

U.S. Patent No. 8,406,733 (“733 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
1[pre] An end-user device comprising:	Samsung Galaxy phones and tablets and Samsung Tizen based devices are each “an end-user device.” For example, the Galaxy S22 is a mobile device. <i>See e.g.</i>, https://www.samsung.com/us/smartphones/galaxy-s22/; As another example, Samsung’s Tizen based devices are user devices. <i>See e.g.</i>,

	 https://www.samsung.com/us/televisions-home-theater/tvs/the-frame/55-class-the-frame-qled-4k-smart-tv-2022-qn55ls03bafxza/.
l[a] a modem for enabling communication with a network system over a service control link provided by the network system over a wireless access network, the service control link secured by an encryption protocol and configured to support control-plane communications between the network system and a service control device link agent on the end-user device;	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise "a modem for enabling communication with a network system over a service control link provided by the network system over a wireless access network, the service control link secured by an encryption protocol and configured to support control-plane communications between the network system and a service control device link agent on the end-user device." For example, the Galaxy S22 includes a plurality of wireless modems for communicating with a network system over a wi-fi network and/or a cellular network (a "wireless access network"). <i>See e.g.,</i>

	Network & Connectivity 5G 5G Non-Standalone (NSA), Standalone (SA), Sub6 / mmWave LTE Enhanced 4x4 MIMO, Up to 7CA, LTE Cat.20 Up to 2.0Gbps Download / Up to 200Mbps Upload Wi-Fi Wi-Fi 802.11 a/b/g/n/ac/ax 2.4G+5GHz+6GHz, HE160, MIMO, 1024-QAM Up to 2.4Gbps Download / Up to 2.4Gbps Upload Bluetooth Bluetooth® v5.2, USB type-C, NFC, Location(GPS, Galileo, Glonass, BeiDou) Ultra Wide Band *Requires optimal connection. Actual speed may vary depending on country, carrier and user environment. *The bandwidths supported by the device may vary depending on the region or service provider. *Download and upload speeds reaching up to 2.4Gbps only available with Wi-Fi 6E. Wi-Fi 6E only supported on Galaxy S22 Ultra and S22+. Galaxy S22 has Wi-Fi 6. *Galileo and BeiDou coverage may be limited. BeiDou may not be available for certain countries. https://www.samsung.com/us/smartphones/galaxy-s22/models/. Samsung Galaxy phones and tablets use encryption protocols, e.g., using encryption keys, signatures, secure buses, etc., 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

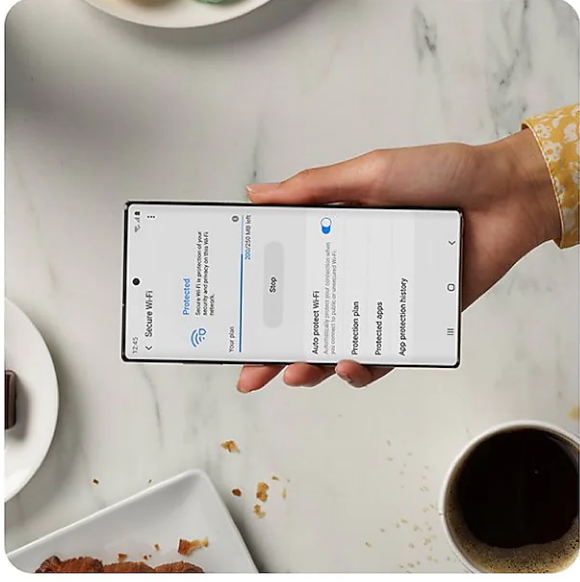
Data isolation is built into every component of our mobile devices. It's a double-lock for sensitive data, meaning the only person who ever has access is you.

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 [\[i\]](#)

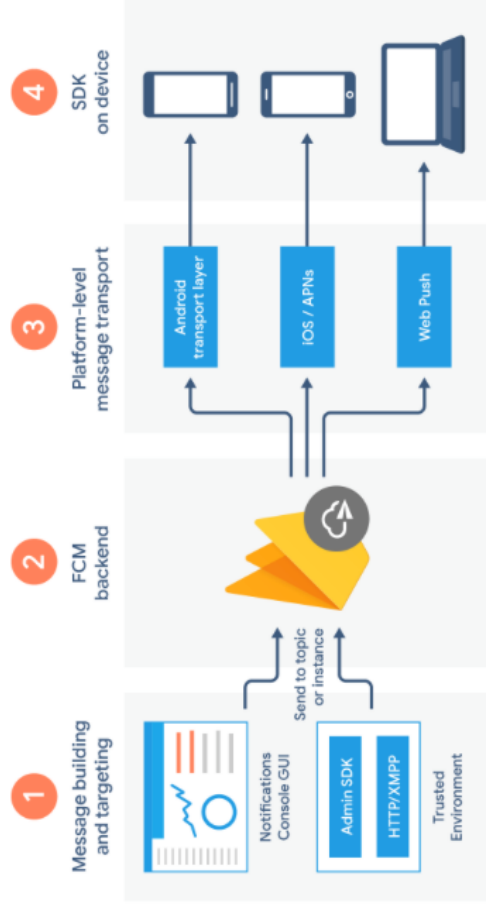
<https://www.samsung.com/us/security/>;

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.

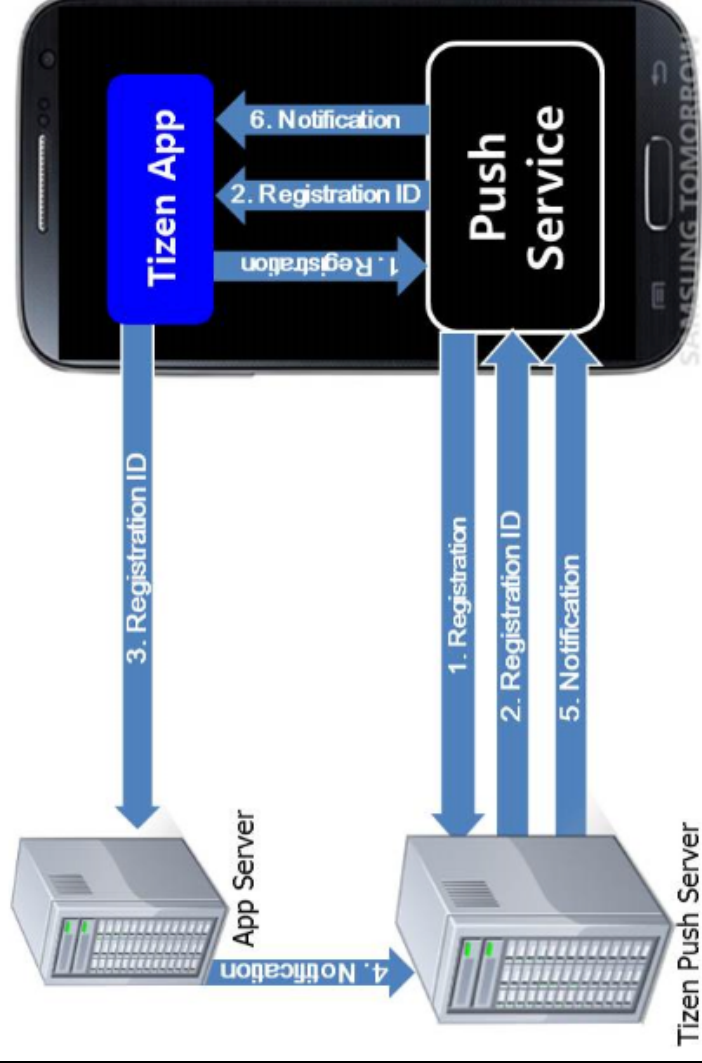
As another example, Samsung's Tizen based devices include wireless modems for communicating with a network system over a wi-fi network (a "wireless access network"). See *e.g.*,

	Wireless Connectivity WI-FI ? Yes (WIFI5) WI-FI Direct Yes Bluetooth ? Yes (BT5.2) https://www.samsung.com/us/televisions-home-theater/tvs/the-frame/55-class-the-frame-qlled-4k-smart-tv-2022-qn55ls03bafxza/#specs; Wireless security protocols The TV only supports the following wireless network security protocols. The TV cannot connect to non-certified wireless access point. – Authentication Modes: WEP, WPA2PSK, WPA2PSK – Encryption Types: WEP, TKIP, AES https://downloadcenter.samsung.com/content/UM/202203/202203092001587/OSNATSCB-3.2.0_EM_Oscar_Pontus_Nike_Kant-SU2e_USA_ENG_220216.0.pdf. Further, Samsung's Tizen based devices use encryption protocols, e.g., using encryption keys, signatures, secure buses, etc., to secure communications between the device and the network system. <i>See e.g.</i>,
--	---

Service Architecture

The following figure illustrates the service architecture of the Tizen push messaging service.

Figure: Service architecture



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.

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

Security

The security features introduce how private information remains private, and how the user knows when they are trying to access privileged information. You can use a repository and cryptographic operations to manage keys, certificates, and sensitive user data. When the user tries to access privileged information, you can display information about the data.

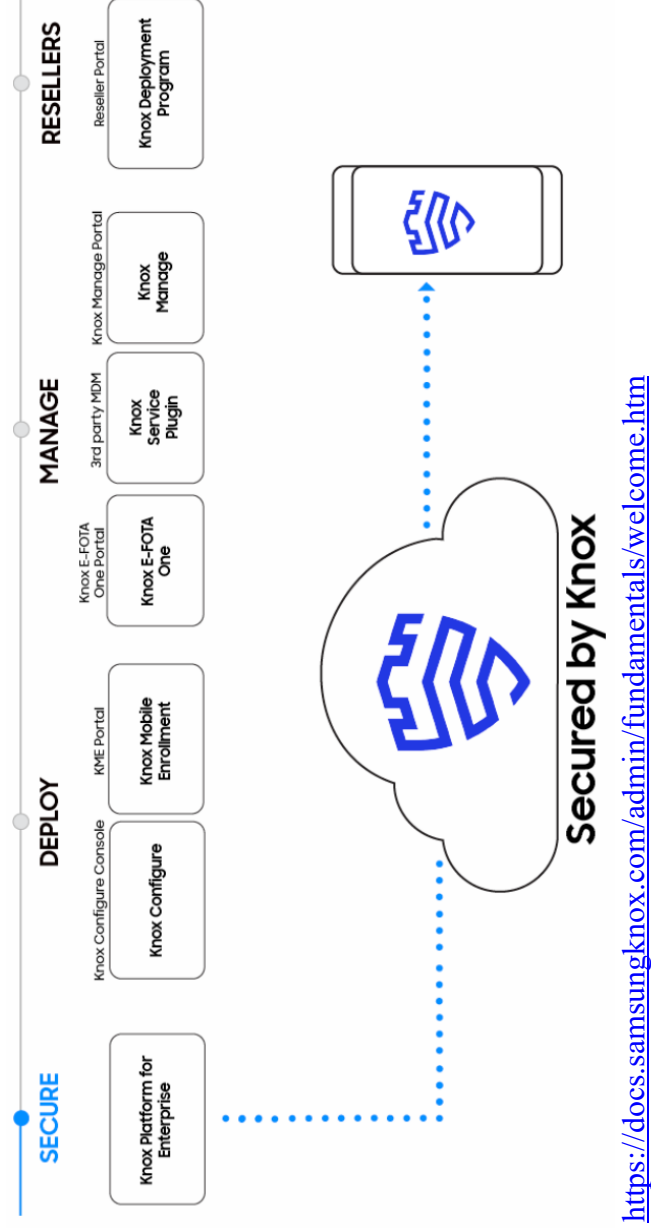
You can use the following security features in your native applications:

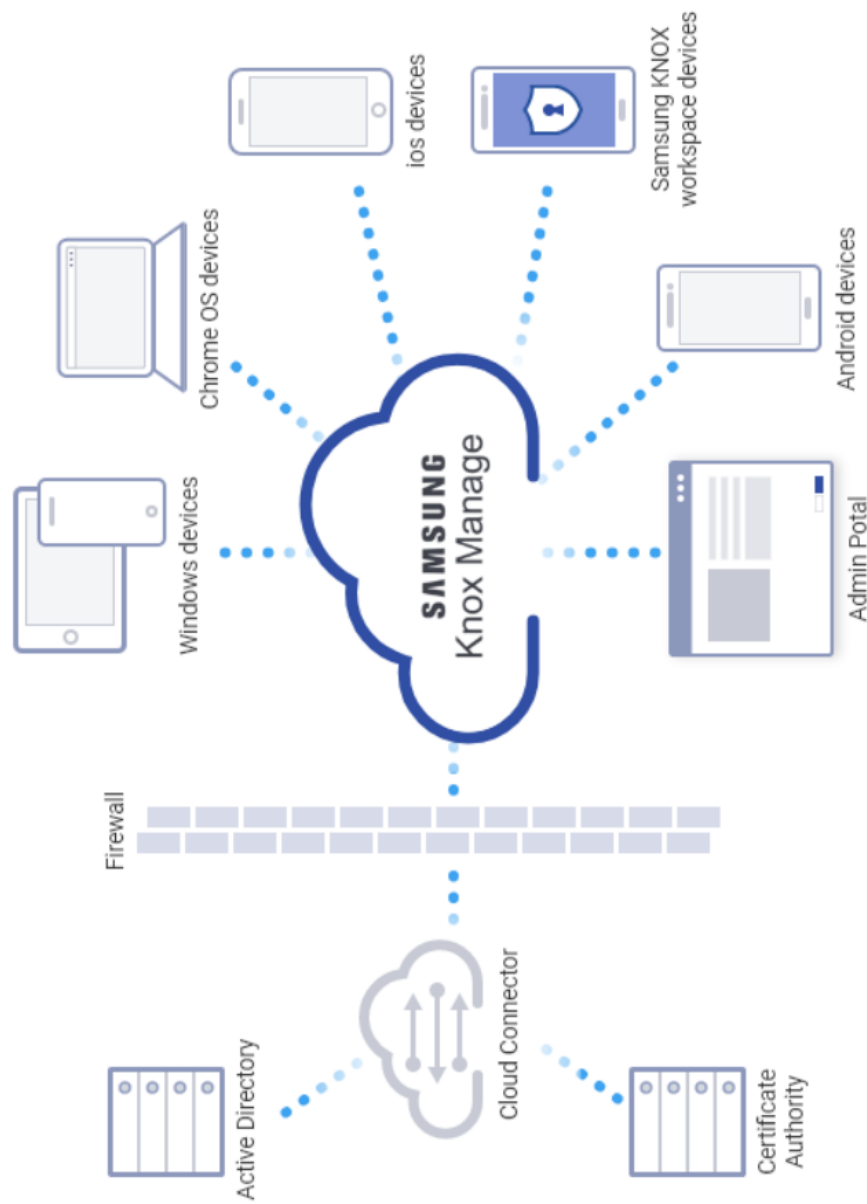
- [Cryptographic Operations](#)

You can encrypt and decrypt data with symmetric or asymmetric encryption, and manage keys with YACA (Yet Another Crypto API). You can also digest messages and create digital signatures.

<https://docs.tizen.org/application/native/guides/security/overview/>.

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, e.g., using encryption keys, signatures, secure buses, etc., programmed into the Samsung Knox MDM platform and e.g., Samsung Knox end-user device software for secure communications with a network system:





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

As a further example, the communications link Samsung Knox Manage servers or Knox MDM platform and devices being managed by such servers or MDM platform, for example including the Knox workspace devices, Android devices, iOS devices, Chrome OS devices, and Windows devices illustrated above, comprise a “service control link” (e.g., a communication link between the server and device) Data and messages transmitted over the secure control link between the Samsung Knox Manage servers or Knox MDM platform and managed devices is encrypted, and commands and notifications transmitted from such servers to devices are “agent messages.” These agent messages are received by “service control device link agent[s],” e.g., software on the device, such as the Knox Manage (KM) agent on Samsung’s devices.

As a further example, the communications link between Samsung's Tizen servers and Tizen OS devices, and Firebase messaging servers and Samsung's Android devices, comprises a "service control link." Notifications, data, and messages transmitted over these service control links from such servers to devices are encrypted, as illustrated above, and are "agent messages." These agent messages are received by "service control device link agent[s]," e.g., software on the device, such as Samsung's Tizen software running on Samsung's devices.

As a further example, Samsung's Knox MDM platform, Knox Manage servers and Tizen servers are all examples of servers which provide control-plane communications (e.g., links between the device agents related to various aspects of device based network service policy implementation), and which are service control server link elements, including because they control, manage, and apply service policies to user devices to which they are connected.

As a further example, devices managed by Samsung Knox Manage or the Knox MDM platform are managed by software including specific end-user device applications and software, such as the Knox Manage (KM) agent on such devices (see, e.g., <https://docs.samsungknox.com/admin/knox-manage/enroll-a-single-device.htm>).

1[b] a plurality of device agents communicatively coupled to the service control device link agent through an agent communication bus, each of the plurality of device agents identifiable by an associated device agent identifier; and

Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise “a plurality of device agents communicatively coupled to the service control device link agent through an agent communication bus, each of the plurality of device agents identifiable by an associated device agent identifier.”

For example, Samsung Galaxy phones and tablets comprise multiple device agents (e.g., operating system software and/or applications) which have an identifier. *See e.g.*,

Firestore > Documentation > FCM > Engage

Was this helpful?

Send feedback

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](#)

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_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:

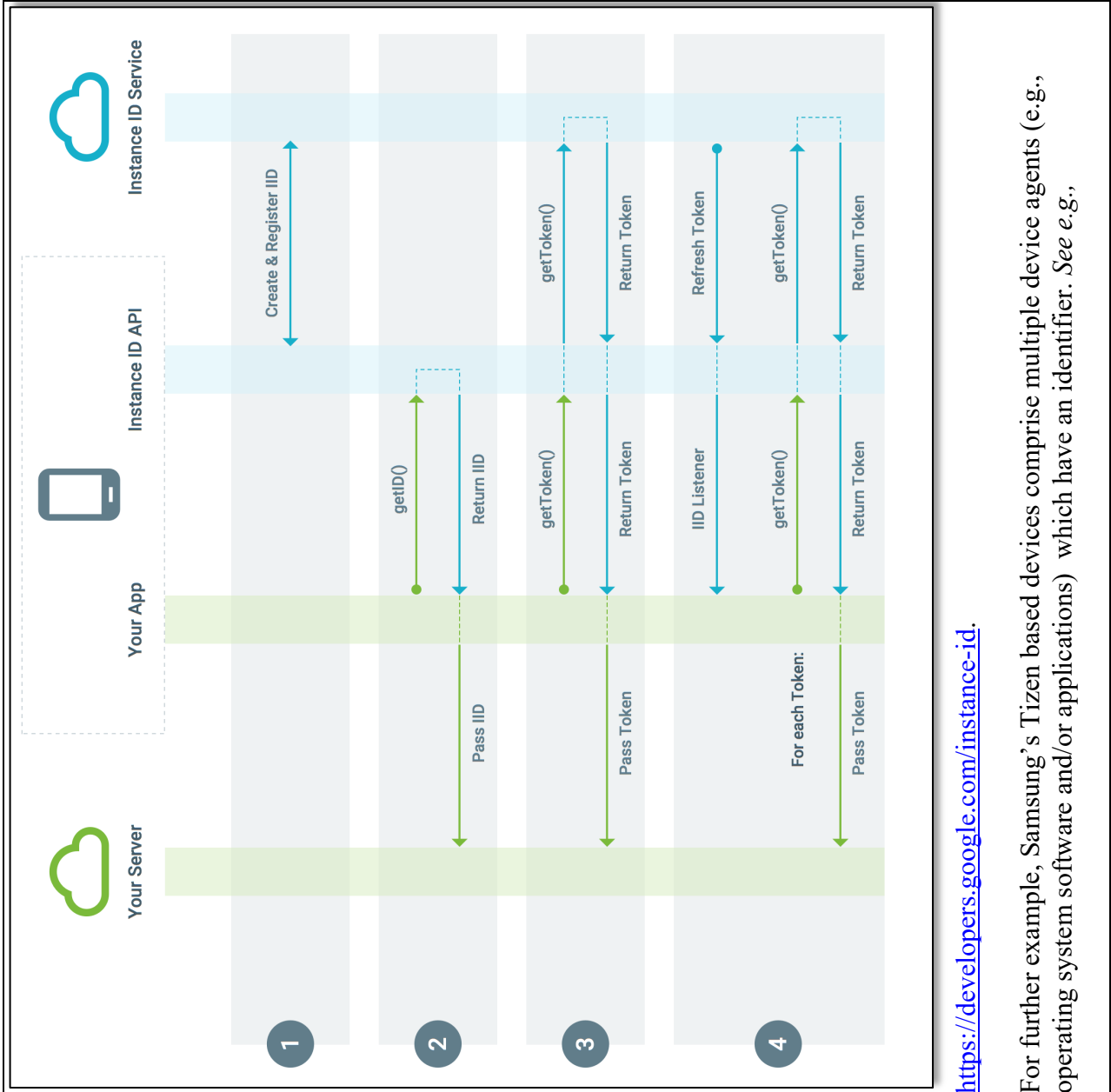
- **Notification messages delivered when your app is in the background.** In this case, the notification is delivered to the device's system tray. A user tap on a notification opens the app launcher by default.
- **Messages with both notification and data payload, when received in the background.** In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

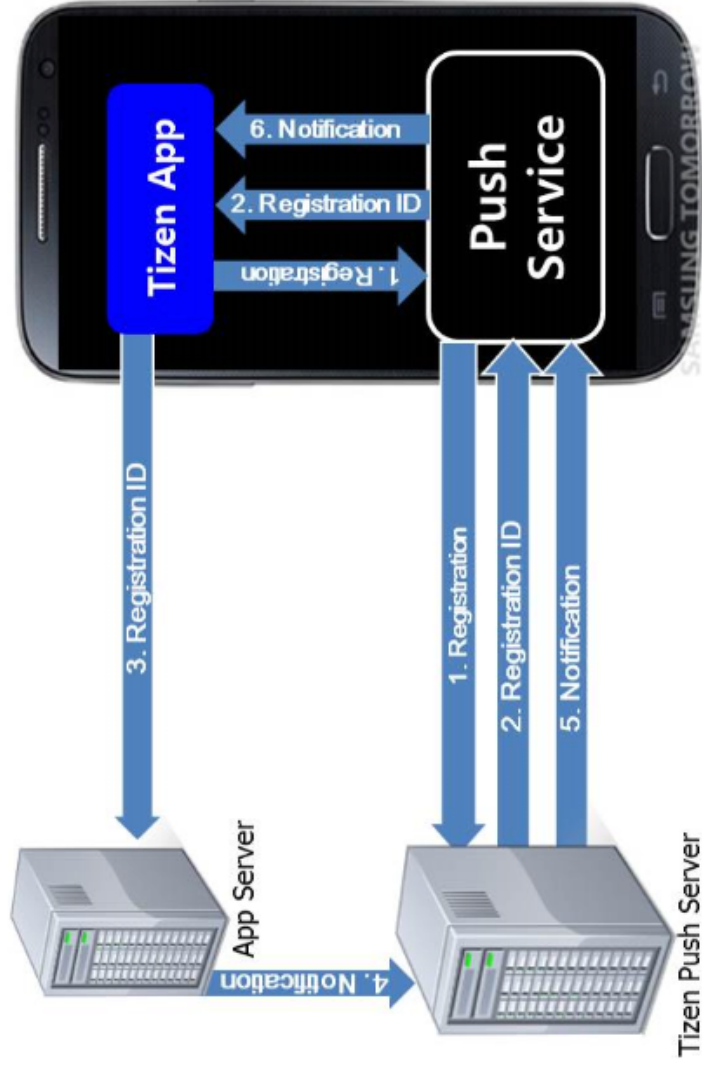
Instance ID provides a unique ID per instance of your apps. You can implement Instance ID for [Android](#) and [iOS](#) apps as well as [Chrome apps/extensions](#).



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);
```

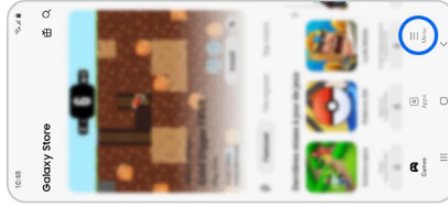
If the registration is successful, the `registerSuccessfulCallback()` callback is called, and the registration ID is passed as a parameter. Any time after a successful registration, you can get the registration ID using the `getRegistrationId()` method:

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

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

How to set up push notifications

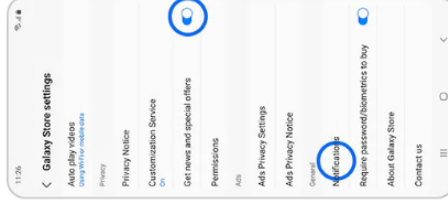
If you want to set up push notifications, follow these steps.



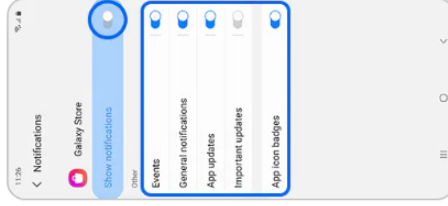
Step 1. Log in to Galaxy Store and tap the "Menu" icon from the main screen.



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



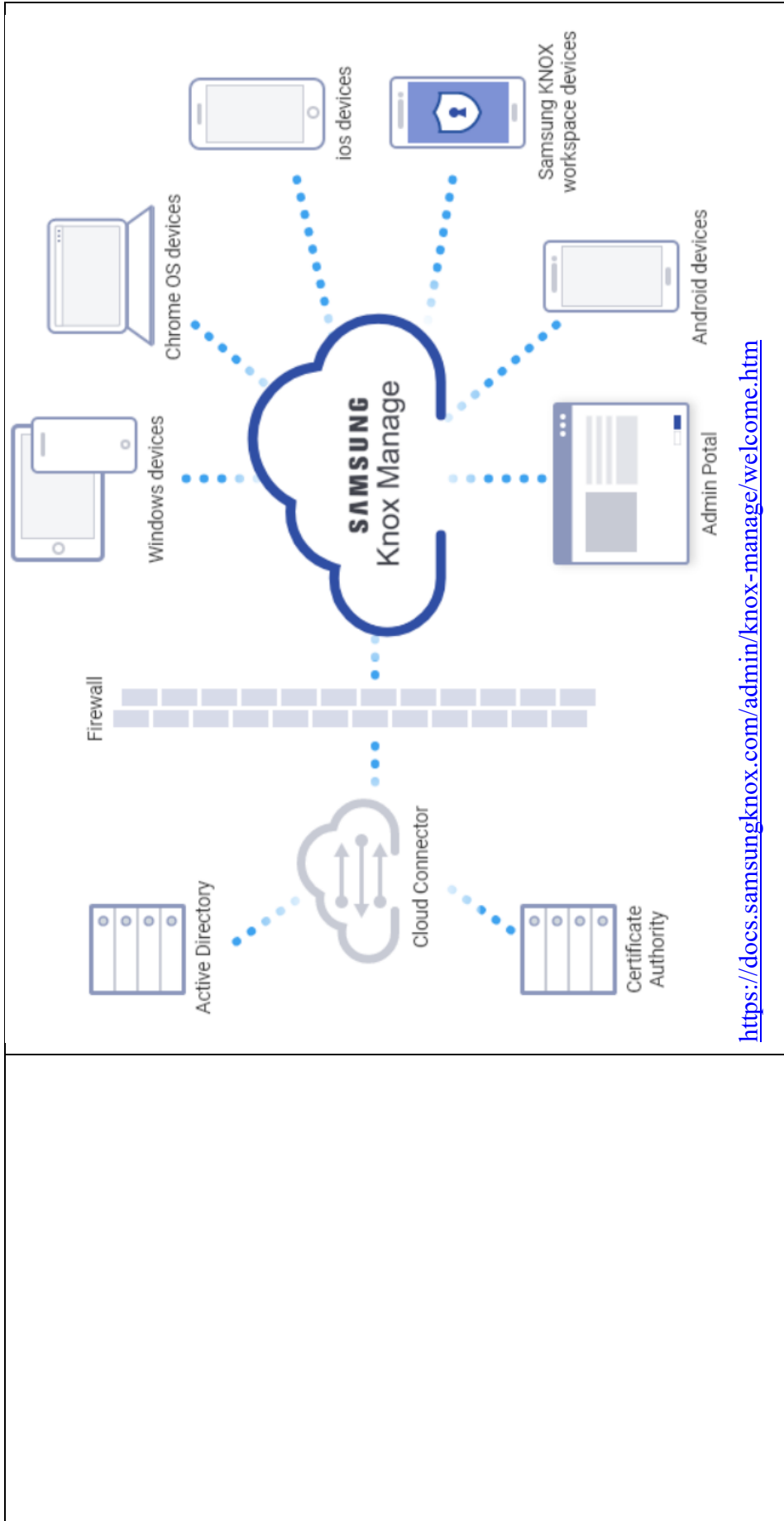
Step 3. Enable "Get news and special offers" and tap "Notifications."



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, comprise multiple device agents which have an identifier.



	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 As a further illustration, the service control device link agents on Samsung's devices are coupled to various other device agents within the device by way of the Android operating system through an agent communication bus (e.g., communication path for various agents and functions to communicate over). That communication structure allows for intercommunication between agents in the device, and enables the service control device link agents to send messages to (and control) behavior of other components within the system. The communications bus connecting device agents to the service control device link agents need not be internal to the device, and the communications channel between Samsung Knox and MDM servers and managed devices is an example of such a communications bus.
1[c] memory configured to store an encryption key, the encryption key shared between the service control device link agent and a service control server link element of the network system;	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise "memory configured to store an encryption key, the encryption key shared between the service control device link agent and a service control server link element of the network system." For example, Samsung Galaxy phones and tablets include memory that stores an encryption key (e.g. keys, tokens, or signatures used by device and server to encrypt communications, such as public/private keys, shared keys, partially shared keys, symmetric and/or asymmetric encryption, etc., such as those used in transport layer encryption, end-to-end encryption, point-to-point encryption, etc.) that is shared between the agent and the network system. See, e.g.,

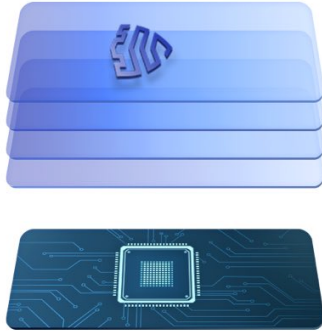
Storage

Save your heart out.⁸

1TB	-	-
512GB	-	-
256GB	256GB	256GB
128GB	128GB	128GB

<https://www.samsung.com/us/smartphones/galaxy-s22/models/>;

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

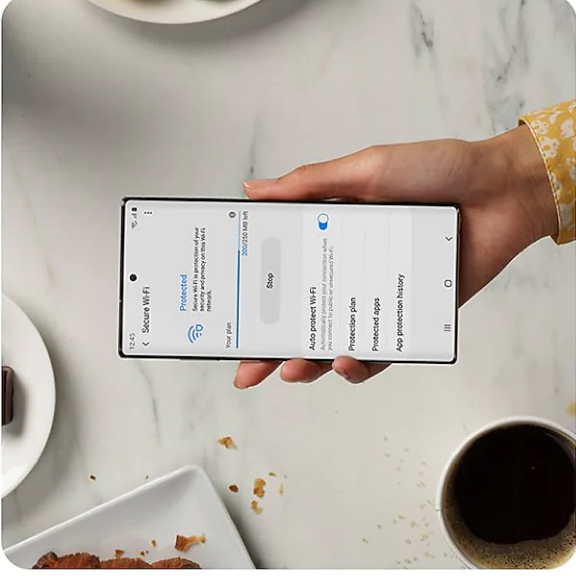
Run-time protection protects your device against data attacks or malware. Any unauthorized or unintended attempts to access or modify your phone's core are blocked in real time.

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](#)



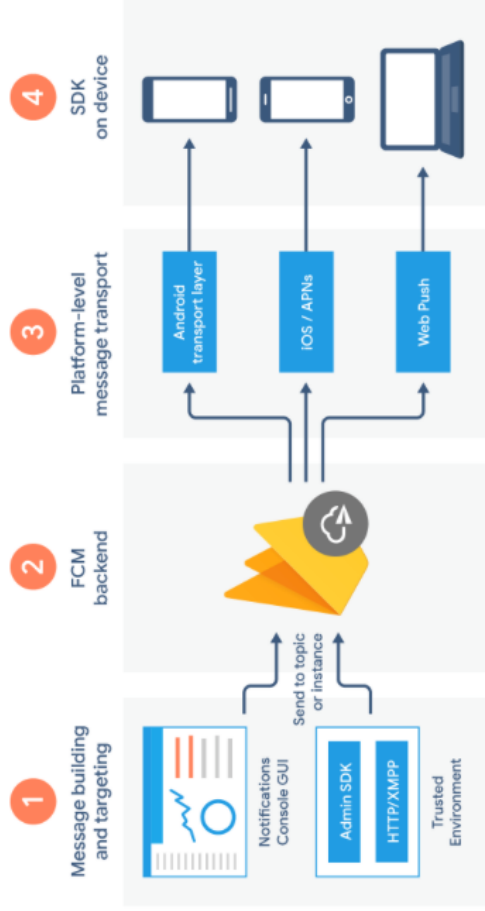
<https://www.samsung.com/us/security/>;

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.

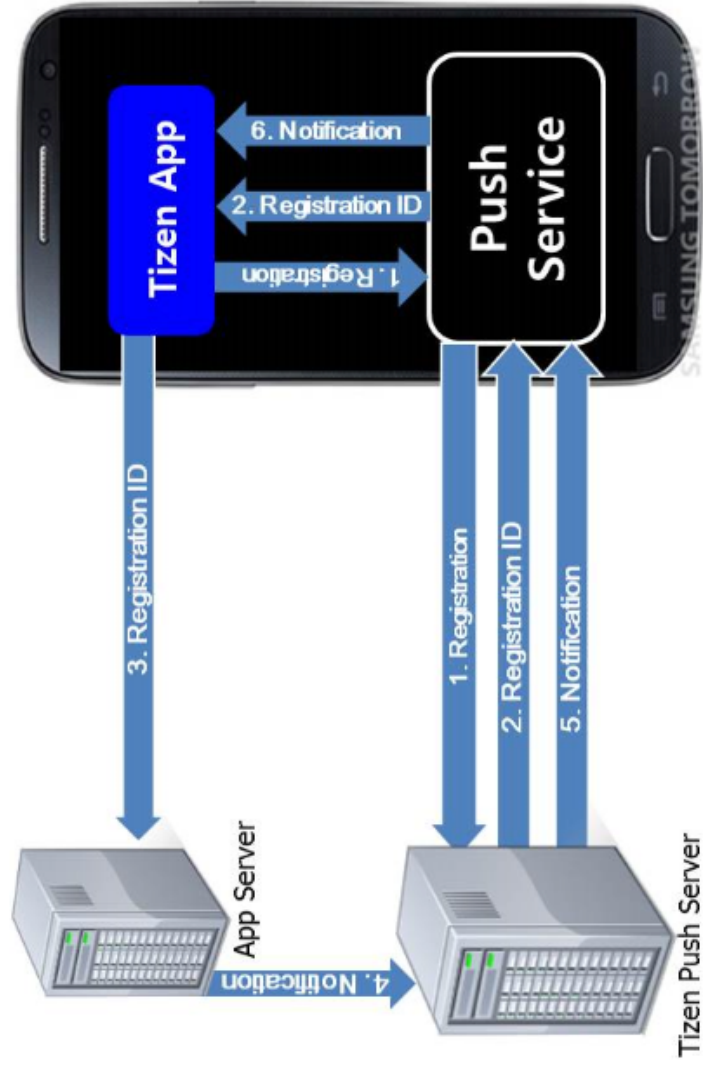
As another example, Samsung's Tizen based devices include memory that stores an encryption key (e.g. keys, tokens, or signatures used by device and server to encrypt communications, such as public/private keys, shared keys, partially shared keys, symmetric and/or asymmetric encryption, etc., such as those used in transport layer encryption, end-to-end encryption, point-to-point encryption, etc.) that is shared between the agent and the network system. See e.g.:

	Wireless Connectivity Wi-Fi ? Yes (WiFi5) Wi-Fi Direct Yes Bluetooth ? Yes (BT5.2) https://www.samsung.com/us/televisions-home-theater/tvs/the-frame/55-class-the-frame-qlc4k-smart-tv-2022-qn55ls03bafxza/#specs; Wireless security protocols The TV only supports the following wireless network security protocols. The TV cannot connect to non-certified wireless access point. – Authentication Modes: WEP, WPA2PSK, WPA2PSK – Encryption Types: WEP, TKIP, AES https://downloadcenter.samsung.com/content/UM/202203/202203092001587/OSNATSCB-3.2.0_EM_Oscar_Pontus_Nike_Kant-SU2e_USA_ENG_220216.0.pdf;
--	--

Service Architecture

The following figure illustrates the service architecture of the Tizen push messaging service.

Figure: Service architecture



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.

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

Security

The security features introduce how private information remains private, and how the user knows when they are trying to access privileged information. You can use a repository and cryptographic operations to manage keys, certificates, and sensitive user data. When the user tries to access privileged information, you can display information about the data.

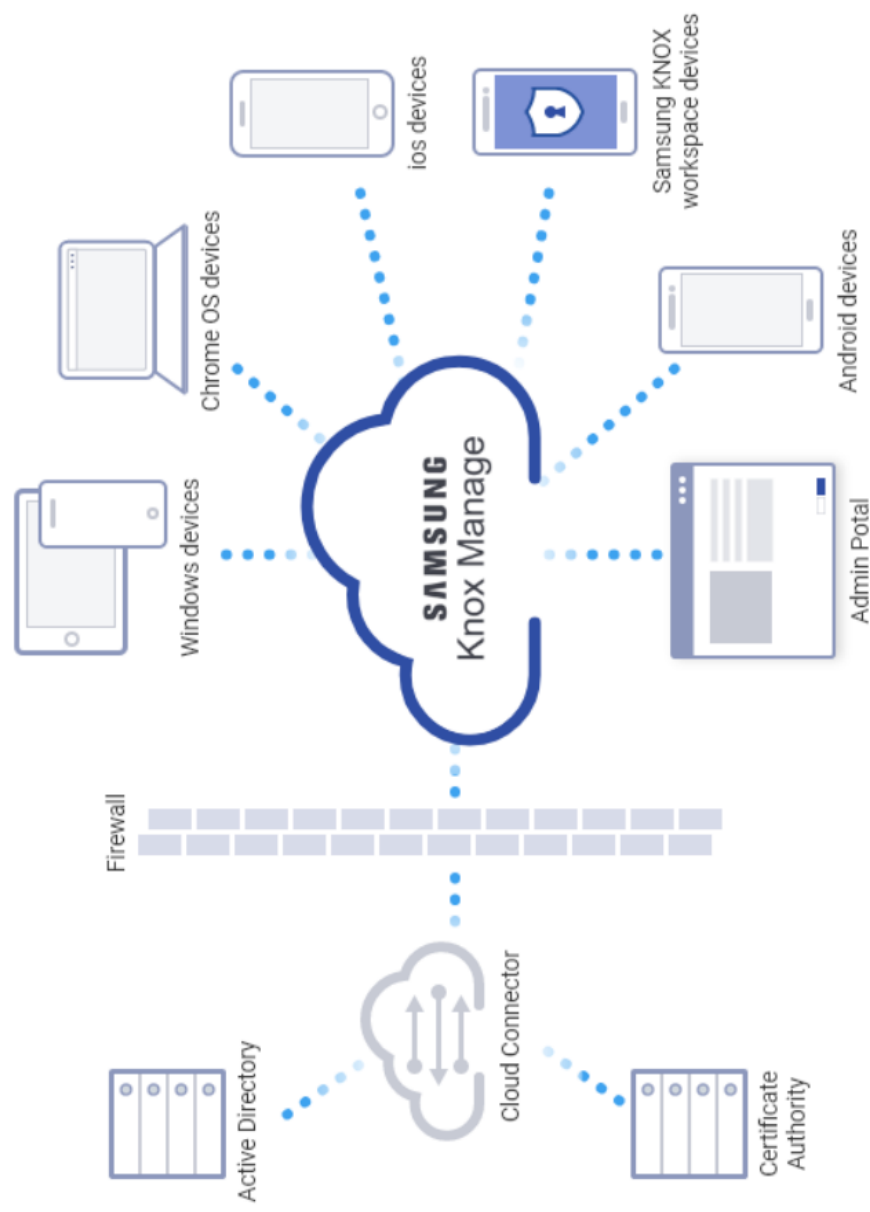
You can use the following security features in your native applications:

- [Cryptographic Operations](#)

You can encrypt and decrypt data with symmetric or asymmetric encryption, and manage keys with YACA (Yet Another Crypto API). You can also digest messages and create digital signatures.

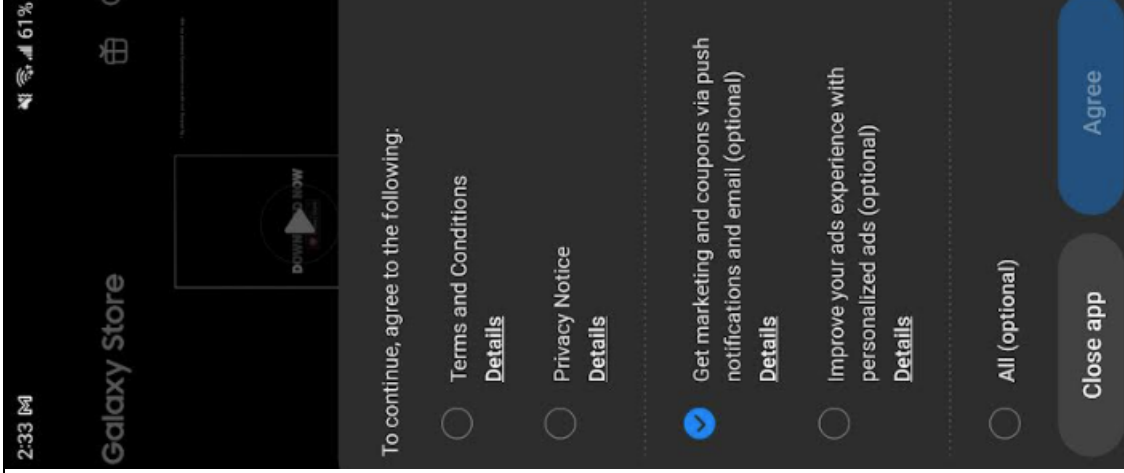
<https://docs.tizen.org/application/native/guides/security/overview/>.

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 memory that stores an encryption key (e.g. keys, tokens, or signatures used by device and server to encrypt communications, such as public/private keys, shared keys, partially shared keys, symmetric and/or asymmetric encryption, etc., such as those used in transport layer encryption, end-to-end encryption, point-to-point encryption, etc.) that is shared between the agent and the network system.



<https://docs.samsungknox.com/admin/knox-manage/welcome.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 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. <i>See, e.g.:</i>
--	--



Upon information and belief, the information needed for encryption/decryption of messages between nodes and devices in the architecture of the systems used with Samsung's Accused Products are all stored in memory, which is a standard and conventional practice for how information needed to, for example, decrypt incoming communications or messages or to

	authenticate that a given message received from an external source (such as another part of the network of devices) is to be trusted.
1[d] wherein the service control device link agent is configured to: receive, over the service control link, an encrypted agent message from the service control server link element,	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise a “service control device link agent [which] is configured to: receive, over the service control link, an encrypted agent message from the service control server link element.” For example, Samsung Galaxy phones and tablets receive encrypted messages (e.g., messages and/or payloads) from a server using software on the device, such as the operating system or applications on the device, to receive messages, notifications, payloads, etc. The server and <i>See e.g.,</i> Firestore > Documentation > FCM > Engage Was this helpful?  Send feedback 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

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:

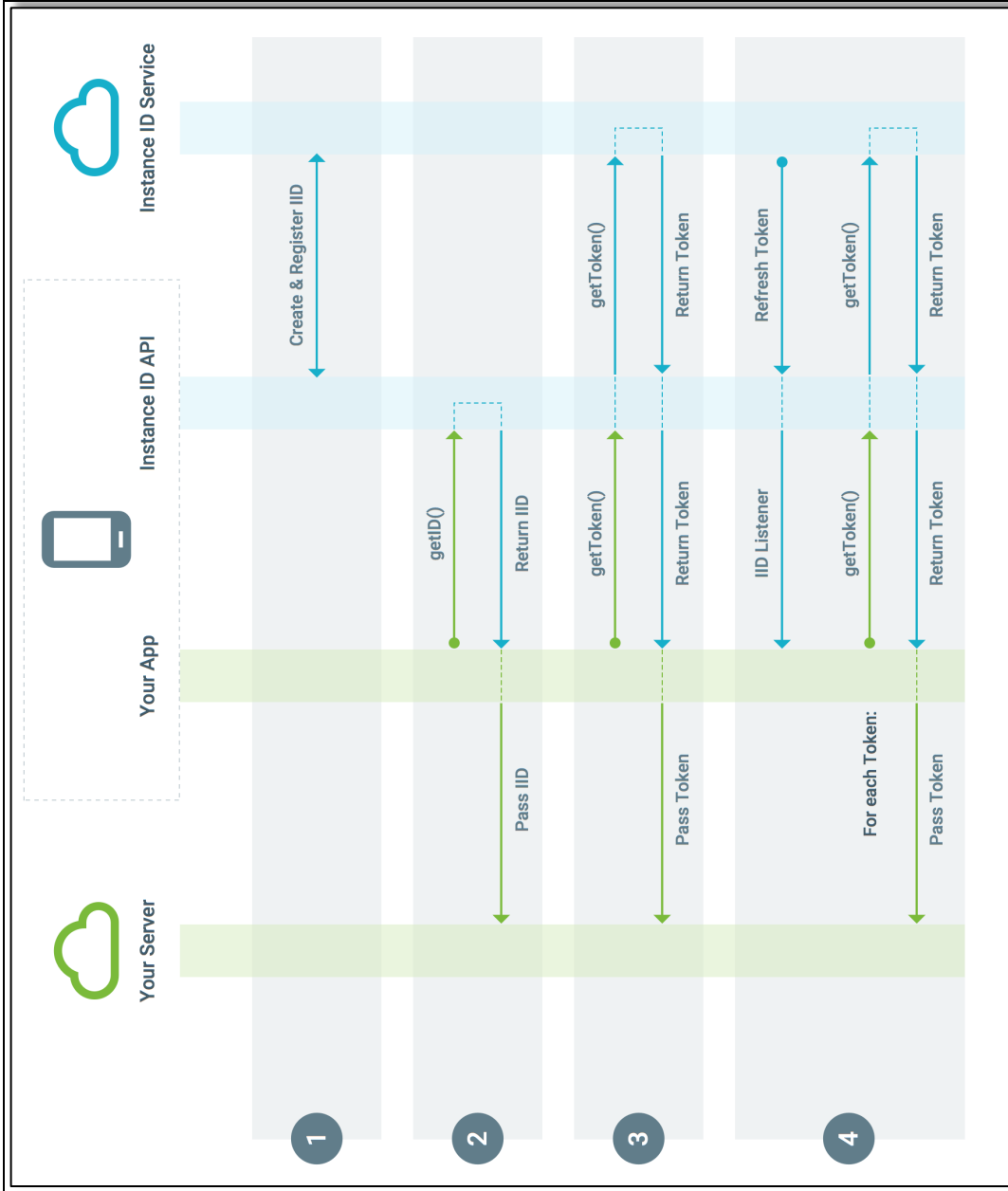
- **Notification messages delivered when your app is in the background.** In this case, the notification is delivered to the device's system tray. A user tap on a notification opens the app launcher by default.
- **Messages with both notification and data payload, when received in the background.** In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



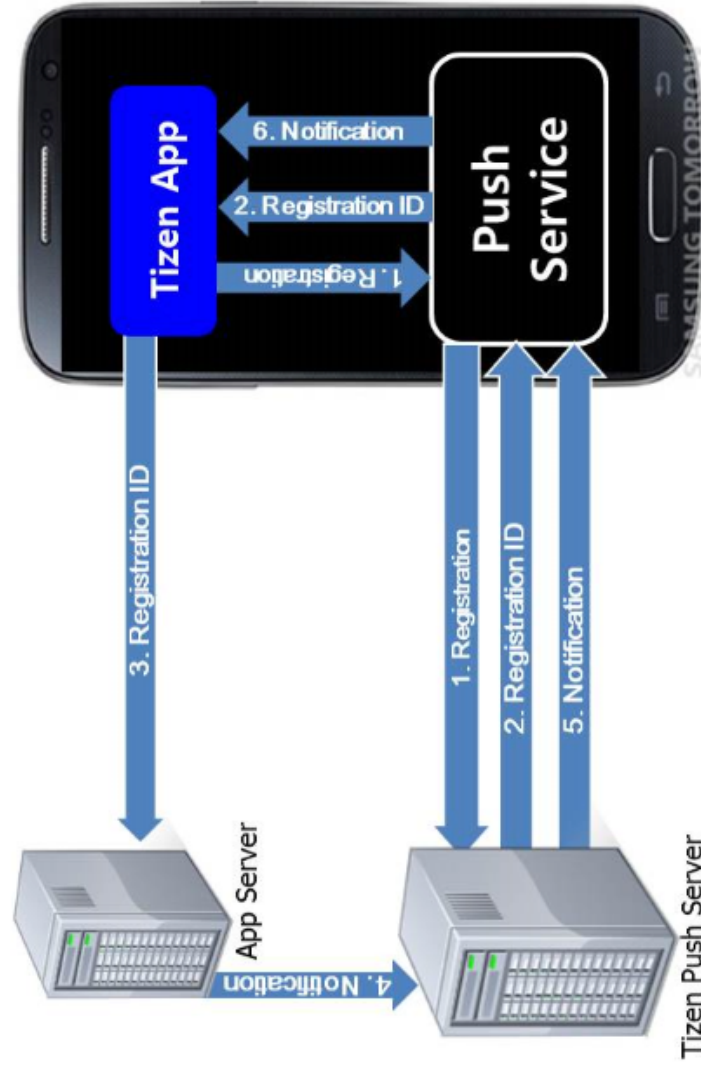
<https://developers.google.com/instance-id>.

For further example, Samsung's Tizen based devices receive encrypted messages from a server.
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 `PushRegistrationStateChangedCallback` (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.

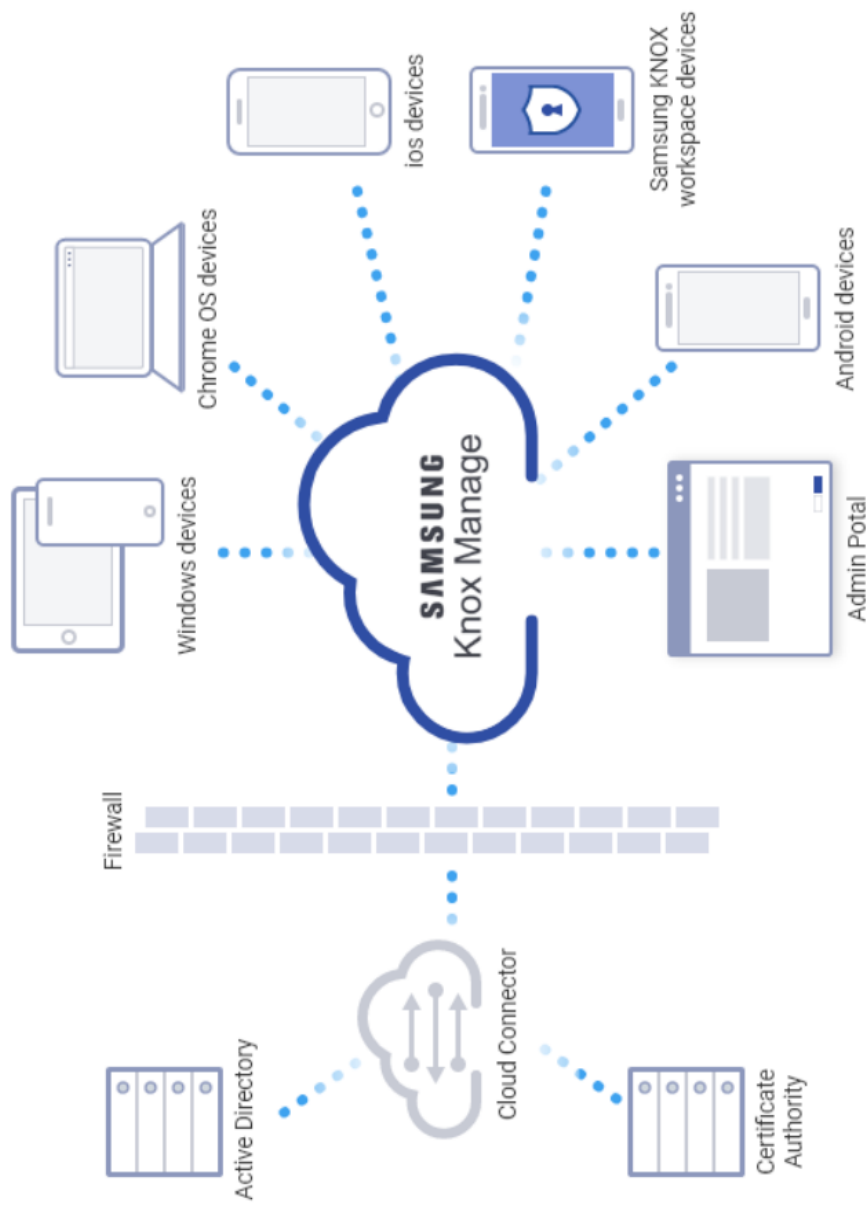
The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification.

- 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.

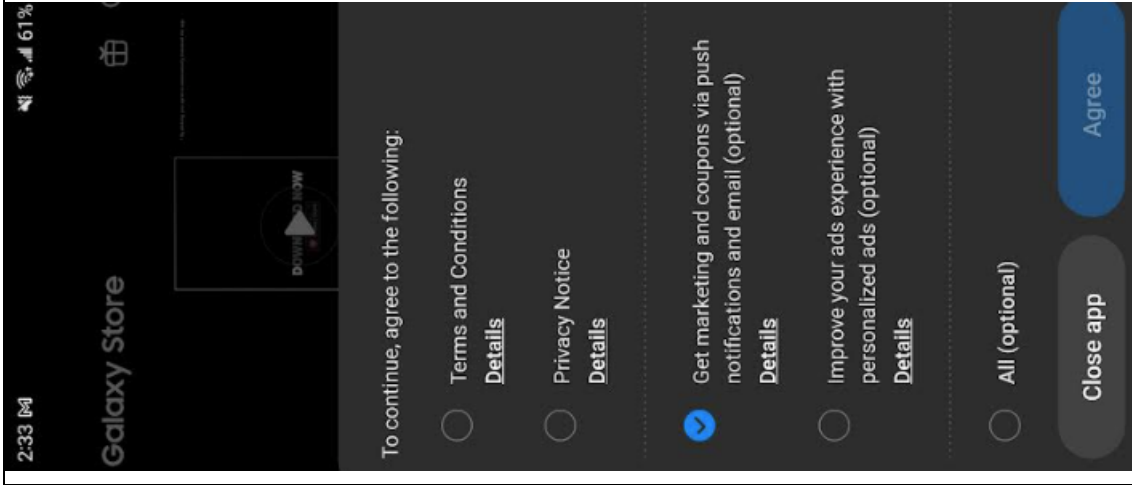
<https://docs.tizen.org/application/web/guides/messaging/push/>.

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, receive encrypted messages from a server.



<https://docs.samsungknox.com/admin/knox-manage/welcome.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 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. <i>See, e.g.:</i>
--	--

	 For avoidance of doubt, service control server link element
1[e] using the encryption key, obtain a decrypted agent message, the decrypted	Samsung Galaxy phones and tablets and Samsung’s Tizen based devices “us[e] the encryption key, obtain a decrypted agent message, the decrypted agent message comprising a particular

agent message comprising a particular agent identifier and message content for delivery to a particular device agent of the plurality of device agents, the particular agent identifier identifying the particular device agent, the message content from a particular server of a plurality of servers communicatively coupled to the service control server link element, and

agent identifier and message content for delivery to a particular device agent of the plurality of device agents, the particular agent identifier identifying the particular device agent, the message content from a particular server of a plurality of servers communicatively coupled to the service control server link element.”

For example, Samsung Galaxy phones and tablets decrypt messages from a plurality of servers which have identifiers and content for delivery to particular device agents. *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

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:

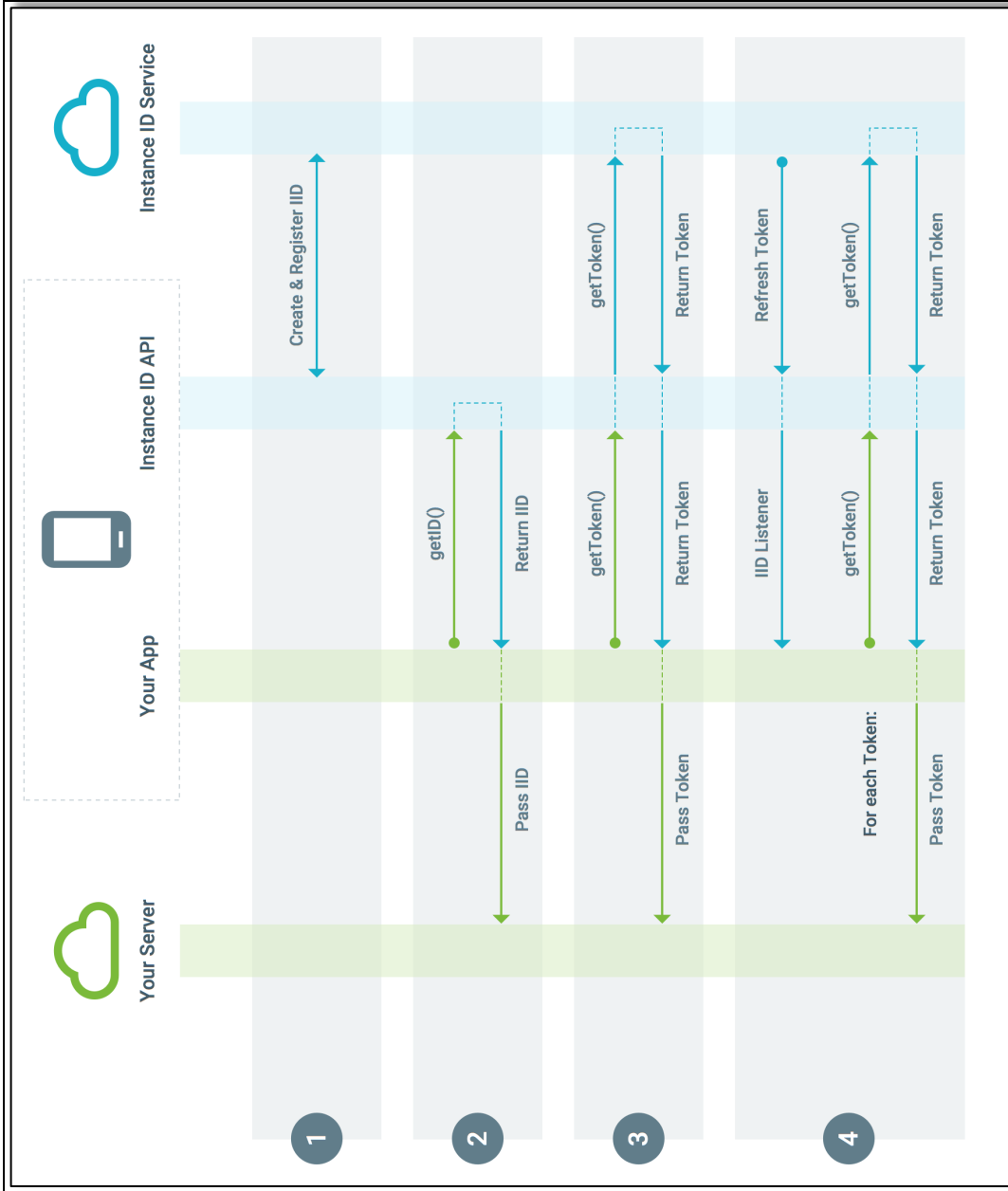
- **Notification messages delivered when your app is in the background.** In this case, the notification is delivered to the device's system tray. A user tap on a notification opens the app launcher by default.
- **Messages with both notification and data payload, when received in the background.** In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



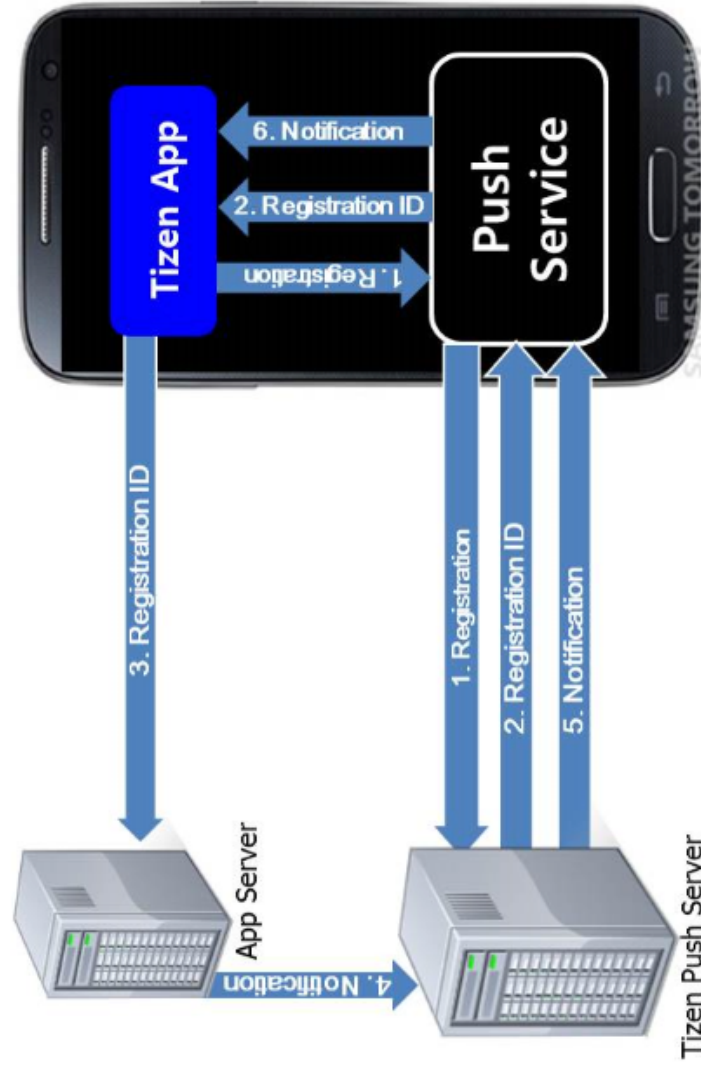
<https://developers.google.com/instance-id>.

For further example, Samsung's Tizen based devices receive encrypted messages from a server.
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.

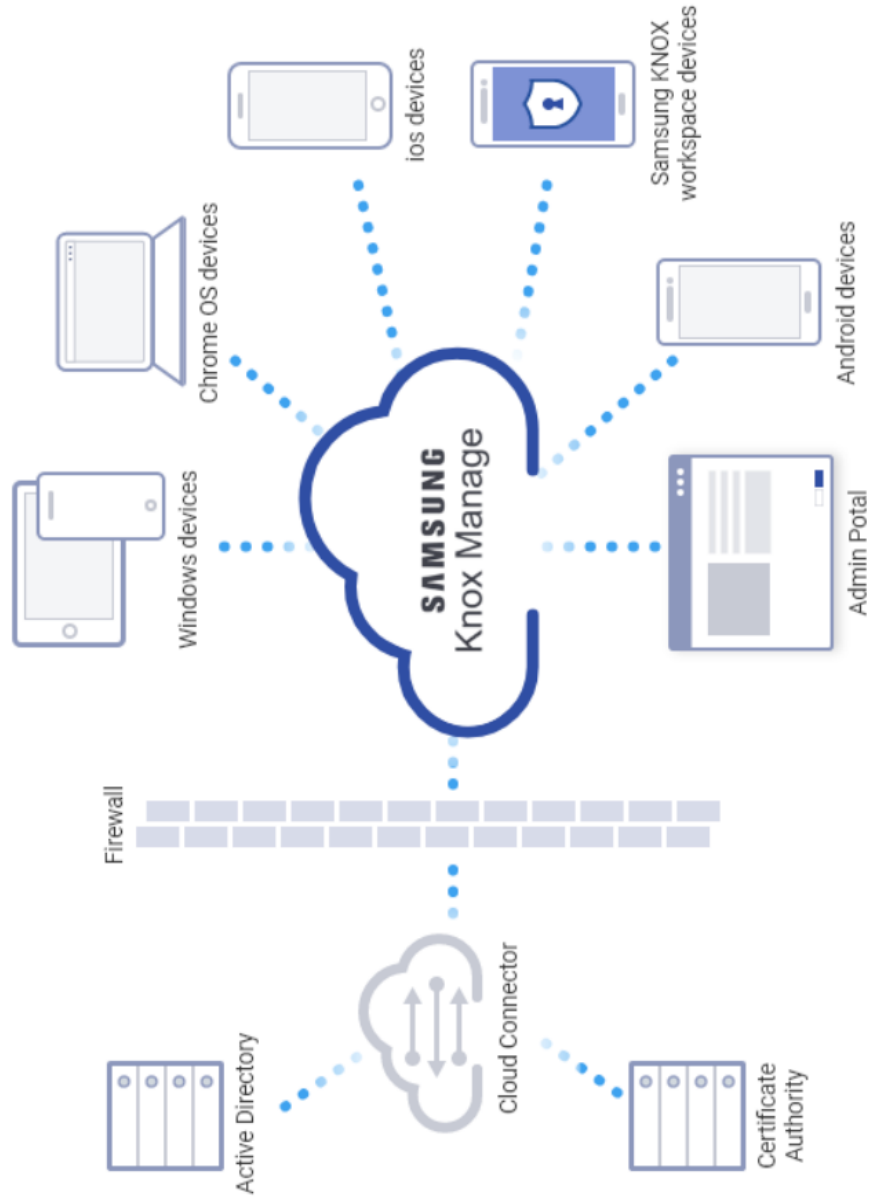
- Discard the notification.

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, decrypt

messages from a plurality of servers which have identifiers and content for delivery to particular device agents.



<https://docs.samsungknox.com/admin/knox-manage/welcome.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
1[f] based on the particular agent identifier, deliver the message content to the particular device agent over the agent communication bus.	Samsung Galaxy phones and tablets and Samsung's Tizen based devices “based on the particular agent identifier, deliver the message content to the particular device agent over the agent communication bus.” For example, Samsung Galaxy phones and tablets deliver message content to particular device agents based on identifier. <i>See e.g.,</i>

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:

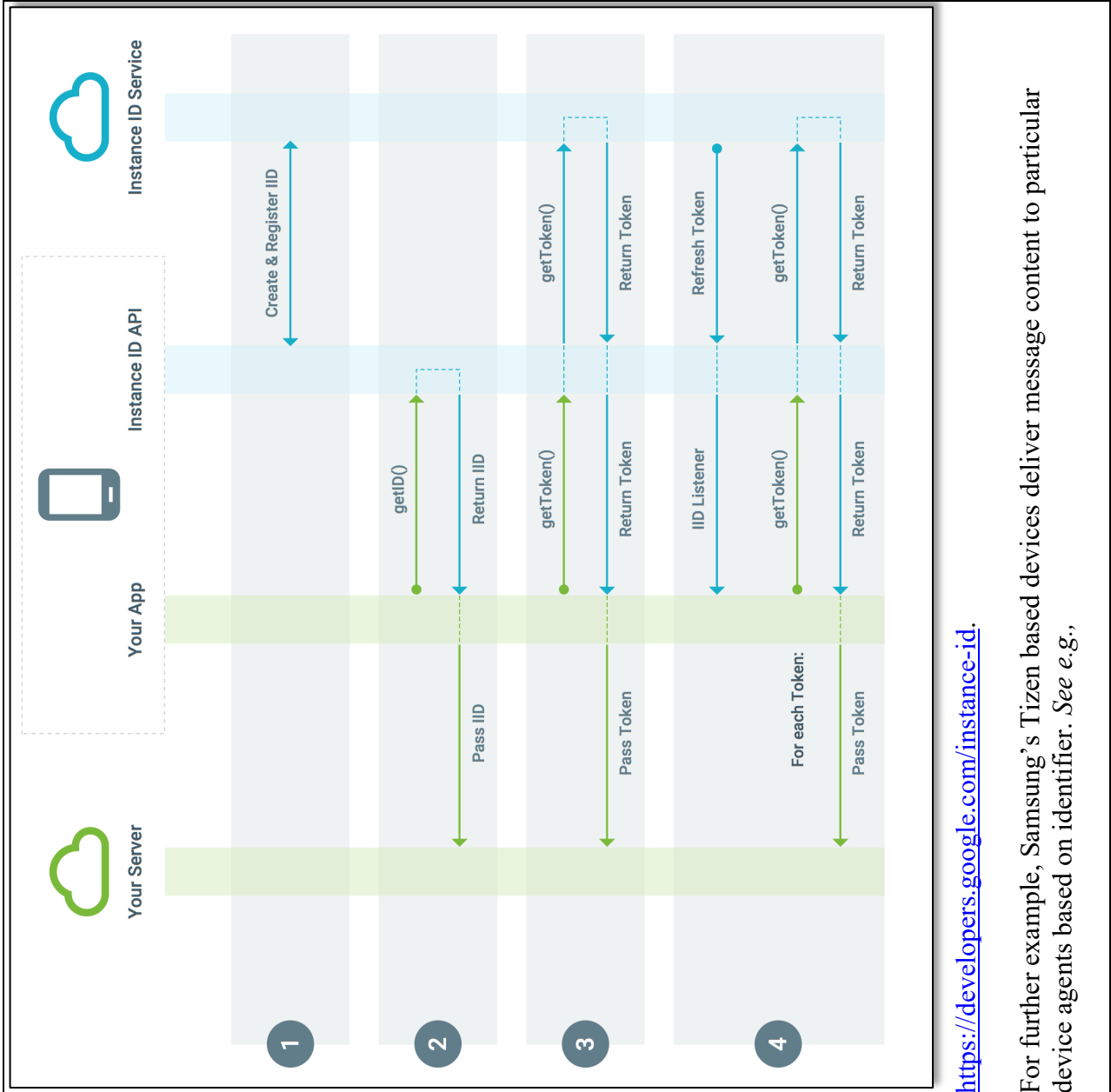
- **Notification messages delivered when your app is in the background.** In this case, the notification is delivered to the device's system tray. A user tap on a notification opens the app launcher by default.
- **Messages with both notification and data payload, when received in the background.** In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

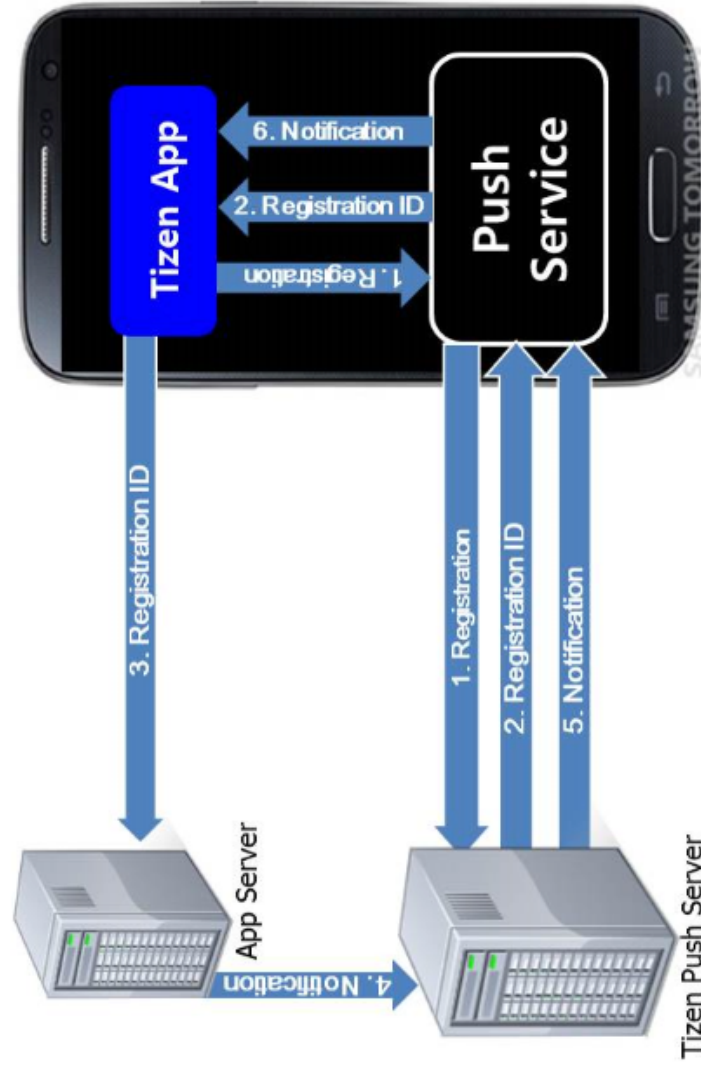
Instance ID provides a unique ID per instance of your apps. You can implement Instance ID for [Android](#) and [iOS](#) apps as well as [Chrome apps/extensions](#).



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);
}
```

decrypt messages from a plurality of servers which have identifiers and content for delivery to particular device agents.

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

`PushRegistrationStateChangedCallback` (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.

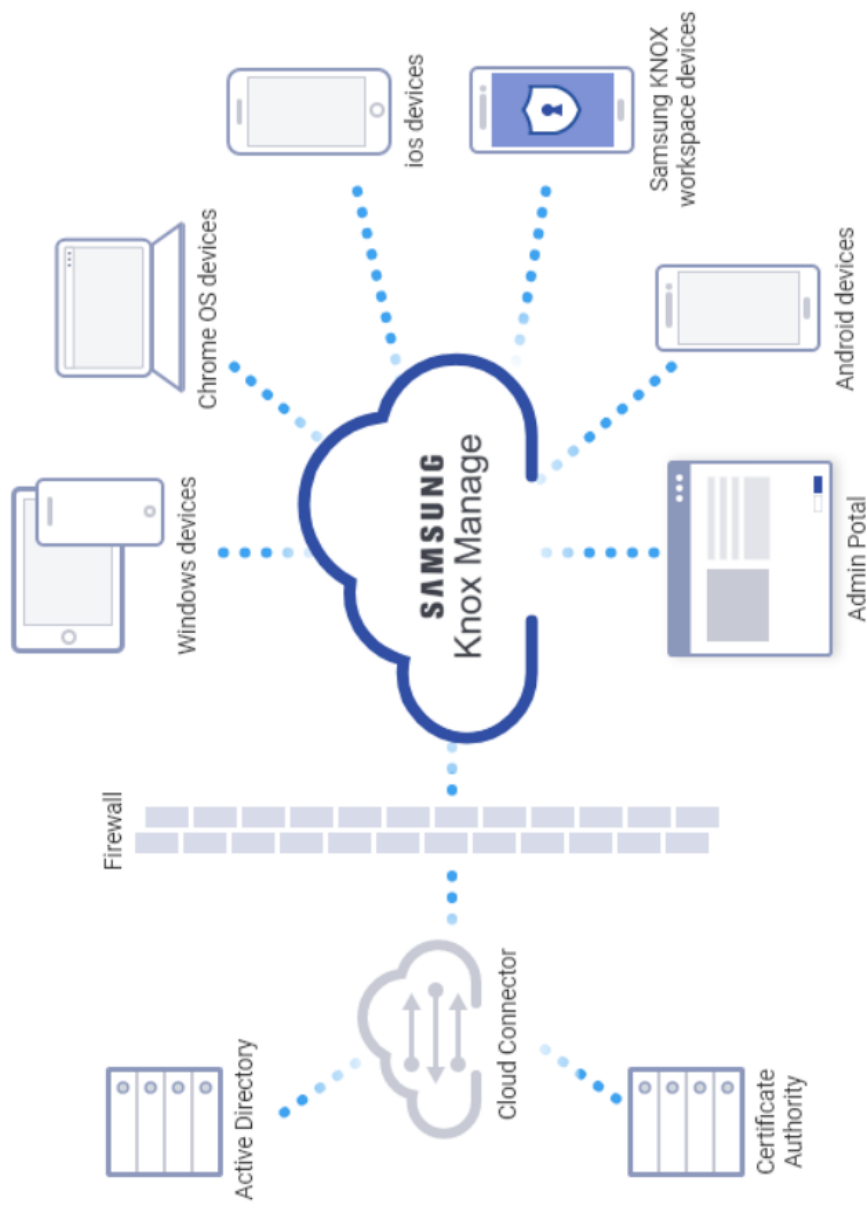
The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification.

- 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, deliver message content to particular device agents based on identifier.



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

Push an update

To push an update to an existing profile:

1. Select **Profiles** from the left-hand navigation menu.
2. Select the check boxes of the target device(s) to receive the profile push update.
3. Select **Push update** from the **ACTIONS** drop-down menu.
4. Select the **Push update without requesting end user consent** to push the updates directly to the devices without consent from the device end user.

×

Push ZEN Profile Update

☒ Push to all associated devices

☒ Only changed items

☐ Full profile update

☐ Push to specific associated devices

☒ Push update without requesting end-user consent

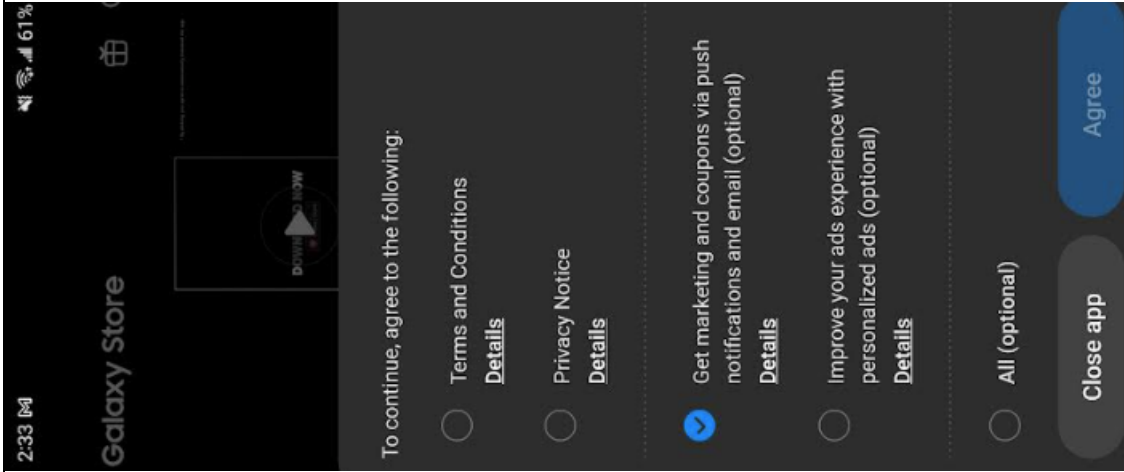
CANCEL

SCHEDULE

PUSH NOW

<https://docs.samsungknox.com/admin/knox-configure-wearables/update-existing-profile.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. <i>See, e.g.:</i>
--	--



Claim 2

2. The end-user device recited in claim 1, wherein the particular server comprises a service usage history server, a policy management server, an access control integrity server, a network traffic analysis server, a beta test server, a service download control server, a billing event server, an activation server, a transaction server, an authentication server, or a content management server.

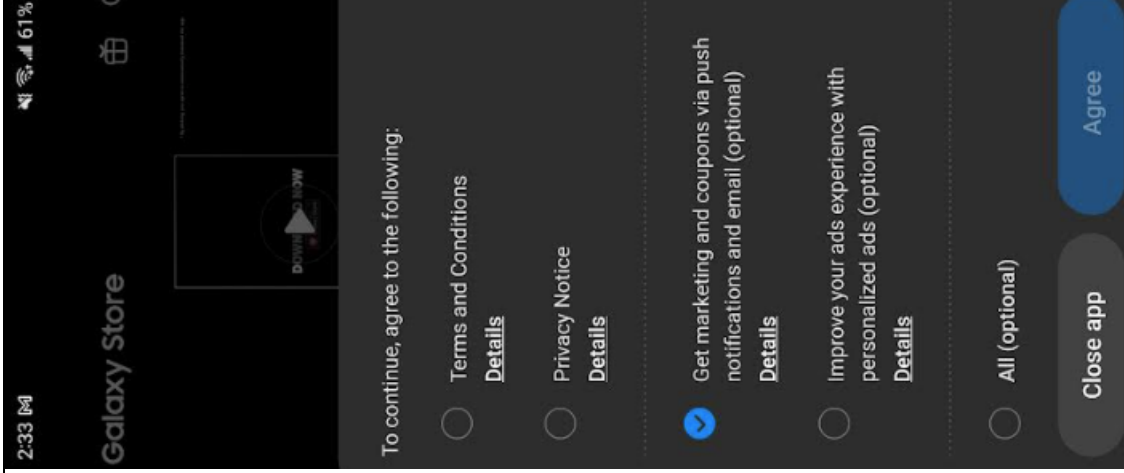
Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise the "end-user device recited in claim 1, wherein the particular server comprises a service usage history server, a policy management server, an access control integrity server, a network traffic analysis server, a beta test server, a service download control server, a billing event server, an activation server, a transaction server, an authentication server, or a content management server."

See supra claim 1.

For end-user devices being used within the Samsung Knox ecosystem (including devices with administrator privileges within a particular Knox MDM), for example mobile phones and other devices enrolled in the Samsung Knox MDM platform, the "particular server" of claim 2 includes servers which comprise any of the specific types of servers listed in this limitation. This includes, without limitation, servers which are a "service usage history server" (<https://docs.samsungknox.com/admin/knox-manage/monitor/service-history/>; <https://docs.samsungknox.com/admin/knox-manage/monitor/device-logs/>; <https://docs.samsungknox.com/admin/knox-manage/monitor/audits/audit-logs/view-audit-logs/>), "policy management server" (<https://www.samsung.com/us/business/solutions/services/mobility-software/knox-manage/>; <https://docs.samsungknox.com/admin/knox-mobile-enrollment/how-to-guides/manage-devices/device-status/>), "access control integrity server" (<https://docs.samsungknox.com/admin/fundamentals/knox-licenses/>), "network traffic analysis server" (<https://docs.samsungknox.com/dev/knox-sdk/network-platform-analytics.htm>), "activation server" (<https://docs.samsungknox.com/admin/knox-mobile-enrollment/how-to-guides/manage-devices/device-status/>), or "authentication server" (<https://docs.samsungknox.com/admin/knox-mobile-enrollment/how-to-guides/manage-devices/device-status/>). See also: <https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-devices/send-notifications/>. Each of the components in the Knox MDM framework responsible for these functions (i.e., activation, enrollment, usage monitoring, push messaging to managed devices, etc.) are referred to by Samsung as being a part of the "Knox Manage" servers, but in reality are functionalities distributed across at least multiple servers.

As another example, Samsung's Tizen based devices receive message content from a "particular server" where the particular server may any of the app servers communicating with Samsung's Tizen push server in causing message content to be delivered to Tizen-based devices, which

	may comprise any of the specific types of servers listed in this limitation. For example, see https://developer.samsung.com/smarttv/develop/api-references/tizenfx-tv-api-references/tizen.tv.service.billing.html (comprising transaction, authentication, billing event servers). As another example, Samsung’s Android-based devices receive push messages (message content) via functionality provided by Firebase cloud messaging servers, which in turn receives message content from servers programmed to send push messages to instances of their application running on Samsung’s Android-based devices. Such servers are “particular servers” and include, depending on the use case, any of the specific types of servers listed in this limitation. For further example, 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’s SmartThings application uses push notifications depending on the use case. <i>See, e.g.,</i> https://www.samsung.com/us/support/troubleshooting/TSG01110148/. 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. <i>See, e.g.:</i>
--	---



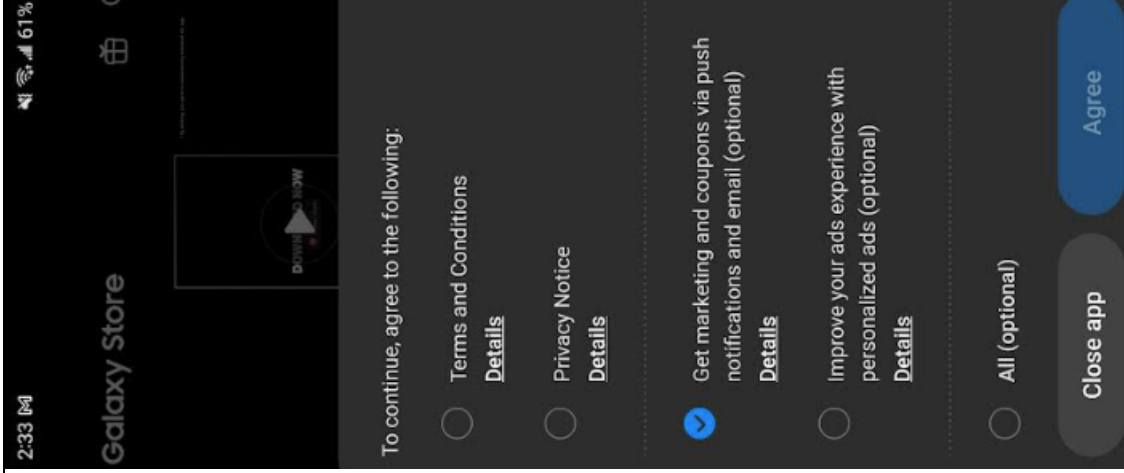
3. The end-user device recited in claim 1, wherein the message content comprises information associated with a service usage.

Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise the "end-user device recited in claim 1, wherein the message content comprises information associated with a service usage."

See supra claim 1.

For end-user devices being used within the Samsung Knox ecosystem (including devices with administrator privileges within a particular Knox MDM), for example mobile phones and other devices enrolled in the Samsung Knox MDM platform, message content received by such devices may comprise information associated with service usage. *See, e.g.,* <https://docs.samsungknox.com/admin/knox-manage/monitor/service-history/>; <https://docs.samsungknox.com/admin/knox-manage/kbas/kba-360046891233/>; <https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-devices/send-notifications/>; <https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-devices/start-and-stop-blinking-reminder/>. As another example, Samsung's Tizen based devices receive message content from a "particular server" where the message content is information associated with service usage. Notifications received by Samsung's Tizen-based devices from Tizen push servers may comprise information the user's particular use of network data service through a given application. As another example, Samsung's Android-based devices receive push messages (message content) via functionality provided by Firebase cloud messaging servers, which in turn receives message content comprising information associated with service usage.

For further example, 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's SmartThings application uses push notifications depending on the use case. *See, e.g.,* <https://www.samsung.com/us/support/troubleshooting/TSG01110148/>. 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.:*



Claim 4

4. The end-user device recited in claim 3, wherein the information associated with the service usage comprises information about one or more of a service usage value, a projected service usage value, a projected service usage plan limit, a projected service usage plan period time cost overage, a service plan period time duration, a service plan time remaining before end of period, and a service overage.

Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise the "end-user device recited in claim 3, wherein the information associated with the service usage comprises information about one or more of a service usage value, a projected service usage value, a service usage plan limit, a projected service usage overage, a projected service cost overage, a service plan period time duration, a service plan time remaining before end of period, and a service overage."

See supra claim 1, 3.

For end-user devices being used within the Samsung Knox ecosystem (including devices with administrator privileges within a particular Knox MDM), for example mobile phones and other devices enrolled in the Samsung Knox MDM platform, message content received by such devices may comprise information associated with service usage. *See, e.g.,*

<https://docs.samsungknox.com/admin/knox-manage/monitor/service-history/>;

<https://docs.samsungknox.com/admin/knox-manage/kbas/kba-360046891233/>;

<https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-devices/send-notifications/>. As another example, Samsung's Tizen based devices receive message content

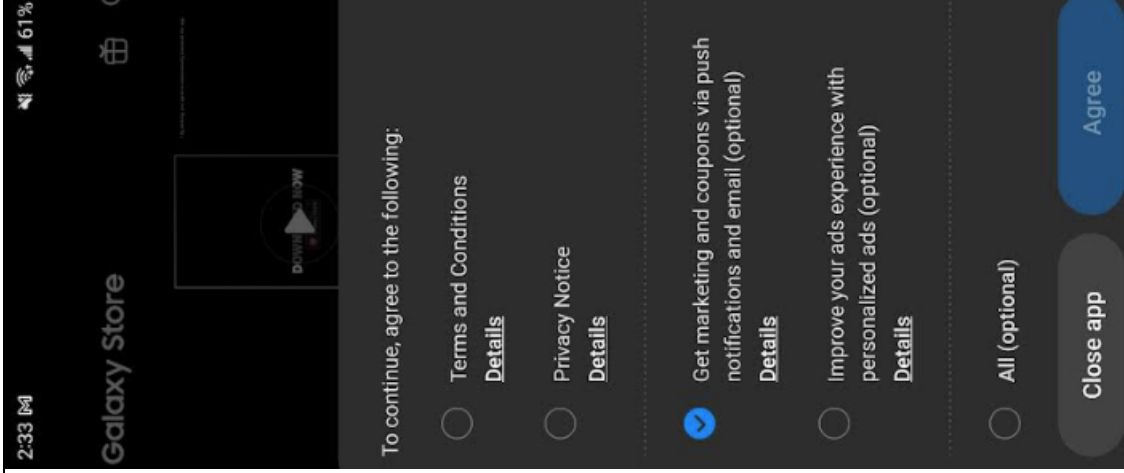
where the message content is information associated with service usage. Notifications received by Samsung's Tizen-based devices from Tizen push servers may comprise information the user's particular use of network data service through a given application. As another example, Samsung's Android-based devices receive push messages (related to message content) via functionality provided by Firebase cloud messaging servers, which in turn receives message content comprising information associated with service usage.

For further example, 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's SmartThings application uses push notifications depending on the use case. *See, e.g.,* <https://www.samsung.com/us/support/troubleshooting/TSG01110148/>. 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.:*



5. The end-user device recited in claim 1, wherein the message content is based, at least in part, on a user preference.	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise the "end-user device recited in claim 1, wherein the message content is based, at least in part, on a user preference." <i>See supra</i> claim 1. For end-user devices being used within the Samsung Knox ecosystem (including devices with administrator privileges within a particular Knox MDM), for example mobile phones and other devices enrolled in the Samsung Knox MDM platform, message content received by such devices may comprise information based at least in part on user preference, where the "user" may be the administrator having the end user device with MDM administrative privileges or the end-user of a device being managed in the Knox MDM. The user preference includes be the options and preferences selected by the MDM group administrator using the ACTIONS > Manage policy menu (https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-devices/send-notifications/; https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-policies/common/enrollment-notices/; https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-devices/lock-and-unlock-devices-and-update-lock-message/; https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-devices/start-and-stop-blinking-reminder/) , which affects the content and format of messages received by managed devices in the MDM group. User preferences at the managed device level may also affect the content of messages, as well. As another example, Samsung's Tizen based devices receive message content where the message content is information based at least in part on a user preference. As another example, Samsung's Android-based devices receive push messages (message content) via functionality provided by Firebase cloud messaging servers, which messages are based at least in part on a user preference. The end user device may have particular settings and options (selection of which is a user preference) for enabling receipt of push messages for certain applications.
---	--

Claim 7

7. The end-user device recited in claim 1, wherein the message content comprises a	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise the "end-user device recited in claim 1, wherein the message content comprises a service offer, an advertisement, or a transaction offer."
---	--

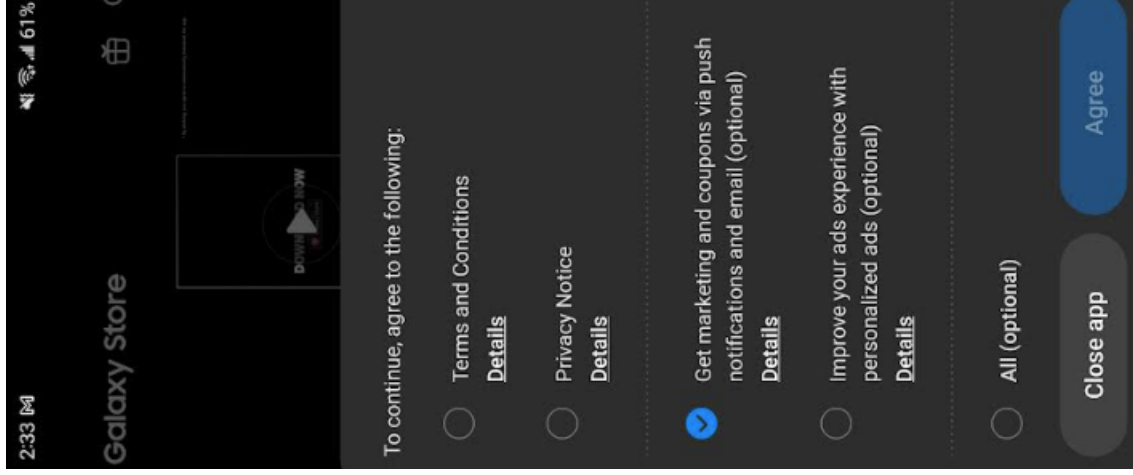
service offer, an advertisement, or a transaction offer.

See supra claim 1.

For example, for end-user devices being used within the Samsung Knox ecosystem (including devices with administrator privileges within a particular Knox MDM), for example mobile phones and other devices enrolled in the Samsung Knox MDM platform, message content received by end user devices may be enrollment requests to user devices (which comprise both service offers and transaction offers) (<https://docs.samsungknox.com/admin/knox-guard/how-to-guides/samsung-knox-deployment-app/>). As another example, Samsung's Tizen-based devices are able to receive advertisements from advertising/app servers communicating with Samsung's Tizen push servers (<https://developer.samsung.com/smarttv/develop/api-references/tizenfx-tv-api-references/tizen.tv.service.adinfo/adinfo-class.html>). As another example, Samsung's Android-based devices receive push messages (message content) via functionality provided by Firebase cloud messaging servers, which messages are comprise advertisements, service offers, and transaction offers depending on the use case and particular application utilizing push messaging through the Firebase framework (<https://firebase.google.com/docs/in-app-messaging> (“Firebase In-App Messaging helps you engage your app's active users by sending them targeted, contextual messages that encourage them to use key app features. For example, you could send an in-app message to get users to subscribe, watch a video, complete a level, or buy an item. You can customize messages as cards, banners, modals, or images, and set up triggers so that they appear exactly when they'd benefit your users most. Firebase In-App Messaging helps you engage your app's active users by sending them targeted, contextual messages that encourage them to use key app features. For example, you could send an in-app message to get users to subscribe, watch a video, complete a level, or buy an item. You can customize messages as cards, banners, modals, or images, and set up triggers so that they appear exactly when they'd benefit your users most.”)).

For further example, 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's SmartThings application sends push notifications containing content based on user preference. *See, e.g.*, <https://www.samsung.com/us/support/troubleshooting/TSG01110148/>. 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 based on user preference. *See, e.g.:*



Claim 8

8. The end-user device recited in claim 1, wherein the message content comprises information from a third party configured to provide control of a service or a billing for a service.	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise the "end-user device recited in claim 1, wherein the message content comprises information from a third party configured to provide control of a service or a billing for a service." <i>See supra</i> claim 1. For example, for end-user devices being used within the Samsung Knox ecosystem (including devices with administrator privileges within a particular Knox MDM), for example mobile phones and other devices enrolled in the Samsung Knox MDM platform, message content received by the end user device may include applications licensed from third-party platforms which are then installed and managed on devices supporting Samsung Knox. For example, Samsung Knox supports the Apple Volume Purchase program which permits devices to receive message content configured to provide control regarding applications that may be installed from a list of available applications. <i>See, e.g.,</i> https://docs.samsungknox.com/admin/knox-manage/configure/applications/volume-purchase-program-for-ios/use-the-volume-purchase-program-ios-only/. As yet another example, Samsung Knox permits devices to receive message content configured to provide control regarding applications that may be installed from a list of available applications that are part of the Google Play Store and/or the Microsoft Store. <i>See, e.g.,</i> https://docs.samsungknox.com/admin/knox-manage/configure/applications/add-apps-using-managed-google-play/; https://docs.samsungknox.com/admin/knox-manage/configure/applications/add-applications/add-public-apps-using-the-microsoft-store/.
---	--

Claim 9

9. The end-user device recited in claim 1, wherein the message content comprises an agent instruction, a setting value, an agent configuration, or a software update.	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise the "end-user device recited in claim 1, wherein the message content comprises an agent instruction, a setting value, an agent configuration, or a software update." <i>See supra</i> claim 1.
--	---

	For example, for end-user devices being used within the Samsung Knox ecosystem (including devices with administrator privileges within a particular Knox MDM), for example mobile phones and other devices enrolled in the Samsung Knox MDM platform, receive messages that are instructions to the Knox Manage or Knox Portal applications running on the devices (“agent instructions”) (https://docs.samsungknox.com/admin/knox-manage/configure/devices/send-commands-to-devices/send-commands-to-devices/), which may cause various options and parameters of the Knox software agents to be set or modified (“setting value” or “agent configuration”) (https://docs.samsungknox.com/admin/knox-manage/configure/devices/send-commands-to-devices/send-commands-to-devices/), or software (including both Knox agents and other non-Knox applications) to be updated (“software update”) (https://docs.samsungknox.com/admin/knox-manage/configure/applications/manage-applications/update-an-app/ (updating applications at end user devices managed by MDM)). As another example, Samsung’s Tizen-based devices are able to receive message content from servers that instruct the Tizen operating system’s push service to display a notification to the user (which is at least an “agent instruction”). As another example, Samsung’s Android-based devices receive push messages (message content) via functionality provided by Firebase cloud messaging servers, which messages comprise instructions to the Firebase SDK on the user device to display notifications (i.e., an “agent instruction”). <i>See, e.g.,</i> Message types With FCM, you can send two types of messages to clients: • Notification messages, sometimes thought of as “display messages.” These are handled by the FCM SDK automatically. • Data messages, which are handled by the client app. Notification messages contain a predefined set of user-visible keys. 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://firebase.google.com/docs/cloud-messaging/concept-options
--	---

10. The end-user device recited in claim 1, wherein the message content comprises software or a media file.	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise the "end-user device recited in claim 1, wherein the message content comprises software or a media file." <i>See supra</i> claim 1. For example, for end-user devices being used within the Samsung Knox ecosystem (including devices with administrator privileges within a particular Knox MDM), for example mobile phones and other devices enrolled in the Samsung Knox MDM platform, may receive message content comprising software. <i>See, e.g.,</i> https://docs.samsungknox.com/admin/knox-manage/configure/applications/add-applications/add-internal-android-and-ios-apps/ (administrators of Knox MDM group able to cause software to be delivered to managed devices) and https://docs.samsungknox.com/admin/knox-manage/configure/content/managed-content/distribute-content/ (media content distributable by Knox Manage)).
--	--

Claim 11

11. The end-user device recited in claim 1, wherein the message content comprises information associated with a service policy.	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise the "end-user device recited in claim 1, wherein the message content comprises information associated with a service policy." <i>See supra</i> claim 2 & 3. The information received by Samsung's Galaxy and Tizen devices may comprise information associated with a service policy. For end-user devices being used within the Samsung Knox ecosystem (including devices with administrator privileges within a particular Knox MDM), for example mobile phones and other devices enrolled in the Samsung Knox MDM platform, message content received by such devices may comprise information associated with a service policy. <i>See, e.g.,</i> https://docs.samsungknox.com/admin/knox-manage/monitor/service-history/; https://docs.samsungknox.com/admin/knox-manage/kbas/kba-360046891233/; https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-devices/send-notifications/; https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-devices/start-and-stop-blinking-reminder/. As another example, Samsung's Tizen based devices receive message content from a "particular server" where the message content is information associated with a service policy. Notifications received by Samsung's Tizen-based devices from Tizen push servers may comprise information the user's particular use of network data service
--	---

	through a given application. As another example, Samsung's Android-based devices receive push messages (message content) via functionality provided by Firebase cloud messaging servers, which in turn receives message content comprising information associated with service policies.
--	--

Claim 12

12. The end-user device recited in claim 1, wherein the message content comprises service usage accounting information.	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise the "end-user device recited in claim 1, wherein the message content comprises service usage accounting information." See supra claim 2 & 3. The information received by Samsung's Galaxy and Tizen devices may comprise information associated with a service policy. For end-user devices being used within the Samsung Knox ecosystem (including devices with administrator privileges within a particular Knox MDM), for example mobile phones and other devices enrolled in the Samsung Knox MDM platform, message content received by such devices may comprise information associated with a service usage. See, e.g., https://docs.samsungknox.com/admin/knox-manage/monitor/service-history/; https://docs.samsungknox.com/admin/knox-manage/kbas/kba-360046891233/; https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-devices/send-notifications/; https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-devices/start-and-stop-blinking-reminder/. As another example, Samsung's Tizen based devices receive message content from a "particular server" where the message content is information associated with a service usage. Notifications received by Samsung's Tizen-based devices from Tizen push servers may comprise information the user's particular use of network data service through a given application. As another example, Samsung's Android-based devices receive push messages (message content) via functionality provided by Firebase cloud messaging servers, which in turn receives message content comprising information associated with service usage.
---	---

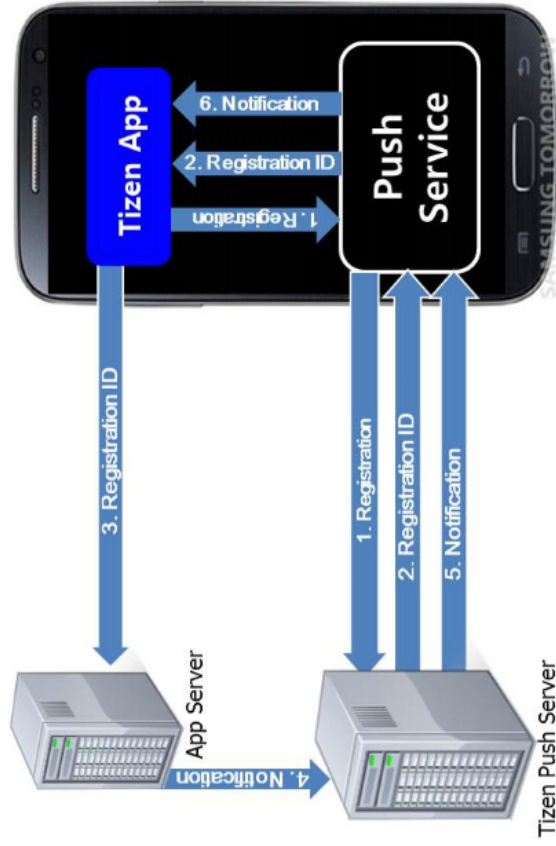
Claim 13

13. The end-user device recited in claim 1, wherein the service control device link agent is further configured to send a device message to the service control server link element over the service control link.	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise the "end-user device recited in claim 1, wherein the service control device link agent is further configured to send a device message to the service control server link element over the service control link." <i>See supra</i> claims 1 and 3. For example, with the device verification and authentication steps taken by the accused instrumentalities with respect to Tizen devices and Android devices receiving message content through the FCM framework, the service control device link agents on the end user devices comprise the push service software running on the Tizen and Android devices (e.g., the Tizen Push Service and the Firebase SDK), which is configured to send device messages (e.g., the Tizen Push Service sends registration information to the Tizen Push Server; the Android device sends authentication tokens and instance IDs to the server; <i>see below</i>) over a service control link.
---	---

Architecture

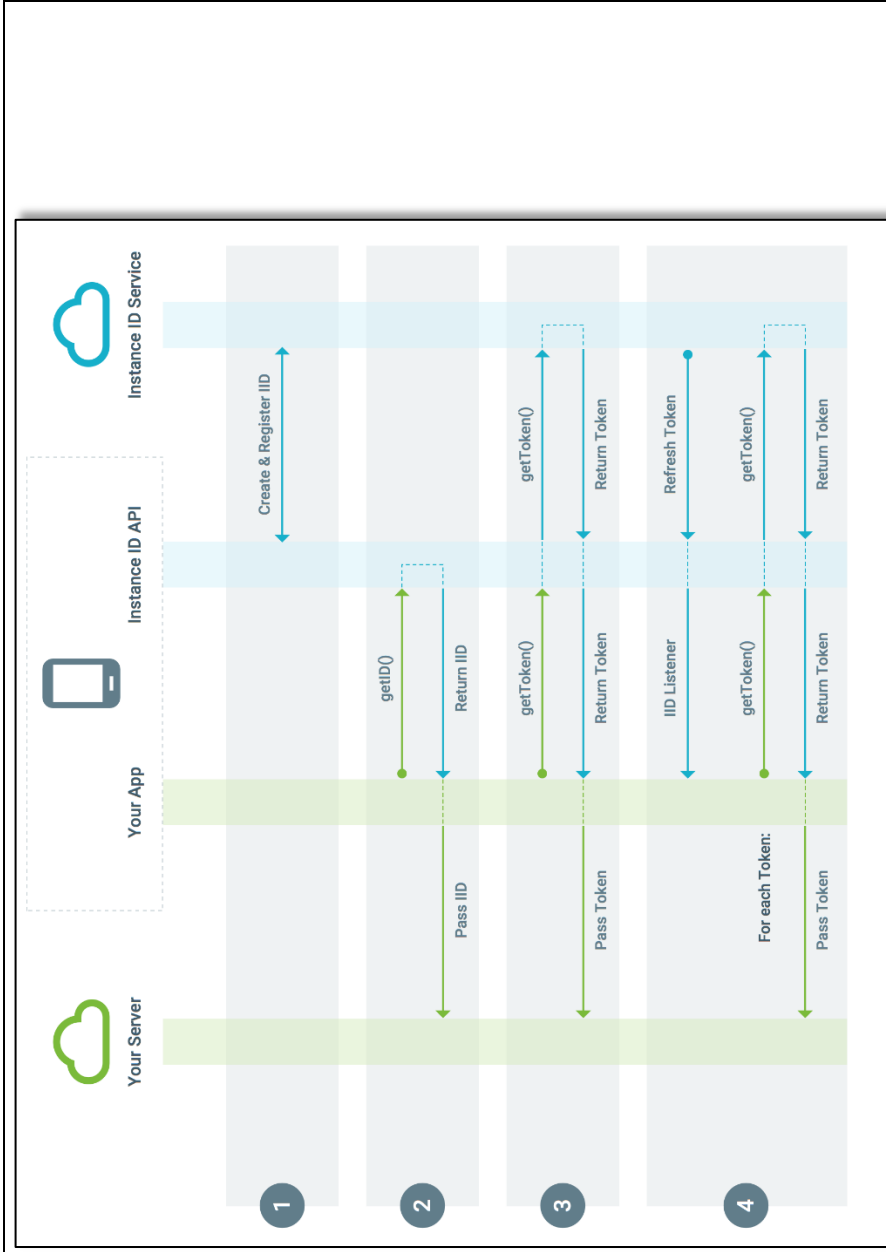
The architecture of the Tizen Push service is described in detail in the [mobile native Push guide](#).

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.



As another example, managed devices in a Samsung Knox enroll with the Knox service by communicating with Knox servers (service control server link elements) over the service control link by providing to Knox servers, for example, information like the device's IMEI number (a mobile device identification number), model number, and so on, which information may then be presented to a Knox administrator. See, e.g.:

	Accept devices Accepting multiple devices in a single operation can only be accomplished when the devices are from a single reseller. 1. Select one or more device checkboxes from the left-hand side of the Devices screen. 2. Select the ACTIONS drop-down menu from the Devices page and select Manage device > Accept Devices. Accept device(s) If you accept, Samsung Test PROD Reseller (Reseller ID: 3965091263) will be added to your registered reseller list. Selected devices will be added to your device inventory with the 'Activating' status. <table> <tr> <th>IMEI/SN</th><th>Device model</th></tr> <tr> <td>357190100093010</td><td>Engineering Model</td></tr> </table> Approval ID (only for this device) Enter Approval ID, Loan ID, or Loan number CANCEL ACCEPT https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-devices/accept-or-reject-devices/	IMEI/SN	Device model	357190100093010	Engineering Model
IMEI/SN	Device model				
357190100093010	Engineering Model				

Claim 14

14. The end-user device recited in claim 13, wherein the device message comprises a service usage report or an integrity report.	Samsung Galaxy phones and tablets and Samsung’s Tizen based devices comprise the “end-user device recited in claim 13, wherein the device message comprises a service usage report or an integrity report.” See <i>supra</i> claim 1. See, for example:
---	---

Device logs

Last updated July 26th, 2023

View device logs for monitoring device operation. Device logs include the inventory and log messages of applied policies and device commands. The device with policy applied collects the log based on the policy. For more information about the log policy, see [Logging](#).

The default settings for device logs are as follows:

- Maximum collection capacity: 10 MB
- Maximum storage period: 7 days
- Log level: Records about debugging and other logs for developers

To view device logs, complete the following steps:

1. Navigate to **History** > **Device Log**.
2. Search for a device by using the search field and/or defining the collecting period.
3. Click the row of the device.
4. In the **Log Files** area, view the log files.
 - You can filter the collection period.
 - The name format of log files is **EMMAgent_yyyymmdd**.

<https://docs.samsungknox.com/admin/knox-manage/monitor/device-logs/>

App usage

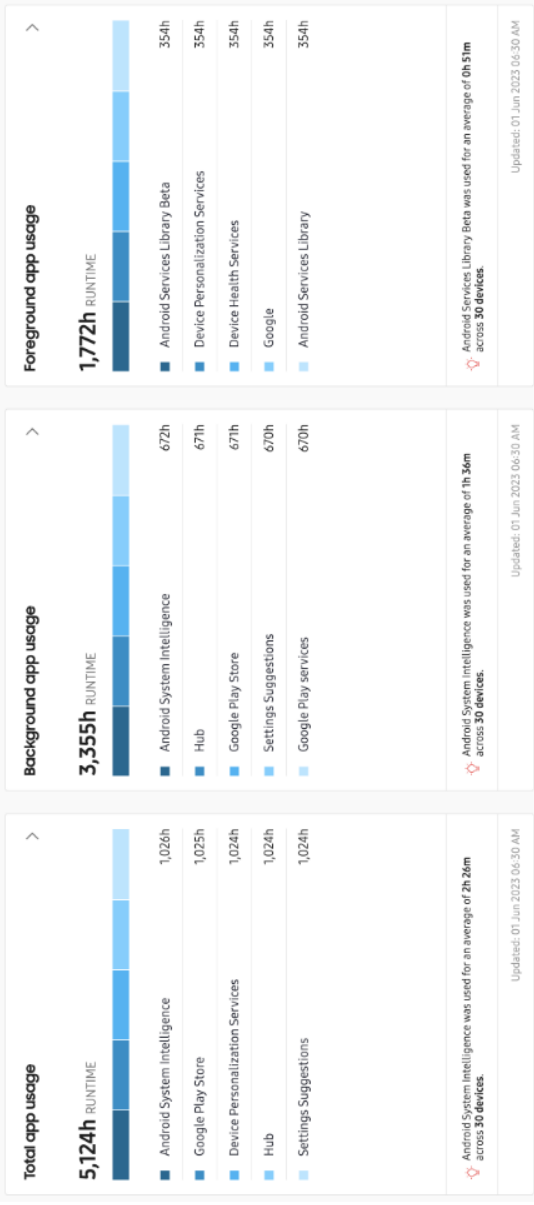
Last updated July 26th, 2023

To better understand how your devices are used on the frontline, you can monitor which apps workers are using, how long they're being used for, and how much battery power and mobile data (Wi-Fi and cellular) are being consumed. Use the cascading charts to view aggregated data among all apps, or narrow it down to a single app. Only work profile app usage is tracked on personally-enabled devices.

Module views

For a description of dashboard tiles, refer to [Dashboard tiles explained](#).

Dashboard tile



<https://docs.samsungknox.com/admin/knox-asset-intelligence/dashboard/app-insights/app-usage/>

19. The end-user device recited in claim 1, further comprising a user interface, and wherein the particular device agent is configured to assist in presenting a notification through the user interface, the notification based on the message content.	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise the "end-user device recited in claim 1, further comprising a user interface, and wherein the particular device agent is configured to assist in presenting a notification through the user interface, the notification based on the message content." <i>See supra</i> claim 1. Message content sent to the end user devices in the accused instrumentalities may be shown in visual form to a user of the end user device in the form of a notification, as explained in the discussion above, including the discussion for claims 1 and 3.
<u>Claim 21</u>	
21. The end-user device recited in claim 1, wherein the service control link supports asynchronous transmissions by the service control server link element.	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise the "end-user device recited in claim 1, wherein the service control link supports asynchronous transmissions by the service control server link element." <i>See supra</i> claim 1. As explained for 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 22

88

22. The end-user device recited in claim 1, wherein the service control link supports periodic transmissions by the service control server link element.	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise the "end-user device recited in claim 1, wherein the service control link supports periodic transmissions by the service control server link element." <i>See supra</i> claim 1. As explained for 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, such as periodically. <i>See supra</i> claim 21.
--	--

Claim 23

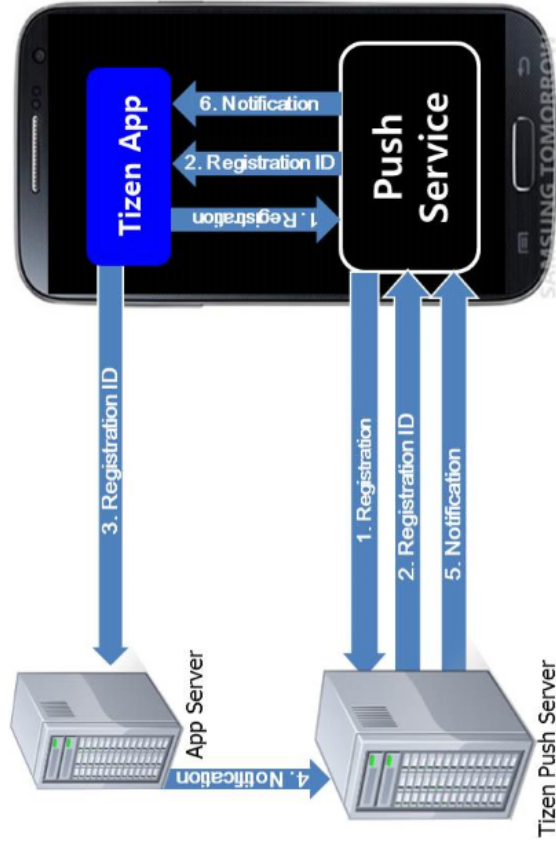
23. The end-user device recited in claim 1, wherein the service control device link agent is further configured to send a	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise the "end-user device recited in claim 1, wherein the service control device link agent is further configured to
---	---

device credential to the network system or receive the device credential from the network system during a service authorization sequence.	send a device credential to the network system or receive the device credential from the network system during a service authorization sequence.” <i>See supra</i> claims 1, 3, and 13. For example, with the device verification and authentication steps taken by the accused instrumentalities with respect to Tizen devices and Android devices receiving message content through the FCM framework, the service control device link agents on the end user devices comprise the push service software running on the Tizen and Android devices (e.g., the Tizen Push Service and the Firebase SDK), which is configured to send device credentials (e.g., the Tizen Push Service sends registration information to the Tizen Push Server; the Android device sends authentication tokens and instance IDs to the server; <i>see below</i>).
--	---

Architecture

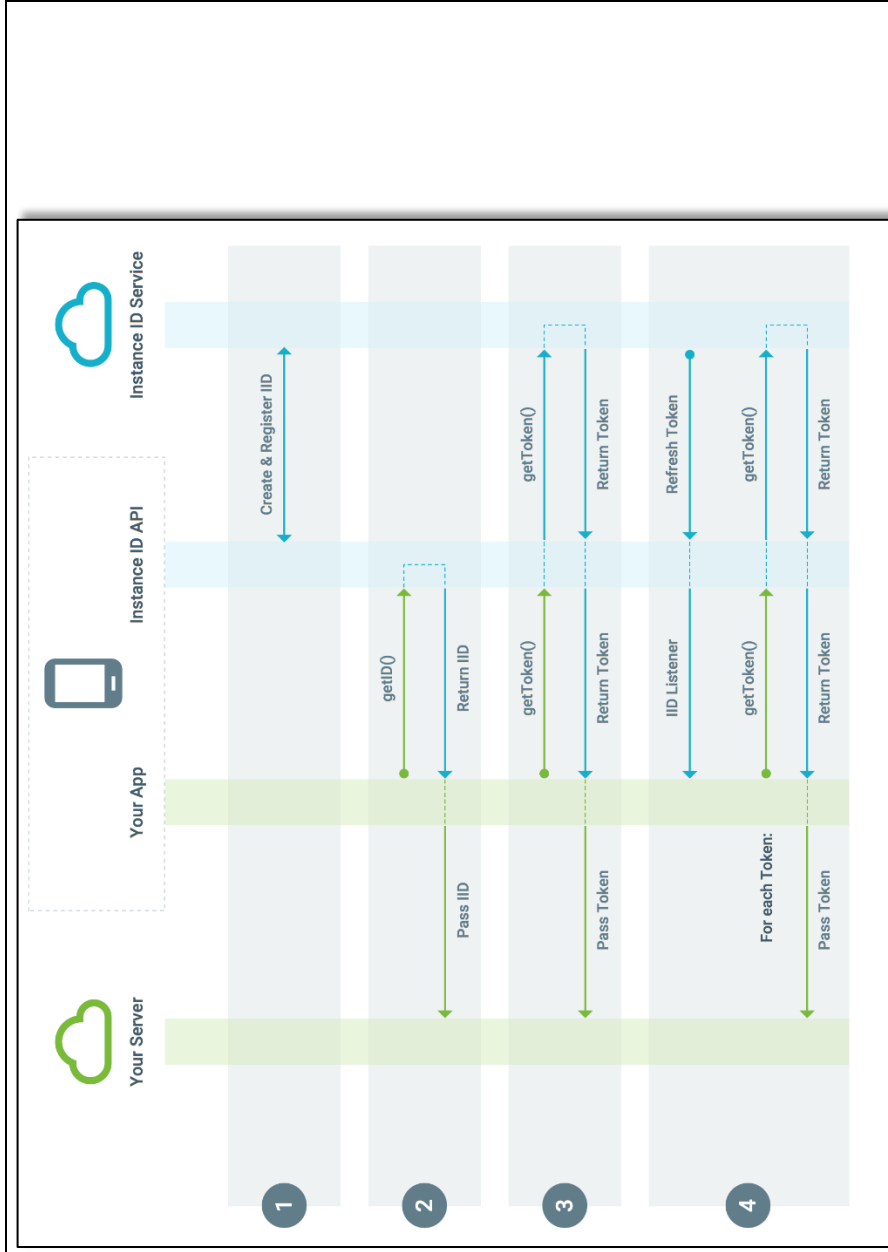
The architecture of the Tizen Push service is described in detail in the [mobile native Push guide](#).

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.



As another example, managed devices in a Samsung Knox enroll with the Knox service by communicating with Knox servers (service control server link elements) over the service control link by providing to Knox servers, for example, information like the device's IMEI number (a mobile device identification number), model number, and so on, which information may then be presented to a Knox administrator. See, e.g.:

	Accept devices Accepting multiple devices in a single operation can only be accomplished when the devices are from a single reseller. 1. Select one or more device checkboxes from the left-hand side of the Devices screen. 2. Select the ACTIONS drop-down menu from the Devices page and select Manage device > Accept Devices. × Accept device(s) If you accept, Samsung Test PROD Reseller (Reseller ID: 3965091263) will be added to your registered reseller list. Selected devices will be added to your device inventory with the 'Activating' status. <table> <tr> <td>IMEI/SN</td><td>Device model</td></tr> <tr> <td>357190100093010</td><td>Engineering Model</td></tr> </table> Approval ID (only for this device) Enter Approval ID, Loan ID, or Loan number CANCEL ACCEPT https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-devices/accept-or-reject-devices/ <i>See also:</i> https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-devices/device-management/; https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-devices/upload-devices/; https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-devices/accept-or-reject-devices/	IMEI/SN	Device model	357190100093010	Engineering Model
IMEI/SN	Device model				
357190100093010	Engineering Model				

Claim 24

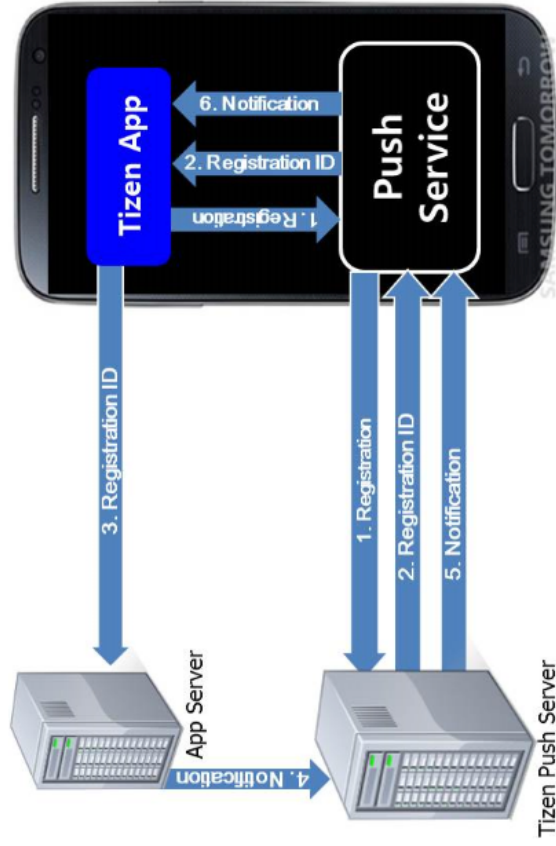
24. The end-user device recited in claim 23, wherein the device credential	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise the “end-user device recited in claim 23, wherein the device credential comprises one or more of a phone
--	---

comprises one or more of a phone number, an identification number, a security signature, a security credential, a subscriber identity module (SIM) identifier, a mobile equipment identifier (MEID), and a device identifier.	number, an identification number, a security signature, a security credential, a subscriber identity module (SIM) identifier, a mobile equipment identifier (MEID), and a device identifier.” <i>See supra</i> claims 1, 3, and 13. For example, with the device verification and authentication steps taken by the accused instrumentalities with respect to Tizen devices and Android devices receiving message content through the FCM framework, the service control device link agents on the end user devices comprise the push service software running on the Tizen and Android devices (e.g., the Tizen Push Service and the Firebase SDK), which is configured to send device credentials (e.g., the Tizen Push Service sends registration information to the Tizen Push Server; the Android device sends authentication tokens and instance IDs to the server; <i>see below</i>).
--	---

Architecture

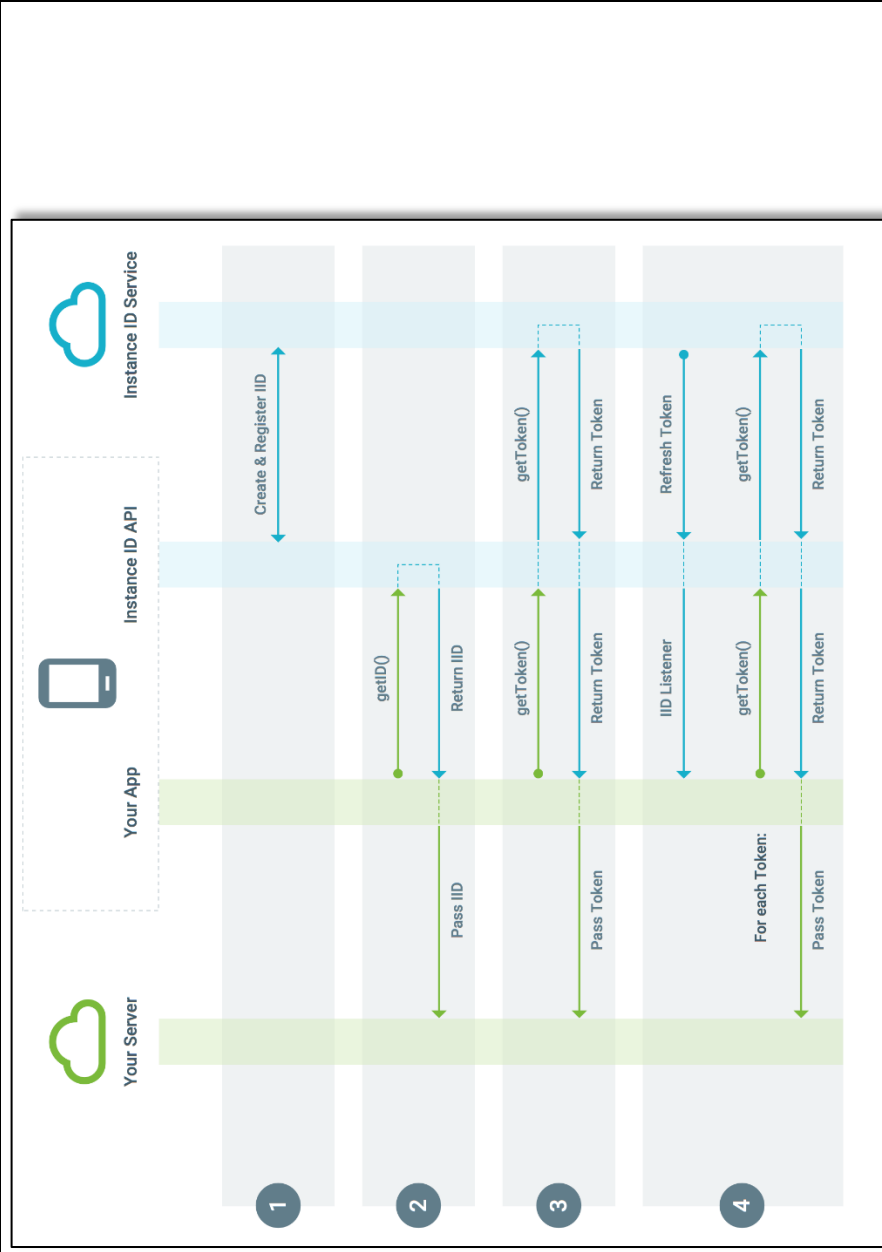
The architecture of the Tizen Push service is described in detail in the [mobile native Push guide](#).

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.



As another example, managed devices in a Samsung Knox enroll with the Knox service by communicating with Knox servers (service control server link elements) over the service control link by providing to Knox servers, for example, information like the device's IMEI number (a mobile device identification number), model number, and so on, which information may then be presented to a Knox administrator. See, e.g.:

Accept devices

Accepting multiple devices in a single operation can only be accomplished when the devices are from a single reseller.

1. Select one or more device checkboxes from the left-hand side of the Devices screen.
2. Select the **ACTIONS** drop-down menu from the Devices page and select **Manage device** > **Accept Devices**.

×

Accept device(s)

If you accept, Samsung Test PROD Reseller (Reseller ID: 3965091263) will be added to your registered reseller list. Selected devices will be added to your device inventory with the 'Activating' status.

IMEI/SN	Device model
357190100093010	Engineering Model

Approval ID (only for this device)

CANCEL

ACCEPT

<https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-devices/accept-or-reject-devices/>

See also: <https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-devices/device-management/>; <https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-devices/upload-devices/>; <https://docs.samsungknox.com/admin/knox-guard/how-to-guides/manage-devices/accept-or-reject-devices/>

Claim 26

26. The end-user device recited in claim 1, wherein the particular device agent comprises software.	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise the "end-user device recited in claim 1, wherein the particular device agent comprises software." <i>See supra</i> claims 1 and 3. The device agents in the accused instrumentalities are all software; e.g., Samsung Knox software on the user device is software; the Samsung Tizen Push Service on the device is software; and the FCM SDK on the device is software.
---	---

Claim 29

29. The end-user device recited in claim 1, wherein the service control link is configured to support control-plane communications using an Internet protocol.	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise the "end-user device recited in claim 1, wherein the service control link is configured to support control-plane communications using an Internet protocol." <i>See supra</i> claims 1 and 3. The control-plane communications discussed herein are all made over the Internet and using an Internet protocol.
--	---

Claim 30

30. A method performed by an end-user device, the method comprising: receiving, over a service control link, an encrypted agent message from a network element, the service control link secured by an encryption protocol, the service control link supporting control-plane communications between a service control device link agent on the end-user device and the network element;	Samsung Galaxy phones and tablets and Samsung's Tizen based devices perform the method of claim 30, as set forth below. Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise "receiving, over a service control link, an encrypted agent message from a network element, the service control link secured by an encryption protocol, the service control link supporting control-plane communications between a service control device link agent on the end-user device and the network element." <i>See supra</i> claim 1[a].
using an encryption key shared between the service control device link agent and the network element, obtaining a decrypted agent message, the decrypted	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise "using an encryption key shared between the service control device link agent and the network element, obtaining a decrypted agent message, the decrypted agent message comprising a particular agent identifier and message content for delivery to a particular device agent of a plurality of device

agent message comprising a particular agent identifier and message content for delivery to a particular device agent of a plurality of device agents on the end-user device, each of the plurality of device agents identifiable by an associated device agent identifier and communicatively coupled to the service control device link agent through an agent communication bus, the particular agent identifier identifying the particular device agent, the message content from a particular server of a plurality of servers communicatively coupled to the network element; and	agents on the end-user device, each of the plurality of device agents identifiable by an associated device agent identifier and communicatively coupled to the service control device link agent through an agent communication bus, the particular agent identifier identifying the particular device agent, the message content from a particular server of a plurality of servers communicatively coupled to the network element.” <i>See supra</i> claim 1[b]-1[d].
delivering the message content to the particular device agent over the agent communication bus based on the particular agent identifier.	Samsung Galaxy phones and tablets and Samsung’s Tizen based devices comprise “delivering the message content to the particular device agent over the agent communication bus based on the particular agent identifier.” <i>See supra</i> claim 1[b]-1[f].