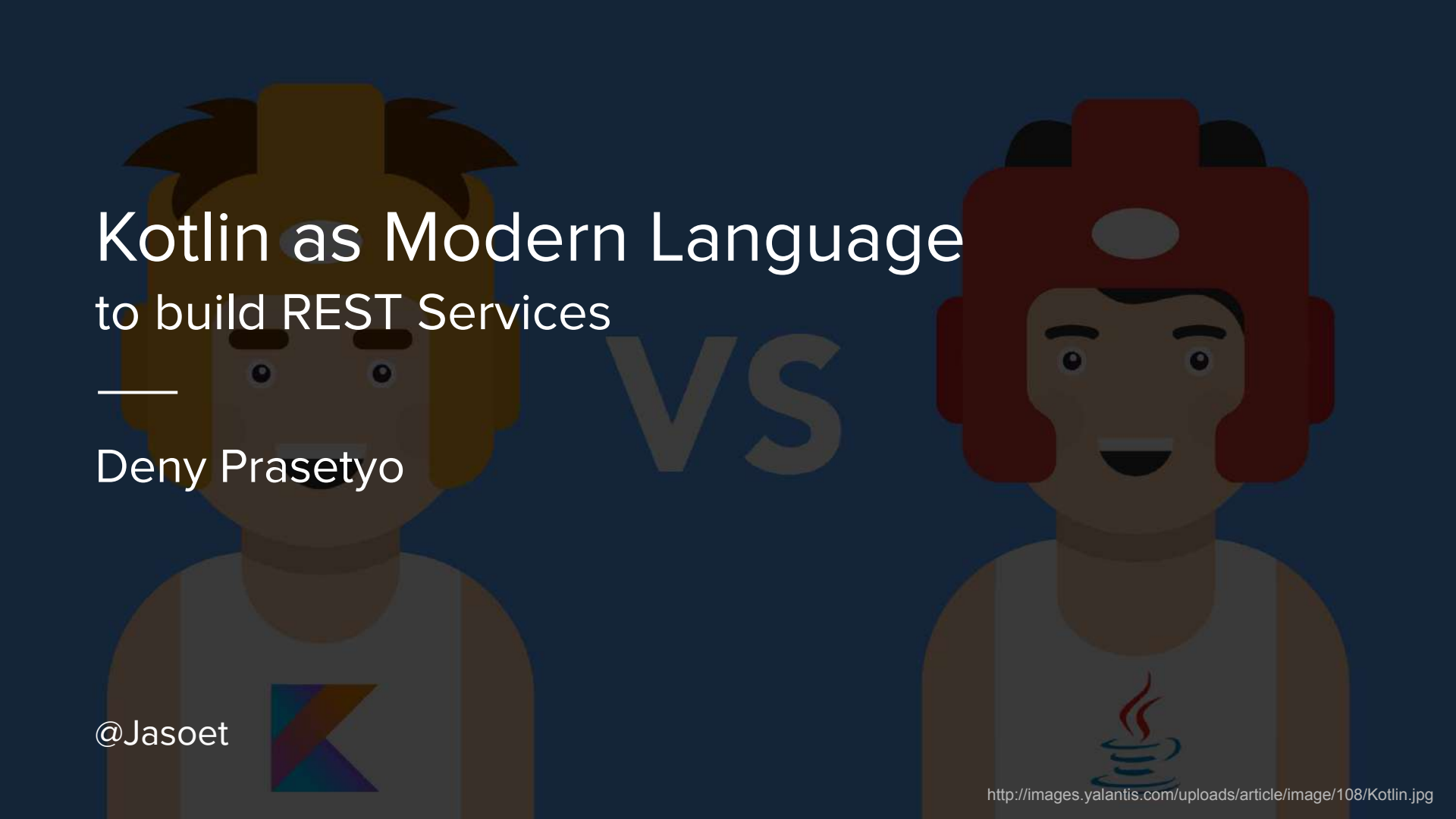
Kotlin as Modern Language

to build REST Services



Deny Prasetyo

@Jasoet

http://images.yalantis.com/uploads/article/image/108/Kotlin.jpg

@Jasoet

> **Deny Prasetyo**

- 9+ years experience as a developer, instructor, and community organizer
- Kotlin Language and Cloud Native Technology Enthusiast
- Senior Software and Systems Engineer, Gopay Indonesia
- Kotlin Indonesia and Cloud Native Indonesia Community

What is Kotlin?

- ➔ Programming **Language**
- ➔ Multi-Platform: **JVM**, **Android**, **JavaScript**, and LLVM (**Native** for iOS, Linux, Mac)
- ➔ 100% **Java Compatible (JVM** and **Android)**
- ➔ Android **Official Language**
- ➔ **Transpile** to JavaScript
- ➔ Compiled to Native using LLVM Compiler Infrastructure
- ➔ By **JetBrains** (Creator of **Android Studio** and **Intellij IDEA**)
- ➔ **Framework** and **Tools** Friendly (Use existing Frameworks/Tools)

What is Not?

- ➔ **Not** a **Java Killer!**
- ➔ **Not** a new **Platform**!

Just a **Language!**

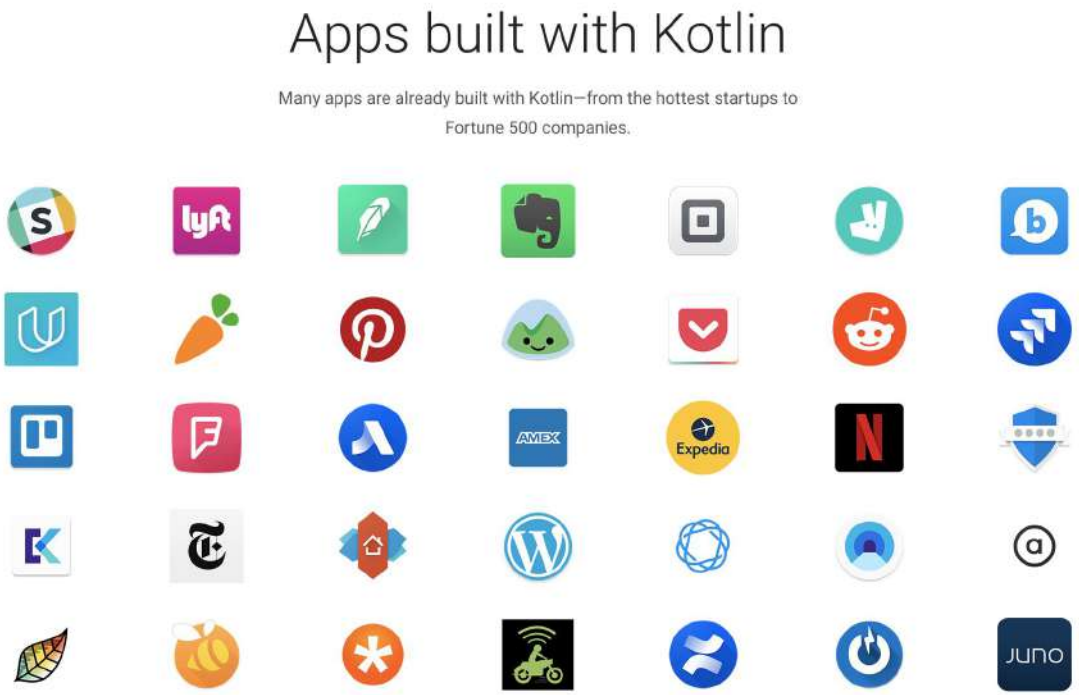
Ecosystem

- ➔ Use **existing framework/tools** (Java, Android, Node.js, LLVM)
- ➔ **Kotlin Indonesia** => 13K FB Group and 8K Telegram Group Member
- ➔ (30000+) Kotlin lang Slack users (**https://kotlin.link**)

Learning Curve

- ➔ Statically-typed. Object-oriented/Functional Paradigm.
- ➔ Kotlin and Java **working side-by-side** in single project.
- ➔ Kotlin allowing us to **write concise** and **expressive code** while maintaining full compatibility with **existing Java-based technology stacks** and a **smooth learning curve.**

Success Story



What Kotlin offers!

- **Null Safety**
  - **Null are part of type system**
- **Mutability Protection**
  - **Mutability notation is a requirement.**
  - You must choose val or var.
- The Magical Fun
  - Normal Function
  - Higher order function and Lambda
  - Inline fun
  - Function Literal with Receiver
  - **Extension Function (Kotlin's white Magic)**
- Class and Object
  - Class
  - Inheritance
  - **Data Class**
  - Object Notation (the real form)
  - Companion Object
  - Sealed Class
  - **Destructuring**
  - Enum Class
- Safety Belt
  - `is` operator
  - **Smart Cast**
  - **Unchecked Exception** (No more "Exception Driven Programming")
  - Type Alias
- **Coroutine**

Null Safety

```
var nonNullType: Type
nonNullType = null // compilation error

var nullableType: Type?
nullableType = null // ok
```

Working with Null

- Safe accessor: [?]
- Elvis operator: [?:]

```
// if messageFromServer is null, we'll display a default msg
val toDisplay: String = messageFromServer ?: defaultMessage

// cascading null safety
val maybeNullThing: Thing? = /* something */
val maybeName: String? = maybeNullThing?.someProperty?.name

// safe scoping
maybeNullThing?.let {
  // code will only be executed if not null
}
```

Mutability Protection

- ➔ You must choose `val` or `var`.
- ➔ “`val a: Int`” = “`final int a`”
- ➔ The compiler is your friend!
- ➔ Use `var` for something that will vary with time.
- ➔ Use `val` for a value that won’t change.
- ➔ Always default to `val` until something needs to be made into `var`.

Using “final” / “val” :

- clearly communicates your intent
- allows the compiler and virtual machine to perform minor optimizations
- clearly flags items which are simpler in behaviour - final says, *"If you are looking for complexity, you won't find it here."*

Extension Function

```
fun <T> MutableList<T>.swap(index1: Int, index2: Int) {
    val tmp = this[index1] // 'this' corresponds to the list
    this[index1] = this[index2]
    this[index2] = tmp
}


val l = mutableListOf(1, 2, 3)
l.swap(0, 2) // 'this' inside 'swap()' will hold the value of 'l'
```

Data Class and Destructuring

```
data class User(val name: String, val age: Int)
```

— `equals()` / `hashCode()` pair,

— `toString()` of the form `"User(name=John, age=42)"`,

— componentN() functions corresponding to the properties in their order of declaration,

— `copy()` function (see below).

```
val jane = User("Jane", 35)
val (name, age) = jane
println("$name, $age years of age") // prints "Jane, 35 years of age"
```

Static Typing + Smart Cast

Java

```java
View childView = this.getChildAt(0);
if (childView instanceof ImageView) {
    ((ImageView) childView).setImageAlpha(255);
}
```

Kotlin

```kotlin
val view = this.getChildAt(0)
if (view is ImageView) {
    view.imageAlpha = 255
}
```

Coroutine

In the beginning

There are Process and Thread

- Processes are often seen as synonymous with programs or applications.
- Single application may in fact be a set of cooperating processes.
- A process has a self-contained execution environment. A process generally has a complete, private set of basic run-time resources; in particular, each process has its own memory space.
- Threads are sometimes called *lightweight processes*.
- Creating new thread requires fewer resources than creating a new process.
- Threads exist within a process — every process has at least one.
- Threads share the process's resources, including memory and open files.
- This makes for efficient, but potentially problematic, communication.

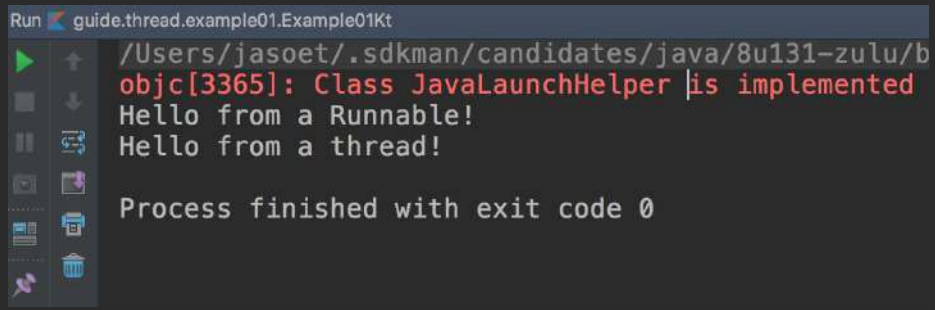
Process Builder

```
jasoet at Go-Soet in ~/Documents/GoCommunity/coroutine-guide-example
± cat /Users/jasoet/output.out
total 2538776
drwxr-xr-x+  74 jasoet  staff   2.5K Jul 27 10:22 .
drwxr-xr-x    6 root    admin   204B Jul  4 16:31 ..
drwxr-xr-x    3 jasoet  staff   102B Jul 13 07:17 .3T
-r--------    1 jasoet  staff     7B Jul  4 16:29 .CFUserTextEncoding
-rw-r--r--@   1 jasoet  staff    14K Jul 26 13:55 .DS_Store
drwxr-xr-x   25 jasoet  staff   850B Jul 23 13:30 .SpaceVim
drwxr-xr-x    3 jasoet  staff   102B Jul 23 13:30 .SpaceVim.d
drwx------    2 jasoet  staff    68B Jul 27 08:51 .Trash
drwxr-xr-x   19 jasoet  staff   646B Jul 22 09:26 .android
-rw-r--r--    1 jasoet  staff   7.2M Jul 22 09:30 .babel.json
-rw-------    1 jasoet  staff   272B Jul 11 09:17 .bash_history
-rw-r--r--    1 jasoet  staff   650B Jul 26 21:27 .bash_profile
drwx------   13 jasoet  staff   442B Jul  5 00:27 .bash_sessions
-rw-r--r--    1 jasoet  staff   218B Jul  5 00:18 .bashrc
drwxr-xr-x   10 jasoet  staff   340B Jul 26 21:23 .cache
drwx------    9 jasoet  staff   306B Jul 23 13:30 .config
drwx------    3 jasoet  staff   102B Jul 26 14:11 .cups
drwxr-xr-x    5 jasoet  staff   170B Jul 23 13:30 .data
-rw-------    1 jasoet  staff    16B Jul 21 00:23 .emulator_console_auth
-rw-r--r--    1 jasoet  staff   487B Jul 15 00:47 .gitconfig
-rw-r--r--    1 jasoet  staff    13B Jun 19 13:43 .gitignore_global
drwxr-xr-x    7 jasoet  staff   238B Jul 14 13:34 .gitkraken
drwxr-xr-x    3 jasoet  staff   102B Jul 11 00:23 .glide
drwxr-xr-x    6 jasoet  staff   204B Jul  5 18:05 .gradle
-rw-r--r--    1 jasoet  staff    27B Jun 19 13:43 .hgignore_global
drwxr-xr-x    4 jasoet  staff   136B Jul 17 01:24 .ideaLibSources
drwxr-xr-x    7 jasoet  staff   238B Jul 14 13:43 .ivy2
drwx------    3 jasoet  staff   102B Jul  5 01:00 .local
drwxr-xr-x    3 jasoet  staff   102B Jul 15 23:46 .m2
```

```
fun main(args: Array<String>) {
    val command = listOf("ls", "-alh", "/Users/jasoet/")
    val workingDirectory = File("/Users/jasoet/")
    val output = File("/Users/jasoet/output.out")
    val processBuilder = ProcessBuilder(command).apply {
        directory(workingDirectory)
        redirectOutput(output)
    }

    processBuilder.start().waitFor()
}
```

Creating Thread




```
class HelloRunnable : Runnable {
    override fun run() {
        println("Hello from a Runnable!")
    }
}

class HelloThread : Thread() {
    override fun run() {
        println("Hello from a thread!")
    }
}

fun main(args: Array<String>) {
    Thread(HelloRunnable()).start()
    HelloThread().start()
}
```

Thread Pool

- **Java Thread pool** represents a group of worker threads that are waiting for the job and reuse many times.
- **Better performance** It saves time because there is no need to create new thread.
- **Real Time Usage** It is used in Servlet, JSP and Spark Java where container creates a thread pool to process the request.

➔ `Executors.newFixedThreadPool(5)`
➔ `Executors.newCachedThreadPool()`
➔ `Executors.newSingleThreadExecutor()`

Using Thread Pool

```
fun main(args: Array<String>) {
    val executor = Executors.newFixedThreadPool( nThreads: 5)
    (1..20).forEachIndexed { i, _ ->
        executor.execute {
            println("Run Worker $i from Thread Pool")
        }
    }
    executor.shutdown()
    while (!executor.isTerminated) {
    }

    System.out.println("Finished all threads")
}
```

Modern Solutions

Future / Promise

- **A Future** represents the pending result of an asynchronous computation.
- It offers a method — `get` — that returns the result of the computation when it's done.
- Can be chained using `CompletableFuture` class.
- **Promise** is similar with **Future**.
- **Kovenant** is **Promise Library** for Kotlin http://kovenant.komponents.nl/

Using Future

```
fun main(args: Array<String>) {
    CompletableFuture
            .supplyAsync {
                println("Some fake heavy load Data!")
                listOf("Data1", "Data2", "Data3", "etc")
            }
            .exceptionally { throwable ->
                when (throwable) {
                    is IllegalStateException -> listOf("Default Data 2")
                    is Exception -> listOf("Default Data 1")
                    else -> emptyList()
                }
            }
            .thenAccept { data ->
                data.forEach {
                    println(it)
                }
            }
            .get()
}
```

Using Promise

```
fun main(args: Array<String>) {
    task {
        //some (long running) operation, or just:
        1 + 1
    } success {
        //called when no exceptions have occurred
        println("result: $it")
    } fail {
        //called when an exceptions has occurred
        println("that's weird ${it.message}")
    } always {
        //no matter what result we get, this is always called once.
    }
}
```

Rx Java

- **Rx** a library for composing asynchronous and event-based programs by using observable sequences.
- To use RxJava you create **Observables** (which emit data items), **transform** those Observables in various ways to get the precise data items that interest you (by using Observable operators), and then **observe** and **react** to these sequences of interesting items (by implementing Observers or Subscribers and then subscribing them to the resulting transformed Observables).

Rx Java Example

```
Observable.from(contacts)
        .compose(new NewThreadTransformer<Contact>())
        .flatMap(new Func1<Contact, Observable<String>>() {
            @Override
            public Observable<String> call(Contact contact) {
                return Observable.just(contact.username);
            }
        })
        .filter(new Func1<String, Boolean>() {
            @Override
            public Boolean call(String number) {
                return ChatUtil.isContactValidIndonesianNumber(number);
            }
        })
        .buffer(CONTACT_BATCH)
        .subscribe(new Subscriber<List<String>>() {
            @Override
            public void onCompleted() {
                Log.d(TAG, "sync contact bulk complete");
                localPreferences.contactSyncOnFinish();
            }

            @Override
            public void onError(Throwable e) {
                Log.e(TAG, "sync contact bulk error, "+e.toString());
                localPreferences.contactSyncOnFinish();
            }

            @Override
            public void onNext(List<String> batchContacts) {
                Chat mSmackChat = ChatManager.getInstanceFor(connection).createChat(
                        XmppAddress.HTTP_GATEWAY, mMessageListener);
```

The Problem

```
fun loadSomeData():Promise<List<String>,Throwable>{
    return task {
        println("Load some important data")
        listOf("Important Data 1","Important Data 2›")
    }
}

fun loadFutureData(): Future<List<String>> {
    return CompletableFuture.supplyAsync {
        println("Load some Future Data")
        listOf("Future Data 1", "Future Data 2")
    }
}
```

The Callback

```
api.getAppNotification()
        .subscribe(new Action1<AppNotificationData>() {
            @Override
            public void call(AppNotificationData appNotificationData) {
                if (success) {
                    api.getAppSettings().subscribe(new Action1<AppProfile>() {
                        @Override
                        public void call(AppProfile appProfile) {

                            // do something for api 2
                        }
                    });

                    //do something for api 1 success
                } else {
                    // do something for api 1 fail
                }
            }
        });
```

Kotlin Coroutine

Using Kotlin Coroutine

```
fun postItem(item: Item) {
    launch(CommonPool) {
        val token = preparePost()
        val post = submitPost(token, item)
        processPost(post)
    }
}
```

explicit coroutine context

suspending function

natural signature

```
suspend fun preparePost(): Token {
    // makes request & suspends coroutine
    return suspendCoroutine { /* ... */ }
}
```

Features

- Regular Loops

```
for((token,item) in list) submitPost(token,item)
```

- Regular Exception Handling

```
try { submitPost(tokenI, itemI) }
catch (e: Exception) { /*...*/}
```

- Regular High Level Operator
  - `let, apply, forEach, filter, map, use, etc`

```
Everything Like Blocking Code!!
```

Library vs Language

- https://kotlinlang.org/docs/reference/coroutines-overview.html
- Kotlin language only has `suspend` keyword.
  - Transform suspend function to callback.
  - Compiles code to State Machine.
  - Stdlib has Continuation and `CoroutineContext`.
- Everything else is in library.
  - It includes `launch/join`, `async/await`, `runBlocking`, `etc`
  - We were using **kotlinx.coroutines** library.
  - https://github.com/Kotlin/kotlinx.coroutines

Blocking threads

Callbacks

Promises/Futures/Rx

Kotlin coroutines

REST Services with Kotlin

Options

- Spring Boot (https://start.spring.io/)
- Eclipse Vertx (https://vertx.io/)
- JetBrains Ktor (https://ktor.io)
- Quarkus (https://quarkus.io/)
- Micronaut (https://micronaut.io/)

Talk is cheap. Show me the code.

— *Linus Torvalds* —

Ktor

- Lightweight
- Asynchronous with Coroutine
- Developed by JetBrains
- 100% Kotlin Awesomeness
- Collection of Features
- Multiple Engine Support (Jetty, Netty, Tomcat, Coroutine IO)

https://github.com/jasoet

Thank You