**Giới Thiệu**

NodeJS tuy phát triển mạnh mẽ những năm gần đây nhưng tại Việt Nam vẫn còn ít lựa chọn để triển khai (deploy) lên Internet cho công chúng. Các dịch vụ “app engine” cho NodeJS của nước ngoài như [Heroku](https://www.heroku.com/), [Nodejitsu](https://nodejitsu.com/), [now.sh](https://zeit.co/now)… khó tiếp cận cho developer ở Việt Nam. Ngoài ra các dịch vụ trên còn nhiều hạn chế chẳng hạn: chưa có data center đặt tại Việt Nam, không kèm giải pháp database hoặc nếu có cũng là dạng add-on.

Để cài đặt một web app viết bằng NodeJS, cách đơn giản nhất và tiết kiệm chi phí nhất hiện nay chính là sử dụng **Cloud VPS**[1](https://int3ractive.com/2017/09/huong-dan-cai-dat-nodejs-app-len-vps-phan-1.html#fn:1). Cloud VPS là một server toàn quyền được ảo hóa trên một server vật lý, có tài nguyên và băng thông được dành riêng. Ở Việt Nam, hiện đã có nhiều nhà cung cấp dịch vụ Cloud VPS với chi phí khá tiết kiệm (cấu hình thấp nhất có chi phí khoảng 150k/tháng). (Vì đây là bài hướng dẫn kỹ thuật nên tôi sẽ không giới thiệu cụ thể nhà cung cấp nào).

Trong bài viết này, tôi sẽ hướng dẫn bạn cài đặt một NodeJS app (cụ thể viết bằng [KeystoneJS](http://keystonejs.com/)) lên một VPS **Ubuntu 16.04 LTS**[2](https://int3ractive.com/2017/09/huong-dan-cai-dat-nodejs-app-len-vps-phan-1.html#fn:2) với kết quả mong muốn như sau:

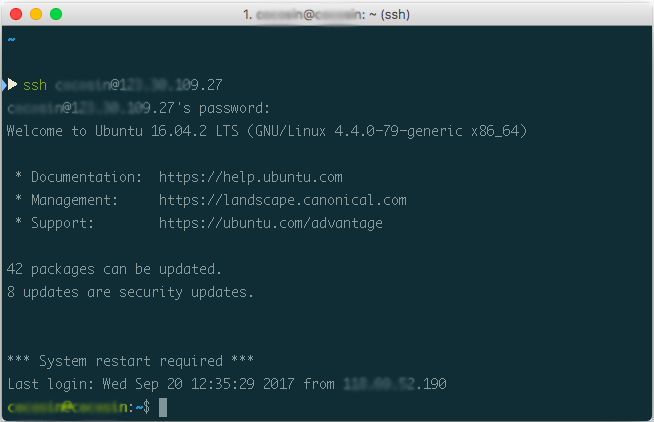
* Website tải cực nhanh
* Các tài nguyên tĩnh có cache header để tối ưu thời gian tải cho lần xem sau
* Các tài nguyên tĩnh được gzip để tối ưu băng thông
* Được mã hóa TLS với chữ ký số hợp lệ (miễn phí)
* Có thể cài đặt và chạy thêm app trên cùng server

**Cài Đặt Server**

Sau khi bạn tạo một Cloud VPS mới, thường thì nó sẽ ở trạng thái mặc định, tức là chưa có thêm bất cứ ứng dụng nào khác được cài thêm. Một số nhà cung cấp VPS thì có thể “nhanh nhảu” cài sẵn Apache HTTP server, tuy nhiên chúng ta sẽ không dùng Apache trong hướng dẫn này, nên có thể bạn phải cần [remove](https://askubuntu.com/questions/176964/permanently-removing-apache2) nó khỏi hệ thống.

Ngoài ra, tùy nhà cung cấp VPS mà bạn sẽ được cấp tài khoản **root** hoặc tài khoản user bình thường nhưng có quyền sudo. Trong trường hợp bạn có tài khoản root thì sẽ không cần chạy những dòng lệnh bắt đầu bằng sudo[3](https://int3ractive.com/2017/09/huong-dan-cai-dat-nodejs-app-len-vps-phan-1.html#fn:3).

Bạn sẽ truy cập vào server thông qua giao thức SSH và sẽ cấu hình server bằng dòng lệnh trong một cửa sổ terminal. Nếu bạn dùng Windows, tham khảo hướng dẫn kết nối SSH vào Linux server tại [đây](https://support.rackspace.com/how-to/connecting-to-linux-from-windows-by-using-putty/) và [đây](http://www.wikihow.com/SSH-to-a-Server)

*Giao diện dòng lệnh SSH terminal trên macOS*

Trước khi tiến hành cài đặt các phần mềm cần thiết, bạn cần cập nhật server Ubuntu với những package mới nhất:

sudo apt update && sudo apt upgrade -y

**Shell**

Copy

**Cài đặt NGINX và Git**

Đầu tiên chúng ta sẽ cài 2 phần mềm có sẵn trong repository của Ubuntu: **nginx** & **git**

sudo apt install -y nginx git

**Shell**

Copy

***NGINX****sẽ đóng vai trò reversed proxy và static file server để tiếp nhận request thông qua port mặc định 80 (http) và 443 (https).****Git****dùng để lấy source code của app để tiến hành build và deploy.*

**Cài đặt NodeJS VM:**

Tùy vào yêu cầu phiên bản Node của web app, bạn sẽ cài phiên bản Node engine tương ứng. Ở đây chúng ta sẽ dùng Node 8 LTS làm ví dụ.

Thêm repository source cho **NodeJS 8 LTS** [(link tham khảo)](https://nodejs.org/en/download/package-manager/#debian-and-ubuntu-based-linux-distributions):

curl -sL https://deb.nodesource.com/setup\_8.x | sudo -E bash -

**Shell**

Copy

Sau đó cài NodeJS lên server:

sudo apt install -y nodejs

**Shell**

Copy

*Hướng dẫn này chỉ sử dụng một phiên bản Node. Nếu có yêu cầu cài đặt nhiều app trên cùng một server và sử dụng nhiều phiên bản Node khác nhau, bạn cân nhắc cài đặt Node thông qua trình quản lý nhiều phiên bản Node như*[*nvm*](https://github.com/creationix/nvm)*hoặc*[*n package*](https://www.npmjs.com/package/n)

**Cài đặt MongoDB:**

KeystoneJS yêu cầu [database MongoDB](https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/). Đầu tiên là thêm PGP key cho MongoDB repository:

sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 0C49F3730359A14518585931BC711F9BA15703C6

**Shell**

Copy

Thêm repository source của MongoDB dành riêng cho Ubuntu 16.04:

echo "deb [ arch=amd64,arm64 ] http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list

**Shell**

Copy

Tiến hành cài MongoDB:

sudo apt update && sudo apt install -y mongodb-org

**Shell**

Copy

Khởi động dịch vụ mongod lần đầu tiên và đăng ký để nó tự chạy lúc restart server:

*# Khởi động mongod service*

sudo systemctl start mongod

*# Bật chức năng tự chạy khi restart Ubuntu*

sudo systemctl enable mongod

**Shell**

Copy

**Tất cả trong một**

Trên đây là hướng dẫn từng bước để các bạn hiểu rõ mình đang làm những gì để khởi tạo server mới. Một khi đã hiểu rõ, bạn có thể viết tất cả những dòng lệnh đó vào một file bash script như [ở Gist này](https://gist.github.com/trongthanh/e05524446249ad7727b78941f0e93a01) và thực thi nó:

*# Thêm quyền thực thi cho file bash script*

chmod +x initialize.sh

*# Chạy file bash script này*

./intialize.sh

**Shell**

Copy

**Tải mã nguồn của app từ Git host và build app**

Trong hướng dẫn này, chúng ta sẽ lấy mã nguồn của app từ một trong 3 Git host phổ biến là Github, Bitbucket, và Gitlab và build ngay tại server.

Trước tiên, chúng ta sẽ tạo khóa SSH mặc định trên server bằng các bước sau (Enter mặc định với tất cả các câu hỏi):

ssh-keygen -t rsa -b 4096 -C "your\_email@example.com"

**Shell**

Copy

Xuất nội dung của public key ra terminal để copy:

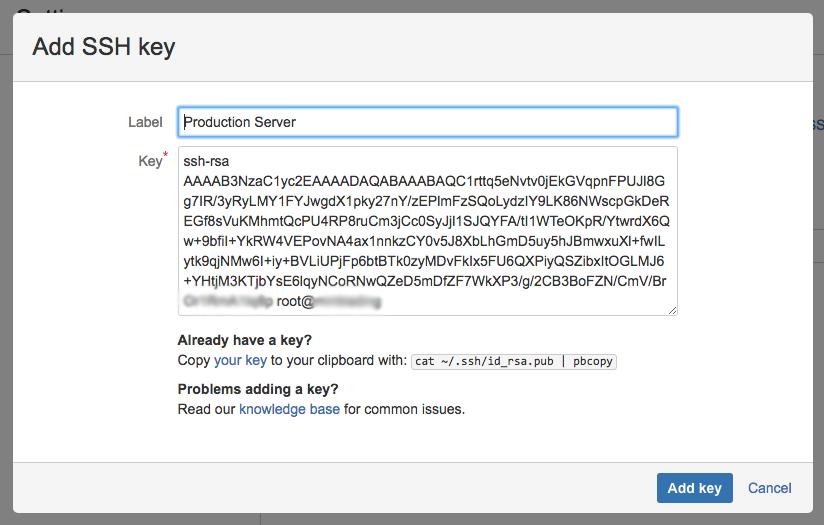
cat ~/.ssh/id\_rsa.pub

**Shell**

Copy

Tại trang settings của project trên Git host, thêm deploy key và paste nội dung của file id\_rsa.pub vừa mới copy ở trên.

* Tại Bitbucket, vào Settings > Chọn tiếp Access Keys
* Tại Github, vào Settings > Chọn Deploy Keys
* Tại Gitlab, vào Settings > Repository > Deploy Keys

*Ảnh chụp popup thêm Access key của Bitbucket.*

Quay trở lại terminal của server, tạo thư mục để chứa mã nguồn của app sẽ được clone vào:

sudo mkdir -p /apps/my-node-app

*# Đổi owner của thư mục về user hiện tại để tiện chạy các lệnh sau đó*

sudo chown -R $USER /apps/my-node-app/

**Shell**

Copy

Lệnh ở trên sẽ tự động tạo 2 cấp thư mục /apps/my-node-app tại root.

*Theo quy ước, các app 3rd-party cài trên Linux thường được cài tại thư mục /opt. Tuy nhiên, đây là ứng dụng đặc biệt do chúng ta viết riêng nên chúng ta sẽ cài vào /apps để tách bạch.*

Tiếp theo, clone Git repo của app vào thư mục vừa tạo (lưu ý sử dụng lệnh clone với giao thức SSH).

git clone git@bitbucket.org:<username>/<repo-name>.git /apps/my-node-app/

**Shell**

Copy

*Với bản Gitlab mới nhất, bạn còn một lựa chọn nữa để lấy source từ Gitlab đó là dùng*[*Deploy Token*](https://docs.gitlab.com/ee/user/project/deploy_tokens/)*. Khi đó, URL để clone source có dạng: https://<username>:<deploy\_token>@gitlab.com/user/my-node-app.git*

**Build app**

Với một Node.js app chuẩn, việc đầu tiên chúng ta cần làm là **cd** vào thư mục gốc của project, và chạy lệnh cài tự động các dependency package được liệt kê trong **package.json**:

npm install

**Shell**

Copy

Sau đó, tùy vào cài đặt của dự án, ta cần chạy các lệnh để build các thành phần cần biên dịch hoặc tối ưu hóa từ mã nguồn.

Theo thông lệ chung của các NodeJS app, việc build project sẽ thông qua một lệnh script được config sẵn trong package.json, VD:

{

"scripts": {

"build": "gulp build"

}

}

**JSON**

Copy

Do đó, việc tiếp theo sẽ là chạy script build này bằng npm:

npm run build

**Shell**

Copy

Tới đây, chúng ta đã có thể chạy thử app bằng lệnh node keystone.js (giả sử keystone.js là điểm start của app) và preview tại IP của server và port mặc định 3000 (VD: http://12.34.56.789:3000).

Với những bước cài đặt vừa rồi, bạn đã có thể chạy app cho môi trường TEST hoặc STAGING và có thể demo với khách hàng. Để giữ cho app demo tiếp tục chạy sau khi thoát SSH, bạn có thể dùng dòng lệnh screen (xem [hướng dẫn](https://askubuntu.com/questions/904373/how-to-use-screen-command)).

Trong phần tiếp theo tôi sẽ hướng dẫn chạy app như service, đưa ra ngoài port chuẩn của web, bật mã hóa TLS/SSL… để sẵn sàng cho Production.

**Hạn chế quyền truy cập của app**

Đây là bước không bắt buộc, tuy nhiên tôi khuyến cáo không nên bỏ qua, nhất là bạn đang truy cập vào VPS bằng tài khoản root. Chúng ta sẽ tạo một Linux user mới và gán cho thư mục của app.

*# Tạo một Linux user mới, nên dùng tên của app và không để khoảng trắng*

sudo useradd my-node-app

*# Chuyển quyền sở hữu (owner) thư mục app và các thư mục con cho user mới*

sudo chown -R my-node-app /apps/my-node-app

**Sh**

Copy

Trong bước tiếp theo, khi chạy app như dịch vụ, app sẽ được chạy với quyền của user mới này để hạn chế quyền truy cập của app lên các thư mục hệ thống.

**Chạy app như dịch vụ**

Tiếp theo chúng ta sẽ cài đặt để chạy app như dịch vụ và tự động chạy lại khi server được restart. Có một số tutorial hướng dẫn chạy app với [**pm2**](https://www.digitalocean.com/community/tutorials/how-to-set-up-a-node-js-application-for-production-on-ubuntu-16-04), nhưng hôm nay tôi sẽ hướng dẫn dùng công cụ quản lý ứng dụng dịch vụ **systemd** trên Ubuntu 16 và các HĐH Linux mới.

Đầu tiên, chúng ta sẽ chạy một số lệnh để lấy thông tin cho config systemd:

*# Lấy đường dẫn tuyệt đối đến node VM*

which node

*# -> /usr/bin/node*

*# Lấy đường dẫn tuyệt đối đến thư mục của app*

pwd

*# -> /apps/my-node-app*

*# Lấy thông tin user hiện tại (bỏ qua nếu chúng ta đã tạo user ở trên)*

id

*# -> uid=0(root) gid=0(root) groups=0(root)*

**Shell**

Copy

Tiếp theo, tạo file service config cho systemd:

sudo nano /etc/systemd/system/my-node-app.service

**Shell**

Copy

* nano là editor trên terminal dễ dùng nhất. Bạn có thể thay nano bằng vim nếu nó là editor quen thuộc với bạn hơn.
* Tên file my-node-app.service là do bạn đặt nhưng cần giữ nguyên đuôi .service

Paste vào editor nội dung sau:

[Unit]

Description=Thay bằng nội dung mô tả app của bạn

Requires=mongod.service

After=mongod.service

[Service]

ExecStart=/usr/bin/node /apps/my-node-app/keystone.js

WorkingDirectory=/apps/my-node-app

Restart=always

StandardOutput=syslog

StandardError=syslog

SyslogIdentifier=my-node-app

User=my-node-app

Group=my-node-app

Environment=PORT=3000

Environment=MONGO\_URL=mongodb://localhost:27017/my-node-app

[Install]

WantedBy=multi-user.target

**Ini**

Copy

Một vài điều lưu ý khi chỉnh sửa file .service trên:

* Hai thuộc tính Requires và After để cho systemd biết chỉ kích hoạt dịch vụ này khi service chỉ định tại Requires và After đã được chạy (Trong trường hợp server phải restart và có rất nhiều service cần chạy lúc khởi động). Trong ví dụ này, mongod.service là tên service của MongoDB khi được cài như hướng dẫn trong phần một.
* ExecStart là nơi thực thi lệnh cho dịch vụ, và yêu cầu đường dẫn phải tuyệt đối.
* WorkingDirectory là thư mục ngữ cảnh khi lệnh được thực thi.
* SyslogIdentifier là tên định danh của dịch vụ trong logger hệ thống, chúng ta sẽ dùng nó để lọc các output hoặc báo lỗi của app như trong phần tiếp theo.
* User và Group là tên của Linux user mà chúng ta muốn app sử dụng khi chạy. Như đã nói ở trên, bạn có thể dùng user mới tạo là my-node-app hoặc tên của user trả về ở dòng lệnh id.
* Environment là nơi để bạn gán các biến môi trường lúc thực thi app. Với Keystonejs thì có thể bạn không cần sử dụng thuộc tính này vì nó sử dụng .env (dotenv).
* Những thuộc tính còn lại nên để giá trị như ví dụ.

Sau khi bạn đã tạo xong file .service, bạn có thể thử chạy app ngay thông qua lệnh sau:

*# Bắt đầu chạy app như dịch vụ (không cần đường dẫn đến file .service)*

sudo systemctl start my-node-app.service

*# Kiểm tra trạng thái app đang chạy hay không*

sudo systemctl status my-node-app.service

*# -> Nếu app chạy thành công, output sẽ có dòng: ...Active: active (running)...*

**Sh**

Copy

Cuối cùng, sau khi app đã được chạy thành công và bạn có thể vào qua port 3000, bật chức năng dịch vụ tự khởi động khi server được restart:

sudo systemctl enable my-node-app.service

*# -> Created symlink from ... to ...*

**Sh**

Copy

**Proxy app ra cổng 80 bằng NGINX**

**Nginx** sẽ đóng vai trò *reversed proxy* và *static file server*. Nó sẽ tiếp nhận request từ ngoài Internet thông qua port mặc định 80 (http) và 443 (https) và forward request qua port của app là 3000.

Đối với Ubuntu[1](https://int3ractive.com/2018/09/huong-dan-cai-dat-nodejs-app-len-vps-phan-2.html#fn:1), 2 file config chính của Nginx nằm ở: /etc/nginx/nginx.conf (config toàn server) và /etc/nginx/sites-available/default (config cho từng web host ảo[2](https://int3ractive.com/2018/09/huong-dan-cai-dat-nodejs-app-len-vps-phan-2.html#fn:2)).

Bước này yêu cầu bạn đã cấu hình DNS của domain trỏ đến địa chỉ IP tĩnh của VPS (VD: mynodeapp.com). Sau đó bạn vào chỉnh sửa file /etc/nginx/sites-available/default, thay toàn bộ nội dung mặc định bằng mẫu bên dưới.

server {

listen 80 default\_server;

listen [::]:80 default\_server;

*# File mặc định khi vào thư mục*

index index.html index.htm;

*# Điều chỉnh kích thước gói upload tối đa*

client\_max\_body\_size 25M;

*# Domain name của web app, có thể một hoặc nhiều domain cùng trỏ đến*

server\_name mynodeapp.com www.mynodeapp.com;

*# Forward toàn bộ request sang web app*

location / {

*# Thay đổi port nếu node-web-app chạy trên port khác 3000*

proxy\_pass http://127.0.0.1:3000;

proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection 'upgrade';

proxy\_set\_header Host $host;

proxy\_cache\_bypass $http\_upgrade;

}

}

**nginx**

Copy

Nếu muốn dùng cùng config cho nhiều web host ảo, bạn có thể chuyển một số directive ở trên như index, client\_max\_body\_size vào trong config toàn server tại /etc/nginx/nginx.conf.

Sau khi chỉnh sửa xong Nginx config như trên. Bạn có thể thử kiểm tra config mới có hợp lệ hay không bằng lệnh:

sudo nginx -t

*# -> nginx: the configuration file /etc/nginx/nginx.conf syntax is ok*

*# -> nginx: configuration file /etc/nginx/nginx.conf test is successful*

**Sh**

Copy

Nếu config mới OK, bạn khởi động lại Nginx bằng lệnh sau:

sudo systemctl restart nginx

**Sh**

Copy

Sau khi Nginx được restart, bạn vào thử website tại domain đã cài đặt ở trên với URL không thêm port (http://my-node-app.com). Nếu website hiện ra thì bạn đã cài đặt thành công. Nếu bạn thấy lỗi “Bad Gateway”, tức là cấu hình Nginx vừa rồi chưa thành công và cần phải rà soát lại.

**Serve file tĩnh bằng NGINX**

Một trong những lý do tôi hướng dẫn bạn sử dụng Nginx, ngoài việc để bật https, còn là để serve file tĩnh (static file) hiệu quả hơn. Nếu như các file tĩnh của web app được thiết kế sử dụng CDN thì bạn có thể không cần bước này.

Nginx serve file tĩnh cực nhanh với lượng kết nối đồng thời cao (concurrency). Việc bật header cache-ontrol với Nginx rất dễ dàng sẽ giúp tăng hiệu quả tải trang. Ngoài ra bạn còn có thể bật **gzip** khi serve file tĩnh, là một yêu cầu không thể thiếu khi tối ưu hóa việc tải trang từ phía server.

Để serve file tĩnh, chúng ta sẽ thêm bộ lọc location vào block server của host config ở trên. Đối với KeystoneJS, chúng ta có một thư mục file tĩnh mặc định đó là public và với ví dụ từ đầu đến giờ, đường dẫn đến thư mục này sẽ là /apps/my-node-app/public. Ngoài ra, nếu bạn có một thư mục để upload riêng và nằm ngoài thư mục public này, thì bạn cũng cần ghi lại đường dẫn để config như tiếp theo sau đây:

*# thêm directive location trong config /etc/nginx/sites-available/default*

server {

*# ...Các config khác đã ẩn...*

*# Hướng dẫn cho Nginx serve tĩnh các file và folder bên trong public,*

*# có tên bắt đầu bằng một trong những pattern như bên dưới*

location ~ ^/(fonts/|img/|javascript/|js/|script/|css/|stylesheets/|flash/|media/|static/|upload/|robots.txt|humans.txt|favicon.ico) {

*# đường dẫn tuyệt đối đến thư mục file tĩnh*

root /apps/my-node-app/public;

access\_log off;

*# bật cache-control lên với thời gian expire tối đa*

expires max;

}

*# Nếu bạn có thư mục upload bên ngoài, bạn cần thêm một directive `location` đến thư mục này*

*# Xem document của Nginx để biết thêm cách cấu hình location*

*# Đặt directive location / (app reversed proxy) ở dưới cùng*

location / {

*# ...*

}

}

**nginx**

Copy

Một vài vấn đề có thể gặp với việc serve file tĩnh bằng Nginx:

* Nếu bạn bị lỗi 403 Forbidden: khả năng là Nginx chưa có quyền truy cập thư mục của app. Bạn cần thêm quyền truy cập cho thư mục public bằng lệnh sudo chmod o+rx -R public.

Để bật gzip, bạn có thể vào file /etc/nginx/nginx.conf, và bỏ comment (dấu #) ở những dòng config gzip như bên dưới:

http {

*# ...Các config khác đã ẩn...*

*##*

*# Gzip Settings*

*##*

gzip on;

gzip\_disable "msie6";

gzip\_vary on;

gzip\_proxied any;

*# gzip\_comp\_level 6;*

*# gzip\_buffers 16 8k;*

*# gzip\_http\_version 1.1;*

gzip\_types text/plain text/css application/json application/javascript text/xml application/xml application/xml+rss text/javascript image/svg+xml;

}

**nginx**

Copy

Trong danh sách các loại file tĩnh được nén với gzip ở trên (gzip\_types), tôi đã thêm SVG. Bạn có thể thêm các loại file khác nếu danh sách chưa có, nhưng không nên thêm các file ảnh bởi chúng đã được nén bằng thuật toán riêng và sẽ không hiệu quả khi nén tiếp với gzip.

**Lấy chữ ký số từ Let’s Encrypt và cài đặt cho NGINX**

Để một trang web được mã hóa an toàn thông qua giao thức HTTPS, bạn cần một chứng thư số (certificate) TLS/SSL để chứng thực cho những domain mà bạn sử dụng. TLS/SSL certificate phải được ký bởi một CA (Certificate Authority) hợp lệ, và trước đây chúng ta phải trả một khoản phí để đăng ký với CA. Tuy nhiên, từ bây giờ chúng ta đã có thêm [Let’s Encrypt](https://letsencrypt.org/) là một CA cung cấp TLS/SSL certificate hoàn toàn miễn phí.

Có nhiều cách để cài đặt certificate Let’s Encrypt nhưng đơn giản nhất trên Linux kết hợp với Nginx là sử dụng công cụ dòng lệnh [Certbot](https://certbot.eff.org/lets-encrypt/ubuntuxenial-nginx). Ở đây tôi sẽ tóm tắt các bước cần thực hiện:

*# Chạy các lệnh sau để cài đặt lệnh certbot kèm plugin Nginx trên Ubuntu*

sudo apt-get update

sudo apt-get install software-properties-common

sudo add-apt-repository ppa:certbot/certbot

sudo apt-get update

sudo apt-get install python-certbot-nginx

**Sh**

Copy

Sau khi **certbot** được cài, bạn có thể chạy lệnh sau để bắt đầu tiến trình cài đặt certificate cho website với Nginx plugin:

sudo certbot --nginx

**Sh**

Copy

Sau khi gọi lệnh trên, bạn sẽ nhập thông tin theo yêu cầu:

* Đầu tiên, certbot sẽ đọc Nginx config và liệt kê các tên miền hiện có. Bạn sẽ nhập vào con số gắn với tên miền (1, 2..) hoặc Enter bỏ trống để chọn tất cả.
* Certbot cũng sẽ hỏi bạn có muốn thêm Nginx config để tự động redirect từ HTTP sang HTTPS không[3](https://int3ractive.com/2018/09/huong-dan-cai-dat-nodejs-app-len-vps-phan-2.html#fn:3). Tùy vào yêu cầu dự án, nhưng khả năng cao là bạn nên chọn “2: Redirect”.
* Certbot cũng sẽ hỏi một số thông tin về email liên lạc và một số thỏa thuận khác.

Certbot sẽ tự động xác nhận quyền sở hữu domain thông qua Nginx server và domain của bạn đã được trỏ về IP của VPS trước đó. Certbot cũng sẽ tự động lưu certificate xuống VPS, cấu hình lại Nginx config và khởi động lại Nginx giùm bạn. Ngay sau khi kết thúc cài đặt với certbot, bạn đã có thể thử vào website với scheme https://.

Certificate mà Let’s Encrypt cấp cho bạn chỉ có thời hạn 3 tháng. Trước đây, khi gần đến thời hạn 3 tháng, bạn phải gọi lại sudo certbot renew để xin cấp lại certificate mới và phải khởi động lại Nginx. Giờ đây với Nginx plugin thì certbot sẽ tự động renew cho bạn và bạn không cần phải làm gì nữa cả sau khi hoàn tất bước này.

**Thêm: Xem log và thông báo lỗi của app**

Ngoài lệnh systemctl status để biết trạng thái chạy của app, với các chương trình chạy bằng systemd, bạn có thể xem log (cả console log và thông báo lỗi) của chúng bằng lệnh journalctl.

Sau đây là một số lệnh tôi thường dùng để xem lại console log của app trong lúc đang chạy:

*# Xem lại tất cả các log output của app*

*# Bạn còn nhớ `SyslogIdentifier` ở trên?*

*# Đặt chuỗi đó sau tham số -u để chỉ hiển thị log cho my-node-app*

sudo journalctl -u my-node-app

*# Nếu log quá nhiều, bạn có thể nhảy dòng bằng các phím tắt của vim*

*# Hiển thị 100 dòng log gần nhất*

sudo journalctl -n 100 -u my-node-app

*# Hiển thị log của app trong khoảng thời gian chỉ định*

sudo journalctl -u my-node-app --since "2018-09-20" --until "2018-09-26 03:00"

*# Hiển thị log gần đây nhất của app, sau đó tiếp tục chờ để*

*# hiển thị các log tiếp theo khi app đang chạy*

*# `-f` là tiếp tục chờ,*

*# `-o cat` là hiển thị log không có timestamp và id phía trước*

sudo journalctl -f -o cat -u my-node-app

**Sh**

Copy

**Lời kết**

Vậy là chúng ta đã hoàn tất cài đặt một Node Web app lên Ubuntu VPS với cách thức tối ưu nhất và tiết kiệm tài nguyên nhất. Với cấu hình này, bạn có thể chạy nhiều web app trên cùng VPS, bằng cách chạy app thứ hai trên một port mới (VD: 3001) và thêm một web host ảo trong Nginx config như ở trên, với server\_name là tên domain cho web app thứ hai…

Tuy nhiên đây là cách cài đặt gắn chặt với môi trường của HĐH Ubuntu. Nói như vậy để phân biệt với một số cách cài đặt sản phẩm phần mềm được cho là *hiện đại* hơn, sử dụng container, mà phổ biến nhất là [Docker](https://www.docker.com/). Theo tôi việc deploy sản phẩm bằng container sẽ giúp quy trình CI[4](https://int3ractive.com/2018/09/huong-dan-cai-dat-nodejs-app-len-vps-phan-2.html#fn:4) được thực hiện dễ dàng hơn, không còn phụ thuộc vào Linux distro, và developer có thể giao việc deploy hoàn toàn cho devops mà không phải bận tâm. Tuy nhiên nó đòi hỏi developer phải có thêm kiến thức khá sâu về devops cũng như cách sử dụng docker ở vai trò người tạo.

Docker vẫn là một thứ khá mới mẻ đối với tôi và tôi vẫn đang nghiên cứu cách thức triển khai sản phẩm Node app trên Docker. Nếu nó thật sự hiệu quả và dễ tiếp cận, tôi sẽ viết tiếp chủ đề này với container.

**Triển khai ứng dụng node js bằng PM2**

Node.js là một môi trường chạy JavaScript nguồn mở giúp bạn có thể dễ dàng xây dựng các ứng dụng server-side và networking. Nền tảng này chạy trên [Linux](https://tech.vccloud.vn/he-dieu-hanh-linux-la-gi-uu-diem-va-nhuoc-diem-cua-hdh-linux-20180704090927779.htm), OS X, FreeBSD và [Windows](https://tech.vccloud.vn/he-dieu-hanh-window-server-la-gi-chuc-nang-cua-window-server-20180703090659189.htm). Các ứng dụng Node.js có thể được chạy ở dòng lệnh, nhưng chúng ta sẽ tập trung vào việc chạy chúng như một dịch vụ, do đó chúng sẽ tự động restart lại khi reboot hoặc khi gặp sự cố, và các ứng dụng này có thể được sử dụng một cách an toàn trong môi trường production.

Trong bài viết hôm nay, chúng ta sẽ tìm hiểu cách thiết lập môi trường production-ready Node.js trên một máy chủ Ubuntu 16.04 đơn lẻ. Server này sẽ chạy ứng dụng Node.js được quản lý bởi PM2, và cung cấp cho người dùng quyền truy cập an toàn vào ứng dụng thông qua Nginx reverse proxy. Nginx server sẽ cung cấp HTTPS, sử dụng một chứng chỉ miễn phí do Let's Encrypt cung cấp.

## Điều kiện tiên quyết

Các điều kiện tiên quyết bạn cần phải đảm bảo trước khi tiến hành cài đặt:

- Một máy chủ Ubuntu 16.04, được cấu hình với non-root user quyền sudo.

- Một domain name trỏ về public IP của server. Hướng dẫn này sẽ sử dụng example.com làm ví dụ.

- Nginx đã được cài đặt.

- Nginx được cấu hình bằng SSL sử dụng Let's Encrypt certificate.

Khi đã hoàn tất các điều kiện tiên quyết, bạn sẽ sở hữu một server phục vụ cho trang Nginx mặc định tại *https://example.com/*.

Hãy bắt đầu bằng cách cài đặt runtime Node.js trên server.

## Cài đặt Node.js

Chúng ta sẽ cài đặt bản phát hành LTS mới nhất của Node.js, sử dụng [NodeSource](https://github.com/nodesource/distributions).

Trước tiên, cần phải cài đặt PPA NodeSource để có quyền truy cập vào nội dung. Hãy đảm bảo bạn đang ở trong thư mục chính và sử dụng curl để truy xuất tập lệnh cài đặt cho Node.js 6.x archives:

*$ cd ~*

*$ curl -sL https://deb.nodesource.com/setup\_6.x -o nodesource\_setup.sh*

Bạn có thể kiểm tra nội dung của tập lệnh này bằng nano:

*$ nano nodesource\_setup.sh*

Và chạy script dưới sudo:

*$ sudo bash nodesource\_setup.sh*

PPA sẽ được thêm vào cấu hình và local package cache sẽ được cập nhật một cách tự động. Sau khi chạy tập lệnh thiết lập từ nodesource, bạn có thể cài đặt gói Node.js giống như cách bạn đã làm ở trên:

*$ sudo apt-get install nodejs*

Nodejs package chứa nhị phân nodejs cũng giống như npm, do đó bạn không cần phải cài đặt riêng npm. Tuy nhiên, để một số npm package hoạt động (chẳng hạn như những package yêu cầu biên dịch mã từ nguồn), bạn sẽ cần phải cài đặt*build-essential* package:

*$ sudo apt-get install build-essential*

Vậy là Node.js runtime hiện đã được cài đặt và sẵn sàng chạy ứng dụng! Bạn hãy bắt tay viết một ứng dụng Node.js.

Lưu ý: Khi cài đặt từ PPA NodeSource, thực thi Node.js được gọi là nodejs, chứ không phải là node.

## Tạo ứng dụng Node.js

Chúng ta sẽ viết một ứng dụng Hello World đơn giản: trả về "Hello World" cho bất kỳ HTTP requests nào. Đây là ứng dụng mẫu giúp bạn thiết lập Node.js, bạn có thể thay thế bằng ứng dụng của riêng mình - chỉ cần đảm bảo rằng bạn sửa đổi ứng dụng để nó lắng nghe các địa chỉ IP và port thích hợp.

### Hello World Code

Đầu tiên, hãy tạo và mở ứng dụng Node.js để tiến hành các thao tác chỉnh sửa. Đối với hướng dẫn này, chúng ta sẽ sử dụng nano để chỉnh sửa ứng dụng mẫu có tên *hello.js*:

*$ cd ~*

*$ nano hello.js*

Chèn đoạn mã sau vào tập tin. Nếu muốn, bạn có thể thay thế highlighted port, 8080, ở cả hai vị trí (đảm bảo sử dụng non-admin port, tức là 1024 trở lên):

[](https://techvccloud.mediacdn.vn/2018/7/27/photo-1-1532665134269716391036.jpg)

Save và thoát ra.

Ứng dụng Node.js này chỉ đơn giản là lắng nghe địa chỉ được chỉ định (localhost) và port (8080), và trả về "Hello World" với 200 HTTP success code. Vì chúng ta đang nghe trên localhost, các remote clients sẽ không thể kết nối với ứng dụng của chúng ta.

### Kiểm tra ứng dụng

Để kiểm tra ứng dụng, đánh dấu hello.js thực thi:

$ chmod x ./hello.js

Và chạy chúng như sau:

$ ./hello.js

Output:

Server running at http://localhost:8080/

Lưu ý: Chạy ứng dụng Node.js theo cách này sẽ chặn các lệnh bổ sung cho đến khi ứng dụng được kết thúc bằng cách nhấn Ctrl-C.

Để kiểm tra ứng dụng, hãy mở một terminal session khác trên server và kết nối với localhost bằng curl:

*$ curl http://localhost:8080*

Nếu bạn thấy output sau chứng tỏ ứng dụng đang hoạt động đúng và đang lắng nghe trên địa chỉ và port thích hợp:

*Output*

*Hello World*

Nếu bạn không nhận được output thích hợp, hãy đảm bảo rằng ứng dụng Node.js đang chạy và được cấu hình để nghe trên địa chỉ và port thích hợp.Khi bạn chắc chắn nó đang hoạt động, hãy xóa ứng dụng bằng cách nhấn Ctrl C.

## Cài đặt PM2

Chúng ta sẽ cài đặt PM2, trình quản lý quy trình cho ứng dụng Node.js. PM2 giúp quản lý và thực hiện các ứng dụng một cách khá dễ dàng (chạy ứng dụng trong background dưới dạng một dịch vụ).

Chúng ta sẽ sử dụng npm, một package manager cho các mô-đun Node cài đặt với Node.js, để cài đặt PM2 trên server, hãy sử dụng lệnh sau:

*$ sudo npm install -g pm2*

Tùy chọn -g sẽ ra lệnh cho npm cài đặt module trên toàn hệ thống.

## Quản lý ứng dụng với PM2

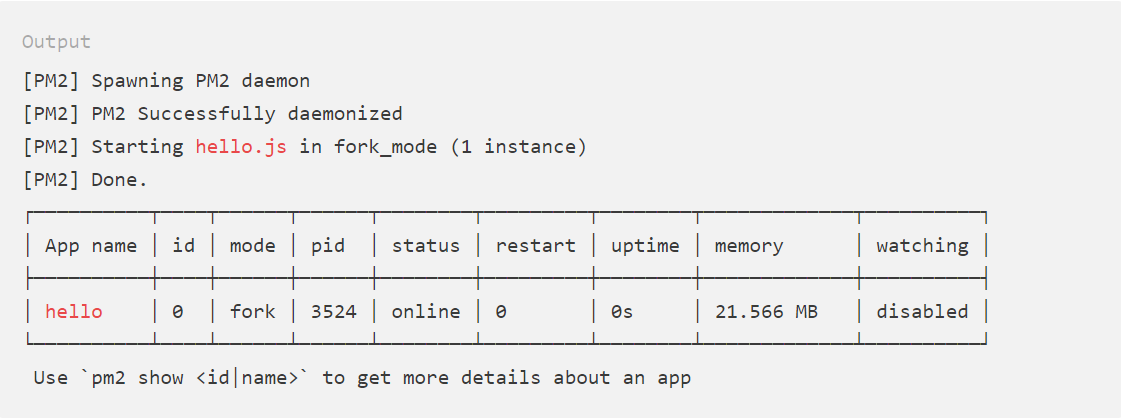
PM2 khá dễ sử dụng. Chúng ta sẽ tìm hiểu một vài điều cơ bản của PM2 ngay sau đây.

### Khởi động ứng dụng

Điều đầu, sử dụng lệnh pm2 start để chạy ứng dụng hello.js trong background:

*$ pm2 start hello.js*

Thao tác này sẽ thêm ứng dụng vào danh sách quy trình của PM2, được xuất mỗi khi bạn khởi động ứng dụng:

[](https://lh6.googleusercontent.com/rSIExvqWUX4I4fmUONR1uPSOKcdiHJ1Q2Ek76IP9UL6egyf1HJWH52NeKHfAc45rE7PllParDsHAZyNeO78rpqqnLSIhfsJC2fYYtZZGGnuh4vFtvnwbdYjJNB4aLx7lxptgQGP9)

Như bạn có thể thấy, PM2 tự động gán *App name* (dựa trên filename, không có phần mở rộng .js) và PM2 id. PM2 cũng duy trì các thông tin khác, ví dụ như PID của quá trình, trạng thái hiện tại và mức sử dụng bộ nhớ.

Các ứng dụng đang chạy dưới PM2 sẽ khởi động lại tự động nếu ứng dụng bị treo hoặc bị dừng hoạt động, nhưng bạn cần thực hiện thêm một bước để ứng dụng khởi chạy khi khởi động hệ thống. May mắn thay, PM2 cung cấp một cách dễ dàng để làm điều này: startup subcommand.

*startup* subcommand tạo và cấu hình một startup script để khởi động PM2 và các quy trình được quản lý của nó:

*$ pm2 startup systemd*

Dòng cuối của kết quả đầu ra sẽ bao gồm một lệnh mà bạn phải chạy với các đặc quyền của superuser:

*Output*

*[PM2] Init System found: systemd [PM2] You have to run this command as root. Execute the following command: sudo env PATH=$PATH:/usr/bin /usr/lib/node\_modules/pm2/bin/pm2 startup systemd -u sammy --hp /home/sammy*

Chạy lệnh đã được tạo (tương tự như output ở trên, nhưng với username thay vì sammy) để thiết lập PM2 và bắt đầu khởi động (sử dụng lệnh từ output của riêng bạn):

*$ sudo env PATH=$PATH:/usr/bin /usr/lib/node\_modules/pm2/bin/pm2 startup systemd -u sammy --hp /home/sammy*

Điều này sẽ tạo ra một systemd unit chạy pm2 cho user khi khởi động. Lần này, đến lượt pm2 chạy hello.js.

Bạn có thể kiểm tra trạng thái của systemd unit bằng systemctl:

*$ systemctl status pm2-sammy*

### Cách sử dụng PM2 (tùy chọn)

PM2 cung cấp nhiều subcommands cho phép bạn quản lý hoặc tìm kiếm thông tin về các ứng dụng. Lưu ý, khi chạy pm2 mà không có bất kỳ đối số nào, trang trợ giúp sẽ hiển thị.

Dừng ứng dụng bằng lệnh này (chỉ định PM2 App name hoặc id):

*$ pm2 stop app\_name\_or\_id*

Khởi động lại ứng dụng bằng lệnh này (chỉ định PM2 App name hoặc id):

*$ pm2 restart app\_name\_or\_id*

Danh sách các ứng dụng hiện đang được quản lý bởi PM2 cũng có thể được tra cứu với list subcommand:

*$ pm2 list*

Bạn có thể tìm thêm thông tin về một ứng dụng cụ thể bằng cách sử dụng info subcommand (chỉ định PM2 App name hoặc id):

*$ pm2 info example*

Màn hình quá trình PM2 có thể được kéo lên với monit subcommand. Điều này sẽ hiển thị trạng thái ứng dụng, CPU và mức sử dụng bộ nhớ:

*$ pm2 monit*

Đến đây ứng dụng Node.js đã hoàn tất khởi tạo và sẽ bắt đầu hoạt động và được quản lý bởi PM2, hãy thiết lập reverse proxy.

## Cài đặt Nginx như một Reverse Proxy Server

Bây giờ ứng dụng của bạn đang hoạt động và lắng nghe trên localhost, bạn cần thiết lập cách để người dùng có thể truy cập vào.

Do đó, chúng ta sẽ thiết lập webserver Nginx như một reverse proxy để đạt được mục đích này.

Trong hướng dẫn về các điều kiện tiên quyết, chúng ta đã thiết lập cấu hình Nginx trong tệp/etc/nginx/sites-available/default. Mở tệp để chỉnh sửa:

*$ sudo nano /etc/nginx/sites-available/default*

Trong server block, bạn nên sở hữu một location/ block. Thay thế nội dung của block đó bằng cấu hình sau. Nếu ứng dụng của bạn được cài đặt để lắng nghe trên một port khác, hãy cập nhật phần được đánh dấu với port number chính xác.

[](https://lh3.googleusercontent.com/NZd1lqR-cw-Yj_WkqNTq8wc33nnEv3nBpfw1izuc_tqYA0Emy5WUVEmK5xwT5OTw3NqyOlcTyNSQEyuwNUQ0pDvWwDUFMa8bHnknyF0D8tCr_K34OEqqYwjV5Ef1jVidAIO9zJpX)

Thao tác này nhằm cấu hình server để đáp ứng các requests tại root. Giả sử server của chúng ta có sẵn tại example.com, truy cập https://example.com/ thông qua web browser sẽ gửi requests đến hello.js, lắng nghe trên port 8080 tại localhost.

Bạn có thể thêm các location block bổ sung vào cùng một server block để cấp quyền truy cập vào các ứng dụng khác trên cùng một server. Ví dụ: nếu bạn cũng đang chạy một ứng dụng Node.js khác trên port 8081, bạn có thể thêm location block này để cho phép truy cập vào nó thông qua *http://example.com/app2*:

[](https://lh6.googleusercontent.com/fykvtHnio2oML2nN4DNOmrT-nfmnIVoWQ-oVcN--ntNJ6HcZo2fZb3OKpfdtmNC8n2n8MIfm99pcFGCCeZCg9G9lXEtK8K-YRe0x47iqMAsY3YayiYB1r4rvszXjlRO0Oxwazh_l)

Khi bạn đã hoàn tất việc thêm các location blocks cho các ứng dụng, hãy lưu và thoát ra.

Chắc chắn phải đảm bảo bạn tạo ra bất kỳ lỗi cú pháp nào bằng cách nhập:

*$ sudo nginx -t*

Tiếp theo, khởi động lại Nginx:

*$ sudo systemctl restart nginx*

Giả sử ứng dụng Node.js đang chạy, cấu hình ứng dụng và Nginx chính xác, vậy thì bây giờ bạn có thể truy cập ứng dụng thông qua Nginx reverse proxy. Hãy dùng thử bằng cách truy cập URL của server (public IP address hoặc domain name).

## Kết luận

Đến đây là bạn đã hoàn thành xong cài đặt. Vậy là bạn đã sở hữu một ứng dụng Node.js hoạt động phía sau một Nginx reverse proxy trên Ubuntu 16.04 server. Thiết lập reverse proxy này đủ linh hoạt để cung cấp cho người dùng quyền truy cập vào các ứng dụng khác hoặc nội dung web tĩnh mà bạn muốn chia sẻ. Chúc bạn may mắn với sự phát triển của Node.js!

**Triển khai ứng dụng bằng docker**

[Docker ra đời](https://blog.duyetdev.com/2015/12/docker-la-gi-co-ban-ve-docker.html#.VyYMd4N94_M) cho giải pháp đóng gói, vận chuyển và triển khai ứng dụng hết sức nhanh chóng và đơn giản. Với Docker, các thành viên trong team (cũng như với người muốn trải nghiệm thử project) sẽ triển khai ngay được môi trường ứng dụng mà không phải mất nhiều thời gian, công việc của SysAdmin cũng nhẹ nhàng hơn.

Sau mình sẽ hướng dẫn cách đóng gói build image docker cho ứng dụng Nodejs một cách cơ bản nhất.

Bài viết được viết tại blog của mình, gửi qua đây để mọi người cùng biết:   
[Đóng gói và triển khai ứng dụng Nodejs bằng Docker](https://blog.duyetdev.com/2016/05/dockerizing-nodejs.html#.Vyh96oN94_M)

[[](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/1-_MtS4HqN2srTcrSyet61DQ.jpeg_9m4awzzsek)](https://s3-ap-southeast-1.amazonaws.com/kipalog.com/1-_MtS4HqN2srTcrSyet61DQ.jpeg_9m4awzzsek)

(Ảnh: Airpair)

# Cài đặt Docker

Mình sử dụng Ubuntu, xem cách cài Docker trên Ubuntu (<https://blog.duyetdev.com/2016/05/cai-dat-docker-tren-ubuntu.html>) . Với các nền tảng khác, xem tại trang Guide tại trang chủ Docker (<https://docs.docker.com/engine/installation/>).

# Chuẩn bị ứng dụng cần đóng gói

Bạn chuẩn bị ứng dụng Nodejs cần đóng gói và triển khai. Ở đây mình sẽ sử dụng [source](https://github.com/saveto-co/saveto) của trang [https://saveto.co](https://saveto.co/) viết bằng Koajs làm ví dụ. Mã nguồn tải về ở đây: <https://github.com/saveto-co/saveto>

Mỗi ứng dụng viết bằng Nodejs bắt buộc đều phải có package.json, lưu thông tin về các gói cần thiết, nhiều thông tin khác về tên, phiên bản, ...

{

"name": "quick-koa",

"version": "1.0.0",

"description": "Quick by Koa",

"main": "app.js",

"scripts": {

"start": "node app.js"

},

"keywords": [

"Quick",

"shorten",

"link",

"jslab",

"duyetdev"

],

"homepage": "http://saveto.co",

"repository": "git://github.com/duyetdev/quick.git",

"author": "Van-Duyet Le <me@duyetdev.com>",

"license": "MIT",

"dependencies": {

"bcrypt-nodejs": "0.0.3",

"co": "^4.6.0",

...

"validator": "^4.8.0"

},

"devDependencies": {

"nodemon": "^1.8.1"

}

}

[saveto.co](https://saveto.co/) còn có thêm bower.json mô tả những thư viện frontend nào sẽ được sử dụng.

# Tạo Dockerfile

Dockerfile sẽ mô tả các bước để Docker build ảnh của ứng dụng, gồm các layer nào.

**touch** Dockerfile

Mở file Dockerfile bằng Editor. Đầu tiên, cần định nghĩa ứng dụng sẽ build trên image nào. Do saveto.co yêu cầu **Nodejs >= v4.0**, mình sẽ sử dụng bản **Nodejs 5.11**ổn định. Danh sách các image Nodejs có [tại đây](https://hub.docker.com/_/node/).

FROM node:5.11

Maintainer mô tả về người build docker image

MAINTAINER Van-Duyet Le <me@duyetdev.com>

Kế tiếp, ta tạo thư mục chứa source code trong Docker container:

# Create app directory

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

Docker Image gốc là Nodejs nên sẽ có sẵn NPM, tiếp mình sẽ copy package.jsonvào và cài đặt các package ứng dụng yêu cầu:

# Install app dependencies

COPY package.json /usr/src/app/

RUN npm install

Sử dụng COPY để đưa hết source code vào container:

*# Bundle app source*

**COPY** . /usr/src/app

Cài thêm Bower và chạy Bower để cài các thư viện frontend

# Install and run Bower

RUN npm install -g bower

RUN bower install

[saveto.co](https://saveto.co/) mặc định chạy ở port 6969, sử dụng EXPOSE để mở **PORT 6969** trên Container, thay 6969 bằng port ứng dụng của bạn:

**EXPOSE** 6969

Cuối cùng quan trọng nhất là lệnh để khởi động ứng dụng Nodejs, lệnh CMD sẽ chạy npm start, cũng như start node app.js được định nghĩa trong package.json

**CMD** [ "npm", "start" ]

Cuối cùng được Dockerfile hoàn chỉnh như thế này:

FROM node:5.11

MAINTAINER Van-Duyet Le <me@duyetdev.com>

# Create app directory

RUN mkdir -p /usr/src/app

WORKDIR /usr/src/app

# Install NPM

COPY package.json /usr/src/app/

RUN npm install

# Bundle app source

COPY . /usr/src/app

# Install and run Bower

RUN npm install -g bower

RUN bower install

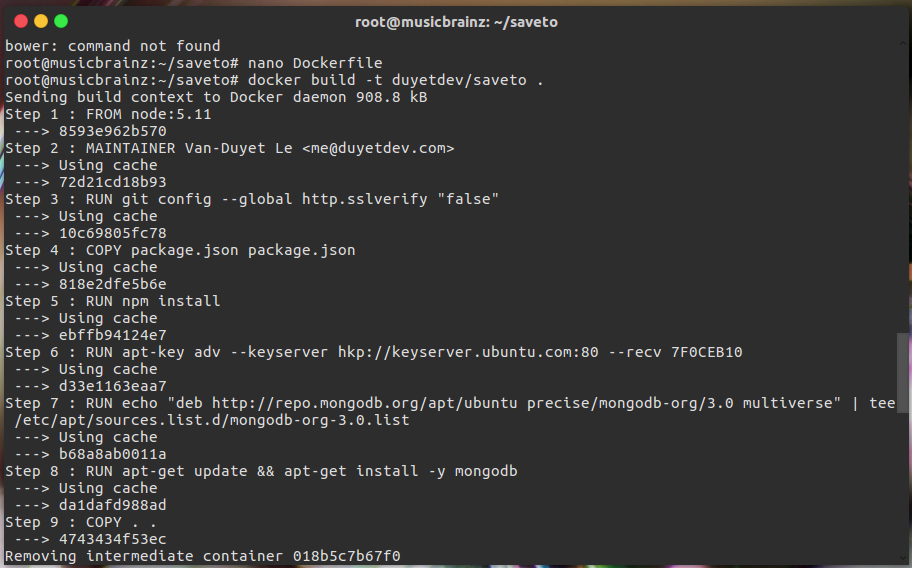
EXPOSE 6969

CMD ["npm","start"]

# Build image

Cd đến thư mục chứa Dockerfile, chạy lệnh bên dưới để build image. --tag dùng để đặt tên cho ảnh dễ nhớ, nhớ đừng bỏ xót dấu chấm ở cuối

$ docker build -t duyetdev/saveto .

[[](https://4.bp.blogspot.com/-jD52kEZB4-8/VyYuFInnraI/AAAAAAAAUZ0/-c21J642VJoPF0pHN44nNf1owMfg7F1rACLcB/s1600/Screenshot%2Bfrom%2B2016-05-01%2B23-25-04.png)](https://4.bp.blogspot.com/-jD52kEZB4-8/VyYuFInnraI/AAAAAAAAUZ0/-c21J642VJoPF0pHN44nNf1owMfg7F1rACLcB/s1600/Screenshot%2Bfrom%2B2016-05-01%2B23-25-04.png)

Sau khi build thành công, xem lại danh sách Docker image bằng:

root@duyetdev:~/saveto*# docker images*

REPOSITORY TAG IMAGE ID CREATED SIZE

node 5.11 8593e962b570 9 days ago 644.3 MB

duyetdev/saveto latest 5e8c25e32c8b 5 minutes ago 1.067 GB

# Run the image

Run docker image với tùy chọn -d sẽ chạy container dưới background, tùy chọn -p sẽ mapping port của **máy thật** (public) với port của **container** (private)

root@duyetdev:~/saveto*# docker run -p 6969:6969 -d duyetdev/saveto*

62cd21863c2fce3a1e9076d07c68fb32c0172d37c5fb9f2ea536a16a12fa527b

62cd21863c2fce3a1e9076d07c68fb32c0172d37c5fb9f2ea536a16a12fa527b là ID của container.   
Xem danh sách các container và log đang chạy:

*# Get container ID*

$ docker ps

*# Print app output*

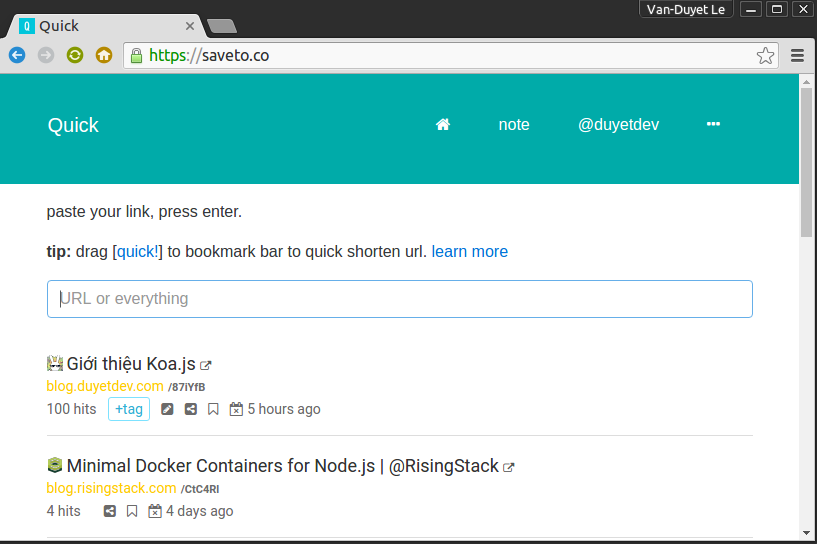
$ docker logs <container id>

Nếu cần truy cập vào container đang chạy, sử dụng:

*# Enter the container*

$ docker exec -it <container id> /bin/bash

Test: truy cập thử ứng dụng bằng trình duyệt: [http://localhost:6969](http://localhost:6969/)

[[](https://4.bp.blogspot.com/-lBI9EPH-ck4/VyYy72_trTI/AAAAAAAAUaM/O8qnBPlgyv4oWGDEqs2O5zXiE6aRIfmHwCLcB/s1600/Screenshot%2Bfrom%2B2016-05-01%2B23-45-24.png)](https://4.bp.blogspot.com/-lBI9EPH-ck4/VyYy72_trTI/AAAAAAAAUaM/O8qnBPlgyv4oWGDEqs2O5zXiE6aRIfmHwCLcB/s1600/Screenshot%2Bfrom%2B2016-05-01%2B23-45-24.png)

Chúc bạn thành công.