Migrate from Hilt to Koin.

A cheat sheet.

Kotlin Multiplatform

Koin is fully KMP compatible.

kotzilla

Koin is developed by Kotzilla, with open-source contributors.

kotzilla.io

Migrate from Hilt to Koin

Cheat Sheet for Android

Koin is a Kotlin dependency injection framework to help you build any kind of Kotlin application, from Android mobile to backend Ktor server applications, **including Kotlin Multiplatform** and Compose.

What is it for ?

This cheat sheet provides a quick reference for Android developpers migrating from Hilt to Koin, covering the most common scenarios and transformations.

It shoud help you streamline the migration process and serve as a handy guide during the transition.

1. Set up Koin

Add Koin framework to your build.grade.kts file dependency section :

Kotlin v

```kotlin
// Koin
implementation("io.insert-koin:koin-android:3.5.6")
```

2. Set up Koin Annotations

Add KSP plugin (or follow official setup) :

Kotlin v

```kotlin
plugins {
  id("com.google.devtools.ksp") version "1.9.23-1.0.20"
}
```

See also official doc : Kotlin Help KSP with Kotlin Multiplatform | Kotlin.

Add Gradle dependencies :

Kotlin v

```kotlin
// Koin Annotations
implementation("io.insert-koin:koin-annotations:1.3.1")
ksp("io.insert-koin:koin-ksp-compiler:1.3.1")
```

Don't forget to set up your source sets to see KSP-generated code :

Kotlin v

```kotlin
android {
  applicationVariants.all {
      val variantName = name
      sourceSets {
        getByName("main") {
          java.srcDir(File("build/generated/ksp/$variantName/kotlin"))
        }
      }
    }
}
```

If you want to activate compile safety in Koin, it's done via KSP configuration and `KOIN_CONFIG_CHECK` property :

Kotlin v

```kotlin
ksp {
  arg("KOIN_CONFIG_CHECK", "true")
}
```

3. Replacing Hilt Annotations with Koin

Hilt Annotation	-->	Koin Annotation
@HiltAndroidApp		No direct equivalent - use startKoin in Application class
@AndroidEntryPoint		Not needed in Koin
@Module		@Module
@Installin		Not needed in Koin
@Provides		@Factory (or @Single if you need a singleton)
@Singleton		@Single
@HiltViewModel		@KoinViewModel

Example Module Definition :

Kotlin v

```kotlin
@Module
class AppModule {

  @Single
  fun provideRepository(api: Api): Repository = RepositoryImpl(api)
}
```

The same can be achieved with @ComponentScan on a module class, and an annotated class component :

```kotlin
@Single
class RepositoryImpl(val API : Api) : Repository

@Module
@ComponentScan("com.my.package")
class AppModule
```

@ComponentScan is targeting a package name, or the current package if no name.

Module class in Koin can include another modules with `includes` property :

```kotlin
@Module
class DataModule

@Module(includes = [DataModule::class])
class AppModule
```

4. Initializing Koin

Replace Hilt initialization with `startKoin` in your Application class :

```kotlin
class MyApplication : Application() {

  override fun onCreate() {
    super.onCreate()
    startKoin {
      androidContext(this@MyApplication)
      modules(AppModule().module)
    }
  }
}
```

To load Koin Annotations modules classes, you need to use the generated `.module` extension on your module class. This contains all the needed Koin DSL configuration, generated for you.

5. Injecting Dependencies

Hilt	-->	Koin
@Inject lateinit var repository: Repository		val repository: Repository by inject0
@Inject constructor(...)		constructor(...) // No annotation needed

6. ViewModel Injection

Hilt

```kotlin
@HiltViewModel
class MyViewModel @Inject constructor(private val repository: Repository) :
ViewModel()
```

koin

```kotlin
@KoinViewModel
class MyViewModel(private val repository: Repository) :
ViewModel()
```

7. Composable Injection

Hilt

```kotlin
@Composable
fun MyComposable(viewModel: MyViewModel = hiltViewModel())
```

koin

```kotlin
@Composable
fun MyComposable(viewModel: MyViewModel = KoinViewModel())
```

Any other component than ViewModel can be injected with `koinInject()` :

```kotlin
@Composable
fun MyComposable(myComponent: MyComponent = KoinInject())
```

Injection with `koinInject` function allows your injected dependency to follow your Composable lifecycle. The attached instance will be dropped only when the composable is dropped too.

8. Migrating Modules

1. **Replace** @Module and @InstallIn with @Module from Koin
2. **Change** @Provides methods to @Single or @Factory
3. **Remove** @Inject from constructors
4. **Update** module declarations in `startKoin`.
5. **Bridge** components between Hilt and Koin- set up both DI frameworks to coexist temporarily, enabling you to migrate parts of your codebase incrementally.

9. Testing

Replace Hilt test rules with Koin test API :

```kotlin
class MyTest : KoinTest {

  @Before
  fun setup() {
    startKoin { modules(testModule) }
  }

  @After
    fun tearDown() {
      stopKoin()
  }
}
```

10. Compose Previews

Use KoinApplication for previews:

```kotlin
@Preview
@Composable
fun MyPreviewComposable() {
  KoinApplication(application = { modules(previewModule) }) {
    MyComposable()
  }
}
```

Migration tips

- Start with smaller, less dependent modules.
- Update one component at a time (e.g., repositories, then ViewModels).
- Leverage Koin Annotations for easier migration from Hilt.
- Update tests alongside component migrations.
- Verify dependency graph after migration.

Remember to consult Koin's official documentation for more detailed information and advanced usage.