**playbook基础**

基础知识

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，这种习惯于使用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，科一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，不希望我其他人随意查看，ansible-valut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，队运维人员的技术水平和前瞻性规划有较高id要求

JSON简介

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

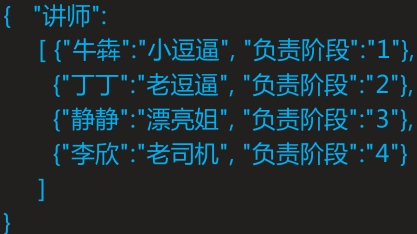
名称/值对包括字段名称（在括号中），后面写一个冒号，然后是值，例如：



json语法规则之数组



复合复杂类型



yaml简介

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参考了其它多种语言，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，另外Ingy döt Net与Oren Ben-Kiki也是这语言的共同设计者。目前已经有书中编程语言或脚本语言支持（或解析）这种语言。

yaml基本语法

YAML的结构通过空格来展示

数组使用”- ”来表示（短横线后台有一个空格）

键值使用”: ”来表示（冒号后头有一个空格）

YAML使用一个固定的缩进风格表时数据层级结构关系（空格加缩进）

一般每个缩进级别由两个以上的空格组成（不准使用tab键对齐）

# 表示注释（井号后头有空格）

注意：

不要使用tab，缩进是初学者容易出错的地方之一

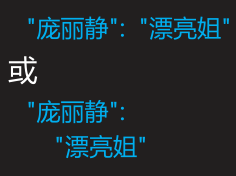
同一层级缩进必须对齐

YAML的键值表示方法

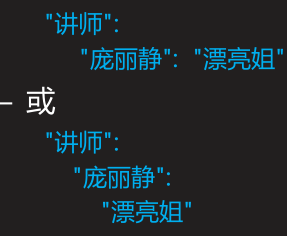
才用冒号分隔

冒号后面必须有一个空格

YAML键值对例子



复杂YAML的键值对嵌套



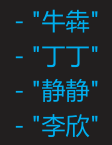
数组



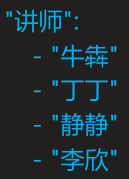
YAML数组表示方法

使用一个短横杠加一个空格

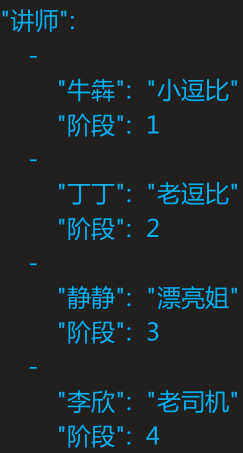
YAML数组例子



哈希数组符合表达式



高级复合表达式



yaml高级语法

|与>表示对应的值为多行字符，>与|的区别是会把\n转换为空格

!可以设置类型，!!可以强制类型转换

为了未出文件的简洁，并避免数据输入的错误，YAML提供了结点参考（\*）和散列合并（<<）参考到其他结点标签的锚点标记（&）。参考会将树状结构加入锚点标记的内容，并可以在所有数据结构中运作，合并只有散列表可以使用，可以将键值自锚点标记复制到指定的散列表中

jinja2模板简介

jinja2是什么

jinja2是基于python的模版引擎，包含变量和表达式两部分，这两者在模板求值的时候会被替换为值。模板中海油标签，控制模板的逻辑

为什么要学习jinja2模版

要使用ansible就要深入学习playbook配置及模版，playbook的模版使用jinja2模块来处理

jinja2模版基本语法

模版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，都包含在分隔符”{# #}”内，支持注释块

调用变量

{{varname}}

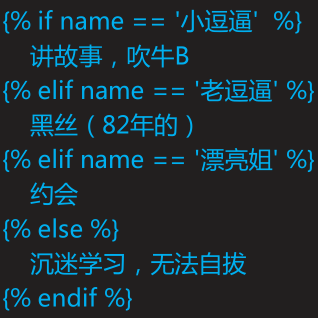
计算

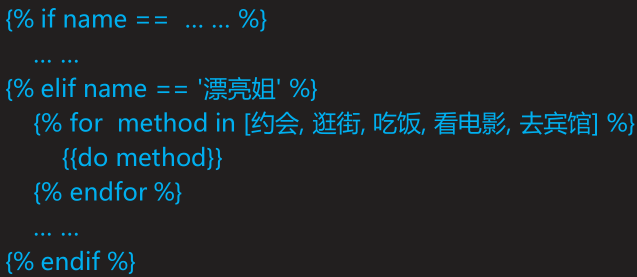
{{2+3}}

判断

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

jinja2模版控制语句





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，并且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。



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命令是不合适的，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，会发现，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以”:”分隔表示

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

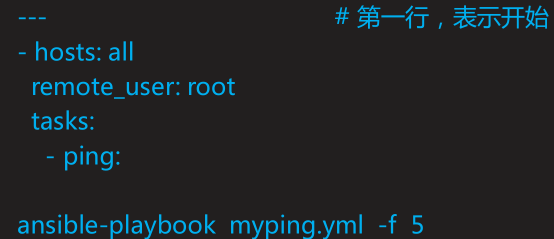
使用ansible-playbook运行playbook文件，得到输出内容为JSON格式。并且由不同颜色组成，便于识别。一般而言

绿色代表成功

\*\*\*代表系统状态发生改变

红色代表执行失败

第一个playbook



-f并发进程数量，默认是5

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

tasks

每一个play包含了一个task列表（任务列表）

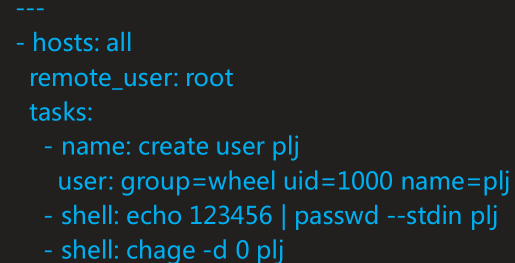
一个task在其所对应的所有主机上（通过host pattern匹配的所有主机）执行完毕后，下一个task才会执行

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

playbook执行命令

给所有主机添加用户plj，设置默认密码123456

要求第一次登陆修改密码



**playbook进阶**

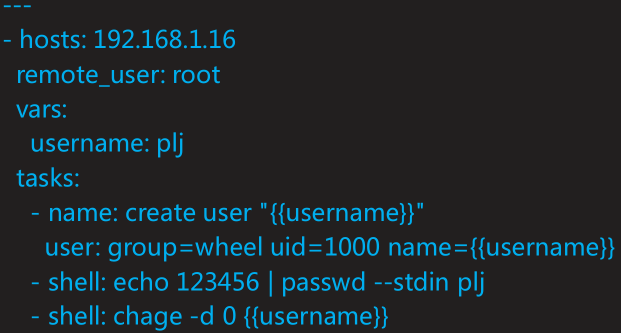
语法进阶

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）



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

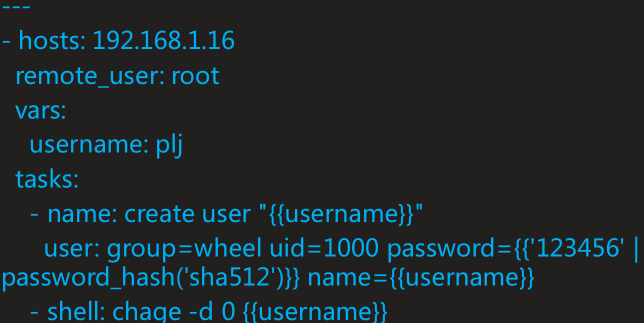
变量过滤器password hash

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

变量过滤器

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）

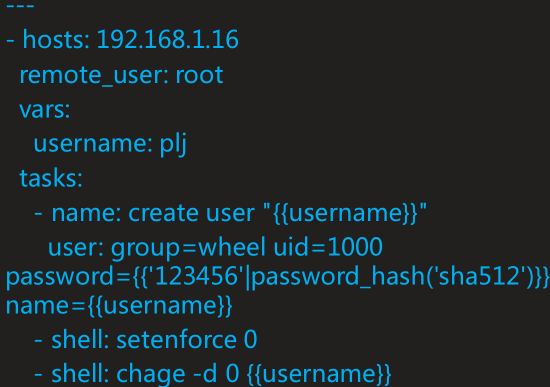


error

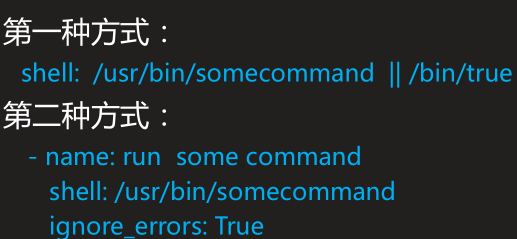
ansible-playbook对错误的处理

默认情况判断$?，如果值不为0就停止执行

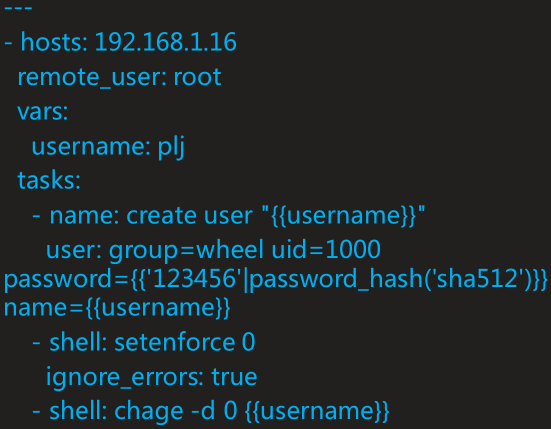
但某些情况需要忽略错误继续执行



要关闭selinux，如果selinux已经是关闭的，返回1，但目的就是关系，已经关闭不算错误，这种情况就要忽略错误继续运行，忽略错误有两种方法



完整playbook



handlers

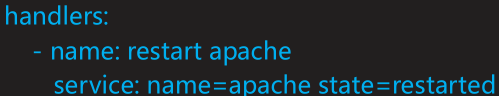
用于当关注的资源发生变化时采取一定的操作

“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

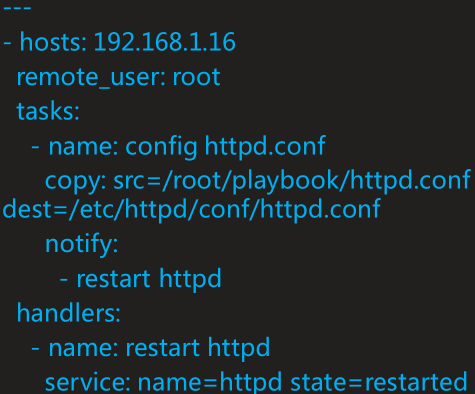
在notify中列出的操作称为Handler也即notify中调用handler中定义的操作

前面装了apache，很多情况是要修改配置文件的，修改配置文件以后要重新载入配置文件让服务生效

这时候，可以使用Handlers实现



结束之前的实验，完整的playbook



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时，同一服务只会触发一次

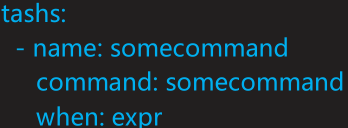
notify可以触发多个条件，在生产环境中往往涉及到某一配置文件的改变要重启若干服务的场景，Handler用到这里非常合适

结合vars可以写出非常合适的服务管理脚本

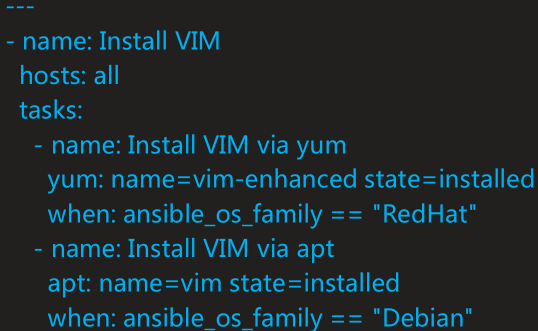
when

某些时候肯泵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U型哟啊进行条件判断，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可以通过setup模块查看

when的样例



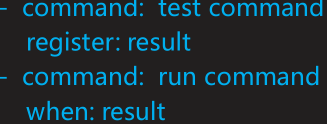
一个使用when的例子



register

register

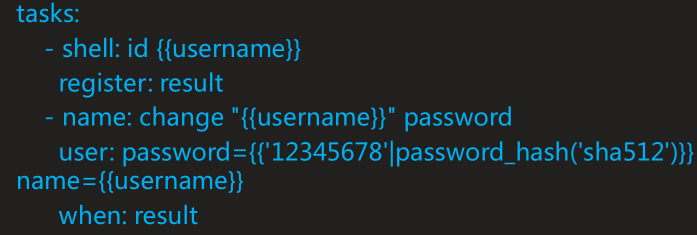
有时候可能还需要更复杂的例子，比如判断前一个命令的执行结果，根据结果处理后面的操作，这时候就需要register模块来保存前一个命令的返回状态，在后面进行调用



变量注册

如果判断plj这个用户是否存在

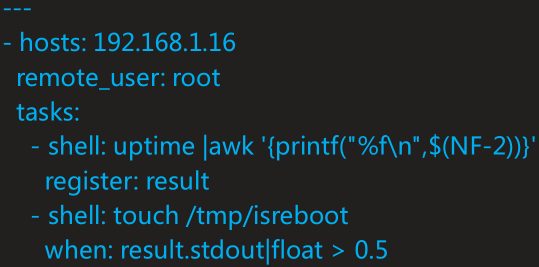
如果存在就修改密码，如果不存在就跳过



变量注册进阶

还可以针对运行命令结果的返回值做判定

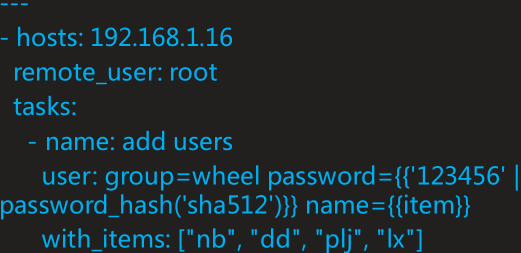
当系统负载超过一定值的时候做特殊处理



with\_items

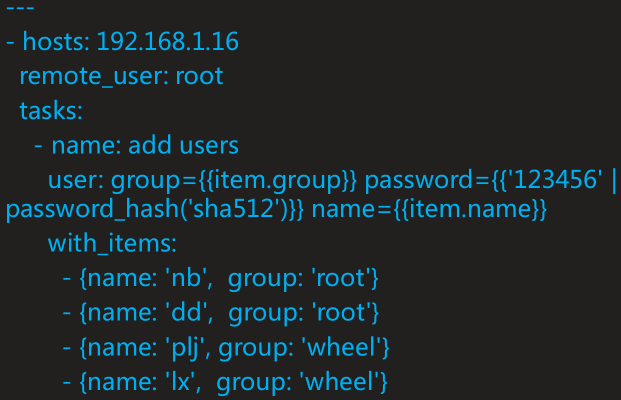
with\_items是playbook标准循环，最常用到的就是它，with\_items科一用于迭代一个列表或字典，通过{{item}}获取每次迭代的值

例如创建多个用户



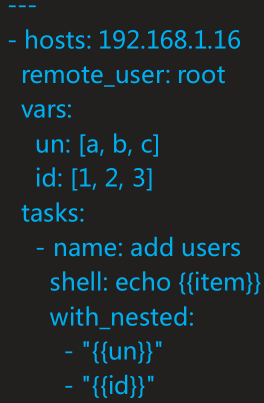
with\_items进阶

为不同用户定义不同组



with\_nested

嵌套循环



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

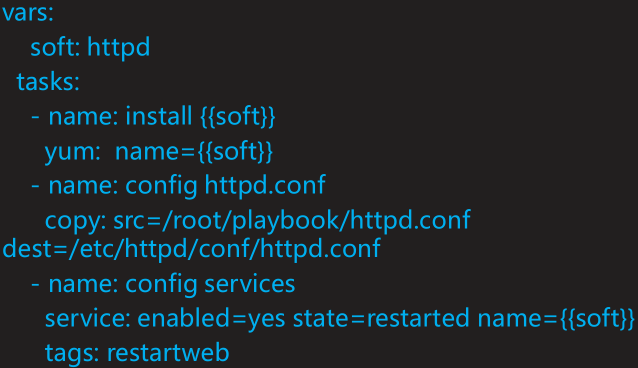
playbook调用方式

-t TAGS,--tags=TAGS

--skip-tags=SKIP\_TAGS

--start-at-task=START\_AT

tags样例

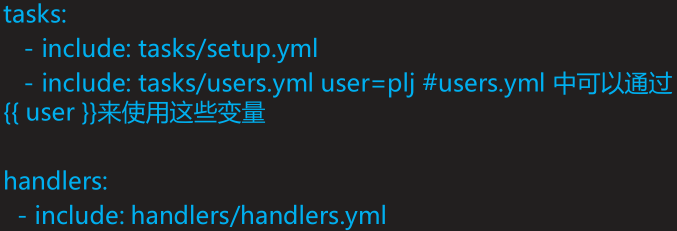


调用方式



include and roles

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



role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

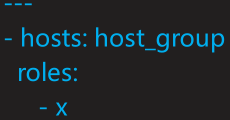
handlers 触发条件

files 文件

template模版

defaults 默认，优先级最低

假如又一个play包含了一个叫”x”的role，则



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

调试

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方法

检测语法

ansible-playbook --synta-check playbooki.yaml

测试运行

ansible-playbook -C playbook.yaml

显示收到影像的主机--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，来帮助我们排错，debug使用样例

