Deployment Packaging on Docker

This document details the instructions to set up a production server for the storyweaver application.This document details the instructions to set up a production server for the storyweaver application.

Author	Version/Date	Change
Manoj Sukhavasi	**V0.1** 13th Apr 2018	Initializing the document
Manoj Sukhavasi	**V0.2** 23rd May 2018	Added CI/CD flow and the steps required to sign up for the third party services.

Table of contents:

External Dependencies

- Google Analytics credentials
- Facebook Oauth credentials
- Google Oauth credentials and setup
- Google storage credentials
- Domain name with SSL setup
- Mailchimp credentials
- On the server need to open port 80 (http), 443(https), 8091(couchbase)

Setting up a new instance

First step is to set up a new instance with the recommended configuration. Recommended configuration is - Ubuntu 16.04 System with at least 8GB RAM (Preferable 16GB) and 4-cores(/threads).

Install Docker and Docker-compose

Intro

Docker is a tool designed to make create,deploy and run an application by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package. In a way, Docker is a bit like a virtual machine. But unlike a virtual machine, rather than creating a whole virtual operating system, Docker allows applications to use the same Linux kernel as the system that they're running on and only requires applications be shipped with things not already running on the host computer.

The docker containers are created from the images. For our application we created four images :

1. nginx - Runs the nginx application
2. postgres_elasticsearc - Runs the postgresql and elasticsearch
3. rails - Runs the rails application, puma server
4. couchbase - Runs the couchbase application

Docker has very good documentation , a good way to get started with docker is with https://docs.docker.com/get-started/ .

First thing we need to do to deploy out application is to install docker on the host system.

1. Install Docker from the instructions given in the link.
   https://docs.docker.com/install/linux/docker-ce/ubuntu/#install-docker-ce

```
$ sudo apt-get update
$ sudo apt-get install \
       apt-transport-https \
       ca-certificates \
       curl \
       software-properties-common

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
$ sudo add-apt-repository \
     "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
     $(lsb_release -cs) \
     stable"

 $ sudo apt-get update

 $ sudo apt-get install -y docker-ce
```

2. **Optional** - Install docker-compose on all the systems. If you want to use docker compose/stack this is required. Otherwise move on to next step.

```
 $ sudo curl -L https://github.com/docker/compose/releases/download/1.20.1/docker-compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose

 $ sudo chmod +x /usr/local/bin/docker-compose

 $ docker-compose --version
```

Check to see if installation is completed.

Initialize the swarm and volumes

A **swarm** is a group of machines that are running Docker and joined into a cluster. After that has happened, you continue to run the Docker commands you're used to, but now

they are executed on a cluster by a swarm manager. Multi-container, multi-machine applications are made possible by joining multiple machines into a swarm.

To enable swarm mode run the following command.This makes your current machine a swarm manager, then run docker swarm join on other machines to have them join the swarm as workers.

- `$ sudo docker swarm init`

Whenever docker container stops, the data doesn't persist. But we need to persist in some of the cases such as databases. Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. For our application we a volume **sw-pgdata**. The **sw-pgdata** is to persist the db data in the host.

- `$ sudo docker volume create sw-pgdata`

Docker networks connects different docker containers and opens the ports between them. We create a docker network type overlay to connect multiple Docker daemons together and enable swarm services to communicate with each other.
For more on networking read https://docs.docker.com/network/

- `$ sudo docker network create -d overlay --subnet 192.168.1.0/24 --gateway 192.168.1.100 sw-net`

Currently we are saving all the images we created on docker hub. So login in to that account to access the images. You need to ask the administrator to give you access to the images on the storyweaver docker hub account.

- `$ sudo docker login`

The following command is a configuration setting to run elasticsearch in production mode. https://www.elastic.co/guide/en/elasticsearch/reference/current/docker.html#docker-cli-run-prod-mode

- `$ sudo sysctl -w vm.max_map_count=262144`

Before starting up the production server you need to signup to the third party services and get the respective API keys to be used. Please follow these instructions to setup API keys

Deploy the services

Services are really just “containers in production.” A service only runs one image, but it codifies the way that image runs—what ports it should use, how many replicas of the container should run so the service has the capacity it needs, and so on. Scaling a service changes the number of container instances running that piece of software, assigning more computing resources to the service in the process.

Here we create four services for each of the images we mentioned above.

First we create a database service names sw-db connected to the network we created above, and connected the container to the volume sw-pgdata.

- `$ sudo docker service create --network sw-net --with-registry-auth --mount type=volume,source=sw-pgdata,destination=/var/lib/postgresql/9.3/main --constraint 'node.role==manager' --name sw-db carmageddon888/storyweaver:postgres_elasticsearch-1.0`

Then we create a couchase service named sw-couchbase connected to sw-net and opens a port 8091 to outside to configure anyy settings we might need.

- `$ sudo docker service create --network sw-net --with-registry-auth -p 8091:8091 --name sw-couchbase carmageddon888/storyweaver:couchbase-1.0`

Next we start the rails service where we need to provide the HOST_IP (Ip address or DNS name of the website) along with other API keys and parameters in the env.list file. We also connect a bind mount to save the logs and system files.

- Checkout deploy code from https://github.com/kuchlous/sw-deploy-r2r
- Create a directory <shared-dir>.
- Create 3 sub directories named ‘log’, ‘system’ and ‘tmp’ in <shared-dir>
- Replace source=/shared with source=<shared-dir> in the commands below. This directory is bind mounted to the /shared directory in docker container.
- cd build
- Populate the env.lst (get it from the deploy repo) with appropriate values.

- `$ sudo docker service create --network sw-net --env-file ./env.list --with-registry-auth  --mount type=bind,source=/shared,destination=/shared --name sw-rails carmageddon888/storyweaver:rails-1.0`

Finally we start a nginx service opening a port 80. We need to specify the HOST_IP as before in rails service.

- `$ sudo docker service create --network sw-net -e HOST_IP=’’ --with-registry-auth --mount type=bind,source=/shared,destination=/shared -p 80:80 --name sw-nginx carmageddon888/storyweaver:nginx-1.0`

Go inside nginx docker and update the SSL details.

Running the Containers (no Jenkins)

1. Set appropriate variables in the build/env-cmd.list, and deploy/env-cmd.list.
2. Set appropriate variables in build/config-files, search for ENV and replace with your appropriate values.
3. Checkout the rails repo (https://github.com/kuchlous/sw-r2r) in build/spp and reactjs repo (https://github.com/kuchlous/sw-js-r2r) into build/spp/react (Note that these need to be git repos as the build step relies on that).

- git clone https://github.com/kuchlous/sw-whitelabel-production.git spp
- cd spp
- git clone https://github.com/kuchlous/sw-js-whitelabel-production.git react

Build

Set the FILEPATH in build/build.sh

- cd build
- ./build.sh

Sometimes due to a bug is rails the build step fails at precompile stage. Just fire it again and it will go through.

Deploy

Set the BUILD_PATH, DEPLOY_PATH and VERSIONS_PATH in deploy.sh

- cd deploy
- ./deploy

The first time you will need to login into the sw-prod-build container and fire the db:create command. Just copy paste from start.sh where it is commented out. After db:create, run start.sh again. This is needed only the first time.

Setting up CI/CD Architecture

Jenkins

Jenkins is an open source automation server used in software development process for continuous integration and delivery. Documentation can be found at https://jenkins.io/doc/

We use an official jenkins docker image, hence no need to create an image explicitly.

Since we want to save the configuration and settings for jenkins we'll use a volume to persist data.

```
$ sudo docker volume create sw-jenkins-home
```

Then you can startup the jenkins server using docker with the following command:

Single container

```
$ sudo docker run --name sw-jenkins -p 8080:8080 -v
sw-jenkins-home:/var/jenkins_home jenkins:2.60.3
```

Or

Running it as a Service

```
$ sudo docker service create -p 8080:8080 --mount
type=volume,source=sw-jenkins-home,destination=/var/jenkins_home --name
sw-jenkins jenkins:2.60.3
```

You can open the jenkins GUI on the http://$host_ip:8080. When you open the jenkins for the first time 'Administrator' user is already created and password code is visible on the console. Then select the custom plugins to install and add the following plugins to the already selected default plugins:

1. copy artifacts
2. publish over ssh
3. multiple SCM's

Then create an user profile for regular usage.

Now having Jenkins setup we need to create two projects on it : one is '*build*' and another is '*deploy*'.

'build' is for testing and building the code. We connect out github repo's here and run the test cases and build the code base to make it ready for deployment.

'*deploy*' is take the code which was tested and build in the *'build'* phase and deploy it to the production server.

The scripts required for the jenkins '*build*' and '*deploy*' can be found at whitelabel-docker-deploy.

Configuring build step

Follow the below steps to configure the sw-build.

1. Create a new freestyle project and name it **build**.
2. Open the configure window of the **build**.
3. Under Source Code management window, select '*multiple SCM's*' and add the details for the both repositories(spp and sw-js), credentials and branch.
4. Checkout the sw-js to a subdirectory called react.
5. Under the 'Build Environment' window select 'send files over ssh'. Fill in the details of the server used for building and testing (Which is the current server hosting jenkins.). We need to add the jenkins ssh key to the authorized_keys of the host server. Please generate the ssh-keys for the jenkins container using the following commands:
    a. First login in to the jenkins container using 'sudo docker exec -it sw-jenkins bash'
    b. Use the command 'ssh-keygen' and create the ssh keys which should be added to the host server.
    c. In the jenkins GUI you need to go to configure jenkins and add in the private key and the host server credentials.

6. Setup a folder **build** in the host server with the deployment scripts from the *whitelabel-docker-deploy* repo.
7. Modify build/config_files/fog.yml to provide your google storage credentials.
8. Modify build/config_files/production.yml to replace ENV with appropriate values.

9. Then add this command to the exec command column: `IMAGE=spp_build_test CONTAINER=spp_testing VOLUME=sw-jenkins-home FILE_PATH=/*/build bash /*/build/jenkins_test.sh`
10. `In advanced options increase timeout by 100 times.`
11. Under the post build actions archive the artifacts ‘pkg/*.tgz’.

Configuring deploy step

Follow the below steps to configure the sw-deploy:

1. Create a new freestyle project and name it sw-deploy.
2. Open the configure window of the sw-deploy..
3. Under Build window select copy artifacts‘ and select the ‘build’ and artifact path to copy in to this workspace.
4. Setup a *deploy* folder with the scripts form whitelabel-docker-deploy.
5. Next add another build step to copy the packed spp to the production server using “publish over ssh” in to the *deploy* folder setup in the production server. Add “pkg/*” to source files and set remote directory to /whitelabel-docker-deploy/versions/
6. Next Run the deployment process through following command : `IMAGE=sw-prod-build:0.1 NGINX_IMAGE=nginx-prod:0.1 FILE_PATH=/home/deploy VERSION_PATH=/home/deploy/versions bash /home/deploy/jenkins_deploy.sh`
7. `In advanced options increase timeout by 100 times.`

Third Party services - Sign Up

Facebook Login API

This is for setting up the login with facebook account. Please follow the instructions given in the following link.
https://developers.facebook.com/docs/facebook-login

Google Login API

This is for setting up the login using google account. Please follow the instructions given in the following link.
https://support.google.com/googleapi/answer/6158849?hl=en

`In addition creating a clientId and secret, you will also need to turn on Google+ API access. This is available from the API dashboard in Google console.`

GA Tracking ID

This is for tracking the analytics using Google Analytics. Please follow the link to setup the GA.
`https://support.google.com/analytics/answer/1008080?hl=en`

MailChimp and Mandrill App

First Open https://mandrillapp.com/login/ and then click on login through mailchimp, create an account in Mailchimp and add mandrill app in the account settings by following `https://kb.mailchimp.com/mandrill/add-or-remove-mandrill`

Then reopen mandrill app page and set up the Domain name, Go to Settings, And create an API key for login.

Setting Up New Relic

Sign Up for the New Relic to monitor the errors in the production environement. New relic sends the errors info the the setup mail adresses. You can follow the instructions at https://docs.newrelic.com/docs/accounts/install-new-relic/account-setup/create-your-new-relic-account .
Select the New Relic APM service after that signup and set the Rails agent as given in the service. You can also follow these instructions
https://docs.newrelic.com/docs/agents/ruby-agent/getting-started/introduction-new-relic-ruby

Sample env.list:

```
#Main
HOST_IP=https://uat.pbees.party
COUCHBASE_IP=http://172.16.97.56
ELASTICSEARCH_IP=sw-db
#For facebook and google sign-ins.
FACEBOOK_APP_ID=
FACEBOOK_SECRET_KEY=
GOOGLE_APP_ID=
GOOGLE_SECRET_KEY=
# for GA events
GA_PROPERTY_ID=
# for google translate
GOOGLE_TRANSLATE_APP_ID=
MAILCHIMP_API_KEY=
DEVISE_SECRET_KEY_BASE=
SECRET_KEY_BASE=
GOOGLE_STORAGE_ACCESS_KEY_ID=
GOOGLE_STORAGE_SECRET_ACCESS_KEY=
RAILS_PATH=/workspace/demo-build
REACT_PATH=/workspace/demo-build/react
```

NOTE: Before starting with Data Import, make sure to set following constants:
Settings.org_info.prefix
Settings.org_info.url
In these files:
<RAILS_APP_DIR>/config/settings/development.yml
<RAILS_APP_DIR>/config/settings/test.yml
<RAILS_APP_DIR>/config/settings/production.yml

Example:

```
org_info:
 prefix: "SW"
 url: "https://storyweaver.org.in"
```