Docker 高级篇

- Docker 高级篇
  - 私有仓库的认证
    - 创建私有仓库的自签名证书
    - 基于私有证书的认证
      - 使用证书的方式启动私有仓库
      - 在客户端测试
    - 基于自签名证书和 `htpasswd` 的双重认证
      - 创建用户密码认证文件`htpasswd`
      - 使用自签名证书启动一个私有仓库
      - 配置客户端信任私有仓库的证书
      - 登录私有仓库并测试 `push` 和 `pull`
    - 常见问题：
    - 使用 nginx 反向代理
      - 介绍
      - 陷阱
      - 设置
      - 启动和停止
  - Docker Registry API
  - Docker 容器夸主机通信
    - Network NS
    - Docker overlay networks
      - 基本介绍
    - 服务的创建和调度
      - 基本使用
    - 搭建一个集群，并使用默认的 overlay network
      - 实验环境
      - 创建集群
      - 创建服务
  - 部署我的 python 项目
    - Dockerfile

私有仓库的认证

创建私有仓库的自签名证书

假设场景：

搭建的私有仓库的域名： `docker.qf.com`

下面我们介绍使用 `openssl` 自行签发 `docker.qf.com` 的站点 SSL 证书。

1. 安装相关软件

```
# 假如没有 openssl 命令，则需要安装软件
yum install openssl*
```

2. 创建 `CA` 的私钥

```
openssl genrsa -out "root-ca.key" 4096
```

3. 利用私钥创建 `CA` 根证书请求文件。

```
openssl req \
 -new -key "root-ca.key" \
 -out "root-ca.csr" -sha256 \
 -subj '/C=CN/ST=BeiJing/L=HaiDian/O=QF/CN=QF Docker Registry CA'
```

以上命令中 `-subj` 参数里的:

`/C` 表示国家，如 `CN`；

`/ST` 表示省；

`/L` 表示城市或者地区；

`/O` 表示组织名；

`/CN` 通用名称。

4. 配置 `CA` 根证书。

新建 `root-ca.cnf` 文件，内容如下：

```
[root_ca]
basicConstraints = critical,CA:TRUE,pathlen:1
keyUsage = critical, nonRepudiation, cRLSign, keyCertSign
subjectKeyIdentifier=hash
```

5. 签发根证书。

执行如下命令：

```
openssl x509 -req  -days 3650  -in "root-ca.csr" \
        -signkey "root-ca.key" -sha256 -out "root-ca.crt" \
        -extfile "root-ca.cnf" -extensions \
        root_ca
```

上面的步骤是创建了 `CA` 的根证书和私钥，现在我们的私有仓库就有资格给其他客户端签发证书了。

下面的步骤是为私有仓库创建整数和私钥

6. 生成私有仓库的 `SSL` 私钥。

执行如下命令

```
openssl genrsa -out "docker.qf.com.key" 4096
```

7. 使用私钥生成私有仓库的证书请求文件。

```
openssl req -new -key "docker.qf.com.key" \
-out "site.csr" -sha256 \
-subj '/C=CN/ST=BeiJing/L=HaiDian/O=QF/CN=docker.qf.com'
```

8. 配置私有仓库的证书。

   新建 `site.cnf` 文件，输入如下内容：

```
[server]
authorityKeyIdentifier=keyid,issuer
basicConstraints = critical,CA:FALSE
extendedKeyUsage=serverAuth
keyUsage = critical, digitalSignature, keyEncipherment
subjectAltName = DNS:docker.qf.com, IP:127.0.0.1
subjectKeyIdentifier=hash
```

9. 签署私有仓库的 `SSL` 证书。

执行如下命令：

```
openssl x509 -req -days 750 -in "site.csr" -sha256 \
-CA "root-ca.crt" -CAkey "root-ca.key"  -CAcreateserial \
-out "docker.qf.com.crt" -extfile "site.cnf" -extensions \
server
```

这样已经拥有了域名为 `docker.qf.com` 私有仓库的如下信息：

- SSL 私钥 `docker.qf.com.key`
- SSL 证书 `docker.qf.com.crt`

10. 创建 `certs` 文件夹，并将 `docker.qf.com.key` `docker.qf.com.crt` 这两个文件移入，删除其他文件。

执行如下命令：

```
mkdir certs
mv docker.qf.com.* certs/
rm root-ca.* site.c*
```

基于私有证书的认证

使用证书的方式启动私有仓库

```
docker run -d \
  --restart=always \
  --name registry \
  --mount type=bind,src=$(pwd)/certs,dst=/certs \
  -e REGISTRY_HTTP_ADDR=0.0.0.0:443 \
  -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/docker.qf.com.crt \
  -e REGISTRY_HTTP_TLS_KEY=/certs/docker.qf.com.key \
  -p 443:443 \
  registry
```

在客户端测试

1. 配置客户端信任私有仓库的证书

就是哪个 Docker 主机要和此仓库进行 `push` 和 `pull` 等交互动作，就在哪台主机上进行如下操作。

把刚才制作的私有仓库证书，拷贝到此 `Docker` 主机的 `certs`目录下

先配置当前主机的 `/etc/hosts` ，对私有仓库的域名进行解析

```
# 私有仓库的 IP      私有仓库的域名
172.16.153.136    docker.qf.com
```

接下来，创建目录并拷贝私有仓库的证书文件到此 `Docker` 主机

```
mkdir certs
scp  root@docker.qf.com:/var/ftp/docker/certs/docker.qf.com.crt \
     certs/docker.qf.com.crt
```

2. 配置让此 `Docker` 主机信任此证书

运行如下命令：

```
mkdir -p /etc/docker/certs.d/docker.qf.com
cp ./certs/docker.qf.com.crt \
/etc/docker/certs.d/docker.qf.com/docker.qf.com.crt
```

此动作不需要启动 `Docker`

3. 向私有仓库推送和拉取镜像

记得改标签

推送

```
[root@docker docker_qf_auth]# docker push docker.qf.com/nginx
The push refers to repository [docker.qf.com/nginx]
7ab428981537: Pushed
82b81d779f83: Pushed
d626a8ad97a1: Pushed
latest: digest:
sha256:e4f0474a75c510f40b37b6b7dc2516241ffa8bde5a442bde3d372c9519c84d90 size: 948
```

删除本地的，再从私有仓库拉取

```
[root@docker docker_qf_auth]# docker pull docker.qf.com/nginx
Using default tag: latest
latest: Pulling from nginx
Digest: sha256:e4f0474a75c510f40b37b6b7dc2516241ffa8bde5a442bde3d372c9519c84d90
Status: Downloaded newer image for docker.qf.com/nginx:latest
[root@docker docker_qf_auth]# docker image ls docker.qf.com/nginx
REPOSITORY            TAG                 IMAGE ID            CREATED
SIZE
docker.qf.com/nginx   latest              ae513a47849c        4 weeks ago
109MB
```

基于自签名证书和 `htpasswd` 的双重认证

此实验中的证书是沿用上面 `创建私有仓库的自签名证书` 中的自签名证书。

创建用户密码认证文件 `htpasswd`

```
mkdir auth
docker run --rm  \
 --entrypoint htpasswd \
 registry -Bbn yangge 123 > auth/htpasswd
```

或者：

```
yum install httpd-tools
htpasswd -Bbn yangge 123 > auth/htpasswd
```

使用自签名证书启动一个私有仓库

1. 停掉并删除原来的私钥仓库容器

```
docker container stop registry && docker container rm -v registry
```

- -v 是同时把数据卷也删了

2. 使用 `TLS` 证书并使用基本身份认证的方式启动一个私有仓库容器

```
docker run -d \
  -p 443:443 \
  --restart=always \
  --name registry \
  -v `pwd`/auth:/auth \
  -e REGISTRY_HTTP_ADDR=0.0.0.0:443 \
  -e REGISTRY_AUTH=htpasswd \
  -e REGISTRY_AUTH_HTPASSWD_REALM="Registry Realm" \
  -e REGISTRY_AUTH_HTPASSWD_PATH=/auth/htpasswd \
  --mount type=bind,src=$(pwd)/certs,dst=/certs \
  -e REGISTRY_HTTP_TLS_CERTIFICATE=/certs/docker.qf.com.crt \
  -e REGISTRY_HTTP_TLS_KEY=/certs/docker.qf.com.key \
  registry
```

配置客户端信任私有仓库的证书

就是哪个 Docker 主机要和此仓库进行 `push` 和 `pull` 等交互动作，就在哪台主机上进行如下操作。

1. 把刚才制作的私有仓库证书，拷贝到此 `Docker` 主机的 `certs` 目录下

先配置当前主机的 `/etc/hosts` ，对私有仓库的域名进行解析

```
# 私有仓库的 IP      私有仓库的域名
172.16.153.136    docker.qf.com
```

在创建目录并拷贝私有仓库的证书文件到此 `Docker` 主机

```
mkdir certs
scp  root@docker.qf.com:/var/ftp/docker/certs/docker.qf.com.crt \
     certs/docker.qf.com.crt
```

2. 配置让此 `Docker` 主机信任此证书

运行如下命令：

```
mkdir -p /etc/docker/certs.d/docker.qf.com
cp ./certs/docker.qf.com.crt \
/etc/docker/certs.d/docker.qf.com/docker.qf.com.crt
```

此动作不需要启动 `Docker`

登录私有仓库并测试 `push` 和 `pull`

1. 登录到私有仓库

```
[root@docker docker_qf_auth]# docker login docker.qf.com
Username: yangge
Password:
Login Succeeded
```

2. 假如还不成功，`Centos` 系统还需配置在系统级别信任此证书

依次执行如下命令：

```
# 第一条命令
cp ./certs/docker.qf.com.crt \
/etc/pki/ca-trust/source/anchors/docker.qf.com.crt

# 第二条命令
update-ca-trust
```

之后改本地仓库的标签，推送拉取镜像即可

常见问题：

1. 出现`x509`错误，是客户端没有配置证书

2. 出现 `no basic auth credentials` 的提示，是客户端没有登录到私有仓库

使用 nginx 反向代理

介绍

使用此处介绍的方法，您可以在位于私有仓库前的反向代理中实施Docker引擎的基本身份验证。

虽然我们使用一个简单的htpasswd文件作为示例，但一旦完成该示例，任何其他nginx身份验证后端应该相当容易实现。

陷阱

虽然此模型允许您通过代理内部实现的辅助身份验证机制来使用任何所需的身份验证后端，但它还要求您停掉私有仓库，将TLS移至代理本身。

此外，在通信管道中引入额外的http层会使其部署，维护和调试更加复杂。确保需要额外的复杂性。

例如，HTTPS模式下的Amazon Elastic Load Balancer（ELB）已经设置了以下客户端标头：

```
X-Real-IP
X-Forwarded-For
X-Forwarded-Proto
```

所以如果你有一个Nginx实例在它后面，请从下面的示例配置中删除这些行：

```
proxy_set_header  Host              $http_host;   # required for docker client's
sake
proxy_set_header  X-Real-IP         $remote_addr; # pass on real client's IP
proxy_set_header  X-Forwarded-For   $proxy_add_x_forwarded_for;
proxy_set_header  X-Forwarded-Proto $scheme;
```

否则Nginx将重置ELB的值，并且请求没有正确路由。有关更多信息，请参阅 #970。

设置

`docker-compose` 简单介绍

是用于多容器部署的利器。

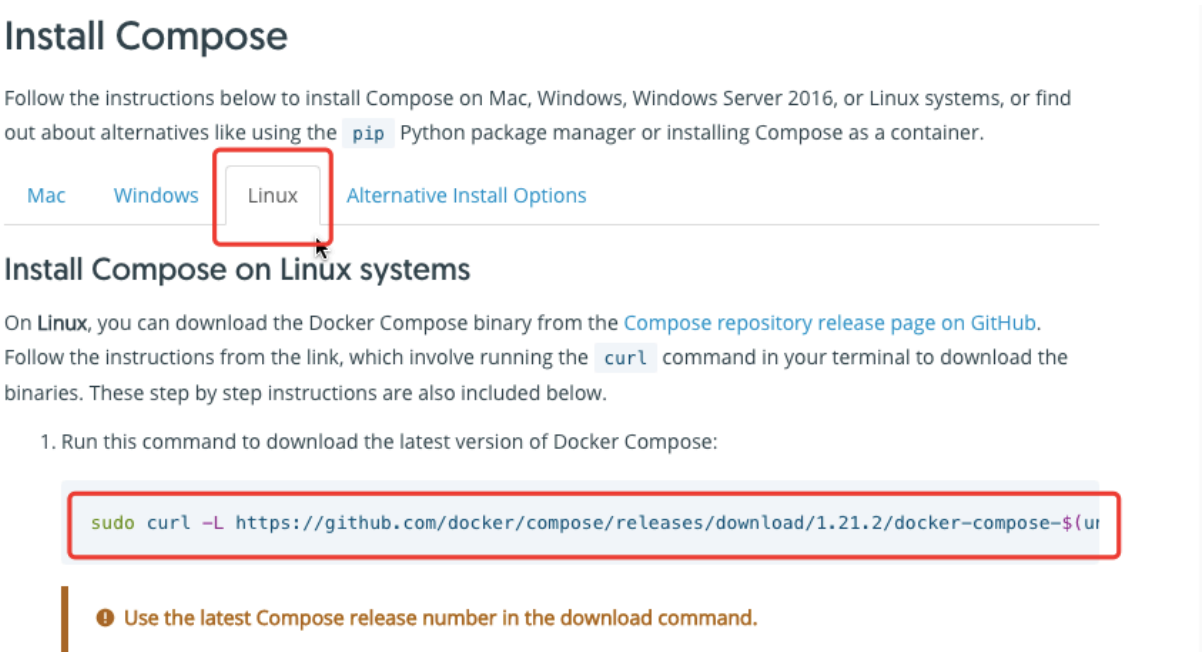
把需要运行的一组容器的配置信息，放在一个 `yaml` 格式的文件中，之后使用 `docker-compose` 命令对这些容器操作。

linux 版本下载：

```
sudo curl -L https://github.com/docker/compose/releases/download/1.21.2/docker-
compose-$(uname -s)-$(uname -m) -o /usr/local/bin/docker-compose
```

假如上面的安装命令出现了错误。请参考官网安装即可

https://docs.docker.com/compose/install/#install-compose




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

默认操作的 `yaml` 文件名： `docker-compose.yml` 此文件默认必须在当前目录下。

使用 `-f` 参数也可指定。

```
docker-compose up -d      # 启动在 docker-compose.yml 文件内的所有容器
docker-compose ps         # 查看所有容器
docker-compose stop       # 停止所有容器
docker-compose rm         # 删除处于退出状态的所有容器
docker-compose ps  服务名  # 就是在 yml 文件中定义的服务名
```

开始

1. 创建所需的目录

   ```
   mkdir -p auth data
   ```

2. 创建主要的nginx配置。将这个代码块粘贴到一个名为`auth/nginx.conf`：

   ```
   events {
       worker_connections  1024;
   }

   http {
   ```

```
  upstream docker-registry {
    server registry:5000;
  }

  ## Set a variable to help us decide if we need to add the
  ## 'Docker-Distribution-Api-Version' header.
  ## The registry always sets this header.
  ## In the case of nginx performing auth, the header is unset
  ## since nginx is auth-ing before proxying.
  map $upstream_http_docker_distribution_api_version
$docker_distribution_api_version {
    '' 'registry/2.0';
  }

  server {
    listen 443 ssl;
    # 私有仓库的域名
    server_name docker.qf.com;

    # SSL 注意，这里的两个文件需要填写本地的证书和私钥文件名
    ssl_certificate /etc/nginx/conf.d/docker.qf.com.crt;
    ssl_certificate_key /etc/nginx/conf.d/docker.qf.com.key;

    # Recommendations from
https://raymii.org/s/tutorials/Strong_SSL_Security_On_nginx.html
    ssl_protocols TLSv1.1 TLSv1.2;
    ssl_ciphers 'EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH';
    ssl_prefer_server_ciphers on;
    ssl_session_cache shared:SSL:10m;

    # disable any limits to avoid HTTP 413 for large image uploads
    client_max_body_size 0;

    # required to avoid HTTP 411: see Issue #1486
(https://github.com/moby/moby/issues/1486)
    chunked_transfer_encoding on;

    location /v2/ {
      # Do not allow connections from docker 1.5 and earlier
      # docker pre-1.6.0 did not properly set the user agent on ping, catch
"Go *" user agents
      if ($http_user_agent ~ "^(docker\/1\.(3|4|5(?!\.[0-9]-dev))|Go ).*$" )
{
        return 404;
      }

      # To add basic authentication to v2 use auth_basic setting.
      auth_basic "Registry realm";
```

```
      auth_basic_user_file /etc/nginx/conf.d/nginx.htpasswd;

      ## If $docker_distribution_api_version is empty, the header is not
added.
      ## See the map directive above where this variable is defined.
      add_header 'Docker-Distribution-Api-Version'
$docker_distribution_api_version always;

      proxy_pass     http://docker-registry;
      proxy_set_header  Host              $http_host;   # required for docker
client's sake
      proxy_set_header  X-Real-IP         $remote_addr; # pass on real
client's IP
      proxy_set_header  X-Forwarded-For   $proxy_add_x_forwarded_for;
      proxy_set_header  X-Forwarded-Proto $scheme;
      proxy_read_timeout                  900;
    }
  }
}
```

3. 为用户为： "yangge"，密码为："123" 的用户创建一个密码文件:`auth/nginx.htpasswd`。

```
yum install httpd-tools
htpasswd -Bbn yangge 123 > auth/nginx.htpasswd
```

> **注意**：如果您不想使用`bcrypt`，可以省略该`-B`参数。

4. 将您的证书文件复制到`auth/`目录。

```
cp docker.qf.com.crt auth
cp docker.qf.com.key auth
```

5. 将以下YAML粘贴到名为的新文件中`docker-compose.yml`。

```
nginx:
  # Note : Only nginx:alpine supports bcrypt.
  # If you don't need to use bcrypt, you can use a different tag.
  # Ref. https://github.com/nginxinc/docker-nginx/issues/29
  image: "nginx:alpine"
  ports:
    - 443:443
  links:
    - registry:registry
  volumes:
    - ./auth:/etc/nginx/conf.d
    - ./auth/nginx.conf:/etc/nginx/nginx.conf:ro

registry:
```

```
    image: registry:2
    ports:
      - 127.0.0.1:5000:5000
    volumes:
      - ./data:/var/lib/registry
```

启动和停止

启动

```
docker-compose up -d
```

用授权用户（使用`yangge`和`123`）登录，然后标记并推送和拉取您的第一个镜像：

```
[root@docker docker_nginx_auth]# docker login docker.qf.com
Username (yangge): yangge
Password:
Login Succeeded
```

推送和拉取的操作和之前的一样

Docker Registry API

官网： https://docs.docker.com/registry/spec/api/#introduction

待整理

```

```

Docker 容器夸主机通信

Network NS

```
# 查看
[root@docker ~]# ip netns

# 添加
[root@docker ~]# ip netns add test1
[root@docker ~]# ip netns add test2
[root@docker ~]# ip netns
```

```
test2
test1
[root@docker ~]# ip netns list
test2
test1

# 在一个网络名称空间中执行命令
[root@docker ~]# ip netns exec test2 ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
[root@docker ~]# ip netns exec test2 ip link
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

# up 一个网络名称空间中的一个接口
[root@docker ~]# ip netns exec test2 ip link set dev lo up
[root@docker ~]# ip netns exec test2 ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
[root@docker ~]# ip netns exec test2 ip link
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT
qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00

# 删除
[root@docker ~]# ip netns del test1
[root@docker ~]# ip netns del test2
```

创建两个网络名称空间，并实现互相通信

1. 创建两个名称空间和一对儿 `link` 接口

```
[root@docker ~]# ip netns add net-ns1
[root@docker ~]# ip netns add net-ns2
[root@docker ~]# ip link add veth-if1 type  veth peer name veth-if2
```

2. 把 `veth-if1` 添加到 `net-ns1`， 再把 `veth-if2` 添加到 `net-ns2` 中

```
[root@docker ~]# ip link set veth-if1 netns net-ns1
[root@docker ~]# ip link set veth-if2 netns net-ns2
```

查看一下

```
[root@docker ~]# ip netns exec net-ns1 ip link
[root@docker ~]# ip netns exec net-ns2 ip link
```

3. 分别给两个接口配置 IP 地址

```
ip netns exec net-ns1 ip addr add 192.168.10.1/24 dev veth-if1

ip netns exec net-ns2 ip addr add 192.168.10.2/24 dev veth-if2
```

4. 使他们 up

```
ip netns exec net-ns2 ip link set dev veth-if2 up

ip netns exec net-ns1 ip link set dev veth-if1 up
```

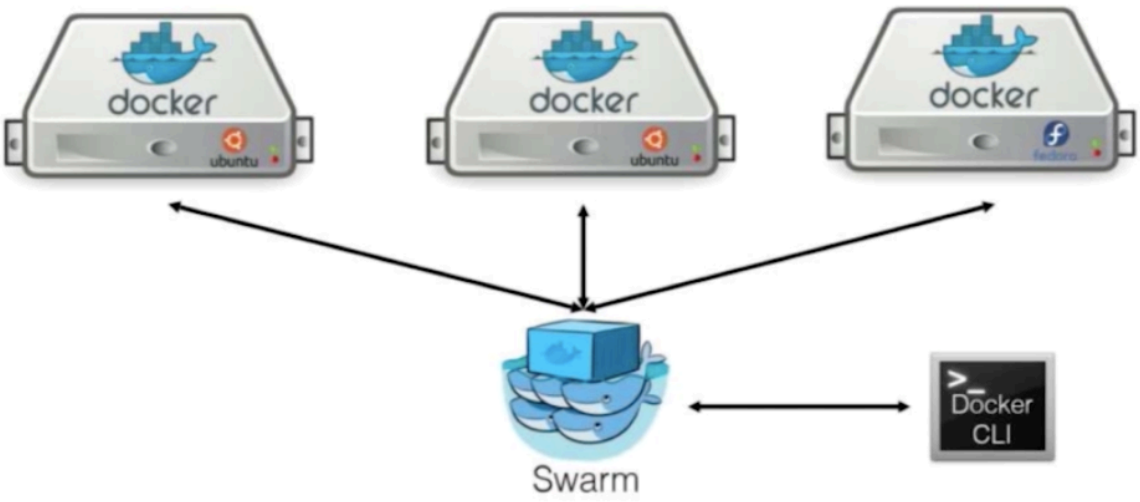
5. 分别在各自所在的网络名称空间里 `ping` 对方的 ip 地址，测试连通性

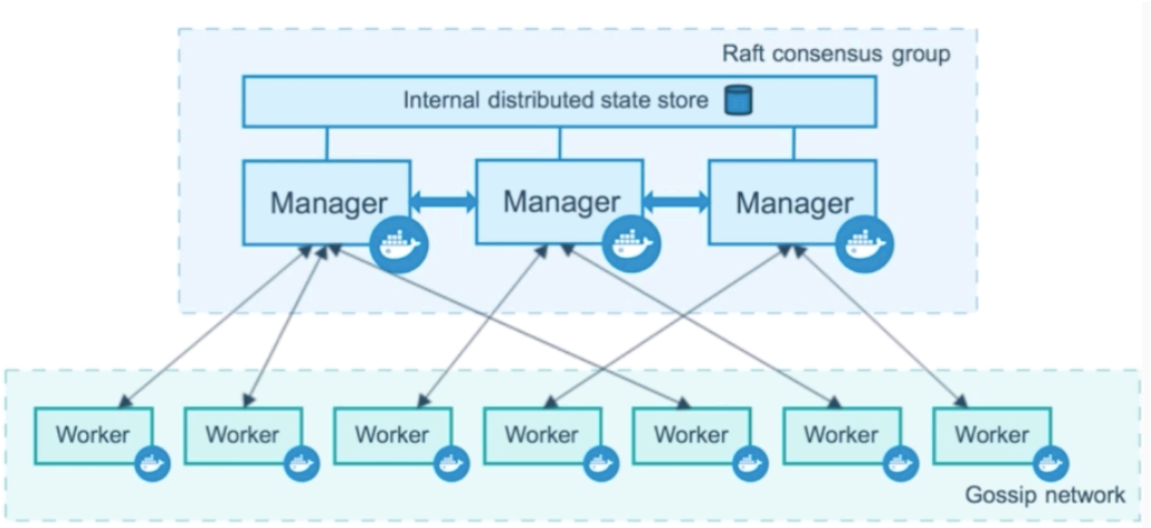
```
ip netns exec net-ns1 ping 192.168.10.2

ip netns exec net-ns2 ping 192.168.10.1
```

Docker overlay networks

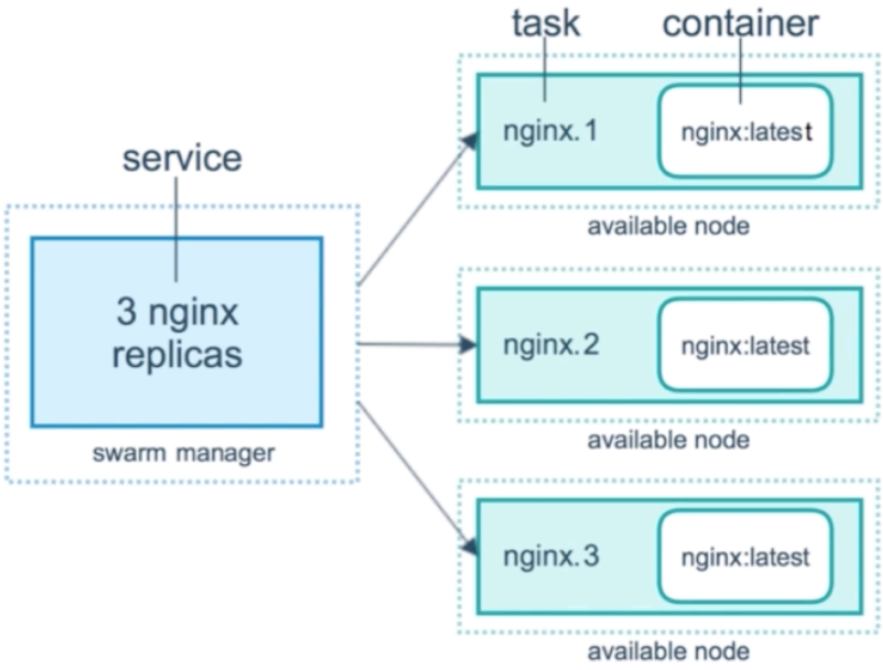
基本介绍

Swarm Mode

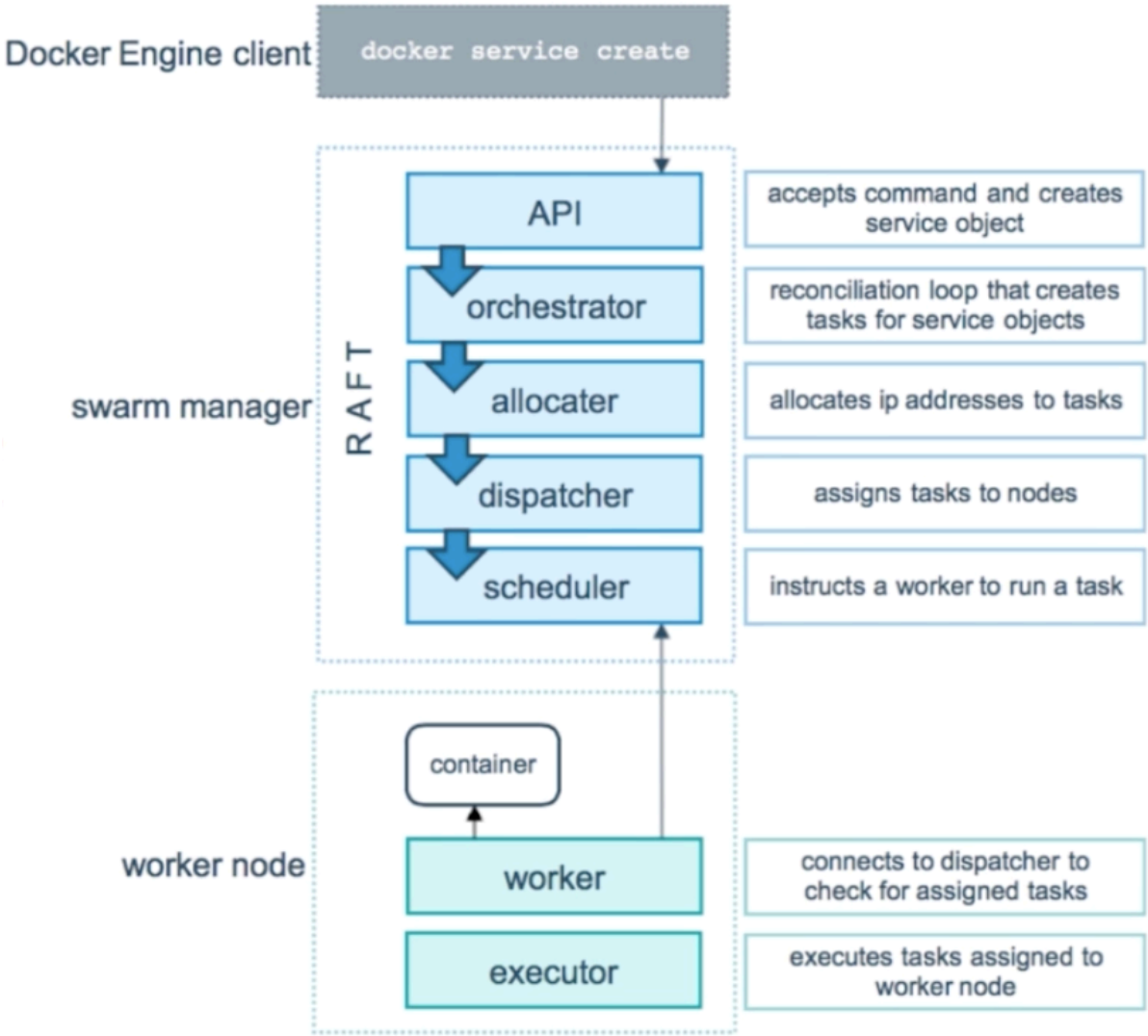

docker
ubuntu
docker
ubuntu
docker
fedora
Swarm
Docker
CLI


Raft consensus group
Internal distributed state store
Manager
Manager
Manager
Worker
Worker
Worker
Worker
Worker
Worker
Worker
Gossip network

Service和Replicas



服务的创建和调度



https://labs.play-with-docker.com/

基本使用

在创建覆盖网络之前，您需要使用`docker swarm init`命令初始化

这会创建一个默认的集群服务的默认 overlay network

即便你不需要使用一个 swarm services，以上的操作也是不需的。

之后，就可以创建用户自定义的其他 overlay network

```
docker swarm init
```

当主机上有多个网卡和地址时需要指定一个 IP 地址

```
[root@docker ~]# docker swarm init --advertise-addr 192.168.199.140
Swarm initialized: current node (see4dykosejm53zvzzq3o2lih) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-
4vkep2az7uz51zu4pls06ine2wjktgh3bmcor7bc330i2yj364-85ibthr27gqp1drhgl4amcxuw
192.168.199.140:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow
the instructions.
```

此时，Docker 会在本地创建一个 overlay network 的默认网络： `ingress`入口网络

```
[root@docker ~]# docker network ls
p1pebwhi5qmb        ingress             overlay         swarm
```

生产中需要遵循的防火墙规则：

- TCP端口2377用于集群管理通信
- TCP和UDP端口7946用于节点之间的通信
- UDP端口4789用于`overyay network`通信

创建一个集群中的 overlay network

```
[root@docker ~]# docker network create -d overlay my-overlay
8w5vf1sxlal2cb2fx96q7iw5v
```

所有群的服务管理数据流默认都是加密的，在GCM模式下使用AES算法。

群中的管理节点会每12小时更新一次用于加密数据的密钥。

自定义 overlay networlk

首先需要删除之前默认的 `ingress`

```
docker network rm ingress
```

在创建自定义的 `ingress`

```
docker network create \
  --driver overlay \
  --ingress \
  --subnet=10.11.0.0/16 \
  --gateway=10.11.0.2 \
  --opt com.docker.network.driver.mtu=1200 \
  my-ingress
```

上面的指定信息如下：

网段：10.11.0.0/16

网关：10.11.0.2

最大输出单元： 1200

名字： my-ingress

这个种网络只能有一个，接着重启 `Docker` 服务

`docker_gwbridge`

docker_gwbridge 是一个虚拟网桥，它将 `overlay network` （包括入口网络）连接到单独的Docker守护进程的物理网络。当您初始化群集或将Docker主机加入群集时，Docker会自动创建它，但它不是Docker设备。它存在于Docker主机的内核中。

搭建一个集群，并使用默认的 overlay network

实验环境

三台 Docker 主机，一个作为管理节点和工作节点，另外两个作为工作节点

- master ip 主机名： 192.168.199.140 docker
- node1 ip 主机名: 192.168.199.152 server1
- node2 ip 主机名: 192.168.199.102 server2

创建集群

1. 在 master 节点上执行初始化集群的命令，同时指定 ip。

```
[root@docker ~]# docker swarm init --advertise-addr 192.168.199.140
Swarm initialized: current node (see4dykosejm53zvzzq3o2lih) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-
4vkep2az7uz51zu4pls06ine2wjktgh3bmcor7bc330i2yj364-85ibthr27gqp1drhgl4amcxuw
192.168.199.140:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow
the instructions.
```

注意：记住上面的信息，其中包含了其他节点加入到此集群的 token

2. 分别在工作节点执行如下命令，以便假如集群

```
docker swarm join --token <TOKEN> \
  --advertise-addr <IP-ADDRESS-OF-WORKER-1> \
  <IP-ADDRESS-OF-MANAGER>:2377
```

node2

```
[root@server1 ~]# docker swarm join \
--token SWMTKN-1-4vkep2az7uz51zu4pls06ine2wjktgh3bmcor7bc330i2yj364-
85ibthr27gqp1drhgl4amcxuw \
--advertise-addr 192.168.199.153 \
192.168.199.140:2377

# 下面是加入成功后的输出信息
This node joined a swarm as a worker.
```

node1

```
[root@server2 ~]# docker swarm join \
> --token SWMTKN-1-4vkep2az7uz51zu4pls06ine2wjktgh3bmcor7bc330i2yj364-
85ibthr27gqp1drhgl4amcxuw \
> --advertise-addr 192.168.199.102 \
> 192.168.199.140:2377
This node joined a swarm as a worker.
```

3. 在 master 上可以列出所有的节点，此命令仅限于在管理节点操作。

查看集群的所有节点

```
[root@docker ~]# docker node  ls
```

查看集群的管理节点

```
[root@docker ~]# docker node ls --filter role=manager
```

查看集群的工作节点

```
[root@docker ~]# docker node ls --filter role=worker
```

4. 可以在任何节点上查看 overlay network 的列表

```
[root@docker ~]# docker network  ls
```

```
[root@server1 ~]# docker network ls
```

```
[root@server2 ~]# docker network ls
```

docker_gwbridge 相当于有创建了一个虚拟网桥，它可以将overlay network 的 `ingress` 网络数据封装在其数据包的数据部分里，把本身是连接到Docker主机的网络的。

创建服务

1. 在 master 节点创建一个新的 overlay network 名字： nginx-net

```
[root@docker ~]# docker network create -d overlay nginx-net
sjqe0oerzu3rdc5vmqce7r5tr
```

这个动作不需要在其他节点进行，因为当集群中的其他节点需要此网络时，会自动创建。

2. 在 master 节点，建立 nginx 容器，指定有 5 个副本，将其加入到网络`nginx-net`并向外部暴露 80 端口。

```
[root@docker ~]# docker service create \
>   --name my-nginx \
>   --publish target=80,published=80 \
>   --replicas=5 \
>   --network nginx-net \
>   nginx
phaq0wy7e227ztl01ejhpjg3s
overall progress: 5 out of 5 tasks
1/5: running
2/5: running
3/5: running
4/5: running
5/5: running
verify: Service converged
```

注意: Services can only be created on a manager.

3. 在 master 上删除服务，并删除创建的网络

```
$ docker service rm my-nginx
$ docker network rm nginx-net nginx-net-2
```

部署我的 python 项目

Dockerfile

```
FROM nginx:alpine
RUN apk update && apk add --no-cache \
    python2 \
    uwsgi-python3 \
    py-setuptools \
    gcc \
    libc-dev \
    linux-headers \
    libuuid \
    pcre \
    mailcap \
```

```
    pcre-dev \
    make \
    && rm -rf /tmp/*
# 对一个纯净的Alpine来说，uWSGI有gcc、libc、linux-headers三个编译依赖。
# 编译完成后如果删除依赖，则还需要一个运行时依赖libuuid。
WORKDIR /code/supervisor-3.3.4
RUN python2 setup.py install \
    && pip3 install -r \
    /code/SharkAPAMP/requirements.txt \
    && apk del \
    gcc \
    libc-dev \
    linux-headers
```