**ansible**

**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。**

**json简介**

**• json 是什么?**

**– 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 : 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 进行解析**

**• 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"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**

**{ "诗人":**

**[ {"李白":"诗仙", "年代":"唐"},**

**{"杜甫":"诗圣", "年代":"唐"},**

**{"白居易":"诗魔", "年代":"唐"},**

**{"李贺":"诗鬼", "年代":"唐"}**

**]**

**}**

**• yaml 是什么**

**– 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参考了其他多种语言,包括:C语言、Python、Perl,并从XML、电子邮件的数据格式(RFC 2822)中获得灵感。Clark Evans在2001年首次发表了这种语言[1],另外Ingy döt Net与Oren Ben-Kiki也是这语言的共同设计者[2]。目前已经有数种编程语言或脚本语言支持(或者说解析)这种语言。**

**• yaml 基础语法**

**– YAML的结构通过空格来展示**

**– 数组使用"- "来表示 后面有空格**

**– 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**

**• 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**

**• 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**

**"诗仙" : "李白"**

**– 或**

**"李白":**

**"诗仙"**

**– 复杂YAML的键值对嵌套**

**"诗人":**

**"李白": "诗仙"**

**或**

**"诗人":**

**"李白":**

**"诗仙"**

**数组**

**["李白", "杜甫", "白居易", "李贺"]**

**• YAML 数组表示方法**

**– 使用一个短横杠加一个空格**

**– YAML数组例子**

**- "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"**

**– 高级复合表达式**

**"诗人":**

**-**

**"李白": "诗仙"**

**"年代": "唐"**

**-**

**"杜甫": "诗圣"**

**"年代": "唐“**

**-**

**"白居易": "诗魔"**

**"年代": "唐"**

**-**

**"李贺": "诗鬼"**

**"年代": "唐"**

**• Jinja2是什么**

**– Jinja2是基于Python的模板引擎,包含变量和表达式两部分,两者在模板求值时会被替换为值,模板中还有标签,控制模板的逻辑**

**• 为什么要学习Jinja2模版**

**– 因为playbook的模板使用Python的Jinja2模块来处理**

**• 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**

**– 模板支持注释,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 == '诗仙' %}**

**李白**

**{% elif name == '诗圣' %}**

**杜甫**

**{% elif name == '诗魔' %}**

**白居易**

**{% else %}**

**李贺**

**{% endif %}**

**• Jinja2模版控制语句**

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

**... ...**

**{% elif name == '于谦' %}**

**{% for method in [抽烟, 喝酒, 烫头] %}**

**{{do method}}**

**{% endfor %}**

**... ...**

**{% endif %}**

**• 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( | )分割,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**

**• 例如**

**– 把一个列表用逗号连接起来:{{ list|join(', ') }}**

**– 过滤器这里不再一一列举,需要的可以查询在线文档**

**http://docs.jinkan.org/docs/jinja2/templates.html**

**#builtin-filters**

**playbook是什么**

**– playbook是ansible用于配– 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状态**

**– 也可以说,playbook字面意思即剧本,现实中由演员按剧本表演,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**

**• 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,但有时一个设施过于复杂时,执行ad-hoc命令是不合适的,最好使用playbook**

**– playbook可以反复使用编写的代码,可以放到不同的机器上面,像函数一样,最大化的利用代码,在使用ansible的过程中,处理的大部分操作都是在编写playbook**

**playbook语法基础**

**• playbook语法格式**

**– playbook由YAML语言编写,遵循YAML标准知**

**– 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,它的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,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**

**– \*\*\*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**

**– 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**

**[root@ansible ~]# vim ping.yml**

**---**

**- hosts: all**

**remote\_user: root**

**tasks:**

**- ping:**

**[root@ansible ~]# ansible-playbook ping.yml**

****

**• 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,即将一组选出的hosts映射到task,执行相同的操作**

**Tasks 下的- name 注释说明的意思**

**• playbook执行命令**

**– 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**

**---**

**- hosts: all**

**remote\_user: root**

**tasks:**

**- user:**

**name: plj**

**group: users**

**- shell: echo 1 | passwd --stdin plj**

**- shell: chage -d 0 plj**

**[root@ansible ~]# ansible-playbook an.yml**

**1. 安装Apache并修改监听端口为8080**

**2. 修改ServerName配置,执行apachectl -t命令不报错**

**3. 设置默认主页hello world**

**4. 启动服务并设开机自启**

**---**

**- hosts: all**

**remote\_user: root**

**tasks:**

**- yum:**

**name: httpd**

**- lineinfile:**

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

**regexp: Listen 80**

**line: Listen 8080**

**- lineinfile:**

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

**regexp: "#ServerName www.example.com 80"**

**line: ServerName www.example.com 8080**

**- shell : echo 11 > /var/www/html/index.htmlZZ**

**- service:**

**name: httpd**

**state: started**

**enabled: yes**

**语法进阶**

**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: all**

**remote\_user: root**

**vars:**

**username: li4**

**tasks:**

**- user:**

**name: '{{username}}'**

**group: users**

**- shell: echo 1 | passwd --stdin '{{username}}'**

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

**[root@ansible ~]# ansible-playbook an.yml -e sun 传参数 -e**

**[root@ansible ~]# vim an.yml**

**---**

**- hosts: all**

**remote\_user: root**

**vars:**

**user:**

**name: 'ss'**

**group: 'mail'**

**tasks:**

**- user:**

**name: '{{user.name}}'**

**group: '{{user.group}}'**

**- shell: echo 1 | passwd --stdin '{{user.name}}'**

**- shell: chage -d 0 '{{user.name}}'**

**[root@ansible ~]# ansible-playbook an.yml -e '{"user":{"name":"dasha","group":"root"}}'**

**[root@ansible ~]# cat usu.vars**

**user:**

**name: "xxx"**

**group: "users"**

**[root@ansible ~]# ansible-playbook an.yml -e ['@usu.vars'](mailto:'@usu.vars') 传文件**

**• 设密码**

**– 解决密码明文问题**

**– user模块的password为什么不能设置密码呢**

**– 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')}}**

**---**

**- hosts: all**

**remote\_user: root**

**vars:**

**user:**

**name: 'ss'**

**group: 'mail'**

**tasks:**

**- user:**

**name: '{{user.name}}'**

**group: '{{user.group}}'**

**password: "{{'1'|password\_hash('sha512')}}"**

**- shell: chage -d 0 '{{user.name}}'**

**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**

**True echo $?=0 flase echo $?=1**

**ignore\_errors: True 添加在可忽略错误下 只对上一条命令有效**

**• 错误处理方法**

**– 关闭selinux,如果selinux已经关闭,返回1 ,若之前**

**已经关闭则不算错误,可以忽略错误继续运行,忽略**

**错误有两种方法**

**– 第一种方式:**

**shell: /usr/bin/somecommand || /bin/true**

**– 第二种方式:**

**- name: run some command**

**shell: /usr/bin/someco**

**[root@ansible ~]# vim an.yml**

**---**

**- hosts: all**

**remote\_user: root**

**vars:**

**user:**

**name: 'ss'**

**group: 'mail'**

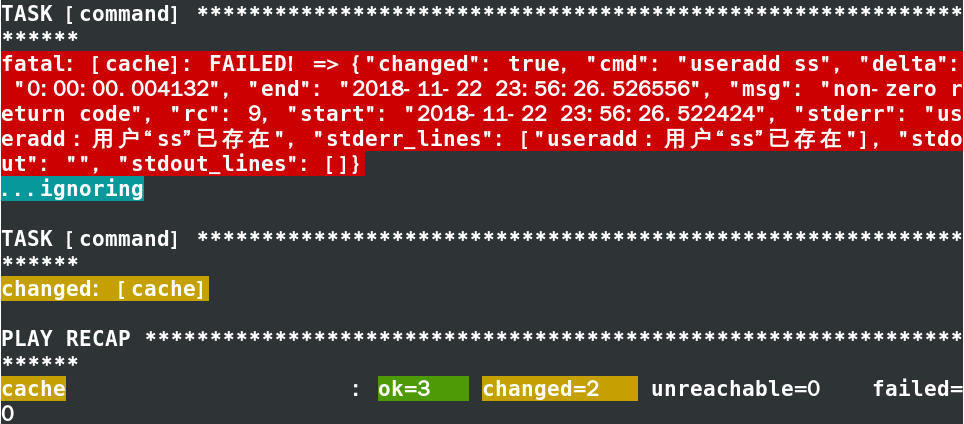
**tasks:**

**- shell: useradd {{user.name}}**

**ignore\_errors: True**

**- shell: chage -d 0 {{user.name}}**

**[root@ansible ~]# ansible-playbook an.yml**



**handlers**

**• 当关注的资源发生变化时采取的操作**

**• notify这个action可用于在每个play的最后被触发,这样可以避免有多次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,即notify调用handler中定义的操作**

**• 前面安装了Apache,修改httpd的配置文件,重新载入配置文件让服务生效**

**• 使用handlers来实现**

**handlers:**

**- name: restart apache 在handlers里面必须定义 可以理解为函数名**

**service: name=apache state=restarted**

**在执行ansible-playbook时不会执行handlers**

**在需要引用的模块下面添加**

**Notify:**

**- restart apache**

**• 注意事项:**

**– notify调用的是handler段的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**

**• 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**

**– when 的样例**

**tashs:**

**- name: somecommand**

**command: somecommand**

**when: expr**

**• register**

**– 有时候我们还需要更复杂的例子,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**

**- command: test command**

**register: result**

**- command: run command**

**when: result**

**• 变量注册**

**– 例如需要判断plj这个用户是否存在**

**– 如果存在就修改密码,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**

**变量注册进阶**

**– 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**

**---**

**- hosts: cache**

**remote\_user: root**

**tasks:**

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

**register: result**

**- shell: touch /tmp/isreboot**

**when: result.stdout|float > 0.5**

**负载超过0.5 关闭httpd**

**[root@ansible ~]# cat load.yml**

**---**

**- hosts: web**

**remote\_user: root**

**tasks:**

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

**register: result**

**- service:**

**name: httpd**

**state: stopped**

**when: result.stdout|float >0.5**

**[root@web1 ~]# awk 'BEGIN {while(1){}}' & 死循环测试增加负载**

**• with\_items是playbook标准循环,可以用于迭代一**

**个列表或字典,通过{{ item }}获取每次迭代的值**

**知**

**– 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进阶**

**– 为不同用户定义不同组**

**---**

**- hosts: 192.168.1.16**

**remote\_user: root**

**tasks:**

**- name: add users**

**user: group={{item.group}} password={{'123456' |**

**password\_hash('sha512')}} name={{item.name}}**

**with\_items:**

**- {name: 'nb', group: 'root'}**

**- {name: 'dd', group: 'root'}**

**- {name: 'plj', group: 'wheel'}**

**- {name: 'lx', 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: restartweb**

**• 调用方式**

**ansible-playbook i.yml --tags=restartweb**

**include and roles**

**• 在编写playbook的时候随着项目越来越大,playbook越来越复杂,修改也很麻烦。这时可以把一些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**

**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,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**

**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**

**– 显示受到影响的主机 --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,帮助**

**我们排错**

**• debug使用样例**

**---**

**- 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**

**- name: Show debug info**

**debug: var=result**