

- 本地执行
- 任务委托
- 任务暂停
- 滚动执行
- 只执行一次
- 设置环境变量
- 交互式提示

## 本地执行

如果希望在控制主机本地运行一个特定的任务，可以使用local\_action语句。

假设我们需要配置的远程主机刚刚启动，如果我们直接运行playbook，可能会因为sshd服务尚未开始监听而导致失败，我们可以在控制主机上使用如下示例来等待被控端sshd端口监听：

```
- name: wait for ssh server to be running
  wait_for:
    port: 22
    host: "{{ inventory_hostname }}"
    search_regex: OpenSSH
    connection: local
```

## 任务委托

在有些时候，我们希望运行与选定的主机或主机组相关联的task，但是这个task又不需要在选定的主机或主机组上执行，而需要在另一台服务器上执行。

这种特性适用于以下场景：

- 在告警系统中启用基于主机的告警
- 向负载均衡器中添加或移除一台主机
- 在dns上添加或修改针对某个主机的解析
- 在存储节点上创建一个存储以用于主机挂载
- 使用一个外部程序来检测主机上的服务是否正常

可以使用delegate\_to语句来在另一台主机上运行task：

```
- name: enable alerts for web servers
  hosts: webservers
  tasks:
    - name: enable alerts
```

```
nagios: action=enable_alerts service=web host="{{ inventory_hostname }}"
delegate_to: nagios.example.com
```

| 如果delegate\_to: 127.0.0.1的时候，等价于local\_action

## 任务暂停

有些情况下，一些任务的运行需要等待一些状态的恢复，比如某一台主机或者应用刚刚重启，我们需要等待它上面的某个端口开启，此时就需要将正在运行的任务暂停，直到其状态满足要求。

Ansible提供了wait\_for模块以实现任务暂停的需求

wait\_for模块常用参数：

- connect\_timeout: 在下一个任务执行之前等待连接的超时时间
- delay: 等待一个端口或者文件或者连接到指定的状态时，默认超时时间为300秒，在这等待的300s的时间里，wait\_for模块会一直轮询指定的对象是否到达指定的状态，delay即为多长时间轮询一次状态。
- host: wait\_for模块等待的主机的地址，默认为127.0.0.1
- port: wait\_for模块等待的主机的端口
- path: 文件路径，只有当这个文件存在时，下一任务才开始执行，即等待该文件创建完成
- state: 等待的状态，即等待的文件或端口或者连接状态达到指定的状态时，下一个任务开始执行。当等的对象为端口时，状态有started, stopped, 即端口已经监听或者端口已经关闭；当等待的对象为文件时，状态有present或者started, absent, 即文件已创建或者删除；当等待的对象为一个连接时，状态有drained, 即连接已建立。默认为started
- timeout: wait\_for的等待的超时时间，默认为300秒

示例：

```
#等待8080端口已正常监听，才开始下一个任务，直到超时
- wait_for:
  port: 8080
  state: started

#等待8000端口正常监听，每隔10s检查一次，直至等待超时
- wait_for:
  port: 8000
  delay: 10

#等待8000端口直至有连接建立
- wait_for:
  host: 0.0.0.0
  port: 8000
  delay: 10
  state: drained

#等待8000端口有连接建立，如果连接来自10.2.1.2或者10.2.1.3，则忽略。
- wait_for:
```

```

host: 0.0.0.0
port: 8000
state: drained
exclude_hosts: 10.2.1.2,10.2.1.3

#等待/tmp/foo文件已创建
- wait_for:
  path: /tmp/foo

#等待/tmp/foo文件已创建，而且该文件中需要包含completed字符串
- wait_for:
  path: /tmp/foo
  search_regex: completed

#等待/var/lock/file.lock被删除
- wait_for:
  path: /var/lock/file.lock
  state: absent

#等待指定的进程被销毁
- wait_for:
  path: /proc/3466/status
  state: absent

#等待openssh启动, 10s检查一次
- wait_for:
  port: 22
  host: "{{ ansible_ssh_host | default(inventory_hostname) }}"
  search_regex: OpenSSH
  delay: 10

```

## 滚动执行

默认情况下，ansible会并行的在所有选定的主机或主机组上执行每一个task，但有的时候，我们会希望能够逐台运行。最典型的例子就是对负载均衡器后面的应用服务器进行更新时。通常来讲，我们会将应用服务器逐台从负载均衡器上摘除，更新，然后再添加回去。我们可以在play中使用serial语句来告诉ansible限制并行执行play的主机数量。

下面是一个在amazon EC2的负载均衡器中移除主机，更新软件包，再添加回负载均衡的配置示例：

```

- name: upgrade pkgs on servers behind load balancer
hosts: myhosts
serial: 1
tasks:
  - name: get the ec2 instance id and elastic load balancer id
    ec2_facts:

  - name: take the host out of the elastic load balancer id
    local_action: ec2_elb
    args:
      instance_id: "{{ ansible_ec2_instance_id }}"
      state: absent

```

```
- name: upgrade pkgs
  apt:
    update_cache: yes
    upgrade: yes

- name: put the host back n the elastic load balancer
  local_action: ec2_elb
  args:
    instance_id: "{{ ansible_ec2_instance_id }}"
    state: present
    ec2_elbs: "{{ items }}"
  with_items: ec2_elbs
```

在上述示例中，serial的值为1，即表示在某一个时间段内，play只在一台主机上执行。如果为2，则同时有2台主机运行play。

一般来讲，当task失败时，ansible会停止执行失败的那台主机上的任务，但是继续对其他主机执行。在负载均衡的场景中，我们会更希望ansible在所有主机执行失败之前就让play停止，否则很可能面临所有主机都从负载均衡器上摘除并且都执行失败导致服务不可用的场景。这个时候，我们可以使用serial语句配合max\_fail\_percentage语句使用。`max_fail_percentage` 表示当最大失败主机的比例达到多少时，ansible就让整个play失败。示例如下：

```
- name: upgrade pkgs on fservers behind load balancer
  hosts: myhosts
  serial: 1
  max_fail_percentage: 25
  tasks:
    ....
```

假如负载均衡后面有4台主机，并且有一台主机执行失败，这时ansible还会继续运行，要让Play停止运行，则必须超过25%，所以如果想一台失败就停止执行，我们可以将max\_fail\_percentage的值设为24。如果我们希望只要有执行失败，就放弃执行，我们可以将max\_fail\_percentage的值设为0。

## 只执行一次

某些时候，我们希望某个task只执行一次，即使它被绑定到了多个主机上。例如在一个负载均衡器后面有多台应用服务器，我们希望执行一个数据库迁移，只需要在一个应用服务器上执行操作即可。

可以使用run\_once语句来处理：

```
- name: run the database migrations
  command: /opt/run_migrations
  run_once: true
```

还可以与local\_action配合使用，如下：

```
- name: run the task locally, only once
  command: /opt/my-custom-command
  connection: local
  run_once: true
```

还可以与delegate\_to配合使用，让这个只执行一次的任务在指定的机器上运行：

```
- name: run the task locally, only once
  command: /opt/my-custom-command
  run_once: true
  delegate_to: app.a1-61-105.dev.unp
```

## 设置环境变量

我们在命令行下执行某些命令的时候，这些命令可能会需要依赖环境变量。比如在安装某些包的时候，可能需要通过代理才能完成安装。或者某个脚本可能需要调用某个环境变量才能完成运行。

ansible 支持通过 environment 关键字来定义一些环境变量。

在如下场景中可能需要用到环境变量：

- 运行shell的时候，需要设置path变量
- 需要加载一些库，这些库不在系统的标准库路径当中

下面是一个简单示例：

```
---
- name: upload a remote file to aws s3
  hosts: test
  tasks:
    - name: install pip
      yum:
        name: python-pip
        state: installed

    - name: install the aws tools
      pip:
        name: awscli
        state: present

    - name: upload file to s3
      shell: aws s3 put-object --bucket=my-test-bucket --key={{ ansible_hostname }}/fstab
      --body=/etc/fstab --region=eu-west-1
      environment:
        AWS_ACCESS_KEY_ID: xxxxxxxx
        AWS_SECRET_ACCESS_KEY: xxxxxxxx
```

事实上，environment也可以存储在变量当中：

```
- hosts: all
  remote_user: root
  vars:
    proxy_env:
      http_proxy: http://proxy.example.com:8080
      https_proxy: http://proxy.bos.example.com:8080
  tasks:
    - apt: name=cobbler state=installed
      environment: proxy_env
```

## 交互式提示

在少数情况下，ansible任务运行的过程中需要用户输入一些数据，这些数据要么比较秘密不方便，或者数据是动态的，不同的用户有不同的需求，比如输入用户自己的账户和密码或者输入不同的版本号会触发不同的后续操作等。ansible的vars\_prompt关键字就是用来处理上述这种与用户交互的情况的。

```
- hosts: all
  remote_user: root
  vars_prompt:
    - name: share_user
      prompt: "what is your network username?"
      private: yes

    - name: share_pass
      prompt: "what is your network password"
      private: yes

  tasks:
    - debug:
        var: share_user
    - debug:
        var: share_pass
```

vars\_prompt常用选项说明：

- private: 默认为yes，表示用户输入的值在命令行不可见
- default: 定义默认值，当用户未输入时则使用默认值
- confirm: 如果设置为yes，则会要求用户输入两次，适合输入密码的情况