Building, Testing & Deploying

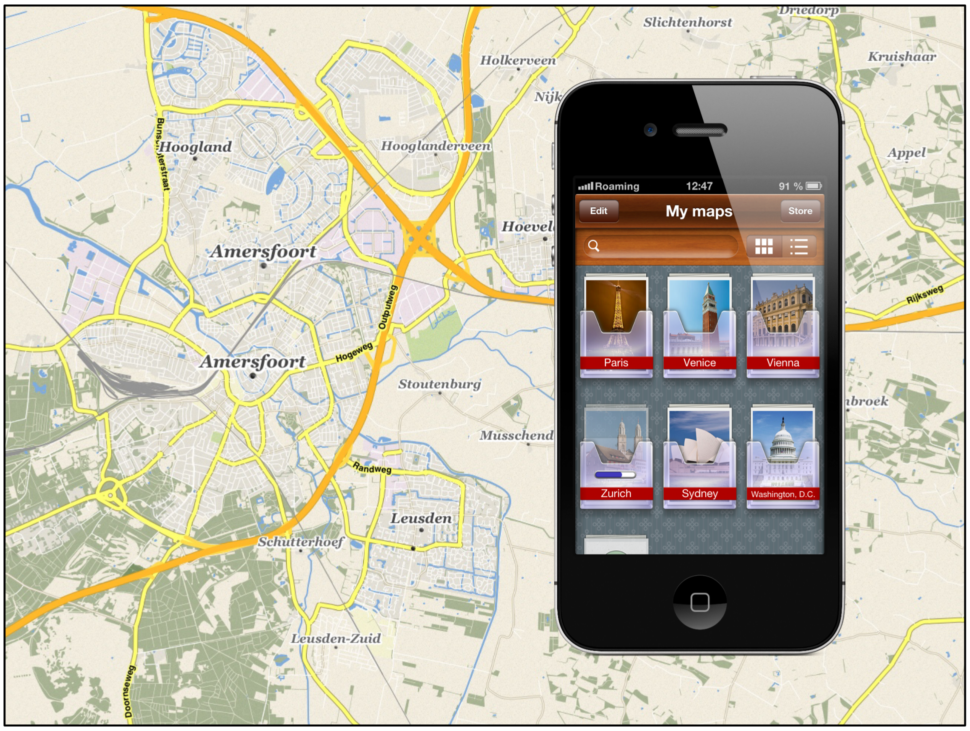
Android Apps with Jenkins

Christopher Orr

First, a quick bit about what my connection is to this topic...

iosphere

I work in Cologne, Germany for a company called iosphere. Who kindly gave me the time to come here today.
We create mobile apps for various customers, and have some of our own products.



Our main app product at the moment is OffMaps for iOS.

This is an offline maps app which uses OpenStreetMap data, which we process and clean-up, add relevant Wikipedia articles for offline use, plus we also have our own datasets making the app useful for tourists.

All of this is displayed using our own vector rendering engine.

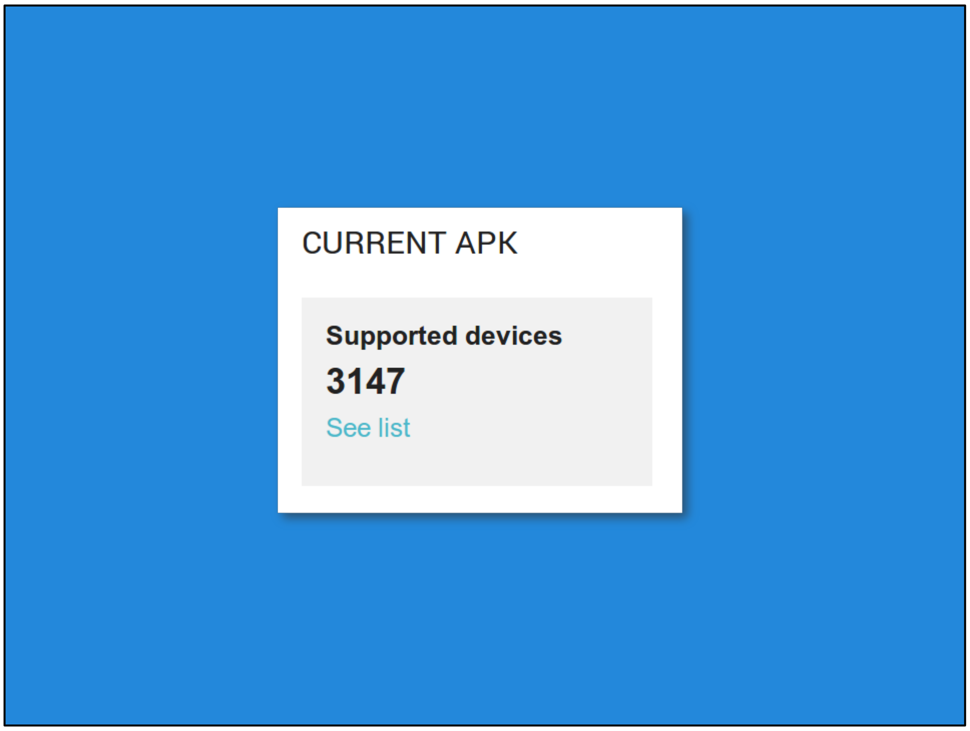
But we also make Android apps. This is what I do, and I've been a full-time Android developer for four years.

We use Jenkins for making maps for OffMaps from OpenStreetMap data, building and testing our apps, plus more. I helped set up a lot of our Jenkins setup, extending plugins and so on.

So, on the subject of Android, there are a couple of problems we face when trying to develop great apps...

The first problem is Android.

People talk about Android "fragmentation", but I think of it more as "diversity" -- it's amazing that Android runs on smartphones, tablets, TVs, games consoles, glasses, vending machines...


CURRENT APK

Supported devices

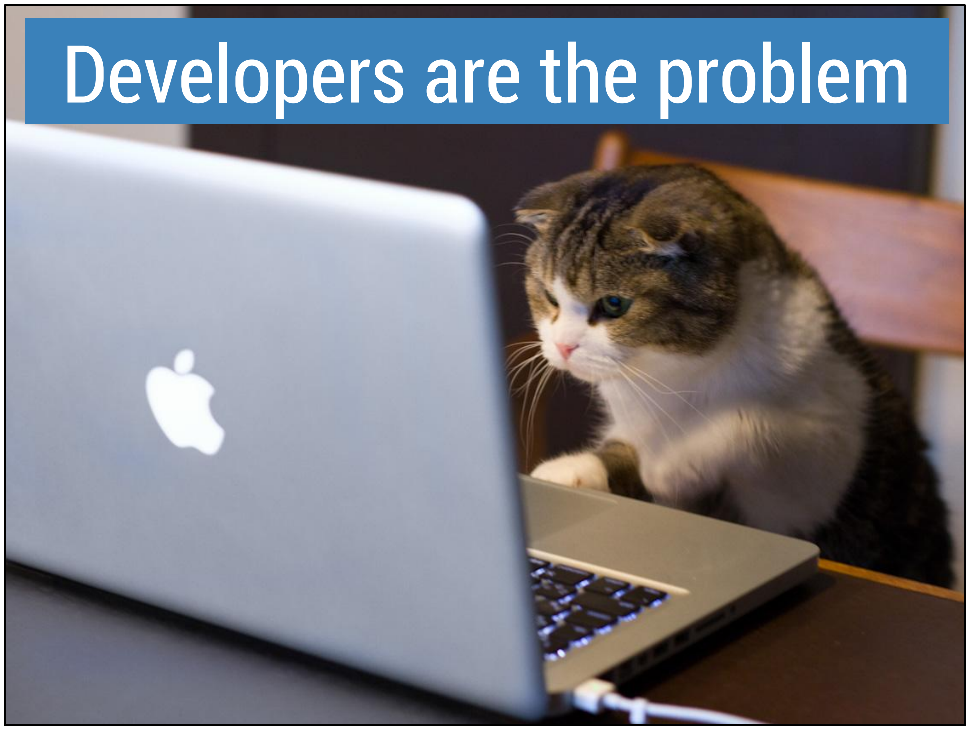
3147

See list

Google Play confirms this diversity for us. Whenever you upload an APK, it shows this ever-increasing number of supported device models.

But the downside is, this diversity is a pain for anyone hoping to provide apps with a consistent user experience.

So Android itself is one problem.



Developers are the problem

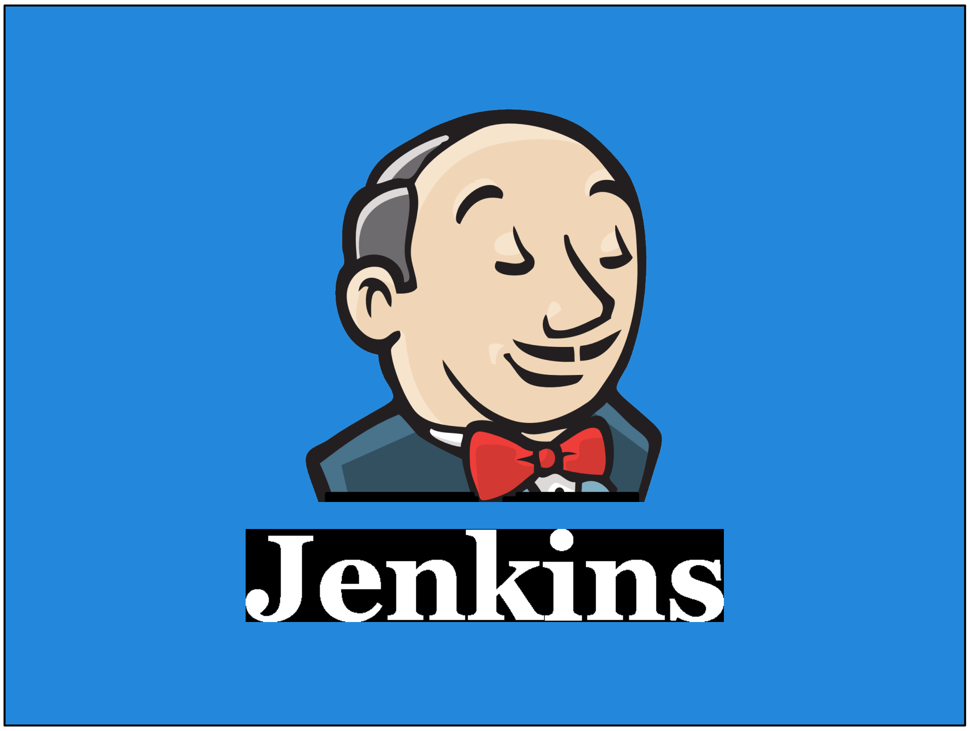
The next problem we encounter during the development process is... developers.

If you're on your own and have no deadline, you can *maybe* write the perfect app that runs perfectly on every device.
As you add people and deadlines, this becomes even less realistic.

Plus with developers you end up with issues with build and build tools:
- "Runs on my machine"
- Building in different ways with different SDK tools
- Compiling and testing with different OS versions and devices

Plus people sometimes just merges a load of crap and someone has to go track it down.

So, how can we try and make life easier for us?



Jenkins

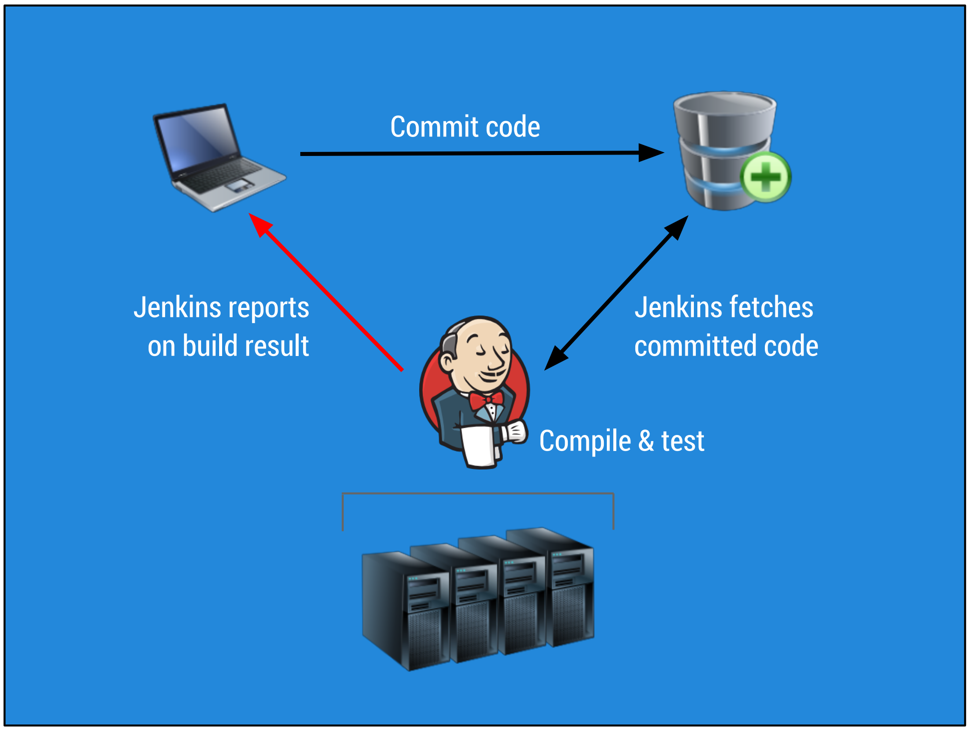
Jenkins is the most popular continuous integration server.

CI means continuously doing small quality checks, like compiling your app and its dependencies and running automated tests (ideally for every code change), so that developers get feedback fast.
By doing this ASAP you're less likely to end up with time-consuming issues later in the development process.

Jenkins is a flexible piece of software that helps you to do this.

It's open source, super-simple to install, actively developed, and can be used to automate just about anything...
There are packages or installers for all commonly-used operating systems and has over 65,000 installs.
And it's used by some big names: eBay, GitHub, Google, NASA, Netflix, Samsung...

Let's look a little about how Jenkins fits into your workflow, and how continuous integration works in general.


Commit code
Jenkins reports on build result
Jenkins fetches committed code
Compile & test

To do something in Jenkins, **create a job**.
This is a **list of instructions**.

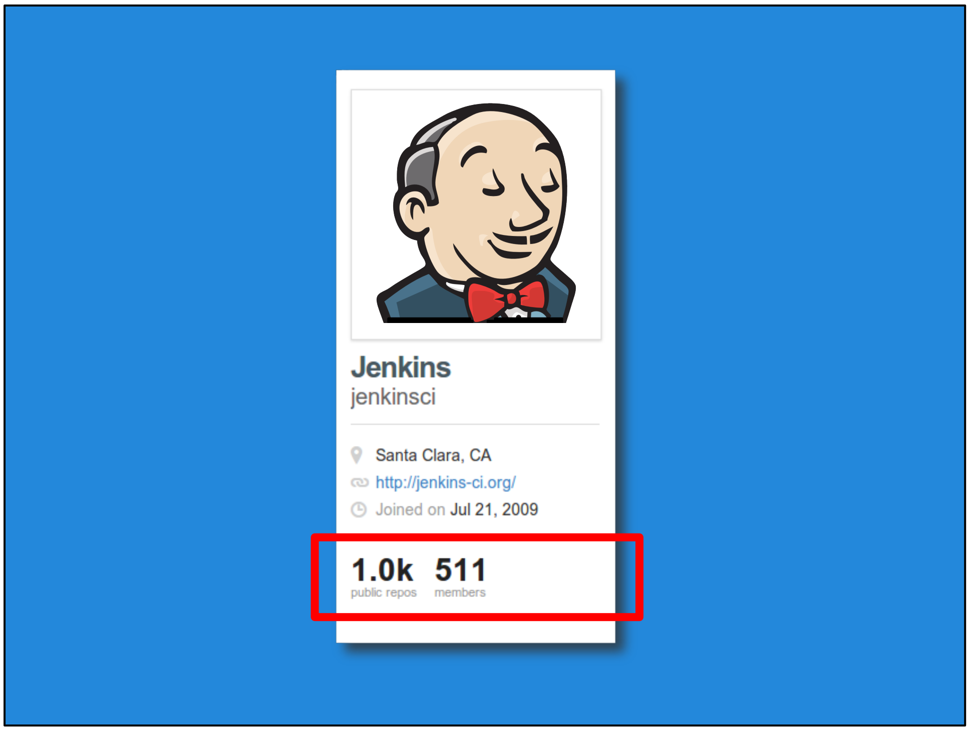
Set up a job, and commit your code as usual.

Repo pushes changes, or Jenkins polls for updates. Or cron.
The code is then built, and tested etc...
Jenkins reports on the results, e.g. sends an email to culprits.

So you get feedback within minutes, while the culprit is still around...

You can run Jenkins on any OS, so here you can have your Linux machine for Android, Mac for iOS builds etc.
These machines can be your own computer, a server in your network, or anything you can reach via SSH. Including EC2 etc...

So, this sounds good. But how does Jenkins get my code, and how does it work with Android or iOS etc.?


Jenkins
jenkinsci
Santa Clara, CA
http://jenkins-ci.org/
Joined on Jul 21, 2009
1.0k public repos
511 members

The answer is: **Plugins**.

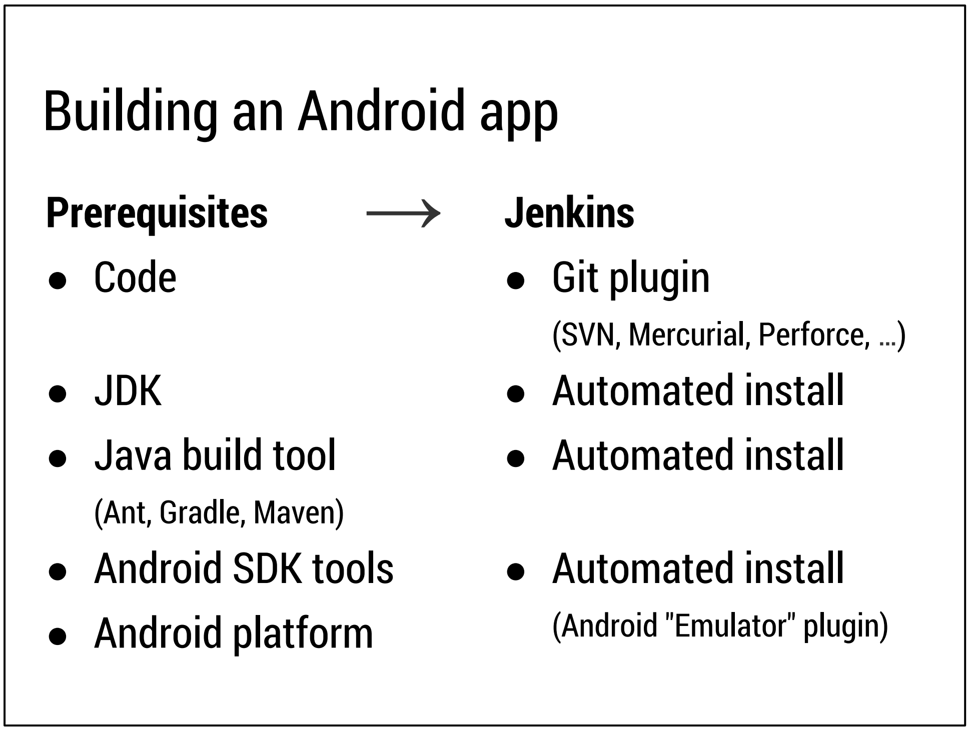
There are over 700 plugins released for Jenkins, so every use case is covered. If not, you can write your own plugin.

Categories of plugins include:
- SCM
- Execution environment
- Build steps
- Notifiers

So, let's see how this works with Android.

Building

The first step with Jenkins is **building an app**.



Building an Android app

Prerequisites →	**Jenkins**
Code	Git plugin (SVN, Mercurial, Perforce, ...)
JDK	Automated install
Java build tool (Ant, Gradle, Maven)	Automated install
Android SDK tools	Automated install (Android "Emulator" plugin)
Android platform	

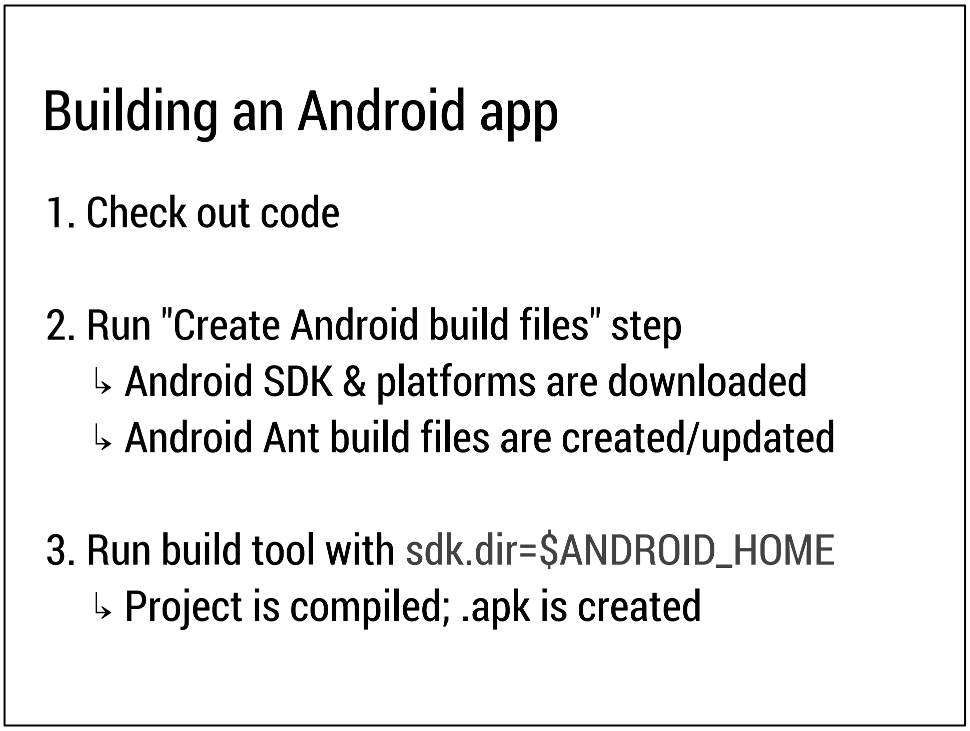
What do we need to build Android apps?

Ant / Gradle / Maven
+ Android SDK with its custom Java tools etc.

Jenkins has support for all these tools, and can automatically install many of them for you.

Jenkins also has the "Android Emulator Plugin", which has a slightly misleading name -- it will also auto-install all the Android dependencies and help with building your app. This is cool -- it automates a *lot* of things for you.

Ok, getting started is pretty easy, so how do we actually build an app?



Building an Android app

1. Check out code

2. Run "Create Android build files" step
   ↳ Android SDK & platforms are downloaded
   ↳ Android Ant build files are created/updated

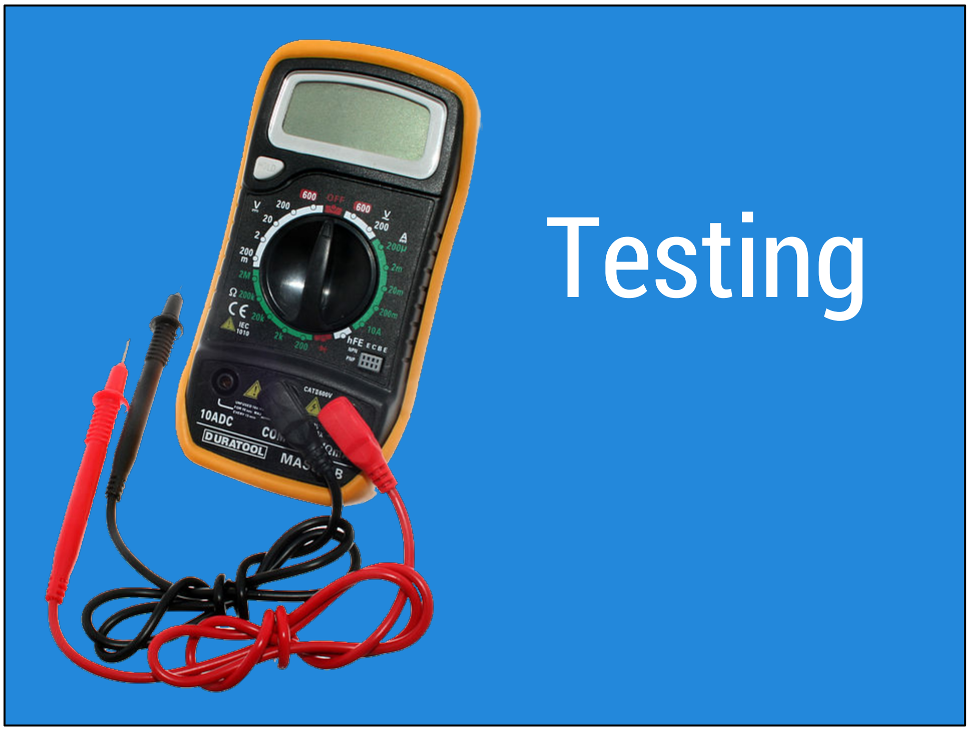
3. Run build tool with `sdk.dir=$ANDROID_HOME`
   ↳ Project is compiled; .apk is created

We create a new job, and configure it with three basic building blocks.

When you check out a repo, you need to create the build files, or at least tell Android where your SDK directory is (e.g. the local.properties file).

We'll take a look at this in a minute.

Demo



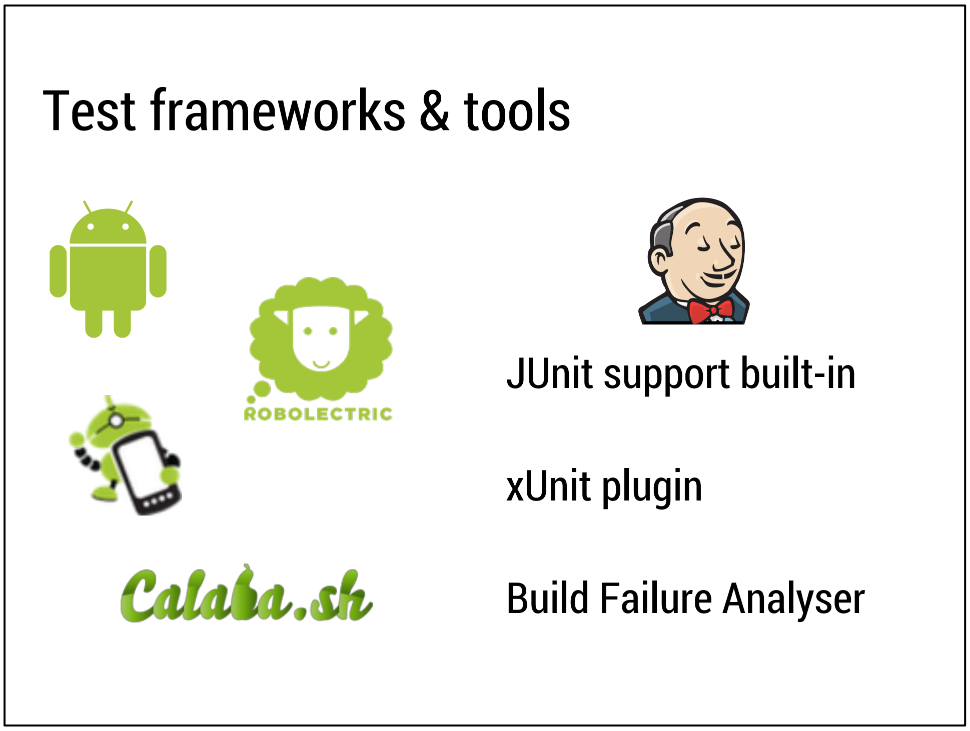
Testing

Ok, now we have a Jenkins job which is building your APK.

This builds our software in an independent environment and gives us feedback as soon as possible. Great.

What next? We can do some automated testing?

So what's the first step?



Test frameworks & tools

JUnit support built-in

xUnit plugin

Build Failure Analyser

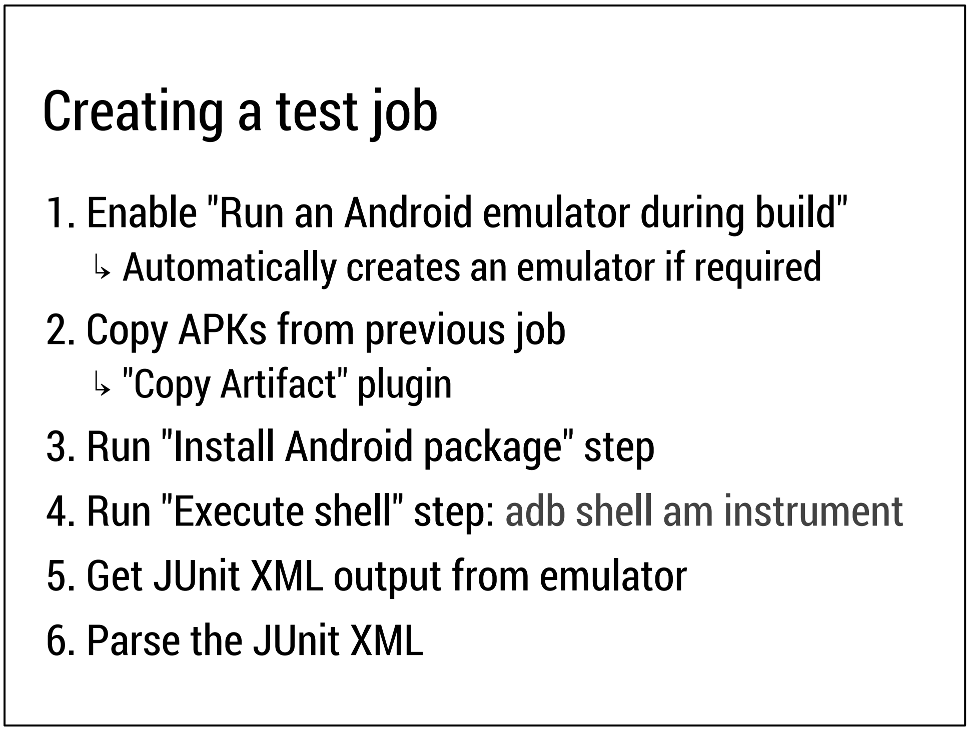
In order to run tests, we first need to write tests!

While I won't cover writing tests here, there are various frameworks which make it easy to get started.
Android uses JUnit3; there are projects like Robotium, Roboelectric...

So how do these fit into Jenkins?

Basically, so long as your tests can output JUnit XML (or similar), Jenkins can handle it, out-of-the-box.
You can use the android-junit-report project to easily output JUnit XML from your Android test runs.
If you can't, for some reason output this, use the BFA plugin.

So, how do we run tests in Jenkins?



Creating a test job

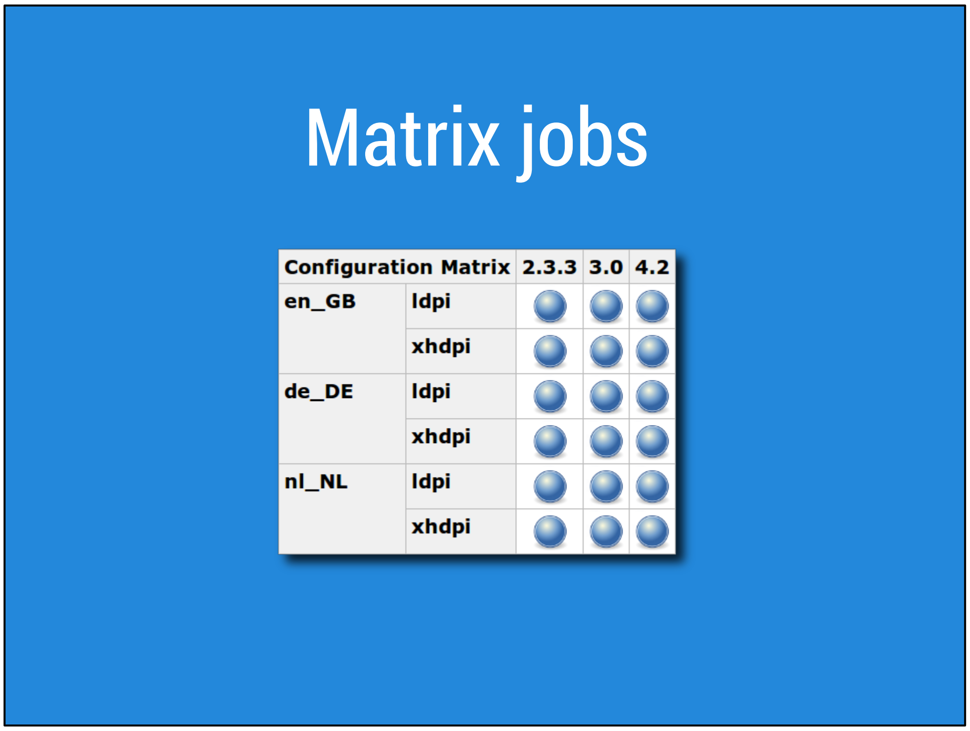
1. Enable "Run an Android emulator during build"
   ↳ Automatically creates an emulator if required
2. Copy APKs from previous job
   ↳ "Copy Artifact" plugin
3. Run "Install Android package" step
4. Run "Execute shell" step: `adb shell am instrument`
5. Get JUnit XML output from emulator
6. Parse the JUnit XML

Android emulator plugin

This lets you tell Jenkins, for example, "I want an Android 4.1 emulator, with a 10MB SD card, set to British English", and Jenkins will then automatically generate that emulator, start it up, and start saving all of the log output to disk.

And, if you don't have the Android SDK installed on your build machine, or the correct platform version for the emulator you want, then Jenkins will automatically install them for you.

Once you're done, Jenkins will automatically shut down the emulator at the end of the build.



Matrix jobs

Configuration Matrix		2.3.3	3.0	4.2
en_GB	ldpi	●	●	●
	xhdpi	●	●	●
de_DE	ldpi	●	●	●
	xhdpi	●	●	●
nl_NL	ldpi	●	●	●
	xhdpi	●	●	●

Ok, so Jenkins can generate new emulators for us automatically, which saves us some time.
But this doesn't really help us against the problem of multiple OS versions, screen sizes, languages etc...

Multi-configuration or "matrix" jobs are a very cool Jenkins feature -- these run the **same basic job configuration** multiple times, but each time with one **variable** changed, until all combinations have run.

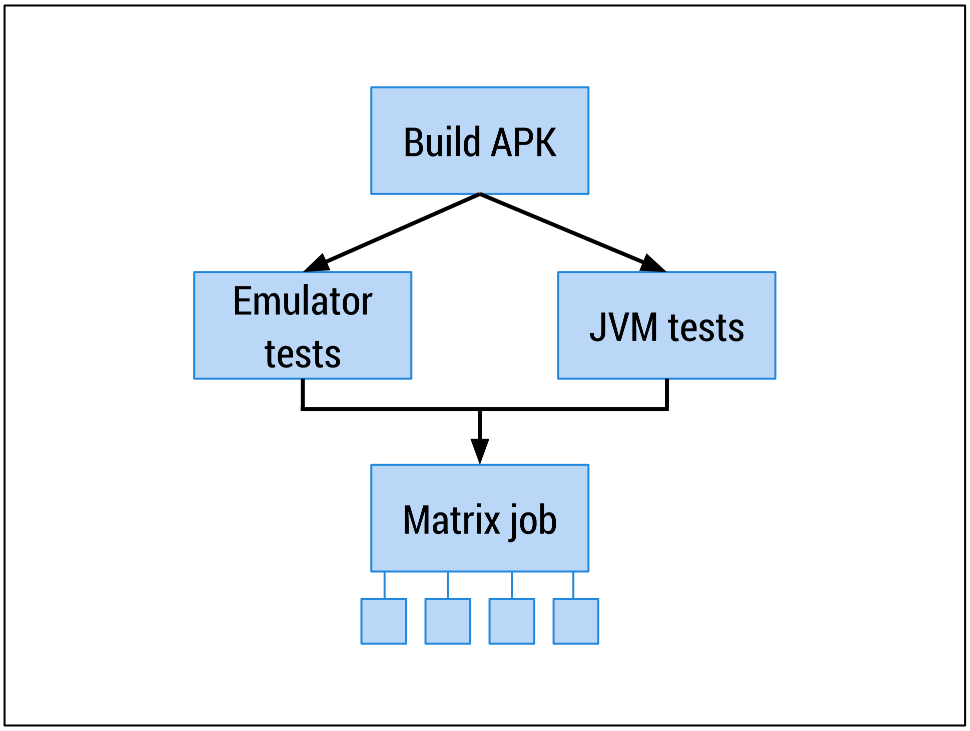
These variables can be things like operating system, compiler version, anything you care about.

Important to us:
- Android version
- Screen size or density
- Locale

The advantage of running your tests across these combinations is that you can find subtle bugs which are specific to a particular screen size, language or OS version.

I've personally found some problems that probably would not have been found, even with lots of manual testing:
- Forgetting or mistyping "android:id" constants in obscure layouts (e.g. layout-de-ldpi-land), causing crashes on certain devices
- Not including the correct number of %s parameters in strings in other languages

(though Lint helps with this now)
- String formatting problems with decimal point vs decimal comma in German/English
- Important! SQLite version changes often (undocumented) between OS versions: I found certain advanced SQL queries were broken on older OS versions



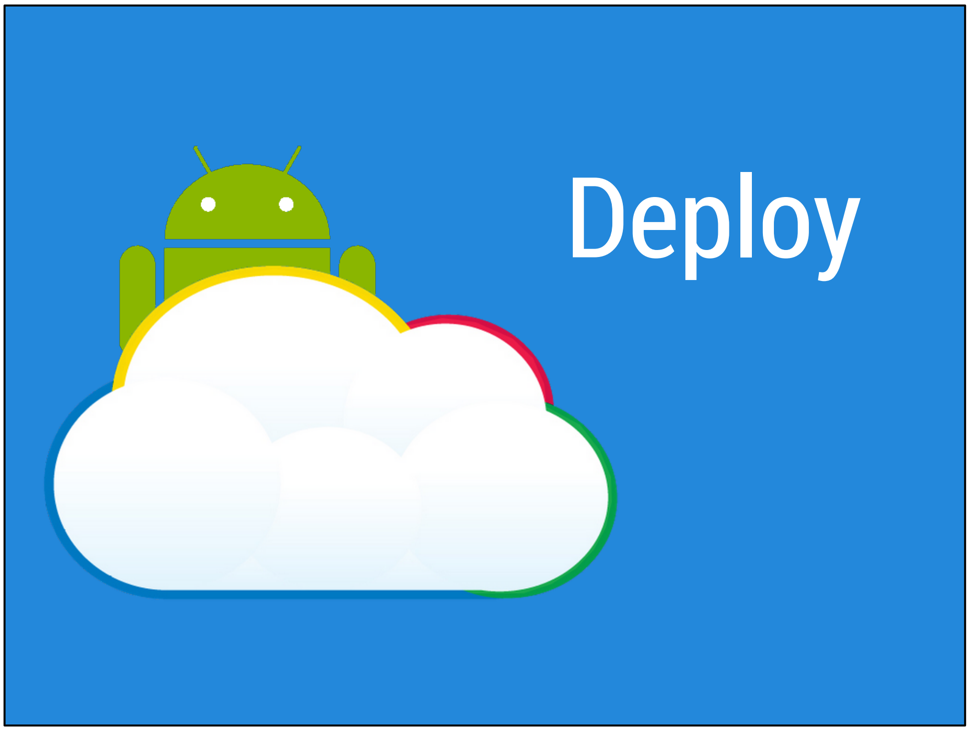
Jobs are easy to create in Jenkins, so create as many as you need to **split things up logically**.

You can also create **relationships** between jobs.

e.g. Create a fast-feedback build job, then kick off a fast JVM test job and a slow integration test job at the same time.
You can use the Build Flow plugin to create graphs like this one.

So let's take a look at testing.

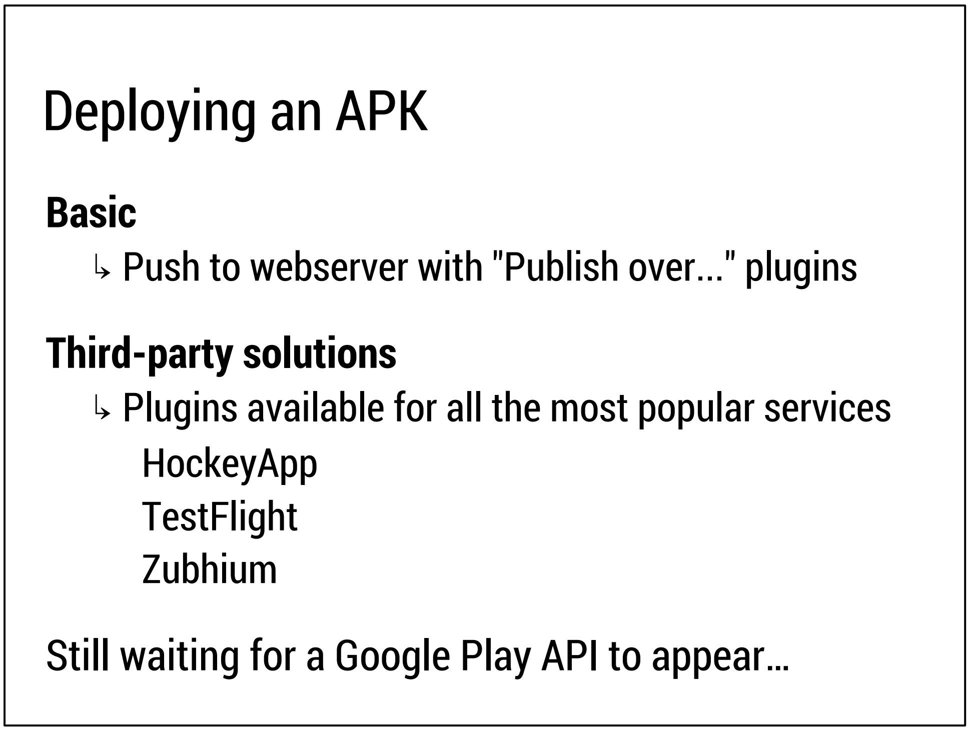
Demo



Distribution
So we’ve automated building and running of tests. Automation is fun.

But how about getting the app into the hands of users?

For example:
- QA department
- curious product managers
- your list of beta testers



Deploying an APK

Basic

↳ Push to webserver with "Publish over..." plugins

Third-party solutions

↳ Plugins available for all the most popular services

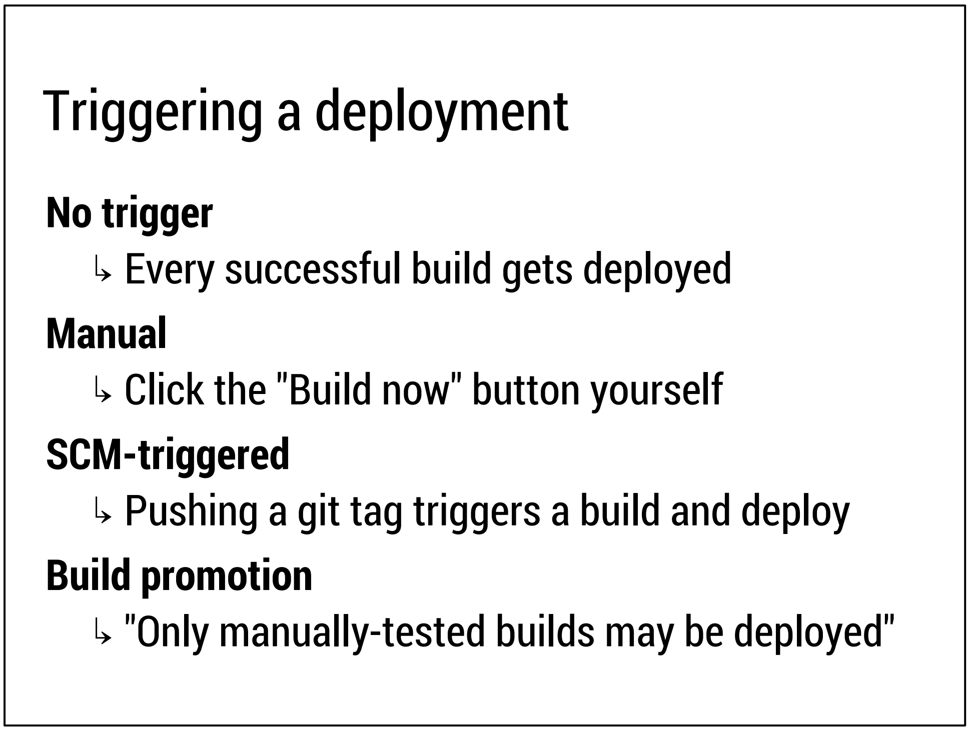
HockeyApp

TestFlight

Zubhium

Still waiting for a Google Play API to appear…

There are various ways we can get an APK out from Jenkins to users.



Triggering a deployment

No trigger
↳ Every successful build gets deployed

Manual
↳ Click the "Build now" button yourself

SCM-triggered
↳ Pushing a git tag triggers a build and deploy

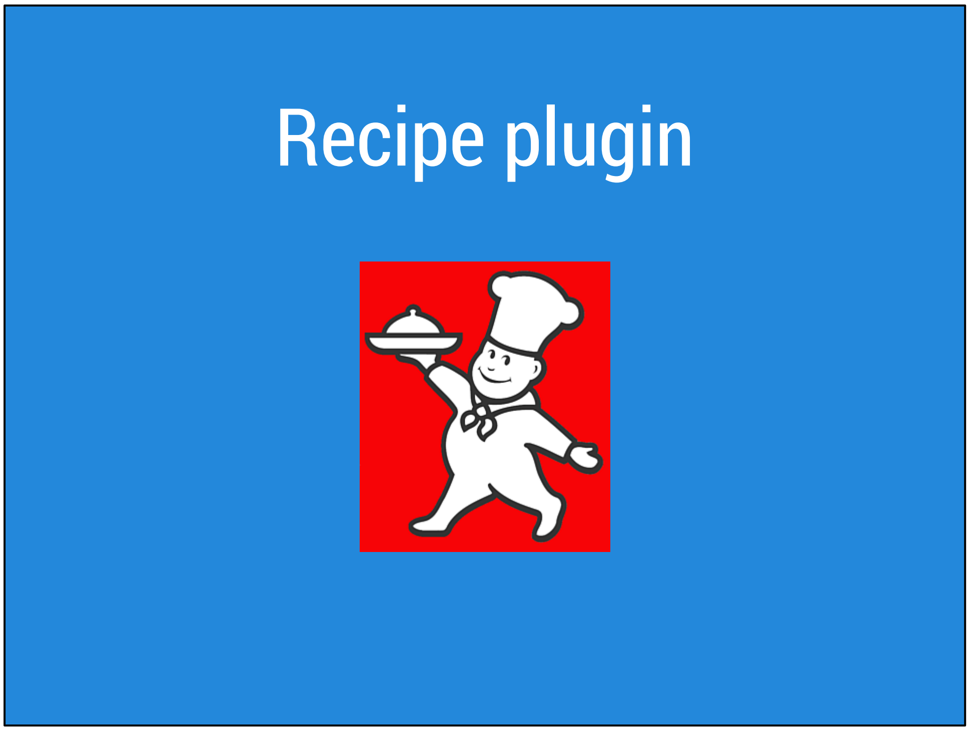
Build promotion
↳ "Only manually-tested builds may be deployed"

Ok, so that’s **how** we can upload an APK, but **when** can this be done with Jenkins?

The first two are simple.

The third is something I do: set up Jenkins to only publish your APK to HockeyApp (or wherever) when you apply a certain git tag (e.g. “beta/*”).
This means the only manual step involved to compile, test, upload and notify beta testers is to run “git tag && git push --tags”. Jenkins automates the rest.

The Build Promotion plugin is a more advanced way to do this; allowing you to add some logic like “this app must pass all the stress tests before I ‘promote’ it to be a beta APK”, or “this APK needs to be manually approved by QA before going for upload”.



Recipe plugin

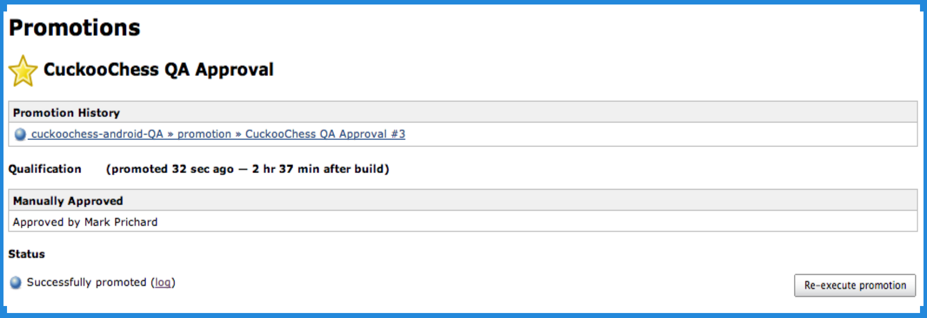
Quick aside

So far we've done a lot of the Jenkins setup manually in the demos.

The new Recipe Plugin for Jenkins lets you capture all the configuration, plugins and dependencies for a series of Jenkins jobs and wraps it into a single file, or “recipe”. You can publish these recipes from the Jenkins UI, or download recipes from others.

When you download a recipe that somebody has created, it will automatically set up all the jobs and install all the plugins required. Magic.

Build promotion



Promotions

CuckooChess QA Approval

Promotion History

cuckoochess-android-QA » promotion » CuckooChess QA Approval #3

Qualification **(promoted 32 sec ago — 2 hr 37 min after build)**

Manually Approved

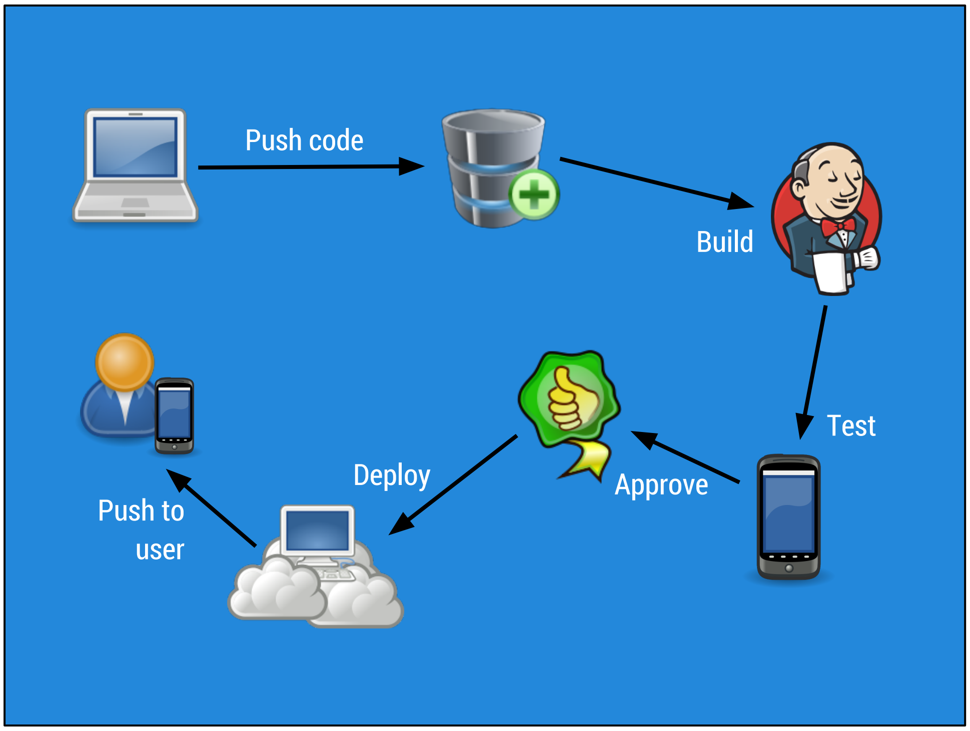
Approved by Mark Prichard

Status

Successfully promoted (log)

A quick example of Build Promotion.

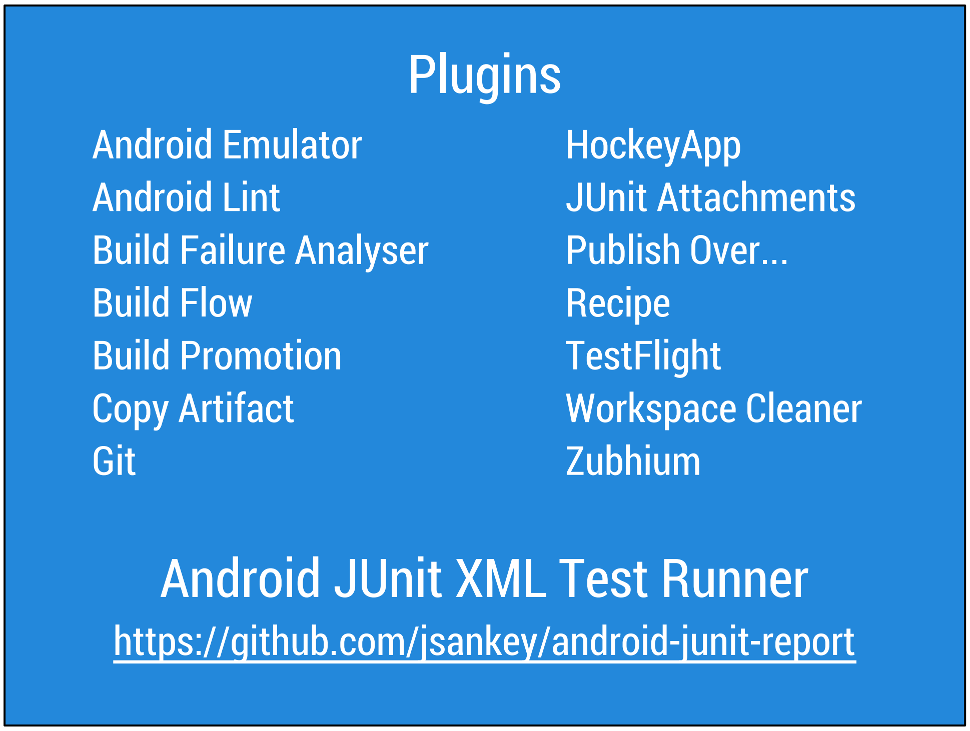
Build the app, run some automated tests. Someone from QA then comes along and clicks “Approve” -- only then is the build promoted.
Promoting the build will then do other stuff, like upload it ready for beta testers to download.



It's maybe hard to visualise the whole process, so here's an overview.

Essentially the **only** manual part is the initial code push by the developer.

Everything else after this is completely automated.



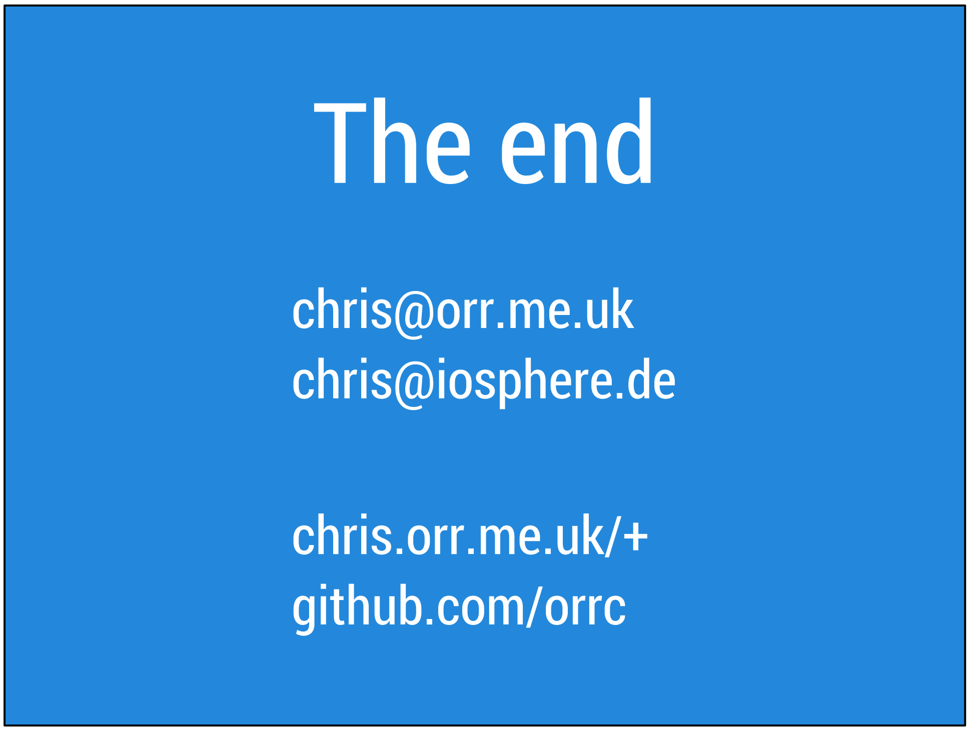
Plugins

Android Emulator	HockeyApp
Android Lint	JUnit Attachments
Build Failure Analyser	Publish Over...
Build Flow	Recipe
Build Promotion	TestFlight
Copy Artifact	Workspace Cleaner
Git	Zubhium

Android JUnit XML Test Runner

https://github.com/jsankey/android-junit-report

Here are the plugins and projects mentioned...



The end

chris@orr.me.uk
chris@iosphere.de

chris.orr.me.uk/+
github.com/orrc

Conclusion:
- Use Jenkins.
- Use the plugins.