**久**

**杨**

**珍**

**精**

**（下）**

**云原生与DevOps**

悟尽归原 几近于道。

create by Yangcl

2023-03-10

Nginx高级特性与实战

1、Nginx基础功能使用与概念

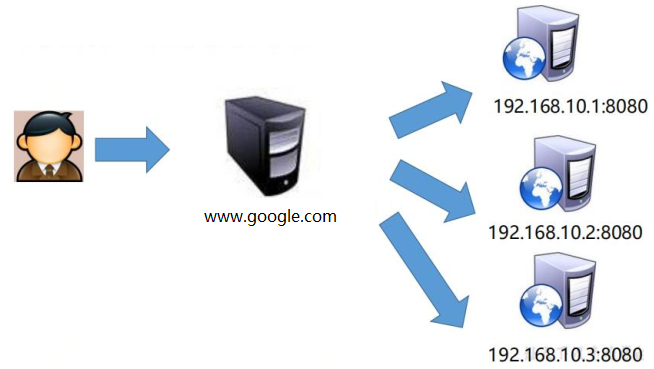
Nginx("engine x")是一款是由俄罗斯的程序设计师Igor Sysoev所开发高性能的 Web和 反向代理服务器，也是一个 IMAP/POP3/SMTP 代理服务器。在高连接并发的情况下，Nginx是Apache服务器不错的替代品。它能包含了如下基础功能：正向代理、反向代理、负载均衡和动静分离。

官网地址：http://nginx.org/

**【正向代理】**



**【反向代理】与【负载均衡】**



【反向代理】客户端无感知，具体请求由反向代理服务器转发，从而隐藏真实的Ip地址。

【负载均衡】将原先请求集中到单个服务器上的情况改为将请求分发到多个服务器上，即：将负载分发到不同的服务器，也就是我们所说的负载均衡。

【动静分离】

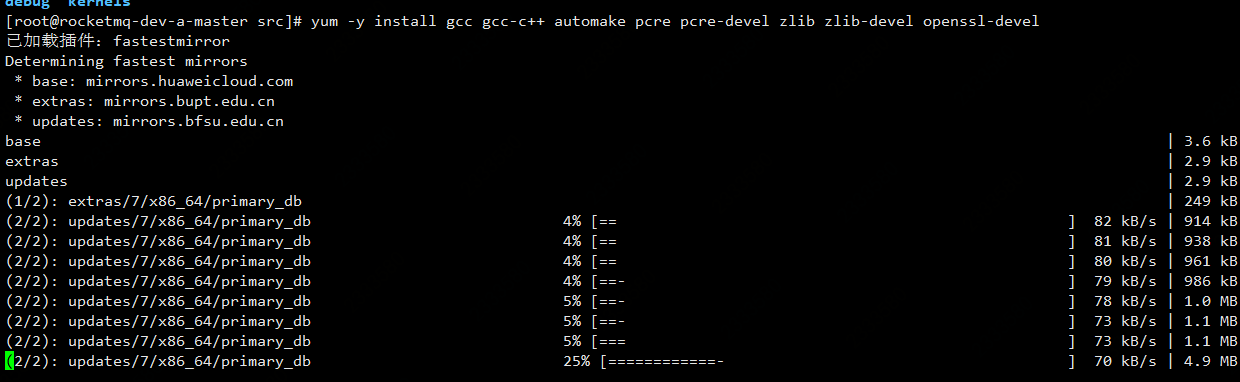


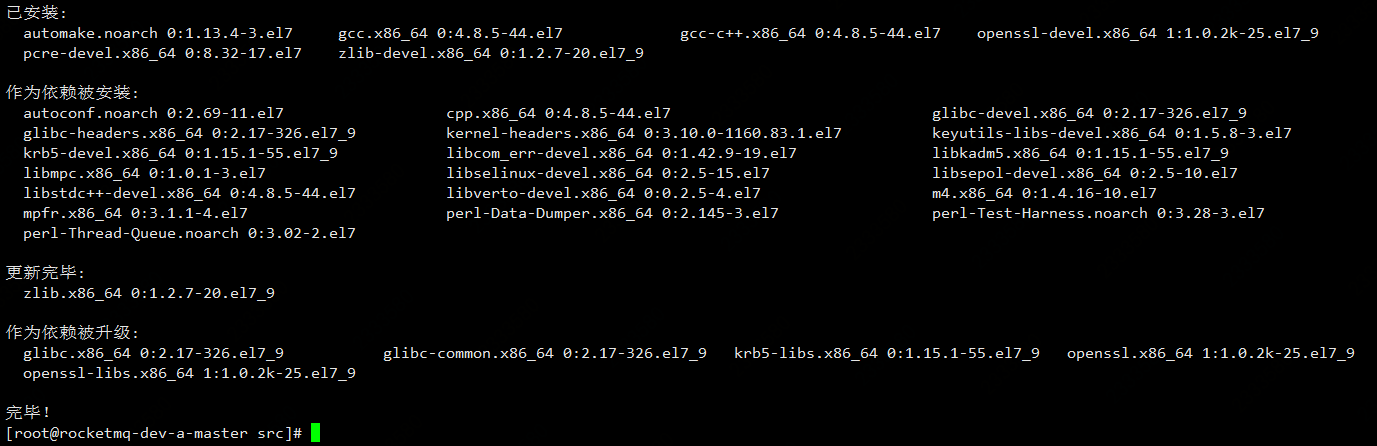
为了加快网站的解析速度，可以把动态请求(http 接口)和静态页面(js/css等静态资源)由不同的服务器来解析，加快解析速度，降低原来单个服务器的压力。

1.1 Nginx基础安装过程

1.1.1 安装基础依赖

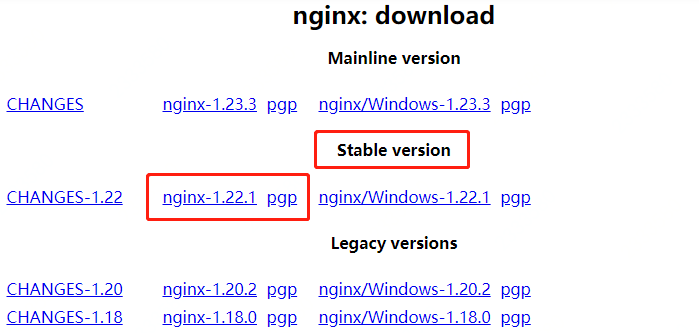
yum -y install gcc gcc-c++ automake pcre pcre-devel zlib zlib-devel openssl-devel



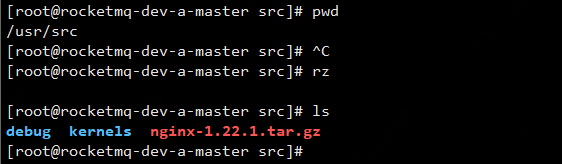


1.1.2 下载nginx稳定版本 解压后进行安装

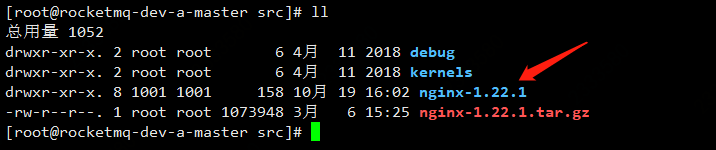
登录官网：http://nginx.org/en/download.html

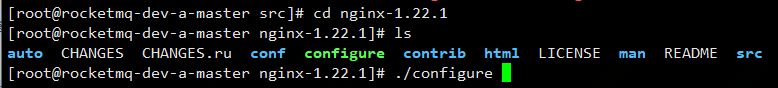


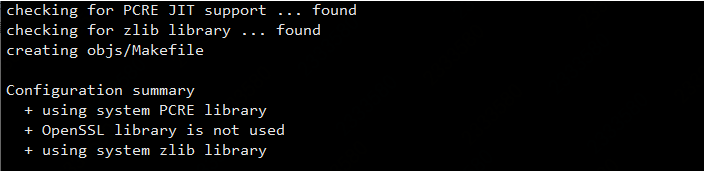
上传到：**/usr/src** 目录下。

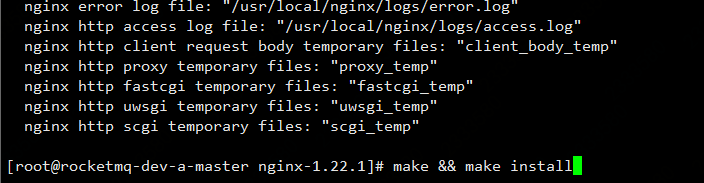


执行命令：tar -zxvf nginx-1.22.1.tar.gz 解压到当前文件夹。

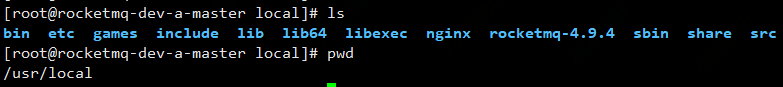








主目录执行命令 ./configure 此命令主要用来执行check操作，完成后回到目录下执行 make && make install 。nginx将会被安装到**/usr/local/nginx**目录下，文件压缩包的下载路径和安装路径是不同的。



1.1.3 开放防火墙端口

查看开放的端口号

firewall-cmd --list-all

设置开放的端口号

firewall-cmd --add-service=http –permanent

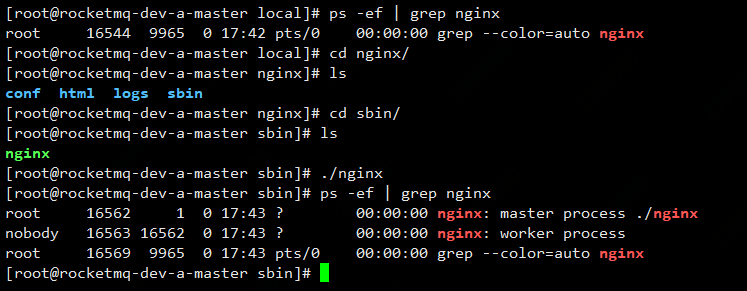
sudo firewall-cmd --add-port=80/tcp --permanent

重启防火墙

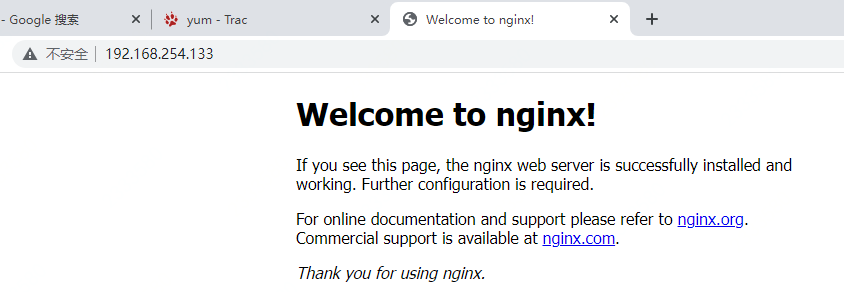
firewall-cmd –reload

1.1.4 启动nginx同时访问欢迎页

进入/usr/local/nginx/sbin目录下执行命令：./nginx 启动程序。



查看默认页面



至此，nginx基础安装过程已经完成。

1.2 Nginx常用命令

查看版本信息：在/usr/local/nginx/sbin 目录下执行 ./nginx -v

启动nginx：在/usr/local/nginx/sbin 目录下执行 ./nginx

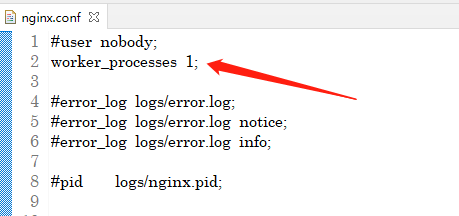
关闭nginx：在/usr/local/nginx/sbin 目录下执行 ./nginx -s stop

重加载nginx：在/usr/local/nginx/sbin 目录下执行 ./nginx -s reload

1.3 Nginx配置文件详解

他的核心配置文件位于：/usr/local/nginx/conf目录下，打开nginx.conf文件。

1.3.1 全局块



从配置文件开始到 events 块之间的内容，主要会设置一些影响 nginx 服务器整体运行的配置指令，主要包括配置运行 Nginx 服务器的用户（组）、允许生成的 worker process 数，进程 PID 存放路径、日志存放路径和类型以及配置文件的引入等。比如：worker\_processes 1; 这是Nginx服务器并发处理服务的关键配置，worker\_processes值越大，可以支持的并发处理量也越多，但是会受到硬件、软件等设备的制约，通常worker\_processes会设置为当前机器的内核数。

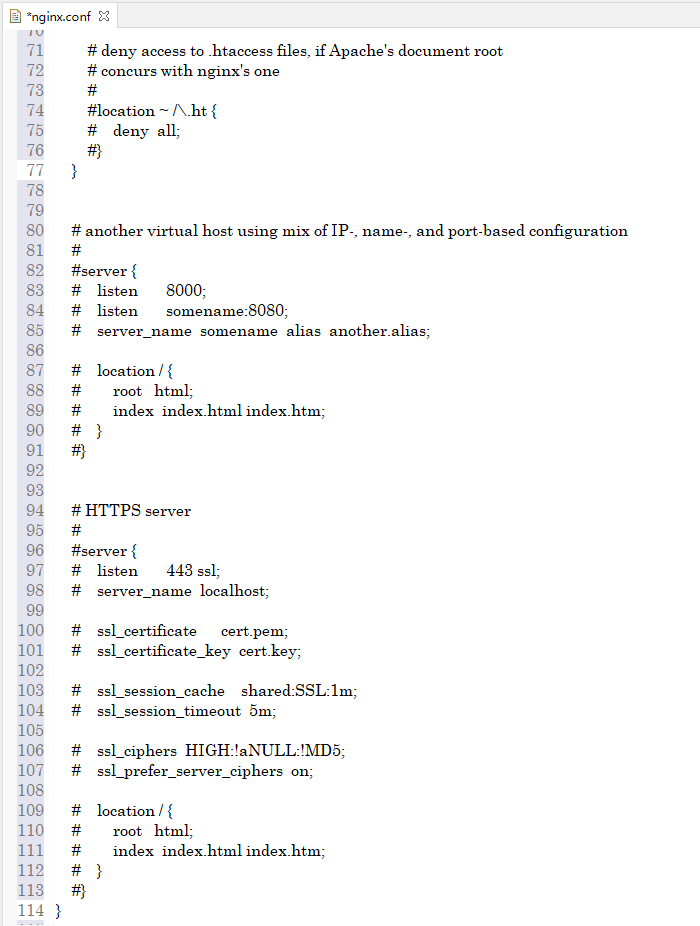
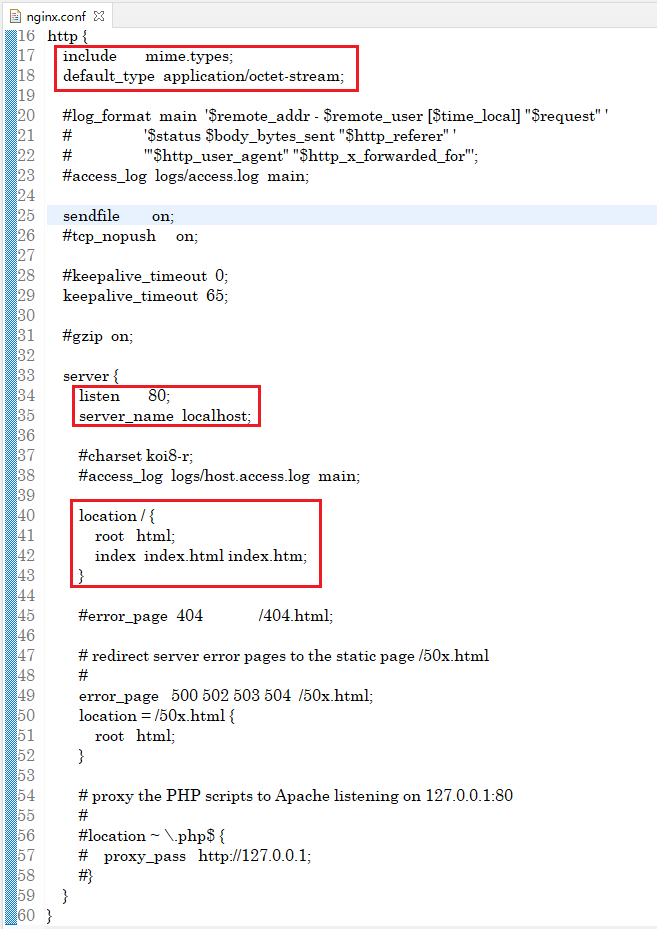
1.3.2 events块



events 块涉及的指令主要影响 Nginx 服务器与用户的网络连接，常用的设置包括是否开启对多work process下的网络连接进行序列化，是否允许同时接收多个网络连接，选取哪种事件驱动模型来处理连接请求，每个 work process 可以同时支持的最大连接数等。默认每个work process支持的最大连接数为1024，这部分的配置对Nginx的性能影响较大，在实际中应该灵活配置。

1.3.3 http块

这是Nginx服务器配置中最频繁的部分，代理、缓存和日志定义等绝大多数功能和第三方模块的配置都在这里。



需要注意的是：http块也可以包括http全局块、server 块。

* http全局块

http全局块配置的指令包括文件引入、MIME-TYPE 定义、日志自定义、连接超时时间、单链接请求数上限等。

* server块

这块和虚拟主机有密切关系，虚拟主机从用户角度看，和一台独立的硬件主机是完全一样的，该技术的产生是为了节省互联网服务器硬件成本。每个 http 块可以包括多个 server 块，而每个 server 块就相当于一个虚拟主机。而每个server块也分为全局server块，以及可以同时包含多个locaton 块。

【全局server块】最常见的配置是本虚拟机主机的监听配置和本虚拟主机的名称或 IP 配置。

【location块】支持正则表达式进行路径分隔、转发等。

一个server 块可以配置多个location块。这块的主要作用是基于Nginx服务器接收到的请求字符串（例如：server\_name/uri-string），对虚拟主机名称(也可以是IP别名)之外的字符串（例如：前面的/uri-string）进行匹配，对特定的请求进行处理。地址定向、数据缓存和应答控制等功能，还有许多第三方模块的配置也在这里进行。

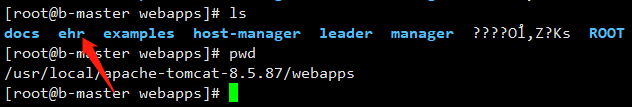
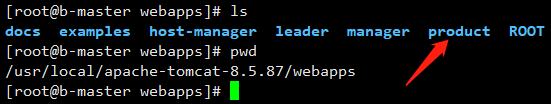
1.4 反向代理示例

1.4.1 准备工作

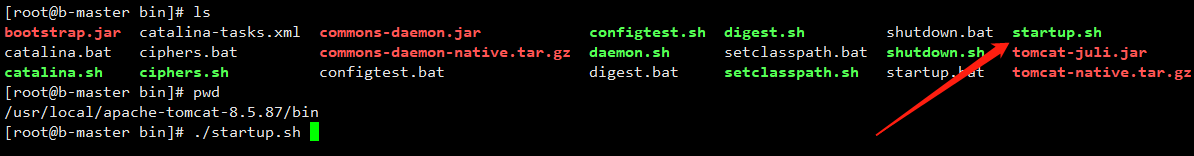
统一部署路径为：/usr/local/apache-tomcat-8.5.87/webapps

192.168.254.128：此台虚拟机部署tomcat服务器，存放前端页面ehr项目，即：/usr/local/apache-tomcat-8.5.87/webapps/ehr/

192.168.254.129：此台虚拟机部署tomcat服务器，存放前端页面product项目，即：/usr/local/apache-tomcat-8.5.87/webapps/product/

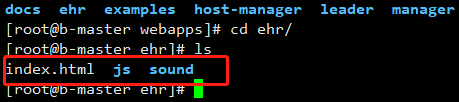
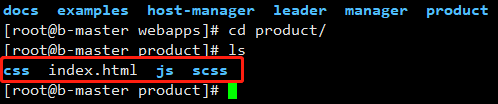
启动tomcat，同时验证两台虚拟机是否可以正常访问项目。



192.168.254.128:8080/ehr/index.html

192.168.254.129:8080/product/index.html

所部署项目的真实结构：

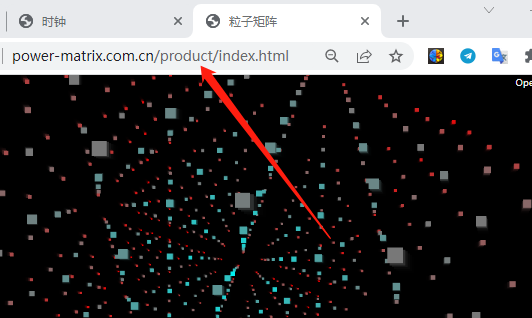
1.4.2 配置反向代理

192.168.254.133：此台虚拟机部署nginx服务，准备修改nginx.conf文件。

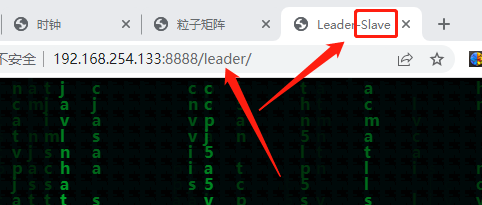
1. worker\_processes  1;
2. events {
3. worker\_connections  1024;
4. }
6. http {
7. include       mime.types;
8. default\_type  application/octet-stream;
10. sendfile        on;
11. keepalive\_timeout  65;
13. server {  # 第一个server块：监听域名
14. listen       80;
15. server\_name  power-matrix.com.cn;
17. location ~/**ehr**/ {
18. proxy\_pass http://192.168.254.128:8080;  #此处无需增加：/ehr/index.html
19. }
20. location ~/**product**/ {
21. proxy\_pass http://192.168.254.129:8080;  #此处无需增加：/product/index.html
22. }
23. error\_page   500 502 503 504  /50x.html;
24. location = /50x.html {
25. root   html;
26. }
27. }
29. server {  # 第二个server块：监听端口
30. listen       8888;
31. server\_name  192.168.254.133;   # 监听本机IP地址的8888端口
33. location / { # 如果不加入任何过滤条件，则展示默认nginx欢迎页
34. root   html;
35. index  index.html index.htm;
36. }
37. location ~/**leader**/ {    # 如果包含关键字，则执行配置规则
38. proxy\_pass http://192.168.254.129:8080;  #/此处无需增加：leader/index.html
39. }
40. error\_page   500 502 503 504  /50x.html;
41. location = /50x.html {
42. root   html;
43. }
44. }
45. }

配置完成后，启动nginx服务

在浏览器中输入：http://power-matrix.com.cn/ehr/ 或 http://power-matrix.com.cn/ehr/index.html都可以访问到页面

在浏览器中输入：http://192.168.254.133:8888/leader/ 或 http://192.168.254.133:8888后：

192.168.254.129虚拟机中leader目录下正是放置的Slave页面。

1.5 location指令详解

此处参考：<https://zhuanlan.zhihu.com/p/137146854>，非常透彻的讲解，特此收录。

Nginx中的块级指令，location指令的功能是用来匹配不同的url请求，进而对请求做不同的处理和响应。明确一些约定，我们输入的网址叫做请求URI，nginx用请求URI与location中配置的URI做匹配。location有两种匹配规则：

1. 匹配URL类型，有四种参数可选，当然也可以不带参数：location [ = | ~ | ~\* | ^~ ] uri { … }
2. 命名location，用@标识，类似于定于goto语句块：location @name { … }

1.5.1 location匹配参数解释

* “=”精确匹配

内容要同表达式完全一致才匹配成功。

1. location = /abc/ {
2. .....
3. }
5. # 只匹配http://power-matrix.com.cn/abc
6. # http://power-matrix.com.cn/abc [匹配成功]
7. # http://power-matrix.com.cn/abc/index [匹配失败]

* “~”执行正则匹配

区分大小写。

1. location ~ /Abc/ {
2. .....
3. }
4. # http://power-matrix.com.cn/Abc/ [匹配成功]
5. # http://power-matrix.com.cn/abc/ [匹配失败]

* “~\*”执行正则匹配

忽略大小写。

1. location ~\* /Abc/ {
2. .....
3. }
4. # 则会忽略 uri 部分的大小写
5. # http://power-matrix.com.cn/Abc/ [匹配成功]
6. # http://power-matrix.com.cn/abc/ [匹配成功]

* “^~”普通字符串匹配

表示普通字符串匹配上以后不再进行正则匹配。

1. location ^~ /index/ {
2. .....
3. }
4. # 以 /index/ 开头的请求，都会匹配上
5. # http://power-matrix.com.cn/index/index.page  [匹配成功]
6. # http://power-matrix.com.cn/error/error.page [匹配失败]

* 普通前缀匹配

不加任何规则时，优先级低于带参数前缀匹配；默认是大小写敏感、前缀匹配，相当于加了“~”与“^~”。

1. location /index/ {
2. ......
3. }
4. # http://power-matrix.com.cn/index  [匹配成功]
5. # http://power-matrix.com.cn/index/index.page  [匹配成功]
6. # http://power-matrix.com.cn/test/index  [匹配失败]
7. # http://power-matrix.com.cn/Index  [匹配失败]
8. # 匹配到所有uri

* “@” nginx内部跳转

一种较少使用的规则。

1. location /index/ {
2. error\_page 404 @index\_error;
3. }
4. location @index\_error {
5. .....
6. }
7. #以 /index/ 开头的请求，如果链接的状态为 404。则会匹配到 @index\_error 这条规则上。

1.5.2 location匹配顺序(优先级)

匹配优先级：【=】高于 【^~】 高于 【~】【~\*】高于 【最长前缀匹配】高于【/】。【=】的匹配优先级最高，一旦匹配成功则不再查找其他匹配项。

常见规则举例如下：

1. location = /  {
2. #规则A
3. }
5. location = /login {
6. #规则B
7. }
9. location ^~ /static/ {
10. #规则C
11. }
13. location ~ \.(gif|jpg|png|js|css)$ {
14. #规则D区分大小写，【$】正则含义为：匹配输入字符串的结尾位置，即以git、jpg、png等等这些后缀结尾会被匹配。
15. }
17. location ~\* \.png$ {
18. #规则E：不区分大小写，且以png结尾。
19. }
21. location !~ \.xhtml$ {
22. #规则F：区分大小写，且不能是以xhtml结尾的后缀会被匹配。
23. }
25. location !~\* \.xhtml$ {
26. #规则G：不区分大小写，且不能是xhtml后缀
27. }
29. location / {
30. #规则H任何没有匹配成功的，都会匹配这里处理。
31. }

访问根目录/， 比如http://localhost/ 将匹配规则A。

访问 http://localhost/login 将匹配规则B，http://localhost/register 则匹配规则H。

访问 http://localhost/static/a.html 将匹配规则C。

访问 http://localhost/b.jpg 将匹配规则D和规则E，但是规则D顺序优先，规则E不起作用；而 http://localhost/static/c.png 则优先匹配到规则C。

访问 http://localhost/a.PNG 则匹配规则E，而不会匹配规则D，因为规则E不区分大小写。

访问 http://localhost/a.xhtml 不会匹配规则F和规则G；

访问 http://localhost/a.XHTML不会匹配规则G，因为不区分大小写。规则F、规则G属于排除法，符合匹配规则但是不会匹配到。

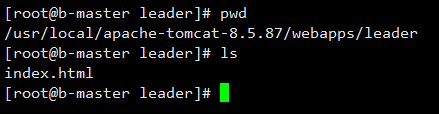
访问 http://localhost/qll/id/1111 则最终匹配到规则H，因为以上规则都不匹配。

1.6 负载均衡配置

负载均衡在配置上不同于反向代理，我们将以如下方式来展示：

192.168.254.128:8080/leader/index.html 这台虚拟机中的index.html 将其文件设置为：<title>Leader-Master</title>

192.168.254.129:8080/leader/index.html 这台虚拟机中的index.html 将其文件设置为：<title>Leader-Slave</title>

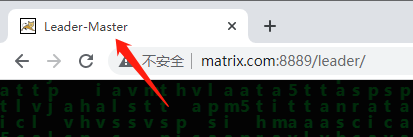
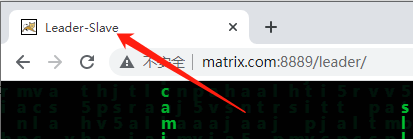
nginx.conf配置文件如下，分别展示基于权重和ip hash这两种个情况的负载均衡：

1. worker\_processes  1;
2. events {
3. worker\_connections  1024;
4. }
6. http {
7. include       mime.types;
8. default\_type  application/octet-stream;
10. sendfile        on;
11. keepalive\_timeout  65;

14. # 演示负载均衡基于【权重】  负载均衡需要用到upstream
15. upstream leaderLoadBalence{
16. server 192.168.254.128:8080 weight=1;
17. server 192.168.254.129:8080 weight=2;
18. }
20. server {
21. listen       8889;
22. server\_name  matrix.com;    # 监听新的域名，不是.com.cn，需要配置host:192.168.254.133 power-matrix.com
24. location / {
25. proxy\_pass http://leaderLoadBalence;
26. }
27. }
29. # 演示负载均衡基于【ip\_hash】，浏览器访问：http://matrix.com:8890/leader/
30. upstream leaderLoadBalenceIphash{
31. ip\_hash;
32. server 192.168.254.128:8080;
33. server 192.168.254.129:8080;
34. }
36. server {
37. listen       8890;
38. server\_name  matrix.com;    # 监听新的域名，需要配置host:192.168.254.133 matrix.com
40. location / {
41. proxy\_pass http://leaderLoadBalenceIphash;
42. root   html;
43. index  index.html index.htm;
44. }
45. }
46. }

1.6.1基于【权重】的负载均衡

如果不加入权重的参数设置，那么nginx会按照轮训的方式来分发请求。轮训是nginx默认的转发策略，每个请求按照时间顺序逐一分配到不同的后端服务器，如果后端服务器宕机，nginx能将其自动剔除。访问：http://matrix.com:8889/leader/后，负载均衡如下图所示：

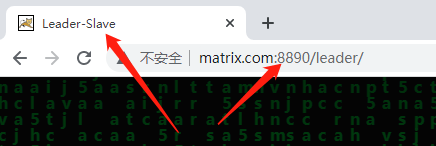
加入weight参数后意味着你选择了权重的方式来做负载均衡，weight的值越高意味着访问转发到这台机器上的概率会越高。

server 192.168.254.128:8080 weight=1;

server 192.168.254.129:8080 weight=2;

1.6.2基于【ip\_hash】的负载均衡

每个请求会按照ip地址的hash结果来分配，这样处理后每个访客会固定访问一个后端服务器，从一定程度上解决了session共享的问题，但这种解决方式很鸡肋。



如上图所示，在本机上请求http://matrix.com:8890/leader/，无论如何刷新，请求都会一直转发到Leader-Slave节点。

1.6.3 基于【fair】

按后端服务器的响应时间来分配请求，响应时间短的优先分配，但需要安装第三方模块。

1. upstream leaderLoadBalenceIphash{
2. fair;
3. server 192.168.254.128:8080;
4. server 192.168.254.129:8080;
5. }

1.7 动静分离配置

Nginx动静分离简单来说就是把动态跟静态请求分开，不能理解成只是单纯的把动态页面和静态页面物理分离。动态请求跟静态请求分开，可以理解成使用Nginx处理静态页面与相关资源(如网站图标、css、js脚本等)；Tomcat处理接口请求，因为这部分任务通常需要查询数据库，然后组装后给出结果，故称为动态请求。

动静分离从目前实现角度来讲大致分为两种，一种是纯粹把静态文件独立成单独的域名，放在独立的服务器上，也是目前主流推崇的方案；另外一种方法就是动态跟静态文件混合在一起发布，通过nginx来分开。通过location指定不同的后缀名实现不同的请求转发；再结合expires参数设置浏览器缓存过期时间，从而减少与服务器之间的静态请求和流量。具体 expires 定义：是给一个资源设定一个过期时间，也就是说无需去服务端验证，直接通过浏览器自身确认是否过期即可，所以不会产生额外的流量。此种方法非常适合不经常变动的资源。（如果经常更新的文件，不建议使用expires来缓存）。

环境准备如下：

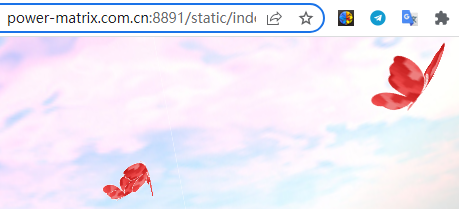
/usr/service-file/static 存放静态html文件。

/usr/service-file/images 存放静态图片。

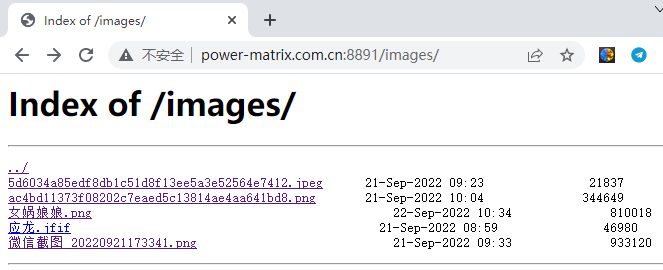
nginx.conf配置文件如下：

1. worker\_processes  1;
2. events {
3. worker\_connections  1024;
4. }
6. http {
7. include       mime.types;
8. default\_type  application/octet-stream;
9. charset utf-8;    # 防止文件出现中文乱码
10. sendfile        on;
11. keepalive\_timeout  65;
13. server {  # 动静分离验证
14. listen       8891;
15. server\_name  power-matrix.com.cn;
17. location /static/ {  # 存放静态html文件、css文件等，访问：http://power-matrix.com.cn:8891/static/index.html
18. root /usr/service-file/;
19. index  index.html index.htm;
20. }
22. location /images/ {  # 存放图片文件，访问：http://power-matrix.com.cn:8891/images/
23. root /usr/service-file/;
24. autoindex on;
25. }
26. }
27. }

访问：http://power-matrix.com.cn:8891/static/index.html效果如下：



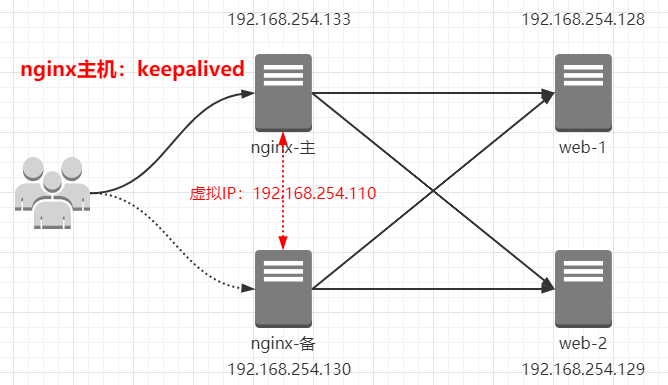
访问：http://power-matrix.com.cn:8891/images/ 效果如下：

因为设置了autoindex on; 所以这里列出了文件列表。

1.8 nginx高可用集群配置

1.8.1 基础示例架构图



需要准备两台虚拟机来做高可用配置，所以新增节点：192.168.254.130，同时安装好nginx环境和服务。

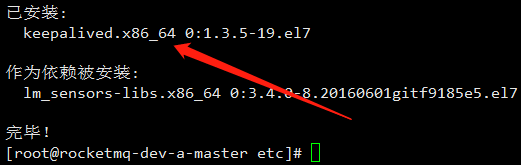
1.8.2 安装步骤与配置过程

Nginx高可用集群配置中需要用到Keepalived，它是一个基于VRRP协议来实现的服务高可用方案，可以利用其来避免IP单点故障，类似的工具还有heartbeat、corosync、pacemaker。但是它一般不会单独出现，而是与其它负载均衡技术（如lvs、haproxy、nginx）一起工作来达到集群的高可用。

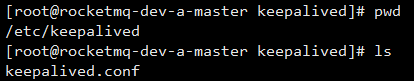
VRRP全称 Virtual Router Redundancy Protocol，即 虚拟路由冗余协议。可以认为它是实现路由器高可用的容错协议，即将N台提供相同功能的路由器组成一个路由器组(Router Group)，这个组里面有一个master和多个backup，但在外界看来就像一台一样，构成虚拟路由器，拥有一个虚拟IP（vip，也就是路由器所在局域网内其他机器的默认路由），占有这个IP的master实际负责ARP相应和转发IP数据包，组中的其它路由器作为备份的角色处于待命状态。master会发组播消息，当backup在超时时间内收不到vrrp包时就认为master宕掉了，这时就需要根据VRRP的优先级来选举一个backup当master，保证路由器的高可用。参考：[*https://developer.aliyun.com/article/47355*](https://developer.aliyun.com/article/47355)*。*

* 安装keepalived插件

执行：yum install keepalived -y



进入/etc/keepalived文件夹，找到keepalived.conf文件。keepalived服务监听的是你这台机器是否已经宕机，不是nginx服务是否已经挂掉。



核心配置展示如下：

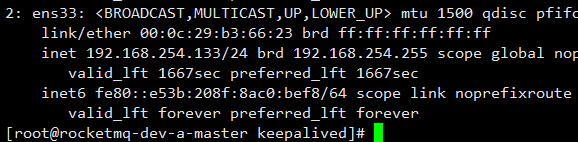
1. ! Configuration File **for** keepalived
2. # 全局定义
3. global\_defs {
4. notification\_email {         #邮件告警
5. 794867067@qq.com
6. }
7. #notification\_email\_from xxxx@richinfo.cn   #设置邮件的发送地址
8. #smtp\_server 192.168.254.133     # 设置邮件的发送地址
9. #smtp\_connect\_timeout 30     # 设置连接smtp server的超时时间
10. router\_id 192.168.254.133    #   设置为本机当前地址即可
11. }
13. # 自定义探活脚本
14. vrrp\_script check\_http\_port {
15. script "/usr/local/src/nginx\_master\_slave\_alive\_check.sh" # 脚本存放路径
16. interval 10 # 检测脚本执行时间间隔
17. weight -20  # 权重设置
18. }
20. # 虚拟IP配置
21. vrrp\_instance VI\_1 {
22. state MASTER            # 备份服务器上将MASTER改为BACKUP
23. interface ens33         # 当前机器主网卡eth0的名称
24. virtual\_router\_id 51    # 主备机器的虚拟路由ID，必须相同
25. priority 100            # 主备优先级设置，主机值较大，备机较小，备机写为90
26. advert\_int 1
27. authentication {        # 验证规则：密码方式
28. auth\_type PASS
29. auth\_pass 1111
30. }
31. virtual\_ipaddress {     # VRRP虚拟IP地址
32. 192.168.254.110
33. }
34. }

自定义探活脚本：

1. #!/bin/bash
2. A=`ps -C nginx –no-header | wc -l`
3. **if** [ $A -eq 0 ];then
4. /usr/local/nginx/sbin/nginx
5. sleep 2
6. **if** [ `ps -C nginx --no-header |wc -l` -eq 0 ];then
7. killall keepalived
8. fi
9. fi

* 虚拟IP地址绑定状态

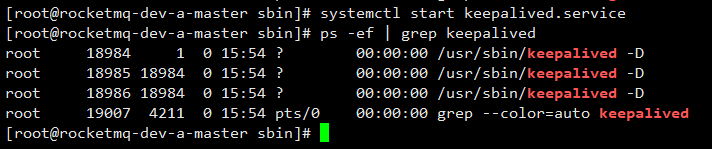
此时尚未绑定任何虚拟IP地址。



* 集群启动步骤

1、先启动nginx服务

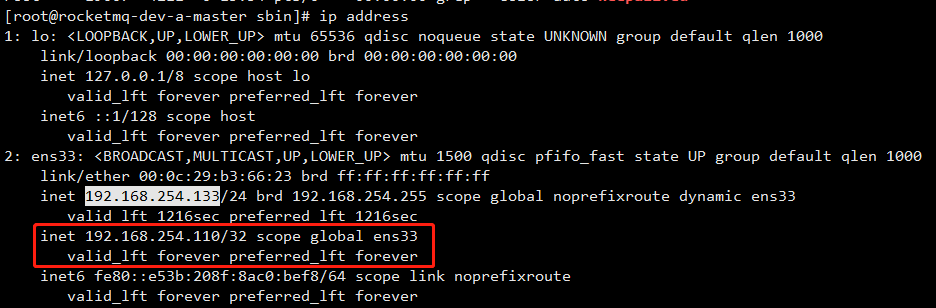
2、再启动keepalived服务：systemctl start keepalived.service

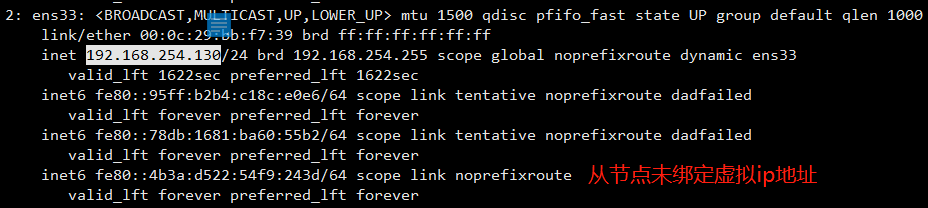


* 启动状态验证

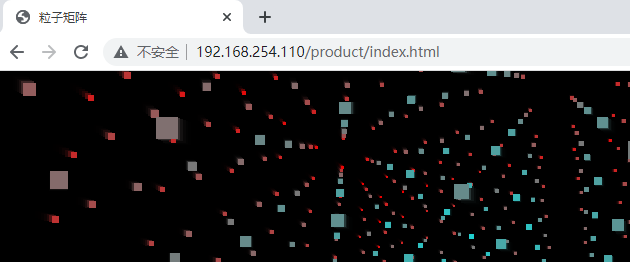
由于192.168.254.133这台机器是主nginx节点，所以当我们输入ip address后能够看到已经绑定的虚拟ip地址，但是nginx从节点是尚未绑定的。

只有当主节点宕机，切换到从节点，这时从节点才会绑定下图的虚拟ip地址。

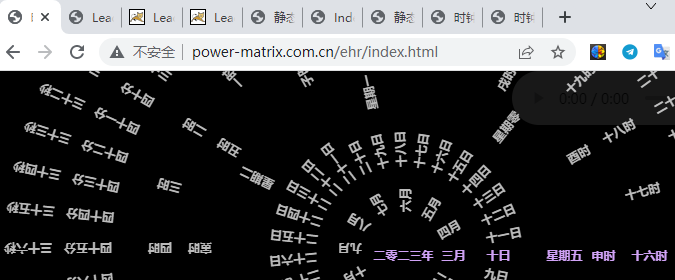




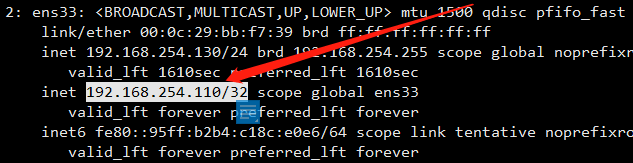
访问：http://192.168.254.110/product/index.html



将虚拟IP地址配置host：192.168.254.110 power-matrix.com.cn



尝试关闭主节点的nginx服务后(注意！需要同时关闭keepalived服务)，查看从节点绑定关系：



从上图可见，从节点在主节点下线宕机以后，自动绑定了虚拟IP地址，同时网页访问正常，至此证明nginx高可用集群配置成功。

如果此时再次启动主节点，从节点这里的虚拟ip绑定关系会自动解除。

2、Nginx高级特性：亿级流量架构设计

云原生与DevOps

云原生的概念可以分为两部分来理解；原生的概念：用任意编程语言(比如Java、Python、C++、vue、Go等)开发的应用，称为原生应用。当我们选择部署到云上时就被称为云原生，所以云原生指的就是：原生应用上云的过程，以及云上的一系列解决方案。

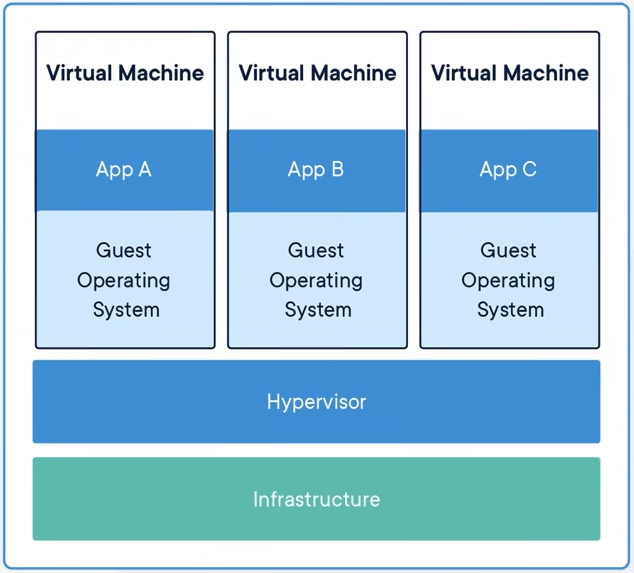
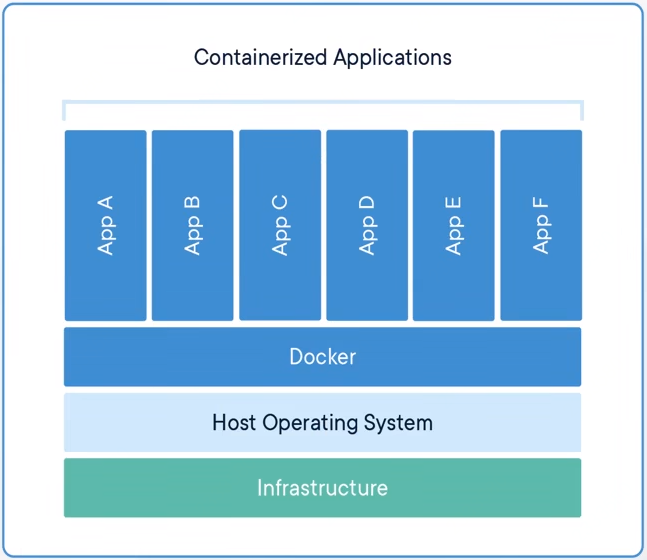
云原生是基于分布部署和统一运管的一套云技术产品体系，以容器、微服务、DevOps等技术为基础建立。使用云原生技术后，开发者无需考虑底层的技术实现，可以充分发挥云平台的弹性和分布式优势，实现快速部署、按需伸缩、不停机交付等。

云原生更侧重发挥kubernetes平台的能力，选择谁的开源产品更显得重要，本文档将着重围绕KubeSphere平台来展开介绍。

Docker In Action

1、docker基础概念

在Docker出现之前，一直使用虚拟机来做资源隔离，这种方式以强隔离性著称，可以有效防止因为一个应用的内存泄露而引起整台物理机下线。

【虚拟机架构】 【容器化架构】

但这种方式有如下弊端：

1、基础镜像至少在GB以上的级别，过于庞大，导致移植与分享都不是很方便；

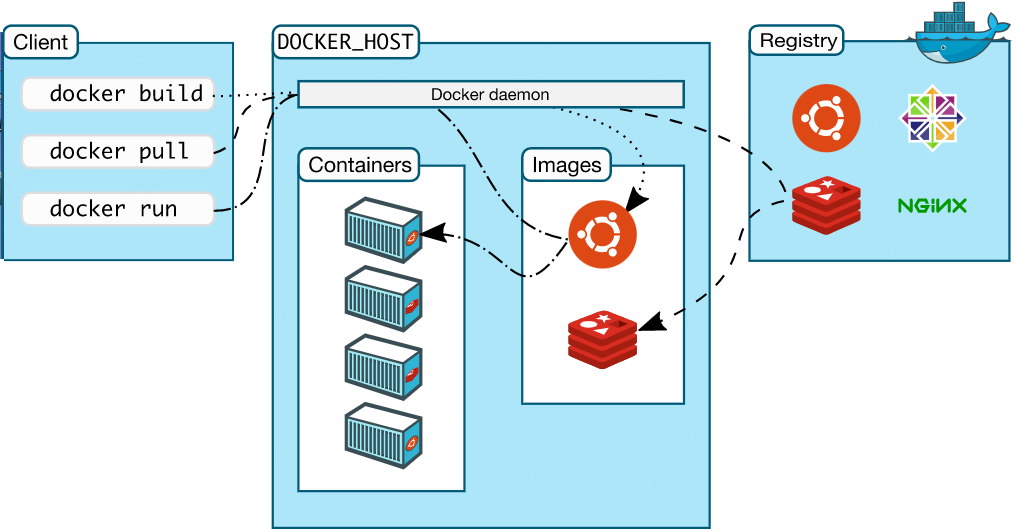
2、启动速度比较慢；

随着docker的出现，在依旧强调【资源隔离】的同时，还解决了如下问题：

1、MB级别大小，利于移植和分享，Linux基础镜像只有5M左右。镜像都会上传到统一的位置：docker hub，类似安卓的应用市场。

2、应用构建：Java/C++/Go/java script/pythen，全部通过docker build编译成镜像，不再区分其最终可执行结果是\*.jar还是\*.exe，屏蔽语言差异。

3、应用运行：极速启动；统一的标准镜像，使用docker run来运行。



【docker架构图】

Docker\_Host：安装docker的主机。

Docker Daemon：运行在docker主机上的docker后台进程。作为Server端接收Docker客户端的请求，并负责创建、运行和分发Docker容器。

Client：操作Docker主机的客户端（命令行、UI等）。

Registry：镜像仓库，docker hub。

Images：镜像，带环境打包好的程序，可以直接启动运行。

Containers：容器，由镜像启动起来正在运行中的程序。

Linux基金会做了容器化技术，是一种轻量级的虚拟化技术，目的和虚拟机一样，都是为了创造“隔离环境”。但是它不像VMware虚拟机那样采用操作系统级的资源隔离，而是采用的是进程级的系统隔离。docker是一种创建容器的主流工具，所以docker并不等于容器，认知到这一点非常重要。在虚拟机的时代，每一台虚拟机都是一个完整的操作系统；容器化技术抛弃了虚拟机的笨重设计，采用了差异化设计。既然宿主机已经有了操作系统，那么docker所保存的是那些存在差异化的内容，然后提取出来形成差异化镜像，而不是保存整个操作系统镜像，这就使得docker镜像文件整体非常小。

从【docker架构图】中可以看到，docker是围绕着容器设计的。其中最重要的一个环节就是docker镜像：它是一个特殊的文件系统，除了提供容器运行时所需的程序、库、资源、配置等文件外，还包含了一些为运行时准备的配置参数。镜像不包含任何动态数据，其内容在构建之后也不会被改变。docker可以用镜像来创建容器，用户可以使用设备上已有的镜像来安装多个相同的docker容器。

2、docker环境安装

官方文档如下：https://docs.docker.com/engine/install/centos/

2.1 移除以前docker相关包

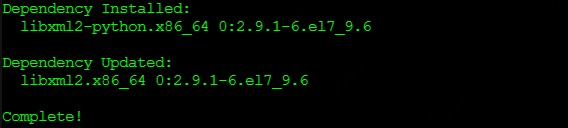
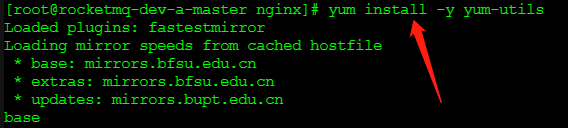
1. sudo yum remove docker \
2. docker-client \
3. docker-client-latest \
4. docker-common \
5. docker-latest \
6. docker-latest-logrotate \
7. docker-logrotate \
8. docker-engine

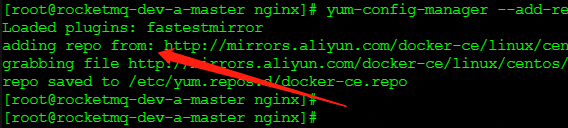
也可以使用：yum remove docker\* 全部移除。

2.2 配置yum源

1、yum install -y yum-utils 安装相关工具，-y代表自动确认，一路默认安装。

2、yum-config-manager --add-repo http://mirrors.aliyun.com/docker-ce/linux/centos/docker-ce.repo 从阿里云的镜像线路处理。

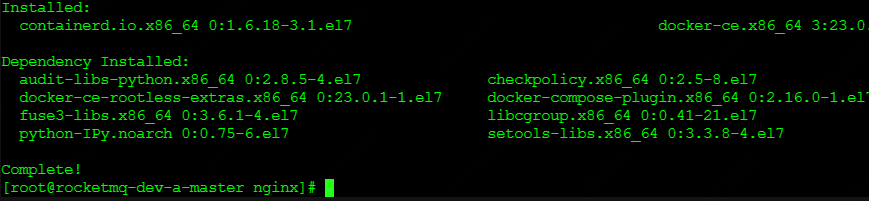




2.3 安装docker

1、yum install -y docker-ce docker-ce-cli containerd.io # 执行默认版本安装

2、yum install -y docker-ce-20.10.7 docker-ce-cli-20.10.7 containerd.io-1.4.6 # 执行指定版本安装，安装k8s的时候使用



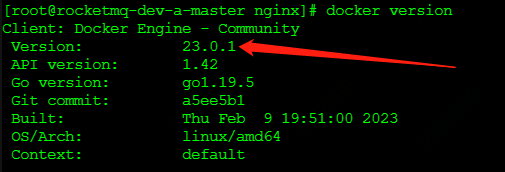
docker-ce：docker主服务程序，docker-ce为社区免费版，docker-ee为企业版(收费)。

docker-ce-cli：docker client客户端。

containerd.io：docker容器化运行环境。

2.4 设置docker开机启动

1、systemctl enable docker --now # 设置docker开机启动且现在立即启动。



Client: Docker Engine - Community：社区版23.0.1。

2.5 配置镜像下载加速器

【命令脚本如下】：

sudo mkdir -p /etc/docker

sudo tee /etc/docker/daemon.json <<-'EOF'

{

"registry-mirrors": ["https://4uwfmlks.mirror.aliyuncs.com"],

"exec-opts": ["native.cgroupdriver=systemd"],

"log-driver": "json-file",

"log-opts": {

"max-size": "100m"

},

"storage-driver": "overlay2"

}

EOF

sudo systemctl daemon-reload

sudo systemctl restart docker

全部复制后在xshell中执行。

【阿里云镜像加速器地址】：

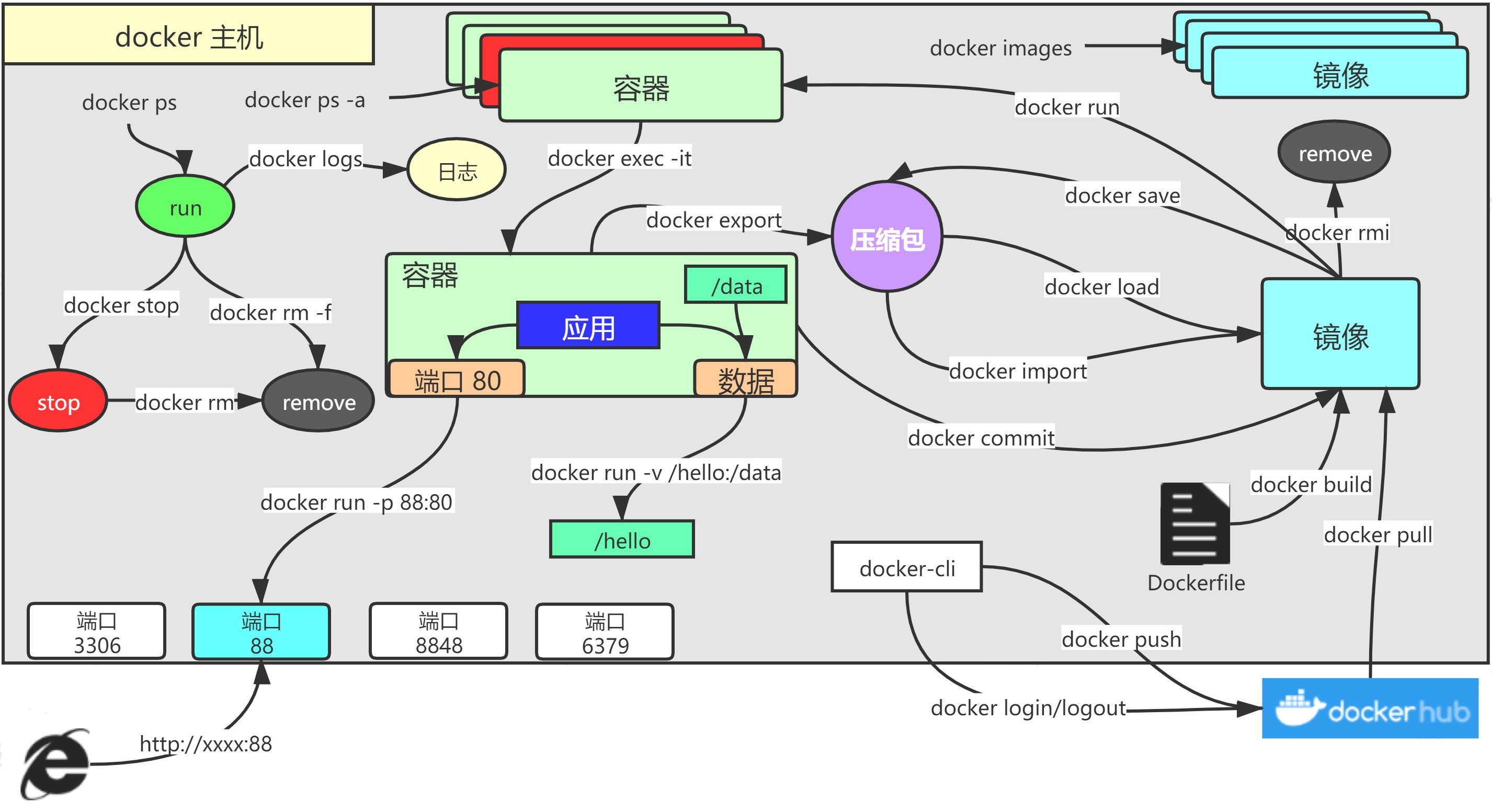






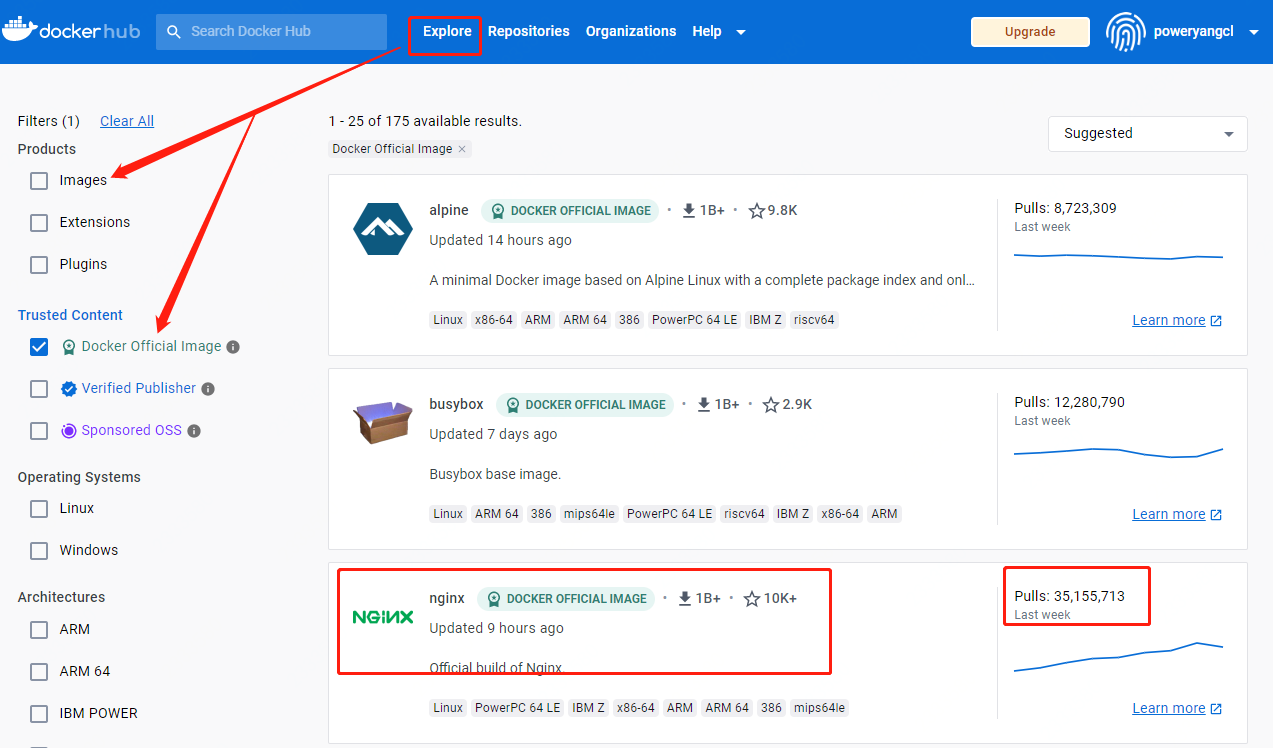
3、docker命令使用

3.1 docker核心命令图

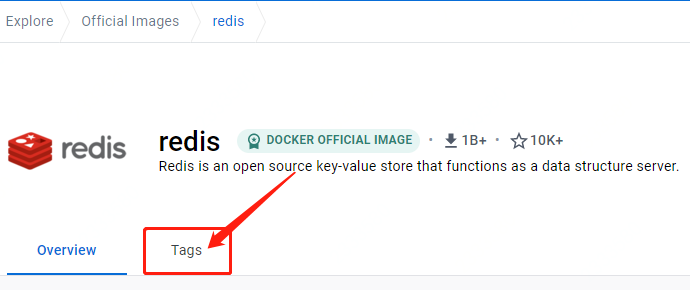


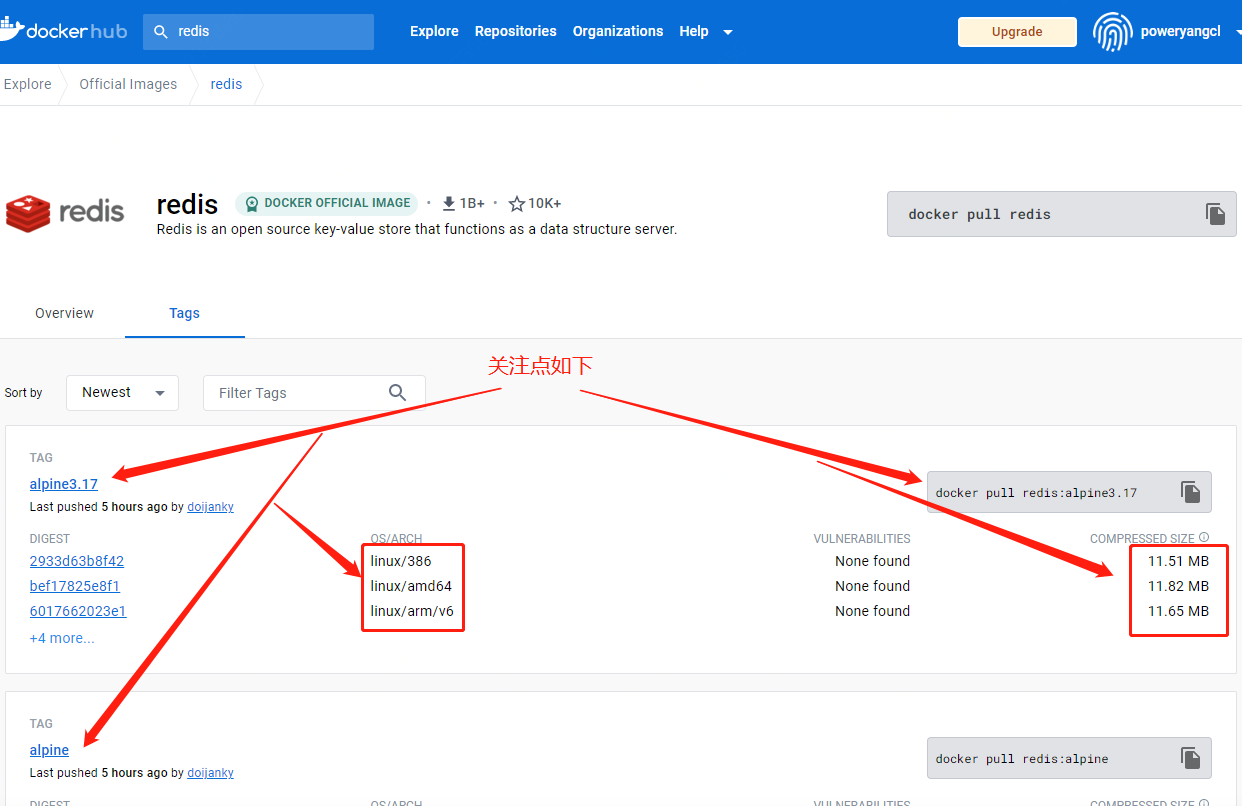
3.2 dockerHub 速通

访问：<https://hub.docker.com/>



* 搜索到redis镜像组





对于相对陈旧的版本，在列表的后面会有，但他们通常包含比较严重的安全漏洞，如下图：



3.3 docker镜像相关命令

* docker pull 下载镜像

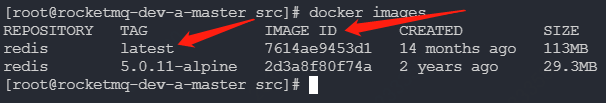
下载镜像的命令格式为：docker pull 镜像名称:版本名称(即：tag)。如果下载最新版本则直接使用docker pull 镜像名称即可。

docker pull redis #下载最新的镜像 等同于：docker pull redis:latest，如下图：



我们再下一个镜像：docker pull redis:5.0.11-alpine 做对比。

* docker images 查看本地镜像



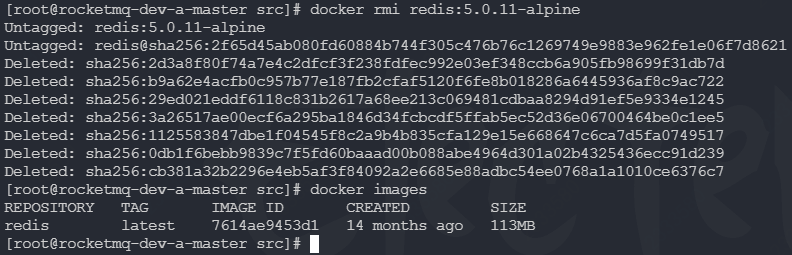
镜像名称、镜像标签、镜像ID、创建时间、镜像大小。

docker images #查看所有镜像

* docker rmi 删除镜像

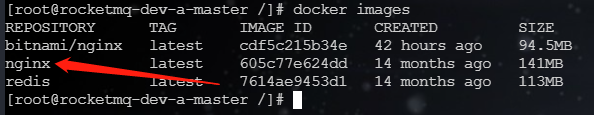
具体命令格式为：docker rmi 镜像名:版本号/镜像id。我们可以通过镜像名称+版本号的方式来删除一个镜像，也可以直接通过镜像ID来删除。

执行命令：docker rmi redis:5.0.11-alpine



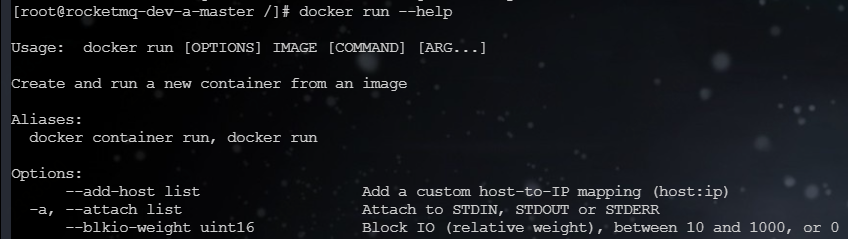
3.4 容器启动命令

下载一个bitnami(比特纳米，一个镜像贡献量较高的组织)的nginx镜像，用于测试容器运行命令。擦擦！比特纳米起不来，还是用官方的nginx镜像吧。



* docker run

通过docker run --help查看run命令所携带的参数非常多，部分截图不一一列举。



其命令格式为：docker run [OPTIONS] IMAGE [COMMAND] [ARG...]；描述为：Create and run a new container from an image。

可以理解为：【docker run 设置项 镜像名】 镜像启动运行的命令(镜像里面默认有的，一般不会写)

**【经典使用示例】**

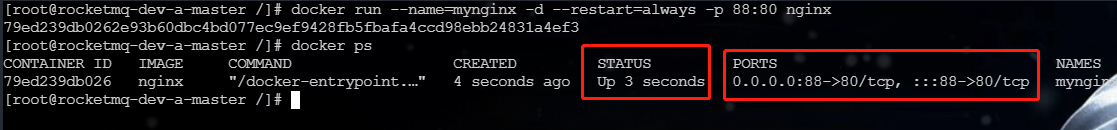
# --name：Assign a name to the container，为容器起一个名字。

# -d：--detach的缩写Run container in background and print container ID，以后台方式运行。不加-d运行容器时会阻塞控制台，控制台如果退出则容器自动销毁。

# --restart：Restart policy to apply when a container exits (default "no")，容器退出时应用的重启策略。重启服务器后，镜像会随着docker一起启动。

# -p：--publish的缩写Publish a container's port(s) to the host，执行端口映射。

docker run --name=mynginx -d --restart=always -p 88:80 nginx





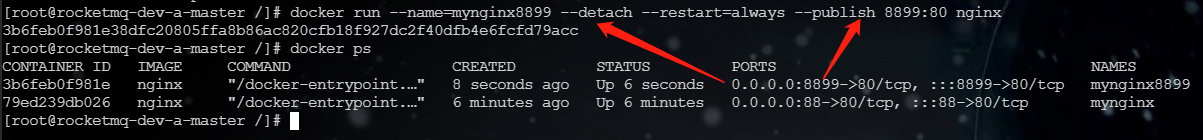
STATUS：Up，代表容器启动成功。

PORTS：0.0.0.0:88->80/tcp, :::88->80/tcp。0.0.0.0：代表接受任意IP地址的请求，无黑白名单。:::88->80/tcp：代表宿主机的88端口映射到容器的80端口。

**【全称参数与简写参数】**

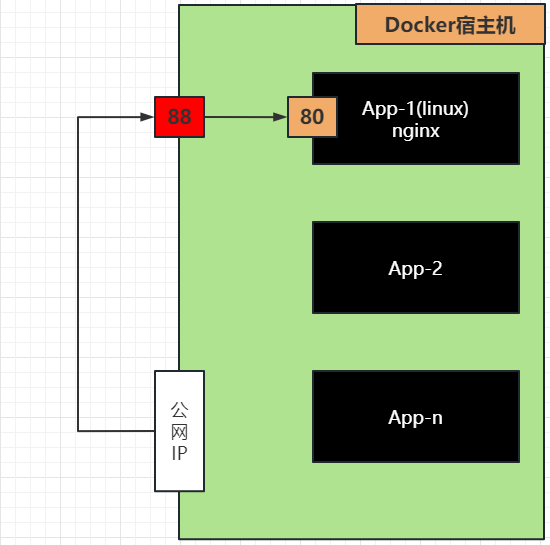
docker中一个“-”代表简写的命令参数，两个“--”的代表全称形式的参数；下面使用非简写参数来执行命令：

docker run --name=mynginx8899 --detach --restart=always --publish 8899:80 nginx





* 关于端口映射问题



https://www.yuque.com/leifengyang/oncloud/mbvigg

代码样式

http://www.codeinword.com/

|  |  |  |
| --- | --- | --- |
| 方法名 | 返回值 | 功能描述 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |
| --- | --- |
| 方法名 | 功能描述 |
|  |  |
|  |  |