OTUS

ОНЛАЙН-ОБРАЗОВАНИЕ

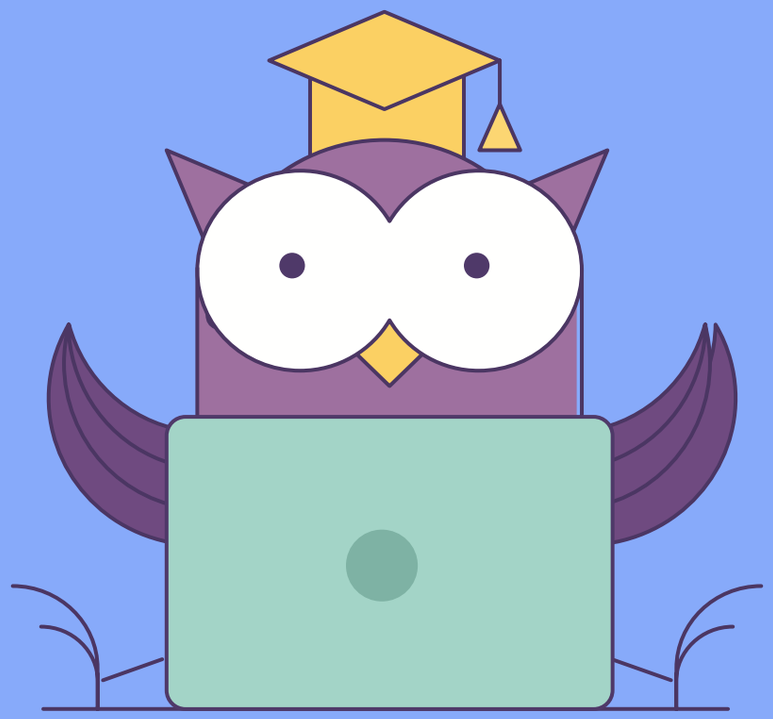
OTUS

Управление конфигурациями. Ansible Part II

Курс «Администратор Linux»

Альбрехт Леонид

Меня хорошо слышно && видно?



Напишите в чат, если есть проблемы!

Ставьте + если все хорошо
Ставьте - если есть проблемы

Маршрут вебинара

Плейбуки

Плюсы:

- Просто писать и читать
- Просто отлаживать

Минусы:

- Быстро разрастаются в размере
- Разрастаются в кол-ве
- Не версионируются
- Не подходят для распространения и переиспользования

Ansible. Роли

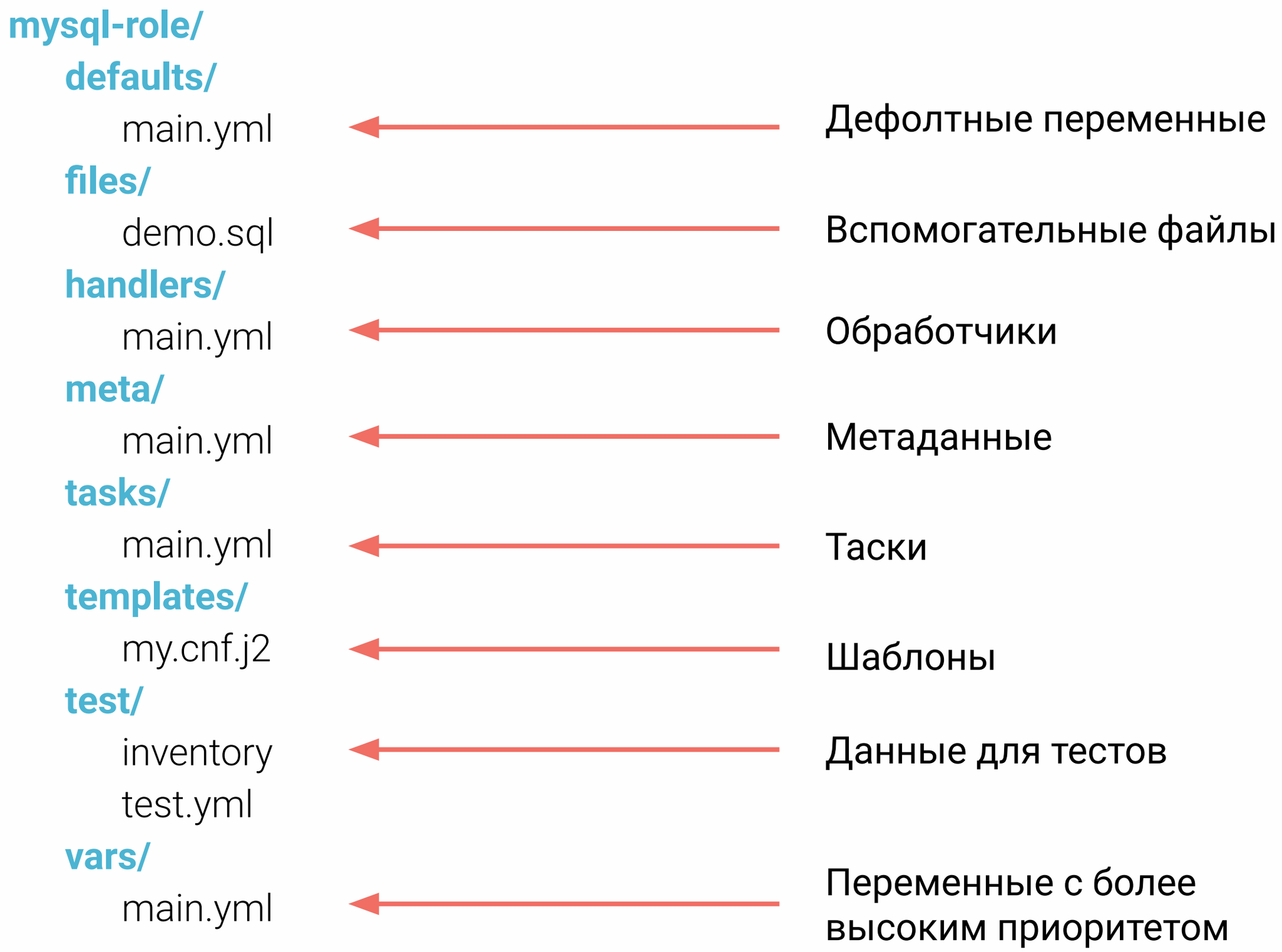
По сути это директория с определенной структурой и файлами внутри нее

- Роль состоит из:
  - Таксов и хендлеров
  - Переменных
  - Метаданных
  - Тестов
  - Вспомогательных файлов
  - Шаблонов

Инициализировать древо каталогов можно одной командой:

[root@ansible ~]$ ansible-galaxy init mysql-role

Ansible. Роли



Что делать плейбуку?

В плейбуке остается:

- Обозначение хостов
- Указание на роль(и)
- Указание переменных

```
# playbooks/role-nginx.yml
- name: Install some packages
  hosts: web
  vars:
    some_var: some_thing

  roles:
  -   nginx
```

Версионирование

- Роли гибко версионируются
- Это позволяет избежать ситуации когда применена неправильная версия роли (человеческий фактор не исключен)
- Отлично встраивается в работу команды над проектом

Ansible. Requirements.yml

- Зависимости необходимые вашей роли или всему репозиторию

```
- src: zaiste.nginx

- src: https://github.com/geerlingguy/ansible-role-nginx
  version: 1.2.1
```

[root@ansible ~]$ ansible-galaxy install -r requirements.yml

Циклы

Циклы:

- Для сокращения кол-во задач есть смысл использовать циклы
- Задача в выполняется в цикле пока условие действительно или другими словами пока переменная не пуста
- Вызывается ключевым словом **loop**

Примеры циклов

```
# tasks/main.yml
- name: Install some packages
  yum:
    name: {{ item }}
    state: present
  loop:
    - htop
    - bind-utils
    - mtr
```

```
# vars/main.yml
packages:
  - htop
  - bind-utils
  - mtr
```

```
# tasks/main.yml
name: Install some packages
yum:
  name: {{ item }}
  state: present
loop: {{ packages }}
```

Словари

- Иногда удобней под одним именем объединить несколько переменных для переиспользования в задаче
- Позволяет драматически сократить ваши playbook-и

Пример словаря

```
dba_users:
  - name: 'petrov'
    dest: '/home/petrov/'
    pg_password: ‘gfhjkm’
  - name: 'vase4kin'
    dest: '/home/vase4kin/'
    pg_password: ‘gfhjkm’
  - name: 'sidorov'
    dest: '/home/sidorov/'
    pg_password: ‘gfhjkm’
```

```
- name: Configure profile settings for DBA users
  template:
    src: bash_profile.j2
    dest: "{{ item.dest }}/.bash_profile"
    owner: "{{ item.name }}"
    group: "{{ item.name }}"
    mode: 0600
  loop: "{{ dba_users }}"
  tags: bash_profile

- name: Configure psqlrc file for DBA users
  template:
    src: psqlrc.j2
    dest: "{{ item.dest }}/.psqlrc"
    owner: "{{ item.name }}"
    group: "{{ item.name }}"
    mode: 0600
  loop: "{{ dba_users }}"
  tags: psqlrc
```

Ansible. Переиспользование. Include

- С версии 2.4 добавлены директивы include_* и import_* для задач и плейбуков. До этого была доступна только include.
- Сложные сценарии можно разбивать на несколько файлов с задачами и переиспользовать их

import vs. include

Все import* блоки пре-обрабатываются во время парсинга playbook

Все include* блоки обрабатываются когда встречаются уже во время выполнения playbook

Ansible. Пример Include

```
---
- name: Setup and use a role
  hosts: localhost
  tasks:
  - name: Get geerlingguy.apache
    command: ansible-galaxy install geerlingguy.apache

  - name: Include geerlingguy.apache
    include_role:
      name: geerlingguy.apache
```

Ansible. Условие с include

- Можно настроить ветвление с помощью include и when:

```
---
- name: Include OS-specific variables.
  include_vars: "{{ ansible_os_family }}.yml"

- include_tasks: setup-RedHat.yml
  when: ansible_os_family == 'RedHat'

- include_tasks: setup-Ubuntu.yml
  when: ansible_os_family == 'Ubuntu'

- import_tasks: vhosts.yml
```

Ansible. Handlers

- **Handlers (обработчики)** - 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 выполнится 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.
- Принудительное выполнение хендлера возможно при помощи модуля meta: flush_handlers

Ansible. Пример Handler

```
handlers:
- name: reload nginx
  service:
    name: nginx
    state: reloaded

tasks:
- name: Create nginx.conf file for NGINX
      template:
        src: templates/nginx/nginx.conf
        dest: /etc/nginx/nginx.conf
   notify:
    - reload nginx
```

Ansible. Vault

- Иногда нужно хранить аутентификационные данные в зашифрованном файле, а не в plain-text, для этого можно (и нужно) воспользоваться ansible vault
- В процессе выполнения Ansible получает к ним доступ. Файлы на диске при этом остаются зашифрованными

Ansible. Vault

- Для запуска play с использованием зашифрованного файла нужно передать параметр --ask-vault-pass

```
[root@ansible ~]$ ansible-playbook nginx.yml --ask-vault-pass
```

- Либо можно указать в ansible.cfg опцию vault_password_file, которая указывает на файл с ключом

Ansible. Vault

- Для большего удобства можно создать зашифрованную строку:

  [root@ansible ~]$ ansible-vault encrypt_string --vault-id dba@passfile ‘gfhjkm’ --name pg_pass:

```
pg_pass: !vault |
        $ANSIBLE_VAULT;1.1;AES25;dba
        65323662363236363135353735613937386561626137373434303930663133316461643063666161
        32626135363765363436353164613038333933383732326 40a3631613163383232653330373033 32
        39643266656262666436666162363761666637663730396664626361363037346539386531396661
        64323234643933630a353435313438633161313262313336386538396564633861623231396364
        3265
```

- Ваши скрипты и конфигурации - это ваш код:
  - Пользуйтесь системами контроля версий
  - Реализуйте от простого к сложному. Начните с playbooks и статических inventory файлов.
  - Проводите рефакторинг

- Следите за стилем кода:
  - Тэги
  - Пробелы
  - Имена задач, переменных, ролей
  - Содержимое директорий
  - Документация

- Используйте особенности YAML синтаксиса для увеличения читабельности:
  - Чтение по вертикали
  - Поддержка сложных параметров
  - Поддержка синтаксиса многими редакторами
- Избегайте усложнений. Много простых плейбуков лучше чем 1 сложный
- Избегайте использования command и shell. Пользуйтесь модулями.
- Разделяйте задачи конфигурации и провижининга:
  - cat site.yml

    - import_playbook: provision.yml

    - import_playbook: configure.yml

Debug

- Уровень verbosity -vvvv
- Модуль Debug
- Пошаговое выполнение роли

Модуль Debug

```
vars:
  my_var: ‘My Name’

tasks:
`  - name: Print my_var
    debug:
      var: my_var

`  - name: Print message
    debug:
      msg: “My_var is {{ my_var }}”
```

Модуль Debug

- Пошаговое выполнение позволяет управлять этапами выполнения и не навредить инфраструктуре
- Для запуска просто добавить ключ --step

```
~/P/O/s/ansible_i  master ±  ansible-playbook -i inventories/production/ playbooks/nginx/nginx.yml --step

PLAY [Install NGINX web server] ************************************************************************
Perform task: TASK: Gathering Facts (N)o/(y)es/(c)ontinue: y

Perform task: TASK: Gathering Facts (N)o/(y)es/(c)ontinue: ************************************************

TASK [Gathering Facts] *********************************************************************************
ok: [nginx01]
Perform task: TASK: Install {{ nginx_repo_name }} repository (N)o/(y)es/(c)ontinue: y

Perform task: TASK: Install {{ nginx_repo_name }} repository (N)o/(y)es/(c)ontinue: ************************

TASK [Install epel-release repository] *****************************************************************
changed: [nginx01]
```

Ваши вопросы?

Заполните, пожалуйста, опрос в ЛК о занятии

Спасибо
за внимание!
До встречи в Slack и на вебинаре