CI/CD AUTOMATION WITH GITLAB FOR IOS PROJECTS

Presented by Václav Vidoň
Agenda

- Importance of CI automation
- Manual procedures, automatable with CI
- Platform-specific iOS CI automation
- Full iOS test automation in CI?
- Implementation examples
Agenda

• Importance of CI automation

• Manual procedures, automatable with CI

• Platform-specific iOS CI automation

• Full iOS test automation in CI?

• Implementation examples
Importance of CI automation

Aka. why would you do stuff manually?

• In the ideal world:
  • Commit -> push -> **pipeline** -> MR -> CR-> approve & merge
    -> manual testing -> release

• In the slightly more realistic world:
  • Run auto-tests locally -> commit -> push -> MR -> CR ->
    approve & merge -> set release version -> assemble release
    build locally -> manual testing -> release
Pipeline in CI/CD

• Set of automatic commands

• Shapeable for any project

• All DevOps & Git tools have this option
Pipeline in CI/CD
Common manual procedures, automatable with CI

- Building your app for testing
- Release version increase
- TestFlight / AppCentre or similar tools for inter-project versioning
- Issues-tracking tools integrations
- Code quality and analysis
- Communication and reporting tools integrations
- Automatic testing – UNIT, API, GUI – smoke and regression sets
Platform-specific iOS CI automation

• Pods, builds, tests, TestFlights, releases - all are slowed down and complicated mainly by:

1. No dockerization:
   • All tasks run on the same server (runner)
   • Or on multiple runners

2. XCode (and CLI tools) are quite performance-demanding
Full iOS test automation in CI?

• All kinds of tests are possible

• Simulator or real device mode

• Challenging to set the environment

• Prevents human error
Implementation examples

- TestFlight build:
  - Occasional
  - Just one at a time
- Steps:
  - Dependencies
  - Build number increase
  - Building
  - Code signing
  - IPA export
  - Upload to store
  - Change log
  - Screenshots
  - Release notes
testFlight:
stage: build
<<: *normalTFPipeline
script:
  - ${/scripts/ci/concurrentPodUpdateInstall.py $COCOAPODS_UPDATE_LOCK_FILE
  - xcodebuild -workspace xxx xcworkspace --scheme xxx -derivedDataPath $PWD | xcpretty --s
  - Build/Products/Debug/CoreDataExporter --outputDirectory CoreDataStack/CoreDataStack/Bundled/World
  - buildNumber=${/scripts/ci/updateBuildNumberTag.sh}
  - xcrun agvtool new-version --all $buildNumber
  - fastlaneParameters=$(/scripts/ci/slackbot.py --TestFlight | tr "," "\n")
  - bundle exec fastlane beta jiraLink:${fastlaneParameters[0]} slackUser:$fastlaneParameters[1] appstoreActivityUrl:${fastlaneParameters[2]} buildNumber:
when: manual

```bash
#!/bin/sh

git fetch --tags --force --quiet

BUILD_NUMBER_TAG_PREFIX="Build"

LATEST_BUILD_NUMBER=$(git tag | grep $BUILD_NUMBER_TAG_PREFIX | cut -d "_" -f 2 | sort --sort=human-numeric | tail -n 1)
INCEMENTED_BUILD_NUMBER=$(($LATEST_BUILD_NUMBER + 1))
INCEMENTED_TAG=${BUILD_NUMBER_TAG_PREFIX}_${INCEMENTED_BUILD_NUMBER}

git tag $INCEMENTED_TAG
git push git@github.skypicker.com:mobile/ios.git $INCEMENTED_TAG --quiet
echo $INCEMENTED_BUILD_NUMBER
```
def main():
    if "[skip jira]" in os.environ['CI_COMMIT_MESSAGE'].lower():
        print("Commit message contains exiting code.")
        sys.exit(0)
    if "devel" in os.environ['CI_COMMIT_REF_NAME']:
        print("Functionality for RC builds is not implemented.")
        sys.exit(0)

    jira = jiraAuthentication()
    marketingVersion = getMarketingAppVersion()
    buildNumber = getBuildNumberFromImportParameters()
    versionName = assembleReleaseName(marketingVersion, buildNumber)
    branchingName, storyName = getSummaryAndStoryNameFromBranchName()
    component = getComponentFromStory(jira, storyName)
    orginalStory = jira.issue(storyName)

    with open(openJson("version.json")) as f:
        versionDictionary = json.loads(f.read())

    with open(openJson("qaRequest.json")) as f:
        qaReqData = json.loads(f.read())

    for key, value in versionDictionary.items():
        versionDictionary["name"] = versionName
        versionDictionary["description"] = branchingName

    for key, value in qaReqData.items():
        qaReqData["summary"] = branchingName
        qaReqData["description"] = "Build: " + buildNumber
        qaReqData["components"] = component

    jira.create_version(versionDictionary["name"], "MOBILE", versionDictionary["description"],
                        originalStory.update({'fixVersions': [versionDictionary]}))
    newQARequest = jira.create_issue(fields=qaReqData)
    jira.create_issue_link("Relates", originalStory, newQARequest)

    if __name__ == '__main__':
        main()
Questions?