Automate your workflows with Kotlin

Fosdem - 2020
Automate your workflows with Kotlin

@martinbonnin

@mgauzins

dailymotion
A daily work...

1. Assign a ticket
2. Create a branch
3. Code...
4. Create a pull request
5. Move ticket state
6. Merge pull request
7. Move ticket state
8. Create an alpha
   a. Increment a version
   b. Tag
   c. Push
9. Send a message to designers/product owners
10. Integrate feedbacks
11. Back to step 1

But also...

- Manage app translations
- Keep the store app up to date (images, listings, archives)
- Manage app rollout
- Notify about the updates
- Publish to alternative stores
Environment

- Github
- Appcenter
- Slack
- Transifex
- PlayStore
- CI
- Jira
Why automating?

- It takes times.
- Reliability
- Reproducibility
- Documentation
- Fun
- Kotlin to the rescue

"I spend a lot of time on this task. I should write a program automating it!"

Theory:
- Writing code
- Work on original task
- Automation takes over
- Free time

Reality:
- Writing code
- Debugging
- Rethinking
- No time for original task anymore

https://xkcd.com/1319/
Why Kotlin?

- The language we use every day
  - No context change (bash syndrome)
  - Every team member knows it

- Modern

- Great IDE

- Great Ecosystem
### What did we replace?

<table>
<thead>
<tr>
<th>Ad-hoc scripts</th>
<th>General purpose tools</th>
<th>3rd party tools</th>
<th>Build system</th>
</tr>
</thead>
<tbody>
<tr>
<td>generate_docs.sh (Bash)</td>
<td>Fastlane (Ruby)</td>
<td>Github hub (Go)</td>
<td>build.gradle (Groovy)</td>
</tr>
<tr>
<td>after_success.sh (Bash)</td>
<td></td>
<td>Transifex cli (Ruby)</td>
<td></td>
</tr>
</tbody>
</table>
Tools we used

- Kotlin scripts
  - Based on Kscript

- Kotlin command line app (cli)
  - Called Kinta
  - Based on Clikt
Kscript
Scripting - motivations

- For short projects/single file projects
- No need for gradle
- Easy to setup/use
Scripting - simple example

// hello.kts
println("Hello \${args[0]}!")

// running the script
$ kotlinc -script hello.kts Fosdem
Hello Fosdem!

https://github.com/Kotlin/KEEP/blob/master/proposals/scripting-support.md
Kscript – scripting improvements

- Compiled script caching
- Shebang and interpreter usage
- Dependencies
- Standalone binaries
- IDE support
- https://github.com/holgerbrandl/kscript
Kscript - installation

curl -s "https://get.sdkman.io" | bash
source ~/.bash_profile  # install sdkman

# add sdkman to PATH

sdk install kotlin
sdk install kscript

# install Kotlin
# install Kscript

touch hello.kts
kscript --idea hello.kts

# start the IDE
// weekend.kts
#!/usr/bin/env kscript
@file:DependsOn("com.squareup.okhttp3:okhttp:4.3.1")

import okhttp3.OkHttpClient
import okhttp3.Request

val weekend = Request.Builder().get().url("http://isitweekendyet.com/").build().let {
  OkHttpClient().newCall(it)
}.execute().body!!.string().toLowerCase().contains("yes")

if (weekend) {
  println("It is the weekend!")
} else {
  println("Not yet :-|")
}
$ kscript --idea weekend.kts
Kscript - Debugger
Kscript - Real life examples

- Generating project website (mkdocs + github pages)
- Install scripts
- Migration from build.gradle to build.gradle.kts
- Finding duplicates in string.xml
- Categorizing my expenses!
- etc...
Scripting - limitations

- JVM required
- JVM startup time
- Multiple files is hard to maintain
- No Gradle => no plugins, kapt, etc...
Kinta: a Kotlin Cli
CLIKT : Presentation

- Open Source library
- Command Line Interface for Kotlin
  [https://github.com/ajalt/clikt](https://github.com/ajalt/clikt)

Help

- Commands: customisable, nested commands
- Argument/Option: composable, type safe, prompt, default
CLIKT to Kinta

- We need entry points for workflows then commands
- Provide a simple way to launch these commands by anyone. (Command line interface)
- Reach even more platforms
Kinta CLI integration

- apply plugin: 'application'
- Create a jar
- Specify the ‘Main’ class
- Generate starting scripts

```kotlin
plugins {
    application
}

tasks.withType<Jar> {
    archiveFileName.set("kinta-cli.jar")
    manifest {
        attributes("Main-Class" to com.dailymotion.kinta.MainKt")
    }
    from(configurations.runtimeClasspath.get().map {
        if (it.isDirectory) it else zipTree(it)
    })
}

application {
    mainClassName = "com.dailymotion.kinta.MainKt"
}
```

build.gradle.kts
PublishPlayStore workflow

- What is a workflow?
- Workflow detail
  - Upload archive
  - Create a release on a specific track
  - Find a local changelog for the version
  - Upload the changelog

```
kinta publish beta --archiveFile=app-release.aab
```
PublishPlayStore workflow

```kotlin
override fun run() {

object PublishPlayStore : CliktCommand(
    name = "publish",
    archiveFile = File(archiveFile),
    help = "Publish a version on the given track") {

    private val track by argument("--track", help = "The Play Store track")
    PlayStoreIntegration.uploadDraft(
        track = track
    )

    private val archiveFile by argument("--archiveFile")
    PlayStoreIntegration.createRelease(
        track = track
        listVersionCodes = listOf(versionCode),
        percent = percentToApply
    )

    private val percent by option("--percent", help = "The user fraction receiving the update").double()

    override fun run() {
        val changeLogs = LocalMetadataHelper.getChangelog(versionCode)

        // Beautiful code is coming...
        PlayStoreIntegration.uploadWhatsNew(
            versionCode = versionCode,
            whatsNewProvider = changeLogs
        )
    }
}
```
A Swiss knife

Git - tickets
- startWork
- PR

Translations
- txPull
- txPush
- txPR

Builds
- nightly
- buildPR
- buildTag

Common tools
- trigger
- branch
- hotfix
- cleanLocal
- cleanRemote

Play Store metadatas
- uploadWhatsNew
- uploadListing
- uploadScreenshots
- generateScreenshots

Play Store releases
- beta
- release
The daily work becomes simpler!

1. Assign a ticket
2. Create a branch

3. Code

4. Create a pull request
5. Move ticket state

6. Merge pull request

7. Move ticket state
8. Create an alpha
   Increment version
   Tag
   Push
9. Send a message

- kinta startWork {TICKET_ID}
- (Sorry you definitely have to write code)
- kinta pr
- Keep the validation on GitHub interface
- kinta nightly
What’s next
Kinta - customization

- Make the kinta tool usable outside Dailymotion
- 3rd party services have a well defined API...
- ... but every organization has their own processes and workflows.
- There's a fine line between customization and reuse
An Integration is:

- A Kotlin class linked to a specific domain:
  - Github
  - Transifex
  - etc...
- Highly reusable
- Redistributed
- Static
  - It doesn’t change often
- Composed of atomic methods
- Documented using Kdoc
- Inside the redistributed `kinta-integrations` artifact

A Workflow is:

- A Clikt command for a specific complex task:
  - Publish a release
  - Create a Translation PR
  - etc...
- Inside the host project
- Most of the times specific to the host project
- Uses integrations to accomplish complex tasks
- Documented using clikt
Custom workflows are built using gradle and loaded at runtime using a ServiceLocator.

Kinta also comes with default built in workflows:
- Publishing to the play store
- Opening a pull request
- etc...

- kinta-integrations.jar
- custom-workflows.jar
- builtin-workflows.jar
What’s next

- Figuring out a way to distribute the kinta binary
- Also distribute the backend/webapp that hosts artifacts
- https://github.com/dailymotion/kinta
- Feedbacks welcome
- Disclaimer: it’s still very early stage and things may break
Thanks.