第四阶段-----云平台与架构

CLOUD(云平台)目录

day01: 虚拟化技术,virsh管理,云平台概述

day02: Openstack概述,部署Openstack及基础操作

day03: Openstack用户,配额管理,云主机类型管理,镜像管理,网络管理,安全与实例管理,计算节点扩容

day04: Docker概述,部署Docker,Docker镜像,Docker基础命令

day05: Docker自定义镜像与仓库,持久化存储,Docker网络架构

ARCHITECTURE(架构)目录

day01: ansible基础,ad-hoc(配置文件与host文件),批量配置管理(多模块)

day02: ansibleplaybook基础,playbook进阶

day03: 分布式ELK平台,ES集群安装,扩展插件

day04: Kibana安装与使用,Logstash配置扩展插件

day05: 大数据概述,Hadoop概述,Hadoop安装与配置,HDFS

day06: 完全分布式,节点管理,NFS网关

day07: 常用组件,kafka集群配置

##################################################

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

git地址: git://43.254.90.134/nsd1903.git

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

一.KVM简介

1.1.0 第四阶段专用虚拟机模板

--修改文件/etc/clone.conf

# vim /etc/clone.conf

name=base

# clone-vm7

密码是 a

1.1.2 virtualization 资源管理

--x个物理资源-->y个逻辑资源

--实现程度: 完全,部分,硬件服务(CPU)

1.1.3 虚拟化主要厂商及产品

系列 PC/服务器版代表

VMware vSphere, VMware Workstation, esxi(生产环境)

RedHat KVM,RHEV

Microsoft VirtualPC, Hyper-V

Citrix Xen

Oracle Oracle VM VirtualBox

1.2.0 安装虚拟化服务器平台

\*KVM /QEMU /LIBVIRTD

--KVM是linux内核的模块,它需要CPU的支持,采用硬件辅助虚拟化技术 Intel-VT,AMD-V,内

存的相关如Intel的EPT和AMD的RVI技术

--QEMU是虚拟化的仿真工具,通过ioctl与内核kvm交互完成对硬件的虚拟化支持

--Libvirt是一个对虚拟化管理的接口和工具,提供用户端程序 virsh,virt-

install,virt-manager,virt-view与用户交互.

\*示所有进程信息，连同命令行

# ps -efww

1.2.1 安装虚拟化服务器平台

1) 必备软件

\*qemu-kvm

--为kvm提供底层仿真支持

\*libvirt-daemon

--libvirtd守护进程,管理虚拟机

\*libvirt-client

--用户端软件,提供客户端管理命令

\*libvirt-daemon-driver-qemu

--libvirtd 连接 qemu 的驱动

2) 可选功能

--virt-install //系统安装工具

--virt-manager //图形管理工具

--virt-v2v //虚拟机迁移工具

--virt-p2v //物理机迁移工具

3) 虚拟化平台的安装

#yum install -y qemu-kvm libvirt-daemon \

libvirt-client libvirt-daemon-driver-qemu

#systemctl start libvirtd

1.2.2 KVM虚拟机的组成

1) 虚拟机的组成

--内核虚拟化模块(KVM)

--系统设备仿真(QEMU)

--虚拟机管理程序(LIBVIRT)

--一个XML文件(虚拟机配置声明文件)

--位置 /etc/libvirt/qemu/

--一个磁盘镜像文件(虚拟机的硬盘)

--位置 /var/lib/libvirt/images

1.3.0 管理KVM平台

1.3.1 virsh命令工具介绍

\*提供管理各虚拟机的命令接口

--支持交互模式,查看/创建/停止/关闭...

--格式: virsh 控制指令 [虚拟机名称] [参数]

1.3.2 virsh命令工具介绍

--list [--all] 列出虚拟机

--start|shutdown|reboot 虚拟机启动,停止,重启

--destroy 强制停止虚拟机

--define|undefine 根据xml文件 创建/删除 虚拟机

--console 连接虚拟机的console

2) virsh 虚拟机管理

--edit 修改虚拟机的配置

--autostart 设置虚拟机自启动

--domiflist 查看虚拟机网卡信息

--domblklist 查看虚拟机硬盘信息

vda(目标) (源) /var/lib/libvirt/images/tedu\_node20.img

3) virsh 虚拟网络管理.(虚拟交换机)

net-list [--all] 列出虚拟网络

net-start 启动虚拟交换机

net-destroy 强制停止虚拟交换机

net-define 根据xml文件 创建虚拟网络

net-undefine 删除虚拟网络设备

net-edit 修改虚拟交换机配置

net-autostart 设置虚拟交换机自启动

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

virsh基本管理操作

1.列出当前正在运行的虚拟机

2.查看虚拟机的信息

3.管理虚拟机

4.设置虚拟机开机自动运行

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2.1.0 常用镜像盘类型

\*虚拟机的磁盘镜像文件格式

特点 RAW QCOW2

KVM默认 否 是

I/O效率 高 较高

占用空间 大 小

压缩 不支持 支持

后端盘复用 不支持 支持

快照 不支持 支持

cow copy on write 写时复制

镜像盘:

2.1.1 qemu-img

\* qemu-img是虚拟机的磁盘管理命令,支持非常多的磁盘格式,例如raw,qcow2,vdi,vmdk等.

\* qemu-img命令格式

--qemu-img 命令 参数 块文件名称 大小

--常用命令

--create 创建一个磁盘

--convert 转换磁盘格式

--info 查看磁盘信息

--resize 扩容磁盘信息

\*基于后端盘可以创建很多前端盘,还原脚本,创建虚拟机脚本都可以通过这个来实现.

# qemu-img create -f qcow2 ./f1.img 20G -b node.qcow2//创建爱你一个磁盘文件

-f 格式 ./f1.img 保存路径 20G 最大容量. -b 指定后端盘

# qemu-img info /var/lib/libvirt/images/node1.img //查看磁盘文件

2.1.2 COW技术原理

\* copy on write 写时复制

--直接映射原始盘的数据内容

--当数据有修改要求时,在修改之前自动将旧数据拷贝入前端盘后,对前端盘进行修改.

--原始盘始终是只读的.

老牛原始盘:

/var/lib/libvirt/images/.node\_tedu.qcow2

2.2.0 xml管理

2.2.1 创建xml文件

\*对虚拟机的配置进行调整

--编辑: virsh edit 虚拟机名

#virsh edit tedu\_node20

2 <name>tedu\_node20</name>

3 <uuid>6e1f66fe-54a7-4842-89d2-ecb44ebea23a</uuid> //uuid

4 <memory unit='KiB'>1488282</memory> //最大内存

5 <currentMemory unit='KiB'>1488282</currentMemory> //当前内存

6 <vcpu placement='static'>2</vcpu> //cpu个数

eg: 修改xml文件,修改cpu,内存等参数

# virsh edit tedu\_node20

4 <memory unit='KiB'>4488282</memory> //最大内存

5 <currentMemory unit='KiB'>4488282</currentMemory> //当前内存

6 <vcpu placement='static'>4</vcpu>

#virsh shutdown tedu\_node20

#virsh reset tedu\_node20 //刷新xml配置

#virsh start teduu\_node20

ssh 192.168.1.100 :

# free -h //查看内存,更新至4G

2) 对虚拟机的配置进行调整

--编辑: virsh edit 虚拟机名

--若修改磁盘

#virsh edit tedu\_node20

<disk type='file' device='disk'>

<driver name='qemu' type='qcow2'/>

<source file='/var/lib/libvirt/images/tedu\_node20.img'/> //磁盘路径

<target dev='vda' bus='virtio'/>

<address type='pci' domain='0x0000' bus='0x00' slot='0x05' function='0x0'/>

</disk>

若修改网络

<interface type='bridge'>

<mac address='52:54:00:33:25:c0'/>

<source bridge='vbr'/> //网卡连接哪个交换机

<model type='virtio'/>

<address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>

</interface>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

批量创建虚拟机

3.1.1 克隆脚本命令

--创建虚拟机

#cd /var/lib/libvirt/images

# qemu-img create -f qcow2 -b .node\_base.qcow2 nsd1903.img 30G

--修改xml文件

#sed 's,node,node1,' node.xml >

--定义xml文件

--开启虚拟机

1) 创建磁盘文件

# cd # cd /var/lib/libvirt/images/

# qemu-img create -f qcow2 -b .node\_base.qcow2 nsd1903.img 30G

2) 修改xml文件

# cp .node\_base.xml /etc/libvirt/qemu/nsd1903.xml

# vim /etc/libvirt/qemu/nsd1903.xml

2 <name>nsd1903</name>

26 <source file='/var/lib/libvirt/images/nsd1903.img'/>

3) 创建虚拟机

# cd //etc/libvirt/qemu

# virsh define nsd1903.xml //创建虚拟机

# virsh start nsd1903

# virsh console nsd1903 //连接虚拟机

3.1.2 网卡及配置文件

网络配置文件说明

-- /etc/sysconfig/network-scripts/ifcff-eth0

ONBOOT="yes" //开机启动

NM\_CONTROLLED="no" //不接受NetworkManager控制

TYPE="Ethernet" //类型

BOOTPROTO="static" //协议(static静态|dhcp|none聚合链路)

IPADDR="192.168.1.10" //IP地址

NETMASK="255.255.255.0" //子网掩码

GATEWAY="192.168.1.254" //默认网关

3.1.3 virsh 扩容磁盘

\* virsh虚拟机管理

--domblklist 查看虚拟机硬盘信息

--blockresize --path [绝对路径] --size 50G tedu\_node20

\*扩容思路:

--首先扩容硬盘

--其次分区

1) 硬件扩容 192.168.1.254

# virsh domblklist tedu\_node20

# virsh blockresize --path /var/lib/libvirt/images/tedu\_node20 --size 50G tedu\_node20

2)分区扩容 192.168.1.100

# LANG=C //语言设置为英文

# lsblk //查看硬盘设置是否有多余空间

#growpart /dev/vda 1 //扩容第一个分区

#xfs\_growfs / //扩容文件系统

#df -h //检查扩容结果

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

云平台

4.1.0 云平台:

4.1.1 Openstack 简介

\*Openstack是由一个NASA(美国国家航空航天局)和Rackspace合作研发并发起的项目

\*Openstack 是一套IaaS解决方案

\*Openstack是一个开源的云计算管理平台

\*以Apache许可证为授权

2) 什么是云

现阶段广为接受的是美国国家标准与技术研究源(NST)定义:

云计算是一种按使用量付费的模式,...

#####################################################

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

云计算部署与管理

1.0什么是云计算

\*基于互联网的相关服务的增加,使用和交付模式

1.1 IaaS,即基础设施即服务.

1.2 PaaS,平台即服务

1.3 SaaS,软件即服务

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Openstack

1.1.0 什么是Openstack

\*Openstack是一套IaaS解决方案

\*Openstack是一个开源的云计算管理平台

1.1.1 penstack的七大组件

1) Horizon

--用于管理Openstack各种服务的,基于web的管理接口.(Apache)

--通过图形界面实现创建用户.管理网络,启动实例等操作.

2)Nova 三.

--在节点上用管理虚拟机的服务

--Nova是一个分布式的服务,能够与Keystone交互实现认证,与Glance交互实现镜像管理.

--Nova被设计成在标准硬件上能够进行水平扩展

--启动实例时,如果有则需要下载镜像.

3)Keystone 一

--为其他服务提供认证和授权的集中身份管理服务

--也提供了集中的目录服务

--支持多种身份认证模式,如密码认证,令牌认证,以及AWS(亚马逊web服务)登录

--为用户和其他服务提供SSO认证服务

4)Glance [类似yum] 后端盘磁盘镜像组件 二.

--扮演虚拟机镜像注册的角色

--允许用户为直接存储拷贝服务器镜像 //后端盘

--这些镜像可以用于新建虚拟机的模板

5)Cinder //存储组件.逻辑卷存储 五.

--为虚拟机管理存储卷的服务

--为运行在Nova中的实例提供永久的块存储

--可以通过快照进行数据备份

--经常应用在实例存储环境中,如数据库文件

6)Swift //存储组件 对象存储 六.

7)Neutron //网络组件 四.

--一种软件定义网络服务

--用户创建网络,子网,路由器,管理浮动IP地址

--可以实现虚拟交换级,虚拟路由器

--可用于在项目中创建VPN.

1.2.1 安装Openstack, 虚拟机准备

1.2.2 虚拟机配置

1) 准备虚拟机3台,配置如下:

\* openstack 管理主机

--2CPU,6G内存,50G硬盘. //内存设7.0G

--配置静态IP:192.168.1.10

2) nova01,nova02 计算节点\*2

--2CPU,4.5G内存,100G硬盘

--配置静态IP:192.168.1.11(12)

eg: 配置 openstack管理主机

扩容硬盘:

真机: 修改虚拟机硬盘大小

# virsh domblklist openstack //显示opentack虚拟机的磁盘文件

# virsh blockresize --path /var/lib/libvirt/images/openstack.img --

size 50G openstack

openstack 192.168.1.10:

# LANG=C

# growpart /dev/vda 1

# xfs\_growfs /

# df -h

扩容内存

真机 192.168.1.254:

#virsh edit openstack //修改openstack虚拟机的内存大小

<memory unit='KiB'>7340032</memory> //修改为7G

<currentMemory unit='KiB'>7340032</currentMemory>

# virsh reset openstack //刷新虚拟机配置文件

# virsh shutdown openstack

# virsh start openstack

修改虚拟机IP:

192.168.1.10/11/12/13

# vim /etc/sysconfig/network-scripts/ifcfg-eth0

BOOTPROTO="static"

IPADDR="192.168.1.10"

NETMASK="255.255.255.0"

GATEWAY="1992.168.1.254"

# /etc/init.d/network restart //重启网卡配置

1.2.3 配置DNS

\*系统环境准备

--openstack安装时需要使用外部dns来解析域名

192.168.1.10/11/12

# vim /etc/resolv.conf

# search localhost //删除search的行

nameserver 192.168.1.254

# vim /etc/hosts

192.168.1.10 openstack

192.168.1.11 nova01

192.168.1.12 nova02

1.2.4 NTP服务

\*nova服务器之间的时间必须保持一致

\*编辑配置文件 /etc/chrony.conf

192.168.1.10/11/12

# vim /etc/chronyd.conf

server gateway iburst改为->

server 192.168.1.254 iburst

# systemctl restart chronyd

# systemctl enable chronyd

# chronyc sources -v //查看与真机时间同步是否正常

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

^\* gateway

//显示为 '^' = server, '\*' = current synced 正常

1.3.0 配置yum仓库

\* CentOS7-1804.iso 系统软件

192.168.1.10/11/12

# vim /etc/yum.repos.d/openstack.repo //yum源配置为包含repodata文件的目录

[extras]

name=extras

baseurl="ftp://192.168.1.254/extras"

enabled=1

gpgcheck=0

[openstack]

name=openstack package

baseurl="ftp://192.168.1.254/openstack/rhel-7-server-openstack-10-rpms"

enabled=1

gpgcheck=0

[packstack]

name=packstack

baseurl="ftp://192.168.1.254/openstack/rhel-7-server-openstack-10-devtools-rpms/"

enabled=1

gpgcheck=0

# yum repolist

repolist: 10,670 //最终yum源为 10670个包

# rsync -avXSH /etc/yum.repos.d nova02:/etc/ //同步yum源

-a 递归拷贝,保留权限属性

-v 显示详细信息

-X 显示保留扩展权限, setfacl设置的

-S 软连接单独处理,

-H 硬链接单独处理,

--delete 减少屏幕输出

1.3.1 安装额外软件包

需要提前安装的软件包: //三台虚拟机均需要安装

--qemu-kvm

--libvirt-daemon

--libvirt-daemon-driver-qemu

--libvirt-client

--python-setuptools //openstack 是Python写的

\*软件安装

# yum -y install qemu-kvm libvirt-client libvirt-daemon //共91个包

libvirt-daemon-driver-qemu.x86\_64 python-setuptools

1.3.2 检查基础环境

\* 是否卸载firewall和NetworkManager

# rpm -qa | grep firewalld

# rpm -qa | grep -i networkmanager

\* 检查配置主机网络参数(静态IP)

\* 主机名必须能够相互ping通

\* 检查配置主机yum源(4个,10670)

\* 依赖软件包是否安装

\* 检查NTP服务器是否可用

\* 检查/etc/resolv.conf不能有search开头的行

1.3.3 配置packstack

\*安装openstack需要使用packstack

--首先安装packstack 只需在192.168.1.10 上安装

# yum -y install openstack-packstack

\*使用packstack创建通用应答文件

# packstack --gen-answer-file=answer.ini

42: 是否使用对象存储

45: openstack计费相关

49: openstack计费相关

53: openstack计费相关

75: 时间服务器NTP

98: 在哪个机器上安装nova虚拟机

102: 在哪个机器上需要配置network网络

333: 与330行对应,用户密码.

840: newwork需要支持什么网络协议(直连网络,vxlan:openstack内部vlan)

910: 虚拟交换机

921: 将eth0网卡变为虚拟交换机的一个接口.

1179: DEMON禁用教学安装.

42 CONFIG\_SWIFT\_INSTALL=n

45 CONFIG\_CEILOMETER\_INSTALL=n

49 CONFIG\_AODH\_INSTALL=n

53 CONFIG\_GNOCCHI\_INSTALL=n

75 CONFIG\_NTP\_SERVERS=192.168.1.254

95 CONFIG\_CONTROLLER\_HOST=192.168.1.10

98 CONFIG\_COMPUTE\_HOSTS=192.168.1.11

102 CONFIG\_NETWORK\_HOSTS=192.168.1.10,192.168.1.11

333 CONFIG\_KEYSTONE\_ADMIN\_PW=a

840 CONFIG\_NEUTRON\_ML2\_TYPE\_DRIVERS=flat,vxlan

910 CONFIG\_NEUTRON\_OVS\_BRIDGE\_MAPPINGS=physnet1:br-ex

921 CONFIG\_NEUTRON\_OVS\_BRIDGE\_IFACES=br-ex:eth0

1179 CONFIG\_PROVISION\_DEMO=n

1.3.4 一键部署Openstack

\*如果前期环境准备无误,只要耐心等待安装结束即可

\*根据主机配置不同,安装过程需要20分钟左右或更久

\*如果出现错误,根据屏幕上给出的日志文件进行拍错

# packstack --answer-file=answer.ini

成功提示:

\*\*\*\* Installation completed successfully \*\*\*\*\*\*

1.3.5 Horizon

\* Horizon BUG处理

\*安装虽然没有报错,但默认无法打开Horizon,这是一个软件的配置BUG.

\* /etc/httpd/conf.d/15-horizon\_vhost.conf

\* WSGIApplicationGroup %{GLOBAL}

1) 修改apache配置文件:

# cd /etc/httpd/conf.d

# vim 15-horizon\_vhost.conf //在文件末尾请添加

WSGIDaemonProcess apache group=apache processes=3 threads=10

user=apache

WSGIProcessGroup apache

WSGIApplicationGroup %{GLOBAL} //添加此行

# apachectl graceful //重新载入配置文件

\*在真机上访问 http://192.168.1.10

1.4.0 网络

1) 查看外部OVS网桥

\* cr-ex为外部OVS网桥

# cat /etc/sysconfig/network-scripts/ifcfg-br-ex

2)eth0为外部OVS网桥的端口

# cat /etc/sysconfig/network-scripts/ifcfg-eth0

3) 验证OVS配置

# ovs-vsctl show //显示ovs信息,查看交换机信息

Bridge br-ex //虚拟交换机

... ...

Port "eth0" //端口eth0

Interface "eth0" //接口

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Horizon配置

2.1.0 Horizon是一个管理,控制OpenStack服务的web控制面板.也称为Dashboard仪表盘

2.1.1 Horizon

\* Horizon BUG处理

\*安装虽然没有报错,但默认无法打开Horizon,这是一个软件的配置BUG.

\* /etc/httpd/conf.d/15-horizon\_vhost.conf

\* WSGIApplicationGroup %{GLOBAL}

1) 修改apache配置文件:

# cd /etc/httpd/conf.d

# vim 15-horizon\_vhost.conf //在文件末尾请添加

WSGIDaemonProcess apache group=apache processes=3 threads=10

user=apache

WSGIProcessGroup apache

WSGIApplicationGroup %{GLOBAL} //添加此行

# apachectl graceful /systemctl reload //重新载入配置文件

\*在真机上访问 http://192.168.1.10

2.1.2 文件keystonerc\_admin

# ls /root/keystonerc\_admin //存储账户和密码

2.1.3 openstack 命令行用法

# # source keystonerc\_admin //登录

####################################################

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

openstack 项目管理

项目:一组隔离的资源和对象,由一组关联的用户进行管理,在旧版里,也用租户tenant表示.

缺省情况下,packstack安装的openstack中有两个独立的项目

--admin:为admin账户创建的项目

--services:与安装的各个服务相关联

1.1.1 在Horizon管理项目

\*通过Horizon可以进行项目的创建和删除

创建 tedu项目,配置默认. \*为必填项.

1.1.2 基本概念

\*用户在openstack中用于身份认证

\*管理员用户admin一般在packstack安装过程中创建

\*其他用户由管理员创建,并指定可以访问的项目

\*非管理员用户创建后,保存到MariaDB中

\*非管理员用户具有以下权限

--启动实例

--创建卷和快照

--创建镜像

--分配浮动IP

--创建网络和路由器

--创建防火墙以及规则,规则策略

--查看网络 拓扑,项目使用概况等

1.1.3 创建用户

用户名: uu

密码: a

主项目: tedu

角色: member

1.1.4 修改用户密码:

在192.168.1.10

普通用户修改密码:

# source keystonerc\_admin //载入环境

# openstack user list

# openstack user set --password aa uu //将用户uu密码修改为aa

1.1.5 配额基础

\*管理员可以通过配额限制,防止资源的过度使用

\*配额基本项目,限制每个项目可以使用多少资源

\*这些操作上的功能限制,赋予了管理员对每个项目的精准控制

1.1.6 资源参数

\*安全组规则: 指定每个项目可用的规则数

\*核心: 指定每个项可用的VCPU核心数

\*固定IP地址: 指定每个项目可用的固定IP数

\*浮动IP地址: 指定每个项目可用的浮动IP数

\*注入文件大小: 指定每个项目内容大小

\*注入文件路径: 指定每个项目注入的文件路径长度

\*注入文件: 指定每个项目允许注入的文件数目

\*实例: 指定每个项目可创建的虚拟机实例数目

\*密钥对: 指定每个项可创建的秘钥数

\*元数据: 指定每个项目可用的元数据数目

\*内存: 指定每个项目可用的最大内存

\*安全组: 指定项目可创建的安全组数目

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

云主机类型:

2.1.1 基本概念

\*云主机类型就是资源的模板

\*它定义了一台云主机可以使用的资源,如内存大小,次哦按容量和CPU核心数等

\*Openstack提供了几个默认的云主机类型

\*管理员还可以自定义云主机类型

2.1.2 云主机类型参数

\*Name: 云主机类型名称

\*ID: 云主机类型ID,系统自动生成一个UUID

\*VCPUs: 虚拟CPU数目

\*RAM(MB): 内存大小

\*Root disk(GB): 外围磁盘大小,如果希望使用本地此案,设置为0

\*临时磁盘: 第二个外围磁盘

\*swap磁盘: 交换磁盘大小

2.1.2 创建云主机类型

名称: hana

vcpu: 2

内存: 512MB

根磁盘: 1G //不能比后端盘小

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

镜像基础

3.1.1 镜像

1) 基本概念

\*在红帽Openstack平台中,镜像指的是虚拟磁盘文件,磁盘文件中应该已经安装了可启动的操作系统

\*镜像管理功能由Glance服务提供

\*它形成了创建虚拟机实例最底层的快结构

\*镜像可以由用户上传,也可以通过红帽官方站点下载

2) Glance磁盘格式

\* raw: 非结构化磁盘镜像格式

\* vhd: VMware,Xen,Microsoft,VirtualBox等均支持的通用磁盘格式

\* vmdk: 是Vmware的虚拟磁盘格式

\* vdi: VIrtualBox虚拟机和QEMU支持磁盘格式

\* iso: 光盘数据内容的归档格式

\* qcow2: QEMU支持的磁盘格式,空间自动扩展,并支持写时复制copy-on-write

3) 镜像服务

\*镜像服务提供了服务器镜像的拷贝,快照功能,可以作为模板快速简历,启动服务器

\*镜像服务维护了镜像的一致性

\*当上传镜像时,容器格式必须指定

\*容器格式指示磁盘文件格式是否包含了虚拟机元数据

4) 镜像容器格式

\*bare: 镜像中没有容器或元数据封装

\*ovf: 一种开源的文件规范,描述了一个开源,安全,有效,可拓展的便携式虚拟打包以及软件分布格式

\*ova: OVA归档文件

\*aki: 亚马逊内核镜像

\*ami: 亚马逊主机镜像

3.2.1 创建镜像:

# qemu-img info small.img //查看镜像格式

镜像--> 添加镜像:

镜像名称: CentOS6

文件: small.img

镜像格式: qcow2

镜像共享: 公有镜像

创建镜像

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

4.1.0 网络和路由

4.1.1 Openstack网络工作原理

\*实例被分配到子网中,以实现网络连通性

\*每个项目可以有一到多个子网

\*在红帽的Openstack平台中,Openstack网络服务是缺省的网络选项,Nova网络服务作为备用.

\*管理员能够配置丰富的网络,将其他Openstack服务连接到这些网络的接口上

\*每个项目都能拥有多个私有网络,各个项目的私有网络互相不受干扰.

4.1.2 网络类型

\*项目网络:由Neutron提供的项目内部网络,网络间可用VLAN隔离.

4.1.3 配置网络

1) 配置公共网络 (外网)

用户: admin

管理员---->网络--> 创建网络

名称: wan

项目: tedu

供应商网络类型: Flat (直连网络)

物理网络:physnet (br-ex桥接的eth0)

共享的 打钩

外部网络 打钩

2) 配置私有网络

用户:uu 密码:aa

网络---->网络拓扑--->创建网络

网络名称: lan

下一步

子网名称: lan\_subnet

网络地址:10.10.10.0/24 //内网

网关IP: 10.10.10.254 //预留网关

下一步

分配地址池:

10.10.10.100,10.10.10.200 //起始地址,终止地址

DNS服务器:

192.168.1.254 //只要能解析地址就行

3) 创建wan的子网

鼠标左键点wan,点击右上角增加子网

子网名称:wan\_subnet

网络地址:192.168.1.0/24 //与br-ex交换机配置一致

网关IP: 192.168.1.254

下一步

分配地址池:

192.168.1.150,192.168.1.200

已创建

4)添加路由

用户: uu 密码: aa

路由名称: hana

外部网络: wan

创建路由

增加接口:

子网:lan:10.10.10.0/24(lan\_subnet)

IP地址:10.10.10.254

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

5.1.0 创建云主机

5.1.1 创建云主机

计算--->云主机数量

详情: 云主机名称: ecs-1

源: 选择源: 镜像 ,

创建新卷: 否

可用: 加号

云主机类型:

hana 加号

网络: lan 加号

创建云主机 两台云主机

5.1.2 exs-1 云主机控制台:

点击此处只显示控制台:

计算-->访问和安全,分配IP给项目

云主机数量--->绑定浮动IP-->选择一个IP地址--->关联

防火墙默认: 云主机能访问任何人,任何人不能访问云主机

访问和安全

创建安全组: host1

管理规则

添加规则 :规则 ALL ICMP

方向:入口

云主机数量---->编辑安全组--->添加host 1

ls openstack-nova-compute

/etc/nova/nova.conf

8430

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

扩容nova节点

安装nova02

1.1 删除云主机,(内存原因)

1) 是否卸载firewalld和NetworkManager

检查基础环境: 参考Nova01

2)是否是静态IP

3)主机名相互能ping通

4)检查yum源配置 (10670个包)

5) 依赖软件包是否安装(共5个包)

qemu-kvm libvirt-daemon libvirt-client libvirt-daemon-

python-setuptools

6)NTP服务器是都可用

#chronyc source -v

7) 检查/etc/resolve.conf,没有search的行

1.2

openstack:

# vim answer.ini

98,102行

192.168.1.11,192.168.1.12

# packstack --answer-file=answer.ini

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

迁移云主机

admin:

主机-->云主机热迁移 nova01--->nova02.(允许块设备迁移)

nova01

#watch -n 1 'virsh list --name' //每1秒执行命令

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

切屏软件: screen

#screen //启动screen

ctrl+a,shift+s //橫切

ctrl+a,ctrl+\ //竖切

ctrl+a,tab //切换控制窗口

ctrl+a,c //创建新的终端

ctrl+a,n //与其他终端同步

ctrl+d //退出

vim -p //跳转标签gt

#######################################################

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1.1.0 什么是容器

\*容器技术已经成为应用程序封装和交付的核心技术.

\*容器技术的核心有以下几个内核技术组成:

--Cgroups(Control Groups)--资源管理

--NameSpace--进程隔离(命名空间)

--SELinux安全

\*由于是在物理机上实施隔离,启动一个容器,可以像启动一个进程一样快速

1.1.1 什么是Docker

\*Docker是完整的一套容器管理系统

\*Docker提供了一组命令,让用户更加方便直接地使用容器技术,而不需要过多关心底层内核技术

\*物理机上实施的隔离.

1.1.2 Docker优点

\*相比于传统的虚拟化技术,容器更加简洁高效

\*传统虚拟机需要给每个VM安装操作系统

\*容器使用的共享公共库和程序.(真机的)

\*容器直接使用真机的操作系统. 虚拟化是自己虚拟操作系统.

1.1.3 Docker缺点

\*容器的隔离性没有虚拟化强

\*共用Linux啮合,安全性有先天缺陷

\*SELinux难以驾驭

\*监控容器和容器排错是挑战

1.1.4 NameSpace 隔离指标,六大命名空间

1) hostname //主机名

2) process //只能列出容器内部的进程

3) network //网络

4) filesystem //文件系统,被限制在某个文件夹中.

5) user //用户

6) ipc //信号向量,信号只在容器内有效.

nginx事件型驱动 epull触发机制

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

安装Docker

2..1.0 安装前准备

\* 需要64位操作系统

\* 强烈推荐RHEL7 # cat /etc/redhat-release

\* 关闭防火墙

2.1.1 安装Docker

在两台base虚拟机 ip为192.168.1.31, 192.168.1.32上安装

1) 配置yum源.

--docker软件包位于光盘RHEL7-extras.iso中

--在物理机ftp目录下创建文件夹 extras

#mkdir /var/ftp/extras

--把光盘挂载到这个目录下

mount

2) 在虚拟机中配置yum源,指向刚刚创建的目录

# vim /etc/yum.repos.d/local.repo

[Docker]

name=extra

baseurl="ftp://192.168.1.254/extras"

enabled=1

gpgcheck=0

3) 软件包安装

# yum -y install dacker

# systemctl restart docker

# systemctl enable docker

4) docker安装好后可以查看到网卡信息:

docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500

inet 172.17.0.1 netmask 255.255.0.0 broadcast 0.0.0.0

2.2.0 Docker镜像

2.2.1 什么是镜像

\*在Docker中容器是基于镜像启动的

\*镜像是启动容器的核心

\*镜像采用分层设计, 镜像包含所有层.

\*使用快照的COW技术,确保底层数据不丢失

虚拟机: tedu\_node15.img 3层

老牛 tedu层 .node\_tedu.qcow2 2层

base层 .node\_base.qcow2 1层

2.2.2 Docker hub镜像仓库

\* https://hub.docker.com

\* Docker官方提供公共镜像的仓库(Registry)

# docker search busybox

# docker search centos

# docker search nginx

2.2.3 镜像操作

1) 下载镜像(从镜像仓库中下载镜像)

# docker search busybox //搜索镜像仓库的 busybox镜像

# docker help pull //查看命令帮助

# docker pull docker.io/busybox

# docker images //查看本地已有镜像

镜像名称+TAG 表示唯一性. 表示为: REPOSITORY:TAG

IMAGE ID: 表示物理文件

2) 上传镜像(上传镜像到仓库)

# docker help push //查看命令帮助

# docker push docker.io/busybox

3) 导出镜像(将本地镜像导出为tar文件). -o: output

# docker save dacker.io/busybox:latest -o busybox.tar

#ls

4) 导入镜像(通过tar包文件导入镜像)

# scp nginx.tar root@192.168.1.31:/root

# ls

nginx.tar

# docker load -i busybox.tar //-i:input

# docker images //可以查看到busybox镜像

5)启动centos镜像生成一个容器

# docker images

# docker run -it docker.io/centos:latest /bin/bash

开启另一个终端(查看容器信息), 创建一个新的容器

/bin/bash 是指容器内的执行路径. 未知时,可以不填,默认启动. eg:/usr/bin/python

# docker ps //查看正在运行中的容器

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

案例2:镜像基本操作

1.导入镜像

2.导出镜像

3.启动镜像

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

3.1.0容器: 隔离特性验证

1.hostname //主机名

2.ps -ef //进程

3.ifconfig //ip

yum源删除原先的,创建与真机一致的yum源.

# yum provides ifconfig //查询未安装的命令对应的包

# rpm -qf /usr/sbin/ifconifg //查询已安装的命令软件

4.user //用户

5.filesystem //文件系统. ls /

6.ipc //信号向量

2) 区别容器与虚拟机

# pstree //查看进程树,区别容器和虚拟机

bash---pstree //没有操作系统,直接跑应用

# sleep 12345

真机上查看,并能结束容器上的进程.

# ps -ef | grep sleep

sleep 12345

# kill 25154

3.1.1 启动容器:

1) # docker run -it docker.io/centos:latest /usr/bin/python

2) 启动nginx镜像

# docker run -it docker.io/nginx:latest /bin/bash

/usr/sbin/sshd -D $OPTIONS //启动ssh服务

能与用户交互的放前台,不能与用户交互的放后台

ubuntu镜像:

linux系统: Redhat : centos

Debian : ubuntu

不同命令: dpkg (.deb): --->rpm -l查看已安装包,-i

apt-get :----> yum

3.1.2 docker 基本命令

1) 镜像常用命令

--docker images //查看镜像列表

--docker history +镜像 //查看镜像制作历史

--docker inspect +镜像/容器/虚拟交换机 //查看镜像底层信息

--docker pull +镜像 //下载镜像

--docker push +镜像 //上传镜像

# docker inspect docker.io/nginx

环境变量:

"Env": [

"PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",

默认启动命令:

"Cmd": [

"nginx",

"-g",

"daemon off;"

],

2) 镜像常用命令

--docker rmi //删除本地镜像,需要先删除对应容器,再删除镜像

--docker save //镜像另存为tar包

--docker load //使用tar包导入镜像

--docker search //搜索镜像

--docker tag //修改镜像名和标签

eg: 修改镜像名和标签:

# docker tag docker.io/centos:latest nsd:1903

eg: 镜像另存为tar包:

# dacker save docker.io/centos:latest -o centos.tar

eg: 导入镜像

# docker load -i centos.tar

3) 容器常用命令

--docker run

--docker inspect 容器id //查看容器底层信息

--docker ps //查看容器列表 -q 只显示容器id

--docker stop 容器id //关闭容器

--docker start 容器id //启动容器

--docker restart 容器id //重启容器

--docker attach|exec //进入容器,建议使用exec,exec可以启动子进程

--docker top 容器id //查看容器进程列表.(在容器外执行ps)

--docker rm 容器id //删除容器. 容器停了在删

eg: 运行容器

# docker run -it docker.io/centos:latest /bin/bash //-it启动交互式的终端

# docker run -itd docker.io/nginx:latest //-d 后台运行

eg:

# docker exec -it 72 /bin/bash

# echo $$ 查看当前进程的pid

# docker attach 连接的是pid为1的上帝进程,退出时attach关闭,pid=1的进程死了.

按ctrl+p,ctrl+q可以不杀死上帝退出.

eg: 过滤出运行容器的ip地址

# docker inspect -f "{{.NetworkSettings.IPAddress}}" 8f4d1f3c4d11 | grep

"{{.NetworkSettings.IPAddress}}" //获取IP地址 .下的NetworkSettings目录下的IPAddress项. .为最顶级

eg: 删除所有容器

# docker rm $(docker ps -aq)

# docker rm $(docker stop $(docker ps -aq)) //停止的容器并删除所有容器

##########################################################

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1.0 自定义镜像

1.1.1 docker commit

\*使用镜像启动容器,在该容器基础上修改

\*另存为一个新镜像

1)配置自定义镜像 (手工编排镜像)

# docker un -it docker.io/centos:latest //增删改数据,安装软件,修改配置文件等

①.修改yum源

②.安装常用的软件及命令

# yum -y install net-tools vim psmisc iproute tree lftp rsync

③.打包容器

docker commit askjdla46 myos:latest

④.删除容器

# docker rm askjdla46

⑤.查看myos共封装了几层

docker history myos:latest

⑥.查看自定义镜像:

# docker iamges

myos latest 5caa9a8dbc0a 3 minutes ago 319.3

1.1.2 Dockerfile

\*Dockerfile语法格式

--FROM: 基础镜像

--MAINTAINER:镜像创建者

--EXPOSE:开放的端口

--ENV:设置变量

--ADD:复制文件到镜像, 本地文件(当前目录下) 容器内路径

--RUN:制作镜像时执行的命令,可以有多个, 在容器内执行. 类似ssh.

--WORKDIR:定义容器默认工作目录, 永久性cd

--CMD:容器启动时执行的命令,仅可以有一条CMD.(启动的默认命令)

格式: CMD ["/bin/ls"] , CMD ["/bin/ls","-a","-l"]

\*使用Dockerfile工作流程

--mkdir base

--cd base

--vim Dockerfile

--docker build -t testos:latest Dockerfile所在目录

1) 编写dockerfile

# mkdir base

# vim Dockfile

FROM docker.io/centos:latest //基于哪个镜像,基础镜像

RUN rm -rf /etc/yum.repos.d/\*.repo //在容器内执行

ADD local.repo /etc/.yum.repos.d/local.repo //将本地文件放到镜像中

RUN yum -y install net-tools vim psmisc iproute tree lftp rsync

2)使用dockerfile 创建镜像

docker build -t testos:latest . //. 为Dockerfile所在

3) 查看新建镜像

# docker images

testos latest bdb0ce30ba42

4) 设置默认启动命令

FROM myos:latest

CMD ["/usr/sbin/python"]

1.1.3 制作ssh服务镜像

# mkdir ooxx

# vim Dockerfile

FROM myos:latest

RUN yum -y install openssh-server initscripts && \

sshd-keygen && \

echo "aa" | passwd --stdin root

ENV EnvironmentFile=/etc/sysconfig/sshd //设置环境变量,镜像里设置

EXPOSE 22 //开放的端口

CMD ["/usr/sbin/sshd","-D"]

# docker build -t myos:ssh . //创建镜像

手动创建过程

yum search sshd

rpm -ql openssl-server | grep service

yum provides /etc/rc.d/init.d/function //查看文件在哪个包里

sshd-keygen

/usr/sbin/sshd -D

pass root

aa

/usr/sbin/sshd -D //启动成功

1.1.4制作appache服务镜像

# vim Dockerfile

FROM myos:latest

RUN yum -y install httpd

ADD index.html /var/www/html/index.html

WORKDIR var/www/html/

ENV /etc/sysconfig/httpd

EXPORT 80

EXPORT 443

CMD ["/usr/sbin/httpd","-DFOREGROUND"]

docker内起服务:

rpm -ql httpd | grep service //systemctl 通过service文件启动

/usr/sbin/httpd -DFOREGROUND

1.1.5 同时启动多个服务的镜像

# vim Dockerfile

FROM myos:ssh //基于ssh的镜像做httpd服务

RUN yum -y install httpd

ADD index.html /var/www/html/index.html

ADD run.sh /etc/init.d/run.sh

WORKDIR var/www/html/

ENV /etc/sysconfig/httpd

EXPORT 80

EXPORT 443

CMD["/etc/init.d/run.sh"]

编写run.sh脚本:

vim /etc/init.d/run.sh

EnvironmentFile=/etc/sysconfig/sshd

/usr/sbin/sshd -D & //启动ssh

EnvironmentFile=/etc/sysconfig/httpd

/usr/sbin/httpd -DFOREGROUND //启动apache

# chmod 777 /etc/init.d/run.sh

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

自定义镜像仓库

2.1.1 自定义私有仓库

1)安装私有仓库(服务端)

# yum install docker-distribution

2)启动私有仓库,并设置开机自启

# systemctl start docker-distribution

# systemctl enable docker-distribution

3)仓库配置文件及数据存储路径

--/etc/docker-disribution/registry/config.yml

--/var/lib/registry

4) 客户端配置:

--修改配置文件 /etc/sysconfig/docker

--允许非加密方式访问仓库

24 INSECURE\_REGISTRY='--insecure-registry 192.168.1.31:5000' //授权地址

--docker 仓库地址

13 ADD\_REGISTRY='--add-registry 192.168.1.31:5000'

5) 重启docker服务

# systemctl restart docker

2.1.2 私有仓库\*上传镜像

1) 为镜像创建标签

格式: docker tag 镜像:标签 IP:5000/镜像:latest

# docker tag docker.io/busybox:latest 192.168.1.31:5000/busybox:latest

2)上传镜像

上传镜像的标签内包含地址和端口号

# docker push 192.168.1.31:5000/busybox:latest

2.1.3 docker2 上使用192.168.1.31的镜像仓库

1) 修改docker配置文件

# vim /etc/sysconfig/docker

13 ADD\_REGISTRY='--add-registry 192.168.1.31:5000'

24 INSECURE\_REGISTRY='--insecure-registry 192.168.1.31:5000'

# systemctl restart docker

2) 下载使用镜像

# docker run -it busybox:latest

2.1.4 查看私有仓库

\*

# curl http://192.168.1.31:5000/v2/\_catalog

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

docker持久化存储

3.1.0 卷的概念

\* docker容器不保持任何数据

\* 重要数据请使用外部卷存储(数据持久化)

\* 容器可以挂载真实机目录或共享存储为卷

eg: mount --bind webroot abc //将 abc 绑定到webroot,即成同一文件夹

3.1.1 主机卷的映射. (就是同一个文件夹)

\* 将真实机目录挂载到容器中提供持久化存储

--目录不存在就自动创建

--目录存在就直接覆盖掉

# docker run -it -v 真机dir:容器:dir docker.io/centos

案例3:NFS共享存储

1).服务器创建NFS共享存储目录,权限为rw

2).客户端挂载共享,并将共享目录映射到容器中

3).docker1 启动apache

4).docker2 启动apache

5.nginx和apache共享同一web目录

docker01

docker02:

yum -y install nfs-utils

docker run -itd /data:/var/www/html/ myos:httpd

curl -i 172.17.0.2

nginx -T //查看html目录

/usr/share/nginx/html

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

docker网络架构

4.1.0 docker网络三种模式

# docker network list

NETWORK ID NAME DRIVER SCOPE

21757f981d60 bridge bridge local

e521540f671e host host local

5c89b9909e19 none null local

4.1.1

# docker network --help //docker网络管理命令

1) 创建新交换机

# docker network create --subnet=10.10.10.0/24 docker1

# docker network list //查看网络列表

# ifconfig //查看网卡

案例4 创建自定义网桥:

1.启动4台容器

2.要求:容器1与容器2能够互通

容器3与容器4能够互通

容器(12)与容器(34)不能互通

创建交换机docker1

# docker network create --subnet=10.10.10.0/24 docker1

容器 1

# docker run -itd docker.io/myos

ip: 172.17.0.2

容器2

# docker run -itd docker.io/myos

ip: 172.17.0.3

容器3:

# docker run -itd --network=daocker1 docker.io/myos

ip: 10.10.10.2

容器4:

# docker run -itd --network=daocker2 docker.io/myos

ip: 10.10.10.3

4.2.1 客户端访问容器内的资源

\* 默认容器可以访问外网

\* 但外部网络的主机不可以访问容器内的资源

\* 容器的特性是可以把宿主主机变成对应的服务

--可以使用-p 参数把容器端口和宿主机端口绑定

-- -p 宿主机端口:容器端口

eg: 把docker1 变成httpd

# docker run -itd -p 80:80 -v /var/webroot:/var/www/html myos:httpd

eg: 把docker1 变成nginx

# docker run -itd -p80:80 -v /var/webroot:/usr/share/nginx/html nginx:latest

vim /etc/httpd/conf/httpd.conf

ServerName localhost:80

/usr/lib/systemd/system/httpd.service

#####################################################

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1.1.0ansible:

1.1.1 什么是ansible

1) 是2013一款IT自动化和DevOps软件,基于Python研发,糅合很多老运维工具的优点,实现了

批量操作系统配置,批量程序部署,批量运行命令等功能.

2)ansible可以实现

--自动化部署APP

--自动化管理配置项

--自动化持续交付

--自动化(AWS)云服务管理

3)优势

--活跃度(社区)

--学习成本

--使用成本

--编码语言

--性能

--使用是否广泛

4) ansible优点

--只需要SSH和Python即可使用

--无客户端

--ansible功能强大,模块丰富

--上手容易,门槛低

--基于Python开发,做二次开发更容易

--使用公司比较多,社区活跃

5) 特性

--模块化设计.调用特定的模块完成特定任务

--基于Python语言实现

--paramiko

--PyYAML(半结构化语言)

--Jinja2

--其模块支持JSON等标准输出格式,可以采用任何编程语言重写

6) 特性

--部署简单

--主从模式工作

--支持自定义模块

--支持playbook

--易于使用

--支持多层部署

--支持异构IT环境

1.1.2 工作流程:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2.1.1 ansible安装前提

\* /linux/04/ansible/\*.rpm 做yum源,给虚拟机用

1) 对管理主机要求: Python 2.6 或 Python 2.7

2) ansible使用以下模块,都需要安装

--paramiko #paramiko是Python的一个库，实现了SSHv2协议

--PyYAML

--Jinja2

--httplib2

--six

# yum -y install python-paramiko PyYAML python-jinja2 python-httplib2 python-six

3) 对于被托管主机

--ansible默认通过SSH协议管理机器

--被管理主机要开启SSH服务,允许ansible主机登录

--在托管节点上也需要安装Python2.5或以上版本

--如果托管节点上开启了SElinux,需要安装]libselinux-python

2.1.2 ansible环境准备

1) 启动6台虚拟机

--2cpu,1.5G以上内存,10G以上硬盘,1快网卡

主机名 IP地址 角色

ansible 192.168.1.40 管理主机

web1 192.168.1.41 托管主机

web2 192.168.1.42 托管主机

db1 192.168.1.43 托管主机

db2 192.168.1.44 托管主机

cache 192.168.1.45 托管主机

2) Yum安装

--把软件拷贝到(真机)ftp共享目录中

--更新索引文件

# createrepo --update /var/ftp/ansible

3) 安装及验证

--在ansible管理主机上配置yum配置文件

--安装 yum install ansible

--验证 ansible --version

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

3.1.1 主机定义与分组

1) ansible配置文件查找顺序

--首先检测\_ANSIBLE\_CONFIG变量定义的配置文件

--其次检查当前目录下的./ansible.cfg文件

--再次检查当前用户家目录下~/ansible.cfg文件

--最后检查/etc/ansible/ansible.cfg文件

2) /etc/ansible/ansible.cfg是ansible的默认配置文件路径

3) ansible.cfg 配置文件

--inventory定义托管主机地址配置文件路径名

--inventory指定的配置文件,写入远程主机的地址

# vim /etc/ansible/ansible.cfg

14 inventory = /etc/ansible/hosts //定义主机集合

61 host\_key\_checking = False //不检测

4) 配置集合

# vim /etc/ansible/hosts

末行

[web] [db] //集合

web1 db[1:2] //元素 1:20 表示db1-db20

web2

# vim /etc/hosts //添加主机名解析

192.168.1.40 ansible

192.168.1.41 web1

192.168.1.42 web2

192.168.1.43 db1

192.168.1.44 db2

192.168.1.45 cache

5) ansible命令基础

①. ansible 主机集合 -m 模块名称 -a 模块参数

--主机集合 主机名或分组名,多使用逗号分隔

-- -m 模块名称, 默认 command 模块

-- -a or --args 模块参数

②. 其他参数

-- -i inventry文件路径,或可执行脚本

-- -k 使用交互式登录密码

-- -e 定义变量

-- -v 显示详细信息

6) ansible 命令基础

①. 列出要执行的主机

# ansible all --list-hosts

# ansible web,db --list-hosts

②.批量检测主机

# ansible web -m ping -k //-m 模块 -k 交互式

//连接后会缓存连接信息, 缓存在 /root/.ansible/cp/

3.1.2 部署证书文件 (key)

1) 创建一堆秘钥

# cd /root/.ssh

# ssh-keygen -t rsa -b 2048 -N '' -f key

# ssh-copy-id -i key.pub 主机名

eg: ssh -i /root/.ssh/key web1 //-i指定私钥位置,对应公钥

也可以:

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

# ssh-copy-id web1 ...

2) inventory扩展参数

\*incentory参数说明

--ansible\_ssh\_port

ssh端口号:如果不是默认的端口号,通过此变量设置

--ansible\_ssh\_user

默认的ssh用户名

--ansible\_ssh\_pass

ssh密码

--ansible\_ssh\_private\_key\_file

ssh使用的私钥文件,适用于多个秘钥,而你不想使用SSH代理的情况

# vim /etc/ansible/hosts

[db]

db1

db2 ansible\_ssh\_port=222 //修改连接db2时的ssh端口

[web/all:vars] //给所有web组的主机配置变量

ansible\_ssh\_private\_key\_file="/root/.ssh/key" //指定私钥文件

[app:childen] //子组定义,后面加组名. 一定要写children.

web

db

3.1.3 自定义配置文件

# cd /var

# mkdir ooxx && cd ooxx

# touch ansible.cfg

[defaults]

inventory = myhost //定义主机集合

host\_key\_checking = False //不检测

# vim myhost

[app1]

web1

web2

[app2]

web1

db1

# ansible app1

# ansible all --list-hosts

# ansible web -m ping -k //-m 模块 -k 交互式

3..1.4 动态主机---调用语言

\*无线可能

--ansible Inventory包含静态和动态的Inventory,静态Inventory只在文件/etc/

ansible/hosts中指定的主机和组,动态Inventory指通过外部脚本获取主机列表,按照其

要求格式返回给ansible命令

\*Json

--Json(JavaScript Object Notation, JavaScript对象表示法), 一种基于文本独立

于语言的轻量级数据交换格式

--定义个人主机集合文件

# vim my.py

import json

hosts = {} //字典

hosts["web"] = ["web1","web2"]

hosts["db"] = ["db1","db2"]

hosts["other"]=["cache"]

print json.dumps(hosts)

# chmod 777 my.py

# ansible web list-hosts [-i my/py] //在自定义配置文件中修改 hosts=my.py

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

模块

4.1.1 ansible-doc和ping模块

1) ansible-doc

--模块的手册相当与shell的man,很重要

--ansible-doc -l //列出所有模块

--ansible-doc modulename //查看帮助

2) ping模块

--测试网络连通性,ping模块没有参数

--注:测试ssh的连通性,与能否ping通没有关系

# ansible web -m ping

3) command模块(默认模块)

--默认模块,远程执行命令

# ansible web -m command -a '[args]'

--查看所有机器负载

# ansible all -m command -a 'uptime' //-a + 命令

--查看日期和时间

# ansible all -m command -a 'date +%F\_%T'

command模块注意事项:

--该模块通过-a跟上要执行的命令可以直接执行,若命令里有如下字符则执行不成功

"<", ">","|","&"

--command模块不能解析系统变量

--该模块不启动shell直接在ssh进程中执行,所有使用到shell的命令执行都会失败

4) shell模块

--shell 模块用法基本和command一样,区别是shell模块是通过/bin/bash进行执行命令,可以执行任意命令

--不能执行交互式的命令,列如 vim top等

# ansible db -m shell -a 'cat /etc/passwd | grep bash'

\* shell 执行一下命令查看结果

# ansible web -m shell -a "echo ${HOSTNAME}"

ansible

注: //1.bash解析,进行变量替换,2.bash调用ansible执行

# ansible web -m shell -a 'echo ${HOSTNAME}'

web1

注: //1. bash变量替换, ''不进行变量替换 2. bash调用ansible执行 3.ansible 吃掉 ' ',

4. web上bash进行变量替换, 5.最后输出web1,web2

案例5: 创建用户

\*添加用户

--给web1,db2添加用户nb

--设置nb的密码为123

# ansible web1 db1 -m shell -a 'useradd nb && echo 123 | passwd --stdin nb'

\*思考:添加用户

--给所有web主机添加用户wk

--要求nb用户与wk用户不能出现在同一台主机上

--设置wk用户密码为456

# ansible web -m shell -a 'grep nb /etc/passwd || useradd wk ;echo 456 | passwd --stdin wk'

5) script模块

--在本地写脚本,然后使用script模块批量执行

# ansible web -m script -a '/root/name.sh'

--注意:该脚本包含但不限于shell脚本,只要指定Sha-bang解释器的脚本都可以运行

6) yum模块

--使用yum包管理器来管理软件包

--name:要进行操作的软件包名字

--state:动作(installed,removed)

eg: 给所有db主机安装mariadb

# ansible db -m yum -a 'name="mariadb-server" state=installed'

eg: cache主机删除 lrzsz

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

7) service模块

\*service模块

--name:必选项,服务名称

--enabled:是否开机启动 yes | no

--sleep:执行restarted,会在stop和start之间沉睡几秒钟

--state: 对当前服务执行启动,停止,重启,重新加载等操作

(started,stopped,restarted,reloaded)

eg: 开启mariadb服务

# ansible db1 -m service -a 'name=mariadb enabled=yes state=started'

8) copy模块 --一般用于管理配置文件

--复制文件到远程主机

--src: 复制本地文件到远程主机,绝对路径和相对路径都可,路径为目录时会递归复制.若路径

以"/"结尾,只复制目录里的内容,若不以"/"结尾,则复制包含目录在内的整个内容,类似

于rsync

--dest:必选项.远程主机的绝对路径,如果源文件是一个目录,那该路径必须是目录

--backup: 覆盖前先备份原文件,备份文件包含时间信息.有连个选项:yes | no

--force: 若目标主机包含该文件,但内容不同,如果设置为yes,则强制覆盖,设为no,则只有当

目标主机的目标位置不存在该文件时才复制.默认为yes.

eg: 复制文件,批量修改dns

# ansible all -m copy -a 'src=/etc/resolv.conf dest=/etc/resolv.conf'

eg: 复制目录

# ansible all -m copy -a 'src=//etc/yum.repos.d/ dest=/etc/yum.repos.d/'

案例7: 批量修改配置文件

--批量修改所有机器的dns配置 /etc/resolv.conf

--批量同步所有及其的yum配置文件

--给所有db主机开启binlog日志

log\_bin=mysql-bin

binlog-format="mixed"

# scp db1:/etc/my.cnf .

# ansible db -m copy -a 'src=my.cnf dest=/etc/my.cnf'

# ansible db -m service -a 'name=mariadb state=restarted'

9) lineinfile 模块

--类似sed的一种行编辑替换模块

--path 目标文件

--regexp 正则表达式,要修改的行

--line 最终修改的结果

eg: 修改my.cnf,中bin-log的格式mixed --> row //混合模式改为行模式

# ansible db -m lineinfile -a '

path=/etc/my.cnf //需要修改的行

regexp="^binlog-format"

line="binlog-format = row"'

10) replace模块

--类似sed的一种行编辑替换模块

--path目的文件

--regexp正则表达式

--replace 替换后的结果 //需要修改的地方

eg: 替换指定字符 row --> mixed

# ansible db -m replace -a '

path=/etc/my.cnf

regexp=""row"$"

replace=""mixed""'

10) setup模块

--主要用于获取主机信息,playbooks里经常会用的另一个参数gather\_facts与该米快相关,setup模块下经常用的是filter参数

eg:

# ansible cache -m setup -a 'filter=ansible\_distribution'

cache | SUCCESS => {

"ansible\_facts": {

"ansible\_hostname": "cache"

},

"changed": false

}

################################################

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ansible playbook基础

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1.1.0 ansible七种武器

JSON简介

YANML简介

Jinja2模板简介

1.1.1 ansible七种武器

\* 1) ansible命令,用于执行临时性工作,必须掌握

\* 2) ansble-doc是]ansible模块的文档说明,针对每个模块都有详细的说明及应用案例介绍,

功能和linux系统man命令类似,必须掌握

3) ansible-console是ansible为用户提供的交互式工具,用户可以在ansible-console

虚拟出来的终端上像shell一样使用ansible内置的各种命令.这为习惯使用shell交互方

式的用户提供了良好的使用体验

4)ansible-galaxy从github上下载管理Roles的一款工具,与python的pip类似.

\* 5) ansible-playbook是日常应用中使用频率最高的命令,工作机制;通过读取先编写好的

playbook文件实现批量管理,可以理解为按一定条件组成的ansible任务集,必须掌握.

6) ansible-vault主要用于配置文件加密,如编写的playbook文件中包含敏感信息,不想其

他人随意查看,可用它加密/解密这个文件

7) ansible-pull

--ansible有两种工作模式pull/push,默认使用push模式工作,pull和push工作模式机制刚

好相反

--使用场景,有大批量机器需要配置,即使使用高并发线程依旧要花费很多时间.

--通常在配置大批量机器的场景下使用,灵活性稍有欠缺,但效率几乎可以无线提升,对运维人员

的技术水平和前瞻性规划有较高要求.

1.2.0 JSON简介

1.2.1 JSON

--JSON是JavaScript对象表示法,它是一种基于文本独立于语言的轻量级数据交换格式

--JSON中的分隔符限于单引号 ' ,小括号(),中括号 [] ,大括号 {} ,冒号: 逗号,

数据结构:

数组: [1,3,4,5,6]

键值对: {key:value}, {k1:v1},{k2:v2,k3,v3}}

JSON中数据结构可以相互嵌套

2)JSON特性

--JSON是纯文本

--JSON具有"自我描述性"(人类可读)

--JSON具有层级结构(值中存在值)

--JSON可通过JavaScript进行解析

eg:key:[[1,2,k1:v1],k2:v2,k3:[1,2,3]]

3) JSON 语法规则

--数据在名称/值对中

--数据由逗号分隔

--大括号保存对象

--中括号保存数据

--小括号保存分组

4) JSON数据的书写格式是:名称/值对

--名称/值对包括字段名称(在双引号中),后面写一个冒号,然后是值:

eg: "诗仙": "李白"

JSON语法规则之数组:

{ "诗人":

["李白","杜甫","白居易","李贺"]

}

复合复杂类型:

eg:{ "诗人":

[{"李白":"诗仙","年代":"唐"},

{"李杜甫":"诗圣","年代":"唐"},

{"白居易":"使魔","年代":"唐"},

{"李贺":"诗鬼","年代":"唐"},

]

}

1.3.0 YAML简介

--是一个可读性高,用来表达数据序列的格式

--YAML (YAML Ain't Markup Language)

--YAML参考了多种语言,如:C语言, Python,Perl等,并从XML,电子邮件的数据格式中获得灵感

感,Clack Evans在2011年首次发表了这种语言,目前已有数种变成语言或脚本语言支持这

种语言

1.3.1 YAML 基础语法

--YAML的结构通过空格来展示

--数组使用 "- " 来表示. 必须有空格

--键值对使用": "来表示

--YAML使用一个固定的缩进风格表示数据层级结构关系

--一般每个缩进级别由两个以上空格组成

--#表示注释

2) 注意:

--不要使用tab,缩进是初学者容易出错的地方之一

--同一层级缩进必须对齐

--尽量使用vim 去编辑

3) YAML的键值表示方法

--采用冒号分隔

--: 后面必须有一个空格

--YAML键值对例子

格式:

eg: "诗仙": "李白"

eg: "诗仙":

"李白" 两个空格

4) 复杂YAML的键值嵌套:

"诗人":

"李白": "诗仙"

或

"诗人":

"李白":

"诗仙"

5) YAML数组表示方法

--使用一个短横杠加一个空格

--YANL数组列子

- "李白"

- "杜甫"

- "白居易"

- "李贺"

--哈希数组复合表达式:

"诗人":

- "李白"

- "杜甫"

- "白居易"

- "李贺"

6) 高级符合表达式

"诗人":

- //空格,空格 - (表示数组)

"李白": "诗仙" //键值,没有逗号,

"出生": "唐"

-

"李贺": "诗鬼"

"出生": "唐"

... ....

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1.4.0 Jinja2模板

1.4.1 Jinja2模板基本语法

--模板的表达式都是包含在分隔符"{{ }}"内的

--控制语句都是包含在分隔符"{% %}"内的

--模板支持注释,都是包含在分隔符"{# #}"内,支持快注释

--调用变量

{{varname}}

--计算

{{2+3}}

--判断

{{1 in [1,2,3]}}

1.4.2 Jinja2模板控制语句

{% if name == '诗仙' %}

李白

{% elif name == '诗圣' %}

杜甫

{% elif name == '使魔' %}

白居易

{% else %}

李贺

{% endif %}

2) Jinja2模板控制语句2

{% if name == ... ... %}

.. ..

{% elif name == '于谦' %}

{% for method in [aa,bb,cc] %}

{{do method}}

{% endfor %}

.. ..

{% endif%}

1.4.3 Jinja2过滤器

--变量可以通过过滤器修改,过滤器与变量用管道符号(|)分隔,也可以用圆括号传递可选参数,多

　个过滤器可以链式调用,前一个过滤器的输出会被作为后一个过滤器的输入. 与 shell的｜

　用法一样

ｅｇ：

--加密一个字符串：{{'astr'|password\_hash('sha512')}}

--过滤器在线文档: http://docs.jinkan.org/docs/jinja2/templates.html

#builtin-filters

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1.5.0 playbook

1.5.1 playbook是什么

1) playbook是ansible用于配置,部署和管理托管主机基本,通过playbook的详细描述,执行

其中的一系列tasks,可以让远端主机达到预期状态.

2) 在ansible中由计算机进行安装,部署应用,提供对外服务,以及组织计算机处理各种各样的事

情

3)playbook可以反复使用编写的代码,可以当道不同的机器上面,像函数一样,最大化的利用代

码,在使用ansible的过程中,处理的大部分操作都是在编写playbook.

1.5.2 playbook语法格式

--playbook由YAML语言编写,遵循YAML标准

--在同一行中,#之后的内容表示注释

--同一列表中的元素应该保持相同的缩进

--playbook由一个或多个play组成

--play中hosts,variables,roles,tasks等对象的表示方法都是键值中间": "分隔表示

--YAML还有一个小的怪癖,它的文件行都应该是 ---,这是YAML格式的一部分,表明一个文件的开

始..

1.5.3 playbook构成

--hosts: 定义将要执行olaybook的远程主机组

--vars: 定义playbook运行时需要使用的变量

--tasks: 定义将要在远程主机上执行的任务列表

--handlers: 定义task执行完成以后需要调用的任务

1.5.4 playbook执行结果

--使用ansible-playbook运行playbook文件,输出内容为JSON格式,由不同颜色组成便于识

别.

--绿色代表执行成功

--黄色代表系统状态发生改变

--红色代表执行失败 #只看红色

1.5.5 编写playbook

# vim ping.yml

---

- hosts: all #主机集合

remote\_user: root #远程执行的用户

task: #命令集合

- ping: #多个命令,放入数组中, 模块名-m:参数-a. -前空两格

# ansible-playbook ping.yml -f 5 #执行playbook

-- -f并发进程数量,默认是5. 一般为cpu的2-4倍

--hosts行 内容是一个(多个)组或主机的patterns,以逗号为分隔符

--remote\_user 账户名

--task

命令的集合

每一个play包含了一个task列表(任务列表)

一个task在其所对应的所有主机上()

单个主机之间是顺序执行,且在上一个命令执行成功后,顺序执行

--hosts

主机的集合

定义要执行任务的主机

1.5.6 playbook编写 2

\* playbook执行命令

--给web主机添加用户z3

--设置默认密码123,并把z3用户添加到users组

eg:

# ansible-doc -l | grep user #找到user模块

# ansible-doc user #查看文档

# ansible-doc shell

# vim

---

hosts: web

remote\_user: root

tasks:

- user:

name: z3

group: users

- name: passwd #所有tasks下一级的name都是注释说明

shell: echo 123|passwd --stdin z3

1.5.7 playbook 练习

--所有web主机安装Apache

--修改配置文件的监听端口为8080

--设置默认主页 hello world

--启动服务

--设置开机自启

yum, lineinfile, copy,service

# ansible-doc \*

# vim http.yml

---

- hosts: web

remote\_user: root

tasks:

- name: http install

yum:

name: httpd.x86\_64

state: latest

- lineinfile:

path: /etc/httpd/conf/httpd.conf

regexp: '^Listen 80'

line: 'Listen 8080'

- copy:

src: /root/index.html #原地址

dest: /var/www/html/index.html #目标地址

owner: apache

group: apache

mode: 0644

- service:

name: httpd

state: restarted

enabled: yes

# ansible-playbook http.yml -f 5

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2.1.0 playbook进阶

2.1.1 语法进阶

1) 变量

\*添加用户

--给web主机添加用户l4,设置默认密码123

# vim

---

hosts: web

remote\_user: root

vars: //设置变量,方便运维维护

username:l4

tasks:

- name: adduser "{{username}}"

user:

name: "{{username}}"

group: users

password: {{'123'|password\_hash('SHA512')}}

#密码使用SHA512加密,在/etc/login.defs内可以查找到,

# - name: passwd #所有tasks下一级的name都是注释说明# shell: echo 123|passwd --stdin "{{l4}}"

2) 设密码

--解决密码明文问题

--经测试发现,password是把字符串直接写入shadow,Linux的shadow密码是经过加密的,不能用

\*解决方法:

--变量过滤器password\_hash

{{'123'|password\_hash('sha512')}}

3) 变量参数 赋值

# vim user01.yml

---

- hosts: web

remote\_user: root

# vars:

# username: 5

tasks:

- name: adduser "{{username}}" #已改成调用文件参数

user:

name: "{{user.name}}"

group: "{{user.group}}"

password: "{{'user.pwd'|password\_hash('sha512')}}"

# ansible-playbook user01.yml -e'{"username": nb}' #JSON格式,变量外置

YAML格式

# vim args.yaml #创建YAML的文件内容

# ansible-playbook user01.yml -e @args.yaml

4) 多参数

# vim arg.yaml

---

username:

name: plj

group:users

pwd: 456

# vim user1.yaml

... ...

name: "{{"uasername.name": nb}}"

# ansible-playbook user01.yml -e @arg.yaml # 将上述参数设定

5) ERROR

\* ansible-playbook对错误处理

--默认情况判断$?.如果值不为0就停止执行

--但某些情况我们需要忽略错误继续执行

--例如 创建缓存目录,重启apche

\* 错误处理方法

--ignorw\_errors: 对错误的处理方式

--True表示忽略错误继续执行

--False表示遇到错误就停止执行

--默认 False

---

- hosts: web

remote\_user: root

tasks:

- shell: mkdir /temp/cache

ignore\_error: True # 忽略错误继续运行

- copy:

src: httpd.conf #原地址

dest: /etc/httpdcpmf #目标地址

owner: root

group: root

mode: 0644

- service:

name: httpd

state: restarted

enabled: yes

6) tags 标签,只运行一部分

--tags: 给指定的任务定义一个调用标识

--使用格式:

--name: NAME

--module: arguments

--tags: TAG\_ID

-- playbook调用方式

# ansible-playbook test.yml -t TAG\_ID

---

- hosts: web

remote\_user: root

tasks:

- shell: mkdir /temp/cache

ignore\_error: True # 忽略错误继续运行

- copy:

src: httpd.conf #原地址

dest: /etc/httpdcpmf #目标地址

owner: root

group: root

mode: 0644

tag: webconf

handlers: #触发器要与tasks对齐.

- name: reload webserver #触发器名字. (空两格)

service:

name: httpd

state: restarted

enabled: yes

# ansible-p

7) handlers

\* 安装Apache,修改httpd配置文件,重新载入配置文件让服务生效

---

- hosts: web

remote\_user: root

tasks:

- shell: mkdir /tmp/cache

ignore\_errors: True

- copy:

src: httpd.conf

dest: /etc/httpd/conf/httpd.cnf

owner: root

group: root

mode: 0644

tags: webconf #tag标签与copy对齐,触发条件copy成功

notify:

- reload webserver

handlers: #与tasks对齐

- name: reload webserver #name必须与tag中一致

service:

name: httpd

state: restarted

enabled: yes

# ansible-playbook http.yml -t webconf #-t tag

8) when

\* 有些时候需要在满足特定的条件后再触发某一项操作,或在特定的条件下终止某个行为,这个时候需要进行条件判断,when正是解决这个问题的最佳选择,远程中的系统变量facts作为when的条件,可以通过setup模块查看

eg:

9) register

\* 有时候我们还需哟啊更复杂的例子,如判断前一个命令的执行结果去处理后面的操作,这时候就需要register模块来保存前一个命令的返回状态,在后面进行调用

eg: 判断web服务器的系统负载,当web负载大于0.7时,关闭web服务器

# uptime | awk '{printf("%.2f",$(NF-2))}'

//%.2f打印浮点,保留两位小数,printf可以输出转化字符类型

#

---

hosts: web

remote\_user: root

task:

- name: system liadavgs

shell: uptime | awk '{printf("%.2f",$(NF-2))}'

register: result #上面执行结果保存在result中

- service:

name: httpd

state: stopped

when: result.stdout|float > 0.7 #result.stdout 标准输出,转化为浮点

# 当web服务器负载超过0.7,关闭web服务

# ansible-playbook load.yml

web1:

# awk 'BEGIN{while(1){}}' & # awk死循环

10) with\_items 循环

\* with\_items是playbook标准循环,可以用于迭代一个列表或字典,通过{{item}}获取 迭代的值.

eg:给所有讲师添加用户,设置不同的密码,并放置不同的组中

---

hosts: db

remote\_user: root

task:

- name: adduser "{{item.name}}"

user:

name: "{{item.name}}"

group: "{{item.group}}"

password: "{{item.pwd|password\_hash('sha512')}}"

with\_items:

-

name: nb

group: nobody

pwd: "123456"

-

name: wk

group: users

pwd: "654321"

-

name: plj

group: mysql

pwd: 123qqq...A

11) include and roles

\* playbook文件越来越大,这时把一些play,task或handler放到其他文件中,通过include指令包含进来是一个不错的选择.

eg:

task:

- include: tasks/setup.yml

- include: task/users.yml user=plj #users.yml中可以通过{{user}}来使用这些变量

handlers:

- include: hangdlers/handlers.yml

\* 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

\* 假如有一个play包含了一个叫"x"的role,则:

---

- hosts: host\_group

roles:

- x

-- x/tasks/main.yml

-- x/vars/main.yml

-- x/handler/main.yml

-- x/... .../main.yml

-- 都会自动添加进这个play

12) debug 调试

\* 拍错方法

--检测语法:

# ansible-playbook --suyntax-check playbook.yaml

--测试运行

# ansible-playbook -C playbook.yaml # 不太准

--显示受到影响的主机 --list-hosts

--显示工作的task --list-tasks #每次检查

--显示将要运行的tag --list-tags

\* result: 既有数组,又有键值对, 是一个对象

- debug: var=result

//将result对象打印出来,与when,service等同列.独立模块

################################################

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1.1.0 ELK

\*ELK分别代表:

--Elasticsearch:负责日志检索和储存

--Logstash:负责日志的收集和分析,处理

--Kibana:负责日志的可视化

\*这三款软件都是开源软件,通常配合使用,而且又先后归于Elastic.co公司名下,故称ELK.

1.1.1 ELK能做什么

\*ELK组件在海量日志系统的运维中,可用于解决:

--分布式日志数据集中式查询和管理

--系统监控,包含系统硬件和应用各个组件的监控

--故障排查

(企业内带宽涨了100M) 1.查交换机流量,查子接口,

-->web集群出口流量涨-->并发量qps,总访问量-->查每一台web出口带宽-->ELK查询各个接口带宽排序,url首页-->单网页跟踪-->首页图片没有压缩处理

--安全信息和事件管理

--报表功能

1.1.2 ELK

--E: Elasticsearch 非关系型数据库, 类似mysql

--L: Logstash 处理日志数据,类似php

--K: Kibana web服务器,提供特定页面展示, 类似nginx

1.2.0 Elasticsearch

\* Elasticsearch是一个基于Lucene的搜索服务器.它提供了一个分布式多用户能力的全文搜索引擎,基于RESful API的Web接口.

\* Elasticsearch是用Java开发的,并作为Apache许可条框下的开放源码发布,是当前流行的企业级搜索引擎.设计用于云计算中,能够达到实时搜索,稳定,可靠,快速,安装使用方便

1.2.1Elasticsearch 特点

-- 实时分析

-- 分布式实时文件存储,并将每一个字段都编入索引

--文档导向,所有的对象全都是文档

--高可用性,易扩展,支持集群(Cluster),[分片和复制(Shards和Replicas)

--接口友好,支持JSON

2) ES没有什么

--Elasticsearch没有典型意义的事务

--Elasticsearch是一种面向文档的数据库

--Elasticsearch没有提供授权和认证特性

3)相关概念

--Node:装有一个ES服务器的节点

--CLuster:有多个Node组成的集群

--Document:一个可被搜索的基础信息单元

--Index:拥有相似特征的文档的集合

--Type: 一个索引中可以定义一种或多种类型

--Filed: 是ES的最小单位,相当于数据的某一列

--Shards: 索引的分片,每一个分片就是一个SHard

--Replicas: 索引的拷贝

1.2.2 SQL与NoSQL 对比

--在ES中文档归属于一种类型(type).而这些类型存在于索引(index)中,类比传统关系型数据库

--DB-> Databases-> Tables-> Rows-> Columns

- 关系型 数据库 表 行 列

--ES-> Indices-> Types->Documents->Fields

- ES 索引 类型 文档 域

1.3.0 ES集群安装

\*安装第一台ES服务器

--设置主机名称和IP对应关系

--解决依赖关系

--安装软件包

--修改配置文件

--启动服务

--检查服务

1.3.1 ES集群安装

1) 更改主机名,配置IP,搭建 elk下的rpm yum源.

# vim /etc/hosts #修改hosts文件

es1 192.168.1.51

es2 192.168.1.52

... ...

# vim /etc/yum.repos.d/local.repo

2) 部署 Elasticsearch 第一个节点 // 所有节点都要做

# vim /etc/hosts

192.168.1.51 es1

192.168.1.52 es2

192.168.1.53 es3

192.168.1.54 es4

192.168.1.55 es5

# yum -y install java-1.8.0

# java -version

# yum -y install elasticsearch

# vim /etc/elasticsearch/elasticsearch.yml

17 cluster.name: my-application

23 node.name: es1

54 network.host: 0.0.0.0 #监听所有地址

68 discovery.zen.ping.unicast.hosts: ["es1", "es2","es3"]

#声明集群里的主机成员,不需要全部写进去 [集群发现相关]

# systemctl restart elasticsearch

# systemctl enable elasticsrarch 启动elasticsearch

# ss -auntpl | grep 9200

# ss -auntpl | grep 9300 #查看端口

# firefox http://192.168.1.51:9200 # 查看是否安装成功

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2.1.0 ES插件使用

192.168.1.254:

# firefox http://192.168.1.51:9200/\_cluster/health?pretty #查看集群状态

2.1.1 head插件安装

1) 将/linux/04/elk/\*.zip 放到/var/ftp/elk下

# scp /linux/04/elk/\*.zip /var/ftp/elk/

1) bigdesk, head, kopf 插件的安装

# cd /usr/share/elasticsearch/bin #zip拷贝到本地安装.方法1

# ./plugin install file:///usr/share/elasticsearch/bin/bigdesk

# /usr/share/elasticsearch/bin/plugin install ftp://192.168.1.254/elk/elasticsearch-head-master.zip

# /usr/share/elasticsearch/bin/plugin install ftp://192.168.1.254/elk/elasticsearch-kopf-master.zip

# /usr/share/elasticsearch/bin/plugin install ftp://192.168.1.254/elk/bigdesk-master.zip # 从ftp上安装插件,一台安装即可

# ./plugin list # 列出插件列表

\* # firefox http://192.168.1.55:9200/\_plugin/head #访问head插件

# firefox http://192.168.1.55:9200/\_plugin/kopt #访问kopt插件

# firefox http://192.168.1.55:9200/\_plugin/bigdesk #访问bigdesk插件

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HTTP协议简介

3.1.1

1) http请求由三部分组成

\*分别是: 请求行,消息报头,请求正文

\*请求行以一个方法符号开头,以空格分开,后面跟着请求的URI和协议版本,格式如下:

Method Request-URI HTTP-Version CRLF

2) http请求方法

--常用方法: GET, POST, HEAD

--其他方法 OPTIONS,PUT,DELETE,TRACE和CONNECT

--ES 常用

-PUT --增

-DELETE --删

-POST --改

-GET --查

3.1.2 系统命令curl

1) 在linux中curl是一个利用URL规则在命令行下工作的文件传输工具,是一款很强大的http命令工具.它支持多种请求模式,自定义请求等强大功能,是一款综合工具.

2) curl 常用参数介绍

-- -A 修改请求 agent

-- -X 设置请求方法

-- -i 显示返回头信息

eg: curl -A "Windows /IE9.0" http://118.144.89.240/info.php #修改请求头信息

eg: curl -XPUT http://118.144.89.240/info.php #指定访问方式,一定要大写

eg: curl -i http://118.144.89.240/info.php #可以探测到对方使用的服务器

3.1.3 RESTful API调用

\* Elasticsearch提供了一系列RESTFUL的API

--检查集群,节点,索引的健康度,状态和统计

--管理集群,节点,索引的数据及元数据

3.1.4 API调用

- \_CAT api查询集群状态,节点信息

- v参数显示详细信息

# curl -XGET HTTP://192.168.1.51:9200/\_cat/shards/ #查看集群状态,节点

# curl -XGET HTTP://192.168.1.51:9200/\_cat/health?v # 查看集群成员信息

# curl -XGET HTTP://192.168.1.51:9200/\_cat/health?help #显示帮助信息

- nodes 查询节点状态信息

# curl -XGET HTTP://192.168.1.51:9200/\_cat/nodes

- 索引信息

# curl -XGET http://192.168.1.51:9200/\_cat/indices?v

2) RESTful API 增加

- 创建一个索引,并设置分片数与副本数量

curl -XPUT http://192.168.1.51:9200/tedu -d '

{ #创建一个tedu的索引

"settings":{

"index":{

number\_of\_shards: 5, #分片数量

numbeer\_of\_replicas: 1 # 副本数量

}

}

} '

3) RESTful API插入数据

- (增)加数据,使用PUT方法

- 调用方式: 数据库地址/索引/类型/id值

# curl -XPUT http://192.168.1.51:9200/tedu/taecher/1 -d '{

"姓名": "牛奔",

"性别": "男"

"年龄": "19"

}'

# curl -XPUT http://192.168.1.51:9200/tedu/taecher/2 -d '{

"姓名": "静静",

"性别": "女",

"年龄": "17",

"爱好": "撸猫"

}'

# curl -XPUT http://192.168.1.52:9200/tedu/teacher/3 -d '{

> "姓名": "紫",

> "性别": "老太婆",

> "年龄": "老太婆",

> "爱好": "傅丽灵梦"

> }'

# curl -XPUT http://192.168.1.51:9200/tedu/teacher/4 -d '{

"姓名": "总司",

"性别": "秀吉",

"年龄": "18",

"爱好": "病弱"

}'

4) POST修改

--修改数据,使用POST方法

--在修改数据的时候必须调用\_update 关键字

--调用方式: 数据库地址/索引/类型/id值/\_update

# curl -XPOST http://es4:9200/tedu/teacher/1/\_update -d '{

"doc": {"性别": "秀吉"}}'

5) 查询:

# curl -XGET http://192.168.1.51:9200/tedu/teacher/3 # 查询

# curl -XGET http://192.168.1.51:9200/tedu/teacher/3?pretty #终端显示

6)删除:(无法删类型)

# curl -XDELETE http://192.168.1.52:9200/tedu/teacher/2 #删除单条记录

# curl -XDELETE http://192.168.1.52:9200/tedu #删除整个库

# curl -XDELETE http://192.168.1.52:9200/\* #删根

3.1.5 批量导入数据

\*使用\_bulk;批量导入数据

--批量导入数据使用POST方式.数据格式为json,url编码使用data-binary

--导入含有index配置的json文件

# gzip -d logs.jsonl.gz #解压gz打包文件

# curl -XPOST http://192.168.1.52:9200/\_bulk --data-binary @logs.jsonl

# curl -XPOST http://192.168.1.52:9200/a/a/\_bulk --data-binary @accounts.json

3.1.6 查看apache日志文件

http://www.baidu.com

insert 金步国 -->金步国[作品集--->Apache 手册--->定制日志格式

-->systemd openssh openssl sshd rpm mount php.ini

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Kibana

ELK 虚拟机角色:

192.168.1.51 es1

192.168.1.52 es2

192.168.1.53 es3

192.168.1.54 es4

192.168.1.55 es5

192.168.1.56 kibana

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1.1.0 Kibana特点

1.1.1 kibana安装

kibana虚拟机: 192.168.1.56

--从es1 拷贝/etc/hosts, /etc/yum.repos.d/local.repo 文件

--使用rpm方式安装

# yum -y install kibana

--修改kibana配置文件: /opt/kibana/config/kibana.yml

# vim /opt/kibana/config/kibana.yml

2 server.port: 5601 # 端口号

5 server.host: "0.0.0.0" # 监听地址

15 elasticsearch.url: "http://es2:9200" #集群服务器地址

23 kibana.index: ".kibana" # 在Elasticsearch 中创建一个信息库

26 kibana.defaultAppId: "discover" # 默认主页

53 elasticsearch.pingTimeout: 1500

57 elasticsearch.requestTimeout: 30000

64 elasticsearch.startupTimeout: 5000

# systemctl start kibana

# netstat -auntpl | grep 5601

# fierfox 192.168.1.56:5601/status

在 Installed Plugins 栏,查看Status 都是ready状态 则表示安装成功

1.1.2 Kibana 访问页

Indices: 提示. 分析的数据在哪

Warning No default index pattern. You must select or create one to continue.

1) 配置分析日志:

Settings:

Index name or pattern

logstash-\* # 支持通配符 \*

Time-field name refresh fields # 横坐标 时间戳

@timestamp

create(绿)

2) Discover 柱状图

扩展时间范围--> 右上角Last 15 minutes-->Absolute (自定义)-->

From: 2015-05-15 09:28:30.764

To: 2015-05-22 09:43:30.764 # 包含导入日志时间

鼠标左键点击图标,拖拉选中有数据的横轴,放大图形,一遍查看

3) Visualize 饼图 .(只能分析以记录的数据)

--> Pie chart --> From a new search --> Split Slices--> Terms

--> geo.src(国家源地址) Size 7 (显示前7名) --> ▷ (显示图标)

添加统计操作系统的子查询:

--> Select buckets type --> Split Slices--> Terms --> machine,os--> ▷

统计ip访问量

--> Terms --> clientip

--> sub-buckets --> Terms --> extension

分析: 访问.php(动态页面)的极有可能是爬虫

--> sub-buckets --> Terms --> request

4) Dashboard 查看图表,可以查看多个已保存的图表

--> Dashboard --> + (右上角)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2.1.0 Logstash

虚拟机环境: 192.168.1.57 Logstash 4G内存,2CPU

192.168.1.58 apache

# virsh edit 57 # 扩展内存

<memory unit='KiB'>4188282</memory>

# virsh shutdown 57

# virsh start 57

2.1.1 Logstash

\* 是一个数据采集,加工处理以及传输的工具

\* 特点:

--所有类型的数据集中处理

--不同模式和格式数据的正常化

--自定义日志格式的迅速扩展

--为自定义数据源轻松添加插件

2.1.2 Logstash 安装

--Logstash依赖Java环境,需要安装java-1.8.0-openjdk

--Logstash没有默认的配置文件,需要手动配置

--Logstash安装在/opt/logstash目录下

# yum -y install logstash

2.1.3 Logstash 配置文件详解

1) 手动创建 logstash.conf 配置文件

# touch /etc/logstash/logstash.conf

2) Logstash工作结构

{数据源}==>

input{}==> # 负责获取日志数据

filter{}==> # 对数据加工处理

output{}==> # 数据放到logstash外部的数据源

{ES}

3) Logstash类型及条件判断

类型:

--布尔值类型: ssl\_enable => true

--字节类型: bytes => "1MiB"

--字符串类型: name => "xkops"

--数值类型: port => 22

--数组: match => ["a","b"]

--哈希: option => {k => "v",k2 => "v2"}

--编码解码

4) 配置文件编辑

\* 配置文件也分三部分配置:

# vim /etc/logstash/logstash.conf

input{ stdin{} } # 标准输入(键盘鼠标)

filter{} # 不对数据做处理

output{ stdout{} } # 标准输出

# /opt/logstash/bin/logstash -f /etc/logstash/logstash.conf #启动程序

Setting: Default pipeline workers: 2 # 2个线程,输入 输出

Pipeline main started

hana ha saiko dawa

2019-07-17T03:58:59.953Z logstash hana ha saiko dawa

5) Logstash插件

-插件有logstash-output, logstash-input, logstash-filter, logstash-codec

插件及文档地址: https://github.com/logstash-plugins

--> 点击一个插件 --> 点击 location

# /opt/logstash/bin/logstash-plugin --list #查看插件列表

\* cedec类插件

--常用的插件: plain,json,joson\_lines,rubydebug,multiline等

logstash-output-email # 以email的形式输出结果

logstash-output-file # 以文件的形式输出

logstash-input-file # 从文件中读取数据

logstash-filter-

logstash-codec-json # codec 编解码

6) 编写 logstash 配置文件

--将输出改为 rubydebug格式

vim /etc/logstash/logstash.conf

input{

stdin{}

}

filter{}

output{

stdout{ codec => "rubydebug"} # 以rubydebug格式输出

}

\* 7) file input plugin //未看

查看file 文档: 搜索 option 查看参数

Setting Input type Required

path array Yes # 看必须要的配置

#vim /etc/logstash/logstash.conf

input{

file {

path => ["/tmp/a.log","var/tmp/b.log"]

start\_position => "beginning" # begin读取所有文件,默认end,丢弃之前的文件(sincedb不存在时生效)

type => "testlog" # 给数据打标签,多个file 区分

sincedb\_path => "/var/lib/logstash/since.db" #指定指针文件

}

}

filter{}

output{

stdout{ codec => "rubydebug"}

}

# rm -rf /root/sincedb\_path

# /var/lib/logstash/since.db

# /opt/logstash/bin/logstash -f /etc/logstash/logstash.conf

# echo "A\_$(RANDOM)" >> /tmp/a.log # 另起终端插入数据

# echo "B\_$(RANDOM)" >> /var/tmp/b.log

8) filter grok插件练习:

# 将apache日志一行写入 /tmp/a.log 中

input{

file {

path => ["/tmp/a.log","var/tmp/b.log"]

start\_position => "beginning"

type => "testlog"

#sincedb\_path => "/var/lib/logstash/since.db"

sincedb\_path =>

}

}

filter {

grok {

patterns\_dir => ["./patterns"]

match => { "message" => "(?<ip>^[0-9.]+).+\[(?<time>.+)\] \"(?<method>[A-Z]+) (?<usrl>\S+) (?<proto>[A-Z]+)/(?<ver>\d\.\d\" (?<rc>\d+) (?<size>\d+) \"(?<ref>[^\']+)\" \"(?<agent>[^\"]+)\""}

# ?<name>reg 表示字段 .%{apche的正则表达式,文件匹配IP:client\_ip}宏

match => { "message" => "%{COMBINEDAPACHELOG}"} # 添加日志的宏

} # client\_ip}宏

}

output{

stdout{ codec => "rubydebug"}

}

# cd /opt/logstash/vendor/bundler/jurby/1.9/gems/logstash-patters-core-2.0.5/patterns

# vim grok=patterns #宏文件

9) output ES插件

output{

stdout{}

elasticsearch {

hosts => ["es1:9200","es2:9200","es3:9200"]

index => "weblog"

flush\_size => 2000 # 达到2000字节插入

idle\_flush\_time => 10 # 超过10 没有数据,则写入

}

}

2.2.1 部署web3服务器

在192.168.1.58 web3上安装apache,及 filebeate软件

# yum -y install httpd

# vim /etc/yum.repos.d/local.repo

# yum -y install filebeat

# vim /etc/filebeat/filebeat.yml

14 paths:

15 - /var/log/httpd/access.log

72 document\_type: apache\_log

183 # elasticsearch: # 注释掉

188 # hosts: ["localhost:9200"]

278 logstash:

280 hosts: ["192.168.1.37:5044"] #logstash主机ip

192.168.1.57 logstash:

# vim /etc/logstach/logstach.conf

input{

stdin{ codec => "json" }

beats{

port => 5044

}

file {

path => [ "/tmp/a.log", "/tmp/b.log" ]

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

start\_position => "beginning"

type => "testlog"

}

filter{

if [type] == "apachelog"{

grok{

match => ["message", "%{COMBINEDAPACHELOG}"]

}}

}

output{

stdout{ codec => "rubydebug" }

if [type] == "filelog"{

elasticsearch {

hosts => ["192.168.1.51:9200", "192.168.1.52:9200"]

index => "filelog"

flush\_size => 2000

idle\_flush\_time => 10

}}

}

插件及文档地址:

https://github.com/logstash-plugins

#################################################

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

1.1.0 大数据

1) 定义: 大数据指无法在一定时间范围内用常规软件工具捕捉,管理和处理的数据集合,需要新处理模式才能具有更强的决策力,洞察发现力和流程优化能力的海量,高增长率和多样化的信息资产

是指从各种各样类型的数据中,快速获得价值的信息

2) 大数据能做什么

--企业组织利用相关数据分析帮助他们降低成本,提高效率,开发新产品,做出更明智的业务决策

--把数据集合并后进行分析得出的信息和数据关系性,用来察觉商业趋势,判定研究质量,避免疾病扩散,打击犯罪或测定即时交通路况等

--大规模并行处理数据库,数据挖掘电网,分布式文件系统或数据库,云计算和可扩展的存储统计等.

1.1.1 大数据特性

1) 数据 TB级,记录/日志,事务,表&文件

2) 速度 实时,批处理,多进程,数据流

3) 种类 结构化,非结构化,多因素,概率性

4) 价值 统计学,事件性,相关性,假设性

5) 真实性 可信性,真伪性,来源&信誉,有效性,可审计性

1.1.2 大数据5V特性是:

-Volume (大体量)

可从数百TB到数十数百PB,甚至EB的规模

-Variety (多样性)

大数据包括各种格式和形态的数据

-Velocity (时效性)

很多大数据需要在一定的时间限度下得到即时处理

-Veracity (准确性)

处理的记过要保证一定的准确性

-Value (大价值)

大数据包含很多深度的价值,大户数据分析挖掘和利用将带来巨大的商业价值

1.1.3 大数据与Haddop

1) Hadoop是一种分析和处理海量数据的软件平台

Hadoop是一款开源软件,使用JAVA开发

Hadoop可以提供一个分布式基础架构

2) Hadoop特点

--高可靠性,高扩展性,高效性,高容错性,低成本

1.1.4 Hadoop起源

1) GFS --- HDFS

2) MapReduce---MapReduce

3) BigTable---Hbase

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2.1.0 Hadoop常用组件

\* HDFS: Hadoop分布式文件系统 (核心组件)

\* MapReduce: 分布式计算框架 (核心组件)

\* Yarm: 集群资源管理系统( 核心组件)

\* Zookeeper: 分布式协作服务

\* Hbase: 分布式列存数据库

\* Hive: 基于Hadoop的数据仓库

\* Sqoop: 数据同步工具

\* Pig: 基于Hadoop的数据流系统

\* Mahout: 数据挖掘算法库

\* Flume: 日志收集工具

2.1.1 HDFS: Hadoop分布式文件系统 图见word

1) Hadoop体系中数据存储管理的基础,是一个高度容错的系统,用于低成本的童工硬件上运行

2) 角色和概念

--Client

--Namenode

--Secondarynode

--Datanode

3) NameNode

--Master节点,管理HDFS的名称空间和数据快映射信息,配置副本策略,处理所有客户端请求

4) Secondary NameNode

--定期合并fsimage和fsedits,推送给NameNode

--紧急情况下,可辅助恢复NameNode, 存有的fsimage和fsedits 不是最新的

--并非NameNode的热备

fsedits: 数据变更日志, 补丁日志.用户读取数据时,取出元数据在安装补丁, 再给用户

fsimage: 数据块映射信息 存储信息

5) DataNode

--数据存储节点,存储实际的数据

--汇报存储信息给NameNode

6) Client

--切分文件 (128MB一块)

--访问HDFS

--与NameNode交互,获取文件位置信息

--与DataNOde交互,读取和写入数据

7) Block

--每块缺省128MB大小

--每块可以多个 副本

2.2.1 MapReduce结构

详见: 百度

2.3.1 Yarn 结构

详见:百度

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

3.1.0 Hadoop安装部署

3.1.1 Hadoop模式

1) 单机 --->学习

2) 伪分布式 -----开发,测试

3) 完全分布式 -----生成环境

3.1.2 单机模式

1) 获取软件:

http://hadoop.apache.org

2) 安装配置Java环境,安装jps工具

安装 Openjdk和Openjdk-devel

3)设置环境变量,启动运行

4) hadoop-env.sh

JAVA\_HOME="JAVA安装路径"

HADOOP\_CONF\_DIR="hadoop配置文件路径"

5) 配置虚拟机 ip 192.168.1.61 hostname: node1

# echo "192.168.1.61 node1" >> /etc/hosts

# yum install java-1.8.0-openjdk java-1.8.0-openjdk-devel

# jps

759 Jps //pid 角色名

6) 安装hadoop

# scp /linux-soft/04/hadoop/hadoop-2.7.7.tar.gz root@192.168.1.61:/root # 真机传hadoop软件到 61 虚拟机

# tar zxf hadoop-2.7.7.tar.gz

# mv hadoop-2.7.7 /usr/local/hadoopp

# /usr/local/hadoop/bin/hadoop

Error: JAVA\_HOME is not set and could not be found.

# rpm -ql java-1.8.0-openjdk # 查看java安装位置

7) 设置环境变量,启动运行

vim /usr/local/hadoop/etc/hadoop.env.sh

25 export JAVA\_HOME="/usr/lib/jvm/java-1.8.0-

openjdk-1.8.0.161-2.b14.el7.x86\_64 "

33 export HADOOP\_CONF\_DIR=${HADOOP\_CONF\_DIR:-"/usr/local/hadoop/etc/hadoop"}

3.1.3 单机模式安装简单,只需要配置好环境变量即可运行,这个模式一般用来学习和测试Hadoop的功能.

eg: 测试统计词频

# cd /usr/local/hadoop

# mkdir oo

# cp \*.txt oo

# ./bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-exampes-2.7.7.jar wordcount input output

# jar运行一个jar文件

# 对应的jar文件: share/hadoop/mapreduce/hadoop-mapreduce-exampes-2.7.7.jar

# 调用wordcount 参数 input 本地输入目录 output 本地输出目录

3.2.0 HDFS 分布式与伪分布式

1) Hadoop-env.sh

JAVA\_HOME

HADOOP\_CONF-DIR

2) xml文件配置格式

<property>

<name>g关键字</name>

<value>变量值</value>

<description>描述</description>

</property>

3) 系统规划

主机 角色 软件

192.168.1.60 NameNode HDFS

nn01 SecondaryNameNode

192.168.1.61

node1 DataNode HDFS

node2 DataNode HDFS

node3 DataNode HDFS

3.2.1 搭建完全分布式

1) 基础环境准备

--新开3台虚拟机

--禁用selinux

--禁用firewalld

# systemctl stop firewalld

# systemctl mask firewalld

2) 安装java-1.8.0-openjdk-devel

# yum -y install java-1.8.0-openjdk-devel

3) 基础环境准备

--3台机器上配置/etc/hosts

--注意所有主机都能ping通namenode的主机名,namenode能ping通所有节点

--java -version验证java安装

--jps 验证角色

4) 配置SSH信任关系(NameNode)

--注意:不能出现要求输入yes的情况,每台机器都要能登录成功,包括本机.

# vim /etc/ssh/ssh\_config

StrictHostKeyChecking no

# ssh-keygen

# ssh-copy-id 192.168.1.X

5) HDFS完全分布式系统配置

--环境配置文件: hadoop-env.sh

--核心配置文件: core-site.xml

--HDFS配置文件: hdfs-site.xml

--节点配置文件: slaves

6) 环境配置文件: hadoop-env.sh

# vim /usr/local/hadoop/etc/hadoop/hadoop-env.sh

25 export JAVA\_HOME="/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.161-2.b14.el7.x86\_64 "

33 export HADOOP\_CONF\_DIR=${HADOOP\_CONF\_DIR:-"/usr/local/hadoop/etc/hadoop"}

7) 核心配置文件 core-site.xml

--fs.defaultFS : 文件系统配置参数

--hadoop.tmp.dir : 数据目录配置参数

--

hadoop.apache.org/docs/ -->2.7.7 ---> 最底下condiguration -->core 查看当前文档

# vim /usr/local/hadoop/etc/core-site.xml

<property>

<name>fs.default.name</name>

<value>hdfs://nn01:9000</value> # 改的值,默认使用文件系统

<description>描述</description> # 注释说明, 不写也可以

</property>

<property>

<name>hadoop.tmp.dir</name>

<value>/var/hadoop</value> # 改的值,var下放数据

</property>

8) HDFS配置文件hdfs-site.xml

--Namenode: 地址声明

--Secondarynamenode: 地址声明

-- 文件冗余份数

#vim /usr/local/hadoop/etc/hadoop/hdfs-site.xml

<property>

<name>dfs.namenode.http-address</name> # 声明启动namenode服务

<value>nn01:50070</value>

</property>

<property>

<name>dfs.namenode.secondary.http-address</name>

<value>hdfs:nn01:50090</value>

</property>

<property>

<name>dfs.replication</name> #配置副本策略

<value>2</value>

</property>

9) 节点配置文件slaves

--只写DataNode节点的主机名称

# vim etc/hadoop/slaves

node1 node2 node3

--同步配置

node1 node2 node3

# rm -rf /usr/local/hadoop

# scp/ansible /usr/local/hadoop root@node1:/usr/local/hadoop

--Hadoop所有节点的配置参数完全一样

10) 格式化,启动HDFS集群

--在namenode创建 /var/hadoop 文件夹

# 扩容nn01磁盘

# mkdir /var/hadoop

--在namenode上执行格式化操作

# /usr/local/hadoop/bin/hdfs namenode -format

--启动集群

# ./sbin/start-dfs.sh

--验证角色

nn01 # jps

20066 NameNode

20249 SecondaryNameNode

20521 Jps

node1 # jps

19752 Jps

19679 DataNode

--节点/集群验证

# ./bin/hdfs dfsadmin -report

Live datanodes (3):

3.3.1 安装与部署

1) Hadoop三大核心组件

--HDFS : 分布式文件系统

--MapReduce: 分布式计算框架

--yarm: 集群资源管理

2) 系统规划

主机 角色 软件

192.168.1.60 NameNode HDFS

nn-1 SecondaryNameNode YARN

192.168.1.61 DataNode HDFS

node1 NodeManager YARN

192.168.1.62 DataNode HDFS

node2 NodeManager YARN

192.168.1.63 DataNode HDFS

node3 NodeManager YARN

3) 更改分布式计算框架 mapred配置模板, mapred-site.xml

--只支持local和yarn两种,单机使用local,集群使用yarn.

# vim etc/hadoop/mapred-site.xml

<property>

<name>mapreduce.framework.name</name>

<value>yarn</value> #由yarm管理mapred

</property>

4) yarn部署,

--资源管理yarn的配置文件: yarn-site.xml

# vim etc/hadoop/yarn-site.xml

<configuration> # 指定resourcemanager 地址

<property>

<name>yarn.resourcemanager.hostname</name>

<value>nn01</value>

</property>

<property> # 指定yarn的计算框架 (由研发提供)

<name>yarn.nodemanager.aux-services</name> #此处为yarn默认框架

<value>mapreduce\_shuffle</value>

</property>

</configuration>

# ./sbin/start-yarn.sh #启动yarn

# ./bin/yarn node -list #查看node列表,

5) web访问Hadoop

namende web页面

192.168.1.60:50070

secondary namenode web页面:

192.168.1.60:50090

resourcemanager web页面:

192.168.1.60:8088

datanode web页面:

192.168.1.60:50075

nodemanager web页面()node1,node2,node3

192.168.1.60:8042

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Hadoop 2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

firefox 192.168.1.60:50070

创建HDFS文件:

1) 扩容四台设备的磁盘.

2) HDFS基本命令使用:

# ./bin/hadoop fs -mkdir /abc #创建文件夹

# ./bin/hadoop fs -ls / #查看/ 下文件

# ./bin/hadoop fs -touchz /ooxx #创建文件ooxx

# ./bin/hadoop fs -put \*.txt /abc/ # 将本地磁盘下的txt文件,上传到/abc/中

# ./bin/hadoop fs -get /ooxx ./ # 下载, 虚拟磁盘路径, 本地磁盘路径

3) 案例:

1. 在集群文件系统中创建文件夹

2. 上传要分析的文件到目录中

3.分析上传文件

4. 展示分析结果

# ./bin/hadoop fs -mkdir /abc

# ./bin/hadoop fs -put \*.txt /abc/

# ./bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.7.jar wordcount /abc /dir

# ./bin/hadoop fs -cat /dir/part-r-00000

# /abc -完整路径-->hdfs://nn01:9000/abc, 为HDFS磁盘文件

# /dir 也是HDFS磁盘的路径

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2.2.0 Hadoop节点管理

2.2.1 增加节点, 集群扩容

--启动一个新的系统,设置SSH免密码登录

--在所有接电脑修改/etc/hosts,增加新节点的主机信息

--安装java运行环境(java-1.8.0-openjdk-devel)

--修改NameNode的slaves文件增加该节点

--拷贝NameNode的/usr/local/hadoop到本机

--在该节点启动DataNode

nn01

# ssh-copy-id -i

# echo "192.168.1.64 newnode" >> /etc/hosts

# rsync -av /etc/hosts root@newnode:/etc/hosts

newnode:

# yum -y install java-1.8.0-openjdk-devel

nn01

# cat etc/hadoop/slaves << newnode

# rsync -av /usr/local/hadoop root@newnode:/usr/local/hadoop

newnode:

# ./sbin/hadoop-daemon.sh start datanode

# jps # 查看角色

nn01:

# ./bin/hdfs dfsadmin -report #查看node成员

Live datanodes (4):

2.2.2 HDFS节点管理

--设置同步带宽,并同步数据

# ./bin/hdfs dfsadmin -setBalancerBandwidth 600 000 000 # 设置为500Mb/s

# ./sbin/start-balancer.sh #同步数据

--查看集群状态

# ./bin/hdfs dfsadmin -report

2.2.3 修复节点

\* HDFS修复节点

--修复节点与增加节点基本一致

--注意:新节点的ip和主机名要与损坏节点的一致

--启动服务:

# ./sbin/hadoop-daemon.sh start datanode

--数据恢复是自动的

--上线以后会自动恢复数据,如果数据量非常巨大,可能需要一定的时间

2.2.4 删除节点 eg: 删除newnode

\*HDFS删除节点

-- 添加删除节点文件 .不需要同步数据

# vim hdfs.hosts.exclude配置

>dfs.hosts.exclude<

>/usr/local/hadoop/etc/hadoop/exclude< # 存放文件位置

--增加exclude配置文件.写入要删除的节点主机名

# vim etc/hadoop/exclude << newnode

--数据迁移

# ./bin/hdfs dfsadmin -refreshNodes

--查看状态

# ./bin/hdfs dfsadmin -report

status: Normal

status: Decommission in progress # 迁移数据中 ! 等....!

status: Decommissioned # 迁移完毕.(拷贝)

注: 仅当状态变成Decommissioned才能down机下线

2.3.1 yarn 节点管理

--增加节点

# sbin/yarn-daemon.sh start nodemanager

--删除节点

# sbin/yarn-daemon.sh stop nodemanager

--查看节点 (ResourceManager)

# ./bin/yarn node -list

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

3.1.0 NFS配置

3.1.1 NFS网关用途

--永不可以通过操作系统兼容的本地NFSv3客户端来浏览HDFS文件系统

--用户可以从HDFS文件系统下载文档到本地文件系统

--用户可以通过挂载点直接流化数据,支持文件附加,但是不支持随机写

--NFS网关支持NFSv3和允许HDFS作为客户端文件系统的一部分被挂载

3..1.2 配置用户

1) 配置代理用户

--在NameNode和NFSGW上添加代理用户

--代理用户的UID,GID,用户名必须完全相同

--如果因特殊原因客户端的用户和NFS网关的用户UID不能保持一致,需要我们配置nfs.map的静态映射关系

-- nfs.map

uid 10 100 # Map the remote UID 10 the local UID 100

gid 11 101 # Map the remote GID 11 the local GID 101

2) 分别nfsgw,namenode添加用户及组

nfsgw:

# groupadd -g 800 nfsuser # 添加组

# useradd -u800 -g800 -r -d /var/hadoop nfsuser # 创建用户

nn01:

# groupadd -g 800 nfsuser

# useradd -u800 -g800 -r -d /var/hadoop nfsuser

3.1.3 配置core-site.xml

1) 停止集群

nn01:

# ./sbin/stop-all.sh

2) 编辑core-site.xml配置文件:

--用户为主机上真实运行的nfs3的用户, 为上面添加用户 nfsuser

--在非安全模式下,运行nfs网关的用户为代理用户

--groups为挂载点用户所使用的组

--hosts为挂载点主机地址

# vim etc/hadoop/core-site.xml

... ...

<property> # 设置组授权

<name>hadoop.proxyuser.nfsuser.groups</name>

<value>\*</value>

</property>

<property> #设置主机授权

<name>hadoop.proxyuser.nfsuser.groups</name>

<value>\*</value>

</property>

3) 同步所有node的配置文件

# rsync -aXSH --delete /usr/local/hadoop/etc/ node1:/usr/local/hadoop/etc/

# rsync -aXSH --delete /usr/local/hadoop/etc/ node2:/usr/local/hadoop/etc/

# rsync -aXSH --delete /usr/local/hadoop/etc/ node3:/usr/local/hadoop/etc/

4) 启动hdfs

# vim etc/hadoop/slave #删除newnode的配置

# vim etc/hadoop/exclude #删除newnode的配置

# ./sbin/start-dfs.sh

# ./bin/hdfs dfsadmin # 查看几圈状态.

Live datanodes (3):

3.2.1 NFSGW配置 192.168.1.65 NFSGW 主机

1) 卸载rpcbind,nfs-utils

# rpm -qa | grep -i rpcbind

# rpm -qa | grep -i nfs-utils

2) 配置/etc/hosts,添加所有NameNode和DataNode的主机与ip对应关系

# rsync -av 192.168.1.60:/etc/hosts /etc/hosts

3) 安装java运行环境(java-1.8.0-openjdk-devel)

# yum -y install java-1.8.0-openjdk-devel

4) 同步NameNode的/usr/local/hadoop到本机

# rsync -aXSH nn01:/usr/local/hadoop /usr/local/

5) 配置文件hdfs-site.xml

--nfs.exports.allowed.hosts

默认情况下,export可以被任何客户端挂载. 为了更好的控制访问,可以设置属性. 值和字符串对应机器名和访问策略,通过空格来分隔.机器名的格式可以是单一的主机,Java的正则表达式或者IPV4地址.

使用rw或ro可以指定导出目录的读写 或只读权限.

如果访问策略没有被提供,默认为只读,每个条目使用 ; 来分隔

--nfs.dump.dir

用户 需要更新文件转储目录参数.NFS客户端经常重新安排写操作,顺序的写操作会随机到达NFS网关.这个目录常用于临时存储无序的写操作.对于每个文件,无序的写操作会在他们积累在内存中超过一定阈值(如: 1M)时被转储.需要确保有足够的空间的目录

如: 应用上传10个100M,那么这个转储目录推荐1GB左右空间,以便每个文件都发生最坏的情况,只有NFS网关需要在设置该属性后重启.

# vim etc/hadoop/hdfs-site.xml

<property> #设置授权主机

<name>nfs.exports.allowed.hosts</name>

<value>\* rw</value>

</property>

<property> #设置转储目录 #

<name>nfs.dump.dir</name>

<value>/var/nfstmp</value>

</property>

8) 创建文件夹, /var/nfstmp授权. /usr/local/hadoop/logs 授权

# mkdir /var/nfstmp

# chown nfsuser.nfsuser /var/nfstmp

6) NFS启动与挂载

1) 设置usr.local.hadoop/logs权限,为代理用户赋予读写执行权限

# setfacl -m user:nfsuser:rwx /usr/local/hadoop/logs

2) 使用root用户启动portmap服务

# ./sbin/hadoop-daemon.sh --script ./bin/hdfs start portmap

3) 使用代理用户启动nfs3 (之前设置了家目录为/var/hadoop)

# su -l nfsuser

# cd /usr/local/hadoop

# ./sbin/hadoop-daemon.sh --scripe ./bin/hdfs start nfs3

4) 挂载 192.168.1.66 (client):

# yum -y install nfs-utils

# showmount -e 192.168.1.65

# mount -t nfs -o vers=3,proto=tcp,nolock,noacl,sync,noatime 192.168.1.65:/ /mnt/

# cd /mnt

# ll #查看到HDFS中的信息

# vim /etc/fstab

192.168.1.65:/ /mnt nfs vers=3,proto=tcp,nolock,noacl,sync,noatime,\_net 0 0

# df -h # 完成自动挂载

5) 警告:

--启动portmap需要使用root用户

--启动nfs3需要使用core-site里面设置的代理用户

--必须先启动portmap之后再启动nfs3

--如果portmap重启了,在重启之后nfs3也需要重启

6) 启动与挂载详述

--vers=3, 目前NFS只能使用v3版本

--proto=tcp, 仅使用TCP作为传输协议

--nolocak, 不支持NLM

--noatime, 禁用access time的时间更新

--noacl, 禁用acl扩展权限

--sync, 最小化避免重排序写入造成不可预测的吞吐量,

未指定同步选项可能会导致上传大文件时出现不可靠行为

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

hadoop组件: 1.Zookeeper 2.kafka

1.1.1 Zookeeper (做Hadoop高可用必须用到)

1) Zookeeper是一个开源的分布式应用程序协调服务

2) Zookeeper是用来保证数据在集群间的事务一致性

3) Zookeeper应用场景:

--集群分布式锁

--集群同一命名服务

--分布式协调服务

1.1.2 Zookeeper角色与特性

1) leader: 接收所有Follower的提案请求统一协调发起提案的投票,负责与所有的Follower进行内部数据交换.

2) Follower: 直接为客户端服务并参与提案的投票,同时与Leader进行数据交换

3)Observer: 直接为客户端服务但并不参与提案的投票,同时也与Leader进行数据交换

1.1.3 Zookeeper角色与选举

1) 角色与选举

--服务在启动的时候是没有角色的(LOOKING)

--角色是通过选举产生的

--选举产生一个Leader,剩下的是Folloer

2) 选举Leader原则

--句群中超过半数机器投票选择Leader

--加入集群中拥有n台服务器,那么Leader必须得到n/2+1 台服务器的投票

3) 角色与选举

--如果Leader死亡,重新选举Leader

--如果死亡的机器数量达到一半,则集群挂掉

--如果无法得到足够的投票数量,就重新发起投票,如果参与投票的机器不足n/2+1,则集群停止工作.

--如果Follwer死亡过多,生育机器不足n/2+1.则集群也会停止工作

--Observer不计算在投票总设备数量里面

4)Zookeeper可伸缩扩展性原则与设计

--Leader所有写相关操作

--Fllower读操作与响应Leader提议

--在Observer出现之前,Zookeeper的伸缩性由Follwer来实现,我们可以通过添加Follwer节点的数量来保证Zookeeper服务的读性能,但是随着Follwer节点数量的增加,Zookeeper服务的写性能受到影响

--客户端提交 一个请求,若是读请求,则由每台server的本地副本数据库直接响应,若是写请求,需要通过一致性协议(Zab)来处理

--Zab协议规定: 来之Client的所有请求都要转发给ZK服务中唯一的Leader,由leader根据该请求发起一个Proposal.然后其他的server对该propasal进行Vote.之后Leader对Vote进行收集,当Vote数量过半时Leader会向所有的server发送一个通知消息.最后当Client所连接的server收到该消息时,会把该操作更新到内存中并对client的写请求作出回应

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2.1.0 Zookeeper集群

192.168.1.254:

# scp /linux-soft/04/hadoop/zookeeper-3.4.13.tar.gz 192.168.1.60:/root/

192.168.1.60:

# tar -xf zookeeper-3.4.13.tar.gz

# mv okeeper-3.4.13.tar.gz /usr/local/zookeeper

1) 配置文件改名zoo.cfg

# mv zoo\_sample.cfg zoo.cfg

2) zoo.cfg 文件最后添加如下内容

server.1=node1:2888:3888

server.2=node2:2888:3888

server.3=node3:2888:3888

server.4=nnn01:2888:3888:observer

3) 将zookeeper传给61 62 63

# scp /usr/local/zookeeper 192.168.1.61:/usr/local/zookeeper

4) 给每个zookeeper设置myid

ssh 192.168.1.61 "mkdir /tmp/zookeeper; echo "1" > /tmp/zookeeper/myid"

ssh 192.168.1.62 "mkdir /tmp/zookeeper; echo "2" > /tmp/zookeeper/myid"

ssh 192.168.1.63 "mkdir /tmp/zookeeper; echo "3" > /tmp/zookeeper/myid"

ssh 192.168.1.60 "mkdir /tmp/zookeeper; echo "4" > /tmp/zookeeper/myid"

5) 启动Zookeeper服务

192.168.1.60/61/62/63

# /usr/local/zookeeper/bin/zkServer.sh start

# /usr/local/zookeeper/bin/zkServer.sh status //等选举完

当前目录会生成 zookeeper.out的日志文件, 可以用来排错

# /usr/local/zookeeper/bin/zkServer.sh status

Mode: follower

Mode: leader

Mode: observer

6) Zookeeper 管理文档

http: //zookeeper.apache.org/doc/r3.4.10/zookeeperAdmin.html

eg: socat -TCP:node2:2181 //交互

conf //配置信息

7) 李欣脚本:

# scp /root/zkstats 192.168.1.60:/root/

# zkstats node{1..3} nn01 # 查看状态

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

3.1.0 kafka

3.1.1 kafka

--由LinkedIn开发的一个分布式的消息系统

--使用Scala编写

--一种消息中间件

--解耦,冗余,提高扩展性,缓冲

--保证顺序,灵活,削峰填谷

--异步通信

3.1.2 kafka角色与集群结构

--producer: 生产者,负责发布消息

--consumer: 消费者,负责读取处理消息

--topic: 消息类别

--Parition: 每个Topic包含一个或多个Partition

--Broker: Kafka集群包含一个或多个服务器

\*Kafka通过Zookeeper管理集群配置,选举Leader.

\* 也就是说kafka 必须要有一个高可用的zookeeper集群

3.1.3 Kafka的安装配置

--Kafka集群的安装配置依赖Zookeeper,搭建Kafka集群之前先创建好一个可用的Zookeeper集群

--安装 OpenJDK运行环境

--同步Kafka拷贝到所有集群主机

--修改配置文件

--启动与验证

# tar -xf

# vim

123 zookeeper.connect=node1:2181,node2:2181,node3:2181

21 broker.id=10

# rsync -av /usr/local/kafka 192.168.1.62/63:/usr/local/kafka

192.168.1.61/62/63

# ./bin/kafka-server-start.sh -daemon config/server.properties //启动服务

3.1.4 集群雅正与消息发布. (验证消息队列 集群正常使用)

1) 创建一个topic. 名字: mymsg --help可查看

node1:

# ./bin/kafka-topic.sh --create --partitions 2 --replication-factor 2 --zookeeper node1:2181 --topic mymsg

2) 生产者

node2:

# ./bin/kafka-console-producer.sh --broker-list localhost:9092 --topic mymsg

// 使用本机的消息队列, localhost:9092,指定topic mymsg

3) 消费者

node3:

# ./bin/kafka-console-consumer.sh --bootstrap-server localhosts:9092 --topic mymsg

//使用本机的消息队列, localhost:9092,指定topic mymsg

4) 生产者输入 消息, 消费者端可以看到消息

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

4.1.0 Hadoop高可用

4.1.1 为什么需要NameNode

--NameNode是HDFS的核心配置,HDFS又是Hadoop核心组件

4.2.0 系统规划:

192.168.1.60 NameNode1 Hadoop

192.168.1.66 NameNode2 Hadoop

192.168.1.61 DataNode HDFS

journalNode Zookeeper

zookeeper

192.168.1.62 DataNode HDFS

journalNode Zookeeper

zookeeper

192.168.1.63 DataNode HDFS

journalNode Zookeeper

zookeeper

4.2.1 准备nn02

1) ip 192.168.1.66

2) hostname nn02

3) vim /etc/hosts

192.168.1.60 nn01

192.168.1.61 node1

192.168.1.62 node2

192.168.1.63 node3

192.168.1.66 nn02

4) 配置ssh信任

nn01:

ssh-copy-id root@192.168.1.66

scp /root/.ssh/id\_rsa root@192.168.1.66:/root/.ssh/

5) yum -y install java-1.8.0-openjdk-devel

4.2.2 配置文件

1）配置 core-site

[root@nn01 .ssh]# vim /usr/local/hadoop/etc/hadoop/core-site.xml

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://nsdcluster</value>

//nsdcluster是随便起的名。相当于一个组，访问的时候访问这个组

</property>

<property>

<name>hadoop.tmp.dir</name>

<value>/var/hadoop</value>

</property>

<property>

<name>ha.zookeeper.quorum</name>

<value>node1:2181,node2:2181,node3:2181</value> //zookeepe的地址

</property>

<property>

<name>hadoop.proxyuser.nfs.groups</name>

<value>\*</value>

</property>

<property>

<name>hadoop.proxyuser.nfs.hosts</name>

<value>\*</value>

</property>

</configuration

2）配置 hdfs-site

[root@nn01 ~]# vim /usr/local/hadoop/etc/hadoop/hdfs-site.xml

<configuration>

<property>

<name>dfs.replication</name>

<value>2</value>

</property>

<property>

<name>dfs.nameservices</name>

<value>nsdcluster</value>

</property>

<property>

<name>dfs.ha.namenodes.nsdcluster</name>

//nn1,nn2名称固定，是内置的变量，nsdcluster里面有nn1，nn2

<value>nn1,nn2</value>

</property>

<property>

<name>dfs.namenode.rpc-address.nsdcluster.nn1</name>

//声明nn1 8020为通讯端口，是nn01的rpc通讯端口

<value>nn01:8020</value>

</property>

<property>

<name>dfs.namenode.rpc-address.nsdcluster.nn2</name>

//声明nn2是谁，nn02的rpc通讯端口

<value>nn02:8020</value>

</property>

<property>

<name>dfs.namenode.http-address.nsdcluster.nn1</name>

//nn01的http通讯端口

<value>nn01:50070</value>

</property>

<property>

<name>dfs.namenode.http-address.nsdcluster.nn2</name>

//nn01和nn02的http通讯端口

<value>nn02:50070</value>

</property>

<property>

<name>dfs.namenode.shared.edits.dir</name>

//指定namenode元数据存储在journalnode中的路径

<value>qjournal://node1:8485;node2:8485;node3:8485/nsdcluster</value>

</property>

<property>

<name>dfs.journalnode.edits.dir</name>

//指定journalnode日志文件存储的路径

<value>/var/hadoop/journal</value>

</property>

<property>

<name>dfs.client.failover.proxy.provider.nsdcluster</name>

//指定HDFS客户端连接active namenode的java类

<value>org.apache.hadoop.hdfs.server.namenode.ha.ConfiguredFailoverProxyProvider</value>

</property>

<property>

<name>dfs.ha.fencing.methods</name> //配置隔离机制为ssh

<value>sshfence</value>

</property>

<property>

<name>dfs.ha.fencing.ssh.private-key-files</name> //指定密钥的位置

<value>/root/.ssh/id\_rsa</value>

</property>

<property>

<name>dfs.ha.automatic-failover.enabled</name> //开启自动故障转移

<value>true</value>

</property>

</configuration>

3）配置yarn-site

[root@nn01 ~]# vim /usr/local/hadoop/etc/hadoop/yarn-site.xml

<configuration>

<!-- Site specific YARN configuration properties -->

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.resourcemanager.ha.enabled</name>

<value>true</value>

</property>

<property>

<name>yarn.resourcemanager.ha.rm-ids</name> //rm1,rm2代表nn01和nn02

<value>rm1,rm2</value>

</property>

<property>

<name>yarn.resourcemanager.recovery.enabled</name>

<value>true</value>

</property>

<property>

<name>yarn.resourcemanager.store.class</name>

<value>org.apache.hadoop.yarn.server.resourcemanager.recovery.ZKRMStateStore</value>

</property>

<property>

<name>yarn.resourcemanager.zk-address</name>

<value>node1:2181,node2:2181,node3:2181</value>

</property>

<property>

<name>yarn.resourcemanager.cluster-id</name>

<value>yarn-ha</value>

</property>

<property>

<name>yarn.resourcemanager.hostname.rm1</name>

<value>nn01</value>

</property>

<property>

<name>yarn.resourcemanager.hostname.rm2</name>

<value>nn02</value>

</property>

</configuration>

4)具体[配置文件在hadoop-ha中,共7个配置文件.

core-site.xmln

excluden

hadoop-env.shn

hdfs-site.xmln

mapred-site.xmln

slavesn

yarn-site.xmln

4.3.0 集群初始化

1). 删除数据 node{1..2} nn01 nn02

# rm -rf /var/hadoop/\*

2) 同步hadoop数据

for i in nn01 node1 node2 node3

do

rsync -aHSX /usr/local/hadoop $i:/usr/local/

done

3) nn01 初始化ZK集群

# ./bin/hdfs zkfc -formatZK

4) node1/2/3: 启动journalnode服务

# /usr/local/hadoop/sbin/hadoop-daemon.sh start journalnode

# jps //查看角色

5) nn01: 格式化

# ./bin/hdfs namenode -format

//var/hadoop 创建文件夹,及初始数据

6) nn02 数据同步到nn02的/var/hadoop/dfs

# rsync -avXHS /var/hadoop/dfs root@192.168.1.66:/var/hadoop/

7) nn01初始化JNS

# ./bin/hdfs namenode -initializeSharedEdits

8) node1/2/3 停止journalnode服务. 初始化完,交由电脑管理

# ./sbin/hadoop-daemon.sh stop journalnode

9) 启动集群

nn01: 启动dhfs

# ./sbin/start-dfs.sh

nn01: 启动yarn

# ./sbin/start-yarn.sh #./sbin/start-all.sh //同时启动

nn02: 启动热备ResourceManager

# ./sbin/yarn-daemon.sh start resourcemanager

4.3.1 查看集群状态

nn01:

# ./bin/hdfs headmin -getServiceState nn01

#./bin/hdfs headmin -getServiceState nn02

2) 获取ResourceManager状态

# ./bin/yarn rmadmin -getServiceState rm1

# ./bin/yarn rmadmin -getServiceState rm1

3) 查看集群状态

--获取节点信息

# ./bin/hdfs dfsadmin -report

# ./binyarn node -list

4) 访问集群文件

# ./bin/hadoop fs -mkdir /input

# ./bin/hadoop fs -ls hdfs://mycluster/

5) 主从切换Active

# ./sbin/hadoop-daemon.sh stop namenode