# ansible简介

• Ansible是2013年推出的一款IT自动化和DevOps软件,目前由Redhat已签署Ansible收购协议。其是基于Python研发,糅合了很多老运维工具的优点实现了批量操作系统配置,批量程序的部署,批量运行命令等功能

• ansible可以让我们实现:

– 自动化部署APP

– 自动化管理配置项

– 自动化的持续交付

– 自动化的(AWS)于服务管理

devops运维开发需要的技能ansible、centens、git、shell、python

为什么要选择ansible

• 选择一款配置管理软件总的来说,无外乎从以下几点来权衡利弊

– 活跃度(社区活跃度)

– 学习成本

– 使用成本

– 编码语言

– 性能

– 使用是否广泛

图形比较 见ppt

• ansible优点

– 是仅需要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

• ansible缺点

– 对于几千台、上万台机器的操作,还不清楚性能、效率情况如何,需要进一步了解。

ansible特性

• 模块化设计,调用特定的模块来完成特定任务

• 基于pythone语言实现

– paramiko

– PyYAML (半结构化诧言)

– jinja2

• 其模块支持JSON等标准输出格式,可采用任何编程语言重写

ansible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

# ansible安装

//在centos官网，找一个aliyun，找到extras文件夹底下找ansible的软件包下载

软件依赖关系

• 对管理主机

– 要求Python 2.6 或 Python 2.7

– ansible 使用了以下模块,都需要安装

– paramiko

– PyYAML

– Jinja2

– httplib2

– six

• 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 主机登彔

– 在托管节点上也需要安装 Python 2.5 或以上的版本

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

安装ansible

• ansible 可以基于源码运行

• 源码安装

– pip,需要配置扩展软件包源 extras

– git

yum install epel-release

yum install git python2-pip

– pip安装依赖模块

pip install paramiko PyYAML Jinja2 httplib2 six

• ansible 源码下载

– git clone git://github.com/ansible/ansible.git

– yum install python-setuptools python-devel

– python setup.py build

– python setup.py install

• pip 方式安装

– pip install ansible

• yum 扩展源安装简单,自动解决依赖关系(推荐)

– http://mirror.centos.org/.../.../extras/

– yum install ansible

• 安装完成以后验证

– ansible -version

ansible 01

1 ansible 管理机器

2 web1 托管机器

3 web2 托管机器

4 db1 托管机器

5 db2 托管机器

6 cache 托管机器

配置一个 yum 源，使用 CentOS-7-x86\_64-Everything-1511.iso

mount -t iso9660 -o ro,loop /ISO/CentOS-7-x86\_64-Everything-1511.iso /var/ftp/rhel7

在创建一个目录用于存放 ansible 的安装包文件

mkdir /var/ftp/ansible

拷贝安装文件到该目录，后创建索引

cd /var/ftp/ansible

createrepo .

配置 ansible 管理机器，添加 repo 文件

[ansible]

name=ansible

baseurl=ftp://192.168.4.254/ansible

enabled=1

gpgcheck=0

在 ansible 管理机器上

yum clean all

yum install -y ansible

安装完成以后执行，没有报错，正确显示版本即可

ansible --version

## 主机定义与分组

ansible 的配置文件是 ansible.cfg

ansible.cfg 的查找顺序是

1 ANSIBLE\_CONFIG 变量定义的配置文件

2 当前目录下的 ./ansible.cfg 文件

3 当前用户家目录下 ~/ansible.cfg 文件

4 /etc/ansible/ansible.cfg 文件

ansible.cfg 配置文件

– inventory 是定义托管主机地址配置文件

– 首先编辑 /etc/ansible/hosts 文件,写入一些进程主机的地址。

• 格式

– # 表示注释

[组名称]

主机名称或ip地址,登彔用户名,密码、端口等信息

• 测试

– ansible [组名称] --list-hosts

ansible.cfg 中 inventony 指定主机分组文件的路径和地址，默认分组文件 hosts，取消注释

hosts 的配置

[web]

web1

web2

xx3

l4

[web:vars] 对一个组进行授权连接

ansible\_ssh\_user="root"

ansible\_ssh\_pass="123456"

[db]

db[1:2] ansible\_ssh\_user="root" ansible\_ssh\_pass="123456"

[other]

cache ansible\_ssh\_user="root" ansible\_ssh\_pass="123456"

[app:children] # 指定子组

web

db

[app:vars] 对app组进行授权连接

ansible\_ssh\_user="root"

ansible\_ssh\_pass="123456"

– ansible [组名称] -m ping 测试主机连通情况

ansible app -m -ping

[root@ansible ~]# vim /etc/ansible/ansible.cfg

61 host\_key\_checking = False //将这行的注释取消，连接时不用输入yes或no

或者 修改/etc/ssh/ssh\_config

35 StrictHostKeyChecking no //将这行的注释取消，连接时不用输入yes或no

## ansible 命令基础

ansible 主机分组 -m 模块 -a '命令和参数'

• 自定义配置文件

– 创建文件夹 myansible

– 创建配置文件 ansible.cfg

[defaults]

inventory = myhost

– 配置主机文件

vim myhost

[nginx]

192.168.1.11

192.168.1.12

192.168.1.13

– ansible nginx --list-hosts

## 动态主机

• 无限可能

– Ansible Inventory实际上是包含静态Inventory和动态Inventory两部分,

静态Inventory指的是在文件/etc/ansible/hosts中指定的主机和组,

Dynamic Inventory指通过外部脚本获取主机列表,并按照ansible 所要求的格式返回给ansilbe命令的。

• json

– JSON的全称是”JavaScript Object Notation”,意思是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。

• 注意事项:

– 1、主机部分必须是列表格式的;

– 2、hostdata行,其中的"hosts" 部分可以省略,但如果使用时,必须是"hosts"

• 脚本输出主机列表

#!/usr/bin/python

import json

hostlist = {}

hostlist["bb"] = ["192.168.1.15", "192.168.1.16"]

hostlist["192.168.1.13"] = {

"ansible\_ssh\_user":"root","ansible\_ssh\_pass":"pwd"

}

hostlist["aa"] = {

"hosts" : ["192.168.1.11", "192.168.1.12"],

"vars" : {

"ansible\_ssh\_user":"root","ansible\_ssh\_pass":"pwd"

}

}

print( json.dumps(hostlist))

将配置文件中，inventory = 脚本绝对路径

会调用脚本的输出信息

• 脚本输出样例

{

"aa" : {

"hosts" : ["192.168.1.11", "192.168.1.12"],

"vars" : {

"ansible\_ssh\_user" : "root",

"ansible\_ssh\_pass" : "pwd"

}

},

"bb" : ["192.168.1.15", "192.168.1.16"],

"192.168.1.13": { "ansible\_ssh\_user" : "root",

"ansible\_ssh\_pass" : "pwd"}

}

批量执行

ansible命令基础

• ansible <host-pattern> [options]

– host-pattern 主机或定义的分组

– -M 指定模块路径

– -m 使用模块,默认 command 模块

– -a or --args 模块参数

– -i inventory 文件路径,戒可执行脚本

– -k 使用交亏式登彔密码

– -e 定义变量

– -v 详绅信息,-vvvv 开吭 debug 模式

• 列出要执行的主机,不执行任何操作

– ansible all --list-hosts

• 批量检测主机

– ansible all -m ping

• 批量执行命令

– ansible all -m command -a 'id' -k //-k 交互式输入密码

批量部署证书文件

• 每次交互输入密码比较麻烦

• 密码写入配置文件安全性很差

• 不同主机不同密码,配置文件要上天

• 使用 key 方式认证,是一个不错的选择

• 给所有主机部署公钥

创建密钥对 id\_rsa 是私钥， id\_rsa.pub 是公钥

ssh-keygen -t rsa -b 2048 -N ''

给所有主机部署密钥，执行命令之前，把hosts 文件里面的认证用户和密码删除

ansible all -m authorized\_key -a "user=root exclusive=true manage\_dir=true key='$(< /root/.ssh/id\_rsa.pub)'" -k -v

• 报错

– "msg": "Using a SSH password instead of a key is

not possible because Host Key checking is

enabled and sshpass does not support this.

Please add this host's fingerprint to your

known\_hosts file to manage this host."

– 解决方法:

– 修改 ansible.cfg

host\_key\_checking = False

# 模块

• ansible-doc

– 模块的手册,相当于 shell 的 man

– 非常重要,非常重要,非常重要

ansible-doc 查看帮助，必须掌握

ansible-doc -l 列出所有模块

ansible-doc 模块名 查看该模块的帮助信息

ping

没有参数，检测主机的连通性，与 ping 无关，主要检测 ssh 是否可以连接

command | shell | raw

command 是默认模块，没有启用 shell ，所有shell 相关特性命令无法使用，例如 < > | &

raw 模块，没有 chdir create remove 等参数，能执行大部分操作

shell 模块，启动 /bin/sh 运行命令，可以执行所有操作

– ansible host-pattern -m command -a '[args]'

– 查看所有机器负载

ansible all -m command -a 'uptime'

– 查看日期和时间

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

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

– 下列命令执行会失败

ansible all -m command -a 'ps aux|grep ssh'

ansible all -m command -a 'set'

测试

ansible cache -m command -a 'chdir=/tmp touch f1' 创建成功

ansible cache -m shell -a 'chdir=/tmp touch f2' 创建成功

ansible cache -m raw -a 'chdir=/tmp touch f3' 文件可以创建，但无法切换目录，文件在用户家目录下生成

复杂操作怎么办，使用脚本来解决

#!/bin/bash

adduser zhang3 # 创建用户 zhang3

echo 123456 |passwd --stdin zhang3 # 修改密码

chage -d 0 zhang3 # 第一次登录必须修改密码

ansible all -m script -a "urscript.sh"

• copy 模块

– 复制文件到进程主机

– src:要复制到进程主机的文件在本地的地址,可以是绝对路径,也可以是相对路径。

如果路径是一个目彔,它将递归复制。在这种情况下,如果路径使用"/"来结尾,

则只复制目彔里的内容。如果没有使用"/"来结尾,则包含目彔在内的整个内容全部复制,类似于rsync

– dest:必选项。进程主机的绝对路径,如果源文件是

一个目彔,那么该路径也必须是个目彔

– backup:在覆盖之前将原文件备份,备份文件包含时

间信息。有两个选项:yes|no

– force:如果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,如果为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

– 复制文件

ansible t1 -m copy -a 'src=/root/alog dest=/root/a.log'

– 复制目彔

ansible t1 -m copy -a 'src=urdir dest=/root/'

lineinfile | replace 模块

copy 把文件发布到远程其他主机上面

lineinfile 修改一个文件的一行，以行为基础，整行修改

replace 修改文件的某一部分，以正则表达式匹配为基础修改

– 类似 sed 的一种行编辑替换模块

– path 目的文件

– regexp 正则表达式

– line 替换后的结果

//使用lineinfile时，如果正则匹配不到对应的行，就会在最后一行添加line=的内容

//如果要关闭这个自动添加，需要在line=下一行添加backrefs: yes

//这个在playbook中也是一样的

ansible t1 -m lineinfile -a 'path="/etc/selinux/config" regexp="^SELINUX=" line="SELINUX=disabled"'

– 替换指定字符

ansible t1 -m replace -a 'path="/etc/selinux/config"

regexp="^(SELINUX=).\*" replace="\1disabled"'

利用 copy 模块修改所有机器的 /etc/resolv.conf 为 nameserver 8.8.8.8

利用 lineinfile 修改 /etc/sysconfig/network-scriopts/ifcfg-eth0

PEERDNS=yes|no

利用 replace 修改 /etc/sysconfig/network-scriopts/ifcfg-eth0

PEERROUTES=yes|no

yum 模块 installed 安装， removed 删除

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

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

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

• yum模块

– 使用yum包管理器来管理软件包

– config\_file:yum的配置文件

– disable\_gpg\_check:关闭gpg\_check

– disablerepo:不启用某个源

– enablerepo:启用某个源

– name:要进行操作的软件包的名字,也可以传递一个

url或者一个本地的rpm包的路径

– state:状态(present,absent,latest)

service 模块 name 指定服务名称，enabled= yes|no 设置开机启动， state=stopped|started |restarted|reloaded启动关闭服务

设置 sshd 服务开启启动，并启动服务；sleep:如果执行了restarted,在则stop和start之间

沉睡几秒钟

ansible other -m service -a 'name="sshd" enabled="yes" state="started"'

setup 模块，查看信息 filter 过滤指定的关键字

[root@ansible ~]# ansible cache -m setup查看客户端设备的所有信息

• setup模块

– 主要用亍获取主机信息,在playbooks里经常会用到的

一个参数gather\_facts就不该模块相关。setup模块下

经常使用的一个参数是filter参数

– filter 可以过滤到我们需要的信息

ansible t1 -m setup -a 'filter=ansible\_distribution'

ansible t1 -m setup -a 'filter="ansible\_os\_family"' 查看linux系统版本，这里为redhat

课堂练习

• 综合练习

– 安装 apache

– 修改 apache 监听的端口为 8080

– 为 apache 增加 NameServer 配置

– 设置默认主页

– 启动服务

– 设置开机自启动

[root@ansible ~]# ansible web -m yum -a 'name="httpd" state=installed'

[root@ansible ~]# ansible web -m lineinfile -a 'path="/etc/httpd/conf/httpd.conf" regexp="^Listen 80" line="Listen 8080"'

[root@ansible ~]# ansible web -m replace -a 'path="/etc/resolv.conf" regexp="^(nameserver).\*" replace="\1 192.168.1.254"'

[root@ansible ~]# ansible web -m service -a 'name="httpd" enabled="yes" state="started"'

[root@ansible ~]# ansible web -m shell -a 'ss -antpul | grep 8080'

作业讲解和回顾

1 在 web 上安装 apache

ansible web -m yum -a 'name="httpd" state=installed'

2 修改 httpd.conf 的配置

3 把 httpd 同步到远程机器上

ansible web -m copy -a 'src="/root/httpd.conf" dest="/etc/httpd/conf/httpd.conf"'

4 设置默认主页 hello nsd1711

ansible web -m shell -a 'echo "hello nsd1711" >/var/www/html/index.html'

5 设置开机自启动，开启服务

ansible web -m service -a 'name="httpd" enabled=yes state=started'

大型架构及配置技术

# playbook基础

ansible七种武器

• 第一种武器

– ansible 命令,用于执行临时性的工作,也是我们之前主要学习的功能,必须掌握

• 第二种武器

– ansible-doc 是 Ansible模块文档说明,针对每个模块都有详细的用法说明及应用案例介绍,

功能和Linux系统man命令类似,必须掌握

• 第三种武器

– ansible-console 是 Ansible 为用户提供的一款交互式工具,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,这为习惯于使用Shell交互方式的用户提供了良好的

使用体验。

• 第四种武器

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

• 第五种武器

– ansible-playbook 是日常应用中使用频率最高的命令,其工作机制是:通过读取预先编写好的

playbook 文件实现批量管理。要实现的功能与命令 ansible 一样,可以理解为按一定条件组成的

ansible任务集,必须掌握

• 第六种武器

– ansible-vault 主要用于配置文件加密,如编写的Playbook 配置文件中包含敏感信息,不希望其他人随

意查看, ansible-vault 可加密/解密这个配置文件

• 第七种武器

– ansible-pull

– Ansible 有两种工作模式 pull/push ,默认使用 push模式工作,pull 模式和通常使用的 push 模式工作机

理刚好相反，

– 适用场景:有数量巨大的机器需要配置,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会使用,灵活性稍有欠缺,但效率几乎可以无限提升,

对运维人员的技术水平和前瞻性规划有较高要求。

playbook

• playbook 是什么?

– playbook 是 ansible 用于配置,部署,和管理托管主机剧本。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的状态。

– 也可以这么理解,playbook 字面意思,即剧本,现实中由演员按照剧本表演,

在 Ansible 中由计算机进行表演,由计算机安装,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

• 为什么要使用playbook

– 执行一些简单的任务,使用ad-hoc命令可以方便的解决问题,但是有时一个设施过于复杂,

需要大量的操作时候,执行的 ad-hoc 命令是不适合的,这时最好使用playbook,

就像执行 shell 命令与写 shell 脚本一样,也可以理解为批处理任务

– 使用 playbook 你可以方便的重用编写的代码,可以移植到不同的机器上面,

像函数一样,最大化的利用代码在使用 Ansible 的过程中,你也会发现,你所处理的大

部分操作都是编写 playbook

playbook语法基础

• 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

– 同一个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

方法都是键值中间以 ": " 分隔表示

– YAML 还有一个小的怪癖. 所有的 YAML 文件开始行都

应该是 ---. 这是 YAML 格式的一部分, 表明一个文件的开始

• playbook 构成

– Target: 定义将要执行 playbook 的远程主机组

– Variable: 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

– Handler: 定义 task 执行完成以后需要调用的任务

• Playbook执行结果

• 使用 ansible-playbook 运行playbook文件,得到输出内容为 JSON 格式。并且由不同颜色组成,便于识别。

一般而言

• 绿色代表执行成功

• \*\*\*代表系统代表系统状态发生改变

• 红色代表执行失败

• 第一个playbook

---

- hosts: all

remote\_user: root

tasks:

- ping:

# 第一行,表示开始

ansible-playbook myping.yml -f 5

– -f 并发进程数量,默认是 5，不建议超过cpu核心数的2倍

– hosts 行的内容是一个或多个组或主机的 patterns,以逗号为分隔符

– remote\_user 就是账户名

续 ... ...

– tasks

– 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(通过 host

pattern 匹配的所有主机)执行完毕之后,下一个 task

才会执行.

– 有一点需要明白的是(很重要),在一个 play 之中,

所有 hosts 会获取相同的任务指令,这是 play 的一个

目的所在,也就是将一组选出的 hosts 映射到 task,执

行相同的操作

给所有主机添加用户 plj，设置默认密码 123456，要求第一次登录修改密码

先通过ansible-doc user 查看使用案例

---

- hosts: web

remote\_user: root

tasks:

- name: aaabbb

user: name="plj" group="wheel"

- name: xxx

shell: echo 123456 |passwd --stdin plj

- name: ooo

shell: chage -d 0 plj

编写 playbook 实现以下效果

安装 apache

修改 apache 监听的端口为 8080

为 apache 增加 NameServer 配置

设置默认主页 hello world

启动服务，并设置设置开机自启动

---

- hosts: cache

remote\_user: root

tasks:

- name: install the latest version of Apache

yum:

name: httpd

state: installed

- copy:

src: /root/httpd.conf

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

owner: apache

group: apache

mode: 0644

# - lineinfile:

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

# regexp: '^Listen'

# line: 'Listen 8080'

# - lineinfile:

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

# regexp: '#ServerName'

# line: 'ServerName localhost'

- service:

name: httpd

enabled: yes

state: started

- shell: echo "hello world" >index.html

args:

chdir: /var/www/html

playbook进阶

变量

• 添加用户

– 给所有主机添加用户 plj,设置默认密码 123456

– 要求第一次登录修改密码(使用变量)

- hosts: 192.168.1.16

remote\_user: root

vars:

username: plj

tasks:

- name: create user "{{username}}"

user: group=wheel uid=1000 name={{username}}

- shell: echo 123456 | passwd --stdin plj

- shell: chage -d 0 {{username}}

user 模块的 password不能设置密码

– 经过测试发现,password 是把字符串直接写入shadow,并没有改变,而 Linux 的 shadow 密码是

经过加密的,所以不能使用

– 解决方案:

– 变量过滤器 password\_hash

– {{ 'urpassword' | password\_hash('sha512')}}

ansible 进阶，变量过滤器，条件与判断

---

- hosts: cache

remote\_user: root

vars:

user: jj

tasks:

- user:

name: "{{user}}"

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

group: wheel

- shell: chage -d 0 "{{user}}"

变量过滤器 password\_hash

error

• ansible-playbook 对错误的处理

– 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

ansible-playbook 在执行命令时，如果出现错误就会停止执行

可以使用忽略错误的方式来继续执行，让错误的命令输出正确的$?

- shell： userdel -r zhang3 || true

或者

- shell： userdel -r zhang3

ignore\_errors: True

handlers

• 用于当关注的资源发生变化时采取一定的操作。

• "notify" 这个action可用于在每个play的最后被触发这样可以避免多次有改变发生时

每次都执行指定的操作取而代之仅在所有的变化发生完成后一次性地执行指定操作。

• 在 notify 中列出的操作称为 handler 也即 notify 中调用 handler 中定义的操作

触发器 handlers 的设置

只有当notify上面的命令导致结果发生变化时，notify的命令才会执行

---

- hosts: web

remote\_user: root

tasks:

- name: install the latest version of Apache

yum:

name: httpd

state: installed

- copy:

src: /root/httpd.conf

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

owner: apache

group: apache

mode: 0644

notify:

- restart apache //这里的notify下的名字一定要和handlers的name一致

- shell: echo "hello world" >index.html

args:

chdir: /var/www/html

handlers:

- name: restart apache

service: name=httpd state=restarted

• 注意事项:

– notify 调用的是 handler 段 name 定义的串,必须一致,否则达不到触发的效果

– 多个 task 触发同一个 notify 的时候,同一个服务只会触发一次

– notify 可以触发多个条件,在生产环境中往往涉及到某一个配置文件的改变要重启若干服务的场景,

handler 用到这里非常适合.

– 结合 vars 可以写出非常普适的服务管理脚本

when

• 某些时候我们可能需要在满足特定的条件后在触发某一项操作,或在特定的条件下终止某个行为,这个时

候我们就需要进行条件判断,when 正是解决这个问题的最佳选择,远程中的系统变量 facts 变量作为

when 的条件,这些 facts 我们可以通过 setup 模块查看

---

- name: Install VIM

hosts: all

tasks:

- name: Install VIM via yum

yum: name=vim-enhanced state=installed

when: ansible\_os\_family == "RedHat"

- name: Install VIM via apt

apt: name=vim state=installed

when: ansible\_os\_family == "Debian"

• register

– 有时候我们可能还需要更复杂的例子,比如判断前一个命令的执行结果,根据结果处理后面的操作,这时

候我们就需要 register 模块来保存前一个命令的返回状态,在后面进行调用

- command: test command

register: result

- command: run command

when: result

• 变量注册

– 例如我们需要判断 plj 这个用户是否存在

– 如果存在我就修改密码,如果不存在就跳过

tasks:

- shell: id {{username}}

register: result

- name: change "{{username}}" password

user: password={{'12345678'|password\_hash('sha512')}} name={{username}}

when: result

××××register和ignore\_errors 冲突

• 变量注册进阶

– 我们还可以针对运行命令结果的返回值做判定

– 当系统负载超过一定值的时候做特殊处理

结合 shell 脚本判断，当系统负载超过 0.7 的时候 终止 httpd 进程

---

- hosts: web

remote\_user: root

tasks:

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

register: result

- service:

name: httpd

state: stopped

when: result.stdout|float >0.7

//也可以这样写uptime | awk -F [,\ ] '{print $(NF-4)}'

with\_items

• with\_items 是 playbook 标准循环,最常用到的就是它,with\_items

可以用于迭代一个列表或字典,通过{{ item }}获取每次迭代的值

– 例如创建多个用户

---

- hosts: 192.168.1.16

remote\_user: root

tasks:

- name: add users

user: group=wheel password={{'123456' |password\_hash('sha512')}} name={{item}}

with\_items: ["nb", "dd", "plj", "lx"]

with\_items 循环添加用户

---

- hosts: cache

remote\_user: root

vars:

pwd: "123456"

tasks:

- user:

name: "{{item.name}}"

group: "{{item.group}}"

with\_items: [

{name: "nb", group: "root"},

{name: "dd", group: "root"},

{name: "jj", group: "wheel"},

{name: "lx", group: "wheel"}

]

循环嵌套

---

- hosts: cache

remote\_user: root

vars:

un: [a, b, c]

id: [1, 2, 3]

tasks:

- name: add users

shell: echo {{item}}

with\_nested:

- "{{un}}"

- "{{id}}"

tags

• 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

– -t TAGS, --tags=TAGS

– --skip-tags=SKIP\_TAGS

– --start-at-task=START\_AT

• tags样例:

vars:

soft: httpd

tasks:

- name: install {{soft}}

yum: name={{soft}}

- name: config httpd.conf

copy: src=/root/playbook/httpd.conf dest=/etc/httpd/conf/httpd.conf

- name: config services

service: enabled=yes state=restarted name={{soft}}

tags: restartweb

• 调用方式

ansible-playbook i.yml --tags=restartweb

include and roles

• 我们在编写 playbook 的时候随着项目越来越大,playbook 也越来越复杂,修改起来也越来越麻烦。

这时候可以把一些 play、task 或 handler 放到其他文件中,然后通过include指令包含进来是一个不错的选择

tasks:

- include: tasks/setup.yml

- include: tasks/users.yml user=plj

#users.yml 中可以通过{{ user }}来使用这些变量

handlers:

- include: handlers/handlers.yml

• roles 像是加强版的 include,他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

debug

• 对于 python 语法不熟悉的同学,playbook 书写起来容易出错,且排错困难,这里介绍几种简单的排错

调试方法

– 检测语法

ansible-playbook --syntax-check playbook.yaml

– 测试运行

ansible-playbook -C playbook.yaml

– 显示收到影响到主机 --list-hosts

– 显示工作的 task --list-tasks

– 显示将要运行的 tag --list-tags

• debug 模块可以在运行时输出更为详细的信息,来

帮助我们排错,debug 使用样例:

调试模块 debug，显示对象的所有属性

---

- hosts: web

remote\_user: root

tasks:

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

register: result

- service:

name: httpd

state: stopped

when: result.stdout|float >0.7

- name: Show debug info

debug: var=result

运行侯会显示result显示的过程信息

会显示result的过程信息：

rc：0

stderr：

stderr\_lines:

stdout:

stdout\_lines: