**Create React App with SASS, Storybook and Yarn in a Docker Environment**

So let's setup a workflow here that satisfies the following statements:

a) I would like to use [**create-react-app**](https://github.com/facebookincubator/create-react-app) because I have no desire to micromanage webpack OR boilerplates.

b) I would like to use [**SASS/SCSS**](http://sass-lang.com/) and also have it easily convert to CSS Modules if create-react-app ever includes them with scss support. Also I'm not interested in punishing myself with vanilla css.

c) I would like to use [**React Storybook**](https://getstorybook.io/) so that I can develop/test component UI's more easily and work with Product Managers and UX Designers.

d) I would like to use [**Yarn**](https://yarnpkg.com/), because npm is hilarious (npm-shrinkwrap is some cruel joke)

e) I'd like to develop in a [**Docker**](https://www.docker.com/) development environment so that I can transfer it to other machines and also not micromanage multiple versions of dependencies on my local machine.

All code can be found at this repo: [**cra-storybook-sass-yarn**](https://github.com/jcolemorrison/cra-storybook-sass-yarn/tree/cra-storybook-sass-yarn)

Also, if you're interested in learning about how to deploy this to AWS I've written a **very** extensive guide on it here:

[**Guide to Fault Tolerant and Load Balanced AWS Docker Deployment on ECS**](http://start.jcolemorrison.com/guide-to-fault-tolerant-and-load-balanced-aws-docker-deployment-on-ecs/)

The How

**1) Have**[**Docker**](https://www.docker.com/)**on your computer. Really straight forward.**

**2) Create a new directory called code. We'll work from here.**

**3) Create the file code/docker-images/cra-storybook/Dockerfile**

**4) Inside of this Dockerfile input the following code:**

# Start with Node

FROM node:6.9.4

# Install Yarn, because lolnpm

RUN curl -o- -L https://yarnpkg.com/install.sh | bash

# Make yarn available to SH, and thus your compose file

ENV PATH="/root/.yarn/bin:${PATH}"

# Add CRA and Storybook to your Dev Image

RUN yarn global add create-react-app && \

yarn global add getstorybook

# All operations that are run from on this image will assume

# this to be the directory the commands are run from

WORKDIR /usr/src/app/

Comments should give you a pretty good idea of what's going on inside there. This is going to let us run Node, Yarn, Storybook and CRA in it's own contained environment so that we don't have to put up with it cluttering up our local machine

Now to build it...

**5) Navigate to the file. Assuming you're in our code directory - docker build -t <username>/cra-storybook-dev ./docker-images/cra-storybook/**

This is going to pull down all of the image layers for our custom dev image. It'll take a minute or so.

Now we'll do the same thing for our SASS image...

*be sure to change <username> to your actual username*

**6) Create the SASS development Dockerfile at code/docker-images/sass-dev-image/Dockerfile and input the following code:**

FROM ruby:2.1

RUN su -c "gem install sass"

**7) Similar to step 5, now we're going to build the sass image. Assuming you're in our code directory run - docker build -t <username>/sass-dev ./docker-images/sass-dev-image/**

*be sure to change <username> to your actual username*

**8) In our code directory still, create an app directory.**

We'll put all of our create-react-app code here

**9) In our code directory, create a cmd.yml file and input the following code:**

version: '2'

services:

# the name of our service that will run the CRA, docker-compose will reference it as web

web:

environment:

NODE\_ENV: development

image: <username>/cra-storybook-dev # the image used for the service

ports:

- 3001:3000 # app will be at 3001

volumes:

- ./app:/usr/src/app # map our app directory to the volumes app working directory

# the name of our service that will run storybook, docker-compose will reference it as story

story:

environment:

NODE\_ENV: development

image: <username>/cra-storybook-dev # the image used for the service

ports:

- 3009:9009 # storybook will be at 3009

volumes:

- ./app:/usr/src/app # map our app directory to the volumes app working directory

# the name of our service that will run SASS, docker-compose will reference it as sass

sass:

image: <username>/sass-dev # the image used for the service

volumes:

- ./app:/usr/src/app # map our app directory to the volumes app working directory

We're defining 3 distinct services that we're going to work with and that will interact and be available to each other - web, story and sass. Although two of them (web and story) share the same image, that's exactly what their doing, their SHARING the same image. The only difference is their top writable layer (their container).

Alright that was a lot of words - it basically means, don't sweat it that we're using the same image twice. It's not duplicating it entirely or anything. Also, running storybook in the same container as the CRA clashes and this lets you keep an easy separation of concerns / logs while developing.

*be sure to change <username> to your actual username*

**10) Navigate inside of our app directory and run the following command:**

docker-compose -f ../cmd.yml run web create-react-app .

This will run the create-react-app command inside of your app directory and scaffold it out. Go grab a drink or something this can take a bit, especially if you're using Docker for Mac.

*note (a): if installation fails, try remakeing the app dir and running the command again, sometimes latency will mess things up*

*note (b): if you're on Docker for Mac, after all the installation, Docker might start running really slow. If it does, your best bet is to just close down the app and reopen it.*

**11) Inside of our app directory, run another command:**

docker-compose -f ../cmd.yml run web getstorybook .

This will setup our application to use React Storybook.

**12)** Navigate back to our root code directory and create a docker-compose.yml file. From this point on, this is mainly what you'll be interacting with when you're developing. Insert the following code:

version: '2'

services:

# the `extends` command references the `web` service in our

# cmd.yml file. We use everything from that one AND we run

# the command in this file.

web:

extends:

file: cmd.yml

service: web

command: yarn start

story:

extends:

file: cmd.yml

service: story

command: yarn run storybook

sass:

extends:

file: cmd.yml

service: sass

command: sass --watch /usr/src/app/src:/usr/src/app/src

Could we have just used this straight up? Sure, but I thought it'd be nice to show a Version 2 docker-compose extends pattern.

**13) Once this is done run docker-compose up -d. Run docker-compose logs -f to see when the different services are ready to go.**

Once they are you can navigate to localhost:3001 for the React Application and localhost:3009 for the React Storybook Application.

You can also use just use docker-compose logs -f <service>, for example docker-compose logs -f web, to just see the logs of one service.

Now to test out SCSS and our Stack in general.

**14) Navigate to your /app directory. Inside, rename App.css to App.scss. You should see a new App.css file created by our sass service.**

**15) Create a new file at /app/src/components/widget/index.js and input the following code:**

import React from 'react'

const Widget = () => {

return (

<div className='widget'>Yay we have SASS AND CRA AND STORYBOOK AND YARN!!!!</div>

)

}

export default Widget

Simple, simple, simple

**16) Create a new file at /app/src/components/widget/\_widget.scss (note the underscore, don't forget it) and put in the following code:**

.widget {

background-color: #CC7A6F;

color: white;

text-align: center;

padding: 20px;

width: 50%;

margin: 0 auto;

}

**17) In your App.scss import your \_widget.scss. App.scss should look like this:**

.App {

text-align: center;

}

.App-logo {

animation: App-logo-spin infinite 20s linear;

height: 80px;

}

.App-header {

background-color: #222;

height: 150px;

padding: 20px;

color: white;

}

.App-intro {

font-size: large;

}

@keyframes App-logo-spin {

from { transform: rotate(0deg); }

to { transform: rotate(360deg); }

}

@import "./components/widget/widget";

Before we finish up, quick aside here:

* We can now define variables just like you would in any other SASS project and use them in all the imports.
* Yes, you'll import any scss files you use here. You don't have to, but if you want to leverage global style variables, it's easier this way.
* If create-react-app goes full blown css-modules with scss, our scss files will already be in the right place!

finally..

**18) In your App.js import the Widget!**

import React, { Component } from 'react'

import logo from './logo.svg'

import './App.css'

import Widget from './components/widget/'

class App extends Component {

render() {

return (

<div className="App">

<div className="App-header">

<img src={logo} className="App-logo" alt="logo" />

<h2>Welcome to React</h2>

</div>

<p className="App-intro">

<Widget />

</p>

</div>

)

}

}

export default App

Now you can travel to your localhost:3001 and see your SCSS/CRA/YARN/DOCKER application! You can change any of the scss files and the changes are watched and updated by our sass service.

**19) When you're down developing, just do a docker-compose down in your root code directory and docker will take down all of the containers and networks for you.**

**20) If you want to build your react application, in your code directory just do a docker-compose run web yarn build.**

Now in your /app directory, you'll have a /build/ directory that will contain your fully built down react application (without storybook or scss files).

Notes

If you ever want to just command line into one of your containers, just do a docker-compose run <service> bash and you'll be able to work directly in the container.

If you're looking to run in Jest TDD mode, and are on Docker for Mac, there's an outstanding issue where mounted volumes run kind of slow:

[**https://forums.docker.com/t/file-access-in-mounted-volumes-extremely-slow-cpu-bound/8076/261**](https://forums.docker.com/t/file-access-in-mounted-volumes-extremely-slow-cpu-bound/8076/261)

It's not really apparent for anything short of Jest with this stack though. If you want to do TDD, I'd suggest just downloading yarn (or use npm lol) and just run your tests through that. The steps would be:

a) Install Yarn on your local machine  
b) in your /code/app folder run yarn test  
c) go about your business

Obviously, you'd want your docker-compose up -d stack running as well, but it's not required.

Without Docker

If you're bent on just doing this on your local machine, it's even simpler. Just install create-react-app, sass and storybook on your machine (hopefully via yarn). Create your react application, storybookify it and tell sass to watch your src folder with the command:

sass --watch /src

(assuming you're in you're in your /app directory)

Edit 2/4/2017

After some experimentation and frustration surrounding Docker for Mac and CPU % spiking over 200, it turns out the culprit is unfortunately SASS. This is also the result of both:

[**https://forums.docker.com/t/file-access-in-mounted-volumes-extremely-slow-cpu-bound/8076/261**](https://forums.docker.com/t/file-access-in-mounted-volumes-extremely-slow-cpu-bound/8076/261)

and

[**https://forums.docker.com/t/com-docker-hyperkit-up-cpu-to-340/13057/2**](https://forums.docker.com/t/com-docker-hyperkit-up-cpu-to-340/13057/2)

Hopefully those get improved some day soon...

The best solution I've wound up working with on a regular basis is:

a) Still use Docker and Docker Compose to do everything react related, yarn and storybook related. Remove the SASS component of the docker-compose file though.

b) Install rbenv, ruby and sass locally.

$ brew install rbenv

$ rbenv install 2.4.0 (or whatever you want)

$ rbenv global 2.4.0

$ gem install sass

Now we have access to the sass command so simply have it watch your src directory:

c) In the directory where we docker-compose up just run sass --watch ./app/src in another tab

Wherever you are in your directory, the goal is simply to tell the sass command to watch your code/app/src directory.

This will allow you to still run your commands and interact with node, yarn, webpack, etc versions WITHOUT clobbering other ones. However it will keep your CPU from spiking to hell and back again.

**YOUR TITLE HERE**