UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

Headwater Research LLC, *Plaintiff*, vs. Samsung Electronics Co., Ltd., Samsung Electronics America, Inc., *Defendants*.	CASE NO. 6:23-CV-00103-JRG-RSP **PLAINTIFF HEADWATER RESEARCH LLC'S DISCLOSURE OF ASSERTED CLAIMS AND INFRINGEMENT CONTENTIONS** **JURY DEMANDED**

PLAINTIFF HEADWATER RESEARCH LLC'S DISCLOSURE OF ASSERTED CLAIMS AND INFRINGEMENT CONTENTIONS

Pursuant to P.R. 3-1 and P.R. 3-2, patent owner Headwater Research LLC ("Headwater") hereby provides its disclosure of asserted claims and infringement contentions and its accompanying document production. This disclosure is based on the information available to Headwater as of the date of this disclosure, before Headwater has received any discovery on the design or operation of the defendants' products. Headwater reserves the right to amend this disclosure to the full extent permitted under the court's rules and orders.

P.R. 3-1: DISCLOSURE OF ASSERTED CLAIMS AND INFRINGEMENT CONTENTIONS

P.R. 3-1(A): ASSERTED CLAIMS

Headwater asserts that defendants Samsung Electronics America, Inc. and Samsung Electronics Co., Ltd. (collectively "Samsung") infringe one or more of the following claims, directly, by inducement, by contributory infringement:

SAMSUNG 1021

U.S. Patent No.	*Asserted Claims*
8,406,733	1-5, 6-14, 19, 21-24, 26, 29, 30
9,198,117	1, 2, 4 -6, 9-14, 16-18
9,615,192	1-9, 11-13, 15

Collectively, these three patents are referred to herein as the Asserted Patents, and these claims as the Asserted Claims.

P.R. 3-1(B): ACCUSED INSTRUMENTALITIES OF WHICH HEADWATER IS AWARE

In this section, Headwater provides lists of accused products that Headwater is aware of infringing based upon information presently available to it and its investigation to date. Headwater's infringement claims are not limited to these listed products and specifically extend to all products and apparatuses of Samsung similar to the listed products that include the claimed elements. Unless otherwise stated, Headwater's infringement assertions apply to all variations, versions, editions, and applications of each of the listed products.

U.S. PATENT NO. 8,406,733

Headwater accuses the following Samsung products which implement Samsung Knox, Tizen, and/or push messaging functionality (e.g., communicating with Google's Firebase Cloud Messaging service, Samsung's Push Service, Knox, or Tizen functionality) of infringing each of the Asserted Claims of the '733 patent:

Samsung phones, tablets, and devices made, used, sold, offered for sale, or imported within six years of the filing of the date of the lawsuit comprising Knox functionality, Tizen functionality, and/or push messaging functionality, including but not limited to the following: Galaxy S II Plus, Galaxy Grand, Galaxy Young, Galaxy Xcover 2, Galaxy S4, Galaxy Fame, Galaxy Mega, Galaxy Win, Galaxy Y Plus, Galaxy Core, Galaxy Star, Galaxy Pocket Neo, Galaxy Ace 3, Galaxy S4 Zoom, Galaxy S4 Active, Galaxy S4 Mini, Galaxy Gear, Galaxy Note 3, Galaxy Core Plus, Galaxy Light, Galaxy Fame Lite, Galaxy Trend Lite, Galaxy Round, Galaxy Express 2, Galaxy J, Galaxy Star Pro, Galaxy Grand 2, Galaxy Trend Plus, Galaxy S Duos 2, Galaxy J, Galaxy Win Pro (SM-

G3812), Galaxy Grand Lite, Galaxy Grand Neo, Galaxy Note 3 Neo, Galaxy Win 2, Galaxy S3 Neo, Galaxy S5, Galaxy Ace Style, Galaxy K Zoom, Galaxy Core, Galaxy Tab S 10.5, Galaxy S5 Mini, Galaxy Core 2, Galaxy S Duos 3, Galaxy Ace 4, Galaxy Star 2 Plus, Galaxy Mega 2, Galaxy Pocket 2, Galaxy Core Prime, Galaxy Grand Prime, Galaxy Alpha, Galaxy Young 2, Galaxy Note 4, Galaxy Note Edge, Galaxy A3 (2015), Galaxy A5 (2015) (released 2014), Galaxy Pocket Plus, Galaxy E7, Galaxy A7 (2015), Galaxy E5, Galaxy J1, Galaxy Xcover 3, Galaxy S6, Galaxy S6 Edge, Galaxy J7, Galaxy J5, Galaxy S6 Active, Galaxy V Plus, Galaxy Trend 2 Lite, Galaxy S5 Neo, Galaxy Note 5, Galaxy A8 (2015), Galaxy S6 Edge+, Galaxy J2, Galaxy J1 Ace, Galaxy A3 (2016), Galaxy A5 (2016), Galaxy A7 (2016), Galaxy A9 (2016), Galaxy J1 (2016), Galaxy S7, Galaxy S7 Edge, Galaxy J1 Mini, Galaxy A9 Pro (2016), Galaxy Tab A6, Galaxy J3 (2016), Galaxy J7 (2016), Galaxy J5 (2016), Galaxy C7, Galaxy C5, Galaxy Tab A 10.1 (2016), Galaxy J3 Pro, Galaxy S7 Active, Galaxy J2 Prime, Galaxy Note 7, Galaxy J7 Prime, Galaxy On7 (2016), Galaxy On5 Pro (2016), Galaxy On5 (2016), Galaxy A8 (2016), Galaxy C9 Pro, Galaxy J2 Prime, Galaxy J1 mini Prime/Galaxy V2, Galaxy C5 Pro, Galaxy A3 (2017), Galaxy A5 (2017), Galaxy A7 (2017), Galaxy Tab S3, Galaxy S8+, Galaxy S8, Galaxy Xcover 4, Galaxy J7 Max, Galaxy J7 Pro (2017), Galaxy J7 (2017), Galaxy J5 (2017), Galaxy J3 Prime, Galaxy J3 (2017), Galaxy Note Fan Edition (FE), Galaxy S8 Active, Galaxy Note 8, Galaxy J7+, Galaxy J7 Sky Pro, Galaxy J7 Nxt, Galaxy J7 V, Galaxy C8 / C7 (2017), Galaxy Tab A 8.0 (2017), Galaxy Tab Active 2, Galaxy J4 (2017), Galaxy A8+ (2018), Galaxy A8 (2018), Galaxy J4 Pro (2018), Galaxy S9+, Galaxy S9, Galaxy J7 Duo, Galaxy J6 (2018), Galaxy J4 (2018), Galaxy A6+ (2018), Galaxy A6 (2018), Galaxy J7 (2018), Galaxy On6, Galaxy J8 (2018), Galaxy Tab S4, Galaxy Tab A 10.5 (2018), Galaxy Note 9, Galaxy J6+, Galaxy J4+, Galaxy A7 (2018), Galaxy A9 (2018), Galaxy A6s, Galaxy A8+ (2018), Galaxy A8 (2018), Galaxy A8s, Galaxy M20, Galaxy M10, Galaxy M30, Galaxy Tab A 10.1 (2019), Galaxy Tab S5e, Galaxy A10, Galaxy A30, Galaxy A50, Galaxy S10e, Galaxy S10, Galaxy S10+, Galaxy A2 Core, Galaxy A20e, Galaxy A20, Galaxy A40, Galaxy A70, Galaxy A80, Galaxy A60, Galaxy View 2, Galaxy S10 5G, Galaxy M40, Galaxy Note 10+, Galaxy Note 10, Galaxy A10e, Galaxy A10s, Galaxy Fold, Galaxy A90 5G, Galaxy A30s, Galaxy A50s, Galaxy A70s, Galaxy M10s, Galaxy A20s, Galaxy M30s, Galaxy A51, Galaxy A71, Galaxy A01, Galaxy S10 Lite, Galaxy Note 10 Lite, Galaxy Xcover Pro, Galaxy Tab S6, Galaxy Tab S6 5G, Galaxy Z Flip, Galaxy M31, Galaxy A11, Galaxy M21, Galaxy A41, Galaxy A31, Galaxy M11, Galaxy S20 Ultra, Galaxy S20+, Galaxy S20, Galaxy Tab S6 Lite, Galaxy A21, Galaxy A51 5G, Galaxy A51 5G UW, Galaxy A71 5G, Galaxy A21s, Galaxy M01, Galaxy M01s, Galaxy A01 Core, Galaxy M31s, Galaxy Tab S7+, Galaxy Tab S7, Galaxy Z Flip 5G, Galaxy Note 20 Ultra, Galaxy Note 20 Ultra 5G, Galaxy Note 20, Galaxy M51, Galaxy Z Fold2 5G, Galaxy S20 FE, Galaxy S20 FE 5G, Galaxy S20 FE 2022, Galaxy F41, Galaxy A42 5G, Galaxy M21s, Galaxy A12, Galaxy A02s, Galaxy M02s, Galaxy A32 5G, Galaxy S21 Ultra, Galaxy S21+, Galaxy S21, Galaxy A02, Galaxy M02, Galaxy M12, Galaxy F62, Galaxy M62, Galaxy A32, Galaxy A52, Galaxy A52 5G, Galaxy A72, Galaxy F12, Galaxy F02s, Galaxy M42 5G, Galaxy F52 5G, Galaxy Tab A7 Lite, Galaxy Tab S7 FE, Galaxy A22, Galaxy A22 5G, Galaxy M32, Galaxy F22, Galaxy Z Flip3 5G, Galaxy Z Fold3 5G, Galaxy A52s 5G, Galaxy A03s, Galaxy M32 5G, Galaxy M22, Galaxy M52 5G, Galaxy F42 5G, Galaxy A03 Core, Galaxy A03, Galaxy Tab A8 10.5, Galaxy A13 5G, Galaxy S21 FE, Galaxy S22, Galaxy S22+, Galaxy S22 Ultra, Galaxy A13, Galaxy A23 5G, Galaxy A23, Galaxy F23, Galaxy A53 5G, Galaxy Tab S8, Galaxy Tab S8+, Galaxy Tab S8 Ultra, Galaxy A73 5G, Galaxy A33 5G, Galaxy M33 5G, Galaxy M23, Galaxy M53 5G, Galaxy F13, Galaxy M13, Galaxy M13 5G, Galaxy Xcover6 Pro, Galaxy Z Flip 4, Galaxy Z Fold 4, Galaxy Tab Active4 Pro, Galaxy A04s, Galaxy A04, Galaxy A04e, Galaxy Tab A7 10.4 (2022), Galaxy M04, Galaxy F04,

Galaxy A14 5G, Galaxy S23 Ultra, Galaxy S23+, Galaxy S23, Galaxy Xcover 5, Galaxy Tab A7, Galaxy Tab A7 10.4 (2020), Galaxy Tab A 8.0 (2019), Galaxy J3 (2018), and Galaxy Tab E.

Samsung televisions made, used, sold, offered for sale, or imported within six years of the filing of the date of the lawsuit comprising Knox functionality, Tizen functionality, and/or push messaging functionality, with chassis numbers including, but not limited to the following: MCWS1, QCQ800, QCQ900, G75NC, G85NC, G95SC, QCQ700, S90PC, LS03C QCQ80, QCQ80D, QCQ80TL, QCQ83, QCQ85, QCQ85D, QCQ85TL, QCQ88, QCQ8X, QCQ90, QCQ90D, QCQ90TL, QCQ95, QCQ95D, QCQ95TL, QCQ9X, QCQS90, QCQS90D, QCQS90TL, QCQS95, QCQS95D, QCQS95TL, QCQS9X, M70C,M80C,QCQ70, QAQ70, QAQ7A, QAQ7D, QCQ70D_NM2, QCQ70_NM2, QCQ7X_NM2, QTQ70, M50C,QCQ60, LSP3C,UCU7000, UCU8000TL_PML,UCU8000_PML,UCU800M_PML,UCU80MD_PML, UCU8000, QBQ900, QBQ850, QBQ800, QBQ100, MBWS1, G97NB, QBQ700, G85NB, G75NB, G95NB, LS04B, LS03B, QBQ75, QBQS95, QBQS90,QBQD90, QBQ9X, QBQ9D, QBQ95, QBQ90S, QBQ90, QBQ8D, QBQ85D, QBQ85, QBQ83, QBQ80, QBQ7X, QBQ70, QBQX3, LS01B, LS05B, G65B, G70B, G8QNB, M80B, G65B, G70B, G70NC, G85SB, LS01B_ML, LS05B_ML, QBQ70_ML, QBQ7X_ML, BQ80_ML, QBQ83_ML, QBQ85D_ML, QBQ85_ML, QBQ8D_ML, QBQ90_ML, QBQ9D_ML, QBQ9X_ML, QAQ70, QAQ7D, QAQ90, QBQ6D, QBQ63, QBQ60_DPP, QBQ60, M70BO, M70B, M50B, UAU8000_L, QAQ60_K, UAU850D_K65, UAU8000_K65, UAU8000_K, LS03BS, UBU7000, UBU8000, LSP3B, UBU8000_X, QAQ9X, QAQ9D, QAQ95, QAQ90_NS, QAQ90, QAQ8D, QAQ85_NS, QAQ85D, QAQ85, QAQ80_NS, QAQ80, QAQ7D, QAQ70_NS, QAQ70, LST5A, QAQ7A, LS03A_NS, LS03AP, LS03A, LS03APD, LS03AD, QAQ900, QAQ850, QAQ800, QAQ700_P, MAWS1, QAQ700, QAQ700_S, UAU9070, UAU9000, QAQ6D, QAQ60, UAU850D, UAU8000, LS05AL, QAQ6D_K, QAQ60_K, QAQ50_K, UAUE60A, UAU7500, UAU7000, UAUE70A, UAU850D_K, UAU8000_K, UAU7700, LSP3, BEA_HB, KTSU2EL_ATM, KTSU2EL_HKC, KTSU2EL_HKC, KTSU2EL_REF, KTSU2EL_TPO, KTSU2EL_TPO, QTQ800, QTQ800_NS, QTQ850, QTQ900, QTQ950, QTQ950_NS, LSR9, LST7T, QTQ70, QTQ70_NS, QTQ72, QTQ7D, QTQ7X, QTQ80, QTQ80_NS, QTQ8D, QTQ8E, QTQ90, QTQ90_NS, QTQ95, QTQ95_NS, LS05T, LS03T, LS03T_NS, LS01T, LS01T_NS, MTWS1, QTQ60, QTQ62, QTQ6D, QTQ6E, QTQ6X, UTU8500, UTU850D, UTU8510, UTU8570, UTU85A0, QTQ6AC, QTQ6SC, QTQ6XC, UTU8000, UTU800D, UTU8200, UTU8300, UTU8400, LS03TS, UTU7000, UTU700D, UTU7090, UT4310, UT4500, UT4700, UT5310, UT5500, UT5770, UT4300, UT4400, UT5300, UT430E, UT530E, QRQ900B, QRQ90, QRQ85_OC, QRQ80_OC, QRQ80, QRQ75S, QRQ75, QRQ70, QRQ6X, QRQ69, QRQ65A, QRQ60, LS03R, LS05R, LS01R, URU800D, URU8000, MLS07R, QRQ50S, QRQ50, URU7800, URU7700, URU74A0, URU7470, URU7450, URU7410, URU740D, URU7400, URU730D, URU7300, URU710D, URU7100,URU7790,URU77FA, URU7000, URU6900, URUF58T, RUF58T, UR6000, UR5500, UR4500, QRQ900,QNQ9S, QNQ9F, QNQ8FB, QNQ8F, QNQ8C, QNQ7FH, QNQ7F, QNQ7C, QNQ75F, QNQ75C, QNQ6FK, QNQ6F, QNQ65FB, QNQ65F,UNU850D, UNU8500, UNU80A0, UNU800D, UNU8000,UNU76A0, UNU7500, UNU74A0, UNU7450, UNU7400,ULS03NU, UNU730D, UNU7300, UNU710D, UNU7103, UNU7100, URU7000, UNU7120, UNU709D, UNU7090, UNU7080, UNU7050, UNU6950, UN5510, UN5500, UN5350, UN5305, UN5300, UN5200, UN4510, UN4500, UN4350, UN4310, UN4300, BERTB, BERTA, QMS9Y, QMQ9FR, QMQ9F, QMQ8CR, QMQ8C, QMQ7FVRE, QMQ7FRE, QMQ7FR, QMQ7FDR, QMQ7FD, QMQ7F, QMQ7CRE, QMQ7CR, QMQ7CDR,

QMQ7CD, QMQ7C, QMQ6FRE, UMQS9, UMU900R, UMU9000, UMU800R, UMU8000, UMU750R, UMU7500, UMU70AR, UMU70A0, UMU700R, UMU7000, UMU66A0, UMU6500, UMU64A0, UMU6470, UMU6400, UMLS003, ULS003U, UMU66AR, UMU650R, UMU64AR, UMU640R, UMU6303, UMU6302, UMU6300, UMU6310,UMU6320,UMU6103, UMU6102, UMU6100, UMU6070, UMU6000,UMUF31E, UMUF30E, UM6303, UM6302, UM6300, UM5603, UM5602, UM5600, UM5523, UM5522, UM5520, UM5513, UM5512, UM5510, UM5503, UM5502, UM5500, UM5300, UM4500, UM4300,TH39S,VF39S, UNU7000, UMU6100, or, alternatively model numbers in the following formats: QN***C, LS0*C, QN**C, S90C, S95C, Q**C, CU****,QN***B, LS0*B, QN**B, S90B, S95B, Q**B, BU****, QN***A, LS0*A, QN**A, Q**A, AU****, QN***T, LS0*T, QN**T, Q**T, TU****, QN***R, LS0*R, QN**R, Q**R, RU****, QN*FN, NU***, LS0*NU, Q7F, Q6F, MU***, U7000, MU6100.

Samsung wearables made, used, sold, offered for sale, or imported within six years of the filing of the date of the lawsuit comprising Knox functionality, Tizen functionality, and/or push messaging functionality including, but not limited to the following: SM-R800 (46mm), SM-R805 (46mm/Cellular), SM-R810 (42mm), SM-R815 (42mm/Cellular), SM-R855F, SM-845F (Global, LTE model) SM-R855U SM-R845U (USA, LTE model) SM-R850, SM-840 (Wi-Fi only), SM-R860 SM-R865 SM-R870 SM-R875 SM-R880 SM-R885 SM-R890 SM-R895, SM-R905 (40mm), SM-R915 (44mm), SM-R925 (45mm), SM-R930 (40mm/Wi-Fi), SM-R935 (40mm/LTE), SM-R940 (44mm/Wi-Fi), SM-R945 (44mm/LTE), SM-R950 (43mm/Wi-Fi), SM-R955 (43mm/LTE), SM-R960 (47mm/Wi-Fi), SM-R965 (47mm/LTE), SM-R500 (40mm), SM-R820 (44mm), SM-R825 (44mm/Cellular), SM-R830 (40mm), SM-R835 (40mm/Cellular).

The Samsung products in the preceding list, including all variations, editions, and applications of the foregoing, and all products and apparatuses of Samsung similar to the foregoing that include the claimed elements are the ’733 Accused Instrumentalities.

U.S. PATENT NO. 9,198,117

Headwater accuses the following Samsung products which implement Samsung Knox, Tizen, and/or push messaging functionality (e.g., communicating with Google’s Firebase Cloud Messaging service, Samsung’s Push Service, Knox, or Tizen functionality) of infringing each of the Asserted Claims of the ’117 patent:

Servers leased, owned, supported, and/or operated by Samsung comprising Knox functionality, Tizen functionality and/or push messaging functionality.

Samsung phones, tablets, and devices made, used, sold, offered for sale, or imported within six years of the filing of the date of the lawsuit comprising Knox functionality, Tizen functionality,

and/or push messaging functionality, including but not limited to the following: Galaxy S II Plus, Galaxy Grand, Galaxy Young, Galaxy Xcover 2, Galaxy S4, Galaxy Fame, Galaxy Mega, Galaxy Win, Galaxy Y Plus, Galaxy Core, Galaxy Star, Galaxy Pocket Neo, Galaxy Ace 3, Galaxy S4 Zoom, Galaxy S4 Active, Galaxy S4 Mini, Galaxy Gear, Galaxy Note 3, Galaxy Core Plus, Galaxy Light, Galaxy Fame Lite, Galaxy Trend Lite, Galaxy Round, Galaxy Express 2, Galaxy J, Galaxy Star Pro, Galaxy Grand 2, Galaxy Trend Plus, Galaxy S Duos 2, Galaxy J, Galaxy Win Pro (SM-G3812), Galaxy Grand Lite, Galaxy Grand Neo, Galaxy Note 3 Neo, Galaxy Win 2, Galaxy S3 Neo, Galaxy S5, Galaxy Ace Style, Galaxy K Zoom, Galaxy Core, Galaxy Tab S 10.5, Galaxy S5 Mini, Galaxy Core 2, Galaxy S Duos 3, Galaxy Ace 4, Galaxy Star 2 Plus, Galaxy Mega 2, Galaxy Pocket 2, Galaxy Core Prime, Galaxy Grand Prime, Galaxy Alpha, Galaxy Young 2, Galaxy Note 4, Galaxy Note Edge, Galaxy A3 (2015), Galaxy A5 (2015) (released 2014), Galaxy Pocket Plus, Galaxy E7, Galaxy A7 (2015), Galaxy E5, Galaxy J1, Galaxy Xcover 3, Galaxy S6, Galaxy S6 Edge, Galaxy J7, Galaxy J5, Galaxy S6 Active, Galaxy V Plus, Galaxy Trend 2 Lite, Galaxy S5 Neo, Galaxy Note 5, Galaxy A8 (2015), Galaxy S6 Edge+, Galaxy J2, Galaxy J1 Ace, Galaxy A3 (2016), Galaxy A5 (2016), Galaxy A7 (2016), Galaxy A9 (2016), Galaxy J1 (2016), Galaxy S7, Galaxy S7 Edge, Galaxy J1 Mini, Galaxy A9 Pro (2016), Galaxy Tab A6, Galaxy J3 (2016), Galaxy J7 (2016), Galaxy J5 (2016), Galaxy C7, Galaxy C5, Galaxy Tab A 10.1 (2016), Galaxy J3 Pro, Galaxy S7 Active, Galaxy J2 Prime, Galaxy Note 7, Galaxy J7 Prime, Galaxy On7 (2016), Galaxy On5 Pro (2016), Galaxy On5 (2016), Galaxy A8 (2016), Galaxy C9 Pro, Galaxy J2 Prime, Galaxy J1 mini Prime/Galaxy V2, Galaxy C5 Pro, Galaxy A3 (2017), Galaxy A5 (2017), Galaxy A7 (2017), Galaxy Tab S3, Galaxy S8+, Galaxy S8, Galaxy Xcover 4, Galaxy J7 Max, Galaxy J7 Pro (2017), Galaxy J7 (2017), Galaxy J5 (2017), Galaxy J3 Prime, Galaxy J3 (2017), Galaxy Note Fan Edition (FE), Galaxy S8 Active, Galaxy Note 8, Galaxy J7+, Galaxy J7 Sky Pro, Galaxy J7 Nxt, Galaxy J7 V, Galaxy C8 / C7 (2017), Galaxy Tab A 8.0 (2017), Galaxy Tab Active 2, Galaxy J4 (2017), Galaxy A8+ (2018), Galaxy A8 (2018), Galaxy J4 Pro (2018), Galaxy S9+, Galaxy S9, Galaxy J7 Duo, Galaxy J6 (2018), Galaxy J4 (2018), Galaxy A6+ (2018), Galaxy A6 (2018), Galaxy J7 (2018), Galaxy On6, Galaxy J8 (2018), Galaxy Tab S4, Galaxy Tab A 10.5 (2018), Galaxy Note 9, Galaxy J6+, Galaxy J4+, Galaxy A7 (2018), Galaxy A9 (2018), Galaxy A6s, Galaxy A8+ (2018), Galaxy A8 (2018), Galaxy A8s, Galaxy M20, Galaxy M10, Galaxy M30, Galaxy Tab A 10.1 (2019), Galaxy Tab S5e, Galaxy A10, Galaxy A30, Galaxy A50, Galaxy S10e, Galaxy S10, Galaxy S10+, Galaxy A2 Core, Galaxy A20e, Galaxy A20, Galaxy A40, Galaxy A70, Galaxy A80, Galaxy A60, Galaxy View 2, Galaxy S10 5G, Galaxy M40, Galaxy Note 10+, Galaxy Note 10, Galaxy A10e, Galaxy A10s, Galaxy Fold, Galaxy A90 5G, Galaxy A30s, Galaxy A50s, Galaxy A70s, Galaxy M10s, Galaxy A20s, Galaxy M30s, Galaxy A51, Galaxy A71, Galaxy A01, Galaxy S10 Lite, Galaxy Note 10 Lite, Galaxy Xcover Pro, Galaxy Tab S6, Galaxy Tab S6 5G, Galaxy Z Flip, Galaxy M31, Galaxy A11, Galaxy M21, Galaxy A41, Galaxy A31, Galaxy M11, Galaxy S20 Ultra, Galaxy S20+, Galaxy S20, Galaxy Tab S6 Lite, Galaxy A21, Galaxy A51 5G, Galaxy A51 5G UW, Galaxy A71 5G, Galaxy A21s, Galaxy M01, Galaxy M01s, Galaxy A01 Core, Galaxy M31s, Galaxy Tab S7+, Galaxy Tab S7, Galaxy Z Flip 5G, Galaxy Note 20 Ultra, Galaxy Note 20 Ultra 5G, Galaxy Note 20, Galaxy M51, Galaxy Z Fold2 5G, Galaxy S20 FE, Galaxy S20 FE 5G, Galaxy S20 FE 2022, Galaxy F41, Galaxy A42 5G, Galaxy M21s, Galaxy A12, Galaxy A02s, Galaxy M02s, Galaxy A32 5G, Galaxy S21 Ultra, Galaxy S21+, Galaxy S21, Galaxy A02, Galaxy M02, Galaxy M12, Galaxy F62, Galaxy M62, Galaxy A32, Galaxy A52, Galaxy A52 5G, Galaxy A72, Galaxy F12, Galaxy F02s, Galaxy M42 5G, Galaxy F52 5G, Galaxy Tab A7 Lite, Galaxy Tab S7 FE, Galaxy A22, Galaxy A22 5G, Galaxy M32, Galaxy F22, Galaxy Z Flip3 5G, Galaxy Z Fold3 5G, Galaxy A52s 5G, Galaxy A03s, Galaxy M32 5G, Galaxy M22, Galaxy M52

5G, Galaxy F42 5G, Galaxy A03 Core, Galaxy A03, Galaxy Tab A8 10.5, Galaxy A13 5G, Galaxy S21 FE, Galaxy S22, Galaxy S22+, Galaxy S22 Ultra, Galaxy A13, Galaxy A23 5G, Galaxy A23, Galaxy F23, Galaxy A53 5G, Galaxy Tab S8, Galaxy Tab S8+, Galaxy Tab S8 Ultra, Galaxy A73 5G, Galaxy A33 5G, Galaxy M33 5G, Galaxy M23, Galaxy M53 5G, Galaxy F13, Galaxy M13, Galaxy M13 5G, Galaxy Xcover6 Pro, Galaxy Z Flip 4, Galaxy Z Fold 4, Galaxy Tab Active4 Pro, Galaxy A04s, Galaxy A04, Galaxy A04e, Galaxy Tab A7 10.4 (2022), Galaxy M04, Galaxy F04, Galaxy A14 5G, Galaxy S23 Ultra, Galaxy S23+, Galaxy S23, Galaxy Xcover 5, Galaxy Tab A7, Galaxy Tab A7 10.4 (2020), Galaxy Tab A 8.0 (2019), Galaxy J3 (2018), and Galaxy Tab E.

Samsung televisions made, used, sold, offered for sale, or imported within six years of the filing of the date of the lawsuit comprising Knox functionality, Tizen functionality, and/or push messaging functionality, with chassis numbers including, but not limited to the following: MCWS1, QCQ800, QCQ900, G75NC, G85NC, G95SC, QCQ700, S90PC, LS03C QCQ80, QCQ80D, QCQ80TL, QCQ83, QCQ85, QCQ85D, QCQ85TL, QCQ88, QCQ8X, QCQ90, QCQ90D, QCQ90TL, QCQ95, QCQ95D, QCQ95TL, QCQ9X, QCQS90, QCQS90D, QCQS90TL, QCQS95, QCQS95D, QCQS95TL, QCQS9X, M70C,M80C,QCQ70, QAQ70, QAQ7A, QAQ7D, QCQ70D_NM2, QCQ70_NM2, QCQ7X_NM2, QTQ70, M50C,QCQ60, LSP3C,UCU7000, UCU8000TL_PML,UCU8000_PML,UCU800M_PML,UCU80MD_PML, UCU8000, QBQ900, QBQ850, QBQ800, QBQ100, MBWS1, G97NB, QBQ700, G85NB, G75NB, G95NB, LS04B, LS03B, QBQ75, QBQS95, QBQS90,QBQD90, QBQ9X, QBQ9D, QBQ95, QBQ90S, QBQ90, QBQ8D, QBQ85D, QBQ85, QBQ83, QBQ80, QBQ7X, QBQ70, QBQX3, LS01B, LS05B, G65B, G70B, G8QNB, M80B, G65B, G70B, G70NC, G85SB, LS01B_ML, LS05B_ML, QBQ70_ML, QBQ7X_ML, BQ80_ML, QBQ83_ML, QBQ85D_ML, QBQ85_ML, QBQ8D_ML, QBQ90_ML, QBQ9D_ML, QBQ9X_ML, QAQ70, QAQ7D, QAQ90, QBQ6D, QBQ63, QBQ60_DPP, QBQ60, M70BO, M70B, M50B, UAU8000_L, QAQ60_K, UAU850D_K65, UAU8000_K65, UAU8000_K, LS03BS, UBU7000, UBU8000, LSP3B, UBU8000_X, QAQ9X, QAQ9D, QAQ95, QAQ90_NS, QAQ90, QAQ8D, QAQ85_NS, QAQ85D, QAQ85, QAQ80_NS, QAQ80, QAQ7D, QAQ70_NS, QAQ70, LST5A, QAQ7A, LS03A_NS, LS03AP, LS03A, LS03APD, LS03AD, QAQ900, QAQ850, QAQ800, QAQ700_P, MAWS1, QAQ700, QAQ700_S, UAU9070, UAU9000, QAQ6D, QAQ60, UAU850D, UAU8000, LS05AL, QAQ6D_K, QAQ60_K, QAQ50_K, UAUE60A, UAU7500, UAU7000, UAUE70A, UAU850D_K, UAU8000_K, UAU7700, LSP3, BEA_HB, KTSU2EL_ATM, KTSU2EL_HKC, KTSU2EL_HKC, KTSU2EL_REF, KTSU2EL_TPO, KTSU2EL_TPO, QTQ800, QTQ800_NS, QTQ850, QTQ900, QTQ950, QTQ950_NS, LSR9, LST7T, QTQ70, QTQ70_NS, QTQ72, QTQ7D, QTQ7X, QTQ80, QTQ80_NS, QTQ8D, QTQ8E, QTQ90, QTQ90_NS, QTQ95, QTQ95_NS, LS05T, LS03T, LS03T_NS, LS01T, LS01T_NS, MTWS1, QTQ60, QTQ62, QTQ6D, QTQ6E, QTQ6X, UTU8500, UTU850D, UTU8510, UTU8570, UTU85A0, QTQ6AC, QTQ6SC, QTQ6XC, UTU8000, UTU800D, UTU8200, UTU8300, UTU8400, LS03TS, UTU7000, UTU700D, UTU7090, UT4310, UT4500, UT4700, UT5310, UT5500, UT5770, UT4300, UT4400, UT5300, UT430E, UT530E, QRQ900B, QRQ90, QRQ85_OC, QRQ80_OC, QRQ80, QRQ75S, QRQ75, QRQ70, QRQ6X, QRQ69, QRQ65A, QRQ60, LS03R, LS05R, LS01R, URU800D, URU8000, MLS07R, QRQ50S, QRQ50, URU7800, URU7700, URU74A0, URU7470, URU7450, URU7410, URU740D, URU7400, URU730D, URU7300, URU710D, URU7100,URU7790,URU77FA, URU7000, URU6900, URUF58T, RUF58T, UR6000, UR5500, UR4500, QRQ900,QNQ9S, QNQ9F, QNQ8FB, QNQ8F, QNQ8C, QNQ7FH, QNQ7F, QNQ7C, QNQ75F, QNQ75C, QNQ6FK, QNQ6F, QNQ65FB,

QNQ65F,UNU850D, UNU8500, UNU80A0, UNU800D, UNU8000,UNU76A0, UNU7500, UNU74A0, UNU7450, UNU7400,ULS03NU, UNU730D, UNU7300, UNU710D, UNU7103, UNU7100, URU7000, UNU7120, UNU709D, UNU7090, UNU7080, UNU7050, UNU6950, UN5510, UN5500, UN5350, UN5305, UN5300, UN5200, UN4510, UN4500, UN4350, UN4310, UN4300, BERTB, BERTA, QMS9Y, QMQ9FR, QMQ9F, QMQ8CR, QMQ8C, QMQ7FVRE, QMQ7FRE, QMQ7FR, QMQ7FDR, QMQ7FD, QMQ7F, QMQ7CRE, QMQ7CR, QMQ7CDR, QMQ7CD, QMQ7C, QMQ6FRE, UMQS9, UMU900R, UMU9000, UMU800R, UMU8000, UMU750R, UMU7500, UMU70AR, UMU70A0, UMU700R, UMU7000, UMU66A0, UMU6500, UMU64A0, UMU6470, UMU6400, UMLS003, ULS003U, UMU66AR, UMU650R, UMU64AR, UMU640R, UMU6303, UMU6302, UMU6300, UMU6310,UMU6320,UMU6103, UMU6102, UMU6100, UMU6070, UMU6000,UMUF31E, UMUF30E, UM6303, UM6302, UM6300, UM5603, UM5602, UM5600, UM5523, UM5522, UM5520, UM5513, UM5512, UM5510, UM5503, UM5502, UM5500, UM5300, UM4500, UM4300,TH39S,VF39S, UNU7000, UMU6100, or, alternatively model numbers in the following formats: QN***C, LS0*C, QN**C, S90C, S95C, Q**C, CU****,QN***B, LS0*B, QN**B, S90B, S95B, Q**B, BU****, QN***A, LS0*A, QN**A, Q**A, AU****, QN***T, LS0*T, QN**T, Q**T, TU****, QN***R, LS0*R, QN**R, Q**R, RU****, QN*FN, NU***, LS0*NU, Q7F, Q6F, MU***, U7000, MU6100.

Samsung wearables made, used, sold, offered for sale, or imported within six years of the filing of the date of the lawsuit comprising Knox functionality, Tizen functionality, and/or push messaging functionality including, but not limited to the following: SM-R800 (46mm), SM-R805 (46mm/Cellular), SM-R810 (42mm), SM-R815 (42mm/Cellular), SM-R855F, SM-845F (Global, LTE model) SM-R855U SM-R845U (USA, LTE model) SM-R850, SM-840 (Wi-Fi only), SM-R860 SM-R865 SM-R870 SM-R875 SM-R880 SM-R885 SM-R890 SM-R895, SM-R905 (40mm), SM-R915 (44mm), SM-R925 (45mm), SM-R930 (40mm/Wi-Fi), SM-R935 (40mm/LTE), SM-R940 (44mm/Wi-Fi), SM-R945 (44mm/LTE), SM-R950 (43mm/Wi-Fi), SM-R955 (43mm/LTE), SM-R960 (47mm/Wi-Fi), SM-R965 (47mm/LTE), SM-R500 (40mm), SM-R820 (44mm), SM-R825 (44mm/Cellular), SM-R830 (40mm), SM-R835 (40mm/Cellular).

The Samsung products in the preceding list, including all variations, editions, and applications of the foregoing, and all products and apparatuses of Samsung similar to the foregoing that include the claimed elements are the ’117 Accused Instrumentalities.

U.S. PATENT NO. 9,615,192

Headwater accuses the following Samsung products which implement Samsung Knox, Tizen, and/or push messaging functionality (e.g., communicating with Google’s Firebase Cloud Messaging service, Samsung’s Push Service, Knox, or Tizen functionality) of infringing each of the Asserted Claims of the ’192 patent:

Servers leased, owned, supported, and/or operated by Samsung comprising Knox functionality, Tizen functionality and/or push messaging functionality.

Samsung phones, tablets, and devices made, used, sold, offered for sale, or imported within six years of the filing of the date of the lawsuit comprising Knox functionality, Tizen functionality, and/or push messaging functionality, including but not limited to the following: Galaxy S II Plus, Galaxy Grand, Galaxy Young, Galaxy Xcover 2, Galaxy S4, Galaxy Fame, Galaxy Mega, Galaxy Win, Galaxy Y Plus, Galaxy Core, Galaxy Star, Galaxy Pocket Neo, Galaxy Ace 3, Galaxy S4 Zoom, Galaxy S4 Active, Galaxy S4 Mini, Galaxy Gear, Galaxy Note 3, Galaxy Core Plus, Galaxy Light, Galaxy Fame Lite, Galaxy Trend Lite, Galaxy Round, Galaxy Express 2, Galaxy J, Galaxy Star Pro, Galaxy Grand 2, Galaxy Trend Plus, Galaxy S Duos 2, Galaxy J, Galaxy Win Pro (SM-G3812), Galaxy Grand Lite, Galaxy Grand Neo, Galaxy Note 3 Neo, Galaxy Win 2, Galaxy S3 Neo, Galaxy S5, Galaxy Ace Style, Galaxy K Zoom, Galaxy Core, Galaxy Tab S 10.5, Galaxy S5 Mini, Galaxy Core 2, Galaxy S Duos 3, Galaxy Ace 4, Galaxy Star 2 Plus, Galaxy Mega 2, Galaxy Pocket 2, Galaxy Core Prime, Galaxy Grand Prime, Galaxy Alpha, Galaxy Young 2, Galaxy Note 4, Galaxy Note Edge, Galaxy A3 (2015), Galaxy A5 (2015) (released 2014), Galaxy Pocket Plus, Galaxy E7, Galaxy A7 (2015), Galaxy E5, Galaxy J1, Galaxy Xcover 3, Galaxy S6, Galaxy S6 Edge, Galaxy J7, Galaxy J5, Galaxy S6 Active, Galaxy V Plus, Galaxy Trend 2 Lite, Galaxy S5 Neo, Galaxy Note 5, Galaxy A8 (2015), Galaxy S6 Edge+, Galaxy J2, Galaxy J1 Ace, Galaxy A3 (2016), Galaxy A5 (2016), Galaxy A7 (2016), Galaxy A9 (2016), Galaxy J1 (2016), Galaxy S7, Galaxy S7 Edge, Galaxy J1 Mini, Galaxy A9 Pro (2016), Galaxy Tab A6, Galaxy J3 (2016), Galaxy J7 (2016), Galaxy J5 (2016), Galaxy C7, Galaxy C5, Galaxy Tab A 10.1 (2016), Galaxy J3 Pro, Galaxy S7 Active, Galaxy J2 Prime, Galaxy Note 7, Galaxy J7 Prime, Galaxy On7 (2016), Galaxy On5 Pro (2016), Galaxy On5 (2016), Galaxy A8 (2016), Galaxy C9 Pro, Galaxy J2 Prime, Galaxy J1 mini Prime/Galaxy V2, Galaxy C5 Pro, Galaxy A3 (2017), Galaxy A5 (2017), Galaxy A7 (2017), Galaxy Tab S3, Galaxy S8+, Galaxy S8, Galaxy Xcover 4, Galaxy J7 Max, Galaxy J7 Pro (2017), Galaxy J7 (2017), Galaxy J5 (2017), Galaxy J3 Prime, Galaxy J3 (2017), Galaxy Note Fan Edition (FE), Galaxy S8 Active, Galaxy Note 8, Galaxy J7+, Galaxy J7 Sky Pro, Galaxy J7 Nxt, Galaxy J7 V, Galaxy C8 / C7 (2017), Galaxy Tab A 8.0 (2017), Galaxy Tab Active 2, Galaxy J4 (2017), Galaxy A8+ (2018), Galaxy A8 (2018), Galaxy J4 Pro (2018), Galaxy S9+, Galaxy S9, Galaxy J7 Duo, Galaxy J6 (2018), Galaxy J4 (2018), Galaxy A6+ (2018), Galaxy A6 (2018), Galaxy J7 (2018), Galaxy On6, Galaxy J8 (2018), Galaxy Tab S4, Galaxy Tab A 10.5 (2018), Galaxy Note 9, Galaxy J6+, Galaxy J4+, Galaxy A7 (2018), Galaxy A9 (2018), Galaxy A6s, Galaxy A8+ (2018), Galaxy A8 (2018), Galaxy A8s, Galaxy M20, Galaxy M10, Galaxy M30, Galaxy Tab A 10.1 (2019), Galaxy Tab S5e, Galaxy A10, Galaxy A30, Galaxy A50, Galaxy S10e, Galaxy S10, Galaxy S10+, Galaxy A2 Core, Galaxy A20e, Galaxy A20, Galaxy A40, Galaxy A70, Galaxy A80, Galaxy A60, Galaxy View 2, Galaxy S10 5G, Galaxy M40, Galaxy Note 10+, Galaxy Note 10, Galaxy A10e, Galaxy A10s, Galaxy Fold, Galaxy A90 5G, Galaxy A30s, Galaxy A50s, Galaxy A70s, Galaxy M10s, Galaxy A20s, Galaxy M30s, Galaxy A51, Galaxy A71, Galaxy A01, Galaxy S10 Lite, Galaxy Note 10 Lite, Galaxy Xcover Pro, Galaxy Tab S6, Galaxy Tab S6 5G, Galaxy Z Flip, Galaxy M31, Galaxy A11, Galaxy M21, Galaxy A41, Galaxy A31, Galaxy M11, Galaxy S20 Ultra, Galaxy S20+, Galaxy S20, Galaxy Tab S6 Lite, Galaxy A21, Galaxy A51 5G, Galaxy A51 5G UW, Galaxy A71 5G, Galaxy A21s, Galaxy M01, Galaxy M01s, Galaxy A01 Core, Galaxy M31s, Galaxy Tab S7+, Galaxy Tab S7, Galaxy Z Flip 5G, Galaxy Note 20 Ultra, Galaxy Note 20 Ultra 5G, Galaxy Note 20, Galaxy M51, Galaxy Z Fold2 5G, Galaxy S20 FE, Galaxy S20 FE 5G, Galaxy S20 FE 2022, Galaxy F41, Galaxy A42 5G, Galaxy M21s, Galaxy A12, Galaxy

A02s, Galaxy M02s, Galaxy A32 5G, Galaxy S21 Ultra, Galaxy S21+, Galaxy S21, Galaxy A02, Galaxy M02, Galaxy M12, Galaxy F62, Galaxy M62, Galaxy A32, Galaxy A52, Galaxy A52 5G, Galaxy A72, Galaxy F12, Galaxy F02s, Galaxy M42 5G, Galaxy F52 5G, Galaxy Tab A7 Lite, Galaxy Tab S7 FE, Galaxy A22, Galaxy A22 5G, Galaxy M32, Galaxy F22, Galaxy Z Flip3 5G, Galaxy Z Fold3 5G, Galaxy A52s 5G, Galaxy A03s, Galaxy M32 5G, Galaxy M22, Galaxy M52 5G, Galaxy F42 5G, Galaxy A03 Core, Galaxy A03, Galaxy Tab A8 10.5, Galaxy A13 5G, Galaxy S21 FE, Galaxy S22, Galaxy S22+, Galaxy S22 Ultra, Galaxy A13, Galaxy A23 5G, Galaxy A23, Galaxy F23, Galaxy A53 5G, Galaxy Tab S8, Galaxy Tab S8+, Galaxy Tab S8 Ultra, Galaxy A73 5G, Galaxy A33 5G, Galaxy M33 5G, Galaxy M23, Galaxy M53 5G, Galaxy F13, Galaxy M13, Galaxy M13 5G, Galaxy Xcover6 Pro, Galaxy Z Flip 4, Galaxy Z Fold 4, Galaxy Tab Active4 Pro, Galaxy A04s, Galaxy A04, Galaxy A04e, Galaxy Tab A7 10.4 (2022), Galaxy M04, Galaxy F04, Galaxy A14 5G, Galaxy S23 Ultra, Galaxy S23+, Galaxy S23, Galaxy Xcover 5, Galaxy Tab A7, Galaxy Tab A7 10.4 (2020), Galaxy Tab A 8.0 (2019), Galaxy J3 (2018), and Galaxy Tab E.

Samsung televisions made, used, sold, offered for sale, or imported within six years of the filing of the date of the lawsuit comprising Knox functionality, Tizen functionality, and/or push messaging functionality, with chassis numbers including, but not limited to the following: MCWS1, QCQ800, QCQ900, G75NC, G85NC, G95SC, QCQ700, S90PC, LS03C QCQ80, QCQ80D, QCQ80TL, QCQ83, QCQ85, QCQ85D, QCQ85TL, QCQ88, QCQ8X, QCQ90, QCQ90D, QCQ90TL, QCQ95, QCQ95D, QCQ95TL, QCQ9X, QCQS90, QCQS90D, QCQS90TL, QCQS95, QCQS95D, QCQS95TL, QCQS9X, M70C,M80C,QCQ70, QAQ70, QAQ7A, QAQ7D, QCQ70D_NM2, QCQ70_NM2, QCQ7X_NM2, QTQ70, M50C,QCQ60, LSP3C,UCU7000, UCU8000TL_PML,UCU8000_PML,UCU800M_PML,UCU80MD_PML, UCU8000, QBQ900, QBQ850, QBQ800, QBQ100, MBWS1, G97NB, QBQ700, G85NB, G75NB, G95NB, LS04B, LS03B, QBQ75, QBQS95, QBQS90,QBQD90, QBQ9X, QBQ9D, QBQ95, QBQ90S, QBQ90, QBQ8D, QBQ85D, QBQ85, QBQ83, QBQ80, QBQ7X, QBQ70, QBQX3, LS01B, LS05B, G65B, G70B, G8QNB, M80B, G65B, G70B, G70NC, G85SB, LS01B_ML, LS05B_ML, QBQ70_ML, QBQ7X_ML, BQ80_ML, QBQ83_ML, QBQ85D_ML, QBQ85_ML, QBQ8D_ML, QBQ90_ML, QBQ9D_ML, QBQ9X_ML, QAQ70, QAQ7D, QAQ90, QBQ6D, QBQ63, QBQ60_DPP, QBQ60, M70BO, M70B, M50B, UAU8000_L, QAQ60_K, UAU850D_K65, UAU8000_K65, UAU8000_K, LS03BS, UBU7000, UBU8000, LSP3B, UBU8000_X, QAQ9X, QAQ9D, QAQ95, QAQ90_NS, QAQ90, QAQ8D, QAQ85_NS, QAQ85D, QAQ85, QAQ80_NS, QAQ80, QAQ7D, QAQ70_NS, QAQ70, LST5A, QAQ7A, LS03A_NS, LS03AP, LS03A, LS03APD, LS03AD, QAQ900, QAQ850, QAQ800, QAQ700_P, MAWS1, QAQ700, QAQ700_S, UAU9070, UAU9000, QAQ6D, QAQ60, UAU850D, UAU8000, LS05AL, QAQ6D_K, QAQ60_K, QAQ50_K, UAUE60A, UAU7500, UAU7000, UAUE70A, UAU850D_K, UAU8000_K, UAU7700, LSP3, BEA_HB, KTSU2EL_ATM, KTSU2EL_HKC, KTSU2EL_HKC, KTSU2EL_REF, KTSU2EL_TPO, KTSU2EL_TPO, QTQ800, QTQ800_NS, QTQ850, QTQ900, QTQ950, QTQ950_NS, LSR9, LST7T, QTQ70, QTQ70_NS, QTQ72, QTQ7D, QTQ7X, QTQ80, QTQ80_NS, QTQ8D, QTQ8E, QTQ90, QTQ90_NS, QTQ95, QTQ95_NS, LS05T, LS03T, LS03T_NS, LS01T, LS01T_NS, MTWS1, QTQ60, QTQ62, QTQ6D, QTQ6E, QTQ6X, UTU8500, UTU850D, UTU8510, UTU8570, UTU85A0, QTQ6AC, QTQ6SC, QTQ6XC, UTU8000, UTU800D, UTU8200, UTU8300, UTU8400, LS03TS, UTU7000, UTU700D, UTU7090, UT4310, UT4500, UT4700, UT5310, UT5500, UT5770, UT4300, UT4400, UT5300, UT430E, UT530E, QRQ900B, QRQ90, QRQ85_OC, QRQ80_OC, QRQ80, QRQ75S, QRQ75, QRQ70, QRQ6X, QRQ69, QRQ65A,

QRQ60, LS03R, LS05R, LS01R, URU800D, URU8000, MLS07R, QRQ50S, QRQ50, URU7800, URU7700, URU74A0, URU7470, URU7450, URU7410, URU740D, URU7400, URU730D, URU7300, URU710D, URU7100,URU7790,URU77FA, URU7000, URU6900, URUF58T, RUF58T, UR6000, UR5500, UR4500, QRQ900,QNQ9S, QNQ9F, QNQ8FB, QNQ8F, QNQ8C, QNQ7FH, QNQ7F, QNQ7C, QNQ75F, QNQ75C, QNQ6FK, QNQ6F, QNQ65FB, QNQ65F,UNU850D, UNU8500, UNU80A0, UNU800D, UNU8000,UNU76A0, UNU7500, UNU74A0, UNU7450, UNU7400,ULS03NU, UNU730D, UNU7300, UNU710D, UNU7103, UNU7100, URU7000, UNU7120, UNU709D, UNU7090, UNU7080, UNU7050, UNU6950, UN5510, UN5500, UN5350, UN5305, UN5300, UN5200, UN4510, UN4500, UN4350, UN4310, UN4300, BERTB, BERTA, QMS9Y, QMQ9FR, QMQ9F, QMQ8CR, QMQ8C, QMQ7FVRE, QMQ7FRE, QMQ7FR, QMQ7FDR, QMQ7FD, QMQ7F, QMQ7CRE, QMQ7CR, QMQ7CDR, QMQ7CD, QMQ7C, QMQ6FRE, UMQS9, UMU900R, UMU9000, UMU800R, UMU8000, UMU750R, UMU7500, UMU70AR, UMU70A0, UMU700R, UMU7000, UMU66A0, UMU6500, UMU64A0, UMU6470, UMU6400, UMLS003, ULS003U, UMU66AR, UMU650R, UMU64AR, UMU640R, UMU6303, UMU6302, UMU6300, UMU6310,UMU6320,UMU6103, UMU6102, UMU6100, UMU6070, UMU6000,UMUF31E, UMUF30E, UM6303, UM6302, UM6300, UM5603, UM5602, UM5600, UM5523, UM5522, UM5520, UM5513, UM5512, UM5510, UM5503, UM5502, UM5500, UM5300, UM4500, UM4300,TH39S,VF39S, UNU7000, UMU6100, or, alternatively model numbers in the following formats: QN***C, LS0*C, QN**C, S90C, S95C, Q**C, CU****,QN***B, LS0*B, QN**B, S90B, S95B, Q**B, BU****, QN***A, LS0*A, QN**A, Q**A, AU****, QN***T, LS0*T, QN**T, Q**T, TU****, QN***R, LS0*R, QN**R, Q**R, RU****, QN*FN, NU***, LS0*NU, Q7F, Q6F, MU***, U7000, MU6100.

Samsung wearables made, used, sold, offered for sale, or imported within six years of the filing of the date of the lawsuit comprising Knox functionality, Tizen functionality, and/or push messaging functionality including, but not limited to the following: SM-R800 (46mm), SM-R805 (46mm/Cellular), SM-R810 (42mm), SM-R815 (42mm/Cellular), SM-R855F, SM-845F (Global, LTE model) SM-R855U SM-R845U (USA, LTE model) SM-R850, SM-840 (Wi-Fi only), SM-R860 SM-R865 SM-R870 SM-R875 SM-R880 SM-R885 SM-R890 SM-R895, SM-R905 (40mm), SM-R915 (44mm), SM-R925 (45mm), SM-R930 (40mm/Wi-Fi), SM-R935 (40mm/LTE), SM-R940 (44mm/Wi-Fi), SM-R945 (44mm/LTE), SM-R950 (43mm/Wi-Fi), SM-R955 (43mm/LTE), SM-R960 (47mm/Wi-Fi), SM-R965 (47mm/LTE), SM-R500 (40mm), SM-R820 (44mm), SM-R825 (44mm/Cellular), SM-R830 (40mm), SM-R835 (40mm/Cellular).

The Samsung products in the preceding list, including all variations, editions, and applications of the foregoing, and all products and apparatuses of Samsung similar to the foregoing that include the claimed elements are the ’192 Accused Instrumentalities.

P.R. 3-1(C): CLAIM CHARTS

Headwater's analysis of Samsung's products and apparatuses is based upon information that is publicly available and based on Headwater's own investigation prior to any discovery in this action.

While the publicly available information constitutes evidence of the methods and apparatuses used by Headwater in the Accused Instrumentalities, direct evidence of the actual apparatuses and methods are at times not publicly available. Accordingly, these infringement contentions are based on the available public information, analysis, and reasonable inferences drawn from that information.

Headwater reserves the right to amend or supplement these disclosures for any of the following reasons (along with any other reason that may be permitted under the court's rules and orders):

(1) Samsung provides evidence of the apparatuses and methods used in the Accused Instrumentalities;

(2) The Asserted Claims may include elements that involve features that are implemented by hardware structures and logic and Headwater's current positions on infringement are set forth without the benefit of access to Samsung's source code, schematics, drawings, or other proprietary specifications or information, which cannot be obtained through publicly available information, for the Accused Instrumentalities. Therefore, it may be necessary for Headwater to supplement its positions on infringement after a complete production of such proprietary specifications or information by Samsung;

(3) Headwater's position on infringement of specific claims will depend on the claim constructions adopted by the Court. Because said constructions have not yet occurred,

Headwater cannot take a final position on the bases for infringement of the Asserted Claims; and

(4) Headwater's investigation and analysis of Samsung's Accused Instrumentalities are based upon information made publicly available by Samsung and by Headwater's own investigations. Headwater reserves the right to amend these contentions based upon discovery of non-public information that Headwater anticipates receiving from Samsung during discovery.

Attached as Exhibits A through C, and incorporated herein in their entirety, are charts identifying where each element of the Asserted Claims of the '733, '117, and '192 patents are found in the Accused Instrumentalities.

Unless otherwise indicated, the information provided that corresponds to each claim element is considered to indicate that each claim element is found within each of the different variations, versions, editions, and applications of each respective Accused Instrumentalities.

P.R. 3-1(D): LITERAL INFRINGEMENT AND DOCTRINE OF EQUIVALENTS

With respect to the patents at issue, Headwater contends that each element of each Asserted Claim is literally present. To the extent that Samsung identifies elements of the Asserted Claims that it contends are not literally present in the Accused Instrumentalities, Headwater contends that such elements are present under the doctrine of equivalents.

P.R. 3-1(E): PRIORITY DATES

U.S. Patent No.	*Priority Date*
8,406,733	January 28, 2009
9,198,117	January 28, 2009
9,615,192	January 28, 2009

P.R. 3-1(F): IDENTIFICATION OF INSTRUMENTALITIES PRACTICING THE CLAIMED INVENTION

The ItsOn software may incorporate or reflect the claims of the Asserted Patents. Headwater reserves the right to supplement this response should further investigation, discovery, or the court's claim construction rulings make such supplementation appropriate.

P.R. 3-2: DOCUMENT PRODUCTION ACCOMPANYING DISCLOSURE

P.R. 3-2(A) DOCUMENTS

Headwater is presently unaware of any documents that evidence any discussion with, disclosure to, or other manner of providing to a third party, or sale of or offer to sell, any of the inventions claimed in the patents in suit prior to their respective application dates.

A diligent search continues for documents, and Headwater reserves the right to supplement this response.

P.R. 3-2(B) DOCUMENTS

Headwater is presently unaware of any documents that evidence the conception, reduction to practice, design, or development of the claimed inventions, which were created on or before the application dates of the patents in suit or priority date identified pursuant to P.R. 3-1(e).

A diligent search continues for documents, and Headwater reserves the right to supplement this response.

P.R. 3-2(C) DOCUMENTS

The file histories for the ’733, ’117, and ’192 patents may be found in Headwater’s production.

Date: September 28, 2023

/s/ Marc Fenster

Marc Fenster
CA State Bar No. 181067
Reza Mirzaie
CA State Bar No. 246953
Brian Ledahl
CA State Bar No. 186579
Ben Wang
CA State Bar No. 228712
Paul Kroeger
CA State Bar No. 229074
Neil A. Rubin
CA State Bar No. 250761
Kristopher Davis
CA State Bar No. 329627
James S. Tsuei
CA State Bar No. 285530
Philip Wang
CA State Bar No. 262239
Amy Hayden
CA State Bar No. 287026
James Milkey
CA State Bar No. 281283
Jason M. Wietholter
CA State Bar No. 337139
RUSS AUGUST & KABAT
12424 Wilshire Blvd. 12th Floor
Los Angeles, CA 90025
Telephone: 310-826-7474
rak_headwater@raklaw.com

ATTORNEYS FOR PLAINTIFF,
Headwater Research LLC

CERTIFICATE OF SERVICE

I certify that this document is being served upon counsel of record for Defendants on September 28, 2023 via electronic mail.

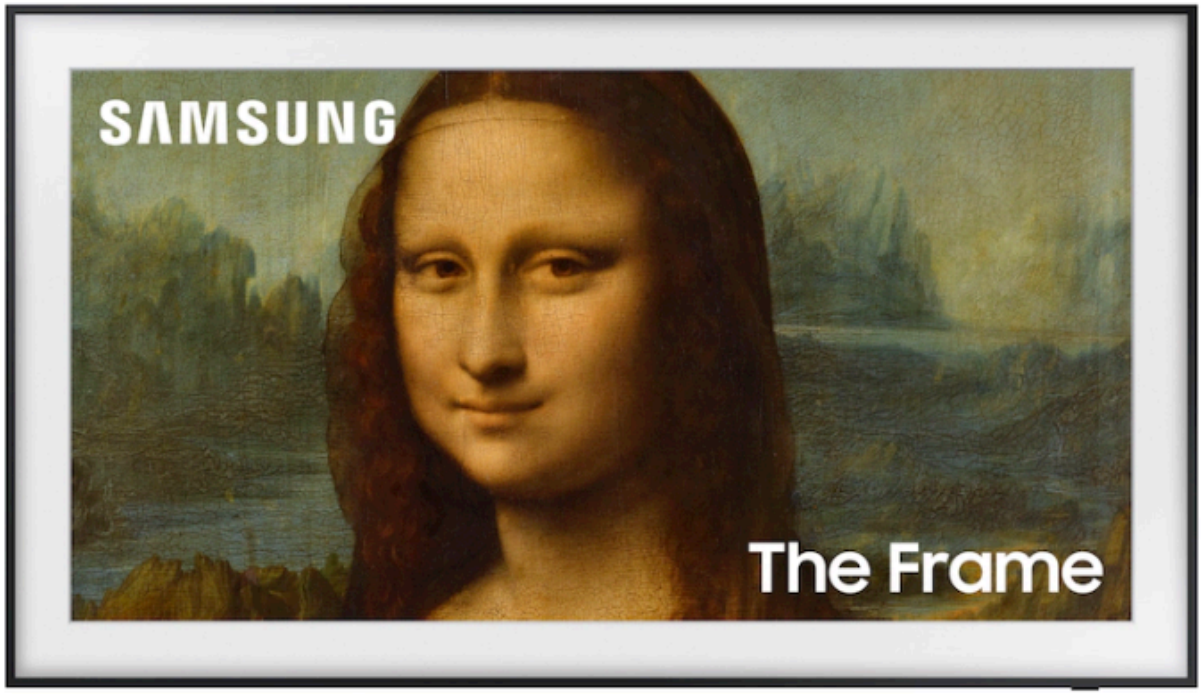
/s/ *Marc Fenster*
Marc Fenster

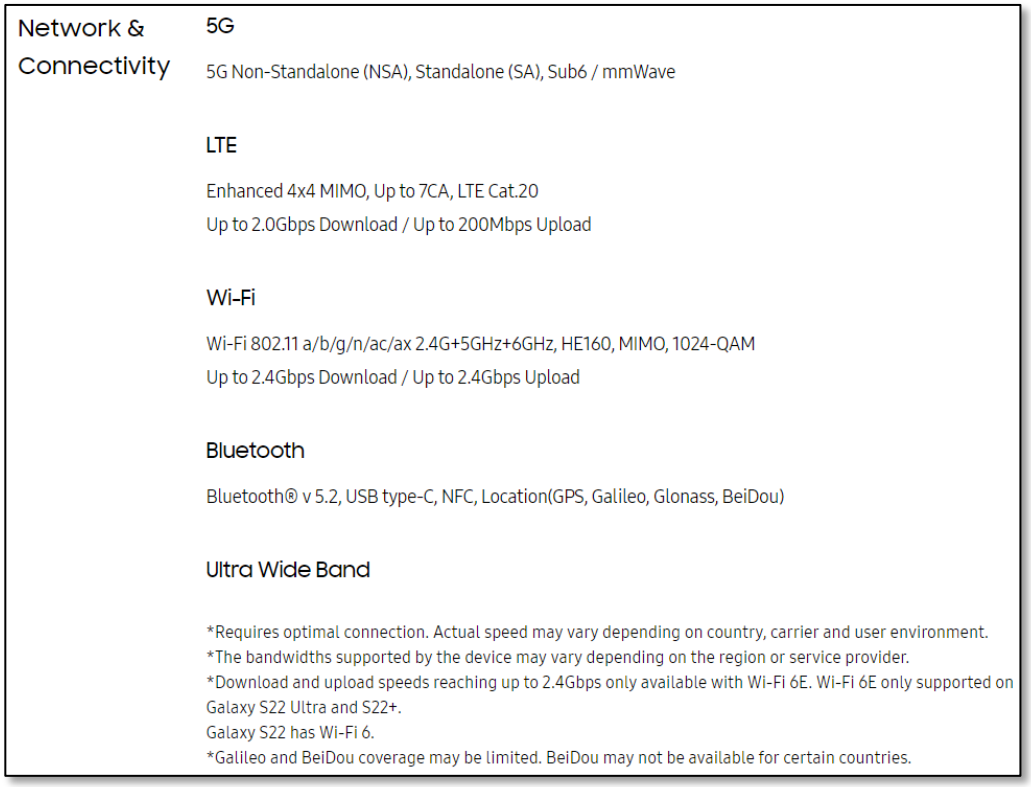
Exhibit A - U.S. Patent No. 8,406,733 ("'733 Patent")

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.

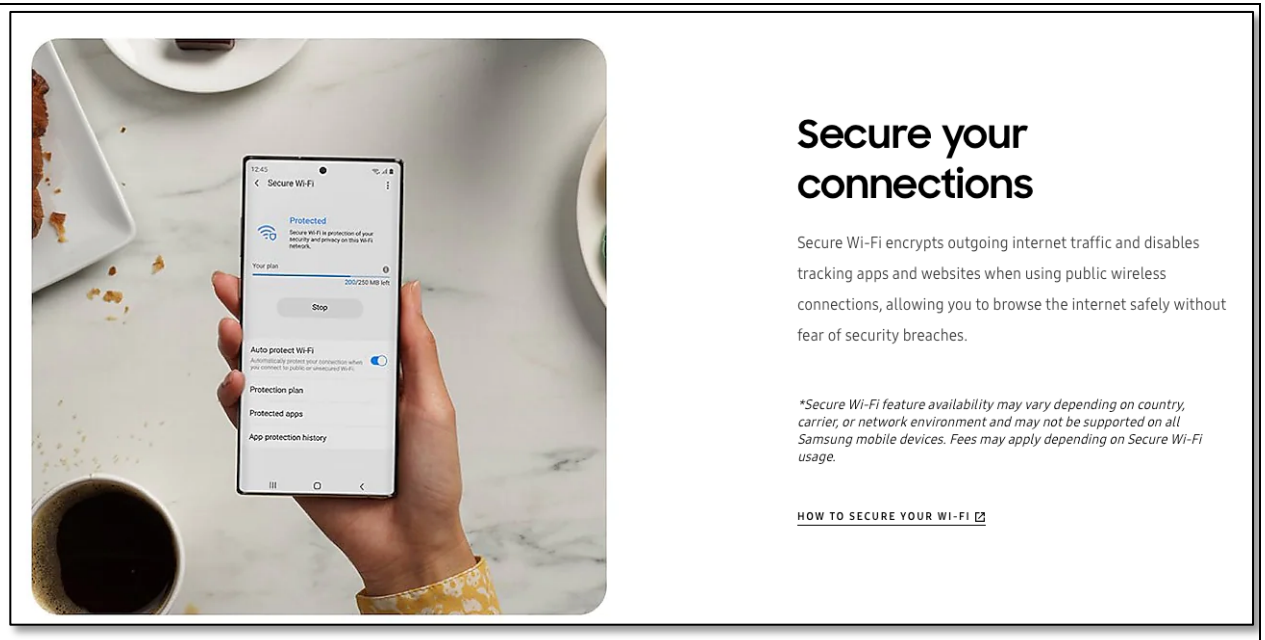
Claim 1

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. *See e.g.*,  https://www.samsung.com/us/smartphones/galaxy-s22/; As another example, Samsung's Tizen based devices are user devices. *See e.g.*,

	 https://www.samsung.com/us/televisions-home-theater/tvs/the-frame/55-class-the-frame-qled-4k-smart-tv-2022-qn55ls03bafxza/.
1[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"). *See e.g.*,

20	 https://www.samsung.com/us/smartphones/galaxy-s22/models/. Samsung Galaxy phones and tablets use encryption protocols to secure communications between the device and the network system. *See e.g.,*

21	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.

22	 https://www.samsung.com/us/security/;

23

Firebase > Documentation > FCM > Engage

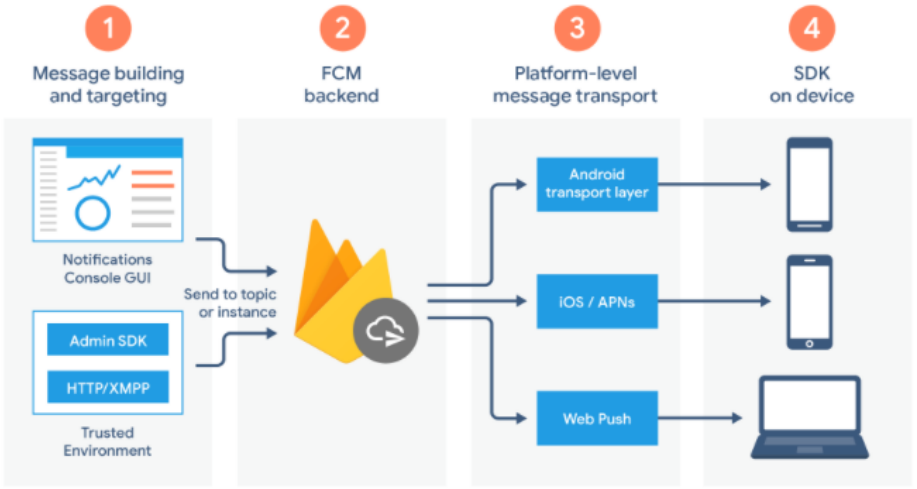
Was this helpful?

FCM Architectural Overview

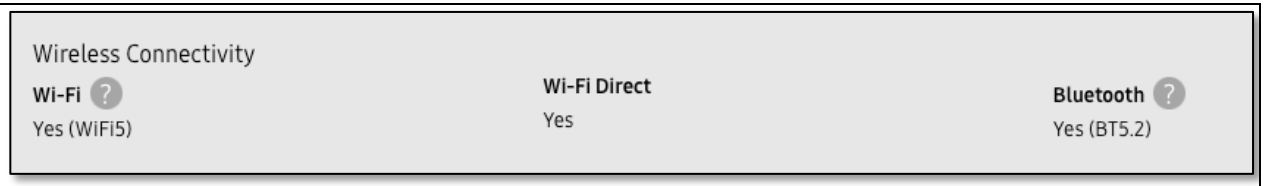
Send feedback

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.



24	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.*,

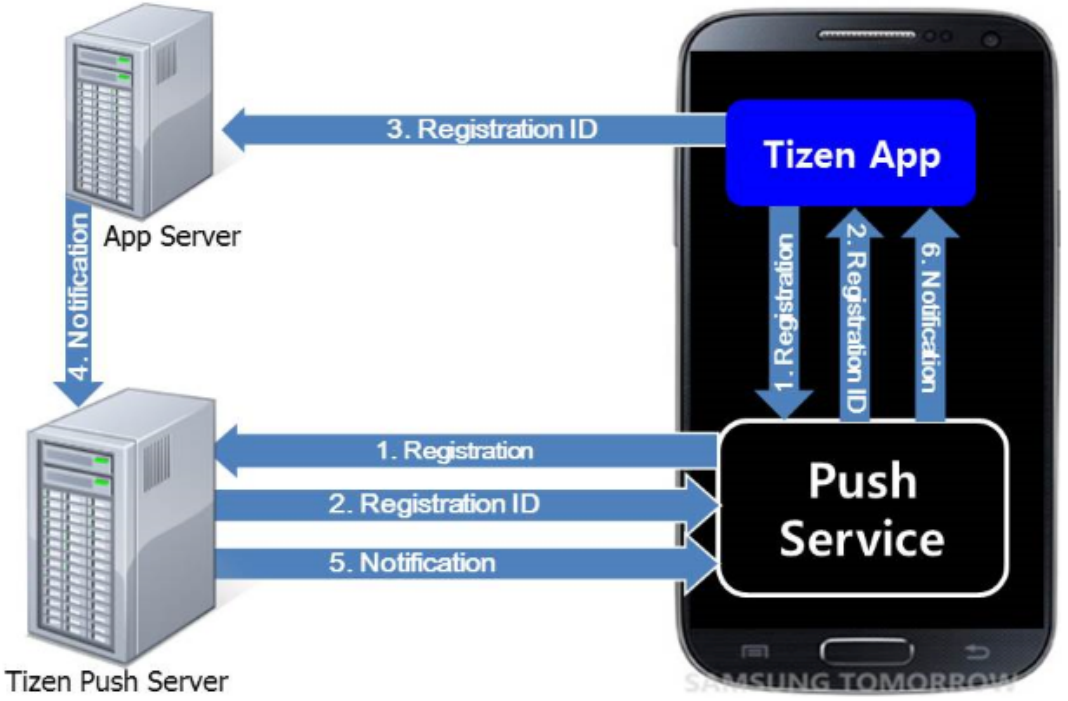
25	 https://www.samsung.com/us/televisions-home-theater/tvs/the-frame/55-class-the-frame-qled-4k-smart-tv-2022-qn55ls03bafxza/#specs; **Wireless security protocols** The TV only supports the following wireless network security protocols. The TV cannot connect to non-certified wireless access point. – Authentication Modes: WEP, WPAPSK, WPA2PSK – Encryption Types: WEP, TKIP, AES https://downloadcenter.samsung.com/content/UM/202203/20220303092001587/OSNATSCB-3.2.0_EM_Oscar_Pontus_Nike_Kant-SU2e_USA_ENG_220216.0.pdf. Further, Samsung's Tizen based devices use encryption protocols to secure communications between the device and the network system. *See e.g.,*

26

Service Architecture

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

Figure: Service architecture



27

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

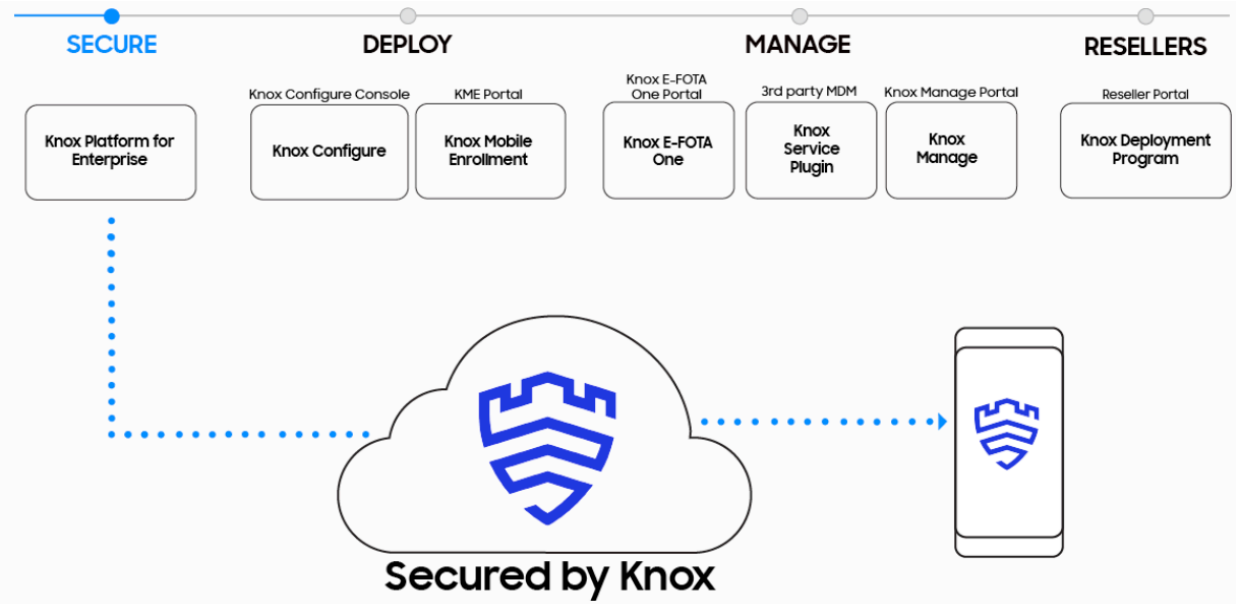
The security features introduce how private information remains private, and how the user knows when they are trying to access privileged information. You can use a repository and cryptographic operations to manage keys, certificates, and sensitive user data. When the user tries to access privileged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

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

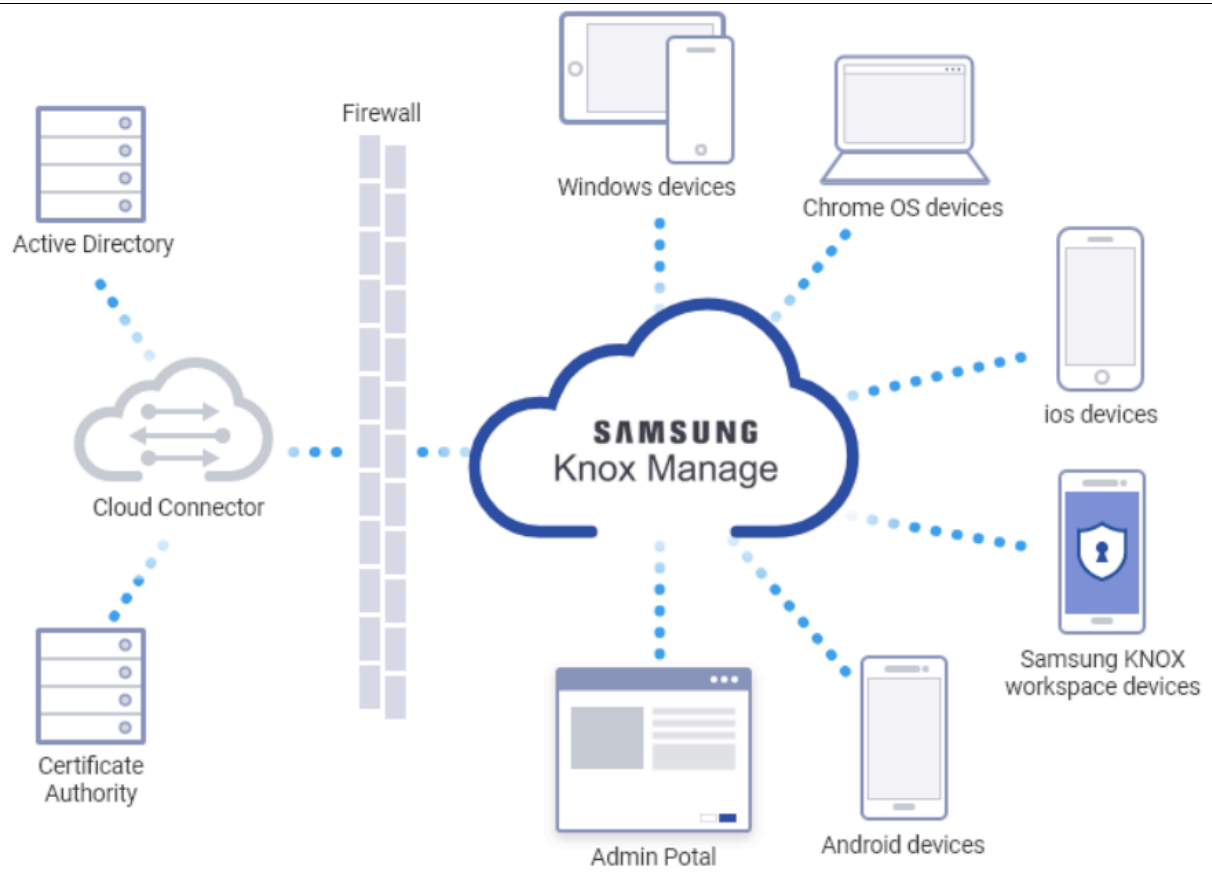
- Cryptographic Operations

  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.

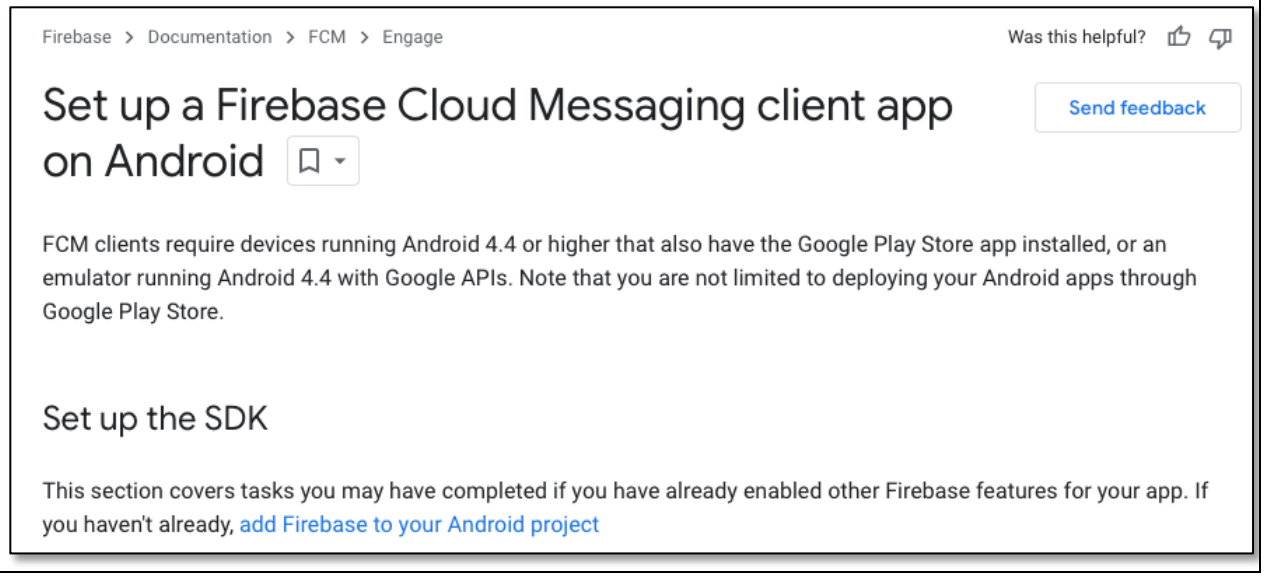
https://docs.tizen.org/application/native/guides/security/overview/.

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

29



https://docs.samsungknox.com/admin/knox-manage/welcome.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 which have an identifier. *See e.g.*, 

31

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`.

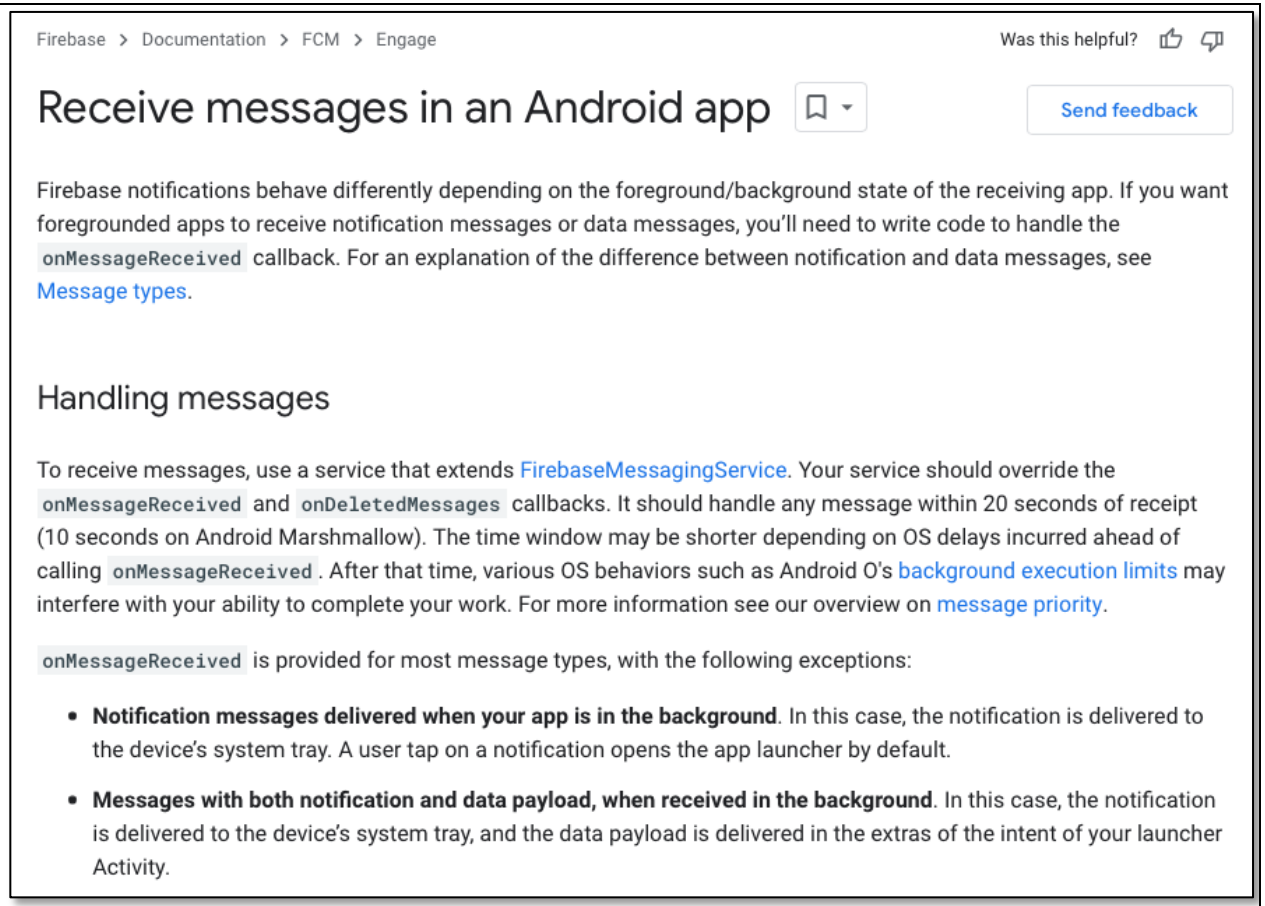
This section describes how to retrieve the token and how to monitor changes to the token. Because the token could be rotated after initial startup, you are strongly recommended to retrieve the latest updated registration token.

The registration token may change when:

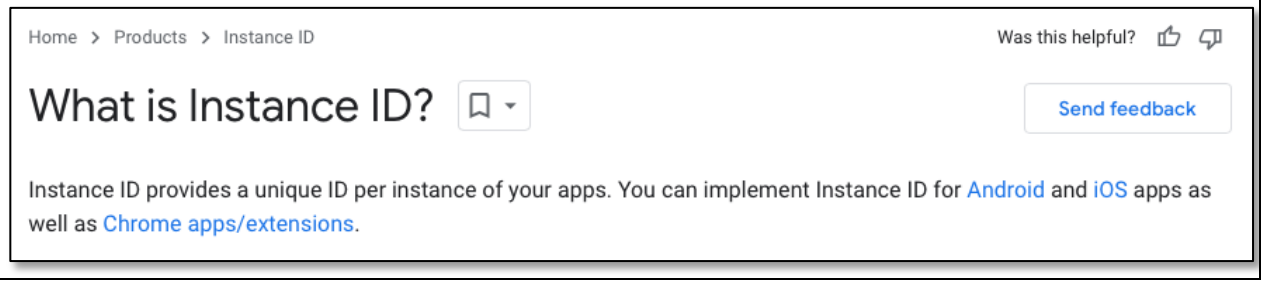
- The app is restored on a new device
- The user uninstalls/reinstall the app
- The user clears app data.

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

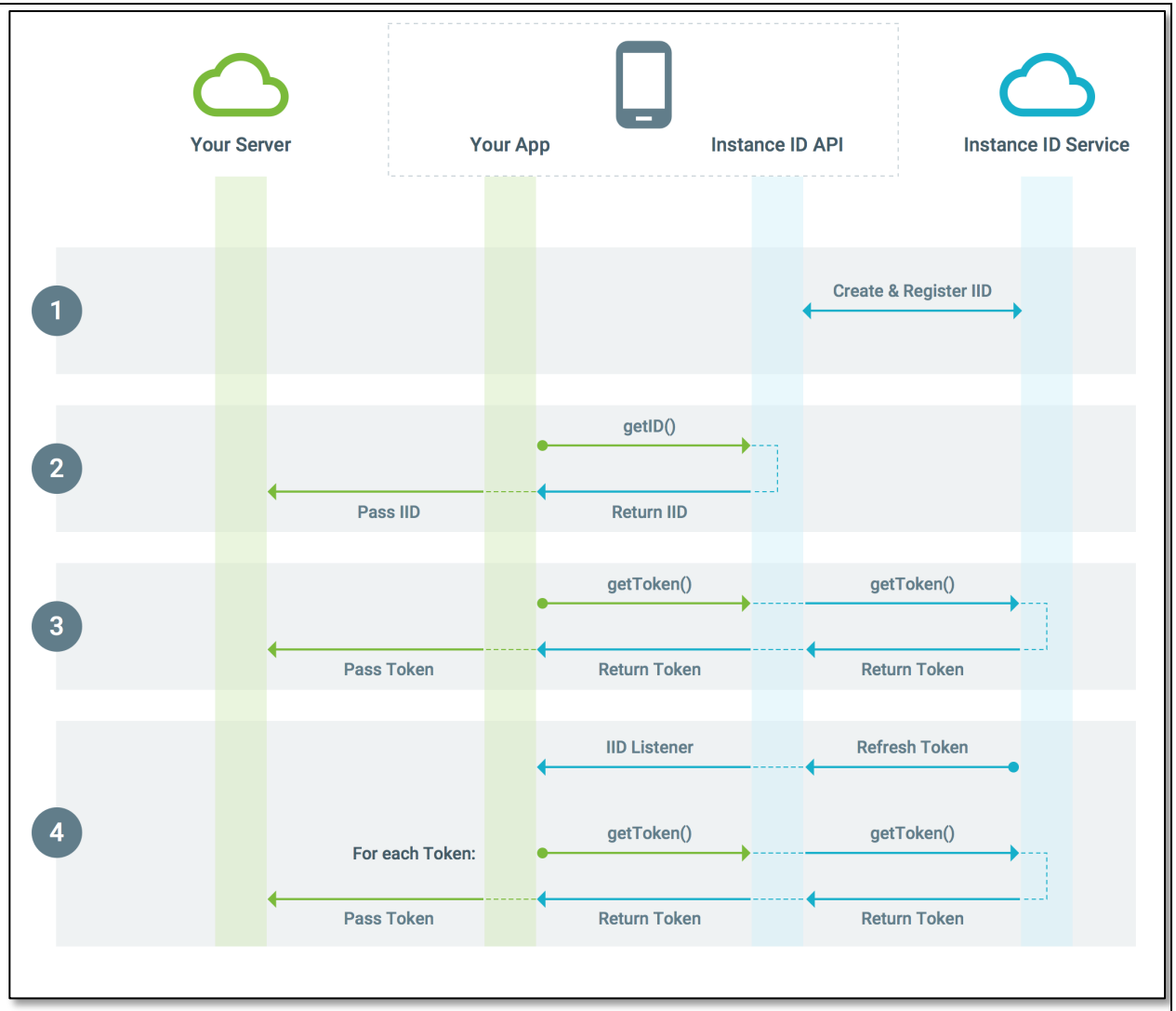
32



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



33



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

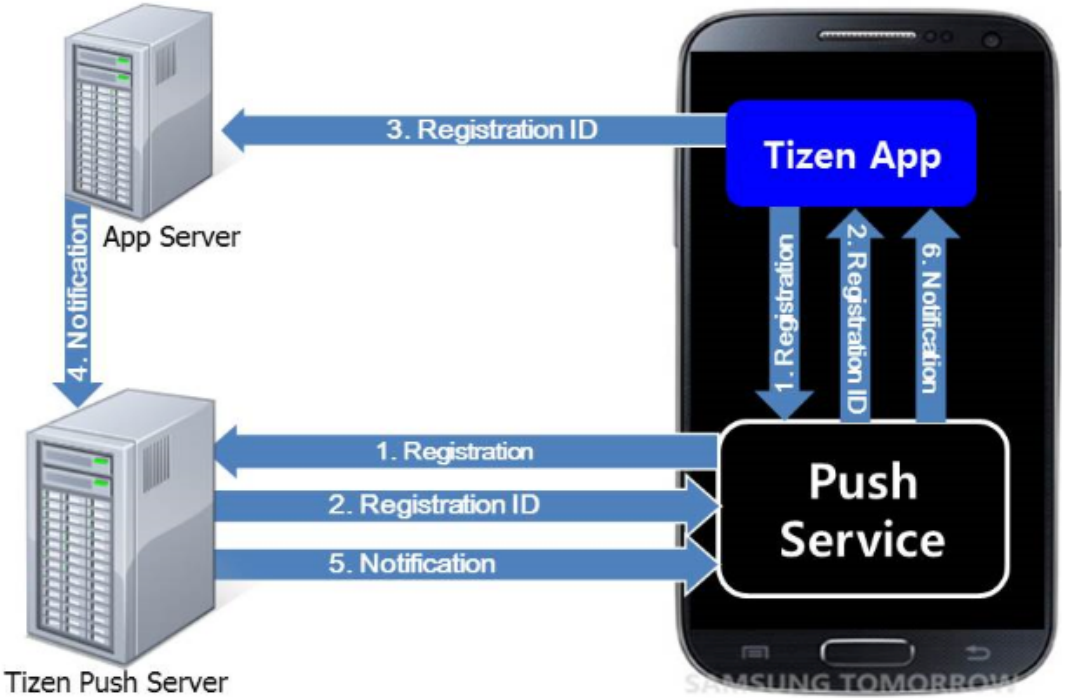
For further example, Samsung's Tizen based devices comprise multiple device agents which have an identifier. *See e.g.,*

34

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.

35

36

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);
```

37

- 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);
```

38

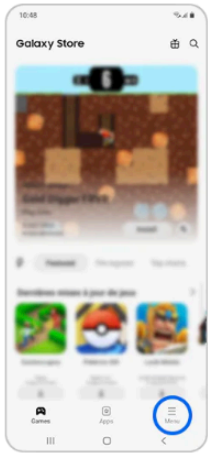
If the registration is successful, the `registerSuccessCallback()` callback is called, and the registration ID is passed as a parameter. Any time after a successful registration, you can get the registration ID using the `getRegistrationId()` method:

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

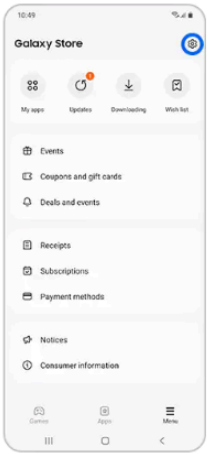
https://docs.tizen.org/application/web/guides/messaging/push/;

How to set up push notifications

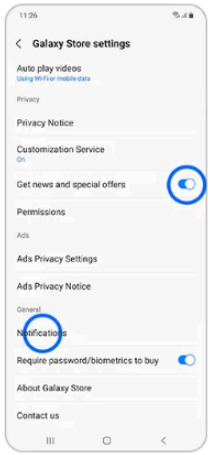
If you want to set up push notifications, follow these steps.



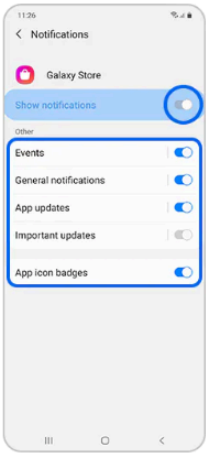
Step 1. Log in to Galaxy Store and tap the "Menu" icon from the main screen.



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."

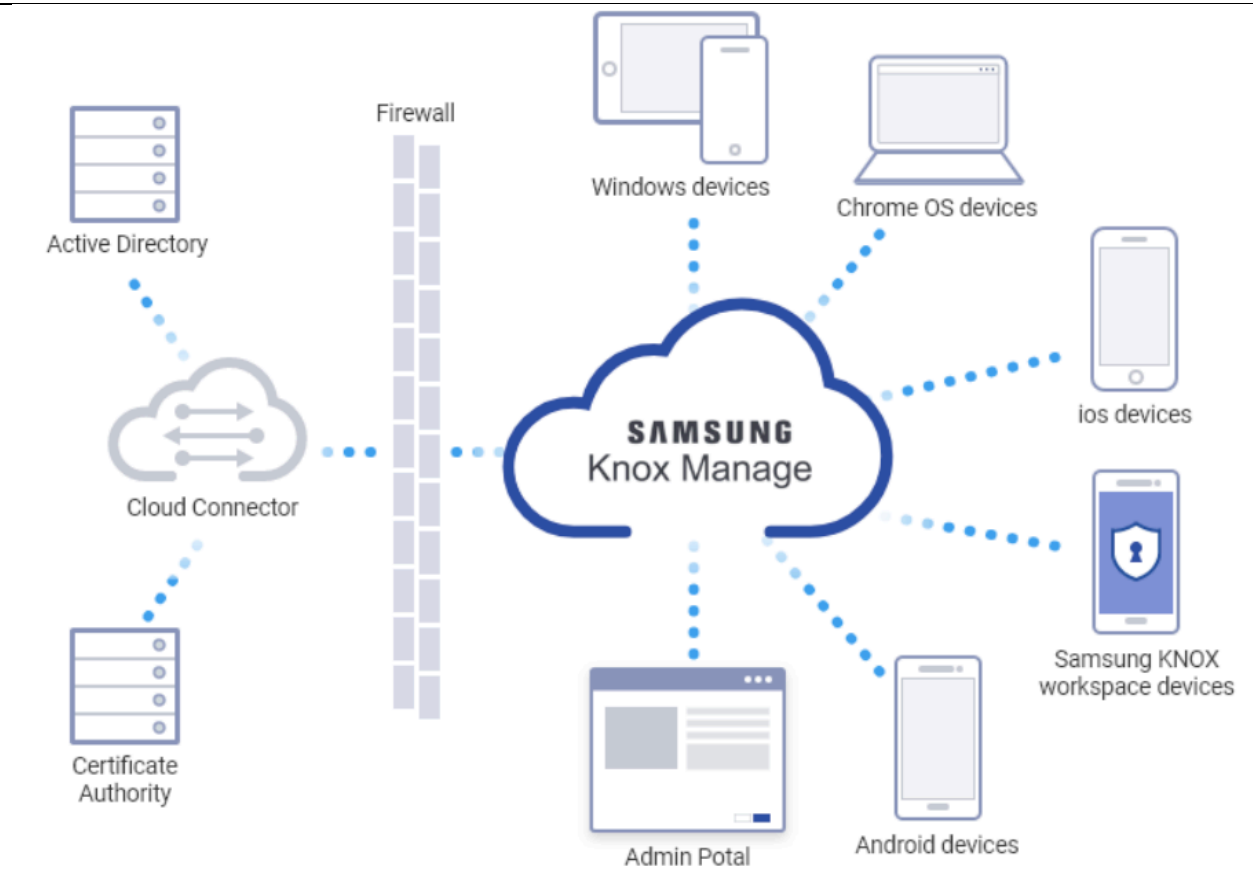


Step 4. Enable "Show notifications" and then enable the types of notifications you want to receive.

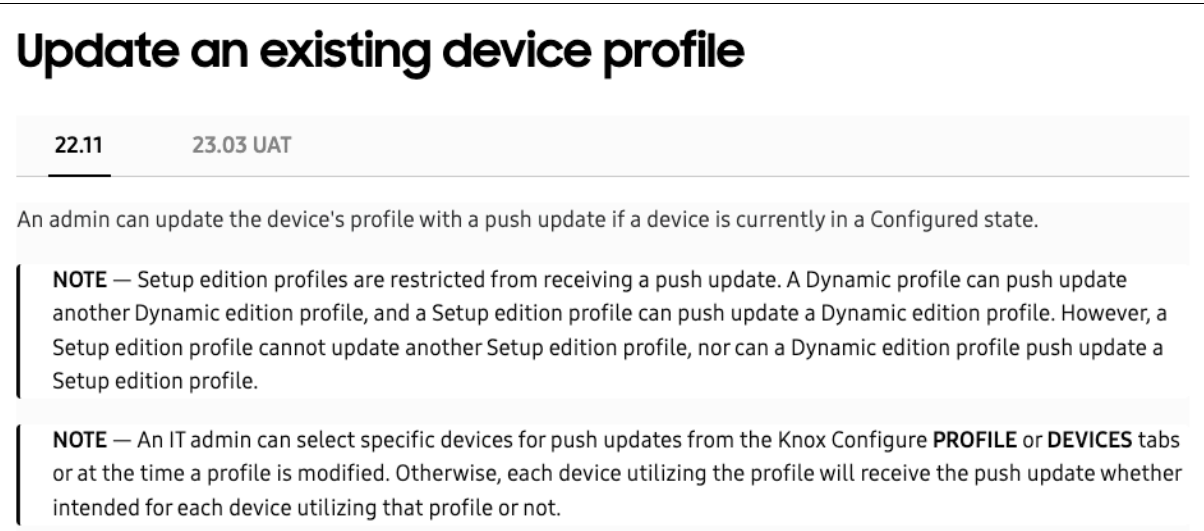
https://www.samsung.com/hk_en/support/apps-services/how-do-i-enable-push-notifications-from-galaxy-store/.

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.

39

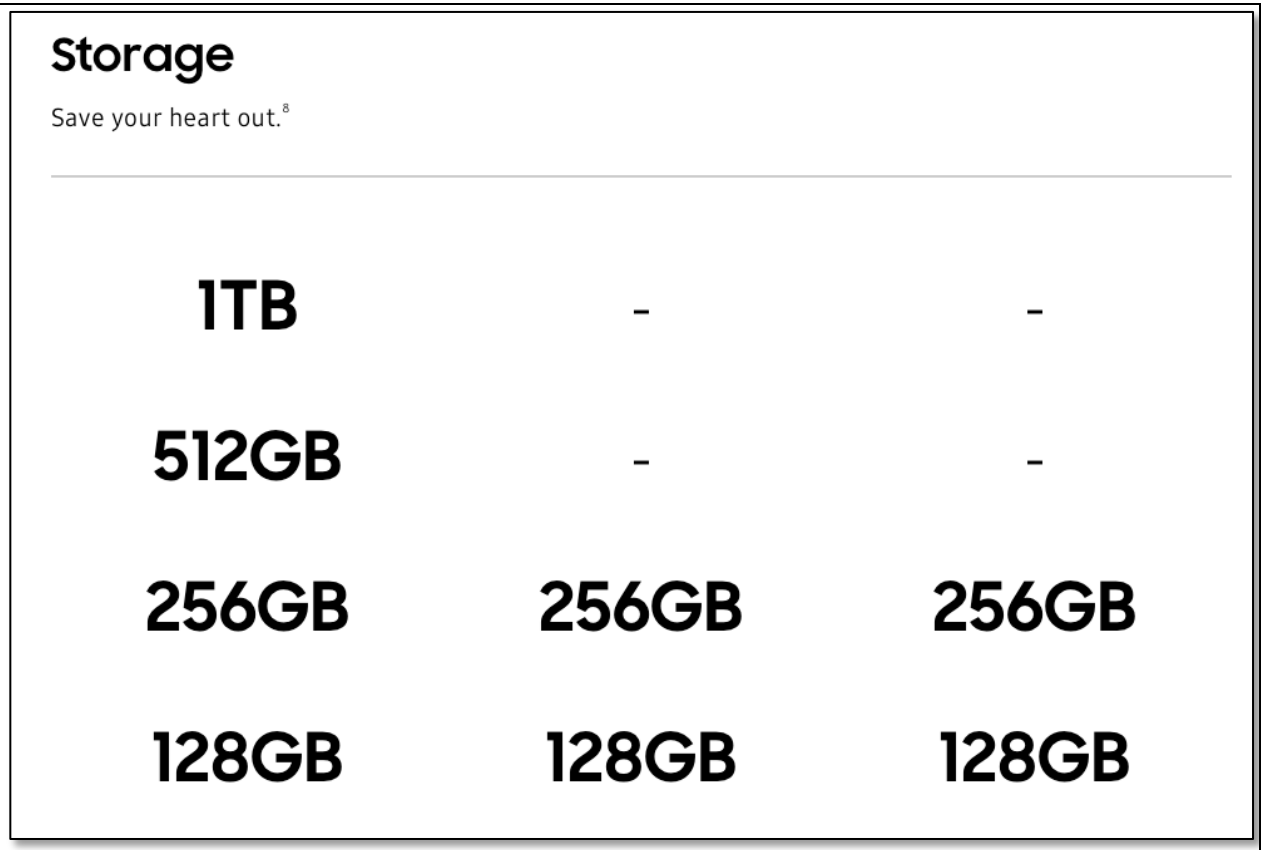


https://docs.samsungknox.com/admin/knox-manage/welcome.htm

	 https://docs.samsungknox.com/admin/knox-configure/updating-an-existing-device-profile.htm
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 that is shared between the agent and the network system. *See, e.g.*,

40

41



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

42

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

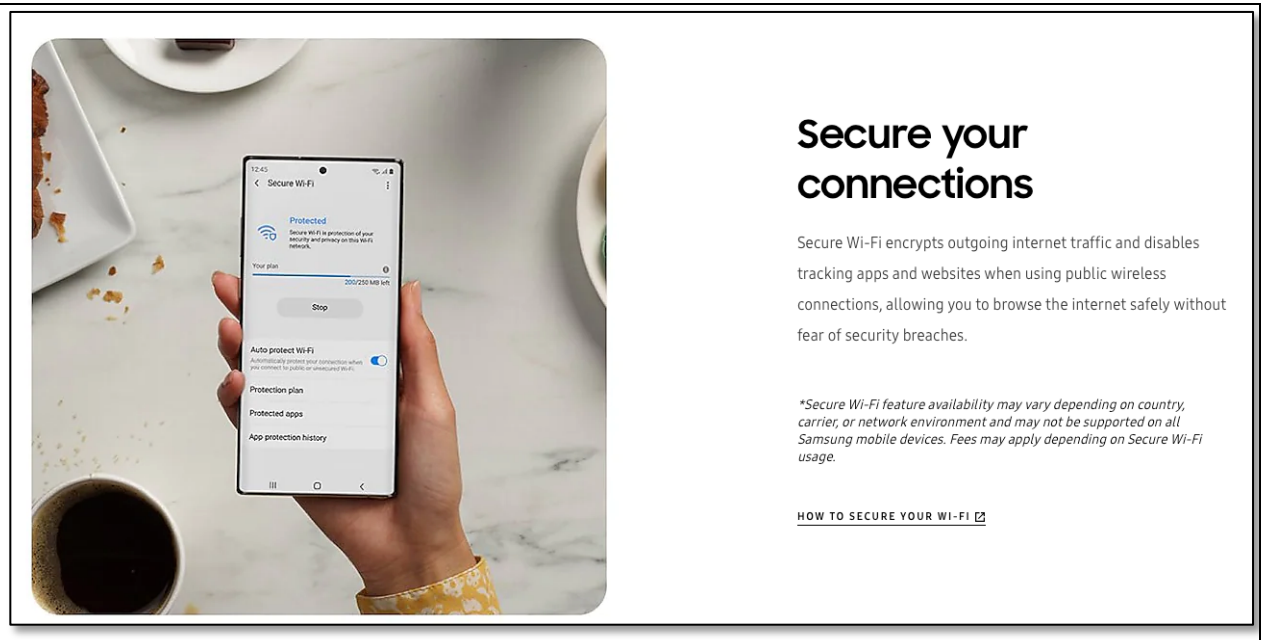
Data isolation is built into every component of our mobile devices. It's a double-lock for sensitive data, meaning the only person who ever has access is you.

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.

43	 https://www.samsung.com/us/security/;

44

Firebase > Documentation > FCM > Engage

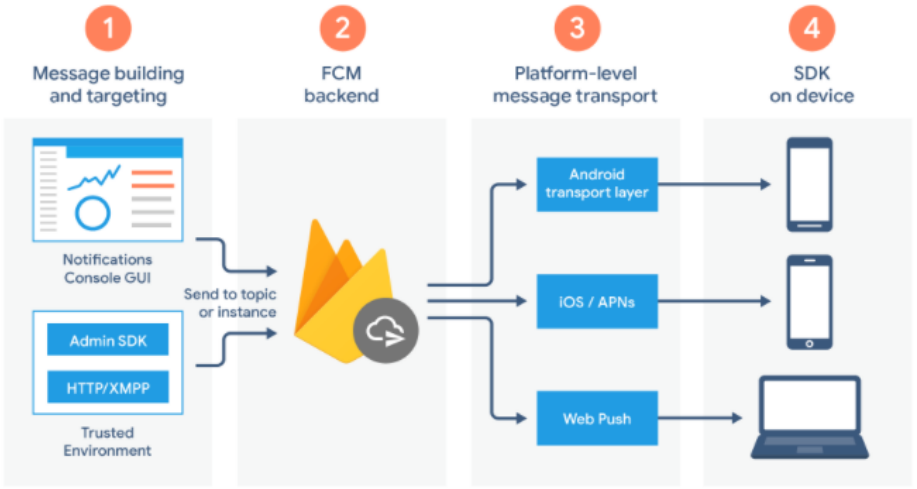
Was this helpful?

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.



45

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 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-qled-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, WPAPSK, WPA2PSK – Encryption Types: WEP, TKIP, AES https://downloadcenter.samsung.com/content/UM/202203/20220303092001587/OSNATSCB-3.2.0_EM_Oscar_Pontus_Nike_Kant-SU2e_USA_ENG_220216.0.pdf;

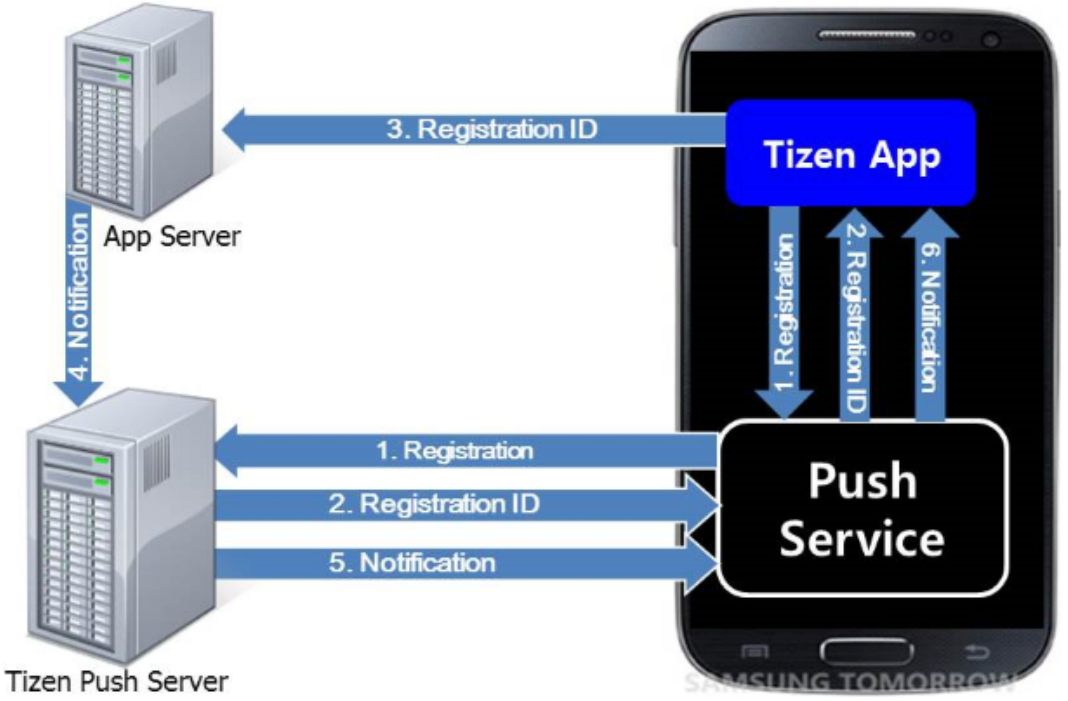
46

47

Service Architecture

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

Figure: Service architecture



48

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

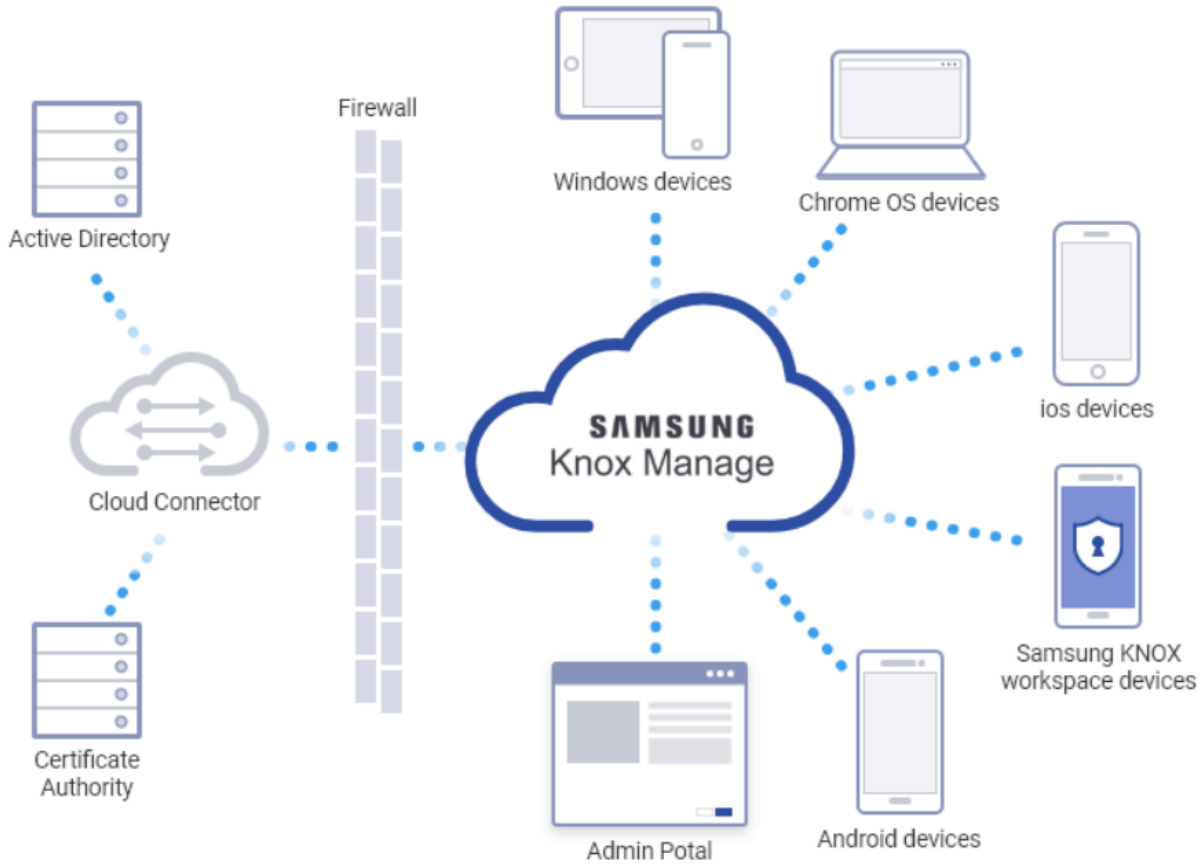
The security features introduce how private information remains private, and how the user knows when they are trying to access privileged information. You can use a repository and cryptographic operations to manage keys, certificates, and sensitive user data. When the user tries to access privileged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

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

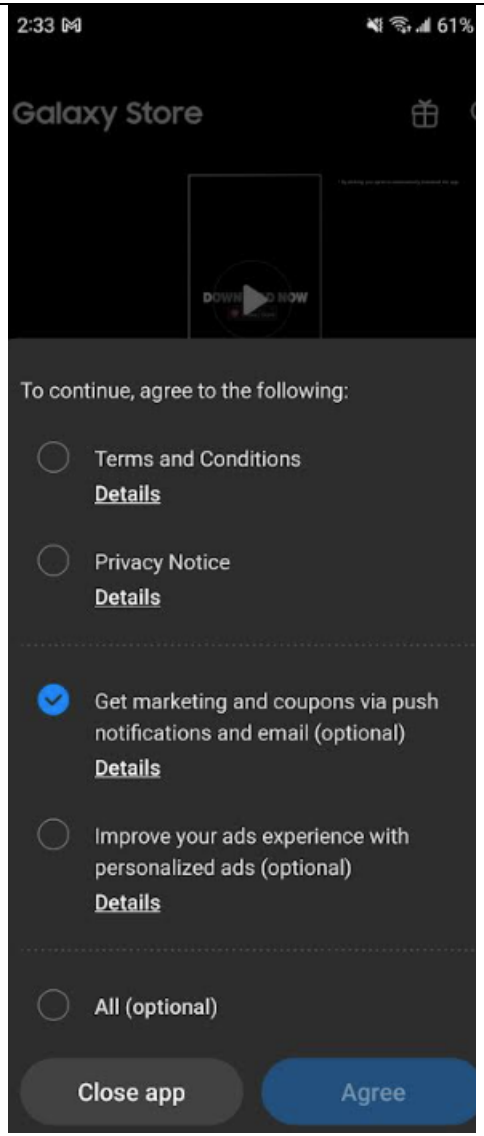
- Cryptographic Operations

  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.

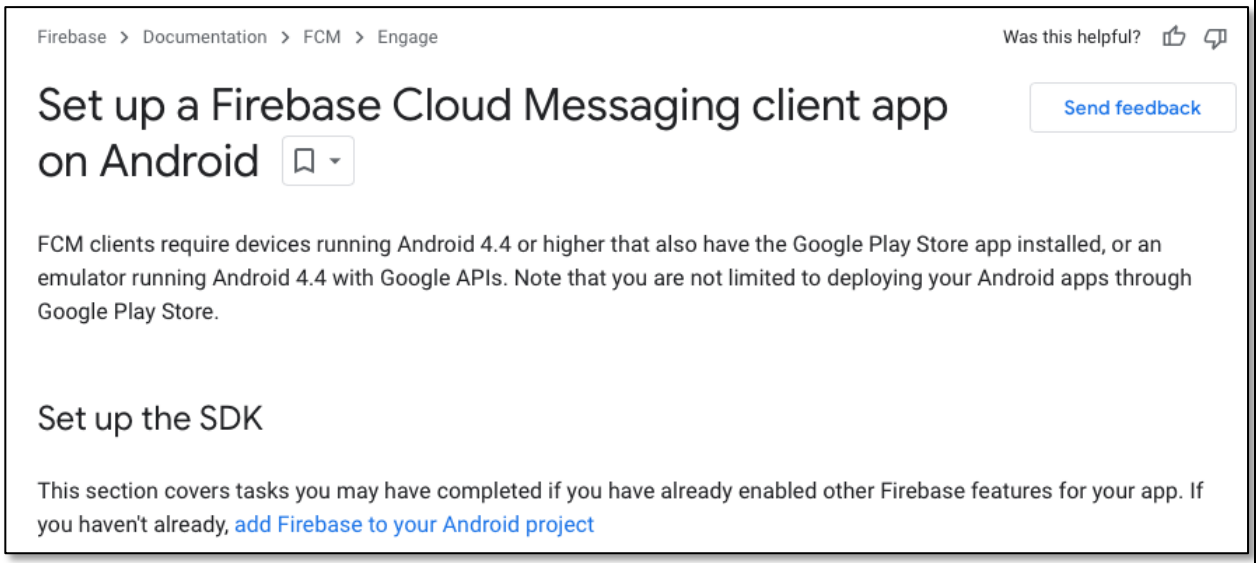
https://docs.tizen.org/application/native/guides/security/overview/.

49	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 that is shared between the agent and the network system.  https://docs.samsungknox.com/admin/knox-manage/welcome.htm

50	**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. *See, e.g.*, https://play.google.com/store/apps/details?id=com.sec.spp.push&hl=en_US&gl=US&pli=1; https://www.lifewire.com/samsung-push-service-4165507. On information and belief, Samsung includes applications in its user devices, such as the "Shop Samsung" and "Galaxy Store" applications, which cause notifications (including marketing notifications) to be sent to user devices via Samsung's push messaging servers. *See, e.g.*:

	
1[d] wherein the service control device link agent is configured to: receive, over the service control link, an encrypted	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."

agent message from the service control server link element,	For example, Samsung Galaxy phones and tablets receive encrypted messages from a server. *See e.g.*,



Firebase > Documentation > FCM > Engage

Was this helpful?

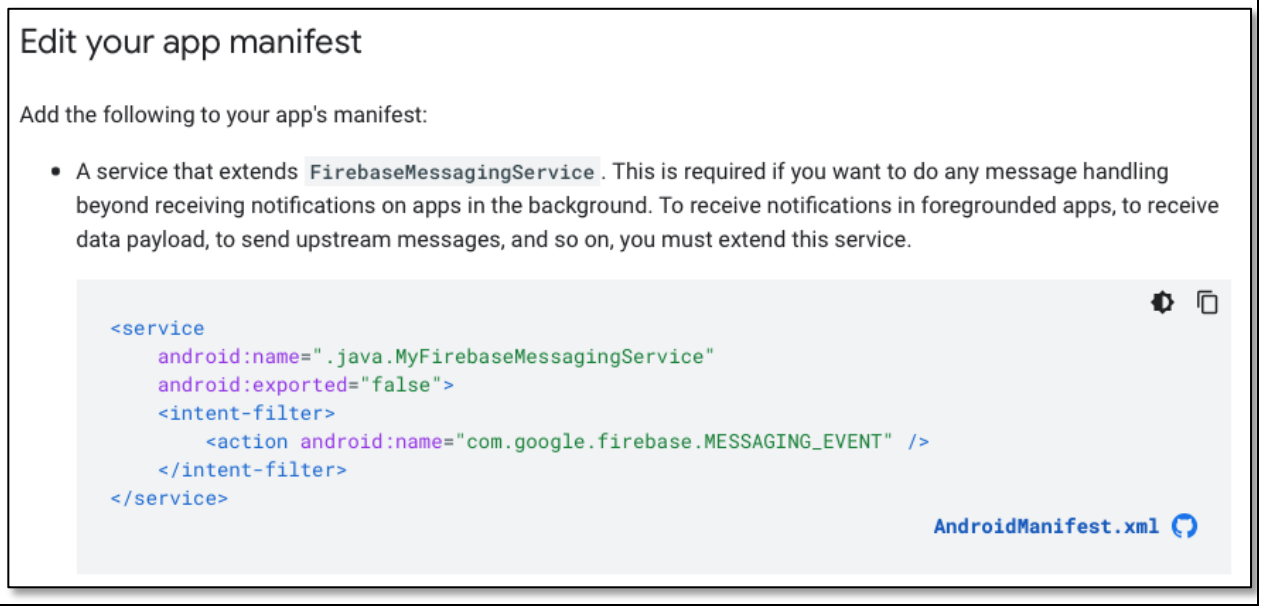
Set up a Firebase Cloud Messaging client app on Android

Send feedback

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

Set up the SDK

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



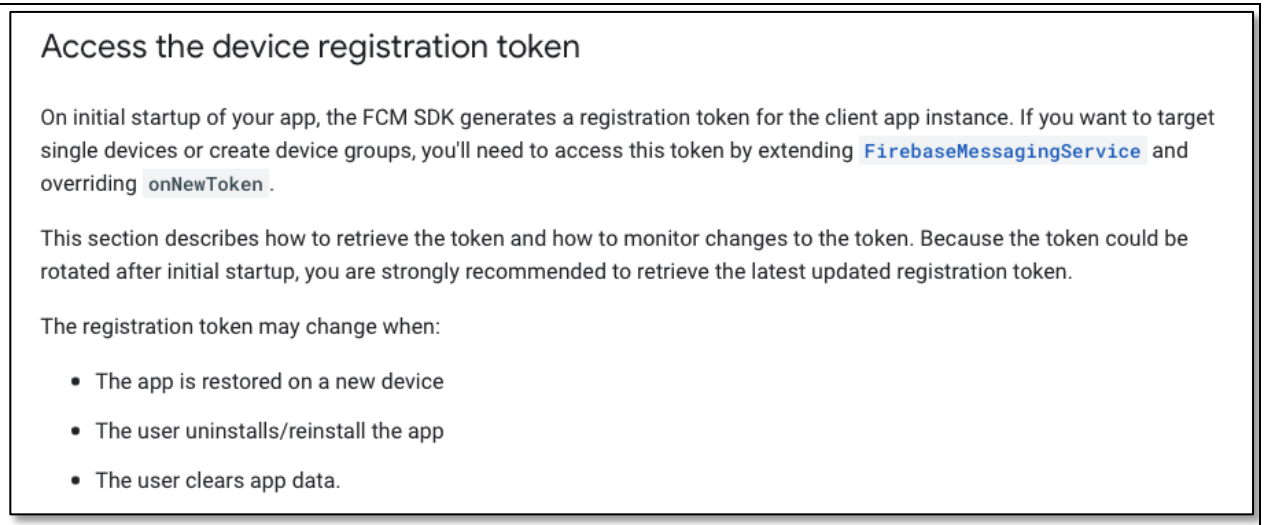
Edit your app manifest

Add the following to your app's manifest:

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

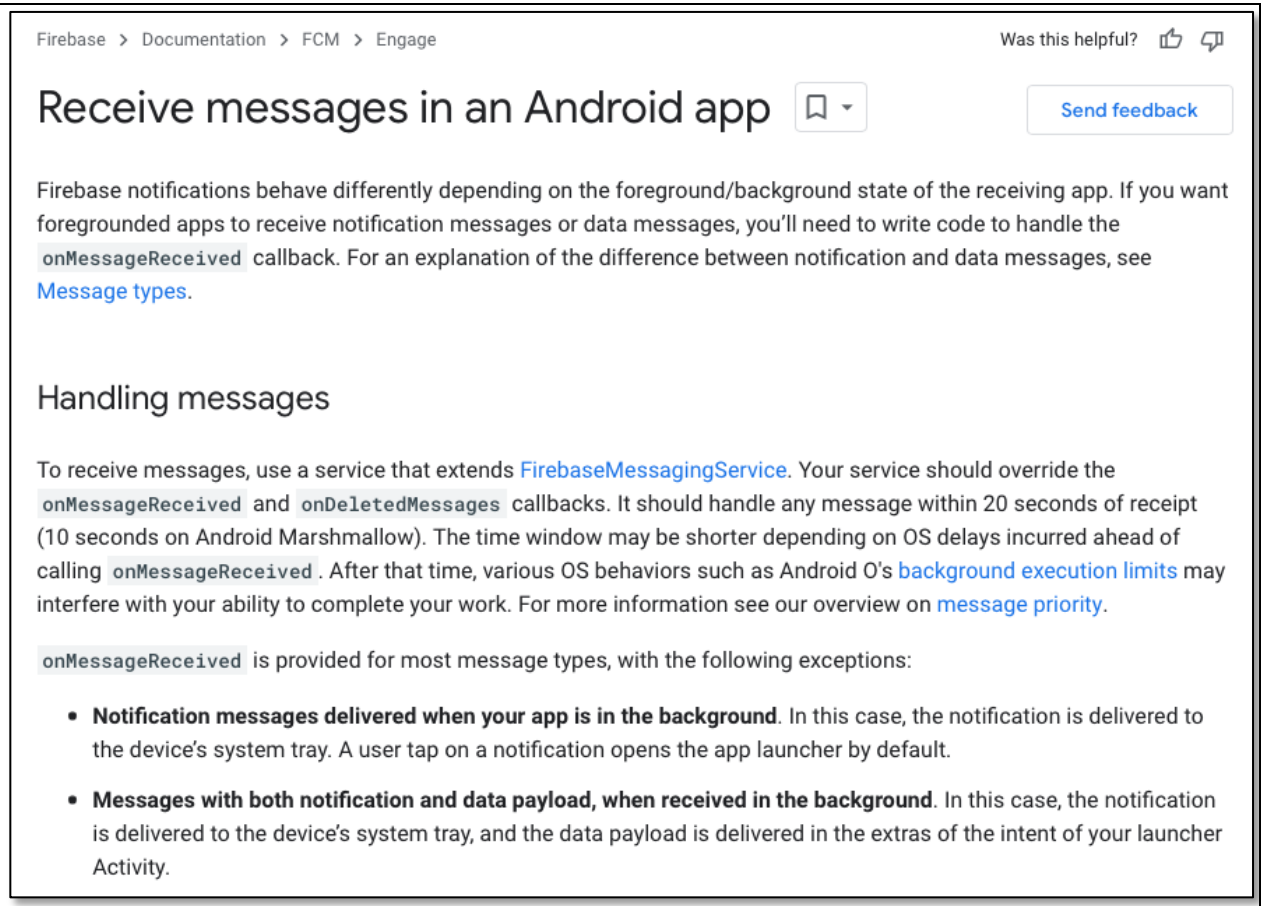
```
<service
    android:name=".java.MyFirebaseMessagingService"
    android:exported="false">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT" />
    </intent-filter>
</service>
```

AndroidManifest.xml

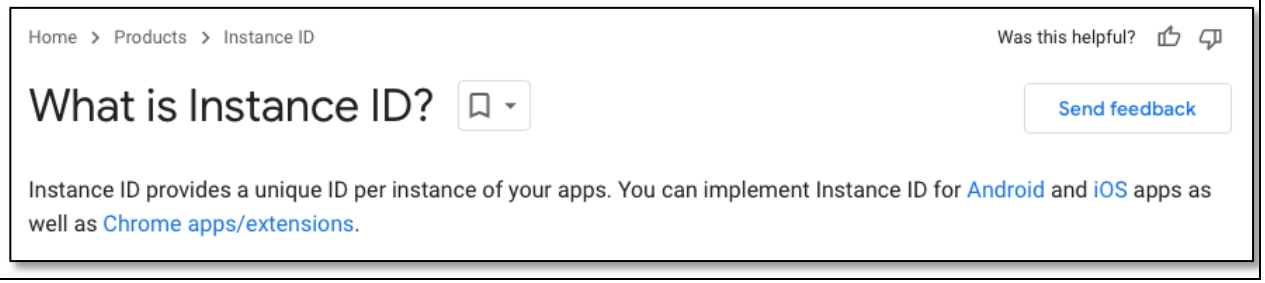
	 https://firebase.google.com/docs/cloud-messaging/android/client;

53

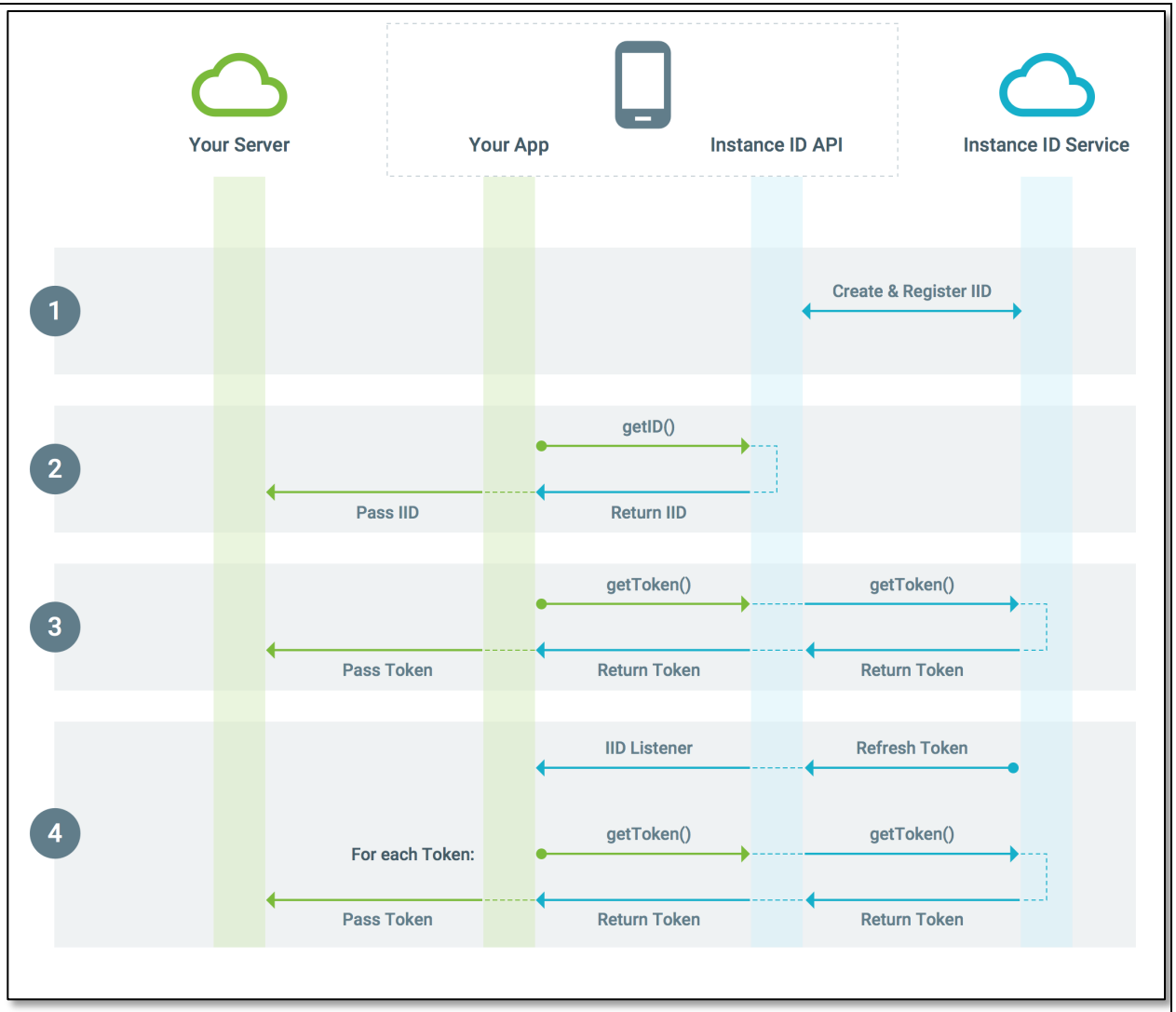
54



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



55



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

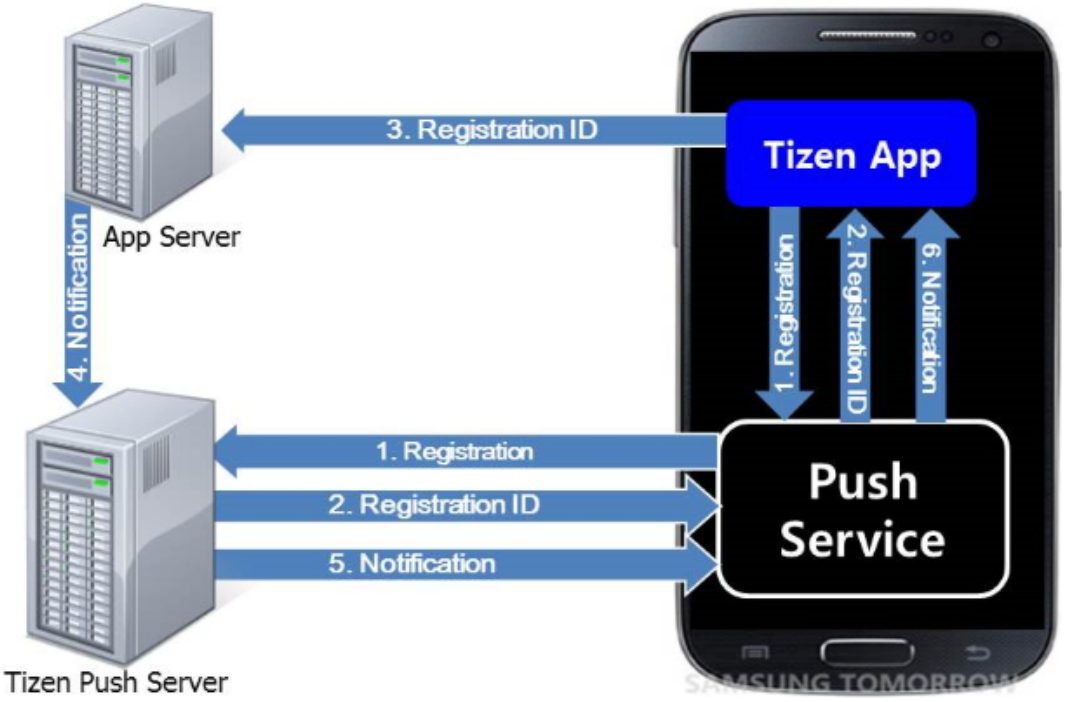
For further example, Samsung's Tizen based devices receive encrypted messages from a server. *See e.g.,*

56

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.

57

58

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);
```

59

- 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);
}
```

61

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.

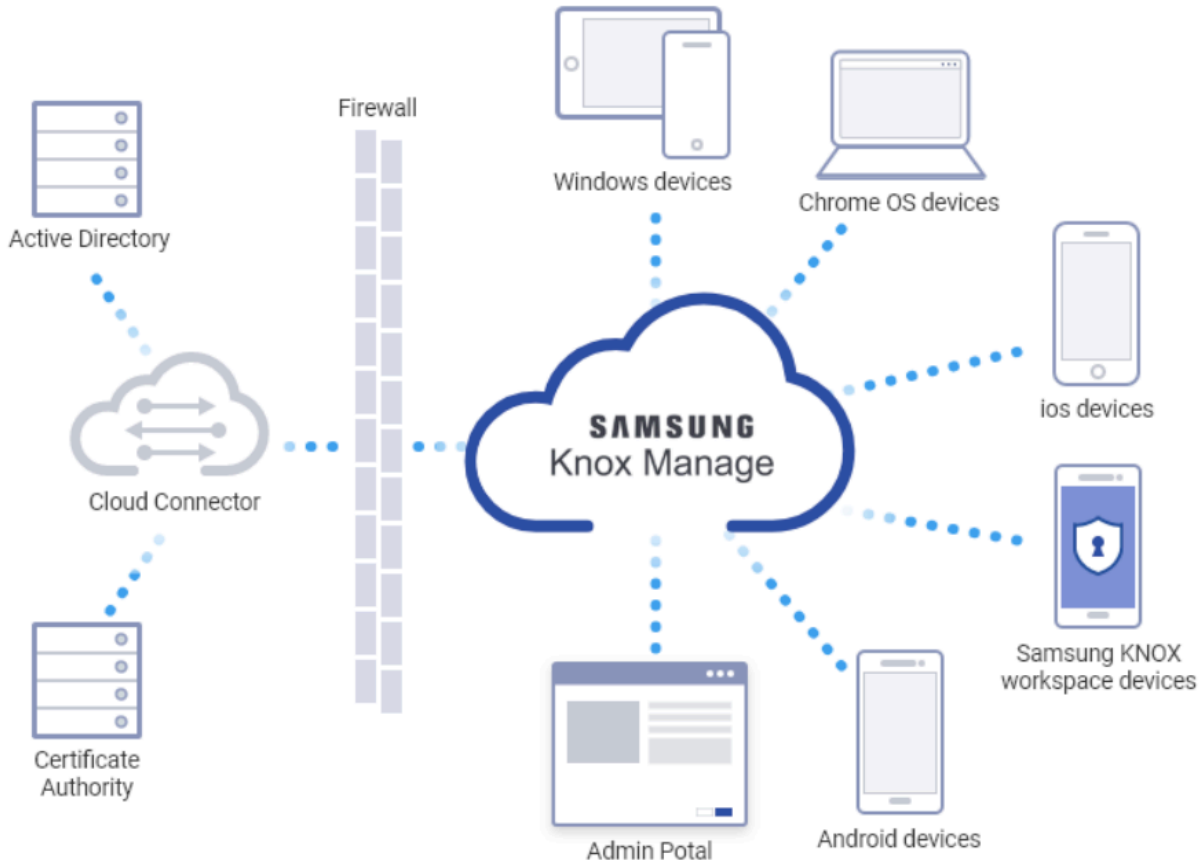
    For more information, see the Retrieving Missed Push Messages use case.

    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/.

62	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

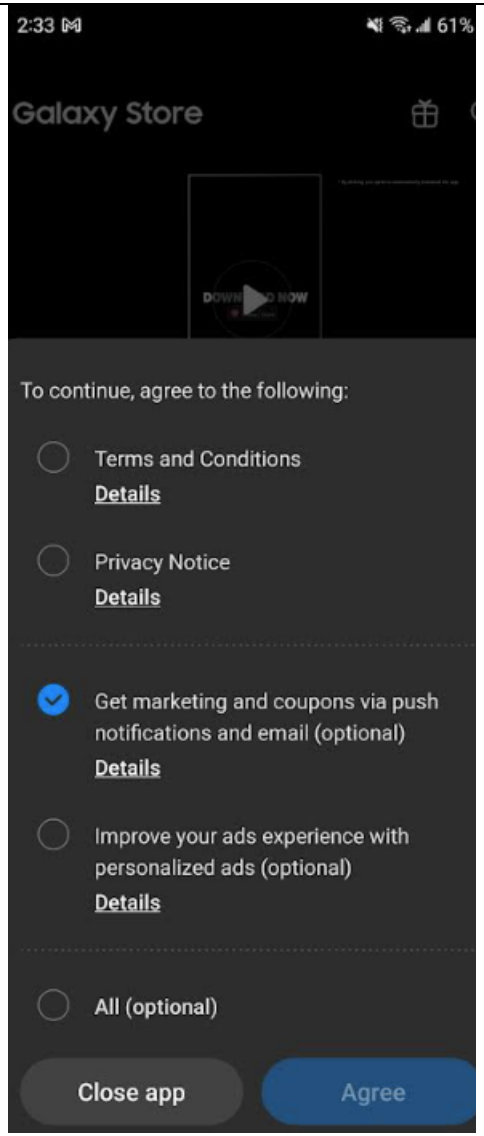
63

Components of Knox Manage

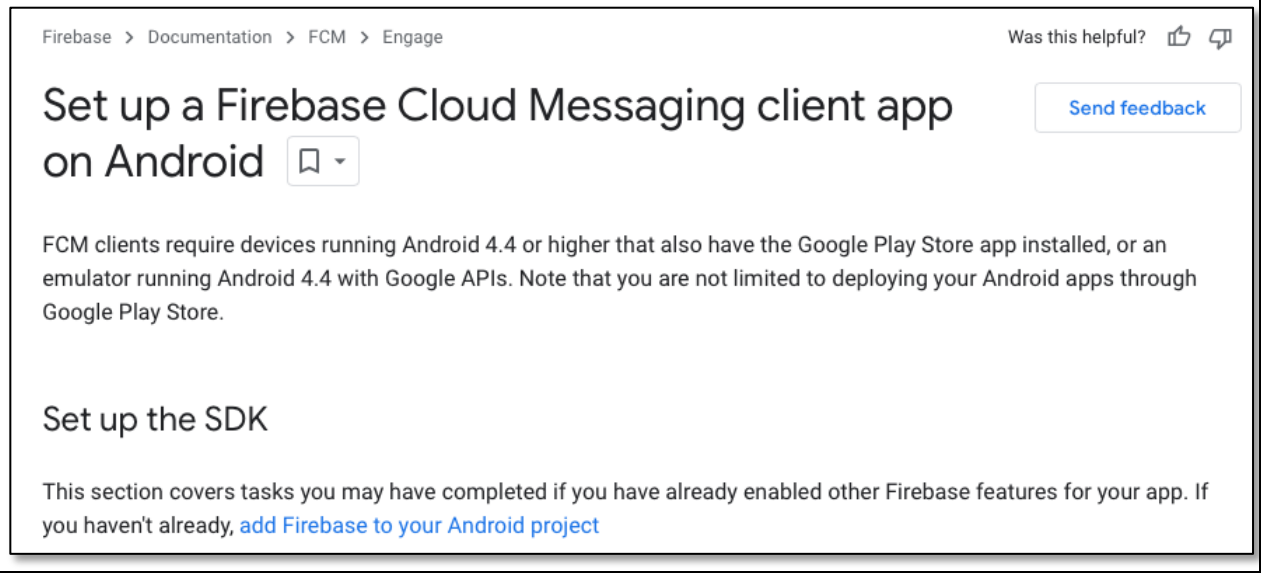
1. **Knox Manage console** — A web console that allows IT admins to configure, monitor, and manage devices, deploy updates, as well as manage certificates and licenses.
2. **Knox Manage MDM Client** — An app that is installed on devices to automate installation and enrollment to Knox Manage.
3. **Knox 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. *See, e.g.*, https://play.google.com/store/apps/details?id=com.sec.spp.push&hl=en_US&gl=US&pli=1; https://www.lifewire.com/samsung-push-service-4165507. On information and belief, Samsung includes applications in its user devices, such as the "Shop Samsung" and "Galaxy Store" applications, which cause notifications (including marketing notifications) to be sent to user devices via Samsung's push messaging servers. *See, e.g.*:

		
1[e] using the encryption key, obtain a decrypted agent message, the decrypted agent message comprising a particular agent identifier and message content for	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 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	

64

65	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	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.*,  Firebase > Documentation > FCM > Engage Was this helpful? Set up a Firebase Cloud Messaging client app on Android Send feedback FCM clients require devices running Android 4.4 or higher that also have the Google Play Store app installed, or an emulator running Android 4.4 with Google APIs. Note that you are not limited to deploying your Android apps through Google Play Store. Set up the SDK This section covers tasks you may have completed if you have already enabled other Firebase features for your app. If you haven't already, add Firebase to your Android project

99

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;

67

Firebase > Documentation > FCM > Engage

Was this helpful?

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;

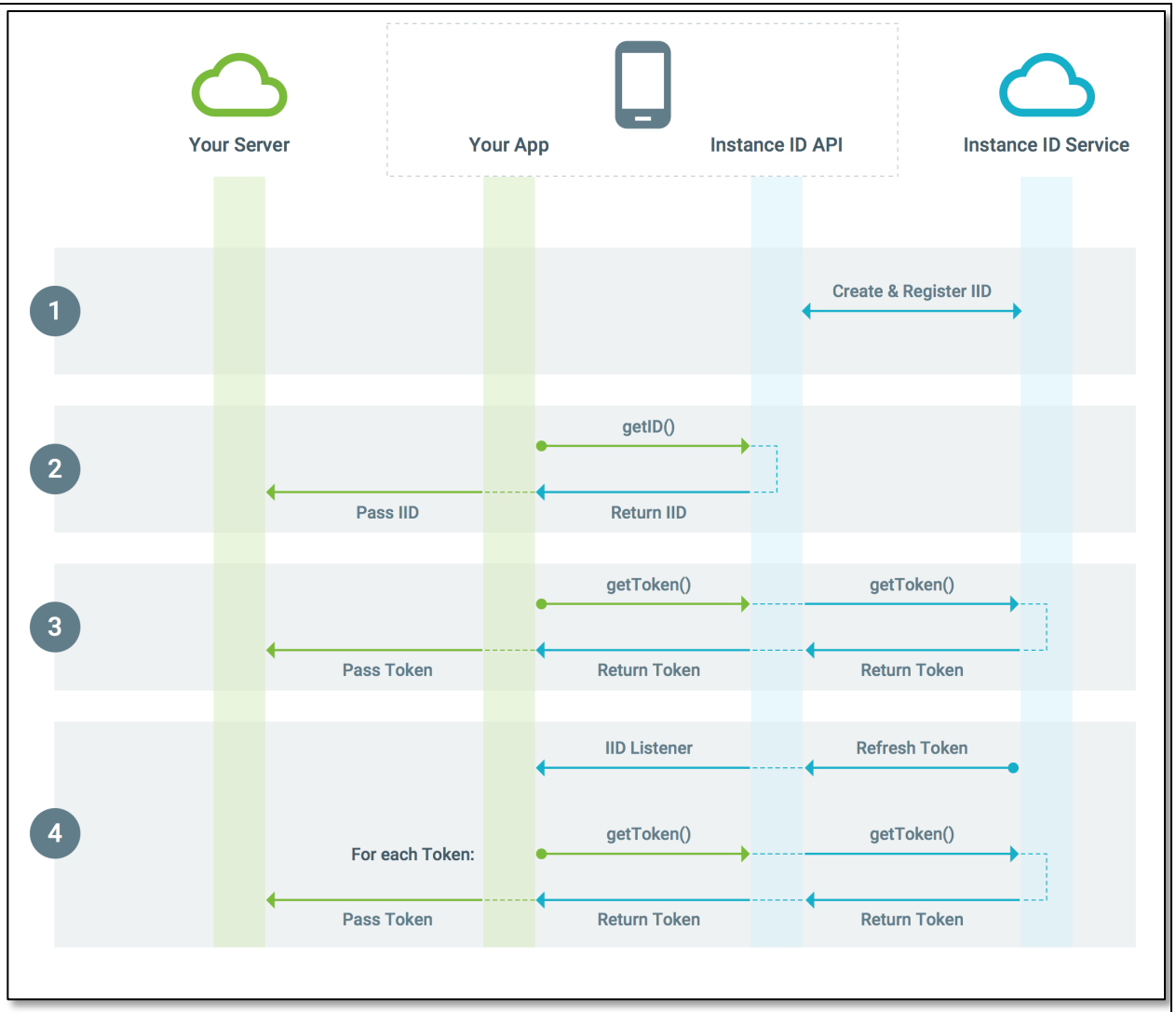
Home > Products > Instance ID

Was this helpful?

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.

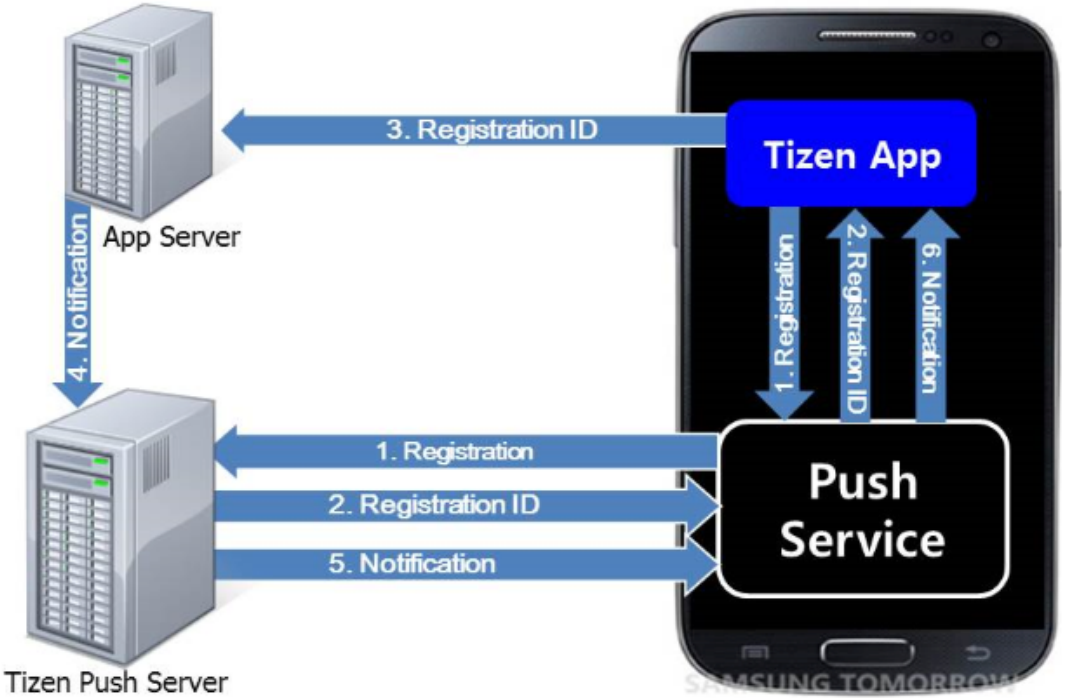
89	 https://developers.google.com/instance-id. For further example, Samsung's Tizen based devices receive encrypted messages from a server. *See e.g.,*

69

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.

70

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.

71

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);
```

72

- 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);
}
```

74

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.

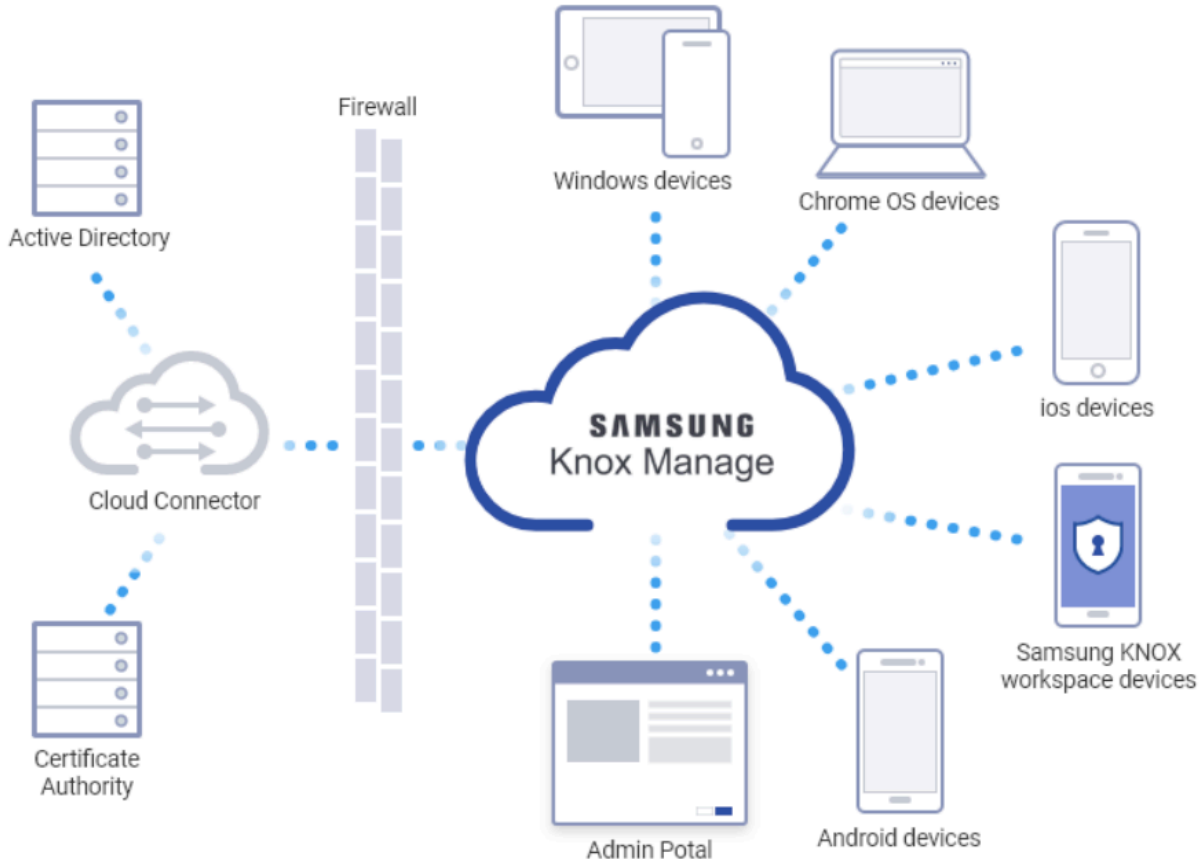
    The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification.

  - 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.

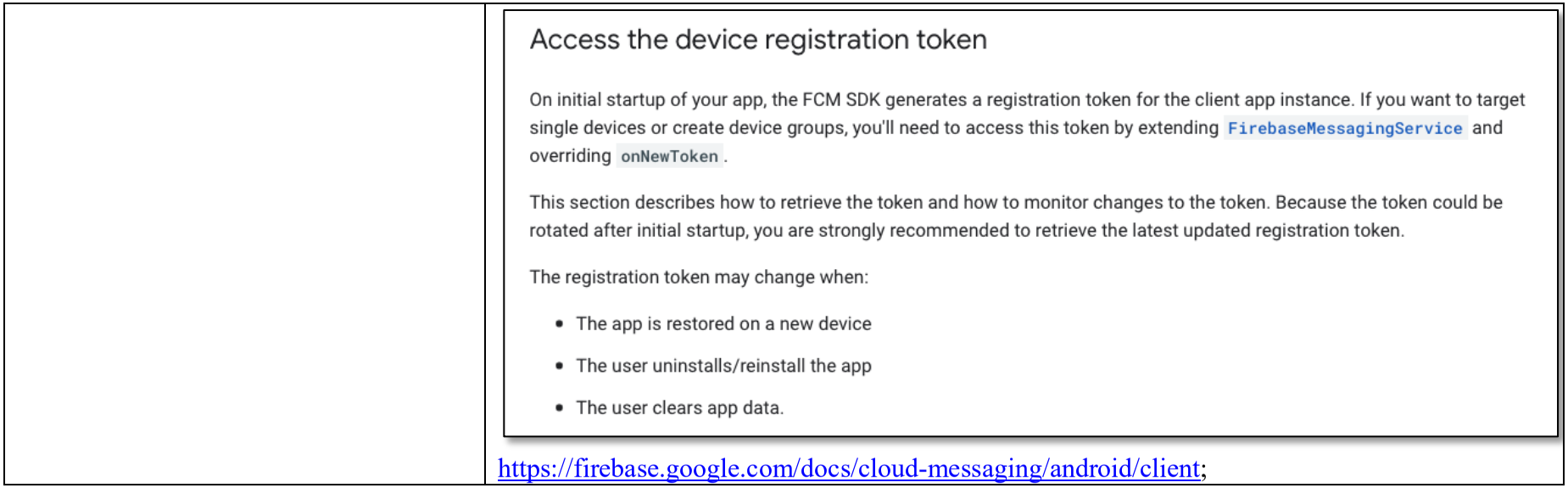
https://docs.tizen.org/application/web/guides/messaging/push/.

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

75	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. *See e.g.*,

76



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

77

78

Firebase > Documentation > FCM > Engage

Was this helpful?

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;

Home > Products > Instance ID

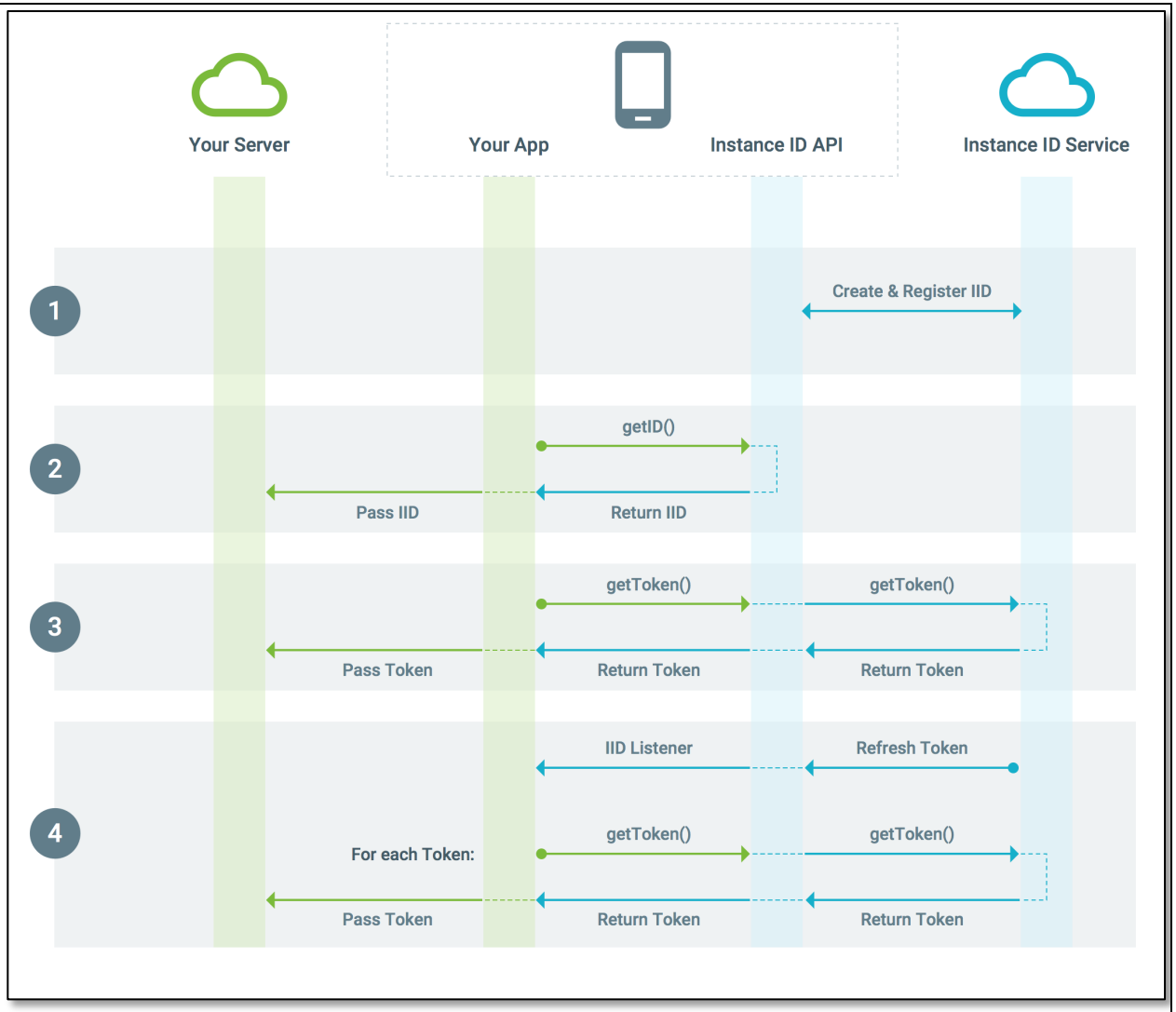
Was this helpful?

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.

79



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

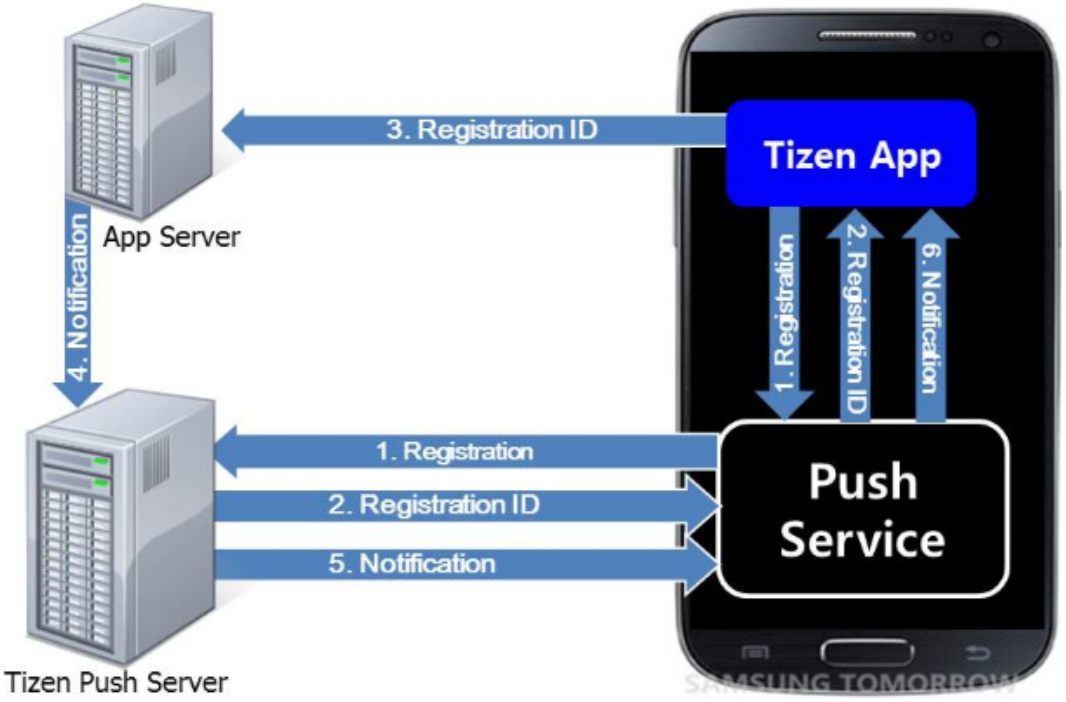
For further example, Samsung's Tizen based devices deliver message content to particular device agents based on identifier. *See e.g.,*

80

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.

81

82

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);
```

83

- 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);
}
```

85

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.

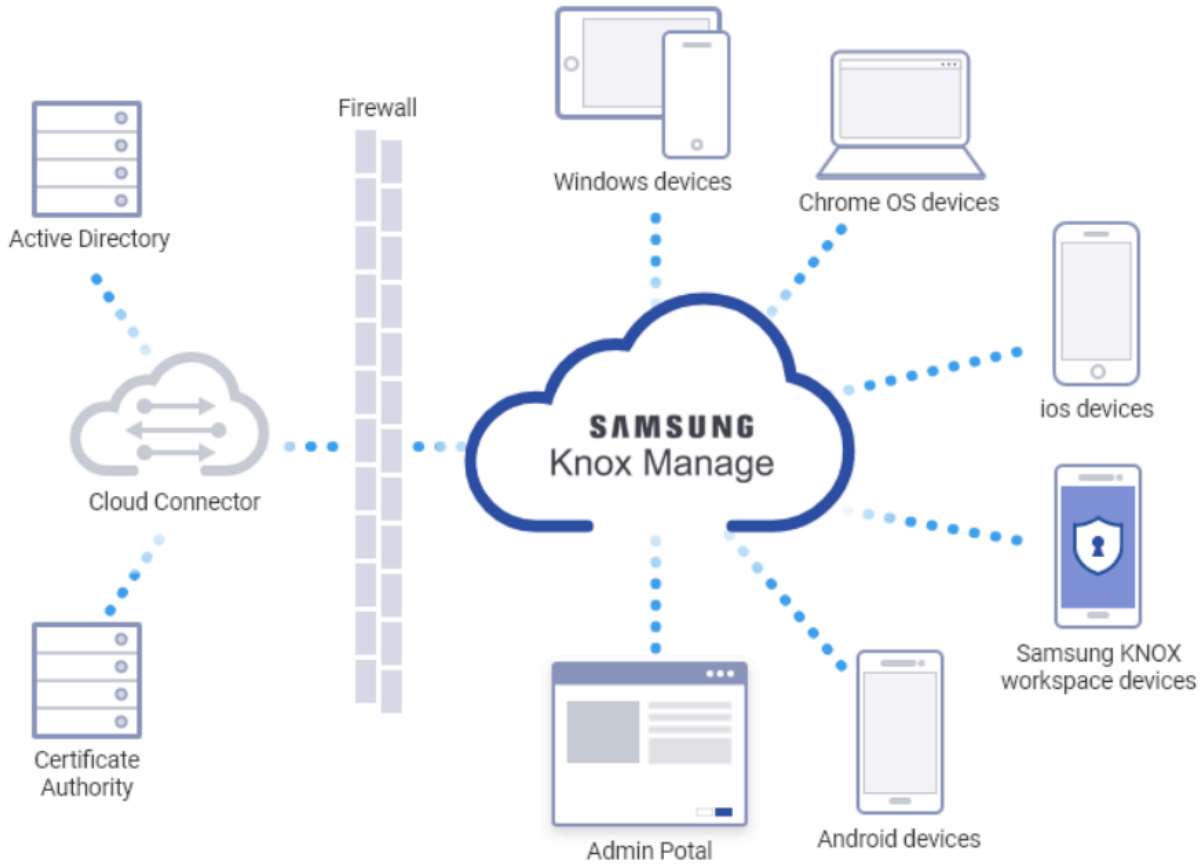
    For more information, see the Retrieving Missed Push Messages use case.

    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/.

98	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

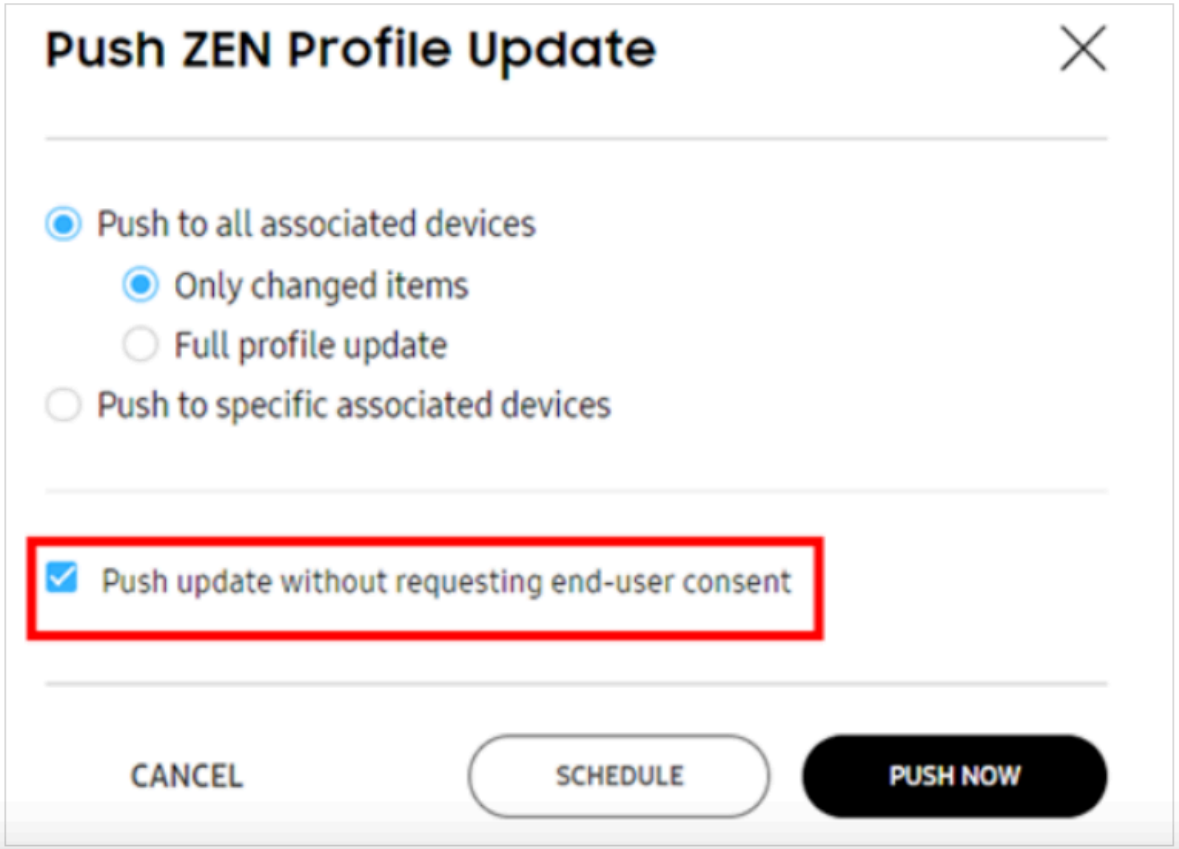
87

88

Push an update

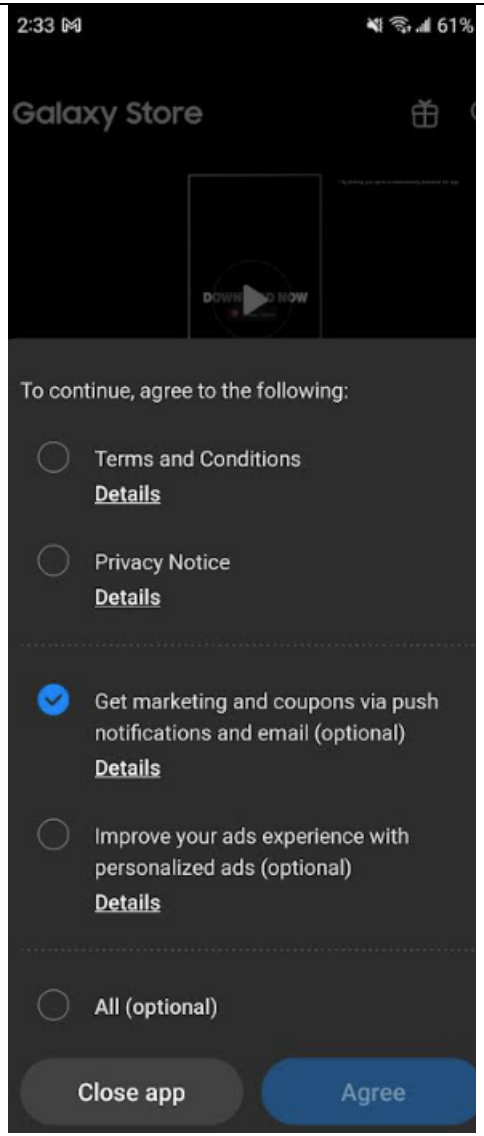
To push an update to an existing profile:

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 assi without consent from the device end user.



	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. *See, e.g.*, https://play.google.com/store/apps/details?id=com.sec.spp.push&hl=en_US&gl=US&pli=1; https://www.lifewire.com/samsung-push-service-4165507. On information and belief, Samsung includes applications in its user devices, such as the "Shop Samsung" and "Galaxy Store" applications, which cause notifications (including marketing notifications) to be sent to user devices via Samsung's push messaging servers. *See, e.g.*:

68

06	 2:33 61% Galaxy Store To continue, agree to the following: Terms and Conditions Details Privacy Notice Details Get marketing and coupons via push notifications and email (optional) Details Improve your ads experience with personalized ads (optional) Details All (optional) Close app Agree	

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.

Claim 3

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.

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 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.	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.

93

Claim 5

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."

	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 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.

94

Claim 7

7. The end-user device recited in claim 1, wherein the message content comprises a service offer, an advertisement, or a transaction offer.	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." *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.")).

95

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." *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 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. *See, e.g.,* 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. *See, e.g.,* https://docs.samsungknox.com/admin/knox-manage/configure/applications/add-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." *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, 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/ (sending 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").

Claim 10

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." *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, may receive message content comprising software. *See, e.g.*, 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/manage-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." *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

97

	devices enrolled in the Samsung Knox MDM platform, message content received by such devices may comprise information associated with a service policy. 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 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.

86 **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

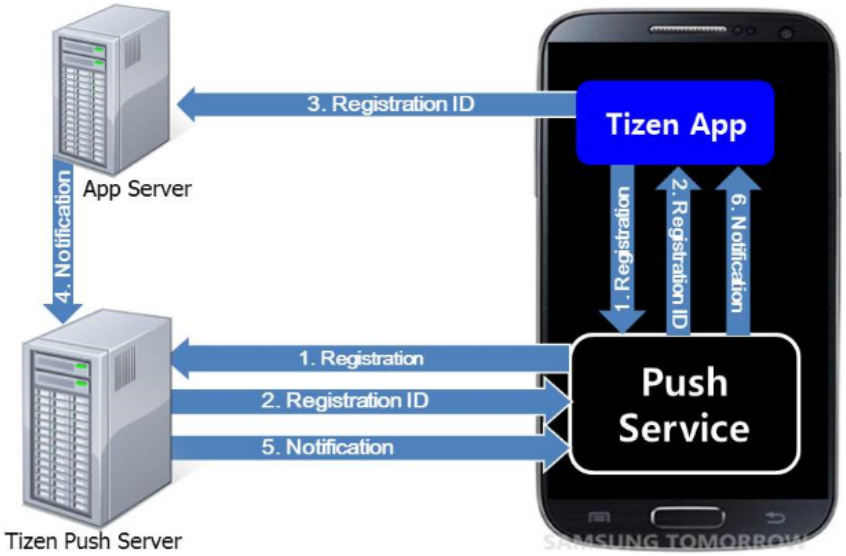
13. The end-user device recited in claim 1, wherein the service control device link agent is further configured to send a device message to the service control server link element over the service control link.	Samsung Galaxy phones and tablets and Samsung's Tizen based devices comprise the "end-user 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." *See supra* 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; *see below*) over a service control link.

100

Architecture

The architecture of the Tizen Push service is described in detail in the mobile native Push guide.

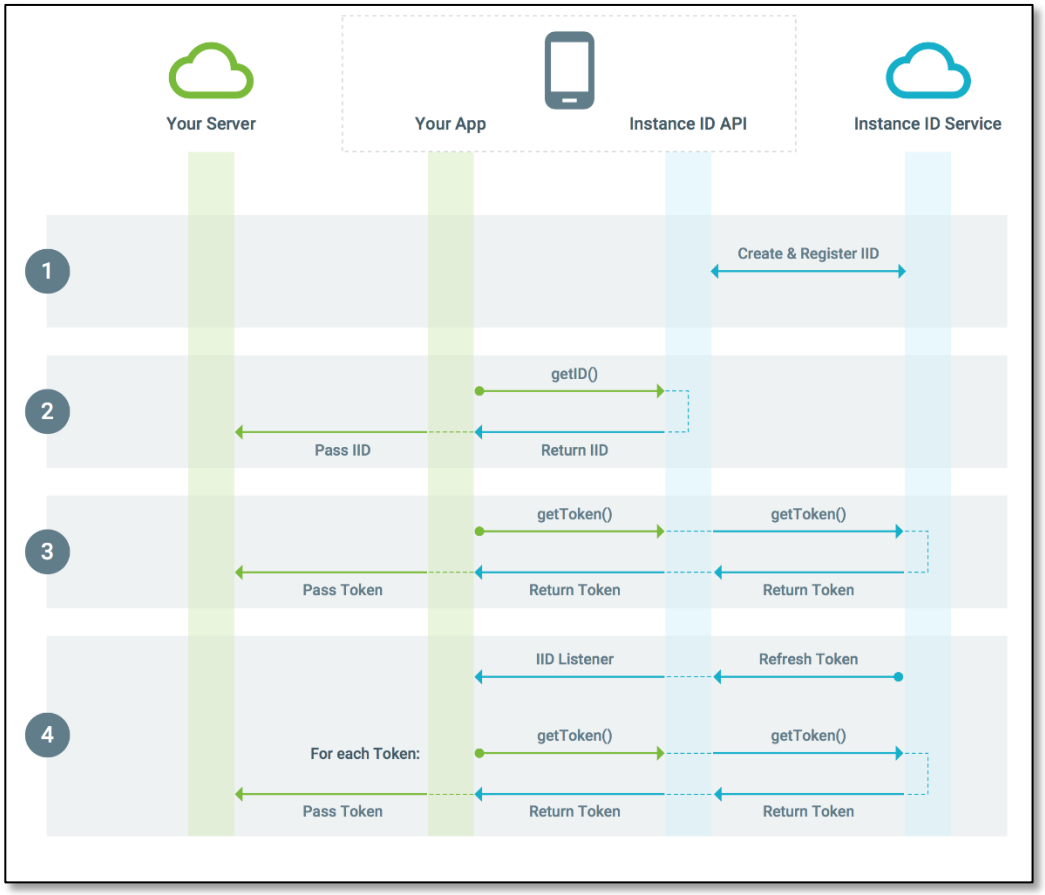
Figure: Service architecture



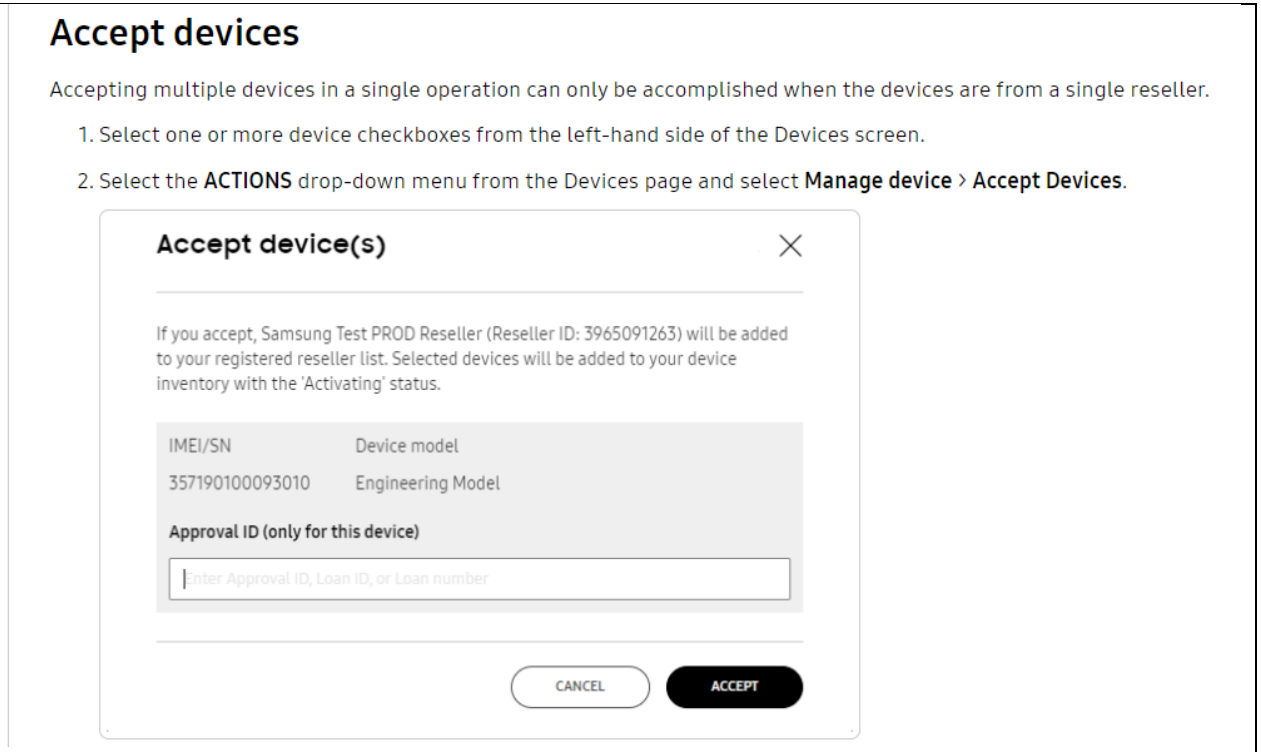
To receive push notifications for your application:

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.

101



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.*:

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

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 supra* claim 1. See, for example:

103

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/

104

App usage

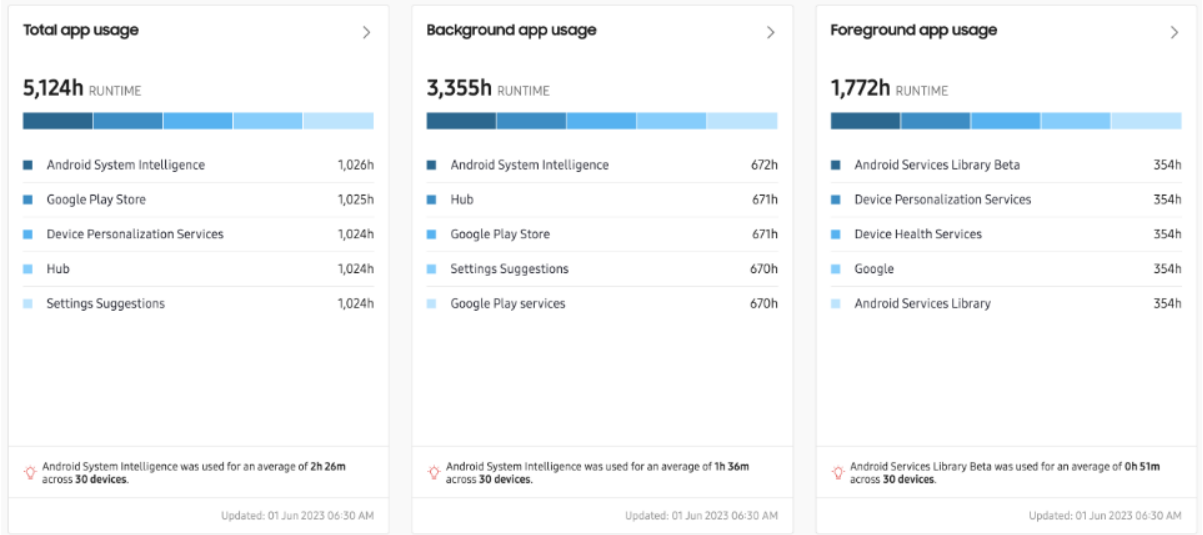
Last updated July 26th, 2023

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/

Claim 19

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." *See supra* 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.

Claim 21

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." *See supra* 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. *See, e.g.*, 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.

105

106

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();
```

107

- 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

108

Claim 22

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." *See supra* 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. *See supra* 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

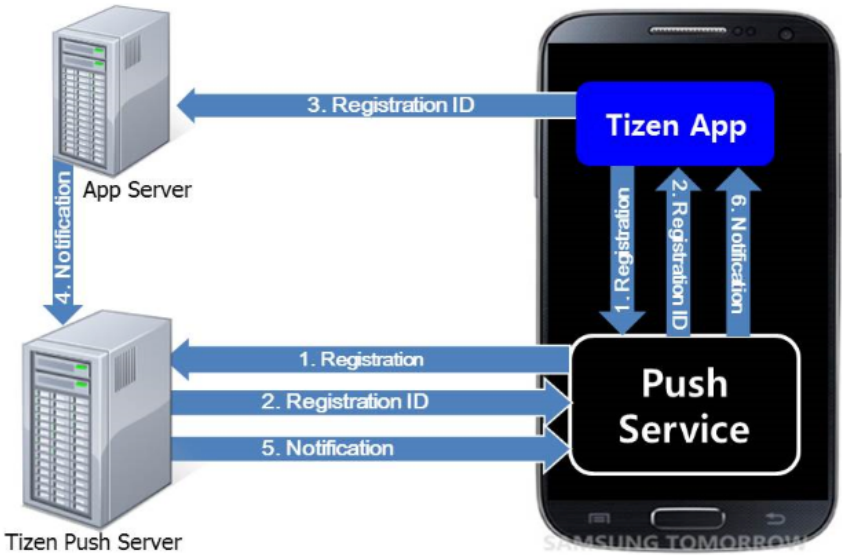
device credential to the network system or receive the device credential from the network system during a service authorization sequence.	send a device credential to the network system or receive the device credential from the network system during a service authorization sequence." *See supra* claims 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; *see below*).

110

Architecture

The architecture of the Tizen Push service is described in detail in the mobile native Push guide.

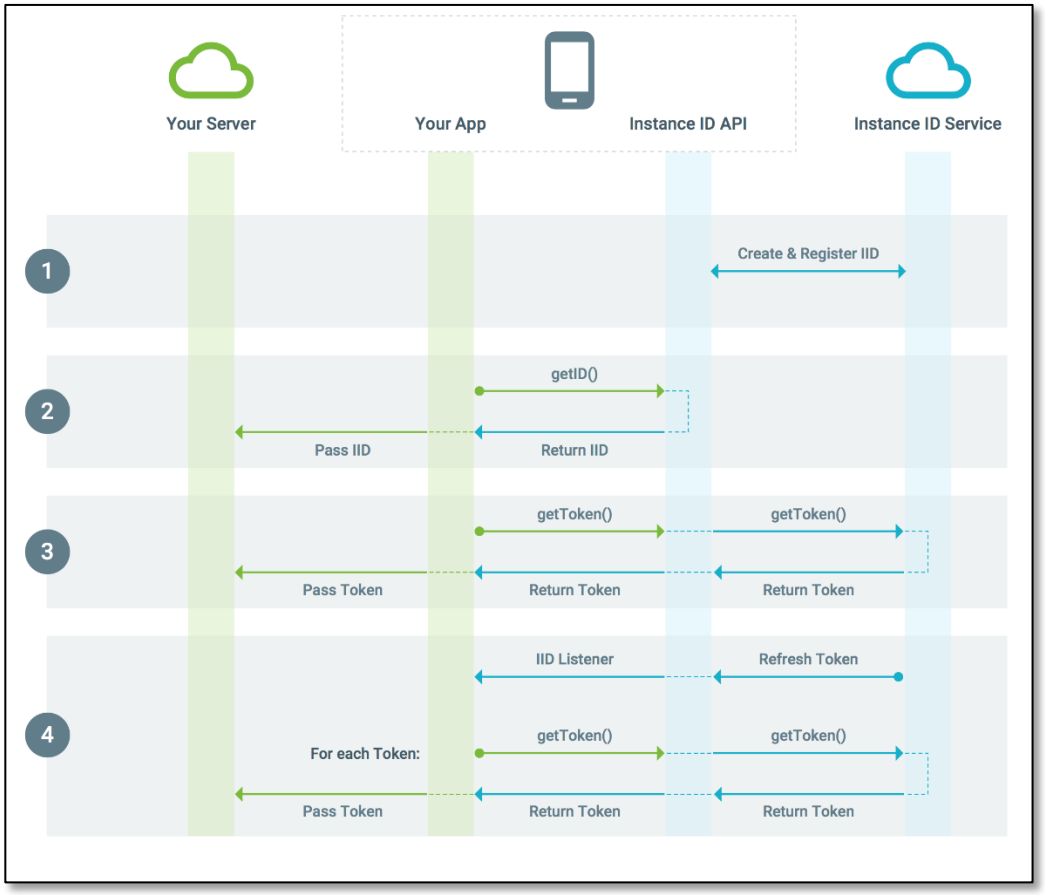
Figure: Service architecture



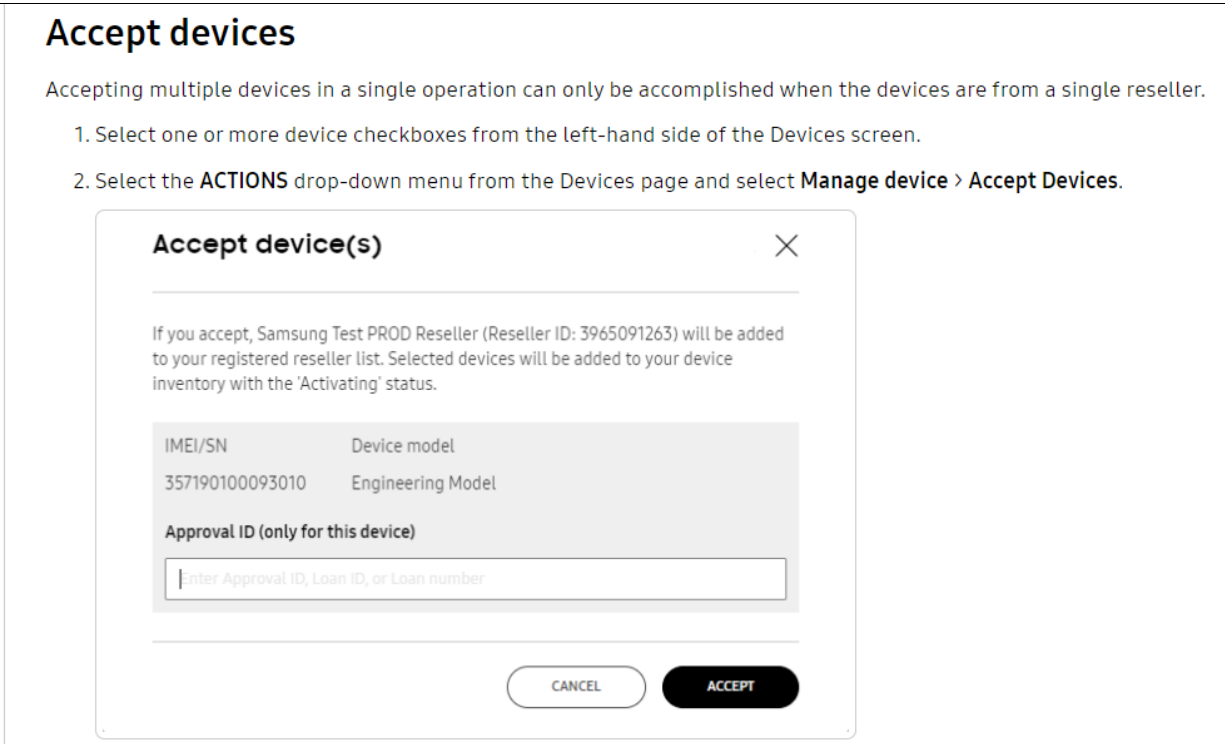
To receive push notifications for your application:

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.

111



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.*:

112	 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 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

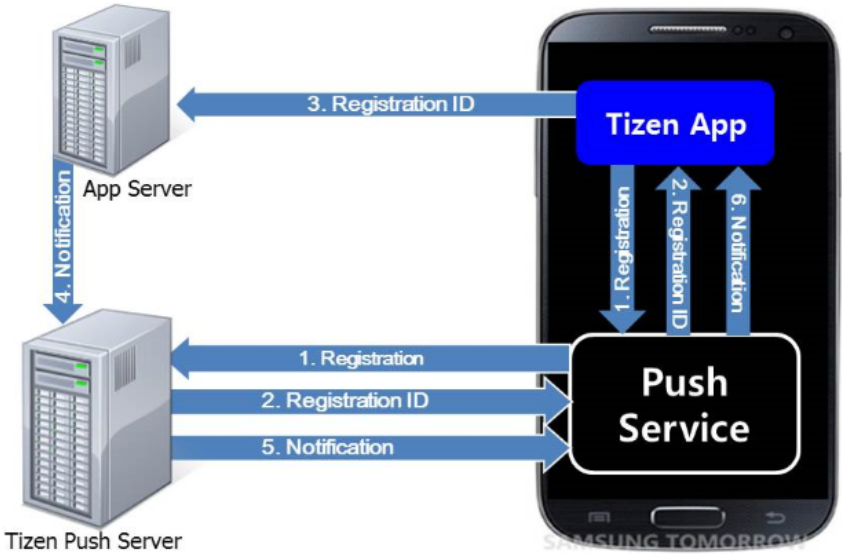
comprises one or more of a phone number, an identification number, a security signature, a security credential, a subscriber identity module (SIM) identifier, a mobile equipment identifier (MEID), and a device identifier.	number, an identification number, a security signature, a security credential, a subscriber identity module (SIM) identifier, a mobile equipment identifier (MEID), and a device identifier." *See supra* 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; *see below*).

114

Architecture

The architecture of the Tizen Push service is described in detail in the mobile native Push guide.

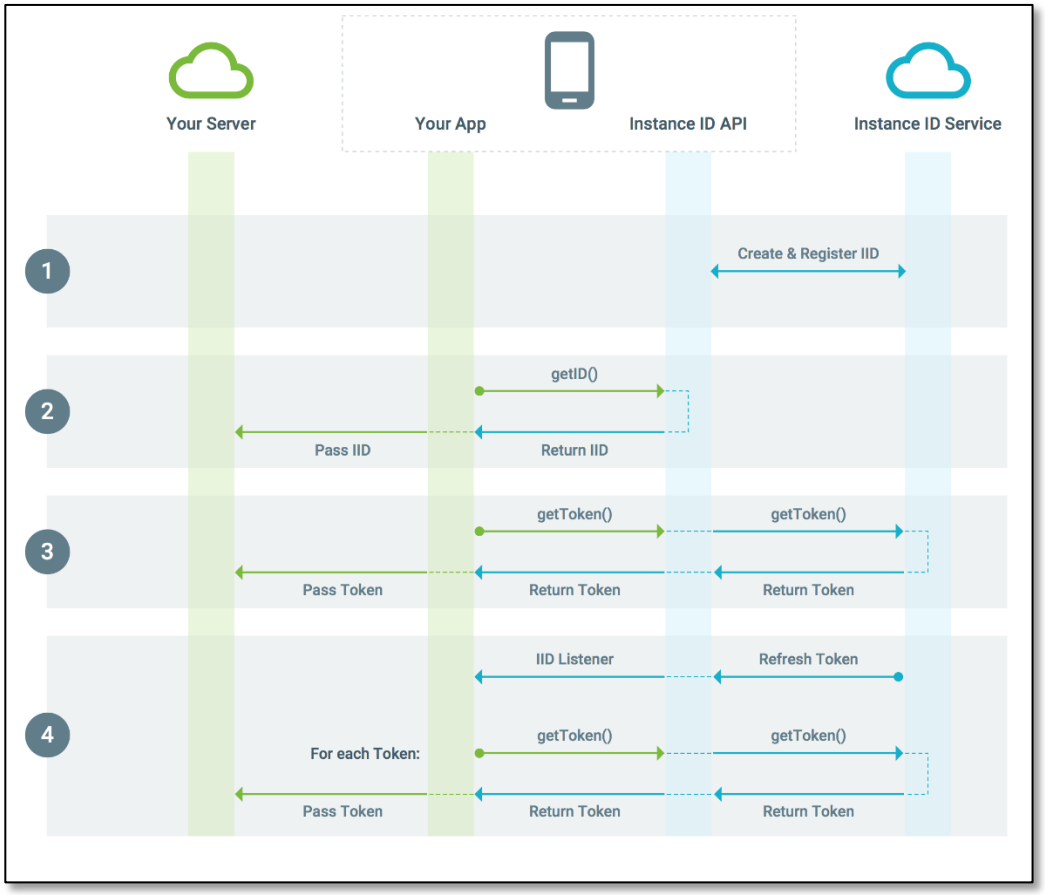
Figure: Service architecture



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.

115



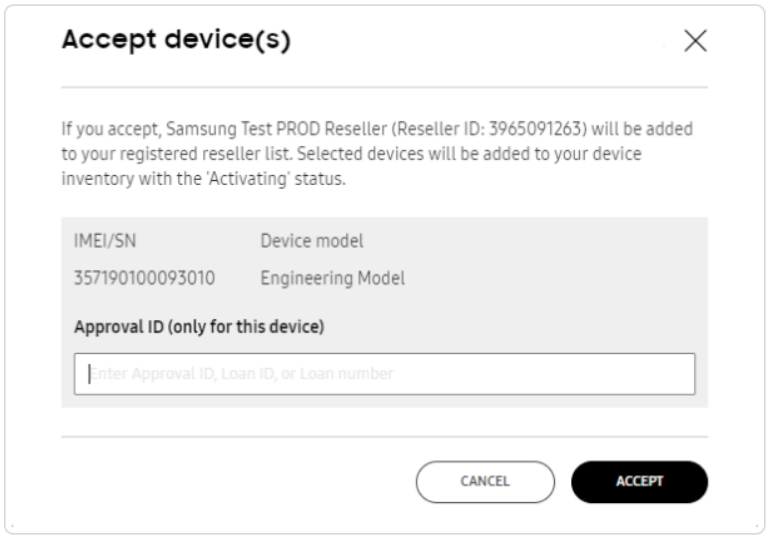
As another example, managed devices in a Samsung Knox enroll with the Knox service by communicating with Knox servers (service control server link elements) over the service control link by providing to Knox servers, for example, information like the device's IMEI number (a mobile device identification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

116

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**.



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." *See supra* 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." *See supra* claims 1 and 3. The control-plane communications discussed herein are all made over the Internet and using an Internet protocol.

117

Claim 30

30. A method performed by an end-user device, the method comprising:	Samsung Galaxy phones and tablets and Samsung's Tizen based devices perform the method of claim 30, as set forth below.
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 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." *See supra* 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

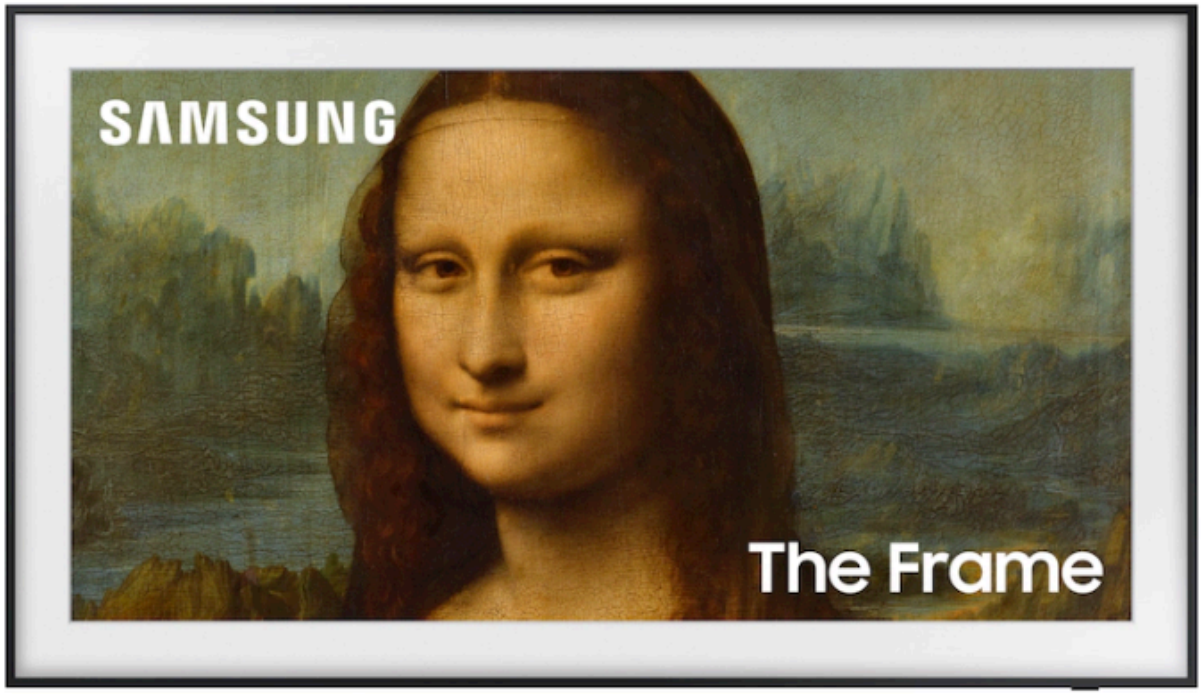
agent message comprising a particular agent identifier and message content for delivery to a particular device agent of a plurality of device agents on the end-user device, each of the plurality of device agents identifiable by an associated device agent identifier and communicatively coupled to the 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." *See supra* 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." *See supra* claim 1[b]-1[f].

Exhibit B - U.S. Patent No. 9,198,117 ("'117 Patent")

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

Claim 1

Issued Claim(s)	Public Documentation
[1pre]. A network system comprising:	Samsung's devices and push messaging servers comprise a "network system." *See, e.g.,*  https://www.samsung.com/us/smartphones/galaxy-s22/;

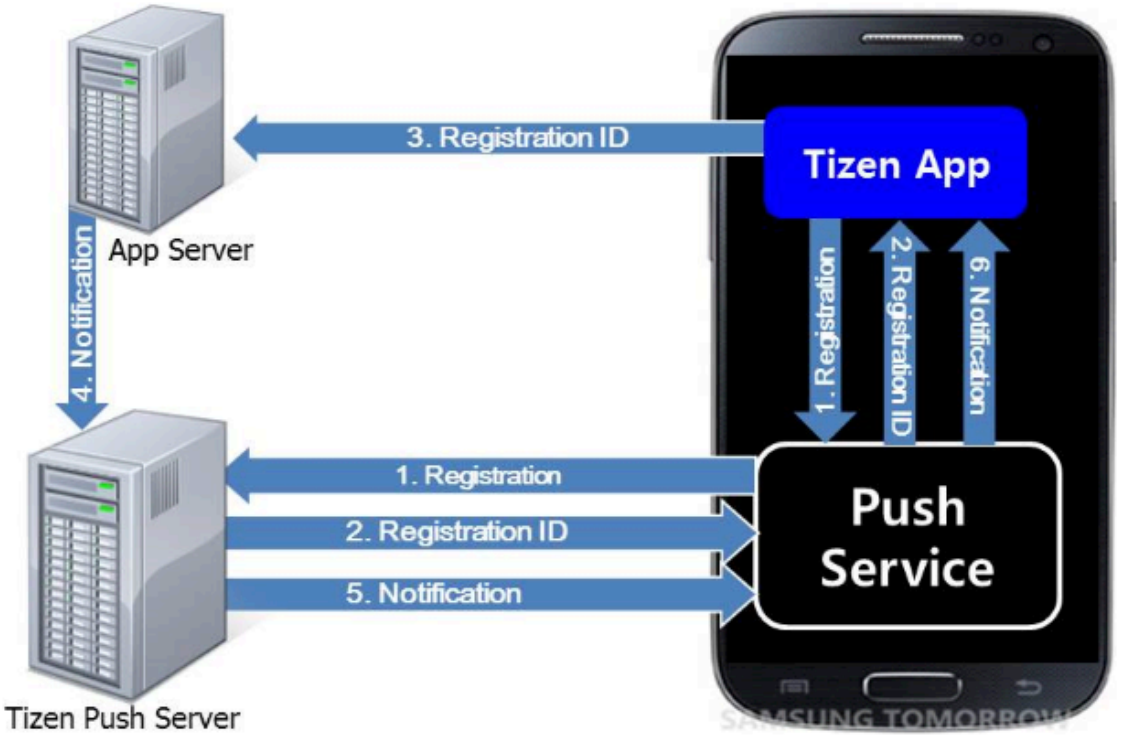
120		 https://www.samsung.com/us/televisions-home-theater/tvs/the-frame/55-class-the-frame-qled-4k-smart-tv-2022-qn55ls03bafxza/;

121

Architecture

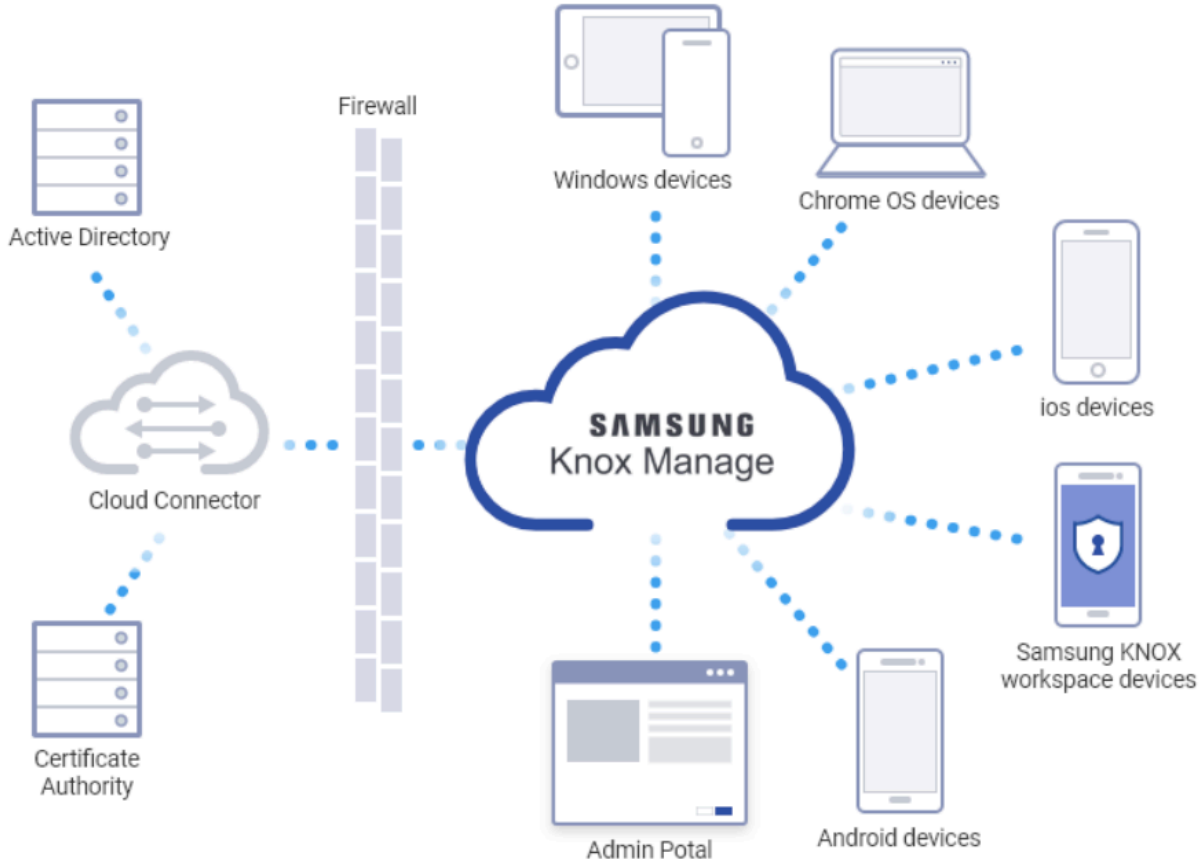
The architecture of the Tizen Push service is described in detail in the mobile native Push guide.

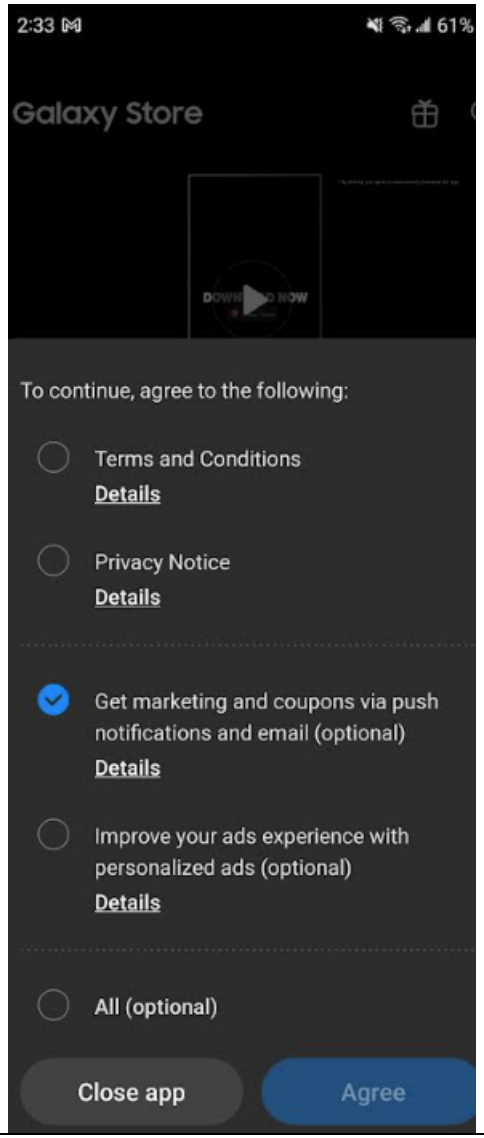
Figure: Service architecture

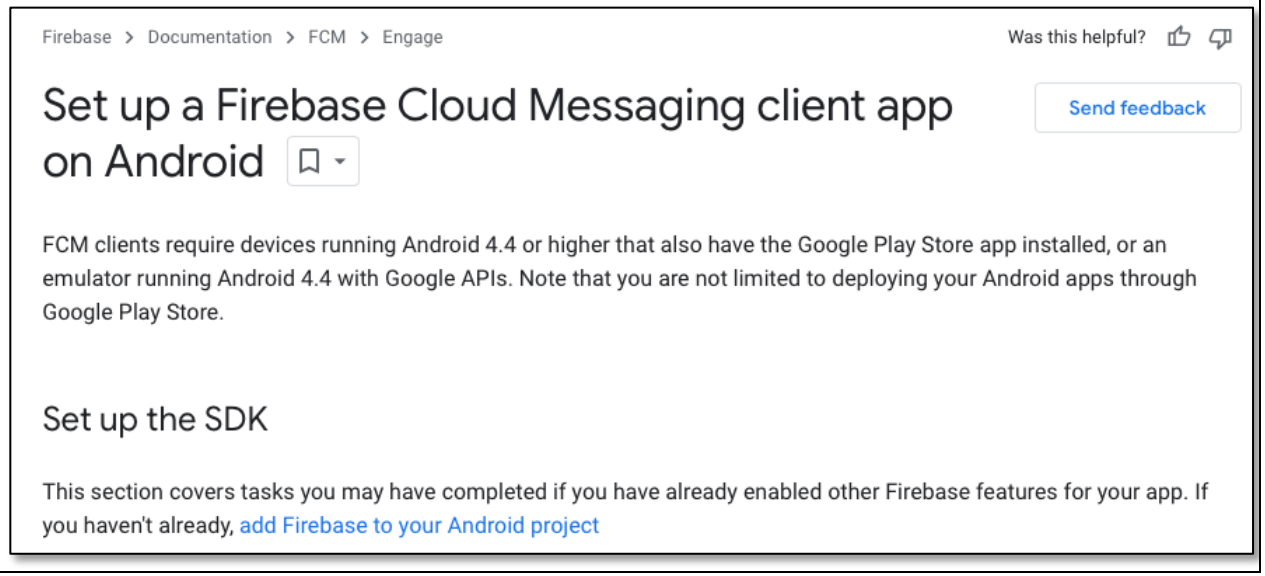


To receive push notifications for your application:

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.

122	https://docs.tizen.org/application/web/guides/messaging/push/. As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, comprise a "network system."  https://docs.samsungknox.com/admin/knox-manage/welcome.htm On information and belief, Samsung owns and operates its own push messaging servers, including servers for communicating with the Samsung Push Service software on its end user devices. *See, e.g.*, https://play.google.com/store/apps/details?id=com.sec.spp.push&hl=en_US&gl=US&pli=1;

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

[1a] a plurality of device messaging agents, each executable on a respective one of a plurality of mobile end-user devices configured to exchange Internet data via a data connection to a wireless network; and	Samsung devices and push messaging servers comprise "a plurality of device messaging agents, each executable on a respective one of a plurality of mobile end-user devices configured to exchange Internet data via a data connection to a wireless network." For example, Samsung Galaxy phones and tablets comprise multiple device agents which have an identifier. *See e.g.*, 

125

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`.

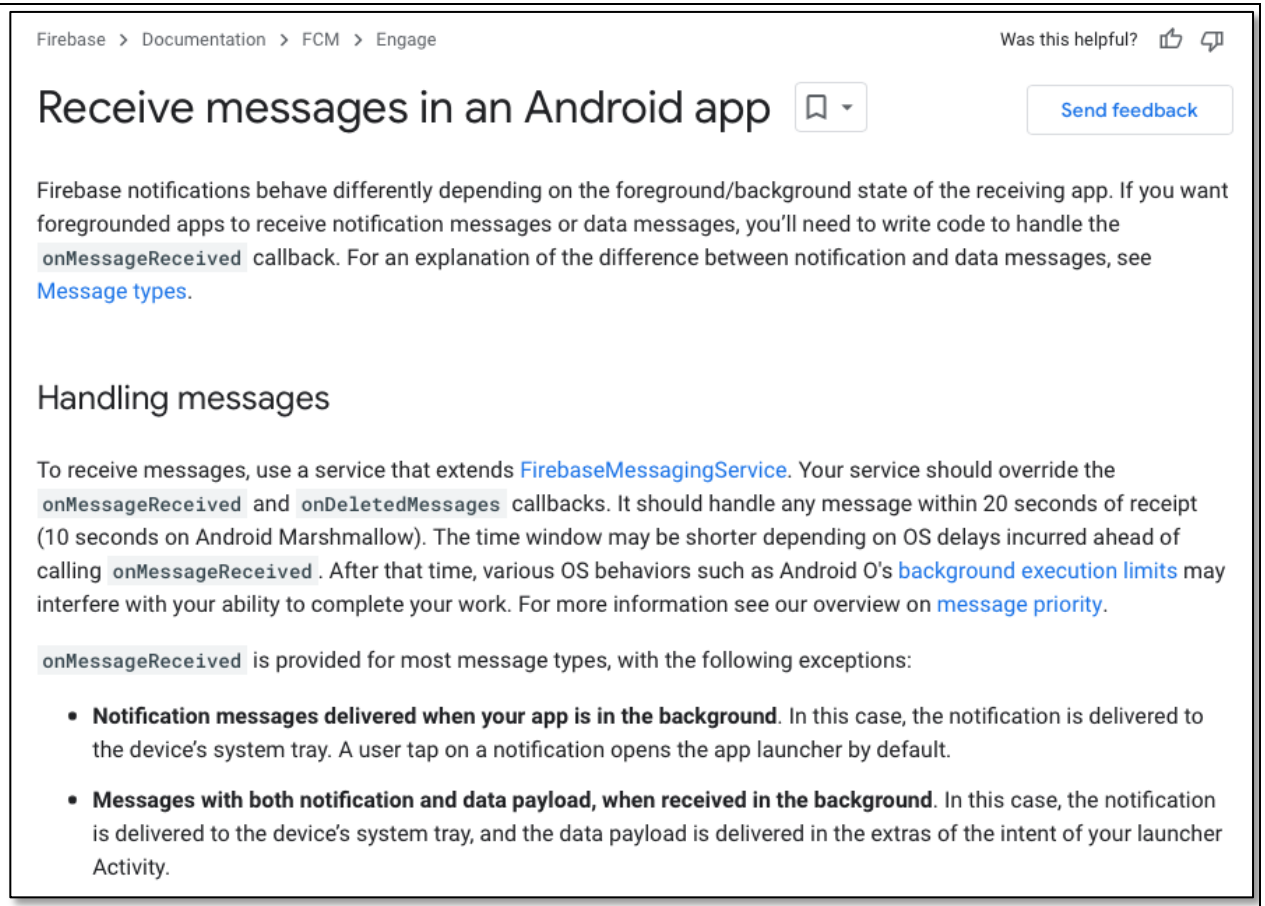
This section describes how to retrieve the token and how to monitor changes to the token. Because the token could be rotated after initial startup, you are strongly recommended to retrieve the latest updated registration token.

The registration token may change when:

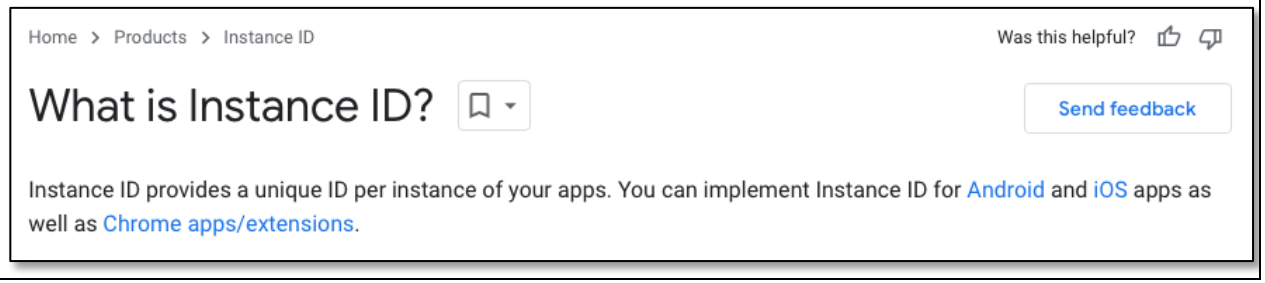
- The app is restored on a new device
- The user uninstalls/reinstall the app
- The user clears app data.

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

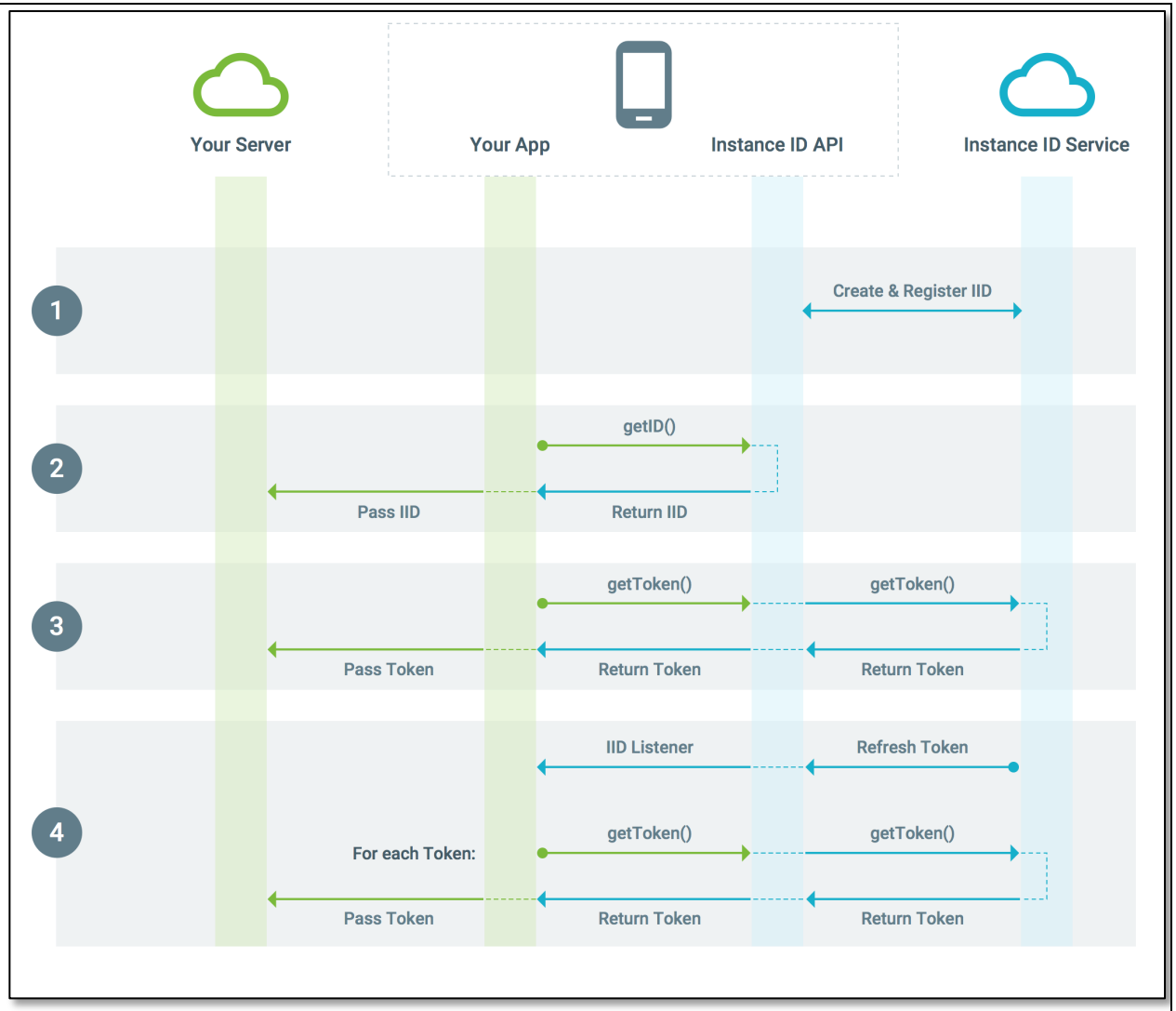
126



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



127



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

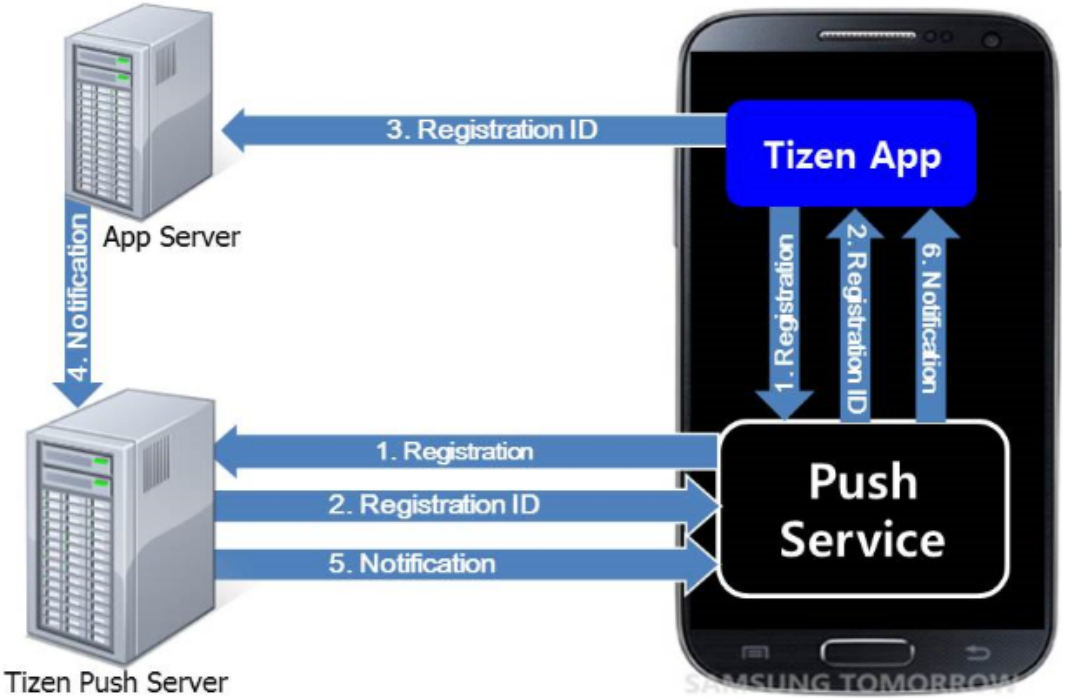
For further example, Samsung's Tizen based devices comprise multiple device agents which have an identifier. *See e.g.,*

128

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.

129

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.

130

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);
```

131

- 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);
```

132

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

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

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

Push Notification

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

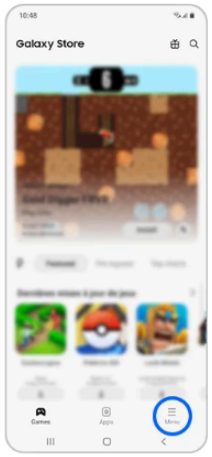
https://developer.tizen.org/development/guides/web-application/messaging/push-notification;

133	# Push API The Push API provides functionality for receiving push notifications from the Tizen push server. The push service is a client daemon that maintains a permanent connection between your device and the Tizen push server. Connection with push server is used to deliver push notifications to the application, and process the registration and deregistration requests. To receive push notifications, follow the steps below: • Connecting to the push service • Registering your application, if the application has not been registered yet • Getting notification data https://developer.samsung.com/smarttv/develop/api-references/tizen-web-device-api-references/push-api.html;

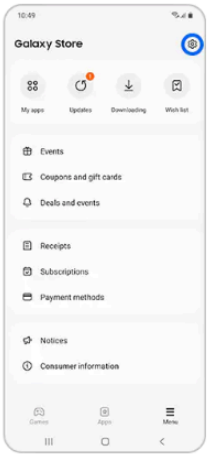
134

How to set up push notifications

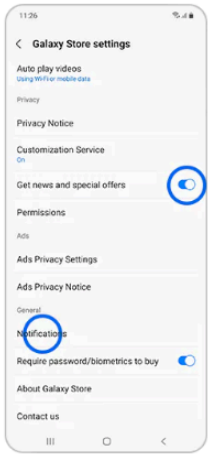
If you want to set up push notifications, follow these steps.



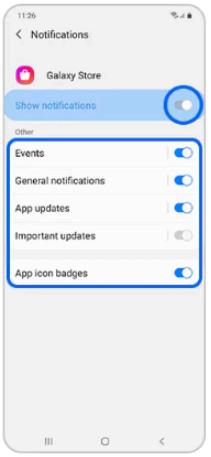
Step 1. Log in to Galaxy Store and tap the "Menu" icon from the main screen.



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



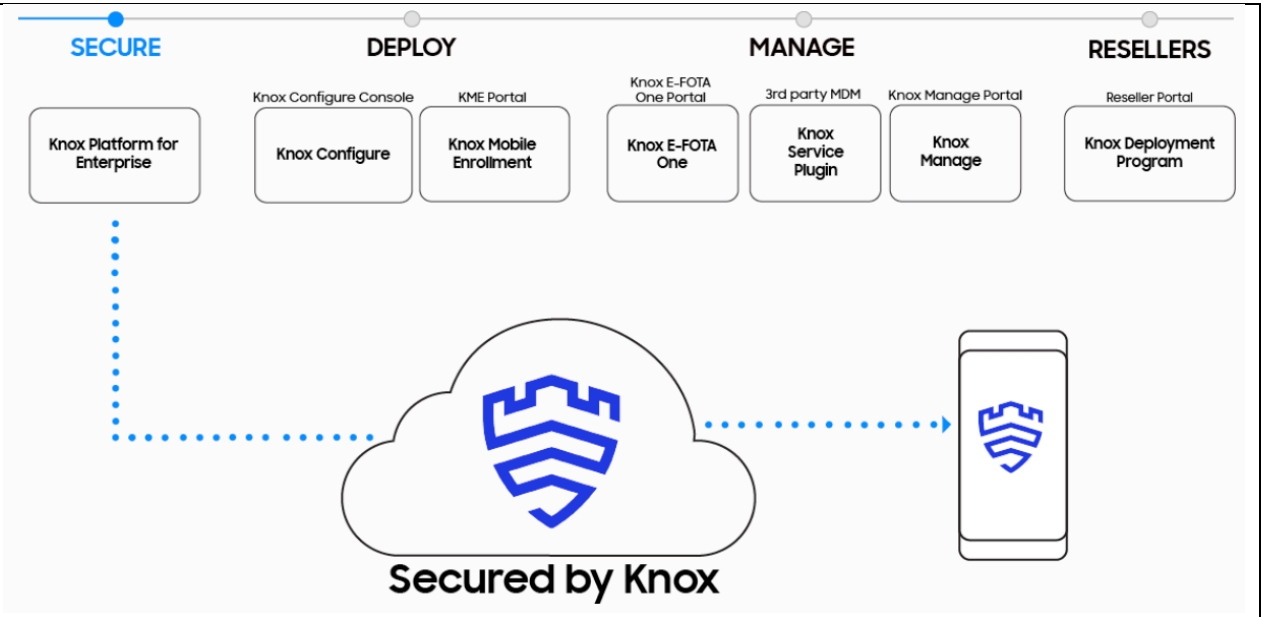
Step 3. Enable "Get news and special offers" and tap "Notifications."



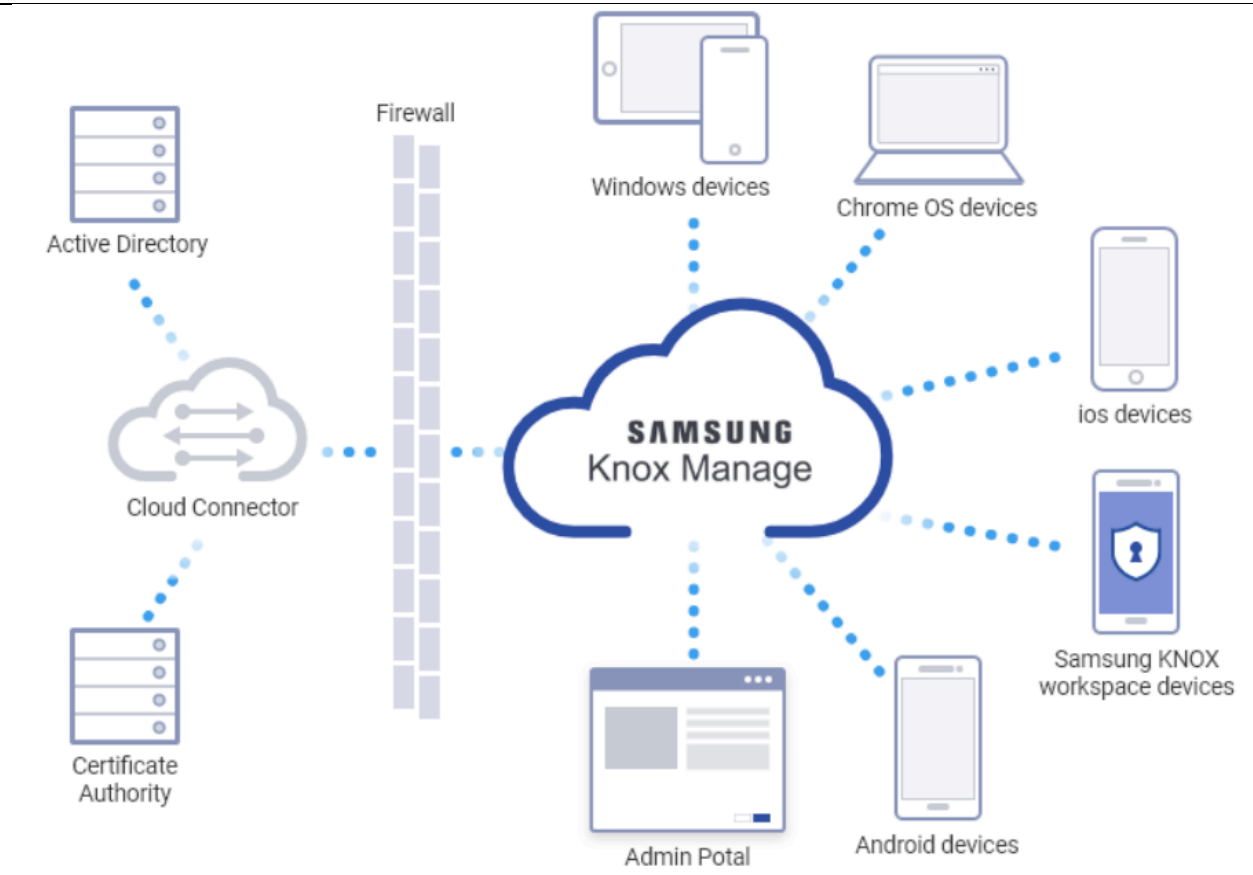
Step 4. Enable "Show notifications" and then enable the types of notifications you want to receive.

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

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

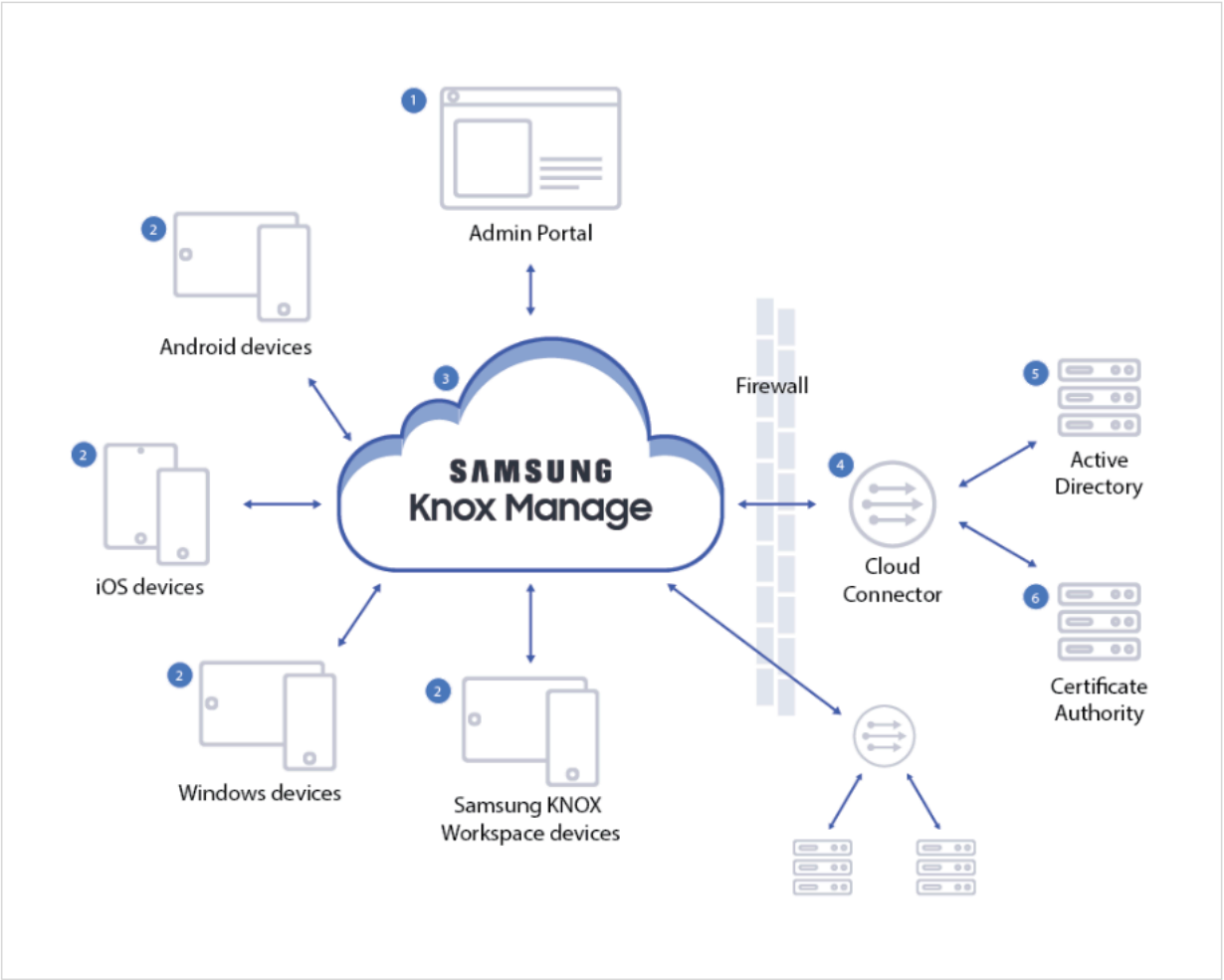
135	 https://docs.samsungknox.com/admin/fundamentals/welcome.htm

136



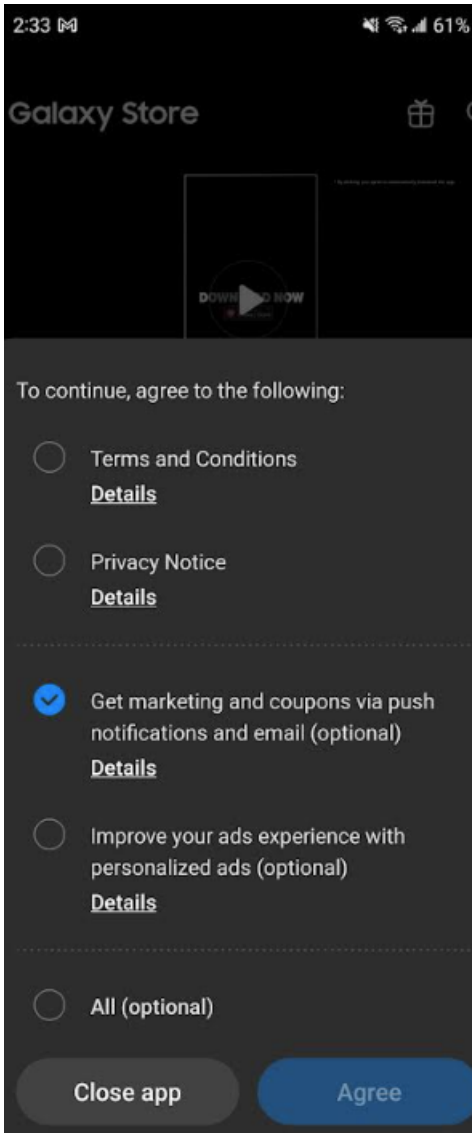
https://docs.samsungknox.com/admin/knox-manage/welcome.htm

137



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

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

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

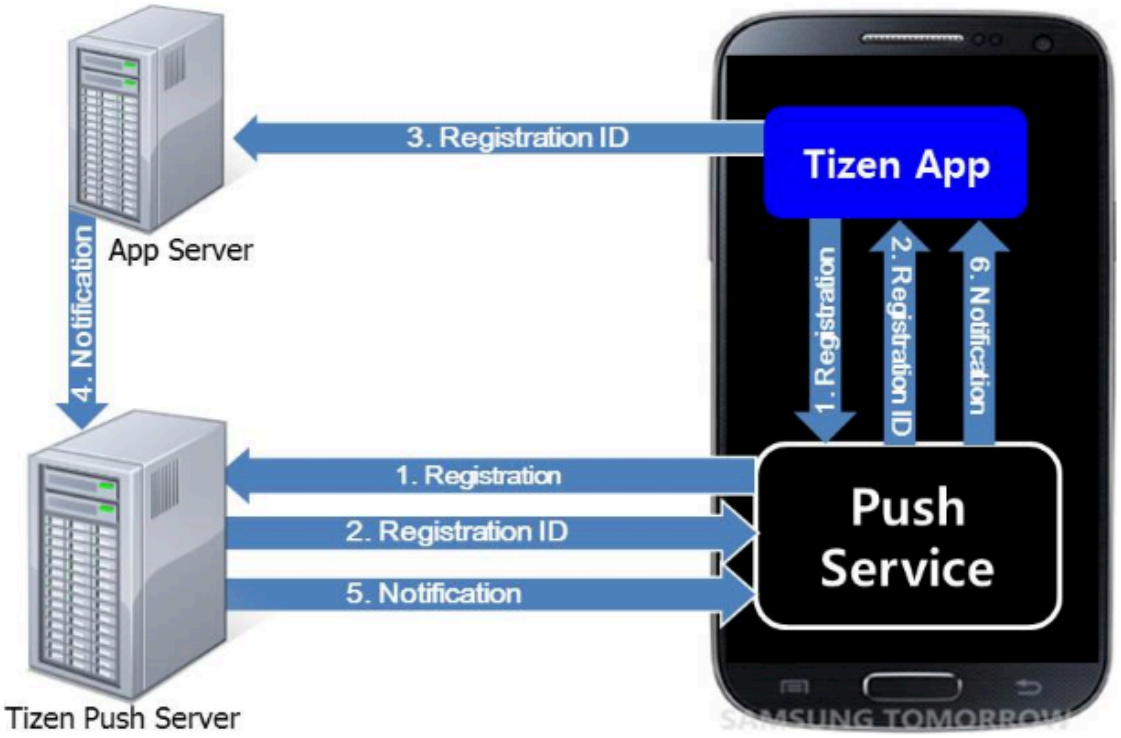
[1b] a network message server	Samsung devices and push messaging servers comprise "a network message server." For example, Samsung's push messaging servers are network servers. *See e.g.*,

140

Architecture

The architecture of the Tizen Push service is described in detail in the mobile native Push guide.

Figure: Service architecture

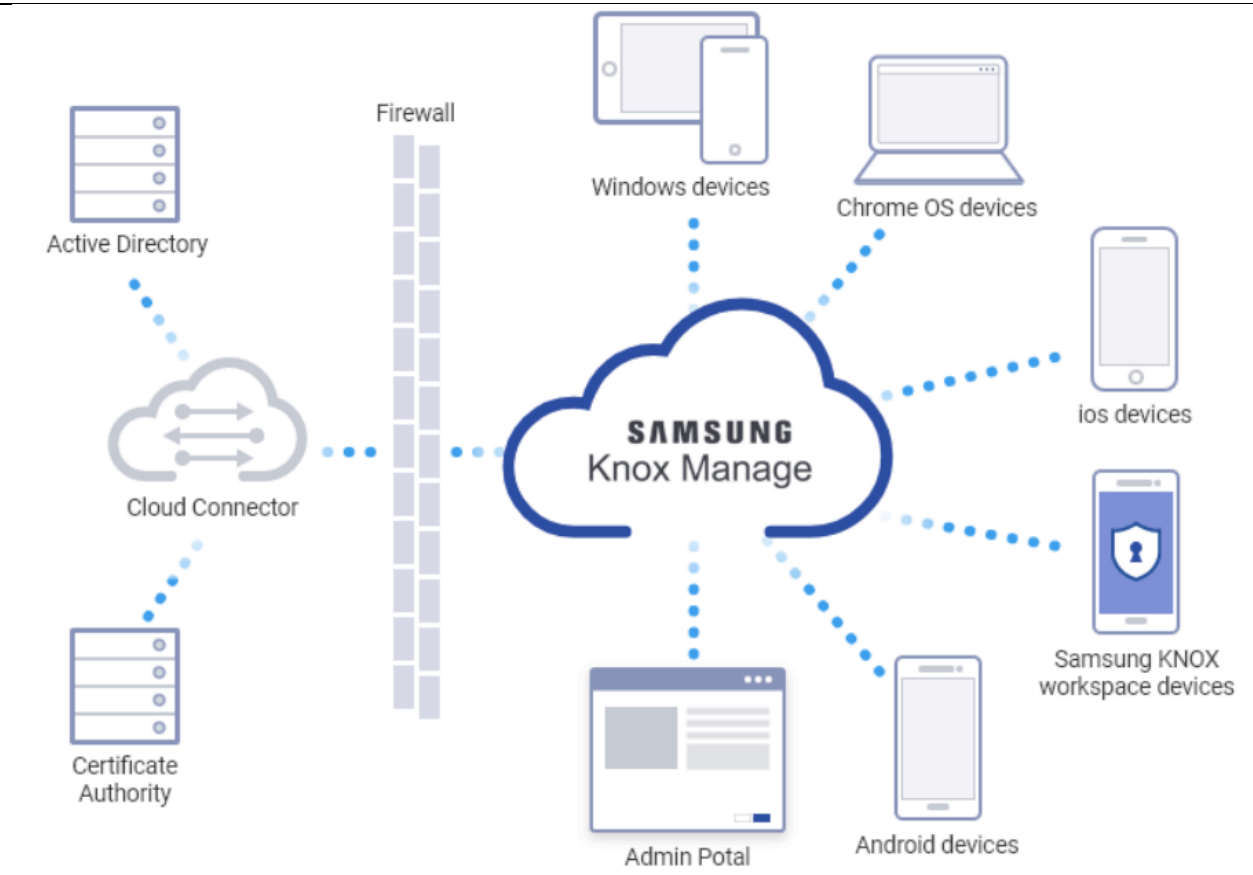


To receive push notifications for your application:

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.

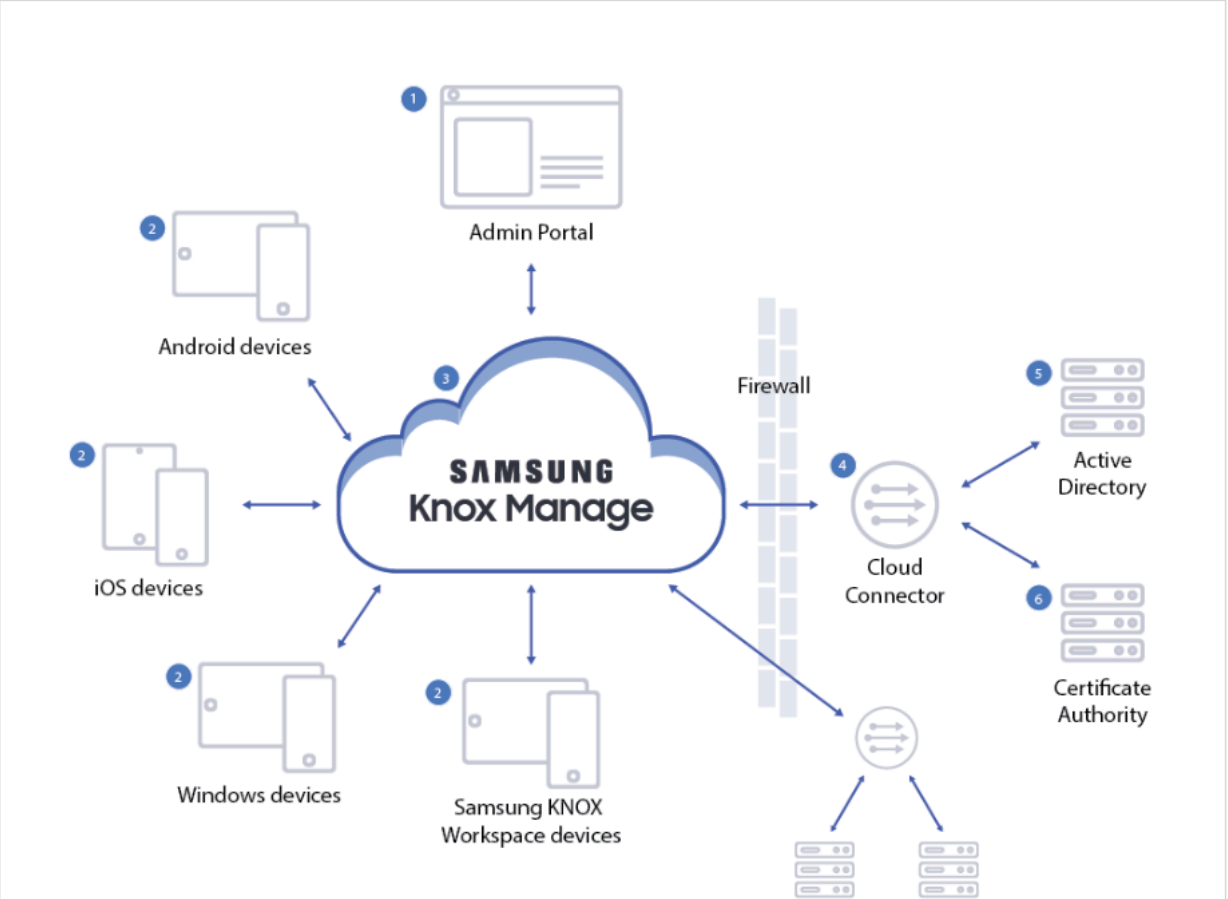
141		https://docs.tizen.org/application/web/guides/messaging/push/; **Push Server** You can push events from an application server to your application on a Tizen device. If the message sending fails for any reason, an error code identifying the failure reason is returned. You can use the error code to determine how to handle the failure. The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices. The main features of the Push API for the server developers include: • Sending push notifications You can send push notifications from the application server to an application. • Decorating push notifications You can add decorations to the push notifications in the quick panel. https://docs.tizen.org/application/native/guides/messaging/push-server/; As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, include network message servers.

142



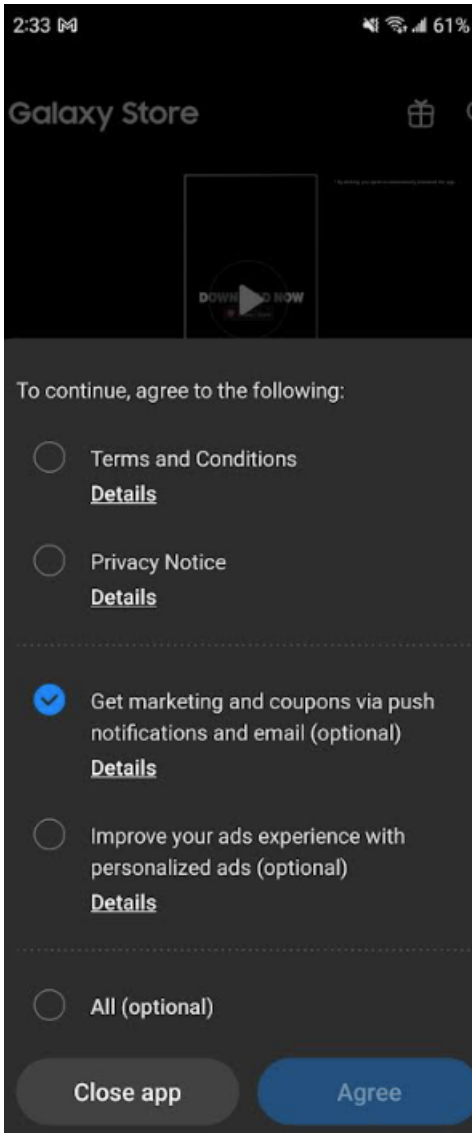
https://docs.samsungknox.com/admin/knox-manage/welcome.htm

143



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

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

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

[1c] supporting a plurality of secure Internet data connections, each secure Internet data connection between the network message server and a respective one of the mobile end-user devices via a device data connection to a wireless network,	Samsung's push messaging servers "support[] a plurality of secure Internet data connections, each secure Internet data connection between the network message server and a respective one of the mobile end-user devices via a device data connection to a wireless network." For example, Samsung's push messaging server supports a plurality of secure connections between the network message server and respective devices. *See, e.g.,* **Push Notification** You can receive notifications from a push server. The push service is a client daemon that maintains a permanent connection between the device and the push server. Push enables you to push events from an application server to your application on a Tizen device. Connection with the push service is used to deliver push notifications to the application, and process the registration and deregistration requests. The Push API is optional for Tizen Mobile, Wearable, and TV profiles, which means that it may not be supported on all mobile, wearable, and TV devices. The Push API is supported on all Tizen emulators. Push notification helps your application server send data to your application on a device over a network, even if the application is not running. Using the push service can reduce battery consumption and data transfer. If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a `LAUNCH` option, the push service forcibly launches the application and hands over the message to the application.

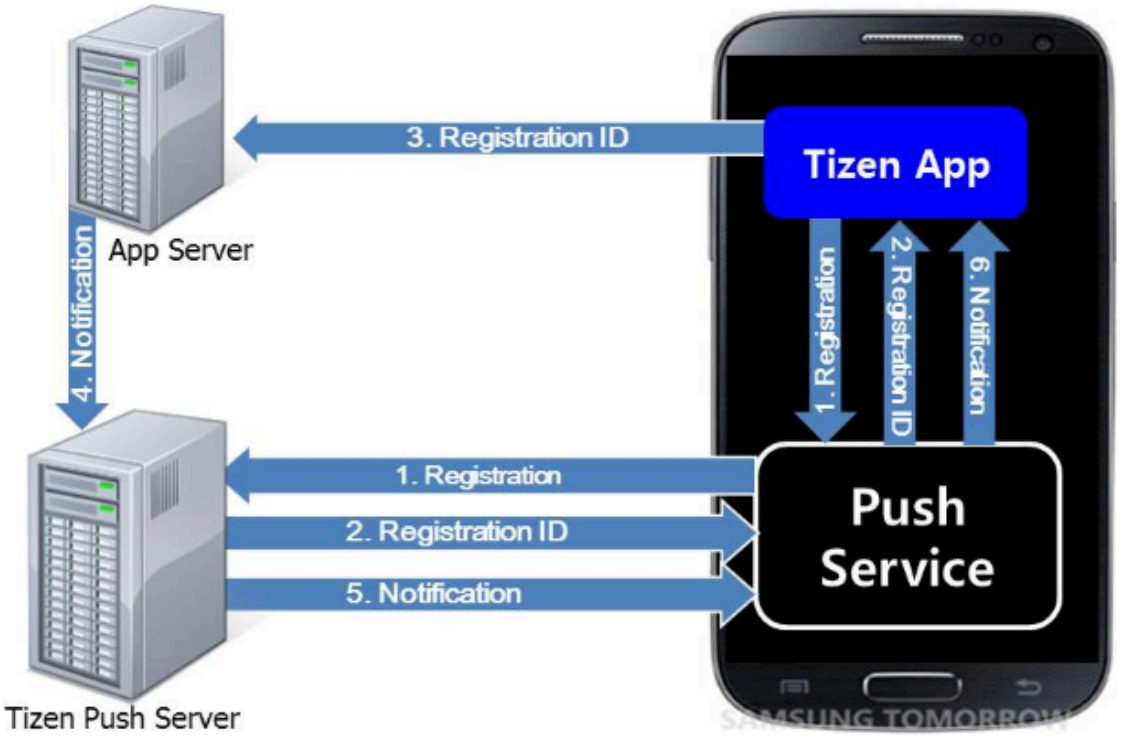
145

146

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.

147

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);
```

148

- Since Tizen 3.0:

  Before registering, you must connect to the push service:

  1. Define event handlers:

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

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

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

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

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

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

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

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

149

Push API

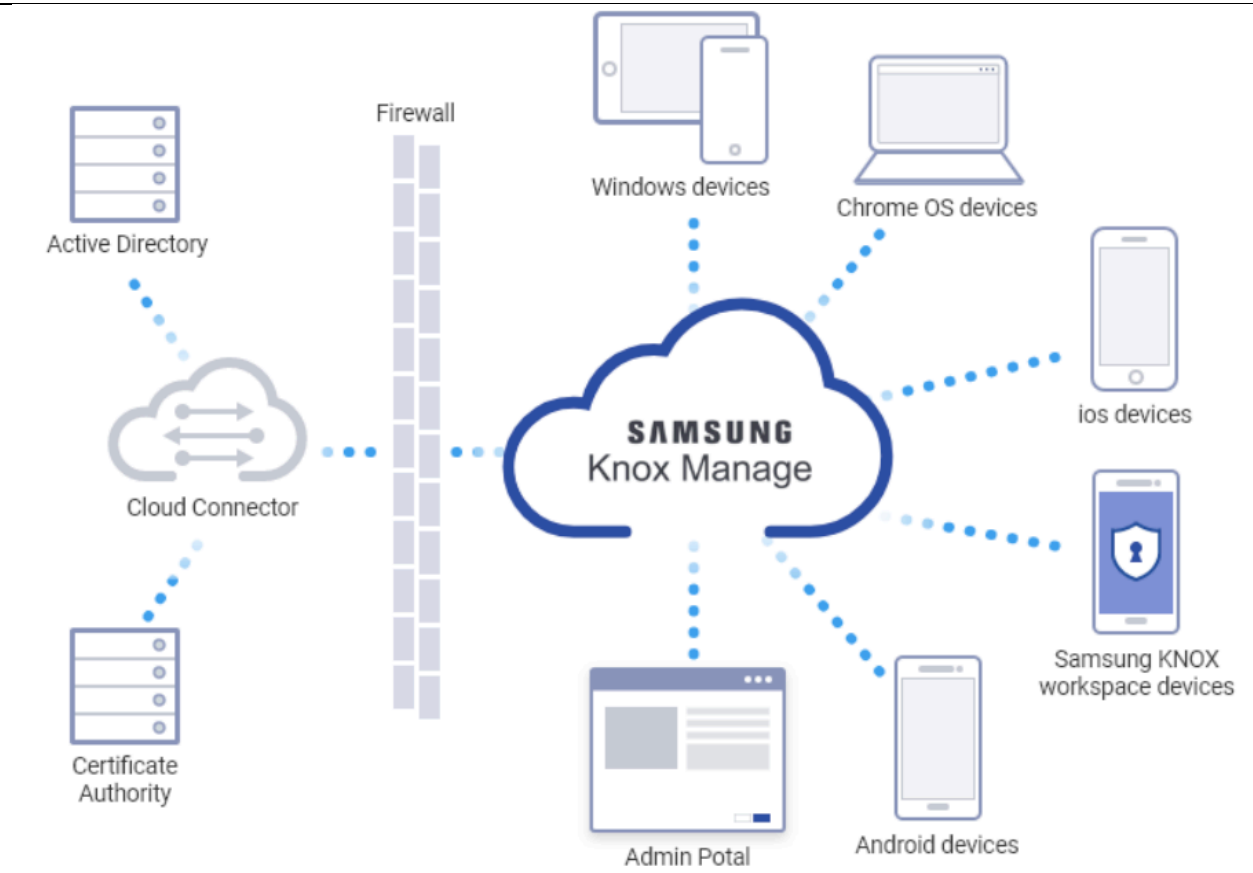
The Push API provides functionality for receiving push notifications from the Tizen push server. The push service is a client daemon that maintains a permanent connection between your device and the Tizen push server. Connection with push server is used to deliver push notifications to the application,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

To receive push notifications, follow the steps below:

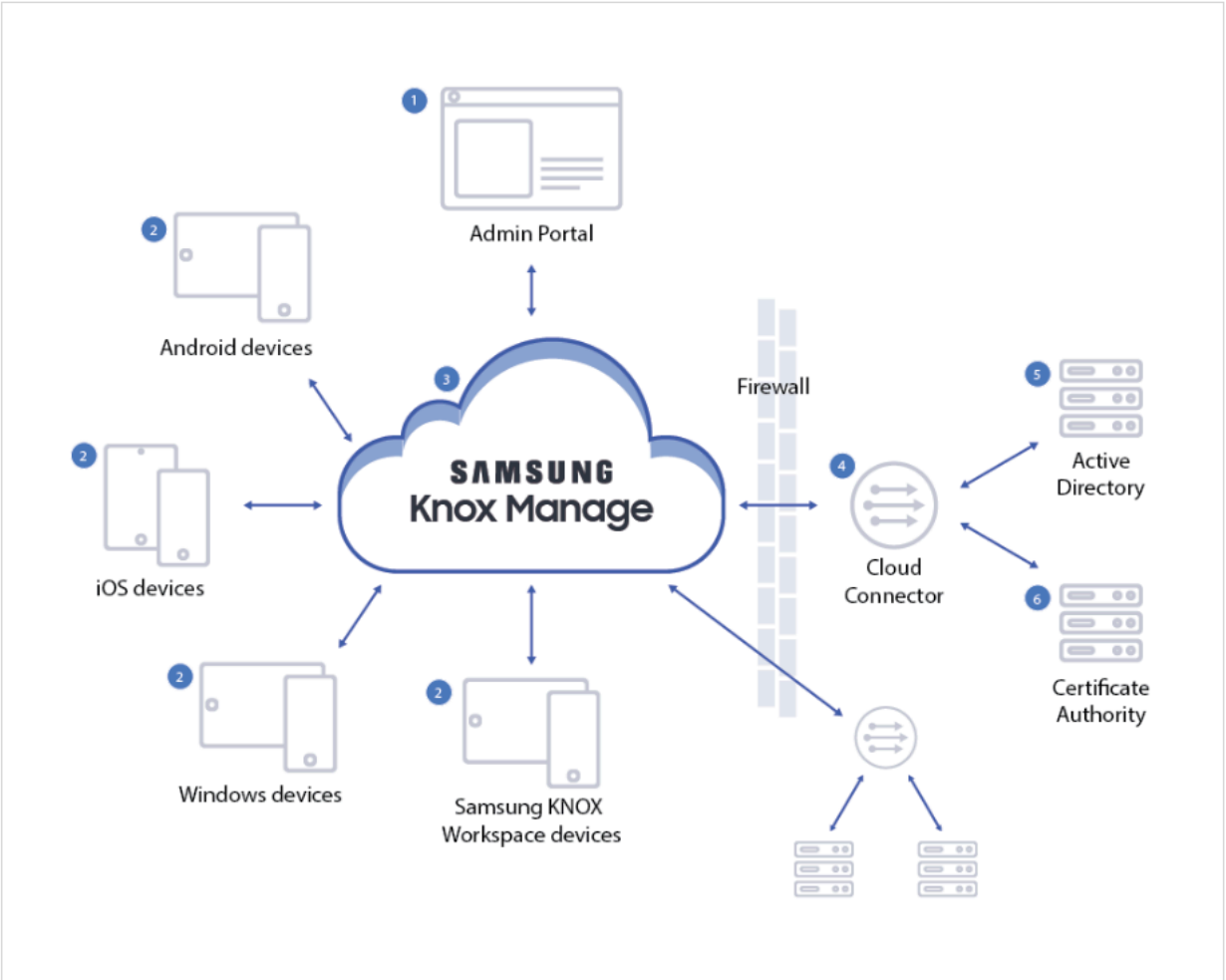
- Connecting to the push service
- Registering your application, if the application has not been registered yet
- Getting notification data

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

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

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

151



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

[1d] the network message server configured to receive, from each of a plurality of network application servers, multiple requests to transmit application data, each such request indicating a corresponding one of the mobile end-user	Samsung's push messaging servers comprise "the network message server configured to receive, from each of a plurality of network application servers, multiple requests to transmit application data, each such request indicating a corresponding one of the mobile end-user devices and one of a plurality of applications."

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

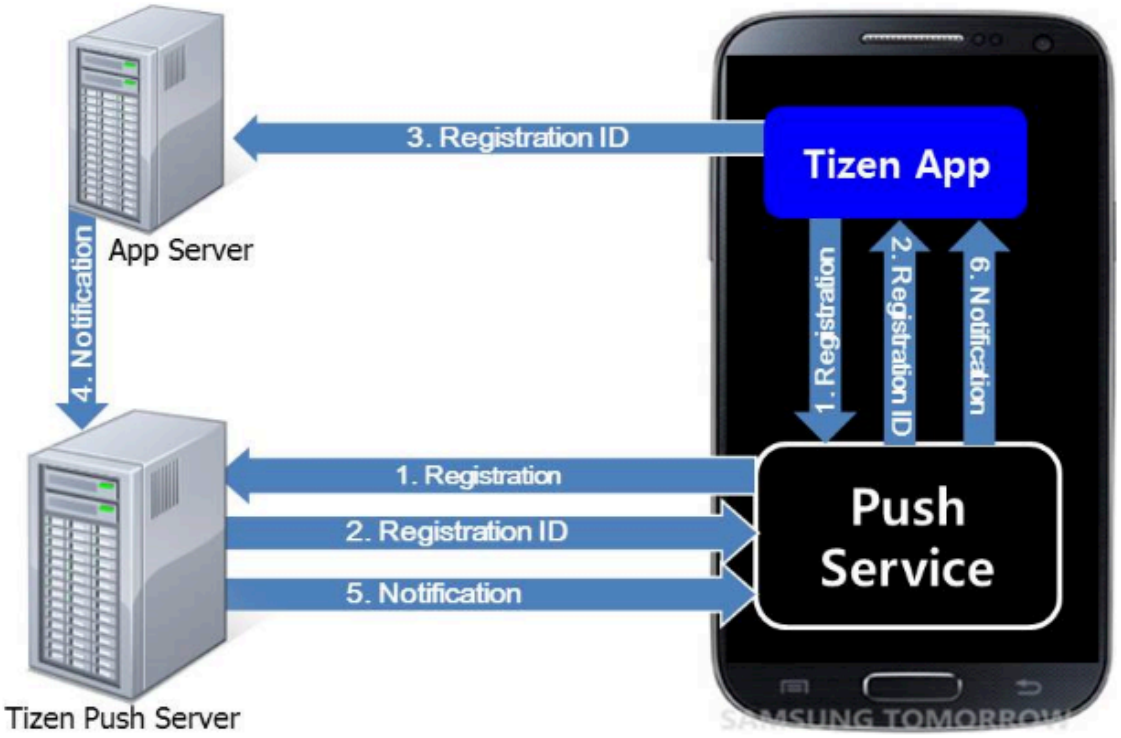
152

153

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.

154	https://docs.tizen.org/application/web/guides/messaging/push/; **Push Server** You can push events from an application server to your application on a Tizen device. If the message sending fails for any reason, an error code identifying the failure reason is returned. You can use the error code to determine how to handle the failure. The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices. The main features of the Push API for the server developers include: • Sending push notifications You can send push notifications from the application server to an application. • Decorating push notifications You can add decorations to the push notifications in the quick panel. https://docs.tizen.org/application/native/guides/messaging/push-server/; **Push** You can push events from an application server to your application on a Tizen device. The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices. Once your application is successfully registered in the push server through the push service (daemon) on the device, your application server can send push messages to the application on that particular device. If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a `LAUNCH` option, the push service forcibly launches the application and hands over the message to the application as an application control.

Figure: Push messaging service



155

156

Sending Push Notifications

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

The following example shows a sample push notification:

- URI: See the Push RQM (Request Manager) server URLs table.
- Method: HTTP POST
- Header:

```
appID: 1234567890987654
appSecret: dYo/o/m11gmWmjs7+5f+2zLNVOc=
```

- Body:

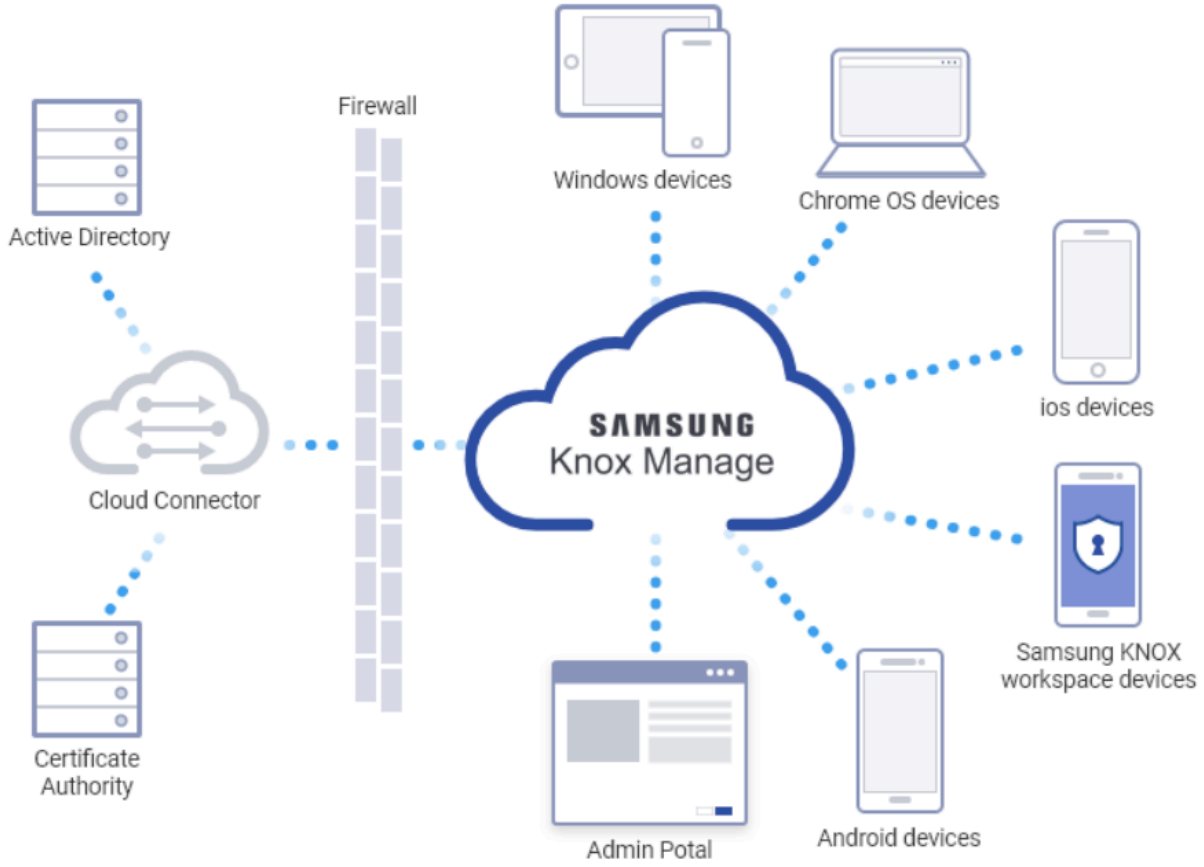
```
{
    "regID": "0501a53f4affdcbb98197f188345ff30c04b-5001",
    "requestID": "01231-22EAX-223442",
    "message": "badgeOption=INCREASE&badgeNumber=1&action=ALERT&alertMessage=Hi",
    "appData": "{id:asdf&passwd:1234}", /* Optional, if the message field is not empty */
}
```

To send a notification:

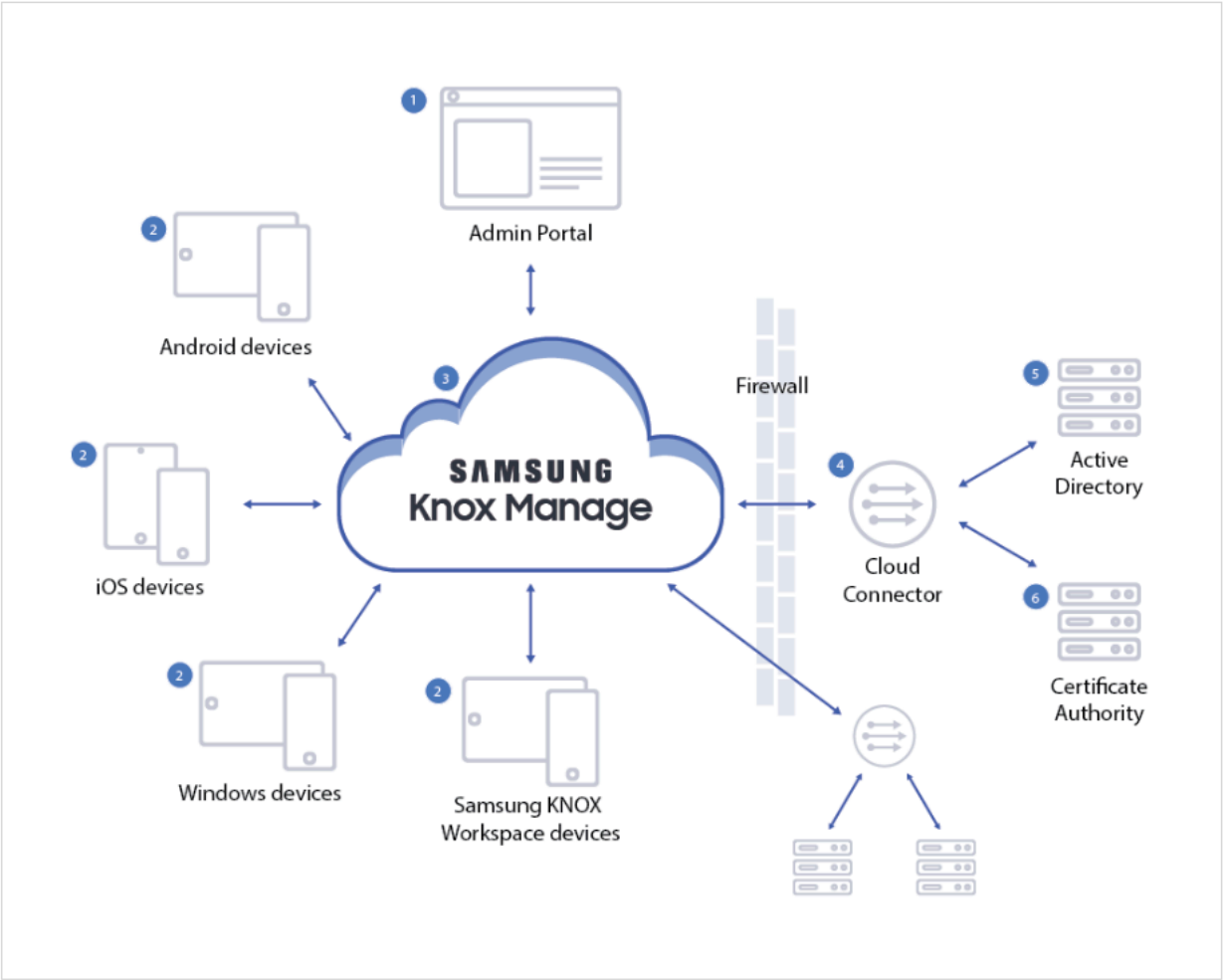
1. Prepare the `appID`, `appSecret`, `regID`, and `requestID`:
    - The `appID` and `appSecret` values are given in the email message that you received when requesting permission to use Tizen push servers.
    - The `regID` value is the one that the application server received from your application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

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

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

157	As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, comprise a plurality of network application servers, multiple requests to transmit application data, each such request indicating a corresponding one of the mobile end-user devices and one of a plurality of applications.  https://docs.samsungknox.com/admin/knox-manage/welcome.htm

158



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

159

Update an existing device profile

22.11 23.03 UAT

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

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

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

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

Components of Knox Manage

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

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

[1e] the network message server to generate corresponding Internet data messages based on the requests, each	Samsung's push messaging servers comprise "the network message server to generate corresponding Internet data messages based on the requests, each such message containing at

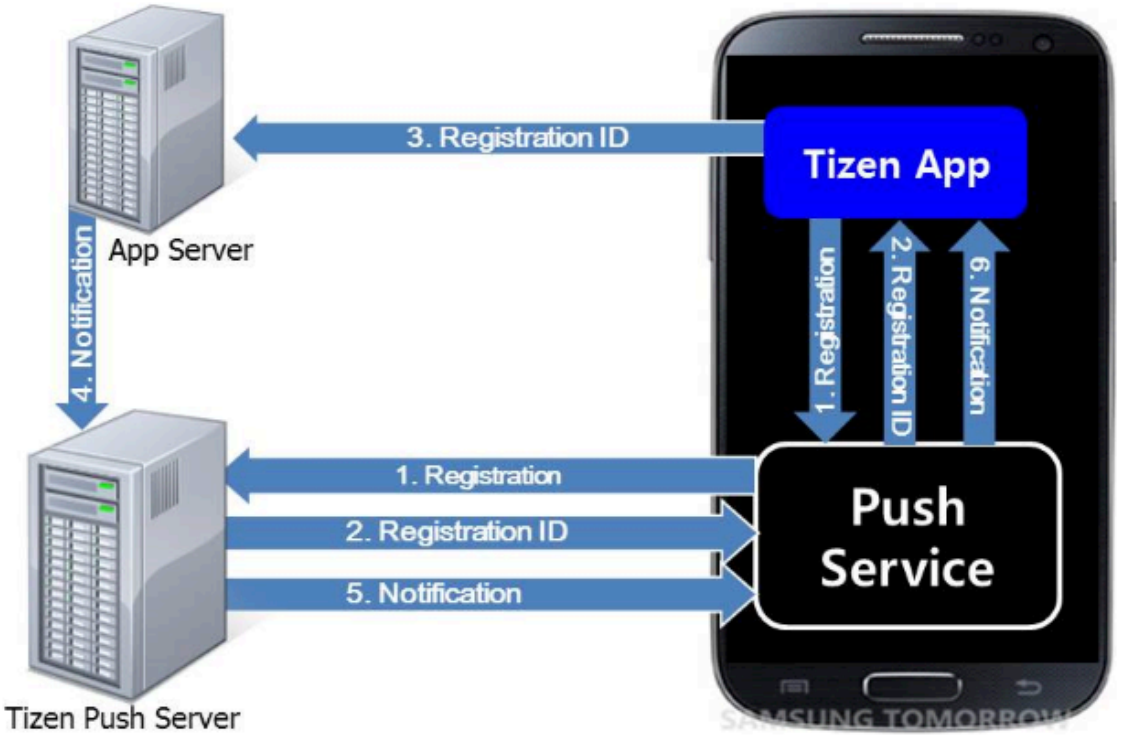
such message containing at least one application identifier for an indicated application and application data corresponding to one of the requests, and	least one application identifier for an indicated application and application data corresponding to one of the requests." For example, Samsung's push messaging servers generate data messages containing an application identifier and data corresponding to the requests. *See, e.g.,*

160

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.

162	https://docs.tizen.org/application/web/guides/messaging/push/; **Push Server** You can push events from an application server to your application on a Tizen device. If the message sending fails for any reason, an error code identifying the failure reason is returned. You can use the error code to determine how to handle the failure. The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices. The main features of the Push API for the server developers include: • Sending push notifications You can send push notifications from the application server to an application. • Decorating push notifications You can add decorations to the push notifications in the quick panel. https://docs.tizen.org/application/native/guides/messaging/push-server/; **Push** You can push events from an application server to your application on a Tizen device. The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices. Once your application is successfully registered in the push server through the push service (daemon) on the device, your application server can send push messages to the application on that particular device. If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a `LAUNCH` option, the push service forcibly launches the application and hands over the message to the application as an application control.

163

Figure: Push messaging service



164

Sending Push Notifications

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

The following example shows a sample push notification:

- URI: See the Push RQM (Request Manager) server URLs table.
- Method: HTTP POST
- Header:

```
appID: 1234567890987654
appSecret: dYo/o/m11gmWmjs7+5f+2zLNVOc=
```

- Body:

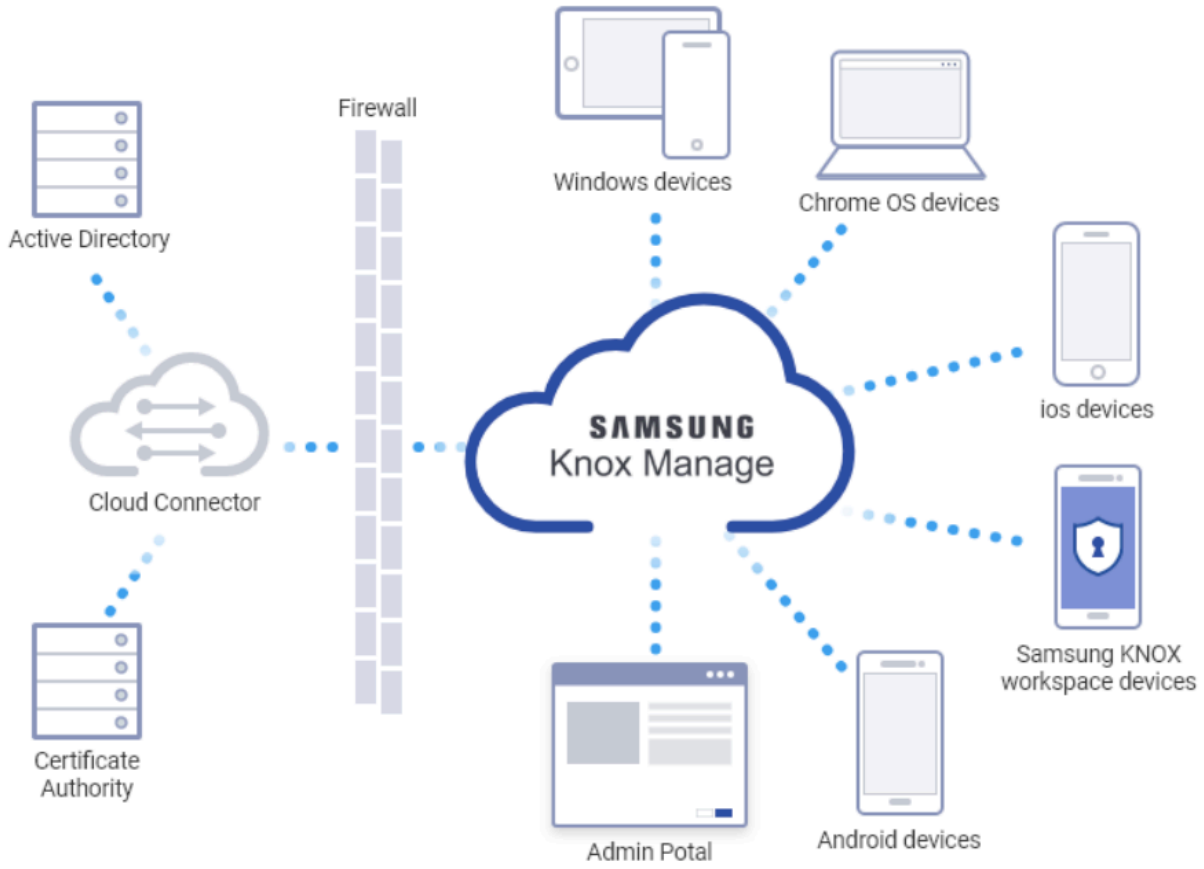
```
{
    "regID": "0501a53f4affdcbb98197f188345ff30c04b-5001",
    "requestID": "01231-22EAX-223442",
    "message": "badgeOption=INCREASE&badgeNumber=1&action=ALERT&alertMessage=Hi",
    "appData": "{id:asdf&passwd:1234}", /* Optional, if the message field is not empty */
}
```

To send a notification:

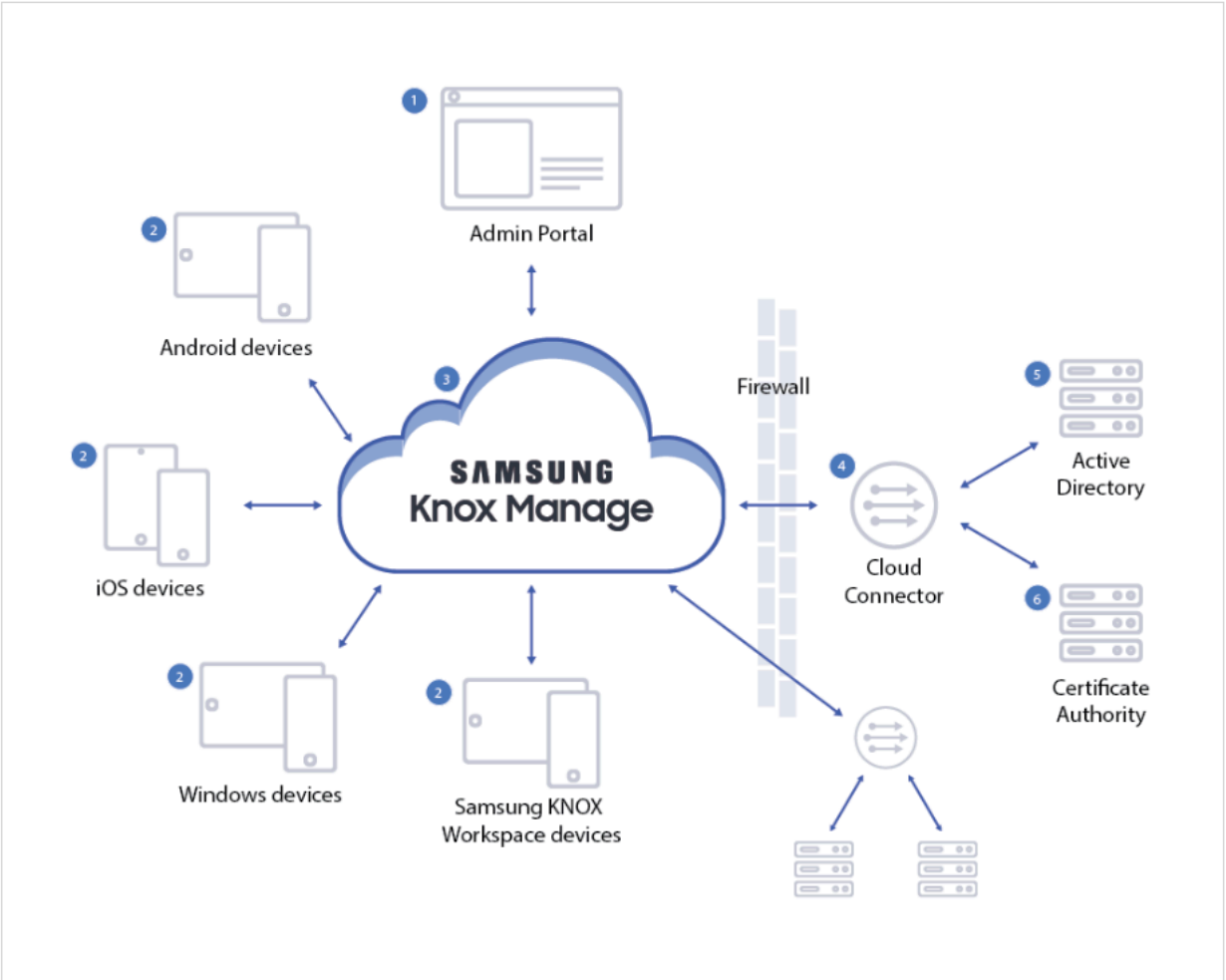
1. Prepare the `appID`, `appSecret`, `regID`, and `requestID`:
    - The `appID` and `appSecret` values are given in the email message that you received when requesting permission to use Tizen push servers.
    - The `regID` value is the one that the application server received from your application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

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

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

165	As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, comprise servers which generate data messages containing an application identifier and data corresponding to the requests.  https://docs.samsungknox.com/admin/knox-manage/welcome.htm

166



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

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

agent located on the device indicated in the corresponding request, using the corresponding secure Internet data connection for the device indicated in the corresponding request;	indicated in the corresponding request, using the corresponding secure Internet data connection for the device indicated in the corresponding request." For example, Samsung's push messaging servers transmit the generated data messages to the device agents on the devices. *See, e.g.,*

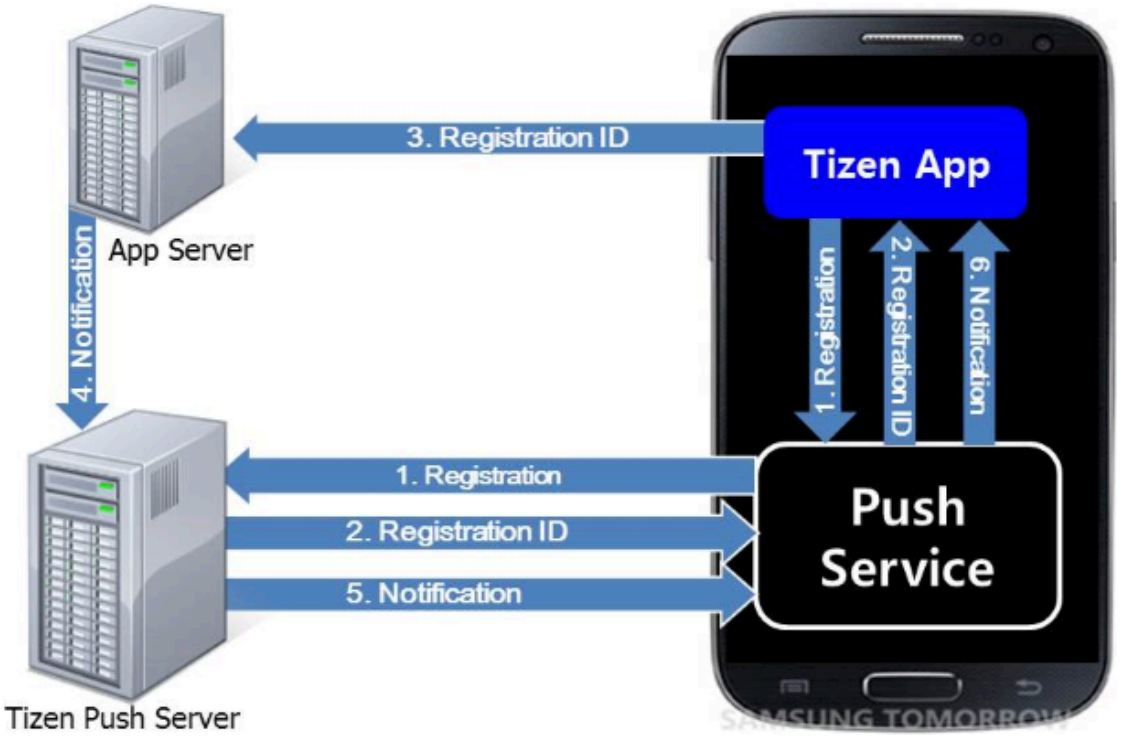
168

169

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.

170	https://docs.tizen.org/application/web/guides/messaging/push/; **Push Server** You can push events from an application server to your application on a Tizen device. If the message sending fails for any reason, an error code identifying the failure reason is returned. You can use the error code to determine how to handle the failure. The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices. The main features of the Push API for the server developers include: • Sending push notifications You can send push notifications from the application server to an application. • Decorating push notifications You can add decorations to the push notifications in the quick panel. https://docs.tizen.org/application/native/guides/messaging/push-server/; **Push** You can push events from an application server to your application on a Tizen device. The Push API is optional for the Tizen Wearable profile, which means that it may not be supported on all wearable devices. Once your application is successfully registered in the push server through the push service (daemon) on the device, your application server can send push messages to the application on that particular device. If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a `LAUNCH` option, the push service forcibly launches the application and hands over the message to the application as an application control.

171

Figure: Push messaging service



172

Sending Push Notifications

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

The following example shows a sample push notification:

- URI: See the Push RQM (Request Manager) server URLs table.
- Method: HTTP POST
- Header:

```
appID: 1234567890987654
appSecret: dYo/o/m11gmWmjs7+5f+2zLNVOc=
```

- Body:

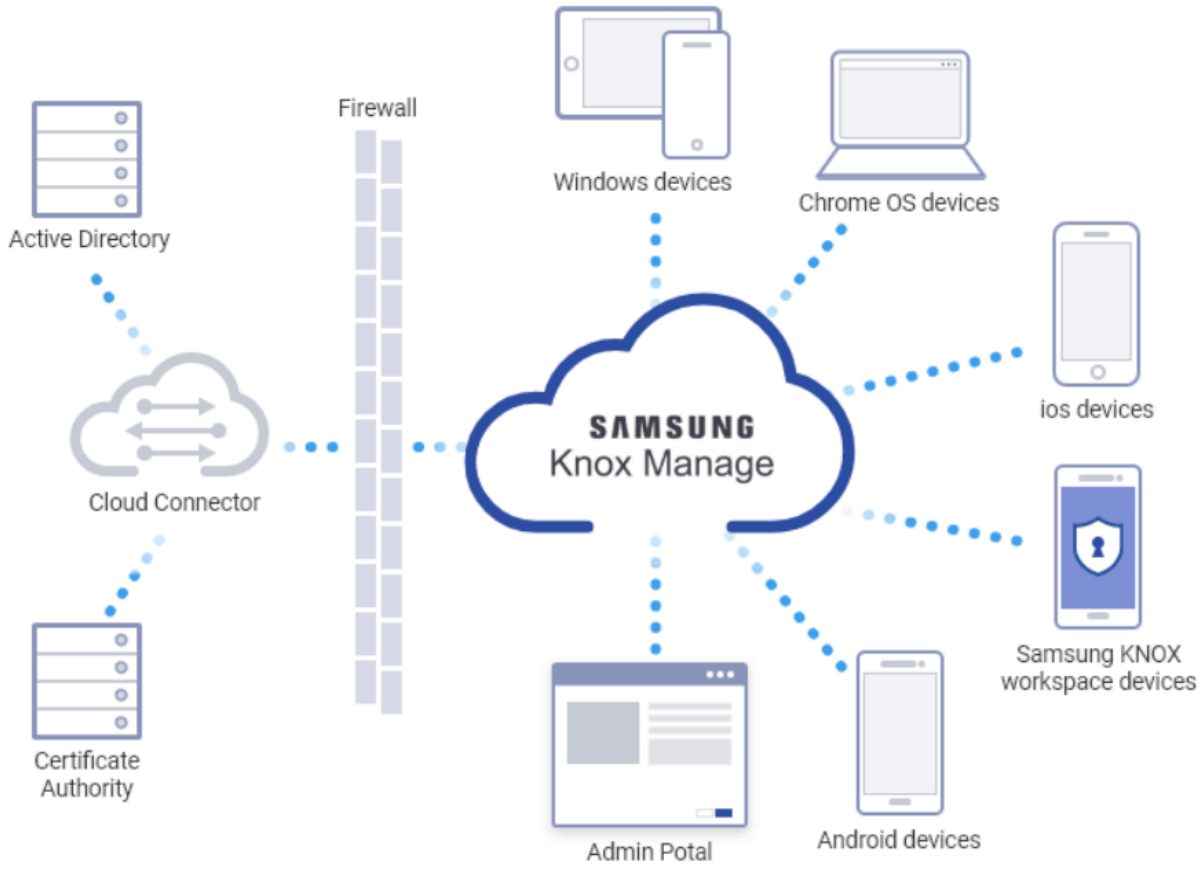
```
{
    "regID": "0501a53f4affdcbb98197f188345ff30c04b-5001",
    "requestID": "01231-22EAX-223442",
    "message": "badgeOption=INCREASE&badgeNumber=1&action=ALERT&alertMessage=Hi",
    "appData": "{id:asdf&passwd:1234}", /* Optional, if the message field is not empty */
}
```

To send a notification:

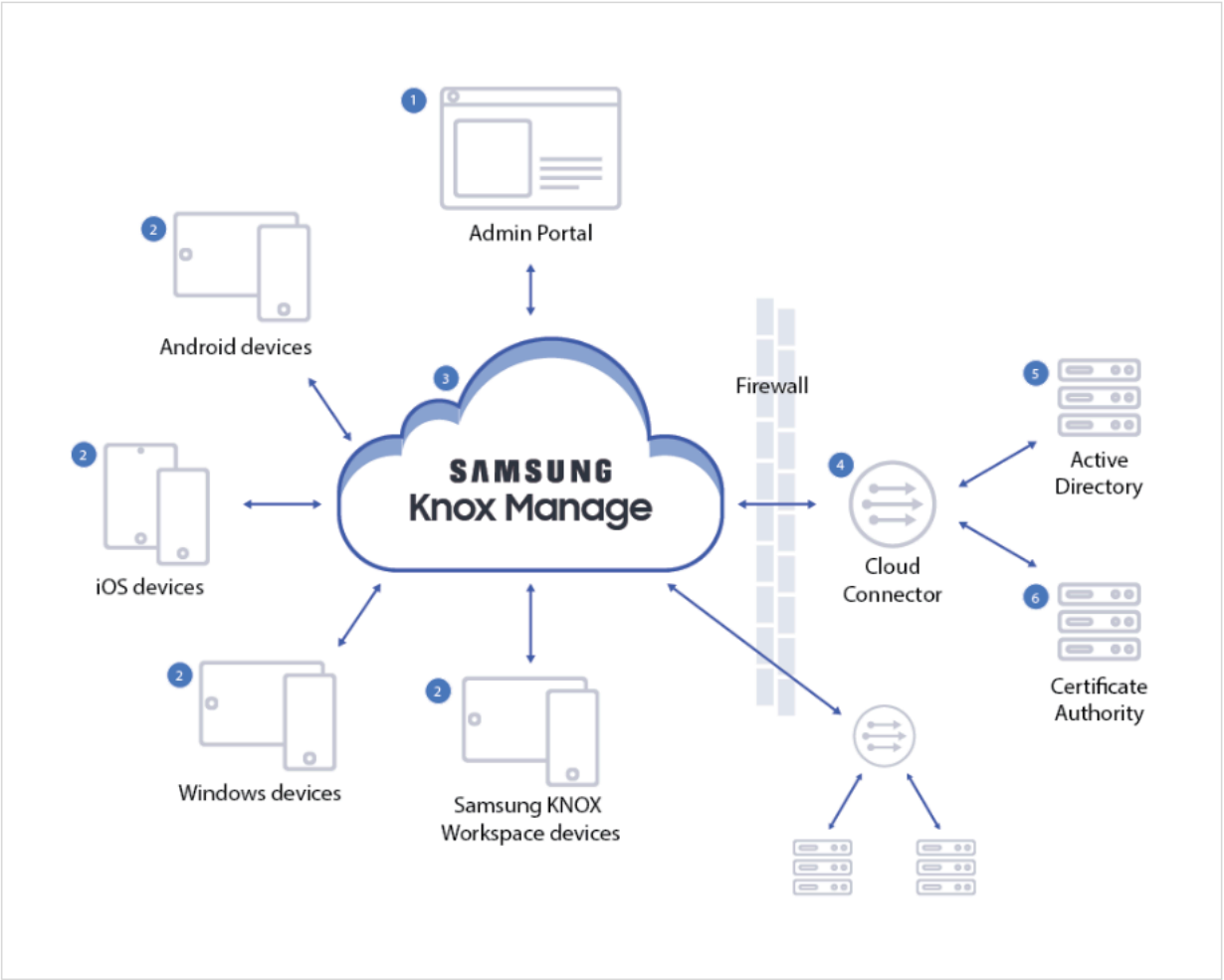
1. Prepare the `appID`, `appSecret`, `regID`, and `requestID`:
    - The `appID` and `appSecret` values are given in the email message that you received when requesting permission to use Tizen push servers.
    - The `regID` value is the one that the application server received from your application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

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

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

173	As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, comprise servers which transmit the generated data messages to the device agents on the devices.  https://docs.samsungknox.com/admin/knox-manage/welcome.htm

174



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

175

Update an existing device profile

22.11 23.03 UAT

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

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

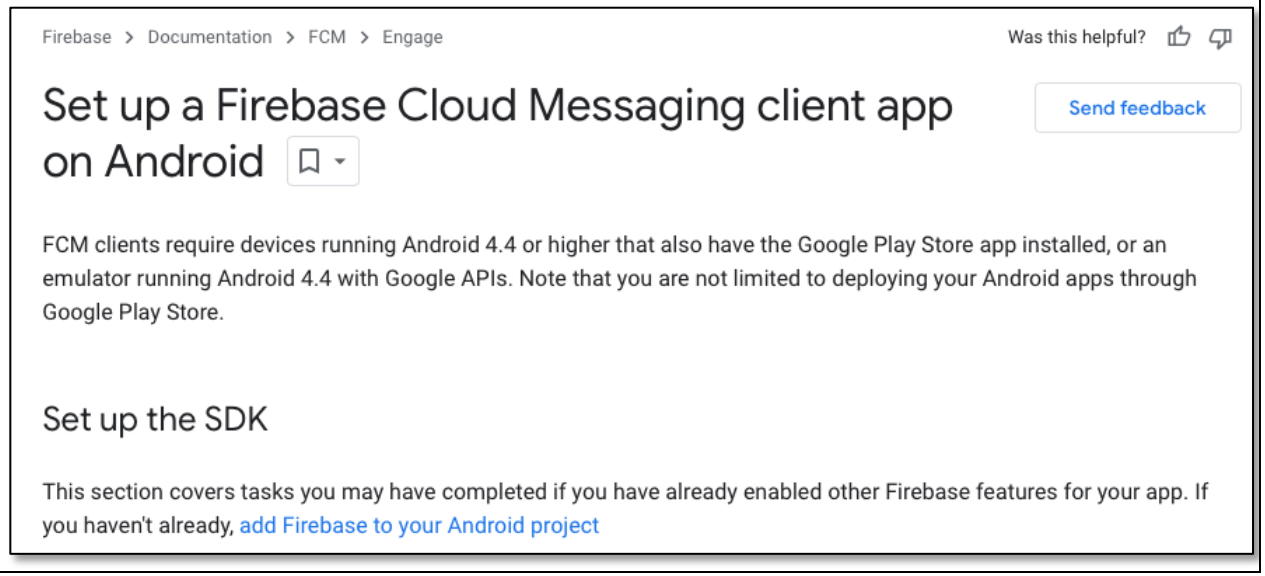
> **NOTE** — An IT admin can select specific devices for push updates from the Knox Configure **PROFILE** or **DEVICES** tabs or at the time a profile is modified. Otherwise, each device utilizing the profile will receive the push update whether intended for each device utilizing that profile or not.

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

Components of Knox Manage

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

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

[1g] each device messaging agent, when executing, to receive the Internet data messages from the secure Internet data connection corresponding to the device executing the device messaging agent, and	Samsung's devices each comprise a "device messaging agent, when executing, to receive the Internet data messages from the secure Internet data connection corresponding to the device executing the device messaging agent." For example, Samsung devices receive data over a secure internet data connection. *See e.g.*,  Firebase > Documentation > FCM > Engage Was this helpful? Set up a Firebase Cloud Messaging client app on Android Send feedback FCM clients require devices running Android 4.4 or higher that also have the Google Play Store app installed, or an emulator running Android 4.4 with Google APIs. Note that you are not limited to deploying your Android apps through Google Play Store. Set up the SDK This section covers tasks you may have completed if you have already enabled other Firebase features for your app. If you haven't already, add Firebase to your Android project

176

177

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`.

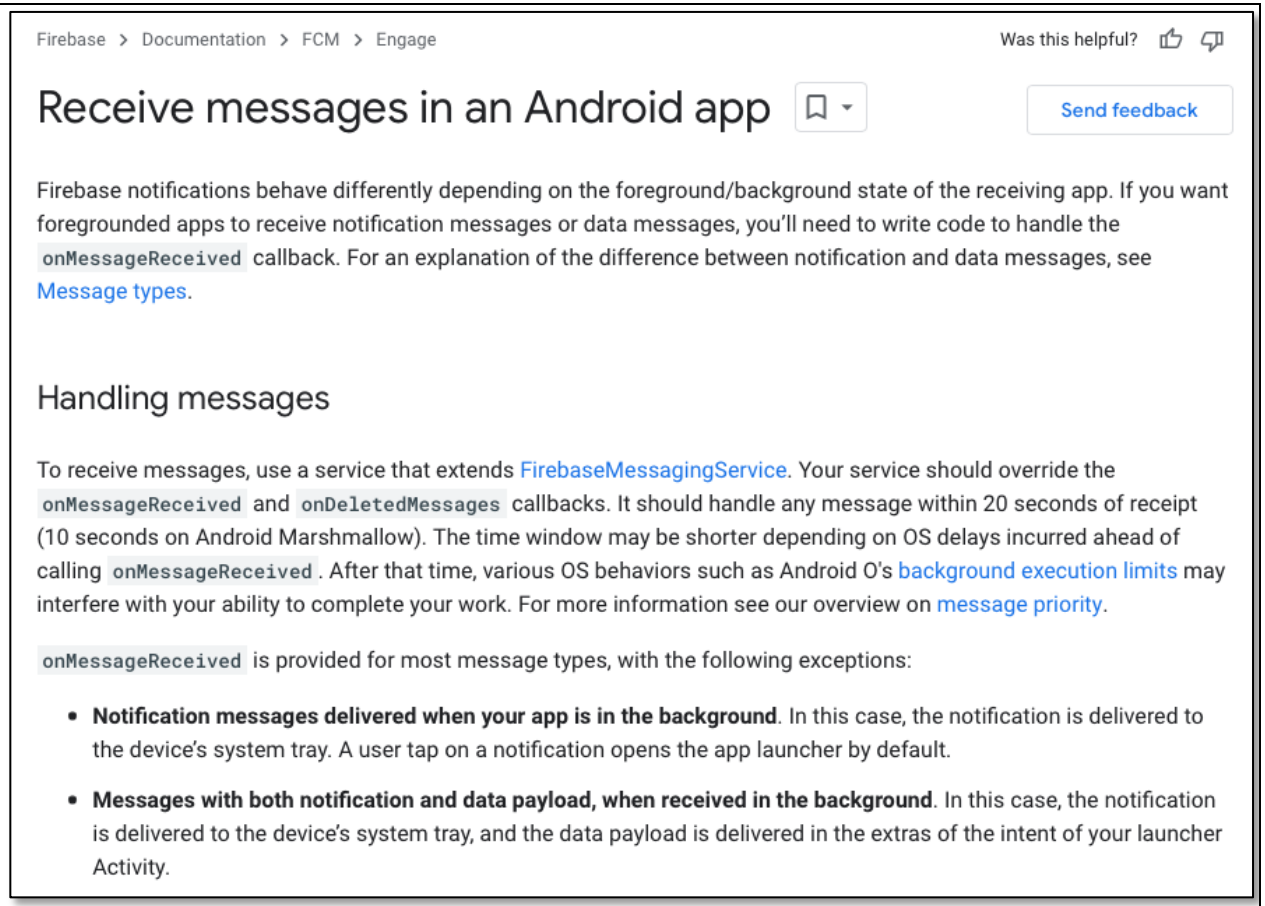
This section describes how to retrieve the token and how to monitor changes to the token. Because the token could be rotated after initial startup, you are strongly recommended to retrieve the latest updated registration token.

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;

178



Firebase > Documentation > FCM > Engage

Was this helpful?

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.

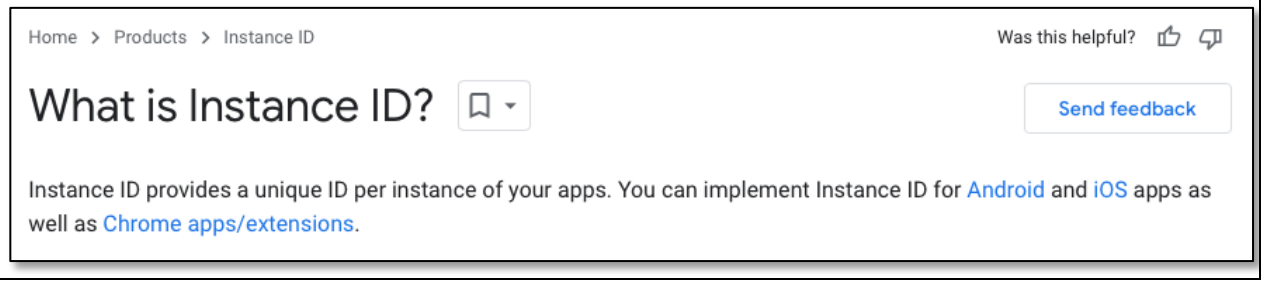
Handling messages

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;



Home > Products > Instance ID

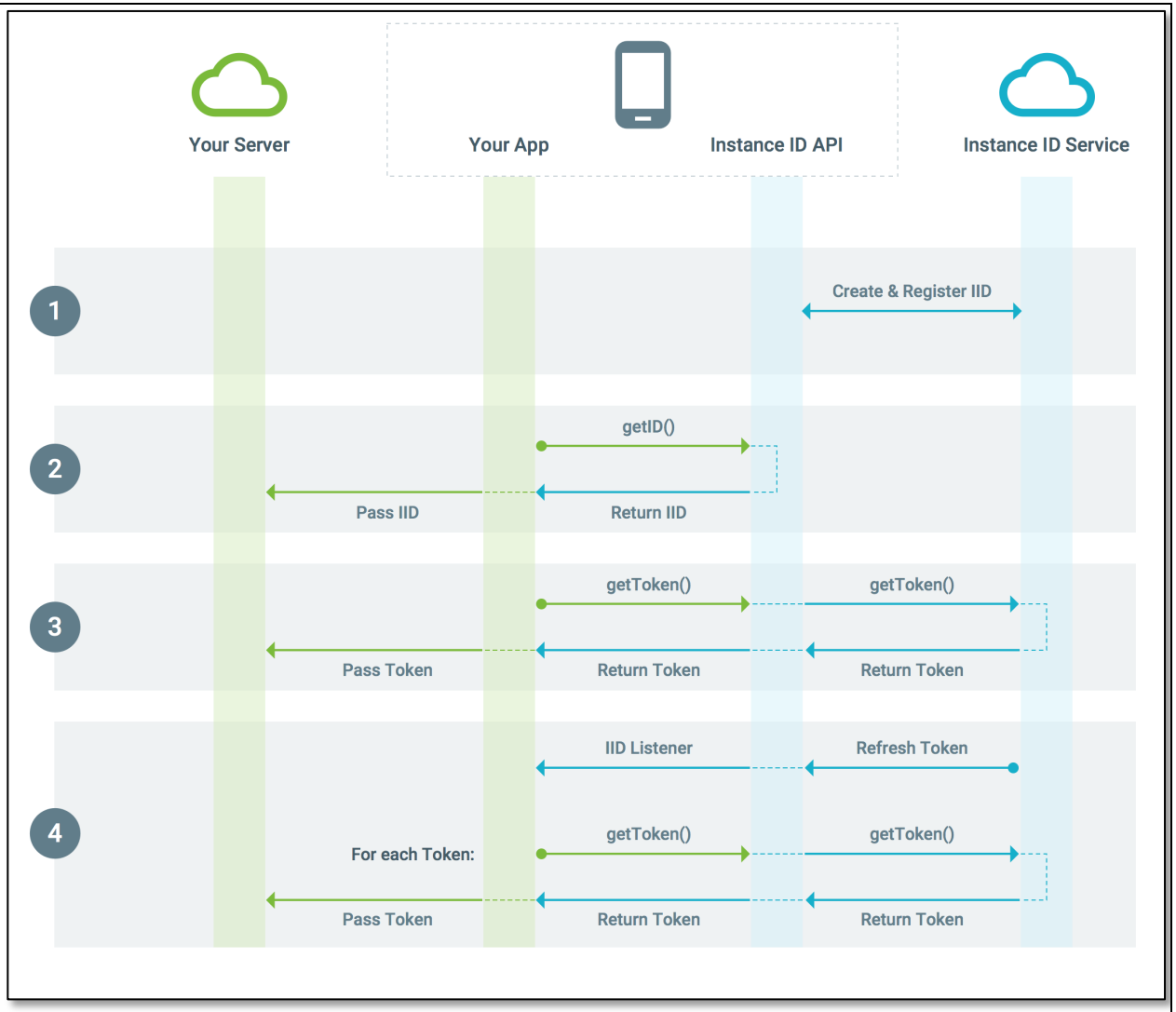
Was this helpful?

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.

179



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

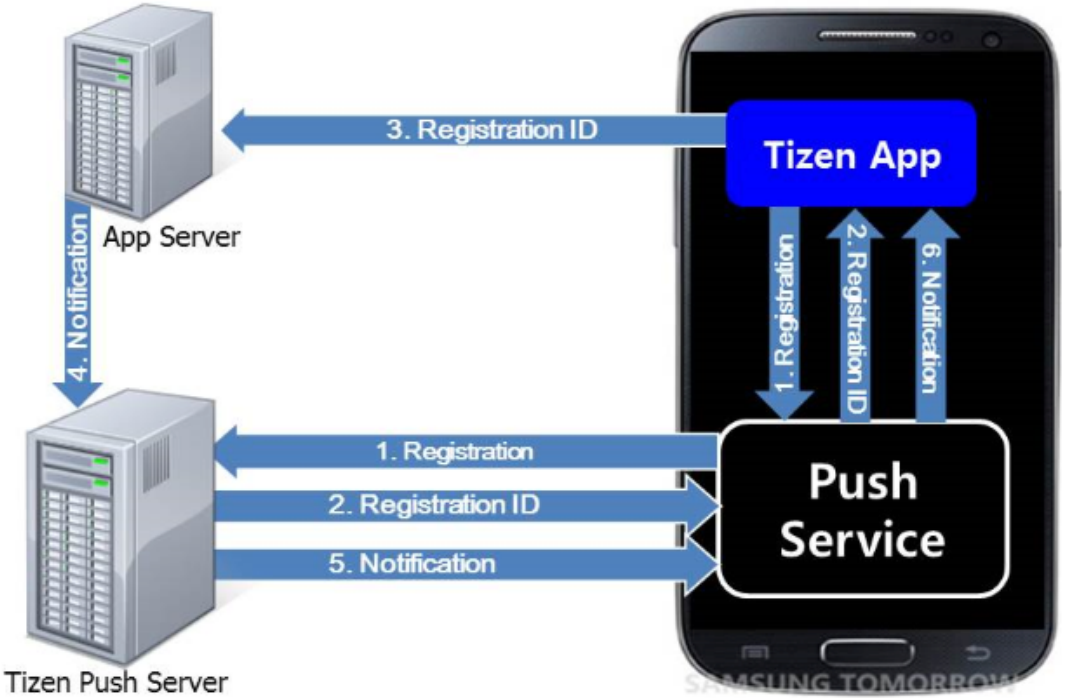
For further example, Samsung's Tizen based devices receive data over a secure internet data connection. *See e.g.,*

180

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.

181

182

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);
```

183

- 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);
}
```

185

Receiving Push Notifications

You can connect to the push service and start receiving push notifications with the `connectService()` method up to Tizen 2.4, or with the `connect()` method since Tizen 3.0. Up to Tizen 2.4, you must pass the `PushNotificationCallback` listener (in mobile and wearable applications) as a parameter in the method to receive push notifications. Since Tizen 3.0, you must pass the `PushRegistrationStateChangeCallback` (in mobile, wearable, and TV applications) and `PushNotificationCallback` callbacks (in mobile, wearable, and TV applications) as parameters in the method. The first callback is called when the registration change state changes. This callback is called at least once, just after the connection is established. The second callback is called when notification messages arrive. You can also pass the `ErrorCallback` as a parameter to be called if the connection request fails.

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

- When the application is running

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

- When the application is not running

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

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

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

    For more information, see the Retrieving Messages When Launched by the Push Service use case.

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

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

    For more information, see the Retrieving Missed Push Messages use case.

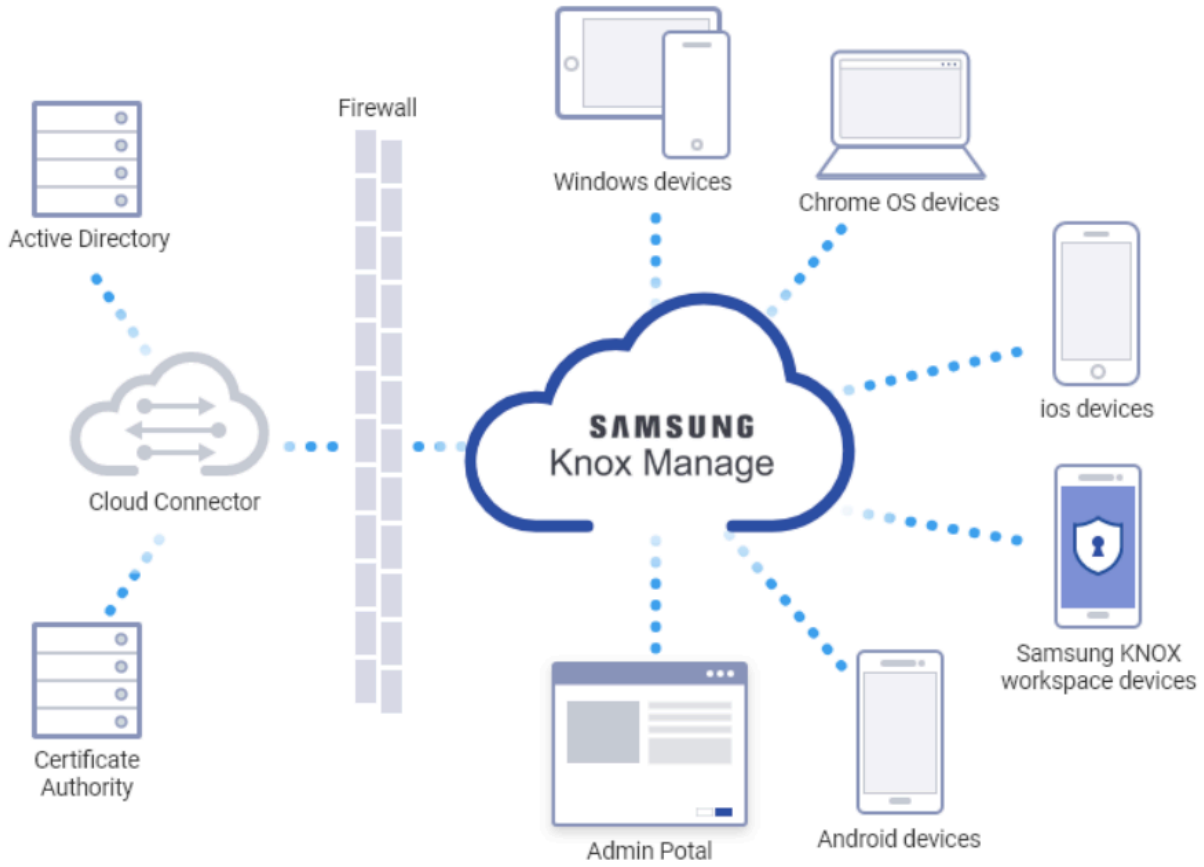
    The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification.

  - Discard the notification.

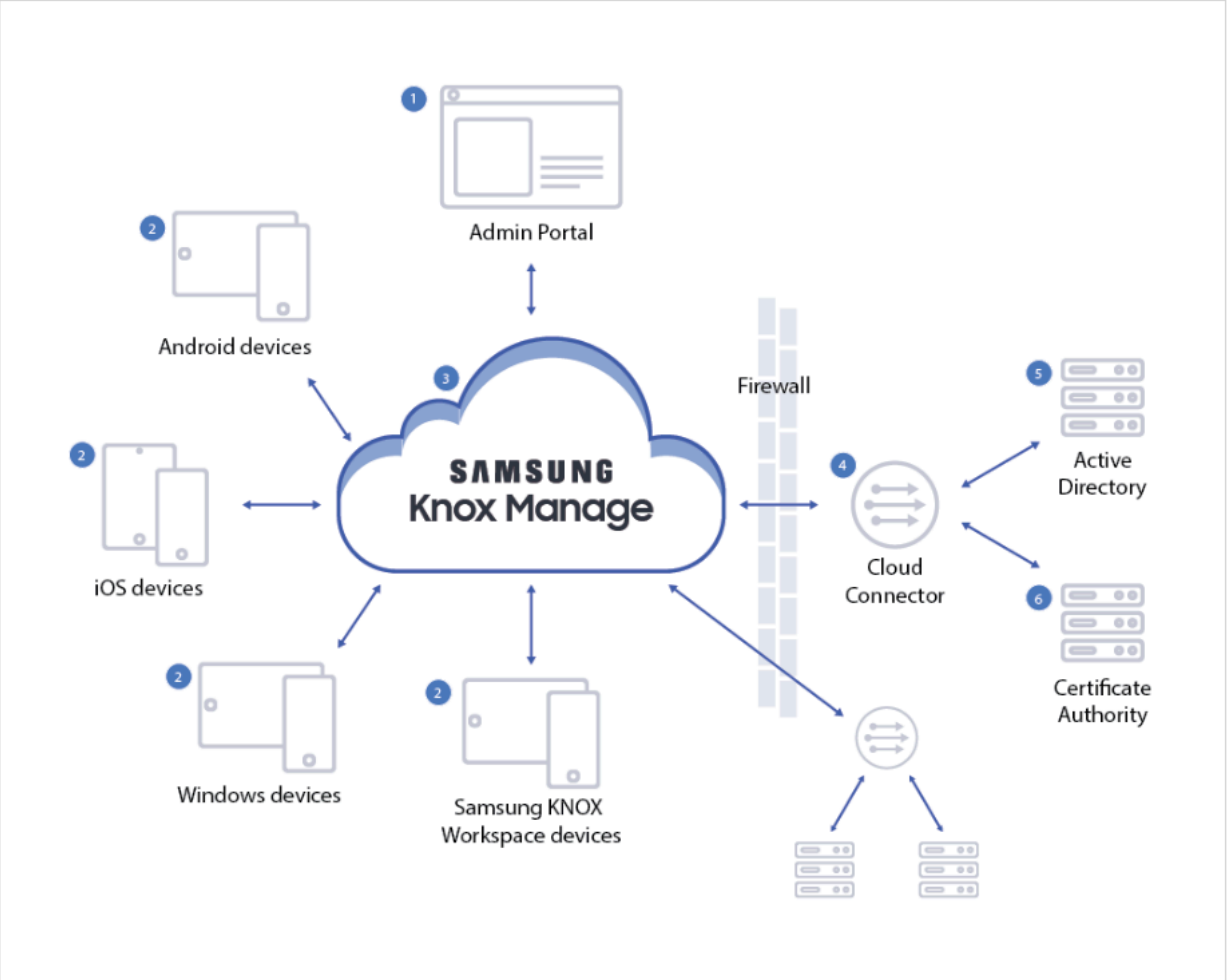
    This happens when the action is set to `DISCARD` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service discards the notification unless the application is running.

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

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

186	Samsung Knox servers managing those phones and devices, comprise devices receiving data over a secure connection.  https://docs.samsungknox.com/admin/knox-manage/welcome.htm

187



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

188

Update an existing device profile

22.11 23.03 UAT

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

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

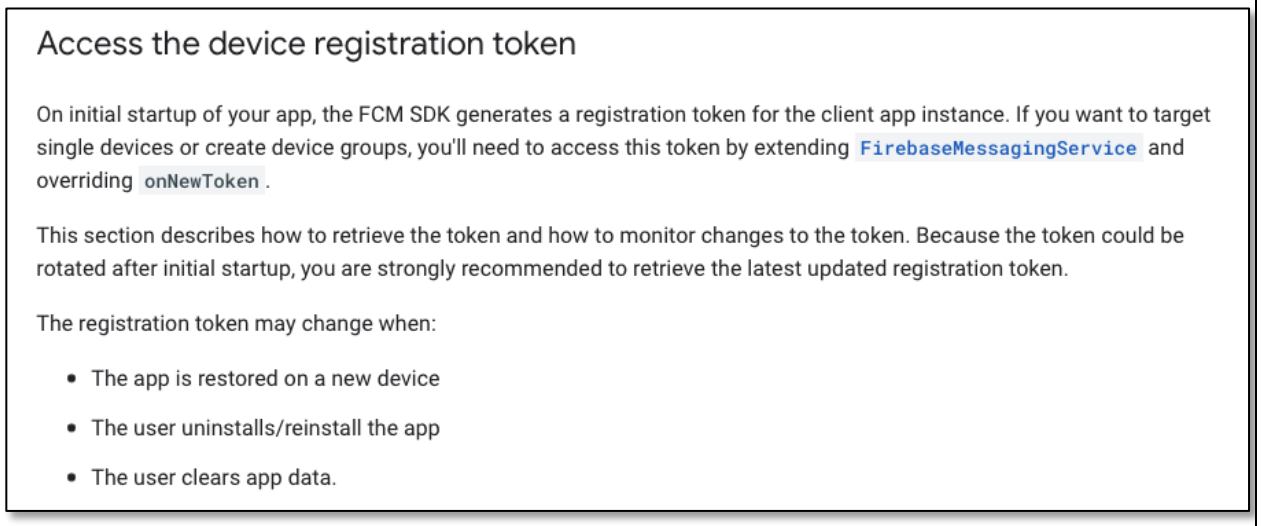
> **NOTE** — An IT admin can select specific devices for push updates from the Knox Configure **PROFILE** or **DEVICES** tabs or at the time a profile is modified. Otherwise, each device utilizing the profile will receive the push update whether intended for each device utilizing that profile or not.

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

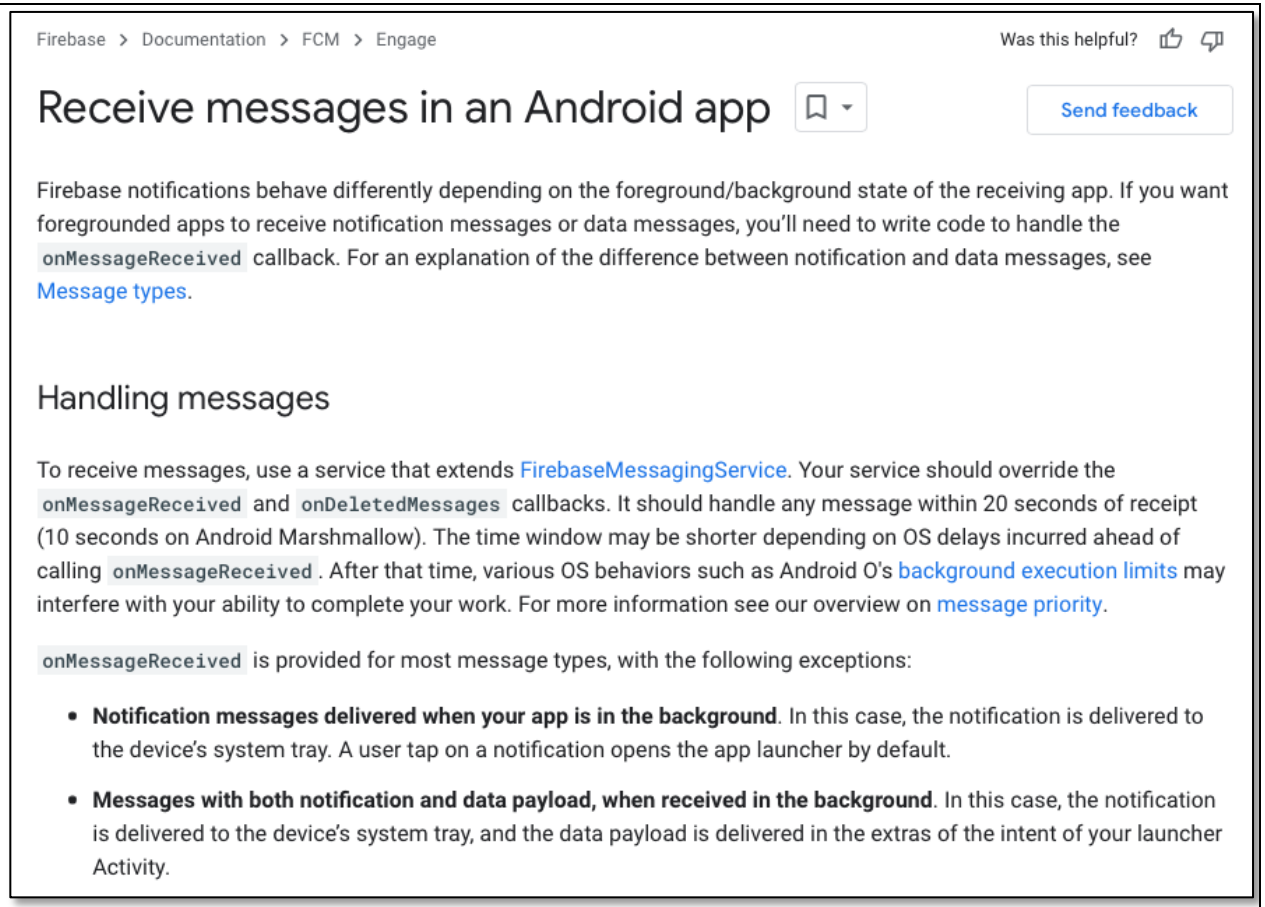
Components of Knox Manage

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

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

189	[1h] to, for each received message, map the application identifier in the message to a software process corresponding to the application identifier, and forward the application data in the message to the software process via a secure interprocess communication service.	Samsung's devices "for each received message, map the application identifier in the message to a software process corresponding to the application identifier, and forward the application data in the message to the software process via a secure interprocess communication service." For example, Samsung devices map the identifier in the message to a corresponding software process and forward the data to the application via a secure service. *See e.g.*,  https://firebase.google.com/docs/cloud-messaging/android/client;

190



Firebase > Documentation > FCM > Engage

Was this helpful?

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.

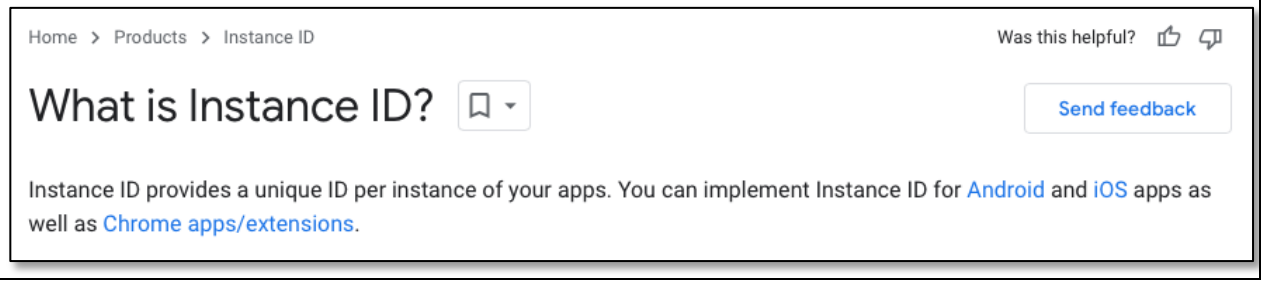
Handling messages

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;



Home > Products > Instance ID

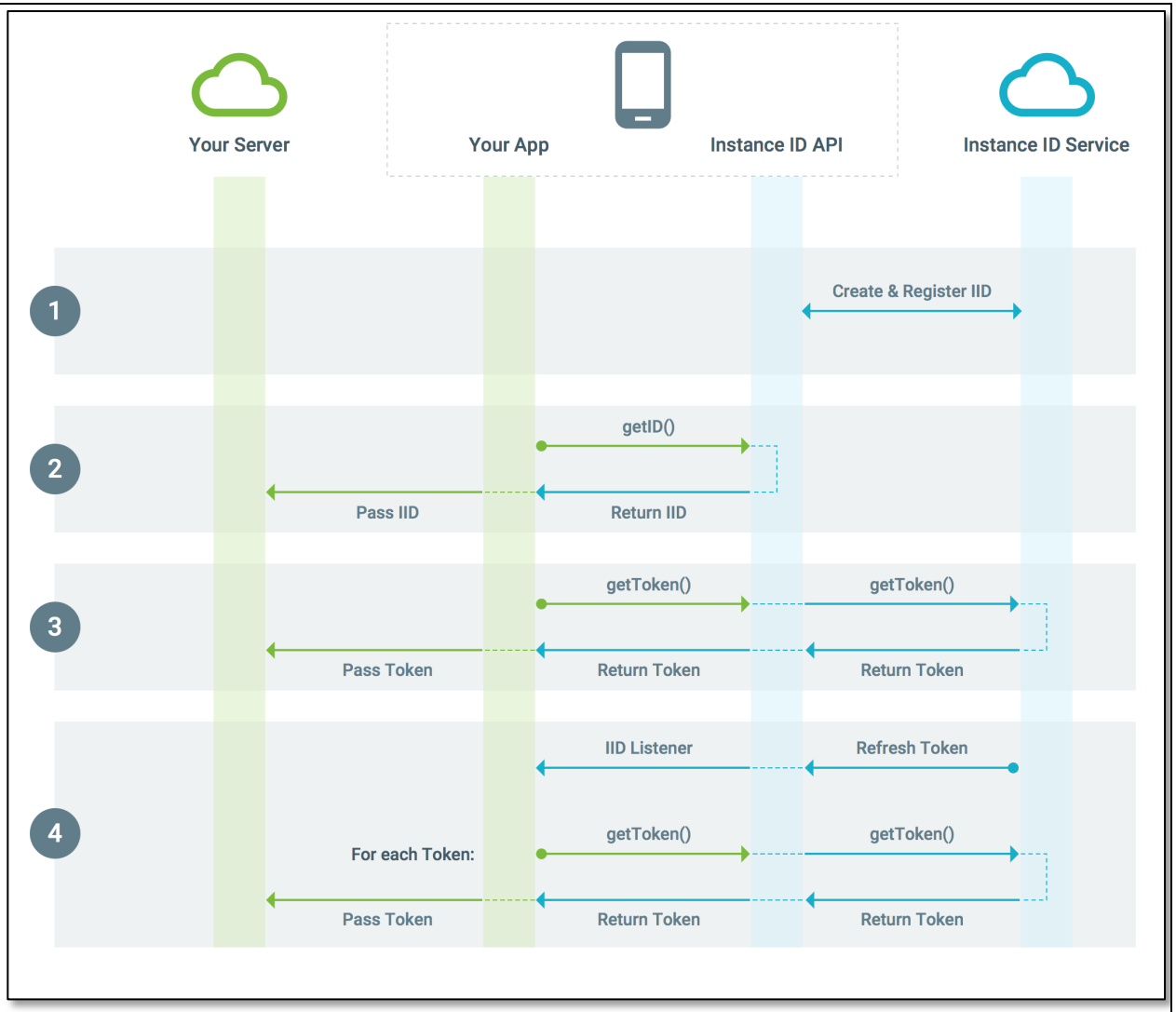
Was this helpful?

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.

191



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

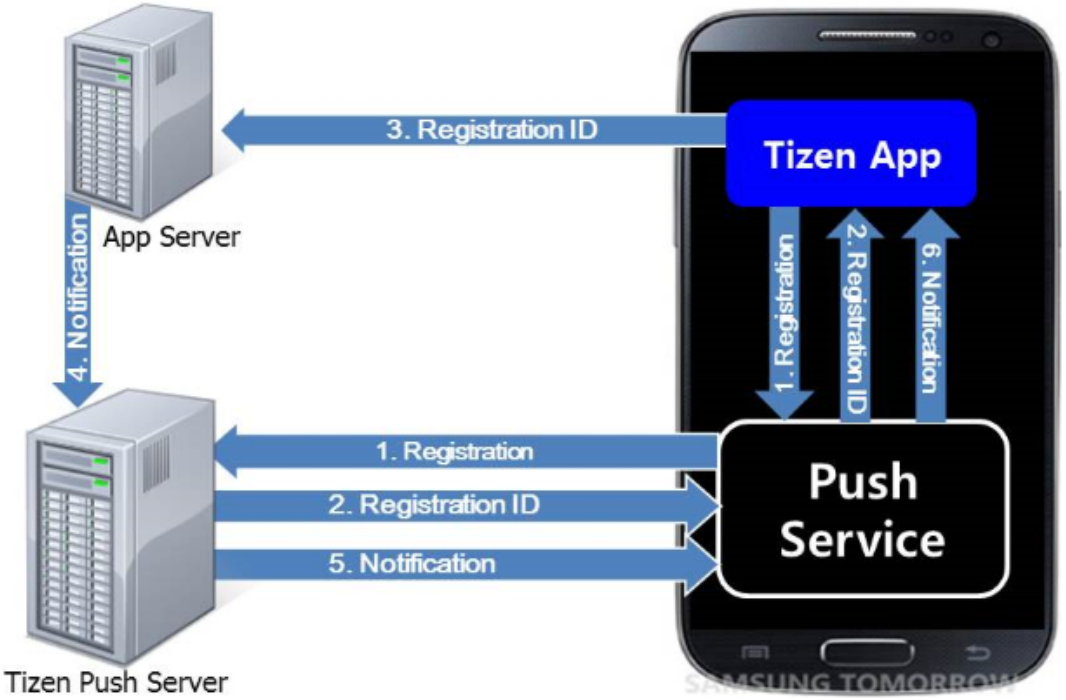
For further example, Samsung's Tizen based devices map the identifier in the message to a corresponding software process and forward the data to the application via a secure service. *See e.g.,*

192

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.

194

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);
}
```

197

Receiving Push Notifications

You can connect to the push service and start receiving push notifications with the `connectService()` method up to Tizen 2.4, or with the `connect()` method since Tizen 3.0. Up to Tizen 2.4, you must pass the `PushNotificationCallback` listener (in mobile and wearable applications) as a parameter in the method to receive push notifications. Since Tizen 3.0, you must pass the `PushRegistrationStateChangeCallback` (in mobile, wearable, and TV applications) and `PushNotificationCallback` callbacks (in mobile, wearable, and TV applications) as parameters in the method. The first callback is called when the registration change state changes. This callback is called at least once, just after the connection is established. The second callback is called when notification messages arrive. You can also pass the `ErrorCallback` as a parameter to be called if the connection request fails.

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

- When the application is running

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

- When the application is not running

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

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

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

    For more information, see the Retrieving Messages When Launched by the Push Service use case.

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

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

    For more information, see the Retrieving Missed Push Messages use case.

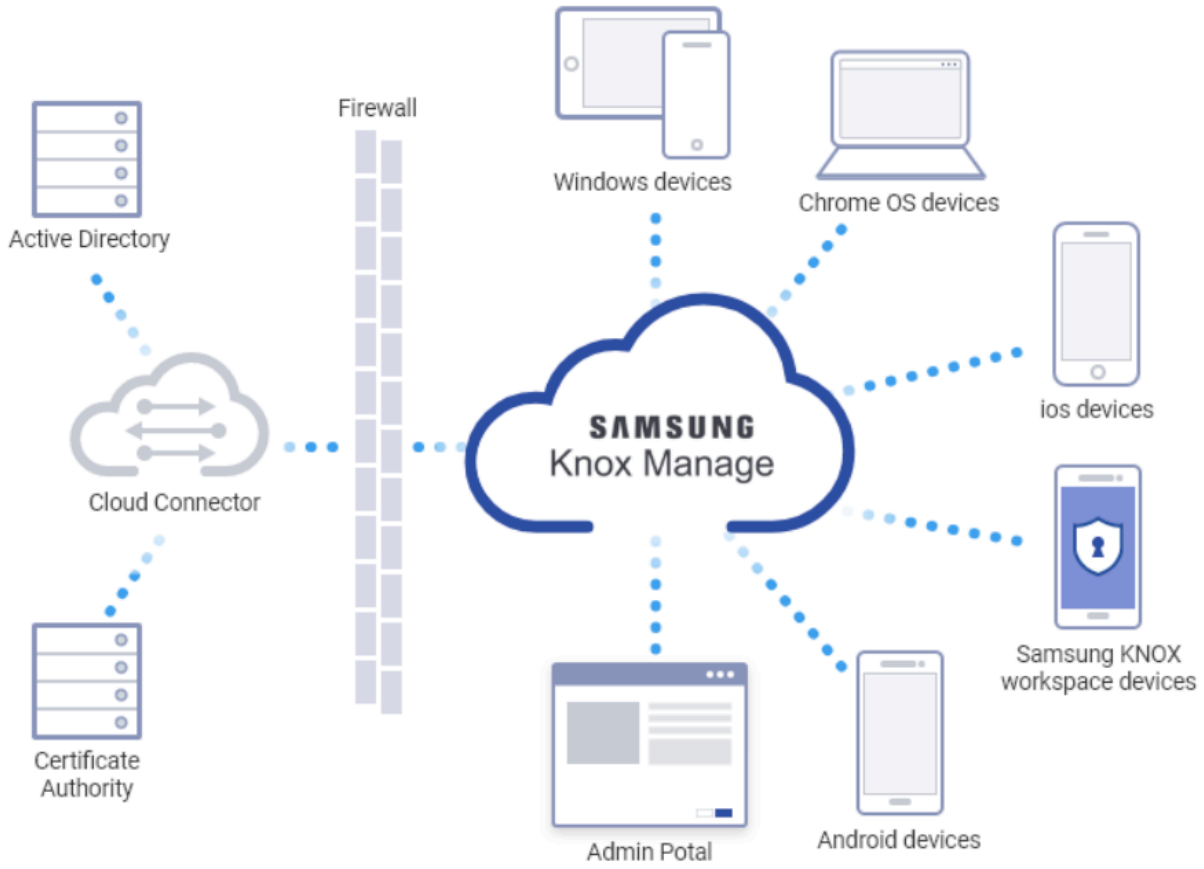
    The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification.

  - Discard the notification.

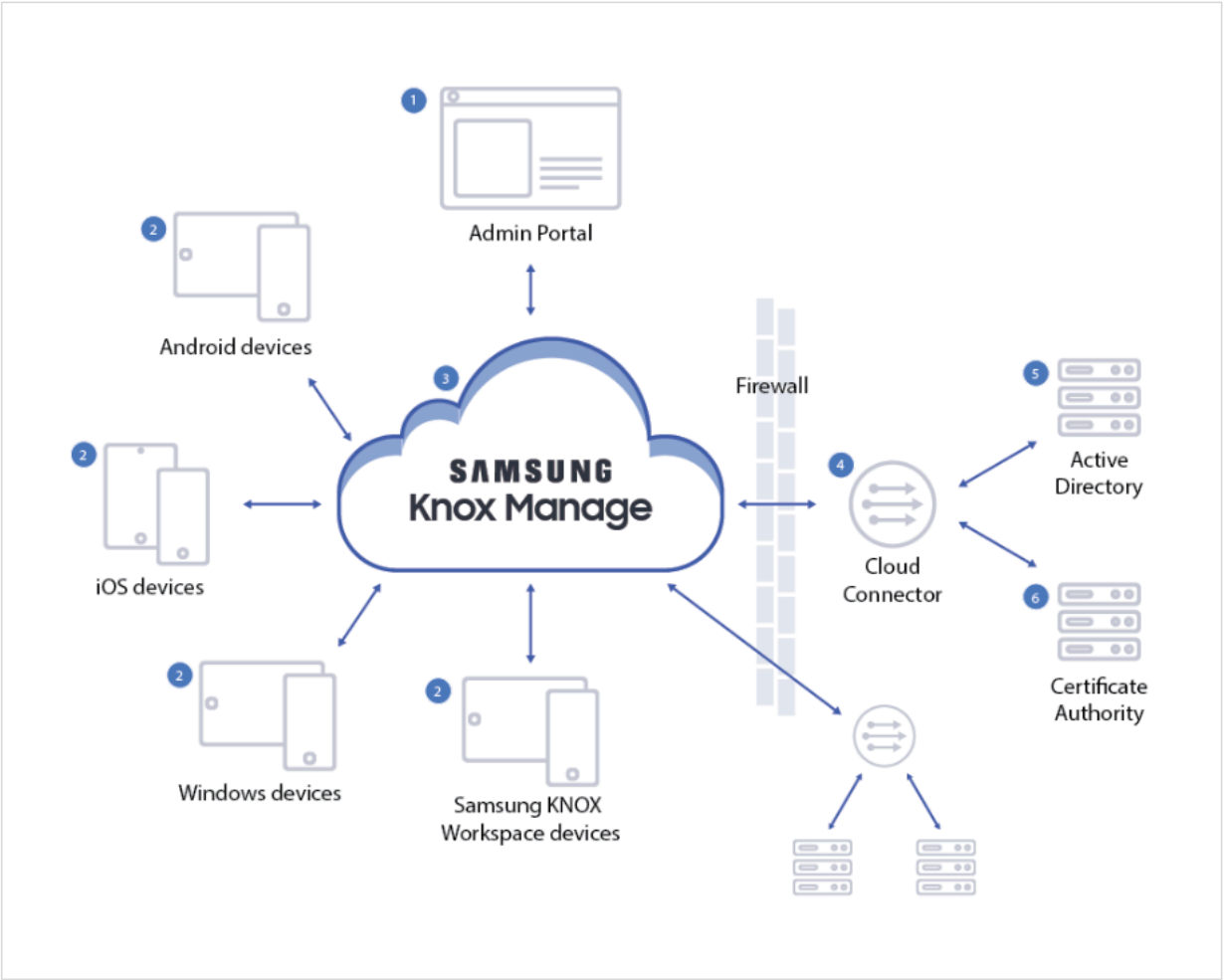
    This happens when the action is set to `DISCARD` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service discards the notification unless the application is running.

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

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

198	Samsung Knox servers managing those phones and devices, comprise mapping the identifier in the message to a corresponding software process and forward the data to the application via a secure service.  https://docs.samsungknox.com/admin/knox-manage/welcome.htm

199



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

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

200 **Claim 2**

2. The network system of claim 1, the network message server further to collect and buffer multiple requests to transmit application data to a particular one of the devices.	The accused instrumentalities comprise the "network system of claim 1, the network message server further to collect and buffer multiple requests to transmit application data to a particular one of the devices." *See supra* 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. *See, e.g.*, 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.

201

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();
```

202

- 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

203

Claim 4

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

Managing Security

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

- Keep the push application ID confidential.

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

- Do not store the registration ID on the device.

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

- Encrypt sensitive information.

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

- Do not hardcode the AppSecret in the source code.

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

Sending Push Notifications

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

The following example shows a sample push notification:

- URI: See the Push RQM (Request Manager) server URLs table.
- Method: HTTP POST
- Header:

```
appID: 1234567890987654
appSecret: dYo/o/m11gmWmjs7+5f+2zLNVOc=
```

- Body:

```
{
    "regID": "0501a53f4affdcbb98197f188345ff30c04b-5001",
    "requestID": "01231-22EAX-223442",
    "message": "badgeOption=INCREASE&badgeNumber=1&action=ALERT&alertMessage=Hi",
    "appData": "{id:asdf&passwd:1234}", /* Optional, if the message field is not empty */
}
```

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

Claim 5

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

205

Managing Security

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

- Keep the push application ID confidential.

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

- Do not store the registration ID on the device.

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

- Encrypt sensitive information.

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

- Do not hardcode the AppSecret in the source code.

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

Sending Push Notifications

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

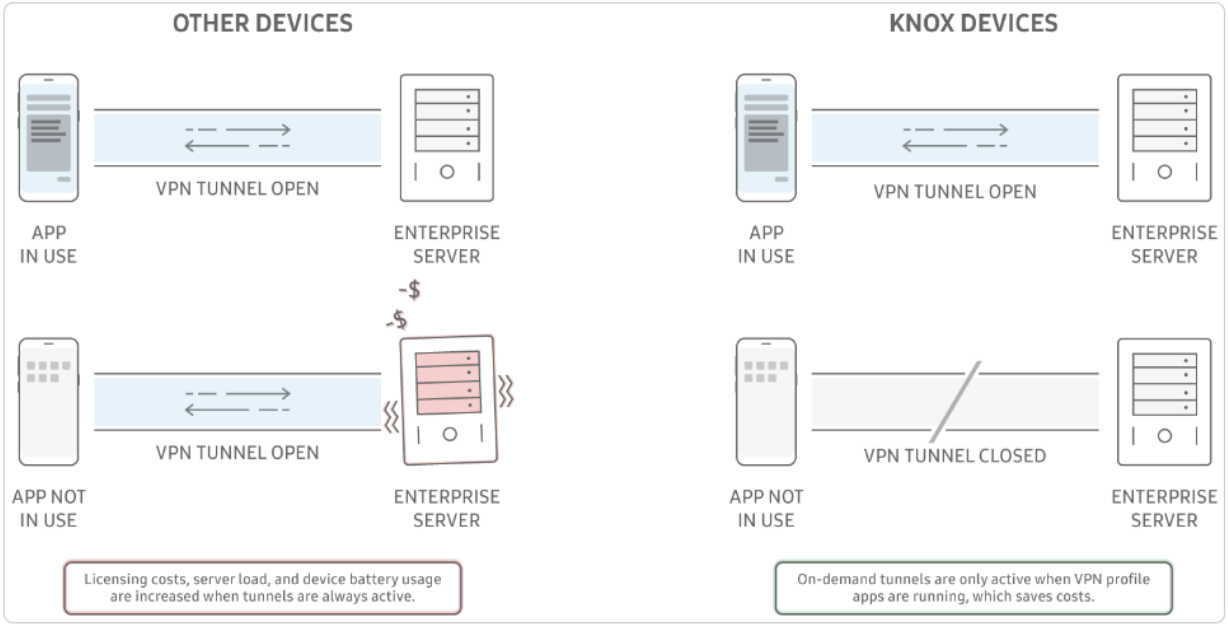
The following example shows a sample push notification:

- URI: See the Push RQM (Request Manager) server URLs table.
- Method: HTTP POST
- Header:

```
appID: 1234567890987654
appSecret: dYo/o/m11gmWmjs7+5f+2zLNVOc=
```

- Body:

```
{
    "regID": "0501a53f4affdcbb98197f188345ff30c04b-5001",
    "requestID": "01231-22EAX-223442",
    "message": "badgeOption=INCREASE&badgeNumber=1&action=ALERT&alertMessage=Hi",
    "appData": "{id:asdf&passwd:1234}", /* Optional, if the message field is not empty */
}
```

207	https://docs.tizen.org/application/native/guides/messaging/push/#managing-security As another example, the Firebase framework encrypts messages in transit. See: https://firebase.google.com/docs/cloud-messaging/concept-options ("Encryption for data messages The Android Transport Layer (see FCM architecture) uses point-to-point encryption."). Likewise, messages sent with Knox Manage instrumentalities are also encrypted. See https://docs.samsungknox.com/admin/fundamentals/whitepaper/core-platform-security/knox-vault/; https://docs.samsungknox.com/admin/fundamentals/whitepaper/network-security/virtual-private-networks/: **Unique advantages of Knox VPN framework** The Knox Platform VPN framework supports all common VPN types, protocols, and configuration options. When deploying VPN solutions, enterprise IT admins must ensure VPN deployments work smoothly, don't waste server resources, limit the VPN solution licensing costs, and enforce strict security policies that prevent data leakage. The following is an example showing how Knox on-demand VPNs save cost: 

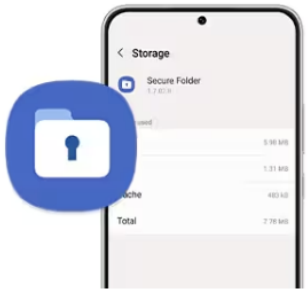
Claim 6

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

208

209

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



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

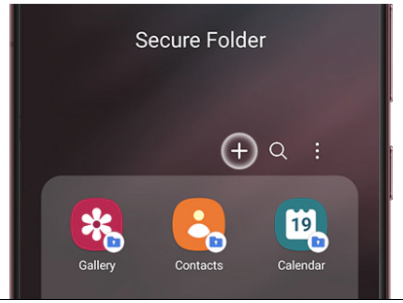
https://www.samsung.com/uk/support/mobile-devices/what-is-the-secure-folder-and-how-do-i-use-it/

Add apps or files in Secure Folder

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

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

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



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

Claim 9

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

210

211

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.

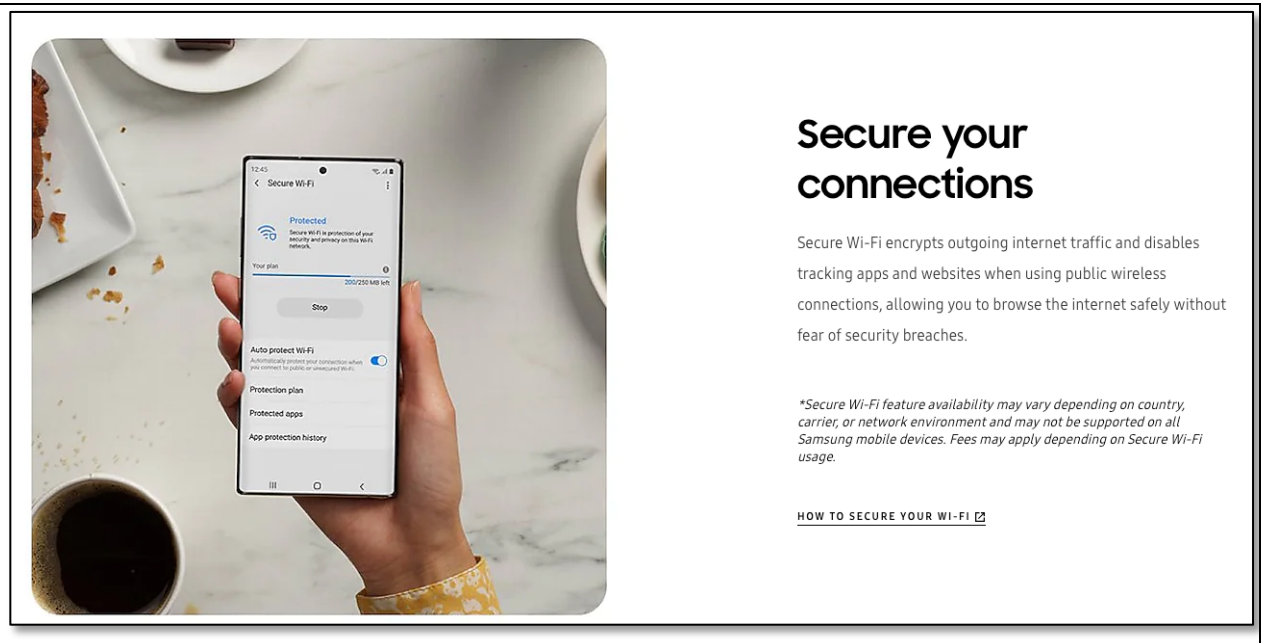
Data encryption

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.

212



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

213

Firebase > Documentation > FCM > Engage

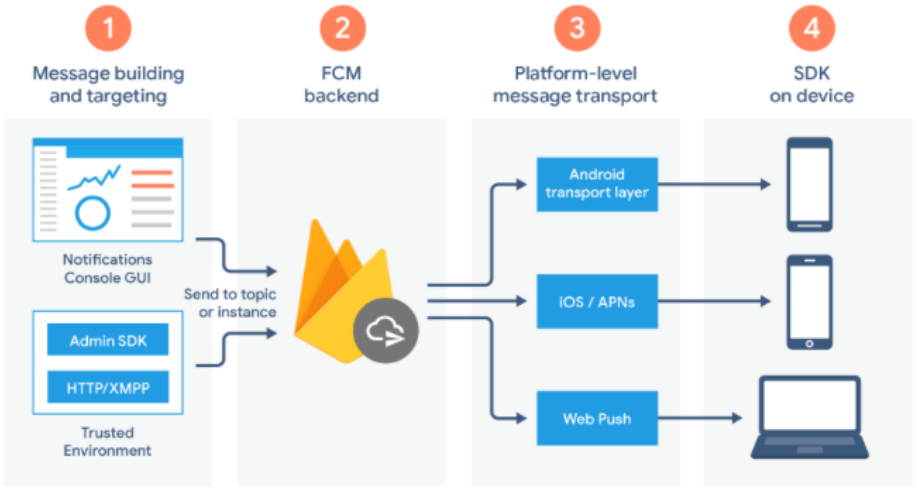
Was this helpful?

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.



214

2. The FCM backend, which (among other functions) accepts message requests, performs fanout of messages via topics, and generates message metadata such as the message ID.
3. A platform-level transport layer, which routes the message to the targeted device, handles message delivery, and applies platform-specific configuration where appropriate. This transport layer includes:
    - Android transport layer (ATL) for Android devices with Google Play services
    - Apple Push Notification service (APNs) for Apple devices
    - Web push protocol for web apps

    ★ **Note:** Platform-level transport layers are outside the core FCM product. FCM messages routed to a platform-level transport layer may be subject to terms specific to that platform rather than FCM's terms of service. Android message routing via ATL falls under the Google APIs terms of service.
4. The FCM SDK on the user's device, where the notification is displayed or the message is handled according to the app's foreground/background state and any relevant application logic.

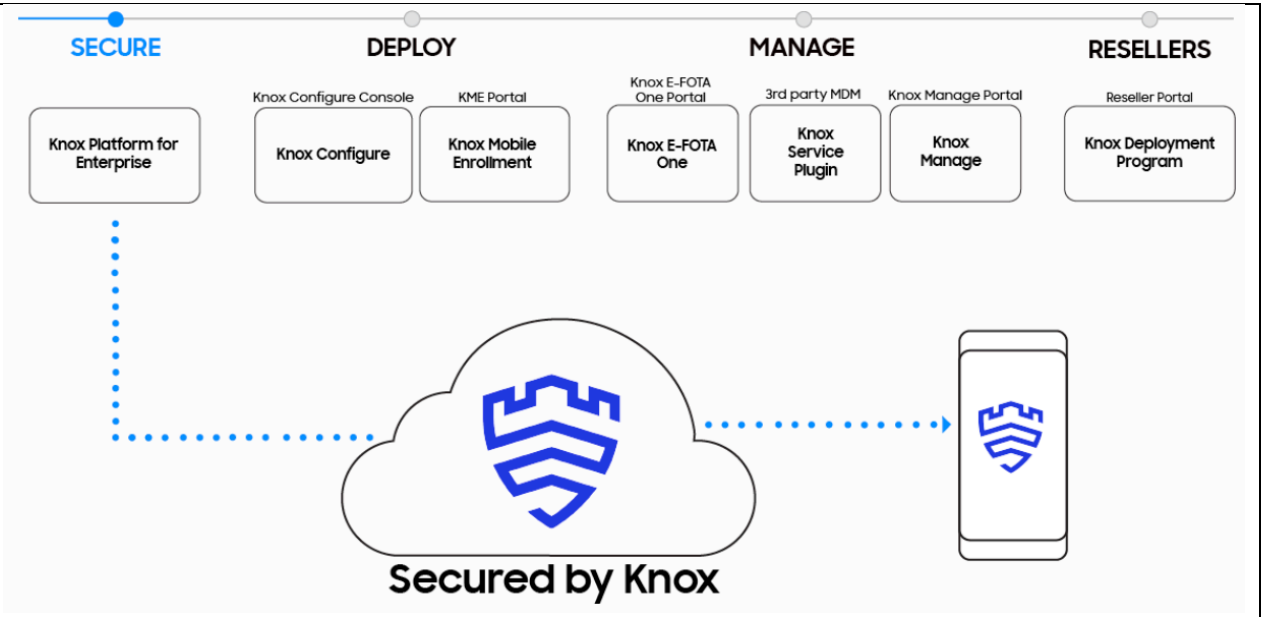
https://firebase.google.com/docs/cloud-messaging/fcm-architecture;

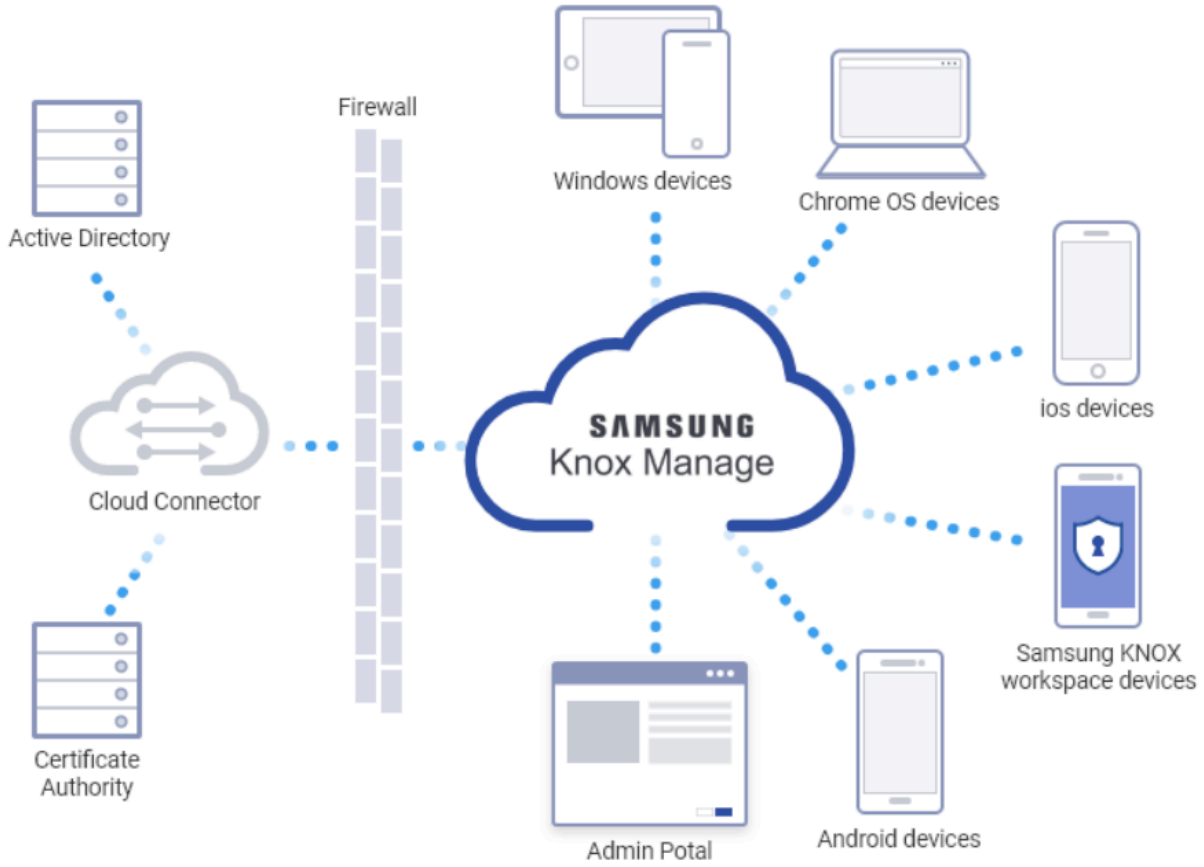
Encryption for data messages

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

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

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

215	 https://docs.samsungknox.com/admin/fundamentals/welcome.htm

216	 https://docs.samsungknox.com/admin/knox-manage/welcome.htm The message content that is passed from messaging servers of the accused instrumentalities and end user devices for which the message content is intended is separately secured, depending on the messaging service used. For example, the Tizen Push Server receives encrypted information from app servers and transmits such encrypted information on to device messaging agents residing on user devices, which in turn decrypt received messages.

Managing Security

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

- Keep the push application ID confidential.

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

- Do not store the registration ID on the device.

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

- Encrypt sensitive information.

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

- Do not hardcode the AppSecret in the source code.

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

Sending Push Notifications

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

The following example shows a sample push notification:

- URI: See the Push RQM (Request Manager) server URLs table.
- Method: HTTP POST
- Header:

```
appID: 1234567890987654
appSecret: dYo/o/m11gmWmjs7+5f+2zLNVOc=
```

- Body:

```
{
    "regID": "0501a53f4affdcbb98197f188345ff30c04b-5001",
    "requestID": "01231-22EAX-223442",
    "message": "badgeOption=INCREASE&badgeNumber=1&action=ALERT&alertMessage=Hi",
    "appData": "{id:asdf&passwd:1234}", /* Optional, if the message field is not empty */
}
```

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

Claim 10

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

218

Claim 11

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

Claim 12

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

219

Claim 13

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

Claim 14

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

Claim 16

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

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

221

Claim 17

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

Claim 18

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

Exhibit C - U.S. Patent No. 9,615,192 ("'192 Patent")

Accused Devices: Samsung's push messaging servers (e.g., Samsung's Knox and Tizen servers), and all versions and variations thereof since the issuance of the asserted patent.

Claim 1

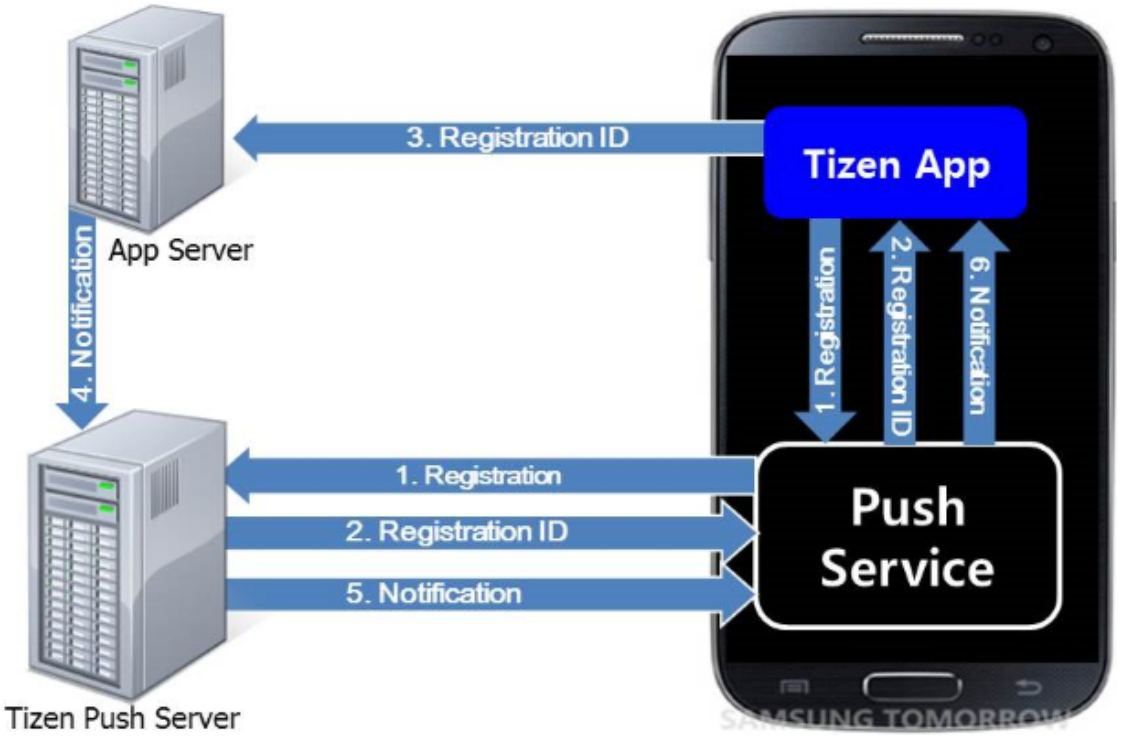
Issued Claim(s)	Public Documentation
[1pre]. A message link server comprising:	Samsung's push messaging servers are each "message link server[s]." *See, e.g.,*

223

Architecture

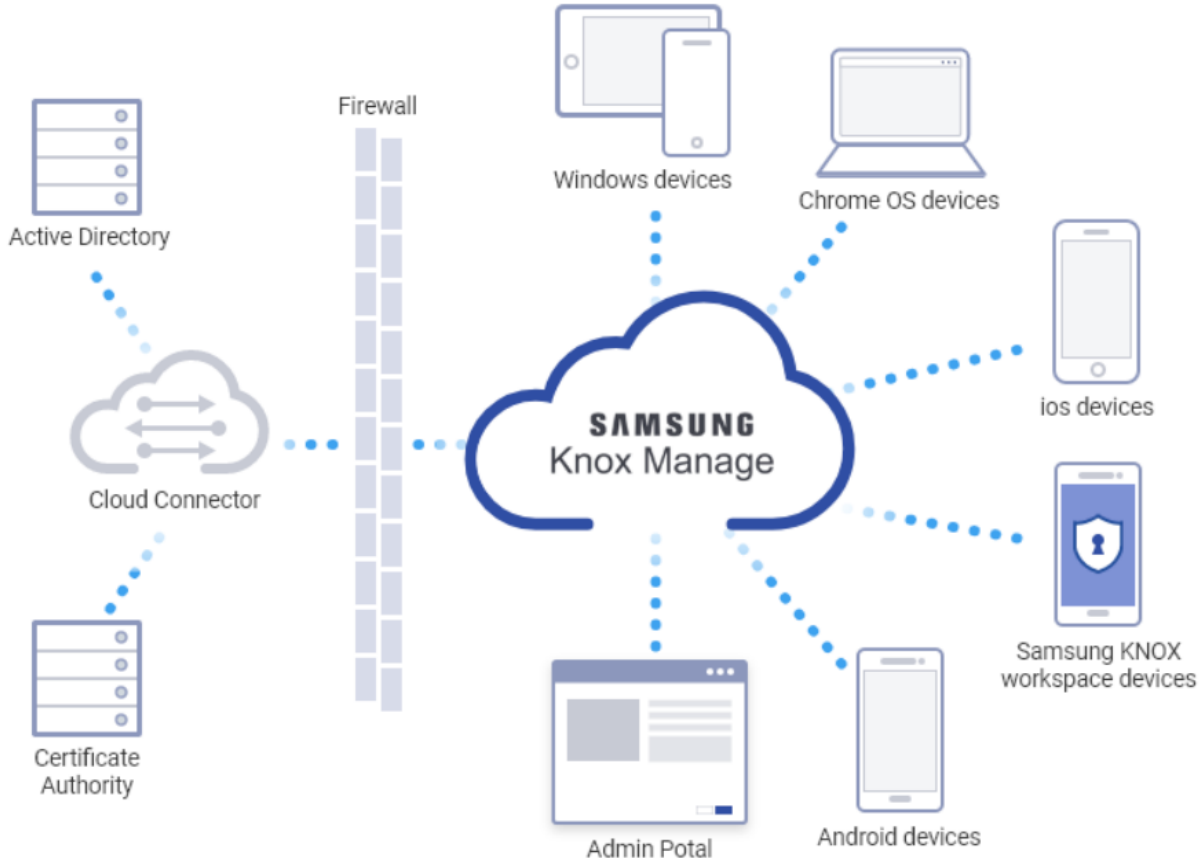
The architecture of the Tizen Push service is described in detail in the mobile native Push guide.

Figure: Service architecture

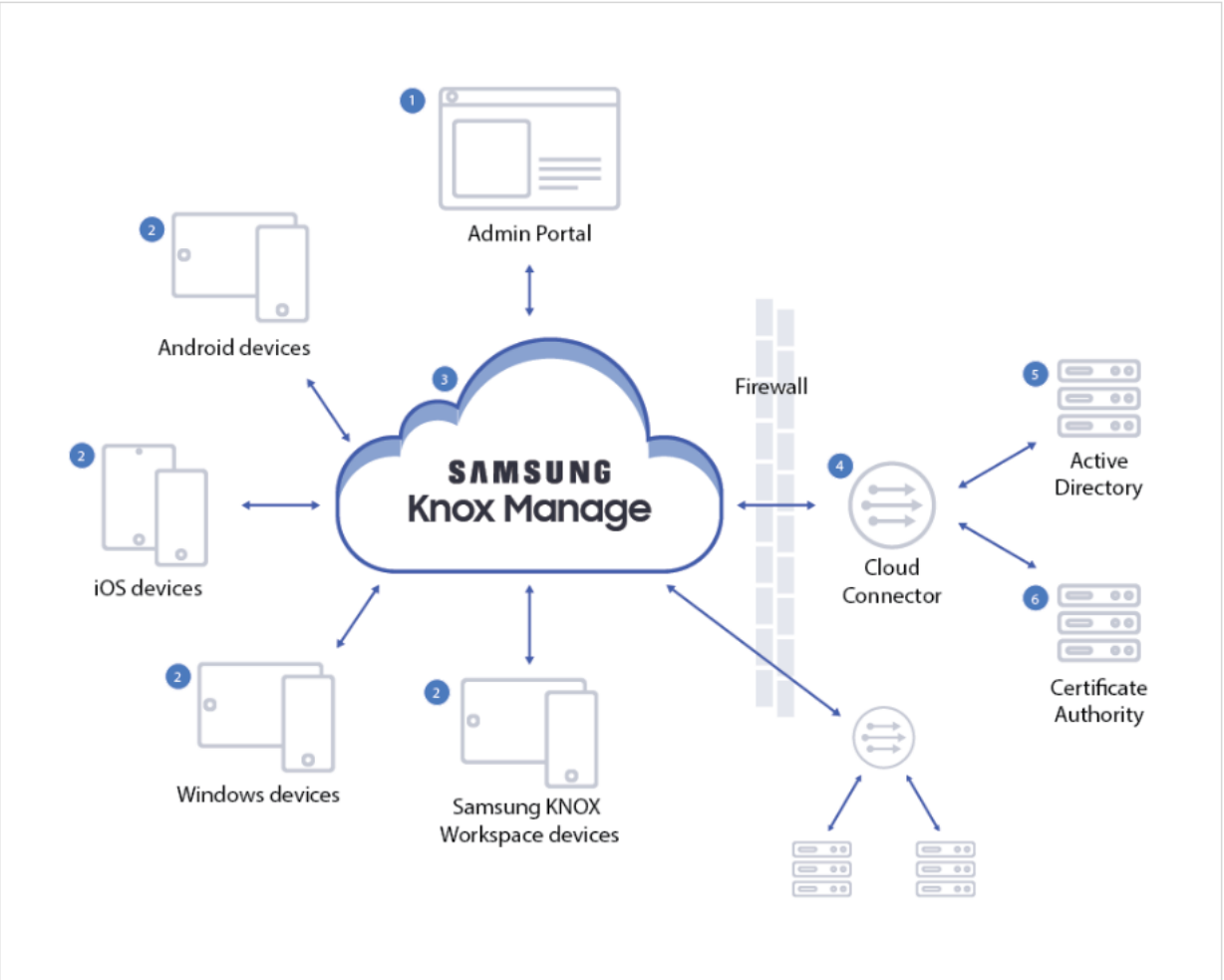


To receive push notifications for your application:

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.

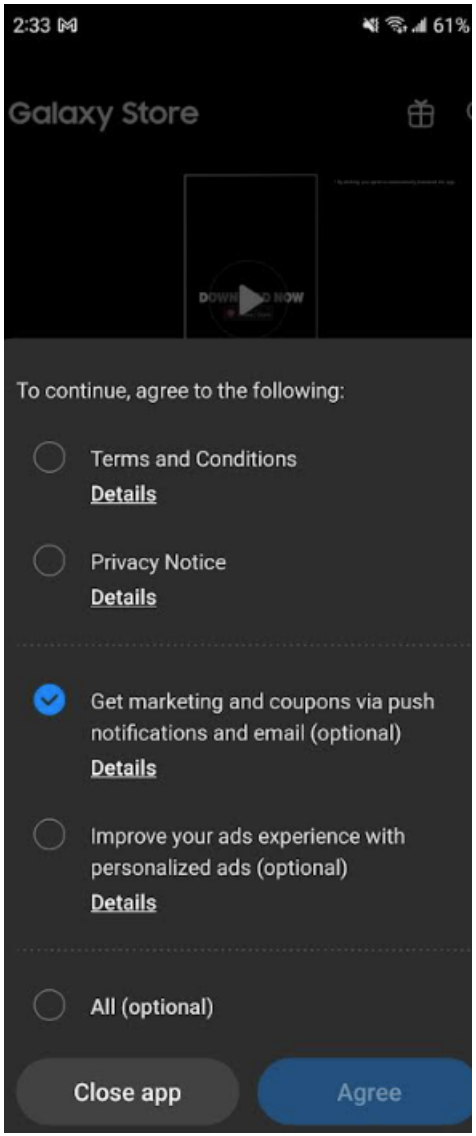
225	https://docs.tizen.org/application/web/guides/messaging/push/;  https://docs.samsungknox.com/admin/knox-manage/welcome.htm

226



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

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

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

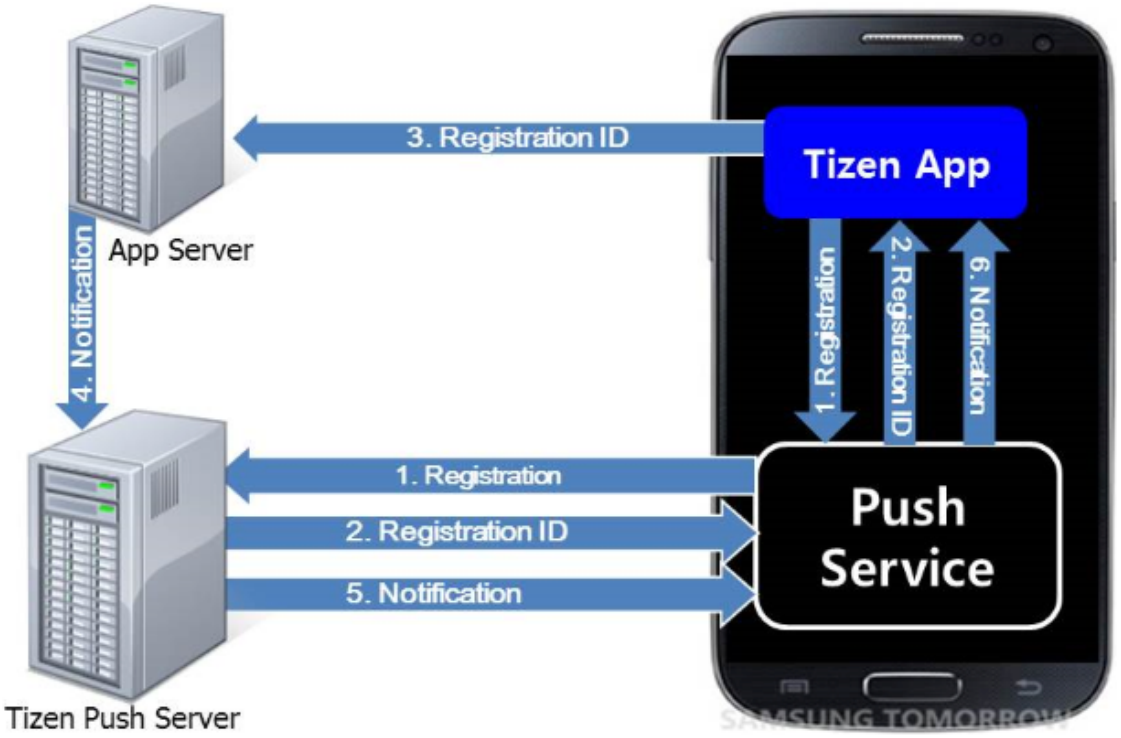
[1a] a transport services stack to maintain a respective secure message link through an Internet network between the message link server and a respective device link agent on each of a plurality of wireless end-user devices, each of the wireless end-user devices comprising multiple software components authorized to receive and process data from secure message link messages received via a device link agent on that device;	Samsung's push messaging servers comprise "a transport services stack to maintain a respective secure message link through an Internet network between the message link server and a respective device link agent on each of a plurality of wireless end-user devices, each of the wireless end-user devices comprising multiple software components authorized to receive and process data from secure message link messages received via a device link agent on that device." For example, Samsung's push messaging server maintains a secure message link between the Tizen Push Server and a device link agent (e.g., Tizen App and/or Push Service) on a plurality of devices which comprise multiple software components and receive/process data from messages sent over the secure message link. *See, e.g.,* **Push Notification** You can receive notifications from a push server. The push service is a client daemon that maintains a permanent connection between the device and the push server. Push enables you to push events from an application server to your application on a Tizen device. Connection with the push service is used to deliver push notifications to the application, and process the registration and deregistration requests. The Push API is optional for Tizen Mobile, Wearable, and TV profiles, which means that it may not be supported on all mobile, wearable, and TV devices. The Push API is supported on all Tizen emulators. Push notification helps your application server send data to your application on a device over a network, even if the application is not running. Using the push service can reduce battery consumption and data transfer. If a push message arrives when the application is running, the message is automatically delivered to the application. If the application is not running, the push service makes a sound or vibrates and adds a ticker or a badge notification to notify the user. By touching this notification, the user can check the message. If the application server sends a message with a `LAUNCH` option, the push service forcibly launches the application and hands over the message to the application.

229

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.

230

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);
```

231

- Since Tizen 3.0:

  Before registering, you must connect to the push service:

  1. Define event handlers:

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

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

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

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

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

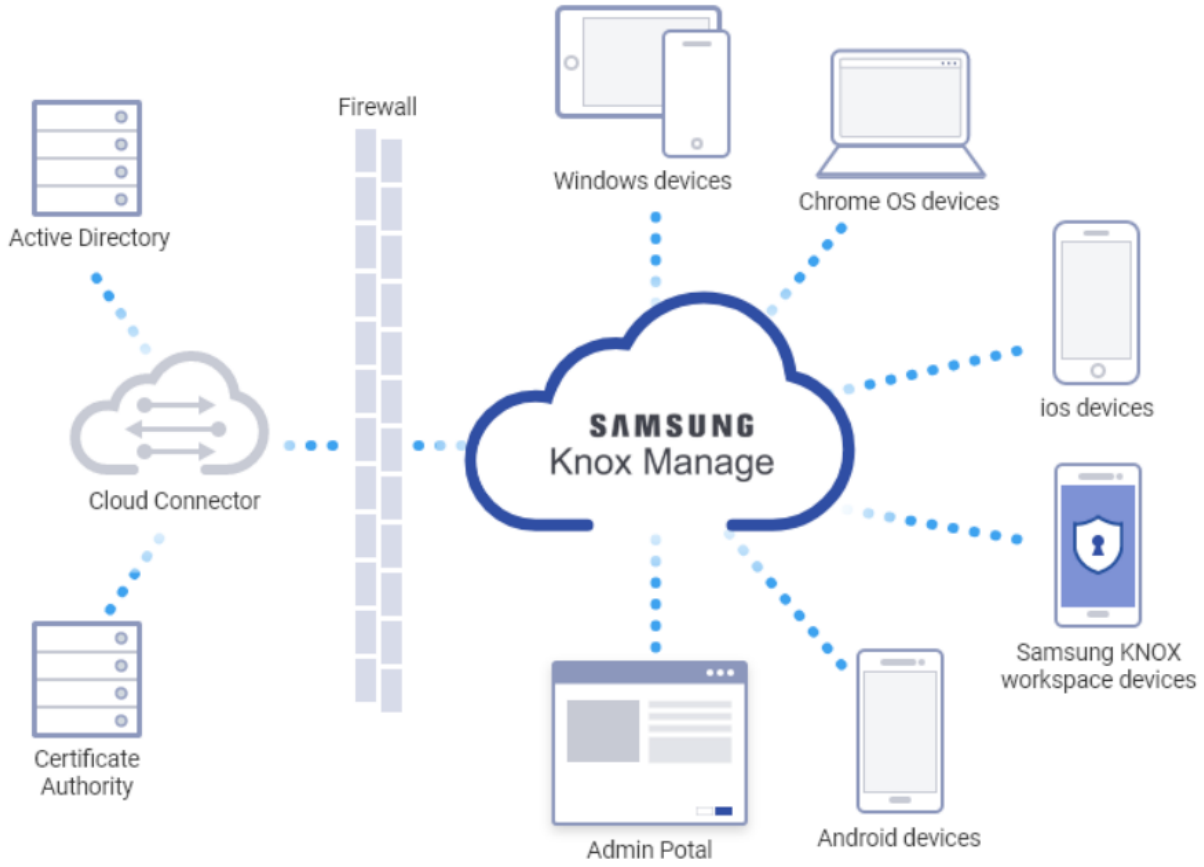
  2. Connect to the push service with the `connect()` method. The `register()` method is called in the `stateChangeCallback()` callback. This operation has to be done only once.

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

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

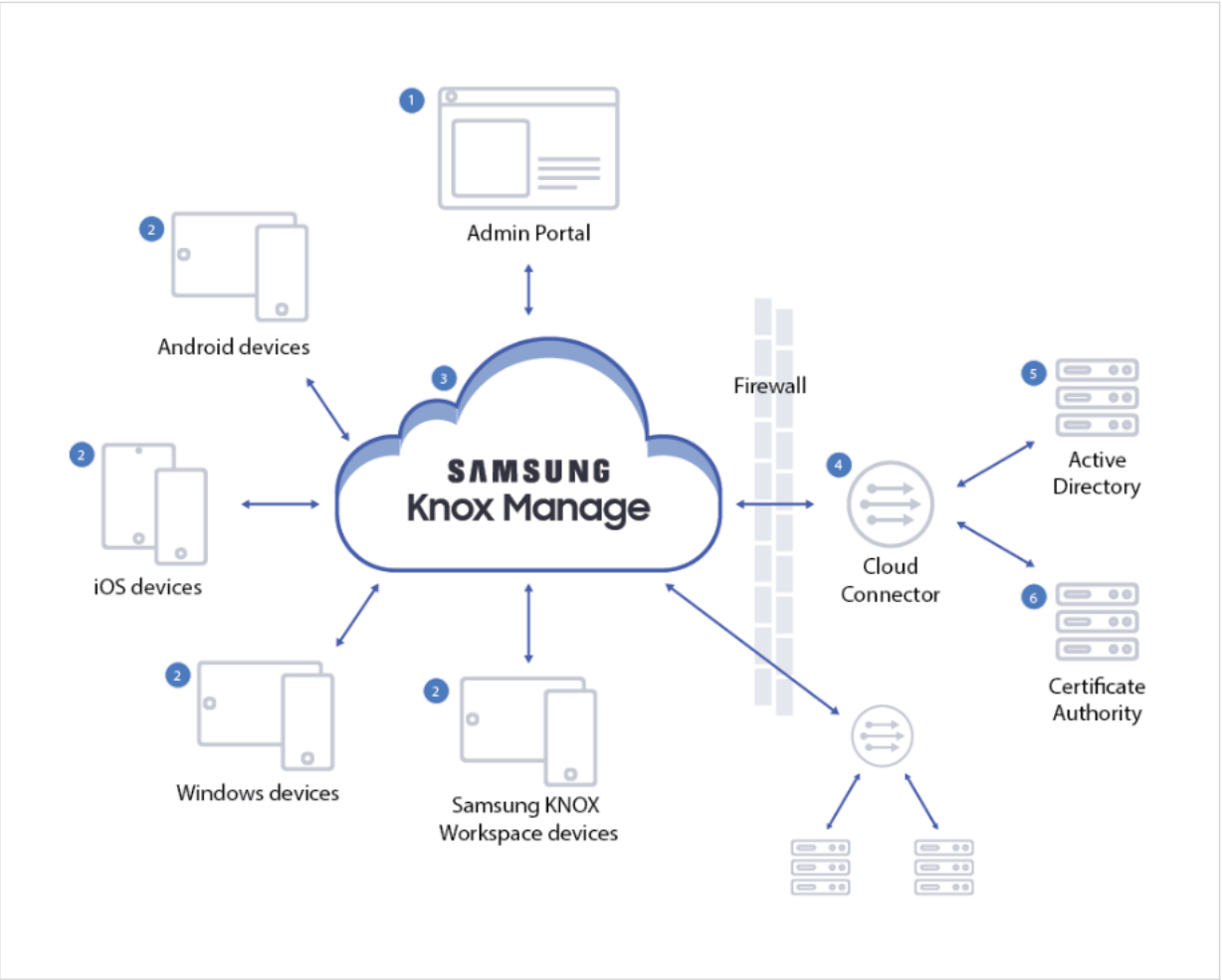
As another example, Samsung's devices operating in the Samsung Knox ecosystem, for example mobile phones and other devices enrolled in the Samsung Knox MDM platform and the Samsung Knox servers managing those phones and devices, maintains a secure message link between the server and a device link agent on a plurality of devices which comprise multiple software components and receive/process data from messages sent over the secure message link.

232



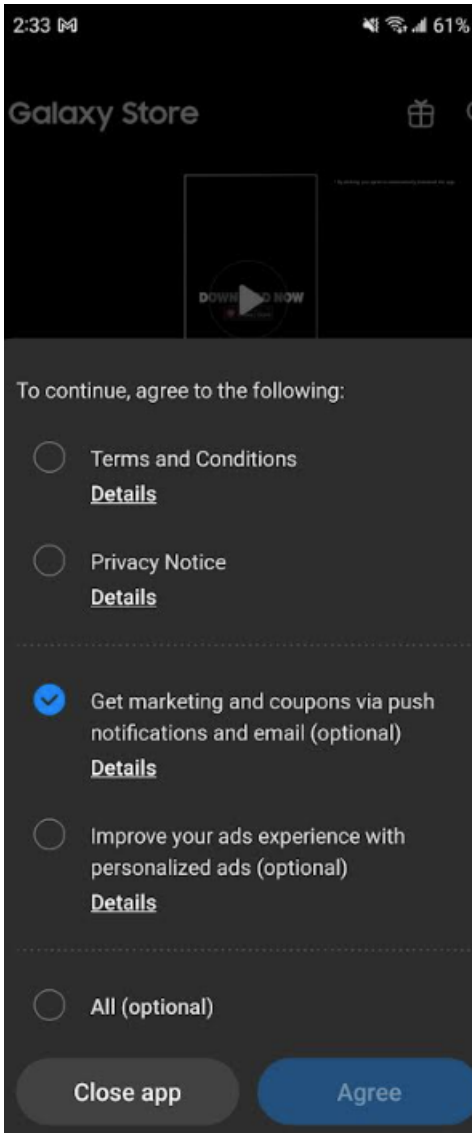
https://docs.samsungknox.com/admin/knox-manage/welcome.htm

233



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

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

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

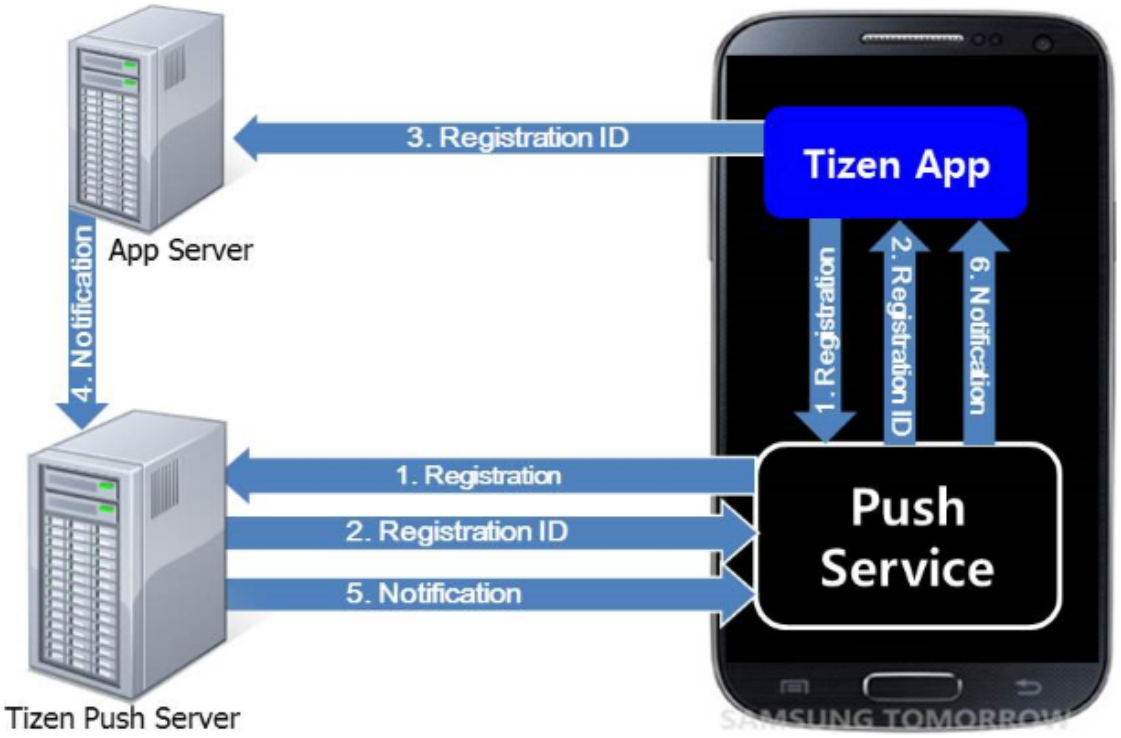
[1b] an interface to a network to receive network element messages from a plurality of network elements, the received network element messages comprising respective message content and requests for delivery of the respective message content to respective wireless end-user devices, the respective message content including data for, and an identification of, a respective one of the authorized software components; and	Samsung's push messaging servers comprise "an interface to a network to receive network element messages from a plurality of network elements, the received network element messages comprising respective message content and requests for delivery of the respective message content to respective wireless end-user devices, the respective message content including data for, and an identification of, a respective one of the authorized software components." For example, Samsung's push messaging servers receive messages from a plurality of network elements (e.g., App Servers) which comprise content and delivery requests to devices and software components. *See, e.g.,*

235

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.

237

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

Push Server

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

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

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

- Sending push notifications

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

- Decorating push notifications

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

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

Push

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

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

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

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

Figure: Push messaging service



238

239

Sending Push Notifications

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

The following example shows a sample push notification:

- URI: See the Push RQM (Request Manager) server URLs table.
- Method: HTTP POST
- Header:

```
appID: 1234567890987654
appSecret: dYo/o/m11gmWmjs7+5f+2zLNVOc=
```

- Body:

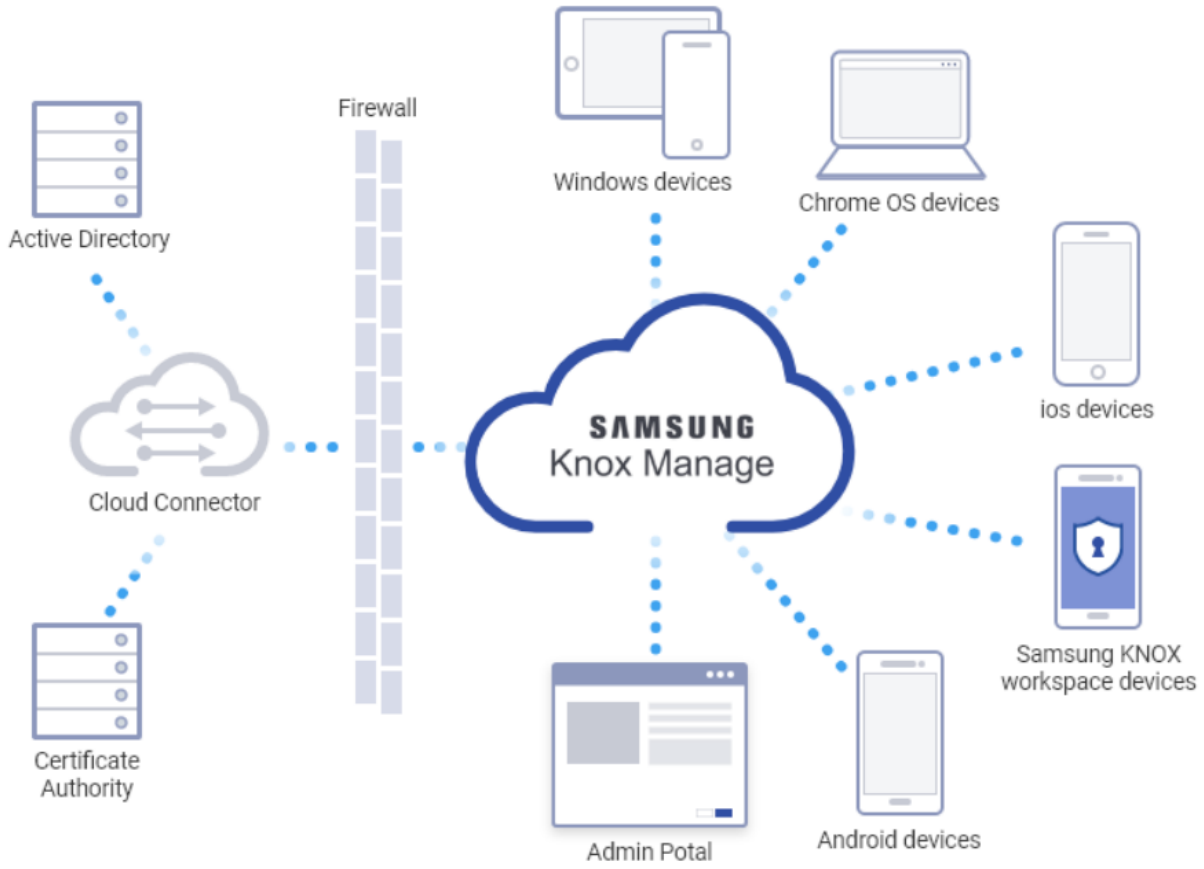
```
{
    "regID": "0501a53f4affdcbb98197f188345ff30c04b-5001",
    "requestID": "01231-22EAX-223442",
    "message": "badgeOption=INCREASE&badgeNumber=1&action=ALERT&alertMessage=Hi",
    "appData": "{id:asdf&passwd:1234}", /* Optional, if the message field is not empty */
}
```

To send a notification:

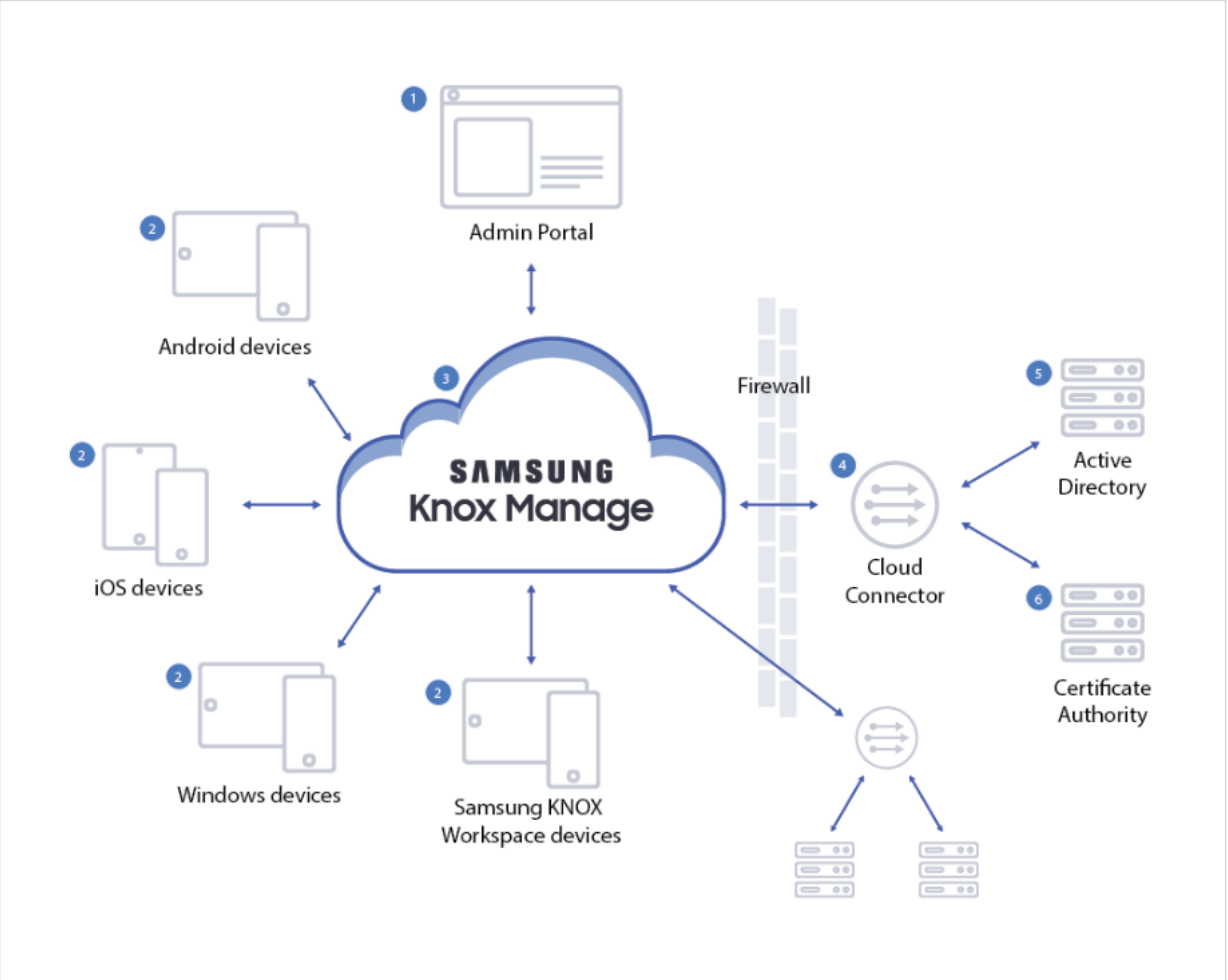
1. Prepare the `appID`, `appSecret`, `regID`, and `requestID`:
    - The `appID` and `appSecret` values are given in the email message that you received when requesting permission to use Tizen push servers.
    - The `regID` value is the one that the application server received from your application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

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

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

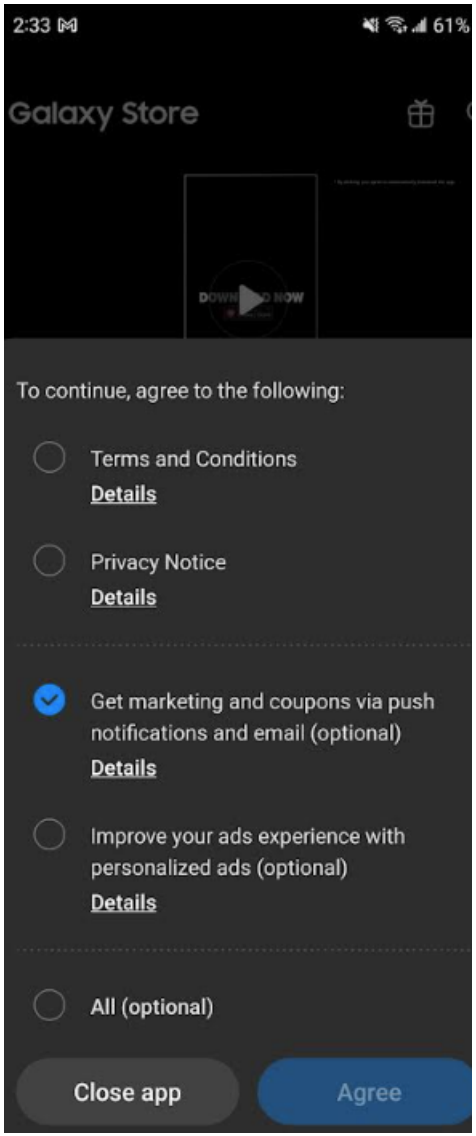
240	As another example, Samsung's receive messages from a plurality of network elements (e.g., App Servers) which comprise content and delivery requests to devices and software components in the Knox ecosystem.  https://docs.samsungknox.com/admin/knox-manage/welcome.htm

241



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

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

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

[1c] a message buffer system including a memory and logic,	The Accused Devices include "a message buffer system including a memory and logic." For example, Samsung's push messaging servers include memory to store messages and logic. *See e.g.*, 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/;

243

244

If the notification arrives when the application is not running, it can be handled in 3 ways:

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

  You need to set the action 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.

  When you create a project in the Tizen Studio, the `app_control()` function is created automatically. When the application is launched by another application or process (in this case, by the push service), all related information regarding this launch request is delivered through the `app_control` parameter. From this handle, retrieve the `op` operation using the `app_control_get_operation()` function. With `app_control` and `op`, retrieve the notification data using the `push_service_app_control_to_noti_data()` function.

  If the application is not launched by the push service, this function returns as `NULL`.

```
static void
app_control(app_control_h app_control, void *data)
{
    char *op = NULL;
    push_service_notification_h noti = NULL;
    int ret;

    if (app_control_get_operation(app_control, &op) < 0)
        return;

    /* Retrieve the noti from the bundle */
    ret = push_service_app_control_to_notification(app_control, op, &noti);

    if (noti) {
        /* Handle the noti */

        /* Free the noti */
        push_service_free_notification(noti);
    } else {
        /* Case when the application is not launched by the push service */
    }
    if (op)
        free(op);
}
```

Since Tizen 3.0, the push service provides launch types when the application is launched by the service. Use the following code to figure out why the application is launched, as the `app_control()` function is invoked in both cases of receiving notification and changing registration state.

```
#define EXTRA_DATA_FROM_REGISTRATION_CHANGE "registration_change"
#define EXTRA_DATA_FROM_NOTIFICATION "notification"

static void
app_control(app_control_h app_control, void *data)
{
    char *value = NULL;
    app_control_get_extra_data(app_control, APP_CONTROL_DATA_PUSH_LAUNCH_TYPE, &value);
    if (value) {
        if (!strcmp(value, EXTRA_DATA_FROM_NOTIFICATION))
            /* Add your code here when push messages arrive */
        else if (!strcmp(value, EXTRA_DATA_FROM_REGISTRATION_CHANGE))
            /* Add your code here when registration state is changed */
    }
}
```

245

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

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

  You can request for unread notifications from the push service. The request can be performed after connecting to the push server when the application is launched.

  ```
  if (push_conn) {
      int ret = push_service_request_unread_notification(push_conn);
      if (ret != PUSH_SERVICE_ERROR_NONE)
          dlog_print(DLOG_ERROR, LOG_TAG, "ERROR: push_service_request_unread_notification()
  failed.");
  }
  ```

  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 through the app control callback function.

- Discard the notification.

  You need to set the action 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 delivers the notification only when the application is up and running. Otherwise, the push service does not store the notification and discards it.

247

- Request unread notifications.

  If the user does not launch the application from the quick panel, the application requests the unread notifications after start-up using the asynchronous `push_service_request_unread_notification()` function.

  The following example shows a synchronous request using the `push_service_get_unread_notification()` function:

```
push_service_notification_h noti;
int ret;
do {
    ret = push_service_get_unread_notification(push_conn, &noti);

    /* Process the unread message noti */

    push_server_free_notification(&noti);
} while (1);
```

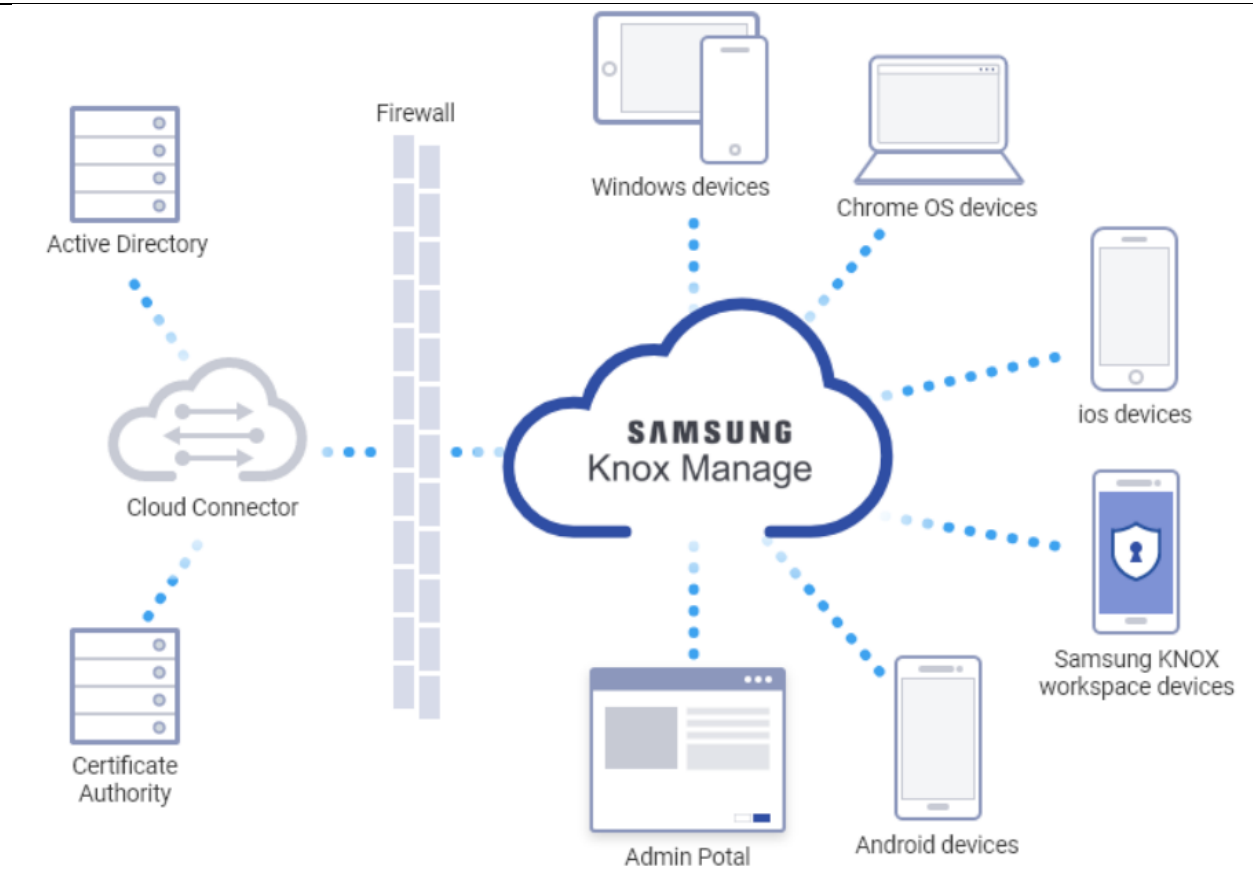
  Call this function repeatedly until no notification is returned. If there are multiple unread notifications, the notifications are retrieved in their arrival order.

  The `push_service_get_unread_notification()` function blocks the code while it receives a notification from the service. Unless you need this kind of synchronous behavior, use the asynchronous function.

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

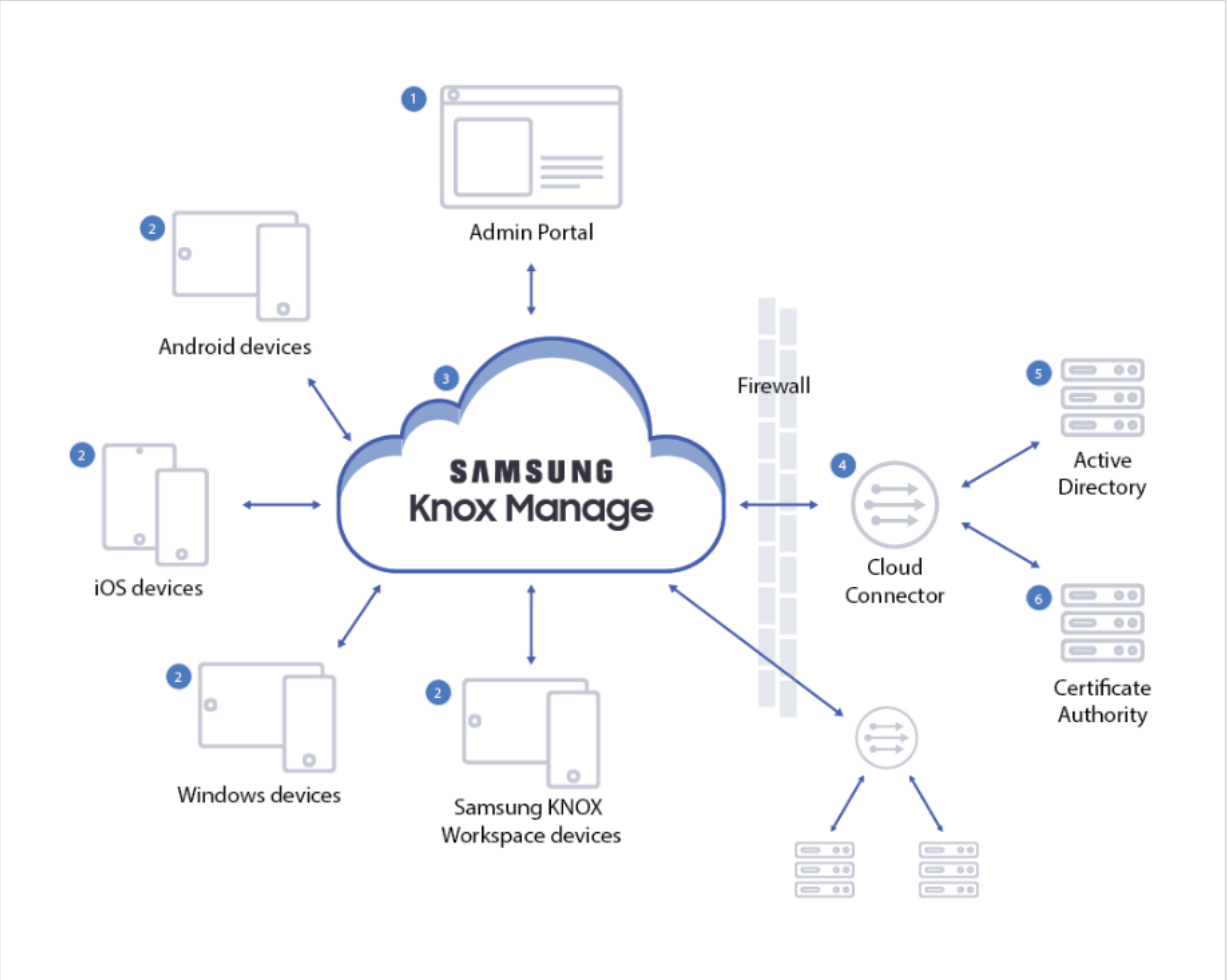
As another example, Samsung's Knox servers include memory to store messages and logic.

248



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

249



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

250

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.

6. If displayed, select the **Push update without requesting end user consent** to push the updates directly to the configured state devices without consent from the device end user. Leave this option unselected affords the device user the ability to approve the profile update before its pushed to their device.
7. Select **YES** to proceed with the device push update and profile overwrite. Select **Back** to move back to the previous **Assign devices with profiles** screen.

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

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

251

	# Set the profile update schedule You can set the schedule to update the latest policy applied to user devices on a regular basis. You can define multiple schedules. Click + to add a schedule. You can add or edit up to 20 schedules when you save the settings. To configure a policy update schedule: 1. Navigate to **Setting** > **Configuration** > **Profile Update Scheduler**. 2. Click next to Profile Update Schedule to enable the scheduler feature. 3. Select a target type from the following. • **Global Setting**—All profiles are updated according to the schedule. • **Set by Group / Organization**—You can configure multiple schedules and select groups or organizations for each schedule setting. 4. Select days and set the start time and time zone. • Select groups or organizations for each schedule setting if you selected the group/organization target type. • The policy update time may vary depending on the time and time zone of the user device. 5. Click **Save & Apply**. https://docs.samsungknox.com/admin/knox-manage/set-the-profile-update-schedule.htm.
[1d] the memory to buffer content from the received network element messages for which delivery is requested to a given one of the wireless end-user devices,	The Accused Devices include "the memory to buffer content from the received network element messages for which delivery is requested to a given one of the wireless end-user devices." For example, Samsung's push messaging servers include memory to store content from received messages for delivery to a device. *See e.g.*,

253

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/;

254

If the notification arrives when the application is not running, it can be handled in 3 ways:

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

  You need to set the action 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.

  When you create a project in the Tizen Studio, the `app_control()` function is created automatically. When the application is launched by another application or process (in this case, by the push service), all related information regarding this launch request is delivered through the `app_control` parameter. From this handle, retrieve the `op` operation using the `app_control_get_operation()` function. With `app_control` and `op`, retrieve the notification data using the `push_service_app_control_to_noti_data()` function.

  If the application is not launched by the push service, this function returns as `NULL`.

```
static void
app_control(app_control_h app_control, void *data)
{
    char *op = NULL;
    push_service_notification_h noti = NULL;
    int ret;

    if (app_control_get_operation(app_control, &op) < 0)
        return;

    /* Retrieve the noti from the bundle */
    ret = push_service_app_control_to_notification(app_control, op, &noti);

    if (noti) {
        /* Handle the noti */

        /* Free the noti */
        push_service_free_notification(noti);
    } else {
        /* Case when the application is not launched by the push service */
    }
    if (op)
        free(op);
}
```

Since Tizen 3.0, the push service provides launch types when the application is launched by the service. Use the following code to figure out why the application is launched, as the `app_control()` function is invoked in both cases of receiving notification and changing registration state.

```
#define EXTRA_DATA_FROM_REGISTRATION_CHANGE "registration_change"
#define EXTRA_DATA_FROM_NOTIFICATION "notification"

static void
app_control(app_control_h app_control, void *data)
{
    char *value = NULL;
    app_control_get_extra_data(app_control, APP_CONTROL_DATA_PUSH_LAUNCH_TYPE, &value);
    if (value) {
        if (!strcmp(value, EXTRA_DATA_FROM_NOTIFICATION))
            /* Add your code here when push messages arrive */
        else if (!strcmp(value, EXTRA_DATA_FROM_REGISTRATION_CHANGE))
            /* Add your code here when registration state is changed */
    }
}
```

255

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

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

  You can request for unread notifications from the push service. The request can be performed after connecting to the push server when the application is launched.

  ```
  if (push_conn) {
      int ret = push_service_request_unread_notification(push_conn);
      if (ret != PUSH_SERVICE_ERROR_NONE)
          dlog_print(DLOG_ERROR, LOG_TAG, "ERROR: push_service_request_unread_notification()
  failed.");
  }
  ```

  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 through the app control callback function.

- Discard the notification.

  You need to set the action 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 delivers the notification only when the application is up and running. Otherwise, the push service does not store the notification and discards it.

257

- Request unread notifications.

  If the user does not launch the application from the quick panel, the application requests the unread notifications after start-up using the asynchronous `push_service_request_unread_notification()` function.

  The following example shows a synchronous request using the `push_service_get_unread_notification()` function:

```
push_service_notification_h noti;
int ret;
do {
    ret = push_service_get_unread_notification(push_conn, &noti);

    /* Process the unread message noti */

    push_server_free_notification(&noti);
} while (1);
```

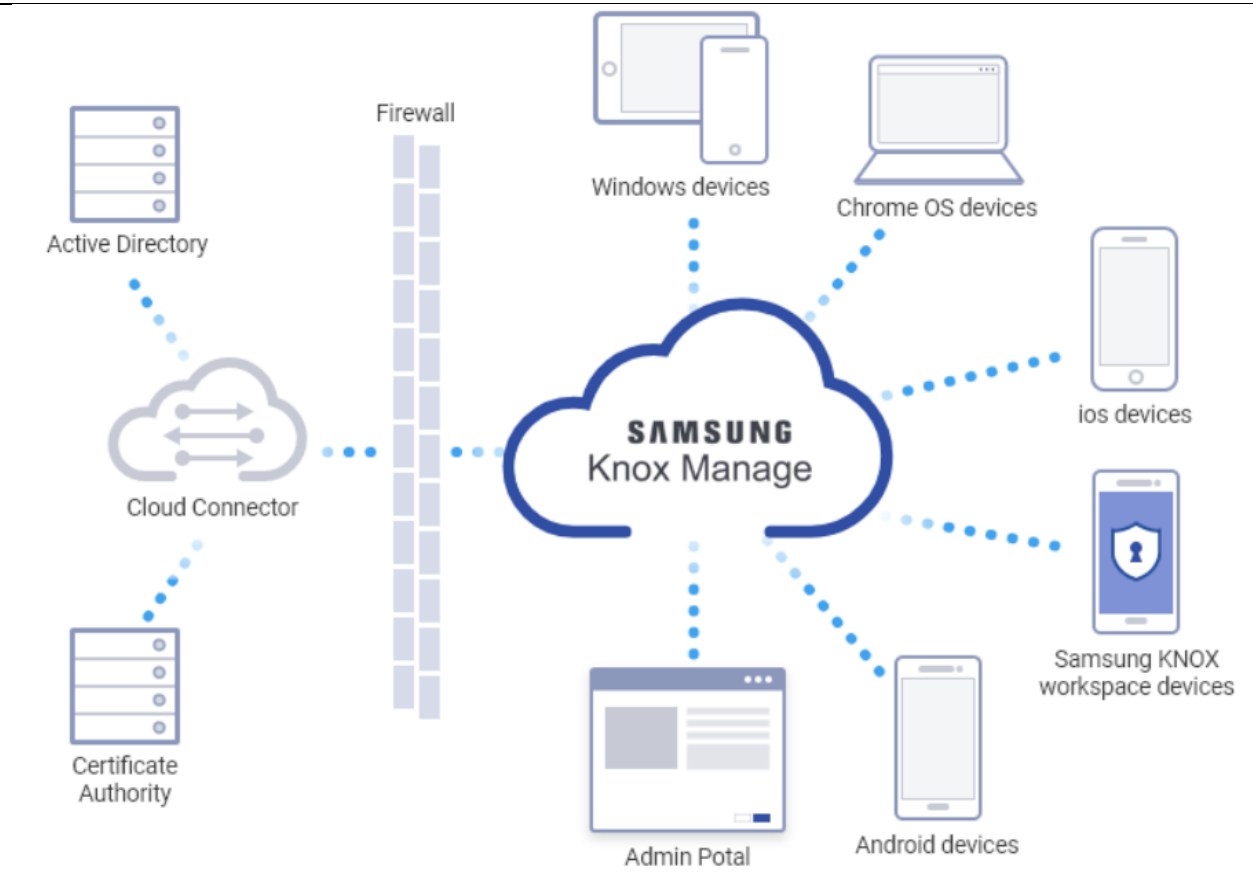
  Call this function repeatedly until no notification is returned. If there are multiple unread notifications, the notifications are retrieved in their arrival order.

  The `push_service_get_unread_notification()` function blocks the code while it receives a notification from the service. Unless you need this kind of synchronous behavior, use the asynchronous function.

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

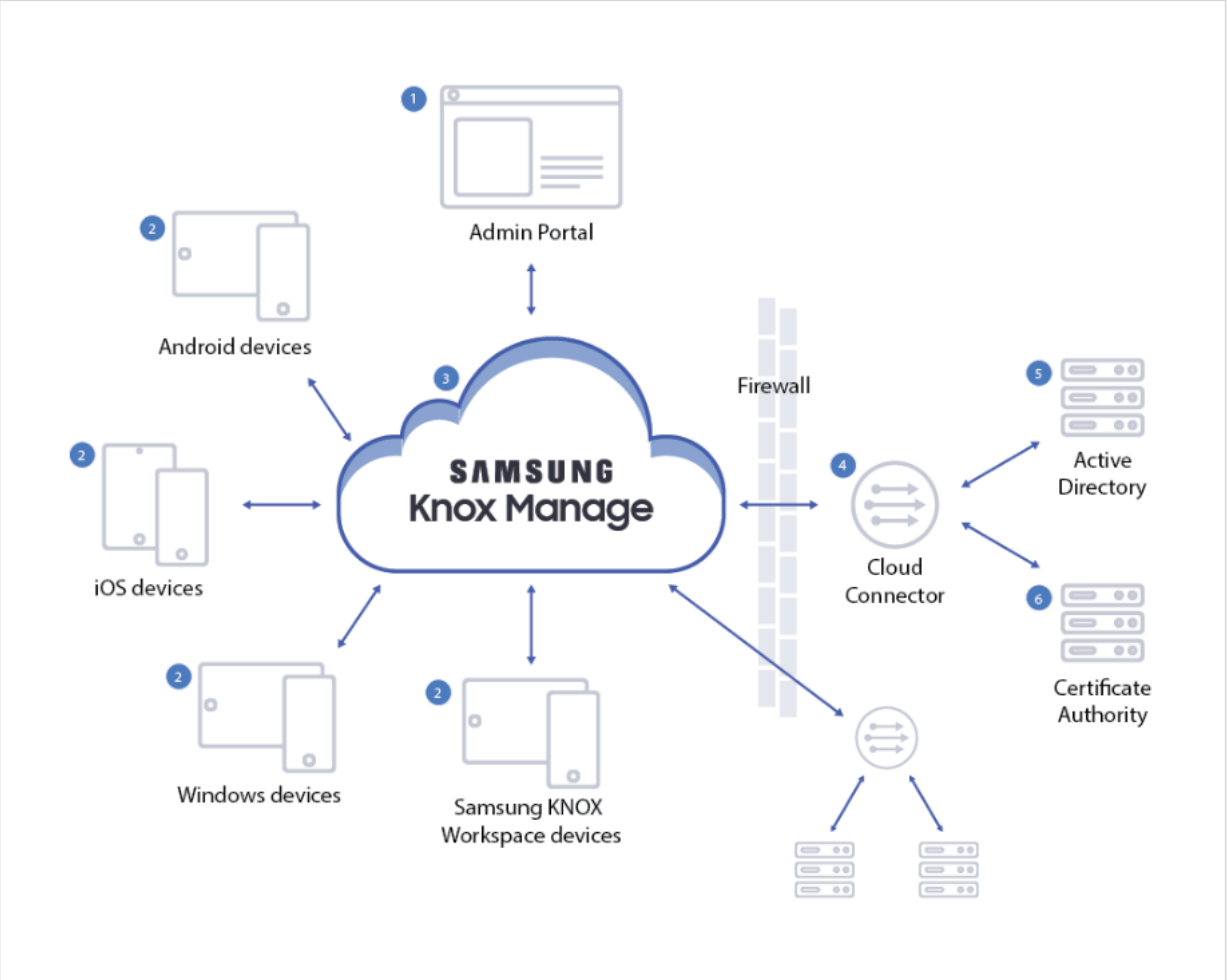
As another example, Samsung's Knox servers include memory to store content from received messages for delivery to a device.

258



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

259



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

260

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.

6. If displayed, select the **Push update without requesting end user consent** to push the updates directly to the configured state devices without consent from the device end user. Leave this option unselected affords the device user the ability to approve the profile update before its pushed to their device.
7. Select **YES** to proceed with the device push update and profile overwrite. Select **Back** to move back to the previous **Assign devices with profiles** screen.

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

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

261

	# Set the profile update schedule You can set the schedule to update the latest policy applied to user devices on a regular basis. You can define multiple schedules. Click + to add a schedule. You can add or edit up to 20 schedules when you save the settings. To configure a policy update schedule: 1. Navigate to **Setting** > **Configuration** > **Profile Update Scheduler**. 2. Click next to Profile Update Schedule to enable the scheduler feature. 3. Select a target type from the following. • **Global Setting**—All profiles are updated according to the schedule. • **Set by Group / Organization**—You can configure multiple schedules and select groups or organizations for each schedule setting. 4. Select days and set the start time and time zone. • Select groups or organizations for each schedule setting if you selected the group/organization target type. • The policy update time may vary depending on the time and time zone of the user device. 5. Click **Save & Apply**. https://docs.samsungknox.com/admin/knox-manage/set-the-profile-update-schedule.htm.
[1e] the logic to determine when one of a plurality of message delivery triggers for the given one of the wireless end-user devices has occurred, wherein for at least some of the received network element messages, the receipt of such a message by the message buffer system is not a	The Accused Devices include "the logic to determine when one of a plurality of message delivery triggers for the given one of the wireless end-user devices has occurred, wherein for at least some of the received network element messages, the receipt of such a message by the message buffer system is not a message delivery trigger, and for at least one of the message delivery triggers, the trigger is an occurrence of an asynchronous event with time-critical messaging needs."

message delivery trigger, and for at least one of the message delivery triggers, the trigger is an occurrence of an asynchronous event with time-critical messaging needs, and	For example, Samsung's push messaging servers include logic to determine whether a delivery trigger that is not based on the receipt of the message has occurred. *See e.g.*, 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/;

263

264

If the notification arrives when the application is not running, it can be handled in 3 ways:

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

  You need to set the action 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.

  When you create a project in the Tizen Studio, the `app_control()` function is created automatically. When the application is launched by another application or process (in this case, by the push service), all related information regarding this launch request is delivered through the `app_control` parameter. From this handle, retrieve the `op` operation using the `app_control_get_operation()` function. With `app_control` and `op`, retrieve the notification data using the `push_service_app_control_to_noti_data()` function.

  If the application is not launched by the push service, this function returns as `NULL`.

```
static void
app_control(app_control_h app_control, void *data)
{
    char *op = NULL;
    push_service_notification_h noti = NULL;
    int ret;

    if (app_control_get_operation(app_control, &op) < 0)
        return;

    /* Retrieve the noti from the bundle */
    ret = push_service_app_control_to_notification(app_control, op, &noti);

    if (noti) {
        /* Handle the noti */

        /* Free the noti */
        push_service_free_notification(noti);
    } else {
        /* Case when the application is not launched by the push service */
    }
    if (op)
        free(op);
}
```

Since Tizen 3.0, the push service provides launch types when the application is launched by the service. Use the following code to figure out why the application is launched, as the `app_control()` function is invoked in both cases of receiving notification and changing registration state.

```
#define EXTRA_DATA_FROM_REGISTRATION_CHANGE "registration_change"
#define EXTRA_DATA_FROM_NOTIFICATION "notification"

static void
app_control(app_control_h app_control, void *data)
{
    char *value = NULL;
    app_control_get_extra_data(app_control, APP_CONTROL_DATA_PUSH_LAUNCH_TYPE, &value);
    if (value) {
        if (!strcmp(value, EXTRA_DATA_FROM_NOTIFICATION))
            /* Add your code here when push messages arrive */
        else if (!strcmp(value, EXTRA_DATA_FROM_REGISTRATION_CHANGE))
            /* Add your code here when registration state is changed */
    }
}
```

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

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

  You can request for unread notifications from the push service. The request can be performed after connecting to the push server when the application is launched.

  ```
  if (push_conn) {
      int ret = push_service_request_unread_notification(push_conn);
      if (ret != PUSH_SERVICE_ERROR_NONE)
          dlog_print(DLOG_ERROR, LOG_TAG, "ERROR: push_service_request_unread_notification()
  failed.");
  }
  ```

  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 through the app control callback function.

- Discard the notification.

  You need to set the action 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 delivers the notification only when the application is up and running. Otherwise, the push service does not store the notification and discards it.

267

- Request unread notifications.

  If the user does not launch the application from the quick panel, the application requests the unread notifications after start-up using the asynchronous `push_service_request_unread_notification()` function.

  The following example shows a synchronous request using the `push_service_get_unread_notification()` function:

```
push_service_notification_h noti;
int ret;
do {
    ret = push_service_get_unread_notification(push_conn, &noti);

    /* Process the unread message noti */

    push_server_free_notification(&noti);
} while (1);
```

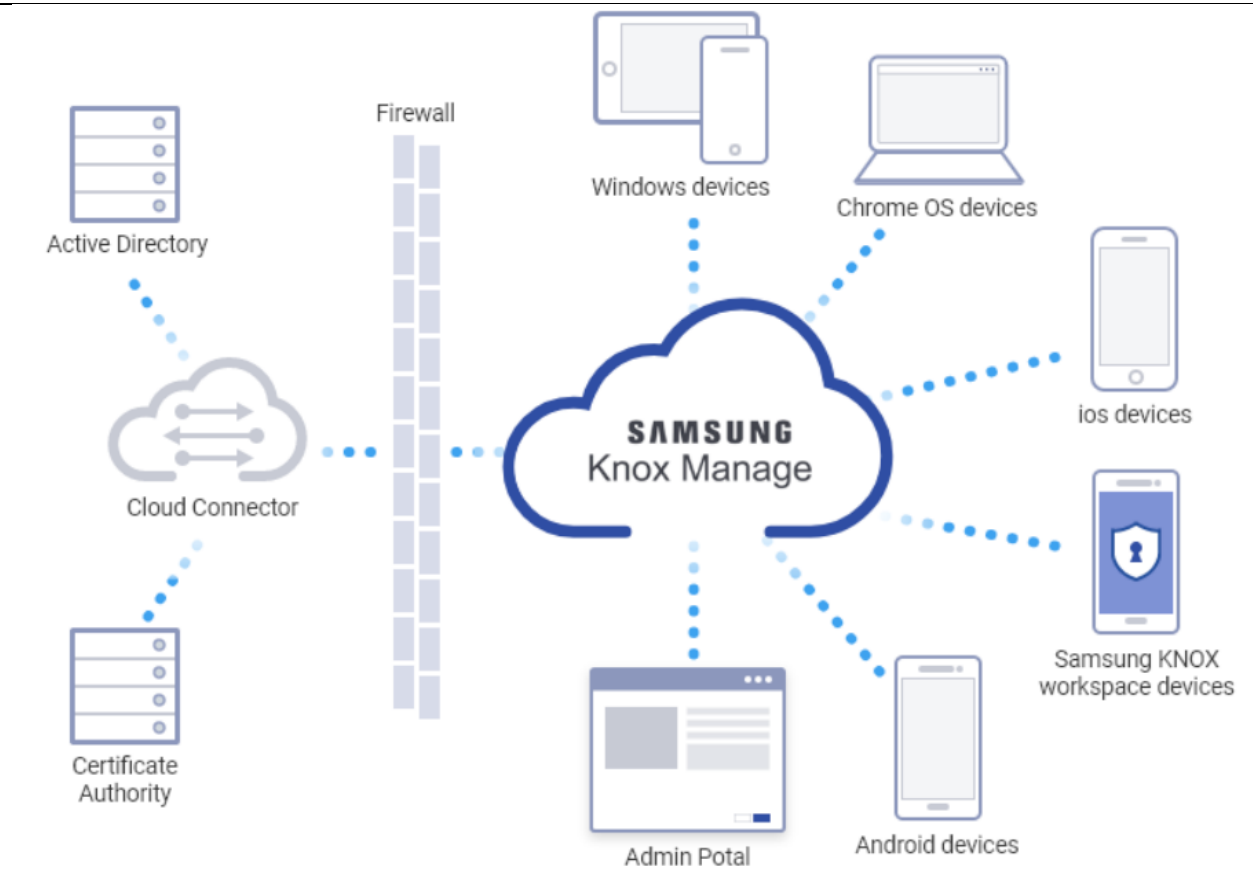
  Call this function repeatedly until no notification is returned. If there are multiple unread notifications, the notifications are retrieved in their arrival order.

  The `push_service_get_unread_notification()` function blocks the code while it receives a notification from the service. Unless you need this kind of synchronous behavior, use the asynchronous function.

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

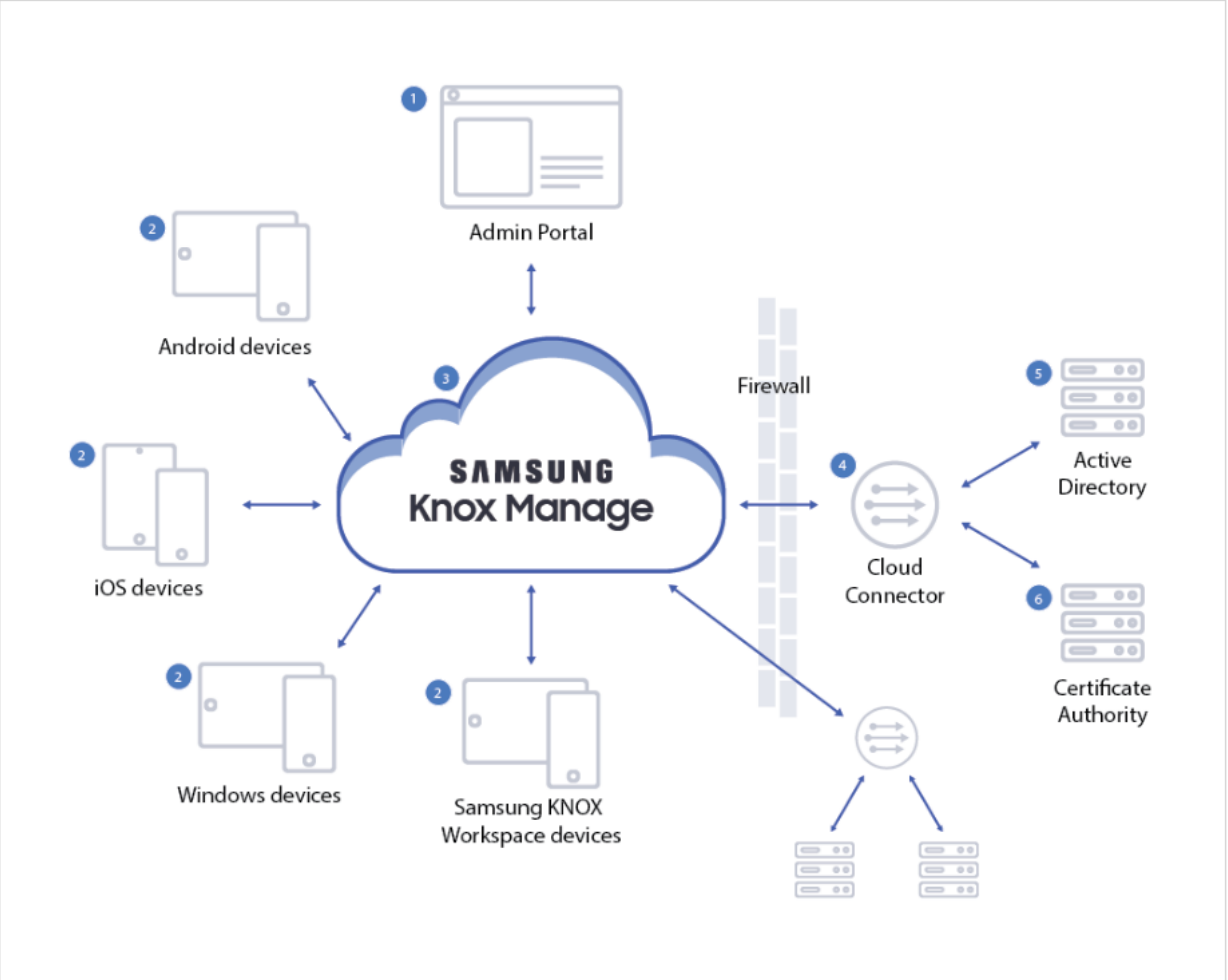
As another example, Samsung's Knox servers include logic to determine whether a delivery trigger that is not based on the receipt of the message has occurred.

268



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

269



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

270

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.

6. If displayed, select the **Push update without requesting end user consent** to push the updates directly to the configured state devices without consent from the device end user. Leave this option unselected affords the device user the ability to approve the profile update before its pushed to their device.
7. Select **YES** to proceed with the device push update and profile overwrite. Select **Back** to move back to the previous **Assign devices with profiles** screen.

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

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

271

	# Set the profile update schedule You can set the schedule to update the latest policy applied to user devices on a regular basis. You can define multiple schedules. Click + to add a schedule. You can add or edit up to 20 schedules when you save the settings. To configure a policy update schedule: 1. Navigate to **Setting** > **Configuration** > **Profile Update Scheduler**. 2. Click next to Profile Update Schedule to enable the scheduler feature. 3. Select a target type from the following. • **Global Setting**—All profiles are updated according to the schedule. • **Set by Group / Organization**—You can configure multiple schedules and select groups or organizations for each schedule setting. 4. Select days and set the start time and time zone. • Select groups or organizations for each schedule setting if you selected the group/organization target type. • The policy update time may vary depending on the time and time zone of the user device. 5. Click **Save & Apply**. https://docs.samsungknox.com/admin/knox-manage/set-the-profile-update-schedule.htm.
[1f] upon determining that one of the message delivery triggers has occurred, the logic further to supply one or more messages comprising the buffered content to the transport services stack for delivery on the secure message link maintained between the transport	The Accused Devices include "upon determining that one of the message delivery triggers has occurred, the logic further to supply one or more messages comprising the buffered content to the transport services stack for delivery on the secure message link maintained between the transport services stack and a device link agent on the given one of the wireless end-user devices."

273	services stack and a device link agent on the given one of the wireless end-user devices.	For example, Samsung's push messaging servers comprise logic which supplies buffered content for delivery. *See e.g.*, 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/;

274

If the notification arrives when the application is not running, it can be handled in 3 ways:

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

  You need to set the action 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.

  When you create a project in the Tizen Studio, the `app_control()` function is created automatically. When the application is launched by another application or process (in this case, by the push service), all related information regarding this launch request is delivered through the `app_control` parameter. From this handle, retrieve the `op` operation using the `app_control_get_operation()` function. With `app_control` and `op`, retrieve the notification data using the `push_service_app_control_to_noti_data()` function.

  If the application is not launched by the push service, this function returns as `NULL`.

```
static void
app_control(app_control_h app_control, void *data)
{
    char *op = NULL;
    push_service_notification_h noti = NULL;
    int ret;

    if (app_control_get_operation(app_control, &op) < 0)
        return;

    /* Retrieve the noti from the bundle */
    ret = push_service_app_control_to_notification(app_control, op, &noti);

    if (noti) {
        /* Handle the noti */

        /* Free the noti */
        push_service_free_notification(noti);
    } else {
        /* Case when the application is not launched by the push service */
    }
    if (op)
        free(op);
}
```

Since Tizen 3.0, the push service provides launch types when the application is launched by the service. Use the following code to figure out why the application is launched, as the `app_control()` function is invoked in both cases of receiving notification and changing registration state.

```
#define EXTRA_DATA_FROM_REGISTRATION_CHANGE "registration_change"
#define EXTRA_DATA_FROM_NOTIFICATION "notification"

static void
app_control(app_control_h app_control, void *data)
{
    char *value = NULL;
    app_control_get_extra_data(app_control, APP_CONTROL_DATA_PUSH_LAUNCH_TYPE, &value);
    if (value) {
        if (!strcmp(value, EXTRA_DATA_FROM_NOTIFICATION))
            /* Add your code here when push messages arrive */
        else if (!strcmp(value, EXTRA_DATA_FROM_REGISTRATION_CHANGE))
            /* Add your code here when registration state is changed */
    }
}
```

275

276

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

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

  You can request for unread notifications from the push service. The request can be performed after connecting to the push server when the application is launched.

  ```
  if (push_conn) {
      int ret = push_service_request_unread_notification(push_conn);
      if (ret != PUSH_SERVICE_ERROR_NONE)
          dlog_print(DLOG_ERROR, LOG_TAG, "ERROR: push_service_request_unread_notification()
  failed.");
  }
  ```

  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 through the app control callback function.

- Discard the notification.

  You need to set the action 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 delivers the notification only when the application is up and running. Otherwise, the push service does not store the notification and discards it.

277

- Request unread notifications.

  If the user does not launch the application from the quick panel, the application requests the unread notifications after start-up using the asynchronous `push_service_request_unread_notification()` function.

  The following example shows a synchronous request using the `push_service_get_unread_notification()` function:

```
push_service_notification_h noti;
int ret;
do {
    ret = push_service_get_unread_notification(push_conn, &noti);

    /* Process the unread message noti */

    push_server_free_notification(&noti);
} while (1);
```

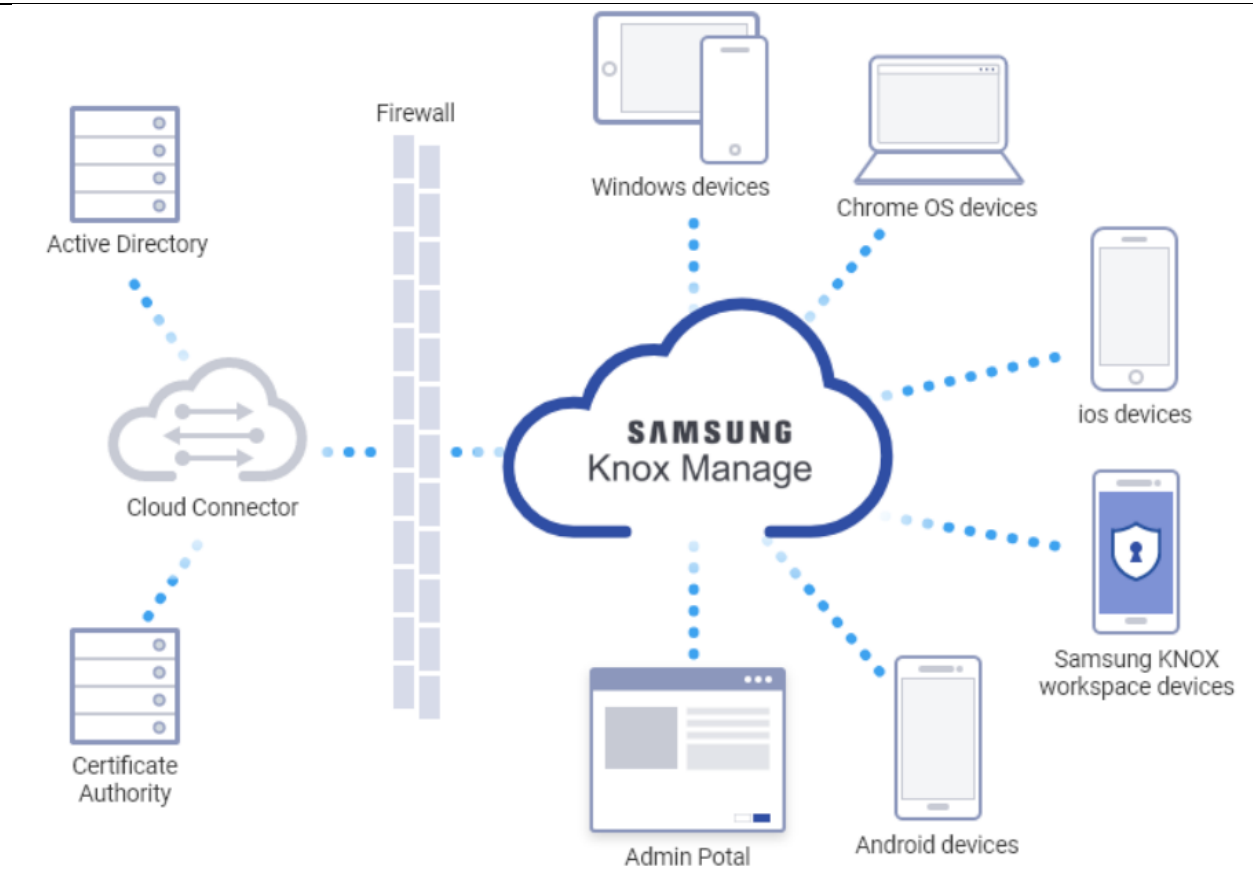
  Call this function repeatedly until no notification is returned. If there are multiple unread notifications, the notifications are retrieved in their arrival order.

  The `push_service_get_unread_notification()` function blocks the code while it receives a notification from the service. Unless you need this kind of synchronous behavior, use the asynchronous function.

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

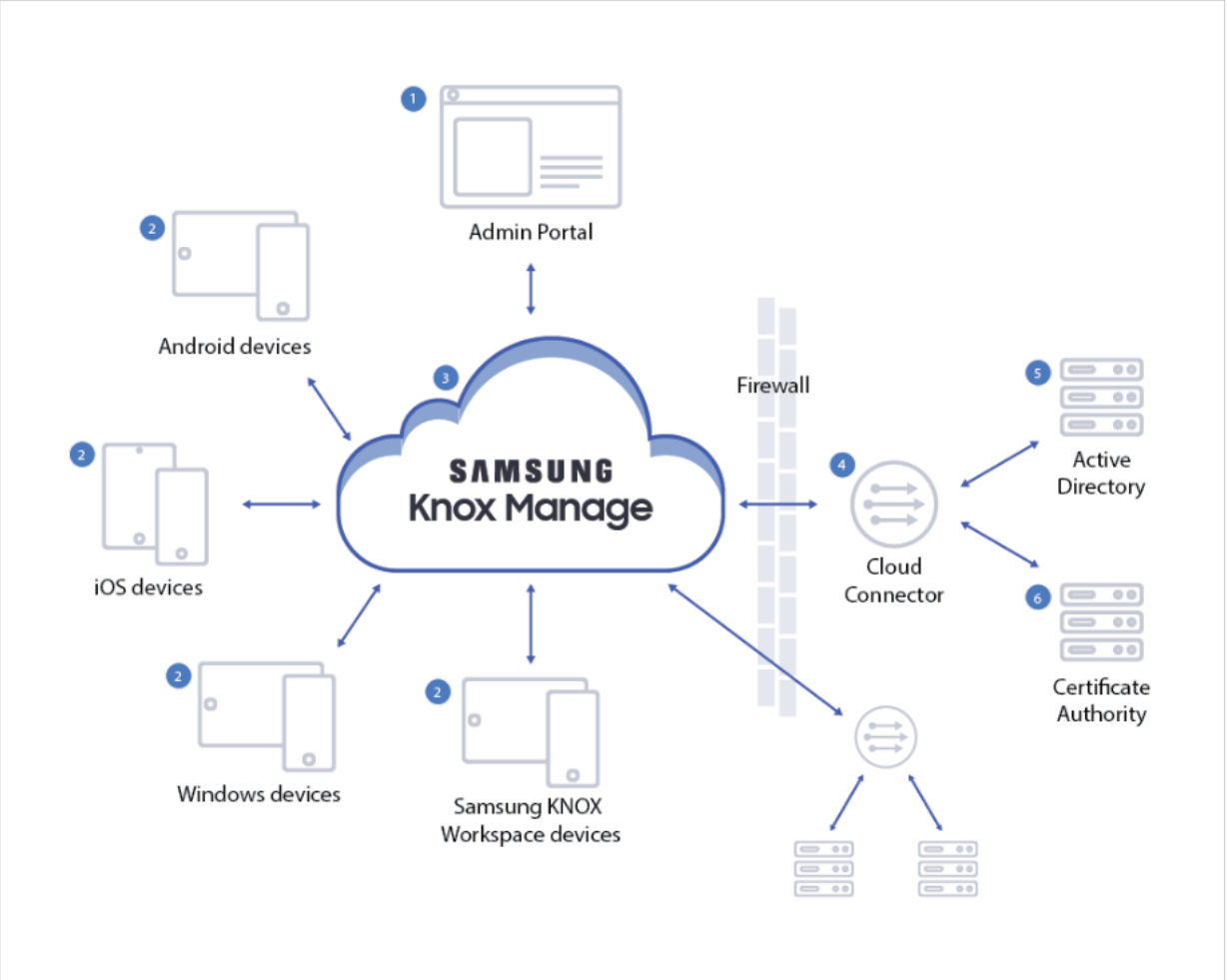
As another example, Samsung's Knox servers comprise logic which supplies buffered content for delivery.

278



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

279



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

280

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.

6. If displayed, select the **Push update without requesting end user consent** to push the updates directly to the configured state devices without consent from the device end user. Leave this option unselected affords the device user the ability to approve the profile update before its pushed to their device.
7. Select **YES** to proceed with the device push update and profile overwrite. Select **Back** to move back to the previous **Assign devices with profiles** screen.

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

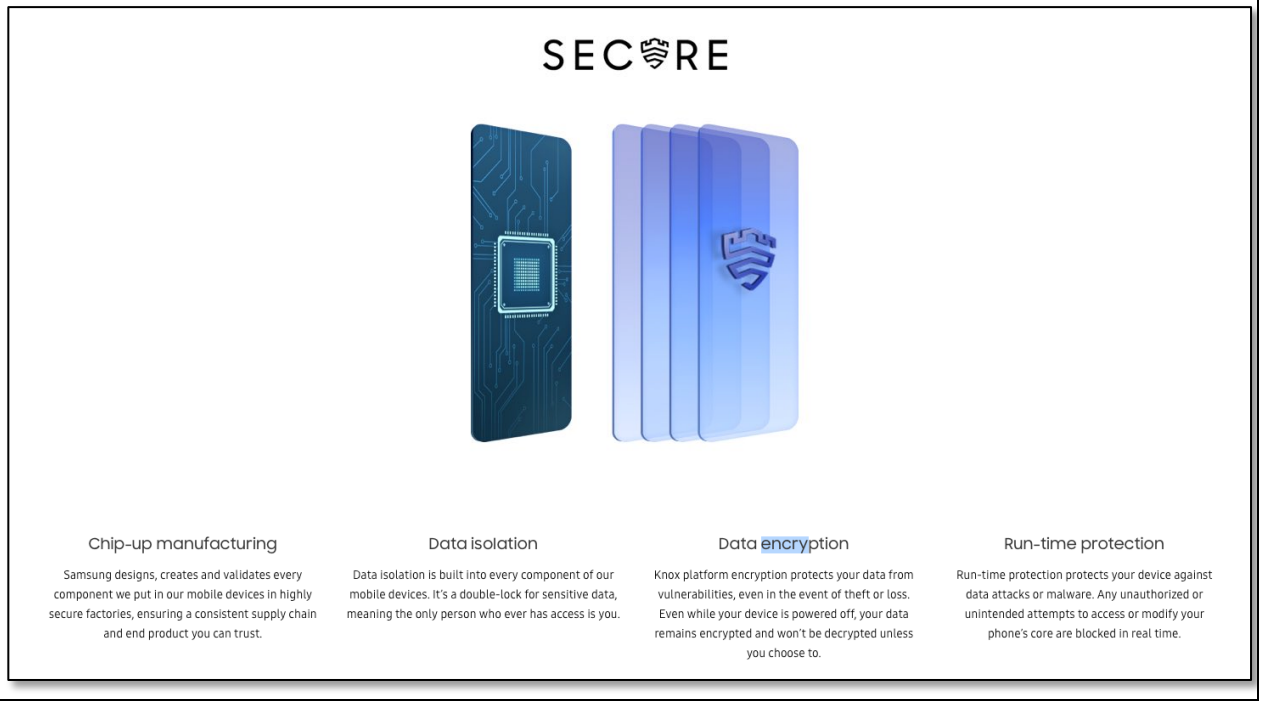
	Components of Knox Manage 1. **Knox Manage console** — A web console that allows IT admins to configure, monitor, and manage devices, deploy updates, as well as manage certificates and licenses. 2. **Knox Manage MDM Client** — An app that is installed on devices to automate installation and enrollment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

281

282	# Set the profile update schedule You can set the schedule to update the latest policy applied to user devices on a regular basis. You can define multiple schedules. Click + to add a schedule. You can add or edit up to 20 schedules when you save the settings. To configure a policy update schedule: 1. Navigate to **Setting** > **Configuration** > **Profile Update Scheduler**. 2. Click next to Profile Update Schedule to enable the scheduler feature. 3. Select a target type from the following. • **Global Setting**—All profiles are updated according to the schedule. • **Set by Group / Organization**—You can configure multiple schedules and select groups or organizations for each schedule setting. 4. Select days and set the start time and time zone. • Select groups or organizations for each schedule setting if you selected the group/organization target type. • The policy update time may vary depending on the time and time zone of the user device. 5. Click **Save & Apply**. https://docs.samsungknox.com/admin/knox-manage/set-the-profile-update-schedule.htm.

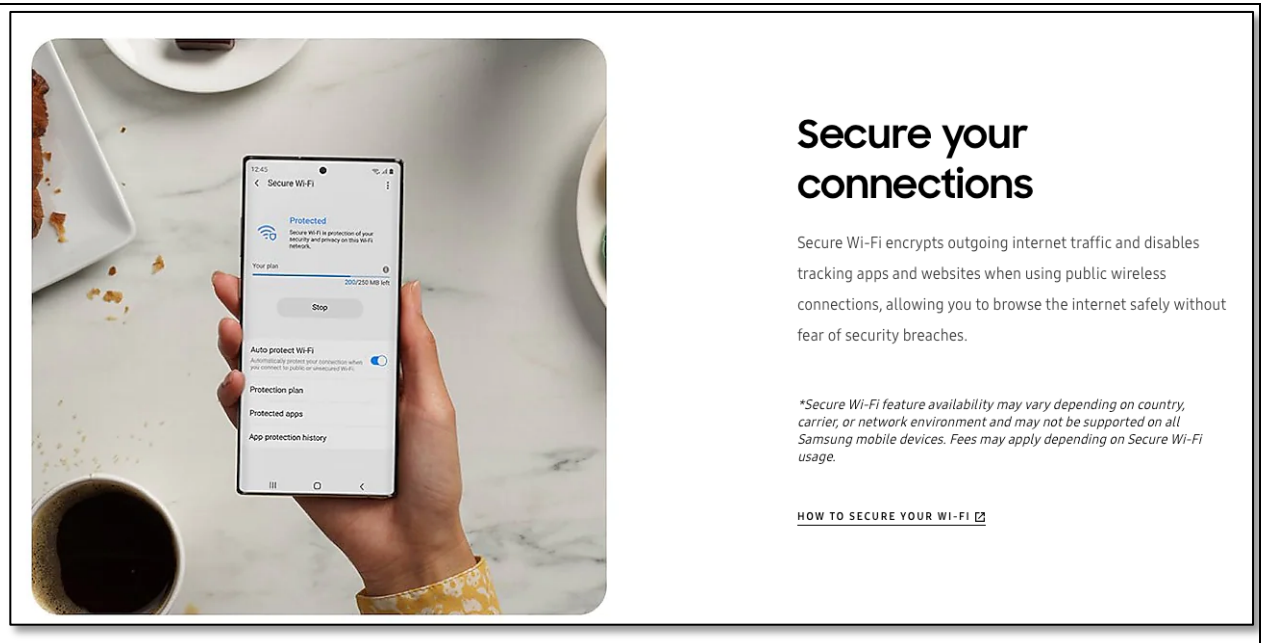
Claim 2

2. The message link server of claim 1, further comprising an encrypt function to encrypt the one or more messages supplied to the transport services stack	The Accused Devices comprise the "message link server of claim 1, further comprising an encrypt function to encrypt the one or more messages supplied to the transport services stack for delivery on the secure message link maintained between the message link server and the device link agent on the given one of the wireless end-user devices."

for delivery on the secure message link maintained between the message link server and the device link agent on the given one of the wireless end-user devices.	For example, the message link servers of the accused instrumentalities use encryption protocols to secure communications between the device and the network system. *See e.g.,* 

283

284



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

285

Firebase > Documentation > FCM > Engage

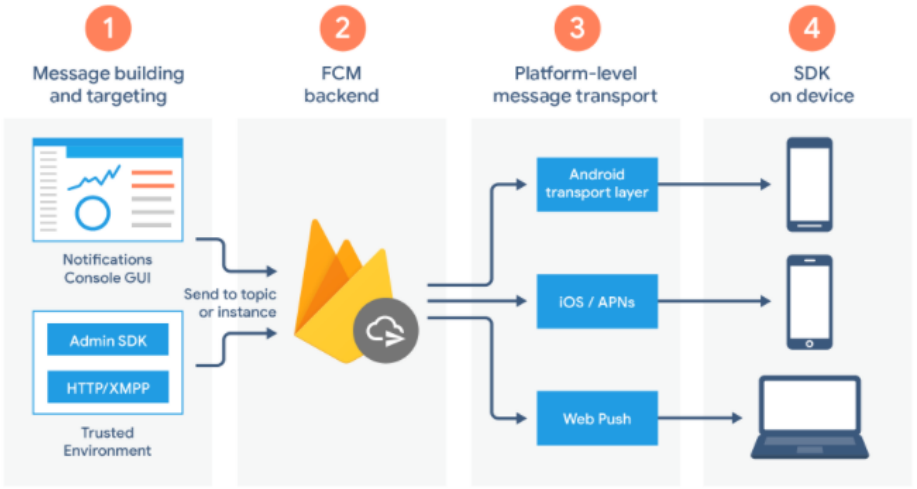
Was this helpful?

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.



286

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.

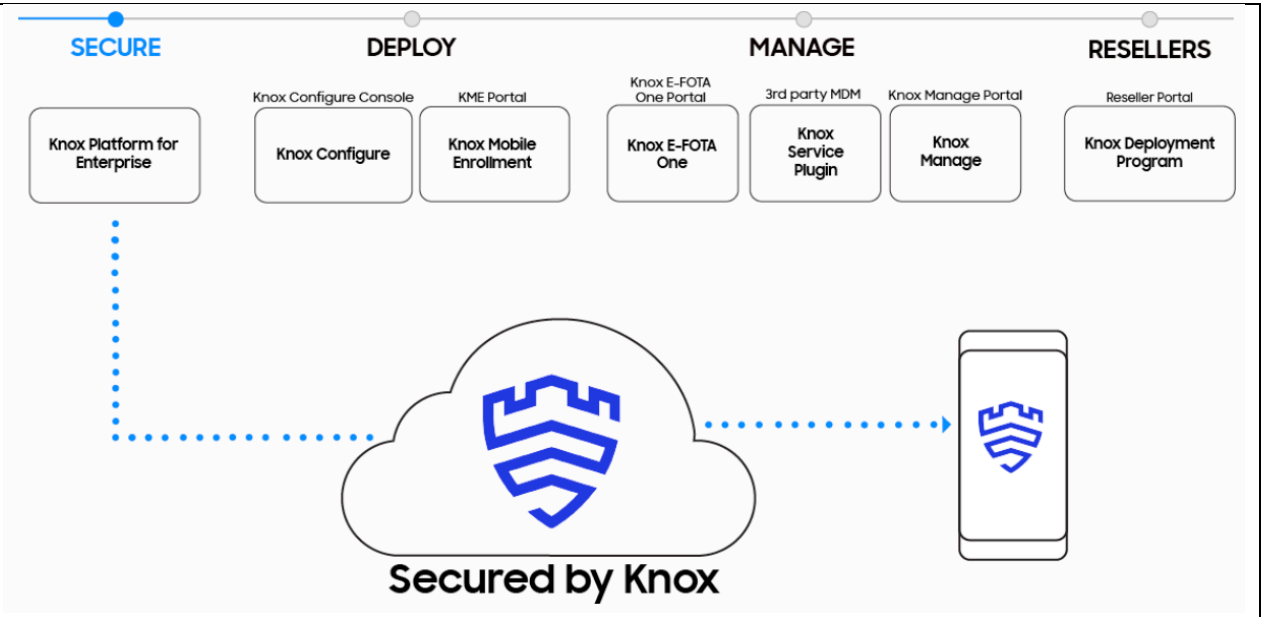
https://firebase.google.com/docs/cloud-messaging/fcm-architecture;

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.

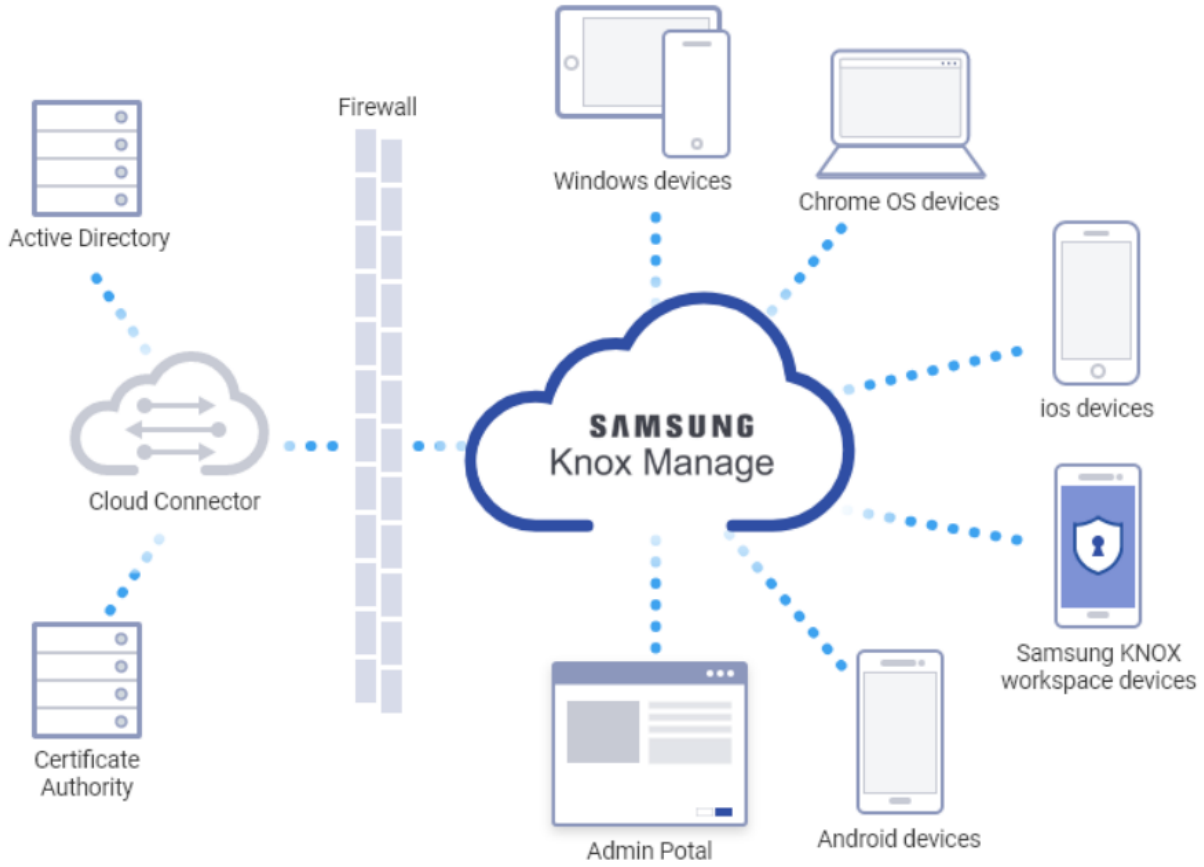
https://firebase.google.com/docs/cloud-messaging/concept-options#encryption_for_data_messages.

287



https://docs.samsungknox.com/admin/fundamentals/welcome.htm

288



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

The messages passed between servers of the accused instrumentalities and end user devices are also secured. As another example, the Tizen Push Server receives encrypted information from app servers and transmits such encrypted information on to device messaging agents residing on user devices, which in turn decrypt received messages.

Managing Security

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

- Keep the push application ID confidential.

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

- Do not store the registration ID on the device.

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

- Encrypt sensitive information.

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

- Do not hardcode the AppSecret in the source code.

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

Sending Push Notifications

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

The following example shows a sample push notification:

- URI: See the Push RQM (Request Manager) server URLs table.
- Method: HTTP POST
- Header:

```
appID: 1234567890987654
appSecret: dYo/o/m11gmWmjs7+5f+2zLNVOc=
```

- Body:

```
{
    "regID": "0501a53f4affdcbb98197f188345ff30c04b-5001",
    "requestID": "01231-22EAX-223442",
    "message": "badgeOption=INCREASE&badgeNumber=1&action=ALERT&alertMessage=Hi",
    "appData": "{id:asdf&passwd:1234}", /* Optional, if the message field is not empty */
}
```

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

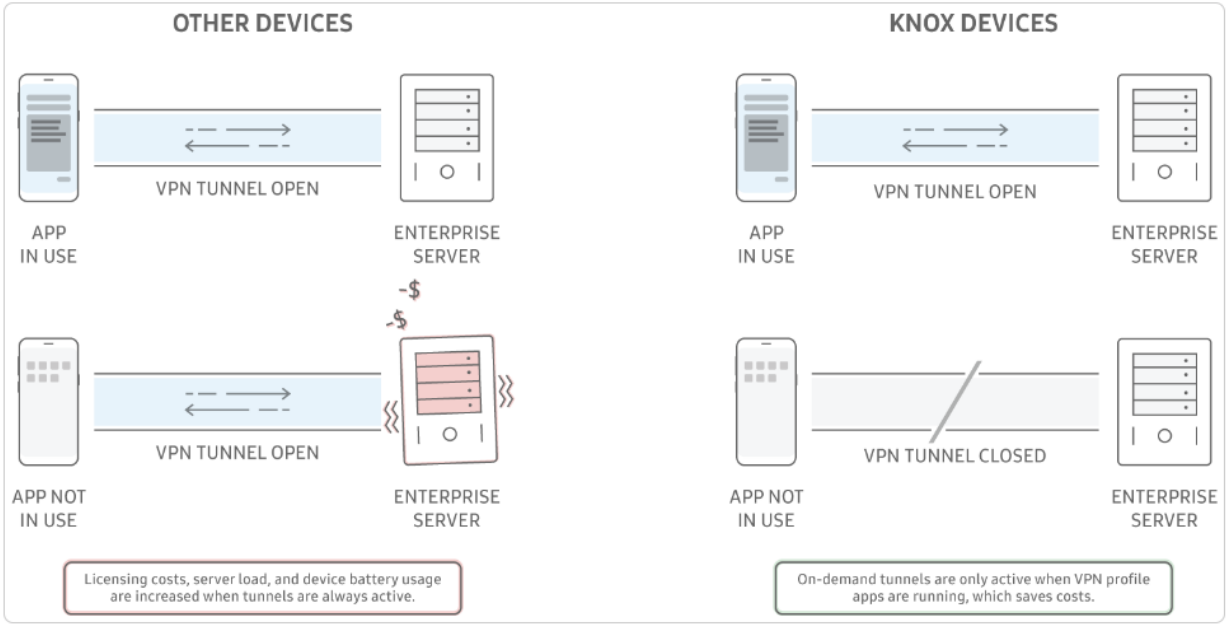
As another example, the Firebase framework encrypts messages in transit. See: https://firebase.google.com/docs/cloud-messaging/concept-options ("Encryption for data messages The Android Transport Layer (see FCM architecture) uses point-to-point encryption.").

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

Unique advantages of Knox VPN framework

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

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



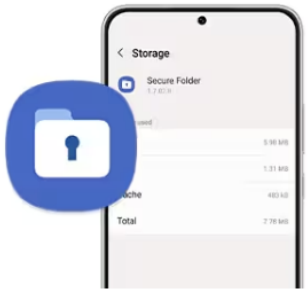
Claim 3

3. The message link server of claim 2, wherein the encrypted one or more messages are transported to the device link agent on the given one of the wireless end-user devices using one or more of encryption on the transport services stack, IP (Internet Protocol) layer encryption, and tunneling.	The accused instrumentalities comprise the "message link server of claim 2, wherein the encrypted one or more messages are transported to the device link agent on the given one of the wireless end-user devices using one or more of encryption on the transport services stack, IP (Internet Protocol) layer encryption, and tunneling." *See supra* claims 1 and 2.

Claim 4

4. The message link server of claim 1, wherein the device link agent executes in a secure execution environment on at least one of the devices, and at least one of the software components executes outside of the secure execution environment on that device.	The accused instrumentalities comprise the "message link server of claim 1, wherein the device link agent executes in a secure execution environment on at least one of the devices, and at least one of the software components executes outside of the secure execution environment on that device." *See supra* claim 1. As just one example, with Knox-managed devices, the Knox agent software residing on the user device executes from a specially protected partition of local memory. This specially protected partition is sometimes referred to as the Knox Secure Folder. Applications stored in the Secure Folder partition are managed separately with respect to push notifications. Other applications on the user device may be run from outside of the Knox Secure Folder partition.

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



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

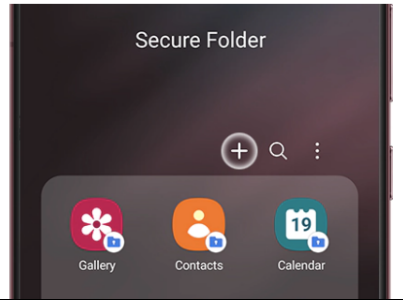
https://www.samsung.com/uk/support/mobile-devices/what-is-the-secure-folder-and-how-do-i-use-it/

Add apps or files in Secure Folder

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

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

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



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

Claim 5

5. The message link server of claim 1, wherein the transport services stack is further to receive, over each of the respective secure message links, upload messages forwarded by the respective device link agents from at least a subset of the device software components, each of the upload messages identifying a corresponding one of the network elements to which the device respective software component has requested delivery, the network server system using the interface to a network to deliver content from the upload messages to the respective identified network elements.	The accused instrumentalities comprise the "message link server of claim 1, wherein the transport services stack is further to receive, over each of the respective secure message links, upload messages forwarded by the respective device link agents from at least a subset of the device software components, each of the upload messages identifying a corresponding one of the network elements to which the device respective software component has requested delivery, the network server system using the interface to a network to deliver content from the upload messages to the respective identified network elements." *See supra* claim 1. For example, the end user devices of the accused instrumentalities cause their respective device link agents of a subset of the software components on the device to send messages to servers such as the App Servers (e.g., from Tizen devices), Knox Manage servers (from Knox managed devices), and push messaging servers (e.g., from Samsung's Android devices). These messages include those for sending authentication and verification information to servers.

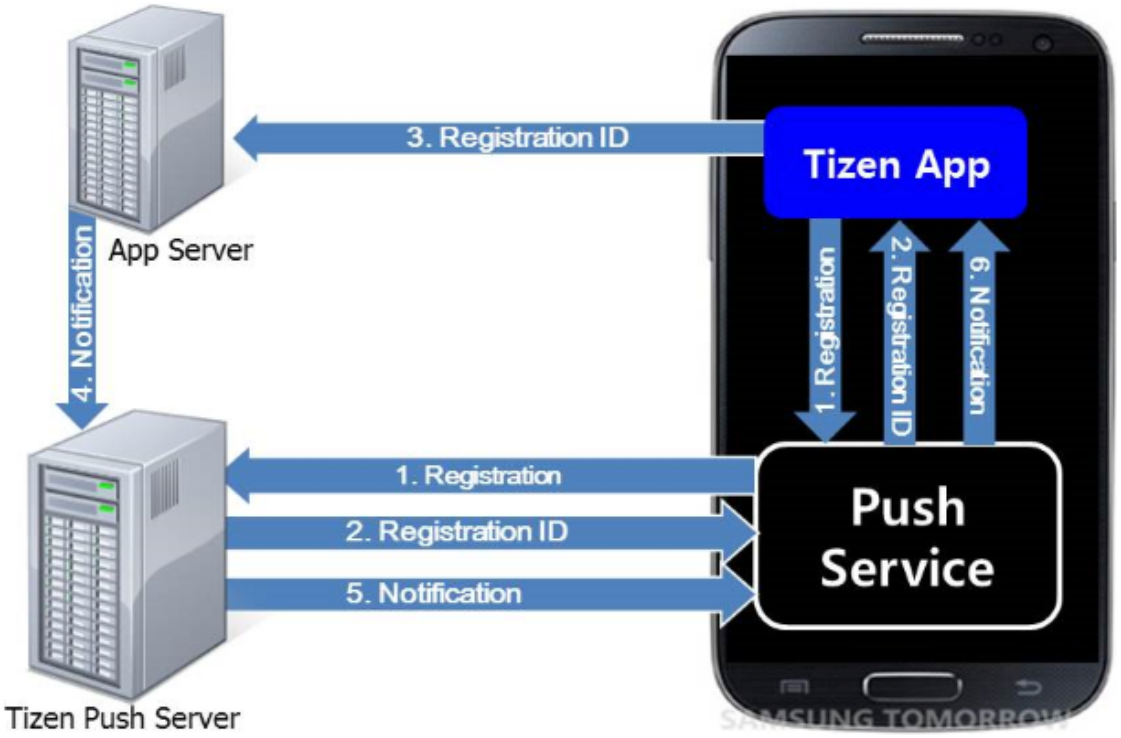
293

294

Architecture

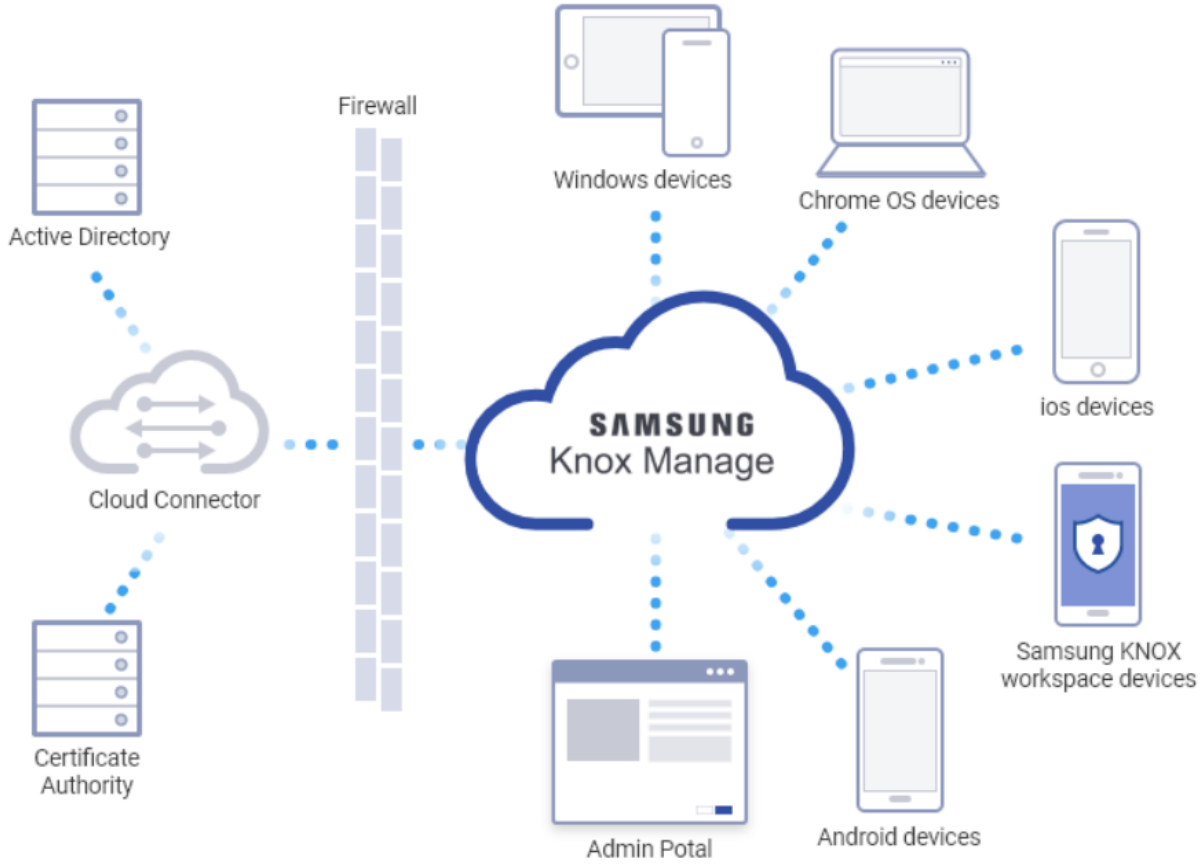
The architecture of the Tizen Push service is described in detail in the mobile native Push guide.

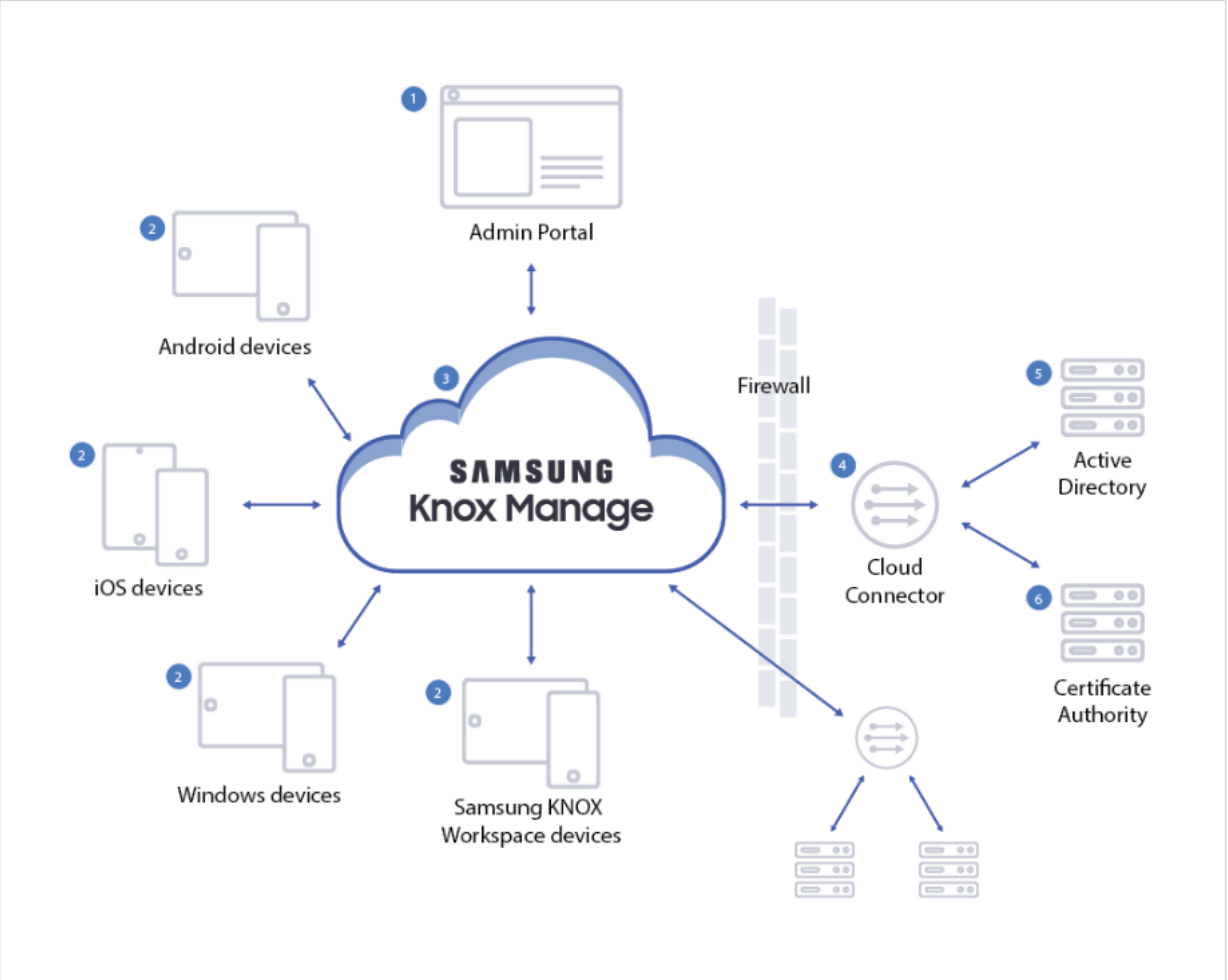
Figure: Service architecture



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.

295	https://docs.tizen.org/application/web/guides/messaging/push/;  https://docs.samsungknox.com/admin/knox-manage/welcome.htm

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

Claim 6

6. The message link server of claim 1, wherein at least one of the one or more messages for delivery by the transport	The accused instrumentalities comprise a "message link server of claim 1, wherein at least one of the one or more messages for delivery by the transport services stack comprises multiple identifier/data pairs."

services stack comprises multiple identifier/data pairs.	*See supra* claim 1. See, for example, https://firebase.google.com/docs/cloud-messaging/concept-options ("Data messages, by contrast, contain only your user-defined custom key-value pairs. Notification messages can contain an optional data payload. Maximum payload for both message types is 4000 bytes, except when sending messages from the Firebase console, which enforces a 1000 character limit."); https://docs.tizen.org/application/native/guides/messaging/push/ ("The message field contains not only the message to show in the quick panel on the device, but also the behaviors that the device must take when receiving the notification. The message field is a string that consists of key-value pairs. The available pair options are given in the following table."). On information and belief, messages transmitted to and from Knox servers also comprise multiple identifier/data pairs.

Claim 7

7. The message link server of claim 1, the device messaging agent on at least one of the wireless end-user devices further to initiate the respective secure Internet data message link to the transport services stack.	The accused instrumentalities comprise a "message link server of claim 1, the device messaging agent on at least one of the wireless end-user devices further to initiate the respective secure Internet data message link to the transport services stack." *See supra* claim 1, 5. As explained above, the device messaging agents on the devices may initiate secure connections to both app servers and the push messaging servers and Knox Manage servers to, inter alia, request push messages and other information and initiate authentication procedures.

Claim 8

8. The message link server of claim 1, further comprising a secure server to provide secure authorization signatures to the given one of the wireless end-user devices, the secure authorization signatures indicating the authorized software components that are allowed to	The accused instrumentalities comprise the "message link server of claim 1, further comprising a secure server to provide secure authorization signatures to the given one of the wireless end-user devices, the secure authorization signatures indicating the authorized software components that are allowed to receive data from secure message link messages via the message link server." *See supra* claim 1, 5. As explained above, the device messaging agents on the devices may initiate secure connections to both app servers and the push messaging servers and Knox Manage servers to, inter alia, request push messages and other information and initiate

receive data from secure message link messages via the message link server.	authentication procedures. Among these communications are transmissions from apps in the user devices to push messaging servers to register said applications as being permitted to receive messages and data from the servers. Likewise, Knox Manage software components on the user device communicate with Knox servers to authenticate the software on the user device to, among other things, allow the device to access Knox-related functionality over the network.

Claim 9

9. The message link server of claim 1, wherein one of the message delivery triggers is the expiration of a periodic timer.	The accused instrumentalities comprise the "message link server of claim 1, wherein one of the message delivery triggers is the expiration of a periodic timer." As explained in claim 1 *supra*, various triggers cause transmissions to occur between devices and push messaging servers and Knox servers. The transmission trigger may be a time period of inactivity or the expiration of a periodic timer. *See, e.g.*, https://docs.tizen.org/application/web/guides/messaging/push/: • "Store the notification at the push service database and request it later when the application is launched. This happens when the action is set to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application. For more information, see the Retrieving Missed Push Messages use case. The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification.

298

	Store the notification at the push service database and request it later when the application is launched. You need to set the action to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application. You can request for unread notifications from the push service. The request can be performed after connecting to the push server when the application is launched. • Request the unread notifications asynchronously. If there is such a notification, it can be received through the `_noti_cb()` function after the `_on_state_registered()` function returns. Once the request for unread notifications is successfully delivered, `PUSH_SERVICE_ERROR_NONE` is returned." 299

Claim 11

11. The message link server of claim 1, wherein one of the message delivery triggers is the receipt of a transmission on the respective secure message link from the device link agent of the given one of the wireless end-user devices, or a response generated to a transmission received from that device link agent.	The accused instrumentalities comprise the "message link server of claim 1, wherein one of the message delivery triggers is the receipt of a transmission on the respective secure message link from the device link agent of the given one of the wireless end-user devices, or a response generated to a transmission received from that device link agent." As explained in claim 1 *supra*, various triggers cause transmissions to occur between devices and push messaging servers and Knox servers. The transmission trigger may be the receipt by the messaging server of a notification from a user device that is has started up and/or is requesting push messages. *See, e.g.*, https://docs.tizen.org/application/web/guides/messaging/push/:

	• “Store the notification at the push service database and request it later when the application is launched. This happens when the action is set to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application. For more information, see the Retrieving Missed Push Messages use case. The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification. Store the notification at the push service database and request it later when the application is launched. You need to set the action to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application. You can request for unread notifications from the push service. The request can be performed after connecting to the push server when the application is launched. • Request the unread notifications asynchronously. If there is such a notification, it can be received through the `_noti_cb()` function after the `_on_state_registered()` function returns. Once the request for unread notifications is successfully delivered, `PUSH_SERVICE_ERROR_NONE` is returned.”

300

Claim 12

12. The message link server of claim 11, wherein the transmission is a heartbeat message generated by the given device link agent, or a request received from the given device link agent.	The accused instrumentalities comprise the "message link server of claim 11, wherein the transmission is a heartbeat message generated by the given device link agent, or a request received from the given device link agent." As explained in claim 1 *supra*, various triggers cause transmissions to occur between devices and push messaging servers and Knox servers. The transmission trigger may be the receipt by the messaging server of a notification from a user device that is has started up and/or is requesting push messages. *See, e.g.*, https://docs.tizen.org/application/web/guides/messaging/push/: • "Store the notification at the push service database and request it later when the application is launched. This happens when the action is set to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application. For more information, see the Retrieving Missed Push Messages use case. The difference between the `ALERT` and `SILENT` actions is that the former shows an alert message in the quick panel and changes the badge count, while the latter does not. If the user clicks the alert message in the quick panel, the push service forcibly launches the application and delivers the notification. Store the notification at the push service database and request it later when the application is launched.	

	You need to set the action to `ALERT` or `SILENT` in the message field when sending the notification from the application server. When such a notification arrives at the device, the push service keeps the notification in the database and waits for the request from the application. You can request for unread notifications from the push service. The request can be performed after connecting to the push server when the application is launched. • Request the unread notifications asynchronously. If there is such a notification, it can be received through the `_noti_cb()` function after the `_on_state_registered()` function returns. Once the request for unread notifications is successfully delivered, `PUSH_SERVICE_ERROR_NONE` is returned."	

302

Claim 13

13. The message link server of claim 1, wherein one of the message delivery triggers is the receipt of a particular network element message from one of the network elements.	The accused instrumentalities comprise the "message link server of claim 1, wherein one of the message delivery triggers is the receipt of a particular network element message from one of the network elements." As explained *supra* for claim 1, app servers may communicate with push messaging servers to cause messages to be sent to user devices.

Claim 15

15. A method of operating a message link server, comprising:	*See* supra claim 1.
maintaining a respective secure message link through an Internet network between the message link server and a respective device link agent on each of a plurality of	The accused instrumentalities comprise a method including "maintaining a respective secure message link through an Internet network between the message link server and a respective device link agent on each of a plurality of wireless end-user devices, each of the wireless end-

wireless end-user devices, each of the wireless end-user devices comprising multiple software components authorized to receive and process data from secure message link messages received via a device link agent on that device;	user devices comprising multiple software components authorized to receive and process data from secure message link messages received via a device link agent on that device." *See* supra claim 1[a].
receiving network element messages from a plurality of network elements, the received network element messages comprising respective message content and requests for delivery of the respective message content to respective wireless end-user devices, the respective message content including data for, and an identification of, a respective one of the authorized software components;	The accused instrumentalities comprise a method including "receiving network element messages from a plurality of network elements, the received network element messages comprising respective message content and requests for delivery of the respective message content to respective wireless end-user devices, the respective message content including data for, and an identification of, a respective one of the authorized software components." *See* supra claim 1[b].
buffering content from the received network element messages for which delivery is requested to a given one of the wireless end-user devices;	The accused instrumentalities comprise a method including "buffering content from the received network element messages for which delivery is requested to a given one of the wireless end-user devices." *See* supra claim 1[c].
determining when one of a plurality of message delivery triggers for the given one of the wireless end-user devices has occurred, wherein for at least some of the received network element messages, the receipt of such a message is not a message delivery trigger, and for at least one of the message delivery triggers, the trigger is an occurrence of an asynchronous event with time-critical messaging needs; and	The accused instrumentalities comprise a method including "determining when one of a plurality of message delivery triggers for the given one of the wireless end-user devices has occurred, wherein for at least some of the received network element messages, the receipt of such a message is not a message delivery trigger, and for at least one of the message delivery triggers, the trigger is an occurrence of an asynchronous event with time-critical messaging needs." *See* supra claim 1[d].
upon determining that one of the message delivery triggers has occurred, supplying one or more messages comprising the	The accused instrumentalities comprise a method including "upon determining that one of the message delivery triggers has occurred, supplying one or more messages comprising the

buffered content for delivery on the secure message link maintained between the message link server and a device link agent on the given one of the wireless end-user devices.	buffered content for delivery on the secure message link maintained between the message link server and a device link agent on the given one of the wireless end-user devices." *See* supra claim 1[e].