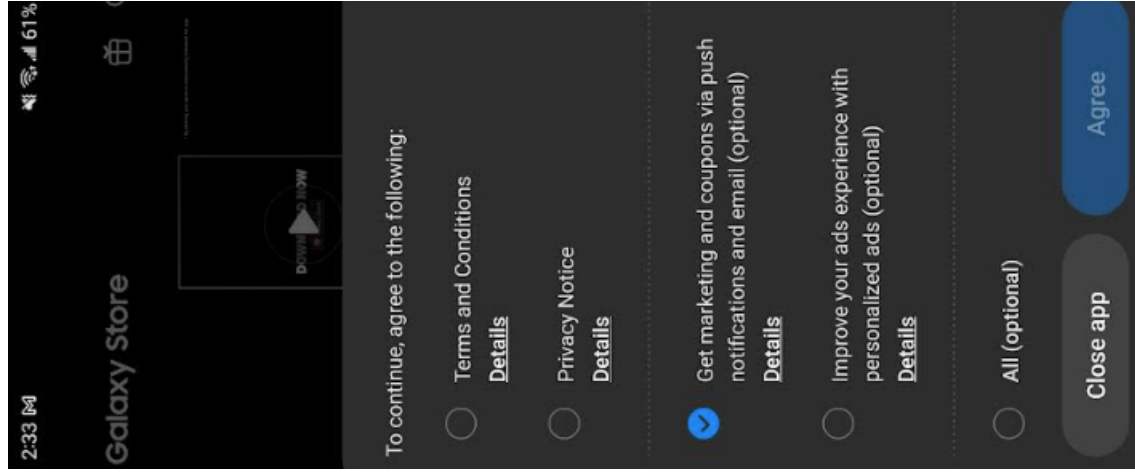
<https://docs.tizen.org/application/native/guides/messaging/push-server/>;

As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, include network message servers.



	https://docs.samsungknox.com/admin/knox-manage/km-features.htm On information and belief, Samsung owns and operates its own push messaging servers, including servers for communicating with the Samsung Push Service software on its end user devices. <i>See, e.g.,</i> https://play.google.com/store/apps/details?id=com.sec.spp.push&hl=en_US&gl=US&pli=1; https://www.lifewire.com/samsung-push-service-4165507. On information and belief, Samsung includes applications in its user devices, such as the “Shop Samsung” and “Galaxy Store”
--	--

applications, which cause notifications (including marketing notifications) to be sent to user devices via Samsung's push messaging servers. See, e.g.:



[1c] supporting a plurality of secure Internet data connections, each secure Internet data connection between the network message server and a respective one of the mobile end-user devices via a device data connection to a wireless network,

Samsung's push messaging servers "support[] a plurality of secure Internet data connections, each secure Internet data connection between the network message server and a respective one of the mobile end-user devices via a device data connection to a wireless network."

For example, Samsung's push messaging server supports a plurality of secure connections between the network message server and respective devices. *See, e.g.,*

Push Notification

You can receive notifications from a push server. The push service is a client daemon that maintains a permanent connection between the device and the push server. Push enables you to push events from an application server to your application on a Tizen device. Connection with the push service is used to deliver push notifications to the application, and process the registration and deregistration requests.

The Push API is optional for Tizen Mobile, Wearable, and TV profiles, which means that it may not be supported on all mobile, wearable, and TV devices. The Push API is supported on all Tizen emulators.

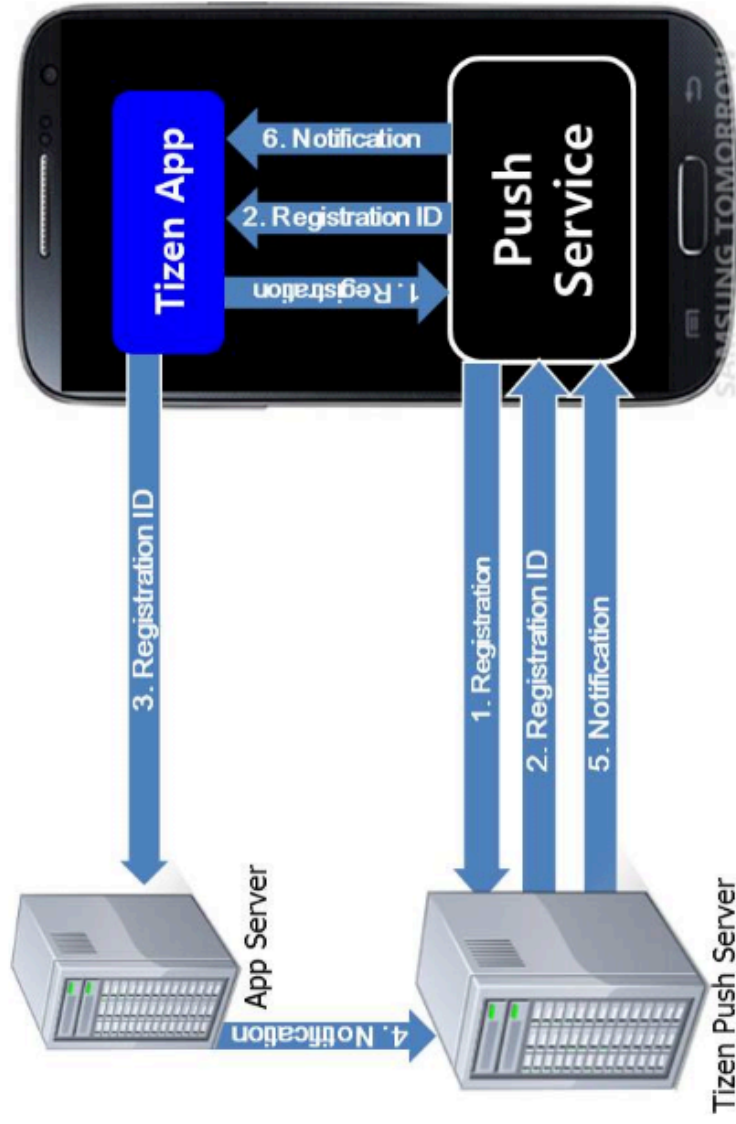
Push notification helps your application server send data to your application on a device over a network, even if the application is not running. Using the push service can reduce battery consumption and data transfer.

If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a **LAUNCH** option, the push service forcibly launches the application and hands over the message to the application.

Architecture

The architecture of the Tizen Push service is described in detail in the [mobile native Push guide](#).

Figure: Service architecture



To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.

Registering to the Push Service

To receive push notifications, you must learn how to register your application to the push service:

- Up to Tizen 2.4:

1. Define event handlers for the registration results:

```
/*
  Define the data to be used when this process
  is launched by the notification service
*/
var service = new tizen.ApplicationControl('http://tizen.org/appcontrol/operation/push_test');

/* Define the error callback */
function errorCallback(response) {
  console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
  console.log('Registration succeeded with id: ' + id);
}
```

2. Register the application for the service with the `register()` method. This operation has to be done only once.

```
/* Request application registration */
tizen.push.registerService(service, registerSuccessCallback, errorCallback);
```

- Since Tizen 3.0:

Before registering, you must connect to the push service:

1. Define event handlers:

```
/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}

/* Define the state change callback */
function stateChangeCallback(state) {
    console.log('The state is changed to: ' + state);
}

if (state == 'UNREGISTERED') {
    /* Request application registration */
    tizen.push.register(registerSuccessCallback, errorCallback);
}

/* Define the notification callback */
function notificationCallback(notification) {
    console.log('A notification arrives.');
```

2. Connect to the push service with the `connect()` method. The `register()` method is called in the `stateChangeCallback()` callback. This operation has to be done only once.

```
/* Connect to push service */
tizen.push.connect(stateChangeCallback, notificationCallback, errorCallback);
```

<https://docs.tizen.org/application/web/guides/messaging/push/>;

Push API

The Push API provides functionality for receiving push notifications from the Tizen push server. The push service is a client daemon that maintains a permanent connection between your device and the Tizen push server.

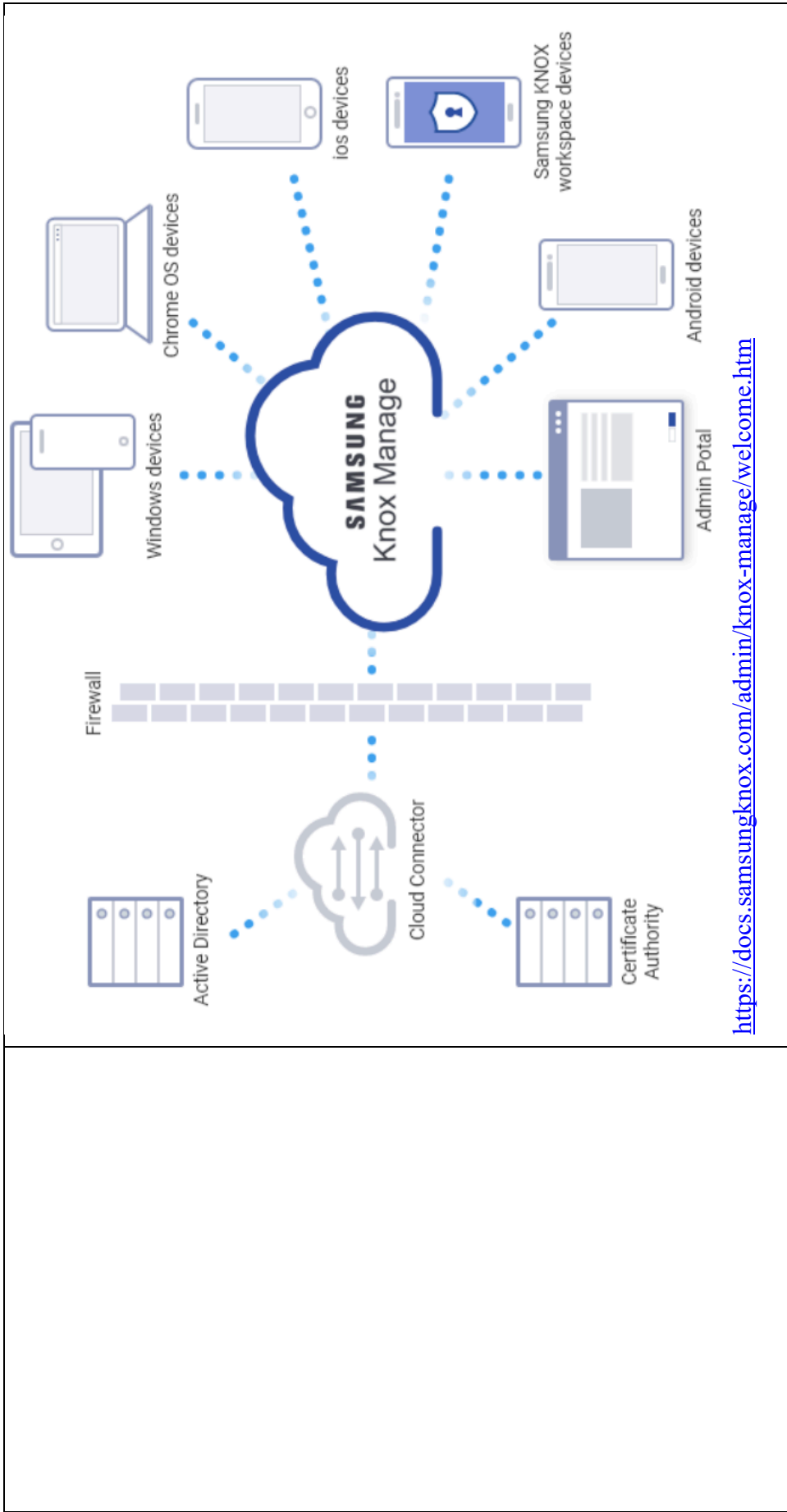
Connection with push server is used to deliver push notifications to the application, and process the registration and deregistration requests.

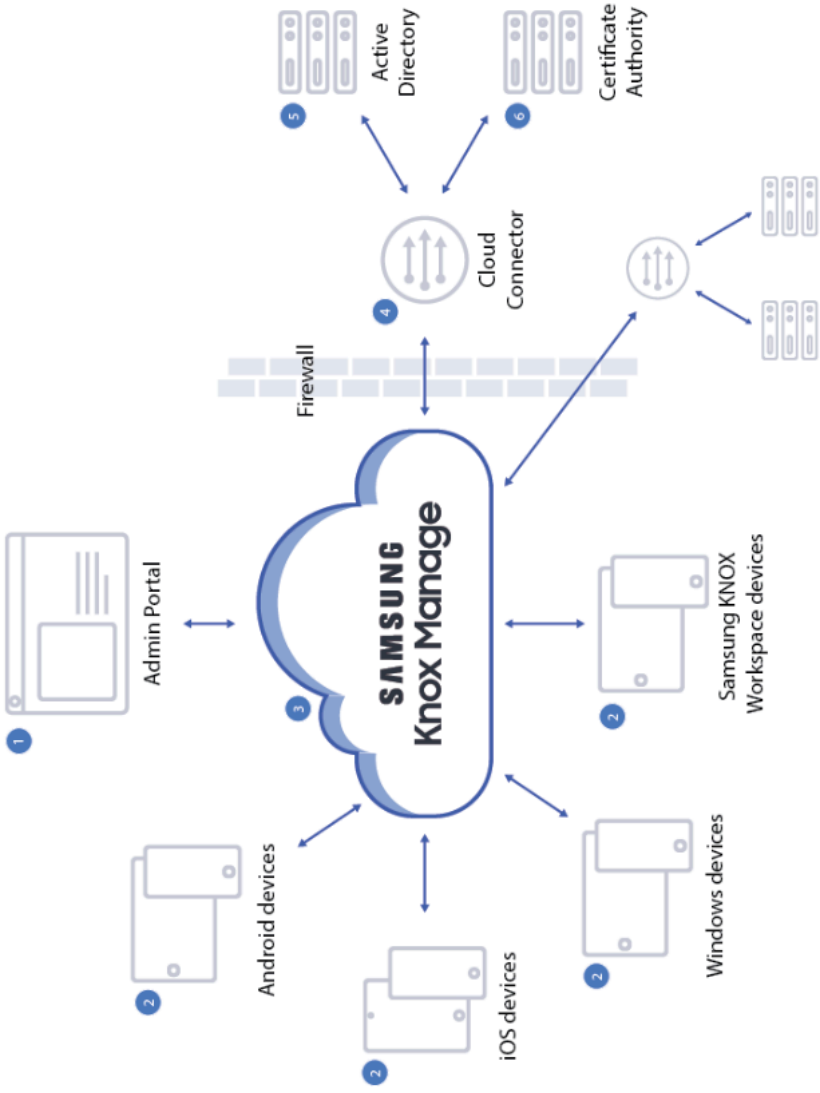
To receive push notifications, follow the steps below:

- Connecting to the push service
- Registering your application, if the application has not been registered yet
- Getting notification data

<https://developer.samsung.com/smarttv/develop/api-references/tizen-web-device-api-references/push-api.html>.

As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, comprise a plurality of secure Internet data connections (between managed devices and Samsung Knox servers).



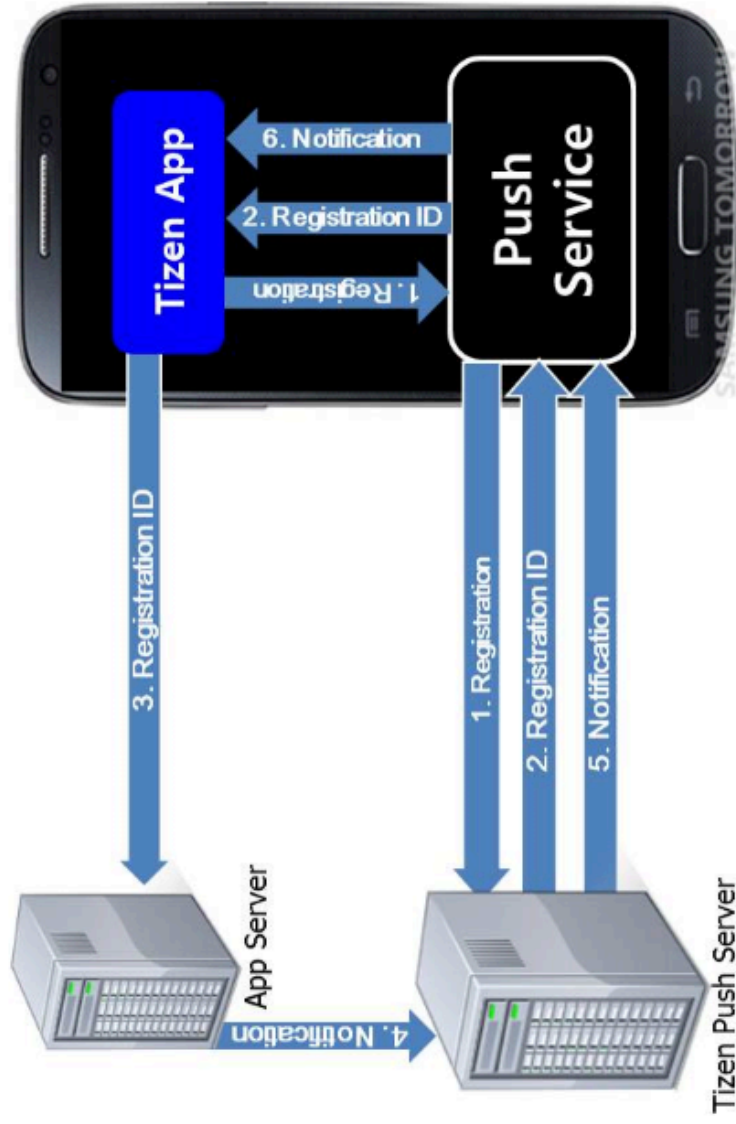
	 https://docs.samsungknox.com/admin/knox-manage/km-features.htm
[1d] the network message server configured to receive, from each of a plurality of network application servers, multiple requests to transmit application data, each such request indicating a corresponding one of the mobile end-user	Samsung's push messaging servers comprise "the network message server configured to receive, from each of a plurality of network application servers, multiple requests to transmit application data, each such request indicating a corresponding one of the mobile end-user devices and one of a plurality of applications."

devices and one of a plurality of applications,	For example, Samsung's push messaging servers receive messages from a plurality of network elements (e.g., App Servers) multiple requests to transmit data indicating a device and application. <i>See, e.g.,</i>
--	--

Architecture

The architecture of the Tizen Push service is described in detail in the [mobile native Push guide](#).

Figure: Service architecture



To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.

<https://docs.tizen.org/application/web/guides/messaging/push/>;

Push Server

You can push events from an application server to your application on a Tizen device. If the message sending fails for any reason, an error code identifying the failure reason is returned. You can use the **error code** to determine how to handle the failure.

The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices.

The main features of the Push API for the server developers include:

- Sending push notifications

You can send **push notifications** from the application server to an application.

- Decorating push notifications

You can add decorations to the push notifications in the quick panel.

<https://docs.tizen.org/application/native/guides/messaging/push-server/>;

Push

You can push events from an application server to your application on a Tizen device.

The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices.

Once your application is successfully registered in the push server through the **push service** (daemon) on the device, your application server can send push messages to the application on that particular device.

If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a **LAUNCH** option, the push service forcibly launches the application and hands over the message to the application as an **application control**.

Figure: Push messaging service



Sending Push Notifications

Once the application successfully sends its registration ID to the application server, you are ready to send push notifications from the application server to the application on that particular device. This use case describes how to send a simple push notification to the device. For advanced features, see the [Push Server](#) guide for server developers.

The following example shows a sample push notification:

- URI: See the [Push RQM \(Request Manager\) server URLs](#) table.

- Method: HTTP POST

- Header:

```
appID: 1234567890987654
appSecret: dYo/o/m11gmMjs7+5f+2zLNV0c=
```

- Body:

```
{
  "regID": "0501a53f4affdcb98197f188345ff30c04b-5001",
  "requestID": "01231-22EAX-223442",
  "message": "badgeOption=INCREASE&badgeNumber=1&action=ALERT&alertMessage=Hi",
  "appData": "{id:asdf&passwd:1234}", /* Optional, if the message field is not empty */
}
```

To send a notification:

1. Prepare the `appID`, `appSecret`, `regID`, and `requestID` :

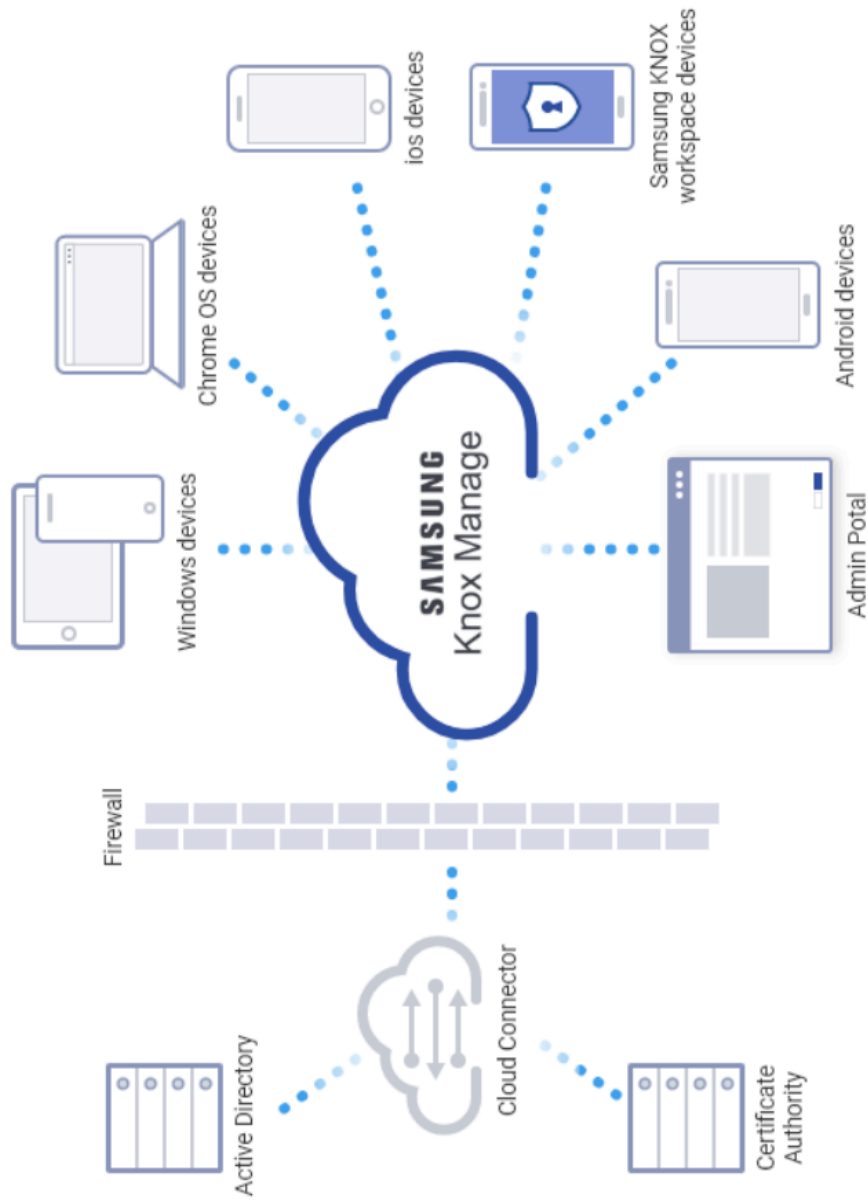
- The `appID` and `appSecret` values are given in the email message that you received when requesting [permission to use Tizen push servers](#).
- The `regID` value is the one that the application server received from your application installed on a Tizen device. Depending on the `regID` value, the URI of the server to which your application server sends the notification varies.
- The `requestID` value is used to identify the notification in the push server. When your application server sends notifications using the same `requestID` value, the last notification overwrites all the previous notifications that are not delivered yet.

2. Use the message field to describe how to process the notification.

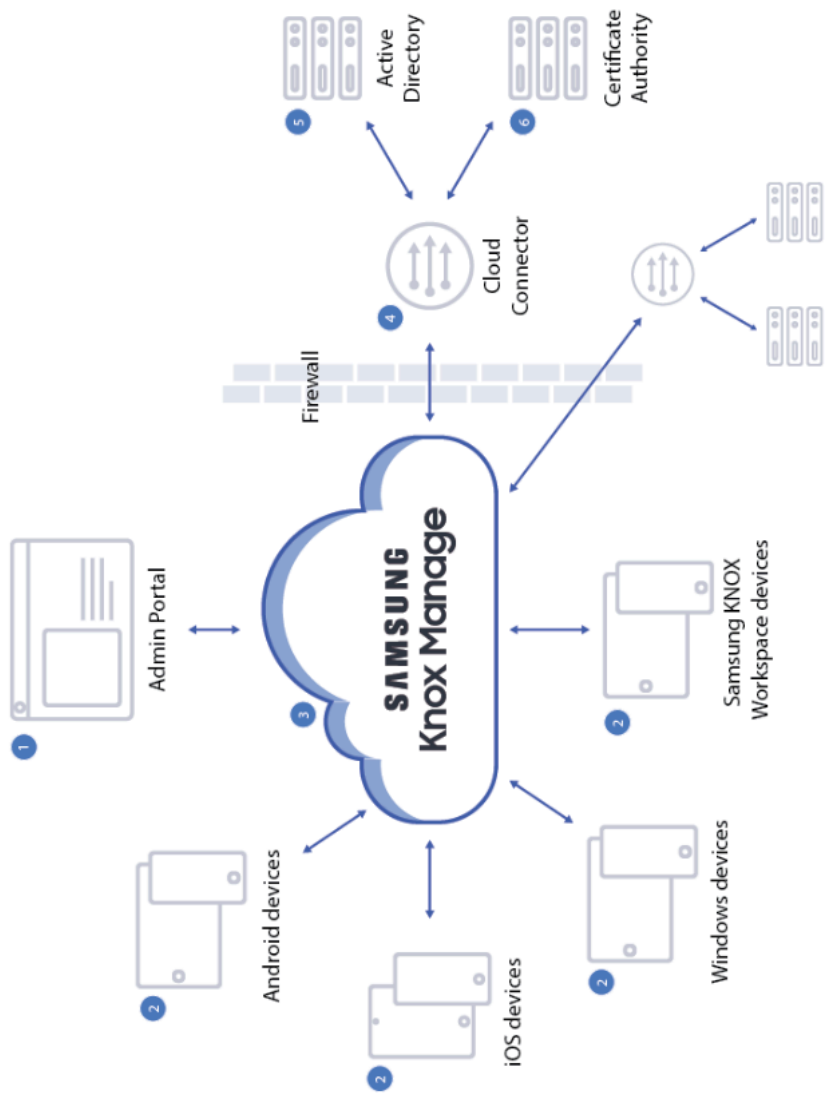
The message field contains not only the message to show in the quick panel on the device, but also the behaviors that the device must take when receiving the notification. The message field is a string that consists of key-value pairs. The available pair options are given in the following table.

<https://docs.tizen.org/application/native/guides/messaging/push/>.

As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, comprise a plurality of network application servers, multiple requests to transmit application data, each such request indicating a corresponding one of the mobile end-user devices and one of a plurality of applications.



<https://docs.samsungknox.com/admin/knox-manage/welcome.htm>



<https://docs.samsungknox.com/admin/knox-manage/km-features.htm>

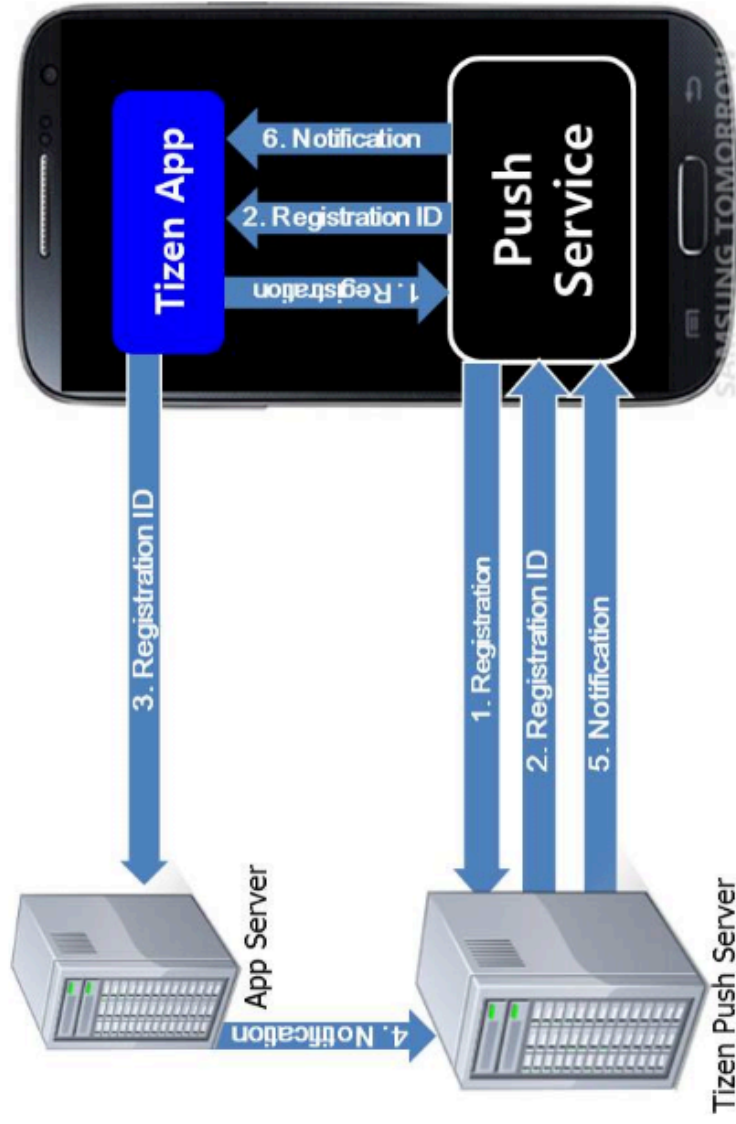
	Update an existing device profile 22.11 23.03 UAT An admin can update the device's profile with a push update if a device is currently in a Configured state. NOTE — Setup edition profiles are restricted from receiving a push update. A Dynamic profile can push update another Dynamic edition profile, and a Setup edition profile can push update a Dynamic edition profile. However, a Setup edition profile cannot update another Setup edition profile, nor can a Dynamic edition profile push update a Setup edition profile. NOTE — An IT admin can select specific devices for push updates from the Knox Configure PROFILE or DEVICES tabs or at the time a profile is modified. Otherwise, each device utilizing the profile will receive the push update whether intended for each device utilizing that profile or not. https://docs.samsungknox.com/admin/knox-configure/updating-an-existing-device-profile.htm Components of Knox Manage 1. Knox Manage console — A web console that allows IT admins to configure, monitor, and manage devices, deploy updates, as well as manage certificates and licenses. 2. Knox Manage MDM Client — An app that is installed on devices to automate installation and enrollment to Knox Manage. 3. Knox Manage Cloud Connector — A service that creates a secure channel for data transfer between your enterprise system and the Knox Manage cloud server. 4. Certificate Authority (CA) — An authority that generates certificates to authenticate devices and users with services such as Wi-Fi, VPN, Exchange, APN, and so on. 5. Active Directory — A Lightweight Directory Access Protocol (LDAP) service that provides access to a customer's directory-based user information. https://docs.samsungknox.com/admin/knox-manage/welcome.htm
[1e] the network message server to generate corresponding Internet data messages based on the requests, each	Samsung's push messaging servers comprise "the network message server to generate corresponding Internet data messages based on the requests, each such message containing at

such message containing at least one application identifier for an indicated application and application data corresponding to one of the requests, and	least one application identifier for an indicated application and application data corresponding to one of the requests.” For example, Samsung’s push messaging servers generate data messages containing an application identifier and data corresponding to the requests. <i>See, e.g.,</i>
--	---

Architecture

The architecture of the Tizen Push service is described in detail in the [mobile native Push guide](#).

Figure: Service architecture



To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.

<https://docs.tizen.org/application/web/guides/messaging/push/>;

Push Server

You can push events from an application server to your application on a Tizen device. If the message sending fails for any reason, an error code identifying the failure reason is returned. You can use the [error code](#) to determine how to handle the failure.

The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices.

The main features of the Push API for the server developers include:

- Sending push notifications

You can send [push notifications](#) from the application server to an application.

- Decorating push notifications

You can add decorations to the push notifications in the quick panel.

<https://docs.tizen.org/application/native/guides/messaging/push-server/>;

Push

You can push events from an application server to your application on a Tizen device.

The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices.

Once your application is successfully registered in the push server through the [push service](#) (daemon) on the device, your application server can send push messages to the application on that particular device.

If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a [LAUNCH](#) option, the push service forcibly launches the application and hands over the message to the application as an [application control](#).

Figure: Push messaging service



Sending Push Notifications

Once the application successfully sends its registration ID to the application server, you are ready to send push notifications from the application server to the application on that particular device. This use case describes how to send a simple push notification to the device. For advanced features, see the [Push Server](#) guide for server developers.

The following example shows a sample push notification:

- URI: See the [Push RQM \(Request Manager\) server URLs](#) table.

- Method: HTTP POST

- Header:

```
appID: 1234567890987654
appSecret: dYo/o/m11gmMjs7+5f+2zLNV0c=
```

- Body:

```
{
  "regID": "0501a53f4affdcb98197f188345ff30c04b-5001",
  "requestID": "01231-22EAX-223442",
  "message": "badgeOption=INCREASE&badgeNumber=1&action=ALERT&alertMessage=Hi",
  "appData": "{id:asdf&passwd:1234}", /* Optional, if the message field is not empty */
}
```

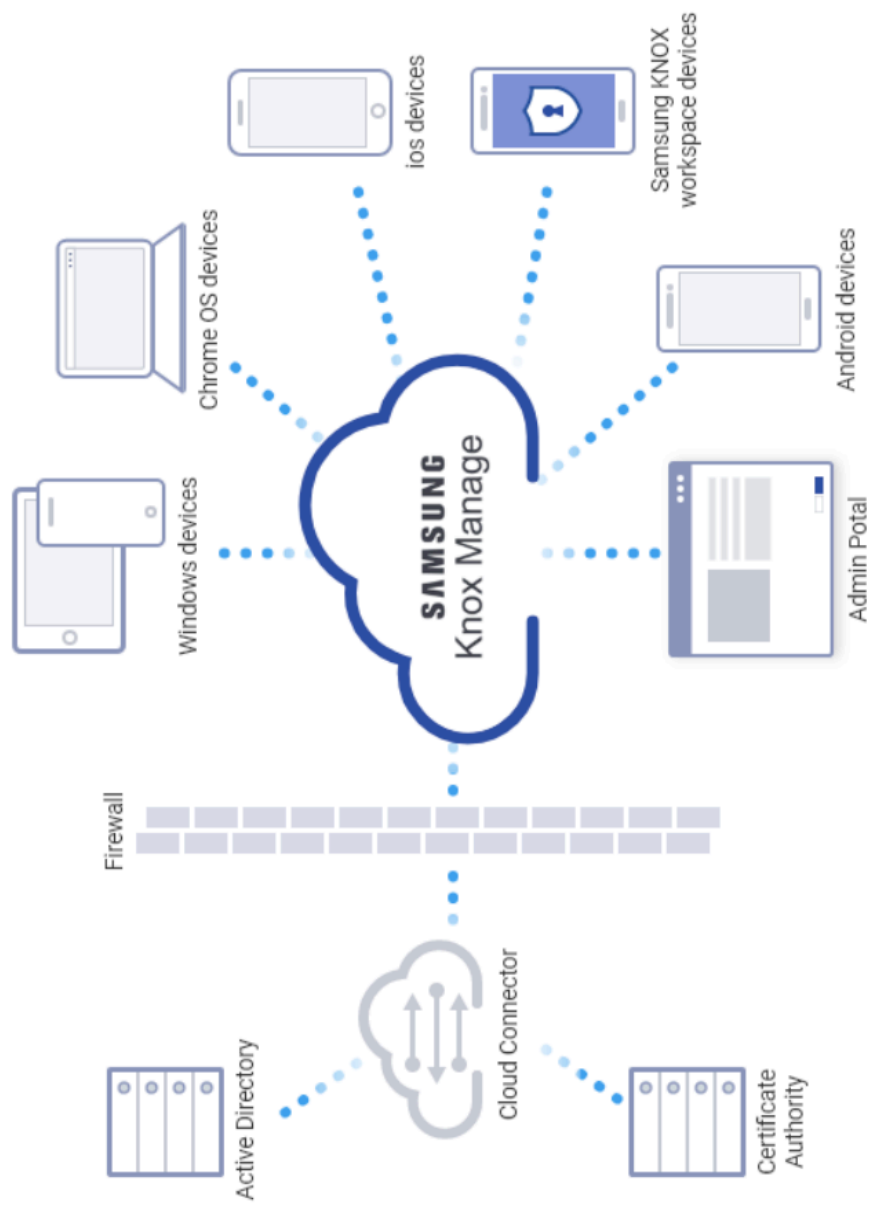
To send a notification:

1. Prepare the `appID`, `appSecret`, `regID`, and `requestID` :
 - The `appID` and `appSecret` values are given in the email message that you received when requesting [permission to use Tizen push servers](#).
 - The `regID` value is the one that the application server received from your application installed on a Tizen device. Depending on the `regID` value, the URI of the server to which your application server sends the notification varies.
 - The `requestID` value is used to identify the notification in the push server. When your application server sends notifications using the same `requestID` value, the last notification overwrites all the previous notifications that are not delivered yet.
2. Use the message field to describe how to process the notification.

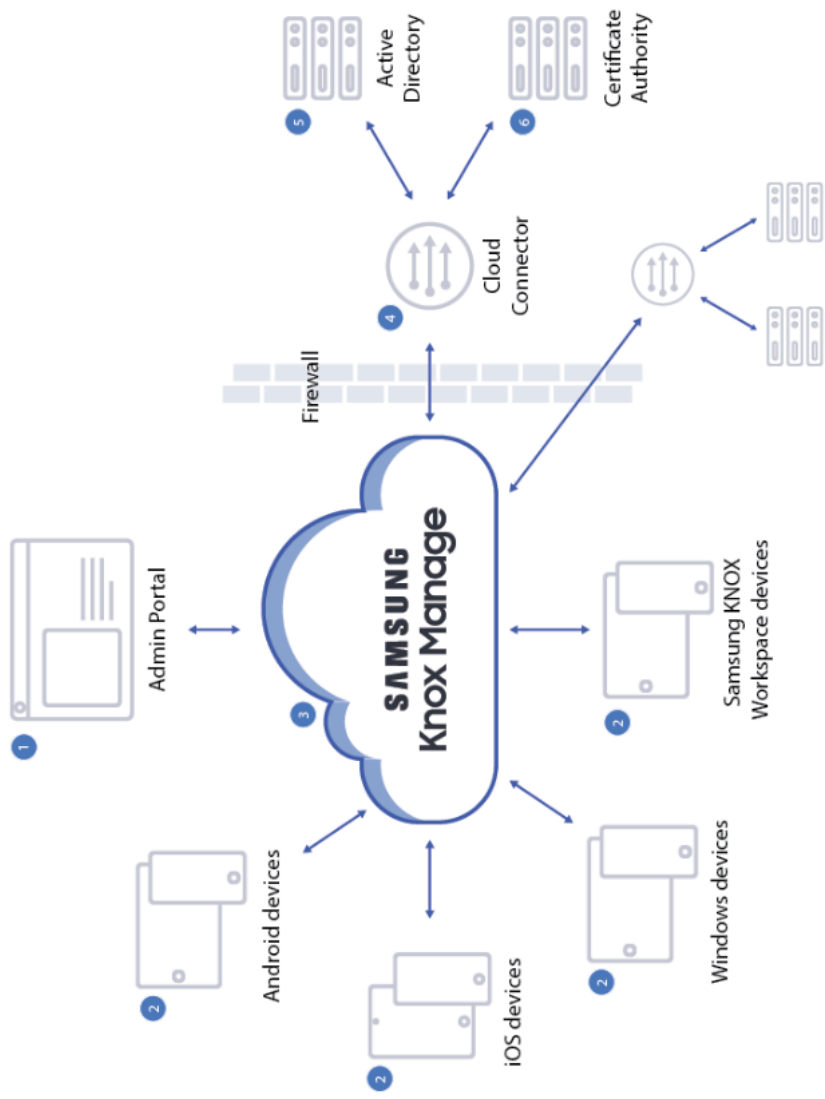
The message field contains not only the message to show in the quick panel on the device, but also the behaviors that the device must take when receiving the notification. The message field is a string that consists of key-value pairs. The available pair options are given in the following table.

<https://docs.tizen.org/application/native/guides/messaging/push/>.

As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, comprise servers which generate data messages containing an application identifier and data corresponding to the requests.



<https://docs.samsungknox.com/admin/knox-manage/welcome.htm>



<https://docs.samsungknox.com/admin/knox-manage/km-features.htm>

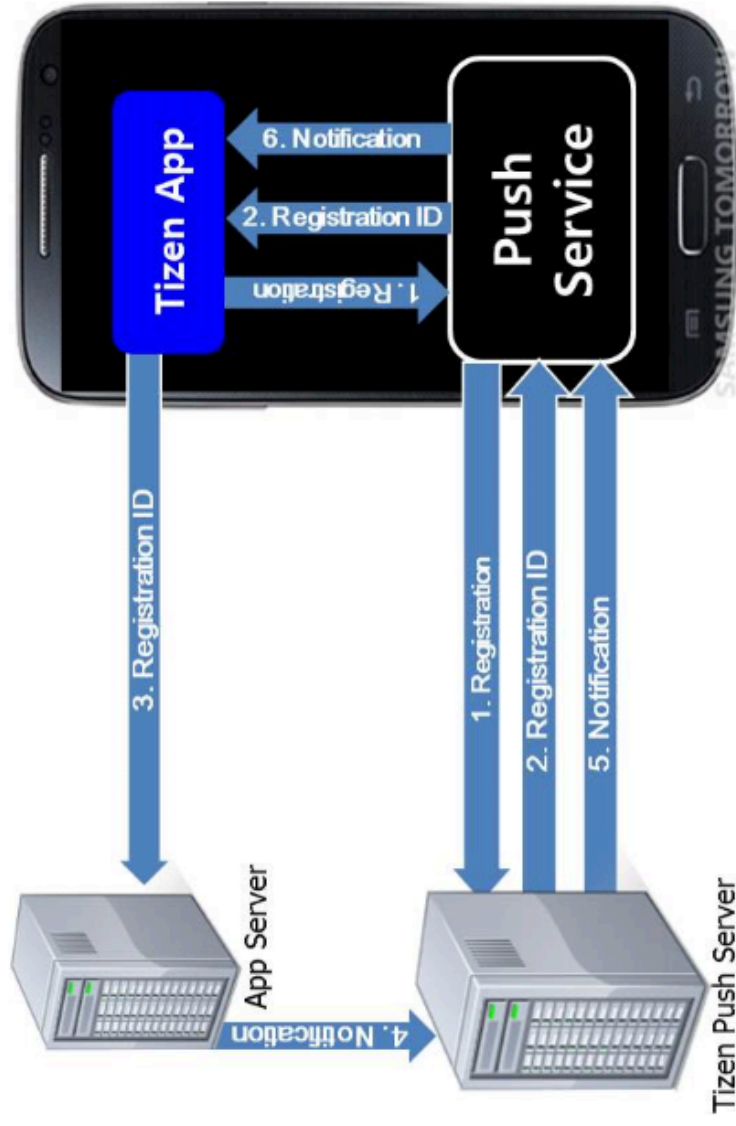
	Update an existing device profile 22.11 23.03 UAT An admin can update the device's profile with a push update if a device is currently in a Configured state. NOTE — Setup edition profiles are restricted from receiving a push update. A Dynamic profile can push update another Dynamic edition profile, and a Setup edition profile can push update a Dynamic edition profile. However, a Setup edition profile cannot update another Setup edition profile, nor can a Dynamic edition profile push update a Setup edition profile. NOTE — An IT admin can select specific devices for push updates from the Knox Configure PROFILE or DEVICES tabs or at the time a profile is modified. Otherwise, each device utilizing the profile will receive the push update whether intended for each device utilizing that profile or not. https://docs.samsungknox.com/admin/knox-configure/updating-an-existing-device-profile.htm Components of Knox Manage 1. Knox Manage console — A web console that allows IT admins to configure, monitor, and manage devices, deploy updates, as well as manage certificates and licenses. 2. Knox Manage MDM Client — An app that is installed on devices to automate installation and enrollment to Knox Manage. 3. Knox Manage Cloud Connector — A service that creates a secure channel for data transfer between your enterprise system and the Knox Manage cloud server. 4. Certificate Authority (CA) — An authority that generates certificates to authenticate devices and users with services such as Wi-Fi, VPN, Exchange, APN, and so on. 5. Active Directory — A Lightweight Directory Access Protocol (LDAP) service that provides access to a customer's directory-based user information. https://docs.samsungknox.com/admin/knox-manage/welcome.htm
[1f] the network message server to transmit each of the generated Internet data messages to the device messaging	Samsung's push messaging servers comprise “the network message server to transmit each of the generated Internet data messages to the device messaging agent located on the device

agent located on the device indicated in the corresponding request, using the corresponding secure Internet data connection for the device indicated in the corresponding request;	indicated in the corresponding request, using the corresponding secure Internet data connection for the device indicated in the corresponding request.” For example, Samsung’s push messaging servers transmit the generated data messages to the device agents on the devices. <i>See, e.g.,</i>
---	---

Architecture

The architecture of the Tizen Push service is described in detail in the [mobile native Push guide](#).

Figure: Service architecture



To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.

<https://docs.tizen.org/application/web/guides/messaging/push/>;

Push Server

You can push events from an application server to your application on a Tizen device. If the message sending fails for any reason, an error code identifying the failure reason is returned. You can use the **error code** to determine how to handle the failure.

The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices.

The main features of the Push API for the server developers include:

- Sending push notifications

You can send **push notifications** from the application server to an application.

- Decorating push notifications

You can add decorations to the push notifications in the quick panel.

<https://docs.tizen.org/application/native/guides/messaging/push-server/>;

Push

You can push events from an application server to your application on a Tizen device.

The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices.

Once your application is successfully registered in the push server through the **push service** (daemon) on the device, your application server can send push messages to the application on that particular device.

If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a **LAUNCH** option, the push service forcibly launches the application and hands over the message to the application as an **application control**.

Figure: Push messaging service



Sending Push Notifications

Once the application successfully sends its registration ID to the application server, you are ready to send push notifications from the application server to the application on that particular device. This use case describes how to send a simple push notification to the device. For advanced features, see the [Push Server](#) guide for server developers.

The following example shows a sample push notification:

- URI: See the [Push RQM \(Request Manager\) server URLs](#) table.

- Method: HTTP POST

- Header:

```
appID: 1234567890987654
appSecret: dYo/o/m11gmMjs7+5f+2zLNV0c=
```

- Body:

```
{
  "regID": "0501a53f4affdcb98197f188345ff30c04b-5001",
  "requestID": "01231-22EAX-223442",
  "message": "badgeOption=INCREASE&badgeNumber=1&action=ALERT&alertMessage=Hi",
  "appData": "{id:asdf&passwd:1234}", /* Optional, if the message field is not empty */
}
```

To send a notification:

1. Prepare the `appID`, `appSecret`, `regID`, and `requestID` :

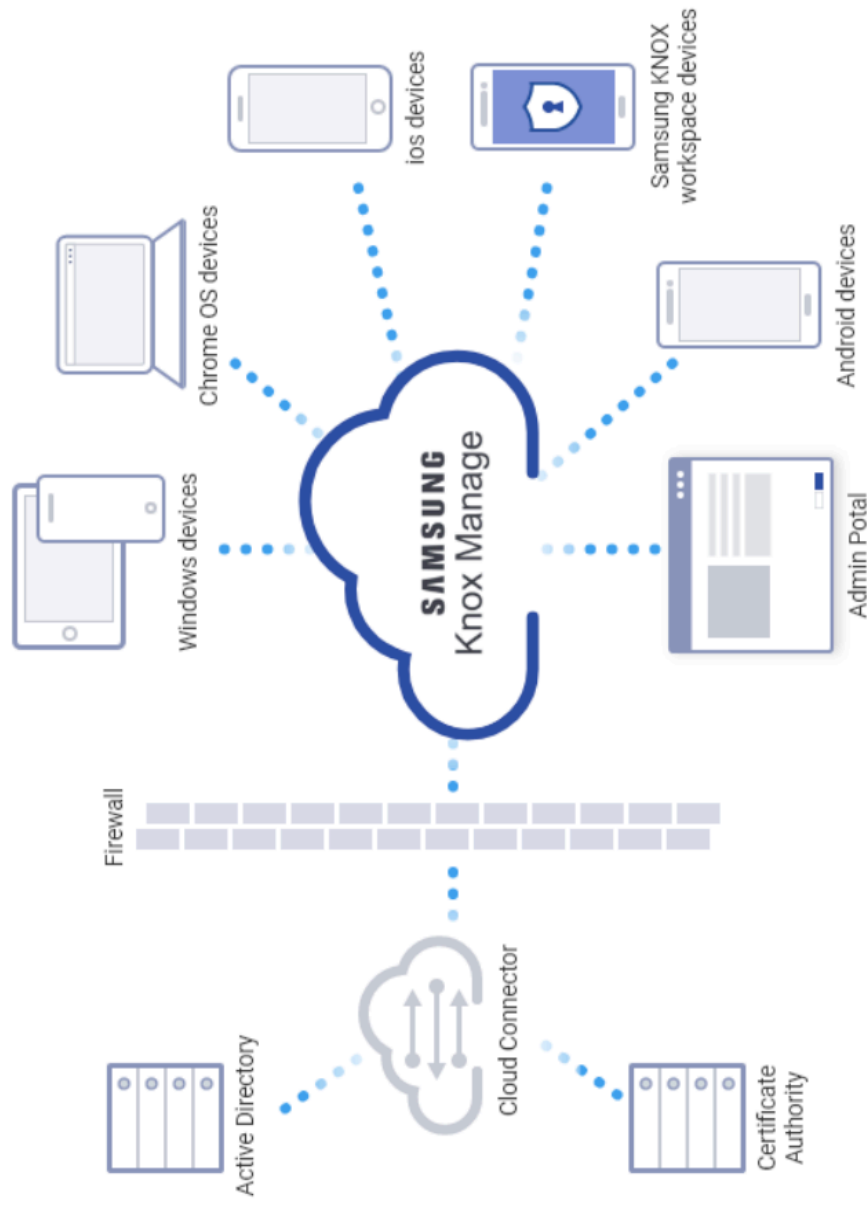
- The `appID` and `appSecret` values are given in the email message that you received when requesting [permission to use Tizen push servers](#).
- The `regID` value is the one that the application server received from your application installed on a Tizen device. Depending on the `regID` value, the URI of the server to which your application server sends the notification varies.
- The `requestID` value is used to identify the notification in the push server. When your application server sends notifications using the same `requestID` value, the last notification overwrites all the previous notifications that are not delivered yet.

2. Use the message field to describe how to process the notification.

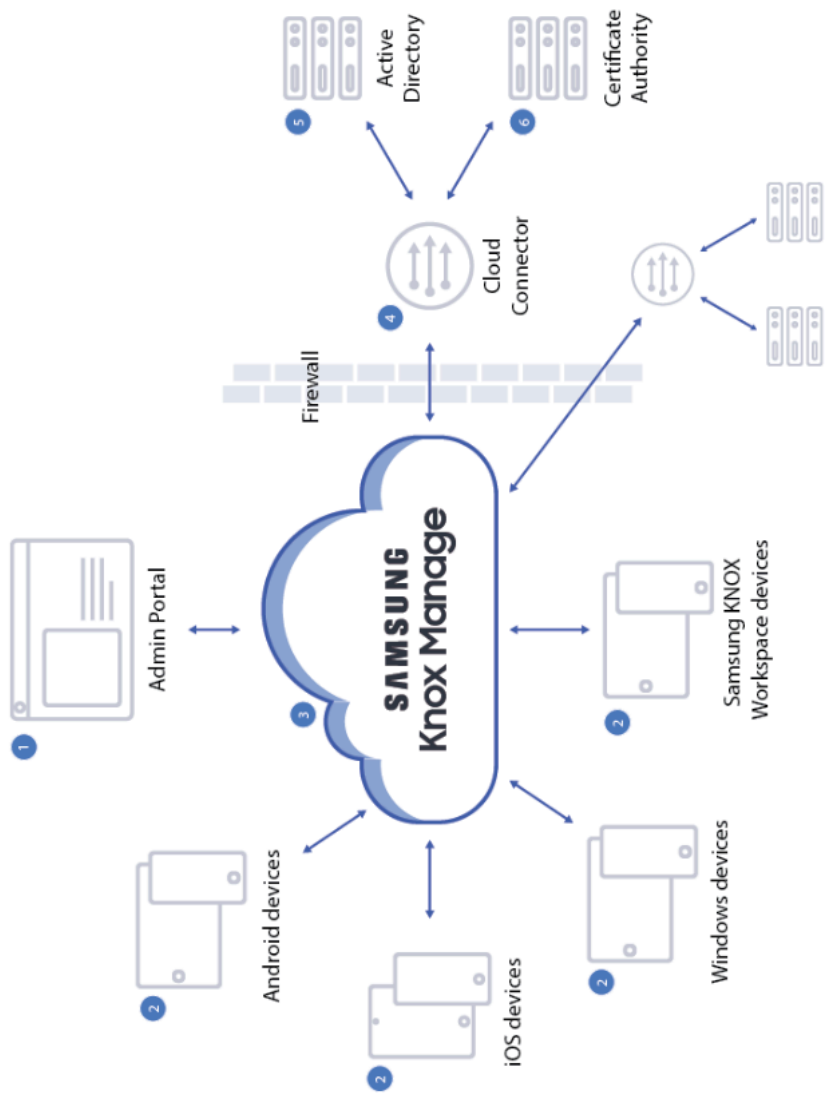
The message field contains not only the message to show in the quick panel on the device, but also the behaviors that the device must take when receiving the notification. The message field is a string that consists of key-value pairs. The available pair options are given in the following table.

<https://docs.tizen.org/application/native/guides/messaging/push/>.

As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, comprise servers which transmit the generated data messages to the device agents on the devices.



<https://docs.samsungknox.com/admin/knox-manage/welcome.htm>



<https://docs.samsungknox.com/admin/knox-manage/km-features.htm>

Update an existing device profile

22.11	23.03 UAT
-------	-----------

An admin can update the device's profile with a push update if a device is currently in a Configured state.

NOTE — Setup edition profiles are restricted from receiving a push update. A Dynamic profile can push update another Dynamic edition profile, and a Setup edition profile can push update a Dynamic edition profile. However, a Setup edition profile cannot update another Setup edition profile, nor can a Dynamic edition profile push update a Setup edition profile.

NOTE — An IT admin can select specific devices for push updates from the Knox Configure **PROFILE** or **DEVICES** tabs or at the time a profile is modified. Otherwise, each device utilizing the profile will receive the push update whether intended for each device utilizing that profile or not.

<https://docs.samsungknox.com/admin/knox-configure/updating-an-existing-device-profile.htm>

Components of Knox Manage

1. **Knox Manage console** — A web console that allows IT admins to configure, monitor, and manage devices, deploy updates, as well as manage certificates and licenses.
2. **Knox Manage MDM Client** — An app that is installed on devices to automate installation and enrollment to Knox Manage.
3. **Knox Manage Cloud Connector** — A service that creates a secure channel for data transfer between your enterprise system and the Knox Manage cloud server.
4. **Certificate Authority (CA)** — An authority that generates certificates to authenticate devices and users with services such as Wi-Fi, VPN, Exchange, APN, and so on.
5. **Active Directory** — A Lightweight Directory Access Protocol (LDAP) service that provides access to a customer's directory-based user information.

<https://docs.samsungknox.com/admin/knox-manage/welcome.htm>

[1g] each device messaging agent, when executing, to receive the Internet data messages from the secure Internet data connection corresponding to the device executing the device messaging agent, and

Samsung's devices each comprise a "device messaging agent, when executing, to receive the Internet data messages from the secure Internet data connection corresponding to the device executing the device messaging agent."

For example, Samsung devices receive data over a secure internet data connection. *See e.g.*,

Firestore > Documentation > FCM > Engage

Was this helpful? 

Set up a Firebase Cloud Messaging client app on Android

FCM clients require devices running Android 4.4 or higher that also have the Google Play Store app installed, or an emulator running Android 4.4 with Google APIs. Note that you are not limited to deploying your Android apps through Google Play Store.

Set up the SDK

This section covers tasks you may have completed if you have already enabled other Firebase features for your app. If you haven't already, [add Firebase to your Android project](#)

[Send feedback](#)

58

Page 58 of 104

Edit your app manifest

Add the following to your app's manifest:

- A service that extends `FirebaseMessagingService`. This is required if you want to do any message handling beyond receiving notifications on apps in the background. To receive notifications in foregrounded apps, to receive data payload, to send upstream messages, and so on, you must extend this service.

```
<service
    android:name=".java.MyFirebaseMessagingService"
    android:exported="false">
    <intent-filter>
        <action android:name="com.google.firebase.messaging.MESSAGEING_EVENT" />
    </intent-filter>
</service>
```

AndroidManifest.xml

Access the device registration token

On initial startup of your app, the FCM SDK generates a registration token for the client app instance. If you want to target single devices or create device groups, you'll need to access this token by extending `FirebaseMessagingService` and overriding `onNewToken`.

This section describes how to retrieve the token and how to monitor changes to the token. Because the token could be rotated after initial startup, you are strongly recommended to retrieve the latest updated registration token.

The registration token may change when:

- The app is restored on a new device
- The user uninstalls/reinstall the app
- The user clears app data.

<https://firebase.google.com/docs/cloud-messaging/android/client>;

Receive messages in an Android app

[Send feedback](#)

Firebase notifications behave differently depending on the foreground/background state of the receiving app. If you want foregrounded apps to receive notification messages or data messages, you'll need to write code to handle the `onMessageReceived` callback. For an explanation of the difference between notification and data messages, see [Message types](#).

Handling messages

To receive messages, use a service that extends [FirebaseMessagingService](#). Your service should override the `onMessageReceived` and `onDeletedMessages` callbacks. It should handle any message within 20 seconds of receipt (10 seconds on Android Marshmallow). The time window may be shorter depending on OS delays incurred ahead of calling `onMessageReceived`. After that time, various OS behaviors such as Android O's [background execution limits](#) may interfere with your ability to complete your work. For more information see our overview on [message priority](#).

`onMessageReceived` is provided for most message types, with the following exceptions:

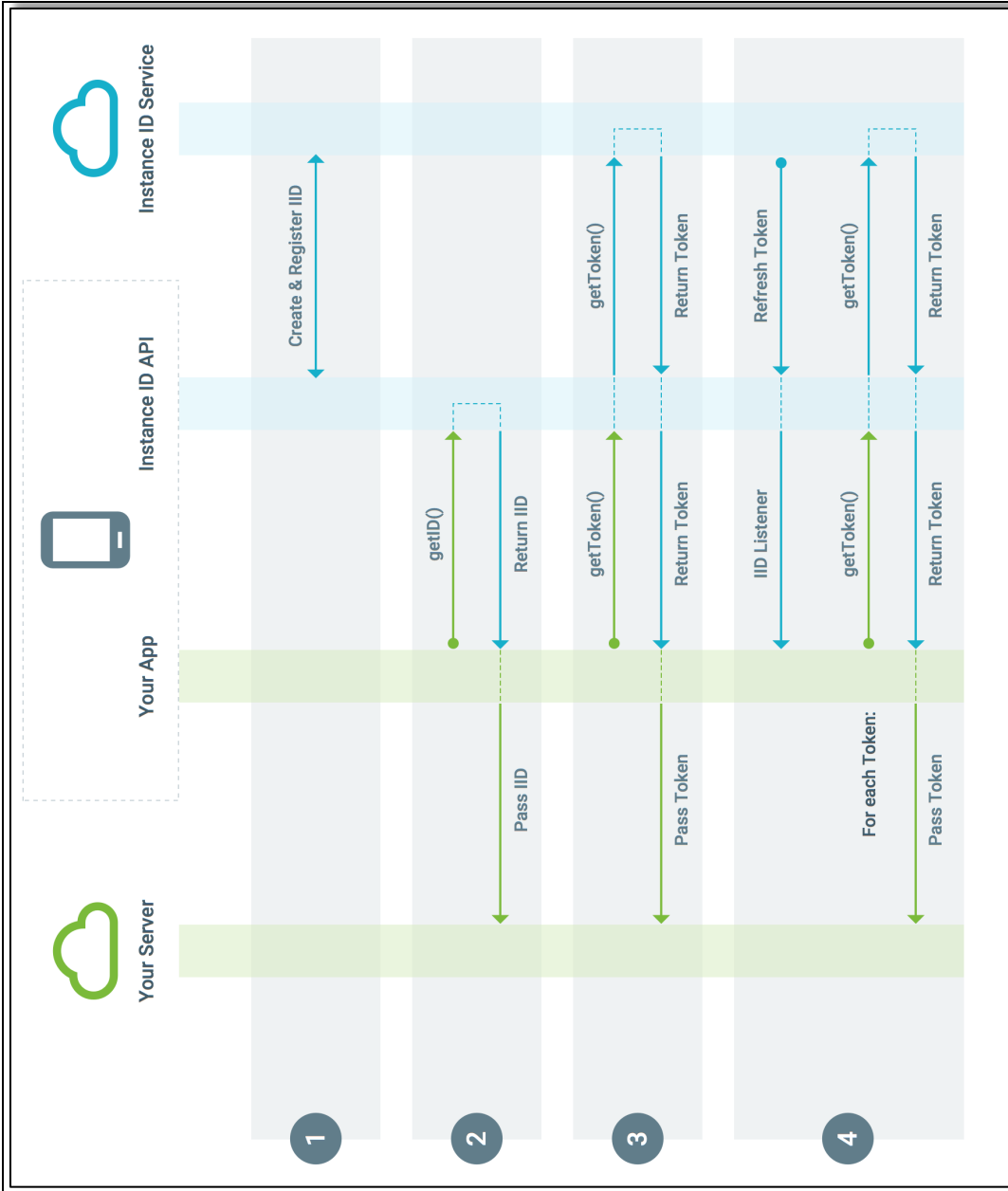
- **Notification messages delivered when your app is in the background.** In this case, the notification is delivered to the device's system tray. A user tap on a notification opens the app launcher by default.
- **Messages with both notification and data payload, when received in the background.** In this case, the notification is delivered to the device's system tray, and the data payload is delivered in the extras of the intent of your launcher Activity.

<https://firebase.google.com/docs/cloud-messaging/android/receive;>

What is Instance ID?

[Send feedback](#)

Instance ID provides a unique ID per instance of your apps. You can implement Instance ID for [Android](#) and [iOS](#) apps as well as [Chrome apps/extensions](#).



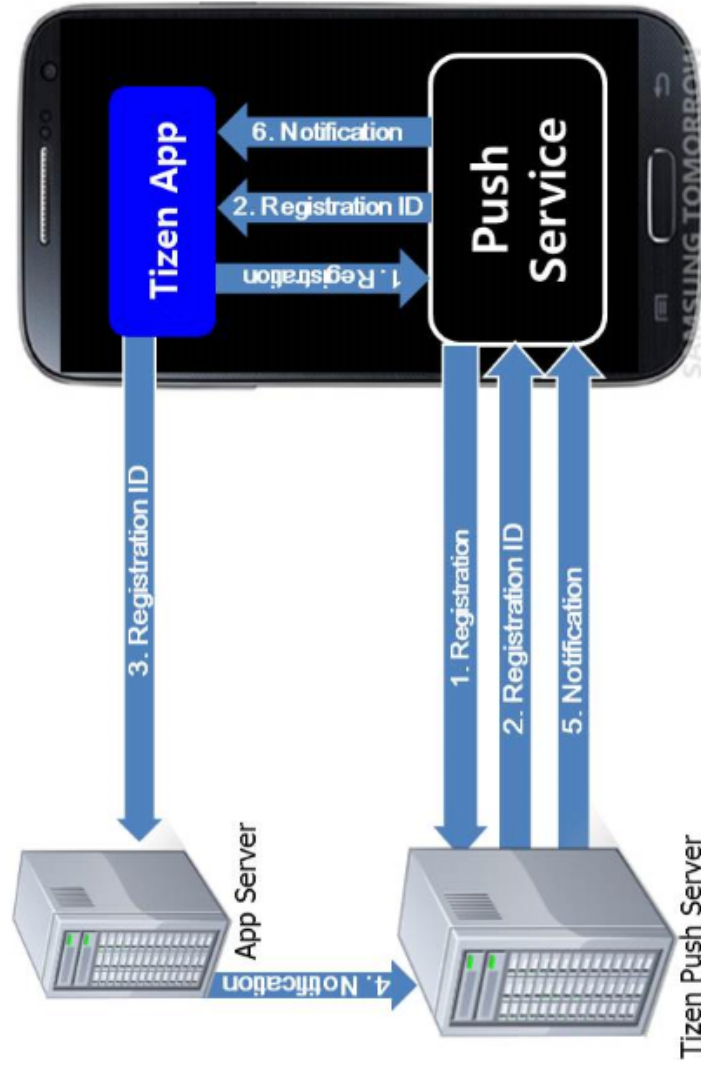
<https://developers.google.com/instance-id>.

For further example, Samsung's Tizen based devices receive data over a secure internet data connection. See e.g.,

Architecture

The architecture of the Tizen Push service is described in detail in the [mobile native Push guide](#).

Figure: Service architecture



To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.

Push Notification

You can receive notifications from a push server. The push service is a client daemon that maintains a permanent connection between the device and the push server. Push enables you to push events from an application server to your application on a Tizen device. Connection with the push service is used to deliver push notifications to the application, and process the registration and deregistration requests.

The Push API is optional for Tizen Mobile, Wearable, and TV profiles, which means that it may not be supported on all mobile, wearable, and TV devices. The Push API is supported on all Tizen emulators.

Push notification helps your application server send data to your application on a device over a network, even if the application is not running. Using the push service can reduce battery consumption and data transfer.

If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a **LAUNCH** option, the push service forcibly launches the application and hands over the message to the application.

The main features of the Push API include:

- Registering to the push service

You can [register to the push service](#) and get the registration identifier.

Registering to the Push Service

To receive push notifications, you must learn how to register your application to the push service:

- Up to Tizen 2.4:

1. Define event handlers for the registration results:

```
/* Define the data to be used when this process
   is launched by the notification service
*/
var service = new tizen.ApplicationControl('http://tizen.org/appcontrol/operation/push_test');

/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}
```

2. Register the application for the service with the `register()` method. This operation has to be done only once.

```
/* Request application registration */
tizen.push.registerService(service, registerSuccessCallback, errorCallback);
```

- Since Tizen 3.0:

Before registering, you must connect to the push service:

1. Define event handlers:

```
/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}

/* Define the state change callback */
function stateChangeCallback(state) {
    console.log('The state is changed to: ' + state);
}

if (state == 'UNREGISTERED') {
    /* Request application registration */
    tizen.push.register(registerSuccessCallback, errorCallback);
}

/* Define the notification callback */
function notificationCallback(notification) {
    console.log('A notification arrives.');
```

2. Connect to the push service with the `connect()` method. The `register()` method is called in the `stateChangeCallback()` callback. This operation has to be done only once.

```
/* Connect to push service */
tizen.push.connect(stateChangeCallback, notificationCallback, errorCallback);
```

If the registration is successful, the `registerSuccessCallback()` callback is called, and the registration ID is passed as a parameter. Any time after a successful registration, you can get the registration ID using the `getRegistrationId()` method:

```
var registrationId = tizen.push.getRegistrationId();
if (registrationId != null) {
    console.log('The registration id: ' + registrationId);
}
```

Receiving Push Notifications

You can connect to the push service and start receiving push notifications with the `connectService()` method up to Tizen 2.4, or with the `connect()` method since Tizen 3.0. Up to Tizen 2.4, you must pass the `PushNotificationCallback` listener (in [mobile](#) and [wearable](#) applications) as a parameter in the method to receive push notifications. Since Tizen 3.0, you must pass the

`PushRegistrationStateChangeCallback` (in [mobile](#), [wearable](#), and [TV](#) applications) and `PushNotificationCallback` callbacks (in [mobile](#), [wearable](#), and [TV](#) applications) as parameters in the method. The first callback is called when the registration change state changes. This callback is called at least once, just after the connection is established. The second callback is called when notification messages arrive. You can also pass the `ErrorCallback` as a parameter to be called if the connection request fails.

When a notification arrives at the device, its delivery mechanism depends on whether the application is running:

- When the application is running

When a notification arrives to the application while it is running (precisely, the application is connected to the service), the push notification callback is called. In this callback, you can read and process the received notification as described in this use case.

- When the application is not running

If the notification arrives when the application is not running, there are 3 ways to handle the notification:

- Forcibly launch the application and deliver the notification to it.

This happens when the action is set to `LAUNCH` in the message field when sending the notification from the application server. When the notification action arrives at the device, the push service forcibly launches the application and delivers the notification as a bundle.

For more information, see the [Retrieving Messages When Launched by the Push Service](#) use case.

- Store the notification at the push service database and request it later when the application is launched.

This happens when the action is set to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application.

For more information, see the [Retrieving Missed Push Messages](#) use case.

The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification.

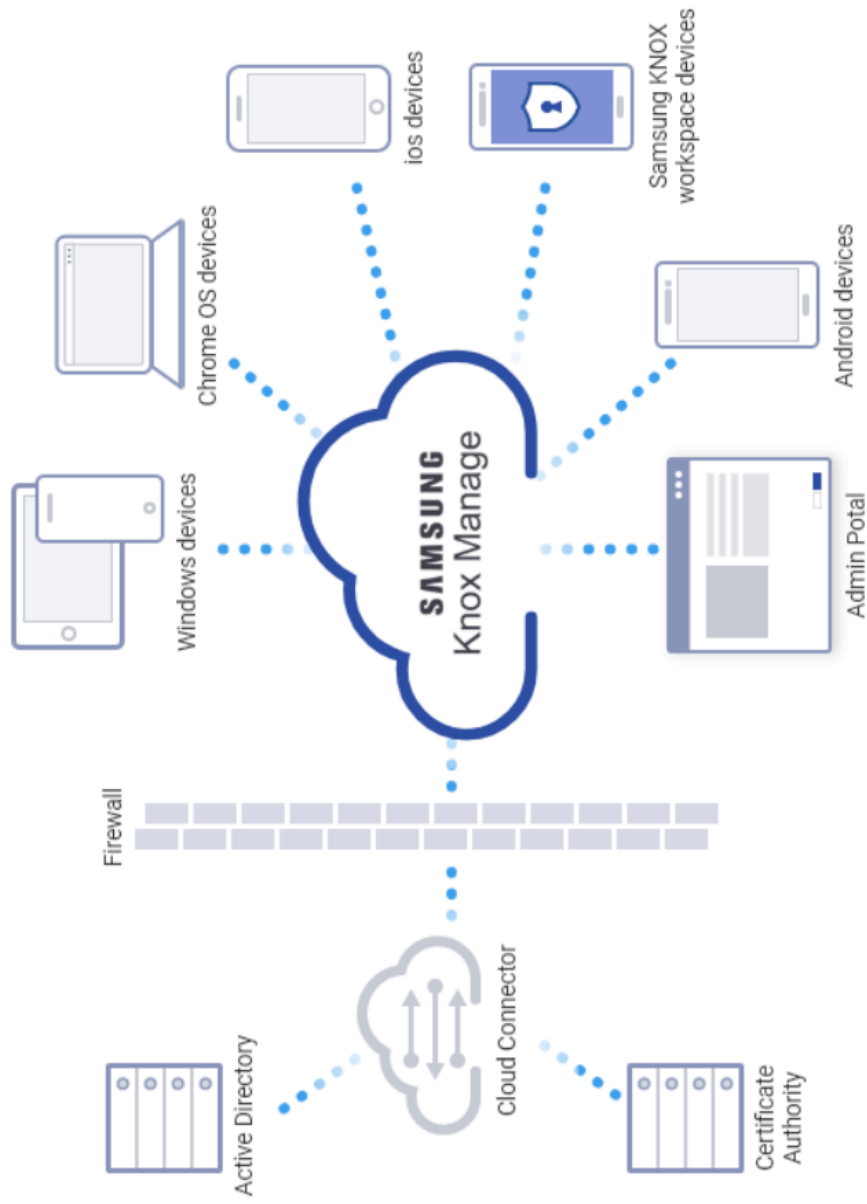
- Discard the notification.

This happens when the action is set to `DISCARD` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service discards the notification unless the application is running.

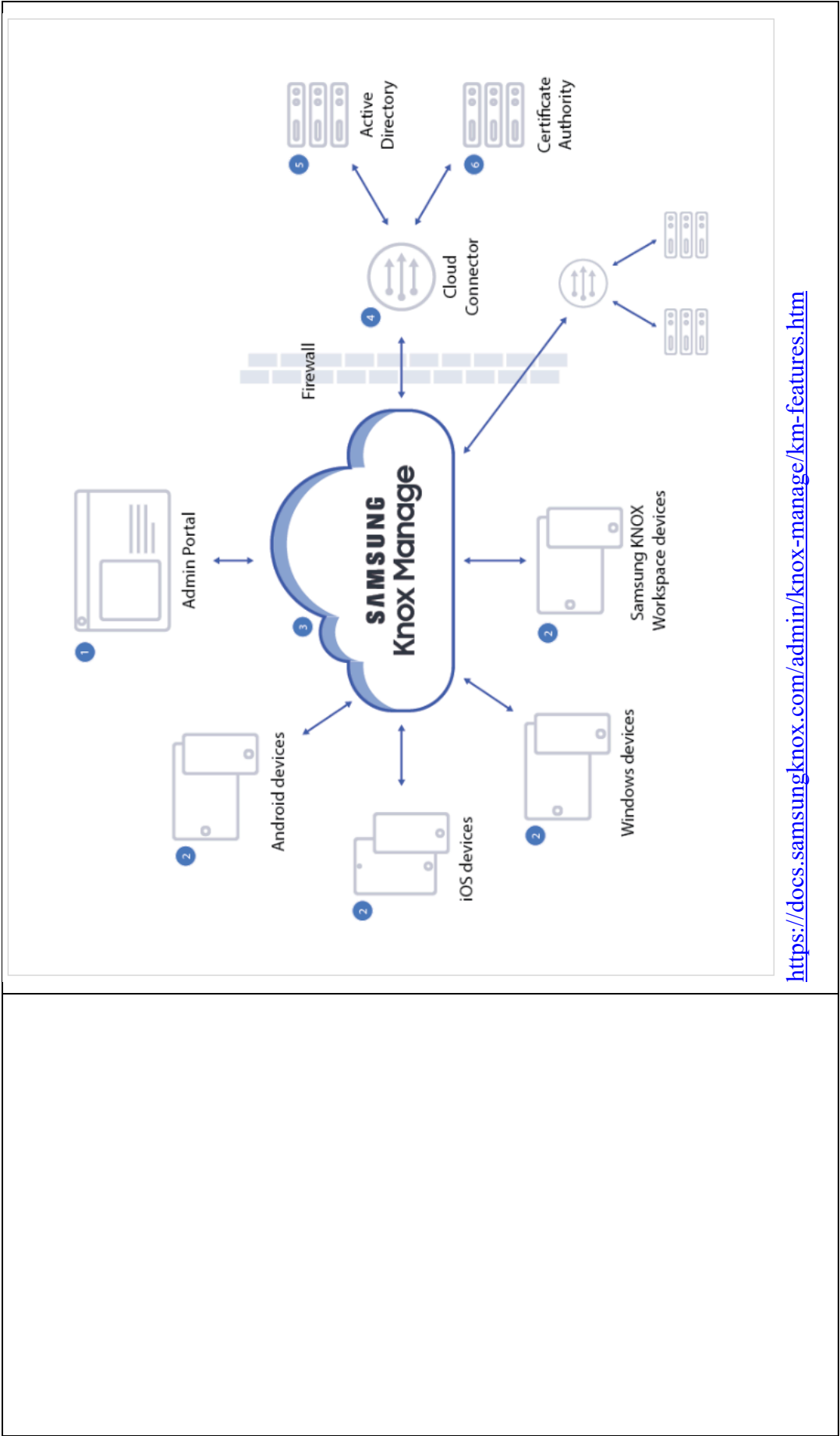
<https://docs.tizen.org/application/web/guides/messaging/push/>.

As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the

Samsung Knox servers managing those phones and devices, comprise devices receiving data over a secure connection.



<https://docs.samsungknox.com/admin/knox-manage/welcome.htm>



Update an existing device profile

22.11 23.03 UAT

An admin can update the device's profile with a push update if a device is currently in a Configured state.

NOTE — Setup edition profiles are restricted from receiving a push update. A Dynamic profile can push update another Dynamic edition profile, and a Setup edition profile can push update a Dynamic edition profile. However, a Setup edition profile cannot update another Setup edition profile, nor can a Dynamic edition profile push update a Setup edition profile.

NOTE — An IT admin can select specific devices for push updates from the Knox Configure **PROFILE** or **DEVICES** tabs or at the time a profile is modified. Otherwise, each device utilizing the profile will receive the push update whether intended for each device utilizing that profile or not.

<https://docs.samsungknox.com/admin/knox-configure/updating-an-existing-device-profile.htm>

Components of Knox Manage

1. **Knox Manage console** — A web console that allows IT admins to configure, monitor, and manage devices, deploy updates, as well as manage certificates and licenses.
2. **Knox Manage MDM Client** — An app that is installed on devices to automate installation and enrollment to Knox Manage.
3. **Knox Manage Cloud Connector** — A service that creates a secure channel for data transfer between your enterprise system and the Knox Manage cloud server.
4. **Certificate Authority (CA)** — An authority that generates certificates to authenticate devices and users with services such as Wi-Fi, VPN, Exchange, APN, and so on.
5. **Active Directory** — A Lightweight Directory Access Protocol (LDAP) service that provides access to a customer's directory-based user information.

<https://docs.samsungknox.com/admin/knox-manage/welcome.htm>

[1h] to, for each received message, map the application identifier in the message to a software process corresponding to the application identifier, and forward the application data in the message to the software process via a secure interprocess communication service.

Samsung's devices "for each received message, map the application identifier in the message to a software process corresponding to the application identifier, and forward the application data in the message to the software process via a secure interprocess communication service."

For example, Samsung devices map the identifier in the message to a corresponding software process and forward the data to the application via a secure service. See e.g.,

Access the device registration token

On initial startup of your app, the FCM SDK generates a registration token for the client app instance. If you want to target single devices or create device groups, you'll need to access this token by extending `FirebaseMessagingService` and overriding `onNewToken`.

This section describes how to retrieve the token and how to monitor changes to the token. Because the token could be rotated after initial startup, you are strongly recommended to retrieve the latest updated registration token.

The registration token may change when:

- The app is restored on a new device
- The user uninstalls/reinstall the app
- The user clears app data.

<https://firebase.google.com/docs/cloud-messaging/android/client>;

Receive messages in an Android app

[Send feedback](#)

Firebase notifications behave differently depending on the foreground/background state of the receiving app. If you want foregrounded apps to receive notification messages or data messages, you'll need to write code to handle the `onMessageReceived` callback. For an explanation of the difference between notification and data messages, see [Message types](#).

Handling messages

To receive messages, use a service that extends [FirebaseMessagingService](#). Your service should override the `onMessageReceived` and `onDeletedMessages` callbacks. It should handle any message within 20 seconds of receipt (10 seconds on Android Marshmallow). The time window may be shorter depending on OS delays incurred ahead of calling `onMessageReceived`. After that time, various OS behaviors such as Android O's [background execution limits](#) may interfere with your ability to complete your work. For more information see our overview on [message priority](#).

`onMessageReceived` is provided for most message types, with the following exceptions:

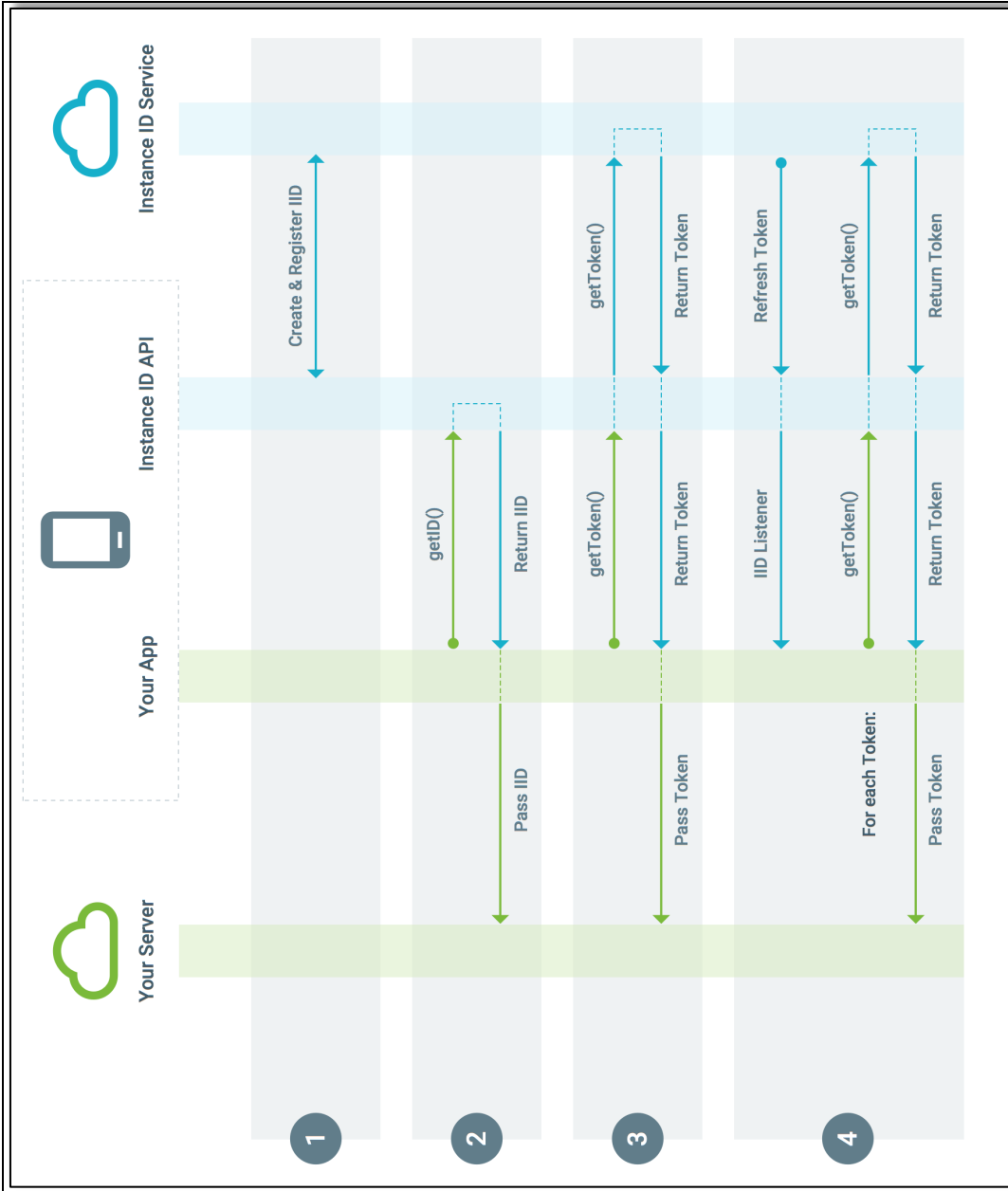
- **Notification messages delivered when your app is in the background.** In this case, the notification is delivered to the device's system tray. A user tap on a notification opens the app launcher by default.
- **Messages with both notification and data payload, when received in the background.** In this case, the notification is delivered to the device's system tray, and the data payload is delivered in the extras of the intent of your launcher Activity.

<https://firebase.google.com/docs/cloud-messaging/android/receive;>

What is Instance ID?

[Send feedback](#)

Instance ID provides a unique ID per instance of your apps. You can implement Instance ID for [Android](#) and [iOS](#) apps as well as [Chrome apps/extensions](#).



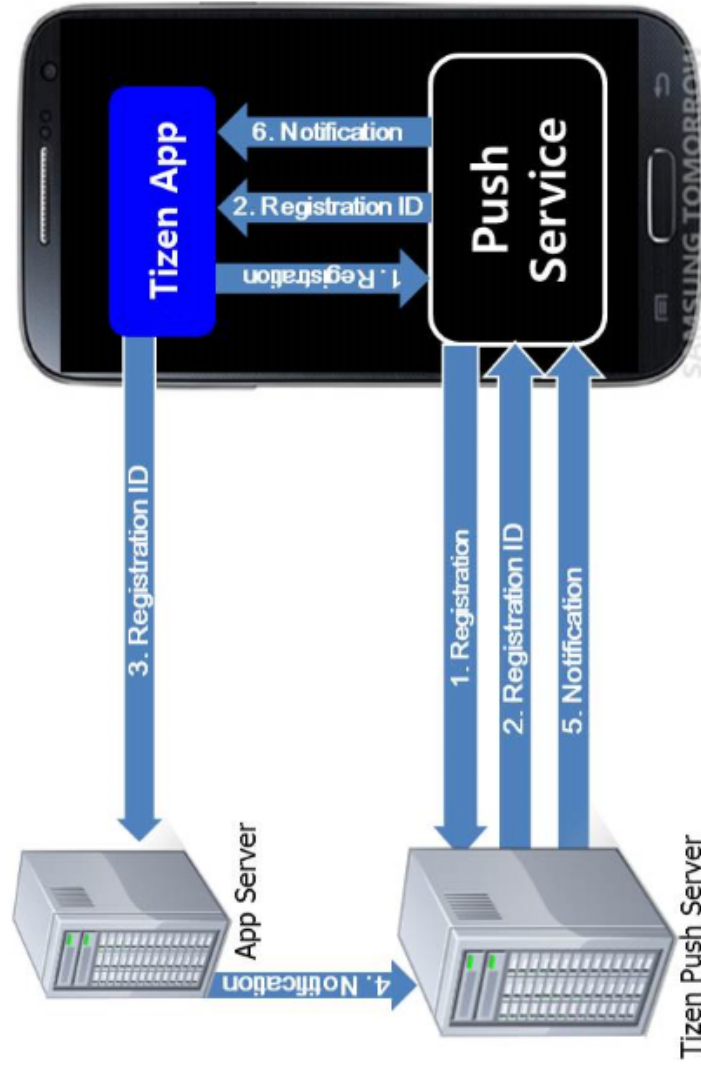
<https://developers.google.com/instance-id>.

For further example, Samsung's Tizen based devices map the identifier in the message to a corresponding software process and forward the data to the application via a secure service. See e.g.,

Architecture

The architecture of the Tizen Push service is described in detail in the [mobile native Push guide](#).

Figure: Service architecture



To receive push notifications for your application:

1. Request permission to access the Tizen push servers for using the push service API.
2. Wait for a confirmation email for the request.
3. Register the installed application on the device.
4. Connect to the push service for receiving push notifications.
5. Receive notifications from the push service.

Push Notification

You can receive notifications from a push server. The push service is a client daemon that maintains a permanent connection between the device and the push server. Push enables you to push events from an application server to your application on a Tizen device. Connection with the push service is used to deliver push notifications to the application, and process the registration and deregistration requests.

The Push API is optional for Tizen Mobile, Wearable, and TV profiles, which means that it may not be supported on all mobile, wearable, and TV devices. The Push API is supported on all Tizen emulators.

Push notification helps your application server send data to your application on a device over a network, even if the application is not running. Using the push service can reduce battery consumption and data transfer.

If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a **LAUNCH** option, the push service forcibly launches the application and hands over the message to the application.

The main features of the Push API include:

- Registering to the push service

You can [register to the push service](#) and get the registration identifier.

Registering to the Push Service

To receive push notifications, you must learn how to register your application to the push service:

- Up to Tizen 2.4:

1. Define event handlers for the registration results:

```
/* Define the data to be used when this process
   is launched by the notification service
*/
var service = new tizen.ApplicationControl('http://tizen.org/appcontrol/operation/push_test');

/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}
```

2. Register the application for the service with the `register()` method. This operation has to be done only once.

```
/* Request application registration */
tizen.push.registerService(service, registerSuccessCallback, errorCallback);
```

- Since Tizen 3.0:

Before registering, you must connect to the push service:

1. Define event handlers:

```
/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}

/* Define the state change callback */
function stateChangeCallback(state) {
    console.log('The state is changed to: ' + state);
}

if (state == 'UNREGISTERED') {
    /* Request application registration */
    tizen.push.register(registerSuccessCallback, errorCallback);
}

/* Define the notification callback */
function notificationCallback(notification) {
    console.log('A notification arrives.');
```

2. Connect to the push service with the `connect()` method. The `register()` method is called in the `stateChangeCallback()` callback. This operation has to be done only once.

```
/* Connect to push service */
tizen.push.connect(stateChangeCallback, notificationCallback, errorCallback);
```

If the registration is successful, the `registerSuccessCallback()` callback is called, and the registration ID is passed as a parameter. Any time after a successful registration, you can get the registration ID using the `getRegistrationId()` method:

```
var registrationId = tizen.push.getRegistrationId();
if (registrationId != null) {
    console.log('The registration id: ' + registrationId);
}
```

Receiving Push Notifications

You can connect to the push service and start receiving push notifications with the `connectService()` method up to Tizen 2.4, or with the `connect()` method since Tizen 3.0. Up to Tizen 2.4, you must pass the `PushNotificationCallback` listener (in [mobile](#) and [wearable](#) applications) as a parameter in the method to receive push notifications. Since Tizen 3.0, you must pass the

`PushRegistrationStateChangeCallback` (in [mobile](#), [wearable](#), and [TV](#) applications) and `PushNotificationCallback` callbacks (in [mobile](#), [wearable](#), and [TV](#) applications) as parameters in the method. The first callback is called when the registration change state changes. This callback is called at least once, just after the connection is established. The second callback is called when notification messages arrive. You can also pass the `ErrorCallback` as a parameter to be called if the connection request fails.

When a notification arrives at the device, its delivery mechanism depends on whether the application is running:

- When the application is running

When a notification arrives to the application while it is running (precisely, the application is connected to the service), the push notification callback is called. In this callback, you can read and process the received notification as described in this use case.

- When the application is not running

If the notification arrives when the application is not running, there are 3 ways to handle the notification:

- Forcibly launch the application and deliver the notification to it.

This happens when the action is set to `LAUNCH` in the message field when sending the notification from the application server. When the notification action arrives at the device, the push service forcibly launches the application and delivers the notification as a bundle.

For more information, see the [Retrieving Messages When Launched by the Push Service](#) use case.

- Store the notification at the push service database and request it later when the application is launched.

This happens when the action is set to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application.

For more information, see the [Retrieving Missed Push Messages](#) use case.

The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification.

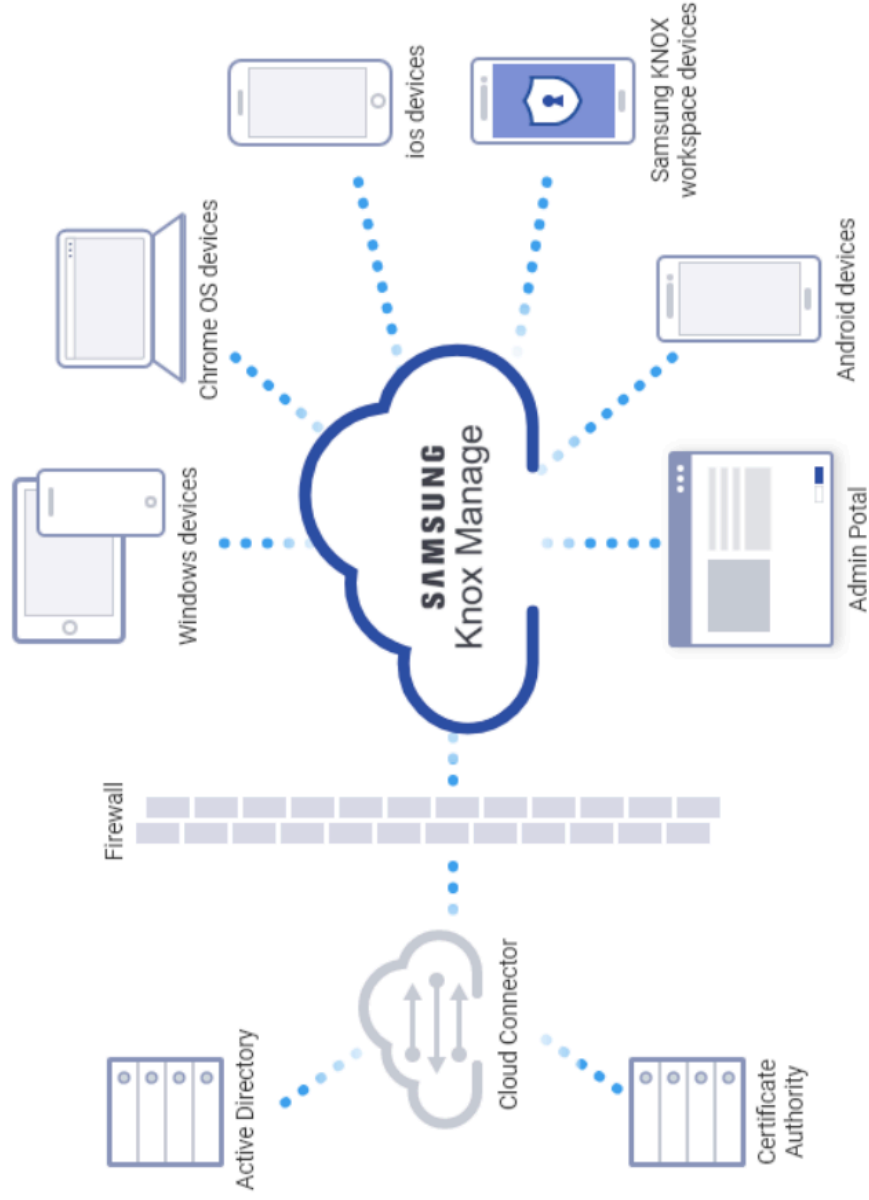
- Discard the notification.

This happens when the action is set to `DISCARD` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service discards the notification unless the application is running.

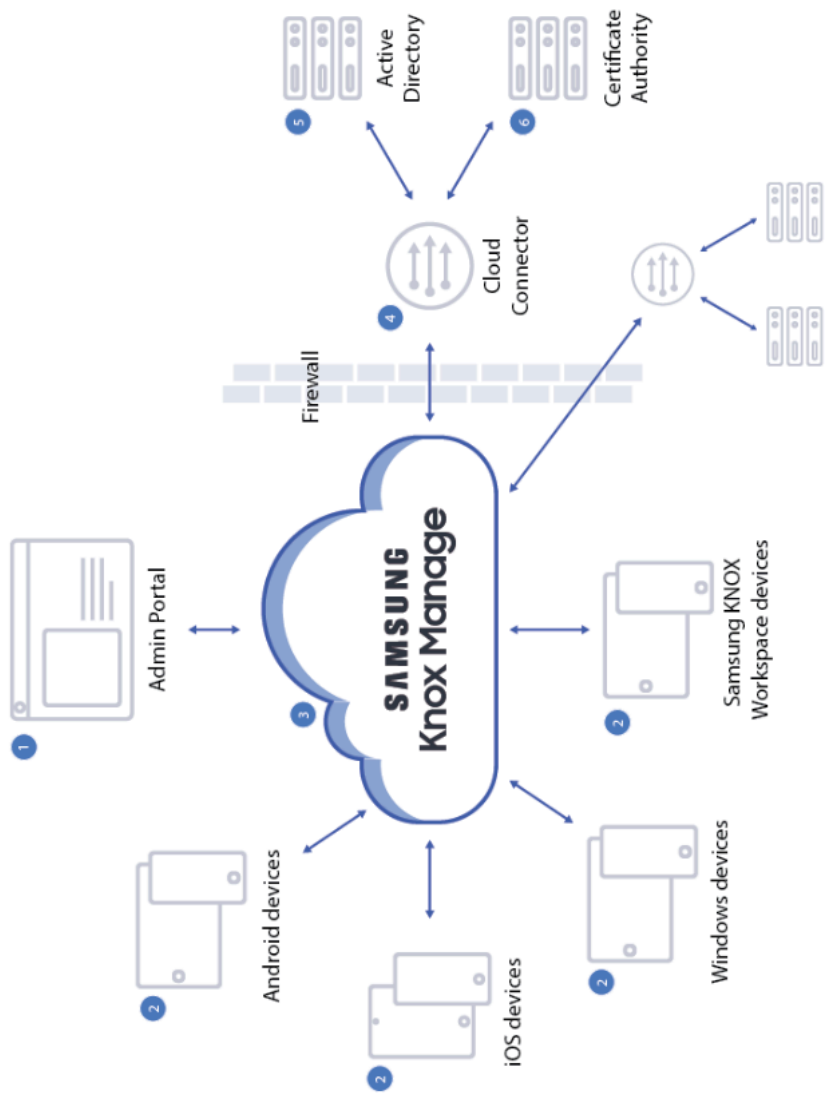
<https://docs.tizen.org/application/web/guides/messaging/push/>.

As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the

Samsung Knox servers managing those phones and devices, comprise mapping the identifier in the message to a corresponding software process and forward the data to the application via a secure service.



<https://docs.samsungknox.com/admin/knox-manage/welcome.htm>



<https://docs.samsungknox.com/admin/knox-manage/km-features.htm>

	Components of Knox Manage 1. Knox Manage console — A web console that allows IT admins to configure, monitor, and manage devices, deploy updates, as well as manage certificates and licenses. 2. Knox Manage MDM Client — An app that is installed on devices to automate installation and enrollment to Knox Manage. 3. Knox Manage Cloud Connector — A service that creates a secure channel for data transfer between your enterprise system and the Knox Manage cloud server. 4. Certificate Authority (CA) — An authority that generates certificates to authenticate devices and users with services such as Wi-Fi, VPN, Exchange, APN, and so on. 5. Active Directory — A Lightweight Directory Access Protocol (LDAP) service that provides access to a customer's directory-based user information. https://docs.samsungknox.com/admin/knox-manage/welcome.htm
--	---

Claim 2

2. The network system of claim 1, the network message server further to collect and buffer multiple requests to transmit application data to a particular one of the devices.	The accused instrumentalities comprise the “network system of claim 1, the network message server further to collect and buffer multiple requests to transmit application data to a particular one of the devices.” <i>See supra</i> claim 1. The accused Tizen and Android devices communicate with Samsung’s service control server link elements both synchronously and asynchronously, including for instance if the end user device is powered off or otherwise not capable of receiving message content from the service control server link elements or not in need of receiving high priority messages. In such case, message content may be delivered at some subsequent time. <i>See, e.g.,</i> https://docs.tizen.org/application/web/guides/messaging/push/ (“While the application is not running, messages cannot be delivered. You can retrieve such missed push messages later.”). Such messages are stored in a buffer for later delivery to user devices.
--	--

Retrieving Missed Push Messages

While the application is not running, messages cannot be delivered. To retrieve such missed push messages:

- Up to Tizen 2.4:

1. Call the `connectService()` method to connect to the Tizen push server and receive new push notifications:

```
/* Method to be called when the notification message arrives */  
function notificationCallback(message) {  
    console.log('New push message: ' + message.alertMessage + ', date: ' + message.date + ',  
    data: ' + message.appData);  
}  
  
tizen.push.connectService(notificationCallback);
```

2. Retrieve any unread messages:

```
tizen.push.getUnreadNotifications();
```

- Since Tizen 3.0:

Call the `connect()` method to connect to the Tizen push server, and retrieve any unread messages with the `getUnreadNotifications()` method in the state change callback when the application is registered:

```
/* Define the error callback */
function errorCallback(response) {
    console.log('The following error occurred: ' + response.name);
}

/* Define the registration success callback */
function registerSuccessCallback(id) {
    console.log('Registration succeeded with id: ' + id);
}

/* Define the state change callback */
function stateChangeCallback(state) {
    console.log('The state is changed to: ' + state);

    if (state === 'UNREGISTERED') {
        /* Request application registration */
        tizen.push.register(registerSuccessCallback, errorCallback);
    } else if (state === 'REGISTERED') {
        /* Gets unread push notifications */
        tizen.push.getUnreadNotifications();
    }
}

/* Define the notification callback */
function notificationCallback(notification) {
    console.log('A notification arrives.');
```

```
    }

    tizen.push.connect(stateChangeCallback, notificationCallback, errorCallback);
```

<https://docs.tizen.org/application/web/guides/messaging/push/#retrieving-missed-push-messages>

- **Normal priority.** Normal priority messages are delivered immediately when the app is in the foreground. For backgrounded apps, delivery may be delayed. For less time-sensitive messages, such as notifications of new email, keeping your UI in sync, or syncing app data in the background, choose normal delivery priority.

	Setting the lifespan of a message FCM usually delivers messages immediately after they are sent. However, this might not always be possible. For example, if the platform is Android, the device could be turned off, offline, or otherwise unavailable. Or FCM might intentionally delay messages to prevent an app from consuming excessive resources and negatively affecting battery life. When this happens, FCM stores the message and delivers it as soon as it's feasible. While this is fine in most cases, there are some apps for which a late message might as well never be delivered. For example, if the message is an incoming call or video chat notification, it is meaningful only for a short period of time before the call is terminated. Or if the message is an invitation to an event, it is useless if received after the event has ended. On Android and Web/JavaScript, you can specify the maximum lifespan of a message. The value must be a duration from 0 to 2,419,200 seconds (28 days), and it corresponds to the maximum period of time for which FCM stores and attempts to deliver the message. Requests that don't contain this field default to the maximum period of four weeks. https://firebase.google.com/docs/cloud-messaging/concept-options
--	---

Claim 4

85

4. The network system of claim 1, the network message server further to encrypt the secure Internet data messages, the device messaging agents further to decrypt each received message to obtain the corresponding application identifier and application data.	The accused instrumentalities comprise the “network system of claim 1, the network message server further to encrypt the secure Internet data messages, the device messaging agents further to decrypt each received message to obtain the corresponding application identifier and application data.” <i>See supra</i> claim 1. For example, the Tizen Push Server receives encrypted information from app servers and transmits such encrypted information on to device messaging agents residing on user devices, which in turn decrypt received messages to obtain the identity of the corresponding application and message content.
---	--

Managing Security

When you send a notification with sensitive information, be aware of the chance that the notification gets hijacked by someone else. It is your responsibility to keep such sensitive information safe from malicious access. The following rules are strongly recommended:

- Keep the push application ID confidential.
- If the application ID is exposed, hackers can try to hijack notifications using a fake application with the exposed ID.
- Do not store the registration ID on the device.

The registration ID can be considered as the destination address for notifications. Without the ID, hackers cannot send fake notifications to your application.

- Encrypt sensitive information.

When you send sensitive information, such as personal information and financial transactions, encrypt it and load it to the notification as a payload instead of the message field. When the notification arrives at the device, the application decrypts the payload and retrieves the sensitive information.

- Do not hardcode the AppSecret in the source code.

The AppSecret is a key to accessing the push server for sending notifications. If notifications are sent from your application server, the application does not need to know the AppSecret at all. Keep the AppSecret in the server and do not load any related information in the application. If you want device-to-device notification delivery without your application server, the application needs the AppSecret to send a notification from a device. In this case, it is your responsibility to keep the AppSecret safe.

Sending Push Notifications

Once the application successfully sends its registration ID to the application server, you are ready to send push notifications from the application server to the application on that particular device. This use case describes how to send a simple push notification to the device. For advanced features, see the Push Server guide for server developers.

The following example shows a sample push notification:

- URI: See the Push RQM (Request Manager) server URLs table.

- Method: HTTP POST

- Header:

```
appId: 1234567890987654
appSecret: dYo/o/m11gmHwJ57+5f+zLNV0c=
```

- Body:

```
{
  "regID": "0501a53f4affdcbb98197f188345ff36c04b-5001",
  "requestID": "01231-22EAX-223442",
  "message": "badgeOption=INCREASE&badgeNumber=1&action=ALERT&alertMessage=Hi",
  "appdata": "{id:asd&passwd:1234}", /* Optional, if the message field is not empty */
}
```

	https://docs.tizen.org/application/native/guides/messaging/push/#managing-security As another example, the Firebase framework encrypts messages in transit. See: https://firebase.google.com/docs/cloud-messaging/concept-options (“Encryption for data messages The Android Transport Layer (see FCM architecture) uses point-to-point encryption.”). Likewise, messages sent with Knox Manage instrumentalities are also encrypted. See https://docs.samsungknox.com/admin/fundamentals/whitepaper/core-platform-security/knox-vault/; https://docs.samsungknox.com/admin/fundamentals/whitepaper/network-security/virtual-private-networks/.
--	--

Claim 5

5. The network system of claim 4, wherein the secure Internet data messages are transported to the device messaging agent on each device using one or more of encryption on a transport services stack, IP (Internet Protocol) layer encryption, and tunneling.	The accused instrumentalities comprise the “network system of claim 4, wherein the secure Internet data messages are transported to the device messaging agent on each device using one or more of encryption on a transport services stack, IP (Internet Protocol) layer encryption, and tunneling.” <i>See supra</i> claim 1, 4. For example, the Tizen Push Server receives encrypted information from app servers and transmits such encrypted information on to device messaging agents residing on user devices, which in turn decrypt received messages to obtain the identity of the corresponding application and message content.
--	--

Managing Security

When you send a notification with sensitive information, be aware of the chance that the notification gets hijacked by someone else. It is your responsibility to keep such sensitive information safe from malicious access. The following rules are strongly recommended:

- Keep the push application ID confidential.

If the application ID is exposed, hackers can try to hijack notifications using a fake application with the exposed ID.

- Do not store the registration ID on the device.

The registration ID can be considered as the destination address for notifications. Without the ID, hackers cannot send fake notifications to your application.

- Encrypt sensitive information.

When you send sensitive information, such as personal information and financial transactions, encrypt it and load it to the notification as a payload instead of the message field. When the notification arrives at the device, the application decrypts the payload and retrieves the sensitive information.

- Do not hardcode the AppSecret in the source code.

The AppSecret is a key to accessing the push server for sending notifications. If notifications are sent from your application server, the application does not need to know the AppSecret at all. Keep the AppSecret in the server and do not load any related information in the application. If you want device-to-device notification delivery without your application server, the application needs the AppSecret to send a notification from a device. In this case, it is your responsibility to keep the AppSecret safe.

Sending Push Notifications

Once the application successfully sends its registration ID to the application server, you are ready to send push notifications from the application server to the application on that particular device. This use case describes how to send a simple push notification to the device. For advanced features, see the Push Server guide for server developers.

The following example shows a sample push notification:

- URI: See the Push RQM (Request Manager) server URLs table.

- Method: HTTP POST

- Header:

```
appId: 1234567890987654
appSecret: dYo/o/m11gmWwJ57+5f+zLlNV0c=
```

- Body:

```
{
  "regID": "0501a53f4affdcbb98197f188345ff36c04b-5001",
  "requestID": "01231-22EAX-223442",
  "message": "badgeOption=INCREASE&badgeNumber=1&action=ALERT&alertMessage=Hi",
  "appdata": "{id:asd&passwd:1234}", /* Optional, if the message field is not empty */
}
```

<https://docs.tizen.org/application/native/guides/messaging/push/#managing-security>

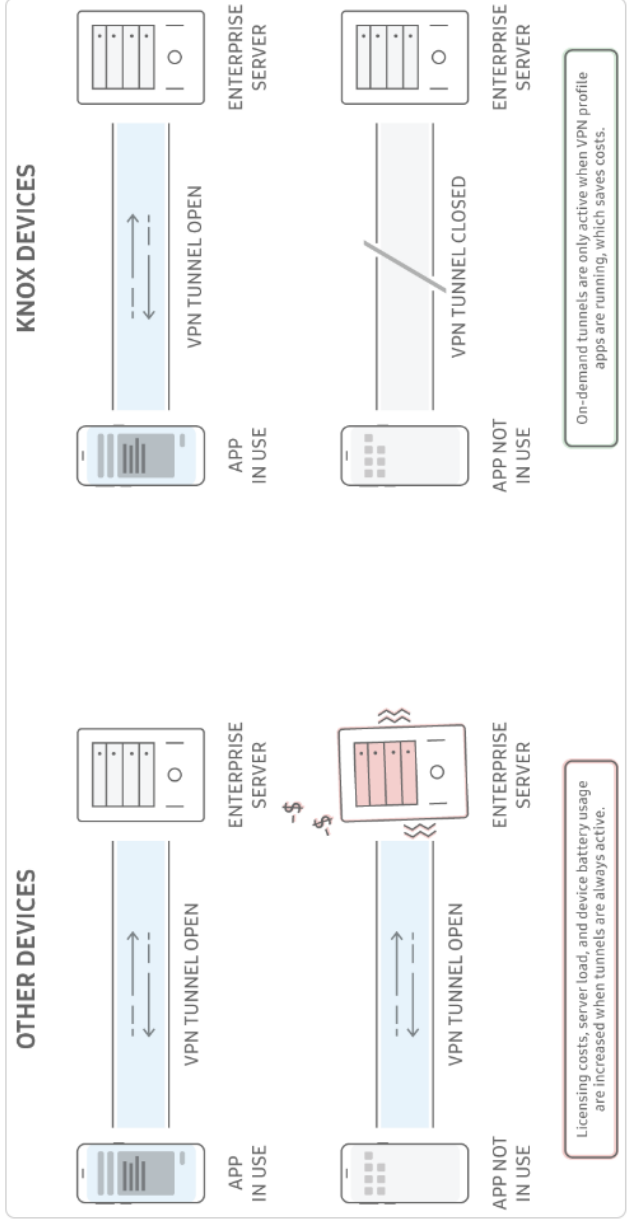
As another example, the Firebase framework encrypts messages in transit. See: <https://firebase.google.com/docs/cloud-messaging/concept-options> (“Encryption for data messages The Android Transport Layer (see FCM architecture) uses point-to-point encryption.”).

Likewise, messages sent with Knox Manage instrumentalities are also encrypted. See <https://docs.samsungknox.com/admin/fundamentals/whitepaper/core-platform-security/knox-vault/>; <https://docs.samsungknox.com/admin/fundamentals/whitepaper/network-security/virtual-private-networks/>:

Unique advantages of Knox VPN framework

The Knox Platform VPN framework supports all common VPN types, protocols, and configuration options. When deploying VPN solutions, enterprise IT admins must ensure VPN deployments work smoothly, don’t waste server resources, limit the VPN solution licensing costs, and enforce strict security policies that prevent data leakage.

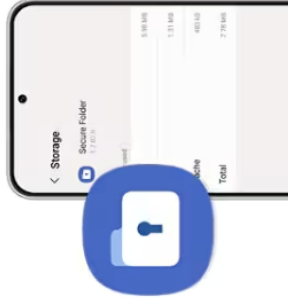
The following is an example showing how Knox on-demand VPNs save cost:



Claim 6

6. The network system of claim 1, wherein the device messaging agent executes in a secure execution environment on at least one of the devices, and at least one of the applications executes outside of the secure execution environment on that device.	The accused instrumentalities comprise the “network system of claim 1, wherein the device messaging agent executes in a secure execution environment on at least one of the devices, and at least one of the applications executes outside of the secure execution environment on that device.” <i>See supra</i> claim 1. As just one example, with Knox-managed devices, the Knox agent software residing on the user device executes from a specially protected partition of local memory. This specially protected partition is sometimes referred to as the Knox Secure Folder. Applications stored in the Secure Folder partition are managed separately with respect to push notifications. Other applications on the user device may be run from outside of the Knox Secure Folder partition.
--	--

What is the Secure Folder and how do I use it?



The Secure Folder is the perfect place to store all the photos, videos, files, apps and data that you want to keep private. The folder is protected by the defence-grade Samsung Knox security platform, which encrypts all data stored there, making sure that your information is kept safe from any malicious attacks. You can also add a passcode or biometric lock to keep your data safe from any prying eyes.

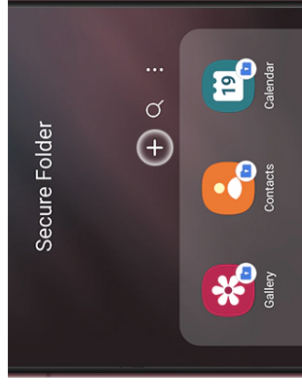
<https://www.samsung.com/uk/support/mobile-devices/what-is-the-secure-folder-and-how-do-i-use-it/>

Add apps or files in Secure Folder

You want to download your bank's app, but you're nervous someone will steal your information. If you add the app to Secure Folder, no one will even know it is on your phone except you.

After Secure Folder is set up, tap **Add apps** (the plus sign). Tap a desired app to put in the folder, and then tap **Add**. You'll see a copy of the app in your Secure Folder.

To add files, tap **More options** (the three vertical dots) and then tap **Add files**. You can now choose the type of file, or use My Files to find any file on your device.

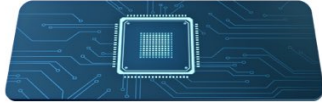


	https://www.samsung.com/us/support/answer/ANS00081123/
--	---

Claim 9

9. The network system of claim 1, further comprising, on at least one of the devices, the secure interprocess communication service, and wherein the secure interprocess communication service and the secure Internet data connection from that device to the network message server are separately secured.	The accused instrumentalities comprise the “network system of claim 1, further comprising, on at least one of the devices, the secure interprocess communication service, and wherein the secure interprocess communication service and the secure Internet data connection from that device to the network message server are separately secured.” <i>See supra</i> claim 1. For example: Samsung Galaxy phones and tablets use encryption protocols to secure communications between the device and the network system. <i>See e.g.</i>,
--	--

SECURE



Chip-up manufacturing

Samsung designs, creates and validates every component we put in our mobile devices in highly secure factories, ensuring a consistent supply chain and end product you can trust.

Data isolation

Data isolation is built into every component of our mobile devices. It's a double-lock for sensitive data, meaning the only person who ever has access is you.

Data encryption

Knox platform encryption protects your data from vulnerabilities, even in the event of theft or loss. Even while your device is powered off, your data remains encrypted and won't be decrypted unless you choose to.

Run-time protection

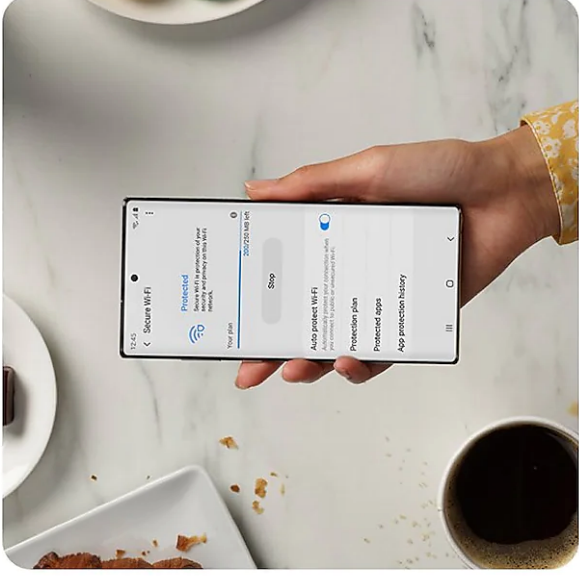
Run-time protection protects your device against data attacks or malware. Any unauthorized or unintended attempts to access or modify your phone's core are blocked in real time.

Secure your connections

Secure Wi-Fi encrypts outgoing internet traffic and disables tracking apps and websites when using public wireless connections, allowing you to browse the internet safely without fear of security breaches.

**Secure Wi-Fi feature availability may vary depending on country, carrier, or network environment and may not be supported on all Samsung mobile devices. Fees may apply depending on Secure Wi-Fi usage.*

HOW TO SECURE YOUR WI-FI [❏](#)



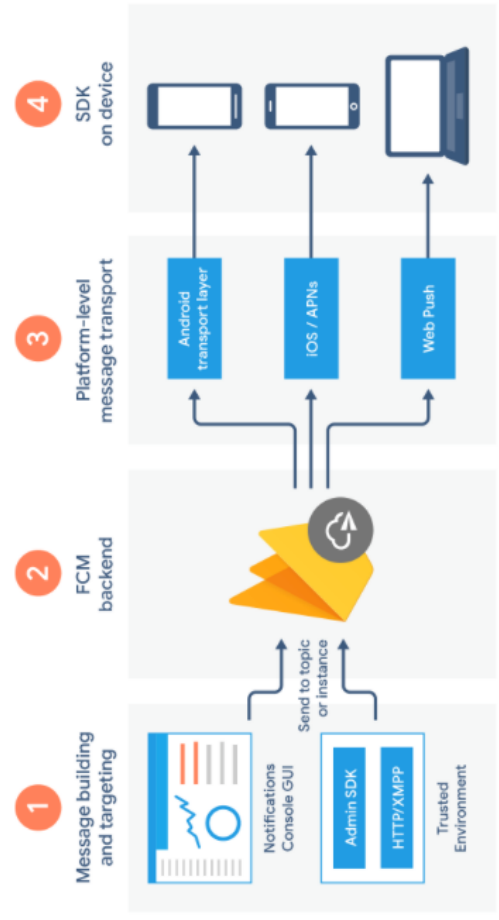
<https://www.samsung.com/us/security/>;

FCM Architectural Overview

[Send feedback](#)

FCM relies on the following set of components that build, transport, and receive messages:

1. Tooling to compose or build message requests. The Notifications composer provides a GUI-based option for creating notification requests. For full automation and support for all [message types](#), you must build message requests in a trusted [server environment](#) that supports the Firebase Admin SDK or the FCM server protocols. This environment could be Cloud Functions for Firebase, App Engine, or your own app server.



2. The FCM backend, which (among other functions) accepts message requests, performs fanout of messages via topics, and generates message metadata such as the message ID.

3. A platform-level transport layer, which routes the message to the targeted device, handles message delivery, and applies platform-specific configuration where appropriate. This transport layer includes:

- Android transport layer (ATL) for Android devices with Google Play services
- Apple Push Notification service (APNs) for Apple devices
- Web push protocol for web apps

★ **Note:** Platform-level transport layers are outside the core FCM product. FCM messages routed to a platform-level transport layer may be subject to terms specific to that platform rather than FCM's terms of service. Android message routing via ATL falls under the [Google APIs terms of service](#).

4. The FCM SDK on the user's device, where the notification is displayed or the message is handled according to the app's foreground/background state and any relevant application logic.

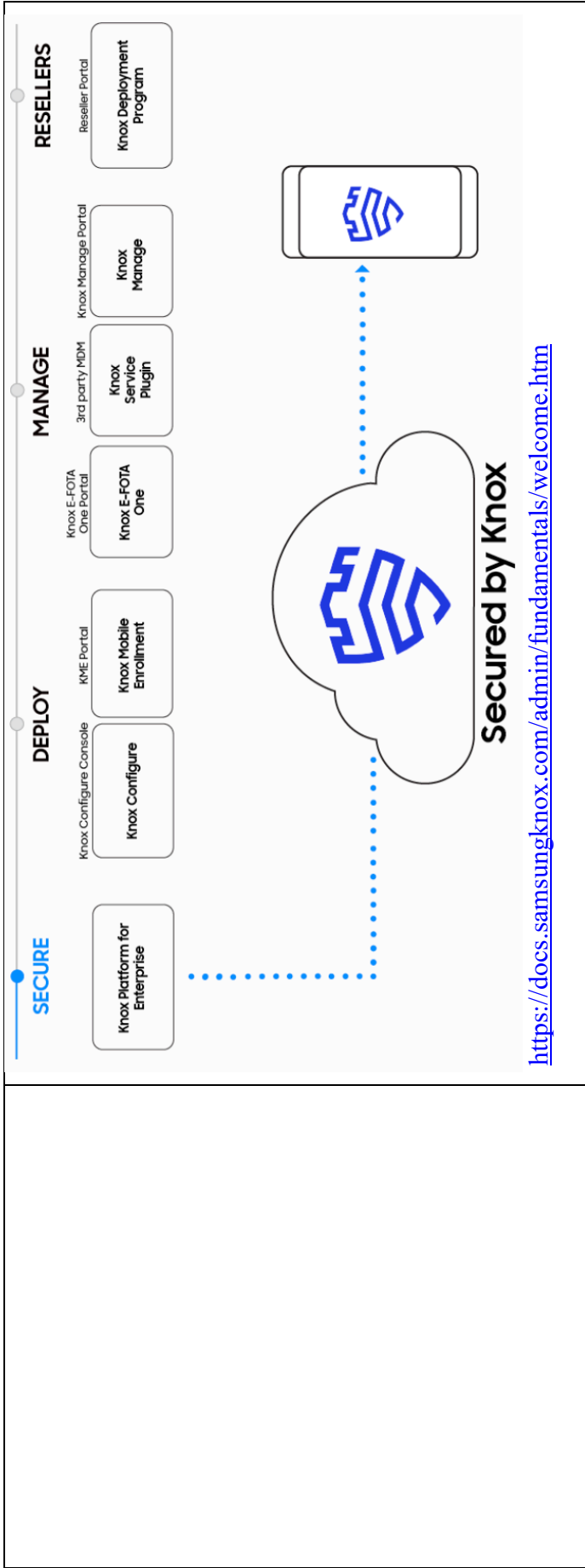
<https://firebase.google.com/docs/cloud-messaging/fcm-architecture>;

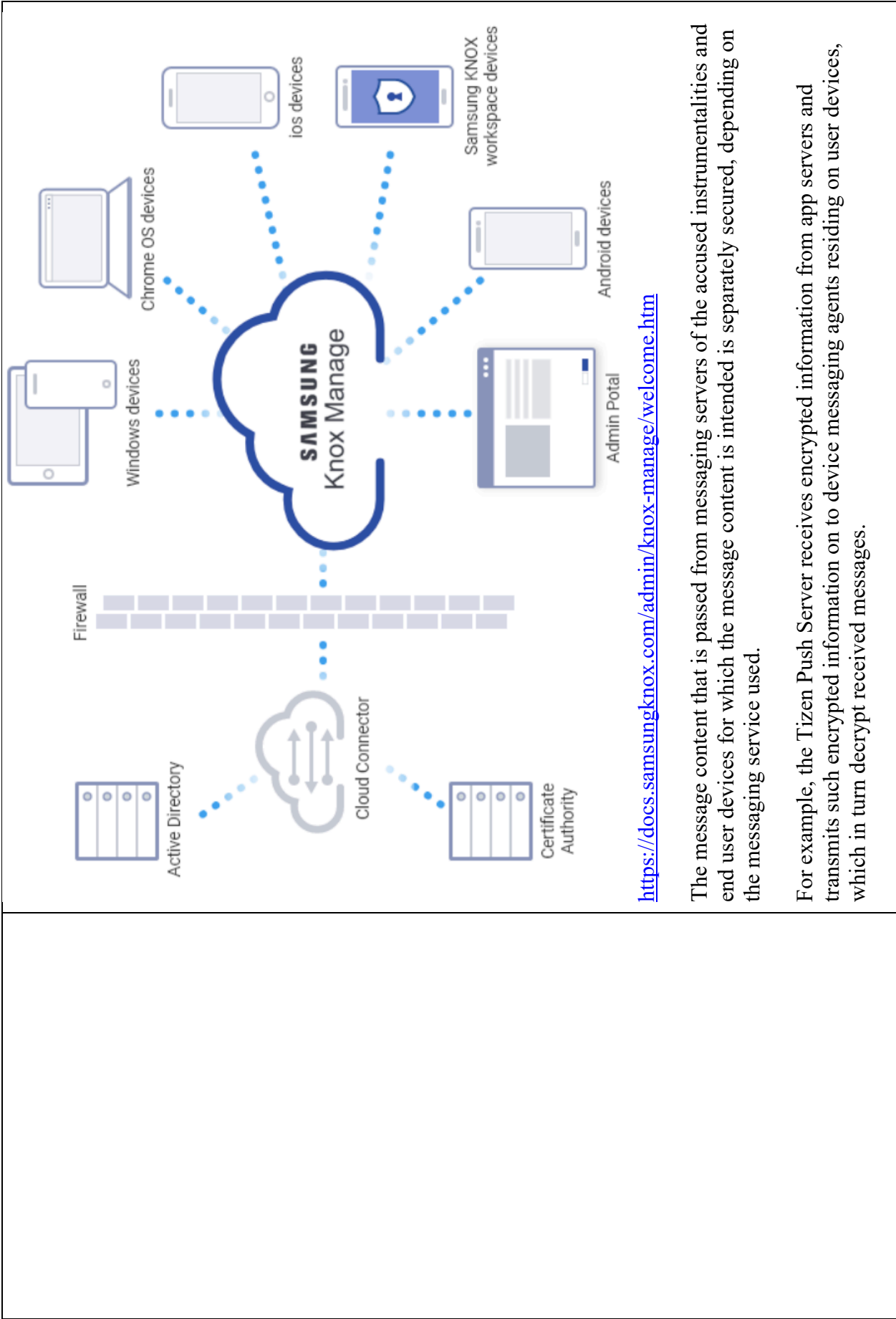
Encryption for data messages

The Android Transport Layer (see [FCM architecture](#)) uses point-to-point encryption. Depending on your needs, you may decide to add end-to-end encryption to data messages. FCM does not provide an end-to-end solution. However, there are external solutions available such as [Capillary](#) or [DTLS](#).

https://firebase.google.com/docs/cloud-messaging/concept-options#encryption_for_data_messages.

As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform, include modems for communicating with a network system over a service control link secured by encryption protocols programmed into the Samsung Knox MDM platform and e.g., Samsung Knox end-user device software for secure communications with a network system:





Managing Security

When you send a notification with sensitive information, be aware of the chance that the notification gets hijacked by someone else. It is your responsibility to keep such sensitive information safe from malicious access. The following rules are strongly recommended:

- Keep the push application ID confidential.

If the application ID is exposed, hackers can try to hijack notifications using a fake application with the exposed ID.

- Do not store the registration ID on the device.

The registration ID can be considered as the destination address for notifications. Without the ID, hackers cannot send fake notifications to your application.

- Encrypt sensitive information.

When you send sensitive information, such as personal information and financial transactions, encrypt it and load it to the notification as a payload instead of the message field. When the notification arrives at the device, the application decrypts the payload and retrieves the sensitive information.

- Do not hardcode the AppSecret in the source code.

The AppSecret is a key to accessing the push server for sending notifications. If notifications are sent from your application server, the application does not need to know the AppSecret at all. Keep the AppSecret in the server and do not load any related information in the application. If you want device-to-device notification delivery without your application server, the application needs the AppSecret to send a notification from a device. In this case, it is your responsibility to keep the AppSecret safe.

Sending Push Notifications

Once the application successfully sends its registration ID to the application server, you are ready to send push notifications from the application server to the application on that particular device. This use case describes how to send a simple push notification to the device. For advanced features, see the Push Server guide for server developers.

The following example shows a sample push notification:

- URI: See the Push RQM (Request Manager) server URLs table.

- Method: HTTP POST

- Header:

```
appId: 1234567890987654
appSecret: dYo/o/m11gmWwJS7+5f+zLNV0c=
```

- Body:

```
{
  "regID": "0501a53f4affdcbb98197f188345ff36c04b-5001",
  "requestID": "01231-22EAX-223442",
  "message": "badgeOption=INCREASE&badgeNumber=1&action=ALERT&alertMessage=Hi",
  "appData": "{id:asd&passwd:1234}", /* Optional, if the message field is not empty */
}
```

	https://docs.tizen.org/application/native/guides/messaging/push/#managing-security As another example, the Firebase framework encrypts messages in transit separately from the channel encryption. See: https://firebase.google.com/docs/cloud-messaging/concept-options (“Encryption for data messages The Android Transport Layer (see FCM architecture) uses point-to-point encryption.”). Likewise, messages sent with Knox Manage instrumentalities are also encrypted. See https://docs.samsungknox.com/admin/fundamentals/whitepaper/core-platform-security/knox-vault/; https://docs.samsungknox.com/admin/fundamentals/whitepaper/network-security/virtual-private-networks/.
--	---

Claim 10

10. The network system of claim 1, wherein at least one of the secure Internet data messages comprises multiple identifier/data pairs.	The accused instrumentalities comprise the “network system of claim 1, wherein at least one of the secure Internet data messages comprises multiple identifier/data pairs.” <i>See supra</i> claim 1. See, for example, https://firebase.google.com/docs/cloud-messaging/concept-options (“Data messages, by contrast, contain only your user-defined custom key-value pairs. Notification messages can contain an optional data payload. Maximum payload for both message types is 4000 bytes, except when sending messages from the Firebase console, which enforces a 1000 character limit.”); https://docs.tizen.org/application/native/guides/messaging/push/ (“The message field contains not only the message to show in the quick panel on the device, but also the behaviors that the device must take when receiving the notification. The message field is a string that consists of key-value pairs. The available pair options are given in the following table.”).
---	---

Claim 11

11. The network system of claim 1, wherein the secure interprocess communication service forwards the application data to at least one of the software processes in an encrypted format.	The accused instrumentalities comprise the “network system of claim 1, wherein the secure interprocess communication service forwards the application data to at least one of the software processes in an encrypted format.” <i>See supra</i> claim 1, 4, and 9. As explained above, the accused instrumentalities cause encrypted messages to be delivered to applications for which the messages are intended.
--	---

Claim 12

12. The network system of claim 1, the device messaging agent on at least one of the devices further to initiate the secure connection to the network message server.	The accused instrumentalities comprise the “network system of claim 1, the device messaging agent on at least one of the devices further to initiate the secure connection to the network message server.” <i>See supra</i> claim 1. As explained above, the device message agents on the devices may initiate secure connections to the push messaging servers and Knox Manage servers to, inter alia, request push messages and other information.
---	--

Claim 13

13. The network system of claim 1, at least one of the devices having a network stack in communication with the device messaging agent, wherein the secure connection between the network message server and that device is terminated within the network stack.	The accused instrumentalities comprise the “network system of claim 1, at least one of the devices having a network stack in communication with the device messaging agent, wherein the secure connection between the network message server and that device is terminated within the network stack.” This occurs in the accused instrumentalities because the user devices communicating with the network message server are communicating with it through a secure SSL-encrypted channel, which terminates in the user device.
--	--

Claim 14

14. The network system of claim 1, wherein at least one of the applications on at least one of the devices and the network application server corresponding to that application authenticate with each other prior to passing application data via the device messaging agent on that device and the network message server.	The accused instrumentalities comprise the “network system of claim 1, wherein at least one of the applications on at least one of the devices and the network application server corresponding to that application authenticate with each other prior to passing application data via the device messaging agent on that device and the network message server.” <i>See supra</i> claim 1. As explained above, applications resident on the user devices authenticate with the push messaging servers in order to enable push messaging capability.
--	--

Claim 16

16. The network system of claim 2, wherein the network message server transmits the collected and buffered requests to the particular device upon the occurrence of a transmission trigger.	The accused instrumentalities comprise the “network system of claim 2, wherein the network message server transmits the collected and buffered requests to the particular device upon the occurrence of a transmission trigger.” <i>See supra</i> claim 2. The transmission triggers may be, for example, the application requesting push messaging, or the device appearing to be powered on to the push messaging server after a period of inactivity. <i>See, e.g.,</i> https://docs.tizen.org/application/web/guides/messaging/push/: • “Store the notification at the push service database and request it later when the application is launched. This happens when the action is set to ALERT or SILENT in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application. For more information, see the Retrieving Missed Push Messages use case. The difference between the ALERT and SILENT actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does
---	--

	not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification. Store the notification at the push service database and request it later when the application is launched. You need to set the action to ALERT or SILENT in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application. You can request for unread notifications from the push service. The request can be performed after connecting to the push server when the application is launched. Request the unread notifications asynchronously. If there is such a notification, it can be received through the <code>_not_i_cb()</code> function after the <code>_on_state_registered()</code> function returns. Once the request for unread notifications is successfully delivered, <code>PUSH_SERVICE_ERROR_NONE</code> is returned.”
--	--

Claim 17

17. The network system of claim 16, wherein the transmission trigger is the expiration of a periodic timer.	The accused instrumentalities comprise the “network system of claim 16, wherein the transmission trigger is the expiration of a periodic timer.” See supra claim 16. The transmission trigger may be a time period of inactivity.
--	---

Claim 18

18. The network system of claim 16, wherein the transmission trigger is the receipt of a transmission from the device messaging agent of the particular device.	The accused instrumentalities comprise the “network system of claim 16, wherein the transmission trigger is the receipt of a transmission from the device messaging agent of the particular device.” As explained above, the network server may receive transmissions (e.g., requests for stored push messages) from a device messaging agent, which would cause push messages to be delivered to the device.
--	---