**Table of Contents**

[Starting a local server (on MacOS): Existing Docker container 2](#_Toc63000638)

[Building server 4](#_Toc63000639)

[Starting server from local build 5](#_Toc63000640)

[Pushing a server build to Dockerhub 6](#_Toc63000641)

[Build server in Xcode 7](#_Toc63000642)

[Run local tests 8](#_Toc63000643)

[Running local tests: Server access using ngrok 11](#_Toc63000644)

[Checking the local database 12](#_Toc63000645)

[Updating Dockerfile’s for new Swift version 13](#_Toc63000646)

[Debugging a failing “slim” Swift image 15](#_Toc63000647)

[Creating test data files for server 16](#_Toc63000648)

[Getting server working on iMac 18](#_Toc63000649)

[Push Notification Certificate 20](#_Toc63000650)

# Starting a local server (on MacOS): Existing Docker container

NOTE: This procedure assumes that mySQL is running locally. See below for instructions on how to do this.

**1. Make sure the Docker app is launched on MacOS.**

**2. Run the server: Do this from MacOS**

**a) Open a terminal window**

**b) Then,**

cd ~/Desktop/Apps/SyncServerII/SyncServerII

(Or, cd ~/Developer/ServerMain/).

**c) Last—run the server as a docker image**

./devops/runLocally.sh ~/Desktop/Apps/SyncServerII/Private/Server/ClientTesting-local.json latest

(Or, ./devops/runLocally.sh ~/Developer/Private/Server/ServerTests.json)

If this succeeds, it will give no output and not return. If it fails, it will give no output and return to the command line. Look at the logs in this case.

**3. Get the logs**

**a) Open a new terminal window**

**b) Change into the directory**

cd ~/Desktop/Apps/SyncServer.Run

**c) tail the log**

tail -f output.log

**4) Test the server:**

From a browser:

<http://localhost:8080/HealthCheck/>

(trailing slash is important)

# Building server

**1. Make sure the Docker app is launched on MacOS.**

**2. Start Docker build container**

docker run --rm -i -t -v /Users/chris/Developer/:/root/Apps crspybits/swift-ubuntu:5.3.1

**3. In the Docker container Terminal**

cd root/Apps/ServerMain

./Tools/clean.sh

./Tools/build.sh

If build.sh has problems (e.g., is very slow to build, and perhaps stalls/blocks), try:

./Tools/build.sh verbose

However, that’s not what helped in my most recent go-around with this (5/30/20). Rather, after floundering for several hours, I updated from Swift 5.0.1 in my Docker build container (devops/Docker/Building) to Swift 5.2.3 and now my

./Tools/build.sh

works with no delays. I assume there have been dependency resolution changes in the last few Swift releases.

I haven’t seen many other references to these issues online. Here’s one:

<https://stackoverflow.com/questions/47431510/swift-package-manager-not-resolving>

# Starting server from local build

This depends on you having carried out the “Building server” steps above. Then, do the following from the MacOS Terminal (not from within the Docker Terminal):

# From the MacOS Terminal (not from within the Docker Terminal), build the docker image syncserver-runnerimage:latest, without pushing it up to docker hub:

**./**devops/buildlatest.sh

# Then, do the following. Note that `latest` refers to the image created in the last step

./devops/runLocally.sh ~/Developer/Private/Server/ServerTests-local.json latest

Note that using ServerTests-local.json turns on the periodic uploader, which is not used in server-only tests.

When this is running (or if it fails to start), you can get the logs from the server run by opening a Terminal window and doing:

cd /Users/chris/Desktop/Apps/SyncServer.Run

tail -f output.log

# Pushing a server build to Dockerhub

This depends on you having carried out the “Building server” steps above. Then, do the following from the MacOS Terminal (not from within the Docker Terminal):

# Execute the following command in ServerMain folder:

./devops/release.sh <ReleaseTag>

# The intent of this is to tag a version on the github repo along with pushing an image to Docker hub

# Build server in Xcode

Make sure you have “My Mac” selected.

****

If you get failures, you may need to regenerate the Xcode project:

**./**generateXcodeproj.sh

# Run local tests

These need to run under Docker. See initial steps under “Building server” above to start the Docker build image.

./Tools/clean.sh

./Tools/runTests.sh filter DatabaseTests

# Or

./Tools/runTests.sh suites all

Test output logs are written into the .testing directory in the same place where you run the tests.

Tests use the ./ServerTests.json configuration file in the main directory (ServerMain). The same file is assumed to contain both the server configuration and test configuration keys—See ServerConfiguration and TestConfiguration.

See the file TestConfiguration.swift for instructions on how to renew or update the Google, Dropbox, Facebook etc. credentials that are accessed in that file. Once updated, the ServerTests.json file needs updating.

Use: multitail -Q 1 '.testing/\*'

To tail the logs

Most tests need:

static let primaryOwningAccount:TestAccount = .google1

SharingTests need:

static let primarySharingAccount:TestAccount = .google2

[spasticmuffin.louisville@gmail.com](mailto:spasticmuffin.louisville@gmail.com)

static let nonOwningSharingAccount:TestAccount = .facebook1

[open\_uwnytjn\_user@tfbnw.net](mailto:open_uwnytjn_user@tfbnw.net)

static let secondarySharingAccount:TestAccount = .google3

[spastic.muffin.biz@gmail.com](mailto:spastic.muffin.biz@gmail.com)

static let secondaryOwningAccount:TestAccount = .google2

AccountAuthenticationTests\_Dropbox

.dropbox1

.facebook1

.google1

.microsoft1

.apple1

To boostrap these account credentials, the Neebla-local.json configuration must be used to run a server to use the Neebla app to create credentials.

1) Delete the tables from the server database

2) Start the server using Neebla-local.json

3) Delete Neebla app from the device/simulator where you are running it.

4) Sign into Neebla with the account you want to bootstrap.

5) For sharing accounts, you must create an invitation from an owning account, and redeem this with the sharing account.

THEN: To run the server tests:

rm ./ServerTests.json

ln -s ../Private/Server/Neebla-local.json

mv Neebla-local.json ServerTests.json

For Facebook accounts, you need to use getLongLivedFacebookToken.sh to make long lived tokens.

./Tools/getLongLivedFacebookToken.sh ~/Developer/Private/Server/Neebla-local.json FacebookLongLivedToken1

# Running local tests: Server access using ngrok

Start your server, running locally. It uses port 8080 by default.

In a new terminal window on Mac OS, do:

~/bin/ngrok http 8080

**Starting mySQL for local running of the server**

# Checking the local database

In a Terminal window, do:

mysql -u crspybits -p

# Get the password from ServerTests.json

# When connected:

use SyncServer;

**Creating** ClientTesting-local.json for running the server.

# Updating Dockerfile’s for new Swift version

**E.g., update to Swift 5.3**

As of 5/30/20, I've moved to basing my Dockerfile on Apple's.

e.g., <https://github.com/apple/swift-docker/blob/38f179345ace24236d6c09de84e77d91384014cd/5.2/ubuntu/16.04/Dockerfile>

Generally:

<https://github.com/apple/swift-docker>

<https://hub.docker.com/_/swift/>

See devops/Docker/Building and devops/Docker/Runtime

**For the Build image**

I’m going to use:

<https://hub.docker.com/r/swiftlang/swift/tags>

These have the tags for the beta builds.

And also see: <https://swift.org/download/#snapshots>

And <https://swift.org/download/#releases> – which has Docker tags for releases.

Create the image based on the Dockerfile using (do this from a Terminal window opened within the devops/Docker/Building folder):

docker build -t swift-ubuntu:latest .

docker tag swift-ubuntu:latest crspybits/swift-ubuntu:latest

docker tag swift-ubuntu:latest crspybits/swift-ubuntu:5.3.1

docker push crspybits/swift-ubuntu:latest

docker push crspybits/swift-ubuntu:5.3.1

Run it with:

docker run --rm -i -t -v /Users/chris/Developer/:/root/Apps crspybits/swift-ubuntu:5.3.1

**For the Runtime image**

See <https://hub.docker.com/r/swiftlang/swift/tags>

Do this from a Terminal window opened within the devops/Docker/Building folder:

docker build -t swift-ubuntu-runtime:latest .

docker tag swift-ubuntu-runtime:latest crspybits/swift-ubuntu-runtime:latest

docker tag swift-ubuntu-runtime:latest crspybits/swift-ubuntu-runtime:5.3.1

docker push crspybits/swift-ubuntu-runtime:latest

docker push crspybits/swift-ubuntu-runtime:5.3.1

Run it, to get a command line:

docker run --rm -i -t -v /Users/chris/Developer/:/root/Apps crspybits/swift-ubuntu-runtime:latest

# Debugging a failing “slim” Swift image

I’m getting a failure when I run:

./devops/runLocally.sh ~/Developer/Private/Server/ServerTests.json latest

The log file just says:

Segmentation fault

I’m going to try running the build as if it was the build image. And try to run the server binary and see what happens.

docker run --rm -i -t -v /Users/chris/Developer/:/root/Apps crspybits/swift-ubuntu-runtime:latest

And that loads!

I’m attempting to debug by doing an `ldd` on the binary and then doing an ls on each of the libraries listed to see if one is missing.

No joy. All of those files appear present.

I’ve reported this issue here

<https://github.com/apple/swift-docker/issues/199>

Next time, try lldb

# Creating test data files for server

I’m adding:

static let catMov = TestFile(

dropboxCheckSum: "d342f6ab222c322e5fccf148435ef32bd676d7ce0baa72ea88593ef93bef8ac2",

md5CheckSum: "5edb34be3781c079935b9314b4d3340d",

sha1Hash: "41CA4AF2CE9C85D4F9969EA5D5C551D1FABD4857",

contents: .url(catMovURL),

mimeType: .mov)

into the test cases, but need to generate the checksums. The above is incorrect and just copy/pasted from an earlier example.

For SHA1, using: <http://onlinemd5.com>

Used that for MD5 too.

My iOSDropbox package (<http://github.com/SyncServerII/iOSDropbox.git>)

Has my hashing implementation for Dropbox.

You have to add the file into the testing target in Package.swift.

Add your new hash creation as a test case into the tests.

Connecting the Apple Server to server Notifications for Sign In to Apple

For testing,

In:

developer.apple.com > Account > Certificates, Identifiers & Profiles > Identifiers > Select your app identifier > Click 'Edit' next to 'Sign In with Apple' > Server to Server Notification Endpoint.

<https://d941e5cd6bfd.ngrok.io/AppleServerServerNotification/>

# Getting server working on iMac

1) Install Docker

Set it up so it launches if iMac reboots.

2) Get Neebla-local.json onto the iMac

Put it on the Desktop

3) Install mySQL

mysql Ver 14.14 Distrib 5.7.29, for osx10.15 (x86\_64) using EditLine wrapper

a) Install brew

https://brew.sh

b) Install mySQL

brew install [mysql@5.7](mailto:mysql@5.7)

server password:

h7!zm#x-3T9

To connect run:

    mysql -uroot -p

Create database SyncServer;

4) Install ngrok

5) git clone <http://github.com/SyncServerII/ServerMain.git>

6) Make the directory on the desktop:

Apps/SyncServer.Run

7) Pull the docker image:

docker pull crspybits/syncserver-runner:1.0.0

8) From within the ServerMain folder, do:

./devops/runLocally.sh ~/Desktop/Neebla-local.json 1.0.0

9) Run ngrok

ngrok http -region=us -hostname=neebla.ngrok.io 80

10) On developer.apple.com, use

<https://neebla.ngrok.io>

And use this in the Neebla .plist too.

In order for your Mac to respond to incoming connections while it is asleep, make sure to enable Wake for Network Access in System Preferences > Energy Saver.

(Already had this turned on)

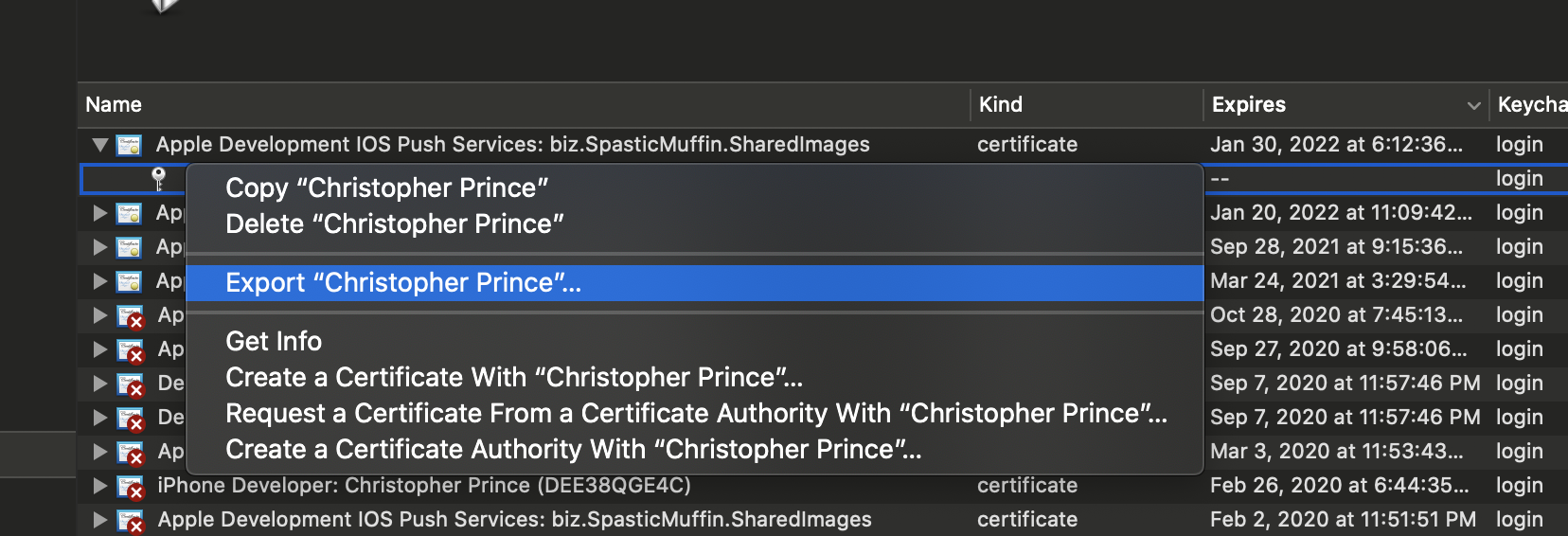
# Push Notification Certificate

First, have to generate the certificate at developer.apple.com

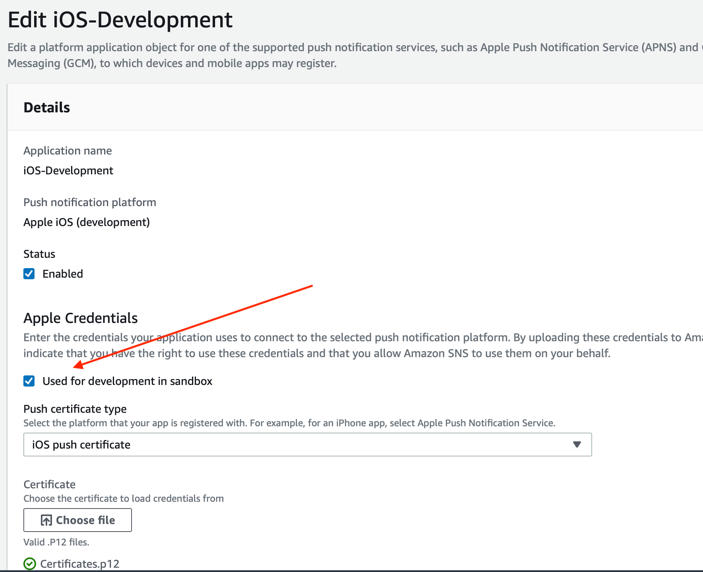
End up with a .cer file on Mac OS.

Then, need to upload to AWS SNS

Need to generate a .p12 file from the .cer



I uploaded the Certificates.p12 file



I’m testing with a build uploaded to TestFlight. I’m using the Sandbox certificate. Is that right? NO: You need to use the Production Certificate for TestFlight builds.

See also <https://stackoverflow.com/questions/24044298/why-push-notifications-is-not-working-on-testflight>