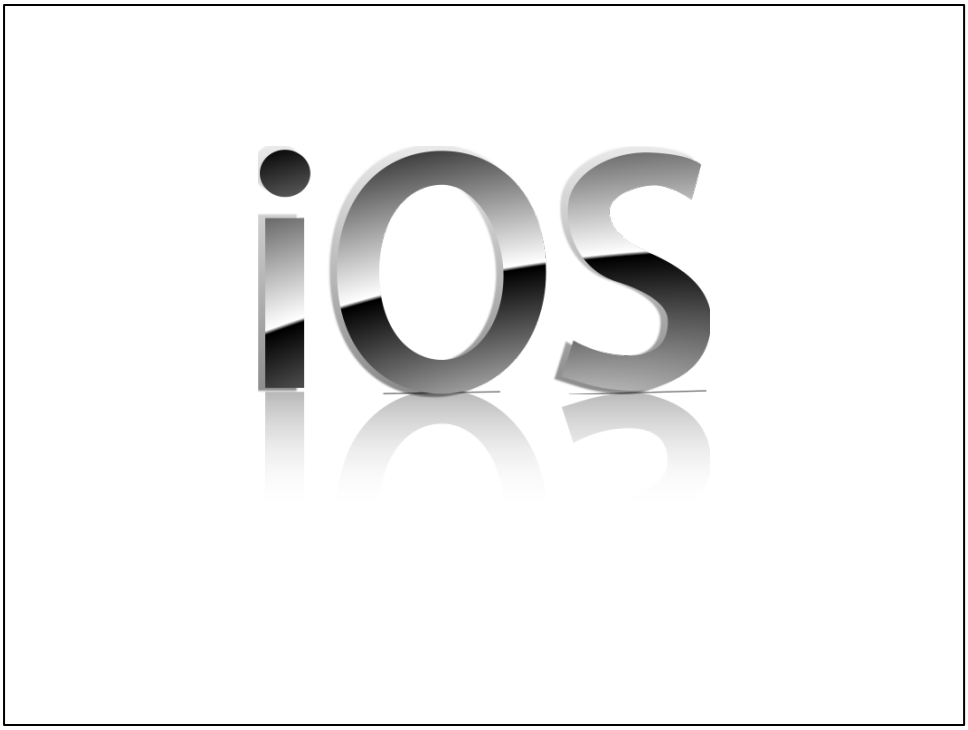
This is a lightning talk which is basically described by its title, where "mobile apps" really means Android and iOS.

We'll try show some useful Jenkins plugins, config and open source tools.

My connection to this topic?

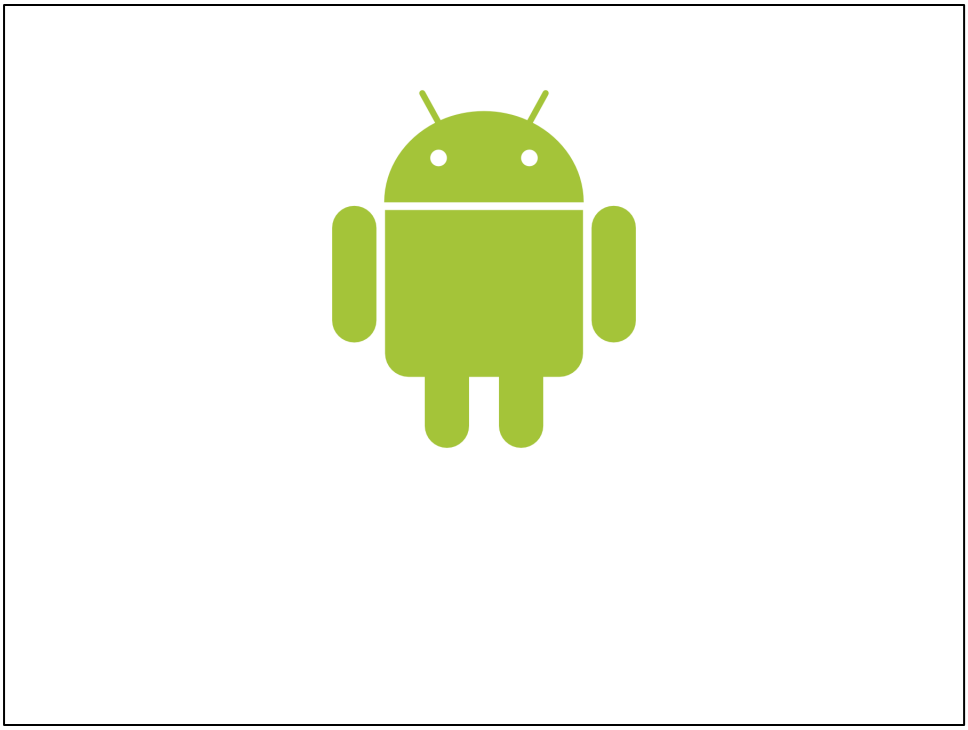
I work in Cologne, Germany for a company called iosphere.

We make mobile apps, mainly for...



iOS!

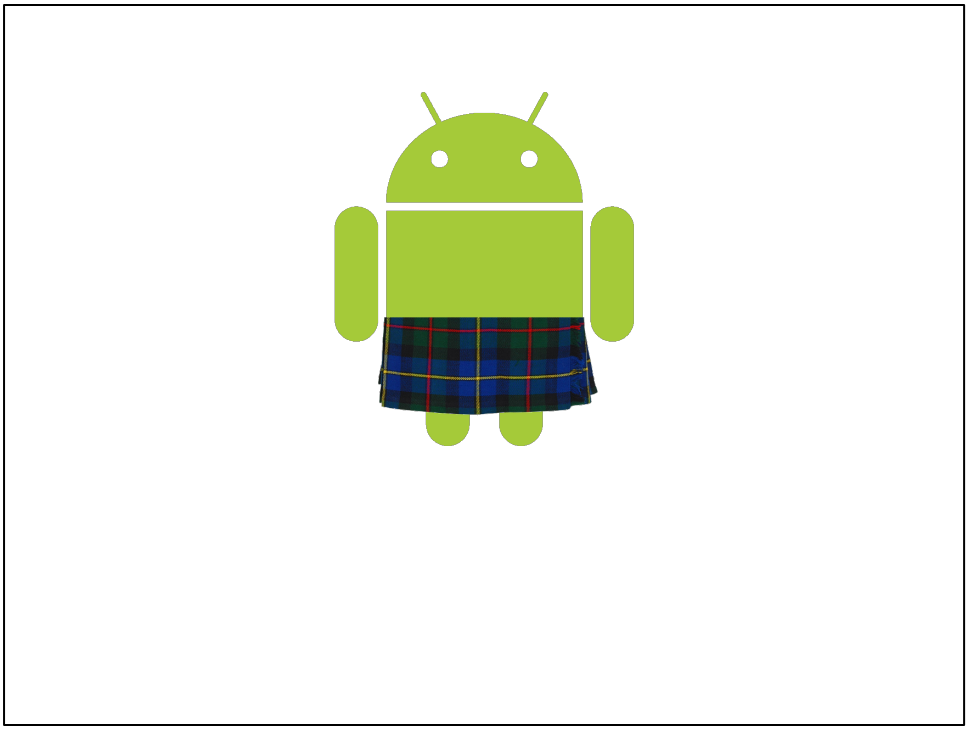
But we also do...



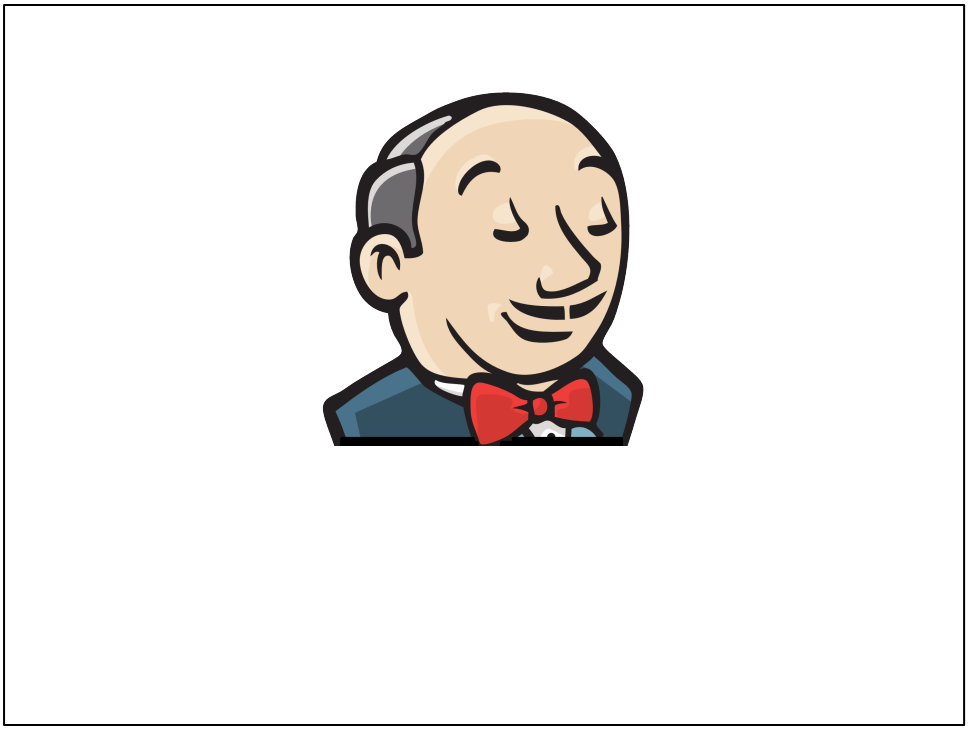
Android!

I've been working full-time as an Android developer since 2009.

Despite working in Germany for this time, I'm originally from...

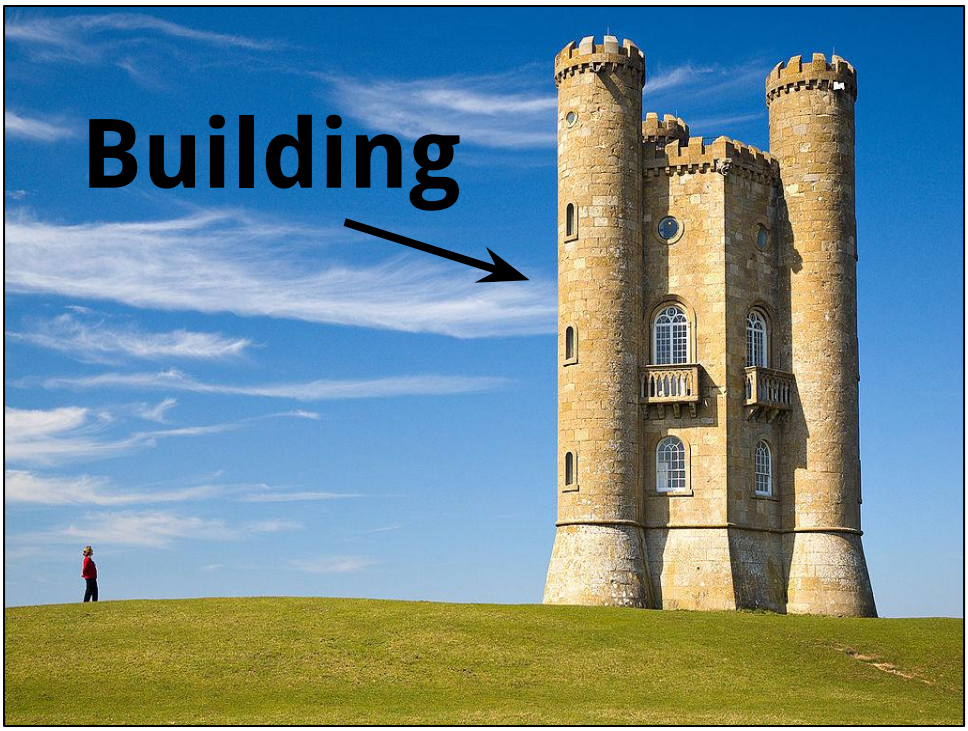


Scotland!

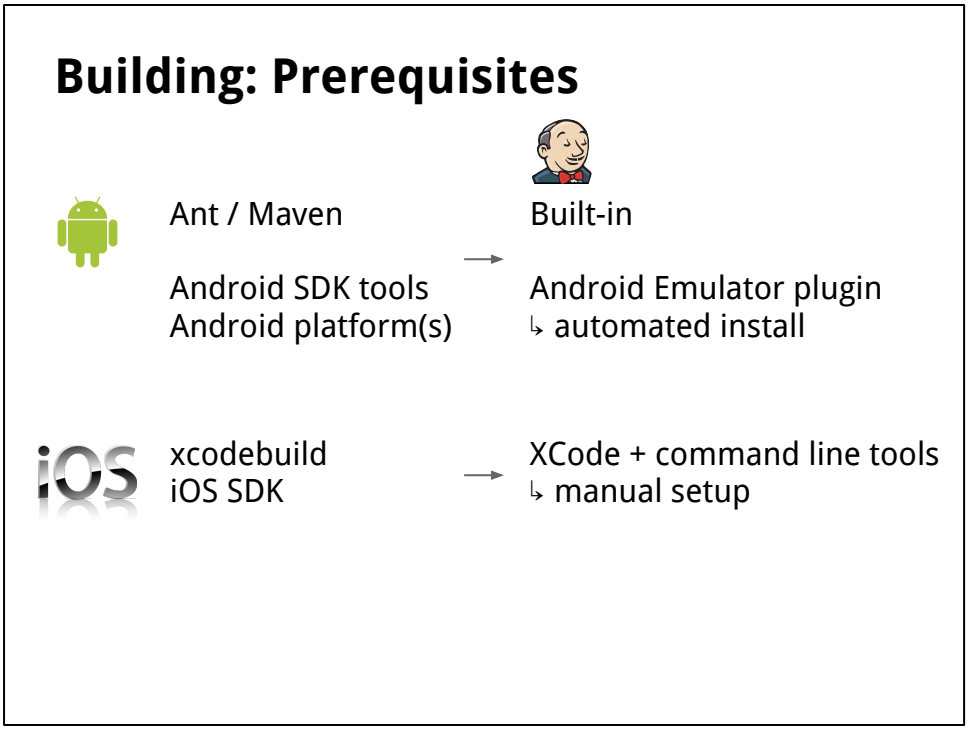


I'm involved in the Jenkins project and have worked quite a lot on Android-related plugins.

Working with my iOS colleagues, I've helped improve our iOS build, test and deployment setup, and also contributed to some more Jenkins plugins along the way...



Ok, let's start with the first topic, **building**.



What do we need to build mobile software with Jenkins?

Android projects normally use Ant or Maven as build tools.
We also need the Android SDK tools and Android platforms to build against.

Ant and Maven support are built-in to Jenkins.
Jenkins has the "Android Emulator Plugin", which can help install the Android stuff.

iOS uses xcodebuild to build, and requires that you have the iOS SDK installed, plus the command line tools.
So there's some manual setup involved -- this stuff isn't all automated yet.

Building mobile apps isn't hugely different from building any other type of project with Jenkins.
You need the code, do some setup, then build and archive.

The Android Emulator plugin can install any required platforms, and create the standard Ant build files for you (if you don't already have them in the repo).

Then you can just use Ant, or Maven, or the XCode plugin.

The XCode plugin automates loads of features -- building, packaging, signing etc... All you need is basically to provide the Target (e.g. Debug or Release) and your XCode file.

Archive the artifacts

**/*.apk

Once your app is built, **archive it!**

This is an important principle in Jenkins: build your app once, and let Jenkins take care of the build artifacts.
Then you can use the artifacts in later jobs, without doing nasty things like hardcoding paths to the workspace where the app was built previously -- you can't rely on the workspace still existing, the build slave still being online etc...

Just enable the "Archive the artifacts" option and enter a pattern like **/*.ipa to gather all the IPA packages...

Recipe Plugin

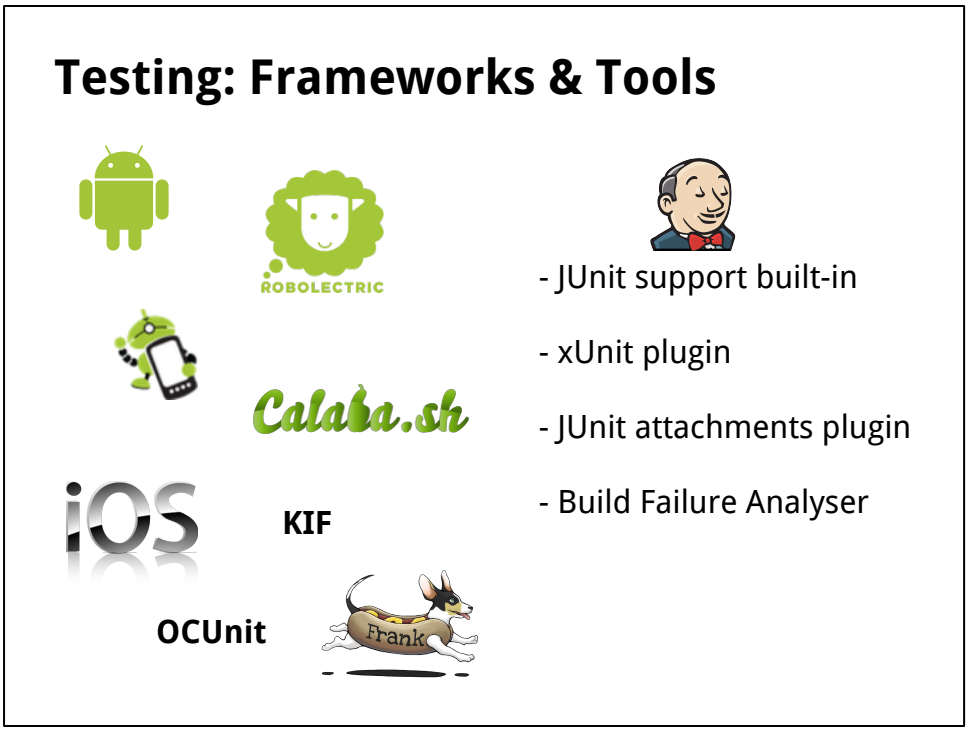
Talking about this config here is a bit abstract. How can you take this info and put it into practice in your own Jenkins instance?
It would be great if there was a way to share this type of configuration.

Thankfully we have the nice new Recipe Plugin.
This lets us share job, view and plugin configurations.

I've shared a couple of recipes relating to the topics in this talk, so go install the plugin!

Ok, with the app built, let's **run some tests**.

The first step is to write some tests...



We can't cover writing tests here, but there are plenty of frameworks which make it easy to get started.
So how do these fit into Jenkins?

Jenkins has built-in support for JUnit; it can parse JUnit XML and present nice trend graphs and tables of tests, time spent executing, regressions, fixes and so on.

Basically, so long as your test framework can output JUnit XML -- or you can write a script to generate it -- you're good. Any format that the xUnit plugin supports is also good.

If you can't or don't want to do this, you can also check out the Build Failure Analyser Plugin, which we'll check out quickly.

Testing: Running & Results

Copy artifacts from latest build (.apk, .ipa...)

Android Emulator Plugin
"Run an Android emulator during build"
"Install Android package"

iOS Device Connector Plugin
"Deploy iOS app to device"
Other possibilities?

Extract **JUnit XML** from device, or generate it

Publish test report

As for actually running the tests, here's a quick look at what we'd do.

We'd create a new Jenkins job where we import the previously-archived app files, i.e. the APK or .ipa or .app...

Then we have Jenkins plugins to help:
- Android Emulator will create an emulator for you, and help you to install APKs into it.
- iOS Device Connector will let you push a .ipa or .app onto an iOS device plugged in anywhere on your Jenkins cluster

--- ***Aside*** ---
While the Android stuff is fairly reliable and the command line tools are fairly cooperative and scriptable, this isn't always the case in the iOS world, in my experience.

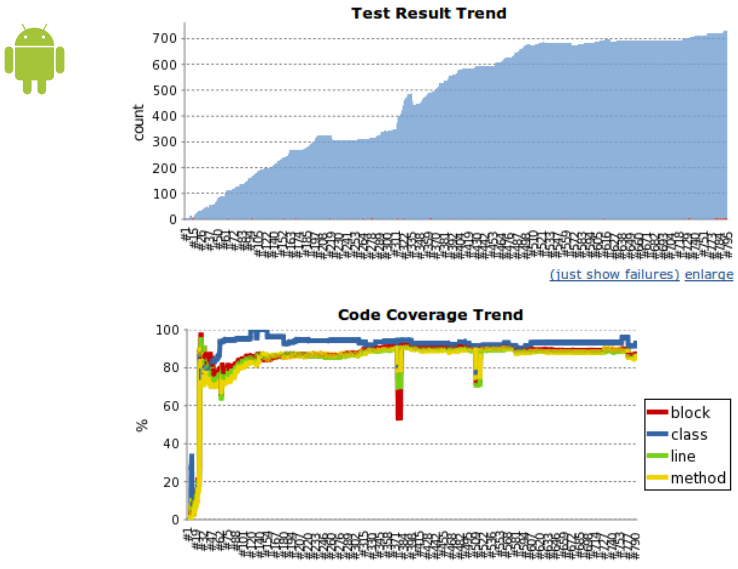
AFAIK, it's difficult to reliably start/stop the simulator in an automated environment without some effort, and it's not really possible to run multiple simulators in parallel.
From experience with the iOS Device Connector, running tests on multiple physical devices in parallel can also be flaky.
Unit testing is somewhat broken in general, I believe. Apple like to break things in XCode now and then...

But I'm very happy to be proven wrong :)
There are **always** things that can be improved for all mobile platforms with Jenkins...

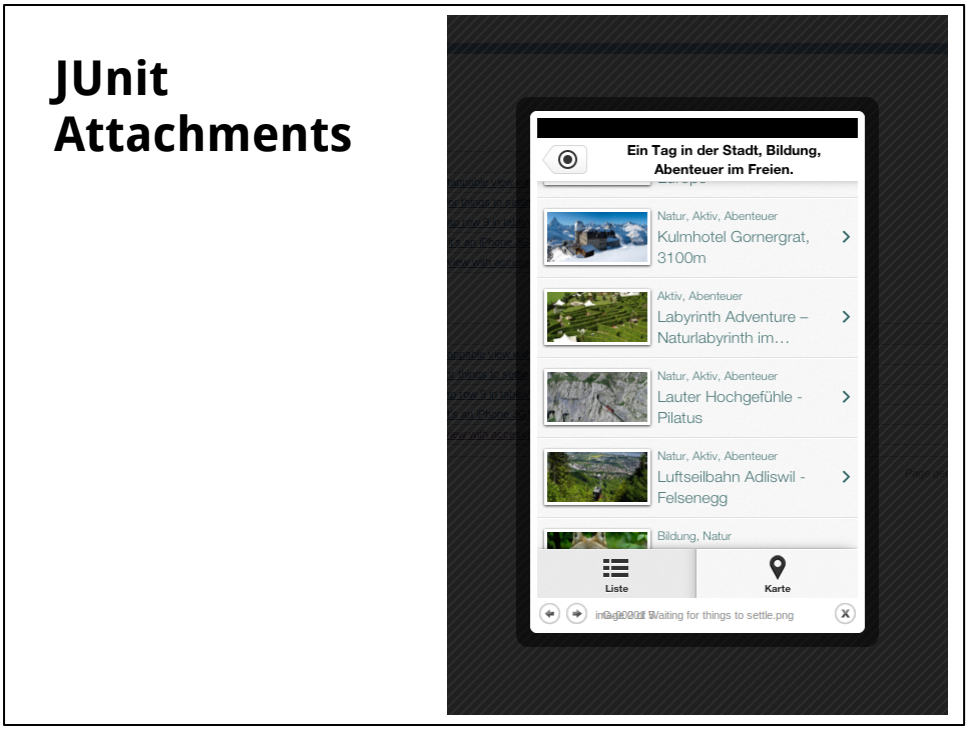
Anyway, once you've run your tests on a device, emulator or simulator, you need to extract or generate the machine-readable result, and let Jenkins parse it.

Then you get pretty graphs!

Testing: Running & Results



Here are some examples of an Android app with JUnit results and Emma code coverage graphs.

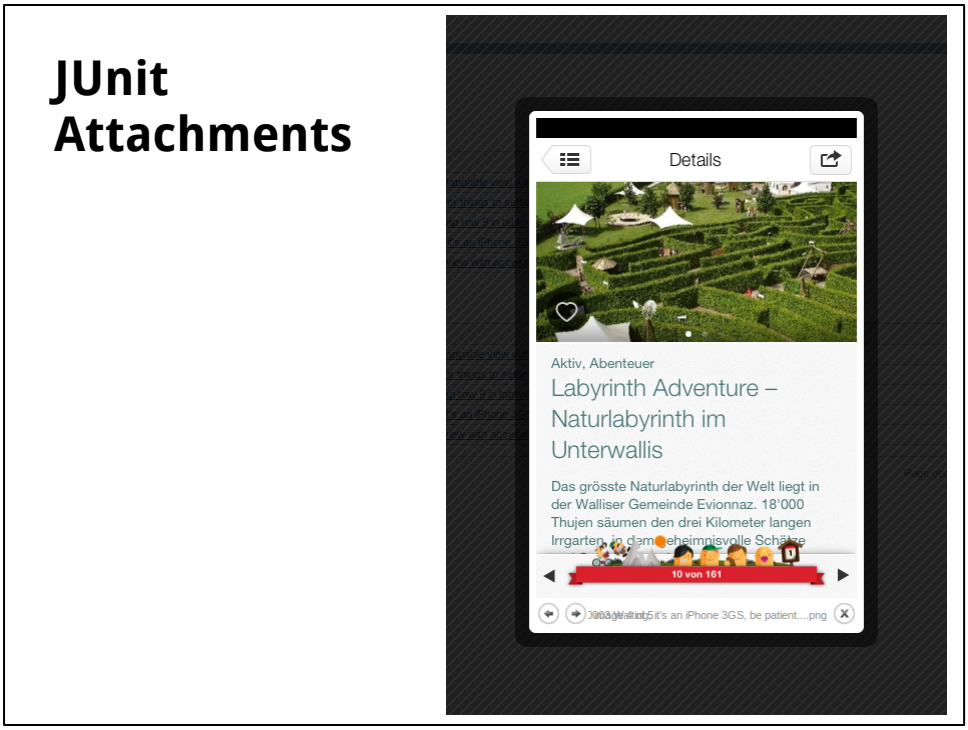


Another handy test-related plugin is the JUnit Attachments Plugin.

If you write "[ATTACHMENT|/path/to/screenshot.png]" into your JUnit XML, this plugin can pick up on that and show those file(s) linked directly to the relevant test results in the Jenkins UI.

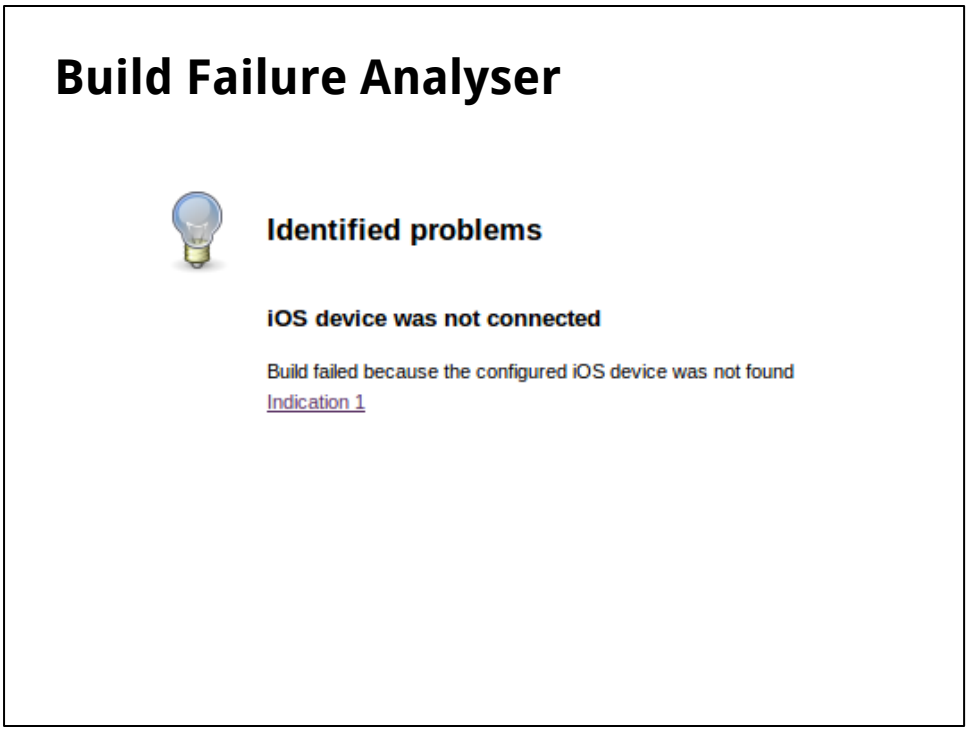
You can use this any type of file, but as this example shows, we've integrated it with our KIF tests for iOS.
We use this for testing, and the screenshots are really helpful to find **very quickly** where and why something broke.
e.g. The Jenkins build will say "Test X failed", and clicking that link will show the screenshot, maybe with a dialog saying "No network connection".

We also use KIF and this plugin for automating the laborious and tedious job of generating app-store screenshots, on multiple devices, for multiple languages.



When the JUnit Attachments plugin detects that you've attached an image, it shows it in a nice lightbox.

You can then use the arrow keys to flip back and forth through the images. Handy.



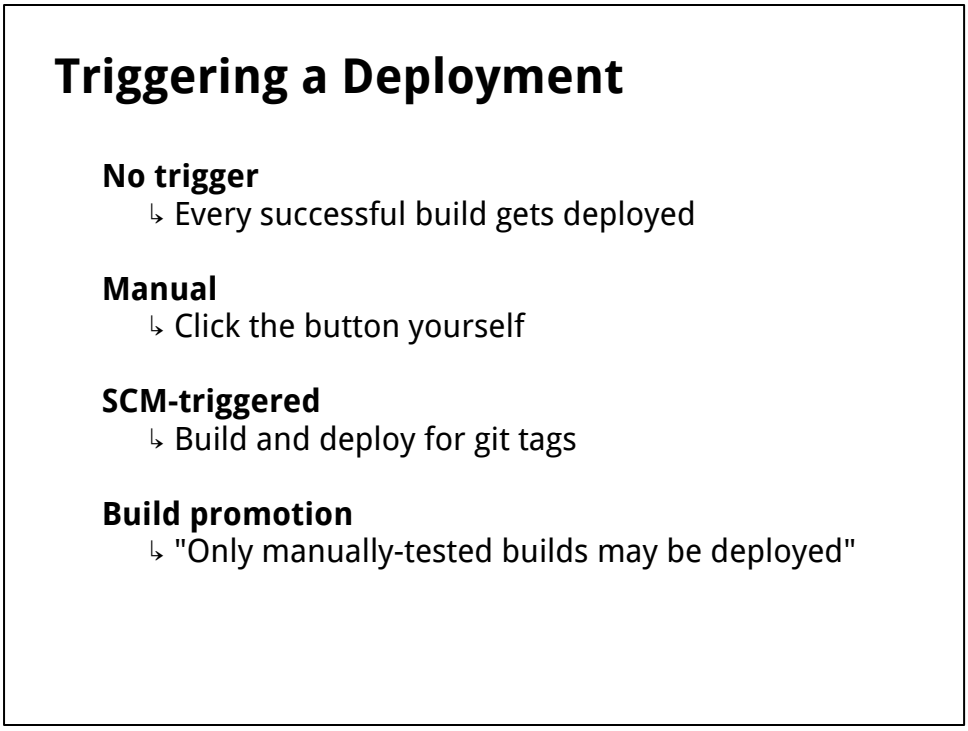
As mentioned, if you can't get nice JUnit output out of your device, you can also try another new plugin.

Robert Sandell and his colleagues at Sony Mobile wrote the BFA Plugin.

If a build fails, oftentimes you have to manually search through the console output looking for the reason.
The BFA plugin automates this -- you define some regular expressions and the associated description once, and then the plugin applies this to any failed builds and shows the result on the build page.

Here's an actual failure from our build system at iosphere -- a build was started using the iOS Device Connector plugin, but somebody had unplugged the configured device...

Having successfully built and tested our mobile apps, let's push them into the hands of our users, beta testers, QA department, or whomever...



Jenkins provides various ways of making a deployment happen.

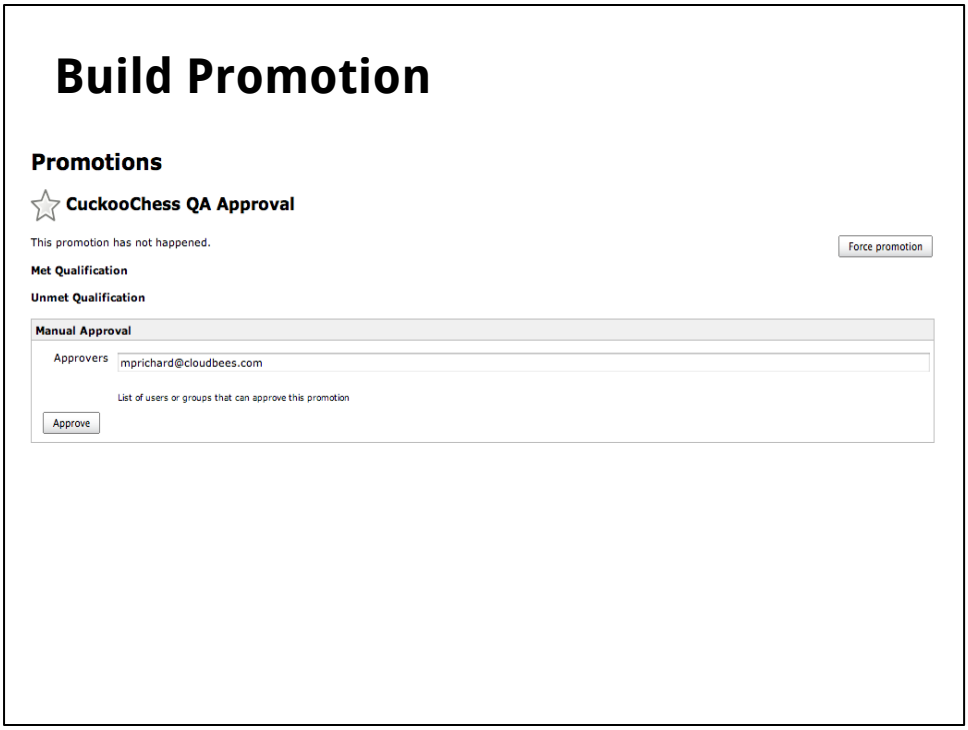
Say you have a Jenkins job which does all of your deployment logic. Again, this job should take an archived app artifact and then perform the relevant deployment logic.

But how to run this job?
- You can simply run this job after every successful build

- If you don't want to deploy every commit, you can simply manually start the job

- You can build the app and deploy it, but only when a certain SCM tag is pushed to the repo (e.g. at iosphere we automatically start deployments when somebody tags the git repo with a pattern like "deploy/beta-1")

- You can use the Build Promotion Plugin to define more advanced workflows...

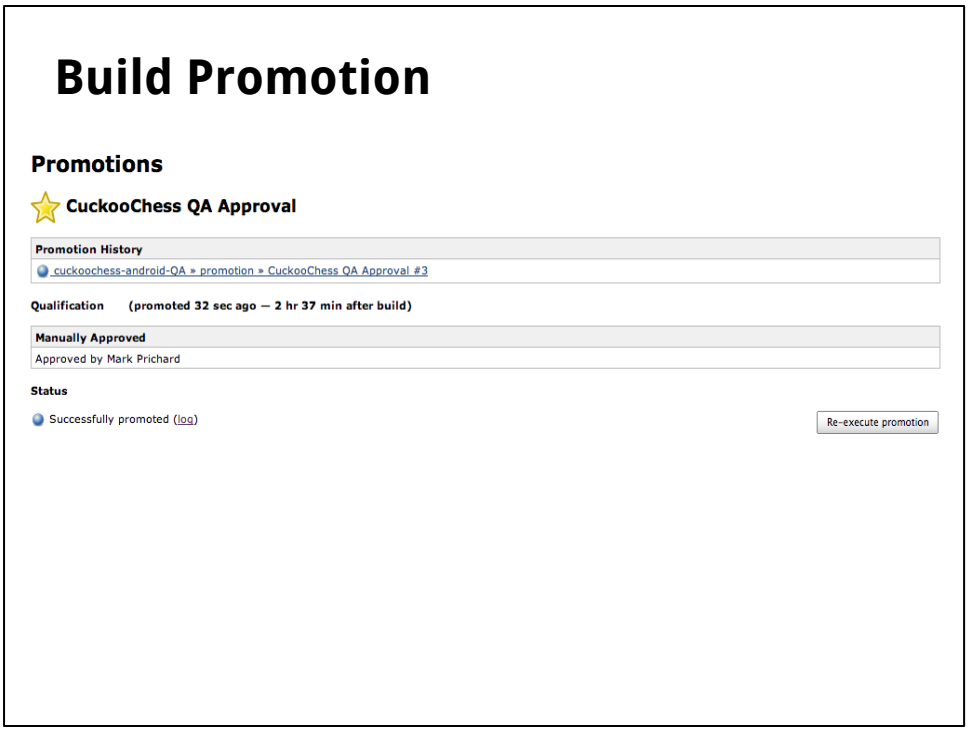


Here's an example of a workflow defined using Build Promotions.

This is the page in Jenkins for a given build of an app.
The app has been built, but it's waiting to be "promoted" -- which then can kick of one or more pre-defined steps.

In this case, the app would be downloaded from this page by a tester, who would manually test the app.

When they're satisfied with the built app, they could click the "Approve" button to promote this build.



Now that this build was approved, you can see that it kicks off another Jenkins job, deploying the app.

Deploying the App

Basic

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

iOS ad-hoc deployment plugin

↳ Generate ad-hoc metadata and push to webserver
(Not quite ready)

Third-party solutions

↳ Plugins available for all the most popular services
HockeyApp
TestFlight
Zubhium

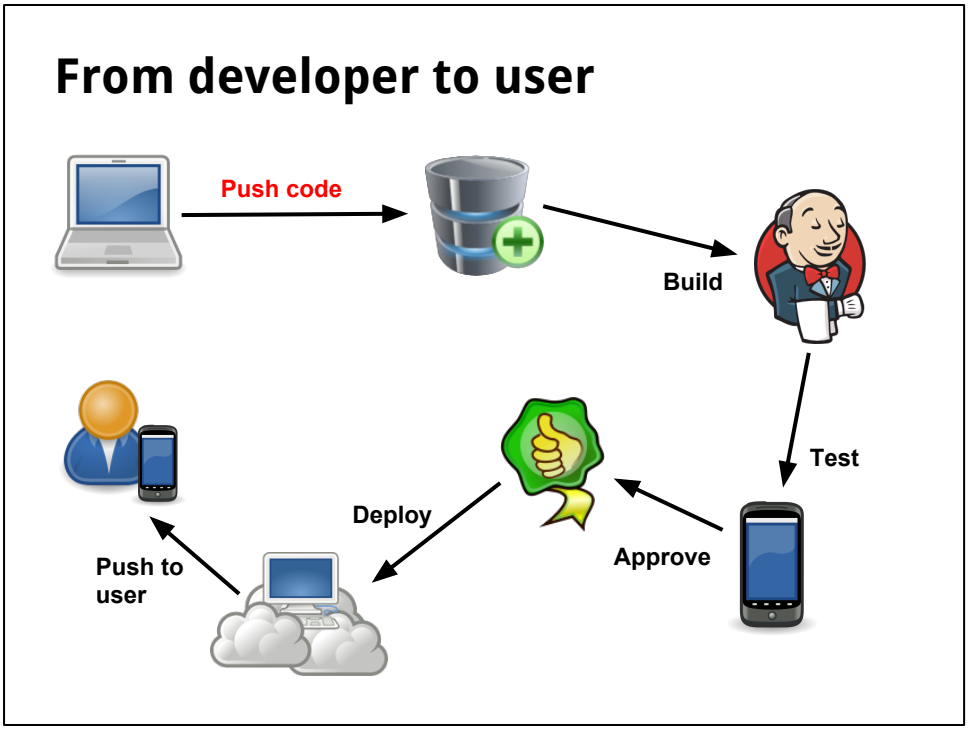
So that's *when* the app would be deployed, but *how* would we do so?

Again, there are multiple options and plugins to help.

- For Android, we could simply use a "Publish over..." plugin to publish an APK to a webserver, via SSH, FTP, SMB etc...

- For iOS, it's less simple. For ad-hoc deployment, before pushing to a webserver you also need to generate some metadata -- a .plist file.
  - There's a plugin currently being built called "iOS ad-hoc deployment plugin" which does this automatically for you.

- Finally, there are third-party solutions which host apps for you, providing ad-hoc distribution, push notifications to users when new versions are available, plus analytics
  - Jenkins has plugins for all the most popular services, so you can simply specify the app artifact plus your service API key and your app is deployed!



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

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

Everything else after this can be completely automated.

Thanks to...

Plugins

Android Emulator
Android Lint
Build Failure Analyser
Copy Artifact
HockeyApp
iOS Device Connector
OTA Ad-hoc Deployment
Promoted Builds
Recipe
Release
Warnings
XCode

Images

"Broadway Tower", by Newton2 at en.wikipedia [CC BY 2.5]
https://commons.wikimedia.org/wiki/File%3ABroadway_tower.jpg

"Digital Multimeter", by oomlout on Flickr [CC BY-SA 2.0]
http://www.flickr.com/photos/snazzyguy/3811449250/

So, that's pretty much it.

Check out some of the plugins I mentioned.
I think the Recipe plugin will really take off for sharing "best practices", which will be excellent.

Of course, there are also about 30 different plugins for various source control systems, like Git, SVN, Mercurial, which I haven't mentioned here.

Recipes!
tinyurl.com/recipeplugin

Slides!
chris.orr.me.uk/jenkins-mobile-recipes

What can be improved?
Next session: Jenkins dev discussion

Donate!
jenkins-ci.org/donate

Check out the Recipe Plugin and the recipes (so far I published a basic Android build recipe, plus a demo recipe showing how to start a build based on a git tag).

Every week, Jenkins itself gets downloaded 50,000 times and there are half-a-million plugin downloads...
Nobody gets paid by the project, but it needs to pay for servers, SSL certificates, stickers, flyers...
So donations are appreciated and, relevant to the FOSDEM audience, you can now donate in Euros! This is even tax-free in Germany :)

Give us feedback!
Please, tell us if and how your use the you use Jenkins and mobile.
How can we improve?
What are your needs?

Happy building!