

Docker 集群入门与实战

这篇文章将侧重介绍 docker 在集群方面的部署，在介绍完集群相关概念之后会进入大家喜闻乐见的实战环节~

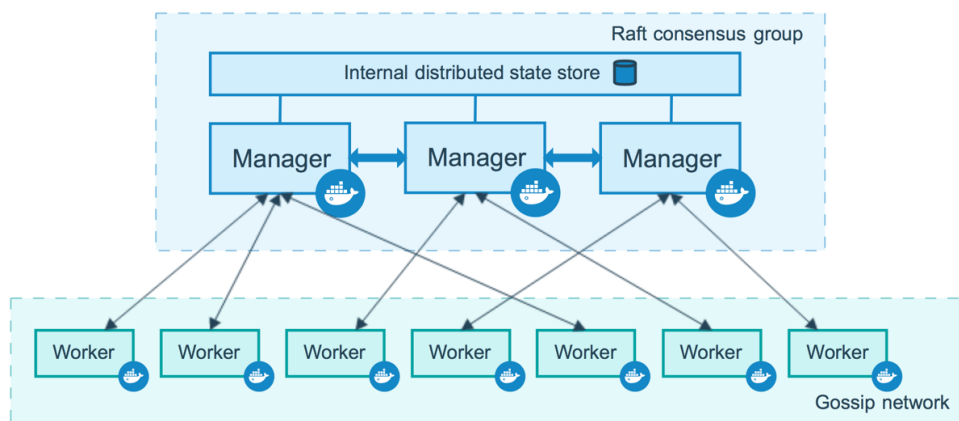
首先让我们来看看与集群相关的对象：

- swarm
- node
- stack
- service

其中 swarm 在上篇推文已经有过介绍，回顾请看->[链接](#)

NODE

node 的概念和 swarm 密切相关，swarm 是指一组包含一个 或多个 运行 docker engine 的主机（物理机或者虚拟机）组成的群组，节点（node）就自然是指在群组中的单体了。



运行 Docker 的主机可以主动初始化一个 Swarm 集群或者加入一个已存在的 Swarm 集群，这样这个运行 Docker 的主机就成为一个 Swarm 集群的节点 (node)。

node 有两类：managers 和 works，在下面的图中很容易理解它们的关系。

managers 负责，

works 负责，

node 的类别可以互相转换，如果想要转换一个 worker 为 manager，可以使用 `docker node promote`，反之则使用 `docker node demote`

```
1 $ docker node --help
2
3 Usage:      docker node COMMAND
4
5 Manage Swarm nodes
6
7 Commands:
8   demote    Demote one or more nodes from manager
9             in the swarm
10  inspect    Display detailed information on one or
11             more nodes
12  ls         List nodes in the swarm
13  promote    Promote one or more nodes to manager
14             in the swarm
15  ps         List tasks running on one or more
16             nodes, defaults to current node
17  rm         Remove one or more nodes from the
18             swarm
19  update     Update a node
20
21 Run 'docker node COMMAND --help' for more
22 information on a command.
```

使用ls命令查看节点信息：

```
1 $ docker node ls
2
3 ID                                HOSTNAME
4 STATUS                            AVAILABILITY    MANAGER STATUS
5 ENGINE VERSION
6 v1oo73vt40hnjivnhnw78po4u *    dameng-00
7 Ready                            Active          Leader
8 18.09.8-ce
9 yrz7drszt4hq5j8c8rbhoexqc      dameng-01
10 Ready                            Active
11 18.09.8-ce
```

SERVICE

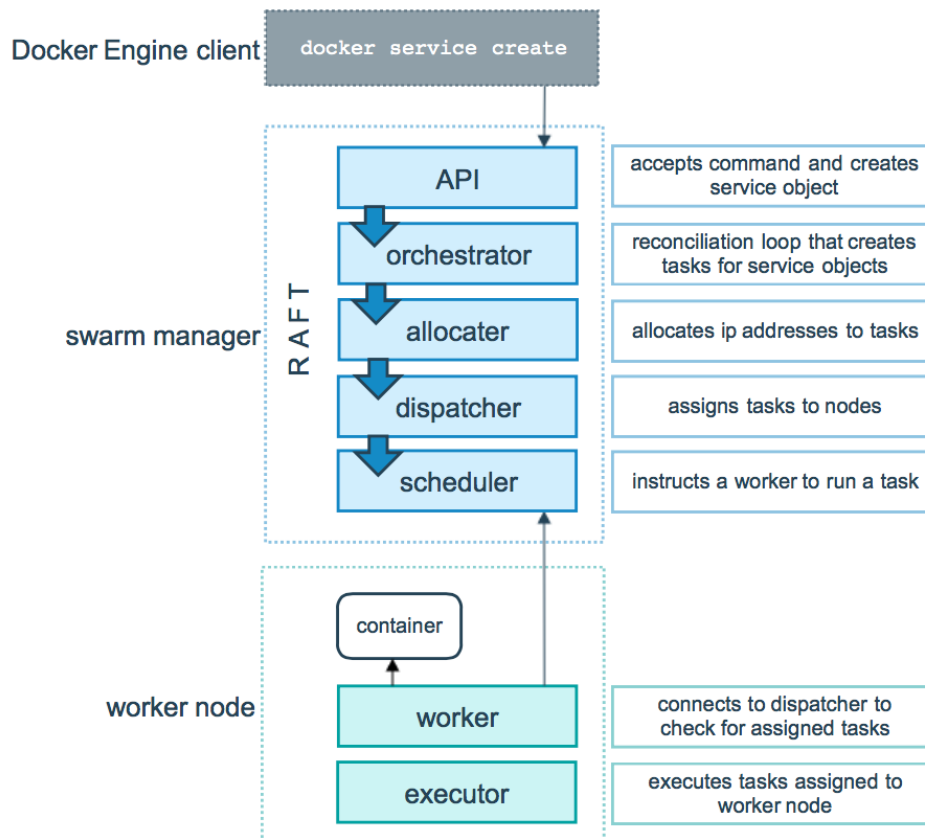
A **service** is the definition of how you want to run your application containers in a swarm.

简单来说，service 定义了 worker node 上要执行的任务。swarm 的主要编排任务就是保证 service 处于期望的状态下。

创建service时，您可以指定要使用的容器镜像以及在运行容器中执行的命令。您还可以定义service的选项，包括：

- 群集在群组外部提供服务的端口
- 服务的覆盖网络，以连接到群中的其他服务
- CPU和内存限制和预留
- 滚动更新政策
- 要在群中运行的图像的副本数

Service 的创建过程：



```
1 $ docker service --help
2
3 Usage:    docker service COMMAND
4
5 Manage services
6
```

```

7  Commands:
8      create      Create a new service
9      inspect     Display detailed information on one or
more services
10     logs        Fetch the logs of a service or task
11     ls          List services
12     ps          List the tasks of one or more services
13     rm          Remove one or more services
14     rollback    Revert changes to a service's
configuration
15     scale        Scale one or multiple replicated
services
16     update      Update a service
17
18 Run 'docker service COMMAND --help' for more
information on a command.

```

查看:

```

1  $ docker service ls
2
3  ID                                NAME                                MODE
4  REPLICAS                          IMAGE                                PORTS
oyiesfc3biiq                        myapp_myapp                        replicated
1/1                                friendlyhello:v3                  *:5000-
>5000/tcp
5  xrk7kskalz76                      myapp_redis                        replicated
1/1                                redis:latest

```

扩容:

```

1  $ docker service scale myapp_myapp=2
2
3  ID                                NAME                                MODE
4  REPLICAS                          IMAGE                                PORTS
oyiesfc3biiq                        myapp_myapp                        replicated
2/2                                friendlyhello:v3                  *:5000-
>5000/tcp
5  xrk7kskalz76                      myapp_redis                        replicated
1/1                                redis:latest

```

STACK

stack是一组相互关联的服务，它们共享依赖关系，并且可以一起 orchestrated (编排)和缩放。单个stack能够定义和协调整个应用程序的功能（尽管非常复杂的应用程序可能希望使用多个stack）。

```
1  $ docker stack --help
2
3  Usage:      docker stack [OPTIONS] COMMAND
4
5  Manage Docker stacks
6
7  Options:
8      --kubeconfig string      Kubernetes config file
9      --orchestrator string    Orchestrator to use
                                (swarm|kubernetes|all)
10
11  Commands:
12      deploy      Deploy a new stack or update an
                    existing stack
13      ls          List stacks
14      ps          List the tasks in the stack
15      rm          Remove one or more stacks
16      services    List the services in the stack
```

部署:

```
1  # compose:
2  $ docker-compose -f CONFIG-YAML up
3
4  # stack:
5  $ docker stack deploy -c CONFIG-YAML STACK-NAME
```

查看:

```
1  # compose:
2  $ docker-compose ps
3  $ docker ps
4
5  # stack:
6  $ docker node ls
7  $ docker stack ls
8  $ docker service ls
```

终止:

```
1  # compose:
2  $ docker-compose -f CONFIG-YAML down
3
4  # stack:
5  $ docker stack rm STACK-NAME
```

网络:

单机 vs 跨节点

注意: `docker stack` 默认使用的是swarm, 但也是可以对接k8s的

TRY IT OUT (4)

这次我们要用 `Swarm` 集群来部署前面做过的 `friendlyhello`。

在前面, 我们了解到可以用 `Docker Machine` 很快地创建一个虚拟的Docker主机, 接下来我们来创建2个新的Docker主机, 并加入到集群中。

STEP 1: 创建Swarm集群——管理节点

首先是一个管理节点, 创建并通过ssh连接:

```
1  $ docker-machine create -d virtualbox manager
2  $ docker-machine ssh manager
```

我们可以看到:



```
learn-docker/ch02/demo4 on v19.03.1
+ docker-machine create -d virtualbox manager
Running pre-create checks...
Creating machine...
(manager) Copying /Users/ronyou/.docker/machine/cache/boot2docker.iso to /Users/ronyou/.docker/machine/machines/manager/boot2docker.iso...
(manager) Creating VirtualBox VM...
(manager) Creating SSH key...
(manager) Starting the VM...
(manager) Check network to re-create if needed...
(manager) Waiting for an IP...
Waiting for machine to be running, this may take a few minutes...
Detecting operating system of created instance...
Waiting for SSH to be available...
Detecting the provisioner...
Provisioning with boot2docker...
Copying certs to the local machine directory...
Copying certs to the remote machine...
Setting Docker configuration on the remote daemon...
Checking connection to Docker...
Docker is up and running!
To see how to connect your Docker Client to the Docker Engine running on this virtual machine, run: docker-machine env manager

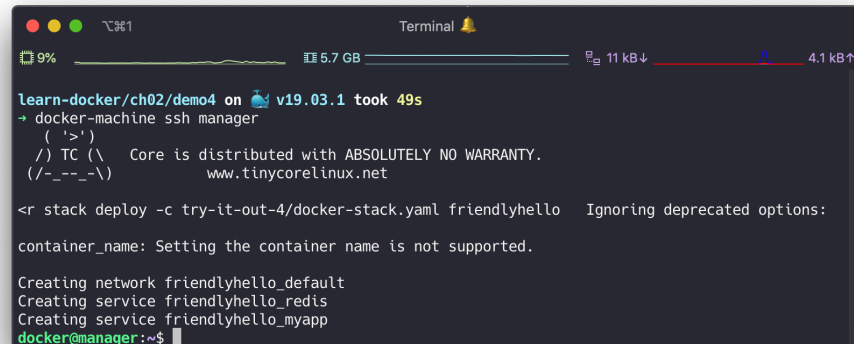
learn-docker/ch02/demo4 on v19.03.1 took 56s
+ docker-machine ssh manager
( ' > ' )
/) TC ( \   Core is distributed with ABSOLUTELY NO WARRANTY.
(/-__-_\)   www.tinycorelinux.net

docker@manager:~$
```

然后，我们用 `docker swarm init` 从这个节点初始化一个 `Swarm` 集群，如果这个 `Docker` 主机有多个 IP（多个网卡），就要用 `--advertise-addr` 指定一个：

```
1 $ docker swarm init --advertise-addr 192.168.99.107
```

我们可以看到：



```
learn-docker/ch02/demo4 on v19.03.1 took 49s
+ docker-machine ssh manager
( ' > ' )
/) TC ( \   Core is distributed with ABSOLUTELY NO WARRANTY.
(/-__-_\)   www.tinycorelinux.net

<r stack deploy -c try-it-out-4/docker-stack.yaml friendlyhello Ignoring deprecated options:
container_name: Setting the container name is not supported.

Creating network friendlyhello_default
Creating service friendlyhello_redis
Creating service friendlyhello_myapp
docker@manager:~$
```

现在我们的 `Manager` 节点就是刚刚创建的集群的管理节点了，

记得复制一下它输出的添加工作节点的那句命令。

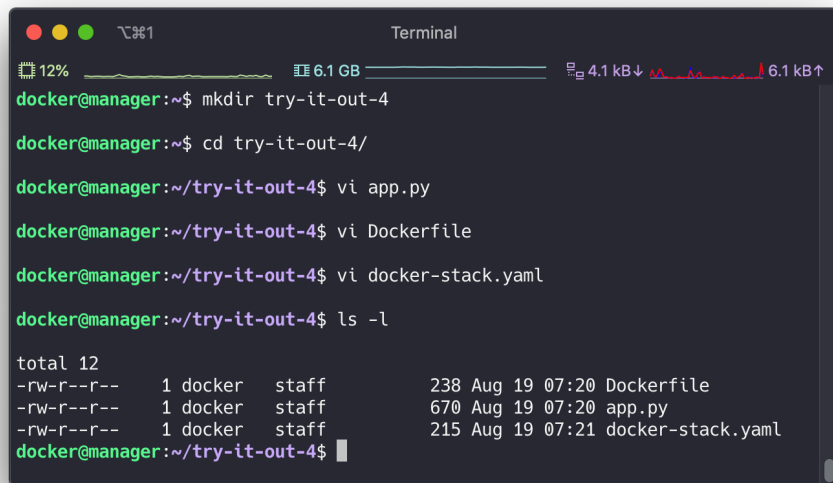
STEP 2: 添加项目文件

接下来我们 `Manager` 节点的 `~/try-it-out-4` 里添加几个文件：

- `app.py`
- `Dockerfile`

- docker-stack.yaml

```
1 $ mkdir try-it-out-4
2 $ cd try-it-out-4
3 $ vi app.py
4 $ vi Dockerfile
5 $ vi docker-stack.yaml
6
7 $ docker build -t friendlyhello .
```

A terminal window titled "Terminal" with a dark background. It shows a series of commands being executed in a Docker environment. The user creates a directory, changes to it, and uses the vi editor to create three files: app.py, Dockerfile, and docker-stack.yaml. Finally, they run 'ls -l' to list the files. The terminal output shows the file permissions, owner (docker), group (staff), size, date, and filename for each file.

```
docker@manager:~$ mkdir try-it-out-4
docker@manager:~$ cd try-it-out-4/
docker@manager:~/try-it-out-4$ vi app.py
docker@manager:~/try-it-out-4$ vi Dockerfile
docker@manager:~/try-it-out-4$ vi docker-stack.yaml
docker@manager:~/try-it-out-4$ ls -l
total 12
-rw-r--r--  1 docker  staff    238 Aug 19 07:20 Dockerfile
-rw-r--r--  1 docker  staff    670 Aug 19 07:20 app.py
-rw-r--r--  1 docker  staff    215 Aug 19 07:21 docker-stack.yaml
docker@manager:~/try-it-out-4$
```

app.py

```
1 from flask import Flask
2 from redis import Redis, RedisError
3 import os
4 import socket
5
6 # Connect to Redis
7 redis = Redis(host="redis", db=0,
8               socket_connect_timeout=2, socket_timeout=2)
9
10 app = Flask(__name__)
11
12 @app.route("/")
13 def hello():
```



```

14         try:
15             visits = redis.incr("counter")
16         except RedisError:
17             visits = "<i>cannot connect to Redis,
counter disabled</i>"
18
19         html = "<h3>Hello {name}!</h3>" \
20             "<b>Hostname:</b> {hostname}<br/>" \
21             "<b>Visits:</b> {visits}"
22         return html.format(name=os.getenv("NAME",
"world"), hostname=socket.gethostname(),
visits=visits)
23
24
25 if __name__ == "__main__":
26     app.run(host='0.0.0.0', port=5000)

```

Dockerfile

```

1 FROM python:3.7-slim
2
3 WORKDIR /app
4
5 COPY . /app
6
7 RUN pip install flask redis -i
https://mirrors.aliyun.com/pypi/simple --trusted-
host mirrors.aliyun.com
8 EXPOSE 5000
9 ENV NAME World
10
11 CMD ["python", "app.py"]

```

docker-stack.yaml

```

1 version: "3"
2
3 services:
4     myapp:
5         image: friendlyhello
6         container_name: myapp
7         ports:

```

```
8      - 5000:5000
9      environment:
10         NAME: World
11
12      redis:
13         image: redis
14         container_name: web
```

STEP 3: 创建Swarm集群——工作节点

继续，我们来创建一个工作节点：

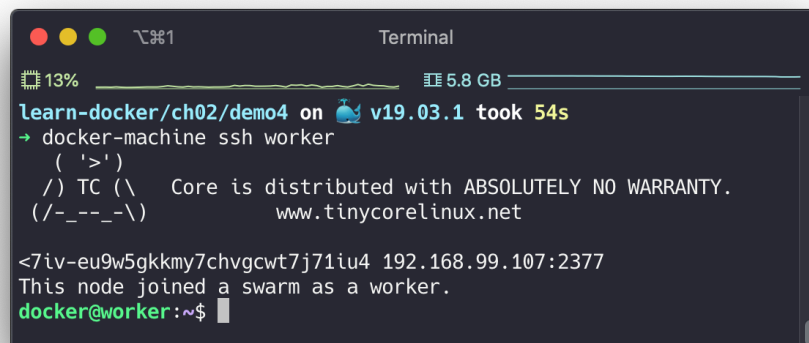
首先回到主机：

```
1  $ exit
```

接着创建一个新的虚拟机 `worker`，并通过上面复制的那句命令加入到集群里：

```
1  $ docker-machine create -d virtualbox worker
2  $ docker-machine ssh worker
3
4  $ docker swarm join --token SWMTKN-1-
   3wd0vdozskitmpw5vofkjc9ie6251wuno2ldmbugqk56pd97iv-
   eu9w5gkkmy7chvgcwt7j71iu4 192.168.99.107:2377
```

我们可以看到：



```
learn-docker/ch02/demo4 on v19.03.1 took 54s
→ docker-machine ssh worker
( '>' )
/) TC (\   Core is distributed with ABSOLUTELY NO WARRANTY.
(/-__-_)   www.tinycorelinux.net

<7iv-eu9w5gkkmy7chvgcwt7j71iu4 192.168.99.107:2377
This node joined a swarm as a worker.
docker@worker:~$
```

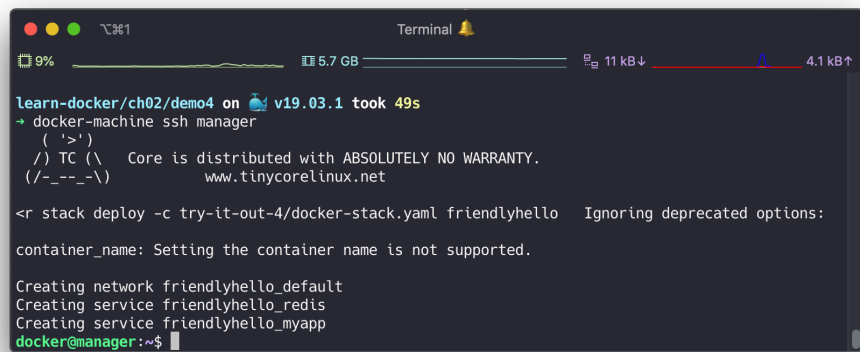
STEP 4: 使用Stack部署服务

我们先回到 `manager` 节点：

```
1 $ exit
2
3 $ docker-machine ssh manager
```

然后使用 `docker stack deploy` 部署服务，其中 `-c` 参数指定 `docker-stack.yaml` 文件：

```
1 $ docker stack deploy -c ~/try-it-out/docker-
  stack.yaml friendlyhello
```



```
learn-docker/ch02/demo4 on v19.03.1 took 49s
→ docker-machine ssh manager
( '>')
/) TC (\ Core is distributed with ABSOLUTELY NO WARRANTY.
(/_--_-\) www.tinycorelinux.net

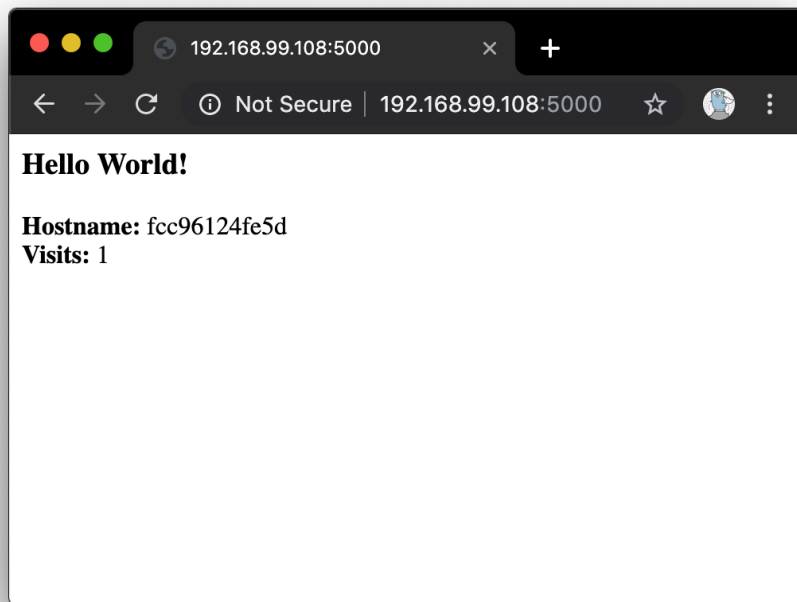
<r stack deploy -c try-it-out-4/docker-stack.yaml friendlyhello Ignoring deprecated options:
container_name: Setting the container name is not supported.

Creating network friendlyhello_default
Creating service friendlyhello_redis
Creating service friendlyhello_myapp
docker@manager:~$
```

部署完毕。

STEP 5: 访问friendlyhello

现在就可以通过集群中的任意一个节点的IP访问到这个 `flask` 项目了：



TRY IT OUT (5)

我们继续用前一个Case的集群。

进入 `manager` 节点，在 `try-it-out-5` 里新建一个 `docker-stack.yaml`：

```
1 $ docker-machine ssh manager
2
3 $ mkdir try-it-out-5
4 $ vi docker-stack.yaml
```

docker-stack.yaml

```
1 version: '3.1'
2
3 services:
4
5   db:
6     image: postgres
7     command: postgres -c 'shared_buffers=512MB' -c
8       'max_connections=2000'
9     restart: always
10    environment:
```

```
10     POSTGRES_USER: dameng
11     POSTGRES_PASSWORD: pythonic
12     ports:
13     - 5432:5432
14     volumes:
15     - pgdata:/var/lib/postgresql/data
16
17
18     adminer:
19     image: adminer
20     restart: always
21     ports:
22     - 8998:8080
23
24     volumes:
25     pgdata:
```

然后部署:

```
1 $ docker stack deploy -c docker-stack.yaml postgresql
```

接着就可以从 8998 端口访问 GUI 了:

