Giacomo Vacca

Saturday, 24 January 2015

Dockerize a node.js WebSocket server in 5 minutes

Docker is an incredibly useful tool to build prototypes of Linux hosts and applications.

You can easily build a network of servers inside a single virtual machine, with each server represented by a docker container. Clients can access the services on the same IP address, but different ports.

In this post I'd like to talk about a common prototype case in WebRTC platforms: a WebSocket server. This will be a node.js server and will run inside a Docker container (hosted by an Ubuntu Trusty VM).

The server logic can be as complex as you can imagine, but since it's not the point of this post I'll keep it as simple as the server example in the node.js websocket module:

```
// Source: https://github.com/einaros/ws

var WebSocketServer = require('ws').Server
  , wss = new WebSocketServer({ port: 8080 });

wss.on('connection', function connection(ws) {
  ws.on('message', function incoming(message) {
    console.log('received: %s', message);
  });

  ws.send('something');
});
```

server.js hosted with ♥ by **GitHub** **view raw**

The WebSocket server will listen on port 8080, accept incoming connections, send back "something" upon client connection, and log the content of the messages from the clients.

We can assume all the files in this article are in the same folder, and we're cd into it. The server logic is inside a 'server.js' file.

As explained in this interesting post from Ogi, you can find docker images with node.js all set and ready to be used, but the purpose of this post is to go a level deeper and build our own image.

Let's create a Dockerfile like this:

```
FROM ubuntu:14.04

MAINTAINER Giacomo Vacca "giacomo.vacca@gmail.com"

ENV REFRESHED_AT 2015-01-19

RUN apt-get update
RUN apt-get upgrade -y
```

GV

About this blog

@giavac
Linkedin
Slideshare
github

Welcome! This is my personal blog about RTC technologies, software development, open source and other related topics.

The ideas expressed here are solely mine and don't represent those of others, either individuals or companies.
This is a non profit blog, so you won't find ads nor posts written under commission.
The code snippets or references to software products or analogous are to be used without any warranty of any kind.
If you enjoy the content, feel free to share it.
Comments are welcome, but will be published only after moderation.

Follow by Email

Email address... Submit

g+1 0

Search This Blog

Search

```
 9
10  RUN apt-get install -y nodejs
11
12  # needs this to find the nodejs exec
13  RUN ln -s /usr/bin/nodejs /usr/bin/node
14
15  RUN apt-get install -y npm
16  RUN /usr/bin/npm install ws
17
18  EXPOSE 8080
19
20  ENTRYPOINT ["/usr/bin/node", "/root/server.js"]
```

Dockerfile hosted with ❤ by **GitHub** view raw

Even if you're not familiar with Dockerfiles, I'm sure you find this self-explanatory. The tricky bits are on line 13, where we symlink the nodejs executable to the desired '/usr/bin/node' (see here why), and line 16, where we install the node.js ws module via the npm package manager.

Line 18 tells docker what port this container is expected to receive connections to.

Line 20, the ENTRYPOINT definition, tells docker what command to execute when running.

(Remember that a docker container will run as long as there's a running command in foreground, and will exit otherwise.)

From inside the same folder, we can build our container image with:

```
docker build -t gvacca/nodejs_ws .
```

'gvacca' is my username, and 'nodejs_ws' is an arbitrary name for this container. Note the '.', which tells docker where to find the Dockerfile. You've probably noticed I've run 'docker build' without 'sudo': for practical purposes I've added docker into the *sudo* group.

The command above, when run for the first time, generates about 1K lines of output; you can find in this gist an example.

I can see the image is available:

```
gvacca@my_vm:/home/gvacca/docker/nodejs_ws$ docker
images|grep nodejs_ws
gvacca/nodejs_ws            latest             332dae6a34f1
 4 minutes ago       493.1 MB
```

Time to run the container:

```
docker run -d -p 8080:8080 -v $PWD:/root gvacca/nodejs_ws
```

This is telling docker a few things:

1. Run the container in daemonized mode (-d)
2. Map the port 8080 on the host with port 8080 on the container (and

Blog Archive

- ▼ 2015 (5)
  - ► February (3)
  - ▼ January (2)
    - Dockerize a node.js WebSocket server in 5 minutes
    - Easy VPN setup accross multiple sites
- ► 2014 (18)
- ► 2013 (8)
- ► 2012 (8)
- ► 2011 (8)
- ► 2010 (12)
- ► 2009 (23)

Labels

debian
perl
linux
webrtc
puppet
docker
CPAN
SIP
VoIP
freeswitch
"Perl Best Practices"
MySQL
asterisk
dh-make-perl
tdd
ubuntu
unit testing
vim
Amazon
Hudson
Internet
Kindle
Open Source
RTP
WebSocket

yes, they can be different)
3. Create a VOLUME, which is a mapping between a folder on the host and a folder on the container (this is handy because allows you to change files without rebuilding the image)
4. Use the 'gvacca/nodejs_ws' image.

The reason why I don't need to specify a command to be executed is that this is already enforced by the Dockerfile with the ENTRYPOINT specification.

The container is up and running:

```
gvacca@my_vm:/home/gvacca/docker/nodejs_ws$ docker
ps|grep nodejs_ws
6ce3498a67e2        gvacca/nodejs_ws:latest
/usr/bin/node /root/    17 seconds ago       Up 16 seconds
0.0.0.0:8080->8080/tcp    ecstatic_feynman
```

```
gvacca@my_vm:/home/gvacca/docker/nodejs_ws$ sudo
netstat -nap |grep 8080
tcp6       0      0 :::8080                 :::*                    LISTEN
     18807/docker
```

Now, if you want to test quickly you can use this Chrome extension, which provides a GUI to instantiate a WebSocket connection and send and receive data through it, and play with it. The URL will be: 'ws://IP_ADDRESS:8080'.

You can also access the server's logs with:

```
docker logs 6ce3498a67e2
```

(where 6ce3498a67e2 is the first part of the container's unique identifier, as shown in the 'docker ps' output).

Once you have this in place, which takes much longer to describe than to do, you can start building your WebSocket server logic.

Posted by Giacomo Vacca at 10:08
Labels: docker, node.js, ubuntu, webrtc, WebSocket

No comments:

Post a Comment

Windows Mobile
astricon
audio codecs
digitalocean
git
kamailio
node.js
perlcritic
python
software development
AppStore
Build-Depends-Indep
C
CANCEL
Encapsulation
Erlang
Expectations
FOSDEM
Facebook
G.711
G.729
ITU
Italy
JsSIP
LAST_INSERT_ID
Lexicon
Literature
MS
Matrix
Microsoft
Mnesia
PHP
PHPUnit
POE
Programming
Putty
Putty Session Manager
RFC 3261
RSA

Enter your comment...

Comment as: Google Accou ▼

Publish Preview

Post a Comment

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

- Ruby on Rails
- SILK
- Script
- TAP::Harness::JUnit
- TLS
- Tahar Ben Jelloun
- Tate
- Tate Modern
- Test::Harness prove
- Test::More
- Verizon
- Windows MarketPlace for Mobile
- aliases
- apt-get
- base64
- bash
- books
- cell phone
- clarify
- cloud computing
- cluecon
- conffiles
- configuration management
- continuous integration
- curl
- data privacy
- debchange
- debian apt madison
- debian apt puppet
- debuild
- development
- diff
- dpkg
- dpkg info
- dpkg-buildpackage
- e-readers
- eclipse
- ejabberd
- emacs

- epic
- freeswitch dtls-srtp voip
- grep
- grep-status
- hackathon
- hashes
- https
- iPhone
- iPod Touch
- openvpn
- order of ignorance
- perl 6
- pjsip
- pjsua
- postinst
- project management
- prove
- reading
- regexp
- reprepro
- respoke
- security
- sh
- shell
- sipp
- sipsak
- sox
- speex
- svn
- tab completion
- truphone
- truphone labs
- twitter
- vi