Основы Ansible для сетевых инженеров

Natasha Samoylenko

июл. 26, 2025

Оглавление

Предупреждение: Книга не обновляется и не дописывается! Используется версия Ansible 2.9.

Ansible - это система управления конфигурациями. Ansible позволяет автоматизировать и упростить настройку, обслуживание и развертывание серверов, служб, ПО и др.

На данный момент существует несколько систем управления конфигурациями. Однако для работы с сетевым оборудованием чаще всего используется Ansible. Связано это с тем, что Ansible не требует установки агента на управляемые хосты. Особенно актуально это для устройств, которые позволяют работать с ними только через CLI.

Кроме того, Ansible активно развивается в сторону поддержки сетевого оборудования, и в нём постоянно появляются новые возможности и модули для работы с сетевым оборудованием. Некоторое сетевое оборудование поддерживает другие системы управления конфигурациями (позволяет установить агента).

Одно из важных преимуществ Ansible заключается в том, что с ним легко начать работать.

Примеры задач, которые поможет решить Ansible:

- подключение по SSH к устройствам
- параллельное подключение к устройствам по SSH (можно указывать, ко скольки устройствам подключаться одновременно)
- отправка команд на устройства
- удобный синтаксис описания устройств:
- можно разбивать устройства на группы и затем отправлять какие-то команды на всю группу
- поддержка шаблонов конфигураций с Jinja2

Это всего лишь несколько возможностей Ansible, которые относятся к сетевому оборудованию. Они перечислены для того, чтобы показать, что эти задачи Ansible сразу снимает, и можно не использовать для этого какие-то скрипты.

1

1. Основы Ansible

Ansible:

- Работает без установки агента на управляемые хосты
- Использует SSH для подключения к управляемым хостам
- Выполняет изменения с помощью модулей Python, которые выполняются на управляемых хостах
- Может выполнять действия локально на управляющем хосте
- Использует YAML для описания сценариев
- Содержит множество модулей (их количество постоянно растет)
- Можно писать свои модули

Терминология:

- **Control machine** — управляющий хост. Сервер Ansible, с которого происходит управление другими хостами
- **Manage node** — управляемые хосты
- **Inventory** — инвентарный файл. В этом файле описываются хосты, группы хостов, а также могут быть созданы переменные
- **Playbook** — файл сценариев
- **Play** — сценарий (набор задач). Связывает задачи с хостами, для которых эти задачи надо выполнить
- **Task** — задача. Вызывает модуль с указанными параметрами и переменными
- **Module** — модуль Ansible. Реализует определенные функции

Список терминов в документации.

С Ansible достаточно просто начать работать. Минимум, который нужен для начала работы:

- инвентарный файл - в нём описываются устройства
- изменить конфигурацию Ansible для работы с сетевым оборудованием
- разобраться с ad-hoc командами - это возможность выполнять простые действия с устройствами из командной строки
- например, с помощью ad-hoc команд можно отправить команду show на несколько устройств

Намного больше возможностей появится при использовании playbook (файлы сценариев). Но ad-hoc команды намного проще начать использовать. И с ними легче начать разбираться с Ansible.

Установка Ansible

Ansible нужно устанавливать только на той машине, с которой будет выполняться управление устройствами.

Требования к управляющему хосту:

- поддержка Python 3 (тестировалось на 3.7)
- Windows не может быть управляющим хостом

Примечание: В книге используется Ansible версии 2.9

Ansible довольно часто обновляется, поэтому лучше установить его в виртуальном окружении. Новые версии выходят примерно раз в полгода.

Установить Ansible можно по-разному.

С помощью pip Ansible можно установить таким образом:

```
$ pip install ansible
```

В примерах раздела используются три маршрутизатора. К ним нет никаких требований, только настроенный SSH.

Параметры, которые используются в разделе:

- пользователь: cisco
- пароль: cisco
- пароль на режим enable: cisco
- SSH версии 2
- IP-адреса:

- R1: 192.168.100.1
- R2: 192.168.100.2
- R3: 192.168.100.3

Примечание: Если вы будете использовать другие параметры, измените соответственно инвентарный файл, конфигурационный файл Ansible и файл group_vars/all.yml.

Инвентарный файл

Инвентарный файл - это файл, в котором описываются устройства, к которым Ansible будет подключаться.

Хосты и группы

В инвентарном файле устройства могут указываться используя IP-адреса или имена. Устройства могут быть указаны по одному или разбиты на группы.

Файл может быть описан в формате INI или YAML. Пример файла в формате INI:

```
r5.example.com

[cisco_routers]
192.168.255.1
192.168.255.2
192.168.255.3
192.168.255.4

[cisco_edge_routers]
192.168.255.1
192.168.255.2
```

Название, которое указано в квадратных скобках - это название группы. В данном случае, созданы две группы устройств: cisco_routers и cisco_edge_routers.

Обратите внимание, что адреса 192.168.255.1 и 192.168.255.2 находятся в двух группах. Это нормальная ситуация, один и тот же адрес или имя хоста, можно помещать в разные группы.

Таким образом можно применять отдельно какие-то политики для группы cisco_edge_routers, но в то же время, когда необходимо настроить что-то, что касается всех маршрутизаторов, можно использовать группу cisco_routers.

К разбиению на группы надо подходить внимательно. Ansible это еще и, в какой-то мере, система описания инфраструктуры. Позже мы будем рассматривать групповые переменные и роли, где значение групп будет заметно в полной мере.

По умолчанию, инвентарный файл находится в `/etc/ansible/hosts`.

При этом обычно лучше создавать свой инвентарный файл и использовать его. Для этого нужно, либо указать его при запуске ansible, используя опцию `-i <путь>`, либо указать файл в конфигурационном файле Ansible.

Если в группу надо добавить несколько устройств с однотипными именами, можно использовать такой вариант записи:

```
[cisco_routers]
192.168.255.[1:5]
```

Такая запись означает, что в группу попадут устройства с адресами 192.168.255.1-192.168.255.5. Этот формат записи поддеживается и для имен хостов:

```
[cisco_routers]
router[A:D].example.com
```

Группа из групп

Ansible также позволяет объединять группы устройств в общую группу. Для этого используется специальный синтаксис:

```
[cisco_routers]
192.168.255.1
192.168.255.2
192.168.255.3

[cisco_switches]
192.168.254.1
192.168.254.2

[cisco_devices:children]
cisco_routers
cisco_switches
```

Группы по-умолчанию

По-умолчанию, в Ansible существует две группы: all и ungrouped. Первая включает в себя все хосты, а вторая, соответственно, хосты, которые не принадлежат ни одной из групп.

Ad Hoc команды

Ad-hoc команды - это возможность запустить какое-то действие Ansible из командной строки.

Такой вариант используется, как правило, в тех случаях, когда надо что-то проверить, например, работу модуля. Или просто выполнить какое-то разовое действие, которое не нужно сохранять. В любом случае, это простой и быстрый способ начать использовать Ansible.

Сначала нужно создать в локальном каталоге инвентарный файл. Назовем его myhosts.ini:

```
[cisco_routers]
192.168.100.1
192.168.100.2
192.168.100.3
```

При подключении к устройствам первый раз, сначала лучше подключиться к ним вручную, чтобы ключи устройств были сохранены локально. В Ansible есть возможность отключить эту первоначальную проверку ключей. В разделе о конфигурационном файле мы посмотрим, как это делать (такой вариант может понадобиться, если надо подключаться к большому количеству устройств).

Пример ad-hoc команды:

```
$ ansible 192.168.100.1 -i myhosts.ini -c network_cli -e ansible_network_os=ios -u cisco -
↪k -m ios_command -a "commands='sh clock'"
```

Разберемся с параметрами команды:

- `192.168.100.1` - устройство, к которому нужно применить действия
  - эта устройство должно существовать в инвентарном файле
  - это может быть группа, конкретное имя или адрес
  - если нужно указать все хосты из файла, можно использовать значение all или *
  - Ansible поддерживает более сложные варианты указания хостов, с регулярными выражениями и разными шаблонами. Подробнее об этом в документации
- `-i myhosts.ini` - параметр -i позволяет указать инвентарный файл
- `-c network_cli` - параметр -c позволяет указать тип подключения. Тип network_cli подразумевает передачу команд через SSH имитируя человека

  - Для работы network_cli обязательно нужно указывать network_os, в данном случае, это IOS `-e ansible_network_os=ios`

- `-u cisco` - подключение выполняется от имени пользователя cisco
- `-k` - параметр, который нужно указать, чтобы аутентификация была по паролю, а не по ключам
- `-m ios_command` - параметр указывает какой модуль используется
- `-a "commands='sh ip int br'"` - параметр `-a` указывает, какую команду отправить

Примечание: Большинство параметров можно указать в интентарном файле или в файле переменных.

Результат выполнения будет таким:

```
$ ansible 192.168.100.1 -i myhosts.ini -c network_cli -e ansible_network_os=ios -u cisco -
↪k -m ios_command -a "commands='sh clock'"
```

```
SSH password:
192.168.100.1 | SUCCESS | rc=0 >>

Interface                  IP-Address      OK? Method Status                Protocol
Ethernet0/0                192.168.100.1   YES NVRAM  up                    up
Ethernet0/1                192.168.200.1   YES NVRAM  up                    up
Ethernet0/2                unassigned      YES manual administratively down down
Ethernet0/3                unassigned      YES manual up                    up
Loopback0                  10.1.1.1        YES manual up                    up
Shared connection to 192.168.100.1 closed.


192.168.100.2 | SUCCESS | rc=0 >>

Interface                  IP-Address      OK? Method Status                Protocol
Ethernet0/0                192.168.100.2   YES manual up                    up
Ethernet0/1                unassigned      YES unset  administratively down down
Ethernet0/2                192.168.200.1   YES manual administratively down down
Ethernet0/3                unassigned      YES manual up                    up
Loopback0                  10.1.1.1        YES manual up                    up
Connection to 192.168.100.2 closed by remote host.
Shared connection to 192.168.100.2 closed.


192.168.100.3 | SUCCESS | rc=0 >>

Interface                  IP-Address      OK? Method Status                Protocol
Ethernet0/0                192.168.100.3   YES manual up                    up
Ethernet0/1                unassigned      YES unset  administratively down down
Ethernet0/2                192.168.200.1   YES manual administratively down down
Ethernet0/3                unassigned      YES manual up                    up
Loopback0                  10.1.1.1        YES manual up                    up
Loopback10                 10.255.3.3      YES manual up                    up
Shared connection to 192.168.100.3 closed.
```

Теперь всё прошло успешно. Команда выполнилась, и отобразился вывод с устройства.

Аналогичным образом можно попробовать выполнять и другие команды и/или на других комбинациях устройств.

Часть параметров можно записать в инвентарный файл и тогда их не нужно будет указывать в команде:

```
[cisco_routers]
192.168.100.1
192.168.100.2
192.168.100.3

[cisco_routers:vars]
```

(continues on next page)

(продолжение с предыдущей страницы)

```
ansible_connection=network_cli
ansible_network_os=ios
ansible_user=cisco
ansible_password=cisco
```

Теперь ad-hoc команду можно вызвать так:

```
$ ansible 192.168.100.1 -i myhosts.ini -m ios_command -a "commands='sh ip int br'"
```

А результат выполнения остается тем же.

Конфигурационный файл

Настройки Ansible можно менять в конфигурационном файле.

Конфигурационный файл Ansible может храниться в разных местах (файлы перечислены в порядке уменьшения приоритета):

- ANSIBLE_CONFIG (переменная окружения)
- ansible.cfg (в текущем каталоге)
- ~/.ansible.cfg (в домашнем каталоге пользователя)
- /etc/ansible/ansible.cfg

Ansible ищет файл конфигурации в указанном порядке и использует первый найденный (конфигурация из разных файлов не совмещается).

В конфигурационном файле можно менять множество параметров. Полный список параметров и их описание можно найти в документации.

В текущем каталоге должен быть инвентарный файл myhosts.ini:

```
[cisco_routers]
192.168.100.1
192.168.100.2
192.168.100.3
```

В текущем каталоге надо создать такой конфигурационный файл ansible.cfg:

```
[defaults]

inventory = ./myhosts.ini
remote_user = cisco
ask_pass = True
```

Настройки в конфигурационном файле:

- `[defaults]` - эта секция конфигурации описывает общие параметры по умолчанию
- `inventory = ./myhosts` - параметр inventory позволяет указать местоположение инвентарного файла. Если настроить этот параметр, не придется указывать, где находится файл, при каждом запуске Ansible
- `remote_user = cisco` - от имени какого пользователя будет подключаться Ansible
- `ask_pass = True` - этот параметр аналогичен опции –ask-pass в командной строке. Если он выставлен в конфигурации Ansible, то уже не нужно указывать его в командной строке.

Теперь вызов ad-hoc команды будет выглядеть так:

```
$ ansible 192.168.100.1 -m ios_command -a "commands='sh ip int br'"
```

gathering

По умолчанию Ansible собирает факты об устройствах.

Факты - это информация о хостах, к которым подключается Ansible. Эти факты можно использовать в playbook и шаблонах как переменные.

Сбором фактов, по умолчанию, занимается модуль setup.

Но для сетевого оборудования модуль setup не подходит, поэтому сбор фактов надо отключить. Это можно сделать в конфигурационном файле Ansible или в playbook.

Примечание: Для сетевого оборудования нужно использовать отдельные модули для сбора фактов (если они есть). Это рассматривается в разделе ios_facts.

Отключение сбора фактов в конфигурационном файле:

```
gathering = explicit
```

host_key_checking

Параметр host_key_checking отвечает за проверку ключей при подключении по SSH. Если указать в конфигурационном файле `host_key_checking=False`, проверка будет отключена.

Это полезно, когда с управляющего хоста Ansible надо подключиться к большому количеству устройств первый раз.

Другие параметры конфигурационного файла можно посмотреть в документации. Пример конфигурационного файла в репозитории Ansible.

Модули Ansible

Вместе с установкой Ansible устанавливается также большое количество модулей (библиотека модулей).

Модули отвечают за действия, которые выполняет Ansible. При этом каждый модуль, как правило, отвечает за свою конкретную и небольшую задачу.

Модули можно выполнять отдельно, в ad-hoc командах или собирать в определенный сценарий (play), а затем в playbook.

Как правило, при вызове модуля ему нужно передать аргументы. Какие-то аргументы будут управлять поведением и параметрами модуля, а какие-то передавать, например, команду, которую надо выполнить.

Например, мы уже выполняли ad-hoc команды, используя модуль ios_command, и передавали ему аргументы:

```
$ ansible 192.168.100.1 -m ios_command -a "commands='sh ip int br'"
```

Выполнение такой же задачи в playbook будет выглядеть так (playbook рассматривается в следующем разделе):

```
- name: run sh ip int br
  ios_command:
    commands: show ip int br
```

После выполнения модуль возвращает результаты в формате JSON.

Модули Ansible, как правило, идемпотентны. Это означает, что модуль можно выполнять сколько угодно раз, но при этом модуль будет выполнять изменения, только если система не находится в желаемом состоянии.

В Ansible модули разделены на категории по тому кто их поддерживает:

- **core** - модули, которые поддерживает основная команда разработчиков Ansible.
- **network** - поддерживает Ansible Network Team.
- **certified** - поддерживают партнеры Ansible
- **community** - поддерживает сообщество Ansible

Также в Ansible модули разделены по функциональности. Список всех категорий находится в документации.

Особенности подключения к сетевому оборудованию

При работе с сетевым оборудованием надо указать, что должно использоваться подключение типа network_cli. Это можно указывать в инвентарном файле, файлах с переменными и т.д.

Пример настройки для сценария (play):

```
---

- name: Run show commands on routers
  hosts: cisco_routers
  connection: network_cli
```

В Ansible переменные можно указывать в разных местах, поэтому те же настройки можно указать по-другому.

Например, в инвентарном файле:

```
[cisco_routers]
192.168.100.1
192.168.100.2
192.168.100.3

[cisco_switches]
192.168.100.100

[cisco_routers:vars]
ansible_connection=network_cli
```

Или в файлах переменных, например, в group_vars/all.yml:

```
---

ansible_connection: network_cli
```

Модули, которые используются для работы с сетевым оборудованием, требуют задания нескольких параметров.

> Все описание и примеры относятся к модулям ios_x и могут отличаться для других модулей.

Для каждой задачи должны быть доступны такие параметры:

- ansible_network_os - например, ios, eos
- ansible_user - имя пользователя
- ansible_password - пароль
- ansible_become - нужно ли переходить в привилегированный режим (enable, для Cisco)

- ansible_become_method - каким образом надод переходить в привилегированный режим
- ansible_become_pass - пароль для привилегированного режима

Пример указания всех параметров в group_vars/all.yml:

```
---

ansible_connection: network_cli
ansible_network_os: ios
ansible_user: cisco
ansible_password: cisco
ansible_become: yes
ansible_become_method: enable
ansible_become_pass: cisco
```

Подготовка к работе с сетевыми модулями

В следующих разделах рассматривается работа с модулями ios_command, ios_facts и ios_config. Для того, чтобы все примеры playbook работали, надо создать несколько файлов (проверить, что они есть).

Инвентарный файл myhosts.ini:

```
[cisco_routers]
192.168.100.1
192.168.100.2
192.168.100.3
```

Конфигурационный файл ansible.cfg:

```
[defaults]

inventory = myhosts.ini
```

В файле group_vars/all.yml надо создать параметры для подключения к оборудованию:

```
---

ansible_connection: network_cli
ansible_network_os: ios
ansible_user: cisco
ansible_password: cisco
ansible_become: yes
ansible_become_method: enable
ansible_become_pass: cisco
```

2. Основы playbooks

Playbook (файл сценариев) — это файл, в котором описываются действия, которые нужно выполнить на какой-то группе хостов.

Внутри playbook:

- play - это набор задач, которые нужно выполнить для группы хостов
- task - это конкретная задача. В задаче есть как минимум:
  - описание (название задачи можно не писать, но очень рекомендуется)
  - модуль и команда (действие в модуле)

Синтаксис playbook

Playbook описываются в формате YAML.

Пример синтаксиса playbook

> Все примеры этого раздела находятся в каталоге 2_playbook_basics

Пример plabook 1_show_commands.yml:

```
---

- name: Run show commands on routers
  hosts: cisco_routers
  gather_facts: false

  tasks:
```

(continues on next page)

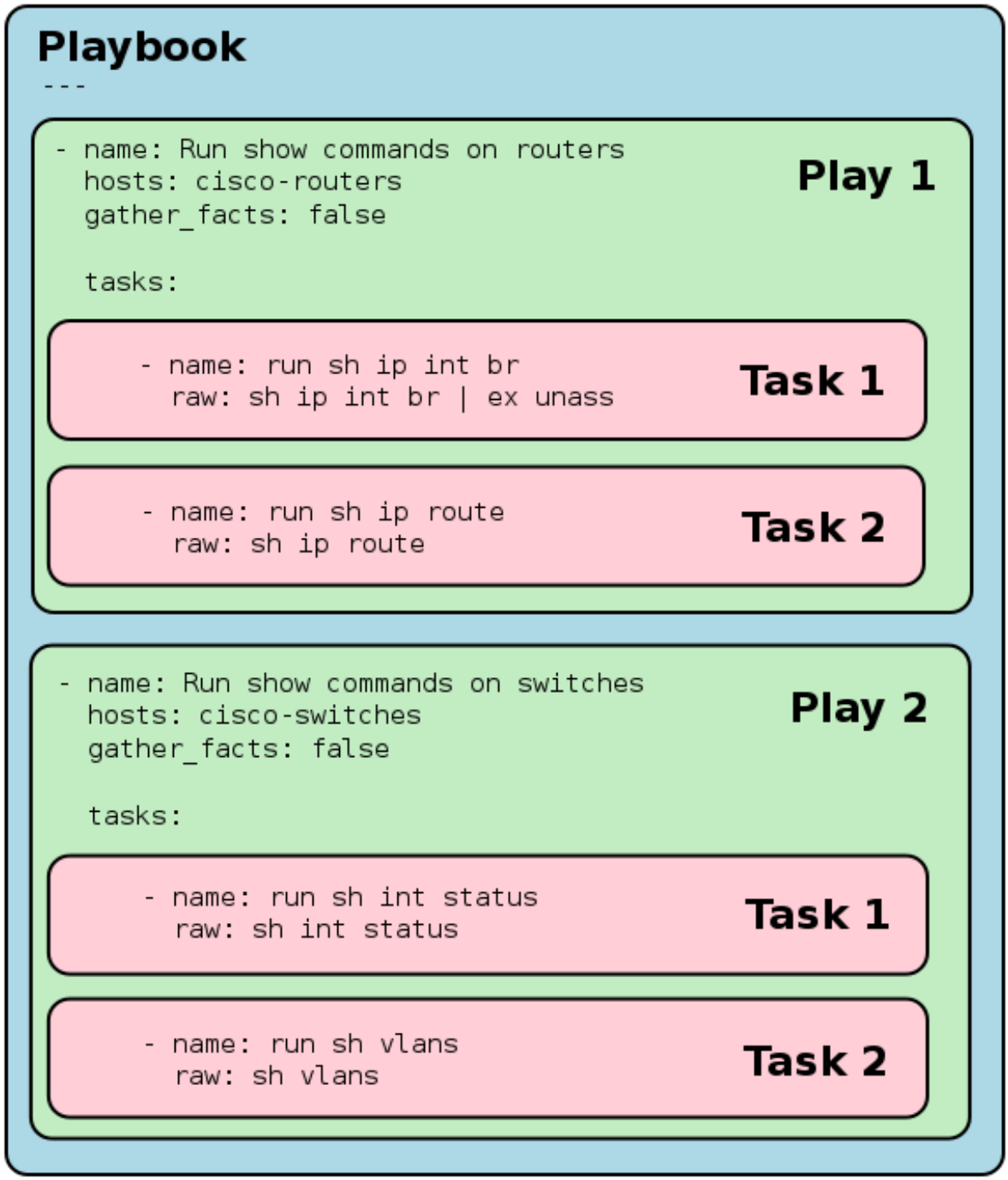
(продолжение с предыдущей страницы)

```
    - name: run sh ip int br
      ios_command:
        commands: sh ip int br

    - name: run sh ip arp
      ios_command:
        commands: sh ip arp


- name: Run command on R1
  hosts: 192.168.100.1
  gather_facts: false

  tasks:

    - name: run sh int status
      ios_command:
        commands: sh clock
```

В playbook два сценария (play):

- `name: Run show commands on routers` - имя сценария (play). Этот параметр обязательно должен быть в любом сценарии
- `hosts: cisco_routers` - сценарий будет применяться к устройствам в группе cisco-routers. Тут может быть указано и несколько групп, например, таким образом: `hosts: cisco_routers:cisco_switches`. Подробнее в документации
- `gather_facts: false` - отключение сбора фактов об устройстве, так как для сетевого оборудования надо использовать отдельные модули для сбора фактов.
  - в разделе «конфигурационный файл» рассматривалось, как отключить сбор фактов по умолчанию. Если он отключен, то параметр gather_facts в play не нужно указывать.
- `tasks:` - дальше идет перечень задач
  - в каждой задаче настроено имя (опционально) и действие. Действие может быть только одно.
  - в действии указывается, какой модуль использовать, и параметры модуля.

И тот же playbook с отображением элементов:



Так выглядит выполнение playbook:

```
$ ansible-playbook 1_show_commands.yml
```

```
PLAY [Run show commands on routers] ********************************************

TASK [run sh ip int br] ********************************************************
changed: [192.168.100.1]
changed: [192.168.100.3]
changed: [192.168.100.2]

TASK [run sh ip route] *********************************************************
changed: [192.168.100.1]
changed: [192.168.100.3]
changed: [192.168.100.2]

PLAY [Run show commands on switches] *******************************************

TASK [run sh int status] *******************************************************
changed: [192.168.100.100]

TASK [run sh vlans] ************************************************************
changed: [192.168.100.100]

PLAY RECAP *********************************************************************
192.168.100.1              : ok=2    changed=2    unreachable=0    failed=0
192.168.100.100            : ok=2    changed=2    unreachable=0    failed=0
192.168.100.2              : ok=2    changed=2    unreachable=0    failed=0
192.168.100.3              : ok=2    changed=2    unreachable=0    failed=0
```

Обратите внимание, что для запуска playbook используется другая команда. Для ad-hoc команды использовалась команда ansible. А для playbook - ansible-playbook.

Для того, чтобы убедиться, что команды, которые указаны в задачах, выполнились на устройствах, запустите playbook с опцией -v (вывод сокращен):

```
$ ansible-playbook 1_show_commands.yml -v
```

```
SSH password:

PLAY [Run show commands on routers] ********************************************

TASK [run sh ip int br] ********************************************************
changed: [192.168.100.1] => {"changed": true, "rc": 0, "stderr": "Shared connection
 to 192.168.100.1 closed.\r\n", "stdout": "\r\nInterface                  IP-Addres
s      OK? Method Status                Protocol\r\nEthernet0/0                192.
168.100.1   YES NVRAM  up                    up      \r\nEthernet0/1
 192.168.200.1   YES NVRAM  up                    up      \r\nLoopback0
      10.1.1.1        YES manual up                    up      \r\n", "stdout_lines
": ["", "Interface                  IP-Address      OK? Method Status
  Protocol", "Ethernet0/0                192.168.100.1   YES NVRAM  up
       up       ", "Ethernet0/1                192.168.200.1   YES NVRAM  up
            up       ", "Loopback0                  10.1.1.1        YES manual up
                 up       "]}
```

В следующих разделах мы научимся отображать эти данные в нормальном формате и посмотрим, что с ними можно делать.

Порядок выполнения задач и сценариев

Сценарии (play) и задачи (task) выполняются последовательно, в том порядке, в котором они описаны в playbook.

Если в сценарии, например, две задачи, то сначала первая задача должна быть выполнена для всех устройств, которые указаны в параметре hosts. Только после того, как первая задача была выполнена для всех хостов, начинается выполнение второй задачи.

Если в ходе выполнения playbook возникла ошибка в задаче на каком-то устройстве, это устройство исключается, и другие задачи на нём выполняться не будут.

Например, заменим пароль пользователя cisco на cisco123 (правильный cisco) на маршрутизаторе 192.168.100.1 и запустим playbook заново:

```
$ ansible-playbook 1_show_commands.yml
```

```
SSH password:

PLAY [Run show commands on routers] ********************************************************************

TASK [run sh ip int br] ********************************************************************************
changed: [192.168.100.2]
changed: [192.168.100.3]
fatal: [192.168.100.1]: FAILED! => {"changed": true, "failed": true, "msg": "non-zero return code", "rc": 5,
"stderr": "", "stdout": "", "stdout_lines": []}

TASK [run sh ip route] *********************************************************************************
changed: [192.168.100.2]
changed: [192.168.100.3]

PLAY [Run show commands on switches] *******************************************************************

TASK [run sh int status] *******************************************************************************
changed: [192.168.100.100]

TASK [run sh vlan] *************************************************************************************
changed: [192.168.100.100]
        to retry, use: --limit @/home/vagrant/repos/pyneng-examples-exercises/examples/15_ansible/2_playbook_
basics/1_show_commands_with_raw.retry

PLAY RECAP *********************************************************************************************
192.168.100.1              : ok=0    changed=0    unreachable=0    failed=1
192.168.100.100            : ok=2    changed=2    unreachable=0    failed=0
192.168.100.2              : ok=2    changed=2    unreachable=0    failed=0
192.168.100.3              : ok=2    changed=2    unreachable=0    failed=0
```

Обратите внимание на ошибку в выполнении первой задачи для маршрутизатора 192.168.100.1.

Во второй задаче „TASK [run sh ip route]“, Ansible уже исключил маршрутизатор и выполняет задачу только для маршрутизаторов 192.168.100.2 и 192.168.100.3.

Параметр –limit позволяет ограничивать, для каких хостов или групп будет выполняться playbook, при этом не меняя сам playbook.

Например, таким образом playbook можно запустить только для маршрутизатора 192.168.100.1:

```
$ ansible-playbook 1_show_commands.yml --limit 192.168.100.1
```

Идемпотентность

Модули Ansible идемпотентны. Это означает, что модуль можно выполнять сколько угодно раз, но при этом модуль будет выполнять изменения, только если система не находится в желаемом состоянии.

Из этого правила есть исключения. Например, модуль raw всегда вносит изменения.

Если, например, в задаче указано, что на сервер Linux надо установить пакет httpd, то он будет установлен только в том случае, если его нет. То есть, действие не будет повторяться снова и снова при каждом запуске, а лишь тогда, когда пакета нет.

Аналогично и с сетевым оборудованием. Если задача модуля - выполнить команду в конфигурационном режиме, а она уже есть на устройстве, модуль не будет вносить изменения.

Переменные

Переменной может быть, например:

- информация об устройстве, которая собрана как факт, а затем используется в шаблоне.
- в переменные можно записывать полученный вывод команды.
- переменная может быть указана вручную в playbook

Имена переменных

В Ansible есть определенные ограничения по формату имен переменных:

- Переменные могут состоять из букв, чисел и символа _
- Переменные должны начинаться с буквы

Кроме того, можно создавать словари с переменными (в формате YAML):

```
R1:
  IP: 10.1.1.1/24
  DG: 10.1.1.100
```

Обращаться к переменным в словаре можно двумя вариантами:

```
R1['IP']
R1.IP
```

Правда, при использовании второго варианта могут быть проблемы, если название ключа совпадает с зарезервированным словом (методом или атрибутом) в Python или Ansible.

Где можно определять переменные

Переменные можно создавать:

- в инвентарном файле
- в playbook
- в специальных файлах для группы/устройства
- в отдельных файлах, которые добавляются в playbook через include (как в Jinja2)

- в ролях, которыезатем используются
- можно даже передавать переменные при вызове playbook

Также можно использовать факты, которые были собраны про устройство, как переменные.

Переменные в инвентарном файле

В инвентарном файле можно указывать переменные для группы:

```
[cisco_routers]
192.168.100.1
192.168.100.2
192.168.100.3

[cisco_switches]
192.168.100.100

[cisco_routers:vars]
ntp_server=192.168.255.100
log_server=10.255.100.1
```

Переменные ntp_server и log_server относятся к группе cisco_routers и могут использоваться, например, при генерации конфигурации на основе шаблона.

Переменные в playbook

Переменные можно задавать прямо в playbook. Это может быть удобно тем, что переменные находятся там же, где все действия.

Например, можно задать переменную interfaces в playbook таким образом:

```
---

- name: Run show commands on routers
  hosts: cisco_routers
  gather_facts: false


  vars:
    interfaces: sh ip int br

  tasks:

    - name: run sh ip int br
      ios_command:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      commands: "{{interfaces}}"

  - name: run sh ip arp
    ios_command:
      commands: sh ip arp
```

Переменные в специальных файлах для группы/устройства

Ansible позволяет хранить переменные для группы/устройства в специальных файлах:

- Для групп устройств, переменные должны находиться в каталоге group_vars, в файлах, которые называются, как имя группы. Кроме того, можно создавать в каталоге group_vars файл all, в котором будут находиться переменные, которые относятся ко всем группам.
- Для конкретных устройств, переменные должны находиться в каталоге host_vars, в файлах, которые соответствуют имени или адресу хоста.

  *Все файлы с переменными должны быть в формате YAML. Расширение файла может быть таким: yml, yaml, json или без расширения
- каталоги group_vars и host_vars должны находиться в том же каталоге, что и playbook, или могут находиться внутри каталога inventory (первый вариант более распространенный). Если каталоги и файлы названы правильно и расположены в указанных каталогах, Ansible сам распознает файлы и будет использовать переменные.

Например, если инвентарный файл myhosts.ini выглядит так:

```
[cisco_routers]
192.168.100.1
192.168.100.2
192.168.100.3

[cisco_switches]
192.168.100.100
```

Можно создать такую структуру каталогов:

```
├── group_vars                  _
│   ├── all.yml                  |
│   ├── cisco_routers.yml        |  Каталог с переменными для групп устройств
│   └── cisco_switches.yml      _|
|
├── host_vars                   _
│   ├── 192.168.100.1            |
│   ├── 192.168.100.2            |
```

(continues on next page)

(продолжение с предыдущей страницы)

```
|   ├── 192.168.100.3           |  Каталог с переменными для устройств
|   └── 192.168.100.100        _|
|
└── myhosts.ini                 |  Инвентарный файл
```

Ниже пример содержимого файлов переменных для групп устройств и для отдельных хостов.

group_vars/all.yml (в этом файле указываются значения по умолчанию, которые относятся ко всем устройствам):

```
---

ansible_connection: network_cli
ansible_network_os: ios
ansible_user: cisco
ansible_password: cisco
ansible_become: yes
ansible_become_method: enable
ansible_become_pass: cisco
```

В данном случае указываются переменные, которые предопределены самим Ansible.

group_vars/cisco_routers.yml

```
---

log_server: 10.255.100.1
ntp_server: 10.255.100.1
users:
  user1: pass1
  user2: pass2
  user3: pass3
```

В файле group_vars/cisco_routers.yml находятся переменные, которые указывают IP-адреса Log и NTP серверов и нескольких пользователей. Эти переменные могут использоваться, например, в шаблонах конфигурации.

group_vars/cisco_switches.yml

```
---

vlans:
  - 10
  - 20
  - 30
```

В файле group_vars/cisco_switches.yml указана переменная vlans со списком VLANов.

Файлы с переменными для хостов однотипны, и в них меняются только адреса и имена:

Файл host_vars/192.168.100.1.yml

```
---

hostname: london_r1
mgmnt_loopback: 100
mgmnt_ip: 10.0.0.1
ospf_ints:
  - 192.168.100.1
  - 10.0.0.1
  - 10.255.1.1
```

Приоритет переменных

Примечание: В этом разделе не рассматривается размещение переменных:

- в отдельных файлах, которые добавляются в playbook через include (как в Jinja2)
- в ролях, которые затем используются
- передача переменных при вызове playbook

Чаще всего, переменная с определенным именем только одна, но иногда может понадобиться создать переменную в разных местах, и тогда нужно понимать, в каком порядке Ansible перезаписывает переменные.

Приоритет переменных (последние значения переписывают предыдущие):

- переменные в инвентарном файле
- переменные для группы хостов в инвентарном файле
- переменные для хостов в инвентарном файле
- переменные в каталоге group_vars
- переменные в каталоге host_vars
- факты хоста
- переменные сценария (play)
- переменные, полученные через параметр register
- переменные, которые передаются при вызове playbook через параметр –extra-vars (всегда наиболее приоритетные)

Более полный список в документации

Работа с результатами выполнения модуля

В этом разделе рассматриваются несколько способов, которые позволяют посмотреть на вывод, полученный с устройств.

Примеры используют модуль raw, но аналогичные принципы работают и с другими модулями.

verbose

В предыдущих разделах один из способов отобразить результат выполнения команд уже использовался - флаг verbose.

Конечно, вывод не очень удобно читать, но, как минимум, он позволяет увидеть, что команды выполнились. Также этот флаг позволяет подробно посмотреть, какие шаги выполняет Ansible.

Пример запуска playbook с флагом verbose (вывод сокращен):

```
ansible-playbook 1_show_commands_with_raw.yml -v
```

```
SSH password:

PLAY [Run show commands on routers] ****************************************

TASK [run sh ip int br] ****************************************************
changed: [192.168.100.1] => {"changed": true, "rc": 0, "stderr": "Shared connection
 to 192.168.100.1 closed.\r\n", "stdout": "\r\nInterface                  IP-Addres
s      OK? Method Status                Protocol\r\nEthernet0/0                192.
168.100.1   YES NVRAM  up                    up      \r\nEthernet0/1
 192.168.200.1   YES NVRAM  up                    up      \r\nLoopback0
      10.1.1.1        YES manual up                    up      \r\n", "stdout_lines
": ["", "Interface                  IP-Address      OK? Method Status
  Protocol", "Ethernet0/0                192.168.100.1   YES NVRAM  up
       up       ", "Ethernet0/1                192.168.200.1   YES NVRAM  up
            up       ", "Loopback0                10.1.1.1        YES manual up
                 up       "]}
```

При увеличении количества букв v в флаге, вывод становится более подробным. Попробуйте вызывать этот же playbook и добавлять к флагу буквы v (5 и больше показывают одинаковый вывод) таким образом:

```
ansible-playbook 1_show_commands_with_raw.yml -vvv
```

В выводе видны результаты выполнения задачи, они возвращаются в формате JSON:

- **changed** - ключ, который указывает, были ли внесены изменения

- **stdout** - вывод команды
- **stdout_lines** - вывод в виде списка команд, разбитых построчно

register

Параметр **register** сохраняет результат выполнения модуля в переменную. Затем эта переменная может использоваться в шаблонах, в принятии решений о ходе сценария или для отображения вывода.

Попробуем сохранить результат выполнения команды.

В playbook 2_register_vars.yml с помощью register вывод команды sh ip int br сохранен в переменную sh_ip_int_br_result:

```
---

- name: Run show commands on routers
  hosts: 192.168.100.1
  gather_facts: false

  tasks:

    - name: run sh ip int br
      ios_command:
        commands: sh ip int br
      register: sh_ip_int_br_result
```

Если запустить этот playbook, вывод не будет отличаться, так как вывод только записан в переменную, но с переменной не выполняется никаких действий. Следующий шаг - отобразить результат выполнения команды с помощью модуля debug.

debug

Модуль debug позволяет отображать информацию на стандартный поток вывода. Это может быть произвольная строка, переменная, факты об устройстве.

Для отображения сохраненных результатов выполнения команды, в playbook 2_register_vars.yml добавлена задача с модулем debug:

```
---

- name: Run show commands on routers
  hosts: 192.168.100.1
  gather_facts: false
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  tasks:

    - name: run sh ip int br
      ios_command:
        commands: sh ip int br
      register: sh_ip_int_br_result

    - name: Debug registered var
      debug: var=sh_ip_int_br_result.stdout_lines
```

Обратите внимание, что выводится не всё содержимое переменной sh_ip_int_br_result, а только содержимое stdout_lines. В sh_ip_int_br_result.stdout_lines находится список строк, поэтому вывод будет структурирован.

Результат запуска playbook выглядит так:

```
$ ansible-playbook 2_register_vars.yml
```

```
SSH password:

PLAY [Run show commands on routers] ****************************************

TASK [run sh ip int br] ****************************************************
changed: [192.168.100.1]
changed: [192.168.100.2]
changed: [192.168.100.3]

TASK [Debug registered var] ************************************************
ok: [192.168.100.1] => {
    "sh_ip_int_br_result.stdout_lines": [
        "",
        "Interface                  IP-Address      OK? Method Status                Protocol",
        "Ethernet0/0                192.168.100.1   YES NVRAM  up                    up      ",
        "Ethernet0/1                192.168.200.1   YES NVRAM  up                    up      ",
        "Loopback0                  10.1.1.1        YES manual up                    up      "
    ]
}
ok: [192.168.100.2] => {
    "sh_ip_int_br_result.stdout_lines": [
        "",
        "Interface                  IP-Address      OK? Method Status                Protocol",
        "Ethernet0/0                192.168.100.2   YES manual up                    up      ",
        "Ethernet0/2                192.168.200.1   YES manual administratively down down    ",
        "Loopback0                  10.1.1.1        YES manual up                    up      "
    ]
}
ok: [192.168.100.3] => {
    "sh_ip_int_br_result.stdout_lines": [
        "",
        "Interface                  IP-Address      OK? Method Status                Protocol",
        "Ethernet0/0                192.168.100.3   YES manual up                    up      ",
        "Ethernet0/2                192.168.200.1   YES manual administratively down down    ",
        "Loopback0                  10.1.1.1        YES manual up                    up      ",
        "Loopback10                 10.255.3.3      YES manual up                    up      "
    ]
}

PLAY RECAP *****************************************************************
192.168.100.1              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.3              : ok=2    changed=1    unreachable=0    failed=0
```

register, debug, when

С помощью ключевого слова **when** можно указать условие, при выполнении которого задача выполняется. Если условие не выполняется, то задача пропускается.

Примечание: when в Ansible используется, как if в Python.

Пример playbook 3_register_debug_when.yml:

```
---

- name: Run show commands on routers
  hosts: 192.168.100.1
  gather_facts: false

  tasks:

    - name: run sh ip int br
      ios_command:
        commands: sh ip int br
      register: sh_ip_int_br_result

    - name: Debug registered var
      debug:
        msg: "IP адрес не найден"
      when: "'4.4.4.4' not in sh_ip_int_br_result.stdout[0]"
```

В последнем задании несколько изменений:

- модуль debug отображает не содержимое сохраненной переменной, а сообщение, которое указано в переменной msg.
- условие when указывает, что данная задача выполнится только при выполнении условия
- `when: "'4.4.4.4' not in sh_ip_int_br_result.stdout[0]"` - это условие означает, что задача будет выполнена только в том случае, если в выводе sh_ip_int_br_result.stdout будет найдена строка 4.4.4.4

Выполнение playbook:

```
$ ansible-playbook 3_register_debug_when.yml
```

```
SSH password:

PLAY [Run show commands on routers] ********************************************

TASK [run sh ip int br] ********************************************************
changed: [192.168.100.2]
changed: [192.168.100.1]
changed: [192.168.100.3]

TASK [Debug registered var] ****************************************************
skipping: [192.168.100.1]
skipping: [192.168.100.2]
skipping: [192.168.100.3]

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=1    unreachable=0    failed=0
```

Обратите внимание на сообщения skipping - это означает, что задача не выполнялась для указанных устройств. Не выполнилась она потому, что условие в when не было выполнено.

Выполнение того же playbook, но после удаления адреса на устройстве:

```
$ ansible-playbook 3_register_debug_when.yml
```

```
SSH password:

PLAY [Run show commands on routers] ********************************************

TASK [run sh ip int br] ********************************************************
changed: [192.168.100.1]
changed: [192.168.100.2]
changed: [192.168.100.3]

TASK [Debug registered var] ****************************************************
ok: [192.168.100.1] => {
    "msg": "Error in command"
}
ok: [192.168.100.2] => {
    "msg": "Error in command"
}
ok: [192.168.100.3] => {
    "msg": "Error in command"
}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.3              : ok=2    changed=1    unreachable=0    failed=0
```

3

3. Сетевые модули привязанные к конкретной ОС

В этом разделе рассматриваются модули, которые работают с сетевым оборудованием через CLI. Глобально модули для работы с сетевым оборудованием можно разделить на две части:

- модули для оборудования с поддержкой API
- модули для оборудования, которое работает только через CLI

Если оборудование поддерживает API, как, например, NXOS, то для него создано большое количество модулей, которые выполняют конкретные действия по настройке функционала (например, для NXOS создано более 60 модулей).

Для оборудования, которое работает только через CLI, Ansible поддерживает, как минимум, такие три типа модулей:

- os_command - выполняет команды show
- os_facts - собирает факты об устройствах
- os_config - выполняет команды конфигурации

Соответственно, для разных операционных систем будут разные модули. Например, для Cisco IOS модули будут называться:

- ios_command
- ios_config
- ios_facts

Аналогичные три модуля доступны для таких ОС:

- Dellos10
- Dellos6
- Dellos9
- EOS

- IOS
- IOS XR
- JUNOS
- SR OS
- VyOS

Полный список всех сетевых модулей, которые поддерживает Ansible, в документации.

Обратите внимание, что Ansible очень активно развивается в сторону поддержки работы с сетевым оборудованием, и в следующей версии Ansible, могут быть дополнительные модули. Поэтому, если на момент чтения книги уже есть следующая версия Ansible (версия в книги 2.9), используйте её и посмотрите в документации, какие новые возможности и модули появились.

В этом разделе все рассматривается на примере модулей для работы с Cisco IOS:

- ios_command
- ios_config
- ios_facts

Примечание: Аналогичные модули command, config и facts для других вендоров и ОС работают одинаково, поэтому, если разобраться, как работать с модулями для IOS, с остальными всё будет аналогично.

Кроме того, рассматривается модуль ntc-ansible, который не входит в core модули Ansible.

Модуль ios_command

Модуль **ios_command** отправляет команду show на устройство под управлением IOS и возвращает результат выполнения команды.

Примечание: Модуль ios_command не поддерживает отправку команд в конфигурационном режиме. Для этого используется отдельный модуль - ios_config.

Модуль ios_command поддерживает такие параметры:

- **commands** - список команд, которые надо отправить на устройство
- **wait_for** (или waitfor) - список условий, на которые надо проверить вывод команды. Задача ожидает выполнения всех условий. Если после указанного количества попыток

выполнения команды условия не выполняются, будет считаться, что задача выполнена неудачно.

- **match** - этот параметр используется вместе с wait_for для указания политики совпадения. Если параметр match установлен в all, должны выполниться все условия в wait_for. Если параметр равен any, достаточно, чтобы выполнилось одно из условий.
- **retries** - указывает количество попыток выполнения команды, прежде чем она будет считаться невыполненной. По умолчанию - 10 попыток.
- **interval** - интервал в секундах между повторными попытками выполнить команду. По умолчанию - 1 секунда.

Перед отправкой самой команды модуль:

- выполняет аутентификацию по SSH
- переходит в режим enable
- выполняет команду `terminal length 0`, чтобы вывод команд show отражался полностью, а не постранично
- выполняет команду `terminal width 512`

Пример использования модуля ios_command (playbook 1_ios_command.yml):

```
---

- name: Run show commands on routers
  hosts: cisco-routers

  tasks:

    - name: run sh ip int br
      ios_command:
        commands: show ip int br
      register: sh_ip_int_br_result

    - name: Debug registered var
      debug: var=sh_ip_int_br_result.stdout_lines
```

Модуль ios_command ожидает, как минимум, такие аргумент commands - список команд, которые нужно отправить на устройство

Обратите внимание, что параметр register находится на одном уровне с именем задачи и модулем, а не на уровне параметров модуля ios_command.

Результат выполнения playbook:

```
$ ansible-playbook 1_ios_command.yml
```

```
SSH password:

PLAY [Run show commands on routers] ****************************************

TASK [run sh ip int br] ****************************************************
ok: [192.168.100.1]
ok: [192.168.100.2]
ok: [192.168.100.3]

TASK [Debug registered var] ************************************************
ok: [192.168.100.1] => {
    "sh_ip_int_br_result.stdout_lines": [
        [
            "Interface                  IP-Address      OK? Method Status                Protocol",
            "Ethernet0/0                192.168.100.1   YES NVRAM  up                    up      ",
            "Ethernet0/1                192.168.200.1   YES NVRAM  up                    up      ",
            "Ethernet0/2                unassigned      YES manual administratively down down    ",
            "Ethernet0/3                unassigned      YES manual up                    up      ",
            "Loopback0                  10.1.1.1        YES manual up                    up      "
        ]
    ]
}
ok: [192.168.100.2] => {
    "sh_ip_int_br_result.stdout_lines": [
        [
            "Interface                  IP-Address      OK? Method Status                Protocol",
            "Ethernet0/0                192.168.100.2   YES manual up                    up      ",
            "Ethernet0/1                unassigned      YES unset  administratively down down    ",
            "Ethernet0/2                192.168.200.1   YES manual administratively down down    ",
            "Ethernet0/3                unassigned      YES manual up                    up      ",
            "Loopback0                  10.1.1.1        YES manual up                    up      "
        ]
    ]
}
ok: [192.168.100.3] => {
    "sh_ip_int_br_result.stdout_lines": [
        [
            "Interface                  IP-Address      OK? Method Status                Protocol",
            "Ethernet0/0                192.168.100.3   YES manual up                    up      ",
            "Ethernet0/1                unassigned      YES unset  administratively down down    ",
            "Ethernet0/2                192.168.200.1   YES manual administratively down down    ",
            "Ethernet0/3                unassigned      YES manual up                    up      ",
            "Loopback0                  10.1.1.1        YES manual up                    up      ",
            "Loopback10                 10.255.3.3      YES manual up                    up      "
        ]
    ]
}

PLAY RECAP *****************************************************************
192.168.100.1              : ok=2    changed=0    unreachable=0    failed=0
192.168.100.2              : ok=2    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=2    changed=0    unreachable=0    failed=0
```

В отличие от использования модуля raw, playbook не указывает, что были выполнены изменения.

Выполнение нескольких команд

Модуль ios_command позволяет выполнять несколько команд.

Playbook 2_ios_command.yml выполняет несколько команд и получает их вывод:

```
---

- name: Run show commands on routers
  hosts: cisco-routers

  tasks:

    - name: run show commands
      ios_command:
        commands:
          - show ip int br
          - sh ip route
      register: show_result

    - name: Debug registered var
      debug: var=show_result.stdout_lines
```

В первой задаче указываются две команды, поэтому синтаксис должен быть немного другим - команды должны быть указаны как список, в формате YAML.

Результат выполнения playbook (вывод сокращен):

```
$ ansible-playbook 2_ios_command.yml
```

```
SSH password:

PLAY [Run show commands on routers] ******************************************

TASK [run show commands] *****************************************************
ok: [192.168.100.3]
ok: [192.168.100.1]
ok: [192.168.100.2]

TASK [Debug registered var] **************************************************
ok: [192.168.100.1] => {
    "show_result.stdout_lines": [
        [
            "Interface                  IP-Address      OK? Method Status                Protocol",
            "Ethernet0/0                192.168.100.1   YES NVRAM  up                    up      ",
            "Ethernet0/1                192.168.200.1   YES NVRAM  up                    up      ",
            "Ethernet0/2                unassigned      YES manual administratively down down    ",
            "Ethernet0/3                unassigned      YES manual up                    up      ",
            "Loopback0                  10.1.1.1        YES manual up                    up      "
        ],
        [
            "Codes: L - local, C - connected, S - static, R - RIP, M - mobile, B - BGP",
            "       D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area ",
            "       N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2",
            "       E1 - OSPF external type 1, E2 - OSPF external type 2",
            "       i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2",
            "       ia - IS-IS inter area, * - candidate default, U - per-user static route",
            "       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP",
            "       + - replicated route, % - next hop override",
            "",
            "Gateway of last resort is not set",
            "",
            "      10.0.0.0/32 is subnetted, 2 subnets",
            "C        10.1.1.1 is directly connected, Loopback0",
            "D        10.255.3.3 [90/409600] via 192.168.100.3, 02:04:51, Ethernet0/0",
            "      192.168.100.0/24 is variably subnetted, 2 subnets, 2 masks",
            "C        192.168.100.0/24 is directly connected, Ethernet0/0",
            "L        192.168.100.1/32 is directly connected, Ethernet0/0",
            "      192.168.200.0/24 is variably subnetted, 2 subnets, 2 masks",
            "C        192.168.200.0/24 is directly connected, Ethernet0/1",
            "L        192.168.200.1/32 is directly connected, Ethernet0/1"
        ]
    ]
}
```

Обе команды выполнились на всех устройствах.

Если модулю передаются несколько команд, результат выполнения команд находится в переменных stdout и stdout_lines в списке. Вывод будет в том порядке, в котором команды описаны в задаче.

За счет этого, например, можно вывести результат выполнения первой команды, указав:

```
- name: Debug registered var
  debug: var=show_result.stdout_lines[0]
```

Обработка ошибок

В модуле встроено распознание ошибок. Поэтому, если команда выполнена с ошибкой, модуль отобразит, что возникла ошибка.

Например, если сделать ошибку в команде и запустить playbook еще раз

```
$ ansible-playbook 2_ios_command.yml
```

```
PLAY [Run show commands on routers] ********************************************

TASK [run show commands] *******************************************************
fatal: [192.168.100.2]: FAILED! => {"changed": false, "failed": true, "msg": "shw ip in
t br\r\n     ^\r\n% Invalid input detected at '^' marker.\r\n\r\nR2#", "rc": 1}
fatal: [192.168.100.3]: FAILED! => {"changed": false, "failed": true, "msg": "shw ip in
t br\r\n     ^\r\n% Invalid input detected at '^' marker.\r\n\r\nR3#", "rc": 1}
fatal: [192.168.100.1]: FAILED! => {"changed": false, "failed": true, "msg": "shw ip in
t br\r\n     ^\r\n% Invalid input detected at '^' marker.\r\n\r\nR1#", "rc": 1}
        to retry, use: --limit @/home/vagrant/repos/pyneng-online-jun-jul-2017/examples
/15_ansible/3_network_modules/ios_command/2_ios_command.retry

PLAY RECAP *********************************************************************
192.168.100.1              : ok=0    changed=0    unreachable=0    failed=1
192.168.100.2              : ok=0    changed=0    unreachable=0    failed=1
192.168.100.3              : ok=0    changed=0    unreachable=0    failed=1
```

Ansible обнаружил ошибку и возвращает сообщение ошибки. В данном случае - „Invalid input“.

Аналогичным образом модуль обнаруживает ошибки:

- Ambiguous command
- Incomplete command

wait_for

Параметр wait_for (или waitfor) позволяет указывать список условий, на которые надо проверить вывод команды.

Пример playbook (файл 3_ios_command_wait_for.yml):

```
---

- name: Run show commands on routers
  hosts: cisco-routers

  tasks:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  - name: run show commands
    ios_command:
      commands: ping 192.168.100.100
      wait_for:
        - result[0] contains 'Success rate is 100 percent'
```

В playbook всего одна задача, которая отправляет команду ping 192.168.100.100, и проверяет, есть ли в выводе команды фраза „Success rate is 100 percent“.

Если в выводе команды содержится эта фраза, задача считается корректно выполненной.

Запуск playbook:

```
$ ansible-playbook 3_ios_command_wait_for.yml -v
```

```
Using /home/vagrant/repos/pyneng-online-jun-jul-2017/examples/15_ansible/3_network_module
s/ios_command/ansible.cfg as config file

PLAY [Run show commands on routers] ********************************************

TASK [run show commands] *******************************************************
ok: [192.168.100.1] => {"changed": false, "failed": false, "stdout": ["Type escape sequen
ce to abort.\nSending 5, 100-byte ICMP Echos to 192.168.100.100, timeout is 2 seconds:\n!
!!!!\nSuccess rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms"], "stdout_lin
es": [["Type escape sequence to abort.", "Sending 5, 100-byte ICMP Echos to 192.168.100.1
00, timeout is 2 seconds:", "!!!!!", "Success rate is 100 percent (5/5), round-trip min/a
vg/max = 1/1/1 ms"]]}
ok: [192.168.100.2] => {"changed": false, "failed": false, "stdout": ["Type escape sequen
ce to abort.\nSending 5, 100-byte ICMP Echos to 192.168.100.100, timeout is 2 seconds:\n!
!!!!\nSuccess rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms"], "stdout_lin
es": [["Type escape sequence to abort.", "Sending 5, 100-byte ICMP Echos to 192.168.100.1
00, timeout is 2 seconds:", "!!!!!", "Success rate is 100 percent (5/5), round-trip min/a
vg/max = 1/1/1 ms"]]}
ok: [192.168.100.3] => {"changed": false, "failed": false, "stdout": ["Type escape sequen
ce to abort.\nSending 5, 100-byte ICMP Echos to 192.168.100.100, timeout is 2 seconds:\n!
!!!!\nSuccess rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1 ms"], "stdout_lin
es": [["Type escape sequence to abort.", "Sending 5, 100-byte ICMP Echos to 192.168.100.1
00, timeout is 2 seconds:", "!!!!!", "Success rate is 100 percent (5/5), round-trip min/a
vg/max = 1/1/1 ms"]]}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

Если указан IP-адрес, который не доступен, результат будет таким:

```
$ ansible-playbook 3_ios_command_wait_for.yml -v
```

```
PLAY [Run show commands on routers] ********************************************************

TASK [run show commands] *******************************************************************
fatal: [192.168.100.1]: FAILED! => {"changed": false, "failed": true, "msg": "timeout trying to
send command: b'ping 192.168.100.10'", "rc": 1}
fatal: [192.168.100.2]: FAILED! => {"changed": false, "failed": true, "msg": "timeout trying to
send command: b'ping 192.168.100.10'", "rc": 1}
fatal: [192.168.100.3]: FAILED! => {"changed": false, "failed": true, "msg": "timeout trying to
send command: b'ping 192.168.100.10'", "rc": 1}
        to retry, use: --limit @/home/vagrant/repos/pyneng-online-jun-jul-2017/examples/15_ansib
le/3_network_modules/ios_command/3_ios_command_wait_for.retry

PLAY RECAP *********************************************************************************
192.168.100.1              : ok=0    changed=0    unreachable=0    failed=1
192.168.100.2              : ok=0    changed=0    unreachable=0    failed=1
192.168.100.3              : ok=0    changed=0    unreachable=0    failed=1
```

Такой вывод из-за того, что по умолчанию таймаут для каждого пакета 2 секунды, и за время выполнения playbook команда еще не выполнена.

По умолчанию есть 10 попыток выполнить команду, при этом между каждыми двумя попытками интервал - секунда. В реальной ситуации при проверке доступности адреса лучше сделать хотя бы две попытки.

Playbook 3_ios_command_wait_for_interval.yml выполняет две попытки, на каждую попытку 12 секунд:

```
---

- name: Run show commands on routers
  hosts: cisco-routers

  tasks:

    - name: run show commands
      ios_command:
        commands: ping 192.168.100.5 timeout 1
        wait_for:
          - result[0] contains 'Success rate is 100 percent'
        retries:  2
        interval: 12
```

В этом случае вывод будет таким:

```
$ ansible-playbook 3_ios_command_wait_for_interval.yml
```

```
PLAY [Run show commands on routers] ********************************************

TASK [run show commands] *******************************************************
fatal: [192.168.100.2]: FAILED! => {"changed": false, "failed": true, "failed_conditions":
 ["result[0] contains 'Success rate is 100 percent'"], "msg": "One or more conditional sta
tements have not be satisfied"}
fatal: [192.168.100.1]: FAILED! => {"changed": false, "failed": true, "failed_conditions":
 ["result[0] contains 'Success rate is 100 percent'"], "msg": "One or more conditional sta
tements have not be satisfied"}
fatal: [192.168.100.3]: FAILED! => {"changed": false, "failed": true, "failed_conditions":
 ["result[0] contains 'Success rate is 100 percent'"], "msg": "One or more conditional sta
tements have not be satisfied"}
        to retry, use: --limit @/home/vagrant/repos/pyneng-online-jun-jul-2017/examples/15
_ansible/3_network_modules/ios_command/3_ios_command_wait_for_interval.retry

PLAY RECAP *********************************************************************************
192.168.100.1              : ok=0    changed=0    unreachable=0    failed=1
192.168.100.2              : ok=0    changed=0    unreachable=0    failed=1
192.168.100.3              : ok=0    changed=0    unreachable=0    failed=1
```

Модуль ios_facts

Модуль ios_facts собирает информацию с устройств под управлением IOS.

Информация берется из таких команд:

- dir
- show version
- show memory statistics
- show interfaces
- show ipv6 interface
- show lldp
- show lldp neighbors detail
- show running-config

Примечание: Чтобы видеть, какие команды Ansible выполняет на оборудовании, можно настроить EEM applet, который будет генерировать лог сообщения о выполненных командах.

В модуле можно указывать, какие параметры собирать - можно собирать всю информацию, а

можно только подмножество. По умолчанию модуль собирает всю информацию, кроме конфигурационного файла.

Какую информацию собирать, указывается в параметре **gather_subset**. Поддерживаются такие варианты (указаны также команды, которые будут выполняться на устройстве):

- **all**
- **hardware**
  - dir
  - show version
  - show memory statistics
- **config**
  - show version
  - show running-config
- **interfaces**
  - dir
  - show version
  - show interfaces
  - show ip interface
  - show ipv6 interface
  - show lldp
  - show lldp neighbors detail

Собрать все факты:

```
- ios_facts:
    gather_subset: all
```

Собрать только подмножество interfaces:

```
- ios_facts:
    gather_subset:
      - interfaces
```

Собрать всё, кроме hardware:

```
- ios_facts:
    gather_subset:
      - "!hardware"
```

Ansible собирает такие факты:

- ansible_net_all_ipv4_addresses - список IPv4 адресов на устройстве
- ansible_net_all_ipv6_addresses - список IPv6 адресов на устройстве
- ansible_net_config - конфигурация (для Cisco sh run)
- ansible_net_filesystems - файловая система устройства
- ansible_net_gather_subset - какая информация собирается (hardware, default, interfaces, config)
- ansible_net_hostname - имя устройства
- ansible_net_image - имя и путь ОС
- ansible_net_interfaces - словарь со всеми интерфейсами устройства. Имена интерфейсов - ключи, а данные - параметры каждого интерфейса
- ansible_net_memfree_mb - сколько свободной памяти на устройстве
- ansible_net_memtotal_mb - сколько памяти на устройстве
- ansible_net_model - модель устройства
- ansible_net_serialnum - серийный номер
- ansible_net_version - версия IOS

Использование модуля

Пример playbook 1_ios_facts.yml с использованием модуля ios_facts (собираются все факты):

```
---

- name: Collect IOS facts
  hosts: cisco-routers

  tasks:

    - name: Facts
      ios_facts:
        gather_subset: all
```

```
$ ansible-playbook 1_ios_facts.yml
```

```
SSH password:

PLAY [Collect IOS facts] ********************************************************

TASK [Facts] ********************************************************************
ok: [192.168.100.1]
ok: [192.168.100.2]
ok: [192.168.100.3]

PLAY RECAP **********************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

Для того, чтобы посмотреть, какие именно факты собираются с устройства, можно добавить флаг -v (информация сокращена):

```
$ ansible-playbook 1_ios_facts.yml -v
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
```

```
SSH password:

PLAY [Collect IOS facts] ********************************************************

TASK [Facts] ********************************************************************
ok: [192.168.100.1] => {"ansible_facts": {"ansible_net_all_ipv4_addresses": ["192.168.
200.1", "192.168.100.1", "10.1.1.1"], "ansible_net_all_ipv6_addresses": [], "ansible_n
et_config": "Building configuration...\n\nCurrent configuration : 6716 bytes\n!\n! Las
t configuration change at 09:09:04 UTC Sun Dec 18 2016\nversion 15.2\nno service times
```

После того, как Ansible собрал факты с устройства, все факты доступны как переменные в playbook, шаблонах и т.д.

Например, можно отобразить содержимое факта с помощью debug (playbook 2_ios_facts_debug.yml):

```
---

- name: Collect IOS facts
  hosts: 192.168.100.1

  tasks:

    - name: Facts
      ios_facts:
        gather_subset: all
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    - name: Show ansible_net_all_ipv4_addresses fact
      debug: var=ansible_net_all_ipv4_addresses

    - name: Show ansible_net_interfaces fact
      debug: var=ansible_net_interfaces['Ethernet0/0']
```

Результат выполнения playbook:

```
$ ansible-playbook 2_ios_facts_debug.yml
```

```
SSH password:

PLAY [Collect IOS facts] ********************************************************

TASK [Facts] ********************************************************************
ok: [192.168.100.1]

TASK [Show ansible_net_all_ipv4_addresses fact] *********************************
ok: [192.168.100.1] => {
    "ansible_net_all_ipv4_addresses": [
        "192.168.200.1",
        "192.168.100.1"
    ]
}

TASK [Show fact] ****************************************************************
ok: [192.168.100.1] => {
    "ansible_net_interfaces['Ethernet0/0']": {
        "bandwidth": 10000,
        "description": null,
        "duplex": null,
        "ipv4": {
            "address": "192.168.100.1",
            "masklen": 24
        },
        "lineprotocol": "up ",
        "macaddress": "aabb.cc00.6500",
        "mediatype": null,
        "mtu": 1500,
        "operstatus": "up",
        "type": "AmdP2"
    }
}

PLAY RECAP **********************************************************************
192.168.100.1              : ok=3    changed=0    unreachable=0    failed=0
```

Сохранение фактов

В том виде, в котором информация отображается в режиме verbose, довольно сложно понять какая информация собирается об устройствах. Для того, чтобы лучше понять, какая информация собирается об устройствах и в каком формате, скопируем полученную информацию в файл.

Для этого будет использоваться модуль copy.

Playbook 3_ios_facts.yml собирает всю информацию об устройствах и записывает в разные файлы (создайте каталог all_facts перед запуском playbook или раскомментируйте задачу Create all_facts dir, и Ansible создаст каталог сам):

```
---

- name: Collect IOS facts
  hosts: cisco-routers

  tasks:

    - name: Facts
      ios_facts:
        gather_subset: all
      register: ios_facts_result

    #- name: Create all_facts dir
    #  file:
    #    path: ./all_facts/
    #    state: directory
    #    mode: 0755

    - name: Copy facts to files
      copy:
        content: "{{ ios_facts_result | to_nice_json }}"
        dest: "all_facts/{{inventory_hostname}}_facts.json"
```

Модуль copy позволяет копировать файлы с управляющего хоста (на котором установлен Ansible) на удаленный хост. Но так как в этом случае, указан параметр `connection: local`, файлы будут скопированы на локальный хост.

Чаще всего, модуль copy используется таким образом:

```
- copy:
    src: /srv/myfiles/foo.conf
    dest: /etc/foo.conf
```

Но в данном случае нет исходного файла, содержимое которого нужно скопировать. Вместо этого, есть содержимое переменной ios_facts_result, которое нужно перенести в файл

all_facts/{{inventory_hostname}}_facts.json.

Для того, чтобы перенести содержимое переменной в файл, в модуле copy вместо src используется параметр content.

В строке `content: "{{ ios_facts_result | to_nice_json }}"`

- параметр to_nice_json - это фильтр Jinja2, который преобразует информацию переменной в формат, в котором удобней читать информацию
- переменная в формате Jinja2 должна быть заключена в двойные фигурные скобки, а также указана в двойных кавычках

Так как в пути dest используются имена устройств, будут сгенерированы уникальные файлы для каждого устройства.

Результат выполнения playbook:

```
$ ansible-playbook 3_ios_facts.yml
```

```
SSH password:

PLAY [Collect IOS facts] ********************************************************

TASK [Facts] ********************************************************************
ok: [192.168.100.1]
ok: [192.168.100.2]
ok: [192.168.100.3]

TASK [Copy facts to files] ******************************************************
changed: [192.168.100.3]
changed: [192.168.100.1]
changed: [192.168.100.2]

PLAY RECAP **********************************************************************
192.168.100.1              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.3              : ok=2    changed=1    unreachable=0    failed=0
```

После этого в каталоге all_facts находятся такие файлы:

```
192.168.100.1_facts.json
192.168.100.2_facts.json
192.168.100.3_facts.json
```

Содержимое файла all_facts/192.168.100.1_facts.json:

```
{
    "ansible_facts": {
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        "ansible_net_all_ipv4_addresses": [
            "192.168.200.1",
            "192.168.100.1",
            "10.1.1.1"
        ],
        "ansible_net_all_ipv6_addresses": [],
        "ansible_net_config": "Building configuration...\n\nCurrent configuration :
...
```

Сохранение информации об устройствах не только поможет разобраться, какая информация собирается, но и может быть полезным для дальнейшего использования информации. Например, можно использовать факты об устройстве в шаблоне.

При повторном выполнении playbook Ansible не будет изменять информацию в файлах, если факты об устройстве не изменились

Если информация изменилась, для соответствующего устройства будет выставлен статус changed. Таким образом, по выполнению playbook всегда понятно, когда какая-то информация изменилась.

Повторный запуск playbook (без изменений):

```
$ ansible-playbook 3_ios_facts.yml
```

```
SSH password:

PLAY [Collect IOS facts] ********************************************************

TASK [Facts] ********************************************************************
ok: [192.168.100.1]
ok: [192.168.100.3]
ok: [192.168.100.2]

TASK [Copy facts to files] ******************************************************
ok: [192.168.100.2]
ok: [192.168.100.1]
ok: [192.168.100.3]

PLAY RECAP **********************************************************************
192.168.100.1              : ok=2    changed=0    unreachable=0    failed=0
192.168.100.2              : ok=2    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=2    changed=0    unreachable=0    failed=0
```

Модуль ios_config

Модуль ios_config позволяет настраивать устройства под управлением IOS, а также генерировать шаблоны конфигураций или отправлять команды на основании шаблона.

Параметры модуля:

- **after** - какие действия выполнить после команд
- **before** - какие действия выполнить до команд
- **backup** - параметр, который указывает, нужно ли делать резервную копию текущей конфигурации устройства перед внесением изменений. Файл будет копироваться в каталог backup относительно каталога, в котором находится playbook
- **config** - параметр, который позволяет указать базовый файл конфигурации, с которым будут сравниваться изменения. Если он указан, модуль не будет скачивать конфигурацию с устройства.
- **defaults** - параметр указывает, нужно ли собирать всю информацию с устройства, в том числе, значения по умолчанию. Если включить этот параметр, то модуль будет собирать текущую конфигурацию с помощью команды sh run all. По умолчанию этот параметр отключен, и конфигурация проверяется командой sh run
- **lines (commands)** - список команд, которые должны быть настроены. Команды нужно указывать без сокращений и ровно в том виде, в котором они будут в конфигурации.
- **match** - параметр указывает, как именно нужно сравнивать команды
- **parents** - название секции, в которой нужно применить команды. Если команда находится внутри вложенной секции, нужно указывать весь путь. Если этот параметр не указан, то считается, что команда должна быть в глобальном режиме конфигурации
- **replace** - параметр указывает, как выполнять настройку устройства
- **save_when** - сохранять ли текущую конфигурацию в стартовую. По умолчанию конфигурация не сохраняется
- **src** - параметр указывает путь к файлу, в котором находится конфигурация или шаблон конфигурации. Взаимоисключающий параметр с lines (то есть, можно указывать или lines, или src). Заменяет модуль ios_template, который скоро будет удален.
- **diff_against**, **diff_ignore_lines**, **intended_config** - параметры, которые указывают, какие конфигурации надо сравнивать

lines (commands)

Самый простой способ использовать модуль ios_config - отправлять команды глобального конфигурационного режима с параметром lines. Для параметра lines есть alias commands, то есть, можно вместо lines писать commands.

Пример playbook 1_ios_config_lines.yml:

```
---

- name: Run cfg commands on routers
  hosts: cisco-routers

  tasks:

    - name: Config password encryption
      ios_config:
        lines:
          - service password-encryption
```

Используется переменная cli, которая указана в файле group_vars/all.yml.

Результат выполнения playbook:

```
$ ansible-playbook 1_ios_config_lines.yml
```

```
PLAY [Run cfg commands on routers] *********************************************

TASK [Config password encryption] **********************************************
changed: [192.168.100.1]
changed: [192.168.100.3]
changed: [192.168.100.2]

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=1    unreachable=0    failed=0
```

Ansible выполняет такие команды:

- terminal length 0
- enable
- show running-config - чтобы проверить, есть ли эта команда на устройстве. Если команда есть, задача выполняться не будет. Если команды нет, задача выполнится
- если команды, которая указана в задаче, нет в конфигурации:

- configure terminal
- service password-encryption
- end

Так как модуль каждый раз проверяет конфигурацию, прежде чем применит команду, модуль идемпотентен. То есть, если ещё раз запустить playbook, изменения не будут выполнены:

```
$ ansible-playbook 1_ios_config_lines.yml
```

```
SSH password:

PLAY [Run cfg commands on routers] ***************************************

TASK [Config password encryption] ***************************************
ok: [192.168.100.2]
ok: [192.168.100.1]
ok: [192.168.100.3]

PLAY RECAP **************************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

Предупреждение: Обязательно пишите команды полностью, а не сокращенно. И обращайте внимание, что для некоторых команд IOS сам добавляет параметры. Если писать команду не в том виде, в котором она реально видна в конфигурационном файле, модуль не будет идемпотентен. Он будет всё время считать, что команды нет, и вносить изменения каждый раз.

Параметр lines позволяет отправлять и несколько команд (playbook 1_ios_config_mult_lines.yml):

```
---

- name: Run cfg commands on routers
  hosts: cisco-routers

  tasks:

    - name: Send config commands
      ios_config:
        lines:
          - service password-encryption
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        - no ip http server
        - no ip http secure-server
        - no ip domain lookup
```

Результат выполнения:

```
$ ansible-playbook 1_ios_config_mult_lines.yml
```

```
SSH password:

PLAY [Run cfg commands on routers] ****************************************

TASK [Send config commands] ***********************************************
changed: [192.168.100.3]
changed: [192.168.100.1]
changed: [192.168.100.2]

PLAY RECAP ****************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=1    unreachable=0    failed=0
```

parents

Параметр parents используется, чтобы указать, в каком подрежиме применить команды.

Например, необходимо применить такие команды:

```
line vty 0 4
 login local
 transport input ssh
```

В таком случае, playbook 2_ios_config_parents_basic.yml будет выглядеть так:

```
---

- name: Run cfg commands on routers
  hosts: cisco-routers

  tasks:

    - name: Config line vty
      ios_config:
        parents:
          - line vty 0 4
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        lines:
          - login local
          - transport input ssh
```

Запуск будет выполняться аналогично предыдущим playbook:

```
$ ansible-playbook 2_ios_config_parents_basic.yml
```

```
SSH password:

PLAY [Run cfg commands on routers] ********************************************

TASK [Config line vty] ********************************************************
changed: [192.168.100.1]
changed: [192.168.100.2]
changed: [192.168.100.3]

PLAY RECAP ********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=1    unreachable=0    failed=0
```

Если команда находится в нескольких вложенных режимах, подрежимы указываются в списке parents.

Например, необходимо выполнить такие команды:

```
policy-map OUT_QOS
 class class-default
  shape average 100000000 1000000
```

Тогда playbook 2_ios_config_parents_mult.yml будет выглядеть так:

```
---

- name: Run cfg commands on routers
  hosts: cisco-routers

  tasks:

    - name: Config QoS policy
      ios_config:
        parents:
          - policy-map OUT_QOS
          - class class-default
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        lines:
          - shape average 100000000 1000000
```

Отображение обновлений

В этом разделе рассматриваются варианты отображения информации об обновлениях, которые выполнил модуль ios_config.

Playbook 2_ios_config_parents_basic.yml:

```
---

- name: Run cfg commands on routers
  hosts: cisco-routers

  tasks:

    - name: Config line vty
      ios_config:
        parents:
          - line vty 0 4
        lines:
          - login local
          - transport input ssh
```

Для того, чтобы playbook что-то менял, нужно сначала отменить команды - либо вручную, либо изменив playbook. Например, на маршрутизаторе 192.168.100.1 вместо строки transport input ssh вручную прописать строку transport input all.

Например, можно выполнить playbook с флагом verbose:

```
$ ansible-playbook 2_ios_config_parents_basic.yml -v
```

```
Using /home/vagrant/repos/pyneng-online-jun-jul-2017/examples/15_ansible/3_network_modu
les/ios_config/ansible.cfg as config file

PLAY [Run cfg commands on routers] **********************************************

TASK [Config line vty] *********************************************************
ok: [192.168.100.2] => {"changed": false, "failed": false}
ok: [192.168.100.3] => {"changed": false, "failed": false}
changed: [192.168.100.1] => {"banners": {}, "changed": true, "commands": ["line vty 0 4
", "transport input ssh"], "failed": false, "updates": ["line vty 0 4", "transport inpu
t ssh"]}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

В выводе в поле updates видно, какие именно команды Ansible отправил на устройство. Изменения были выполнены только на маршрутизаторе 192.168.100.1.

Обратите внимание, что команда login local не отправлялась, так как она настроена. Поле updates в выводе есть только в том случае, когда есть изменения.

В режиме verbose информация видна обо всех устройствах. Но было бы удобней, чтобы информация отображалась только для тех устройств, для которых произошли изменения.

Новый playbook 3_ios_config_debug.yml на основе 2_ios_config_parents_basic.yml:

```
---

- name: Run cfg commands on routers
  hosts: cisco-routers

  tasks:

    - name: Config line vty
      ios_config:
        parents:
          - line vty 0 4
        lines:
          - login local
          - transport input ssh
      register: cfg

    - name: Show config updates
      debug: var=cfg.updates
      when: cfg.changed
```

Изменения в playbook:

- результат работы первой задачи сохраняется в переменную **cfg**.
- в следующей задаче модуль **debug** выводит содержимое поля **updates**.
- но так как поле updates в выводе есть только в том случае, когда есть изменения, ставится условие when, которое проверяет, были ли изменения
- задача будет выполняться, только если на устройстве были внесены изменения.
- вместо `when: cfg.changed` можно написать `when: cfg.changed == true`

Если запустить повторно playbook, когда изменений не было, задача Show config updates пропускается:

```
$ ansible-playbook 3_ios_config_debug.yml
```

```
SSH password:

PLAY [Run cfg commands on routers] ********************************************

TASK [Config line vty] ********************************************************
ok: [192.168.100.2]
ok: [192.168.100.3]
ok: [192.168.100.1]

TASK [Show config updates] ****************************************************
skipping: [192.168.100.1]
skipping: [192.168.100.2]
skipping: [192.168.100.3]

PLAY RECAP ********************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

Если внести изменения в конфигурацию маршрутизатора 192.168.100.1 (изменить transport input ssh на transport input all):

```
$ ansible-playbook 3_ios_config_debug.yml
```

```
SSH password:

PLAY [Run cfg commands on routers] ********************************************

TASK [Config line vty] ********************************************************
ok: [192.168.100.2]
changed: [192.168.100.1]
ok: [192.168.100.3]

TASK [Show config updates] ****************************************************
ok: [192.168.100.1] => {
    "cfg.updates": [
        "line vty 0 4",
        "transport input ssh"
    ]
}
skipping: [192.168.100.2]
skipping: [192.168.100.3]

PLAY RECAP ********************************************************************
192.168.100.1              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

Теперь второе задание отображает информацию о том, какие именно изменения были внесены на маршрутизаторе.

save_when

Параметр **save_when** позволяет указать, нужно ли сохранять текущую конфигурацию в стартовую.

Доступные варианты значений:

- always - всегда сохранять конфигурацию (в этом случае флаг modified будет равен True)
- never (по умолчанию) - не сохранять конфигурацию
- modified - в этом случае конфигурация сохраняется только при наличии изменений

Playbook 4_ios_config_save_when.yml:

```
---

- name: Run cfg commands on routers
  hosts: cisco-routers

```

(continues on next page)

(продолжение с предыдущей страницы)

```
  tasks:

    - name: Config line vty
      ios_config:
        parents:
          - line vty 0 4
        lines:
          - login local
          - transport input ssh telnet
        save_when: modified
```

Выполнение playbook:

```
$ ansible-playbook 4_ios_config_save_when.yml
```

```
SSH password:

PLAY [Run cfg commands on routers] ******************************************

TASK [Config line vty] ******************************************************
ok: [192.168.100.3]
ok: [192.168.100.2]
changed: [192.168.100.1]

TASK [Save config] **********************************************************
skipping: [192.168.100.2]
skipping: [192.168.100.3]
ok: [192.168.100.1]

PLAY RECAP ******************************************************************
192.168.100.1              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

backup

Параметр **backup** указывает, нужно ли делать резервную копию текущей конфигурации устройства перед внесением изменений. Файл будет копироваться в каталог backup относительно каталога, в котором находится playbook (если каталог не существует, он будет создан).

Playbook 5_ios_config_backup.yml:

```
---
```

(continues on next page)

(продолжение с предыдущей страницы)

```
- name: Run cfg commands on routers
  hosts: cisco-routers

  tasks:

    - name: Config line vty
      ios_config:
        parents:
          - line vty 0 4
        lines:
          - login local
          - transport input ssh
        backup: yes
```

Теперь каждый раз, когда выполняется playbook (даже если не нужно вносить изменения в конфигурацию), в каталог backup будет копироваться текущая конфигурация:

```
$ ansible-playbook 5_ios_config_backup.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on routers] *********************************************

TASK [Config line vty] *********************************************************
ok: [192.168.100.1] => {"backup_path": "/home/nata/pyneng_course/chapter15/backup/192
.168.100.1_config.2016-12-10@12:35:38", "changed": false, "warnings": []}
ok: [192.168.100.3] => {"backup_path": "/home/nata/pyneng_course/chapter15/backup/192
.168.100.3_config.2016-12-10@12:35:38", "changed": false, "warnings": []}
ok: [192.168.100.2] => {"backup_path": "/home/nata/pyneng_course/chapter15/backup/192
.168.100.2_config.2016-12-10@12:35:38", "changed": false, "warnings": []}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

В каталоге backup теперь находятся файлы такого вида (при каждом запуске playbook они перезаписываются):

```
192.168.100.1_config.2016-12-10@10:42:34
192.168.100.2_config.2016-12-10@10:42:34
192.168.100.3_config.2016-12-10@10:42:34
```

defaults

Параметр **defaults** указывает, нужно ли собирать всю информацию с устройства, в том числе и значения по умолчанию. Если включить этот параметр, модуль будет собирать текущую конфигурацию с помощью команды sh run all. По умолчанию этот параметр отключен, и конфигурация проверяется командой sh run.

Этот параметр полезен в том случае, если в настройках указывается команда, которая не видна в конфигурации. Например, такое может быть, когда указан параметр, который и так используется по умолчанию.

Если не использовать параметр defaults и указать команду, которая настроена по умолчанию, то при каждом запуске playbook будут вноситься изменения.

Происходит это потому, что Ansible каждый раз вначале проверяет наличие команд в соответствующем режиме. Если команд нет, то соответствующая задача выполняется.

Например, в таком playbook каждый раз будут вноситься изменения (попробуйте запустить его самостоятельно):

```
---

- name: Run cfg commands on routers
  hosts: cisco-routers

  tasks:

    - name: Config interface
      ios_config:
        parents:
          - interface Ethernet0/2
        lines:
          - ip address 192.168.200.1 255.255.255.0
          - ip mtu 1500
```

Если добавить параметр `defaults: yes`, изменения уже не будут внесены, если не хватало только команды ip mtu 1500 (playbook 6_ios_config_defaults.yml):

```
---

- name: Run cfg commands on routers
  hosts: cisco-routers

  tasks:

    - name: Config interface
      ios_config:
        parents:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      - interface Ethernet0/2
    lines:
      - ip address 192.168.200.1 255.255.255.0
      - ip mtu 1500
    defaults: yes
```

Запуск playbook:

```
$ ansible-playbook 6_ios_config_defaults.yml
```

```
SSH password:

PLAY [Run cfg commands on routers] ****************************************

TASK [Config interface] ***************************************************
ok: [192.168.100.3]
ok: [192.168.100.1]
ok: [192.168.100.2]

PLAY RECAP ****************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

after

Параметр **after** указывает, какие команды выполнить после команд в списке lines (или commands).

Команды, которые указаны в параметре after:

- выполняются, только если должны быть внесены изменения.
- при этом они будут выполнены независимо от того, есть они в конфигурации или нет.

Параметр after очень полезен в ситуациях, когда необходимо выполнить команду, которая не сохраняется в конфигурации.

Например, команда no shutdown не сохраняется в конфигурации маршрутизатора, и если добавить её в список lines, изменения будут вноситься каждый раз при выполнении playbook. Если написать команду no shutdown в списке after, она будет применена только в том случае, если нужно вносить изменения (согласно списка lines).

Пример использования параметра after в playbook 7_ios_config_after.yml:

```
---

- name: Run cfg commands on router
  hosts: 192.168.100.1

  tasks:

    - name: Config interface
      ios_config:
        parents:
          - interface Ethernet0/3
        lines:
          - ip address 192.168.230.1 255.255.255.0
        after:
          - no shutdown
```

Первый запуск playbook, с внесением изменений:

```
$ ansible-playbook 7_ios_config_after.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on routers] ******************************************

TASK [Config interface] *****************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["interface Ethernet0/3",
 "ip address 192.168.230.1 255.255.255.0", "no shutdown"], "warnings": []}

PLAY RECAP ******************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

Второй запуск playbook (изменений нет, поэтому команда no shutdown не выполняется):

```
$ ansible-playbook 7_ios_config_after.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on routers] ******************************************

TASK [Config interface] *****************************************************
ok: [192.168.100.1] => {"changed": false, "warnings": []}

PLAY RECAP ******************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
```

Рассмотрим ещё один пример использования after.

С помощью after можно сохранять конфигурацию устройства (playbook 7_ios_config_after_save.yml):

```
---

- name: Run cfg commands on routers
  hosts: cisco-routers

  tasks:

    - name: Config line vty
      ios_config:
        parents:
          - line vty 0 4
        lines:
          - login local
          - transport input ssh
        after:
          - end
          - write
```

Результат выполнения playbook (изменения только на маршрутизаторе 192.168.100.1):

```
$ ansible-playbook 7_ios_config_after_save.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on routers] ******************************************

TASK [Config line vty] ******************************************************
ok: [192.168.100.2] => {"changed": false, "warnings": []}
ok: [192.168.100.3] => {"changed": false, "warnings": []}
changed: [192.168.100.1] => {"changed": true, "updates": ["line vty 0 4", "transpo
rt input ssh", "end", "write"], "warnings": []}

PLAY RECAP ******************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

before

Параметр **before** указывает, какие действия выполнить до команд в списке lines.

Команды, которые указаны в параметре before: * выполняются, только если должны быть внесены изменения. * при этом они будут выполнены независимо от того, есть они в конфигурации или нет.

Параметр before полезен в ситуациях, когда какие-то действия необходимо выполнить перед выполнением команд в списке lines.

При этом, как и after, параметр before не влияет на то, какие команды сравниваются с конфигурацией. То есть, по-прежнему сравниваются только команды в списке lines.

Playbook 8_ios_config_before.yml:

```
---

- name: Run cfg commands on router
  hosts: 192.168.100.1

  tasks:

    - name: Config ACL
      ios_config:
        before:
          - no ip access-list extended IN_to_OUT
        parents:
          - ip access-list extended IN_to_OUT
        lines:
          - permit tcp 10.0.1.0 0.0.0.255 any eq www
          - permit tcp 10.0.1.0 0.0.0.255 any eq 22
          - permit icmp any any
```

В playbook 8_ios_config_before.yml ACL IN_to_OUT сначала удаляется с помощью параметра before, а затем создается заново.

Таким образом, в ACL всегда находятся только те строки, которые заданы в списке lines.

Запуск playbook с изменениями:

```
$ ansible-playbook 8_ios_config_before.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] **************************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["no ip access-list exten
ded IN_to_OUT", "ip access-list extended IN_to_OUT", "permit tcp 10.0.1.0 0.0.0.25
5 any eq www", "permit tcp 10.0.1.0 0.0.0.255 any eq 22", "permit icmp any any"],
"warnings": []}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

Запуск playbook без изменений (команда в списке before не выполняется):

```
$ ansible-playbook 8_ios_config_before.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] **************************************************************
ok: [192.168.100.1] => {"changed": false, "warnings": []}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
```

match

Параметр **match** указывает, как именно нужно сравнивать команды (что считается изменением):

- **line** - команды проверяются построчно. Этот режим используется по умолчанию
- **strict** - должны совпасть не только сами команды, но и их положение относительно друг друга
- **exact** - команды должны в точности совпадать с конфигурацией, и не должно быть никаких лишних строк
- **none** - модуль не будет сравнивать команды с текущей конфигурацией

match: line

Режим `match: line` используется по умолчанию.

В этом режиме модуль проверяет только наличие строк, перечисленных в списке lines в соответствующем режиме. При этом не проверяется порядок строк.

На маршрутизаторе 192.168.100.1 настроен такой ACL:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
```

Пример использования playbook 9_ios_config_match_line.yml в режиме line:

```
---

- name: Run cfg commands on router
  hosts: 192.168.100.1

  tasks:

    - name: Config ACL
      ios_config:
        parents:
          - ip access-list extended IN_to_OUT
        lines:
          - permit tcp 10.0.1.0 0.0.0.255 any eq www
          - permit tcp 10.0.1.0 0.0.0.255 any eq 22
          - permit icmp any any
```

Результат выполнения playbook:

```
$ ansible-playbook 9_ios_config_match_line.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] ***************************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["ip access-list extended
 IN_to_OUT", "permit tcp 10.0.1.0 0.0.0.255 any eq www", "permit icmp any any"], "
warnings": []}

PLAY RECAP **********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

Обратите внимание, что в списке updates только две из трёх строк ACL. Так как в режиме lines модуль сравнивает команды независимо друг от друга, он обнаружил, что не хватает только двух команд из трех.

В итоге конфигурация на маршрутизаторе выглядит так:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 permit icmp any any
```

То есть, порядок команд поменялся. И хотя в этом случае это не важно, иногда это может привести совсем не к тем результатам, которые ожидались.

Если повторно запустить playbook при такой конфигурации, он не будет выполнять изменения, так как все строки были найдены.

match: exact

Пример, в котором порядок команд важен.

ACL на маршрутизаторе:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 deny   ip any any
```

Playbook 9_ios_config_match_exact.yml (будет постепенно дополняться):

```
---

- name: Run cfg commands on router
  hosts: 192.168.100.1

  tasks:

    - name: Config ACL
      ios_config:
        parents:
          - ip access-list extended IN_to_OUT
        lines:
          - permit tcp 10.0.1.0 0.0.0.255 any eq www
          - permit tcp 10.0.1.0 0.0.0.255 any eq 22
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        - permit icmp any any
        - deny   ip any any
```

Если запустить playbook, результат будет таким:

```
$ ansible-playbook 9_ios_config_match_exact.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ******************************************

TASK [Config ACL] **********************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["ip access-list extended
 IN_to_OUT", "permit icmp any any", "deny ip any any"], "warnings": []}

PLAY RECAP *****************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

Теперь ACL выглядит так:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 deny   ip any any
 permit icmp any any
```

Конечно же, в таком случае последнее правило никогда не сработает.

Можно добавить к этому playbook параметр before и сначала удалить ACL, а затем применять команды:

```
---

- name: Run cfg commands on router
  hosts: 192.168.100.1

  tasks:

    - name: Config ACL
      ios_config:
        before:
          - no ip access-list extended IN_to_OUT
        parents:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        - ip access-list extended IN_to_OUT
      lines:
        - permit tcp 10.0.1.0 0.0.0.255 any eq www
        - permit tcp 10.0.1.0 0.0.0.255 any eq 22
        - permit icmp any any
        - deny   ip any any
```

Если применить playbook к последнему состоянию маршрутизатора, то изменений не будет никаких, так как все строки уже есть.

Попробуем начать с такого состояния ACL:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 deny   ip any any
```

Результат будет таким:

```
$ ansible-playbook 9_ios_config_match_exact.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:



PLAY [Run cfg commands on router] ******************************************

TASK [Config ACL] **********************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["no ip access-list exten
ded IN_to_OUT", "ip access-list extended IN_to_OUT", "permit icmp any any"], "warn
ings": []}

PLAY RECAP *****************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

И, соответственно, на маршрутизаторе:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit icmp any any
```

Теперь в ACL осталась только одна строка:

- Модуль проверил, каких команд не хватает в ACL (так как режим по умолчанию match: line),

- обнаружил, что не хватает команды `permit icmp any any`, и добавил её

Так как в playbook ACL сначала удаляется, а затем применяется список команд lines, получилось, что в итоге в ACL одна строка.

Поможет в такой ситуации вариант `match: exact`:

```
---

- name: Run cfg commands on router
  hosts: 192.168.100.1

  tasks:

    - name: Config ACL
      ios_config:
        before:
          - no ip access-list extended IN_to_OUT
        parents:
          - ip access-list extended IN_to_OUT
        lines:
          - permit tcp 10.0.1.0 0.0.0.255 any eq www
          - permit tcp 10.0.1.0 0.0.0.255 any eq 22
          - permit icmp any any
          - deny   ip any any
        match: exact
```

Применение playbook 9_ios_config_match_exact.yml к текущему состоянию маршрутизатора (в ACL одна строка):

```
$ ansible-playbook 9_ios_config_match_exact.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] ***************************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["no ip access-list exten


ded IN_to_OUT", "ip access-list extended IN_to_OUT", "permit tcp 10.0.1.0 0.0.0.25
5 any eq www", "permit tcp 10.0.1.0 0.0.0.255 any eq 22", "permit icmp any any", "
deny   ip any any"], "warnings": []}

PLAY RECAP **********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

Теперь результат такой:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit icmp any any
 deny   ip any any
```

То есть, теперь ACL выглядит точно так же, как и строки в списке lines, и в том же порядке.

И для того, чтобы окончательно разобраться с параметром `match: exact`, ещё один пример.

Закомментируем в playbook строки с удалением ACL:

```
---

- name: Run cfg commands on router
  hosts: 192.168.100.1

  tasks:

    - name: Config ACL
      ios_config:
        #before:
        #  - no ip access-list extended IN_to_OUT
        parents:
          - ip access-list extended IN_to_OUT
        lines:
          - permit tcp 10.0.1.0 0.0.0.255 any eq www
          - permit tcp 10.0.1.0 0.0.0.255 any eq 22
          - permit icmp any any
          - deny   ip any any
        match: exact
```

В начало ACL добавлена строка:

```
ip access-list extended IN_to_OUT
 permit udp any any
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit icmp any any
 deny   ip any any
```

То есть, последние 4 строки выглядят так, как нужно, и в том порядке, котором нужно. Но, при этом, есть лишняя строка. Для варианта match: exact - это уже несовпадение.

В таком варианте, playbook будет выполняться каждый раз и пытаться применить все команды из списка lines, что не будет влиять на содержимое ACL:

```
$ ansible-playbook 9_ios_config_match_exact.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] ***************************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["ip access-list extended
 IN_to_OUT", "permit tcp 10.0.1.0 0.0.0.255 any eq www", "permit tcp 10.0.1.0 0.0.
0.255 any eq 22", "permit icmp any any", "deny   ip any any"], "warnings": []}

PLAY RECAP **********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

Это значит, что при использовании `match:exact` важно, чтобы был какой-то способ удалить конфигурацию, если она не соответствует тому, что должно быть (или чтобы команды перезаписывались). Иначе эта задача будет выполняться каждый раз при запуске playbook.

match: strict

Вариант `match: strict` не требует, чтобы объект был в точности как указано в задаче, но команды, которые указаны в списке lines, должны быть в том же порядке.

Если указан список parents, команды в списке lines должны идти сразу за командами parents.

На маршрутизаторе такой ACL:

```
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit icmp any any
 deny   ip any any
```

Playbook 9_ios_config_match_strict.yml:

```
---

- name: Run cfg commands on router
  hosts: 192.168.100.1

  tasks:

    - name: Config ACL
      ios_config:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        before:
          - no ip access-list extended IN_to_OUT
        parents:
          - ip access-list extended IN_to_OUT
        lines:
          - permit tcp 10.0.1.0 0.0.0.255 any eq www
          - permit tcp 10.0.1.0 0.0.0.255 any eq 22
          - permit icmp any any
        match: strict
```

Выполнение playbook:

```
$ ansible-playbook 9_ios_config_match_strict.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] **************************************************************
ok: [192.168.100.1] => {"changed": false, "warnings": []}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
```

Так как изменений не было, ACL остался таким же.

В такой же ситуации, при использовании `match: exact`, было бы обнаружено изменение, и ACL бы состоял только из строк в списке lines.

match: none

Использование `match: none` отключает идемпотентность задачи: каждый раз при выполнении playbook будут отправляться команды, которые указаны в задаче.

Пример playbook 9_ios_config_match_none.yml:

```
---

- name: Run cfg commands on router
  hosts: 192.168.100.1

  tasks:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  - name: Config ACL
    ios_config:
      before:
        - no ip access-list extended IN_to_OUT
      parents:
        - ip access-list extended IN_to_OUT
      lines:
        - permit tcp 10.0.1.0 0.0.0.255 any eq www
        - permit tcp 10.0.1.0 0.0.0.255 any eq 22
        - permit icmp any any
      match: none
```

Каждый раз при запуске playbook результат будет таким:

```
$ ansible-playbook 9_ios_config_match_none.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ******************************************

TASK [Config ACL] **********************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["no ip access-list exten
ded IN_to_OUT", "ip access-list extended IN_to_OUT", "permit tcp 10.0.1.0 0.0.0.25
5 any eq www", "permit tcp 10.0.1.0 0.0.0.255 any eq 22", "permit icmp any any"],
"warnings": []}

PLAY RECAP *****************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

Использование `match: none` подходит в тех случаях, когда, независимо от текущей конфигурации, нужно отправить все команды.

replace

Параметр replace указывает, как именно нужно заменять конфигурацию:

- **line** - в этом режиме отправляются только те команды, которых нет в конфигурации. Этот режим используется по умолчанию
- **block** - в этом режиме отправляются все команды, если хотя бы одной команды нет

replace: line

Режим `replace: line` - это режим работы по умолчанию. В этом режиме, если были обнаружены изменения, отправляются только недостающие строки.

Например, на маршрутизаторе такой ACL:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit icmp any any
```

Попробуем запустить такой playbook 10_ios_config_replace_line.yml:

```
---

- name: Run cfg commands on router
  hosts: 192.168.100.1

  tasks:

    - name: Config ACL
      ios_config:
        before:
          - no ip access-list extended IN_to_OUT
        parents:
          - ip access-list extended IN_to_OUT
        lines:
          - permit tcp 10.0.1.0 0.0.0.255 any eq www
          - permit tcp 10.0.1.0 0.0.0.255 any eq 22
          - permit icmp any any
          - deny   ip any any
```

Выполнение playbook:

```
$ ansible-playbook 10_ios_config_replace_line.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] ***************************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["no ip access-list exten
ded IN_to_OUT", "ip access-list extended IN_to_OUT", "deny   ip any any"], "warnin
gs": []}

PLAY RECAP **********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

После этого на маршрутизаторе такой ACL:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 deny   ip any any
```

В данном случае модуль проверил, каких команд не хватает в ACL (так как режим по умолчанию match: line), обнаружил, что не хватает команды deny ip any any, и добавил её. Но, так как ACL сначала удаляется, а затем применяется список команд lines, получилось, что у нас теперь ACL с одной строкой.

В таких ситуациях подходит режим `replace: block`.

replace: block

В режиме `replace: block` отправляются все команды из списка lines (и parents), если на устройстве нет хотя бы одной из этих команд.

Повторим предыдущий пример.

ACL на маршрутизаторе:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit icmp any any
```

Playbook 10_ios_config_replace_block.yml:

```
---

- name: Run cfg commands on router
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  hosts: 192.168.100.1

  tasks:

    - name: Config ACL
      ios_config:
        before:
          - no ip access-list extended IN_to_OUT
        parents:
          - ip access-list extended IN_to_OUT
        lines:
          - permit tcp 10.0.1.0 0.0.0.255 any eq www
          - permit tcp 10.0.1.0 0.0.0.255 any eq 22
          - permit icmp any any
          - deny   ip any any
        replace: block
```

Выполнение playbook:

```
$ ansible-playbook 10_ios_config_replace_block.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ******************************************

TASK [Config ACL] **********************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["no ip access-list exten
ded IN_to_OUT", "ip access-list extended IN_to_OUT", "permit tcp 10.0.1.0 0.0.0.25
5 any eq www", "permit tcp 10.0.1.0 0.0.0.255 any eq 22", "permit icmp any any", "
deny   ip any any"], "warnings": []}

PLAY RECAP *****************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

В результате на маршрутизаторе такой ACL:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit icmp any any
 deny   ip any any
```

src

Параметр **src** позволяет указывать путь к файлу конфигурации или шаблону конфигурации, которую нужно загрузить на устройство.

Этот параметр взаимоисключающий с lines (то есть, можно указывать или lines, или src). Он заменяет модуль ios_template, который скоро будет удален.

Конфигурация

Пример playbook 11_ios_config_src.yml:

```
---

- name: Run cfg commands on router
  hosts: 192.168.100.1

  tasks:

    - name: Config ACL
      ios_config:
        src: templates/acl_cfg.txt
```

В файле templates/acl_cfg.txt находится такая конфигурация:

```
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit icmp any any
 deny   ip any any
```

Удаляем на маршрутизаторе этот ACL, если он остался с прошлых разделов, и запускаем playbook:

```
$ ansible-playbook 11_ios_config_src.yml -v
```

```
PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] ***************************************************************
changed: [192.168.100.1] => {"banners": {}, "changed": true, "commands": ["ip access-list e
xtended IN_to_OUT", "permit tcp 10.0.1.0 0.0.0.255 any eq www", "permit tcp 10.0.1.0 0.0.0.
255 any eq 22", "permit icmp any any", "deny   ip any any"], "failed": false, "updates": ["
ip access-list extended IN_to_OUT", "permit tcp 10.0.1.0 0.0.0.255 any eq www", "permit tcp
 10.0.1.0 0.0.0.255 any eq 22", "permit icmp any any", "deny   ip any any"]}

PLAY RECAP **********************************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
```

Теперь на маршрутизаторе настроен ACL:

```
R1#sh run | s access
ip access-list extended IN_to_OUT
 permit tcp 10.0.1.0 0.0.0.255 any eq www
 permit tcp 10.0.1.0 0.0.0.255 any eq 22
 permit icmp any any
 deny   ip any any
```

Если запустить playbook ещё раз, то никаких изменений не будет, так как этот параметр также идемпотентен:

```
$ ansible-playbook 11_ios_config_src.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Config ACL] **************************************************************
ok: [192.168.100.1] => {"changed": false, "warnings": []}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
```

Шаблон Jinja2

В параметре src можно указывать шаблон Jinja2.

Пример шаблона (файл templates/ospf.j2):

```
router ospf 1
 router-id {{ mgmnt_ip }}
 ispf
 auto-cost reference-bandwidth 10000
{% for ip in ospf_ints %}
 network {{ ip }} 0.0.0.0 area 0
{% endfor %}
```

В шаблоне используются две переменные:

- mgmnt_ip - IP-адрес, который будет использоваться как router-id
- ospf_ints - список IP-адресов интерфейсов, на которых нужно включить OSPF

Для настройки OSPF на трёх маршрутизаторах нужно иметь возможность использовать разные значения этих переменных для разных устройств. Для таких задач используются файлы с переменными в каталоге host_vars.

В каталоге host_vars нужно создать такие файлы (если они ещё не созданы):

Файл host_vars/192.168.100.1:

```
---

hostname: london_r1
mgmnt_loopback: 100
mgmnt_ip: 10.0.0.1
ospf_ints:
  - 192.168.100.1
  - 10.0.0.1
  - 10.255.1.1
```

Файл host_vars/192.168.100.2:

```
---

hostname: london_r2
mgmnt_loopback: 100
mgmnt_ip: 10.0.0.2
ospf_ints:
  - 192.168.100.2
  - 10.0.0.2
  - 10.255.2.2
```

Файл host_vars/192.168.100.3:

```
---

hostname: london_r3
mgmnt_loopback: 100
mgmnt_ip: 10.0.0.3
ospf_ints:
  - 192.168.100.3
  - 10.0.0.3
  - 10.255.3.3
```

Теперь можно создавать playbook 11_ios_config_src_jinja.yml:

```
---

- name: Run cfg commands on router
  hosts: cisco-routers

  tasks:

    - name: Config OSPF
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    ios_config:
      src: templates/ospf.j2
```

Так как Ansible сам найдет переменные в каталоге host_vars, их не нужно указывать. Можно сразу запускать playbook:

```
$ ansible-playbook 11_ios_config_src_jinja.yml -v
```

```
PLAY [Run cfg commands on router] ***********************************************

TASK [Config OSPF] *************************************************************
changed: [192.168.100.1] => {"banners": {}, "changed": true, "commands": ["router ospf 1",
"router-id 10.0.0.1", "ispf", "auto-cost reference-bandwidth 10000", "network 192.168.100.1
 0.0.0.0 area 0", "network 10.0.0.1 0.0.0.0 area 0", "network 10.255.1.1 0.0.0.0 area 0"],
"failed": false, "updates": ["router ospf 1", "router-id 10.0.0.1", "ispf", "auto-cost refe
rence-bandwidth 10000", "network 192.168.100.1 0.0.0.0 area 0", "network 10.0.0.1 0.0.0.0 a
rea 0", "network 10.255.1.1 0.0.0.0 area 0"]}
changed: [192.168.100.2] => {"banners": {}, "changed": true, "commands": ["router ospf 1",
"router-id 10.0.0.2", "ispf", "auto-cost reference-bandwidth 10000", "network 192.168.100.2
 0.0.0.0 area 0", "network 10.0.0.2 0.0.0.0 area 0", "network 10.255.2.2 0.0.0.0 area 0"],
"failed": false, "updates": ["router ospf 1", "router-id 10.0.0.2", "ispf", "auto-cost refe
rence-bandwidth 10000", "network 192.168.100.2 0.0.0.0 area 0", "network 10.0.0.2 0.0.0.0 a
rea 0", "network 10.255.2.2 0.0.0.0 area 0"]}
changed: [192.168.100.3] => {"banners": {}, "changed": true, "commands": ["router ospf 1",
"router-id 10.0.0.3", "ispf", "auto-cost reference-bandwidth 10000", "network 192.168.100.3
 0.0.0.0 area 0", "network 10.0.0.3 0.0.0.0 area 0", "network 10.255.3.3 0.0.0.0 area 0"],
"failed": false, "updates": ["router ospf 1", "router-id 10.0.0.3", "ispf", "auto-cost refe
rence-bandwidth 10000", "network 192.168.100.3 0.0.0.0 area 0", "network 10.0.0.3 0.0.0.0 a
rea 0", "network 10.255.3.3 0.0.0.0 area 0"]}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=1    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=1    unreachable=0    failed=0
```

Теперь на всех маршрутизаторах настроен OSPF:

```
R1#sh run | s ospf
router ospf 1
 router-id 10.0.0.1
 ispf
 auto-cost reference-bandwidth 10000
 network 10.0.0.1 0.0.0.0 area 0
 network 10.255.1.1 0.0.0.0 area 0
 network 192.168.100.1 0.0.0.0 area 0

R2#sh run | s ospf
router ospf 1
 router-id 10.0.0.2
```

(continues on next page)

(продолжение с предыдущей страницы)

```
 ispf
 auto-cost reference-bandwidth 10000
 network 10.0.0.2 0.0.0.0 area 0
 network 10.255.2.2 0.0.0.0 area 0
 network 192.168.100.2 0.0.0.0 area 0

router ospf 1
 router-id 10.0.0.3
 ispf
 auto-cost reference-bandwidth 10000
 network 10.0.0.3 0.0.0.0 area 0
 network 10.255.3.3 0.0.0.0 area 0
 network 192.168.100.3 0.0.0.0 area 0
```

Если запустить playbook ещё раз, то никаких изменений не будет:

```
$ ansible-playbook 11_ios_config_src_jinja.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on router] ******************************************

TASK [Config OSPF] ********************************************************
ok: [192.168.100.3] => {"changed": false, "warnings": []}
ok: [192.168.100.2] => {"changed": false, "warnings": []}
ok: [192.168.100.1] => {"changed": false, "warnings": []}

PLAY RECAP ****************************************************************
192.168.100.1              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.2              : ok=1    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=1    changed=0    unreachable=0    failed=0
```

Совмещение с другими параметрами

Параметр **src** совместим с такими параметрами:

- backup
- config
- defaults
- save

Дополнительные материалы

Ansible без привязки к сетевому оборудованию

- У Ansible очень хорошая документация
- Очень хорошая серия видео с транскриптом и хорошими ссылками
- Примеры использования Ansible
- Примеры Playbook с демонстрацией различных возможностей

Ansible for network devices

Документация:

- Gathering facts from network devices
- Interpreter Discovery
- Python 3 Support
- Networking Support
- Network Modules
- Network Debug and Troubleshooting Guide
- ios_command
- ios_facts
- ios_config

Новые состояния rescued и ignored:

- Blocks error handling
- Ignoring Failed Commands

Отличные видео от Ansible:

- AUTOMATING YOUR NETWORK. Репозиторий с примерами из вебинара

Проекты, которые используют TextFSM:

- Модуль ntc-ansible

Шаблоны TextFSM (из модуля ntc-ansible):

- ntc-templates

Статьи:

Примечание: Обращайте внимание на время написания статьи. В Ansible существенно изменились модули для работы с сетевым оборудованием. И в статьях могут быть ещё старые примеры.

Network Config Templating using Ansible (Kirk Byers):

- https://pynet.twb-tech.com/blog/ansible/ansible-cfg-template.html
- https://pynet.twb-tech.com/blog/ansible/ansible-cfg-template-p2.html
- https://pynet.twb-tech.com/blog/ansible/ansible-cfg-template-p3.html

Очень хорошая серия статей. Постепенно повышается уровень сложности:

- http://networkop.github.io/blog/2015/06/24/ansible-intro/
- http://networkop.github.io/blog/2015/07/03/parser-modules/
- http://networkop.github.io/blog/2015/07/10/test-verification/
- http://networkop.github.io/blog/2015/07/17/tdd-quickstart/
- http://networkop.github.io/blog/2015/08/14/automating-legacy-networks/
- http://networkop.github.io/blog/2015/08/26/automating-network-build-p1/
- http://networkop.github.io/blog/2015/09/03/automating-bgp-config/
- http://networkop.github.io/blog/2015/11/13/automating-flexvpn-config/
- http://jedelman.com/home/ansible-for-networking/
- http://jedelman.com/home/network-automation-with-ansible-dynamically-configuring-interface-descriptions/
- http://www.packetgeek.net/2015/08/using-ansible-to-push-cisco-ios-configurations/

4. Модули ресурсов

Ресурс - это часть конфигурации, например, интерфейсы или vlan’ы.

В этом разделе рассматриваются модули, которые появились в Ansible 2.9.

Модули:

- ios_lldp_global
- ios_lldp_intefaces
- ios_lacp
- ios_lacp_interfaces
- ios_vlans
- ios_interfaces
- ios_l2_interfaces
- ios_l3_interfaces
- ios_lag_interfaces

Подробнее о модулях ресурсов

Получение информации о ресурсах

Получить структурированную информацию о ресурсах можно с помощью модулей по сбору фактов, например, ios_facts. В модуле появился дополнительный параметр gather_network_resources, который позволяет получить информацию о ресурсах в одинаковом виде, независимо от платформы.

Пример получения информация о ресурсе interfaces с Cisco IOS (1_ios_facts_network_resources.yml):

```
- name: Collect IOS facts
  hosts: 192.168.100.1

  tasks:

    - name: Facts
      ios_facts:
        gather_subset: min
        gather_network_resources:
          - interfaces

    - name: Show ansible_network_resources
      debug: var=ansible_network_resources
```

Все ресурсы собираются в отдельную переменную ansible_network_resources, а уже внутри нее каждому ресурсу создан отдельный ключ.

```
ansible-playbook 1_ios_facts_network_resources.yml
```

```
PLAY [Collect IOS facts] ********************************************************

TASK [Facts] ********************************************************************
[WARNING]: default value for `gather_subset` will be changed to `min` from
`!config` v2.11 onwards
ok: [192.168.100.1]

TASK [Show ansible_network_resources] *******************************************
ok: [192.168.100.1] => {
    "ansible_network_resources": {
        "interfaces": [
            {
                "enabled": true,
                "name": "loopback0"
            },
            {
                "enabled": true,
                "name": "loopback35"
            },
            {
                "enabled": true,
                "name": "loopback55"
            },
            {
                "enabled": true,
                "name": "loopback90"
            },
            {
                "enabled": true,
                "name": "Ethernet0/0"
            },
```

Важный аспект получения информации о ресурсе - она одинаково структурирована для разных платформ.

ios_l3_interfaces

Playbook:

```
- name: Collect IOS facts
  hosts: 192.168.100.1

  vars:
    l3_intf: "{{ lookup('file', '192.168.100.1_l3_intf.json') | from_json }}"

  tasks:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    - name: Read data from file
      ios_l3_interfaces:
        config: "{{ l3_intf }}"
        state: deleted
      register: result

    - name: Show result
      debug: var=result
```

Файл 192.168.100.1_l3_intf:

```
[
    {
        "ipv4": [
            {
                "address": "4.4.4.4 255.255.255.255"
            }
        ],
        "name": "loopback0"
    },
    {
        "ipv4": [
            {
                "address": "5.5.5.5 255.255.255.255"
            }
        ],
        "name": "loopback55"
    },
    {
        "ipv4": [
            {
                "address": "90.1.1.1 255.255.255.255"
            }
        ],
        "name": "loopback90"
    }
]
```

До выполнения playbook

```
R1#show ip int bri
Interface                  IP-Address      OK? Method Status                Protocol
Ethernet0/0                192.168.100.1   YES NVRAM  up                    up
Ethernet0/1                192.168.200.1   YES NVRAM  up                    up
Loopback0                  4.4.4.4         YES manual up                    up
```

(continues on next page)

(продолжение с предыдущей страницы)

```
Loopback55                 5.5.5.5         YES manual up                    up
Loopback90                 90.1.1.1        YES manual up                    up
```

После выполнения playbook

```
R1#show ip int bri
Interface                  IP-Address      OK? Method Status                Protocol
Ethernet0/0                192.168.100.1   YES NVRAM  up                    up
Ethernet0/1                192.168.200.1   YES NVRAM  up                    up
Loopback0                  unassigned      YES manual up                    up
Loopback55                 unassigned      YES manual up                    up
Loopback90                 unassigned      YES manual up                    up
```

```
$ ansible-playbook 3_ios_l3_interfaces.yml

PLAY [Collect IOS facts] ********************************************************

TASK [Read data from file] ******************************************************
changed: [192.168.100.1]

TASK [Show result] **************************************************************
ok: [192.168.100.1] => {
    "result": {
        "after": [
            {
                "name": "loopback0"
            },
            {
                "name": "loopback55"
            },
            {
                "name": "loopback90"
            },
            {
                "ipv4": [
                    {
                        "address": "192.168.101.1 255.255.255.0",
                        "secondary": true
                    },
                    {
                        "address": "192.168.100.1 255.255.255.0"
                    }
                ],
                "name": "Ethernet0/0"
            },
```

(continues on next page)

(продолжение с предыдущей страницы)


```
            {
                "ipv4": [
                    {
                        "address": "192.168.200.1 255.255.255.0"
                    }
                ],
                "name": "Ethernet0/1"
            }
        ],
        "before": [
            {
                "ipv4": [
                    {
                        "address": "4.4.4.4 255.255.255.255"
                    }
                ],
                "name": "loopback0"
            },
            {
                "ipv4": [
                    {
                        "address": "5.5.5.5 255.255.255.255"
                    }
                ],
                "name": "loopback55"
            },
            {
                "ipv4": [
                    {
                        "address": "90.1.1.1 255.255.255.255"
                    }
                ],
                "name": "loopback90"
            },
            {
                "ipv4": [
                    {
                        "address": "192.168.101.1 255.255.255.0",
                        "secondary": true
                    },
                    {
                        "address": "192.168.100.1 255.255.255.0"
                    }
                ],
                "name": "Ethernet0/0"
```


(continues on next page)

(продолжение с предыдущей страницы)

```
            },
            {
                "ipv4": [
                    {
                        "address": "192.168.200.1 255.255.255.0"
                    }
                ],
                "name": "Ethernet0/1"
            }
        ],
        "changed": true,
        "commands": [
            "interface loopback0",
            "no ip address",
            "interface loopback55",
            "no ip address",
            "interface loopback90",
            "no ip address"
        ],
        "failed": false
    }
}

PLAY RECAP *********************************************************************
192.168.100.1: ok=2  changed=1  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

ios_vlans

Playbook 5_ios_vlans.yml

```
- name: Collect IOS facts
  hosts: 192.168.100.100

  tasks:

    - name: Configure vlans
      ios_vlans:
        config:
          - name: Vlan_10
            vlan_id: 10
          - name: Vlan_20
            vlan_id: 20
        state: merged
```

(continues on next page)

(продолжение с предыдущей страницы)

```
      register: result

    - name: Show result
      debug: var=result
```

Вывод до:

```
SW1#sh vlan br

VLAN Name                              Status    Ports
---- --------------------------------- --------- -------------------------------
1    default                           active    Et0/0, Et0/1, Et0/2, Et0/3
                                                 Et1/0, Et1/1, Et1/2, Et1/3
                                                 Et2/0, Et2/1, Et2/2, Et2/3
                                                 Et3/0, Et3/1, Et3/2, Et3/3
10   VLAN0010                          active
1002 fddi-default                      act/unsup
1003 token-ring-default                act/unsup
1004 fddinet-default                   act/unsup
1005 trnet-default                     act/unsup
```

После:

```
SW1#sh vlan br

VLAN Name                              Status    Ports
---- --------------------------------- --------- -------------------------------
1    default                           active    Et0/0, Et0/1, Et0/2, Et0/3
                                                 Et1/0, Et1/1, Et1/2, Et1/3
                                                 Et2/0, Et2/1, Et2/2, Et2/3
                                                 Et3/0, Et3/1, Et3/2, Et3/3
10   Vlan_10                           active
20   Vlan_20                           active
1002 fddi-default                      act/unsup
1003 token-ring-default                act/unsup
1004 fddinet-default                   act/unsup
1005 trnet-default                     act/unsup
```

Выполнение playbook:

```
$ ansible-playbook 5_ios_vlans.yml

PLAY [Collect IOS facts] ********************************************************

TASK [Configure vlans] **********************************************************
changed: [192.168.100.100]
```

(continues on next page)

(продолжение с предыдущей страницы)

```
TASK [Show result] ********************************************************
ok: [192.168.100.100] => {
    "result": {
        "after": [
            {
                "mtu": 1500,
                "name": "default",
                "shutdown": "disabled",
                "state": "active",
                "vlan_id": 1
            },
            {
                "mtu": 1500,
                "name": "Vlan_10",
                "shutdown": "disabled",
                "state": "active",
                "vlan_id": 10
            },
            {
                "mtu": 1500,
                "name": "Vlan_20",
                "shutdown": "disabled",
                "state": "active",
                "vlan_id": 20
            }
        ],
        "before": [
            {
                "mtu": 1500,
                "name": "default",
                "shutdown": "disabled",
                "state": "active",
                "vlan_id": 1
            },
            {
                "mtu": 1500,
                "name": "VLAN0010",
                "shutdown": "disabled",
                "state": "active",
                "vlan_id": 10
            }
        ],
        "changed": true,
        "commands": [
```

(continues on next page)

(продолжение с предыдущей страницы)

```
            "vlan 10",
            "name Vlan_10",
            "vlan 20",
            "name Vlan_20"
        ],
        "failed": false
    }
}

PLAY RECAP *********************************************************************
192.168.100.100: ok=2  changed=1  unreachable=0  failed=0  skipped=0  rescued=0  ignored=0
```

Дополнительные материалы

Документация:

- Network resource modules
- Network features coming soon in Ansible Engine 2.9

5. Получение структурированного вывода

В этом разделе рассматриваются варианты получения структурированного вывода с устройств. В Ansible есть встроенные варианты, такие как факты и ресурсы, но иногда их недостаточно и тогда нужен какой-то способ обработать вывод команды, например, регулярным выражением.

В этом разделе рассматривается как обработать вывод и получить структурированные данные с помощью регулярных выражений и textfsm.

ntc-ansible

ntc-ansible - это модуль для работы с сетевым оборудованием, который не только выполняет команды на оборудовании, но и обрабатывает вывод команд и преобразует с помощью TextFSM. Этот модуль не входит в число core модулей Ansible, поэтому его нужно установить.

Перед установкой нужно указать Ansible, где искать сторонние модули. Указывается путь в файле ansible.cfg:

```
[defaults]

inventory = ./myhosts

remote_user = cisco
ask_pass = True

library = ./library
```

После этого нужно клонировать репозиторий ntc-ansible, находясь в каталоге library:

```
[~/pyneng_course/chapter15/library]
$ git clone https://github.com/networktocode/ntc-ansible --recursive
```

(continues on next page)

(продолжение с предыдущей страницы)

```
Cloning into 'ntc-ansible'...
remote: Counting objects: 2063, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 2063 (delta 1), reused 0 (delta 0), pack-reused 2058
Receiving objects: 100% (2063/2063), 332.15 KiB | 334.00 KiB/s, done.
Resolving deltas: 100% (1157/1157), done.
Checking connectivity... done.
Submodule 'ntc-templates' (https://github.com/networktocode/ntc-templates) registered for␣
↪path 'ntc-templates'
Cloning into 'ntc-templates'...
remote: Counting objects: 902, done.
remote: Compressing objects: 100% (34/34), done.
remote: Total 902 (delta 16), reused 0 (delta 0), pack-reused 868
Receiving objects: 100% (902/902), 161.11 KiB | 0 bytes/s, done.
Resolving deltas: 100% (362/362), done.
Checking connectivity... done.
Submodule path 'ntc-templates': checked out '89c57342b47c9990f0708226fb3f268c6b8c1549'
```

А затем установить зависимости модуля:

```
pip install ntc-ansible
```

При установке зависимостей может появиться ошибка:

```
No matching distribution found for textfsm==1.0.1 (from pyntc->ntc-ansible)
```

Ее можно игнорировать, если модуль textfsm установлен.

> Если при установке возникнут другие проблемы, посмотрите другие варианты установки в репозитории проекта.

Так как в текущей версии Ansible уже есть модули, которые работают с сетевым оборудованием и позволяют выполнять команды, из всех возможностей ntc-ansible наиболее полезной будет отправка команд show и получение структурированного вывода. За это отвечает модуль ntc_show_command.

ntc_show_command

Модуль использует netmiko для подключения к оборудованию (netmiko должен быть установлен) и, после выполнения команды, преобразует вывод команды show с помощью TextFSM в структурированный вывод (список словарей).

Преобразование будет выполняться в том случае, если в файле index была найдена команда, и для команды был найден шаблон.

Как и с предыдущими сетевыми модулями, в ntc-ansible нужно указывать ряд параметров для подключения:

- **connection** - тут возможны два варианта: ssh (подключение netmiko) или offline (чтение из файла для тестовых целей)
- **platform** - платформа, которая существует в index файле (library/ntc-ansible/ntc-templates/templates/index)
- **command** - команда, которую нужно выполнить на устройстве
- **host** - IP-адрес или имя устройства
- **username** - имя пользователя
- **password** - пароль
- **template_dir** - путь к каталогу, в котором находятся шаблоны (в текущем варианте установки они находятся в каталоге library/ntc-ansible/ntc-templates/templates

Пример playbook 1_ntc_ansible.yml:

```
---

- name: Run show commands on router
  hosts: 192.168.100.1
  gather_facts: false
  connection: local

  tasks:

    - name: Run sh ip int br
      ntc_show_command:
        connection: ssh
        platform: "cisco_ios"
        command: "sh ip int br"
        host: "{{ inventory_hostname }}"
        username: "cisco"
        password: "cisco"
        template_dir: "library/ntc-ansible/ntc-templates/templates"
      register: result

    - debug: var=result
```

Результат выполнения playbook:

```
$ ansible-playbook 1_ntc-ansible.yml
```

```
SSH password:

PLAY [Run show commands on router] ********************************************

TASK [Run sh ip int br] *******************************************************
ok: [192.168.100.1]

TASK [debug] ******************************************************************
ok: [192.168.100.1] => {
    "result": {
        "changed": false,
        "response": [
            {
                "intf": "Ethernet0/0",
                "ipaddr": "192.168.100.1",
                "proto": "up",
                "status": "up"
            },
            {
                "intf": "Ethernet0/1",
                "ipaddr": "192.168.200.1",
                "proto": "up",
                "status": "up"
            },
            {
                "intf": "Ethernet0/2",
                "ipaddr": "unassigned",
                "proto": "down",
                "status": "administratively down"
            },
            {
                "intf": "Ethernet0/3",
                "ipaddr": "unassigned",
                "proto": "up",
                "status": "up"
            },
            {
                "intf": "Loopback0",
                "ipaddr": "10.1.1.1",
                "proto": "up",
                "status": "up"
            }
        ],
        "response_list": []
    }
}

PLAY RECAP ********************************************************************
192.168.100.1              : ok=2    changed=0    unreachable=0    failed=0
```

В переменной response находится структурированный вывод в виде списка словарей. Ключи в словарях получены на основании переменных, которые описаны в шаблоне library/ntc-ansible/ntc-templates/templates/cisco_ios_show_ip_int_brief.template (единственное отличие - регистр):

```
Value INTF (\S+)
Value IPADDR (\S+)
Value STATUS (up|down|administratively down)
Value PROTO (up|down)

Start
  ^${INTF}\s+${IPADDR}\s+\w+\s+\w+\s+${STATUS}\s+${PROTO} -> Record
```

Для того, чтобы получить вывод про первый интерфейс, можно поменять вывод модуля debug таким образом:

```
- debug: var=result.response[0]
```

Сохранение результатов выполнения команды

Для того, чтобы сохранить вывод, можно использовать тот же прием, который использовался для модуля ios_facts.

Пример playbook 2_ntc_ansible_save.yml с сохранением результатов команды:

```
---

- name: Run show commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Run sh ip int br
      ntc_show_command:
        connection: ssh
        platform: "cisco_ios"
        command: "sh ip int br"
        host: "{{ inventory_hostname }}"
        username: "cisco"
        password: "cisco"
        template_dir: "library/ntc-ansible/ntc-templates/templates"
      register: result

    - name: Copy facts to files
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    copy:
      content: "{{ result.response | to_nice_json }}"
      dest: "all_facts/{{inventory_hostname}}_sh_ip_int_br.json"
```

Результат выполнения:

```
$ ansible-playbook 2_ntc-ansible_save.yml
```

```
SSH password:

PLAY [Run show commands on routers] ********************************************

TASK [Run sh ip int br] ********************************************************
ok: [192.168.100.3]
ok: [192.168.100.1]
ok: [192.168.100.2]

TASK [Copy facts to files] *****************************************************
changed: [192.168.100.2]
changed: [192.168.100.1]
changed: [192.168.100.3]

PLAY RECAP *********************************************************************
192.168.100.1              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.3              : ok=2    changed=1    unreachable=0    failed=0
```

В результате, в каталоге all_facts появляются соответствующие файлы для каждого маршрутизатора. Пример файла all_facts/192.168.100.1_sh_ip_int_br.json:

```
[
    {
        "intf": "Ethernet0/0",
        "ipaddr": "192.168.100.1",
        "proto": "up",
        "status": "up"
    },
    {
        "intf": "Ethernet0/1",
        "ipaddr": "192.168.200.1",
        "proto": "up",
        "status": "up"
    },
    {
        "intf": "Ethernet0/2",
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        "ipaddr": "unassigned",
        "proto": "down",
        "status": "administratively down"
    },
    {
        "intf": "Ethernet0/3",
        "ipaddr": "unassigned",
        "proto": "up",
        "status": "up"
    },
    {
        "intf": "Loopback0",
        "ipaddr": "10.1.1.1",
        "proto": "up",
        "status": "up"
    }
]
```

Шаблоны Jinja2

Для Cisco IOS в ntc-ansible есть такие шаблоны:

```
cisco_ios_dir.template
cisco_ios_show_access-list.template
cisco_ios_show_aliases.template
cisco_ios_show_archive.template
cisco_ios_show_capability_feature_routing.template
cisco_ios_show_cdp_neighbors_detail.template
cisco_ios_show_cdp_neighbors.template
cisco_ios_show_clock.template
cisco_ios_show_interfaces_status.template
cisco_ios_show_interfaces.template
cisco_ios_show_interface_transceiver.template
cisco_ios_show_inventory.template
cisco_ios_show_ip_arp.template
cisco_ios_show_ip_bgp_summary.template
cisco_ios_show_ip_bgp.template
cisco_ios_show_ip_int_brief.template
cisco_ios_show_ip_ospf_neighbor.template
cisco_ios_show_ip_route.template
cisco_ios_show_lldp_neighbors.template
cisco_ios_show_mac-address-table.template
cisco_ios_show_processes_cpu.template
cisco_ios_show_snmp_community.template
```

(continues on next page)

(продолжение с предыдущей страницы)

```
cisco_ios_show_spanning-tree.template
cisco_ios_show_standby_brief.template
cisco_ios_show_version.template
cisco_ios_show_vlan.template
cisco_ios_show_vtp_status.template
```

Список всех шаблонов можно посмотреть локально, если ntc-ansible установлен:

```
ls -ls library/ntc-ansible/ntc-templates/templates/
```

Или в репозитории проекта.

Используя TextFSM, можно самостоятельно создавать дополнительные шаблоны.

И для того, чтобы ntc-ansible их использовал автоматически, добавить их в файл index (library/ntc-ansible/ntc-templates/templates/index):

```
# First line is the header fields for columns and is mandatory.
# Regular expressions are supported in all fields except the first.
# Last field supports variable length command completion.
# abc[[xyz]] is expanded to abc(x(y(z)?)?)?, regexp inside [[]] is not supported
#
Template, Hostname, Platform, Command
cisco_asa_dir.template,  .*, cisco_asa, dir
cisco_ios_show_archive.template,  .*, cisco_ios, sh[[ow]] arc[[hive]]
cisco_ios_show_capability_feature_routing.template,  .*, cisco_ios, sh[[ow]]␣
↪cap[[ability]] f[[eature]] r[[outing]]
cisco_ios_show_aliases.template,  .*, cisco_ios, sh[[ow]] alia[[ses]]
...
```

6. Playbook

В прошлых разделах мы разобрались с основами playbook. В этом разделе мы разберемся с другими возможностями playbook.

Для работы с Ansible достаточно использовать базовый функционал. И, по мере использования, вы можете обращаться к этим разделам, когда потребуется добавить более сложный функционал.

> Также не забывайте о документации Ansible. Она очень хорошо написана и в документации вы найдете больше информации по этим темам.

В этой части мы рассмотрим:

- handlers - специальные задачи, которые можно вызывать из обычных задач. Например, с помощью handlers можно выполнять сохранение конфигурации.
- include - способ добавлять задачи, сценарии или переменные из файлов в текущий playbook.
- роли - способ разбития playbook на логические части.
- фильтры и тесты Jinja2 - позволяют делать проверки.
- условия - позволяют указывать в каком случае задача должна выполняться.
- циклы - с помощью циклов можно передавать несколько групп переменных, которые будут подставляться в задачу.

Handlers

Handlers - это специальные задачи. Они вызываются из других задач ключевым словом **notify**.

Эти задачи срабатывают после выполнения всех задач в сценарии (play). При этом, если несколько задач вызвали одну и ту же задачу через notify, она выполниться только один раз.

Handlers описываются в своем подразделе playbook - handlers, так же, как и задачи. Для них используется такой же синтаксис, как и для задач.

Пример использования handlers (playbook 8_handlers.yml):

```
---

- name: Run cfg commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Config line vty
      ios_config:
        parents:
          - line vty 0 4
        lines:
          - login local
          - transport input ssh
        provider: "{{ cli }}"
      notify: save config

    - name: Send config commands
      ios_config:
        lines:
          - service password-encryption
          - no ip http server
          - no ip http secure-server
          - no ip domain lookup
        provider: "{{ cli }}"
      notify: save config

  handlers:

    - name: save config
      ios_command:
        commands:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
        - write
      provider: "{{ cli }}"
```

Запуск playbook с изменениями:

```
$ ansible-playbook 8_handlers.yml
```

```
SSH password:

PLAY [Run cfg commands on routers] ********************************************

TASK [Config line vty] ********************************************************
changed: [192.168.100.3]
changed: [192.168.100.2]
changed: [192.168.100.1]

TASK [Send config commands] ****************************************************
changed: [192.168.100.1]
changed: [192.168.100.3]
changed: [192.168.100.2]

RUNNING HANDLER [save config] **************************************************
ok: [192.168.100.2]
ok: [192.168.100.3]
ok: [192.168.100.1]

PLAY RECAP ********************************************************************
192.168.100.1              : ok=3    changed=2    unreachable=0    failed=0
192.168.100.2              : ok=3    changed=2    unreachable=0    failed=0
192.168.100.3              : ok=3    changed=2    unreachable=0    failed=0
```

Обратите внимание, что handler выполняется только один раз.

Запуск того же playbook с изменениями и режимом verbose:

```
$ ansible-playbook 8_handlers.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on routers] *********************************************

TASK [Config line vty] *********************************************************
changed: [192.168.100.3] => {"changed": true, "updates": ["line vty 0 4", "transp
ort input ssh"], "warnings": []}
changed: [192.168.100.2] => {"changed": true, "updates": ["line vty 0 4", "transp
ort input ssh"], "warnings": []}
changed: [192.168.100.1] => {"changed": true, "updates": ["line vty 0 4", "transp
ort input ssh"], "warnings": []}

TASK [Send config commands] ****************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["service password-encry
ption", "no ip http server", "no ip http secure-server", "no ip domain lookup"],
"warnings": []}
changed: [192.168.100.3] => {"changed": true, "updates": ["service password-encry
ption", "no ip http server", "no ip http secure-server", "no ip domain lookup"],
"warnings": []}
changed: [192.168.100.2] => {"changed": true, "updates": ["service password-encry
ption", "no ip http server", "no ip http secure-server", "no ip domain lookup"],
"warnings": []}

RUNNING HANDLER [save config] **************************************************
ok: [192.168.100.1] => {"changed": false, "stdout": ["Building configuration...\n
[OK]"], "stdout_lines": [["Building configuration...", "[OK]"]], "warnings": []}
ok: [192.168.100.2] => {"changed": false, "stdout": ["Building configuration...\n
[OK]"], "stdout_lines": [["Building configuration...", "[OK]"]], "warnings": []}
ok: [192.168.100.3] => {"changed": false, "stdout": ["Building configuration...\n
[OK]"], "stdout_lines": [["Building configuration...", "[OK]"]], "warnings": []}

PLAY RECAP *********************************************************************
192.168.100.1              : ok=3    changed=2    unreachable=0    failed=0
192.168.100.2              : ok=3    changed=2    unreachable=0    failed=0
192.168.100.3              : ok=3    changed=2    unreachable=0    failed=0
```

Запуск playbook без изменений:

```
$ ansible-playbook 8_handlers.yml
```

```
SSH password:

PLAY [Run cfg commands on routers] ******************************************

TASK [Config line vty] ******************************************************
ok: [192.168.100.3]
ok: [192.168.100.1]
ok: [192.168.100.2]

TASK [Send config commands] **************************************************
ok: [192.168.100.2]
ok: [192.168.100.1]
ok: [192.168.100.3]

PLAY RECAP ******************************************************************
192.168.100.1              : ok=2    changed=0    unreachable=0    failed=0
192.168.100.2              : ok=2    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=2    changed=0    unreachable=0    failed=0
```

Так как в задачах не нужно выносить изменений, handler также не выполняется.

Include

До сих пор, каждый playbook был отдельным файлом. И, хотя для простых сценариев, такой вариант подходит, когда задач становится больше, может понадобиться выполнять одни и те же действия в разных playbook. И было бы намного удобней, если бы можно было разбить playbook на блоки, которые можно повторно использовать (как в случае с функциями).

Это можно сделать с помощью выражений include (и с помощью ролей, которые мы будем рассматриваться в следующем разделе).

С помощью выражения include, в playbook можно добавлять:

- задачи
- handlers
- сценарий (play)
- playbook
- файлы с переменными (используют другое ключевое слово)

Task include

Task include позволяют подключать в текущий playbook файлы с задачами.

Например, создадим каталог tasks и добавим в него два файла с задачами.

Файл tasks/cisco_vty_cfg.yml:

```
---

- name: Config line vty
  ios_config:
    parents:
      - line vty 0 4
    lines:
      - exec-timeout 30 0
      - login local
      - history size 100
      - transport input ssh
    provider: "{{ cli }}"
  notify: save config
```

Файл tasks/cisco_ospf_cfg.yml:

```
---

- name: Config ospf
  ios_config:
    src: templates/ospf.j2
    provider: "{{ cli }}"
  notify: save config
```

Шаблон templates/ospf.j2 (переменные, которые используются в шаблоне, находятся в файлах с переменными для каждого устройства, в каталоге host_vars):

```
router ospf 1
 router-id {{ mgmnt_ip }}
 ispf
 auto-cost reference-bandwidth 10000
{% for ip in ospf_ints %}
 network {{ ip }} 0.0.0.0 area 0
{% endfor %}
```

Теперь создадим playbook, который будет использовать созданные файлы с задачами.

Playbook 8_playbook_include_tasks.yml:

```
---

- name: Run cfg commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Disable services
      ios_config:
        lines:
          - no ip http server
          - no ip http secure-server
          - no ip domain lookup
        provider: "{{ cli }}"
      notify: save config

    - include: tasks/cisco_ospf_cfg.yml
    - include: tasks/cisco_vty_cfg.yml

  handlers:

    - name: save config
      ios_command:
        commands:
          - write
        provider: "{{ cli }}"
```

В этом playbook специально создана обычная задача. А также handler, который мы использовали в предыдущем разделе. Он вызывается и из задачи, которая находится в playbook, и из задач в подключаемых файлах.

Обратите внимание, что строки include находятся на том же уровне, что и задача.

В конфигурации R1 внесены изменения, чтобы playbook мог выполнить конфигурацию устройства.

Запуск playbook с изменениями:

```
$ ansible-playbook 8_playbook_include_tasks.yml
```

```
SSH password:

PLAY [Run cfg commands on routers] *********************************************

TASK [Disable services] ********************************************************
changed: [192.168.100.1]
ok: [192.168.100.2]
ok: [192.168.100.3]

TASK [Config ospf] *************************************************************
ok: [192.168.100.3]
ok: [192.168.100.1]
ok: [192.168.100.2]

TASK [Config line vty] *********************************************************
ok: [192.168.100.2]
ok: [192.168.100.3]
changed: [192.168.100.1]

RUNNING HANDLER [save config] **************************************************
ok: [192.168.100.1]

PLAY RECAP *********************************************************************
192.168.100.1              : ok=4    changed=2    unreachable=0    failed=0
192.168.100.2              : ok=3    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=3    changed=0    unreachable=0    failed=0
```

При выполнении playbook, задачи которые мы добавили через include работают так же, как если бы они находились в самом playbook.

Таким образом мы можем делать отдельные файлы с задачами, которые настраивают определенную функциональность, а затем собирать их в нужной комбинации в итоговом playbook.

Передача переменных в include

При использовании include, задачам можно передавать аргументы.

Например, когда мы использовали команду ntc_show_command из модуля ntc-ansible, нужно было задать ряд параметров. Так как они не вынесены в отдельную переменную, как в случае с модулями ios_config, ios_command и ios_facts, довольно не удобно каждый раз их описывать.

Попробуем вынести задачу с использованием ntc_show_command в отдельный файл tasks/ntc_show.yml:

```
---

- ntc_show_command:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    connection: ssh
    platform: "cisco_ios"
    command: "{{ ntc_command }}"
    host: "{{ inventory_hostname }}"
    username: "cisco"
    password: "cisco"
    template_dir: "library/ntc-ansible/ntc-templates/templates"
```

В этом файле указаны две переменные: ntc_command и inventory_hostname. С переменной inventory_hostname мы уже сталкивались раньше, она автоматически становится равной текущеву устройству, для которого Ansible выполняет задачу.

А значение переменной ntc_command мы будем передавать из playbook.

Playbook 8_playbook_include_tasks_var.yml:

```
---

- name: Run cfg commands on routers
  hosts: 192.168.100.1
  gather_facts: false
  connection: local

  tasks:

    - include: tasks/cisco_ospf_cfg.yml
    - include: tasks/ntc_show.yml ntc_command="sh ip route"

  handlers:

    - name: save config
      ios_command:
        commands:
          - write
        provider: "{{ cli }}"
```

В таком варианте, нам достаточно указать какую команду передать ntc_show_command.

Переменные можно передавать и таким образом:

```
tasks:

  - include: tasks/cisco_ospf_cfg.yml
  - include: tasks/ntc_show.yml
    vars:
      ntc_command: "sh ip route"
```

Такой вариант удобнее, когда вам нужно передать несколько переменных.

Handler include

Include можно использовать и в разделе handlers.

Например, перенесем handler из предыдущих примеров в отдельный файл handlers/cisco_save_cfg.yml:

```
---

- name: save config
  ios_command:
    commands:
      - write
    provider: "{{ cli }}"
```

И добавим его в playbook 8_playbook_include_handlers.yml через include:

```
---

- name: Run cfg commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Disable services
      ios_config:
        lines:
          - no ip http server
          - no ip http secure-server
          - no ip domain lookup
        provider: "{{ cli }}"
      notify: save config

    - include: tasks/cisco_ospf_cfg.yml
    - include: tasks/cisco_vty_cfg.yml

  handlers:

    - include: handlers/cisco_save_cfg.yml
```

Запуск playbook:

```
$ ansible-playbook 8_playbook_include_handlers.yml -v
```

```
Using /home/nata/pyneng_course/chapter15/ansible.cfg as config file
SSH password:

PLAY [Run cfg commands on routers] ***************************************

TASK [Disable services] *************************************************
changed: [192.168.100.1] => {"changed": true, "updates": ["no ip http server"], "
warnings": []}
ok: [192.168.100.3] => {"changed": false, "warnings": []}
ok: [192.168.100.2] => {"changed": false, "warnings": []}

TASK [Config ospf] ******************************************************
ok: [192.168.100.2] => {"changed": false, "warnings": []}
ok: [192.168.100.3] => {"changed": false, "warnings": []}
changed: [192.168.100.1] => {"changed": true, "warnings": []}

TASK [Config line vty] **************************************************
ok: [192.168.100.3] => {"changed": false, "warnings": []}
ok: [192.168.100.2] => {"changed": false, "warnings": []}
changed: [192.168.100.1] => {"changed": true, "updates": ["line vty 0 4", "transp
ort input ssh"], "warnings": []}

RUNNING HANDLER [save config] *******************************************
ok: [192.168.100.1] => {"changed": false, "stdout": ["Building configuration...\n
[OK]"], "stdout_lines": [["Building configuration...", "[OK]"]], "warnings": []}

PLAY RECAP **************************************************************
192.168.100.1              : ok=4    changed=3    unreachable=0    failed=0
192.168.100.2              : ok=3    changed=0    unreachable=0    failed=0
192.168.100.3              : ok=3    changed=0    unreachable=0    failed=0
```

Playbook выполняет handler, как-будто он находится в playbook. Таким образом можно легко добавлять handler в любой playbook.

Play/playbook include

С помощью выражения include можно добавить в playbook и целый сценарий (play) или другой playbook. От добавления задач это будет отличаться только уровнем, на котором выполняется include.

Например, у нас есть такой сценарий 8_play_to_include.yml:

```
---

- name: Run show commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: run show commands
      ios_command:
        commands:
          - show ip int br
          - sh ip route
        provider: "{{ cli }}"
      register: show_result

    - name: Debug registered var
      debug: var=show_result.stdout_lines
```

Добавим его в playbook 8_playbook_include_play.yml:

```
---

- name: Run cfg commands on routers
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Disable services
      ios_config:
        lines:
          - no ip http server
          - no ip http secure-server
          - no ip domain lookup
        provider: "{{ cli }}"
      notify: save config

    - include: tasks/cisco_ospf_cfg.yml
    - include: tasks/cisco_vty_cfg.yml

  handlers:

    - include: handlers/cisco_save_cfg.yml
```

(continues on next page)

(продолжение с предыдущей страницы)

```
- include: 8_play_to_include.yml
```

Если выполнить playbook, то все задачи из файла 8_play_to_include.yml выполняются точно так же, как и те, которые находятся в playbook (вывод сокращен):

```
$ ansible-playbook 8_playbook_include_play.yml
```

```
SSH password:

PLAY [Run cfg commands on routers] ***************************************

TASK [Disable services] **************************************************
ok: [192.168.100.2]
ok: [192.168.100.1]
ok: [192.168.100.3]

TASK [Config ospf] *******************************************************
ok: [192.168.100.1]
ok: [192.168.100.3]
ok: [192.168.100.2]

TASK [Config line vty] ***************************************************
ok: [192.168.100.1]
ok: [192.168.100.2]
ok: [192.168.100.3]

PLAY [Run show commands on routers] **************************************

TASK [run show commands] *************************************************
ok: [192.168.100.1]
ok: [192.168.100.3]
ok: [192.168.100.2]

TASK [Debug registered var] **********************************************
ok: [192.168.100.1] => {
    "show_result.stdout_lines": [
        [
            "Interface                  IP-Address      OK? Method Status                Protocol",
            "Ethernet0/0                192.168.100.1   YES NVRAM  up                    up      ",
            "Ethernet0/1                192.168.200.1   YES NVRAM  up                    up      ",
            "Ethernet0/2                unassigned      YES manual administratively down down    ",
            "Ethernet0/3                unassigned      YES manual up                    up      ",
            "Loopback0                  10.10.1.1       YES manual up                    up      "
        ],
```

Vars include

Несмотря на то, что файлы с переменными могут быть вынесены в каталоги host_vars и group_vars, и разбиты на части, которые относятся ко всем устройствам, к группе или к конкретному устройству, иногда не хватает этой иерархии и файлы с переменными становятся слишком большими. Но и тут Ansible поддерживает возможность создавать дополнительную иерархию.

Можно создавать отдельные файлы с переменными, которые будут относиться, например, к настройке определенного функционала.

include_vars

Например, создадим каталог vars и добавим в него файл vars/cisco_bgp_general.yml

```
---

as: 65000
network: 120.0.0.0 mask 255.255.252.0
ttl_security_hops: 3
send_community: true
update_source_int: Loopback0
ibgp_neighbors:
  - 10.0.0.1
  - 10.0.0.2
  - 10.0.0.3
  - 10.0.0.4
ebgp_neighbors:
  - ip: 15.0.0.5
    as: 500
  - ip: 26.0.0.6
    as: 600
```

Переменные будем использовать для генерации конфигурации BGP по шаблону templates/bgp.j2:

```
router bgp {{ as }}
 network {{ network }}
 {% for n in ibgp_neighbors %}
 neighbor {{ n }} remote-as {{ as }}
 neighbor {{ n }} update-source {{ update_source_int }}
 {% endfor %}
 {% for extn in ebgp_neighbors %}
 neighbor {{ extn.ip }} remote-as {{ extn.as }}
 neighbor {{ extn.ip }} ttl-security hops {{ ttl_security_hops }}
 {% if send_community == true %}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
 neighbor {{ extn.ip }} send-community
 {% endif %}
 {% endfor %}

Шаблон подразумевает настройку одного маршрутизатора, просто чтобы
показать как добавлять переменные из файла.
```

Итоговый playbook 8_playbook_include_vars.yml

```
---

- name: Run cfg commands on router
  hosts: 192.168.100.1
  gather_facts: false
  connection: local

  tasks:

    - name: Include BGP vars
      include_vars: vars/cisco_bgp_general.yml

    - name: Config BGP
      ios_config:
        src: templates/bgp.j2
        provider: "{{ cli }}"

    - name: Show BGP config
      ios_command:
        commands: sh run | s ^router bgp
        provider: "{{ cli }}"
      register: bgp_cfg

    - name: Debug registered var
      debug: var=bgp_cfg.stdout_lines
```

Обратите внимание, что переменные из файла подключаются отдельной задачей (в данном случае, можно было бы обойтись без имени задачи):

```
- name: Include BGP vars
  include_vars: vars/cisco_bgp_general.yml
```

Выполнение playbook выглядит так:

```
$ ansible-playbook 8_playbook_include_vars.yml
```

```
SSH password:

PLAY [Run cfg commands on router] ***********************************************

TASK [Include BGP vars] *********************************************************
ok: [192.168.100.1]

TASK [Config BGP] ***************************************************************
changed: [192.168.100.1]

TASK [Show BGP config] **********************************************************
ok: [192.168.100.1]

TASK [Debug registered var] *****************************************************
ok: [192.168.100.1] => {
    "bgp_cfg.stdout_lines": [
        [
            "router bgp 65000",
            " bgp log-neighbor-changes",
            " network 120.0.0.0 mask 255.255.252.0",
            " neighbor 10.0.0.1 remote-as 65000",
            " neighbor 10.0.0.1 update-source Loopback0",
            " neighbor 10.0.0.2 remote-as 65000",
            " neighbor 10.0.0.2 update-source Loopback0",
            " neighbor 10.0.0.3 remote-as 65000",
            " neighbor 10.0.0.3 update-source Loopback0",
            " neighbor 10.0.0.4 remote-as 65000",
            " neighbor 10.0.0.4 update-source Loopback0",
            " neighbor 15.0.0.5 remote-as 500",
            " neighbor 15.0.0.5 ttl-security hops 3",
            " neighbor 15.0.0.5 send-community",
            " neighbor 26.0.0.6 remote-as 600",
            " neighbor 26.0.0.6 ttl-security hops 3",
            " neighbor 26.0.0.6 send-community"
        ]
    ]
}

PLAY RECAP **********************************************************************
192.168.100.1              : ok=4    changed=1    unreachable=0    failed=0
```

Модуль include_vars поддерживает большое количество вариантов использования. Подробнее об этом можно почитать в документации модуля.

vars_files

Второй вариант добавления файлов с переменными - использование vars_files.

Его отличие в том, что мы создаем переменные на уровне сценария (play), а не на уровне задаче.

Пример playbook 8_playbook_include_vars_files.yml:

```
---

- name: Run cfg commands on router
  hosts: 192.168.100.1
  gather_facts: false
  connection: local

  vars_files:
    - vars/cisco_bgp_general.yml

  tasks:

    - name: Config BGP
      ios_config:
        src: templates/bgp.j2
        provider: "{{ cli }}"

    - name: Show BGP config
      ios_command:
        commands: sh run | s ^router bgp
        provider: "{{ cli }}"
      register: bgp_cfg

    - name: Debug registered var
      debug: var=bgp_cfg.stdout_lines
```

Результат выполнения будет в целом аналогичен предыдущему выводу, но, так как файл с переменными указывался через vars_files, загрузка переменных не будет видна как отдельная задача:

```
$ ansible-playbook 8_playbook_include_vars_files.yml
```

```
SSH password:

PLAY [Run cfg commands on router] ******************************************

TASK [Config BGP] **********************************************************
changed: [192.168.100.1]

TASK [Show BGP config] *****************************************************
ok: [192.168.100.1]

TASK [Debug registered var] ************************************************
ok: [192.168.100.1] => {
    "bgp_cfg.stdout_lines": [
        [
            "router bgp 65000",
            " bgp log-neighbor-changes",
            " network 120.0.0.0 mask 255.255.252.0",
            " neighbor 10.0.0.1 remote-as 65000",
            " neighbor 10.0.0.1 update-source Loopback0",
            " neighbor 10.0.0.2 remote-as 65000",
            " neighbor 10.0.0.2 update-source Loopback0",
            " neighbor 10.0.0.3 remote-as 65000",
            " neighbor 10.0.0.3 update-source Loopback0",
            " neighbor 10.0.0.4 remote-as 65000",
            " neighbor 10.0.0.4 update-source Loopback0",
            " neighbor 15.0.0.5 remote-as 500",
            " neighbor 15.0.0.5 ttl-security hops 3",
            " neighbor 15.0.0.5 send-community",
            " neighbor 26.0.0.6 remote-as 600",
            " neighbor 26.0.0.6 ttl-security hops 3",
            " neighbor 26.0.0.6 send-community"
        ]
    ]
}

PLAY RECAP *****************************************************************
192.168.100.1              : ok=3    changed=1    unreachable=0    failed=0
```

Роли

В прошлом разделе мы разобрались с использованием include. Это был первый способ разбития playbook на части. В этом разделе мы рассмотрим второй способ - роли.

Роли это способ логического разбития файлов Ansible. По сути роли это просто автоматизация выражений include, которая основана на определенной файловой структуре. То есть, нам не нужно будет явно указывать полные пути к файлам с задачами или сценариями, а достаточно лишь соблюдать определенную структуру файлов.

Но, засчет этого, работать с Ansible намного удобней. И у нас рождается модульная структура, которая разбита на роли, например, на основе функциональности.

Для того, чтобы мы могли использовать роли, нужно соблюдать определенную структуру каталогов:

```
├── all_roles.yml
├── cfg_security.yml
├── cfg_ospf.yml
|
└── roles
    ├── ospf
    |   ├── files
    |   ├── templates
    |   ├── tasks
    |   ├── handlers
    |   ├── vars
    |   ├── defaults
    |   └── meta
    └── security
        ├── files
        ├── templates
        ├── tasks
        ├── handlers
        ├── vars
        ├── defaults
        └── meta
```

Первые три файла - это playbook. Они используют созданные роли.

Например, playbook all_roles.yml выглядит так:

```
---

- name: Roles config
  hosts: cisco-routers
  gather_facts: false
  connection: local
  roles:
    - security
    - ospf
```

Остальные файлы: инвентарный, конфигурационный файл Ansible и каталоги с переменными, находятся в тех же местах (в том же каталоге, что и playbook).

Все роли, по умолчанию, должны быть определены в каталоге roles:

- Каталоги следующего уровня определяют названия ролей
- В примере выше, созданы две роли: ospf и security

- Внутри каждой роли могут быть указанные каталоги.
- Как минимум, понадобится каталог tasks, чтобы описать задачи, а все остальные каталоги опциональны.
- Внутри каталогов tasks, handlers, vars, defaults, meta автоматически считывается всё, что находится в файле main.yml
- если в этих каталогах есть другие файлы, их надо добавлять через include
- Внутри роли, на файлы в каталогах files, templates, tasks можно ссылаться не указывая путь к ним (достаточно указать имя файла)

Каталоги внутри роли:

- tasks - если в этом каталоге существует файл main.yml, все задачи, которые в нем указаны, будут добавлены в сценарий
- если в каталоге tasks есть файл с задачами с другим названием, его можно добавить в роль через include, при этом не нужно указывать путь к файлу
- handlers - если в этом каталоге существует файл main.yml, все handlers, которые в нем указаны, будут добавлены в сценарий
- vars - если в этом каталоге существует файл main.yml, все переменные, которые в нем указаны, будут добавлены в сценарий
- defaults - каталог, в котором указываются значения по умолчанию для переменных. Эти значения имеют самый низкий приоритет, поэтому их легко перебить, определив переменную в другом месте. Если в этом каталоге существует файл main.yml, все переменные, которые в нем указаны, будут добавлены в сценарий
- meta - каталог, в котором указаны зависимости роли. Если в этом каталоге существует файл main.yml, все роли, которые в нем указаны, будут добавлены в список ролей
- files - каталог, в котором могут находиться различные файлы. Например, файл конфигурации
- templates - каталог для шаблонов. Если нужно указать шаблон из этого каталога, достаточно указать имя, без пути к файлу

Пример использования ролей

Рассмотрим пример использования ролей.

Структура каталога 8_playbook_roles выглядит таким образом:

```
├── ansible.cfg
├── myhosts
│
```

(continues on next page)

(продолжение с предыдущей страницы)

```
├── all_roles.yml
├── cfg_initial.yml
├── cfg_ospf.yml
│
├── group_vars
│   ├── all.yml
│   ├── cisco-routers.yml
│   └── cisco-switches.yml
├── host_vars
│   ├── 192.168.100.1
│   ├── 192.168.100.100
│   ├── 192.168.100.2
│   └── 192.168.100.3
│
└── roles
    ├── ospf
    │   ├── handlers
    │   │   └── main.yml
    │   ├── tasks
    │   │   └── main.yml
    │   └── templates
    │       └── ospf.j2
    ├── security
    │   └── tasks
    │       └── main.yml
    └── usability
        └── tasks
            └── main.yml
```

Файл конфигурации Ansible, инвентарный файл и каталоги с переменными остались без изменений.

Добавлен каталог roles, в котором находятся три роли: usability, security и ospf.

Для ролей usability и security создан только каталог tasks и в нем находится только один файл: main.yml.

Содержимое файла roles/usability/tasks/main.yml:

```
---

- name: Global usability config
  ios_config:
    lines:
      - no ip domain lookup
    provider: "{{ cli }}"
```

(continues on next page)

(продолжение с предыдущей страницы)

```
- name: Configure vty usability features
  ios_config:
    parents:
      - line vty 0 4
    lines:
      - exec-timeout 30 0
      - logging synchronous
      - history size 100
    provider: "{{ cli }}"
```

В нем находятся две задачи. Они достаточно простые и должны быть полностью понятны.

Обратите внимание, что в файле определяются только задачи. К каким хостам они будут применяться, будет определять playbook, который будет использовать роль.

Содержимое файла roles/security/tasks/main.yml также должно быть понятно:

```
---

- name: Global security config
  ios_config:
    lines:
      - service password-encryption
      - no ip http server
      - no ip http secure-server
    provider: "{{ cli }}"

- name: Configure vty security features
  ios_config:
    parents:
      - line vty 0 4
    lines:
      - transport input ssh
    provider: "{{ cli }}"
```

Примечание: Несмотря на то, что функционал достаточно простой и общий, мы разделили его на две роли. Такое разделение позволяет более четко описать цель роли.

Теперь посмотрим как будет выглядеть playbook, который использует обе роли (файл cfg_initial.yml):

```
---

- name: Initial config
  hosts: cisco-routers
```

(continues on next page)

(продолжение с предыдущей страницы)

```
  gather_facts: false
  connection: local
  roles:
    - usability
    - security
```

Теперь запустим playbook (предварительно на маршрутизаторах сделаны изменения):

```
$ ansible-playbook cfg_initial.yml
```

```
SSH password:

PLAY [Initial config] ************************************************************

TASK [usability : Global usability config] ***************************************
ok: [192.168.100.2]
ok: [192.168.100.3]
ok: [192.168.100.1]

TASK [usability : Configure vty usability features] ******************************
changed: [192.168.100.1]
changed: [192.168.100.2]
changed: [192.168.100.3]

TASK [security : Global security config] *****************************************
changed: [192.168.100.1]
changed: [192.168.100.2]
changed: [192.168.100.3]

TASK [security : Configure vty security features] ********************************
changed: [192.168.100.1]
changed: [192.168.100.3]
changed: [192.168.100.2]

PLAY RECAP ***********************************************************************
192.168.100.1              : ok=4    changed=3    unreachable=0    failed=0
192.168.100.2              : ok=4    changed=3    unreachable=0    failed=0
192.168.100.3              : ok=4    changed=3    unreachable=0    failed=0
```

Обратите внимание, что теперь, когда задачи выполняются, перед именем задачи написано имя роли:

```
TASK [usability : Configure vty usability features]
```

Теперь разберемся с ролью ospf. В этой роли используется несколько файлов.

Файл roles/ospf/tasks/main.yml описывает задачи:

```
---

- name: Collect facts
  ios_facts:
    gather_subset:
      - "!hardware"
    provider: "{{ cli }}"

- name: Set fact ospf_networks
  set_fact:
    current_ospf_networks: "{{ ansible_net_config | regex_findall('network (.*) area 0') }
↪}"

- name: Show var current_ospf_networks
  debug: var=current_ospf_networks

- name: Config OSPF
  ios_config:
    src: ospf.j2
    provider: "{{ cli }}"
  notify: save config

- name: Write OSPF cfg in variable
  ios_command:
    commands:
      - sh run | s ^router ospf
    provider: "{{ cli }}"
  register: ospf_cfg

- name: Show OSPF cfg
  debug: var=ospf_cfg.stdout_lines
```

Разберемся с содержимым файла:

- Сначала мы собираем все факты об устройствах, кроме hardware.
- Затем вручную устанавливаем факт current_ospf_networks
- фильтруем конфигурацию устройства и находим все строки с командами `network ... area 0`. Всё, что находится между указанными словами, запоминается.
- в итоге, мы получим список с командами
- Следующая задача показывает содержимое переменной current_ospf_networks
- Задача «Config OSPF» настраивает OSPF по шаблону ospf.j2
- если изменения были, выполняется handler save config
- Последующие задачи выполняют команду `sh run | s ^router ospf` и отображают

содержимое

Файл roles/ospf/handlers/main.yml:

```
- name: save config
  ios_command:
    commands:
      - write
    provider: "{{ cli }}"
```

Файл roles/ospf/templates/ospf.j2:

```
router ospf 1
 router-id {{ mgmnt_ip }}
 ispf
 auto-cost reference-bandwidth 10000
{% for ip in ansible_net_all_ipv4_addresses %}
 network {{ ip }} 0.0.0.0 area 0
{% endfor %}
{% for network in current_ospf_networks %}
 {% if network.split()[0] not in ansible_net_all_ipv4_addresses %}
   no network {{ network }} area 0
 {% endif %}
{% endfor %}
```

В шаблоне мы используем переменные:

- mgmnt_ip - определена в соответствующем файле каталога host_vars/
- ansible_net_all_ipv4_addresses - эта переменная содержит список всех IP-адресов устройства. Это факт, который обнаруживается благодаря модулю ios_facts
- current_ospf_networks - факт, который мы создали вручную

Получается, что в шаблоне настраиваются команды network, на основе IP-адресов устройства, а затем удаляются лишние команды network.

Проверим работу роли на примере такого playbook cfg_ospf.yml:

```
---

- name: Configure OSPF
  hosts: 192.168.100.1
  gather_facts: false
  connection: local
  roles:
    - ospf
```

Начальная конфигурация R1 такая (две лишних команды network):

```
R1#sh run | s ^router ospf
router ospf 1
 router-id 10.0.0.1
 ispf
 auto-cost reference-bandwidth 10000
 network 10.1.1.1 0.0.0.0 area 0
 network 10.10.1.1 0.0.0.0 area 0
 network 192.168.100.1 0.0.0.0 area 0
 network 192.168.200.1 0.0.0.0 area 0

R1#show ip int bri | exc unass
Interface        IP-Address      OK? Method Status      Protocol
Ethernet0/0      192.168.100.1   YES NVRAM  up          up
Ethernet0/1      192.168.200.1   YES NVRAM  up          up
```

Теперь запустим playbook и посмотрим удалятся ли две лишние команды:

```
$ ansible-playbook cfg_ospf.yml
```

```
SSH password:

PLAY [Configure OSPF] ****************************************************

TASK [ospf : Collect facts] **********************************************
ok: [192.168.100.1]

TASK [ospf : Set fact ospf_networks] *************************************
ok: [192.168.100.1]

TASK [ospf : Show var current_ospf_networks] *********************************
ok: [192.168.100.1] => {
    "current_ospf_networks": [
        "10.1.1.1 0.0.0.0",
        "10.10.1.1 0.0.0.0",
        "192.168.100.1 0.0.0.0",
        "192.168.200.1 0.0.0.0"
    ]
}

TASK [ospf : Config OSPF] ************************************************
changed: [192.168.100.1]

TASK [ospf : Write OSPF cfg in variable] *********************************
ok: [192.168.100.1]

TASK [ospf : Show OSPF cfg] **********************************************
ok: [192.168.100.1] => {
    "ospf_cfg.stdout_lines": [
        [
            "router ospf 1",
            " router-id 10.0.0.1",
            " ispf",
            " auto-cost reference-bandwidth 10000",
            " network 192.168.100.1 0.0.0.0 area 0",
            " network 192.168.200.1 0.0.0.0 area 0"
        ]
    ]
}

RUNNING HANDLER [ospf : save config] *************************************
ok: [192.168.100.1]

PLAY RECAP ***************************************************************
192.168.100.1              : ok=7    changed=1    unreachable=0    failed=0
```

Обратите внимание, что до выполнения конфигурации было 4 команды network (мы их видим по содержимому переменной current_ospf_networks):

```
"current_ospf_networks": [
    "10.1.1.1 0.0.0.0",
    "10.10.1.1 0.0.0.0",
    "192.168.100.1 0.0.0.0",
    "192.168.200.1 0.0.0.0"
]
```

А после конфигурации, осталось две команды network:

```
    "ospf_cfg.stdout_lines": [
        [
            "router ospf 1",
            " router-id 10.0.0.1",
            " ispf",
            " auto-cost reference-bandwidth 10000",
            " network 192.168.100.1 0.0.0.0 area 0",
            " network 192.168.200.1 0.0.0.0 area 0"
        ]
    ]

**Note** Этот пример не идеален. Например, подразумевается, что все
интерфейсы находятся в зоне 0. Но его достаточно, чтобы понять как
использовать роли.
```

Скорее всего, в реальной жизни вы уберете задачи, которые отображают содержимое переменных. Но, для того чтобы лучше разобраться с тем, что делает роль, они полезны.

На этом мы заканчиваем раздел. О других возможностях использования ролей вы можете почитать в документации, в разделе роли.

Фильтры Jinja2

Ansible позволяет использовать фильтры Jinja2 не только в шаблонах, но и в playbook.

С помощью фильтров можно преобразовывать значения переменных, переводить их в другой формат и др.

Ansible поддерживает не только встроенные фильтры Jinja, но и множество собственных фильтров. Мы не будем рассматривать все фильтры, поэтому, если вы не найдете нужный вам фильтр тут, посмотрите документацию.

Мы уже использовали фильтры:

- to_nice_json в разделе ios_facts
- regex_findall в разделе роли

Если вас интересуют фильтр в контексте использования их в шаблонах, это рассматривалось в разделе Фильтры.

Для начала, перечислим несколько фильтров для общего понимания возможностей.

Ansible поддерживает такие фильтры (список не полный):

- фильтры для форматирования данных:
  - `{{ var | to_nice_json }}` - преобразует данные в формат JSON
  - `{{ var | to_nice_yaml }}` - преобразует данные в формат YAML
- переменные
  - `{{ var | default(9) }}` - позволяет определить значение по умолчанию для переменной
  - `{{ var | default(omit) }}` - позволяет пропустить переменную, если она не определена
- списки
  - `{{ lista | min }}` - минимальный элемент списка
  - `{{ lista | max }}` - максимальный элемент списка
- фильтры, которые работают множествами
  - `{{ list1 | unique }}` - возвращает множество уникальных элементов из списка
  - `{{ list1 | difference(list2) }}` - разница между двумя списками: каких элементов первого списка нет во втором
- фильтр для работы с IP-адресами
  - `{{ var | ipaddr }}` - проверяет является ли переменная IP-адресом
- регулярные выражения
  - regex_replace - замена в строке
  - regex_search - ищет первое совпадение с регулярным выражением
  - regex_findall - ищет все совпадения с регулярным выражением
- фильтры, которые применяют другие фильтры к последовательности объектов:
  - map: `{{ list3 | map('int') }}` - применяет другой фильтр к последовательности элементов (например, список). Также позволяет брать значение определенного атрибута у каждого объекта в списке.
  - select: `{{ list4 | select('int') }}` - фильтрует последовательность применяя другой фильтр к каждому из элементов. Остаются только те объекты, для которых тест отработал.
- конвертация типов

- `{{ var | int }}` - конвертирует значение в число, по умолчанию, в десятичное
- `{{ var | list }}` - конвертирует значение в список

to_nice_yaml

Фильтры to_nice_yaml (to_nice_json) можно использовать для того, чтобы записать нужную информацию в файл.

Ansible также поддерживает фильтры to_json и to_yaml, но их сложнее воспринимать визуально.

Повторим пример из раздела ios_facts. Playbook 8_playbook_filters_to_nice_yaml.yml:

```
---

- name: Collect IOS facts
  hosts: cisco-routers
  gather_facts: false
  connection: local

  tasks:

    - name: Facts
      ios_facts:
        gather_subset: all
        provider: "{{ cli }}"
      register: ios_facts_result

    - name: Copy facts to files
      copy:
        content: "{{ ios_facts_result | to_nice_yaml }}"
        dest: "all_facts/{{inventory_hostname}}_facts.yml"
```

Результат выполнения playbook будет таким:

```
$ ansible-playbook 8_playbook_filters_to_nice_yaml.yml
```

```
SSH password:

PLAY [Collect IOS facts] ********************************************************

TASK [Facts] ********************************************************************
ok: [192.168.100.2]
ok: [192.168.100.1]
ok: [192.168.100.3]

TASK [Copy facts to files] ******************************************************
changed: [192.168.100.3]
changed: [192.168.100.2]
changed: [192.168.100.1]

PLAY RECAP **********************************************************************
192.168.100.1              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=2    changed=1    unreachable=0    failed=0
192.168.100.3              : ok=2    changed=1    unreachable=0    failed=0
```

Теперь в каталоге all_facts появились такие файлы:

```
192.168.100.1_facts.yml
192.168.100.2_facts.yml
192.168.100.3_facts.yml
```

Файл all_facts/192.168.100.1_facts.yml:

```
ansible_facts:
    ansible_net_all_ipv4_addresses:
    - 192.168.200.1
    - 192.168.100.1
    ansible_net_all_ipv6_addresses: []
    ansible_net_config: "Building configuration...\n\nCurrent configuration : 7367\
        \ bytes\n!\n! Last configuration change at 16:33:06 UTC Mon Jan 9 2017\nversion\
        \ 15.2\nno service timestamps debug uptime\nno service timestamps log uptime\n\
        service password-encryption\n!\nhostname R1\n!\nboot-start-marker\n
...
```

regex_findall, map, max

Посмотрим пример использования фильтров одновременно и в шаблоне, и в playbook.

Сделаем playbook, который будет генерировать конфигурацию site-to-site VPN (GRE + IPsec) для двух сторон.

В этом случае, мы не будем отправлять команды на устройства, а воспользуемся модулем template, чтобы сгенерировать конфигурацию и записать её в локальные файлы.

Настройка GRE + IPsec выглядит таким образом:

```
crypto isakmp policy 10
 encr aes
 authentication pre-share
 group 5
 hash sha

crypto isakmp key cisco address 192.168.100.2

crypto ipsec transform-set AESSHA esp-aes esp-sha-hmac
 mode transport

crypto ipsec profile GRE
 set transform-set AESSHA

interface Tunnel0
 ip address 10.0.1.2 255.255.255.252
 tunnel source 192.168.100.1
 tunnel destination 192.168.100.2
 tunnel protection ipsec profile GRE
```

Playbook 8_playbook_filters_regex.yml

```
---

- name: Cfg VPN
  hosts: 192.168.100.1,192.168.100.2
  gather_facts: false
  connection: local


  vars:
    wan_ip_1: 192.168.100.1
    wan_ip_2: 192.168.100.2
    tun_ip_1: 10.0.1.1 255.255.255.252
    tun_ip_2: 10.0.1.2 255.255.255.252
```

(continues on next page)

(продолжение с предыдущей страницы)

```
tasks:

  - name: Collect facts
    ios_facts:
      gather_subset:
        - "!hardware"
      provider: "{{ cli }}"

  - name: Collect current tunnel numbers
    set_fact:
      tun_num: "{{ ansible_net_config | regex_findall('interface Tunnel(.*)') }}"

  #- debug: var=tun_num

  - name: Generate VPN R1
    template:
      src: templates/ios_vpn1.txt
      dest: configs/result1.txt
    when: wan_ip_1 in ansible_net_all_ipv4_addresses

  - name: Generate VPN R2
    template:
      src: templates/ios_vpn2.txt
      dest: configs/result2.txt
    when: wan_ip_2 in ansible_net_all_ipv4_addresses
```

Разберемся с содержимым playbook. В этом playbook один сценарий и он применяется только к двум устройствам:

```
- name: Cfg VPN
  hosts: 192.168.100.1,192.168.100.2
  gather_facts: false
  connection: local
```

Наша задача была в том, чтобы сделать playbook, который можно легко повторно использовать. А значит, нужно сделать так, чтобы нам не нужно было повторять несколько раз одни и те же вещи (например, адреса).

И, в данном случае не очень удобно будет, если мы будем создавать переменные в файлах host_vars. Удобней создать их в самом playbook, а когда нужно будет сгенерировать конфигурацию для другой пары устройств, достаточно будет сменить адреса в playbook.

Для этого, в сценарии создан блок с переменными:

```
  vars:
```

(continues on next page)

(продолжение с предыдущей страницы)

```
    wan_ip_1: 192.168.100.1
    wan_ip_2: 192.168.100.2
    tun_ip_1: 10.0.1.1 255.255.255.252
    tun_ip_2: 10.0.1.2 255.255.255.252

Вместо адресов wan_ip_1, wan_ip_2, вам нужно будет подставить
белые адреса маршрутизаторов.
```

Адреса мы задаем вручную. Но, всё остальное, хотелось бы делать автоматически.

Например, для настройки VPN нам нужно знать номер туннеля, чтобы создать интерфейс. Но мы не можем взять какой-то произвольный номер, так как на маршрутизаторе уже может существовать туннель с таким номером. Нам нужно определять автоматически.

Для этого, мы сначала собираем факты об устройстве:

```
- name: Collect facts
  ios_facts:
    gather_subset:
      - "!hardware"
    provider: "{{ cli }}"
```

Теперь мы создадим факт, для каждого из маршрутизаторов, который будет содержать список текущих номеров туннелей. Создаем факт мы с помощью модуля set_fact.

Факт создается на основе того, что нам выдаст результат поиска в конфигурации строки `interface TunnelX` с помощью фильтра regex_findall. Этот фильтр ищет все строки, которые совпадают с регулярным выражением. А затем, запоминает и записывает в список то, что попало в круглые скобки (номер туннеля).

```
- name: Collect current tunnel numbers
  set_fact:
    tun_num: "{{ ansible_net_config | regex_findall('interface Tunnel(.*)') }}"
```

Дальнейшая обработка списка будет выполняться в шаблоне.

Затем, мы генерируем шаблоны для устройств. Для каждого устройства есть свой шаблон. Поэтому, в каждой задаче стоит условие

```
when: wan_ip_1 in ansible_net_all_ipv4_addresses
```

Благодаря этому условию, мы выбираем для какого устройства будет сгенерирован какой конфиг.

ansible_net_all_ipv4_addresses - это список IP-адресов на устройства, вида:

```
ansible_net_all_ipv4_addresses:
    - 192.168.200.1
    - 192.168.100.1
```

Этот список был получен в задаче по сбору фактов.

Задача будет выполняться только в том случае, если в списке адресов на устройстве, был найден адрес wan_ip_1.

Генерация шаблонов:

```
- name: Generate VPN R1
  template:
    src: templates/ios_vpn1.txt
    dest: configs/result1.txt
  when: wan_ip_1 in ansible_net_all_ipv4_addresses

- name: Generate VPN R2
  template:
    src: templates/ios_vpn2.txt
    dest: configs/result2.txt
  when: wan_ip_2 in ansible_net_all_ipv4_addresses
```

Шаблон templates/ios_vpn1.txt выглядит таким образом:

```
{% if not tun_num %}
 {% set tun_num = 0 %}
{% else %}
 {% set tun_num = tun_num | map('int') | max %}
 {% set tun_num = tun_num + 1 %}
{% endif %}

crypto isakmp policy 10
 encr aes
 authentication pre-share
 group 5
 hash sha

crypto isakmp key cisco address {{ wan_ip_2 }}

crypto ipsec transform-set AESSHA esp-aes esp-sha-hmac
 mode transport

crypto ipsec profile GRE
 set transform-set AESSHA

interface Tunnel {{ tun_num }}
```

(continues on next page)

(продолжение с предыдущей страницы)

```
 ip address {{ tun_ip_1 }}
 tunnel source {{ wan_ip_1 }}
 tunnel destination {{ wan_ip_2 }}
 tunnel protection ipsec profile GRE
```

Шаблон templates/ios_vpn2.txt выглядит точно также, меняются только переменные с адресами:

```
{% if not tun_num %}
 {% set tun_num = 0 %}
{% else %}
 {% set tun_num = tun_num | map('int') | max %}
 {% set tun_num = tun_num + 1 %}
{% endif %}

crypto isakmp policy 10
 encr aes
 authentication pre-share
 group 5
 hash sha

crypto isakmp key cisco address {{ wan_ip_1 }}

crypto ipsec transform-set AESSHA esp-aes esp-sha-hmac
 mode transport

crypto ipsec profile GRE
 set transform-set AESSHA

interface Tunnel {{ tun_num }}
 ip address {{ tun_ip_2 }}
 tunnel source {{ wan_ip_2 }}
 tunnel destination {{ wan_ip_1 }}
 tunnel protection ipsec profile GRE
```

В самой конфигурации никаких сложностей нет. Обычная подстановка переменных.

Разберемся с этой частью:

```
{% if not tun_num %}
 {% set tun_num = 0 %}
{% else %}
 {% set tun_num = tun_num | map('int') | max %}
 {% set tun_num = tun_num + 1 %}
{% endif %}
```

Переменная tun_num - это факт, который мы устанавливали в playbook. Если на маршрути-

заторе созданы туннели, эта переменная содержит список номеров туннелей. Но, если на маршрутизаторе нет ни одного туннеля, мы получим пустой список.

Если мы получили пустой список, то можно создавать интерфейс Tunnel0. Если мы получили список с номерами, то мы вычисляем максимальный и используем следующий номер, для нашего туннеля.

Если переменная tun_num будет пустым списком, нам нужно установить её равной 0 (пустой список - False):

```
{% if not tun_num %}
 {% set tun_num = 0 %}
```

Иначе, нам нужно сначала конвертировать строки в числа, затем выбрать из чисел максимальное и добавить 1. Это и будет значение переменной tun_num.

```
{% else %}
 {% set tun_num = tun_num | map('int') | max %}
 {% set tun_num = tun_num + 1 %}
{% endif %}
```

Выполнение playbook (создайте каталог configs):

```
$ ansible-playbook 8_playbook_filters_regex.yml
```

```
SSH password:

PLAY [Cfg VPN] *****************************************************************

TASK [Collect facts] ***********************************************************
ok: [192.168.100.2]
ok: [192.168.100.1]

TASK [Collect current tunnel numbers] ******************************************
ok: [192.168.100.1]
ok: [192.168.100.2]

TASK [debug] *******************************************************************
ok: [192.168.100.1] => {
    "tun_num": [
        "0",
        "1",
        "3",
        "9",
        "10",
        "11",
        "15"
    ]
}
ok: [192.168.100.2] => {
    "tun_num": []
}

TASK [Generate VPN R1] *********************************************************
skipping: [192.168.100.2]
changed: [192.168.100.1]

TASK [Generate VPN R2] *********************************************************
skipping: [192.168.100.1]
changed: [192.168.100.2]

PLAY RECAP *********************************************************************
192.168.100.1              : ok=4    changed=1    unreachable=0    failed=0
192.168.100.2              : ok=4    changed=1    unreachable=0    failed=0
```

На маршрутизаторе 192.168.100.1 специально созданы несколько туннелей. А на маршрутизаторе 192.168.100.2 нет ни одного туннеля.

В результате, мы получили такие конфигурации (configs/result1.txt):

```
crypto isakmp policy 10
 encr aes
 authentication pre-share
```

(continues on next page)

(продолжение с предыдущей страницы)

```
 group 5
 hash sha

crypto isakmp key cisco address 192.168.100.2

crypto ipsec transform-set AESSHA esp-aes esp-sha-hmac
 mode transport

crypto ipsec profile GRE
 set transform-set AESSHA

interface Tunnel 16
 ip address 10.0.1.1 255.255.255.252
 tunnel source 192.168.100.1
 tunnel destination 192.168.100.2
 tunnel protection ipsec profile GRE
```

Файл configs/result2.txt:

```
crypto isakmp policy 10
 encr aes
 authentication pre-share
 group 5
 hash sha

crypto isakmp key cisco address 192.168.100.1

crypto ipsec transform-set AESSHA esp-aes esp-sha-hmac
 mode transport

crypto ipsec profile GRE
 set transform-set AESSHA

interface Tunnel 0
 ip address 10.0.1.2 255.255.255.252
 tunnel source 192.168.100.2
 tunnel destination 192.168.100.1
 tunnel protection ipsec profile GRE
```

7

Скачать PDF/Epub

- Epub
- PDF