自动化运维工具Ansible使用介绍


qiuyi943 关注
0.571 2017.03.21 11:53:41 字数 3,934 阅读 14,780

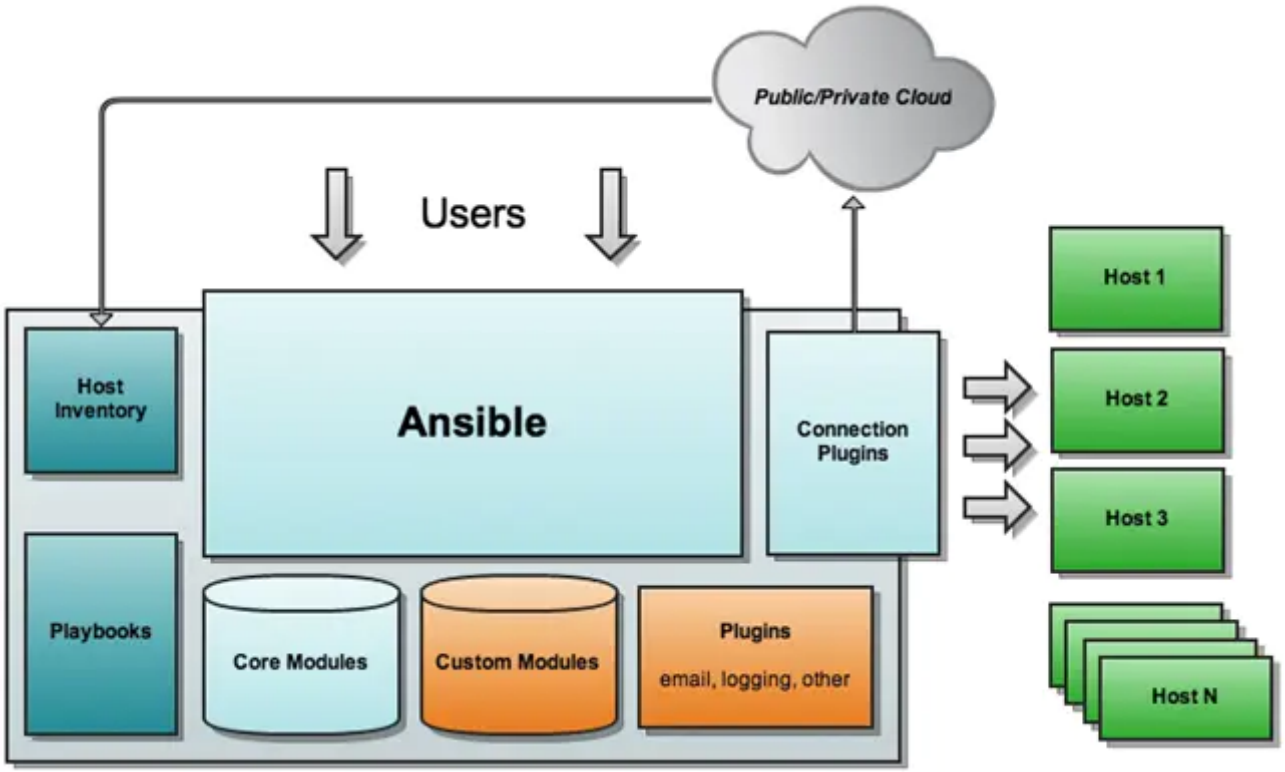
> 本文主要内容均收集于网络上的博文资料，仅以此文作为学习总结。BTW，目前Ansible对python3的支持还不是很完备。

一、基础介绍

1、 简介

ansible是新出现的自动化运维工具，基于Python开发，集合了众多运维工具（puppet、cfengine、chef、func、fabric）的优点，实现了批量系统配置、批量程序部署、批量运行命令等功能。ansible是基于模块工作的，本身没有批量部署的能力。真正具有批量部署的是ansible所运行的模块，ansible只是提供一种框架。主要包括：
(1)、连接插件connection plugins：负责和被监控端实现通信；
(2)、host inventory：指定操作的主机，是一个配置文件里面定义监控的主机；
(3)、各种模块核心模块、command模块、自定义模块；
(4)、借助于插件完成记录日志邮件等功能；
(5)、playbook：剧本执行多个任务时，非必需可以让节点一次性运行多个任务。

2、总体架构



总体架构

3、特性

(1)、no agents：不需要在被管控主机上安装任何客户端；

(2)、no server：无服务器端，使用时直接运行命令即可；

(3)、modules in any languages：基于模块工作，可使用任意语言开发模块；

(4)、yaml，not code：使用yaml语言定制剧本playbook；

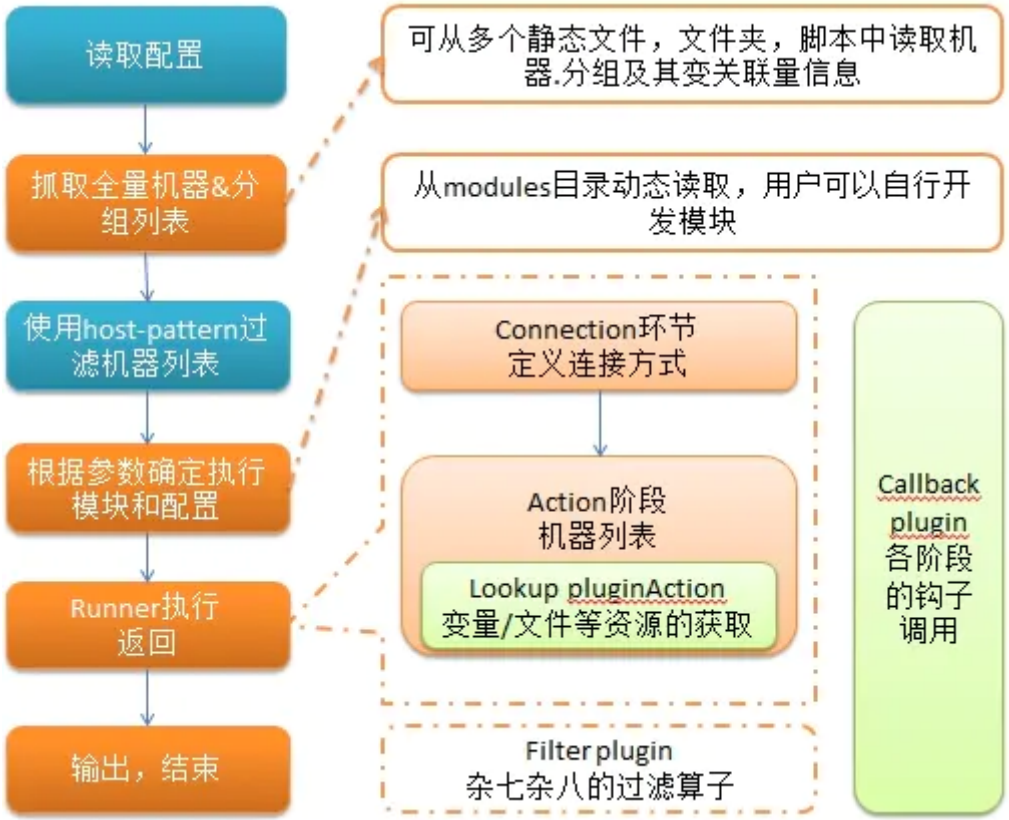
(5)、ssh by default：基于SSH工作；

(6)、strong multi-tier solution：可实现多级指挥。

4、优点

(1)、轻量级，无需在客户端安装agent，更新时，只需在操作机上进行一次更新即可；
(2)、批量任务执行可以写成脚本，而且不用分发到远程就可以执行；
(3)、使用python编写，维护更简单，ruby语法过于复杂；
(4)、支持sudo。

5、任务执行流程



任务执行流程

二、安装Ansible

- 从github下载源码：

```
$ git clone https://github.com/ansible/ansible.git --recursive
$ cd ./ansible
```

- 运行环境配置脚本：

```
$ source ./hacking/env-setup
```

- 如果没有安装pip，请先安装对应于你的Python版本的pip：

```
$ sudo easy_install pip
```

以下的Python模块也需要安装：

```
$ sudo pip install paramiko PyYAML Jinja2 httplib2 six pycrypto
```

如果安装paramiko 的时候系统报openssl相关的错误，那么需要再安装以下文件：

```
$ sudo apt-get install -y libffi-dev libssl-dev python-dev
```

当更新ansible版本时,不只要更新git的源码树，也要更新git中指向Ansible自身模块的“submodules” (不是同一种模块)：

```
$ git pull --rebase
$ git submodule update --init --recursive
```

三、Ansible 配置

1、SSH免密钥登录设置

- 使用ssh-keygen生成key-pair，然后将公钥配置在每台服务器.ssh/authorized_keys文件中：

```
# ssh-keygen -t rsa -P ''
```

- 写入信任文件（将/root/.ssh/id_rsa.pub分发到其他服务器，并在所有服务器上执行如下指令）：

```
# cat /root/.ssh/id_rsa.pub >> /root/.ssh/authorized_keys
# chmod 600 /root/.ssh/authorized_keys
```

> 如果需要配置公钥的服务器数量比较多的话，也可以通过使用ansible中的authorize_key模块来进行分布式部署。

2、Ansible配置

Ansible使用前通常需要配置/etc/anisble/路径下的hosts和ansible.cfg这个两个文件。要注意的是，编辑这两个文件需要root权限。

```
# mkdir -p /etc/ansible`
# vim /etc/ansible/ansible.cfg`
```

```
……
remote_port = 36000
private_key_file = /root/.ssh/id_rsa
……
```

Ansible 可同时操作属于一个组的多台主机，组和主机之间的关系通过 inventory 文件(INI **格式**)配置，默认的文件路径为 /etc/ansible/hosts。一个系统可以属于不同的组，比如一台服务器可以同时属于 webserver组 和 dbserver组。

```
# vim /etc/ansible/hosts
[test]
10.139.13.80
```

/etc/ansible/hosts是配置分组的默认文件，还有一种方式可以通过定义环境变量ANSIBLE_HOSTS来改变配置文件的引用路径：

```
export ANSIBLE_HOSTS=~/test/qiuyi_ansible/test_hosts
```

四、Ansible ad-hoc命令

Ansible提供两种方式去完成任务，一种是 ad-hoc 命令，另一种写 Ansible playbook。前者可以解决一些简单的任务，后者解决较复杂的任务。ad-hoc**只能做些一些轻量级的指令操作，而配置管理或部署这种事还是要借助** playbook **来完成，即使用** '/usr/bin/ansible-playbook' 这个命令。

ad-hoc的命令格式：

```
ansible <host-pattern> [options]
```

ansible有许多模块，默认是 'command'，也就是命令模块，我们可以通过 -m 选项来指定不同的模块。

常用模块：

1. setup：**用来查看远程主机的一些基本信息**

2. ping：**用来测试远程主机的运行状态**

3. file：**设置文件属性**

相关选项如下：

- force：需要在两种情况下强制创建软链接，一种是源文件不存在，但之后会建立的情况下；另一种是目标软链接已存在，需要先取消之前的软链，然后创建新的软链，有两个选项：

yes|no

- group：定义文件/目录的属组
- mode：定义文件/目录的权限
- owner：定义文件/目录的属主
- path：必选项，定义文件/目录的路径
- recurse：递归设置文件的属性，只对目录有效
- src：被链接的源文件路径，只应用于state=link的情况
- dest：被链接到的路径，只应用于state=link的情况
- state

```
state：
directory：如果目录不存在，就创建目录
file：即使文件不存在，也不会被创建
link：创建软链接
hard：创建硬链接
touch：如果文件不存在，则会创建一个新的文件，如果文件或目录已存在，则更新其最后 修改时间
absent：删除目录、文件或者取消链接文件
```

示例

- 远程文件符号链接创建

```
# ansible test -m file -a "src=/etc/resolv.conf dest=/tmp/resolv.conf state=link"
```

- 远程文件信息查看

```
# ansible test -m command -a "ls –al /tmp/resolv.conf"
```

- 远程文件符号链接删除

```
# ansible test -m file -a "path=/tmp/resolv.conf state=absent"
```

- 远程文件信息查看

```
# ansible test -m command -a "ls -al /tmp/resolv.conf"
```

4. copy：**复制文件到远程主机**

相关选项如下：

- backup：在覆盖之前，将源文件备份，备份文件包含时间信息。有两个选项：yes|no
- content：用于替代“src”，可以直接设定指定文件的值

- dest：必选项。要将源文件复制到的远程主机的绝对路径，如果源文件是一个目录，那么该路径也必须是个目录
- directory_mode：递归设定目录的权限，默认为系统默认权限
- 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
- others：所有的file模块里的选项都可以在这里使用
- src：被复制到远程主机的本地文件，可以是绝对路径，也可以是相对路径。如果路径是一个目录，它将递归复制。在这种情况下，如果路径使用“/”来结尾，则只复制目录里的内容，如果没有使用“/”来结尾，则包含目录在内的整个内容全部复制，类似于rsync。

示例：

- 将本地文件“/etc/ansible/ansible.cfg”复制到远程服务器

```
# ansible test -m copy -a "src=/etc/ansible/ansible.cfg dest=/tmp/ansible.cfg owner=root group
```

- 远程文件信息查看

```
# ansible test -m command -a "ls -al /tmp/ansible.cfg"
```

5. command：**在远程主机上执行命令**

相关选项如下：

- creates：一个文件名，当该文件存在，则该命令不执行
- free_form：要执行的linux指令
- chdir：在执行指令之前，先切换到该目录
- removes：一个文件名，当该文件不存在，则该选项不执行
- executable：切换shell来执行指令，该执行路径必须是一个绝对路径

示例：

- 如果远端服务器的~/qiuyi_test目录中已经存在1.txt文件的话，则不执行ls操作：

```
$ ansible test -m command  -a "chdir=~/qiuyi_test creates=~/qiuyi_test/1.txt ls"
10.139.13.80 | skipped
```

- 如果远端服务器的"~/qiuyi_test"目录中已经不存在2.txt文件的话，
  则在"~/qiuyi_test"路径下执行ls操作：

```
$ ansible test -m command  -a "chdir=~/qiuyi_test creates=~/qiuyi_test/2.txt ls"
10.139.13.80 | success | rc=0 >>
1.txt
qiuyi.sh
ssh
```

注意，creates选项并不会创建文件。

6. shell：**切换到某个**shell**执行指定的指令，参数与**command**相同**

与command不同的是，**此模块可以支持命令管道**，同时还有另一个模块也具备此功能：raw

示例：

```
# ansible raleigh -m shell -a "echo $TERM"
```

使用 Ansible ad-hoc 命令行接口时(与使用 Playbooks 的情况相反)，尤其注意 shell 引号的规则.比如在上面的例子中,如果使用双引号”echo $TERM”,会求出TERM变量在当前系统的值,而我们实际希望的是把这个命令传递 到其它机器执行.

示例：

- 先在本地创建一个SHELL脚本

```
$ vim /home/user_00/test/qiuyi_ansible/qiuyi_test.sh
```

```
#!/bin/sh
date +%F_%H:%M:%S
```

```
$ chmod +x /home/user_00/test/qiuyi_ansible/qiuyi_test.sh
```

- 将创建的脚本文件分发到远程

```
$ ansible test -m copy -a "src=/home/user_00/test/qiuyi_ansible/qiuyi_test.sh dest=/home/user
```

```
1 | $ ansible test -m copy -a "src=/home/user_00/test/qiuyi_ansible/qiuyi_testvim dest=/home/user
```

- 远程执行

```
1 | $ ansible test -m shell -a "~/qiuyi_test/qiuyi.sh"
2 | 10.139.13.80 | success | rc=0 >>
3 | 2017-03-22_12:27:18
```

7. **更多模块**

其他常用模块，比如：service、cron、yum、synchronize就不一一例举，可以结合自身的系统环境进行测试。在使用中，如果想查看某个模块的信息，可以使ansible-doc命令。

- service：系统服务管理
- cron：计划任务管理
- yum：yum软件包安装管理
- synchronize：使用rsync同步文件
- user：系统用户管理
- group：系统用户组管理

```
1 | # ansible-doc -l
2 | add_host             add a host (and alternatively a group) to the ansible-playbo
3 | apt                  Manages apt-packages
4 | apt_repository       Manages apt repositores
5 | assemble             Assembles a configuration file from fragments
6 | async_status         Obtain status of asynchronous task
7 | authorized_key       Adds or removes an SSH authorized key
8 | command              Executes a command on a remote node
9 | copy                 Copies files to remote locations.
```

```
cron                Manage crontab entries.
debug               Print statements during execution
easy_install        Installs Python libraries
ec2                 create an instance in ec2, return instanceid
facter              Runs the discovery program `facter' on the remote system...
fail                Fail with custom message
fetch               Fetches a file from remote nodes
file                Sets attributes of files
get_url             Downloads files from HTTP, HTTPS, or FTP to node
git                 Deploy software (or files) from git checkouts
group               Add or remove groups
group_by            Create Ansible groups based on facts
ini_file            Tweak settings in INI files
lineinfile          Ensure a particular line is in a file
mail                Send an email
mount               Control active and configured mount points
mysql_db            Add or remove MySQL databases from a remote host.
mysql_user          Adds or removes a user from a MySQL database.
nagios              Perform common tasks in Nagios related to downtime and notif
ohai                Returns inventory data from `Ohai'
pause               Pause playbook execution
ping                Try to connect to host and return `pong' on success.
pip                 Manages Python library dependencies.
postgresql_db       Add or remove PostgreSQL databases from a remote host.
postgresql_user     Adds or removes a users (roles) from a PostgreSQL database..
raw                 Executes a low-down and dirty SSH command
script              Runs a local script on a remote node
seboolean           Toggles SELinux booleans.
selinux             Change policy and state of SELinux
service             Manage services.
setup               Gathers facts about remote hosts
shell               Execute commands in nodes.
slurp               Slurps a file from remote nodes
subversion          Deploys a subversion repository.
supervisorctl       Manage the state of a program or group of programs running v
svr4pkg             Manage Solaris SVR4 packages
template            Templates a file out to a remote server.
user                Manage user accounts
virt                Manages virtual machines supported by libvirt
```

```
wait_for            Waits for a given port to become accessible on a server....
yum                 Manages packages with the `yum' package manager
```

五、 Palybooks

1、基础介绍

Playbooks与ad-hoc相比，是一种完全不同的运用 ansible 的方式。
简单来说，playbooks是一种简单的配置管理系统与多机器部署系统的基础。与现有的其他系统有不同之处，且非常适合于复杂应用的部署。Playbooks可用于声明配置，更强大的地方在于，在 playbooks中可以编排有序的执行过程，甚至于做到在多组机器间，来回有序的执行特别指定的步骤。并且可以同步或异步的发起任务。

Playbooks 采用的格式是YAML。

```yaml
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: pkg=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running
    service: name=httpd state=started
  handlers:
```

```
    - name: restart apache
      service: name=httpd state=restarted
```

- 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；
- vars为变量列表；

  在 action 行中可以使用变量。假设在 ‘vars’ 那里定义了一个变量 ‘vhost’ ， 可以这样使用它:

```
tasks:
  - name: create a virtual host file for {{ vhost }}
    template: src=somefile.j2 dest=/etc/httpd/conf.d/{{ vhost }}
```

- task的目标在于执行一个moudle；

> 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，**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。**
> 在运行playbook时（从上到下执行），如果一个host执行task失败，这个host将会从整个 playbook的rotation中移除。如果发生执行失败的情况，请修正playbook 中的错误，然后重新执行即可。每一个task必须有一个名称name，这样在运行playbook时，从其输出的任务执行信息中可以很好的辨别出是属于哪一个task的。

一个基本的tasks定义：

```
tasks:
  - name: make sure apache is running
    service: name=httpd state=running
```

- Handlers: 在发生改变时执行的操作

  这里有一个例子,当一个文件的内容被改动时,重启两个 services，‘notify’ 下列出的即是 handlers:

```
- name: template configuration file
  template: src=template.j2 dest=/etc/foo.conf
  notify:
     - restart memcached
     - restart apache
```

> Handlers 也是一些task的列表，通过名字来引用，它们和一般的 task 并没有什么区别。Handlers**是由通知者进行** notify， **如果没有被** notify，handlers **不会执行。**不管有多少个通知者进行了 notify，**等到** play **中的所有** task **执行完成之后**，handlers也只会被执行一次。handlers 会按照声明的顺序执行。

这里是一个 handlers 的示例:

```
handlers:
    - name: restart memcached
      service:  name=memcached state=restarted
    - name: restart apache
      service: name=apache state=restarted
```

Handlers 最佳的应用场景是用来重启服务，或者触发系统重启操作。除此以外很少用到了。

- 执行一个playbook

以下示例是并行的运行playbook，并行的级别是10：

```
ansible-playbook playbook.yml -f 10
```

在执行一个 playbook 之前,想看看这个 playbook 的执行会影响到哪些 hosts,你可以这样做:

```
ansible-playbook playbook.yml --list-hosts
```

playbook是由一个或多个“play”组成的列表，可以让它们联同起来按事先编排的机制执行；所谓task无非是调用ansible的一个module，而在模块参数中可以使用变量；模块执行是幂等的，这意味着多次执行是安全的，因为其结果均一致。

> **幂等**（idempotent、idempotence）是一个数学与计算机学概念，常见于抽象代数中。
> 在编程中，一个幂等操作的特点是其任意多次执行所产生的影响均与一次执行的影响相

> 同。幂等函数，或幂等方法，是指可以使用相同参数重复执行，并能获得相同结果的函数。这些函数不会影响系统状态，也不用担心重复执行会对系统造成改变。

2、Roles和include语句

假如你希望在多个play或者多个playbook中重用同一个task列表，你可以使用include 做到这一点。 当我们希望为系统定义一个角色时，使用include去包含task列表是一种很好用的方法。需要记住的是，一个play所要达成的目标是将一组系统映射为多个角色。

一个 task include file 由一个普通的 task 列表所组成，像这样:

```
---
# possibly saved as tasks/foo.yml

- name: placeholder foo
  command: /bin/foo

- name: placeholder bar
  command: /bin/bar
```

Include 指令看起来像下面这样，在一个 playbook 中，Include 指令可以跟普通的 task 混合在一起使用:

```
tasks:

  - include: tasks/foo.yml
```

Roles 基于一个已知的文件结构，去自动的加载某些 vars_files，tasks 以及 handlers。基于 roles 对内容进行分组，使得我们可以容易地与其他用户分享 roles 。

一个项目的结构如下:

```
site.yml
webservers.yml
fooservers.yml
roles/
   common/
     files/
     templates/
     tasks/
     handlers/
     vars/
     defaults/
     meta/
   webservers/
     files/
     templates/
     tasks/
     handlers/
     vars/
     defaults/
     meta/
```

一个 playbook 如下:

```
---
- hosts: webservers
  roles:
     - common
     - webservers
```

这个 playbook 为一个角色 'x' 指定了如下的行为：

- 如果 roles/x/tasks/main.yml 存在, 其中列出的 tasks 将被添加到 play 中
- 如果 roles/x/handlers/main.yml 存在, 其中列出的 handlers 将被添加到 play 中
- 如果 roles/x/vars/main.yml 存在, 其中列出的 variables 将被添加到 play 中
- 如果 roles/x/meta/main.yml 存在, 其中列出的 "角色依赖" 将被添加到 roles 列表中 (1.3 and later)
- 所有 copy tasks 可以引用 roles/x/files/ 中的文件，不需要指明文件的路径。
- 所有 script tasks 可以引用 roles/x/files/ 中的脚本，不需要指明文件的路径。
- 所有 template tasks 可以引用 roles/x/templates/ 中的文件，不需要指明文件的路径。
- 所有 include tasks 可以引用 roles/x/tasks/ 中的文件，不需要指明文件的路径。

3、小结

本节只对Playbooks的基础内容进行了简单的介绍；而余下的如Roles依赖、条件控制、循环操作、异步处理等相关内容可以参考Ansible中文权威指南进行学习，本文将不再深入介绍。

六、参考文献


```
Reference:
[1] http://www.ansible.com.cn/docs/
[2] http://sofar.blog.51cto.com/353572/1579894
[3] http://baike.baidu.com/item/ansible
```

qiuyi943

总资产2 (约0.16元)　共写了1.0W字　获得66个赞　共65个粉丝

anisble使用及常用模块

Ansible总结 ##### 运维工作: 系统安装（物理机、虚拟机）-->程序包安装、配置、服务启...

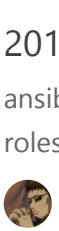
Ansible详解及模块

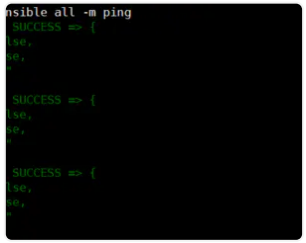
Ansible总结 运维工作: 系统安装（物理机、虚拟机）-->程序包安装、配置、服务启动 --> 对主机批量操...


二郎5　阅读 2,154　评论 0　赞 9

20171118 Ansible

ansible介绍ansible常用模块使用playbooktemplates，模板条件测试和循环迭代roles，...


哈喽别样　阅读 846　评论 0　赞 3

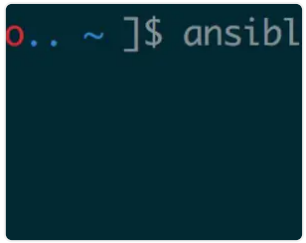


---

ansible总结

安装ansible 1.yum安装:RHEL(Centos)7版本： 2.Apt(Ubuntu)安装方式： 3.h...


爱吃土豆的程序猿　阅读 9,797　评论 0　赞 12



自动化运维工具Ansible详细部署（转）

一、基础介绍

==========...


帅T　阅读 1,122　评论 0　赞 11