**Ansible2:playbook**

七种武器：

**• 第一种武器**

– ansible 命令,用亍执行临时性的工作,也是我们乊前

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

都有详绅的用法说明及应用案例介终,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 虚拟出杢的织

端上像 Shell 一样使用 Ansible 内置的各种命令,

这为习惯亍使用 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,不 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:通过读叏预先编写好的 playbook 文

件实现批量管理。要实现的功能不命令 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 主要用亍配置文件加密,如编写的

Playbook 配置文件中包含敏感信息,丌希望其他人随

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

幵収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

**• json**

– json 是 JavaScript 对象表示法,它是一种基亍文本,

独立亍诧言的轻量级数据交换格式。

– json中的分隔符限亍单引号 ' 、小括号 ()、中括号[ ]、大括号 { } 、冒号 : 和逗号 ,

• 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 迚行解析

• 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

– 中括号保存数组

• 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,例如:

"漂亮姐" : "庞丽静"

• json 诧法规则数组

{ "讲师":

["牛犇", "丁丁", "静静","李欣"]

}

• 复合复杂类型

{ "讲师":

[

{"丁丁":"老逗逼", "负责阶段":"2"},

{"静静":"漂亮姐", "负责阶段":"3"},

{"李欣":"老司机", "负责阶段":"4"}

]

}

**yaml简介**

• yaml 基础语法

– YAML的结构通过空格来展示

– 数组使用"- "来表示

– 键值对使用": "来表示

– YAML使用一个固定的缩迚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

• 注意:

–不要使用tab,缩迚是初学者容易出错的地方之一

– 同一层级缩进必须对齐

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

"庞丽静": "漂亮姐"

– 戒

"庞丽静":

"漂亮姐"

**jinja2 模版基本语法**

– 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;

– 另外,模板也支持注释,都是包含在分隔符 "{# #}"内,支持块注释。

– 调用变量{{varname}}

– 计算 {{2+3}}

– 判断 {{1 in [1,2,3]}}

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

jinja2 模版控制语句

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

... ...

{% elif name == '漂亮姐' %}

{% for method in [约会, 逛街, 吃饭, 看电影, 去宾馆] %}

{{do method}}

{% endfor %}

... ...

{% endif %}

jinja2 过滤器

– 变量可以通过 过滤器 修改。过滤器不变量用管道符号

( | )分割,幵且也 可以用圆括号传递可选参数。

**Playbook**

Ansible的剧本，达到自动化运行。

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亍复杂,需要大量的操作时候,执行的 ad-hoc 命令是不合适的,这时最好使用playbook,就像执行 shell 命令不写 shell 脚本一样,也可以理解为批处理任务。

playbook由 YAML 语言编写,遵循 YAML 标准

– 在同一行中,#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迚

– playbook 由一个戒多个 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 的进程主机组

– Variable: 定义 playbook 运行时需要使用的变量

– Tasks: 定义将要在进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

实际操作：

编写playbook学会查询:ansible-doc yum ---- 查询yum相关的应用（EXAMPLE中有具体的用法）

E.g

配置web机子的apache服务：

---

- hosts: web

remote\_user: root

tasks:

- name: install the latest version of Apache

yum:

name: httpd

state: installed

- lineinfile:

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

regexp: '^Listen '

insertafter: '^#Listen '

line: 'Listen 8080'

- lineinfile:

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

regexp: '^#ServerName'

line: 'ServerName localhost'

- copy:

src: /root/index.html

dest: /var/www/html/index.html

owner: apache

group: apache

mode: 0644

- service:

name: httpd

state: started

enabled: yes

使用playbook增加变量vars: user=nb，引用为：”{{user}}”

注意：user中写：

Password: 12345 --->执行中会直接在/etc/shadow中写明码12345密码不对，需要写加密的密码，故可以两种方式：

1. 在本机上添加用户输入密码，然后将生成的加密密码写入playbook
2. 在playbook中写入过滤器： 密码|password\_hash(‘加密算法’)

/etc/login.defs----系统的密码设置配置

经过过滤器可以设置linux中的密码

---

- hosts: other

remote\_user: root

vars:

un: lixin

tasks:

- user:

name: "{{un}}"

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

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

向上百台机器上增加密码用户：

在有很多台机器又有很多没有密码时，区分判断比较麻烦 ----> 解决思路：全部添加该用户，若用户存在，则不添加，否则则添加：

在执行错误的情况默认情况判断 $?,如果 值不为 0 就停止执行

在playbook中如果上一次的结果没有$?为非0则会结束。--->解决：

1.写一个true再来$?则为0 ： CMD || /bin/true

2.忽略错误继续执行 ： ignore\_errors: True

添加用户，忽略错误

e.g

---

- hosts: db

remote\_user: root

vars:

un: zhang3

tasks:

- shell: adduser "{{un}}"

ignore\_errors: True

- shell: echo 123456|passwd --stdin "{{un}}"

- name: ooxx

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

Handlers与notify:

Handlers类似于函数：可以使一个handlers后面的所有动作重复执行n次

在每个标记的nodify上面进行标记，在所有的tasks的命令执行完成后才执行，

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,同一个服务叧会触发一次（如一个httpd的重启，若都写了这个那么最后就执行最后一次，等价于最后重启一次，但往往handlers中含有多个不同的触发条件，可以大大简化逻辑判断的时间）

– 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现若干服务的场景,

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収某一项操作,在特定的条件下需要提前判断上个命令的执行状态,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,进程中的系统变量 facts 变量作为when 的条件,这些 facts 我们可以通过 setup 模块

---

- hosts: other

remote\_user: root

vars:

un: lixin

tasks:

- shell: id "{{un}}"

- register: ooxx

- user:

name: "{{un}}"

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

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

when: ooxx

When结合register处理负载加大时自动关掉负载大的服务器：

uptime | Awk ‘{printf(“%.2f”,$(NF-2))}’

把负载高的web服务器停止

---

- hosts: web

remote\_user: root

tasks:

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

register: result

- service: name=httpd state=stopped

when: result.stdout|float > 0.7

测试命令 awk 'BEGIN{while(1){}}'

**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 }}获叏每次迭代的值

---

- hosts: other

remote\_user: root

tasks:

- user:

name: "{{item.name}}" --------(要求格式，类似于引用数组)

group: "{{item.group}}" ---------（通上）

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

with\_items: ------（一个数组列表）

- {name: "nb", group: "users" }

- {name: "dd", group: "mail" }

- {name: "jj", group: "root" }

- {name: "son", group: "wheel" }

with\_nested

• 嵌套循环:

---

- hosts: 192.168.1.16

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: restart web

**include and role**

• 调用方式

ansible-playbook i.yml --tags=restartweb

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指令包含进行是一个不错

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 触収条件

– files文件

– template 模板

– default

默认,优先级最低

**Ansible检测方法**

Playbook报错时是执行到哪一步是出错，不一定就是那个位置出错

调试方法

– 检测语法

ansible-playbook --syntax-check playbook.yaml

– 测试运行

ansible-playbook -C playbook.yaml

有些测试运行报错为正常（有些服务并没有实际执行起来）需要时别出真假错误

执行前面，先查看目前的情况：

– 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

执行一次后会打包一个文件，可以保存一个完整的运行信息

---

- hosts: 192.168.1.16

remote\_user: root

tasks:

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

register: result

- shell: touch /tmp/isreboot

when: result.stdout|float > 0.5 //当负载超过0.5时，将debug信息写入result

- name: Show debug info

debug: var=result //查看result中的内容，显示到命令行中