Android CI At Scale - How Square Makes It Work

Paul Hundal, Inez Korczyński
DPE Summit 2023
Our Team

Paul Hundal  Senior Software Engineer

Inez Korczyński  Senior Software Engineer
Agenda

1. General Stats
2. Build Pipeline Composition
3. Git S3 strategy
4. UI Test Avoidance
5. Shard Avoidance
6. Results
7. Next Steps
General Stats
Overview & numbers

Square Go
Discover your go-to's

Photo Studio by Square
eCommerce Product Photography

Square Invoices:
Invoice Maker
Invoicing, Estimates, Bills

Square Team
Square Team App

Square Payroll
Payroll App

Square: Retail Point of Sale
Payment, Inventory Management

Square Appointments
Booking, Scheduling, Payments

Square - Dashboard for POS
Simple, Powerful POS Analytics
Some Stats

- 200 Android developers
- 11 apps in Google Play Store
- 300 demo/development apps
- 5,400 Android modules
Composition of our Build Pipeline
Types Of Jobs

- checks
  - e.g. check-ktlint, check-unused-dependencies
- builds
  - e.g. pos-assemble-release, pos-assemble-android-test, login-screen-assemble-debug,
- unit tests
  - e.g. pos-unit-tests
- ui-tests
  - e.g. pos-ui-mobile, point-of-sale-ui-tablet
- publish
  - pos-sign-and-upload
- Pathfinder
  - Application build shard 1
    - UI tests app 1
  - Test build shard 1
  - Application build shard 2
    - UI tests app 2
  - Test build shard 2
  - Application build shard 3
    - UI tests app 3
    - Test build shard 3
  - Checks (ktilnt)
  - Results collector
    - Publish application 1
    - Publish application 2
    - Publish application 3
Git / S3 Strategy
● Git snapshot
  ○ Created daily
  ○ Shallow clone (depth=50)
● Git bundle
  ○ Created for each SHA
  ○ Differential (snapshot => SHA)
UI Test Avoidance
• 14,000 UI tests
• 500 CI UI tests jobs

pos-mobile-ui
3,100 tests & 120 emulators

login-screen-demo-mobile-ui
5 tests & 1 emulator
# As it turns out in Android compilation with multidexing sometimes some classes will end up in
# for instance smali_classes2 and other time in smali_classes5 (it is not deterministic),
# however it does not have impact on how the application work. To work-around that we are moving
# all files/directories into common (`__smali_classes__`) directory first.
# Note: `smali_classesX` start from 2
FileUtils.mkdir("#{dir}/__smali_classes__/")
i = 2
while true
  if File.directory?("#{dir}/smali_classes#{i}")
    `cp -r #{dir}/smali_classes#{i}/. #{dir}/__smali_classes__/`
    raise "Copying smali_classes* to __smali_classes__ failed" unless $?..success?
    FileUtils.rm_rf("#{dir}/smali_classes#{i}")
    i = i + 1
  else
    break
  end
end
if version_name != nil && File.file?("#{dir}/__smalli_classes__/com/squareup/android/util/RealPosBuild.smali")
  `sed -i#{SED_BACKUP_EXTENSION_SUFFIX} '/"#{version_name_without_postfix}"/d' #{dir}/__smalli_classes__/com/squareup/android/util/RealPosBuild.smali`
  raise "Failed to remove version_name_without_postfix" unless $??.success?
  `sed -i#{SED_BACKUP_EXTENSION_SUFFIX} '/"#{version_name_with_postfix}"/d' #{dir}/__smalli_classes__/com/squareup/android/util/RealPosBuild.smali`
  raise "Failed to remove version_name_with_postfix" unless $??.success?
end

if version_code_hex != nil && File.file?("#{dir}/__smalli_classes__/com/squareup/android/util/RealPosBuild.smali")
  `sed -i#{SED_BACKUP_EXTENSION_SUFFIX} '/0x#{version_code_hex}/d' #{dir}/__smalli_classes__/com/squareup/android/util/RealPosBuild.smali`
  raise "Failed to remove version_code_hex" unless $??.success?
end
Results - hit ratios

98%  Demo/Development applications

25%  Large applications
Shard Avoidance
Shard Avoidance Benefits

- Faster Builds
- Less potentially flakey shards to run
- Reduced worker queue
- Faster developer iterations
Shard Avoidance In Practice

- Compare Git SHA’s
- Analyze modified files
- Map to Gradle Modules
- Find minimum set of CI shards to run

```kotlin
val shardsToDocsDeferred = getShardsToDocsDeferred(analysisResultDeferred)

val docsToShardsDeferred = getDocsToShardsDeferred(shardsToDocsDeferred)

// Gathers all shards that are not mapped to docs
val unmappedShardsDeferred = async(Dispatchers.Default) {
    options.kochiku.targets.map { it.type }.toSet() - shardsToDocsDeferred.await().keys
}

// Run the global file check while spinning up other coroutines
val noGlobalFilesDeferred = async {
    ensureNoGlobalFiles(analysisResultDeferred, unmappedShardsDeferred)
}

// Performs the analysis that produces the shard skipping as well as logging to ES2
val analysisDeferred = getAnalysisDeferred(analysisResultDeferred, docsToShardsDeferred)

// Check for the global files changed on the first await. This gives all jobs a chance to run.
LOGGER.info("Checking global file changes")
val globalFilesCheckResult = noGlobalFilesDeferred.await()

if (globalFilesCheckResult.isNotEmpty()) {
    if (options.kochikuPipelineOnly) {
        LOGGER.warn("Global files changes detected. Not skipping any shards.")
        writeKochikuPipelineToFile(emptySet())
        return@coroutineScope
    } else {
        throw GlobalFilesFoundException(globalFilesCheckResult.files)
    }
}

LOGGER.info("Finding affected shards")
val affectedShards = findAffectedShards(analysisDeferred.await())

LOGGER.info("Calculating skippable")
val skippable = (shardsToDocsDeferred.await().keys - affectedShards).toSortedSet()
```
Non-Convergence CI Pipeline for Shard Avoidance

- Application build shard 1
- UI tests app 1
- Test build shard 1
- Application build shard 2
- UI tests app 2
- Test build shard 2
- Application build shard 3
- UI tests app 3
- Test build shard 3
- Checks (e.g., lint)
Gradle Tooling API

We analyzed how Gradle interacts with IntelliJ using the Tooling API to extract a custom model representation of our dependency graph.

- Extract the build graph
- Create model representation for analysis
- Inject models into our analyzer
- Determine which shards to skip
Shard Avoidance Results

50% Shards Skipped

360 Time Saved (hrs)
Static Build Analyzer

We tried to bypass Gradle using Groovy’s AST Parsing

- Mimic the model representation of Gradle
- Bypass the Tooling API completely
- Reduce the 3-5 minute runtime of the configuration phase to mere seconds!
- Lower level of correctness
Shard Avoidance Cache

We realized that though expensive, Gradle produce an accurate representation of our build graph. But does it need to be invoked every time?

- Global files modified only 8% of the time
- Reusable build graph
- S3 Key/Value Store

```kotlin
interface AvoidanceCache {

    suspend fun save(     
        key: String,     
        value: ByteArray
    )

    suspend fun get(key: String): ByteArray
}
```
Results

- Avoidance Analysis Time Savings: 63%
- Overall Build Time Reduction: 12%
- Avoidance Cache Hit Rate: 90%
Results
Recap

- **Problem Space**
  - 11 Apps
  - 200 Developers
  - 1,200 Shards
  - 14,000 UI Tests
  - 5,400 Android Modules

- **Solutions**
  - UI Test Avoidance
  - Shard Avoidance
  - Shard Caching
THANKS

paul.hundal@block.xyz
inez@block.xyz