# Red Hat Ansible Automation Platform 2.4

## Ansible Playbook 入门

ansible playbook 入门

Last Updated: 2024-06-29

# Red Hat Ansible Automation Platform 2.4 Ansible Playbook 入门

ansible playbook 入门

## 法律通告



## 摘要

本指南介绍了如何创建和使用 playbook 满足您的自动化要求。

# 目录

# 前言

感谢您对 Red Hat Ansible Automation Platform 的关注。Ansible Automation Platform 是一个商业产品，它可以帮助团队通过增加控制、知识、协调基于 Ansible 的环境来更好地管理多阶的复杂部署环境。

本指南介绍了使用 Ansible Playbook。

# 对红帽文档提供反馈

如果您对本文档有任何改进建议，或发现了任何错误，请通过 https://access.redhat.com 联系技术支持，以使用 **docs-product** 组件在 Ansible Automation Platform JIRA 项目中创建一个问题。

# 第 1 章 简介

Ansible Playbook 是自动化任务的蓝图，这是在解决方案清单期间执行有限的手动工作操作。Playbook 告知 Ansible 在哪个设备上执行什么操作。除了在 IT 环境中手动应用相同操作到数百个或数千个类似技术的操作，而是自动执行 playbook 会自动完成对指定类型的清单（如一组路由器）的相同操作。

Playbook 定期用于自动化 IT 基础架构 - 例如操作系统和 Kubernetes 平台网络、安全系统和代码存储库（如 GitHub）。您可以使用 playbook 来编程应用程序、服务、服务器节点和其他设备，而无需从头开始创建所有操作。Playbook 以及它们内的条件、变量和任务可以无限期保存、共享或重复使用。这样，您可以更轻松地协调操作知识，并确保相同的操作是一致的执行。

## 1.1. ANSIBLE PLAYBOOK 的工作原理？

Ansible Playbook 是自动为指定清单或主机组执行的任务列表。可以合并一个或多个 Ansible 任务来制作一个 play，即映射到特定主机的任务分组。

任务按照它们写入的顺序执行。

一个 playbook 可以包含一个或多个 play。

playbook 由排序列表中的一个或多个 **play** 组成。

术语 **playbook** 和 **play** 是 sports analogies。

每个 play 执行 playbook 的整体目标的一部分，运行一个或多个任务。

每个任务都调用 Ansible 模块。

**Playbook**

定义 Ansible 从上到下执行操作的 play 列表，以实现总体目标。

**Play**

映射到清单中的受管节点的任务列表。

**任务**

对定义 Ansible 执行的操作的单个模块的引用。

**角色**

角色是一种将功能放在"库"中可重复利用代码的方法，然后根据需要用于任何 playbook。

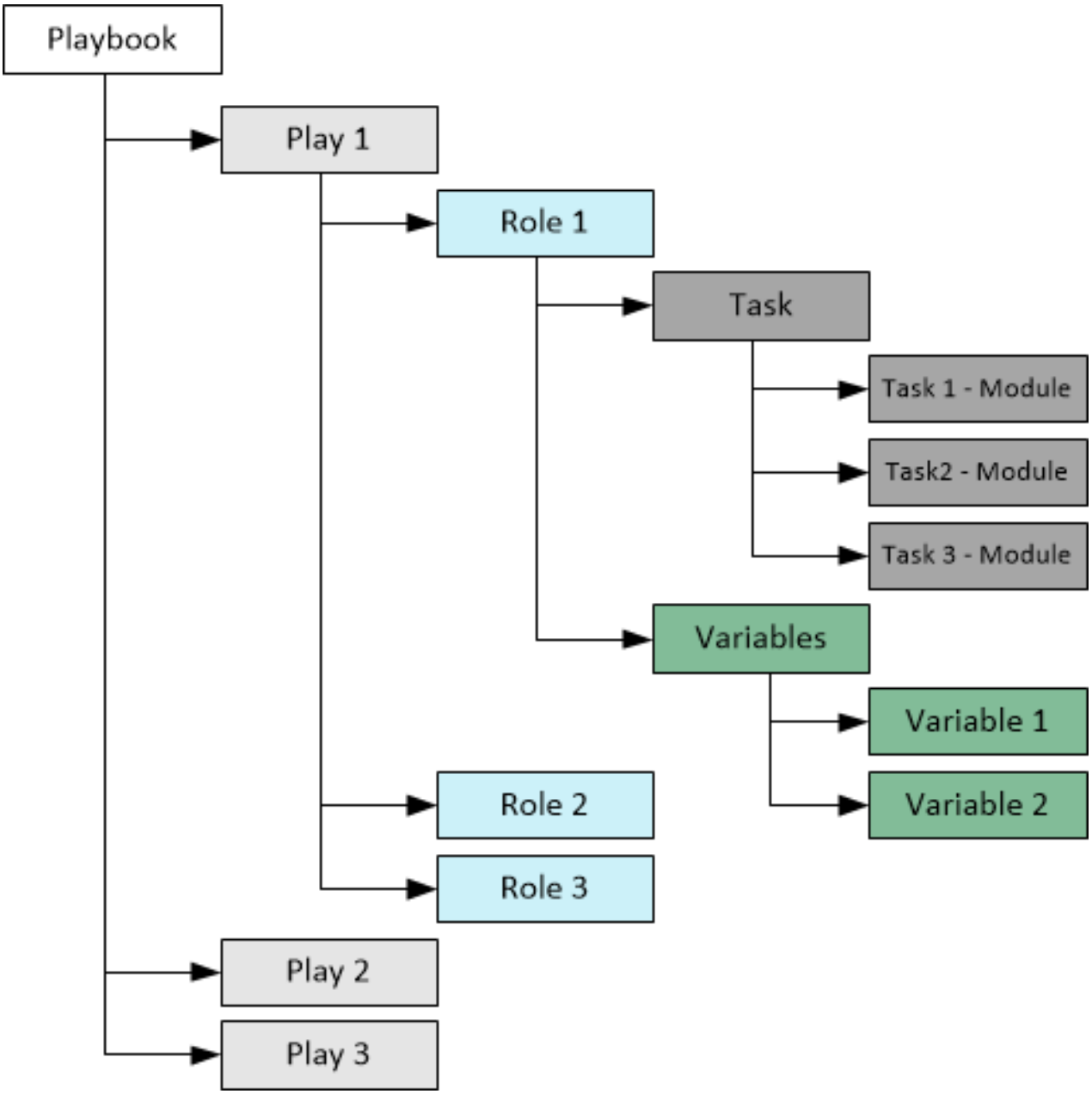
**模块**

Ansible 在受管节点上运行的代码或二进制代码单元。

Ansible 模块以集合的形式分组，*每个模块都有一个完全限定域名* (FQCN)。任务由模块执行，每个模块都在 playbook 中执行特定的任务。模块包含用于决定执行任务的时间和位置以及执行它的元数据。有数千个 Ansible 模块可以执行所有类型的 IT 任务，例如：

- 云管理
- 用户管理
- 网络
- 安全性
- 配置管理

- 通信




## 1.2. 如何使用 ANSIBLE PLAYBOOK？

Ansible 使用 YAML 语法。YAML 是一种人类可读的语言，可让您创建 playbook，而无需学习复杂的编码语言。

如需有关 YAML 的更多信息，请参阅 YAML 语法并考虑为您的文本编辑器安装附加组件，请参阅 其他工具和程序，以帮助您在 playbook 中编写干净的 YAML 语法。

使用 Ansible Playbook 的方法有两种：

- *使用命令行界面* (CLI)
- 使用 Red Hat Ansible Automation Platform 的按钮式部署。

### 1.2.1. 使用 CLI

安装开源 Ansible 项目或 Red Hat Ansible Automation Platform 后，使用以下命令

```
$ sudo dnf install ansible
```

在 Red Hat Enterprise Linux CLI 中，您可以使用 **ansible-playbook** 命令运行 Ansible Playbook。

### 1.2.2. 从平台内

Red Hat Ansible Automation Platform 用户界面提供按钮式 Ansible Playbook 部署，可用于更大作业或作业模板的一部分。这些部署附带额外的保护措施，对较新的 IT 自动化的用户特别有用，或者那些没有大量在 CLI 中工作经验的用户。

## 1.3. 使用 ANSIBLE 启动自动化

通过创建一个自动化项目、构建清单并创建 **Hello World** playbook 来开始使用 Ansible。

**先决条件**

- 必须安装 Ansible 软件包。

**流程**

- 在文件系统上创建一个项目文件夹。

```
mkdir ansible_quickstart
cd ansible_quickstart
```

通过使用单个目录结构，可以更轻松地添加到源控制，并重复使用和共享自动化内容。

## 1.4. 构建清单

清单将受管节点组织到集中文件中，该文件为 Ansible 提供系统信息和网络位置。使用清单文件，Ansible 可以通过单个命令管理大量主机。要完成以下步骤，您需要至少一个主机系统的 IP 地址或完全限定域名 (FQDN)。出于演示目的，主机可以在容器或虚拟机本地运行。

您还必须确保您的公共 SSH 密钥添加到每个主机上的 **authorized_keys** 文件中。使用以下步骤构建清单。

**流程**

在您创建的 **ansible_quickstart** 目录中，创建一个名为 **inventory.ini** 的文件。向 inventory.ini 文件中添加一个新的 **[myhosts]** 组，并指定每个主机系统的 IP 地址或完全限定域名(FQDN)。

```
[myhosts]
192.0.2.50
192.0.2.51
192.0.2.52
```

使用以下命令验证您的清单：

```
ansible-inventory -i inventory.ini --list
```

使用以下方法 ping 清单中的 **myhosts** 组：

```
ansible myhosts -m ping -i inventory.ini
```

如果在控制节点和受管节点上的用户名不同，请将 **-u** 选项与 Ansible 命令一起传递。

```
192.0.2.50 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
192.0.2.51 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
192.0.2.52 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
```

您已成功构建了一个清单。

### 1.4.1. 以 INI 或 YAML 格式的清单

您可以在 INI 文件或 YAML 中创建清单。在大多数情况下，如上例，CIN 文件对于少量受管节点来说比较简单，易于阅读。当受管节点的数量增加时，使用 YAML 格式创建清单会变得可行。

例如，以下等同于 **inventory.ini**，它为受管节点声明唯一名称，并使用 **ansible_host** 字段：

```
myhosts:
  hosts:
    my_host_01:
      ansible_host: 192.0.2.50
    my_host_02:
      ansible_host: 192.0.2.51
    my_host_03:
      ansible_host: 192.0.2.52
```

### 1.4.2. 构建清单的提示

- 确保组名称具有有意义的且唯一性。
- 组名称也区分大小写。
- 不要在组名称中使用空格、连字符或前面的数字（使用 floor_19，而不是 19th_floor）。
- 根据清单中的主机，逻辑上和何时对清单中的主机进行分组：

- what: 根据拓扑对主机进行分组，例如：db、web、leaf、spine。
- 其中：按地理位置对主机进行分组，例如： datacenter、region、floor、build。
- when ：按阶段分组主机，例如：development, test, staging, production。

### 1.4.3. 使用 metagroups

创建一个元组，以使用以下语法在清单中组织多个组：

```
metagroupname:
  children:
```

以下清单演示了数据中心的基本结构。这个示例清单包含一个网络 metagroup，其中包含所有网络设备和一个 datacenter metagroup，其中包含 network group 和 all webservers。

```
leafs:
  hosts:
    leaf01:
      ansible_host: 192.0.2.100
    leaf02:
      ansible_host: 192.0.2.110

spines:
  hosts:
    spine01:
      ansible_host: 192.0.2.120
    spine02:
      ansible_host: 192.0.2.130

network:
  children:
    leafs:
    spines:

webservers:
  hosts:
    webserver01:
      ansible_host: 192.0.2.140
    webserver02:
      ansible_host: 192.0.2.150

datacenter:
  children:
    network:
    webservers:
```

## 1.5. 创建变量

为受管节点设置值的变量，如 IP 地址、FQDN、操作系统和 SSH 用户，因此在运行 Ansible 命令时不需要传递它们。

变量可以应用到特定的主机。

```
webservers:
  hosts:
    webserver01:
      ansible_host: 192.0.2.140
      http_port: 80
    webserver02:
      ansible_host: 192.0.2.150
      http_port: 443
```

变量也可以应用到组中的所有主机。

```
webservers:
  hosts:
    webserver01:
      ansible_host: 192.0.2.140
      http_port: 80
    webserver02:
      ansible_host: 192.0.2.150
      http_port: 443
  vars:
    ansible_user: my_server_user
```

如需有关清单和 Ansible 清单变量的更多信息，请参阅关于安装程序 清单文件和 清单文件变量。

## 1.6. 创建第一个 PLAYBOOK

使用以下步骤创建 ping 主机并输出 "Hello world" 消息的 playbook。

**流程**

1. 在 **ansible_quickstart** 目录中创建一个名为 **playbook.yaml** 的文件，其内容如下：

```
- name: My first play
  hosts: myhosts
  tasks:
   - name: Ping my hosts
     ansible.builtin.ping:

   - name: Print message
     ansible.builtin.debug:
      msg: Hello world
```

2. 使用以下命令运行 playbook：

```
ansible-playbook -i inventory.ini playbook.yaml
```

3. Ansible 返回以下输出：

```
PLAY [My first play] ***********************************************************************

TASK [Gathering Facts] *********************************************************************
ok: [192.0.2.50]
ok: [192.0.2.51]
```

```
ok: [192.0.2.52]

TASK [Ping my hosts] ***********************************************************************
ok: [192.0.2.50]
ok: [192.0.2.51]
ok: [192.0.2.52]

TASK [Print message] ************************************************************************
ok: [192.0.2.50] => {
    "msg": "Hello world"
}
ok: [192.0.2.51] => {
    "msg": "Hello world"
}
ok: [192.0.2.52] => {
    "msg": "Hello world"
}

PLAY RECAP *********************************************************************************************
192.0.2.50: ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
192.0.2.51: ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
192.0.2.52: ok=3    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

在这个输出中，您可以看到：

- 提供 play 和每个任务的名称。始终使用描述性名称，使 playbook 易于验证和故障排除。
- Gather Facts 任务隐式运行。默认情况下，Ansible 会收集有关可以在 playbook 中使用的清单的信息。
- 每个任务的状态。每个任务都有一个 **ok** 状态，表示它成功运行。
- play 总结用于汇总每个主机 playbook 中所有任务的结果。在本例中，有三个任务，因此 **ok=3** 表示每个任务都成功运行。

# 第 2 章 使用 PLAYBOOK 建立与受管节点的连接

要确认您的凭证，您可以手动连接到网络设备并检索其配置。将示例用户和设备名称替换为您的实际凭证。

例如，对于 VyOS 路由器：

```
ssh my_vyos_user@vyos.example.net
show config
exit
```

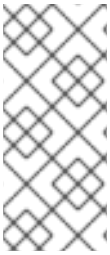
## 2.1. 运行网络 ANSIBLE 命令

您可以使用单个 Ansible 命令检索其配置，而不是在网络设备上手动连接和运行命令。

```
ansible all -i vyos.example.net, -c ansible.netcommon.network_cli -u \
my_vyos_user -k -m vyos.vyos.vyos_facts -e \
ansible_network_os=vyos.vyos.vyos
```

此命令中的标志设置七值：

- 命令应应用到的主机组（本例中为 **all**）
- 清单(**-i**、目标的设备或设备 - 没有结尾的逗号 **-i** 指向清单文件)
- 连接方法(**-c**，连接和执行 ansible 的方法)
- 用户(**-u**，SSH 连接的用户名)
- SSH 连接方法(-**k**，提示输入密码)
- 模块(**-m**，要使用的 Ansible 模块，使用完全限定的集合名称(FQCN))
- 额外的变量（本例中为 **-e**，设置网络操作系统值）



**注意**

如果您将 **ssh-agent** 与 ssh 密钥一起使用，Ansible 会自动加载它们。您可以省略 **-k** 标志。

如果您在虚拟环境中运行 Ansible，还必须添加变量 **ansible_python_interpreter=/path/to/venv/bin/python**。

## 2.2. 运行网络 ANSIBLE PLAYBOOK

如果要每天运行特定命令，可以将其保存在 playbook 中，并使用 **ansible-playbook** 而不是 ansible 运行它。该 playbook 可以存储您在命令行中提供的许多参数，从而减少在命令行中键入的内容。您需要两个文件，即 playbook 和清单文件。

**先决条件**

从此处 下载 **first_playbook.yml**。

playbook 类似如下：

```
---

- name: Network Getting Started First Playbook
  connection: ansible.netcommon.network_cli
  gather_facts: false                    1
  hosts: all
  tasks:

    - name: Get config for VyOS devices
      vyos.vyos.vyos_facts:
        gather_subset: all

    - name: Display the config
      debug:
        msg: "The hostname is {{ ansible_net_hostname }} and the OS is {{ ansible_net_version }}"
```

| 标签 | 描述 |
| --- | --- |
| **gather_facts** | Ansible 的原生事实收集(**ansible.builtin.setup**)已在此处禁用，因为 playbook 依赖于此网络集合中特定于平台的模块(**vyos.vyos_facts**)提供的事实。 |

playbook 从上面的命令行设置三个七个值：

- 组(**主机：all**)
- **连接方法（连接：ansible.netcommon.network_cli**）和
- 模块（在每个任务中）。

使用 playbook 中设置的这些值，您可以在命令行中省略它们。该 playbook 还会添加第二项任务来显示配置输出。

当从系统中收集事实时，通过特定于集合的事实模块（如 **vyos.vyos.vyos_facts** 或 **ansible.builtin.setup** ）收集到系统时，收集的数据会被保留在内存中，而不是写入到控制台。

当模块在 playbook 中运行时，输出会保留在内存中，供将来的任务使用，而不是写入控制台。在大多数其他模块中，您必须明确注册一个变量来存储和重复使用模块或任务的输出。

有关事实的更多信息，请参阅 *Ansiible Playbook 参考指南* 中的 [Ansible 事实]。

以下调试任务可让您在 shell 中看到结果。

**流程**

1. 使用以下命令运行 playbook：

```
ansible-playbook -i vyos.example.net, -u ansible -k -e ansible_network_os=vyos.vyos.vyos
first_playbook.yml
```

playbook 包含一个含有两个任务的 play，并生成如下输出：

```
$ ansible-playbook -i vyos.example.net, -u ansible -k -e ansible_network_os=vyos.vyos.vyos
```

```
first_playbook.yml

PLAY [Network Getting Started First Playbook]
**********************************************************************************************************
**********

TASK [Get config for VyOS devices]
**********************************************************************************************************
**********
ok: [vyos.example.net]

TASK [Display the config]
**********************************************************************************************************
**********
ok: [vyos.example.net] => {
    "msg": "The hostname is vyos and the OS is VyOS 1.1.8"
}
```

2. 现在，您可以检索设备配置，您可以尝试使用 Ansible 更新它。

3. 从此处 下载 **first_playbook_ext.yml**，这是第一个 playbook 的扩展版本：
playbook 类似如下：

```
---

- name: Network Getting Started First Playbook Extended
  connection: ansible.netcommon.network_cli
  gather_facts: false
  hosts: all
  tasks:

    - name: Get config for VyOS devices
      vyos.vyos.vyos_facts:
        gather_subset: all

    - name: Display the config
      debug:
        msg: "The hostname is {{ ansible_net_hostname }} and the OS is {{ ansible_net_version
}}"

    - name: Update the hostname
      vyos.vyos.vyos_config:
        backup: yes
        lines:
          - set system host-name vyos-changed

    - name: Get changed config for VyOS devices
      vyos.vyos.vyos_facts:
        gather_subset: all

    - name: Display the changed config
      debug:
        msg: "The new hostname is {{ ansible_net_hostname }} and the OS is {{
ansible_net_version }}"
```

4. 扩展的第一个 playbook 在单个 play 中有五个任务。

5. 使用以下命令运行 playbook：

```
$ ansible-playbook -i vyos.example.net, -u ansible -k -e ansible_network_os=vyos.vyos.vyos
first_playbook_ext.yml
```

6. 输出显示您对配置所做的更改：

```
$ ansible-playbook -i vyos.example.net, -u ansible -k -e ansible_network_os=vyos.vyos.vyos
first_playbook_ext.yml

PLAY [Network Getting Started First Playbook Extended]
*********************************************************************************************************
*******************

TASK [Get config for VyOS devices]
*********************************************************************************************************
*****************
ok: [vyos.example.net]

TASK [Display the config]
*********************************************************************************************************
*******************
ok: [vyos.example.net] => {
    "msg": "The hostname is vyos and the OS is VyOS 1.1.8"
}

TASK [Update the hostname]
*********************************************************************************************************
*******************
changed: [vyos.example.net]

TASK [Get changed config for VyOS devices]
*********************************************************************************************************
*******************
ok: [vyos.example.net]

TASK [Display the changed config]
*********************************************************************************************************
*******************
ok: [vyos.example.net] => {
    "msg": "The new hostname is vyos-changed and the OS is VyOS 1.1.8"
}

PLAY RECAP
*********************************************************************************************************
*******************
vyos.example.net           : ok=5    changed=1    unreachable=0    failed=0
```

## 2.3. 从网络设备收集事实

**gather_facts** 关键字支持在标准化键/值对中收集网络设备事实。您可以将这些网络事实传送到进一步的任务中，以管理网络设备。您还可以使用 **gather_network_resources** 参数和 network ***_facts** 模块（如 **arista.eos.eos_facts**）返回设备配置的子集，如下所示。

```
- hosts: arista
  gather_facts: True
  gather_subset: interfaces
  module_defaults:
    arista.eos.eos_facts:
      gather_network_resources: interfaces
```

playbook 返回以下接口事实：

```
"network_resources": {
      "interfaces": [
          {
              "description": "test-interface",
              "enabled": true,
              "mtu": "512",
              "name": "Ethernet1"
          },
          {
              "enabled": true,
              "mtu": "3000",
              "name": "Ethernet2"
          },
          {
              "enabled": true,
              "name": "Ethernet3"
          },
          {
              "enabled": true,
              "name": "Ethernet4"
          },
          {
              "enabled": true,
              "name": "Ethernet5"
          },
          {
              "enabled": true,
              "name": "Ethernet6"
          },
      ]
  }
```

**注意**

**gather_network_resources** 将配置数据呈现为所有支持资源的事实 (**interfaces/bgp/ospf/etc'**)，而 **gather_subset** 则主要用于获取操作数据。

您可以存储这些事实，并直接在另一个任务中使用它们，如 **eos_interfaces** 资源模块。

# 第 3 章 ANSIBLE PLAYBOOK 的实际示例

Ansible 可以与许多不同的设备分类通信，从基于云的 REST API 到 Linux 和 Windows 系统、网络硬件等。

以下是两个 Ansible 模块示例，它会自动更新两种类型的服务器。

## 3.1. PLAYBOOK 执行

playbook 运行从上到下的顺序。在每个 play 中，任务也从上到下的顺序运行。具有多个"play"的 playbook 可以编配多计算机部署，在您的 webservers 上运行一个 play，然后在您的数据库服务器上运行另一个 play，然后在网络基础架构上运行第三个 play，等等。

每个 play 至少定义了两个内容：

- 指向目标的受管节点，使用模式
- 至少一个要执行的任务

在 Ansible 2.10 及更高版本中，使用 playbook 中的完全限定集合名称来确保选择了正确的模块，因为多个集合可以包含具有相同名称的模块( **如用户**)。

如需更多信息，请参阅在 playbook 中使用集合。

在本例中，第一个 play 以 web 服务器为目标；第二个 play 以数据库服务器为目标。

```
---
- name: Update web servers
  hosts: webservers
  become: true

  tasks:
    - name: Ensure apache is at the latest version
      ansible.builtin.yum:
        name: httpd
        state: latest
    - name: Write the apache config file
      ansible.builtin.template:
        src: /srv/httpd.j2
        dest: /etc/httpd.conf
        mode: "0644"

- name: Update db servers
  hosts: databases
  become: true

  tasks:
    - name: Ensure postgresql is at the latest version
      ansible.builtin.yum:
        name: postgresql
        state: latest
    - name: Ensure that postgresql is started
      ansible.builtin.service:
        name: postgresql
        state: started
```

playbook 包含两个 play：

- 首先检查 Web 服务器软件是最新的，并在需要时运行更新。
- 第二个检查数据库服务器软件是否是最新的，并在需要时运行更新。

您的 playbook 不仅仅包含主机行和任务。

例如，此示例 playbook 为每个 play 设置 remote_user。这是 SSH 连接的用户帐户。您可以在 playbook、play 或任务级别添加其他 Playbook 关键字，以影响 Ansible 的行为方式。Playbook 关键字可以控制连接插件，是否使用特权升级、如何处理错误等。

为了支持各种环境，Ansible 可让您将这些参数设置为命令行标志、Ansible 配置或清单中。了解这些数据源的优先规则可帮助您扩展 Ansible 生态系统