运营

1. 运营很复杂，首先在 N 多的互联网公司内部，就存在着“内容运营”、“活动运营”、“用户运营”、“新媒体运营”、“产品运营”等不同岗位和职能。

2. 互联网行业内相对比较有一致共识的 4 大运营职能划分是：内容运营、用户运营、活动运营和产品运营。

(1) 内容运营：围绕着内容的生产和消费搭建起来一个良性循环，持续提升各类跟内容相关的数据，如内容数量、内容浏览量、内容互动数、内容传播数等。

(2) 用户运营：围绕着用户的新增—留存—活跃—传播以及用户之间的价值供给关系建立起来一个良性的循环，持续提升各类跟用户有关的数据，如用户数、活跃用户数、精英用户数、用户停留时间等。

(3) 活动运营：围绕着一个或一系列活动的策划、资源确认、宣传推广、效果评估等一系列流程做好全流程的项目推进、进度管理和执行落地。

(4) 产品运营：通过一系列各式各样的运营手段（比如活动策划、内外部资源拓展和对接、优化产品方案、内容组织等），去拉升某个产品的特定数据，如装机量、注册量、用户访问深度、用户访问频次、用户关系对数量、发帖量等。

3. QQ 群、小组运营：某个特定 QQ 群或线上小组的维护，包括但不限于定期群内抛话题、组织活动、活跃气氛等。

4. 平台类产品的运营是比较讲究的，因为平台的运营特别注重“节奏”。例如，淘宝或滴滴出行这样的产品，一开始无论司机或商户太多了，还是买家和乘客太多了，可能都会是问题——会导致大量该类用户得不到好的体验从而离开。

所以，平台类产品的运营会特别关注“节奏”，什么时候要拉什么人进来，拉多少合适，都很考究。

5. 平台类产品会非常注重策略和用户维系。比如淘宝商家，可能需要按地区、按售卖商品、按客单价等各种不同维度分成很多类，然后再分别对其进行维系。尤其是在平台发展大了以后，必须针对用户的不同特征进行精细化运营。

6. 目前游戏产品的运营，重点是两大块。

- 第一块，还是推广，各种对接渠道，各种看转化率，各种盯数据。

- 第二块，则是收入。比如，对于游戏中有更大付费可能的一群人，游戏公司可能会有一个专门的团队来围绕着这群人转，通过各种策略和运营手段促进这群人的付费，最俗的手段，比如冒充美女在游戏里求土豪送装备。

7. 广告的目的在于增强用户对自身品牌的认知度，而非直接产生价值。

8. 用户对一个品牌的认知形成流程有三步：

(1) 广告、用户传播；

(2) 品牌形象初步塑造；

(3) 最后品牌形象定型。

9. 很多产品的长期价值不是一蹴而就的，而是必须借由用户的持续使用和反馈，经过长期优化迭代后才能成立的。所以，要是没有用户的使用和反馈，产品很可能永远无法形成真正的长期价值。因而，在产品的长期价值还不是那么确定的时候，运营需要先通过创造一系列的短期价值让用户先能够来使用你的产品。

滴滴打车在早期上线没有用户时，一方面自己雇大量员工满北京打车，另一方面则给予乘客们大量补贴，以便能够维系住为数不多的司机和乘客们，这才有了后来的种种可能。

10. 对于任何一个早期产品来说，“烂”可能是一种再正常不过的状态，甚至可能是一种必然。这个阶段，恰恰是需要运营发挥作用和价值，维护好早期核心用户，并不断推动着产品来进行更新、迭代和打磨的时候。

11. 目标导向类工作相比纯粹的职能支持类工作，其产出的价值要高。

12. 消费用户价值的前提是创造足够的用户价值，让用户产生依赖心理。

13. 追热点、借势营销是运营人员必备技能。

14. 把自己变成一个真正的典型用户，让自己大量置身于真实用户的真实体验场景下，这样久而久之，你自然会慢慢拥有一种对于你的用户们的“洞察力”。而这样的洞察力，很多时候也会成为一个优秀运营身上不可替代的核心价值。

15. 4 个关键性“运营思维”：

- 流程化思维：一个优秀的运营和一个普通人之间会存在的一个核心差别，就是优秀的运营拿到一个问题后，会先回归到流程，先把整个问题的全流程梳理出来，然后再从流程中去寻找潜在解决方案。

- 精细化思维：就是必须要能够把自己关注的一个大问题拆解为无数细小的执行细节，并且要能够做到对于所有的这些小细节都拥有掌控力。
- 杠杆化思维：先做好做足某一件事，然后再以此为一个核心杠杆点，去撬动更多的事情和成果发生，典型的比如：我先服务好一小群种子用户，给他们制造大量超出他们预期的体验，然后我就有机会借此为杠杆，去撬动他们在此后帮助我进行品牌和产品传播的意愿（还记得小米的米粉吗）；
- 生态化思维：所谓生态，其实就是一个所有角色在其中都可以互为价值、和谐共存、共同驱动其发展和生长的一个大环境，好比，一个几百人的微信群，要是大家在其中都很活跃，彼此也都能给其他人多少提供和创造一些价值，让这个群可以自然良性发展下去，它就已经是一个小生态了。

16. 互联网行业有且仅有两种内容生产模式：PGC（专业内容生产模式）和 UGC（用户生产内容模式）。

C++

1. 在 C++中，头文件在预编译期间就被替换到 CPP 文件中，和 inline 函数一样，都是简单的文本替换。故头文件之间相互包含时，只需要在相应的 CPP 文件中包含 include，而不需要在头文件中 include。（头文件、预编译、文本替换）

而 python 等动态语言没有预编译的步骤，故每个枚举和 int 等数据类型一样，是一种普通的数据类型，枚举变量是普通的变量。（枚举、普通的数据类型）

2. 指针变量和普通变量差不多，既可指向普通数据，也可指向类对象。（指针变量、指向类对象）
3. 指向类成员函数的指针与普通函数指针的区别在于，指向类成员函数的指针不仅要匹配函数的参数类型和个数以及返回值类型，还要匹配该函数指针所属的类类型。总结一下，比较以下几点：（指向类成员函数、匹配所属类的类型）

(1) 参数类型和个数

(2) 返回值类型

(3) 所属的类类型（特别之处）

4. 指向类成员的指针声明只需要在指针前加上类类型即可，格式为：（指向类成员的指针、加上类类型）

```
返回值 (类名::*指针类型名)(参数列表);
```

例如：

```
void (A::*p1)(void);
```

5. 指向类成员的指针赋值：只需要用类的成员函数地址赋值即可，格式为：

```
指针类型名  指针名 = &类名::成员函数名;
```

例如：

```
struct X {
  void f(int){ };
  int a;
};
void (X::* pmf)(int);
pmf = &X::f;
```

这里的这个&符号是比较重要的：不加&，编译器会认为是在这里调用成员函数，所以需要给出参数列表，否则会报错；加了&，才认为是要获取函数指针。这是 C++专门做了区别对待。（&、表示这是个成员指针）

6. 类的静态成员的指针和普通的函数指针没啥区别。（静态成员、普通指针、没区别）

7. 冒号（:）的用法：（冒号、结构体位域、构造函数初始化列表）

(1) 表示结构体内位域的定义：

```
typedef struct _XXX{
        unsigned char a:4;
        unsigned char c;
}XXX;
```

(2) 构造函数后面的冒号起分割作用，是类给成员变量赋值的方法，初始化列表，更适用于成员变量的常量 const 型。

```
struct _XXX{
        _XXX() : y(0xc0) {}
};
```

8．双冒号（::）用法　（双冒号、域操作符、全局函数、作用域名成员）

(1) 表示“域操作符”

```
Using ns::print;
```

(2) 直接用在全局函数前，表示是全局函数

(3) 表示引用成员函数及变量，作用域成员运算符。

9．好代码:

(1) 可读性高

(2) 高内聚、低耦合的

(3) 性能高

(4) BUG 少

10．

Django

1．新式类：由任意内置类型派生出的类。新式类的基类搜索机制是广度优先遍历。（新式类、广度优先）

不由任意内置类型派生出的类，则称之为“经典类”。经典类的基类搜索机制是深度优先遍历。（经典类、深度优先）

Docker

1. 停止所有的 container：（所有、$()）

```
docker stop $(docker ps -a -q)
```

2. 删除所有 container：

```
docker rm $(docker ps -a -q)
```

3. docker logs 用于显示指定容器的日志：（docker logs、显示日志）

```
Docker logs container-name
```

4. docker run：创建一个新的容器并运行一个命令。常见参数如下：

(1) -a stdin：指定标准输入输出内容类型，可选 STDIN/STDOUT/STDERR 三项；

(2) -d：后台运行容器，并返回容器 ID；

(3) -i：以交互模式运行容器，通常与 -t 同时使用；

(4) -t：为容器重新分配一个伪输入终端，通常与 -i 同时使用；

(5) --name="nginx-lb"：为容器指定一个名称；

(6) --dns 8.8.8.8：指定容器使用的 DNS 服务器，默认和宿主一致；

(7) --dns-search example.com：指定容器 DNS 搜索域名，默认和宿主一致；

(8) -h "mars"：指定容器的 hostname；

(9) -e username="ritchie"：设置环境变量；

(10) --env-file=[]：从指定文件读入环境变量；

(11) --cpuset="0-2" or --cpuset="0,1,2"：绑定容器到指定 CPU 运行；

(12) -m ： 设置容器使用内存最大值；

(13) --net="bridge"：指定容器的网络连接类型，支持 bridge/host/none/container： 四种类型；

(14) --link=[]：添加链接到另一个容器；

(15) --expose=[]：开放一个端口或一组端口；

5. 传统分区使用固定大小分区，重新调整大小十分麻烦。但是，LVM 可以创建和管理“逻辑”卷，而不是直接使用物理硬盘。可以让管理员弹性的管理逻辑卷的扩大缩小，操作简单，而不损坏已存储的数据。可以随意将新的硬盘添加到 LVM，以直接扩展已经存在的逻辑卷。（LVM、弹性扩大磁盘大小）

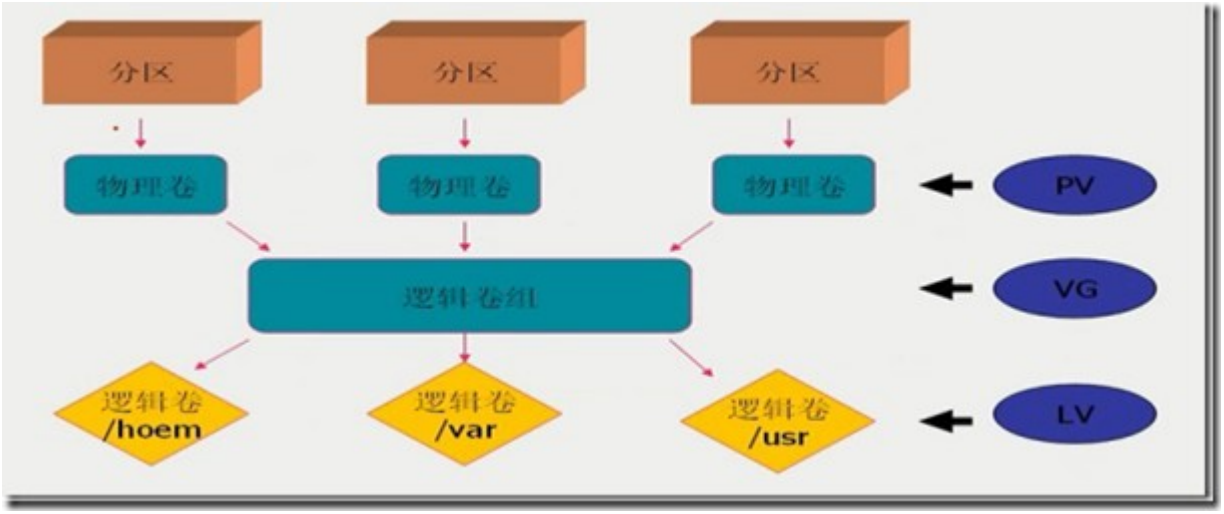
(1) PV：physical volume，物理卷。物理卷在逻辑卷管理系统最底层，可为整个物理硬盘或实际物理硬盘上的分区。（PV、物理卷）

(2) VG：volume group，卷组。卷组建立在物理卷上，一卷组中至少要包括一物理卷，卷组建立后可动态的添加卷到卷组中，一个逻辑卷管理系统工程中可有多个卷组。（VG、卷组）

(3) LV：logical volume，即逻辑卷。逻辑卷建立在卷组基础上，卷组中未分配空间可用于建立新的逻辑卷，逻辑卷建立后可以动态扩展和缩小空间。逻辑卷，事实上就是 linux 里面的各种分区，如 home 分区、根分区等就是一个逻辑卷 （LV、逻辑卷、home 分区、就是逻辑卷）

(4) PE：physical extent，物理区域。物理区域是物理卷中可用于分配的最小存储单元，物理区域大小在建立卷组时指定，一旦确定不能更改，同一卷组所有物理卷的物理区域大小需一致，新的 pv 加入到 vg 后，pe 的大小自动更改为 vg 中定义的 pe 大小。（PE、物理区域、物理卷、最小存储单元）

(5) LE：logical extent 逻辑区域。逻辑区域是逻辑卷中可用于分配的最小存储单元，逻辑区域的大小取决于逻辑卷所在卷组中的物理区域的大小。（LE、逻辑区域、逻辑卷、最小存储单元）



6．docker 的网络模式：

(1) none 网络：这种网络模式下容器只有 lo 回环网络，没有其他网卡。none 网络可以在容器创建时通过—network=none 来指定。这种类型的网络没有办法联网，封闭的网络能很好的保证容器的安全性。

(2) host 网络：通过命令--network=host 指定，使用 host 模式的容器可以直接使用 docker host 的 IP 地址与外界通信，容器内部的服务端口也可以使用宿主机的端口，不需要进行 NAT，host 最大的优势就是网络性能比较好，但是 docker host 上已经使用的端口就不能再用了，网络的隔离性不好。（host 模式、--network=host、共用 IP）

(3) bridge 网络：容器的默认网络模式，docker 在安装时会创建一个名为 docker0 的 Linux bridge，在不指定--network 的情况下，创建的容器都会默认挂到 docker0 上面。（bridge 模式、默认模式）

(4) container 模式：创建容器时使用--network=container:NAME_or_ID 这个模式在创建新的容器的时候指定容器的网络和一个已经存在的容器共享一个 Network Namespace，但是并不为 docker 容器进行任何网络配置，这个 docker 容器没有网卡、IP、路由等信息，需要手动的去为 docker 容器添加网卡、配置 IP 等。

(5) user-defined 模式：用户自定义模式主要可选的有三种网络驱动：bridge、overlay、macvlan。bridge 驱动用于创建类似于前面提到的 bridge 网络；overlay 和 macvlan 驱动用于创建跨主机的网络。

7．docker 会在机器上维护一个网络，并通过虚拟交换机 docker0 和主机本身的网络连接在一起。（虚拟交换机、docker0）

8. bridge 模式相关的 ip 有 3 个，分别是容器 ip、docker0 的 ip 和宿主机的 ip。对外部机器而言，容器 ip 和 docker0 的 ip 都是内部 ip，用于请求转发，对外是不可见的，由宿主机 ip 统一对外服务。
9. 容器之间通信由 docker0 来完成。
10. 所有容器要提供服务，必须要把服务的端口映射到主机的某个端口上，所有的报文都是通过主机进行转发的。（提供服务、通过主机转发）
11. 使用以下命令来显示 docker 的版本：

```
docker --version
docker version
```

使用以下命令显示 docker 的基本信息：

```
docker info
```

12. 显示主机上存在的 image：

```
docker image ls
```

13. Docker 既支持单机模式，也支持群集模式。Swarm 集群是一组运行 Docker 的机器。
14. 使用 docker swarm init 启用 swarm 模式，并让当前机器成为 swarm manager，然后在其他机器上运行 docker swarm join 来作为工作者加入 swarm。
15. 容器的是否随 docker 启动而启动的相关配置文件在下面这个文件里：

```
/var/lib/docker/containers/容器 ID/hostconfig.json/
```

16. 容器内软件的相关设置可以通过 docker exec 命令进入容器内部后再设置。例如，使用以下命令进入 mynginx 容器并执行相关 shell 命令：（Docker exec -it、进入容器内部）

```
Docker exec -it mynginx /bin/bash
```

之后就可以设置 nginx 的 nginx.conf 文件了。

17. RUN、CMD 和 ENTRYPOINT 这三个 Dockerfile 指令看上去很类似，很容易混淆：

- RUN 执行命令并创建新的镜像层，RUN 经常用于安装软件包。
- CMD 设置容器启动后默认执行的命令及其参数，但 CMD 能够被 docker run 后面跟的命令行参数替换。（CMD、默认命令和参数）

```
FROM ubuntu:trusty
CMD ping localhost
```

下面的 hostname 覆盖了默认的 ping 命令。

```
$ docker run demo hostname
```

- ENTRYPOINT 配置容器启动时运行的命令。使用 docker run 的--entrypoint 参数覆盖默认的 ENTRYPOINT。（ENTRYPOINT 、启动时运行的命令）

```
$ docker run --entrypoint hostname demo
```

hostname 覆盖 ping 命令。

18. CMD 有三种格式的配置， 分别是:

- CMD ["executable","param1","param2"]：exec 格式，推荐的格式。
- CMD ["param1","param2"]： 作为 ENTRYPOINT 的默认参数，必须提供 ENTRYPOINT 配置，CMD 作为默认参数。
- CMD command param1 param2： shell 格式。

19. ENTRYPOINT ，有两种格式的配置，分别是:

- ENTRYPOINT [“executable”，“param1”，“param2”]： exec 格式，推荐。
- ENTRYPOINT command param1 param2： shell 格式。

ENTRYPOINT 使用 shell 格式时，会忽略 CMD 和 run 传入的参数，如果要替换默认的 ENTRYPOINT 命令，则需要在执行 docker run 的时候指定--entrypoint 参数。

20. 组合使用 ENTRYPOINT 和 CMD，ENTRYPOINT 指定默认的运行命令，CMD 指定默认的运行参数:（组合使用 ENTRYPOINT 和 CMD）

```
FROM ubuntu:trusty
ENTRYPOINT ["/bin/ping","-c","3"]
CMD ["localhost"]
```

21. CMD 命令定义的是容器的默认的可执行体，而 ENTRYPOINT 命令才是真正的入口。
22. 镜像是由一组只读层组成，容器在只读的镜像上面加了一层可写层。（镜像、只读层、容器、加了可写层）
23. Docker 有两种容器将文件存储在宿主机中的选项，这样即使在容器停止之后这些文件也会被保留：卷和绑定挂载。（卷存储、绑定挂载）

- 卷存储：卷是由 Docker 管理的主机文件系统的一部分中，在 Linux 上的目录为/var/lib/docker/volumes/。非 Docker 进程不应该修改这部分文件系统。卷是在 Docker 中保留数据的最佳方式。（卷、docker 管理）
- 绑定挂载：将主机系统的某个文件或文件夹挂载到容器中。（绑定挂载、主机文件夹）

```
docker run -d \
  -it \
  --name devtest \
  --mount type=bind,source="$(pwd)"/target,target=/app \
  nginx:latest
```

- tmpfs 挂载仅存储在主机系统的内存中，而不会写入主机系统的文件系统。这个是 linux 特有的。

24. 卷由 Docker 创建和管理。有两种方式创建卷，

- 使用 docker volume create 命令创建。（docker volume create、创建卷）
- 创建容器或服务时使用-v 或--mount 参数创建卷。

25. 使用-v 参数时，创建卷指定卷名：（-v、创建卷、指定卷名）

```
$ docker run -d \
  --name=nginxtest \
  -v nginx-vol:/usr/share/nginx/html:ro \
  nginx:latest
```

而绑定挂载使用绝对路径或相对路径：（绑定挂载、绝对路径或相对路径）

```
$ docker run -d \
  -it \
```

```
 --name devtest \
 -v "$(pwd)"/target:/app:ro \
 nginx:latest
```

26. Docker volume 创建和管理卷：

- 创建卷：

```
docker volume create my-vol
```

- 列出卷：

```
docker volume ls
```

- 检查卷：

```
$ docker volume inspect my-vol
[
   {
      "Driver": "local",
      "Labels": {},
      "Mountpoint": "/var/lib/docker/volumes/my-vol/_data",
      "Name": "my-vol",
      "Options": {},
      "Scope": "local"
   }
]
```

- 删除卷：

```
docker volume rm my-vol
```

27. 如果 Docker 容器是使用/bin/bash 命令启动的，则可以使用 attach 来访问它，如果不是，则需要在容器内创建一个 bash 实例。

attach 不会在容器中创建进程执行额外的命令，只是附着到容器上。exec 会在运行的容器上创建进程执行新的命令。（attach、附着到容器上）

使用 attach 连接到一个已经运行的容器的 stdin 时需要注意的是，如果从这个 stdin 中 exit，会导致容器的停止。 （attach、执行 exit、会导致容器停止）

28. shell 程序的执行有交互和非交互之分：（shell、交互 shell、非交互 shell）

- tty 中的 shell 是交互 shell。
- 而后台执行 shell 是非交互 shell。

29. 使用 exec 进入容器时，如果最后执行的命令是/bin/bash，则是进入容器并开启一个人机交互 shell，如下：（exec、命令是/bin/bash、进入并开启交互 shell）

```
Docker exec -it mycontainer /bin/bash
```

如果执行的是诸如 tar 之类的其他命令，则执行该命令，但并不开启交互 shell：（其他命令、执行、不开启交互 shell）

```
sudo docker run -i -t --volumes-from data -v $(pwd):/backup c9fc7f8eec37 tar cvf /backup/backup.tar /data
```

30. -v 标志和--mount 标志差不多，都可用于 docker run 命令指定一个卷，最大的不同在于-v 语法将所有选项组合在一个字段中，而--mount 语法将它们分开。（-v、--mount、差不多）

31. --mount 的选项由多个键值对组成，用逗号分隔，每个键对由<key>=<value>元组组成。 --mount 语法比-v 或--volume 更冗长，但键的顺序并不重要，并且标志的值更易于理解。（--mount、<key>=<value>、逗号分隔）

- type： 卷的类型，可以是 bind 、volume 或 tmpfs 。
- source： 对于命名卷，这是卷的名称。对于匿名卷，该字段被省略。可以指定为 source 或 src 。
- destination：将文件或目录安装在容器中的路径作为其值。可能被指定为 destination 、dst 或 target 。
- readonly：以只读方式挂载。
- volume-opt：选项可以多次指定，它采用由选项名称和值组成的键值对。

```
docker run -d \
  --name=nginxtest \
  --mount source=nginx-vol,destination=/usr/share/nginx/html,readonly \
  nginx:latest
```

32. 使用 docker run 的-v 参数指定卷时，如果卷不存在，则直接创建卷。

```
$ docker run -d --name devtest -v myvol2:/app nginx:latest
```

33. -volumes-from 选项用于指定主容器到底使用哪个卷容器。（-volumes-from、指定卷容器）
34. 卷容器是只包含卷的容器，通常以 busybox 或 alpine 为基础镜像，然后在启动主容器时使用-volumes-from 选项，把数据卷容器的所有卷映射到主容器内。（卷容器、以 busybox 或 alpine 为基础镜像）

```
Docker run -d –name db3 –volumes-from db1 trainning/postgres
```

指定使用卷容器 db1。

35. 要指定容器的重启策略，可以在 Docker run 命令时使用--restart 标志来指定。（重启策略、--restart）
36. Docker 容器的持久存储模式：

- 卷
- 卷容器
- 绑定挂载

37. 在绑定挂载中，如果容器中的目录存在文件，则默认会将这些文件隐藏。（绑定挂载、容器存在文件、会隐藏文件）
38. 卷是由完全由 docker 管理，所以功能上比绑定挂载更强大，绑定挂载无法使用 docker volume 管理。
39. 如下方式使用绑定挂载：

- 从主机共享配置文件到容器。默认情况，docker 会绑定类似 /etc/resolv.conf 的文件用于 DNS 的解析。

- 主机与容器共享源代码或构建工具。如，你可以将 Maven target/ 挂载到容器中，并且每次主机上构建 Maven 项目时，容器都可以访问重建的构件。
- 主机的文件或目录结构与容器所需的一致时。

40. 绑定挂载的行为跟 U 盘差不多，将文件夹挂载到容器之后，就可以在容器里修改文件夹中的数据。（绑定挂载、U 盘差不多）
41. 启动一个创建新卷的容器时，如果容器中对应的目录有文件或目录，则将这些文件或目录复制到新卷以填充该卷，之后卷和绑定挂载在数据覆盖和读写操作上几乎是一样的。（新卷、容器中有文件、复制到新卷）
42. 数据卷容器最基本的用处还是利用数据卷容器来备份、恢复、迁移数据卷。
43. 卷的数据覆盖问题：

- 如果挂载一个空的数据卷到容器中的一个非空目录中，那么这个目录下的文件会被复制到数据卷中。（空卷、非空目录、复制到卷）
- 如果挂载一个非空的数据卷到容器中的一个目录中，那么容器中的目录中会显示数据卷中的数据。如果原来容器中的目录中有数据，那么这些原始数据会被隐藏掉。（非空卷、隐藏容器中的文件）

44. 绑定挂载的数据覆盖问题：如果挂载一个空的目录到容器中的一个非空目录中，那么容器目录下的文件被隐藏。（绑定挂载、隐藏容器目录的文件）
45. docker 提供了一些默认的网络驱动用于创建这些网络。你可以创建桥接网络 bridege、覆盖网络 overlay 或者 MACVLAN 网络。
46. Docker Machine 是一个工具，安装在用户使用的主机，用于管理云主机上的 Docker Engine。（Docker Machine、是个工具、管理 docker engine）

Docker Engine 也就是 linux 中运行的带镜像和容器的 docker，它由几部分组成：（docker engine、带镜像和容器）

- Docker Daemon — docker 的守护进程，属于 C/S 中的 server。
- Docker REST API — docker daemon 向外暴露的 REST 接口。
- Docker CLI — docker 向外暴露的命令行接口。也就是 docker run 等命令。

47. network 命令：

- docker network create：创建网络。（docker network create、创建网络）
- docker network connect：令容器连接到一个指定的网络。连接以后该容器就能访问该网络下的所有容器。（docker network connect、连接指定网络）
- docker network ls：显示当前已经创建的网络。（docker network ls、显示网络）
- docker network rm：删除网络。（docker network rm、删除网络）
- docker network disconnect：指定容器退出某个网络。（docker network disconnect、退出网络）
- docker network inspect：显示网络信息。（docker network inspect、显示网络信息）

48. 自定义容器网络可以实现用户根据需要指定自己的网络，以此来适应某些配置，如 ip 地址规划等等。（自定义网络、根据需要配置、ip 地址规划）

```
docker network create -d bridge --subnet 172.10.0.0/24 --gateway 172.10.0.1 my_net
```

- -d：指定网络类型，默认为 bridge。
- --subnet：指定子网掩码。
- --gateway：指定网关。

49. 每创建一个自定义网络便会在宿主机中创建一个网桥，原理和 docker0 是一致的，而且也是对等的。名字为 br-<网络短 ID>，可以通过 brctl show 命令查看全部网桥信息。（自定义网络、创建网桥）

50. 同一个网络下的容器之间是能 ping 通的，但是不同网络之间的容器由于网络独立性的要求是无法 ping 通的。原因是 iptables-save DROP 掉了 docker 之间的网络。（不同网络、无法 ping 通）

51. 默认情况下，当 docker daemon 停止时，会关闭运行中的容器。有两种方式可以保持容器在 docker daemon 变为不可用的时候保持运行：

- 在 linux 系统上，默认配置文档为 /etc/docker/daemon.json。（/etc/docker/daemon.json、"live-restore": true）

```
{
"live-restore": true
}
```

52. 如果直接启动 docker daemon，只需要传递--live-restore 标识即可：（--live-restore 标识）

```
sudo dockerd --live-restore
```

53.

Ansible

1. 调试和测试类的 module

- ping：ping 一下你的远程主机，如果可以通过 ansible 成功连接，那么返回 pong。（ping 模块、成功、返回 pong）

```
$ ansible -m ping all
192.168.0.116 | SUCCESS => {
   "changed": false,
   "ping": "pong"
}
```

- debug：主要用于打印输出，特别适合于在 playbook 中打印一个变量结果，看看有没有达到自己预期的效果（debug、打印变量结果）

```
$ ansible -m debug all
192.168.0.116 | SUCCESS => {
   "msg": "Hello world!"
}
```

2. ad-hoc 里面的-a 参数后面接的是模块参数，这个模板参数和 playbook 的 task 中的 module:options 的 options 是一样的。以 copy 模块为例，ad-hoc 格式如下：（-a 后的参数、和 options 是一样的）

```
ansible test -m copy -a "src=test.sh backup=yes dest=/root"
```

这条命令是下面这条 task 完成一样的功能：

```
Name:copy a file
copy: src=test.sh backup=yes dest=/root
```

注意：除 command 和 shell 模块之外的所有模块的“module:options”里面的 options 格式类似“key=value”，这里的 key 是模块内置的，比如上面的 src、backup 和 dest 都是 copy 内置的参数。不同的模块有不同的内置参数。（key、内置参数、不同模块、不同的内置参数）

3. ansible 常用模块：

(1) command 模块是 Ansible 默认执行模块，在远程主机执行 shell 指令。（command 模块、默认模块、执行 shell 指令）

(2) copy 模块主要用于复制文件、目录等，并对其进行权限修改。（copy 模块、复制文件）

(3) flie 模块主要用于文件、文件夹权限的变更，以及文件、文件夹、超级链接类的创立、拷贝、移动、删除操作。（file 模块、文件操作）

(4) stat 主要用于查看文件目录属性，包括创建时间、访问时间、修改时间、存在状态、路径位置、文件大小等等。（stat 模块、查看文件目录属性）

(5) user 模块主要用于用户管理，包括用户创建、删除，shell 类型、密钥生成以及用户属性设置等。（user 模块、用户管理）

(6) group 模块主要用于用户组管理，包括创建、删除等。（group 模块、用户组管理）

(7) yum 模块主要用于 rpm 包的安装、更新、删除等等。（yum 模块、rpm 包的安装）

(8) service 模块主要用于系统服务管理，包括启动、关闭、重启、以及设置开机启动等。（service 模块、系统服务管理）

(9) cron 模块主要用于计划任务管理，包括新建，删除等。（cron、计划任务管理）

(10) get_url 模块主要用于下载，类似于 shell 命令 curl 或者 wget。（get_url 模块、用于下载、类似 wget）

(11) unarchive 模块主要用于解压包，支持 zip、tar、jar 等。（unarchive 模块、解压包）

(12) synchronize 模块是同步模块，类似于 rsync。（synchronize 模块、同步）

(13) filesystem 模块主要用于文件系统创建，磁盘格式化。（filesystem 模块、文件统创建）

(14) mount 模块主要用于分区挂载。（mount 模块、分区挂载）

(15) lineinfile 模块主要用于搜索匹配替换、插入：若匹配存在的行则替换行，若不存在则插入。（lineinfile 模块、搜索匹配替换、插入）

(16) replace 模块主要用于搜索匹配替换，类似于 linux 命令 sed。（replace 模块、搜索匹配替换）

(17) template 使用了 Jinjia2 格式作为文件模版，进行文档内变量的替换的模块。它的每次使用都会被 ansible 标记为”changed”状态，但是 template 只能应用到 playbook。

(18) sysctl 模块主要用于修改 sysctl.conf 配置文件。（sysctl 模块、修改 sysctl.conf 配置文件）

(19) debug 模块主要用于打印输出，特别适合于在 playbook 中打印一个变量结果，看看有没有达到自己预期的效果。（debug 模块、打印输出）

(20) wait_for 模块主要用于等待到达到某个条件时才继续执行下一个 task。（wait_for 模块、等待某个条件）

4. shell 和 command 的区别：

- command 模块命令不使用 shell 执行。因此,，像$HOME 这样的变量是不可用的。还有像<、 >、|、&都将不可用。
- shell 模块通过 shell 程序执行， 默认是/bin/sh，<、>、|、& 可用。但这样有潜在的 shell 注入风险。
- command 模块更安全，因为他不受用户环境的影响。也很大的避免了潜在的 shell 注入风险。

5. 在 task 或 template 中引用变量，使用双花括号：（引用变量、双花括号）

```
tasks:
 - name: ensure apache is at the latest version
   yum: name={{ package }} state=latest
```

Wine

1. 通常，安装或启动 exe 时应使用命令行安装，以查看出错信息。（安装或启动、命令行、查看出错信息）
2. 使用 wine 命令安装应用；（wine、安装应用）

```
wine xxx.exe
```

3. wine 使用 windows 路径时需要加上引号：（wine、windows 路径、加引号）

```
wine "C:\Program Files\ appname \ appname.exe "
```

4. 使用“wine control”命令打开控制面板。（wine control、控制面板）
5. 使用“winecfg”命令进行设置，如切换 windows 版本，即选择 win7 还是 win8。（winecfg、设置）
6. 在 wine 下使用“wine regedit”启动注册表。（wine regedit、注册表）
7. wine 使用 IE 的 HTML 渲染引擎会破坏 wine 环境，因此 wine 有自己的 HTML 渲染引擎，名为 gecko，是个 cab 或 msi 文件，需要下载下来并放到指定目录即可。（HTML 渲染引擎、gecko）
8. 使用“wine msiexec /i”来安装 msi 文件：（wine msiexec /i、安装 msi 文件）

```
wine msiexec /i whatever.msi
```

或者使用 wine start：（wine start、安装 msi 文件）

```
wine start whatever.msi
```

9. 使用 wine uninstaller 卸载程序。（wine uninstaller、卸载程序）
10. 使用 winefile 以图形化的方式来管理文件。（winefile、管理文件）
11. Winetricks 是一个辅助脚本，用于在 wine 中下载并安装各种闭源的组件和运行库。（Winetricks、安装组件和运行库）

```
winetricks quartz
```

12. 字体出错时会出现“CoGetClassObject no not registered”等 CoGetClassObject 错误，需要安装“winetricks mdac28”。

```
winetricks mdac28
```

13. desktop 文件是 ubuntu 的桌面快捷方式，通常格式如下：

```
[Desktop Entry]
Name=通达信金融终端
Exec=env WINEPREFIX="/home/lan/.wine" wine C:\\\\new_tdx\\\\Tdxw.exe
Type=Application
StartupNotify=true
Path=/home/lan/.wine/dosdevices/c:/new_tdx/
Icon=F9F2_tdxw.0
```

- Name：快捷方式的名字。
- Exec：表示在执行的命令，这里执行的是 wine 命令
- Type：快捷方式类型，有 Application 和 link 两种。
- Icon：图标。

14.