Continuous Deployment With Sparkle for macOS Apps

robenkleene/continuous-deployment-with-sparkle

@robenkleene
robenkleene.com
thepotionlab.com

What's Continuous Deployment?

Agile + Continuous Integration + Deployment

No manual steps to ship to customers.

…still two manual steps

The Process

☞ Branch `master` is always ready to ship

☞ Ceating a `X.X.X` version tag uploads a build

☞ Manually generate the appcast

Manual Steps

① Creating the tag

② Generating the appcast

Why Continuous Deployment?

"The Continuous Culture" by Kim van Wilgen

☞ **Slow**: 1 release every 100 days

☞ **Fast**: 7448 releases a day (Amazon)

Unused features

- ☞ **Large Deliveries**: 64%
- ☞ **Small Deliveries**: 14%

Chance of Success

- ☞ **Large Project**: 10%
- ☞ **Small Projects**: 74%

2017 State of DevOps Report

"High performers are doing significantly less manual work"

"HP LaserJet was able to increase time spent on developing new features by 700 percent."

Focus on high value work.

Uh, Native Apps?

Releases are more costly and harder to rollback.

But managing releases is still tedious.

Why not Mac App Store?

- ☞ File-System Access
- ☞ Subprocess Management
- ☞ Programming
- ☞ Freedom

Continuous Integration Services

- ☞ **Bitrise**: $36/month (200 free builds a month)
- ☞ **Travis**: $69/month (Free for open source)
- ☞ **CircleCI**: $39/month (Nothing free for macOS)
- ☞ **App Center**: $40/month (250 free build minutes per month)

How about free?

Implementation Summary

Build the app on the continuous integration server

① `xcodebuild` the zip

② `rsync` the zip to the server

Manually create the appcast on a developer machine

① `rsync` down all the zips

② `generate_appcast` to create the `appcast.xml`

③ `rsync` the `appcast.xml` and deltas back to the sever

Signing

Bitrise handles this.

Sparkle EdDSA signing.

Notarizing?

Need to wait an indefinite amount of time after uploading your app package to find out whether notarizing was successful.

Build Steps

```
SCHEME = Potion
EXPORT_PATH = build/
ARCHIVE_PATH = $(EXPORT_PATH)$(SCHEME).xcarchive
APP_PATH = $(EXPORT_PATH)$(SCHEME).app
ZIP_PATH = $(EXPORT_PATH)$(SCHEME).zip

xcodebuild archive \
    -scheme $(SCHEME) \
    -archivePath $(ARCHIVE_PATH)

xcodebuild \
    -exportArchive \
    -archivePath $(ARCHIVE_PATH) \
    -exportOptionsPlist ExportOptions.plist \
    -exportPath $(EXPORT_PATH)

/usr/bin/ditto -c -k --keepParent $(APP_PATH) $(ZIP_PATH)
```

(This is done on the continuous integration server.)

Deploying the App

```
app_version=$(agvtool what-marketing-version -terse1 | tr -d '\n')

rsync --archive --compress --ignore-existing --verbose \
  $zip_path \
user@server:\
"path/to/download.thepotionlab.com/potion/Potion\\ ${app_version}.zip"
```

Only run this for tags! (`"$app_version" != "$tag_match"`)

(This is done on the continuous integration server.)

Generating the Appcast

```
rsync --archive --verbose --delete \
user@server:\
"path/to/download.thepotionlab.com/potion/" \
./updates/

generate_appcast ./updates/potion/
```

(This is done on a developer machine.)

(Also a nice way to create a backup of all your builds.)

Publishing the Appcast

Upload the deltas:

```
rsync --archive --verbose \
    ./updates/potion/*.delta \
    user@server:path/to/download.thepotionlab.com/potion
```

Upload the `appcast.xml`:

```
rsync --archive --verbose \
    ./updates/potion/appcast.xml \
    user@server:path/to/download.thepotionlab.com/potion
```

(This is done on a developer machine.)

Implementation Summary

Build the app on the continuous integration server

① `xcodebuild` the zip

② `rsync` the zip to the server

Manually create the appcast on a developer machine

① `rsync` down all the zips

② `generate_appcast` to create the `appcast.xml`

③ `rsync` the `appcast.xml` and deltas back to the sever

Thanks!

robenkleene/continuous-deployment-with-sparkle

@robenkleene
robenkleene.com
thepotionlab.com