Développement Android avec Kotlin

Cours - 11 - Notifications

Jordan Hiertz

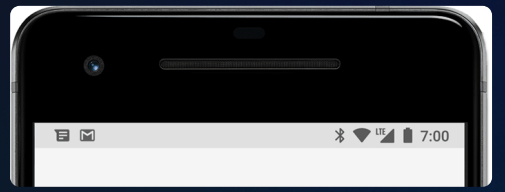
Contact
hiertzjordan@gmail.com
jordan.hiertz@al-enterprise.com

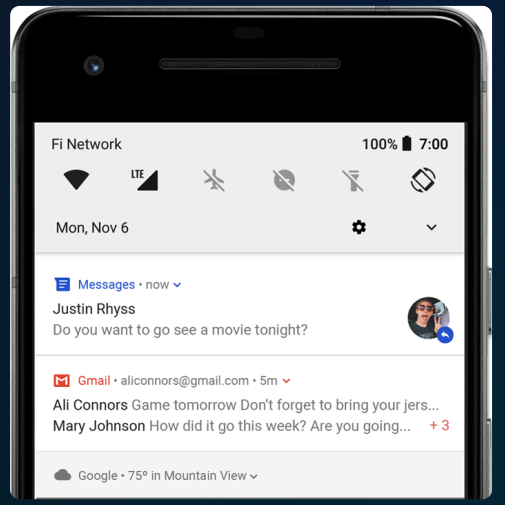
Présentation des notifications

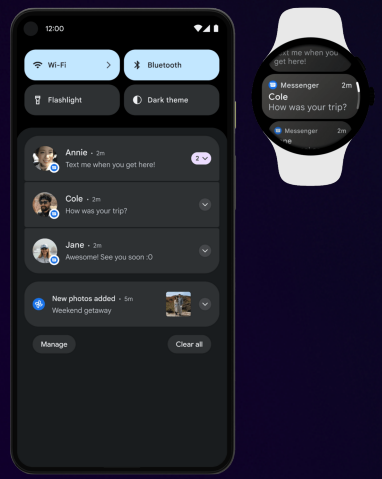
- Une notification est un message qu'Android affiche en dehors de l'UI de l'application.
- Utilisées pour des rappels, messages, mises à jour en temps réel.
- L'utilisateur peut appuyer dessus pour ouvrir l'application ou effectuer des actions directement.

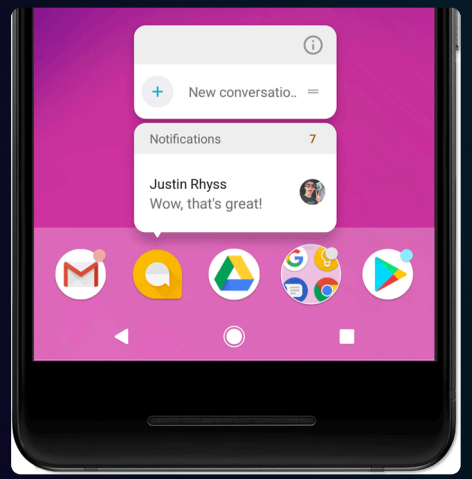
Affichage des notifications

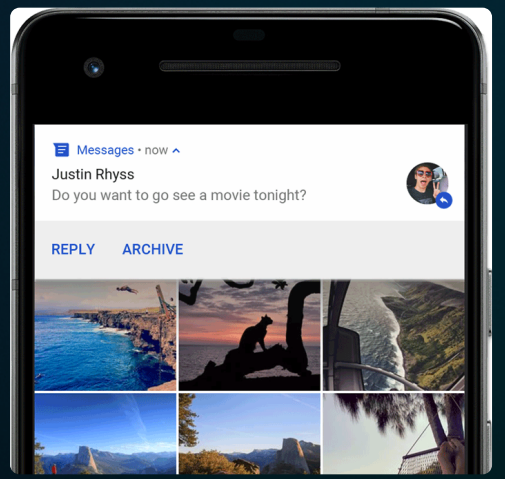
- **Les notifications apparaissent à différents endroits :**
  - Dans la barre d'état sous forme d'icône de notification
  - Détaillé dans le panneau des notifications
  - Dans une fenêtre flottante appelée notification prioritaire
  - Sous forme de badge sur l'icône de l'application
  - Sur l'écran de verrouillage avec le contenu sensible masqué
  - Sur les accessoires connectés (montres etc.)





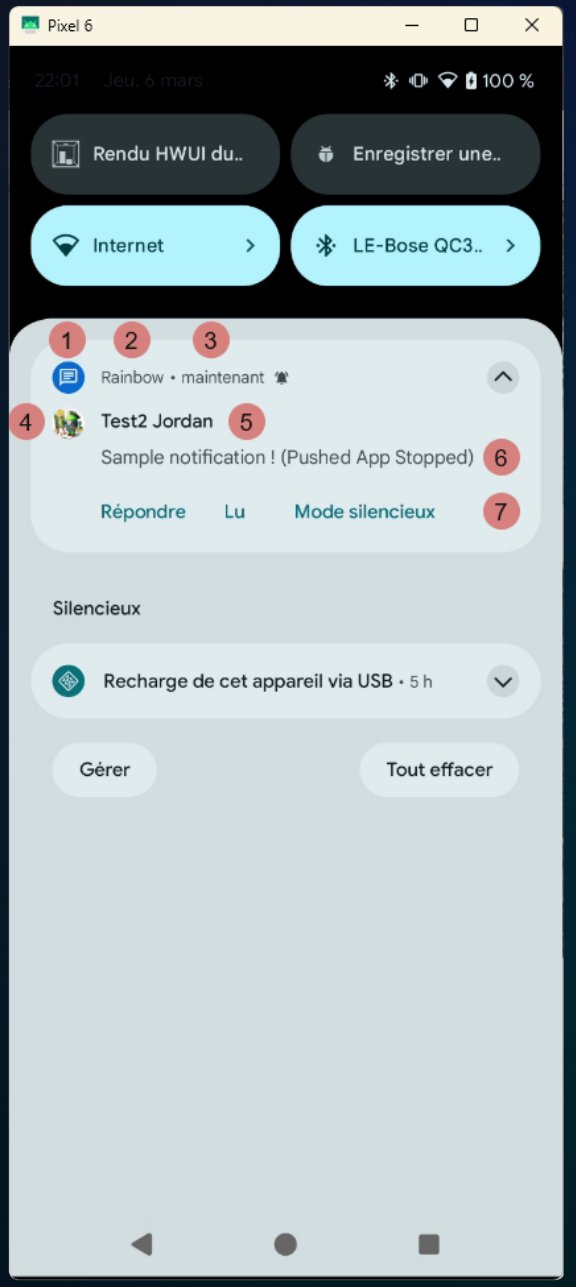






Anatomie d'une notification

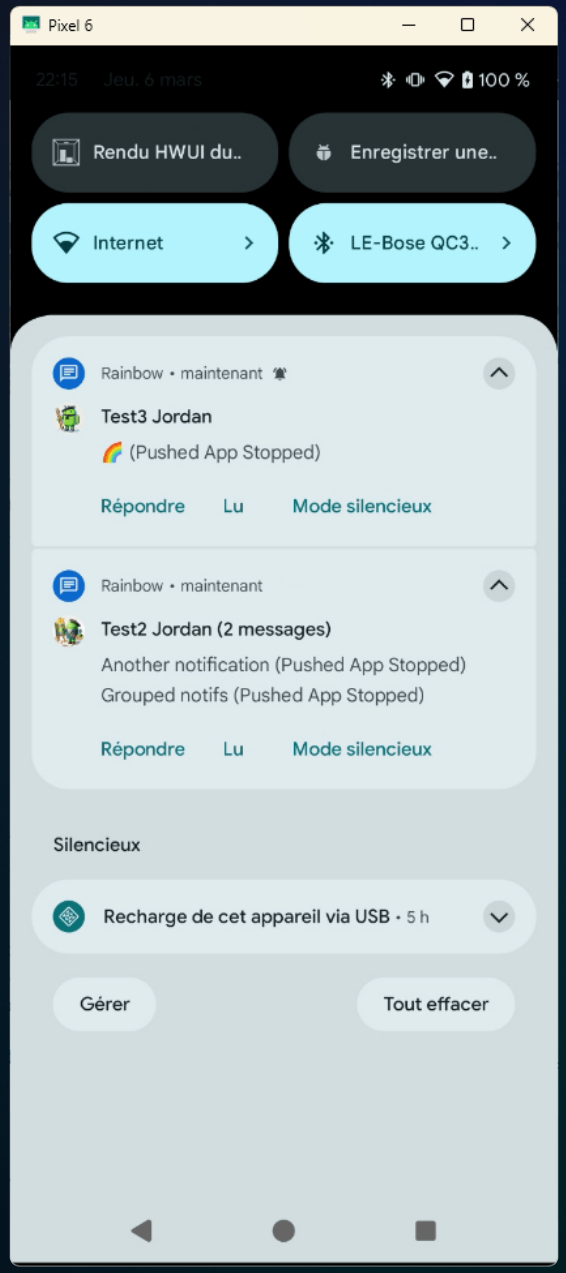
1. Petite icône (obligatoire) → `setSmallIcon()`
2. Nom de l'application (fourni par le système)
3. Timestamp (fourni par le système mais modifiable)
4. Grande icône → `setLargeIcon()`
5. Titre → `setContentTitle()`
6. Texte → `setContentText()`
7. Actions → `addAction()`



Groupes de notifications

- Il est possible de mettre à jour une notification existante, cela peut être pertinent pour ne pas spammer l'utilisateur.

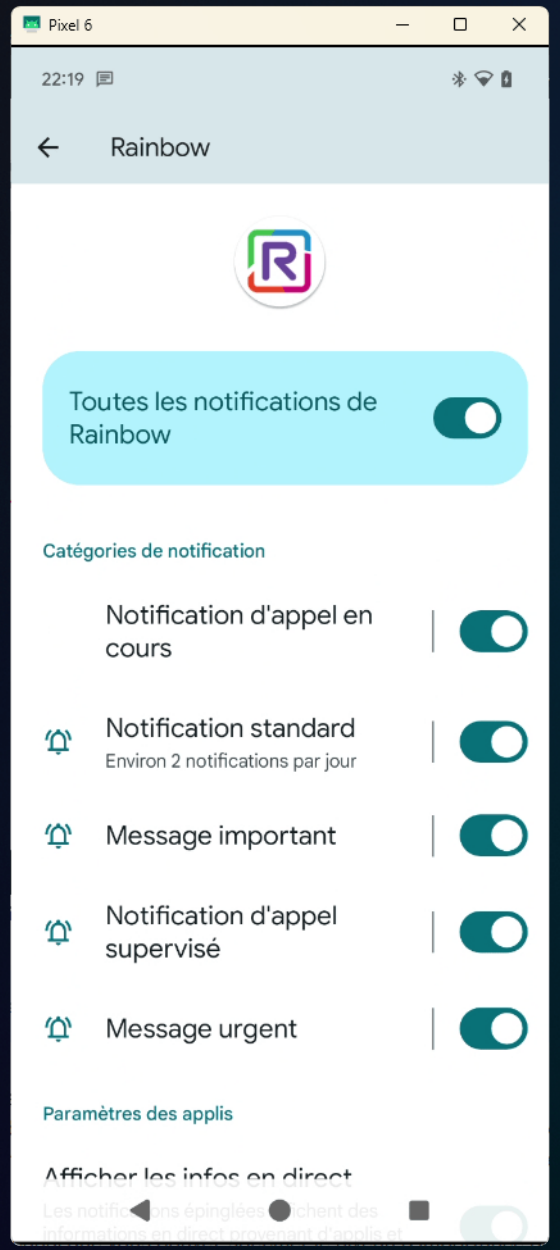
- On peut également groupé les notifications (*Android 7.0*+)
  - Affichage condensé avec un résumé
  - Expansion progressive pour voir les détails

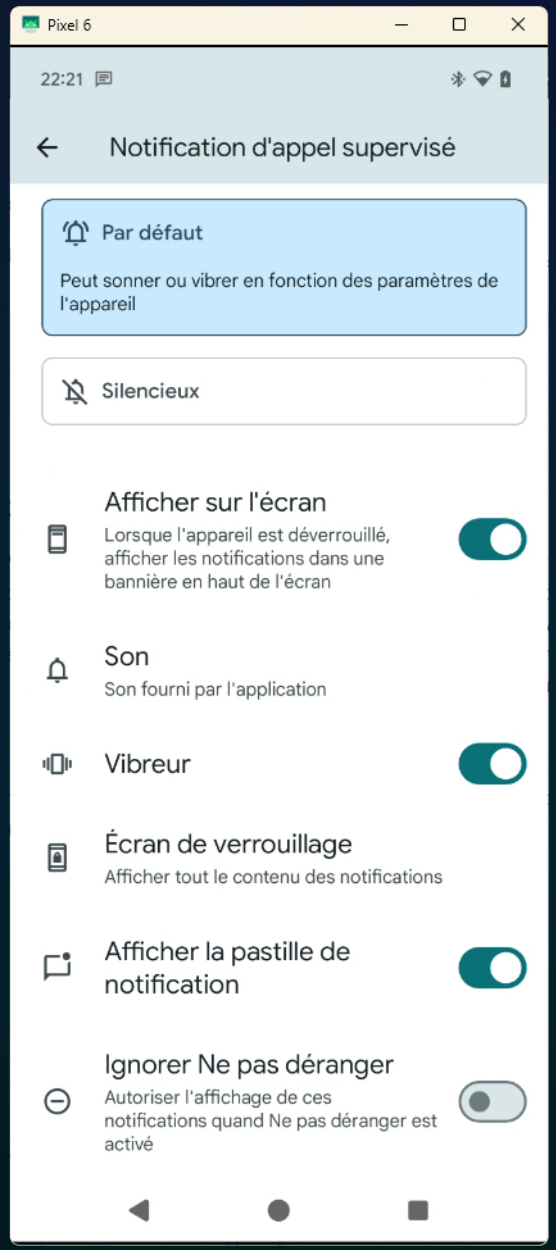


Canaux de notification (Android 8.0+)

- **Obligatoires** → Sans canal, une notification ne s'affiche pas
- **Contrôle utilisateur** → Activer/désactiver des canaux spécifiques
- **Personnalisation** → Options visuelles et sonores ajustables
- **Gestion rapide** → Appui long sur une notification pour gérer son canal

Avant Android 8.0 → Une seule catégorie de notification par application





Compatibilité des notifications

Les notifications évoluent constamment avec les mises à jour d'Android, et pour garantir une compatibilité entre les différentes versions, il est recommandé d'utiliser **NotificationCompat** et **NotificationManagerCompat**. Ces bibliothèques gèrent automatiquement les différences entre les API, évitant ainsi la nécessité de vérifier le niveau de l'API dans votre code.

Autorisation de notification

Depuis Android 13 (API 33), les applications doivent explicitement demander la permission d'envoyer des notifications. Sans cette autorisation, l'application ne pourra pas notifier l'utilisateur.

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android" ...>

  <uses-permission android:name="android.permission.POST_NOTIFICATIONS"/>

  <application ...></application>
</manifest>
```

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
    if (ContextCompat.checkSelfPermission(this, Manifest.permission.POST_NOTIFICATIONS) != PackageManager.PERMISSION_GRANTED
    ) {
        ActivityCompat.requestPermissions(
            this, arrayOf(Manifest.permission.POST_NOTIFICATIONS), REQUEST_CODE_NOTIFICATIONS
        )
    }
}
```

Canaux de notification

- Une fois qu'un canal de notification est créé, vous ne pouvez plus modifier ses comportements (importance, son, vibrations, etc.). L'utilisateur a alors un contrôle total sur ses paramètres.
- Vous pouvez toutefois modifier le nom et la description d'un canal après sa création.
- Il est recommandé de créer un canal pour chaque type de notification que vous souhaitez envoyer.
- Si votre `targetSdkVersion` est inférieur ou égal à 25, votre application fonctionnera sous Android 8.0+ comme elle le ferait sur Android 7.1 et versions antérieures.
- Cependant, pour les applications ciblant Android 8.0+ (API 26+), il est impératif de créer au moins un canal de notification.

Canaux de notification

1. **Créer un objet `NotificationChannel`**
   - Définir un identifiant unique
   - Spécifier un nom visible par l'utilisateur
   - Définir un niveau d'importance (`IMPORTANCE_HIGH`, `IMPORTANCE_DEFAULT`, etc.)
2. **(Facultatif) Ajouter une description** avec `setDescription()`
3. **Enregistrer le canal** auprès du `NotificationManager` avec `createNotificationChannel()`

```
1  if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
2      val channelId = "my_channel_id"
3      val channelName = "My Channel"
4      val channelDescription = "Description of My Channel"
5      val importance = NotificationManager.IMPORTANCE_DEFAULT
6
7      val channel = NotificationChannel(channelId, channelName, importance).apply {
8          description = channelDescription
9      }
10
11     (getSystemService(NOTIFICATION_SERVICE) as? NotificationManager)?.createNotificationChannel(channel)
12 }
```

Canaux de notification

```kotlin
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
    val channelId = "my_advanced_channel"
    val channelName = "Advanced Channel"
    val channelDescription = "This channel includes lights, vibrations, and sound"
    val importance = NotificationManager.IMPORTANCE_HIGH

    val channel = NotificationChannel(channelId, channelName, importance).apply {
        description = channelDescription
        enableLights(true)
        lightColor = Color.RED
        enableVibration(true)
        vibrationPattern = longArrayOf(0, 500, 250, 500)
        setSound(RingtoneManager.getDefaultUri(RingtoneManager.TYPE_NOTIFICATION), null)
    }

    (getSystemService(NOTIFICATION_SERVICE) as? NotificationManager)?.createNotificationChannel(channel)
}
```

Canaux de notification

```
1 // Delete a notification channel
2 val notificationManager = getSystemService(Context.NOTIFICATION_SERVICE) as NotificationManager
3 val id: String = "my_advanced_channel"
4 (getSystemService(NOTIFICATION_SERVICE) as? NotificationManager)?.deleteNotificationChannel(id)
```

Canaux de notification

Le niveau d'importance détermine le comportement d'une notification lorsqu'elle est reçue.

Niveau visible par l'utilisateur	**Importance (Android 8.0+)**	**Priorité (Android 7.1-)**	**Comportement**
Urgent	`IMPORTANCE_HIGH`	`PRIORITY_HIGH` ou `PRIORITY_MAX`	Signal sonore + bannière en haut de l'écran
Élevé	`IMPORTANCE_DEFAULT`	`PRIORITY_DEFAULT`	Signal sonore mais pas de bannière en haut de l'écran
Moyen	`IMPORTANCE_LOW`	`PRIORITY_LOW`	Pas de son, s'affiche dans la barre d'état et panneau de notifications
Faible	`IMPORTANCE_MIN`	`PRIORITY_MIN`	Seulement dans le panneau des notifications
Aucun	`IMPORTANCE_NONE`	N/A	N'apparaît nulle part

Créer une notification - Dépendances

1️⃣ Ajouter la bibliothèque principale AndroidX

```
[versions]
coreKtx = "1.15.0"

[libraries]
androidx-core-ktx = { group = "androidx.core", name = "core-ktx", version.ref = "coreKtx" }
```

```
implementation(libs.androidx.core.ktx)
```

📌 **Remarque :** Si vous utilisez déjà d'autres bibliothèques Jetpack, `core-ktx` est probablement inclus en tant que dépendance transitive. Dans ce cas, vous avez déjà accès à `NotificationCompat`.

Créer une notification - Builder

2️⃣ Exemple de notification avec `NotificationCompat.Builder`

```kotlin
val channelId = "my_channel_id"
val notificationId = 1

val notification = NotificationCompat.Builder(this, channelId)
    .setSmallIcon(R.drawable.ic_notification)
    .setContentTitle("Titre de la notification")
    .setContentText("Ceci est le contenu de la notification.")
    .setPriority(NotificationCompat.PRIORITY_HIGH) // Priorité pour Android < 8.0
    .setAutoCancel(true) // Supprime la notification quand l'utilisateur clique dessus
    .build()

val notificationManager = getSystemService(NotificationManager::class.java)
notificationManager.notify(notificationId, notification)
```

Créer une notification - Action principale

Définir une action à effectuer en appuyant sur la notification

```kotlin
// Create an explicit intent for an Activity in your app.
val intent = Intent(this, AlertDetails::class.java).apply {
    flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLEAR_TASK
}

val pendingIntent: PendingIntent = PendingIntent.getActivity(this, 0, intent, PendingIntent.FLAG_IMMUTABLE)

val channelId = "my_channel_id"
val notificationId = 1

val notification = NotificationCompat.Builder(this, channelId)
    .setSmallIcon(R.drawable.ic_notification)
    .setContentTitle("Titre de la notification")
    .setContentText("Ceci est le contenu de la notification.")
    .setPriority(NotificationCompat.PRIORITY_HIGH) // Priorité pour Android < 8.0
    .setContentIntent(pendingIntent)
    .setAutoCancel(true) // Supprime la notification quand l'utilisateur clique dessus
    .build()

val notificationManager = getSystemService(NotificationManager::class.java)
notificationManager.notify(notificationId, notification)
```

Créer une notification - Actions supplémentaires

Ajouter des boutons d'action à une notification

Une notification peut inclure **jusqu'à trois boutons d'action**, permettant à l'utilisateur d'interagir sans ouvrir l'application. Ces actions doivent être **distinctes** de l'action principale de la notification.

```
val intent = Intent(ACTION_MARK_AS_READ_IM_NOTIFICATION)
    .putExtra(MESSAGE_IDS_PARAMETER, ids)
    .putExtra(CONV_ID_PARAMETER, conv.id)

val pendingIntent = PendingIntent.getBroadcast(
     application,
     nextBroadcastReceiverRequestCode,
     intent,
     pendingIntentFlags
)

val action = NotificationCompat.Action.Builder(R.drawable.ic_check, application.getString(R.string.readLabel), pendingIntent)
    .setSemanticAction(NotificationCompat.Action.SEMANTIC_ACTION_MARK_AS_READ)
    .setShowsUserInterface(false)
    .build()



builder.addAction(action)
```

Créer une notification - Répondre directement

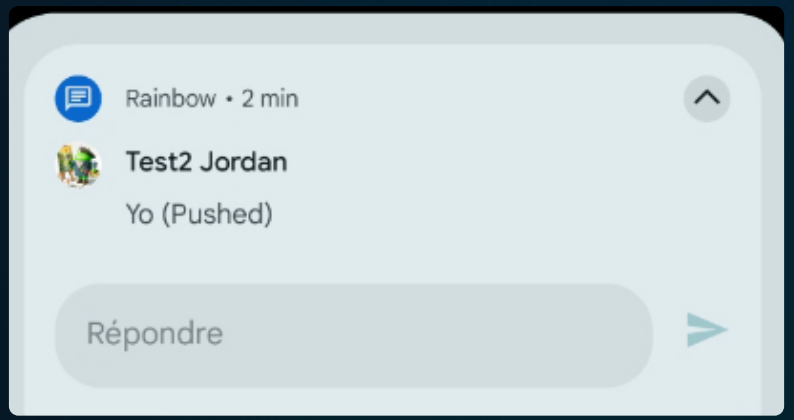
Répondre dans une notification

Disponible depuis Android 7.0 (API 24), l'utilisateur peut saisir du texte directement dans la notification. Le texte est envoyé à l'application sans ouvrir d'activité.

```
// Key for the string that's delivered in the action's intent.
private val KEY_TEXT_REPLY = "key_text_reply"
var replyLabel: String = resources.getString(R.string.reply_label)
var remoteInput: RemoteInput = RemoteInput.Builder(KEY_TEXT_REPLY).run {
    setLabel(replyLabel)
    build()
}

var replyPendingIntent: PendingIntent =
  PendingIntent.getBroadcast(applicationContext,
    conversation.getConversationId(),
    getMessageReplyIntent(conversation.getConversationId()),
    PendingIntent.FLAG_UPDATE_CURRENT)


var action: NotificationCompat.Action =
  NotificationCompat.Action.Builder(R.drawable.ic_reply_icon,
    getString(R.string.label), replyPendingIntent)
  .addRemoteInput(remoteInput)
  .build()
```



Créer une notification - Catégorie

Définir une catégorie au niveau du système

Android utilise des catégories prédéfinies pour déterminer si une notification doit déranger l'utilisateur lorsqu'il active le mode "Ne pas déranger" (Do Not Disturb).

```
var builder = NotificationCompat.Builder(this, CHANNEL_ID)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("My notification")
        .setContentText("Hello World!")
        .setPriority(NotificationCompat.PRIORITY_DEFAULT)
        .setCategory(NotificationCompat.CATEGORY_MESSAGE)
```

Créer une notification - Écran de verrouillage

Visibilité sur l'écran de verrouillage

Lorsque vous créez une notification, vous pouvez contrôler le niveau de détail visible sur l'écran de verrouillage à l'aide de la méthode `setVisibility()`. Cette méthode accepte trois valeurs principales :

- **VISIBILITY_PUBLIC** : Cette option permet d'afficher le contenu complet de la notification sur l'écran de verrouillage.
- **VISIBILITY_SECRET** : Lorsque vous choisissez cette option, aucune partie de la notification ne s'affiche sur l'écran de verrouillage.
- **VISIBILITY_PRIVATE** : Cette option affiche uniquement les informations de base sur l'écran de verrouillage, telles que l'icône de la notification et le titre du contenu.

```
1 val privateNotification = NotificationCompat.Builder(context, CHANNEL_ID)
2     .setContentTitle(title)
3     .setVisibility(NotificationCompat.VISIBILITY_PRIVATE)
4     .build()
```

Mettre à jour et supprimer une notification

Pour mettre à jour une notification déjà affichée, il suffit d'appeler à nouveau `NotificationManagerCompat.notify()` en lui spécifiant le même ID que celui utilisé précédemment. Le système remplacera la précédente si elle existe encore.

Les notifications sur Android restent visibles jusqu'à ce qu'un événement spécifique survienne. Plusieurs mécanismes permettent de gérer leur suppression :

- Si l'utilisateur ignore la notification (en la balayant, par exemple), elle sera automatiquement supprimée.
- Si vous définissez `setAutoCancel()` lors de la création de la notification, elle sera automatiquement supprimée dès que l'utilisateur appuie dessus.
- Vous pouvez appeler la méthode `cancel()` en lui fournissant l'ID de la notification pour supprimer cette notification spécifique.
- Si vous souhaitez supprimer toutes les notifications émises par votre application, vous pouvez appeler `cancelAll()`.
- Lors de la création de la notification, vous pouvez définir un délai avant expiration avec `setTimeoutAfter()`

Présentation de Firebase Cloud Messaging (FCM)

Qu'est-ce que FCM ?

- Service de messagerie push de Google permettant d'envoyer des notifications et des messages de données aux applications Android, iOS et Web.
- Utilisé pour informer l'utilisateur d'un événement important (nouveau message, mise à jour, alerte, etc.).

Pourquoi utiliser FCM ?

- Réduit la consommation d'énergie par rapport aux alternatives (websockets, polling, long polling).
- Prise en charge du ciblage (utilisateur unique, groupe, topic).
- Fonctionne en arrière-plan et peut réveiller l'application si nécessaire.

Fonctionnement global de FCM

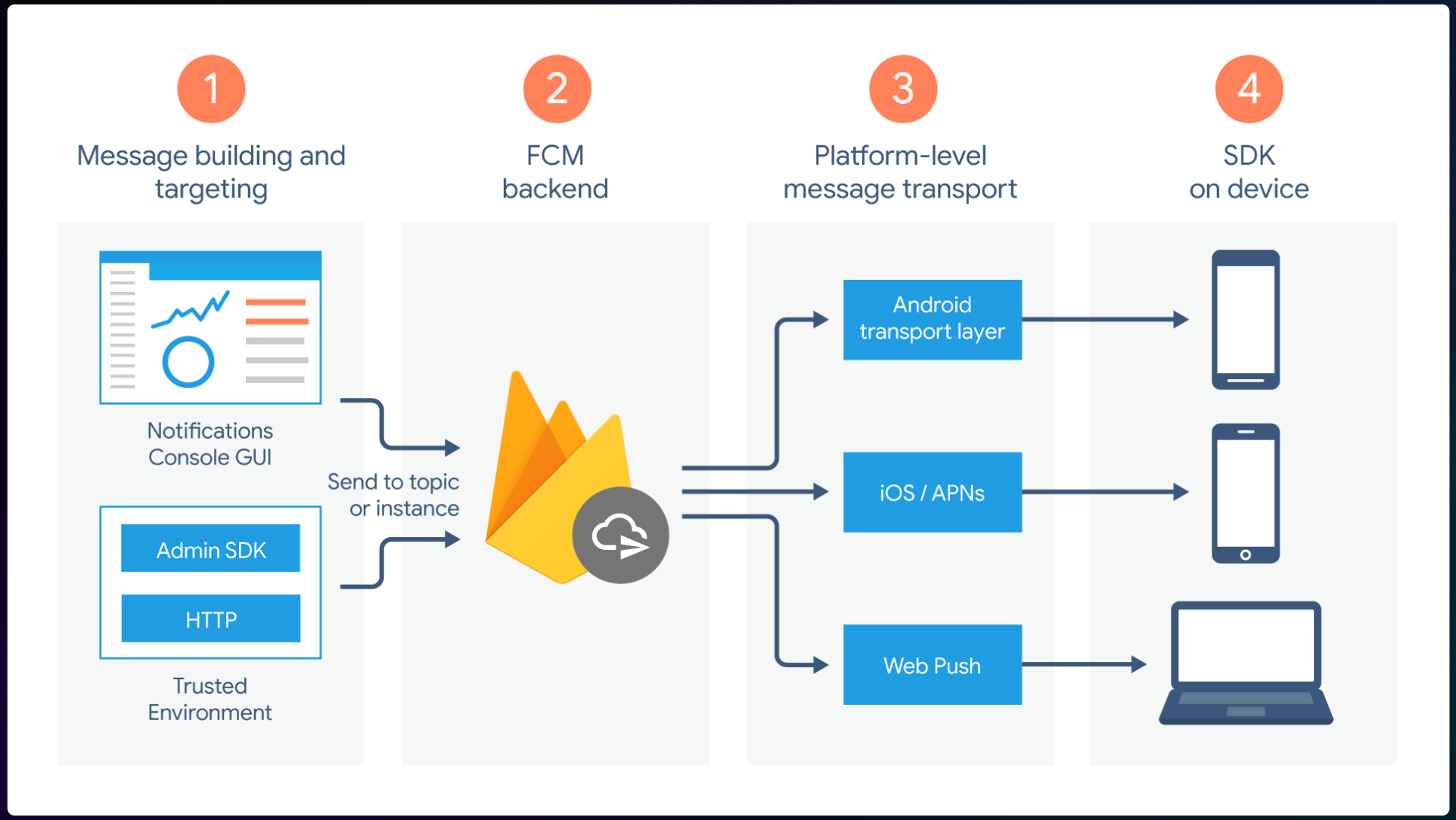
Comment ça marche ?

1. **L'application Android récupère un token FCM**
   - Chaque appareil se voit attribuer un **token unique** par FCM.
   - Ce token est utilisé pour identifier l'appareil auprès du serveur.

2. **Le serveur envoie une requête HTTP ou gRPC à FCM**
   - Le serveur doit être authentifié via une **clé d'API** ou un **Jeton OAuth 2.0**.
   - Il envoie un message contenant le **token cible** et le **contenu de la notification**.

3. **FCM transmet le message aux appareils concernés**
   - Un Service qui hérite de `FirebaseMessagingService` gère les actions à réaliser.

Fonctionnement global de FCM



Fonctionnement global de FCM

```
{
  "message":{
    "token":"bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
    "notification":{
      "title":"Portugal vs. Denmark",
      "body":"great match!"
    }
  }
}
```

```
{
  "message":{
    "token":"bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
    "data":{
      "Nick" : "Mario",
      "body" : "great match!",
      "Room" : "PortugalVSDenmark"
    }
  }
}
```

```
{
  "message":{
    "token":"bk3RNwTe3H0:CI2k_HHwgIpoDKCIZvvDMExUdFQ3P1...",
    "notification":{
      "title":"Match update",
      "body":"Arsenal goal in added time, score is now 3-0"
    },
    "android":{
      "ttl":"86400s",
      "notification":{
        "click_action":"OPEN_ACTIVITY_1"
      }
    },
    "apns": {
      "headers": {
        "apns-priority": "5",
      },
      "payload": {
        "aps": {
          "category": "NEW_MESSAGE_CATEGORY"
        }
      }
    },
    "webpush":{
      "headers":{
        "TTL":"86400"
      }
    }
  }
}
```

Conclusion

- **Les notifications** : Permettent de créer et personnaliser des alertes pour informer l'utilisateur.
- **Les groupes de notifications** : Aident à organiser les notifications similaires pour éviter l'encombrement.
- **Les canaux de notifications** : Permettent de gérer les préférences de notification des utilisateurs.
- **NotificationCompat** : Assure la compatibilité des notifications sur différentes versions d'Android.
- **Firebase Cloud Messaging (FCM)** : Permet d'envoyer des notifications push, même lorsque l'application est en arrière-plan ou fermée.

Ces concepts sont essentiels pour offrir une expérience utilisateur fluide et efficace dans les applications Android modernes.