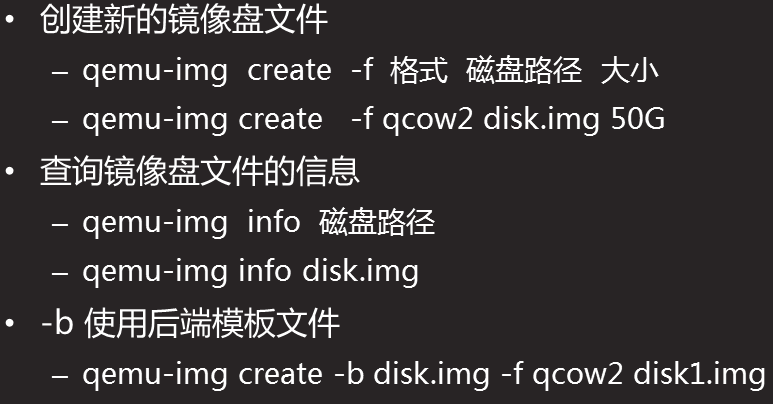
**CLOUD**

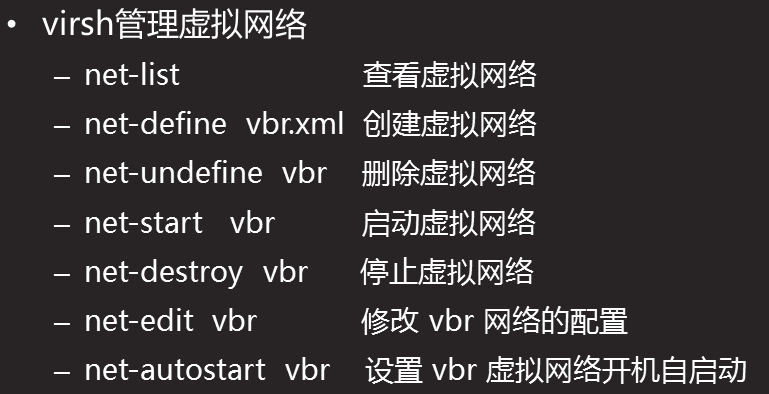
1. KVM







虚拟网络管理



虚拟机 : 镜像文件+配置文件

镜像文件 /var/lib/libvirt/images/虚拟机.qcow2

配置文件 /etc/libvirt/qemu/虚拟机.xml

远程控制远端主机的virt虚拟机系统

virsh -c qemu+ssh://176.121.204.53:99/system

1. Openstack ---------IaaS

组成:Horizon(控制台服务) Keystone(身份认证)

Neutron(网络管理) Cinder(存储卷)

Nova(虚拟机节点管理,配合Keystone认证)

Glance(镜像存储管理)

swift 提供块存储

systemctl list-unit-files 查看系统服务及状态

systemctl -t service (--all) 查看启动的服务(全部服务)

控制节点,网络节点,存储节点,计算节点

+++++++++++++++++++++++++++++++++++DOCKER

1.新开 run -it

-d :分离模式: 在后台运行

-i :即使没有附加也保持STDIN 打开

-t :分配一个伪终端

2.docker ps [-a] 查看开启的容器列表

3.docker 问题列表，及解答

问题：docker run -it busybox /bin/bash 报错问题

原因：

最后的命令是容器内部命令，容器内，存在才可以执行

可以不指定，不指定启动默认命令

问题：docker run -it centos 退出后，配置全部丢失

原因：

run 启动的是新的容器，老的容器退出后就停止了

启动管理一个老的容器可以使用 docker start|stop|restart

问题：老容器启动以后，怎么在进入？

docker exec -it 容器id /bin/bash

docker attach 容器id

exec 与 attach 的区别

exec 单独启动命令运行，与容器启动的终端无关

attach 不启动新的命令，直接连接 console 终端

exec 退出不会影响容器的运行

attach 退出后，容器结束

问题：如果我使用attach 连接容器后，怎么才能不结束容器？

解决方法：把容器放后台，使用快捷键 ctrl + pq

问题：attach 为什么退出后，容器会结束？

因为 attach 连接进容器的 pid 1 的进程，当 attach 结束时候，pid 为 1 的进程被结束

所有整个容器被销毁

问题：docker run -it nginx 没响应？

因为 nginx 启动的默认 cmd 时 nginx daemon，该进程不是一个交互式的进程

4.docker run 使用

-i 交互式的

-t 分配终端

-d 把容器放在后台运行

-c 在容器中执行命令(非交互式)

docker run -it centos cmd 启动一个交互式的容器，在前台运行

docker run -d centos cmd 启动一个非交互式的容器，在后台运行

docker run -itd centos cmd 启动一个交互式的容器，在后台运行

测试

docker run -it centos /bin/bash 成功

docker run -d centos /bin/bash 失败

docker run -itd centos /bin/bash 成功

docker run -it nginx nginx 失败

docker run -d nginx nginx 失败

docker run -itd nginx nginx 失败

docker run -it nginx nginx -g "daemon off;" 成功

docker run -d nginx nginx -g "daemon off;" 成功

docker run -itd nginx nginx -g "daemon off;" 成功

交互式的进程启动要使用 it ， 非交互式的使用 d ，交互式的放后台使用 itd

启动 bash 的正确姿势 docker run -itd centos

启动 nginx 的正确姿势 docker run -d -p 80:80 nginx

1. -p 80:80：将容器的80端口映射到主机的80端口

-p 80:5000

这就将容器中的5000端口映射到本机的80端口；

**pstree -p 0:可知attach 直接连接pid为1的进程,退出后整个容器被销毁**

**而exec时fork一个子bash进程运行的,无影响.**

DOCKER 原理之六大命名空间



容器的优势在于开销小,启动实例快捷方便,但隔离性不如虚拟机,虚拟机有自己的操作系统,驱动程序,接口等等,隔离线极强.

一个容器中的出现安全问题可能会泄漏并影响其他容器或其应用程序。

docker适用于弹性的,需要多次部署的应用环境.

docker inspect -f {{}} name // 查询容器信息

1. Dockerfile

进入所在目录---->docker build -t 镜像名:标签 .

格式:

FROM:基础镜像

MAINTAINER:信息说明

EXPOSE:开放端口

ENV:环境变量

ADD:复制本地文件到镜像中

RUN:制作镜像时执行命令,可以有多个RUN

WORKDIR:容器默认工作目录

CMD: [“cmd1”,”cmd2”] 容器启动命令

一个空格由一个逗号表示分割

2.自定义仓库:

写配置,重启服务

首先运行registry容器 docker -d -p 5000:5000 registry

需要上传的镜像打标签 docker tag a:latest 192.168.1.20:5000/a:latest

上传镜像 docker push 192.168.1.20:5000/a:latest

docker images 查看

网络查看:

curl [http://192.168.1.20:5000/v2/\_catalog](http://192.168.1.10:5000/v2/_catalog) 查看仓库中有什么镜像

curl http://192.168.1.20:5000/v2/busybox/tags/list

{"name":"busybox","tags":["latest"]}

[root@docker1oo]# curl http://192.168.1.20:5000/v2/myos/tags/list

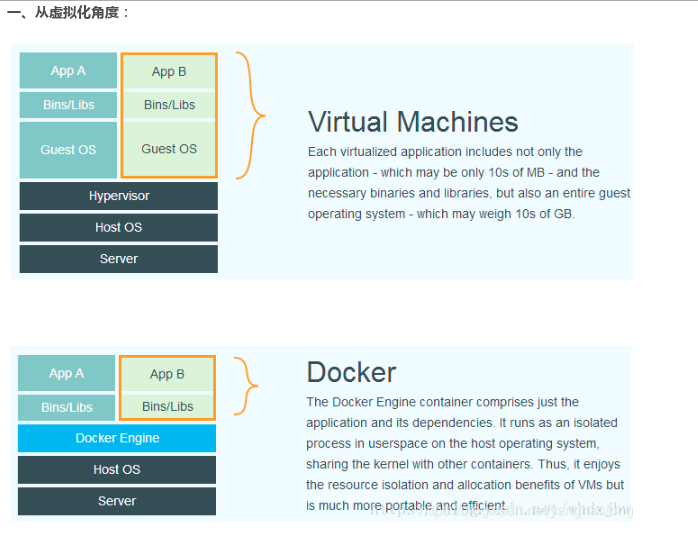
{"name":"myos","tags":["http"]}

查看某一镜像的标签.

openstack 管理 virsh虚拟机 (云主机集群)

对比

k8s 管理调度 docker (容器集群)



上图是Docker容器（可用k8s管理的玩意儿）与传统虚拟化方式的不同之处，传统的虚拟技术，在将物理硬件虚拟成多套硬件后，需要再每套硬件上都部署一个操作系统，接着在这些操作系统上运行相应的应用程序。而Docker容器内的应用程序进程直接运行在宿主机(真实物理机)的内核上，Docker引擎将一些各自独立的应用程序和它们各自的依赖打包，相互独立直接运行于未经虚拟化的宿主机硬件上，同时各个容器也没有自己的内核，显然比传统虚拟机更轻便。 每个集群有多个节点，每个节点可，我们的kuberbete就是管理这些应用程序所在的小运行环境（container）而生。

**ansible**

1. 定义主机,分组和子组

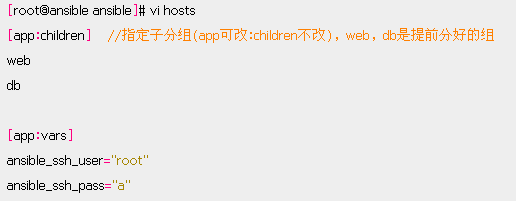


ansible 主机或组或写all -m 模块 -a 命令

设置ssh密码



子分组,也是为了方便调用



自定义的ansible文件只在当前路径生效

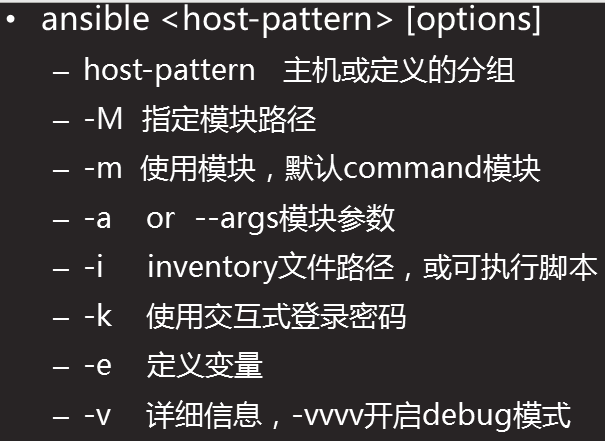
自定义配置文件ansible.cfg

[defaults]

inventory = myhost

host\_key\_checking = False // 相当于ssh时stricthostchecking=no,无须输入yes/no,ansible代为输入.





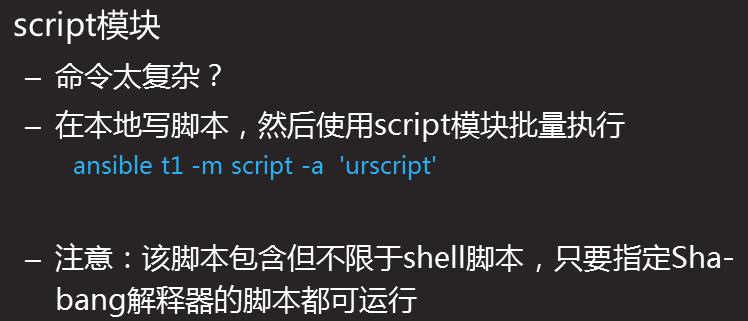
模块的使用:

ansible-doc 模块 :查看指定模块的详细帮助

shell/raw/command

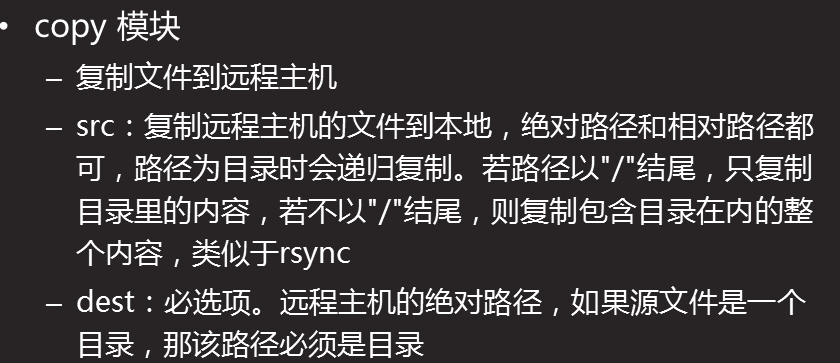


script

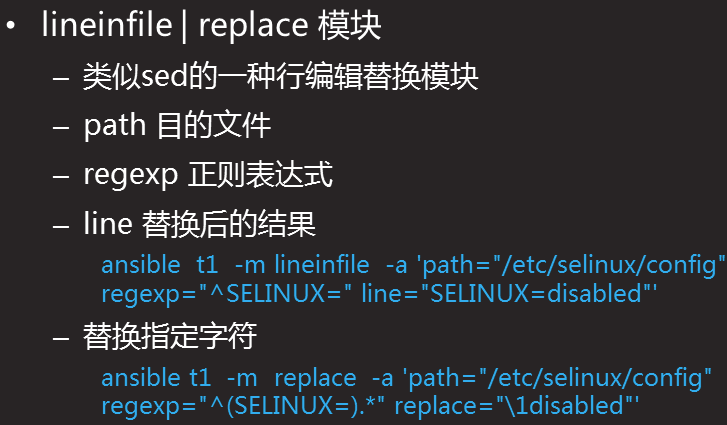


ansible web1 -m script -a './user.sh'

copy



lineinfile | replace

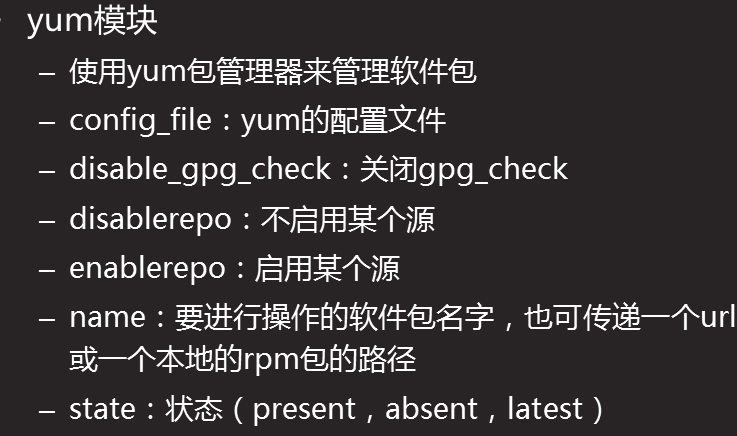


lineinfile相当于 sed c 会整行替换;replace相当于sed s 只替换匹配道德字符.

lineinfile: path=”” regexp=”” line=””

replace: path=”” regexp=”” replace=””

yum



ansible other -m yum -a 'name="lrzsz" state=removed'

//lrzsz软件包名，removed=absent删除

ansible other -m yum -a 'name="lrzsz,lftp" state=installed'

//安装多个软件包，不写state默认为安装(installed)

service模块

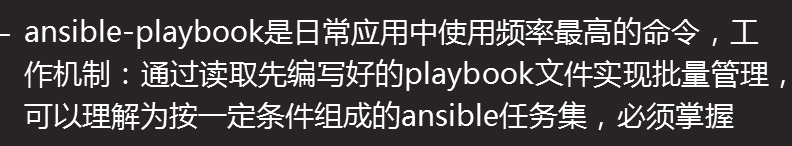
[root@ansible ~]# ansible other -m service -a 'name="sshd" enabled="yes" state="started"' //sshd服务名，开机启动同时启动这个服务

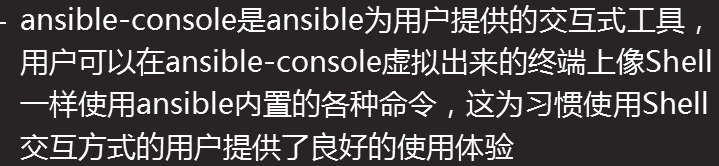
setup模块

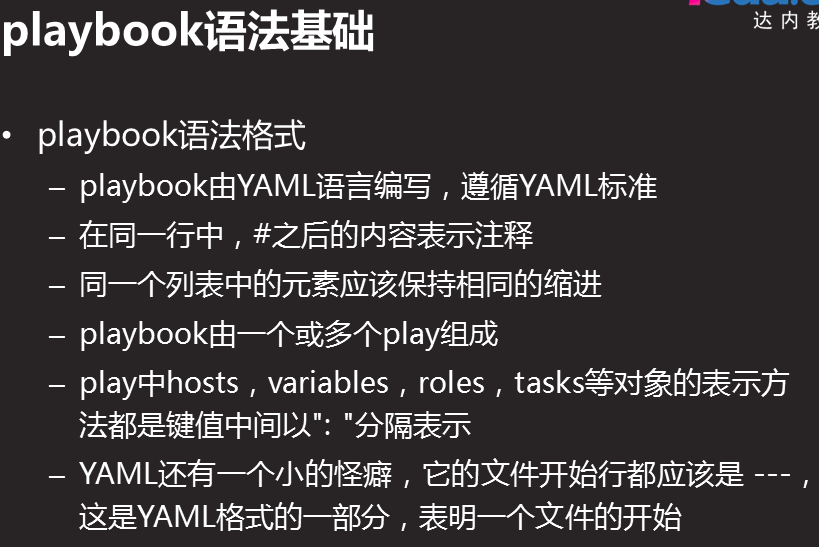
(获取主机信息,一般需要过滤需要的信息)

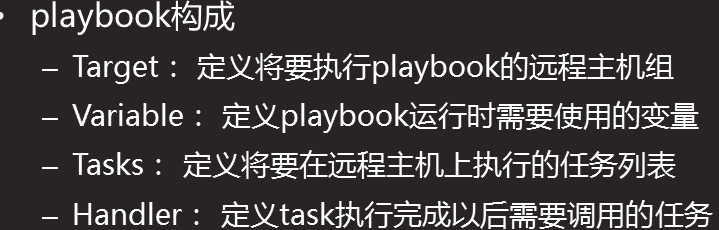
filter 过滤指定的关键字（可以过滤到我们需要的信息）

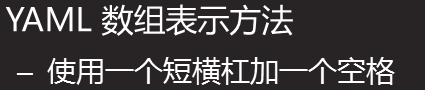
1. playbook!



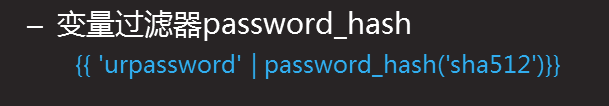








user模块不能直接明文设置密码,解决方法:



password: "{{'123456' | password\_hash('sha512')}}"

1. playbook书写技巧

notify与handlers register: stdout/stderr 与when

①ansible-playbook是将多个play和task集合在一起,其大致构成如下

hosts:主机(主机组,也可以是变量)

tasks:任务

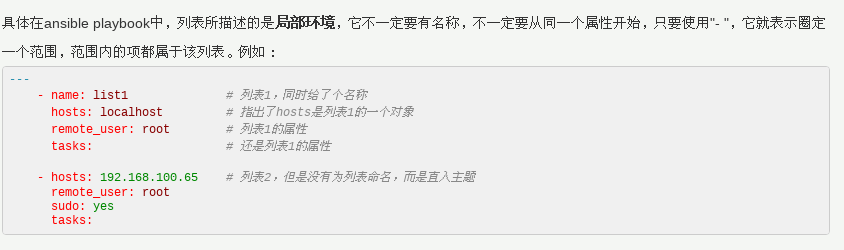
vars:变量

user:指定用哪个用户执行,

templates:模版文件

handles:由特定条件出发tasks

要使用”-”圈定一个范围!!然后一段代码一段代码执行



②分行写

playbook中有3种方式进行续行。

在"key: "的后面使用大于号。

在"key: "的后面使用竖线。这种方式可以像脚本一样写很多行语句。

多层缩进。

例如，下面的3中方法。

---

- hosts: localhost

tasks:

- shell: echo 2 >>/tmp/test.txt

creates=/tmp/haha.txt # 比模块shell缩进更多

- shell: > # 在"key: "后使用大于号

echo 2 >>/tmp/test.txt

creates=/tmp/haha.txt

- shell: | # 指定多行命令

echo 2 >>/tmp/test.txt

echo 3 >>/tmp/test.txt

args:

creates: /tmp/haha.txt

③向模块传递参数

模块的参数一般来说是key=value格式的，有3种传递的方式：

直接写在模块后，此时要求使用"key=value"格式。这是让ansible内部去解析字符串。因为可分行写，所以有多种写法。

写成字典型，即"key: value"。此时要求多层缩进。这是让yaml去解析字典。

使用内置属性args，然后多层缩进定义参数列表。这是让ansible明确指定用yaml来解析。

例如：

---

- hosts: localhost

tasks:

- yum: name=unix2dos state=installed # key=value直接传递

- yum:

name: unxi2dos

state: installed # "key: value"字典格式传递

- yum:

args: # 使用args传递

name: unix2dos

state:installed

④迭代循环

ansibel支持迭代功能。例如，有一大堆要输出的命令、一大堆要安装的软件包、一大堆要copy的文件等等。

例如，要安装一堆软件包。

此处-pkg1 -pkg2 表示一个个数组

---

- hosts: localhost

tasks:

- yum: name="{{item}}" state=installed

with\_items:

- pkg1

- pkg2

- pkg3

它会一个一个迭代到特殊变量"{{item}}"处。

此外,pkg1-3等如果是一个元素组,也可以提高可用性,如下:

---

- hosts: web2

remote\_user: root

tasks:

- user:

name: "{{item.name}}"

group: "{{item.group}}"

password: "{{'123456'|password\_hash('sha512')}}"

with\_items:

-

name: "aa"

group: "users"

-

name: "bb"

group: "mail"

-

name: "cc"

group: "wheel"

-

name: "dd"

group: "root"

再例如，指定一堆文件列表，然后使用grep搜索出给定文件列表中包含"www.example.com"字符串的文件：

---

- hosts: localhost

tasks:

- shell: grep -Rl "www\.example\.com" "{{item}}"

with\_items:

- file1

- file2

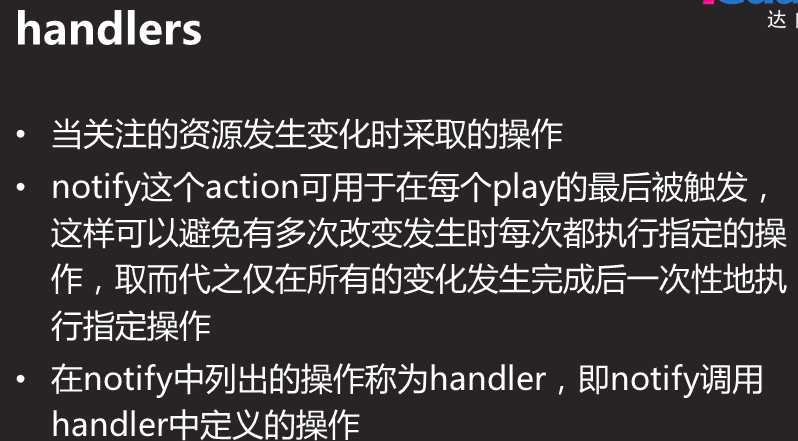
- file3

register: match\_file

- debug: msg="{% for i in match\_file.results %} {{i.stdout}} {% endfor %}"

注意，将with\_items迭代后的结果注册为变量时，其注册结果也是列表式的，且其key为"results"。具体的结果比较长，可以使用debug模块的var或msg参数观察match\_file变量的结果。

⑤



notify关注的内容发生改变,即触发handlers定义的操作,handles与tasks同级,通常结合tags使用,只运行某一个tags修改配置文件,然后重启服务.



与tasks在同一组,其下为所要执行的模块,如例子中的service

⑥include and roles

在编写playbook的时候随着项目越来越大，playbook越来越复杂。可以把一些play、task 或 handler放到其他文件中，通过包含进来是一个不错的选择

roles像是加强版的include，它可以引入一个项目的文件和目录

一般所需的目录层级有

vars：变量层

tasks：任务层

handlers：触发条件

files：文件

template：模板

default：默认，优先级最低

⑦Jinjia2

变量{{}}

条件判断,循环{%..%} :{% if... %}{% elif %}{%endif%}

⑧tags



tags写在所需要调用的某个task的下方

类似于notify

⑨register: stdout/stderr 与when

register变量注册,存储前一个命令返回结果,正确则触发when上的任务.

register: result //返回结果保存到result中

可以通过result.stdout 调用上一条命令的执行输出结果.

⑩



vars:

username: haha

playbook中的vars,如上的haha,只是变量的一个初值处理,可以从命令行-e进行赋值或者调用yml文件赋值.

+++++++++++++++++++++++++++++++++++++++++

ELK

**ELK,Elasticsearch 搜索及实时存储**

**logstash 日志收集,分析**

**kibana 可视化工具,(分析数据)**

1. elasticsearch

配置文件修改主机名,集群名称,监听地址,声明集群成员

集群健康检查 curl 192.168.4.11:9200/\_cluster/health?pretty

ES常用：

PUT --增

DELETE --删

POST --改

GET --查

**结合curl,curl 常用参数介绍：**

**-A 修改请求 agent curl -A ‘hahaha’ 192.168.1.11**

**-X 设置请求方法,不指定时默认为 GET**

**-i 显示返回头信息和文件内容**

**-I 只显示头部信息**

①索引的分片信息:

curl -X GET http://192.168.1.61:9200/\_cat

②显示health的详细信息

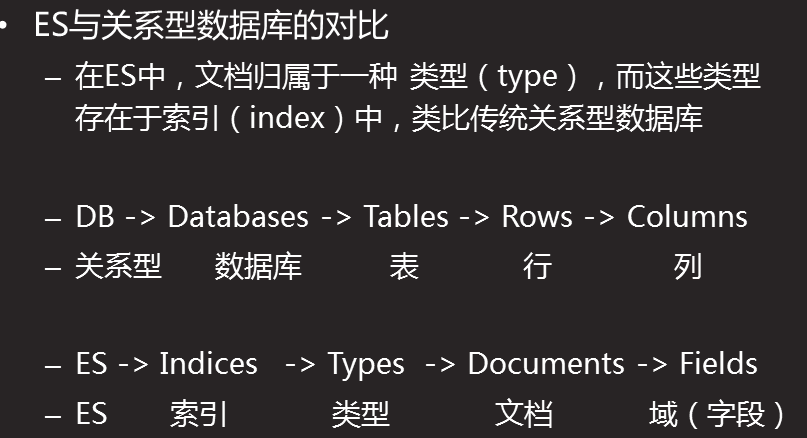
curl -X GET <http://192.168.1.62:9200/_cat/health?v>

③ 查看nodes的帮助

curl -X GET http://192.168.1.61:9200/\_cat/nodes?help

导入数据时,可以看为:库名/表名/id

ES就是一个索引库!



ELK 是一整套的解决方案

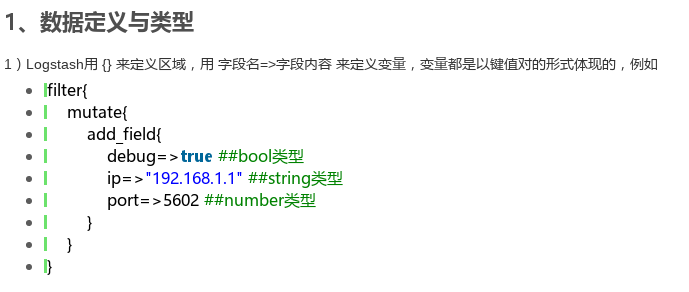
（E）lasticsearch -- 数据库

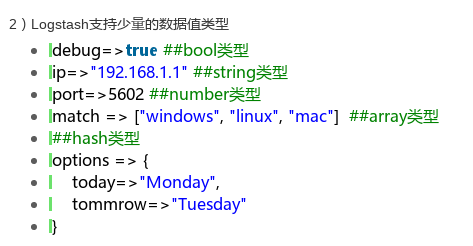
（L）ogstash -- 收集日志、标准化的程序

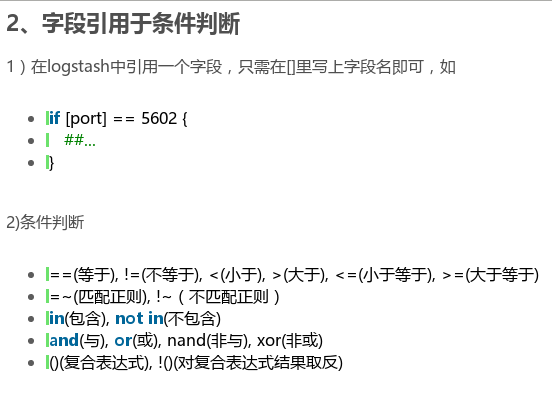
（K）ibana -- 图形的展示工具

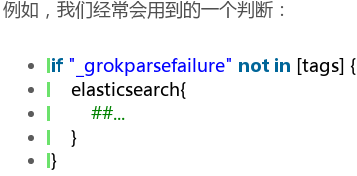
kibana :5601 可视化工具

logstash











**首先,其格式为输入inout加输出output,**

**中间可以设置filter进行过滤,并都用{}包好**

**大体为:**

**input{stdin{...} 模块{...}}**

**filter{}**

**output{stdout{...} 模块{..}}**

**注意 赋值格式 =>**

**重点之模块插件!**

**1.codec类插件:input{**

**stdin{**

**codec => "json" //输入设置为编码json**

**}**

**}**

**filter{**

**}**

**output{**

**stdout{**

**codec => "rubydebug" //输出设置为rubydebug**

**}**

**}**

1. **file类插件,导入本地日志**

**input{**

**file {**

**path => [ "/tmp/a.log", "/var/tmp/b.log" ] //数组类型**

**sincedb\_path => "/var/lib/logstash/sincedb" //记录读取文件的位置,时间戳文件**

**start\_position => "beginning" //配置第一次读取文件从什么地方开始(若没写上述时间戳文件的配置,则安装这个配置开始读取;写了配置,此行相当于无用)**

**type => "testlog" //类型名称**

**}**

**}**

**filter{**

**}**

**output{**

**stdout{**

**codec => "rubydebug"**

**}**

**}**

**又如:**

**input{file{**

**2 path => ["/var/log/message","/var/log/\*.log"]**

**3 type => "syslog"**

**4 sincedb\_path => "/var/lib/logstash/sincedb"**

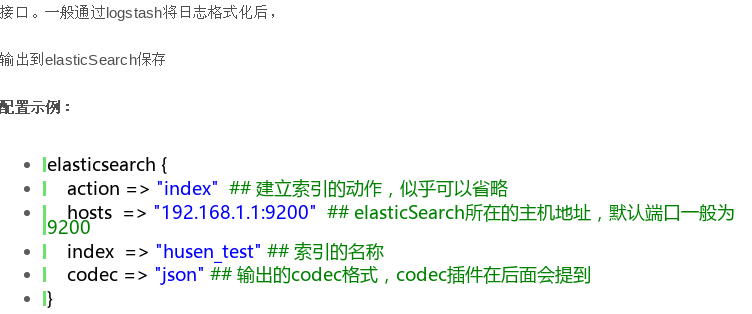
**5 start\_position=> "beginning"**

**6 }}**

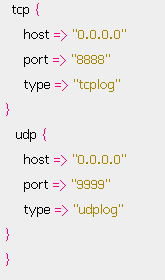
**7 filter{}**

**8 output{stdout{codec=>rubydebug}}**

1. **输出到elasticsearch**



4.tcp/udp模块



导入到elasticsearch应用:

filter中用一个或者多个if[type]==”apachelog”{grok{match=> }} 来分类匹配不同日志;同理,output中对应一下,分别输出存储到不同库.

此时input利用filebeats插件, input{beats{port=>”5044”}},在特定服务器上装包配置filebeats从而获得该服务器运行日志.

============================================

脚本小记:ssh-expect.sh

#!/bin/bash

ssh-keygen -N '' -f /root/.ssh/id\_rsa

code(){

expect<<EOF

spawn ssh-copy-id 192.168.4.1$1

expect "(yes/no)?" { send "yes\r" }

expect "password:" { send "123456\r" }

expect "#" { send "exit"\r }

EOF

}

for a in {1..6}

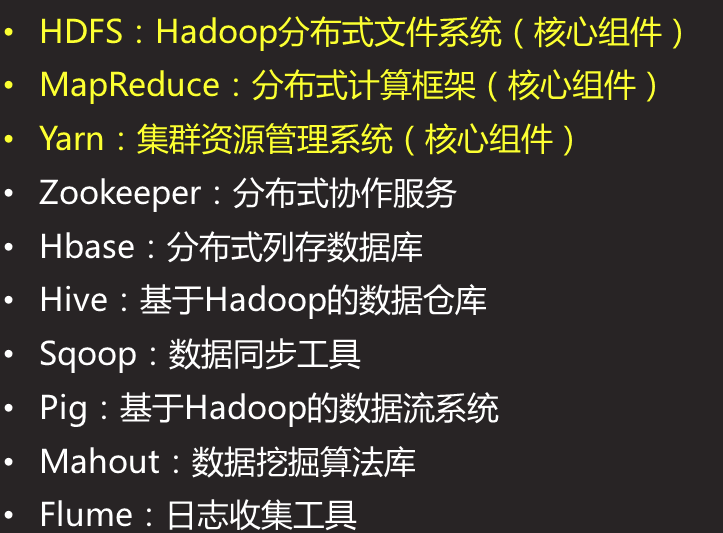
do

code $a

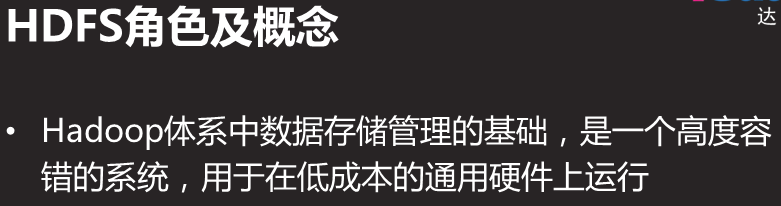
done

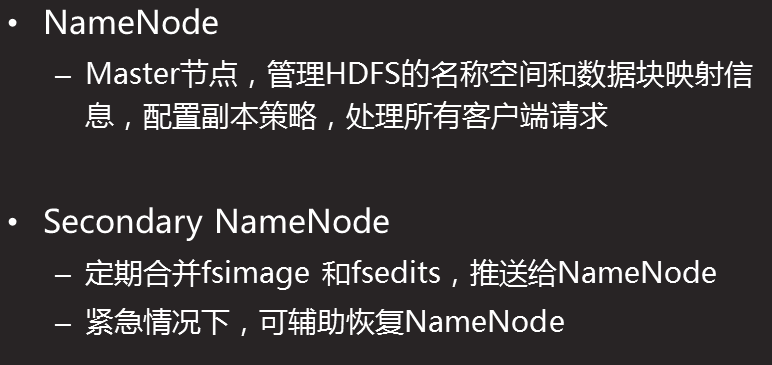
**HADOOP**

1.hadoop常用组件

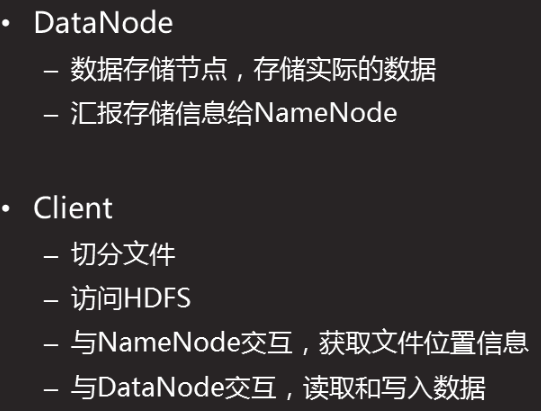


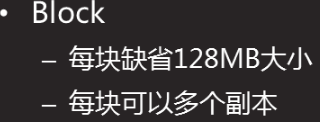
1. hafs的角色











1.DataNode存储有fsimg(文件与数据块的映射表以及数据块与数据节点的映射表)和fsedits(数据块变更日志)

从最新检查点后，hadoop将对每个文件的操作都保存在edits中，为避免edits不断增大，secondary namenode就会周期性合并fsimage和edits成新的fsimage，相当于打补丁,edits再记录新的变化。

这种机制有个问题：因edits存放在Namenode中，当Namenode挂掉，edits也会丢失，导致利用secondary namenode恢复Namenode时，会有部分数据丢失。

1. mapreduce

jobtracker 负责接受客户端请求,分解一系列任务后分派给tasktracker

1. Yarn(一个集群)

分布式资源管理系统,核心思想是将jobtracker和tasktracker进行分离

1. 完全分布式HDFS:

由namenode,secondarynamenode

以及数个datanode组成,搭建集群时可以设置dfs.replication值,意为存盘的份数,冗余份数.

删除节点:迁移数据-->下线服务节点删除

hdfs.site.xml加入

<property>

<name>dfs.hosts.exclude</name>

<value>/usr/local/hadoop/etc/hadoop/exclude</value>

</property>

[root@nn01 hadoop]# vim /usr/local/hadoop/etc/hadoop/exclude

node4

--------> ./bin/hdfs dfsadmin -refreshNodes //进行数据迁移

./bin/hdfs dfsadmin -report

./bin/hdfs namenode -format

./sbin/start-hdfs.sh

1. Zookeeper:

socat - tcp:es1:2181 (连接api,进入zookeeper页面访问)

1. Kafka 消息中间件

读缓存,写队列 ---->读遇到瓶颈,设置缓存;写有困难.设置消息队列

7.

**++++++++++++++++++++++++++++++**

**拓展**

1.Ubuntu系统:deb包管理

apt-get:Advanced Package Tool --->yum

dpkg -L

2.swap空间的作用可简单描述为：当系统的物理内存不够用的时候，就需要将物理内存中的一部分空间释放出来，以供当前运行的程序使用。那些被释放的空间可能来自一些很长时间没有什么操作的程序，这些被释放的空间被临时保存到Swap空间中，等到那些程序要运行时，再从Swap中恢复保存的数据到内存中。这样，系统总是在物理内存不够时，才进行Swap交换。