**Table of Contents**

[Starting a local server (on MacOS): Existing Docker container 2](#_Toc49546337)

[Building server 4](#_Toc49546338)

[Starting server from local build 5](#_Toc49546339)

[Build server in Xcode 6](#_Toc49546340)

[Run local tests 7](#_Toc49546341)

[Checking the local database 8](#_Toc49546342)

[Updating Dockerfile’s for new Swift version 9](#_Toc49546343)

[Debugging a failing “slim” Swift image 11](#_Toc49546344)

# 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 ~/Desktop/NewSyncServer/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 ~/Desktop/NewSyncServer/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/Desktop/NewSyncServer/:/root/Apps crspybits/swift-ubuntu:5.2.3

or

docker run --rm -i -t -v /Users/chris/Desktop/NewSyncServer/:/root/Apps crspybits/swift-ubuntu:5.3-beta

**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 ~/Desktop/NewSyncServer/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

# 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

# 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-beta

docker push crspybits/swift-ubuntu:latest

docker push crspybits/swift-ubuntu:5.3-beta

Run it with:

docker run --rm -i -t -v /Users/chris/Desktop/NewSyncServer/:/root/Apps crspybits/swift-ubuntu:5.3-beta

**For the Runtime image**

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

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-beta

docker push crspybits/swift-ubuntu-runtime:latest

docker push crspybits/swift-ubuntu-runtime:5.3-beta

Run it, to get a command line:

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

# Debugging a failing “slim” Swift image

I’m getting a failure when I run:

./devops/runLocally.sh ~/Desktop/NewSyncServer/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/Desktop/NewSyncServer/:/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