Machine

Docker machine 是一个简化Docker安装的命令行工具，通过一个简单的命令行即可在相应的平台上安装Docker，比如VirtualBox、 Digital Ocean、Microsoft Azure。

目前docker machine beta，支持以下服务：

* Amazon EC2
* Microsoft Azure
* Microsoft Hyper-V
* DigitalOcean
* Google Compute Engine
* OpenStack
* Rackspace
* SoftLayer
* VirtualBox
* VMware Fusion
* VMware vCloud Air
* VMware vSphere

***Note****: Machine is currently in beta, so things are likely to change. We don't recommend you use it in production yet.*

Install Linux

$ curl -L https://github.com/docker/machine/releases/download/v0.2.0/docker-machine\_darwin-amd64 > /usr/local/bin/docker-machine

$ chmod +x /usr/local/bin/docker-machine

Windows

Install the docker clinet

$ curl -L https://get.docker.com/builds/Windows/x86\_64/docker-latest.exe > /bin/docker

install the docker machine

$ curl -L https://github.com/docker/machine/releases/download/v0.2.0/docker-machine\_windows-amd64.exe > /bin/docker-machine

Getting started with Docker machine using a local VM

Using docker machine with a cloud provider

Adding a host without a driver

Using docker machine with docker swarm

Driver

Amazon web services, digital ocean, google compute engine, IBM softlayer

Microsoft azure, Microsoft hyper-V, openstack, Rackspace, oracle virtualbox

Vmware fusion, vmware vCloud Air, Vmware vSphere

Swarm

[Docker Swarm](https://github.com/docker/swarm/)是一个Dockerized化的分布式应用程序的本地集群，它是在[Machine](http://linux.cn/article-4393-1.html)所提供的功能的基础上优化主机资源的利用率和容错服务。具体来说，Docker Swarm支持用户创建可运行Docker Daemon的主机资源池，然后在资源池中运行Docker容器。Docker Swarm可以管理工作负载并维护集群状态

Docker Swarm可以保证应用的高可用性和容错性。Docker Swarm会不断的检查Docker Daemon所在主机的健康状态。当某个主机不可用时，Swarm就会将容器迁移到新的主机上。

Docker Swarm的亮点之一是它可以在应用的生命周期内扩展，也就是说当应用从一个主机扩展到2个、20个或者200个的时候，用户可以保证接口的一致性。

Swarm可插可拔

Swarm serves the standard Docker API.

Swarm 是一个标准的docker image.不依赖外部的框架。

So install swarm: docker pull swarm

***Note****: Swarm is currently in beta, so things are likely to change. We don't recommend you use it in production yet.*

Docker API兼容性，资源管理，约束（constraint），相关性（affinities）,TLS(保证与swarm之间的通信安全)

TLS

Swarm支持CLI and swarm之间的TLS认证证书。以及swarm 和docker之间的认证。

所有docker deamon和cli的证书，都必须用相同的认证CA-certificate.

Swarm 证书必须是常用的。

为clinet and server开启TLS：

Swarm manage –tlsverify –tlscacert=<CACERT> --tlscert=<CERT> --tlskey=<KEY> […]

如何在dokcer上设置TLS认证？？？

<https://docs.docker.com/articles/https/>

启动swarm步骤:

1. 运行一个命令去创建一个集群.
2. 运行另一个命令去启动Swarm.
3. 在运行有Docker Engine的每个主机上，运行一个命令与上面的集群相连

Set up swarm nodes

每个node都会运行一个node agent。每个agent都会注册一个referenced docker deamon,用来监控和更新node的状态。

For example,用docker hub发现服务

1. Create a docker cluster。

docker run –rm swarm create

(return a cluster id, 启动swarm agent 的时候可以用到)

1. Log into each node and do the following.
2. 启动docker deamon。保证在远程的swarm agent API上，docker是可访问tcp的。

$ docker –H tcp://0.0.0.0:2375 –d

1. 注册swarm agent.这node IP必须可以访问swarm manager.替换合适的node\_ip。用cluster\_id启动agent。

$ docker run –d swarm join –addr=<node\_ip:2375> token://<cluster\_id>

1. 启动swarm manager

docker run –d –p <swarm\_port>：2375 swarm manage token://<cluster\_id>

1. 这个manager在运行的时候，检查configuration.

docker –H tcp://<manager\_ip:manager\_port> info

Discovery services????

Swarm中用于维护docker集群状态的机制。

5种：node，file，consul，etcd,zookeeper

Using the hosted discovery service

$ swarm create

$ swarm join --addr=<node\_ip:2375> token://<cluster\_id>

$ swarm manage -H tcp://<swarm\_ip:swarm\_port> token://<cluster\_id>

$ docker -H tcp://<swarm\_ip:swarm\_port> info

$ docker -H tcp://<swarm\_ip:swarm\_port> run ...

$ docker -H tcp://<swarm\_ip:swarm\_port> ps

$ docker -H tcp://<swarm\_ip:swarm\_port> logs ...

$ swarm list token://<cluster\_id>

### Using a static file describing the cluster

$ echo <node\_ip1:2375> >> /tmp/my\_cluster

$ echo <node\_ip2:2375> >> /tmp/my\_cluster

$ swarm manage -H tcp://<swarm\_ip:swarm\_port> file:///tmp/my\_cluster

$ docker -H tcp://<swarm\_ip:swarm\_port> info

$ docker -H tcp://<swarm\_ip:swarm\_port> run ...

$ docker -H tcp://<swarm\_ip:swarm\_port> ps

$ docker -H tcp://<swarm\_ip:swarm\_port> logs ...

$ swarm list file:///tmp/my\_cluster

### Using etcd

$ swarm join --addr=<node\_ip:2375> etcd://<etcd\_ip>/<path>

$ swarm manage -H tcp://<swarm\_ip:swarm\_port> etcd://<etcd\_ip>/<path>

$ docker -H tcp://<swarm\_ip:swarm\_port> info

$ docker -H tcp://<swarm\_ip:swarm\_port> run ...

$ docker -H tcp://<swarm\_ip:swarm\_port> ps

$ docker -H tcp://<swarm\_ip:swarm\_port> logs ...

$ swarm list etcd://<etcd\_ip>/<path>

### Using consul

$ swarm join --addr=<node\_ip:2375> consul://<consul\_addr>/<path>

$ swarm manage -H tcp://<swarm\_ip:swarm\_port> consul://<consul\_addr>/<path>

$ docker -H tcp://<swarm\_ip:swarm\_port> info

$ docker -H tcp://<swarm\_ip:swarm\_port> run ...

$ docker -H tcp://<swarm\_ip:swarm\_port> ps

$ docker -H tcp://<swarm\_ip:swarm\_port> logs ...

$ swarm list consul://<consul\_addr>/<path>

### Using zookeeper

$ swarm join --addr=<node\_ip:2375> zk://<zookeeper\_addr1>,<zookeeper\_addr2>/<path>

$ swarm manage -H tcp://<swarm\_ip:swarm\_port> zk://<zookeeper\_addr1>,<zookeeper\_addr2>/<path>

$ docker -H tcp://<swarm\_ip:swarm\_port> info

$ docker -H tcp://<swarm\_ip:swarm\_port> run ...

$ docker -H tcp://<swarm\_ip:swarm\_port> ps

$ docker -H tcp://<swarm\_ip:swarm\_port> logs ...

$ swarm list zk://<zookeeper\_addr1>,<zookeeper\_addr2>/<path>

### Using a static list of ips

$ swarm manage -H <swarm\_ip:swarm\_port> nodes://<node\_ip1:2375>,<node\_ip2:2375>

Or

$ swarm manage -H <swarm\_ip:swarm\_port> <node\_ip1:2375>,<node\_ip2:2375>

$ docker -H <swarm\_ip:swarm\_port> info

$ docker -H <swarm\_ip:swarm\_port> run ...

$ docker -H <swarm\_ip:swarm\_port> ps

$ docker -H <swarm\_ip:swarm\_port> logs ...

### Range pattern for IP addresses

The file and nodes support a range pattern to specify IP addresses.

File eg:

$ echo "10.0.0.[11:100]:2375" >> /tmp/my\_cluster

$ echo "10.0.1.[15:20]:2375" >> /tmp/my\_cluster

$ echo "192.168.1.2:[2:20]375" >> /tmp/my\_cluster

Start the manager

$ swarm manage -H tcp://<swarm\_ip:swarm\_port> file:///tmp/my\_cluster

Nodes eg:

$ swarm manage -H <swarm\_ip:swarm\_port> "nodes://10.0.0.[10:200]:2375,10.0.1.[2:250]:2375"

Contributing a new discovery backend

type DiscoveryService interface {

Initialize(string, int) error

Fetch() ([]string, error)

Watch(WatchCallback)

Register(string) error

}

<https://github.com/docker/swarm/blob/master/discovery/README.md>

<https://docs.docker.com/swarm/discovery/>

使用bin pack自动优化所在位置的工作负载。

未来

1. 容错调度。Swarm可检测中断，并在其他主机重新调度容器时，遵循限制条件。
2. 高可用性调度。如果一个Swarm主机宕机，另一个主机将会被选举，集群上的容器调度将不会中断。
3. 调度器。调度器驱动API，将允许swarm与其他集群解决方案(mesos, kubernetes)相集成。

<https://github.com/docker/swarm/pull/393>

Compose

compose一个部署环境的工具

Fig has been replaced by docker-compose

用compose来定义的应用，可以在任何环境或供应方上运行。

一个定义和运行包含docker的多容器的工具。

使用compose分三步：

1. 定义app的环境变量在dockerfile。

2. 在docker-compose.yml文件中定义运行app的services信息。

3. run docker-compose up启动app.

Install compose

curl -L https://github.com/docker/compose/releases/download/1.2.0/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose

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

or

sudo pip install –U dokcer-compose

quick start

运行一个简单的python web app在compose上。

1. Install and set-up
2. Create a docker image

用dockerfile build一个Image.

1. Set up services in docker-compose.yml。
2. Build and run app with compose

Ps: start guide for django, rails, wordpress, commands, configuration file, environment variable.

Using compose in production

1. 在deploy changes的时候，我们需要 rebuild image, recreate app’s containers, redeploy service.

Eg: --no-deps prevents to recreate any services which ‘web’ depends on.

$ docker-compose build web

$ docker-compose up --no-deps -d web

### Running Compose on a single server

### Machine+compose

### You can use Compose to deploy an app to a remote Docker host by setting the DOCKER\_HOST,DOCKER\_TLS\_VERIFY, and DOCKER\_CERT\_PATH environment variables appropriately

### Running Compose on a Swarm cluster

### Swarm + compose

Extending services in compose

Why need the extending service.

1. Extending services is useful if you have several applications that reuse commonly-defined services.
2. Using extends you can define a service in one place and refer to it from anywhere.
3. you can deploy the same application to multiple environments with a slightly different set of services in each case

### Example use case

### The example assumes you want to use compose both to develop an application locally and then deploy it to a production environment.

### In development, you mount the application code as a volume so that it can pick up changes

### In production, the code should be immutable from the outside.

### The development environment uses a local Redis container.

### In production another team manages the Redis service, which is listening at redis-production.example.com.

1. Define the web app
2. Create an app.py file.
3. Define the Python dependencies in a requirements.txt file
4. Create a Dockerfile to build an image containing the app
5. Create a Compose configuration file called common.yml, to define how to run the app.
6. Define the development environment
7. Create a docker-compose.yml file.
8. Run docker-compose up

Compose creates, links, and starts a web and redis container linked together. It mounts your application code inside the web container.

1. Verify that the code is mounted by changing the message in app.py
2. Define the production environment
3. create a production.yml file
4. Run docker-compose -f production.yml up.

Reference

You can extend a service that itself extends another. You can extend indefinitely. Compose does not support circular references and docker-compose returns an error if it encounters them.

**This is the default behaviour - all exceptions are listed below.**

# original service

command: python app.py

# local service

command: python otherapp.py

# result

command: python otherapp.py

For build and image.

# original service

build: .

# local service

image: redis

# result

image: redis

For the **multi-value options** ports, expose, external\_links, dns and dns\_search

# original service

expose:

- "3000"

# local service

expose:

- "4000"

- "5000"

# result

expose:

- "3000"

- "4000"

- "5000"

For the environment.

# original service

environment:

- FOO=original

- BAR=original

# local service

environment:

- BAR=local

- BAZ=local

# result

environment:

- FOO=original

- BAR=local

- BAZ=local

For volumes.

# original service

volumes:

- /original-dir/foo:/foo

- /original-dir/bar:/bar

# local service

volumes:

- /local-dir/bar:/bar

- /local-dir/baz/:baz

# result

volumes:

- /original-dir/foo:/foo

- /local-dir/bar:/bar

- /local-dir/baz/:baz

Swarm+compose (integration)

Current, compose can create containers on a swarm cluster. but the majority of Compose apps won’t work out of the box unless all containers are scheduled on one host, defeating much of the purpose of using Swarm in the first place.

Compose and swarm 在批量处理的时候非常有用（多个容器需要上传下载不同的独立算法）。或者共享集群。（多个小组想在集群上部署应用）

在整合之前需要做的事情：

让在容器中已经停止的app可以相互通信。使不同主机上的容器可以相互通信。

Placement strategy(swarm)

<https://github.com/docker/swarm/tree/master/scheduler/strategy>

Etcd????（swarm）

Consul??? (swarm)

Zookeeper??(swarm)

Scheduler filters(swarm)

Advanced scheduling????(swarm)

Swarm API????

<https://docs.docker.com/swarm/API/>

<http://docs.docker.com/reference/api/docker_remote_api/>

boot2docker???(compose)

relate to: django, rails, wordpress, environment variable.