MEAN STACK APP ON DOCKER CONTAINERS : MICRO SERVICES

(http://www.addthis.com/bookmark.php?v=250&username=khhong7)

bogotobogo.com site search:

Search

Ph.D. / Golden Gate Ave, San Francisco / Seoul National Univ / Carnegie Mellon / UC Berkeley / DevOps / Deep Learning / Visualization

Introduction

In this tutorial, we'll deploy MEAN application to two Docker containers, and our local machine will be hosting the two containers:

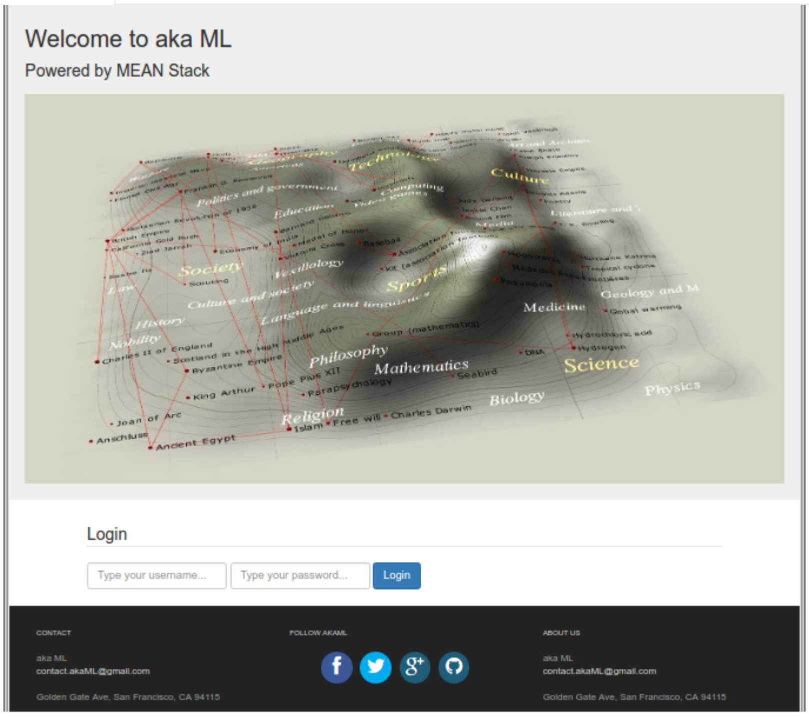
1. mongodb container
2. node/express/angular app

Here is the home page of the app:

Node.JS

Node.js (/AngularJS/NodeJS.php)

MEAN Stack : MongoDB, Express.js, AngularJS, Node.js (/MEAN-Stack/MEAN-Stack-MongoDB-ExpressJS-AngularJS-NodeJS.php)



Source code : Github (https://github.com/Einsteinish/akaML)

The app uses passport for user authentication, and for more features, please consult the README.md of the repo.

Local nginx for testing app

Though we don't need Nginx server in the Docker work flow of this tutorial, before we deploy our app to Docker containers, we may want to test it with Nginx. The configuration file looks like this:

```
        proxy_pass http://localhost: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;
    }
}
```

With **/etc/hosts**:

```
127.0.0.1 akaml.com
```

To run our MEAN app on host machine but not on container, the following line in **config/database.js** should be modified like this instead of "container's ip":

```
var dbURI = 'mongodb://localhost:27017/myApp';
```

Also, mongodb should be running before MEAN app's run:

```
$ sudo service mongodb restart
```

In the project folder, install packages:

```
$ npm install
$ sudo npm install -g bower
$ bower install
```

Then start Nginx proxy server:

```
$ sudo service nginx start
```

Now, we're ready to run MEAN app using the Nginx as a reverse proxy configuration:

```
$ node server.js
```

Or

```
$ nodemon server.js
```

Or

Meteor app with React (/MEAN-Stack/Meteor_React.php)

MEAN Stack app on Docker containers : micro services (/MEAN-Stack/MEAN-Stack-NodeJS-Angular-Docker.php)

MEAN Stack app on Docker containers : micro services via docker-compose (/MEAN-Stack/MEAN-Stack-NodeJS-Angular-Docker-Compose.php)

Sponsor Open Source development activities and free contents for everyone.

Donate with PayPal

Thank you.

- K Hong (http://bogotobogo.com/about_us.php)

AngularJS

MongoDB docker

To get our application running, the MongoDB container needs be started first.

We'll use OFFICIAL REPOSITORY : mongo (https://hub.docker.com/_/mongo/).

```
$ docker run [-p 27017:27017] --name mymongodb -d mongo
```

1. **-p 27017:27017** exposes the MongoDB port so the mean container can connect to it.
2. **-d** runs it as a background process (detached mode).
3. **--name mymongodb** gives this container a name so it can be referenced.
4. **mongo** is the image name that should be run.

Our **mongo** will be listening on 27017 port by default and that's it. We don't have to do anything once we launched the container.

```
$ docker ps
CONTAINER ID   IMAGE  COMMAND                   CREATED         STATUS         PORTS       NAMES
8f333617ed15   mongo  "/entrypoint.sh mongo"    4 minutes ago   Up 3 minutes   27017/tcp   mymongodb
```

MEAN stack on docker

Even though we'll build our container starting from a simple **ubuntu:14.04** images and make it ready for MEAN app, we may use the ready-made image of MEANJS (https://hub.docker.com/r/maccam912/meanjs/ (https://hub.docker.com/r/maccam912/meanjs/)).

If **maccam912/meanjs** is chosen, we can run the container with the following command:

```
$ docker run -i -t --name mymeanjs --link mymongodb:db_1 -p 80:3000 maccam912/meanjs:latest bash
```

Then, we can skip this section.

```
$ docker run -i -t --name mymeanjs --link mymongodb:db_1 -p 80:3000 ubuntu:14.04 bash
```

Here, the **--link mymongodb:db_1** argument is a link between this **mymeanjs** container and **mymongodb** container. This is a docker's way of communicating between containers.

Directives I - ng-app, ng-model, and ng-bind (/AngularJS/AngularJS_ng-directives.php)

Directives II - ng-show, ng-hide, and ng-disabled (/AngularJS/AngularJS_More_Directiv

Directives III - ng-click with toggle() (/AngularJS/AngularJS_More_Directiv

Expressions - numbers, strings, and arrays (/AngularJS/AngularJS_Expressions.pl

Binding - ng-app, ng-model, and ng-bind (/AngularJS/AngularJS_Binding.php)

Controllers - global controllers, controller method, and external controllers (/AngularJS/AngularJS_Controllers.ph

Data Binding and Controllers (Todo App) (/AngularJS/AngularJS_Controller_Dat

Todo App with Node (/MEAN-Stack/MEAN-Stack-MongoDB-ExpressJS-AngularJS-NodeJS-ToDoList-App-B.php)

$scope - A glue between javascript (controllers) and HTML (the view) (/AngularJS/AngularJS_Scopes.php)

Tables and css (/AngularJS/AngularJS_Tables.php)

Dependency Injection - http:fetch json & minification (/AngularJS/AngularJS_Dependency_I

Filters - lower/uppercase, currenty, orderBy, and filter:query with http.get() (/AngularJS/AngularJS_Filters.php)

$http - XMLHttpRequest and json file (/AngularJS/AngularJS_HttpRequest_j

Module - module file and controller file (/AngularJS/AngularJS_Module.php)

Forms (/AngularJS/AngularJS_Forms.php)

db_1 is an alias to reference this connected container. In other words, our MEAN application is set to use **db_1**.

By the argument of **-p 80:3000**, we're mapping the 3000 container port to 80 host machine port. Our MEAN application is set to run on port 3000, and the mapping enables any request on http:80 from outside the container to access our app running deep inside our container.

To install MEAN, we need to work within the docker container and do the following:

```
# apt-get update
# apt-get install nodejs
# ln -s "$(which nodejs)" /usr/bin/node
# apt-get install npm
```

To check if our install:

```
# node -v
v0.10.25
# npm -v
1.3.10
```

Express & bower install:

```
# npm install -g express
# npm install -g bower
```

Get our MEAN app

Install git:

```
# apt-get install git
```

Clone our repo to get the source code:

```
# cd home
# git clone https://github.com/Einsteinish/akaML.git
# cd akaML
```

On our MEAN.JS folder, download all the package dependencies:

```
# npm install
```

Install the front-end dependencies running by running bower:

Run our MEAN app

Make sure our mongodb ip is correct in **config/database.js**:

```
var dbURI = 'mongodb://172.17.0.2:27017/myApp';
```

We can check it by issuing the following command on our host machine, and this will give us two ips:

```
172.17.0.3
172.17.0.2
```

On our MEAN container, we can check ip via "ifconfig":

```
inet addr:172.17.0.3
```

So, the ip for mongodb is '172.17.0.2', and our database configuration is correct!

Let's run our MEAN app:

Meteor Angular App with MongoDB (Part II - Angular talks with MongoDB) (/AngularJS/Meteor-AngularJS-ToDo-App-with-MongoDB-2.php)

Meteor Angular App with MongoDB (Part III - Facebook / Twitter / Google logins) (/AngularJS/Meteor-AngularJS-with-MongoDB-User-Registration-Social-Auth-Facebook-Twitter-GooglePlus.php)

Scala/Java Play app with Angular (/Java/tutorials/ScalaPlay/ScalaPlayIn

Laravel 5 / Angular Auth using JSON Web Token (JWT) - Prod (/Laravel5/Laravel5_Angular_Auth_us

Scala/Java Play app with Angular (/Java/tutorials/ScalaPlay/ScalaPlayIn

Docker & K8s

Docker install on Amazon Linux AMI (/DevOps/Docker/Docker_Install_On_

Docker install on EC2 Ubuntu 14.04 (/DevOps/Docker/Docker_Install_On_

Docker container vs Virtual Machine (/DevOps/Docker/Docker_Container_

Docker install on Ubuntu 14.04 (/DevOps/Docker/Docker_Install_On_

Docker Hello World Application (/DevOps/Docker/Docker_Hello_Wor

Nginx image - share/copy files, Dockerfile (/DevOps/Docker/Docker_Nginx_Wel

Working with Docker images : brief introduction

```
GET /scripts/lib/angular-toastr/dist/angular-toastr.css 200 95ms - 6.64kb
GET /scripts/lib/bootstrap/dist/css/bootstrap.css 200 112ms - 142.59kb
GET /css/app.css 200 226ms - 1.32kb
GET /scripts/lib/requirejs/require.js 200 186ms - 84.24kb
GET /scripts/src/main.js 200 38ms - 1.46kb
GET /favicon.ico 404 5ms
GET /scripts/src/app.js 200 5ms - 3.89kb
GET /scripts/lib/jquery/dist/jquery.min.js 200 31ms - 84.33kb
GET /scripts/lib/angular/angular.min.js 200 37ms - 156.3kb
GET /scripts/src/controllers.js 200 5ms - 7.39kb
GET /scripts/lib/cryptojslib/rollups/pbkdf2.js 200 68ms - 5.4kb
GET /scripts/src/services.js 200 45ms - 4.23kb
GET /scripts/lib/bootstrap/dist/js/bootstrap.min.js 200 41ms - 36.18kb
GET /scripts/lib/angular-route/angular-route.min.js 200 26ms - 4.65kb
GET /scripts/lib/angular-animate/angular-animate.min.js 200 26ms - 25.11kb
GET /scripts/lib/angular-local-storage/dist/angular-local-storage.min.js 200 44ms - 6.25kb
GET /scripts/lib/angular-toastr/dist/angular-toastr.tpls.min.js 200 18ms - 7.02kb
{ REQUEST:
   { HEADERS:
      { host: 'akaml.com',
        connection: 'keep-alive',
        accept: 'application/json, text/plain, */*',
        'user-agent': 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/5
        referer: 'http://akaml.com/',
        'accept-encoding': 'gzip, deflate, sdch',
        'accept-language': 'en-US,en;q=0.8',
        'if-none-match': '"1892644392"' },
     BODY: {} } }
GET /partials/login 304 35ms
{ REQUEST:
   { HEADERS:
      { host: 'akaml.com',
        connection: 'keep-alive',
        accept: 'application/json, text/plain, */*',
        'user-agent': 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/5
        referer: 'http://akaml.com/',
        'accept-encoding': 'gzip, deflate, sdch',
        'accept-language': 'en-US,en;q=0.8',
        'if-none-match': '"1227391045"' },
     BODY: {} } }
GET /partials/nav.html 304 8ms
{ REQUEST:
   { HEADERS:
      { host: 'akaml.com',
        connection: 'keep-alive',
        accept: 'application/json, text/plain, */*',
        'user-agent': 'Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/5
        referer: 'http://akaml.com/',
        'accept-encoding': 'gzip, deflate, sdch',
        'accept-language': 'en-US,en;q=0.8' },
     BODY: {} } }
GET /partials/header.html 200 5ms - 170b
GET /img/home/akaML-home.jpg 200 28ms - 82.08kb
GET /scripts/src/main.js 304 3ms
GET /scripts/src/app.js 304 5ms
GET /scripts/lib/angular/angular.min.js 304 2ms
GET /scripts/src/controllers.js 304 4ms
GET /scripts/src/services.js 304 1ms
GET /scripts/lib/cryptojslib/rollups/pbkdf2.js 304 2ms
GET /scripts/lib/bootstrap/dist/js/bootstrap.min.js 304 2ms
GET /scripts/lib/angular-route/angular-route.min.js 304 4ms
GET /scripts/lib/angular-animate/angular-animate.min.js 304 1ms
GET /scripts/lib/angular-local-storage/dist/angular-local-storage.min.js 304 3ms
GET /scripts/lib/angular-toastr/dist/angular-toastr.tpls.min.js 304 3ms
GET /scripts/lib/requirejs/require.js 304 2ms
```

(/DevOps/Docker/Docker_Command

More on docker run command (docker run -it, docker run --rm, etc.) (/DevOps/Docker/Docker_Run_Comr

Docker Networks - Bridge Driver Network (/DevOps/Docker/Docker-Bridge-Driver-Networks.php)

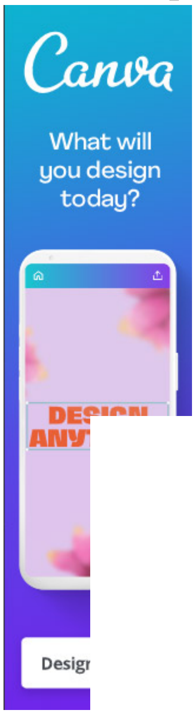
Docker Persistent Storage (/DevOps/Docker/Docker_Container_

File sharing between host and container (docker run -d -p -v) (/DevOps/Docker/Docker_File_Share_

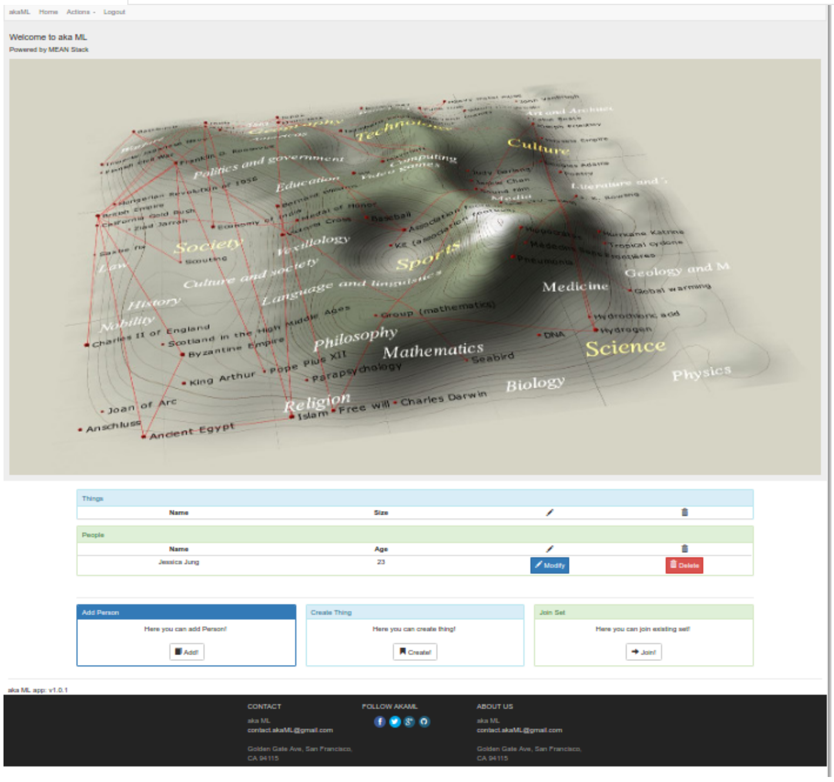
Linking containers and volume for datastore (/DevOps/Docker/Docker_Container_

Dockerfile - Build Docker images automatically I - FROM, MAINTAINER, and build context (/DevOps/Docker/Docker_Dockerfile_

Dockerfile - Build Docker images automatically II - revisiting FROM, MAINTAINER, build context, and caching (/DevOps/Docker/Docker_Dockerfile_



Dockerfile - Build Docker images automatically III - RUN



Welcome to aka ML
Powered by MEAN Stack

Things

Name	Size		

People

Name	Age		
Jessica Jung	23	Modify	Delete

Add Person
Here you can add Person!
Add!

Create Thing
Here you can create thing!
Create!

Join Set
Here you can join existing set!
Join!

aka ML app: v1.0.1

CONTACT
aka ML
contact.akaML@gmail.com
Golden Gate Ave, San Francisco, CA 94115

FOLLOW AKAML

ABOUT US
aka ML
contact.akaML@gmail.com
Golden Gate Ave, San Francisco, CA 94115

Optional : Run node server with PM2

We may want to run node server as a daemon using PM2 (http://pm2.keymetrics.io/).

```
# npm install pm2 -g
```

Now that the PM2 is installed, let's start our Node application:

```
# pm2 start server.js
```

Then, we get the following screen output:

Dockerfile - Build Docker images automatically V - WORKDIR, ENV, ADD, and ENTRYPOINT (/DevOps/Docker/Docker_Dockerfile

Docker - Apache Tomcat (/DevOps/Docker/Docker_Apache_To

Docker - NodeJS (/DevOps/Docker/Docker-NodeJS.php)

Docker - NodeJS with hostname (/DevOps/Docker/Docker-NodeJS-with-hostname.php)

Docker Compose - NodeJS with MongoDB (/DevOps/Docker/Docker-Compose-Node-MongoDB.php)

Docker - Prometheus and Grafana with Docker-compose (/DevOps/Docker/Docker_Promethe

Docker - StatsD/Graphite/Grafana (/DevOps/Docker/Docker_StatsD_Gra

Docker - Deploying a Java EE JBoss/WildFly Application on AWS Elastic Beanstalk Using Docker Containers (/DevOps/Docker/Docker_Container_

Docker : NodeJS with GCP Kubernetes Engine (/DevOps/Docker/Docker-NodeJS-GCP-Kubernetes-Engine.php)

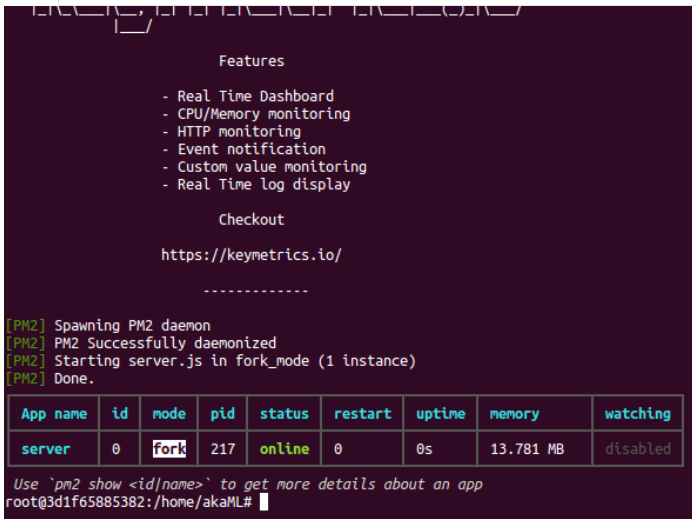
Docker : Jenkins Multibranch Pipeline with Jenkinsfile and Github (/DevOps/Docker/Docker-Jenkins-Multibranch-Pipeline-with-Jenkinsfile-and-Github.php)

Docker : Jenkins Master and Slave (/DevOps/Docker/Docker-Jenkins-Master-Slave-Agent-ssh.php)

Docker - ELK : ElasticSearch, Logstash, and Kibana (/DevOps/Docker/Docker_ELK_Elastic

Docker - ELK 7.6 : Elasticsearch on Centos 7 (/DevOps/Docker/Docker_ELK_7_6_E

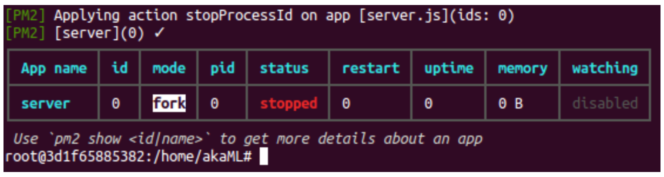
Docker - ELK 7.6 : Filebeat on Centos 7



```
                        Features

                - Real Time Dashboard
                - CPU/Memory monitoring
                - HTTP monitoring
                - Event notification
                - Custom value monitoring
                - Real Time log display

                        Checkout

                https://keymetrics.io/

                     ------------

[PM2] Spawning PM2 daemon
[PM2] PM2 Successfully daemonized
[PM2] Starting server.js in fork_mode (1 instance)
[PM2] Done.
│ App name │ id │ mode │ pid │ status │ restart │ uptime │ memory    │ watching │
│ server   │ 0  │ fork │ 217 │ online │ 0       │ 0s     │ 13.781 MB │ disabled │
 Use `pm2 show <id|name>` to get more details about an app
root@3d1f65885382:/home/akaML#
```

To stop the server:

```
# pm2 stop server.js
```



```
[PM2] Applying action stopProcessId on app [server.js](ids: 0)
[PM2] [server](0) ✓
│ App name │ id │ mode │ pid │ status  │ restart │ uptime │ memory │ watching │
│ server   │ 0  │ fork │ 0   │ stopped │ 0       │ 0      │ 0 B    │ disabled │
 Use `pm2 show <id|name>` to get more details about an app
root@3d1f65885382:/home/akaML#
```

Optional : Run node server with nodemon

The PM2 has relatively bigger footprints, so as an alternative we can use **nodemon**.

```
# npm install -g nodemon
# nodemon server.js
```

Optional : Saving the docker image

We want to save our work (layers in docker terminology) so far.

Let's exit the NodeJS app container:

```
root@3d1f65885382:/home/akaML# exit
$
```

Then, check which containers are running:

Docker - ELK 7.6 : Kibana on Centos 7 Part 1 (/DevOps/Docker/Docker_ELK_7_6_K

Docker - ELK 7.6 : Kibana on Centos 7 Part 2 (/DevOps/Docker/Docker_ELK_7_6_K

Docker - ELK 7.6 : Elastic Stack with Docker Compose (/DevOps/Docker/Docker_ELK_7_6_E

Docker - Deploy Elastic Cloud on Kubernetes (ECK) via Elasticsearch operator on minikube (/DevOps/Docker/Docker_Kubernete

Docker - Deploy Elastic Stack via Helm on minikube (/DevOps/Docker/Docker_Kubernete

Docker Compose - A gentle introduction with WordPress (/DevOps/Docker/Docker-Compose.php)

Docker Compose - MySQL (/DevOps/Docker/Docker-Compose-MySQL.php)

MEAN Stack app on Docker containers : micro services (/MEAN-Stack/MEAN-Stack-NodeJS-Angular-Docker.php)

Docker Compose - Hashicorp's Vault and Consul Part A (install vault, unsealing, static secrets, and policies) (/DevOps/Docker/Docker-Vault-Consul.php)

Since we exited from the NodeJS container, only the mongodb container is running.

To check our recent containers, we can use **docker ps -a** command:

```
$ docker ps -a
CONTAINER ID  IMAGE                     COMMAND  CREATED      STATUS                    PORTS  NAMES
3d1f65885382  maccam912/meanjs:latest   "bash"   2 hours ago  Exited (0) 4 minutes ago         mymeanjs
```

"docker commit" is the command we want to use to save the image:

```
$ docker commit -a khong 3d1f65885382 nodejs-micro-service:0.1
581298efda9f2874ed86f90f47b4834c6eb863650cdf4e3b764d939ece1cfd31
```

Here "-a" flag is for the author, "3d1f65885382" is the **container id**, and after that we specified the name (**repository**) of the image with version **tag**.

Let's check we really create a new image:

```
$ docker images
REPOSITORY            TAG    IMAGE ID       CREATED          VIRTUAL SIZE
nodejs-micro-service  0.1    581298efda9f   3 minutes ago    1.116 GB
```

Let's run our newly created image in background:

```
$ docker run -d -p 80:3000 nodejs-micro-service:0.1 pm2 start /home/akaML/server.js
8d3eb5ab0d89d65243b29ff06ccc781bca9c3698ebaa791ac1a4f8ae3a92a76d
```

Here, the "-d" flag is "detached" meaning background run, "80:3000" is port forwarding, "nodejs-micro-service:0.1" is the container name, and "node /home/akaML/server.js" is the command to run our nodejs app.

Now we should have two running containers - mongo and nodejs

```
$ docker ps
CONTAINER ID  IMAGE                     COMMAND               CREATED        STATUS        PORTS
0af14aa1f952  nodejs-micro-service:0.1  "node /home/akaML/ser" 8 seconds ago Up 5 seconds  80/tcp, 443
8f333617ed15  mongo                     "/entrypoint.sh mongo" 8 hours ago   Up 8 hours    27017/tcp
```

We can use "docker attach container-id" command to see what's going on inside our containers:

Docker Compose - Hashicorp's Vault and Consul Part B (EaaS, dynamic secrets, leases, and revocation) (/DevOps/Docker/Docker-Vault-Consul-B.php)

Docker Compose - Hashicorp's Vault and Consul Part C (Consul) (/DevOps/Docker/Docker-Vault-Consul-C.php)

Docker Compose with two containers - Flask REST API service container and an Apache server container (/DevOps/Docker/Docker-Compose-FlaskREST-Service-Container-and-Apache-Container.php)

Docker compose : Nginx reverse proxy with multiple containers (/DevOps/Docker/Docker-Compose-Nginx-Reverse-Proxy-Multiple-Containers.php)

Docker compose : Nginx reverse proxy with multiple containers (/DevOps/Docker/Docker-Compose-Nginx-Reverse-Proxy-Multiple-Containers.php)

Docker & Kubernetes : Envoy - Getting started (/DevOps/Docker/Docker-Envoy-Getting-Started.php)

Docker & Kubernetes : Envoy -

Note: Unfortunately, the command "pm2 start /home/akaML/server.js" in detached mode does not seem to be working. So, here, I used "node server.js" as a command argument.

Ambassador - Envoy API Gateway on Kubernetes (/DevOps/Docker/Docker-Envoy-Ambassador-API-Gateway-for-Kubernetes.php)

Docker Packer (/DevOps/Docker/Docker-Packer.php)

Docker Cheat Sheet (/DevOps/Docker/Docker-Cheat-Sheet.php)

Docker Q & A (/DevOps/Docker/Docker_Q_and_A.p

Kubernetes Q & A - Part I (/DevOps/Docker/Docker_Kubernete

Kubernetes Q & A - Part II (/DevOps/Docker/Docker_Kubernete

Docker - Run a React app in a docker (/DevOps/Docker/Docker-React-App.php)

Docker - Run a React app in a docker II (snapshot app with nginx) (/DevOps/Docker/Docker-React-App-2-SnapShot.php)

Docker - NodeJS and MySQL app with React in a docker (/DevOps/Docker/Docker-React-Node-MySQL-App.php)

Docker - Step by Step NodeJS and MySQL app with React - I (/DevOps/Docker/Step-by-Step-React-Node-MySQL-App.php)

Installing LAMP via puppet on Docker (/DevOps/Docker/Installing-LAMP-with-puppet-on-Docker.php)

Docker install via Puppet (/DevOps/Docker/Docker_puppet.ph

Nginx Docker install via Ansible (/DevOps/Ansible/Ansible-Deploy-Nginx-to-Docker.php)

Apache Hadoop CDH 5.8 Install with QuickStarts Docker (/Hadoop/BigData_hadoop_CDH5.8_

Docker - Deploying Flask app to ECS (/DevOps/Docker/Docker-