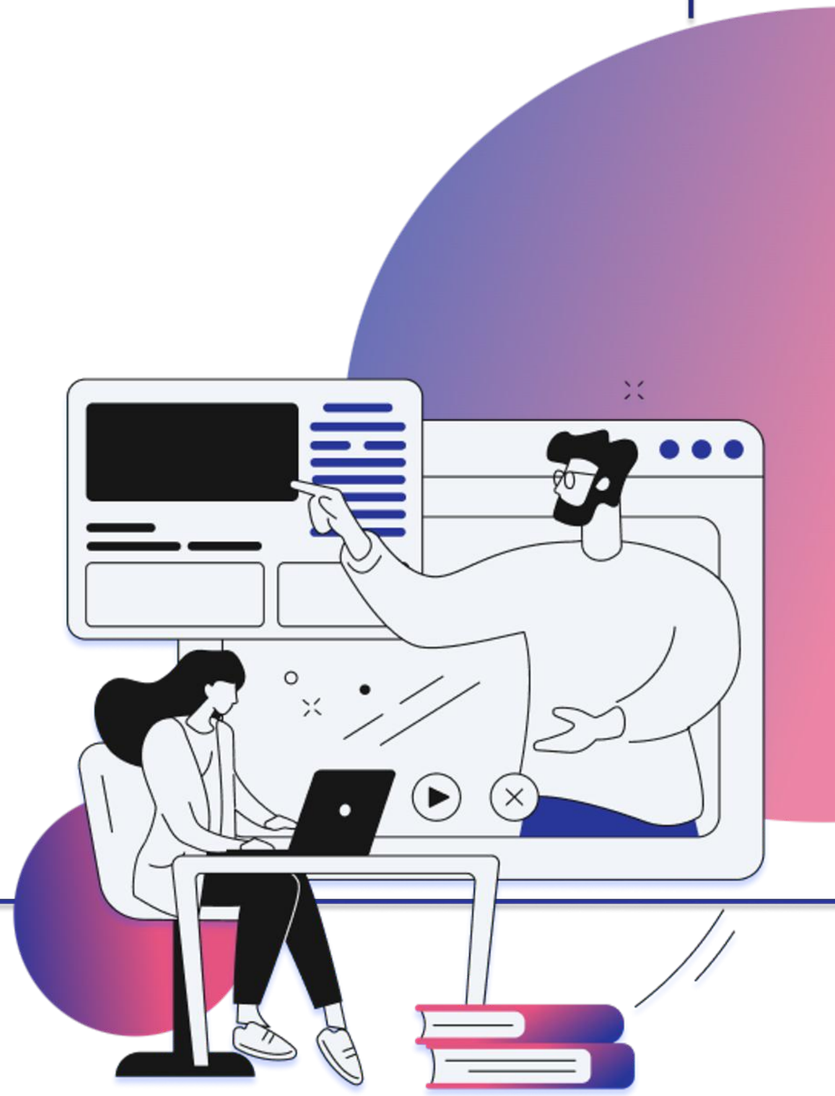
Учебный центр АйТи Клауд

# Практическое использование Ansible

**Учебный курс: практикум на стенде из четырёх машин**

современные технологии простым языком



# О курсе

- Один курс — два трека: «3 дня» (24 ак. ч., практики 1–10) и «5 дней» (40 ак. ч., практики 1–16).
- Первые десять практик трека «3 дня» — это начало полного курса, а не отдельный набор.
- Цель: приводить парк машин Linux к описанному состоянию средствами Ansible — воспроизводимо, проверяемо, без ручных правок на каждой машине.

▸ **Ключевая мысль: плейбук описывает состояние, а не действие. Повторный прогон не должен менять ничего.**

- Каждая практика опирается только на предыдущие; разные семейства ОС (Debian и RedHat) вводятся постепенно.
- Итоговая работа трека «3 дня» — практика 10, роль web_admin на четырёх машинах.
- Итоговая работа трека «5 дней» — практика 16: роль проходит ansible-lint, проверяется molecule и запускается по динамическому инвентарю.

# Стенд курса

- control — 10.0.2.5 — Ubuntu 26.04 LTS, 2 ядра, 4 ГБ, диск 25 ГБ — управляющая машина.
- web1 — 10.0.2.11 — Ubuntu 26.04 LTS, 2 ядра, 2 ГБ, диск 20 ГБ + второй диск 10 ГБ.
- web2 — 10.0.2.12 — Ubuntu 26.04 LTS, 2 ядра, 2 ГБ, диск 20 ГБ + второй диск 10 ГБ.
- db1 — 10.0.2.13 — Rocky Linux 10, 2 ядра, 2 ГБ, диск 20 ГБ + второй диск 10 ГБ.

▸ **Учётная запись engineer с sudo без пароля; учебный пароль Console47 опубликован, потому что стенд изолирован.**

- Вход по SSH-ключу ed25519; после практики 2 вход по паролю запрещён.
- Второй диск (/dev/sdb, 10 ГБ) нужен только в практике 13 (LVM), но добавляется сразу.
- Сеть упрощена намеренно: без DHCP и DNS, адреса в инвентаре, имена — через /etc/hosts.

# Версии и инструменты

- Ansible: ansible-core 2.19 (пакет Ansible community 12.x) в виртуальном окружении Python.
- Python: 3.13 на управляющей машине и Ubuntu-узлах, 3.12 на Rocky Linux 10.
- ОС: Ubuntu 26.04 LTS и Rocky Linux 10.
- Коллекции: ansible.posix, community.general; роль из Galaxy — geerlingguy.nginx; системные роли RHEL — redhat.rhel_system_roles.
- ansible-lint 25.x, molecule 25.x (драйвер delegated).
- ▸ **Установка Ansible — в venv: pip install в системный Python в 26.04 запрещён (PEP 668).**
- FQCN: в плейбуках модули указываются полным именем (ansible.builtin.apt, community.general.lvg).
- Стиль YAML: true/false, mode строкой в кавычках, loop вместо with_*, у каждой задачи name, отступ два пробела.

# Программа. Трек «3 дня»

- 1. Основы Ansible и стенд — 2 ак. ч.
- 2. Инвентаризация и подключение клиентов — 2
- 3. Язык YAML и ad-hoc управление — 2
- 4. Плейбуки: первый рабочий сценарий — 2
- 5. Переменные и секреты — 3
- 6. Факты и шаблоны Jinja2 — 3
- 7. Условия, циклы, блоки, обработчики — 2
- 8. Файлы, права и текстовые конфигурации — 2
- 9. Административные задачи — 2
- 10. Роли и итоговая работа — 4

▸ **Итого 24 ак. ч.: 5,5 теории и 18,5 практики. Дни: 1 — модули 1–4, 2 — 5–7, 3 — 8–10.**

# Программа. Трек «5 дней» (продолжение)

- 11. Регулярные выражения и точечная правка — 2 ак. ч.
- 12. Jinja2 для настоящих конфигураций — 3
- 13. Хранилище: диски, LVM, файловые системы — 3
- 14. Управление автоматизацией в сложных проектах — 3
- 15. Galaxy, коллекции и системные роли RHEL — 3
- 16. Качество автоматизации и инвентаризация — 2

▸ **Итого за дни 4–5: 16 ак. ч. Всего по курсу 40 ак. ч.: 8,5 теории и 31,5 практики.**

- Дни: 4 — модули 11–13, 5 — модули 14–16.

# Содержание


• Практика 1. Основы Ansible и стенд
• Практика 2. Инвентаризация и подключение клиентов
• Практика 3. Язык YAML и ad-hoc управление
• Практика 4. Плейбуки: первый рабочий сценарий
• Практика 5. Переменные и секреты
• Практика 6. Факты и шаблоны Jinja2
• Практика 7. Условия, циклы, блоки, обработчики
• Практика 8. Файлы, права и текстовые конфигурации
• Практика 9. Административные задачи
• Практика 10. Роли и итоговая работа
• Практика 11. Регулярные выражения и точечная правка
• Практика 12. Jinja2 для настоящих конфигов
• Практика 13. Хранилище: диски, LVM, файловые системы
• Практика 14. Управление автоматизацией в сложных проектах
• Практика 15. Galaxy, коллекции и системные роли RHEL
• Практика 16. Качество автоматизации и инвентаризация
▸ **Приложения: 1 — шпаргалка по командам; 2 — частые ошибки; 3 — Windows и сетевое оборудование.**

Учебный центр АйТи Клауд

# Практика 1. Основы Ansible и стенд

**ЭТА ПРАКТИКА — ПЕРВАЯ.**



современные технологии простым языком

# Обзор практики

- ЭТА ПРАКТИКА — ПЕРВАЯ.
- Здесь мы соберём стенд из четырёх машин, поставим Ansible на управляющую машину и убедимся, что он видит остальные три.
- Всё остальное в курсе выполняется на этом стенде, поэтому к конфигурации машин мы больше не возвращаемся.

## Что мы сделаем

- Разберём, что такое инфраструктура как код и почему для этого взят Ansible.
- Соберём четыре виртуальные машины: управляющую и три управляемых.
- Настроим вход по SSH-ключу без пароля.
- Поставим Ansible в отдельное виртуальное окружение Python.
- Запустим первую команду на трёх узлах сразу и разберём её вывод.

# Что понадобится

▸ **ЧЕТЫРЕ ВИРТУАЛЬНЫЕ МАШИНЫ: Управляющая — на ней идёт весь курс:**

`control 10.0.2.5 — Ubuntu 26.04 LTS, 2 ядра, 4 ГБ, диск 25 ГБ`

• Три управляемых узла:

`web1 10.0.2.11 — Ubuntu 26.04 LTS, 2 ядра, 2 ГБ, диск 20 ГБ`
`web2 10.0.2.12 — Ubuntu 26.04 LTS, 2 ядра, 2 ГБ, диск 20 ГБ`
`db1 10.0.2.13 — Rocky Linux 10, 2 ядра, 2 ГБ, диск 20 ГБ`

• ЧТО ЕЩЁ НУЖНО:

– машины в одной подсети и видят друг друга;
– доступ в интернет: нужно скачать пакеты и установить Ansible;
– учётная запись engineer с правами sudo на всех четырёх машинах.
– Учебный пароль — Console47. Он опубликован в курсе потому, что
– стенд изолирован; в рабочей среде так нельзя;
– на web1, web2 и db1 — второй диск 10 ГБ без разметки. Он
– понадобится только в практике 13, но добавляется сейчас:
– дописывать диск в машину посреди занятия дольше, чем выделить
– его сразу.

# ЧАСТЬ 1. БАЗОВЫЕ ШАГИ

-

# ШАГ 1. ИНФРАСТРУКТУРА КАК КОД: ЗАЧЕМ ЭТО ВООБЩЕ

- Обычный способ настроить пять серверов выглядит так: инженер заходит по SSH на каждый и делает одно и то же руками.
- Через месяц он делает то же самое для шестого сервера — и делает чуть иначе, потому что не помнит, что именно набирал в прошлый раз.
- Ещё через месяц никто не может сказать, чем пятый сервер отличается от третьего.
- ИНФРАСТРУКТУРА КАК КОД (Infrastructure as Code, IaC) — это подход, при котором состояние машин описано в файлах, которые лежат в системе контроля версий.
- Файлы можно прочитать, проверить, изменить и применить к любому количеству машин одинаково.
- У подхода три следствия, ради которых его и берут:
  - ОПИСАНО СОСТОЯНИЕ, А НЕ ДЕЙСТВИЕ. Мы пишем «на этой машине
  - должен быть установлен веб-сервер и запущен», а не «выполни
  - apt install». Разница видна на втором запуске: описание
  - состояния можно применить повторно, ничего не сломав.
  - ИЗМЕНЕНИЯ ВИДНЫ. Файл описания можно сравнить с предыдущей
  - версией и увидеть, что поменялось в инфраструктуре, — так же,
  - как сравнивают код.
  - ОДИН РЕЗУЛЬТАТ НА ВСЕХ МАШИНАХ. Не «примерно так же»,
  - а одинаково, потому что описание одно.
- ПОЧЕМУ ИМЕННО ANSIBLE.

# ШАГ 2. КАК УСТРОЕН ANSIBLE: УПРАВЛЯЮЩАЯ МАШИНА И УЗЛЫ

- В Ansible две роли машин, и их важно не путать:
  УПРАВЛЯЮЩАЯ МАШИНА (control node) — там установлен Ansible
  и оттуда выполняются команды. В курсе это control.
  УПРАВЛЯЕМЫЕ УЗЛЫ (managed nodes) — машины, которыми управляют.
  В курсе это web1, web2 и db1. На них Ansible не установлен.
- Как происходит выполнение задачи:
  - Ansible читает описание задачи и список машин.
  - Подключается к каждой машине по SSH — как это сделал бы инженер, только параллельно.
  - Копирует на машину небольшой модуль — программу на Python, которая выполняет одно конкретное действие: поставить пакет,
  - скопировать файл, запустить службу.
  - Запускает этот модуль, получает результат в формате JSON, удаляет модуль с машины.
  - Печатает результат и переходит к следующей задаче.
- **ИЗ ЭТОГО СЛЕДУЕТ ПРАКТИЧЕСКОЕ: на управляемом узле должен быть Python. На Ubuntu и Rocky Linux он есть по умолчанию, так что для нашего стенда ничего делать не нужно. Если Python стоит по нестандартному пути,...**

# ШАГ 3. СОБИРАЕМ СТЕНД

- Соберите четыре машины по таблице из раздела «Что понадобится».
- Дальше — общие настройки, одинаковые на всех четырёх.

▸ **УЧЁТНАЯ ЗАПИСЬ: Пользователь engineer уже создан установщиком ОС; введите пароль Console47. Убедитесь, что он в группе sudo (Ubuntu) или wheel (Rocky):**

```
id
uid=1000(engineer) gid=1000(engineer) groups=1000(engineer),27(sudo)
```

▸ **ИМЕНА МАШИН: Задайте то же имя, что в таблице: control, web1, web2, db1. Именем удобно пользоваться вместо адреса, и дальше в курсе встречаются оба варианта.**

```
sudo hostnamectl set-hostname web1
```

- ФАЙЛ /etc/hosts НА ВСЕХ ЧЕТЫРЁХ МАШИНАХ.
- Пропишите остальные машины вручную — в курсе нет DNS, и это сделано намеренно: курс про Ansible, а не про службу имён.

```
sudo tee -a /etc/hosts > /dev/null <<'EOF'
10.0.2.5 control
10.0.2.11 web1
10.0.2.12 web2
10.0.2.13 db1
```

- Проверьте:
- ПРАВА SUDO БЕЗ ПАРОЛЯ.

# ШАГ 4. SSH-КЛЮЧ: ВХОД БЕЗ ПАРОЛЯ

- Ansible подключается по SSH.
- Пароль в командной строке — плохая идея, поэтому настроим ключ.
- На управляющей машине control:

```
ssh-keygen -t ed25519 -C "engineer@control"
```

- На все вопросы нажмите Enter: путь по умолчанию (/home/engineer/.ssh/id_ed25519) и пустая фраза-пароль.
- Пустая — потому что иначе ключ придётся разблокировать при каждом запуске, и автоматизация перестанет быть автоматизацией.

```
ls -l ~/.ssh/
-rw------- 1 engineer engineer 411 ... id_ed25519
-rw-r--r-- 1 engineer engineer 100 ... id_ed25519.pub
```

- Права важны: если у файла ключа есть доступ для группы или остальных, SSH откажется им пользоваться.
- Поэтому здесь -rw-------.
- Скопируйте открытый ключ на три узла.
- Команда спросит пароль Console47 на каждом:

```
ssh-copy-id engineer@10.0.2.11
ssh-copy-id engineer@10.0.2.12
ssh-copy-id engineer@10.0.2.13
```

# ШАГ 5. УСТАНОВКА ANSIBLE

- Ansible ставим на управляющей машине control.
- Узлы не трогаем: на них Ansible не нужен.
- Почему не пакетом из репозитория ОС: в репозитории Ubuntu версия Ansible отстаёт от текущей на год-два, а нам нужны свежие коллекции.
- Для этого в Ubuntu 24.04 и новее есть вторая причина: установка пакетов в системный Python запрещена (правило PEP 668), и pip отвечает «error: externally-managed-environment».
- Создаём виртуальное окружение Python:

```
sudo apt update
sudo apt install -y python3-venv
python3 -m venv ~/ansible-venv
source ~/ansible-venv/bin/activate
```

- После активации приглашение изменится: в начале строки появится (ansible-venv).
- Ставим Ansible:

```
pip install --upgrade pip
pip install ansible
```

- ПРОВЕРКА:

```
ansible --version
ansible [core 2.19.0]
```

# ШАГ 6. ПЕРВАЯ КОМАНДА: ANSIBLE ALL

- Пока у нас нет файла с описанием машин, перечислим их прямо в командной строке.
- Список адресов через запятую — это тоже инвентарь, только временный:

```
ansible all -i '10.0.2.11,10.0.2.12,10.0.2.13' \
-m ansible.builtin.ping -u engineer
```

- Что здесь написано:

```
all – образец (шаблон) машин. Группа all
существует всегда и содержит все
машины из инвентаря;
-i '10.0.2.11,...' – инвентарь: список машин;
-m ansible.builtin.ping – модуль, который надо выполнить;
-u engineer – пользователь, под которым подключаться.
```

- Вывод:
- Всё это заняло доли секунды: Ansible подключился к трём машинам параллельно, а не по очереди.
- ЧТО ОЗНАЧАЮТ ПОЛЯ:
  - SUCCESS — задача выполнена, ошибки нет;
  - changed: false — на машине ничего не изменено. У модуля ping

# ШАГ 7. ЗАЧЕМ СПРАШИВАТЬ РАЗРЕШЕНИЕ: SUDO ЧЕРЕЗ BECOME

- Ключ -b (сокращение от --become) говорит Ansible выполнять задачи с повышением прав:

```
ansible all -i '10.0.2.11,10.0.2.12,10.0.2.13' -u engineer -b \
-m ansible.builtin.command -a "id -u"
10.0.2.11 | CHANGED | rc=0 >>
0
```

- Ноль означает: команда выполнена от root.
- Так работает become: Ansible вызывает sudo, а sudo на нашем стенде настроен без пароля (шаг 3).

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: почти все задачи настройки требуют root: поставить пакет, записать файл в /etc, запустить службу. Поэтому в плейбуках строка become: true встречается почти в каждом play, и в практике 4 мы...**

▸ **ЧЕГО НЕ ДЕЛАТЬ: не запускайте Ansible от root и не подключайтесь к узлам пользователем root. Работа от root на управляющей машине означает, что любая ошибка в плейбуке сразу применяется с полными правами, а в...**

# ШАГ 8. ЧТО ЗНАЧИТ «ПАРК МАШИН»: РУЧНОЙ ЦИКЛ ДЛЯ СРАВНЕНИЯ

- Чтобы стало понятно, что именно мы автоматизируем, сделаем то же самое без Ansible — циклом по адресам:

```
for h in 10.0.2.11 10.0.2.12 10.0.2.13; do
echo "=== $h ==="
ssh -o BatchMode=yes engineer@$h \
'hostname; grep PRETTY_NAME /etc/os-release'
done
=== 10.0.2.11 ===
web1
PRETTY_NAME="Ubuntu 26.04 LTS"
```

- Работает.
- А теперь посмотрите, чего в этом цикле нет:
  - если одна машина недоступна, цикл продолжит работу, но вы
  - узнаете об этом из перемешанного вывода, а не из отчёта;
  - нет итога: сколько машин обработано, сколько изменилось,
  - сколько не ответило;
  - нет повторного запуска: если команда что-то поставила,

# ШАГ 9. СПРАВКА БЕЗ ИНТЕРНЕТА: ANSIBLE-DOC

- Модулей в Ansible сотни, и запоминать их параметры не нужно.
- Справка встроена:

```
ansible-doc -l | wc -l
172
```

- Число у вас будет другим: оно зависит от установленных коллекций.
- Список длинный, поэтому фильтруем:

```
ansible-doc -l | grep -i '^ansible.builtin.copy'
```

- Полная справка по модулю:

```
ansible-doc ansible.builtin.ping
```

- Краткая — только параметры, без примеров:

```
ansible-doc -s ansible.builtin.ping
- name: Try to connect to host, verify a usable python and return `pong' on success
ping:
data: # (optional) Data to return for the `ping' return value.
```

▸ **ПОЧЕМУ ИМЕННО ТАК: справка ansible-doc всегда соответствует той версии, которая установлена у вас. Документация на сайте — не всегда: она показывает текущий выпуск, а на стенде может стоять предыдущий. Когда...**

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: на изолированных стендах, куда нет доступа в интернет, ansible-doc — единственный источник. К нему быстро привыкаешь как к основному.**

# ШАГ 10. СНИМОК СТЕНДА

- Сделайте снимок всех четырёх машин средствами вашей системы виртуализации (VirtualBox, Proxmox, KVM).
- Это не команда, которую я могу здесь написать: она зависит от того, где развёрнут стенд.

▸ **ЗАЧЕМ ЭТО В РАБОТЕ И НА ЗАНЯТИИ: дальше мы будем ломать конфигурацию намеренно. В практике 8 испортим sshd_config, в практике 13 разметим диск, в практике 9 перезагрузим узел. Вернуться к исправному состоянию...**

▸ **ВАЖНО: снимок делается сейчас, пока стенд в известном состоянии. Позже, когда конфигурация уже изменена, снимок зафиксирует изменённое состояние, и вернуться будет некуда.**

# ЧАСТЬ 2. ЧТО ОТЛИЧАЕТ РАБОТУ ОТ УЧЕБНОГО ПРИМЕРА

-

# ШАГ 11. ОДНА МАШИНА, НЕСКОЛЬКО СПОСОБОВ ЗАДАТЬ ИНВЕНТАРЬ

• Инвентарь можно передать тремя способами, и все три встречаются в работе.

▸ **СПОСОБ 1: Список адресов в командной строке — то, что мы делали:**

```
ansible all -i '10.0.2.11,10.0.2.12,10.0.2.13' -m ansible.builtin.ping
```

• Годится для разовой проверки.

• ВАЖНО: если адрес один, после него нужна запятая — иначе Ansible решит, что это имя файла:

```
ansible all -i '10.0.2.11,' -m ansible.builtin.ping
```

▸ **СПОСОБ 2: Файл. Так делают всегда, и это практика 2.**

▸ **СПОСОБ 3: Внешний источник: скрипт или плагин, который отдаёт список машин. Так делают, когда парк меняется сам — например, машины создаются в облаке. Это практика 16.**

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: увидеть, что «инвентарь» — это не файл, а понятие. Файл — лишь самый удобный способ его задать.**

# ШАГ 12. ЧТО ANSIBLE ДЕЛАЕТ ПАРАЛЛЕЛЬНО

- Ansible по умолчанию работает не с одной машиной за раз, а с пятью.
- Число задаётся параметром forks:

```
ansible-config dump | grep -i forks
DEFAULT_FORKS(default) = 5
```

- На стенде из трёх машин это незаметно.
- На парке из трёхсот — заметно очень: именно forks определяет, сколько машин обрабатываются одновременно, и именно поэтому «Ansible медленный» обычно означает не Ansible, а слишком маленькое значение forks или...

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: параметр forks и параметр serial (практика 14) решают разные задачи. forks — «сколько машин параллельно», serial — «сколько машин за один заход обновления». Первый про скорость, второй про...**

# Если что-то пошло не так

▸ **ПРОБЛЕМА: Permission denied (publickey).**

```
engineer@10.0.2.11: Permission denied (publickey).
```

• Проверяйте по порядку:

– Ключ вообще есть и права на него верные:
– ls -l ~/.ssh/id_ed25519
– -rw------- 1 engineer engineer 411 ... id_ed25519
– Если у файла есть права для группы или остальных — исправьте:
– chmod 600 ~/.ssh/id_ed25519
– Открытый ключ попал на узел:
– ssh engineer@10.0.2.11 'cat ~/.ssh/authorized_keys'
– Ключ используется именно тот, который скопировали. Указать
– файл явно:
– ssh -i ~/.ssh/id_ed25519 engineer@10.0.2.11 hostname
– Самое частое: ключ скопировали под другим пользователем
– (например, под root), и он лежит в /root/.ssh/authorized_keys.

▸ **ПРОБЛЕМА: sudo: a password is required**

# Проверка результата

- Стенд собран и работает, если выполняются все пункты.
- 1.
- Ansible установлен в окружение и виден:

```
ansible --version
ansible [core 2.19.0]
...
config file = None
Строка config file = None — правильная на этом шаге: файл
конфигурации появится в практике 2.
```

- 2.
- Ansible видит все три узла:

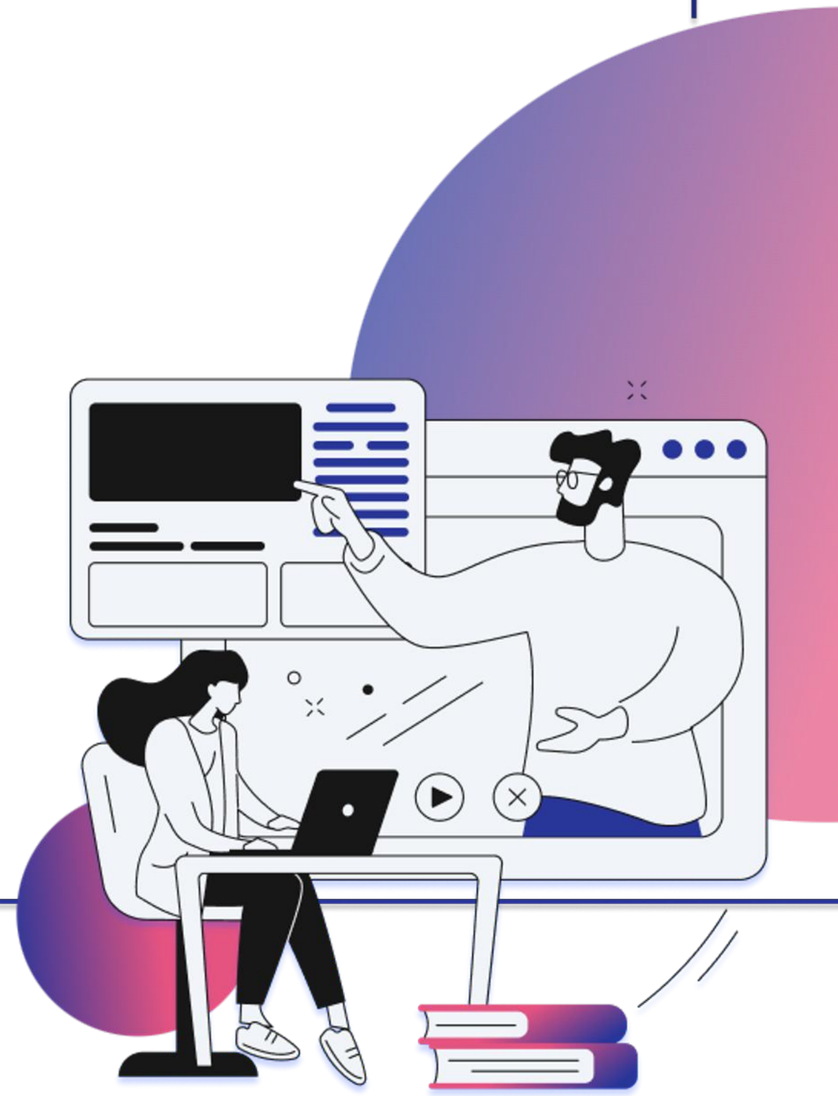
```
ansible all -i '10.0.2.11,10.0.2.12,10.0.2.13' -m ansible.builtin.ping
10.0.2.11 | SUCCESS => { ... "ping": "pong" }
```

- 3.
- Повышение прав работает:
- 4.

# Что дальше

- Далее в курсе:
  - практика 2 — инвентаризация и подключение клиентов: список
  - машин перестанет быть строкой в командной строке и станет
  - файлом inventory.ini, появится ansible.cfg, а подключение
  - настроится один раз на группу машин;
  - практика 3 — язык YAML и ad-hoc управление: те же команды,
  - но модулями и с понятными именами;
  - практика 4 — плейбуки: то, что мы делаем вручную по одной
  - задаче, станет одним файлом, который можно запускать повторно;
  - практика 7 — условия и семейства ОС: зачем курсу машина
  - на Rocky Linux, станет окончательно ясно здесь;
  - практика 9 — административные задачи: пользователи, пакеты,
  - службы и firewall на всём парке;
  - практика 10 — роли: всё, что написано в практиках 4–9,
  - превратится в роль web_admin.

Учебный центр АйТи Клауд



# Практика 2. Инвентаризация и подключение клиентов

**ЭТА ПРАКТИКА ПРЕВРАЩАЕТ СПИСОК АДРЕСОВ ИЗ КОМАНДНОЙ СТРОКИ В ФАЙЛ.**

современные технологии простым языком

# Обзор практики

- ЭТА ПРАКТИКА ПРЕВРАЩАЕТ СПИСОК АДРЕСОВ ИЗ КОМАНДНОЙ СТРОКИ В ФАЙЛ.
- Здесь появятся инвентарь, конфигурация Ansible, переменные для групп и хостов, а вход по паролю на узлах будет запрещён.
- Все следующие практики работают с этим инвентарём и к его устройству не возвращаются.

## Что мы сделаем

- Опишем три узла в файле inventory.ini: группы по назначению, группы по семейству операционной системы и группу, собранную из групп.
- Разберём, чем имя машины отличается от адреса.
- Создадим ansible.cfg: семь параметров — inventory, host_key_checking, remote_user, interpreter_python, stdout_callback, callback_result_format и forks, — четыре места чтения конфигурации и приоритет между ними.
- Перенесём подключение в group_vars и host_vars, чтобы пользователь и ключ были заданы один раз на парк, а различия между Ubuntu и Rocky Linux стали переменными, а не условиями.
- Посмотрим на инвентарь глазами Ansible, отключим вход по паролю и проверим подключение модулем ansible.builtin.ping.
- Обзорно, без практики, разберём Windows и сетевые устройства: подробно это в приложении app_3_windows_network.txt.
- В конце научимся выбирать машины шаблонами: по имени группы, по маске, пересечением и исключением.

# Что понадобится

▸ **СТЕНД ИЗ ПРАКТИКИ 1: Все четыре машины собраны и работают:**

```
control 10.0.2.5 — управляющая машина, здесь мы всё выполняем
web1 10.0.2.11 — Ubuntu 26.04 LTS
web2 10.0.2.12 — Ubuntu 26.04 LTS
db1 10.0.2.13 — Rocky Linux 10
```

- Ansible установлен в ~/ansible-venv, окружение активировано, вход по ключу ed25519 на три узла настроен и проверен.
- ПЕРЕД НАЧАЛОМ сделайте снимок машин: мы запретим вход по паролю, и ошибка в настройке SSH оставит только консоль виртуальной машины.
- Заодно проверьте ключ:

```
ssh -o BatchMode=yes engineer@web1 hostname
web1
```

- ФАЙЛЫ ИЗ КАТАЛОГА КУРСА лежат в files/02_inventory — на случай, если что-то не набралось руками:
  - files/02_inventory/inventory.ini — инвентарь (шаг 2)
  - files/02_inventory/ansible.cfg — настройки (шаг 3)
  - files/02_inventory/group_vars/all.yml — переменные парка (шаг 4)
  - files/02_inventory/group_vars/debian.yml — Ubuntu (шаг 4)
  - files/02_inventory/group_vars/redhat.yml — Rocky Linux (шаг 4)

# ЧАСТЬ 1. БАЗОВЫЕ ШАГИ

-

# ШАГ 1. ИНВЕНТАРЬ: СПИСОК МАШИН ОТДЕЛЬНО ОТ КОМАНД

- В практике 1 список машин жил прямо в командной строке:
  `ansible all -i '10.0.2.11,10.0.2.12,10.0.2.13' -m ansible.builtin.ping`
- Так можно один раз проверить пару машин.
- Для работы это не годится: список пришлось бы набирать в каждой команде, его нельзя положить в систему контроля версий и нельзя приложить к описанию задач.
- Инвентарь (inventory) — это список машин Ansible, записанный в файл.
- В нём три сущности: машины (хосты), к которым Ansible подключается; группы — именованные наборы машин; переменные, привязанные к машине или к группе.
- Инвентарь бывает статический (файл, который правит человек — он у нас) и динамический (скрипт или плагин, который отдаёт список машин сам; об этом в практике 16).
- ▸ **ПОЧЕМУ ИМЕННО ТАК: YAML удобнее, когда структура вложенная — пять уровней подгрупп и переменные у каждой. У нас структура плоская: три машины и пять групп. В INI она видна целиком на одном экране, и это важнее...**
- Создадим рабочий каталог курса:

```
mkdir -p ~/ansible-lab
cd ~/ansible-lab
pwd
/home/engineer/ansible-lab
```

# ШАГ 2. ПИШЕМ INVENTORY.INI

```
cat > inventory.ini <<'EOF'
# Инвентарь учебного стенда курса. Формат INI.
#
# Слева — имя машины, по которому Ansible её знает. Справа —
# переменная ansible_host: адрес, по которому идёт подключение.
[web]
web1 ansible_host=10.0.2.11
web2 ansible_host=10.0.2.12
```

- **ВАЖНО: в тексте практики содержимое файла напечатано с отступом в два пробела — это отступ текста, а не файла. В самом файле строки начинаются с первого столбца, и закрывающее слово EOF тоже: иначе heredoc не...**
- web1 ansible_host=10.0.2.11 — строка машины.
- Первое слово — имя, по которому Ansible знает машину: оно используется в командах, в шаблонах и в паттернах. ansible_host=10.0.2.11 — переменная этой машины: адрес, к которому Ansible подключается.
- Без неё Ansible подключался бы по имени — на стенде оно разрешается через /etc/hosts из практики 1; явный адрес надёжнее.
- [linux:children] — группа из групп: в секции с :children перечислены не машины, а группы, которые в неё входят.
- Так же устроены [debian:children] и [redhat:children].
- Группа all существует всегда и содержит все машины инвентаря: описывать её не нужно, Ansible создаёт её сам.
- Кроме all есть группа ungrouped — машины, не указанные ни в одной группе; у нас таких нет.

# ШАГ 3. ANSIBLE.CFG: ФАЙЛ НАСТРОЕК

- Настройки Ansible живут в файле ansible.cfg в рабочем каталоге:

```
cat > ansible.cfg <<'EOF'
# Конфигурация Ansible для каталога ansible-lab.
# Секция называется [defaults] — во множественном числе.
[defaults]
inventory = ./inventory.ini
host_key_checking = False
remote_user = engineer
interpreter_python = auto_silent
```

- Имя секции здесь — не украшение: Ansible ищет настройки по именам секций, и секцию [defaults] он знает, а секцию [default] — нет.
- Проверка одна — спросить у самого Ansible, откуда взято значение:
- Теперь по параметрам. inventory = ./inventory.ini — откуда брать список машин
- путь относительный, он отсчитывается от текущего каталога, поэтому команды выполняются из ~/ansible-lab. remote_user = engineer — пользователь, под которым Ansible подключается по умолчанию
- в практике 1 мы передавали его флагом -u. interpreter_python = auto_silent — искать Python на узле самому и не рассказывать об этом в выводе
- в практике 1 в ответе модуля ping было поле discovered_interpreter_python — результат той самой находки. forks = 10 — сколько машин обрабатывается одновременно
- по умолчанию их пять, для трёх машин значение роли не играет, но привычка ставить его осознанно окупается на большом парке.

# ШАГ 4. GROUP_VARS И HOST_VARS: ПОДКЛЮЧЕНИЕ ДЛЯ ГРУППЫ И ХОСТА

- ansible.cfg описывает Ansible целиком.
- Переменные описывают машины, и их удобнее держать рядом с инвентарём — в каталогах group_vars и host_vars: Ansible читает их сам, без единой ссылки в команде.

```
mkdir -p group_vars host_vars
cat > group_vars/all.yml <<'EOF'
---
# Переменные для всех машин стенда: файл читается для каждой
# группы и каждого хоста инвентаря.
# Подключение: пользователь engineer и ключ ed25519 из практики 1.
ansible_user: engineer
ansible_ssh_private_key_file: /home/engineer/.ssh/id_ed25519
```

- ansible_user и ansible_ssh_private_key_file — переменные подключения.
- Ansible берёт их раньше значений из ansible.cfg: переменные сильнее конфигурации.
- Поэтому remote_user = engineer и ansible_user: engineer не спорят, а дополняют друг друга, и пользователя с ключом можно переопределить для группы или одной машины, не трогая общий файл.
- ▸ **ВАЖНО: держите эти два значения одинаковыми. Разные значения — это подключение под одним пользователем и долгий разбор вопроса «почему не тот пользователь». Если пользователь на стенде один, выбирайте одно...**
- Теперь различия между семействами.
- Ubuntu и Rocky Linux отличаются именами служб и пакетов, и эти различия удобно держать переменными:

# ШАГ 5. СМОТРИМ НА ИНВЕНТАРЬ ГЛАЗАМИ ANSIBLE

- Файл можно прочитать глазами, но важнее понимать, что из него понял Ansible.
- Для этого есть отдельная команда — ansible-inventory.

```
ansible-inventory --graph
@all:
|--@debian:
| |--@web:
| | |--web1
| | |--web2
|--@linux:
| |--@db:
```

- Символ @ отмечает группы, строки без него — машины.
- Видно главное: web1 и web2 попали сразу в три группы — web, debian и linux, — а db1 тоже в три: db, redhat и linux.
- Группа ungrouped пуста.
- Порядок групп в выводе зависит от того, как они записаны в файле.
- Полный список — в формате JSON:
- Переменные одной машины — то, с чем работает задача:

# ШАГ 6. ПОДКЛЮЧЕНИЕ LINUX: КЛЮЧ, ЗАПРЕТ ПАРОЛЯ, PING

```
ansible all -m ansible.builtin.ping
web1 | SUCCESS => { ... "ping": "pong" }
web2 | SUCCESS => { ... "ping": "pong" }
db1 | SUCCESS => { ... "ping": "pong" }
```

- ansible.builtin.ping — полное имя модуля, FQCN: пространство имён ansible, коллекция builtin, модуль ping.
- Полное имя однозначно: короткое ping может оказаться модулем из другой коллекции, и тогда выполнится не то, что вы имели в виду.
- В ad-hoc командах короткие имена разрешены, но в материалах курса мы пишем полные имена везде; правило разбирается в практике 4.
- **ВАЖНО: модуль ping — это не команда ping. Он не отправляет ICMP-пакеты: Ansible подключается по SSH, запускает на узле Python и получает в ответ pong. Проверка сети — отдельная задача, например ansible all -m...**
- Теперь запретим вход по паролю.
- Делаем это после того, как вход по ключу проверен: если ключ не работает, а пароль запрещён, останется только консоль виртуальной машины.
- На каждом узле создаём файл в /etc/ssh/sshd_config.d:

```
ssh engineer@web1 'sudo tee /etc/ssh/sshd_config.d/10-lab-nopassword.conf >/dev/null' <<'EOF'
PasswordAuthentication no
KbdInteractiveAuthentication no
EOF
```

- Ту же команду повторите для web2 и db1: содержимое одинаковое для Ubuntu и Rocky Linux.

# ЧАСТЬ 2. ЧТО ОТЛИЧАЕТ РАБОТУ ОТ УЧЕБНОГО ПРИМЕРА

-

# ШАГ 7. WINDOWS: ДРУГОЙ СПОСОБ ПОДКЛЮЧЕНИЯ (ОБЗОРНО)

- Этот шаг и следующий — без практики: машин под них в стенде нет.
- Linux-узел отвечает по SSH и имеет Python.
- У Windows-машины нет ни того, ни другого в привычном виде: Ansible работает с ней по WinRM — службе удалённого управления Windows.
- Меняется не инвентарь, а способ подключения и набор модулей.
- Что нужно:
  - на управляющей машине — библиотека pywinrm:
  - pip install pywinrm
  - коллекция с модулями:
  - ansible-galaxy collection install ansible.windows
  - на самой Windows-машине — включённый WinRM; для учебного
  - стенда это делает скрипт ConfigureRemotingForAnsible.ps1.
- Инвентарь выглядит привычно (адрес подставьте свой: машины Windows в стенде курса нет):

```
[windows]
win1 ansible_host=<адрес машины Windows>
[windows:vars]
ansible_connection=winrm
```

# ШАГ 8. СЕТЕВЫЕ УСТРОЙСТВА: ТА ЖЕ ЛОГИКА, ДРУГОЙ ТРАНСПОРТ (ОБЗОРНО)

- У коммутатора нет Python, и модуль на него не скопировать.
- Ansible работает с такими устройствами через подключение network_cli: заходит по SSH и обменивается текстовыми командами командной строки устройства.
- Инвентарь (адрес снова свой — коммутатора в стенде нет):

```
[switches]
sw1 ansible_host=<адрес коммутатора>
[switches:vars]
ansible_connection=network_cli
ansible_network_os=cisco.ios.ios
ansible_user=admin
```

- Подключение network_cli даёт коллекция ansible.netcommon, а модули — коллекции под конкретного производителя: cisco.ios, arista.eos, juniper.junos, community.network:

```
ansible-galaxy collection install ansible.netcommon cisco.ios
```

- Пример команды для устройства — тот же шаблон, что и везде:

```
ansible sw1 -i switches.ini -m cisco.ios.ios_command -a "commands='show version'"
```

- Ни шаблонов Jinja2, ни фактов в привычном виде здесь нет: конфигурация устройства — это набор строк, а не файл, который можно положить на место.
- Поэтому задачи для сети выглядят иначе.
- Подробно: приложение app_3_windows_network.txt.

# ШАГ 9. ШАБЛОНЫ ХОСТОВ И ГРУПП: ВЫБОР МАШИН

- Ansible умеет выбирать машины шаблонами (patterns) — по имени группы, по имени машины и по маске.
- Полезная привычка: сначала посмотреть, кого выбрал шаблон, и только потом подключаться.
- Для этого есть --list-hosts.

```
ansible 'web' --list-hosts
hosts (2):
web1
web2
ansible 'web*' --list-hosts
hosts (2):
web1
web2
```

- Пересечение и исключение.
- Двоеточие разделяет условия, символ & оставляет только общее, символ ! убирает:
- Одиночное исключение тоже работает: '!db1' — это то же самое, что 'all:!db1'.
- Почему шаблон в кавычках: символ & в оболочке означает запуск команды в фоне, а ! — обращение к истории команд.
- ▸ **ВАЖНО: шаблон ищет имя из инвентаря, а не адрес. Команда ansible '10.0.2.11' --list-hosts не найдёт ничего: 10.0.2.11 — это значение переменной ansible_host, а не имя машины.**

# ШАГ 10. СЕМЕЙСТВА ОС: ПЕРЕМЕННЫЕ ВМЕСТО УСЛОВИЙ

- В парке две машины на Ubuntu и одна на Rocky Linux.
- В шаге 4 мы записали различия между ними переменными.
- Посмотрим, что получилось:

```
ansible-inventory --host web1 | grep -E 'web_|firewall_'
"firewall_package": "ufw",
"web_package": "apache2",
"web_service": "apache2"
ansible-inventory --host db1 | grep -E 'web_|firewall_'
"firewall_package": "firewalld",
"web_package": "httpd",
"web_service": "httpd"
```

- Один и тот же запрос, разные значения — и ни одного условия в задачах.
- Описание, которое поставит web_package и запустит web_service, будет работать на обоих семействах без единой строчки «если»: различие уже выражено данными.
- Второе применение видно здесь же: служба SSH называется по-разному, и это тоже переменная — ssh_service.
- Одна и та же задача сможет перезагрузить её и на Ubuntu, и на Rocky Linux.

▸ **ПОЧЕМУ ИМЕННО ТАК: так различия между семействами собираются в двух файлах, а не расползаются по задачам. Практика 7 добавит условия when — они нужны там, где различие не выражается значением переменной,...**

# Если что-то пошло не так

▸ **ПРОБЛЕМА: [WARNING]: Unable to parse ... as an inventory source**

```
[WARNING]: Unable to parse /home/engineer/ansible-lab/inventory.ini as an inventory source
[WARNING]: No inventory was parsed, only implicit localhost is available
```

- Ansible не нашёл инвентарь по тому пути, который получился, и остался с одной машиной localhost.
- Чаще всего дело не в содержимом файла, а в пути: в ansible.cfg написано inventory = ./inventory.ini, а команда запущена не из ~/ansible-lab.
- Проверяйте по порядку:
  - Где мы находимся и что лежит рядом:
  - pwd
  - ls -l inventory.ini
  - Какой файл конфигурации взят и что Ansible видит в инвентаре:
  - ansible --version | grep 'config file'
  - ansible-inventory --list -i inventory.ini

▸ **ПРОБЛЕМА: [WARNING]: Could not match supplied host pattern, ignoring: web3**

```
[WARNING]: Could not match supplied host pattern, ignoring: web3
```

- Шаблон не совпал ни с одним именем машины или группы.
- Ansible не считает это ошибкой: он предупреждает и продолжает работу с теми машинами, которые совпали.

# Проверка результата

- Практика выполнена, если выполняются все пункты.
- 1.
- Файлы на месте и лежат там, где нужно:

```
cd ~/ansible-lab
ls -1 ansible.cfg inventory.ini group_vars host_vars
```

- 2.
- Ansible нашёл конфигурацию и применил настройки:

```
ansible --version | grep 'config file'
config file = /home/engineer/ansible-lab/ansible.cfg
ansible-config dump --only-changed
...в скобках стоит путь к ansible.cfg, а не (default)
```

- 3.
- Инвентарь разбирается целиком, и машин ровно три:

```
ansible-inventory --graph
...в дереве web1, web2 и db1, группа ungrouped пуста
```

- 4.

# Что дальше

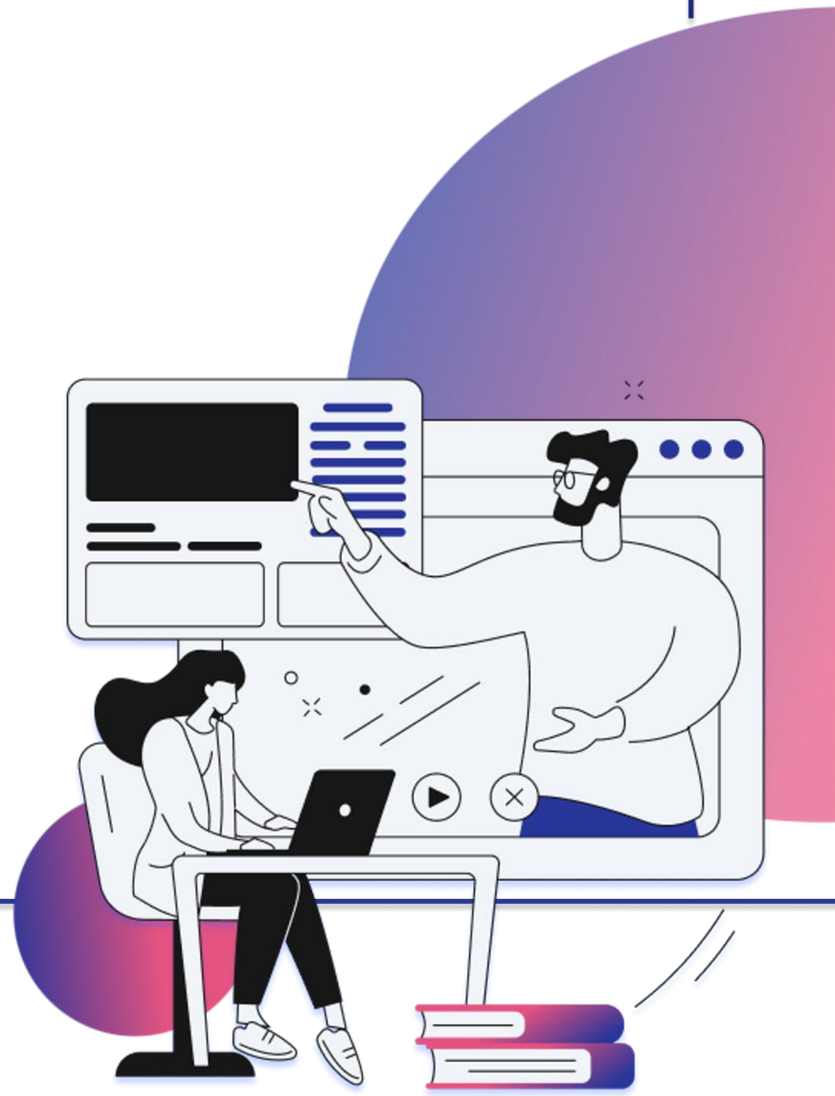
- Далее в курсе:
  - практика 3 — язык YAML и ad-hoc управление: инвентарь и настройки
  - готовы, можно переходить от одиночных команд к модулям
  - и разбирать YAML, на котором написаны group_vars;
  - практика 4 — плейбуки: первый файл с задачами, который берёт
  - машины из нашего инвентаря, а не из командной строки;
  - практика 9 — административные задачи: модуль authorized_key
  - разложит ключи новым пользователям, а ssh_service перезагрузит
  - службу на обоих семействах;
- ПЕРЕД ПЕРЕХОДОМ уберите за собой:

  `убирать нечего: все созданные файлы — часть проекта курса, и они нужны в следующих практиках.`
- Единственное, что стоит сделать сейчас: снимок машин.

Учебный центр АйТи Клауд

# Практика 3. Язык YAML и ad-hoc управление

**ЭТА ПРАКТИКА — ПРО ДВА УМЕНИЯ, КОТОРЫЕ ДЕРЖАТСЯ ДРУГ ЗА ДРУГА: ЯЗЫК YAML, НА КОТОРОМ ОПИСАН ВЕСЬ КУРС, И AD-HOC КОМАНДЫ — ОДНА ЗАДАЧА ОДНОЙ СТРОКОЙ, БЕЗ ФАЙЛА.**



современные технологии простым языком

# Обзор практики

- ЭТА ПРАКТИКА — ПРО ДВА УМЕНИЯ, КОТОРЫЕ ДЕРЖАТСЯ ДРУГ ЗА ДРУГА: ЯЗЫК YAML, НА КОТОРОМ ОПИСАН ВЕСЬ КУРС, И AD-HOC КОМАНДЫ — ОДНА ЗАДАЧА ОДНОЙ СТРОКОЙ, БЕЗ ФАЙЛА.
- YAML РАЗБИРАЕМ ДО ПЕРВОГО ПЛЕЙБУКА, А AD-HOC ДАЁТ БЫСТРЫЙ СПОСОБ ЧТО-ТО ПРОВЕРИТЬ И ПОМЕНЯТЬ.

## Что мы сделаем

- Разберём YAML по частям: отступы, скаляры, списки, словари, вложенность, комментарии, границы документа.
- Отдельно посмотрим на типы, в которых ошибаются чаще всего: логические значения, права доступа, числа, оказавшиеся строками, и многострочные значения.
- Сделаем файл с ошибкой отступа и прочитаем настоящее сообщение разборщика YAML целиком — не пересказ, а строку, которую вы увидите в терминале.
- Научимся проверять YAML без Ansible: короткой командой Python и линтером.
- Потом возьмём ad-hoc управление: разберём ключи, сравним модули command и shell, разберём полное имя модуля, поставим и удалим пакет, посмотрим факты и справку по модулю.
- Закончим разговором о том, почему ad-hoc не заменяет плейбук.

# Что понадобится

- ЧЕТЫРЕ МАШИНЫ ИЗ ПРАКТИКИ 1:

```
control 10.0.2.5 — управляющая машина, на ней идёт вся работа
web1 10.0.2.11 — Ubuntu 26.04 LTS
web2 10.0.2.12 — Ubuntu 26.04 LTS
db1 10.0.2.13 — Rocky Linux 10
```

- СОСТОЯНИЕ СТЕНДА после практики 2: рабочий каталог /home/engineer/ansible-lab, инвентарь /home/engineer/ansible-lab/ inventory.ini с группами web, db, linux, debian и redhat, конфигурация...

▸ **ВАЖНО: окружение нужно активировать в каждом новом сеансе терминала:**

```
cd ~/ansible-lab
source ~/ansible-venv/bin/activate
```

- Без активации команды ansible не найдутся, а python3 не увидит PyYAML.
- Про ошибку «ansible: command not found» подробно сказано в практике 1.
- ЧТО ЕЩЁ НУЖНО:
  - доступ в интернет с control: мы один раз скачаем файл с узла в интернете,
  - чтобы показать модуль get_url;
  - каталог ~/ansible-lab/yaml — в нём мы наберём пять файлов примеров.
  - Эталоны лежат в курсе и напечатаны в практике целиком:

# ЧАСТЬ 1. ЯЗЫК YAML: ОТСТУПЫ, ТИПЫ, ПРОВЕРКА

-

# ШАГ 1. ЗАЧЕМ YAML РАЗБИРАЕТСЯ ВМЕСТЕ С AD-HOC

- В практике 1 мы уже запускали ad-hoc команды, не называя их так.
- Ad-hoc (от латинского «к этому», то есть «для этого случая») — запуск одного модуля на выбранных машинах прямо из командной строки, без файла:

```
ansible all -i '10.0.2.11,10.0.2.12,10.0.2.13' -m ansible.builtin.ping
```

- Дальше в курсе почти всё описывается в файлах на языке YAML.
- И первая же ошибка новичка — не в Ansible, а в YAML: не тот отступ, пропущенный пробел после двоеточия, кавычки не там, где надо.

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: из десяти обращений «Ansible не работает» восемь заканчиваются фразой «в файле не тот отступ». Строка YAML читается глазами, но структуру задаёт отступ — и человек видит не то, что видит...**

▸ **ПОЧЕМУ ИМЕННО ТАК: YAML выбран потому, что читается человеком. Скобки и запятые заменены отступами и переводами строк, а это ровно то, что нужно файлу в репозитории. Плата за читаемость — чуткость к пробелам.**

```
mkdir -p ~/ansible-lab/yaml
cd ~/ansible-lab/yaml
```

▸ **ПРОВЕРКА: каталог создан и мы находимся в нём.**

```
pwd
/home/engineer/ansible-lab/yaml
```

# ШАГ 2. ОТСТУПЫ: ДВА ПРОБЕЛА И НИ ОДНОГО ТАБА

- Структуру YAML задаёт отступ.
- Один уровень вложенности в курсе — два пробела.
- Язык допускает и другое число, лишь бы оно было одинаковым, но у двух пробелов два преимущества: это самый распространённый стиль, и на него настроен lint (практика 16).

▸ **ТАБУЛЯЦИИ ЗАПРЕЩЕНЫ: Это не пожелание стиля, а правило языка: таб в отступе разборщик YAML не принимает. Причина историческая — ширина таба в разных редакторах разная, и файл, который у одного выглядит ровным,...**

```
grep -nP '\t' *.yml
```

- Если команда ничего не вывела — табов нет.
- Выражение стоит в одинарных кавычках, поэтому оболочка знак табуляции не раскрывает.

▸ **ПОЧЕМУ ИМЕННО ТАК: два пробела — самый частый вариант в документации Ansible и в ролях из Galaxy. Читая чужой плейбук, вы почти всегда видите именно их, а разнобой в отступах внутри одного файла — то, за что...**

▸ **ЧЕГО НЕ ДЕЛАТЬ: не выравнивайте отступы вручную «по красоте» и не смешивайте в одном файле два и четыре пробела. YAML такой файл разберёт, если каждый блок внутри себя ровный, а человек — уже нет. Настройте в...**

▸ **ПРОВЕРКА: в каталоге ~/ansible-lab/yaml пока пусто.**

```
ls -A
(пусто)
```

# ШАГ 3. СКАЛЯРЫ, КОММЕНТАРИИ И ГРАНИЦЫ ДОКУМЕНТА

- Скаляр — одиночное значение: строка, число, логическое значение.
- Наберём первый файл целиком и разберём его построчно.

```
---
# Файл 01_scalars.yml. Только скаляры — одиночные значения.
# Комментарий начинается с решётки; внутри строки он должен быть
# отделён пробелом, иначе станет частью значения.
site_name: ansible-lab # строка без кавычек
site_port: 80 # целое число
site_root: /var/www/html # строка, похожая на путь
site_owner: engineer # строка без кавычек
```

- Что здесь стоит заметить:
  - КЛЮЧ И ЗНАЧЕНИЕ РАЗДЕЛЯЕТ ДВОЕТОЧИЕ С ПРОБЕЛОМ. Если пробела нет,
  - «site_name:ansible-lab» — не пара ключ-значение, а одна строка с
  - двоеточием внутри;
  - КАВЫЧКИ НЕОБЯЗАТЕЛЬНЫ, но нужны там, где без них значение сломается:
  - двоеточие с пробелом, решётка, ведущий или концевой пробел, а также слова

# ШАГ 4. СПИСКИ

- Список — набор значений.
- В блочной записи каждый элемент начинается с дефиса и пробела:

```
---
# Файл 02_lists.yml. Списки: каждый элемент начинается с "- ".
# Список строк. Дефисы стоят на одном уровне, отступ — два пробела
# от имени ключа.
packages:
- mc
- curl
- tree
```

- ЧТО ЗДЕСЬ ВАЖНО:
  - ВСЕ ДЕФИСЫ ОДНОГО СПИСКА СТОЯТ НА ОДНОМ УРОВНЕ. Сдвинете один дефис —
  - получится список внутри элемента списка: язык это разберёт, а смысл будет
  - другим;
  - ОТСТУП ПОСЛЕ ДЕФИСА. В элементе-словаре ключи выравниваются по первой
  - букве имени, то есть на два пробела правее дефиса;

# ШАГ 5. СЛОВАРИ И ВЛОЖЕННОСТЬ

- Словарь — набор пар «ключ: значение».
- Словари вкладываются друг в друга, и так выглядит любой плейбук: play — словарь, tasks — список словарей, у каждой задачи есть параметры.

```
---
# Файл 03_dicts.yml. Словари — пары «ключ: значение» — и вложенность.
# Двоеточие пишется сразу после ключа, пробел после него обязателен.
server:
name: web1
address: 10.0.2.11
family: Debian
roles:
```

- ЧТО ЗДЕСЬ ВИДНО:
  - УРОВЕНЬ ВЛОЖЕННОСТИ — ЭТО УРОВЕНЬ СМЫСЛА. service внутри server означает
  - «служба этой машины», а не отдельную сущность;
  - СЛОВАРЬ ВНУТРИ СПИСКА И СПИСОК ВНУТРИ СЛОВАРЯ — обычное дело. roles —
  - список строк внутри словаря server, nodes — список словарей на верхнем
  - уровне;

# ШАГ 6. ТИПЫ, КОТОРЫЕ ЛОВЯТ НОВИЧКОВ

- YAML сам определяет тип значения.
- Строка без кавычек может оказаться числом или логическим значением — и это самая дорогая ловушка языка, потому что выглядит всё правильно.

```
---
# Файл 04_multiline.yml. Типы, которые ловят новичков,
# и многострочные значения.
# 1. Логика: true и false — логические значения, а не строки.
run_backup: true
run_backup_as_text: "true" # строка из четырёх букв
# yes и no YAML тоже читает как логику. Поэтому в курсе пишут
# только true и false, а строку «yes» берут в кавычки.
```

- Теперь посмотрим, во что превратил эти строки разборщик:
- Разбираем построчно — это самое полезное упражнение практики:
  - run_backup — логическое True, run_backup_as_text — строка 'true'. Снаружи
  - разница незаметна, внутри это разные значения;
  - answer_plain — тоже логическое True. Слово yes в YAML 1.1 значит то же,
  - что true. Поэтому в курсе везде пишут true и false, а если нужна именно

# ШАГ 7. МНОГОСТРОЧНЫЕ ЗНАЧЕНИЯ: | И >

- Многострочный текст пишут двумя знаками:
  - ВЕРТИКАЛЬНАЯ ЧЕРТА | (literal) сохраняет переводы строк: каждая строка
  - значения станет отдельной строкой в файле;
  - ЗНАК БОЛЬШЕ > (folded) склеивает соседние строки в одну, а перевод строки
  - остаётся только там, где была пустая строка.
- Выбор зависит от того, куда значение попадёт:
  - файл конфигурации, где важна каждая строка (motd, список параметров,
  - ключ) — вертикальная черта;
  - длинное описание или сообщение — знак больше: он избавляет от ручных
  - переносов посреди абзаца.
- СРАВНИТЕ два значения из файла шага 6:

```
motd_block: |
Добро пожаловать на узел.
Работы ведутся через Ansible.
motd_folded: >
Добро пожаловать на узел.
```

# ШАГ 8. ПРОВЕРКА YAML БЕЗ ANSIBLE

- YAML можно проверить, не запуская ни одной задачи.
- Это экономит время: ошибка находится за секунду, а не после подключения к трём машинам.
- Способ первый — короткая команда на Python 3.

```
python3 -c "import yaml, sys; yaml.safe_load(open(sys.argv[1]))" 05_broken.yml
```

- **ПОЧЕМУ ИМЕННО ТАК: safe_load, а не load. Функция load умеет создавать объекты Python прямо из YAML — это удобно для сериализации и опасно для чужого файла. safe_load разбирает только обычные данные: строки,...**
- PyYAML в системный Python Ubuntu не входит: он приходит вместе с Ansible, поэтому команду запускаем из активированного окружения ~/ansible-venv.
- Если нужен системный python3, поставьте python3-yaml.
- Теперь файл с намеренной ошибкой: строка port сдвинута вправо на один пробел — этого достаточно.

```
---
# Файл 05_broken.yml. Здесь намеренная ошибка: строка port сдвинута
# вправо на один пробел. Файл нужен, чтобы увидеть настоящий текст
# ошибки YAML, а не прочитать про неё.
server:
name: web1
service:
```

- Что скажет разборщик:

# ЧАСТЬ 2. AD-HOC УПРАВЛЕНИЕ: ОДНА ЗАДАЧА ОДНОЙ СТРОКОЙ

-

# ШАГ 9. ЧТО ТАКОЕ AD-HOC И КОГДА ЕГО ХВАТАЕТ

- Ad-hoc — запуск одного модуля на выбранных машинах прямо из командной строки.
- Файла нет, задачи нет, есть одна команда и её результат:

```
ansible <образец машин> -m <модуль> -a "<параметры>" [ключи]
```

- Ad-hoc хорош там, где нужно сейчас и один раз: посмотреть, что за машины в парке и чем они отличаются; проверить, что узел доступен; поставить или убрать пакет
- остановить или запустить службу; скопировать или скачать файл; снять факт с узла.
- Ad-hoc не годится там, где работу нужно повторить.
- Команда из истории терминала не лежит в репозитории, её нельзя отревьюить, она не описывает состояние и не даёт одинакового результата при повторном запуске.
- Для этого есть плейбук — практика 4.

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: в реальной эксплуатации ad-hoc занимает больше времени, чем плейбуки. Инженер ставит пакет на одну машину, смотрит факты, ищет, почему узел отстал от остальных. Ad-hoc — рабочий инструмент...**

▸ **ВАЖНО: ad-hoc работает от того инвентаря и той конфигурации, которые видит Ansible. Мы работаем из каталога ~/ansible-lab, где лежит ansible.cfg из практики 2, поэтому файл инвентаря указывать не обязательно.**

▸ **ПРОВЕРКА: все три узла отвечают.**

```
ansible all -m ansible.builtin.ping
web1 | SUCCESS => { ... "ping": "pong" }
web2 | SUCCESS => { ... "ping": "pong" }
db1 | SUCCESS => { ... "ping": "pong" }
```

- Если вы видите не имена машин, а адреса — Ansible взял не тот инвентарь: проверьте, что вы в каталоге ~/ansible-lab.

# ШАГ 10. КЛЮЧИ AD-HOC: ЧТО ЗА ЧТО ОТВЕЧАЕТ

- Ключей немного, и они повторяются в плейбуках почти один в один.
  -m — МОДУЛЬ: -m ansible.builtin.ping, -m ansible.builtin.command. Короткие имена (ping, command) тоже работают, и в ad-hoc это разрешено (шаг 12).
  -a — АРГУМЕНТЫ МОДУЛЯ одной строкой в кавычках: -a "name=mc state=present". Это те же параметры, что в плейбуке пишутся в столбик, только в одну строку.
  -b — BECOME, повышение прав до root через sudo. Нужно везде, где меняется система: пакеты, службы, файлы вне домашнего каталога.
- Теперь примеры.
- Обычная команда без повышения прав:
- Та же команда с повышением прав — от чьего имени она выполняется:
- Ноль означает root.
- Инвентарь можно указать явно:
- Ограничить одну машину:
- Разница между forks видна на команде с паузой.

# ШАГ 11. COMMAND ПРОТИВ SHELL

- Два модуля запускают команды, и разница между ними не в удобстве, а в том, участвует ли в запуске оболочка.

```
ansible.builtin.command — запускает программу напрямую. Оболочки
(/bin/sh) в цепочке нет, поэтому подстановок она не делает.
ansible.builtin.shell — запускает команду через /bin/sh -c.
Работают |, >, &&, ||, $HOME, *, ~, подстановка переменных.
```

- Проверим на одном и том же:

```
ansible web1 -m ansible.builtin.command -a 'echo $HOME'
web1 | CHANGED | rc=0 >>
$HOME
ansible web1 -m ansible.builtin.shell -a 'echo $HOME'
```

- Команда одна и та же, результат разный: там, где нет оболочки, переменная не раскрывается.
- Обратите внимание на кавычки: они одинарные.
- В двойных переменную раскрыла бы ваша оболочка на управляющей машине, ещё до того как Ansible получит аргумент, — и опыт показал бы не то, что нужно.
- ПОЧЕМУ ansible.builtin.command ПРЕДПОЧТИТЕЛЬНЕЕ: он не даёт выполнить лишнего.
- Ни конвейер, ни перенаправление, ни подстановка переменных в нём не работают — значит, значение, пришедшее из переменной, не превратится в команду.
- Это и защита от ошибок, и защита от чужих данных.

# ШАГ 12. ПОЛНОЕ ИМЯ МОДУЛЯ: ПРАВИЛО FQCN

• У каждого модуля есть полное имя из трёх частей: пространство имён, коллекция, модуль.

```
ansible.builtin.copy ядро Ansible, модуль copy
ansible.builtin.dnf ядро Ansible, модуль dnf
community.general.lvg коллекция community.general, модуль lvg
ansible.posix.sysctl коллекция ansible.posix, модуль sysctl
```

▸ **ПРАВИЛО КУРСА: в плейбуках и ролях модули пишутся полным именем (FQCN — fully qualified collection name). В ad-hoc командах разрешены короткие имена: писать ansible.builtin. в каждой строке терминала...**

▸ **ПОЧЕМУ ИМЕННО ТАК: коротких имён мало, и они могут совпасть. В разных коллекциях встречаются модули с одинаковым именем, и тогда непонятно, какой выполнится. Полное имя снимает вопрос и заодно говорит...**

▸ **ПРОВЕРКА: справка открывается и по полному, и по короткому имени.**

```
ansible-doc ansible.builtin.copy
ansible-doc copy
```

▸ **ВАЖНО: в ad-hoc короткое имя выбирается из ядра Ansible и из установленных коллекций. Если модуль живёт в коллекции, которой на управляющей машине нет, короткое имя не найдётся — появится ошибка couldn't...**

# ШАГ 13. ФАЙЛЫ ОДНОЙ СТРОКОЙ: COPY, FILE, GET_URL

- Три модуля закрывают почти всю работу с файлами.
- COPY переносит файл с управляющей машины на узлы:

```
ansible web -m ansible.builtin.copy -a "src=/home/engineer/ansible-lab/yaml/01_scalars.yml dest=/tmp/01_sca...
web1 | CHANGED => { "changed": true, "dest": "/tmp/01_scalars.yml", ... }
web2 | CHANGED => { "changed": true, "dest": "/tmp/01_scalars.yml", ... }
```

- Повторный запуск той же команды изменений не вносит:

```
ansible web -m ansible.builtin.copy -a "src=/home/engineer/ansible-lab/yaml/01_scalars.yml dest=/tmp/01_sca...
web1 | SUCCESS => { "changed": false, ... }
web2 | SUCCESS => { "changed": false, ... }
```

- Так работает большинство модулей Ansible: они проверяют состояние и меняют систему только тогда, когда это нужно.
- FILE управляет самим объектом: каталогом, ссылкой, правами:

```
ansible web -m ansible.builtin.file -a "path=/tmp/ansible-lab state=directory mode=0755"
web1 | CHANGED => { "changed": true, "path": "/tmp/ansible-lab", "state": "directory" }
```

- Удаление — тот же модуль с другим состоянием.
- «Удалить каталог» — это не команда rm, а описание состояния absent:
- GET_URL скачивает файл из интернета прямо на узел:

# ШАГ 14. ФАКТЫ ОДНОЙ КОМАНДОЙ: МОДУЛЬ SETUP С ФИЛЬТРОМ

- Прежде чем что-то менять на машине, полезно узнать, что это за машина.
- Модуль ansible.builtin.setup собирает факты — сведения об узле: семейство ОС, версию ядра, память, диски, адреса.
- Все факты сразу — это сотни строк, а нам нужен один.
- Параметр filter принимает образец имени факта:

```
ansible db1 -m ansible.builtin.setup -a "filter=ansible_os_family"
db1 | SUCCESS => {
"ansible_facts": {
"ansible_os_family": "RedHat"
},
"changed": false
}
ansible web1 -m ansible.builtin.setup -a "filter=ansible_os_family"
```

- Два значения — RedHat и Debian.
- Это те самые семейства, по которым в практике 7 будут выбираться пакеты и службы: apache2 против httpd, ufw против firewalld.
- Полезно посмотреть и на пакетный менеджер:

▸ **ПОЧЕМУ ИМЕННО ТАК: filter — не «сократить вывод для красоты». Сбор всех фактов на парке из трёхсот машин занимает заметное время, а в плейбуке факты собираются перед задачами. Точечный фильтр — способ не...**

# ШАГ 15. ПАКЕТЫ И СЛУЖБЫ: MC, APT, DNF, SERVICE

- Сначала смотрим факты (шаг 14) и выбираем модуль: на web1 и web2 это apt, на db1 — dnf.
- Один и тот же смысл «пусть пакет будет установлен» на двух семействах записывается двумя модулями.
- СТАВИМ MC НА ВЕБ-СЕРВЕРАХ:

```
ansible web -m ansible.builtin.apt -a "name=mc state=present update_cache=true" -b
web1 | CHANGED => { "changed": true, ... }
web2 | CHANGED => { "changed": true, ... }
ansible web -m ansible.builtin.command -a "which mc"
web1 | CHANGED | rc=0 >>
/usr/bin/mc
web2 | CHANGED | rc=0 >>
/usr/bin/mc
```

- Параметр update_cache=true означает «обнови список пакетов, если он устарел»: тот же apt update, только сделанный модулем.
- Если пакет не найден, модуль отвечает No package matching 'mc' is available — значит, в подключённых репозиториях его нет (в Ubuntu это обычно отключённый universe, на Rocky — не подключённый EPEL).
- ТОТ ЖЕ СМЫСЛ НА ROCKY — ДРУГИМ МОДУЛЕМ.
- Пакет mc на Rocky живёт не в базовых репозиториях, а в EPEL, которого на стенде нет, поэтому покажем модуль dnf на пакете, который там точно есть, — python3:
- Здесь важны две вещи.

# ШАГ 16. ANSIBLE-DOC: СПРАВКА ПО МОДУЛЮ В ТЕРМИНАЛЕ

- Справка встроена в Ansible и не требует интернета.
- В практике 1 мы уже смотрели ansible-doc -l; здесь разберём три режима на примере apt.
- СПИСОК МОДУЛЕЙ с кратким описанием: их сотни, поэтому фильтруем.

```
ansible-doc -l | grep -i '^ansible.builtin.apt'
ansible.builtin.apt Manages apt-packages
```

- **ПОЛНАЯ СПРАВКА: описание, все параметры, примеры использования.**

```
ansible-doc ansible.builtin.apt
> ANSIBLE.BUILTIN.APT (/home/engineer/ansible-venv/lib/python3.13/site-packages/ansible/modules/apt.py)
...
```

- **КРАТКАЯ СПРАВКА: только параметры, в виде заготовки задачи. Именно эта форма нужна чаще всего.**

```
ansible-doc -s ansible.builtin.apt
- name: Manages apt-packages
ansible.builtin.apt:
```

- Теперь главное, ради чего этот шаг в практике: параметры модуля одинаковы и в ad-hoc, и в плейбуке.
- Ключ -a принимает те же самые параметры, только записанные в одну строку.
- Поэтому между ad-hoc и плейбуком нет «двух разных Ansible»: это один модуль с одними параметрами, и справка ansible-doc -s годится для обоих случаев.

# ШАГ 17. ПОЧЕМУ AD-HOC — ЭТО ВСЁ ЕЩЁ НЕ ПЛЕЙБУК

- Ad-hoc удобен, и именно поэтому им легко заменить нормальную работу.
- Так появляется вики-страница «команды, которые надо выполнить по порядку», и это ровно то, от чего Ansible избавляет.
- Чего в ad-hoc нет:
  - ОПИСАНИЯ СОСТОЯНИЯ. Команда говорит «сделай», а не «пусть будет». Разница
  - видна на повторном запуске: плейбук отвечает changed=0, ad-hoc просто
  - выполняется снова;
  - ИСТОРИИ. Команда живёт в истории терминала, а не в репозитории. Её нельзя
  - отревьюить, откатить и сравнить с прошлой версией;
  - МНОГИХ ЗАДАЧ. Одна команда — одна задача. Установить пакет, поправить
  - конфиг и перезапустить службу — это уже три команды, между которыми нет
  - ни порядка, ни обработки ошибки;
  - ОБРАБОТЧИКОВ, ПЕРЕМЕННЫХ, УСЛОВИЙ И ЦИКЛОВ. Всё это появляется в
  - плейбуках: практики 4–7;
  - ВОСПРОИЗВОДИМОСТИ. «Выполните эти двенадцать команд в таком порядке» — не
  - описание инфраструктуры, а инструкция для человека.

▸ **ПОЧЕМУ ЭТО ВАЖНО ИМЕННО СЕЙЧАС: следующая практика начинается с файла. Всё, что мы делали одной строкой, станет одной задачей в плейбуке, а состояние — параметром state, который мы уже видели в модулях apt,...**

# Если что-то пошло не так

▸ **ПРОБЛЕМА: ERROR! Missing target hosts**

```
ansible webb1 -m ansible.builtin.ping
[WARNING]: Could not match supplied host pattern, ignoring: webb1
ERROR! Missing target hosts
```

• Опечатка в имени машины или машины нет в инвентаре: Ansible нашёл инвентарь, но ни один образец машин не совпал ни с чем.

• Порядок проверки:

– Посмотрите имена машин в инвентаре: ansible-inventory --graph.
– Проверьте конкретное имя: ansible-inventory --host web1.
– Сравните написание: web1 и webb1 отличаются одним символом,
– а сообщение об этом не говорит. Если имя верное, но машины
– в выводе нет — её нет в inventory.ini, правьте инвентарь.

▸ **ПРОБЛЕМА: модуль command отказался работать с | или >**

```
ansible web1 -m ansible.builtin.command -a "ls /tmp | wc -l"
web1 | FAILED! => {
"changed": false,
"cmd": ["ls", "/tmp", "|", "wc", "-l"],
```

# Проверка результата

- 1.
- Пять файлов YAML набраны в ~/ansible-lab/yaml и разбираются.

```
На 01, 02, 03 и 04 команда молчит:
python3 -c "import yaml, sys; yaml.safe_load(open(sys.argv[1]))" 01_scalars.yml
```

- 2.
- Ansible видит три узла и берёт инвентарь из ansible.cfg:

```
ansible all -m ansible.builtin.ping
web1 | SUCCESS => { ... "ping": "pong" }
web2 | SUCCESS => { ... "ping": "pong" }
db1 | SUCCESS => { ... "ping": "pong" }
```

- 3.
- Два семейства ОС видны по фактам: web1 — Debian, db1 — RedHat.

```
ansible db1 -m ansible.builtin.setup -a "filter=ansible_os_family"
db1 | SUCCESS => { ... "ansible_os_family": "RedHat" ... }
```

- 4.
- Пакет mc установлен и удалён, повторное удаление изменений

# Что дальше

- Далее в курсе:
  - практика 4 — плейбуки: первый рабочий сценарий. Всё, что мы делали ad-hoc
  - одной строкой, станет задачами в файле; там же появится идемпотентность,
  - которую мы видели в changed: false, и правило FQCN целиком;
  - практика 5 — переменные и секреты: значения из плейбука переедут в файлы
  - переменных, и YAML станет не описанием задачи, а хранилищем данных;
  - практика 6 — факты и шаблоны Jinja2: модуль setup из шага 14 станет
  - основой страницы сайта, а факты — источником данных для шаблона;
  - практика 7 — условия, циклы, блоки: разница ssh и sshd, apt и dnf
  - превратится в условие when по ansible_os_family;
  - практика 8 — файлы и права: модули copy, file и get_url из шага 13 станут
  - задачами с правами, владельцами и проверкой результата;
  - практика 9 — административные задачи: установка пакетов, службы и
  - репозитории на всём парке одним плейбуком;
  - практика 16 — качество автоматизации: ansible-lint, с которым мы
  - познакомились в шаге 8, разбирается подробно, вместе с правилами и

Учебный центр АйТи Клауд

# Практика 4. Плейбуки: первый рабочий сценарий

**ЭТА ПРАКТИКА ПРЕВРАЩАЕТ ОТДЕЛЬНЫЕ AD-HOC КОМАНДЫ В ФАЙЛ.**

современные технологии простым языком



# Обзор практики

- ЭТА ПРАКТИКА ПРЕВРАЩАЕТ ОТДЕЛЬНЫЕ AD-HOC КОМАНДЫ В ФАЙЛ.
- Здесь появится первый плейбук: список play, задачи с именами, состояния вместо команд и обработчик, который перезапускает службу только тогда, когда конфигурация изменилась.
- Главное слово практики — идемпотентность: один и тот же файл запускается дважды, и второй прогон не меняет на машинах ничего.

## Что мы сделаем

- Напишем четыре плейбука, каждый следующий сложнее предыдущего: проверка связи, установка веб-сервера, своя страница сайта и рабочий сценарий целиком — с пакетом, файлом, службой и обработчиком.
- Разберём структуру плейбука: почему корень — список, что означают ключи name, hosts, become и tasks, зачем каждой задаче имя.
- Запустим плейбук дважды и прочитаем PLAY RECAP по колонкам: ok, changed, unreachable, failed, skipped, rescued, ignored.
- Убедимся, что второй прогон даёт changed=0.
- Выясним, почему на db1 (Rocky Linux 10) пакет называется httpd, а плейбук ставит apache2, и почему это не опечатка, а заготовка под практику 7.
- Научимся проверять плейбук до запуска: --syntax-check, --check, --diff, --list-tasks, --list-hosts, и разберём, чего --check не гарантирует.
- Посмотрим, что показывает запуск с -v и -vv, и что делать, когда задача «висит».

# Что понадобится

▸ **СТЕНД ПОСЛЕ ПРАКТИК 1 И 2: Все четыре машины собраны и работают:**

```
control 10.0.2.5 — управляющая машина, здесь мы всё выполняем
web1 10.0.2.11 — Ubuntu 26.04 LTS
web2 10.0.2.12 — Ubuntu 26.04 LTS
db1 10.0.2.13 — Rocky Linux 10
```

• РАБОЧИЙ КАТАЛОГ ~/ansible-lab с файлами практики 2: ansible.cfg, inventory.ini с группами web, db, linux, debian и redhat, каталоги group_vars и host_vars.

• Окружение Python активировано, проверка связи проходит:

```
source ~/ansible-venv/bin/activate
cd ~/ansible-lab
ansible all -m ansible.builtin.ping
web1 | SUCCESS => { ... "ping": "pong" }
```

• ФАЙЛЫ ИЗ КАТАЛОГА КУРСА лежат в files/04_playbooks — на случай, если что-то не набралось руками:

– files/04_playbooks/01_ping.yml — первый плейбук (шаг 2)
– files/04_playbooks/02_web.yml — установка веб-сервера
– (шаг 4)
– files/04_playbooks/03_template.yml — копирование страницы

# ЧАСТЬ 1. БАЗОВЫЕ ШАГИ

-

# ШАГ 1. СТРУКТУРА ПЛЕЙБУКА: СПИСОК PLAY И ЕГО КЛЮЧИ

- Плейбук (playbook) — файл в формате YAML, в котором описано, что должно быть сделано на машинах.
- Слово переводят и как «сценарий», и как «пьеса», и сравнение удачное: пьеса состоит из актов, акт — из действий, действия идут в том порядке, в котором написаны.
- В Ansible акт называется play, действие — task (задача).
- Корень плейбука — СПИСОК.
- От этого зависит вся структура файла: каждый элемент начинается с дефиса, элементов может быть один или несколько, и каждый элемент — это play, то есть «набор машин плюс то, что на них сделать».
- У play четыре ключа, которых достаточно для начала:

```
name название play человеческими словами. Печатается
в выводе после слова PLAY;
hosts на каких машинах работать: группа из инвентаря,
имя машины, шаблон или их комбинация. Обязателен;
become выполнять задачи с повышением прав через sudo:
true или false;
tasks список задач. Задачи выполняются по порядку, сверху
вниз.
```

- ▸ **ПОЧЕМУ ИМЕННО ТАК: имена ключей не выдуманы — это те же понятия, что и в ad-hoc командах практики 3. Шаблон машин там задавал аргумент команды, здесь его задаёт hosts. Модуль там задавал флаг -m, здесь — имя...**
- ПОЧЕМУ КОРЕНЬ — СПИСОК, А НЕ СЛОВАРЬ.

# ШАГ 2. ПЕРВЫЙ ПЛЕЙБУК: 01_PING.YML ЦЕЛИКОМ

- Начнём с плейбука, который ничего не меняет, а только проверяет, что Ansible видит машины и может выполнить задачу.
- Это тот же ansible.builtin.ping из практики 1, но записанный в файл.
- Создайте файл 01_ping.yml в рабочем каталоге:

```
cd ~/ansible-lab
cat > 01_ping.yml <<'EOF'
# 01_ping.yml — шаг 2 практики 4: первый плейбук.
# Один play, одна задача, ничего не меняется на узлах.
- name: Проверить связь с узлами стенда
hosts: web:db
gather_facts: false
tasks:
```

- В тексте практики строки файла напечатаны с отступом в два пробела — это отступ текста, а не файла.
- В самом файле строка начинается с дефиса в первом столбце, а закрывающее слово EOF пишется без отступа: иначе heredoc не закончится, и команда будет ждать ввода.
- Этот же файл лежит в каталоге курса: files/04_playbooks/01_ping.yml.
- Разберём его построчно.
- Дефис и name открывают первый элемент списка, то есть первый play.

# ШАГ 3. ЗАПУСК И РАЗБОР PLAY RECAP

- Запускаем плейбук командой ansible-playbook:

```
ansible-playbook 01_ping.yml
PLAY [Проверить связь с узлами стенда] *********************************
TASK [Ответить pong, если узел доступен] *******************************
ok: [web1]
ok: [web2]
ok: [db1]
PLAY RECAP *************************************************************
db1 : ok=1 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

- Строка PLAY печатается один раз — это название play из ключа name.
- Строка TASK печатается перед каждой задачей, звёздочки справа добивают её до ширины экрана.
- Машины в отчёте идут в алфавитном порядке: db1, web1, web2, — а не в том порядке, в каком записаны в инвентаре.
- PLAY RECAP — последний блок вывода и самая важная его часть.
- Это сводка по каждой машине, и в ней семь колонок:

▸ **ПОЧЕМУ ЭТОГО ОТЧЁТА НЕ БЫЛО В ПРАКТИКЕ 1: Ad-hoc команда ansible отвечает построчно на каждую машину и итога не печатает. Плейбук выполняет десятки задач, и без сводки инженер не понял бы, чем закончился...**

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: PLAY RECAP смотрят первым, когда прогон запускают из системы автоматизации ночью. Если во всех строках changed=0, значит, ничего не изменилось; если failed больше нуля — надо искать, где...**

# ШАГ 4. 02_WEB.YML: УСТАНОВКА ВЕБ-СЕРВЕРА

- Первый плейбук ничего не менял.
- Добавим полезную работу: поставим веб-сервер.
- В практике 3 это делала ad-hoc команда с модулем ansible.builtin.apt; теперь то же самое станет задачей плейбука.

```
cat > 02_web.yml <<'EOF'
# 02_web.yml — шаг 4 практики 4: ставим веб-сервер.
# Имя пакета пока написано прямо в задаче: это заготовка
# под практику 7, где появятся условия для семейств ОС.
- name: Установить веб-сервер
hosts: web:db
become: true
gather_facts: false
```

- Этот файл тоже лежит в каталоге курса: files/04_playbooks/02_web.yml.
- Что появилось по сравнению с 01_ping.yml: ключ become: true на уровне play, модуль ansible.builtin.apt вместо ping и два параметра модуля — name и state.
- ПОЧЕМУ BECOME СТОИТ НА PLAY, А НЕ НА ЗАДАЧЕ.
- Ключ можно написать и у отдельной задачи, и он сделает то же самое.
- Но в реальном плейбуке повышение прав нужно почти каждой задаче: поставить пакет, записать файл в /etc, перезапустить службу.

# ШАГ 5. СОСТОЯНИЯ ВМЕСТО КОМАНД

- Разберём самое важное в модулях Ansible — параметры состояния.
- В плейбуке мы не пишем «выполни apt install», мы пишем «пакет должен быть установлен».

```
ПЕРВОЕ: apt-get install -y apache2
ВТОРОЕ: ansible.builtin.apt: name=apache2 state=present
```

- Первое — команда.
- Она выполняется всегда, когда её встретят.
- Второе — состояние.
- Модуль сначала смотрит, что на машине есть сейчас, и только потом решает, нужно ли что-то делать.
- Состояний, которые встречаются в курсе чаще всего, немного:
  - state: present — «должно быть». У пакетов это установленный
  - пакет, у файлов — существующий файл, у пользователей —
  - существующая учётная запись;
  - state: absent — «должно не быть». Удаление описывается так же,
  - как установка: модулю всё равно, есть объект или нет;
  - state: started — «служба должна работать». Если она уже
  - работает, ничего не произойдёт;

# ШАГ 6. 03_TEMPLATE.YML: СВОЯ СТРАНИЦА САЙТА

- Теперь у нас есть веб-сервер, но отвечает он страницей-заглушкой.
- Положим на место свою страницу.
- Заодно добавим задачу, которой раньше не было, — обновление кэша пакетов.

```
cat > index.html <<'EOF'
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>Стенд курса Ansible</title>
</head>
<body>
```

- Отступ в два пробела у строк HTML — снова отступ текста: в файле index.html строка <!DOCTYPE html> начинается с первого столбца.
- Тот же файл лежит в каталоге курса: files/04_playbooks/index.html.
- Оба файла лежат в каталоге курса: files/04_playbooks/03_template.yml и files/04_playbooks/index.html.
- Разберём задачу copy.
- Модуль ansible.builtin.copy берёт файл с управляющей машины, передаёт его на узел и кладёт по указанному пути:

# ШАГ 7. ИДЕМПОТЕНТНОСТЬ: ВТОРОЙ ПРОГОН ДАЁТ CHANGED=0

- Запустите 03_template.yml второй раз, не меняя ни одной строки в файле:

```
ansible-playbook --limit web 03_template.yml
TASK [Обновить кэш пакетов] ****************************************
ok: [web1]
ok: [web2]
TASK [Положить главную страницу сайта] *********************************
ok: [web1]
ok: [web2]
PLAY RECAP *************************************************************
```

- changed=0.
- Ни одна задача не изменила машины, потому что менять было нечего: кэш свежий, файл на месте, содержимое совпадает с источником, права совпадают.
- Свойство плейбука, при котором повторный прогон ничего не меняет, называется ИДЕМПОТЕНТНОСТЬЮ.
- На что здесь смотреть: колонка ok не изменилась — две задачи и в первом, и во втором прогоне.
- Модуль не «пропускает» задачу при повторном запуске: он её выполняет и сообщает, что изменений не потребовалось.

▸ **ПОЧЕМУ ИМЕННО ТАК: changed — не «задача выполнена», а «состояние машины изменилось». Разница легче всего видна на файле: модуль copy передаёт файл, сравнивает контрольную сумму с тем, что лежит на узле, и либо...**

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: идемпотентность — не красивый отчёт, а возможность применять описание повторно. Плейбук запускают после аварии, после добавления машины в парк, каждую ночь по расписанию, перед выкаткой...**

# ЧАСТЬ 2. ЧТО ОТЛИЧАЕТ РАБОТУ ОТ УЧЕБНОГО ПРИМЕРА

-

# ШАГ 8. 04_WEB_FULL.YML: СЛУЖБА И ОБРАБОТЧИК

- Четвёртый плейбук — рабочий сценарий практики: пакет, файл, состояние службы и обработчик перезапуска.

```
cat > 04_web_full.yml <<'EOF'
# 04_web_full.yml — шаги 8 и 9 практики 4: рабочий сценарий.
# Пакет, файл, состояние службы и обработчик перезапуска.
# Порядок задач — причина, по которой страница ложится после
# пакета.
- name: Веб-сервер со своей страницей
hosts: web:db
become: true
```

- Этот файл лежит в каталоге курса: files/04_playbooks/04_web_full.yml.
- HANDLERS — отдельный раздел play, на одном уровне с tasks.
- Технически это список задач, которые выполняются не по очереди, а по вызову.
- Вызывает их ключ notify у обычной задачи.
- Значение notify — не имя модуля и не имя задачи, а ИМЯ ОБРАБОТЧИКА: строка должна буква в букву совпадать со строкой name в разделе handlers.

▸ **ПОЧЕМУ ОБРАБОТЧИК, А НЕ ОБЫЧНАЯ ЗАДАЧА: Перезапускать службу после каждой задачи — простой способ уронить сайт на время. Обработчик позволяет описать перезапуск один раз и вызывать его из любой задачи, которая...**

- Ещё две особенности обработчиков:

# ШАГ 9. NAME У ЗАДАЧ: ЗАЧЕМ И ЧТО БЫВАЕТ БЕЗ НЕГО

- Ключ name у задачи не влияет на то, что задача делает.
- Его влияние видно в выводе, а точнее — в его отсутствие.
- Задача без name выглядит так:

```
TASK [ansible.builtin.copy] ********************************************
```

- Задача с name — так:

```
TASK [Положить главную страницу сайта] *********************************
```

- Разница проявляется в трёх случаях.
- Первый — разбор ошибки: когда задача падает, в логе остаётся её название.
- По названию «Установить пакет веб-сервера» сразу понятно, что не получилось.
- По «ansible.builtin.apt» понятно, каким модулем, но не понятно, каким из пяти мест плейбука.
- Второй — журнал прогона: прогоны запускают по расписанию, и через полгода в журнале видно набор названий задач.
- Третий — вывод с --list-tasks: команда печатает состав play, и задачи в нём перечислены именно по name.

```
ansible-playbook --list-tasks 04_web_full.yml
playbook: 04_web_full.yml
play #1 (web:db): Веб-сервер со своей страницей TAGS: []
tasks:
```

# ШАГ 10. ЗАПУСК С -V И -VV: ЧТО СМОТРЕТЬ, КОГДА ЗАДАЧА «ВИСИТ»

- При обычном запуске Ansible печатает по задаче одну строку на машину: ok, changed, skipping или fatal.
- Этого хватает, когда всё идёт хорошо, и не хватает, когда надо понять, почему не идёт.
- Флаг -v поднимает подробность вывода на ступень.
- Ступеней четыре, и каждая следующая включает предыдущую:

```
ansible-playbook -v 04_web_full.yml подробные результаты
ansible-playbook -vv 04_web_full.yml плюс параметры задач
ansible-playbook -vvv 04_web_full.yml плюс подключение по SSH
ansible-playbook -vvvv 04_web_full.yml плюс отладка соединений
```

- Что искать в выводе с -v: модуль и его параметры, с которыми задача запущена (частая ошибка — параметр попал не тому модулю или не попал вообще)
- значение переменной после подстановки (если вместо apache2 подставилось пустое место, переменная не определена)
- команду, которую Ansible выполнил на узле, — она видна в поле invocation.
- С -vv к результату добавляется подробный JSON задачи.
- С -vvv в лог попадает обмен с SSH, и именно там видно, почему задача «висит»: Ansible ждёт ответа от узла ещё до того, как начал работу.

▸ **ЧТО ДЕЛАТЬ, ЕСЛИ ЗАДАЧА «ВИСИТ»: Задача, которая не отвечает, почти всегда ждёт не Ansible, а узел:**

– Проверьте, отвечает ли узел вообще. Отдельное окно терминала

– и обычный ssh — самый быстрый способ:

# ШАГ 11. FQCN: ПОЧЕМУ В ПЛЕЙБУКАХ ПОЛНОЕ ИМЯ МОДУЛЯ

- Во всех четырёх плейбуках модули записаны полным именем: ansible.builtin.apt, ansible.builtin.copy, ansible.builtin.service, ansible.builtin.ping.
- Такая запись называется FQCN — fully qualified collection name, полное имя с указанием коллекции.
- Три части: ansible — пространство имён, builtin — коллекция, apt — модуль.
- Работает и короткая запись:
  - name: Установить пакет веб-сервера
  - apt:
  - name: apache2
  - state: present
- Почему в курсе принято полное имя:
  - короткое имя неоднозначно. Модулей с именем copy в разных
  - коллекциях несколько, и Ansible выбирает первый найденный
  - по своему порядку поиска. Это может быть не тот модуль,
  - что вы имели в виду, — и, что хуже, он тоже может работать;
  - полное имя самодостаточно: по строке ansible.builtin.service
  - видно, откуда модуль, и не нужно гадать, входит ли он
  - в ansible-core или приехал с коллекцией;

# ШАГ 12. ПРОВЕРКИ ПЕРЕД ЗАПУСКОМ

- Плейбук, который меняет боевые машины, проверяют до запуска.
- У ansible-playbook для этого есть пять ключей, и все пять встречаются в работе.

▸ **ПРОВЕРКА СИНТАКСИСА: Ansible разберёт файл и скажет, есть ли в нём ошибки разбора. Машины при этом не трогаются вообще:**

```
ansible-playbook --syntax-check 04_web_full.yml
playbook: 04_web_full.yml
```

- Пустой ответ после строки playbook: — это успех. --syntax-check ловит сломанные отступы, незакрытые кавычки, ключ у play, которого не бывает.
- Чего он не ловит: ни одной смысловой ошибки.
- Опечатка в имени модуля или несуществующий параметр на этой проверке не видны.

▸ **СПИСОК ЗАДАЧ И СПИСОК МАШИН: Два ключа отвечают на вопросы «что будет сделано» и «на ком»:**

```
ansible-playbook --list-tasks 04_web_full.yml
ansible-playbook --list-hosts 04_web_full.yml
playbook: 04_web_full.yml
play #1 (web:db): Веб-сервер со своей страницей TAGS: []
pattern: ['web:db']
hosts (3):
```

- --list-hosts особенно полезен с шаблонами: он показывает, кого именно выбрал hosts, до всякого подключения.

# ШАГ 13. ПРОВЕРКА ИДЕМПОТЕНТНОСТИ НА ВСЕХ ТРЁХ УЗЛАХ

- Соберём то, что должно быть к концу практики: прогон настоящий, потом проверочный.
- Между ними не меняется ни одна строка плейбука и ни один файл.
- ▸ **ВАЖНО: числа первого прогона ниже приведены для стенда, возвращённого к снимку, который снят перед этой практикой. Если практика проходится второй раз подряд, изменения будут только при сбросе состояния:...**

```
ansible-playbook 04_web_full.yml
PLAY RECAP *********************************************************************
db1 : ok=0 changed=0 unreachable=0 failed=1 skipped=0 rescued=0 ignored=0
web1 : ok=5 changed=4 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
web2 : ok=5 changed=4 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

- Откуда ok=5, если задач в файле четыре.
- Четыре задачи плюс сработавший обработчик: обработчик — такая же задача play, и его результат попадает в отчёт наравне с остальными.
- Пятая ok — это перезапуск веб-сервера.
- Сбор фактов в этом play выключен (gather_facts: false), поэтому строки Gathering Facts в отчёте нет; будь он включён, факты дали бы ещё одну ok.
- Из пяти успешных записей четыре изменили машину: обновился кэш пакетов, установился apache2, лёг файл страницы и служба была перезапущена обработчиком.
- Задача service в changed не попала: пакет apache2 сам поднял службу при установке, менять было нечего.
- Обработчик перезапустил службу один раз, в конце play, — и попал в changed именно потому, что состояние restarted всегда даёт изменение.
- Обработчик вообще виден в обеих колонках сразу: он увеличивает и ok, и changed.

# Если что-то пошло не так

▸ **ПРОБЛЕМА: ERROR! 'host' is not a valid attribute for a Play**

```
ERROR! 'host' is not a valid attribute for a Play
```

• Ключ у play назван host вместо hosts.
• Проверяйте по порядку:
  – Откройте строку, которую называет Ansible: в тексте ошибки
  – есть путь к файлу и номер строки.
  – Сверьте имя ключа с допустимыми: hosts, name, become, tasks,
  – gather_facts, vars, handlers.
  – Проверьте весь файл разом — команда ловит ошибки разбора,
  – не подключаясь к машинам:
  – ansible-playbook --syntax-check 01_ping.yml

▸ **ПРОБЛЕМА: ERROR! 'notife' is not a valid attribute for a Task**

```
ERROR! 'notife' is not a valid attribute for a Task
```

• Частый случай — опечатка в имени ключа: notife вместо notify.
• Ansible не знает такого атрибута и останавливает разбор плейбука до выполнения задач.
• В тексте ошибки стоит ровно то имя ключа, которое написано в задаче: набрали notife — про notife и сообщит.

# Проверка результата

- Практика выполнена, если выполняются все пункты.
- 1.
- В рабочем каталоге лежат четыре плейбука и файл страницы:

```
cd ~/ansible-lab
ls -1 01_ping.yml 02_web.yml 03_template.yml 04_web_full.yml index.html
01_ping.yml
02_web.yml
03_template.yml
04_web_full.yml
index.html
```

- 2.
- Синтаксис всех четырёх файлов разбирается:

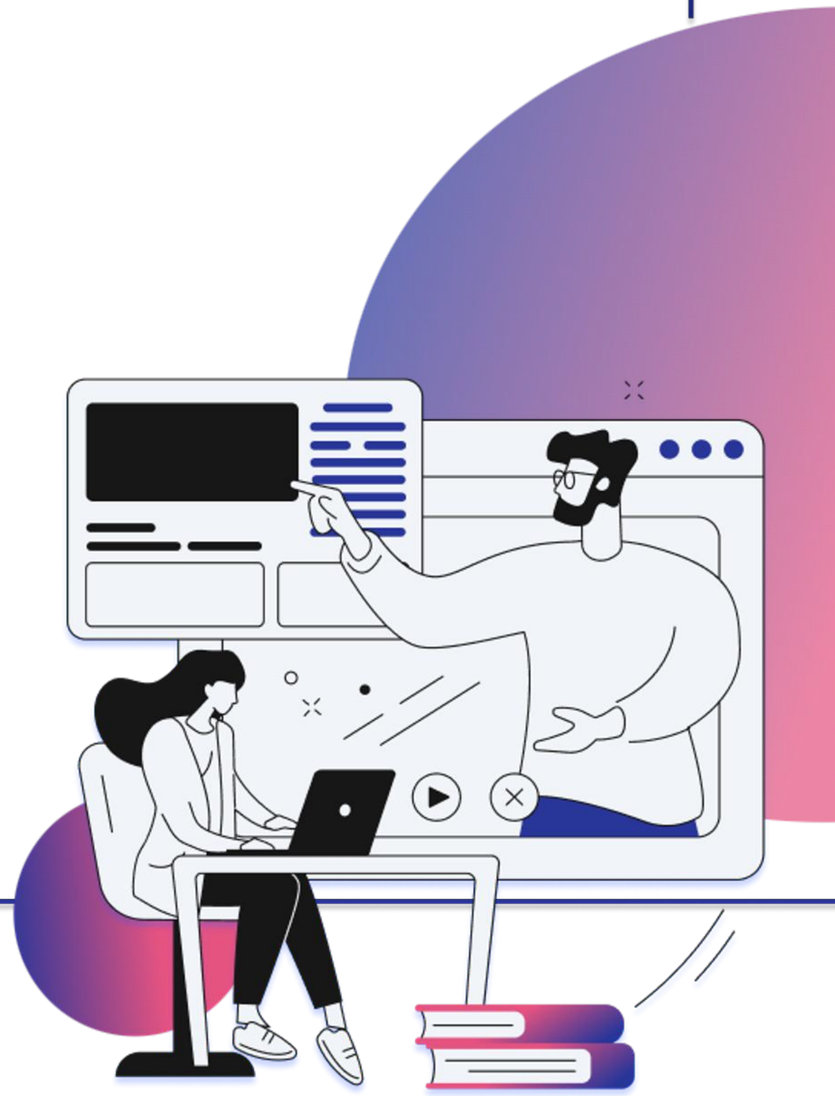
```
ansible-playbook --syntax-check 01_ping.yml
```

- 3.
- Список машин совпадает с задуманным:
- 4.

# Что дальше

- Далее в курсе:
  - практика 5 — переменные и секреты: имя пакета и путь к странице
  - перестанут быть зашитыми в задачу и станут переменными,
  - а приоритет переменных объяснит, почему значение из group_vars
  - иногда не побеждает;
  - практика 6 — факты и шаблоны Jinja2: страница index.html
  - превратится в шаблон index.j2, в котором будут имя хоста,
  - адрес и семейство ОС, — а сбор фактов, выключенный в этой
  - практике, снова станет нужен;
  - практика 7 — условия, циклы, блоки: та же установка веб-сервера
  - заработает и на Ubuntu, и на Rocky Linux, а строка db1
  - исчезнет из отчёта с ошибкой;
  - практика 8 — файлы и права: каталоги сайта, права, режимы
  - и validate для конфигураций;
  - практика 9 — административные задачи: пользователи, sudo,
  - пакеты, службы и firewall на всём парке;

Учебный центр АйТи Клауд



# Практика 5. Переменные и секреты

**ЭТА ПРАКТИКА О ТОМ, ОТКУДА ANSIBLE БЕРЁТ ЗНАЧЕНИЯ: переменные в плейбуке, в отдельных файлах, в инвентаре и в командной строке, порядок их приоритета, register и set_fact, а секрет**

современные технологии простым языком

# Обзор практики

- ЭТА ПРАКТИКА О ТОМ, ОТКУДА ANSIBLE БЕРЁТ ЗНАЧЕНИЯ: переменные в плейбуке, в отдельных файлах, в инвентаре и в командной строке, порядок их приоритета, register и set_fact, а секреты — в Ansible Vault.
- Опирается на практики 2 и 4.

## Что мы сделаем

- Разберёмся, где вообще могут быть заданы переменные и почему одно имя встречается в двух-трёх файлах сразу.
- Соберём порядок приоритета от слабого места к сильному и поставим web_port в пять мест: group_vars/all.yml, host_vars/web1.yml, vars плейбука, vars/app.yml и --extra-vars.
- Разберём именование: почему web_port, а не port.
- Научимся забирать результат задачи в переменную (register) и считать переменную во время прогона (set_fact).
- Вынесем переменные в vars_files, научимся менять поведение плейбука через --extra-vars и спрячем пароль в Ansible Vault.
- Честно разберём, что Vault защищает, а что нет.

# Что понадобится

▸ **СТЕНД ИЗ ПРАКТИК 1 И 2: рабочий каталог ~/ansible-lab на машине control, в нём ansible.cfg, inventory.ini, group_vars, host_vars и плейбуки практики 4.**

```
control 10.0.2.5 — управляющая машина, здесь мы всё выполняем
web1 10.0.2.11 — Ubuntu 26.04 LTS
web2 10.0.2.12 — Ubuntu 26.04 LTS
db1 10.0.2.13 — Rocky Linux 10
```

- Ansible установлен в ~/ansible-venv.
- Перед началом сделайте снимок машин: практика правит групповые переменные.
- ФАЙЛЫ ИЗ КАТАЛОГА КУРСА лежат в files/05_variables_vault:
  - files/05_variables_vault/01_vars_play.yml — шаг 2
  - files/05_variables_vault/02_register.yml — шаги 6-7
  - files/05_variables_vault/03_vars_files.yml — шаг 11
  - files/05_variables_vault/vars/app.yml — шаг 11
  - files/05_variables_vault/extra_vars.yml — шаг 12
  - files/05_variables_vault/group_vars/all/vault.yml.example — шаг 9
  - files/05_variables_vault/README.md — что за файлы

▸ **ОБРАЗЕЦ, А НЕ СЕКРЕТ: зашифрованного файла в каталоге курса нет — Ansible на машине автора не установлен, а шифровать файл чужим паролем бессмысленно, прочитать его вы не сможете. В курсе лежит...**

# ЧАСТЬ 1. ПЕРЕМЕННЫЕ: ГДЕ ОНИ ЖИВУТ И КТО СИЛЬНЕЕ

-

# ШАГ 1. МЕСТА, В КОТОРЫХ ВООБЩЕ БЫВАЮТ ПЕРЕМЕННЫЕ

- В практике 2 переменные уже появились — group_vars и host_vars.
- Это не единственное место.
- Перечислим места, с которыми будем работать, от самого далёкого от задачи к самому близкому:
  - defaults роли — значения по умолчанию для роли (практика 10);
  - переменные инвентаря — например web_port=80 в строке машины;
  - group_vars/all.yml — переменные для всех машин парка;
  - group_vars/debian.yml, group_vars/redhat.yml — для группы;
  - host_vars/web1.yml — для одной машины;
  - vars в плейбуке — для всех машин этого play;
  - vars_files — файлы, перечисленные в play;
  - факты — то, что Ansible узнал о машине сам (практика 6);
  - set_fact и register — то, что посчитано во время прогона;
  - --extra-vars — то, что передано в командной строке.
- Ansible собирает все слои в один набор переменных для каждой задачи: если имя встречается в двух местах, победит более сильное.
- Это не ошибка, а обычная работа.

▸ **ПОЧЕМУ ИМЕННО ТАК: переменные не «объявляются» один раз, как в языке программирования: они вычисляются тогда, когда нужны задаче, и разные задачи одного плейбука могут видеть одно имя по-разному. Отсюда...**

# ШАГ 2. VARS: ПЕРЕМЕННЫЕ В ПЛЕЙБУКЕ

- Начнём с переменных прямо в play.
- Заодно подготовим опыт с приоритетом: одну переменную web_port мы положим в несколько мест.

```
cd ~/ansible-lab
cat >> group_vars/all.yml <<'EOF'
web_port: 80
EOF
cat >> host_vars/web1.yml <<'EOF'
web_port: 8080
EOF
```

- В files/02_inventory этих строк нет: там файлы практики 2, а мы их сейчас дополняем.
- Порт 80 — рабочий порт веб-сервера в курсе.
- Числа 8080, 8081, 8082 и 8083 — учебные метки: ни один плейбук курса на этих портах не слушает, они нужны только для опыта.

```
cat > 01_vars_play.yml <<'EOF'
```

- ▸ **ВАЖНО: содержимое файла напечатано с отступом в два пробела — это отступ текста практики, а не файла. В самом файле строки начинаются с первого столбца, и закрывающее слово EOF тоже: иначе heredoc не...**
- У всех трёх машин web_port равен 8081, хотя в group_vars лежит 80, а в host_vars/web1.yml — 8080: переменные play оказались сильнее переменных инвентаря. inventory_hostname — имя машины из инвентаря, одна из...
- фильтр default подставляет значение, если переменной нет.

# ШАГ 3. ПОРЯДОК ПРИОРИТЕТА: ОТ СЛАБОГО К СИЛЬНОМУ

- Победит то место, что стоит в списке ниже; список идёт от самого слабого к самому сильному:
  - defaults роли; 2. переменные инвентаря и group_vars/all.yml;
  - group_vars/<группа>.yml; 4. host_vars/<машина>.yml;
  - факты о машине и кэшированные set_fact (практика 6);
  - vars в play; 7. vars_prompt; 8. vars_files;
  - vars роли (role vars); 10. vars на блоке задач (block vars);
  - vars на одной задаче (task vars); 12. include_vars;
  - set_fact и register — посчитанное во время прогона;
  - параметры роли и параметры include;
  - --extra-vars — сильнее не бывает.
- Сокращённая цепочка, которую достаточно помнить:

```
defaults роли → group_vars → host_vars → vars плейбука →
vars_files → set_fact/register → --extra-vars
```

- **ПОЧЕМУ ИМЕННО ТАК: полный порядок шире сокращённого — в нём есть role vars, include_vars, block vars и task vars. Заучивать все пятнадцать строк не нужно: нужно правило — чем ближе переменная к задаче и чем...**

# ШАГ 4. ЖИВОЙ ОПЫТ: ОДНА ПЕРЕМЕННАЯ В ЧЕТЫРЁХ МЕСТАХ

- Здесь участвуют четыре места из пяти: групповые переменные, переменные хоста, vars плейбука и командная строка.
- Пятое место — vars_files — появится в шаге 11.
- Закомментируйте в 01_vars_play.yml строки web_port: 8081 и site_root вместе с заголовком vars — и запустите снова:

```
ansible-playbook 01_vars_play.yml
ok: [web1] => { "msg": [ "машина: web1", "web_port: 8080", ... ] }
ok: [web2] => { "msg": [ "машина: web2", "web_port: 80", ... ] }
ok: [db1] => { "msg": [ "машина: db1", "web_port: 80", ... ] }
```

- На web1 победило host_vars (8080), на web2 и db1 — group_vars (80).
- Вернём vars в плейбук и добавим к запуску внешнюю переменную:

```
ansible-playbook 01_vars_play.yml -e web_port=8082
...на всех трёх машинах web_port равен 8082 — победил --extra-vars.
```

- ▸ **ПОМЕНЯЙТЕ МЕСТАМИ: Перенесите web_port из host_vars/web1.yml в group_vars/all.yml: в host_vars строку уберите, а в group_vars поменяйте 80 на 8080. Без -e и без vars в play на всех трёх машинах будет 8080. Для...**
- ▸ **ПРОВЕРКА: спросите у Ansible, что он думает о переменных инвентаря:**

```
ansible-inventory --host web1 | grep web_port
"web_port": 8080,
```

- ▸ **ВАЖНО: ansible-inventory показывает только переменные инвентаря. Переменных из vars плейбука и из --extra-vars там не будет никогда: эти слои появляются позже, когда плейбук уже запущен. Проверять их можно...**

# ШАГ 5. ИМЕНОВАНИЕ ПЕРЕМЕННЫХ

– строчные буквы, слова разделяются подчёркиванием: web_port;
– первая буква — не цифра и не подчёркивание: port_80, а не 80_port;
– пробелов и дефисов нет: web-port в YAML разберётся как вычитание;
– префикс по назначению: web_, db_, nginx_, а в роли — по имени роли;
– никаких имён из одного слова без контекста: не port, а web_port.

- **ПОЧЕМУ ИМЕННО ТАК: port в роли веб-сервера и port в роли базы данных — это два разных port, и Ansible сведёт их в одно имя: победит то, что задано позже. web_port и db_port не столкнутся никогда.**
- **ЧЕГО НЕ ДЕЛАТЬ: не занимайте встроенные имена Ansible — ansible_user, ansible_host, hostvars, groups, item, loop. Переменная ansible_user, заданная «для красоты» ещё раз в другом файле, ломает подключение, а...**

# ШАГ 6. REGISTER: РЕЗУЛЬТАТ ЗАДАЧИ КАК ДАННЫЕ

- Модуль возвращает не только «готово», но и данные: код возврата, вывод команды, факт изменения.
- Ключ register кладёт весь ответ в переменную этого хоста.

```
cat > 02_register.yml <<'EOF'
---
# Результат задачи как данные: register сохраняет ответ модуля
# в переменную этого хоста, set_fact считает новое значение прямо
# во время прогона.
- name: Собрать сведения об узле и посчитать итог
hosts: linux
gather_facts: false
```

- Что лежит в результате: changed — изменила ли задача что-нибудь на машине; rc — код возврата команды, 0 при успехе; stdout — весь вывод одной строкой
- stdout_lines — тот же вывод списком строк; stderr — поток ошибок; failed и skipped — как задача закончилась; results — список ответов, если задача выполнялась в loop (практика 7).

▸ **ПОЧЕМУ ИМЕННО ТАК: register — способ принять решение по факту, а не по предположению. «Поставить пакет, если его нет», «перезапустить службу, если конфигурация изменилась» — всё это строится на данных из...**

- changed_when: false в задачах выше — не украшение.
- Команда date ничего на машине не меняет, но модуль command по умолчанию считает изменением каждый свой запуск, и в отчёте появляется changed.

▸ **ПРОВЕРКА: в выводе найдите строку с отдельными полями и убедитесь, что stdout содержит дату, а rc равен нулю:**

# ШАГ 7. SET_FACT: ПЕРЕМЕННАЯ, ПОСЧИТАННАЯ ВО ВРЕМЯ ПРОГОНА

- Последние две задачи 02_register.yml считают новую переменную:
  - name: Посчитать переменную во время прогона
  - ansible.builtin.set_fact:
  - report_line: "{{ inventory_hostname }}: проверено {{ date_result.stdout }}"
- Вывод:

```
ok: [web1] => { "report_line": "web1: проверено 2026-03-16" }
```

- set_fact создаёт переменную-факт для конкретного хоста и только на время этого прогона.
- Она сильнее vars плейбука, vars_files и include_vars, но слабее --extra-vars.
- В файлы значение не пишется: следующий прогон посчитает его заново.
- Поэтому «сохранить результат прошлого запуска» через set_fact нельзя, и это правильно: устаревшее значение хуже отсутствующего.

▸ **ВАЖНО: set_fact, посчитанный на web1, не виден web2. Общий итог по парку собирают иначе — через delegate_to и run_once (практика 14) или через hostvars, когда нужны факты всех машин (практика 6).**

▸ **ЧЕГО НЕ ДЕЛАТЬ: не считайте переменную через саму себя — ни в set_fact, ни в vars: Ansible останавливается с ошибкой о рекурсии.**

# ЧАСТЬ 2. СЕКРЕТЫ: VAULT, ФАЙЛЫ И ЧЕСТНОСТЬ ПРО БЕЗОПАСНОСТЬ

-

# ШАГ 8. VAULT: ФАЙЛ, КОТОРЫЙ НЕ ЧИТАЕТСЯ ГЛАЗАМИ

- Пока переменные лежат открытым текстом, всё просто.
- Пароль от базы данных так хранить нельзя: он попадёт в репозиторий, в вывод git log и на чужой ноутбук.
- Vault шифрует файл целиком, а Ansible расшифровывает его на лету, когда читает переменные.

```
cd ~/ansible-lab
ansible-vault create secrets_demo.yml
```

- Команда спросит пароль для нового файла (дважды) и откроет редактор.
- Наберите одну строку — demo_password: MyStrongPassword (значение учебное) — и сохраните.
- Что получилось:

```
head -1 secrets_demo.yml
$ANSIBLE_VAULT;1.1;AES256
cat secrets_demo.yml
$ANSIBLE_VAULT;1.1;AES256
623133653966623430613934643361633837643737646136336536343062313864336264366233 61
6134333665353966363534333632666535333761666131626135373137616531373665666338626 4
```

- Первая строка — заголовок формата, дальше шестнадцатеричный текст: ни имён переменных, ни значений не видно.
- Читают файл командами view и edit:

# ШАГ 9. СЕКРЕТ В GROUP_VARS/ALL/VAULT.YML

- Теперь настоящий секрет — пароль, под которым приложение ходит в базу данных.
- Схема такая: сам пароль лежит в зашифрованном файле, а обычные файлы ссылаются на переменную из него.
- В файле group_vars/all/vault.yml живёт vault_db_password, а в обычном файле переменных — строка db_password: "{{ vault_db_password }}".
- Вот содержимое образца из каталога курса (files/05_variables_vault/group_vars/all/vault.yml.example):

```
---
# ОБРАЗЕЦ файла секретов. Это не зашифрованный файл: в материалы курса
# он положен открытым текстом, потому что шифрование проверяется на
# стенде, а не в репозитории. Шифровать его нечем: Ansible на машине
# автора не установлен, а класть в курс непроверенный шифротекст
# нельзя — студент не сможет его прочитать своим паролем.
#
# Как из образца сделать рабочий файл:
```

- Скопируйте файл под именем group_vars/all/vault.yml — без расширения .example — и перенесите в каталог прежний файл all.yml:
- ПОЧЕМУ ПЕРЕНЕСТИ ALL.YML: Ansible ищет переменные группы так — сначала каталог с именем группы (group_vars/all) и, найдя его, на этом останавливается.
- Файл group_vars/all.yml рядом с каталогом того же имени после этого не читается: молча, без предупреждения.
- Вместе с ним пропали бы ansible_user, ansible_ssh_private_key_file и lab_owner.

# ШАГ 10. ПАРОЛЬ VAULT: ЧТОБЫ НЕ ВВОДИТЬ ЕГО КАЖДЫЙ РАЗ

- Пароль можно передать тремя способами: --ask-vault-pass — спросить при запуске; --vault-password-file ~/.vault_pass — прочитать из файла; строка vault_password_file в ansible.cfg — не указывать ничего.

```
ansible-playbook 02_register.yml --ask-vault-pass
Vault password:
```

- Для работы удобнее файл, и держать его нужно вне каталога проекта:

```
echo 'MyStrongPassword' > ~/.vault_pass
chmod 600 ~/.vault_pass
ansible-vault view group_vars/all/vault.yml \
--vault-password-file ~/.vault_pass
```

- Третий способ самый удобный.
- Строка в ansible.cfg, в секции [defaults], рядом с настройками из практики 2:

```
vault_password_file = /home/engineer/.vault_pass
```

- После этого пароль не указывается вообще — ни в ansible-vault, ни в ansible-playbook, ни в ansible-inventory:

```
ansible-vault view group_vars/all/vault.yml
```

▸ **ПРОВЕРКА: спросите у Ansible, откуда взято значение:**

- В скобках — файл, из которого настройка прочитана.
- Если там (default), значение из ansible.cfg не взято: проверьте имя секции.

# ШАГ 11. VARS_FILES: ПЕРЕМЕННЫЕ ОТДЕЛЬНЫМ ФАЙЛОМ

- Если переменных много, держать их в play неудобно.
- Директива vars_files подключает файл целиком и позволяет заменить один набор переменных другим — например, для другого окружения (практика 14).

```
mkdir -p ~/ansible-lab/vars
cat > vars/app.yml <<'EOF'
---
# Переменные приложения. Файл подключается плейбуком 03_vars_files.yml
# директивой vars_files и лежит рядом с ним, в каталоге vars.
app_name: ansible-lab
app_version: "1.0"
app_root: /var/www/html
```

- ▸ **ЗАЧЕМ ЭТО В РАБОТЕ: задача «Убедиться, что секрет дошёл до узла» впервые знакомит с модулем ansible.builtin.assert: он проверяет, что все условия из that истинны, и если хотя бы одно ложно, задача падает с...**
- Путь записан как vars/app.yml, относительно плейбука, — это ровно тот путь, куда мы положили файл.
- Каталог /var/www/html уже есть после практики 4.
- В play тоже есть web_port: 8081, но в vars/app.yml лежит 8083.
- Запускаем (пароль Vault берётся из ansible.cfg):
- Порт 8083 из файла победил порт 8081 из vars плейбука, хотя строки vars стоят в файле выше.

# ШАГ 12. --EXTRA-VARS И ФАЙЛ @EXTRA_VARS.YML

- Последнее по силе место — командная строка.
- Две формы записи: -e web_port=8082 задаёт одну переменную, -e @extra_vars.yml читает целый файл:

```
cat > extra_vars.yml <<'EOF'
---
# Внешние переменные для запуска:
# ansible-playbook 03_vars_files.yml -e @extra_vars.yml
#
# Значения из этого файла сильнее всего, что написано в плейбуке,
# в vars_files и в инвентаре. Числа 8082 — учебные метки порта:
# рабочий порт веб-сервера в курсе 80.
```

- Все три значения из файла победили: и порт из vars_files, и версию из vars/app.yml, и lab_owner из host_vars.

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: --extra-vars — ручка для того, кто запускает плейбук, а не для того, кто его пишет. Типичные примеры: номер версии для установки, имя окружения, адрес стенда. В системе сборки тот же приём...**

▸ **ВАЖНО: --extra-vars сильнее всего, включая set_fact. Если переменная не меняется, хотя вы правите файлы, посмотрите, не передана ли она в командной строке — в этом запуске или в скрипте-обёртке.**

▸ **ЧЕГО НЕ ДЕЛАТЬ: не передавайте через -e пароли. Значение из командной строки попадает в историю оболочки, в список процессов и в журнал системы сборки — туда, куда секрет попадать не должен.**

# ШАГ 13. ПРОВЕРКА: ПАРОЛЬ ИЗ VAULT ДОХОДИТ ДО УЗЛА

- Соберём всё вместе.
- Сначала посмотрим секрет так, как его видит человек:

```
ansible-vault view group_vars/all/vault.yml
vault_db_password: "MyStrongPassword"
vault_api_token: "ЗАМЕНИТЕ_ЭТОТ_КЛЮЧ"
vault_engineer_password: "Console47"
```

- Пароль Vault при этом не спрашивают: он берётся из ansible.cfg.
- Теперь прогон плейбука, который этим секретом пользуется:

```
ansible-playbook 03_vars_files.yml
TASK [Убедиться, что секрет дошёл до узла]
ok: [web1] => {
"msg": "пароль из Vault получен"
```

- Здесь пароль виден на узле открытым текстом, и это нормально: приложению нужен пароль, а не шифротекст.
- Защищают такой файл права и владелец: 0644 читают все, у кого уже есть доступ на машину, поэтому для настоящих секретов берут 0640 и владельца-службу.
- Важно другое: на узел уходит только нужное значение, а не весь зашифрованный файл.
- **ПРОВЕРКА: второй прогон подряд не должен менять ничего:**

# ШАГ 14. ЧЕСТНОСТЬ ПРО БЕЗОПАСНОСТЬ: ЧТО VAULT ЗАЩИЩАЕТ

- Первое и главное: Vault — это шифрование файла, а не «безопасность».
- Он защищает ровно одно — содержимое файла в покое: того самого, который лежит в репозитории, на диске и в резервной копии.
- Чего Vault не защищает:
  - управляющую машину: пароль Vault лежит на ней, расшифрованные
  - значения — в памяти процесса Ansible. Кто получил доступ
  - к control под вашим пользователем, тот получил и секреты;
  - вывод задач: значение, попавшее в параметр задачи, видно в выводе
  - ansible-playbook с -vvv и в журнале, если журнал пишется;
  - файл с паролем Vault: зашифрованный файл и .vault_pass рядом —
  - это открытый текст с лишним шагом;
  - файл, который задача положила на узел: дальше его защищают
  - только права и владелец;
  - историю: пароль, один раз попавший в коммит, остался там
  - навсегда, и менять нужно сам пароль, а не строку в файле.
- Отдельно про пересылку на узел: значение расшифровывается на управляющей машине и уходит на узел внутри задачи.
- Эту пересылку защищает SSH, а не Vault.

## Если что-то пошло не так

▸ **ПРОБЛЕМА: ERROR! Attempting to decrypt but no vault secrets found**

```
ERROR! Attempting to decrypt but no vault secrets found
```

- Ansible нашёл зашифрованный файл и не получил ни одного пароля.
- Проверяйте: зашифрован ли файл (head -1 group_vars/all/vault.yml покажет $ANSIBLE_VAULT; 1.1; AES256)
- передан ли пароль хотя бы одним способом — --ask-vault-pass, --vault-password-file или vault_password_file в ansible.cfg; применилась ли настройка:

```
ansible-config dump --only-changed | grep -i vault
DEFAULT_VAULT_PASSWORD_FILE(/home/engineer/ansible-lab/ansible.cfg) = /home/engineer/.vault_pass
```

- и на месте ли сам файл пароля (ls -l ~/.vault_pass).
- Ошибка выходит и у ansible-inventory, и у ansible-vault view: пока в проекте есть зашифрованный файл, пароль нужен любой команде, которая читает переменные.

▸ **ПРОБЛЕМА: ERROR! Decryption failed (no vault secrets were found that could decrypt)**

```
ERROR! Decryption failed (no vault secrets were found that could decrypt)
```

- Пароль передан, но не подошёл: файл зашифрован другим паролем или другим vault-id.
- Ansible не различает «пароль неверный» и «пароля нет», и это сделано намеренно: сообщение не должно подсказывать, угадали вы пароль или нет.
- Проверяйте: расшифровывается ли этот файл отдельно (ansible-vault view group_vars/all/vault.yml --ask-vault-pass)
- нет ли кавычек в файле пароля — перенос строки и пробелы по краям Ansible убирает сам, а кавычки нет
- не переданы ли сразу два разных пароля — файл и --ask-vault-pass проверяются оба.

# Проверка результата

- 1.
- Файлы лежат там, где нужно:

```
cd ~/ansible-lab
ls -1 01_vars_play.yml 02_register.yml 03_vars_files.yml extra_vars.yml
ls -1 vars/ group_vars/all/
В group_vars/all должны быть vars.yml и vault.yml, а файла
group_vars/all.yml рядом быть не должно.
```

- 2.
- Файл секретов действительно зашифрован:

```
head -1 group_vars/all/vault.yml
$ANSIBLE_VAULT;1.1;AES256
```

- 3.
- Приоритет проверен своими глазами:

```
ansible-playbook 01_vars_play.yml -e web_port=8082
```

- 4. register и set_fact работают:
- 5.

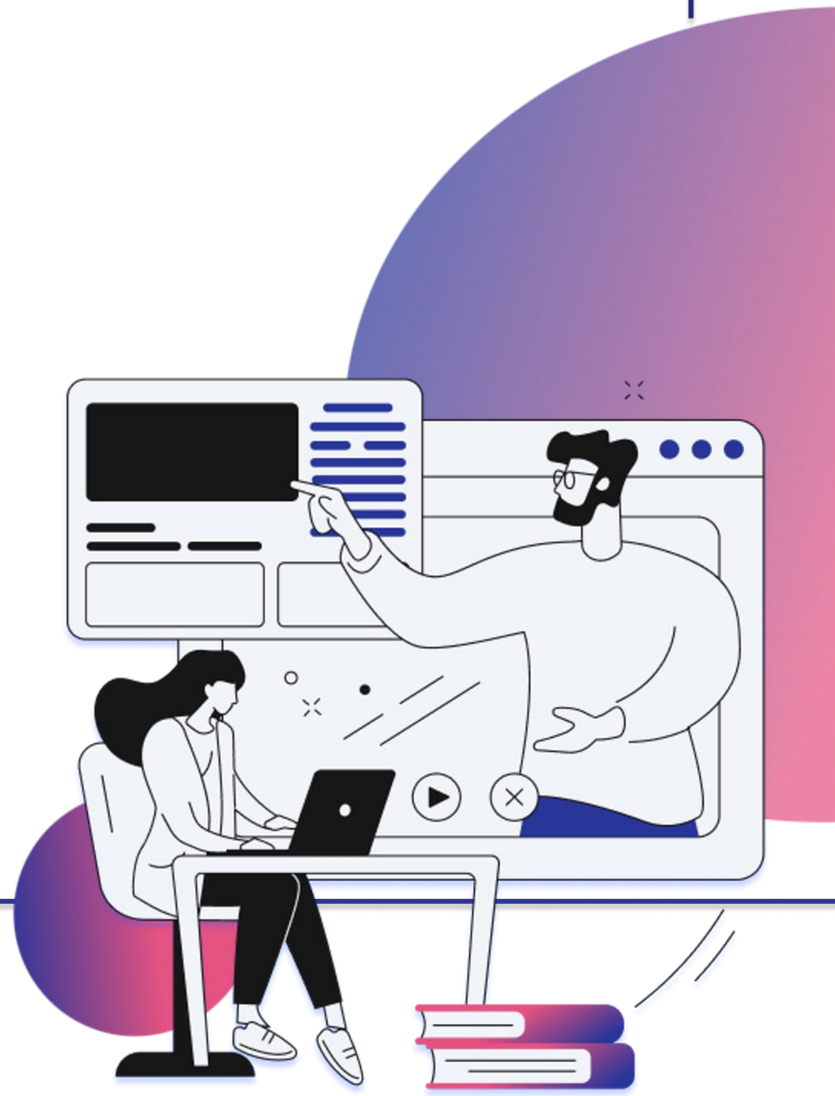
# Что дальше

- Далее в курсе:
  - практика 6 — факты и шаблоны Jinja2: переменные, которые Ansible
  - узнаёт о машине сам, и шаблон, в котором видно и факты, и наши
  - переменные;
  - практика 7 — условия и циклы: решения по данным из register
  - и различия семейств ОС, описанные переменными;
  - практика 9 — административные задачи: пароли пользователей
  - хэшем, а не открытым текстом, и Vault для остальных секретов;
  - практика 10 — роли: defaults и vars роли, то есть два слоя
  - из таблицы приоритета;
  - практика 14 — сложные проекты: отдельный vault.yml на окружение
  - и один плейбук с разными vars_files.
- ПЕРЕД ПЕРЕХОДОМ уберите за собой:
  - учебные метки порта: в рабочем каталоге верните рабочий порт 80,
  - а значения 8080-8083 из переменных проекта удалите. В каталоге
  - files/05_variables_vault они остаются как образец;

Учебный центр АйТи Клауд

# Практика 6. Факты и шаблоны Jinja2

**ЭТА ПРАКТИКА О ТОМ, ОТКУДА ПЛЕЙБУК ЗНАЕТ ПРО МАШИНУ.**



современные технологии простым языком

# Обзор практики

- **ЭТА ПРАКТИКА О ТОМ, ОТКУДА ПЛЕЙБУК ЗНАЕТ ПРО МАШИНУ: Ansible сам собирает о каждом узле сотни значений — это факты. Научимся брать из них нужные, добавлять свои пометки и собирать из этих данных файлы...**

## Что мы сделаем

- Посмотрим, что Ansible знает о машине, и научимся брать только нужные факты: полный сбор на большом парке стоит времени.
- Проверим предположения о хосте модулями ansible.builtin.debug и ansible.builtin.assert.
- Разложим настраиваемые факты в /etc/ansible/facts.d — INI и JSON — и прочитаем их через ansible_local: так машина получает пометку о роли, владельце и версии стенда.
- Напишем два шаблона: index.j2 — страница сайта с фактами и владельцем машины, и motd.j2 — сводка о парке циклом по groups['linux'].
- Разберём волшебные переменные (hostvars, groups, group_names, inventory_dir, ansible_check_mode), фильтры и тесты Jinja2 и познакомимся с validate — проверкой файла до того, как он попал на место.
- В конце посчитаем, во что обходится полный сбор фактов.

# Что понадобится

▸ **СТЕНД ИЗ ПРАКТИК 1-5: четыре машины собраны, вход по ключу настроен.**

```
control 10.0.2.5 — управляющая машина, здесь мы всё выполняем
web1 10.0.2.11 — Ubuntu 26.04 LTS
web2 10.0.2.12 — Ubuntu 26.04 LTS
db1 10.0.2.13 — Rocky Linux 10
```

- В ~/ansible-lab лежат inventory.ini, ansible.cfg, group_vars и host_vars из практик 2 и 5.
- На web1 и web2 из практики 4 работает apache2, на db1 веб-сервера нет: страницу развернём с ключом --limit web.
- Ansible: ansible-core 2.19 (пакет Ansible community 12.x) в ~/ansible-venv.
- ФАЙЛЫ ИЗ КАТАЛОГА КУРСА лежат в files/06_facts_templates:
  - files/06_facts_templates/01_facts.yml — факты и assert (шаги 1-3)
  - files/06_facts_templates/02_custom_facts.yml — настраиваемые факты (шаг 4)
  - files/06_facts_templates/facts.d/site.fact — факт в формате INI (шаг 4)
  - files/06_facts_templates/facts.d/hardware.fact — факт в формате JSON (шаг 4)
  - files/06_facts_templates/03_index_template.yml — страница сайта (шаг 8)
  - files/06_facts_templates/templates/index.j2 — шаблон страницы (шаг 8)
  - files/06_facts_templates/04_motd.yml — /etc/motd (шаг 9)

# ЧАСТЬ 1. ФАКТЫ: ЧТО ANSIBLE ЗНАЕТ О МАШИНЕ

-

# ШАГ 1. ОТКУДА БЕРУТСЯ ФАКТЫ

- ФАКТЫ (facts) — переменные, которые Ansible собирает о машине сам, подключаясь к ней по SSH.
- В начале каждого play он выполняет задачу Gathering Facts:

```
TASK [Gathering Facts]
ok: [web1]
ok: [web2]
ok: [db1]
```

- Тот же сбор запускается вручную модулем ansible.builtin.setup.
- Вывод большой — около 1300 строк на машину:

```
ansible all -m ansible.builtin.setup
web1 | SUCCESS => {
"ansible_facts": {
"ansible_default_ipv4": {
```

- ВСЕ ФАКТЫ ЛЕЖАТ В ОДНОМ СЛОВАРЕ ansible_facts, и обращаться к ним можно двумя способами: длинным ansible_facts['os_family'] и коротким ansible_os_family — второе имя Ansible подставляет сам, беря значение из...
- Работают обе формы, выбор между ними — вопрос стиля: короткую пишем там, где она читается, а форму через словарь — там, где имя факта собирается из частей.
- ФАКТЫ, НА КОТОРЫЕ ОПИРАЕТСЯ КУРС:
  – ansible_facts['os_family'] — Debian или RedHat: по этому

# ШАГ 2. ТОЛЬКО НУЖНЫЕ ФАКТЫ: FILTER И GATHER_SUBSET

- У ansible.builtin.setup два разных инструмента, и их постоянно путают:
  - filter — что ПОКАЗАТЬ из уже собранного;
  - gather_subset — что СОБИРАТЬ вообще.
- FILTER отбирает факты по имени; звёздочка заменяет любую часть строки:

```
ansible all -m ansible.builtin.setup -a "filter=ansible_os_family*"
web1 | SUCCESS => {
"ansible_facts": {
"ansible_os_family": "Debian"
}
}
db1 | SUCCESS => { ... "ansible_os_family": "RedHat" }
```

- Если фильтр совпал с фактом-словарём, возвращается словарь целиком:

```
ansible web1 -m ansible.builtin.setup -a "filter=ansible_default_ipv4"
```

- ▸ **ВАЖНО: filter не ускоряет работу. Модуль всё равно собирает весь набор фактов и лишь отбрасывает лишнее при печати: экономится ваше время на чтение вывода, но не время прогона.**
- GATHER_SUBSET задаёт, какие НАБОРЫ собирать: distribution, network, hardware, virtual, local, processor, mounts и другие.
- Восклицательный знак означает «не собирать»:

# ШАГ 3. DEBUG И ASSERT: ПРОВЕРКА ПРЕДПОЛОЖЕНИЙ О ХОСТЕ

- ansible.builtin.debug печатает то, что вы ему дали: msg — строку, var — значение переменной со всеми типами. ansible.builtin.assert проверяет условие и останавливает задачу, если условие ложно.
- Запишем оба модуля в плейбук 01_facts.yml:

```
---
# Практика 6, шаги 1-3: выбранные факты и проверка предположений о машине.
# Запуск: ansible-playbook 01_facts.yml
- name: Показать выбранные факты стенда
hosts: linux
gather_facts: true
tasks:
- name: Показать сводку по машине
```

- Запускаем:
- changed=0: печать и проверка ничего не меняют.
- Шесть задач ok — это два сбора фактов и четыре задачи в двух play.
- Запись msg: >- — свёрнутый блок YAML: несколько строк превращаются в одну строку текста.

▸ **ПОЧЕМУ ИМЕННО ТАК: assert, а не проверка глазами по выводу debug. На парке из трёхсот машин вывод построчно никто не читает: проверка должна быть в плейбуке, и тогда неподходящая машина останавливает прогон на...**

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: assert записывает требование к машине в исполняемом виде. «Не меньше гигабайта памяти», «семейство ОС из списка поддерживаемых» — это дешевле проверить до первой задачи, чем разбирать отказ...**

# ШАГ 4. НАСТРАИВАЕМЫЕ ФАКТЫ: ПОМЕТКА НА МАШИНЕ

- В фактах нет того, что знает только человек: чей это сервер, какая у него роль, какой версии на нём стенд.
- Для этого есть НАСТРАИВАЕМЫЕ ФАКТЫ (local facts).
- Файл с расширением .fact кладётся на узел в каталог /etc/ansible/facts.d, и модуль setup читает его вместе с остальными фактами: этот путь — значение по умолчанию, оно же параметр fact_path.
- Формат файла — INI или JSON.
- site.fact, формат INI:

```
# Настраиваемый факт стенда курса. Формат INI.
# Лежит на узле в /etc/ansible/facts.d/ и читается модулем setup.
# Доступ к значениям: ansible_local.site.general.<ключ>
# (имя файла, затем имя секции, затем имя ключа).
[general]
owner = engineer
role = webserver
version = 1.0.0
```

- hardware.fact, формат JSON:

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: имя файла становится именем факта, имя секции — вложенным ключом, имя ключа — значением. Машина получает пометку так, как удобно организации: ansible_local.site.general даёт владельца, роль...**

- Раскладываем файлы плейбуком 02_custom_facts.yml:

# ШАГ 5. ВОЛШЕБНЫЕ ПЕРЕМЕННЫЕ

- Кроме фактов Ansible даёт переменные, которые описывают сам прогон.
- Их называют волшебными (magic variables): они есть всегда, их не нужно определять и нельзя переопределить через group_vars.
  - inventory_hostname — имя машины из инвентаря. Именно оно,
  - а не ansible_facts['hostname']: системное имя может быть другим;
  - inventory_hostname_short — то же имя до первой точки;
  - ansible_play_hosts — список машин текущего прогона; при
  - --limit он короче полного списка;
  - hostvars — переменные всех машин инвентаря;
  - groups — словарь групп: имя группы, список машин;
  - group_names — список групп, в которые входит машина;
  - inventory_dir — каталог с инвентарём;
  - ansible_check_mode — true, если прогон идёт с --check;
  - ansible_forks — сколько машин обрабатывается параллельно.
- Живые примеры — где я, кто мои соседи, откуда взят инвентарь:

```
ansible web1 -m ansible.builtin.debug \
-a "msg={{ inventory_hostname }} {{ inventory_dir }}"
```

# ШАГ 6. JINJA2: ПОДСТАНОВКА, ЦИКЛЫ, ФИЛЬТРЫ, ТЕСТЫ

- Jinja2 — язык шаблонов, на котором Ansible собирает текст из данных.
- Он встречается в значении параметра, в шаблоне файла и в условии when (там без фигурных скобок — об этом в практике 7).
- ЧЕТЫРЕ КОНСТРУКЦИИ, ИЗ КОТОРЫХ СОСТОИТ ПОЧТИ ВСЁ:
  - {{ выражение }} — подставить значение;
  - {% for ... %} ... {% endfor %} — повторять кусок текста;
  - {% if ... %} ... {% endif %} — включить текст по условию;
  - {# комментарий #} — комментарий, в результат не попадает.
- ВНУТРИ {{ }} РАБОТАЮТ ФИЛЬТРЫ: знак | отделяет значение от фильтра, как конвейер в оболочке.
  - default('значение') — подставить значение, если переменной нет.
  - Самый нужный фильтр в шаблонах: он превращает ошибку «переменная
  - не определена» в понятную строку;
  - upper / lower — регистр;
  - join(', ') — склеить список в строку;
  - length — сколько элементов в списке или строке;
  - to_nice_json — показать данные как аккуратный JSON;
  - b64encode — base64 для API, которым нужны «безопасные»

# ЧАСТЬ 2. ОТ ДАННЫХ К ФАЙЛАМ: ШАБЛОНЫ

-

# ШАГ 7. TEMPLATE ПРОТИВ COPY: ПОЧЕМУ ШАБЛОН, А НЕ SED

- Три модуля кладут файл на узел, и разница в том, откуда берётся содержимое: ansible.builtin.copy переносит готовый файл как есть, ansible.builtin.template подставляет значения переменных и кладёт результат,...
- До сих пор страница сайта была одинаковой на всех машинах.
- Теперь на каждой своё имя, свой адрес и свой владелец.
- Соблазн — скопировать файл и дописать нужное через sed:
  - name: Подставить имя машины
  - ansible.builtin.shell: >
  - sed -i "s/HOSTNAME/{{ inventory_hostname }}/" /var/www/html/index.html
- ПОЧЕМУ ЭТО ПЛОХО, хотя и работает:
  - sed не идемпотентен: первый прогон заменил HOSTNAME, второй его
  - не находит, а другой рисунок замены сработает повторно. Задача
  - каждый раз показывает changed;
  - shell обходит проверку синтаксиса: Ansible не знает, что делает
  - эта строка;
  - слэш, амперсанд или кавычка в значении ломают выражение sed,
  - а приходят они из данных;
  - у shell нет --diff, поэтому изменение не видно заранее;

# ШАГ 8. ШАБЛОН INDEX.J2: СТРАНИЦА С ФАКТАМИ

- Плейбук 03_index_template.yml целиком:

```
---
# Практика 6, шаг 8: главная страница сайта из шаблона Jinja2.
# На db1 страница не нужна: там нет веб-сервера, поэтому запускаем
# с ограничением по группе web:
# ansible-playbook 03_index_template.yml --limit web
# Условие when, которое уберёт --limit, появится в практике 7.
- name: Развернуть главную страницу сайта
hosts: linux
```

- В плейбуке описано намерение «у всех linux-машин есть каталог сайта», а --limit web — решение этого прогона.
- Без него страница легла бы и на db1, где её никто не отдаёт.
- Шаблон templates/index.j2:
- Три строки, ради которых шаблон и написан. {{ inventory_hostname }} в заголовке — имя машины из инвентаря. {{ ansible_facts['default_ipv4'] ['address'] }} — адрес: сначала словарь по ключу default_ipv4, потом...
- Запускаем и смотрим результат:
- На web2 те же строки, но имя и адрес другие:
- Теперь посмотрим изменение, не меняя ничего на узле, — передадим другой заголовок через --extra-vars:

# ШАГ 9. ШАБЛОН MOTD.J2: ЦИКЛ ПО GROUPS['LINUX']

- Второй шаблон показывает то, чего у страницы не было: цикл и данные других машин. /etc/motd — текст, который показывают при входе по SSH.
- Плейбук 04_motd.yml:

```
---
# Практика 6, шаг 9: /etc/motd из шаблона с циклом по группе linux.
# Запуск: ansible-playbook 04_motd.yml
- name: Положить /etc/motd из шаблона
hosts: linux
become: true
gather_facts: true
tasks:
```

- Шаблон templates/motd.j2:
- {% for host in groups['linux'] %} — начало цикла: список из трёх имён, тело выполняется три раза.
- Вывод одинаков на всех машинах, потому что список берётся из инвентаря, а не из фактов текущего узла: motd — сводка о парке, а не о машине.
- {% set address = ... %} — переменная внутри шаблона, чтобы не повторять длинное выражение.
- Значение берётся из hostvars по имени машины, а фильтр default подстрахует, если факта нет — например, при прогоне с --limit по части машин. {{ '%-8s' | format(host) }} — выравнивание по столбцам: фильтр...
- Посмотрим результат:

# ШАГ 10. HOSTVARS, GROUPS, GROUP_NAMES: ДАННЫЕ ОБО ВСЁМ ПАРКЕ

- Волшебные переменные мы назвали в шаге 5, но не запускали.
- Плейбук 05_hostvars.yml целиком — он же используется в шаге 6:

```
---
# Практика 6, шаг 10: волшебные переменные, hostvars и данные обо всём парке.
# Запуск: ansible-playbook 05_hostvars.yml
#
# Опыт с ограничением по машинам: ansible-playbook 05_hostvars.yml --limit db1
# Тогда факты web2 в hostvars не собраны, и задача «Показать адрес web2»
# падает с ошибкой про undefined. Это разобрано в практике, в разделе
# «Если что-то пошло не так».
```

- Запускаем:
- Строка group_names у web1 — web, debian, linux: готовый ответ на вопрос «кто я», и в практике 7 он станет основой условий when.
- Проверим режим --check: по переменной ansible_check_mode в шаблоне можно, например, не перезапускать службу в холостом прогоне.
- ▸ **ЧЕГО НЕ ДЕЛАТЬ: не стройте шаблон на hostvars, не проверив, что нужная машина есть в прогоне. hostvars знает про все машины инвентаря, но факты есть только у машин текущего play. Запустите пример с...**
- Задача про адрес web2 падает на db1, потому что в этом прогоне собирались факты только db1.
- Та же задача внутри цикла не падает: там стоит фильтр default.

# ШАГ 11. VALIDATE: ПРОВЕРКА ДО ТОГО, КАК ФАЙЛ ПОПАЛ НА МЕСТО

- У модулей copy и template есть параметр validate: в него передают команду проверки, а Ansible подставляет вместо %s путь к временному файлу с новым содержимым.
- Файл проверяют до публикации; не прошёл проверку — задача падает, а рабочий файл остаётся прежним.
- Классический пример — sudoers: ошибка в этом файле закрывает sudo.
- Файл с правилом (sudoers-lab) лежит рядом с плейбуком.
  - name: Положить правило sudo с проверкой
  - ansible.builtin.copy:
  - src: sudoers-lab
  - dest: /etc/sudoers.d/10-lab-nopass
  - owner: root
  - group: root
  - mode: "0440"
  - validate: "visudo -cf %s"
- Второй пример — конфигурация nginx:
  - name: Положить конфигурацию nginx с проверкой
  - ansible.builtin.template:
  - src: templates/nginx.conf.j2

# ШАГ 12. СКОЛЬКО СТОИТ СБОР ФАКТОВ И GATHER_FACTS: FALSE

- Полный сбор фактов — это не мгновение:

```
time ansible all -m ansible.builtin.setup > /dev/null
real 0m2.143s
```

- Две секунды на три машины — параллельно.
- Посчитаем для парка среднего размера: триста машин, forks = 10 (значение из нашего ansible.cfg), полторы-две секунды на машину:

```
300 / 10 * 2 = 60 секунд
```

- Минута только на сбор фактов, и это в каждом прогоне, даже если плейбук всего лишь копирует один файл.
- Поэтому у play есть ключ gather_facts.
- Указывать его явно — хорошая привычка, даже когда значение true:
  - name: Положить статичную страницу
  - hosts: web
  - gather_facts: false
  - tasks:
  - name: Разложить страницу
  - ansible.builtin.copy:
  - src: index.html

# Если что-то пошло не так

▸ **ПРОБЛЕМА: 'dict object' has no attribute 'ansible_distibution'**

```
fatal: [web1]: FAILED! => {
"msg": "The task includes an option with an undefined variable. ...
'dict object' has no attribute 'ansible_distibution'. ..."
}
```

- В части выпусков то же самое печатается как 'ansible_facts' has no attribute 'ansible_distibution' — смысл один: ключа с таким именем в словаре нет.
- Причина в большинстве случаев — опечатка в имени факта.
  - Сверьте имя по буквам: distribution, а не distibution.
  - Есть ли факт на машине:
  - ansible web1 -m ansible.builtin.setup -a "filter=ansible_distribution*"
  - Не смешаны ли длинная и короткая формы: работают
  - ansible_facts['distribution'] и ansible_distribution, а запись
  - ansible_facts.distribution без кавычек — уже ошибка.

▸ **ПРОБЛЕМА: VARIABLE IS NOT DEFINED!**

```
ansible web1 -m ansible.builtin.debug -a "var=lab_owner"
web1 | SUCCESS => {
```

# Проверка результата

- Практика выполнена, если выполняются все пункты.
- 1.
- Факты видны и по ним понятно, что за машина:

```
ansible all -m ansible.builtin.setup -a "filter=ansible_os_family*"
web1 | SUCCESS => { ... "ansible_os_family": "Debian" }
db1 | SUCCESS => { ... "ansible_os_family": "RedHat" }
```

- 2.
- Разница между filter и gather_subset усвоена: первый отбирает

```
вывод, второй — сбор:
ansible web1 -m ansible.builtin.setup -a "gather_subset=!all,!min,local" | wc -l
ansible web1 -m ansible.builtin.setup | wc -l
...около 1300 строк
```

- 3.
- Плейбук с фактами и проверками проходит на трёх узлах, changed=0:

```
ansible-playbook 01_facts.yml
```

- 4.

# Что дальше

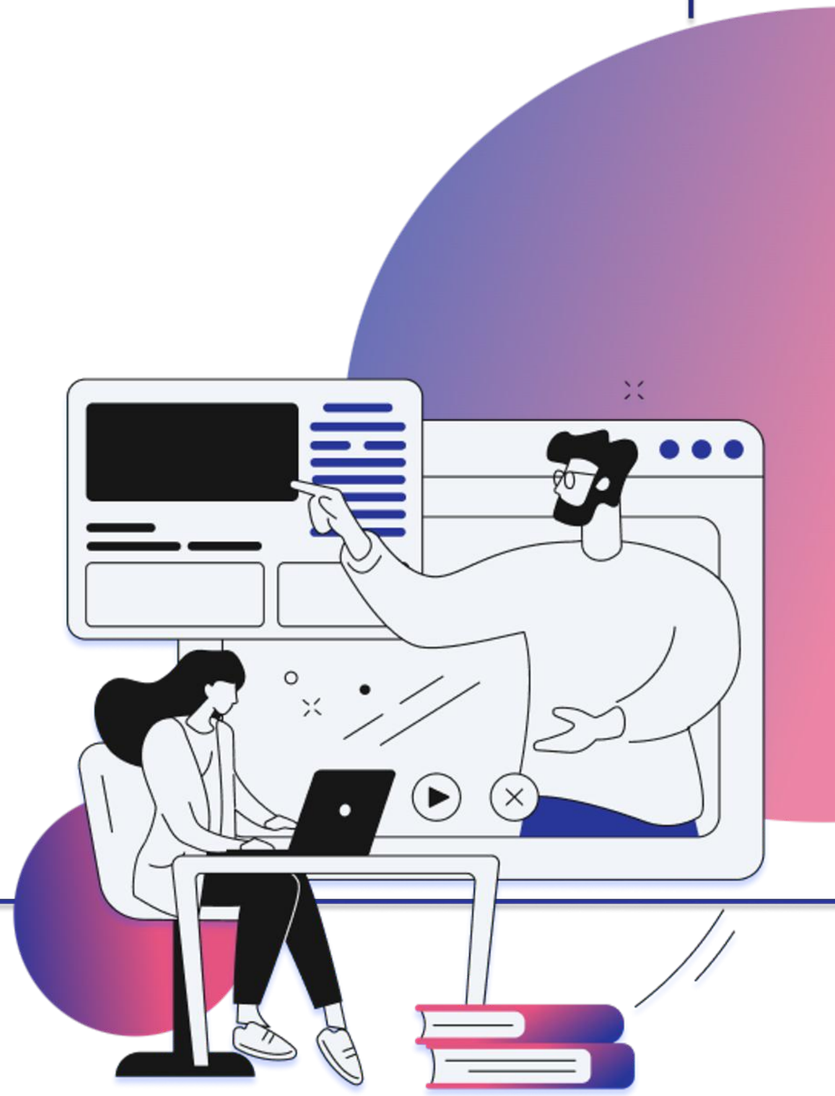
- Далее в курсе:
  - практика 7 — условия, циклы, блоки, обработчики: условие when,
  - которое уберёт --limit web и разделит семейства ОС, станет основным
  - инструментом; факты из этой практики превратятся в условия,
  - а не в отчёт;
  - практика 8 — файлы, права, текстовые конфигурации: lineinfile,
  - blockinfile, find и stat; там же validate на sshd_config
  - с проверкой sshd -t, о которой мы говорили в шаге 11;
  - практика 9 — административные задачи: sudoers раскладывается
  - с validate: "visudo -cf %s", как в шаге 11;
  - практика 10 — роли: шаблоны переезжают в роль web_admin,
  - в её собственный каталог templates/;
  - практика 12 — Jinja2 для настоящих конфигов: там же место
  - для {% raw %}, trim_blocks и lookup('template', ...), и там же
  - validate становится основным приёмом;
  - практика 14 — сложные проекты: отчёт о парке через hostvars

Учебный центр АйТи Клауд

# Практика 7. Условия, циклы, блоки, обработчики

**ЭТА ПРАКТИКА — ПРО ДВА СЕМЕЙСТВА ОС СРАЗУ.**

современные технологии простым языком



# Обзор практики

- ЭТА ПРАКТИКА — ПРО ДВА СЕМЕЙСТВА ОС СРАЗУ.
- В практике 4 плейбук работал на Ubuntu и падал на Rocky Linux.
- Здесь он научится принимать решения: ставить apache2 или httpd, ждать готовности службы и перезапускать её только при изменении конфигурации.

## Что мы сделаем

- Разберём условия when: по фактам, по переменным и по результату register.
- Соберём один плейбук для Ubuntu и Rocky Linux: сначала двумя задачами с when, потом блоками с одним when на блок, потом через переменные семейства ОС.
- Познакомимся с block, rescue и always.
- Разберёмся, чем ошибка отличается от «не ошибки»: ignore_errors, failed_when, changed_when.
- Пройдём циклы loop, dict2items и loop_control, дождёмся готовности службы через until, retries и delay, разберём обработчики с listen и теги.
- В конце тот же плейбук на втором прогоне даст changed=0.

# Что понадобится

- СТЕНД ИЗ ПРАКТИК 1–6 В РАБОЧЕМ СОСТОЯНИИ:

```
control 10.0.2.5 — управляющая машина, Ubuntu 26.04 LTS;
web1 10.0.2.11 — Ubuntu 26.04 LTS;
web2 10.0.2.12 — Ubuntu 26.04 LTS;
db1 10.0.2.13 — Rocky Linux 10.
```

- Нужны: пользователь engineer, вход по ключу /home/engineer/.ssh/id_ed25519, sudo без пароля, активированное окружение (source ~/ansible-venv/bin/activate), инвентарь и ansible.cfg из практики 2.
- Учебный пароль Console47 не нужен.
- ЧТО НУЖНО ИЗ РЕПОЗИТОРИЯ КУРСА — восемь файлов:

```
files/07_conditions_loops/01_when.yml
files/07_conditions_loops/02_two_families.yml
files/07_conditions_loops/03_blocks.yml
files/07_conditions_loops/04_loops.yml
```

- Все они кладутся в рабочий каталог ~/ansible-lab — рядом с ansible.cfg и inventory.ini из практики 2.
- Каталог vars/ создаётся там же: путь vars/… относительный, он отсчитывается от каталога плейбука.
- ▸ **ПОЧЕМУ ИМЕННО ТАК: Ansible ищет ansible.cfg только в четырёх местах — в ANSIBLE_CONFIG, в текущем каталоге, в ~/.ansible.cfg и в /etc/ansible/ansible.cfg — и вверх по каталогам не поднимается. Из подкаталога...**
- ▸ **ВРЕМЯ: 2 академических часа. Межсетевой экран (ufw на Ubuntu, firewalld на Rocky Linux) здесь только упоминается: разбор правил и модулей — практика 9.**

# ЧАСТЬ 1. УСЛОВИЯ И ДВА СЕМЕЙСТВА ОС

-

# ШАГ 1. ПОЧЕМУ ОДИН ПЛЕЙБУК ЛОМАЕТСЯ НА ROCKY LINUX

- Практика 4 закончилась тем, что плейбук 04_web_full.yml работал на web1 и web2 и падал на db1: на Rocky Linux модуль apt не работает вообще, потому что там нет python3-apt.

```
TASK [Обновить кэш пакетов]
fatal: [db1]: FAILED! => {"changed": false, "msg": "Could not import the python3-apt module using /usr/bin/...
```

- Сообщение зависит от версии ansible-core, но падает уже первая задача — та, где вызывается ansible.builtin.apt, — и говорит она про отсутствующий python3-apt, а не про пакет apache2.
- Практика 4 показывает это на первой же задаче.
- Причина для нас всё равно одна: в семействе RedHat веб-сервер Apache ставится пакетом httpd, служба тоже называется httpd, а конфигурация лежит не в /etc/apache2, а в /etc/httpd/conf.d.

▸ **ПОЧЕМУ ИМЕННО ТАК: модуль apt знает только про apt, модуль dnf — только про dnf. Ни один модуль не знает, что apache2 и httpd — один и тот же веб-сервер. Это знание живёт в описании — в плейбуке или в...**

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: парк из одной операционной системы — редкость. Обычно рядом живут Ubuntu и Rocky Linux, Debian и AlmaLinux, и различаются имена пакетов, пути конфигурации и имена служб.**

- Сказать плейбуку, где он находится, можно тремя способами: условием when в задаче, блоком с одним when на группу задач и переменными по семействам ОС.
- Разберём все три по порядку.

# ШАГ 2. WHEN: УСЛОВИЯ ПО ФАКТАМ, ПЕРЕМЕННЫМ И REGISTER

- Значение when — выражение на языке Jinja2, записанное без шаблонных скобок: Ansible сам подставит переменные и вычислит результат.
- Результат обязан быть логическим.
- Работаем из ~/ansible-lab, рядом с ansible.cfg и inventory.ini из практики 2.
- Создайте файл 01_when.yml:

```
cd ~/ansible-lab
# 01_when.yml — шаг 2 практики 7: условия when.
# Плейбук ничего не меняет на узлах: только проверки.
---
- name: Условия по фактам, переменным и результату задачи
hosts: web:db
become: false
gather_facts: true
```

- Запускаем:
- Вывод сокращён.
- Строки skipping — не ошибка: так Ansible сообщает, что задача пропущена, потому что условие ложно.
- На web1 пропущена задача с отрицанием (там Debian), на db1 — задача по группе инвентаря.

# ШАГ 3. ДВА СЕМЕЙСТВА «В ЛОБ»: ДВЕ ЗАДАЧИ С WHEN

- Первый рабочий вариант — на каждое действие по две задачи со своим when.
- Файл 02_two_families.yml:

```
# 02_two_families.yml — шаг 3 практики 7.
# Вариант «в лоб»: на каждое действие по две задачи с when.
---
- name: Веб-сервер на Ubuntu и Rocky Linux двумя задачами с when
hosts: web:db
become: true
gather_facts: true
tasks:
```

- Здесь и дальше используется ansible.builtin.service — тот же модуль, что и в практике 4: он сам определяет, чем управлять службой.
- Второй модуль для служб, ansible.builtin.systemd_service (короткое имя systemd — прежнее, в примерах практики 9 записано именно оно), разбирается в практике 9, шаг 6: на службах и на своём таймере.
- Запускаем:
- Работает, и ни одной строки failed.
- Но посмотрите на цену: половина плейбука не выполняется ни на одной машине.
- Две задачи пропущены на каждой — на Ubuntu это ветка Rocky Linux, на Rocky — ветка Ubuntu.

# ШАГ 4. ОДИН WHEN НА БЛОК И ОТКАТ ЧЕРЕЗ RESCUE

- Ключ block собирает задачи в группу, и условие у блока действует на все задачи внутри.
- Файл 03_blocks.yml целиком:

```
# 03_blocks.yml — шаг 4 практики 7: block, rescue и always.
# Первый play — по блоку на семейство: один when на блок.
# Второй play падает намеренно: так видно, что делает rescue.
---
- name: Веб-сервер на двух семействах двумя блоками
hosts: web:db
become: true
gather_facts: true
```

- Первый play даёт то же состояние, что и файл шага 3, но условие стоит один раз на блок.
- Добавить в блок новую задачу можно, не думая про when: условие уже написано.
- ▸ **ЧЕГО БЛОК НЕ ДЕЛАЕТ: он не ускоряет выполнение. Пропущенные задачи на узел вообще не отправляются, условие вычисляется на управляющей машине. Экономия — время автора и того, кто читает плейбук перед запуском....**
- Запускаем второй play — он падает намеренно:
- Строка fatal с FAILED! — настоящий отказ: пакета нет в репозиториях.
- Дальше сработала ветка rescue: сообщение и откат.

# ЧАСТЬ 2. ЧТО ОТЛИЧАЕТ РАБОТУ ОТ УЧЕБНОГО ПРИМЕРА

-

# ШАГ 5. ПЕРЕМЕННЫЕ ПО СЕМЕЙСТВАМ ВМЕСТО WHEN В КАЖДОЙ ЗАДАЧЕ

- Два блока убрали повтор условий, но имена пакетов и служб остались в задачах.
- Если в парке появится AlmaLinux, придётся править задачи.
- Правильнее держать различия в данных: тогда задачи выглядят одинаково для любого семейства.
- Создайте каталог vars и два файла:

```
cd ~/ansible-lab
mkdir -p vars
---
# Переменные семейства Debian (Ubuntu 26.04 LTS: web1, web2).
web_package: apache2
web_service: apache2
web_config_dir: /etc/apache2/sites-available
web_root: /var/www/html
```

- Подключаются они одной задачей, которая есть в файлах 05_until.yml и 06_handlers_tags.yml:
  - name: Подключить переменные семейства ОС
  - ansible.builtin.include_vars:
  - file: "vars/{{ ansible_os_family }}.yml"

# ШАГ 6. ОШИБКИ, КОТОРЫЕ НЕ ОШИБКИ

- Три ключа меняют отношение Ansible к результату задачи.
- changed_when МЕНЯЕТ ПРИЗНАК ИЗМЕНЕНИЯ.
- Модуль command всегда сообщает changed, даже если команда только читала файл.
- В 01_when.yml и 05_until.yml у таких задач стоит changed_when: false.
- Без него в отчёте всегда было бы changed, и обещание «второй прогон даёт changed=0» не выполнялось бы никогда.
- failed_when МЕНЯЕТ ПРИЗНАК ОТКАЗА: так задаче говорят, что считать отказом.
- Два полезных направления:

```
failed_when: false — не считать отказом ничего;
failed_when: result.rc > 1 — отказом считать только серьёзный код.
```

- Первый вариант нужен, когда неудача задачи — тоже информация.
- Пример: проверка, установлен ли пакет.
- Команда rpm -q вернёт 1, если пакета нет, и это нормальный результат проверки:
  - name: Проверить, установлен ли httpd
  - ansible.builtin.command:
  - cmd: rpm -q httpd
  - register: httpd_check

# ШАГ 7. ЦИКЛЫ: LOOP, DICT2ITEMS И LOOP_CONTROL

- Второй способ убрать повторы — цикл.
- Он нужен там, где действий много, а различаются они одним значением: списком пакетов, списком каталогов, списком пользователей.
- Файл 04_loops.yml:

```
# 04_loops.yml — шаг 7 практики 7: циклы loop, dict2items
# и loop_control. Пакеты с общим именем ставятся одинаково на обоих
# семействах, поэтому здесь хватает общего модуля пакетов.
---
- name: Циклы по списку, словарю и парам значений
hosts: web:db
become: true
gather_facts: true
```

- Запускаем:
- Переменная цикла по умолчанию называется item.
- Дальше три детали, ради которых стоит использовать loop_control.
- label ЗАМЕНЯЕТ ТО, ЧТО ПЕЧАТАЕТСЯ В СКОБКАХ.
- Без него в строке (item={...}) печатался бы весь словарь и вывод было бы не читать.

# ШАГ 8. UNTIL, RETRIES И DELAY: ДОЖДАТЬСЯ ГОТОВНОСТИ СЛУЖБЫ

- Служба запускается не мгновенно: процессу нужно время, чтобы открыть порт.
- Плейбук, который сразу обратится к порту, получит отказ — не потому, что что-то сломано, а потому, что он поторопился.
- Для этого есть until:

```
until — условие, при котором повторы прекращаются;
retries — сколько раз пробовать (по умолчанию 3);
delay — сколько секунд ждать между попытками.
```

- Файл 05_until.yml:

```
# 05_until.yml — шаг 8 практики 7: ожидание готовности службы
# через until, retries и delay.
---
- name: Дождаться готовности веб-сервера
hosts: web:db
```

- Запускаем:
- Если служба поднимается медленнее, чем длится первая попытка, Ansible показывает повторы:
- RETRYING — не ошибка: задача не упала, она ждёт.
- Ошибкой станет только исчерпание попыток:

# ШАГ 9. ОБРАБОТЧИКИ: NOTIFY, LISTEN И ПОРЯДОК ВЫПОЛНЕНИЯ

- Обработчик (handler) выполняется не тогда, когда до него дошла очередь, а тогда, когда его уведомили ключом notify.
- Смысл простой: перезапускать службу нужно только при изменении конфигурации.

```
# 06_handlers_tags.yml — шаги 9–10 практики 7.
# Итоговый плейбук практики: оба семейства, обработчики
# с listen и теги. Второй прогон даёт changed=0.
---
- name: Итоговый веб-сервер для Ubuntu и Rocky Linux
hosts: web:db
become: true
gather_facts: true
```

- Здесь три приёма.

▸ **ПЕРВЫЙ: НЕСКОЛЬКО УВЕДОМЛЕНИЙ. Задача копирования уведомляет двумя строками: тему «перезапустить веб-сервер» и обработчик «Проверить, что служба отвечает». Обработчики выполняются в том порядке, в котором...**

▸ **ВТОРОЙ: LISTEN. Обработчик можно уведомлять не по имени, а по теме. Тема «перезапустить веб-сервер» указана у двух обработчиков, поэтому одно уведомление запускает оба. Это удобно, когда обработчик...**

▸ **ТРЕТИЙ: HANDLER ВЫПОЛНЯЕТСЯ ОДИН РАЗ. Сколько бы задач его ни уведомили, он выполнится один раз и в конце play. Уведомление по имени и по теме в одну и ту же задачу тоже даёт один запуск.**

- Запускаем первый раз:
- Второй прогон:

# ШАГ 10. ТЕГИ: ЗАПУСТИТЬ ЧАСТЬ ПЛЕЙБУКА

- Тег — метка на задаче.
- По меткам запускают не весь плейбук, а нужную часть.
- В итоговом файле четыре тега: always, install, config, service.
- Тег always означает «выполнять всегда» и спасает, когда от тега зависит переменная: без него задача include_vars не попала бы в запуск с --tags config, и копирование упало бы на {{ web_root }}.

```
ansible-playbook --list-tags 06_handlers_tags.yml
playbook: 06_handlers_tags.yml
play #1 (web:db): Итоговый веб-сервер для Ubuntu и Rocky Linux TAGS: []
TASK TAGS: [always, config, install, service]
ansible-playbook --tags install 06_handlers_tags.yml
PLAY RECAP
db1 : ok=3 changed=0 unreachable=0 failed=0 skipped=0 rescued=0 ignored=0
```

- С ключом --tags install выполнились только задачи с тегом install и always: три вместо пяти.
- Ключ --skip-tags config делает обратное — выполняет всё, кроме задач с этим тегом.
- Тегов у задачи может быть несколько: tags: [install, config].

▸ **ПОЧЕМУ ИМЕННО ТАК: имена тегов — часть интерфейса плейбука, как имена задач. Тег packages или databases понятен с первого взгляда, тег test1 через месяц не значит ничего. Держите три-пять тегов на плейбук.**

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: теги нужны при разборе поломки. Когда упала одна задача на одной машине, запуск --tags config --limit web1 --check занимает секунды, а полный прогон по парку — минуты.**

# ШАГ 11. ИТОГ: ОДИН ПЛЕЙБУК НА ДВУХ СЕМЕЙСТВАХ

- Проверим то, ради чего всё делалось.
- В итоговом файле нет ни одной проверки семейства, но он работает и на Ubuntu, и на Rocky Linux: имена приходят из vars/Debian.yml и vars/RedHat.yml.
- В отчёте третьего прогона нет ни failed, ни skipped, ни changed.

```
curl -s http://10.0.2.11/
<h1>Стенд курса Ansible: web1</h1>
curl -s http://10.0.2.13/
<h1>Стенд курса Ansible: db1</h1>
```

- Одинаковые страницы на разных семействах — результат, а не совпадение: задача одна, а данные для неё разные.
- ▸ **ПОЧЕМУ ИМЕННО ТАК: в итоговом файле нет ни одного when — и это правильно. when остаётся для случаев, когда различие нельзя описать данными: разный набор служб, разное поведение, разные модули. Всё, что...**

# Если что-то пошло не так

▸ **ПРОБЛЕМА: conditional statements should not include jinja2 templating delimiters**

```
[WARNING]: conditional statements should not include jinja2 templating delimiters such as {{ }} or {% %}. F...
```

- Так отвечали версии ansible-core до 2.19.
- В 2.19 та же ошибка выглядит иначе: если всё значение обёрнуто в {{ }}, будет предупреждение Conditionals should not be surrounded by templating delimiters, а если скобки стоят внутри выражения — отказ с...
- Причина одна: when — это выражение, а не шаблон.
- Проверяйте по порядку:
  - Найдите строку с when и уберите из неё {{ }}:
  - when: ansible_os_family == "Debian"
  - Поищите остальные случаи: grep -n "when:.*{{" *.yml
  - Запустите --syntax-check: ошибка в условии видна только
  - при выполнении, синтаксис YAML при этом верный.

▸ **ПРОБЛЕМА: Conditional result was derived from value of type 'str'**

```
fatal: [web1]: FAILED! => {"msg": "Conditional result (True) was derived from value of type 'str' at '<исто...
```

- Проверка из ansible-core 2.19: результат условия обязан быть логическим.
- Строка, число, непустой словарь — уже не условие, даже если значение похоже на истину.
- Порядок проверки:

# Проверка результата

- Практика выполнена, если выполняются все пункты.
- 1.
- Все шесть плейбуков разбираются:

```
cd ~/ansible-lab
ansible-playbook --syntax-check 01_when.yml 02_two_families.yml \
03_blocks.yml 04_loops.yml 05_until.yml 06_handlers_tags.yml
```

- 2.
- Условия работают и видно, что пропускается:

```
ansible-playbook 01_when.yml
db1 : ok=7 changed=0 ... skipped=1 ...
На web1 пропущена задача с отрицанием, на db1 — по группе.
```

- 3.
- Каждое семейство получило свой веб-сервер:

```
ansible db1 -b -m ansible.builtin.command -a "systemctl is-active httpd"
db1 | CHANGED | rc=0 >>
```

- 4.

# Что дальше

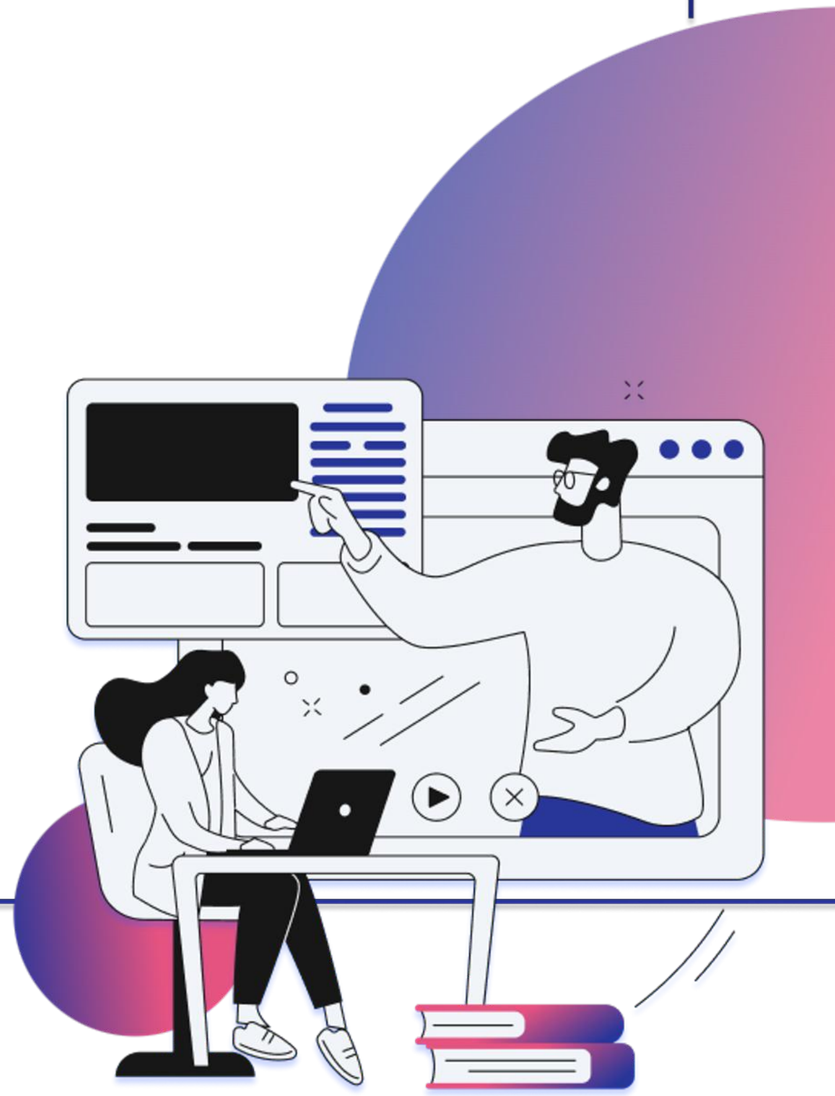
- Далее в курсе:
  - практика 8 — файлы, права и текстовые конфигурации: та же
  - страница и тот же конфиг, но теперь с правами, backup,
  - lineinfile и blockinfile; условия из этой практики там
  - используются, чтобы не трогать чужой файл;
  - практика 9 — административные задачи: пакеты, пользователи,
  - службы и межсетевой экран. ufw и firewalld из шага 1 разбираются
  - там как следует, и снова на двух семействах;
  - практика 10 — роли: всё, что мы написали в файлах 05_until.yml
  - и 06_handlers_tags.yml, станет ролью web_admin; переменные
  - по семействам переедут в vars/ роли, обработчики — в tasks
  - и handlers роли;
  - практика 12 — Jinja2 для настоящих конфигов: шаблоны с циклами
  - и условиями внутри файла, а не в задачах;
  - практика 14 — сложные проекты: serial, delegate_to, include_tasks
  - и теги уже на весь проект.

Учебный центр АйТи Клауд

# Практика 8. Файлы, права и текстовые конфигурации

**ЭТА ПРАКТИКА О ТОМ, КАК ANSIBLE РАБОТАЕТ С ФАЙЛАМИ: КОПИРУЕТ, СОЗДАЁТ, ПРАВИТ ПО СТРОКАМ, ПРОВЕРЯЕТ И УПАКОВЫВАЕТ.**

современные технологии простым языком



# Обзор практики

- ЭТА ПРАКТИКА О ТОМ, КАК ANSIBLE РАБОТАЕТ С ФАЙЛАМИ: КОПИРУЕТ, СОЗДАЁТ, ПРАВИТ ПО СТРОКАМ, ПРОВЕРЯЕТ И УПАКОВЫВАЕТ.
- РАНЬШЕ ФАЙЛЫ ТОЛЬКО ПОЯВЛЯЛИСЬ НА УЗЛАХ, ТЕПЕРЬ МЫ ОТВЕЧАЕМ ЗА ИХ ПРАВА И СОДЕРЖИМОЕ.
- ОПИРАЕТСЯ НА ПРАКТИКИ 2, 4, 6 И 7.

## Что мы сделаем

- Разберём, чем отличаются модули copy, file и template, и когда берут один, а не другой.
- Научимся ставить права восьмеричной и символической записью, поймём, откуда берётся ошибка в записи mode без кавычек и почему 0777 — не «чтобы точно работало».
- Разложим статический сайт на web1 и web2 из каталога files/08_files/site, поставим на файлы 0644, на каталог 0755 и добавим на каталог setgid.
- Разберёмся с каталогами, ссылками и удалением файлов.
- Дальше — точечная правка чужих конфигураций: lineinfile, blockinfile и replace.
- Включим в sshd запрет входа root и входа по паролю через отдельный файл в каталоге sshd_config.d, проверим конфигурацию командой sshd -t и перезапустим службу.
- В конце научимся принимать решения по файлам: find и stat как источник данных, assert как проверка предположений, archive и unarchive как работа с каталогом целиком.

# Что понадобится

```
control 10.0.2.5 — управляющая машина, Ubuntu 26.04 LTS
web1 10.0.2.11 — Ubuntu 26.04 LTS
web2 10.0.2.12 — Ubuntu 26.04 LTS
```

- db1 в этой практике не участвует: sshd мы правим на Ubuntu, а сайт разворачивается только на веб-узлах.

▸ **ЧТО ДОЛЖНО БЫТЬ СДЕЛАНО РАНЬШЕ: стенд собран и вход по ключу работает (практика 1); в каталоге /home/engineer/ansible-lab лежат inventory.ini и ansible.cfg, и команды выполняются из него (практика 2); известны...**

- ФАЙЛЫ ИЗ РЕПОЗИТОРИЯ лежат в files/08_files: site/index.html и site/style.css — содержимое сайта; conf/sshd_hardening.conf и conf/hosts.append — образцы для правки конфигураций
- templates/cron.j2 — шаблон задания cron; плейбуки 01_copy.yml, 02_permissions.yml, 03_lineinfile.yml, 04_blockinfile.yml и 05_archive.yml — по шагам 5, 6, 7, 9 и 13
- README.md каталога объясняет назначение каждого файла.
- Копировать надо содержимое каталога, а не сам каталог, иначе файлы лягут в ~/ansible-lab/08_files, а плейбуки ищут site/ и templates/ рядом с собой:

```
scp -r files/08_files/. engineer@10.0.2.5:/home/engineer/ansible-lab/
```

- Если репозиторий уже лежит на управляющей машине, то же самое делается на месте, из каталога ~/ansible-lab:

```
cp -r /путь/к/репозиторию/files/08_files/. .
```

▸ **СКОЛЬКО ЭТО ЗАЙМЁТ: около двух академических часов. Если стенда на Rocky Linux нет, шаг 7 про sshd_config.d проходится полностью, а различия семейств разбираются по тексту без запуска.**

# ЧАСТЬ 1. ФАЙЛЫ, ПРАВА И СОСТОЯНИЯ

-

# ШАГ 1. COPY, FILE, TEMPLATE: ЧТО ЧЕМ ДЕЛАЕТСЯ

- Три модуля постоянно путают.
- Разница не в том, «что умеет модуль», а в том, откуда берётся содержимое.

```
ansible.builtin.copy берёт готовое содержимое и кладёт как есть:
файл рядом с плейбуком или строку
в параметре content.
ansible.builtin.template берёт шаблон Jinja2, подставляет значения
переменных и кладёт результат.
ansible.builtin.file содержимое не трогает вообще: каталог,
ссылка, пустой файл, права, владелец,
удаление.
```

- Правило выбора: нужны значения переменных — template; файл всегда одинаковый — copy; надо поменять только права или владельца — file; надо поменять одну строку в чужом файле, который принадлежит пакету, —...

▸ **ПОЧЕМУ ИМЕННО ТАК: модуль отвечает за результат, а не за способ. Задача «файл на узле должен быть таким» описывается одним модулем, и Ansible сам решает, надо ли что-то делать: второй прогон не делает ничего,...**

- Параметры: src — что взять; dest — полный путь на узле; content — содержимое строкой вместо src; owner, group, mode — владелец, группа и права
- backup: true — сохранить копию перед заменой; force: false — не перезаписывать существующий файл.
- У модуля file вместо них два своих: state (directory, file, touch, link, hard, absent) и recurse — применить владельца и права ко всему дереву.

▸ **ВАЖНО: copy и template сравнивают содержимое и не перезаписывают файл, если оно не изменилось. Поэтому backup: true не создаёт гору копий при каждом прогоне, а notify у задачи не срабатывает зря. Справка по...**

# ШАГ 2. ПРАВА: ВОСЬМЕРИЧНАЯ И СИМВОЛИЧЕСКАЯ ЗАПИСЬ

- Права в Linux — три группы по три бита: владелец (u, user), группа (g, group), остальные (o, other).
- Внутри группы три бита: чтение r = 4, запись w = 2, выполнение x = 1.
- Сумма битов даёт цифру.

```
0644 = владелец читает и пишет, группа читает, остальные читают.
0755 = владелец читает, пишет и выполняет, группа и остальные
читают и выполняют. Так выглядит каталог.
0600 = только владелец читает и пишет. Так выглядит закрытый ключ.
```

- Четыре цифры вместо трёх — другая группа битов: 0644 обычные права, 1644 плюс sticky bit (удалять файл в каталоге может только его владелец), 2644 плюс setgid (новые файлы получают группу каталога), 4644 плюс...
- Первая ловушка — запись без кавычек:

```
# так делать нельзя
mode: 0644
# так правильно
mode: "0644"
```

- В первом случае YAML разбирает 0644 как восьмеричное число и передаёт в модуль число, а не строку из четырёх символов.
- Ansible 2.19 такое значение принимает и поправляет сам, но выдаёт предупреждение, а в шаблонах и переменных то же самое число ведёт себя иначе.

▸ **ВАЖНО: правило курса — права всегда строкой в кавычках: mode: "0644", mode: "0755", mode: "0600". Это же требует ansible-lint правилом risky-octal, и на нём спотыкаются почти все, кто пишет mode: 0644.**

# ШАГ 3. КАТАЛОГИ, ССЫЛКИ, УДАЛЕНИЕ

- У каталога два отличия от файла.
- Первое: бит x у каталога означает «можно войти и обратиться к файлу внутри», и без него страница не откроется, даже если файл читается всеми.
- Второе: права на каталог и права на содержимое задаются разными задачами.
  - name: Создать каталог сайта
  - ansible.builtin.file:
  - path: /var/www/html/site
  - state: directory
  - owner: root
  - group: root
  - mode: "0755"
  - name: Привести дерево сайта к одному владельцу
  - ansible.builtin.file:
  - path: /var/www/html/site
  - state: directory
  - owner: root
  - group: root

# ШАГ 4. КОПИРУЕМ КАТАЛОГ И ПИШЕМ ШАБЛОН

- Содержимое сайта — каталог site с двумя файлами.
- Скопировать его целиком можно одной задачей, если помнить одно правило: слеш в конце src меняет результат.
- Запись src: site/ кладёт содержимое каталога в dest: получится dest/index.html.
- Запись src: site кладёт сам каталог: получится dest/site/index.html.
- Ошибка здесь встречается почти у всех: задача проходит, а вместо страницы браузер отдаёт список файлов — потому что на месте index.html оказался каталог.

```
backup: true перед заменой сохранить копию с отметкой времени:
index.html.2025-05-14@10-31-07.
force: false не перезаписывать существующий файл. Так защищают
файлы, которые кто-то правил руками.
remote_src: true брать источник на управляемом узле, а не на
управляющей машине. Разбирается в шаге 13.
- name: Положить страницу сайта с сохранением прежней версии
ansible.builtin.copy:
```

- **ЗАЧЕМ ЭТО В РАБОТЕ: backup: true — не замена системе контроля версий, а страховка на первый прогон по живому серверу: содержимое конфигураций всё равно хранят в репозитории, а не в копиях с датой в имени.**
- Теперь template: тот же шаг, но файл собирается из переменных.
- Возьмём шаблон задания cron из files/08_files/templates/cron.j2:

# ШАГ 5. ПЛЕЙБУК 01_COPY.YML: САЙТ НА WEB1 И WEB2

- Собираем первый плейбук практики.
- Его полный текст совпадает с файлом files/08_files/01_copy.yml.

```
---
# 01_copy.yml — шаг 5 практики 8.
- name: Разложить статический сайт
hosts: web
become: true
gather_facts: false
vars:
site_dir: /var/www/html/site
```

- Здесь hosts: web — группа из инвентаря практики 2, то есть web1 и web2; become: true нужен для записи в /var/www и в /etc
- gather_facts: false — факты этому плейбуку не нужны, а сбор фактов стоит времени; vars хранят путь сайта и права, которые в практике 10 переедут в defaults роли.
- Обратите внимание: три переменные cron_minute, cron_hour и cron_mailto объявлены на уровне play, а не у задачи template, и шаблон cron.j2 использует только их и inventory_hostname, поэтому отключение фактов...
- Обратите внимание на пару прав: у задачи file — 0755, у задачи copy — 0644.
- Каталог обязан быть исполняемым для всех, файлы внутри — только читаемыми.
- Если поменять их местами, сайт не откроется, а ошибка будет выглядеть странно: файл на месте, браузер отдаёт 403.

# ШАГ 6. ПЛЕЙБУК 02_PERMISSIONS.YML: SETGID НА КАТАЛОГЕ

- Права на файлы мы расставили при копировании.
- Но у каталога есть свойство, которого копированием не задать: наследование группы новыми файлами.
- За это отвечает бит setgid.
- Пусть сайт наполняют два человека, оба в группе webadmin.
- Без setgid новый файл получает группой основную группу того, кто его создал: у одного это webadmin, у другого — личная группа.
- Через неделю часть файлов недоступна части команды, и никто не может объяснить почему.
- С битом setgid каталог назначает группу новым файлам сам.
- Обозначается он числом 2755 или символически как g+s; в Ansible пишем числом строкой.
- Полный текст плейбука — в файле files/08_files/02_permissions.yml.
- Ключевые задачи:
  - name: Проверить, какие файлы лежат в каталоге сайта
  - ansible.builtin.find:
  - paths: "{{ site_dir }}"
  - file_type: file
  - recurse: true
  - register: site_files

# ЧАСТЬ 2. ТЕКСТОВЫЕ КОНФИГУРАЦИИ И РЕШЕНИЯ ПО ФАЙЛАМ

-

# ШАГ 7. LINEINFILE: ДВЕ СТРОКИ В КОНФИГУРАЦИИ SSHD

- Теперь самая частая задача в работе: поменять одну строку в файле, который мы не писали и не хотим переписывать целиком.
- Для этого три модуля, и первый — lineinfile.
- Он ищет в файле строку по регулярному выражению regexp: нашёл — заменяет её на line, не нашёл — дописывает line в конец файла или в указанное место.

```
path файл, который правим. Обязателен.
line строка, которая должна оказаться в файле.
regexp по какому образцу искать существующую строку.
state present (по умолчанию) или absent – удалить строку.
insertafter вставить после строки, найденной по этому выражению.
insertbefore вставить перед строкой, найденной по этому выражению.
backrefs true – использовать группы выражения в line
(подробно – в практике 11).
```

- Сама задача: закроем два входа, которые открыты по умолчанию почти везде, — вход root по SSH и вход по паролю.
- ПОЧЕМУ ЭТО ДЕЛАЮТ ЧЕРЕЗ ОТДЕЛЬНЫЙ ФАЙЛ, А НЕ ПРАВКОЙ ОСНОВНОГО.
- Файл /etc/ssh/sshd_config принадлежит пакету openssh-server.
- Пакет обновляется — и при обновлении может заменить файл, вернув настройки к исходным.
- Тогда наша правка исчезнет, и никто этого не заметит: служба поднимется с ослабленной конфигурацией.

# ШАГ 8. INSERTAFTER, INSERTBEFORE, STATE: ABSENT И BACKREFS

- lineinfile умеет больше, чем замену существующей строки.
- ПЕРВЫЙ приём.
- Строки в файле нет, и она должна появиться в нужном месте, а не в конце.
- За это отвечает insertafter:
  - name: Разрешить вход группе администраторов
  - ansible.builtin.lineinfile:
  - path: /etc/ssh/sshd_config.d/20-sshd-hardening.conf
  - line: "AllowGroups sudo wheel"
  - insertafter: '^PermitRootLogin'
  - validate: "sshd -t -f %s"
- Правильное значение insertafter — регулярное выражение, а не имя строки.
- Если совпадений нет, строка дописывается в конец файла, и задача завершается успешно: успешная задача не означает, что строка оказалась там, где ждали.
- Параметр insertbefore вставляет перед найденной строкой, а специальное значение insertafter: EOF означает «в конец файла».
- ВТОРОЙ приём.
- Строка в файле есть, и её надо убрать — state: absent вместе с regexp:
  - name: Убрать разрешение входа по паролю для локальной сети

# ШАГ 9. BLOCKINFILE: ЦЕЛЫЙ БЛОК ПОД УПРАВЛЕНИЕМ

- lineinfile управляет одной строкой.
- Когда строк несколько и они связаны между собой, берут blockinfile: он кладёт в файл блок и обрамляет его маркерами.

```
# BEGIN ANSIBLE MANAGED BLOCK
...строки блока...
# END ANSIBLE MANAGED BLOCK
```

- Между этими двумя строками blockinfile считает своей территорией.
- При каждом прогоне блок находится по маркерам, содержимое сравнивается и заменяется целиком, если изменилось.
- Строки за пределами маркеров не трогаются никогда — этим blockinfile и ценен.
- Содержимое блока задаётся так же, как у copy: готовым файлом в src или строкой в block.
- Строку удобно брать из файла через lookup:
  - name: Добавить машины стенда в /etc/hosts
  - ansible.builtin.blockinfile:
  - path: /etc/hosts
  - block: "{{ lookup('ansible.builtin.file', 'conf/hosts.append') }}"
  - marker: "# {mark} ANSIBLE MANAGED BLOCK: стенд курса"
  - owner: root

# ШАГ 10. REPLACE: ЗАМЕНА ПО ВСЕМУ ФАЙЛУ

- Третий инструмент правки — replace.
- Он не ищет строку целиком, а заменяет все совпадения регулярного выражения во всём файле:
  - name: Заменить старый комментарий в настройках sshd
  - ansible.builtin.replace:
  - path: /etc/ssh/sshd_config.d/20-sshd-hardening.conf
  - regexp: '^#\s*PermitRootLogin\s+.*$'
  - replace: '# PermitRootLogin: изменено плейбуком практики 8'
  - backup: true
  - path файл, в котором идёт замена.
  - regexp что искать.
  - replace на что заменить. Пустая строка удаляет найденное.
  - backup true — сохранить копию файла до замены.
  - validate проверить результат до подмены файла.
  - after искать только после строки, найденной по этому выражению.
  - before искать только до строки, найденной по этому выражению.
- Параметры after и before задают не границы файла, а границы участка поиска.

# ШАГ 11. FIND И STAT: РЕШЕНИЯ ПО ФАЙЛАМ, А НЕ ПО ДОГАДКАМ

- Иногда решение зависит от того, что на машине уже есть.
- Гадать («наверное, каталог пустой») нельзя: ответ надо получить у машины.
- Для этого два модуля.
- stat сообщает про один объект: существует ли он, каталог это или файл, каковы права, владелец, размер, время изменения.
- Результат кладут в register и потом читают:
  - name: Узнать состояние каталога сайта
  - ansible.builtin.stat:
  - path: /var/www/html/site
  - register: site_stat
- Поля результата, которые нужны чаще всего: stat.exists — объект есть или нет; stat.isdir, stat.isreg, stat.islnk — каталог, файл, ссылка
- stat.mode — права строкой из четырёх цифр, например 2755; stat.pw_name и stat.gr_name — владелец и группа; stat.size — размер в байтах; stat.mtime — время последнего изменения.
- Все поля сразу видно в выводе debug с var: site_stat.
- find ищет объекты по условиям и возвращает список: paths — где искать (обязателен, список каталогов)
- patterns — образцы имён вроде '*.log', это не регулярные выражения, а обычные маски оболочки; use_regex — если true, patterns читаются как регулярные выражения
- recurse — true, если искать и во вложенных каталогах; file_type — file, directory, link или any; age — отбор по возрасту (+30d старше 30 дней, -1w младше недели)
- size — отбор по размеру (+10M больше 10 МБ); hidden — учитывать скрытые файлы.

# ШАГ 12. STAT И ASSERT: ПРОВЕРКА ПРЕДПОЛОЖЕНИЙ

- Данные, полученные stat и find, кто-то должен проверить.
- В инженерной практике это делает assert: он превращает предположение в условие, которое либо выполняется, либо останавливает плейбук с понятным текстом.
- Условия пишутся на Jinja2 без фигурных скобок: значения подставляются сами, а site_dir_stat.stat.mode == '2755' читается как «права каталога ровно 2755», а не «примерно похожи».
  - name: Отказаться работать при расхождении
  - ansible.builtin.assert:
  - that:
  - site_dir_stat.stat.exists
  - site_dir_stat.stat.isdir
  - site_dir_stat.stat.mode == '2755'
  - site_dir_stat.stat.pw_name == 'root'
  - fail_msg: >-
  - Каталог /var/www/html/site: права {{ site_dir_stat.stat.mode }},
  - владелец {{ site_dir_stat.stat.pw_name }}. Ожидалось 2755
  - и root:root.
  - success_msg: "Каталог сайта описан верно."
  - quiet: true

# ШАГ 13. АРХИВЫ: ARCHIVE И UNARCHIVE

- Каталог целиком удобно передавать архивом.
- В Ansible для этого два модуля: archive упаковывает, unarchive распаковывает.
  - name: Упаковать каталог сайта в архив
  - ansible.builtin.archive:
  - path:
  - /var/www/html/site
  - dest: "/tmp/site-{{ inventory_hostname }}.tar.gz"
  - format: gz
  - mode: "0640"
  - path что упаковать: путь или список путей. Обязателен.
  - dest имя архива. Обязателен.
  - format gz, bz2, xz, zip, tar. По умолчанию gz.
  - remove true — удалить исходные файлы после упаковки.
  - mode права на получившийся архив.
- Имя архива собирается из inventory_hostname.
- Это не украшение: если собирать архивы со всех машин в один каталог, без имени узла в имени файла они затрут друг друга, и заметите вы это только при распаковке.

# ШАГ 14. ИТОГ: САЙТ, ПРАВА И ПОВТОРНЫЙ ПРОГОН

- На web1 и web2 сейчас должно быть: каталог /var/www/html/site с правами 2755 и владельцем root:root; файлы index.html и style.css с правами 0644 внутри него
- файл /etc/cron.d/site-check с заданием из шаблона cron.j2; файл /etc/ssh/sshd_config.d/20-sshd-hardening.conf с двумя настройками и правами 0644
- блок машин стенда в /etc/hosts между маркерами BEGIN и END; архив /tmp/site-web1.tar.gz и его копия в каталоге backup/ на управляющей машине.
- Каждый пункт появился из описания состояния, а не из команды.
- Проверим это прогоном всех плейбуков подряд.
- Порядок важен: сначала каталог и файлы, потом права, потом конфигурации, потом архив.

```
cd ~/ansible-lab
ansible-playbook 01_copy.yml
ansible-playbook 02_permissions.yml
ansible-playbook 03_lineinfile.yml
ansible-playbook 04_blockinfile.yml
ansible-playbook 05_archive.yml
PLAY RECAP *********************************************************
web1 : ok=... changed=1 unreachable=0 failed=0
```

- Один changed из пяти плейбуков — пересборка архива из шага 13.
- Всё остальное уже в описанном состоянии, и обработчик перезапуска SSH в отчёт не попадает: уведомлять его нечем, задачи ничего не меняют.

# Если что-то пошло не так

▸ **ПРОБЛЕМА: Destination directory /var/www/html/site does not exist**

```
fatal: [web1]: FAILED! => {"changed": false, "msg": "Destination
directory /var/www/html/site does not exist"}
```

• Модуль copy не создаёт промежуточные каталоги: он кладёт файл в уже существующий каталог.

• Порядок проверки: каталог действительно есть (ls -ld /var/www/html/site); в плейбуке перед копированием есть задача file со state: directory — именно она создаёт каталог, а не copy

• путь в dest совпадает с путём в задаче file буква в букву, потому что разница между site и site/ меняет результат копирования (шаг 4)

• переменная в path не пустая — пустое значение превращает путь в /var/www/html, и копирование идёт не туда.

▸ **ПРОБЛЕМА: mode must be in octal or symbolic notation**

```
fatal: [web1]: FAILED! => {"changed": false, "msg": "mode must be in
octal or symbolic notation"}
```

• Модуль получил значение, которое не является ни числом в восьмеричной записи, ни символической записью.

• Порядок проверки: как записаны права в задаче — правильно mode: "0644" строкой в кавычках, а число 644 в YAML читается как десятичное, и это другое значение

• нет ли арифметики в выражении вроде mode: "{{ file_mode + 0 }}" — она даёт число, а не строку прав; нет ли пробелов в переменной из group_vars (значение " 0644" ломает разбор)

• есть ли поле mode у элемента цикла, когда задача написана как mode: "{{ item.mode }}".

▸ **ПРОБЛЕМА: lineinfile вставляет строку повторно при каждом прогоне**

```
TASK [Запретить вход по паролю] ***********************************
```

# Проверка результата

- Практика пройдена, если выполняются все пункты.
- Команды выполняются с управляющей машины, из каталога /home/engineer/ansible-lab.
- 1.
- Содержимое сайта лежит на обеих машинах, права описаны верно:

```
ssh engineer@web1 'find /var/www/html/site -printf "%M %u:%g %p\n"'
drwxr-sr-x root:root /var/www/html/site
-rw-r--r-- root:root /var/www/html/site/index.html
Буква s в правах группы каталога — бит setgid из шага 6.
```

- 2.
- Страница отдаётся веб-сервером на обеих машинах:

```
for h in 10.0.2.11 10.0.2.12; do
curl -s -o /dev/null -w "$h %{http_code}\n" http://$h/site/
done
10.0.2.11 200
```

- 3.
- Задание cron разложено из шаблона и имеет правильные права:

# Что дальше

- Далее в курсе:
  - практика 9 — административные задачи: пользователи, пакеты,
  - службы и firewall. Права 0440 на файл в /etc/sudoers.d и проверка
  - visudo -cf — тот же приём validate, что и здесь для sshd -t;
  - практика 10 — роли: пять плейбуков этой практики превратятся
  - в задачи одной роли web_admin, а переменные site_dir и права
  - переедут в defaults;
  - практика 11 — регулярные выражения и точечная правка: разбор
  - regexp, якорей, жадности и backrefs, с которыми здесь мы работали
  - почти вслепую;
  - практика 12 — Jinja2 для настоящих конфигов: sshd_config целиком
  - из шаблона с циклом по словарю параметров вместо нескольких
  - задач lineinfile;
  - практика 13 — хранилище: разметка диска, файловые системы
  - и монтирование. Владельцы и права на точке монтирования — та же
  - тема, что setgid здесь;

Учебный центр АйТи Клауд

# Практика 9. Административные задачи

**ЭТА ПРАКТИКА О ТОМ, ЧТО АДМИНИСТРАТОР ДЕЛАЕТ КАЖДЫЙ ДЕНЬ: ПОЛЬЗОВАТЕЛИ И ПРАВА, ПАКЕТЫ И РЕПОЗИТОРИИ, СЛУЖБЫ, МЕЖСЕТЕВОЙ ЭКРАН, ВРЕМЯ, РАСПИСАНИЕ ЗАДАЧ И ПАРАМЕТРЫ ЯДРА.**



современные технологии простым языком

# Обзор практики

- ЭТА ПРАКТИКА О ТОМ, ЧТО АДМИНИСТРАТОР ДЕЛАЕТ КАЖДЫЙ ДЕНЬ: ПОЛЬЗОВАТЕЛИ И ПРАВА, ПАКЕТЫ И РЕПОЗИТОРИИ, СЛУЖБЫ, МЕЖСЕТЕВОЙ ЭКРАН, ВРЕМЯ, РАСПИСАНИЕ ЗАДАЧ И ПАРАМЕТРЫ ЯДРА. МЫ СОБИРАЕМ ОДИН ПЛЕЙБУК, КОТОРЫЙ...

## Что мы сделаем

- Договоримся, что считаем результатом, и разложим файлы практики в рабочий каталог.
- Опишем переменные по семействам ОС.
- Создадим пользователя с паролем и ключом, удалим лишнюю учётную запись вместе с её домашним каталогом.
- Опишем правило sudo файлом в /etc/sudoers.d из шаблона и проверим его до того, как оно попадёт на место.
- Поставим набор пакетов на Ubuntu и на Rocky Linux одной лестницей задач и подключим репозиторий на каждом семействе.
- Приведём в порядок службы и добавим свой таймер systemd из шаблона.
- Откроем веб-сервер наружу двумя разными модулями — ufw и firewalld — и разберёмся, почему это одно намерение.
- Зададим часовой пояс, ежедневную задачу cron и параметр ядра.
- Разберём перезагрузку с ожиданием возврата узла.
- Соберём всё в один плейбук admin_tasks.yml и добьёмся того, чтобы повторный прогон дал changed=0.

# Что понадобится

▸ **СТЕНД ИЗ ПРАКТИК 1 И 2: Три управляемые машины:**

```
web1 10.0.2.11 — Ubuntu 26.04 LTS
web2 10.0.2.12 — Ubuntu 26.04 LTS
db1 10.0.2.13 — Rocky Linux 10
```

- Управляющая машина control (10.0.2.5) в инвентаре не описана: ей не управляют, с неё управляют.
- Поэтому «весь парк» в этой практике — три машины, а не четыре.
- ЧТО ДОЛЖНО БЫТЬ ГОТОВО:
  - инвентарь inventory.ini, файл ansible.cfg и каталоги group_vars
  - и host_vars — практика 2;
  - вход по ключу engineer без пароля и sudo без запроса пароля —
  - практика 1;
  - веб-сервер, поставленный плейбуком практики 7: дальше мы будем
  - не ставить его, а описывать его состояние.
- ФАЙЛЫ ПРАКТИКИ лежат в каталоге files/09_admin_tasks:
  - files/09_admin_tasks/02_users.yml — шаг 3
  - files/09_admin_tasks/03_sudoers.yml — шаг 4

# ЧАСТЬ 1. БАЗОВЫЕ ШАГИ: ПОЛЬЗОВАТЕЛИ, ПАКЕТЫ, СЛУЖБЫ, ЭКРАН

-

# ШАГ 1. ЧТО ЗНАЧИТ «ПРИВЕСТИ ПАРК К СОСТОЯНИЮ»

- В практике 4 мы поставили веб-сервер и запустили службу, в практике 7 научились выбирать ветку по семейству ОС.
- Там было одно намерение — «на машине есть работающий веб-сервер».
- Здесь намерений становится десяток, и каждое описывается своей задачей.
- Разница не в синтаксисе, а в том, что теперь плейбук отвечает на вопрос «какой должна быть машина», а не «что сегодня надо сделать».
- Список намерений для нашего парка выглядит так:
  - на машинах есть учётная запись для выкладки с ключом и паролем;
  - лишней учётной записи нет — вместе с её домашним каталогом;
  - правило sudo описано файлом, который перед установкой проверен;
  - набор административных пакетов установлен, репозиторий подключён;
  - веб-сервер запущен и включён в автозапуск;
  - есть свой таймер systemd, который раз в сутки пишет отчёт;
  - на Ubuntu работает ufw, на Rocky — firewalld, порт 80 открыт;
  - часовой пояс, задача cron и параметр ядра заданы;
  - базовая проверка: повторный прогон не меняет ничего.

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: именно этот список, а не команды, попадает в документацию и в систему контроля версий. Прочитав его, коллега понимает, что произойдёт с машинами, не читая YAML.**

▸ **ПОЧЕМУ ИМЕННО ТАК: обратный порядок — «сначала команды, потом описание» — работает, пока машин две. На парке из тридцати машин список из десяти команд уже никто не удержит в голове.**

# ШАГ 2. РАСКЛАДЫВАЕМ ФАЙЛЫ: ПЕРЕМЕННЫЕ ПО СЕМЕЙСТВАМ

- Копируем файлы практики в рабочий каталог курса.
- Каталоги templates и vars копируются целиком — плейбуки ссылаются на них по именам:

```
cd ~/ansible-lab
cp -r /путь/к/репозиторию/files/09_admin_tasks/. .
ls
ansible.cfg group_vars host_vars inventory.ini templates vars
02_users.yml 03_sudoers.yml 04_packages.yml 05_services.yml
06_firewall.yml 07_reboot.yml admin_tasks.yml
```

- Теперь главное в этом шаге — переменные.
- Различия между Ubuntu и Rocky Linux мы не размазываем по задачам условиями when, а собираем в два файла.
- Вот файл для Ubuntu — files/09_admin_tasks/vars/Debian.yml:

```
---
# Переменные семейства Debian (Ubuntu 26.04): web1 и web2.
```

- А вот файл для Rocky Linux — files/09_admin_tasks/vars/RedHat.yml.
- Имена переменных те же, значения другие:
- Файл выбирается по факту ansible_os_family, значение которого — Debian или RedHat.

# ШАГ 3. ПОЛЬЗОВАТЕЛИ И ГРУППЫ

- Первый файл практики — files/09_admin_tasks/02_users.yml:

```
---
# 02_users.yml — практика 9, шаг 3: пользователи, группы и ключ.
# Запуск из каталога ~/ansible-lab:
# ansible-playbook 02_users.yml
- name: Пользователи и группы на парке Linux
hosts: linux
become: true
vars:
```

- Запускаем (вывод сокращён):
- Последняя задача показала ok, а не changed: пользователя intern на машинах нет, удалять нечего.
- Так и должно быть — модуль описал состояние «этой учётной записи нет», а не выполнил команду удаления.
- ▸ **ПАРОЛЬ ПЕРЕДАЁТСЯ ХЭШЕМ, А НЕ ТЕКСТОМ: В /etc/shadow не может лежать пароль: там лежит его криптографический хэш. Поэтому и модулю ansible.builtin.user в параметр password передают хэш. Считать его можно...**
- ▸ **ПОЧЕМУ ИМЕННО ТАК: у фильтра password_hash второй аргумент — соль. Если её не указать, Ansible возьмёт случайную, и хэш при каждом прогоне будет другим. Пароль останется рабочим, но задача каждый раз будет...**
- ▸ **ЧЕГО НЕ ДЕЛАТЬ: не пишите в password сам пароль: password: "Console47". Модуль предупредит, что значение не похоже на хэш, но при некоторых настройках всё равно запишет его в shadow — и вход по нему работать...**
- РАЗБОР ПАРАМЕТРОВ ЗАДАЧИ «Создать пользователя deploy»: name — имя учётной записи, comment — описание в /etc/passwd (нужно для аудита), group — основная группа, groups вместе с append: true — дополнительные...

# ШАГ 4. SUDO ИЗ ШАБЛОНА

- Правило sudo описано файлом в /etc/sudoers.d.
- Плейбук files/09_admin_tasks/03_sudoers.yml:

```
---
# 03_sudoers.yml — практика 9, шаг 4: правило sudo из шаблона.
# Запуск из каталога ~/ansible-lab:
# ansible-playbook 03_sudoers.yml
- name: Правило sudo из шаблона
hosts: linux
become: true
vars:
```

- Содержимое шаблона — files/09_admin_tasks/templates/engineer-nopass.j2:
- ПОЧЕМУ ИМЕННО ТАК:
  - отдельный файл в /etc/sudoers.d вместо правки /etc/sudoers:
  - основной файл остаётся нетронутым, а наше правило можно удалить
  - одной задачей с state: absent;
  - права "0440" и владелец root:root. При других правах sudo молча

# ШАГ 5. ПАКЕТЫ И РЕПОЗИТОРИИ

- Файл files/09_admin_tasks/04_packages.yml целиком:

```
---
# 04_packages.yml — практика 9, шаг 5: пакеты, кэш и репозитории.
# Запуск из каталога ~/ansible-lab:
# ansible-playbook 04_packages.yml
- name: Пакеты и репозитории
hosts: linux
become: true
pre_tasks:
```

- ▸ **РАЗБОР: apt и dnf — разные модули, и это не наша прихоть: у каждого свой пакетный менеджер. Задачи-близнецы с одинаковыми параметрами здесь честнее, чем попытка написать одну задачу на два семейства: видно,...**
- КОГДА ПРИГОДИТСЯ ansible.builtin.package: когда задача делает одно и то же и параметры модулей совпадают.
- Мы используем его в шаге 7 для установки пакета межсетевого экрана.
- update_cache И cache_valid_time.
- Пакетные менеджеры работают по локальному списку пакетов, и его надо обновлять. update_cache: true означает «обнови список перед установкой», а без cache_valid_time это лишний поход в интернет при каждом...
- autoremove — уборка зависимостей, которые больше никому не нужны.
- Это единственный режим, где модулю не нужен параметр name: он просто выполняет уборку и сообщает changed только тогда, когда что-то действительно удалил.

# ШАГ 6. СЛУЖБЫ И СВОЙ ТАЙМЕР

- Файл files/09_admin_tasks/05_services.yml:

```
---
# 05_services.yml — практика 9, шаг 6: службы и свой таймер systemd.
# Запуск из каталога ~/ansible-lab:
# ansible-playbook 05_services.yml
- name: Службы и свой таймер systemd
hosts: linux
become: true
pre_tasks:
```

- Два юнита лежат в шаблонах.
- Служба — файл files/09_admin_tasks/templates/timezone-check.service.j2:
- Таймер — files/09_admin_tasks/templates/timezone-check.timer.j2:
- Здесь важно всё:
  - имя таймера и имя службы совпадают до первой точки. Поэтому
  - в таймере достаточно указать Unit=timezone-check.service, но мы
  - пишем его явно — читается лучше;

# ШАГ 7. МЕЖСЕТЕВОЙ ЭКРАН: UFW И FIREWALLD

- Файл files/09_admin_tasks/06_firewall.yml:

```
---
# 06_firewall.yml — практика 9, шаг 7: одно намерение двумя модулями.
# Запуск из каталога ~/ansible-lab:
# ansible-playbook 06_firewall.yml
- name: "Межсетевой экран: веб-сервер доступен снаружи"
hosts: linux
become: true
pre_tasks:
```

- **ОДНО НАМЕРЕНИЕ, ДВА МОДУЛЯ: Намерение формулируется словами: «веб-сервер должен быть доступен снаружи, остальное входящее — закрыто». На Ubuntu это выражается правилами ufw, на Rocky — службой в зоне...**
- ПОЧЕМУ ИМЕННО ТАК:
  - ansible.builtin.package вместо apt и dnf: операция одна и та же,
  - параметры совпадают, различие уже спрятано в переменной;
  - ansible.builtin.service для самой службы: нам нужно только
  - «запущена и включена», а переносимость здесь бесплатна;
  - block с when по семейству — приём из практики 7: ветка из трёх

# ШАГ 8. ВРЕМЯ, CRON, ПАРАМЕТРЫ ЯДРА

- Три задачи, которых не было в шагах: они сразу пишутся в итоговый файл admin_tasks.yml.
- Первая — часовой пояс.
- Значение переменной lab_timezone одно для всего стенда:
  - name: Часовой пояс стенда
  - community.general.timezone:
  - name: "{{ lab_timezone }}"
- Модуль community.general.timezone работает и на Ubuntu, и на Rocky: он знает и про /etc/timezone, и про /etc/sysconfig/clock.
- Данные о часовых поясах лежат в пакете tzdata — он есть в списке admin_packages, поэтому задача не зависит от того, что успел поставить установщик ОС.
- Вторая задача — ежедневный отчёт о свободном месте.
- Здесь важно не столько содержимое, сколько способ:
  - name: "Ежедневная задача cron: отчёт о свободном месте"
  - ansible.builtin.cron:
  - name: "Проверка свободного места"
  - minute: "30"
  - hour: "3"
  - user: root

# ШАГ 9. ПЕРЕЗАГРУЗКА С ОЖИДАНИЕМ

• Файл files/09_admin_tasks/07_reboot.yml:

```
---
# 07_reboot.yml — практика 9, шаг 9: перезагрузка с ожиданием возврата.
# Перезагружаем по одной машине:
# ansible-playbook 07_reboot.yml --limit web1 -e reboot_allowed=true
# Общий для всего парка serial появится в практике 14.
- name: Перезагрузка узла с ожиданием возврата
hosts: linux
become: true
```

▸ **Пока машина не перезагружена, обновление не применено, хотя пакеты стоят.**

• ПОЧЕМУ ИМЕННО ТАК:

– reboot_timeout: 600 — сколько секунд ждать возврата. По умолчанию
– 600, и мы пишем его явно: на медленной виртуалке десяти минут
– может не хватить, и лучше это увидеть в файле, чем вспоминать;
– test_command: /usr/bin/whoami — команда, которой модуль проверяет,
– что узел поднялся и готов принимать задачи. Ждать «просто ответа

# ЧАСТЬ 2. ЧТО ОТЛИЧАЕТ РАБОТУ ОТ УЧЕБНОГО ПРИМЕРА

-

# ШАГ 10. ИТОГОВЫЙ ПЛЕЙБУК ADMIN_TASKS.YML

- Все задачи, которые мы разобрали, собраны в один файл files/09_admin_tasks/admin_tasks.yml.
- Он начинается так:

```
---
# admin_tasks.yml — практика 9: административные задачи на парке Linux.
# Итоговый плейбук практики: пользователи, sudo, пакеты, репозитории,
# службы, межсетевой экран, время, cron и параметры ядра.
#
# Запуск из каталога ~/ansible-lab:
# ansible-playbook admin_tasks.yml
#
```

- Первое отличие от шагов: имена deploy_user, deploy_group, sudo_user вынесены в переменные.
- В отдельных файлах они были прямо в задачах — так короче, когда файл живёт один шаг.
- В итоговом плейбуке те же имена встречаются в четырёх задачах, и менять их в четырёх местах не нужно.
- Дальше идут задачи шагов 3–8 дословно: пользователи, правило sudo, пакеты и репозитории, службы и таймер, межсетевой экран, время, cron и параметр ядра.
- Новых задач три.
- Установка пакетов передаёт список целиком:

# ШАГ 11. ПОВТОРНЫЙ ПРОГОН: ЧТО ЗНАЧИТ CHANGED=0

- Теперь запустите тот же плейбук второй раз, ничего не меняя:

```
ansible-playbook admin_tasks.yml
PLAY RECAP
db1 : ok=13 changed=0 unreachable=0 failed=0 skipped=5
web1 : ok=14 changed=0 unreachable=0 failed=0 skipped=2
web2 : ok=14 changed=0 unreachable=0 failed=0 skipped=2
```

- changed=0 означает, что описание совпало с состоянием машин: ничего приводить в порядок не пришлось.
- Это и есть главный результат практики: плейбук можно запускать в любой момент.
- Числа ok меньше, чем в первом прогоне, — на единицу у web-машин и у db1, а skipped те же.
- Разница как раз про обработчик: он попадает в ok наравне с задачами.
- При первом прогоне на Ubuntu сработал обработчик обновления кэша apt — изменился файл репозитория, — и добавил единицу в ok.
- На втором прогоне файл не менялся, обработчик не вызывался, и его строка из ok пропала.
- Skipped при этом не изменился: в него попадают только задачи и блоки, чей when оказался ложным.
- ЧТО МОЖЕТ ПОКАЗАТЬ CHANGED НА ПОВТОРНОМ ПРОГОНЕ — и это нормально:
  - задача обновления кэша apt, если с прошлого обновления прошло
  - больше часа: кэш действительно обновляется, работа сделана;

# ШАГ 12. ОБНОВЛЕНИЯ, ПЕРЕЗАГРУЗКА ПАРКА И ДОСТУПЫ

• Три решения, которые в учебном примере выглядели бы одной строкой, а в работе требуют отдельного разговора.

▸ **РЕШЕНИЕ ПЕРВОЕ: ОБНОВЛЯТЬ ЛИ ВЕРСИИ. Соблазн написать так:**

– name: Обновить веб-сервер до последней версии
– ansible.builtin.apt:
– name: "{{ web_package }}"
– state: latest

• Строка рабочая.

• Но state: latest означает «поставь самую свежую версию, какая есть в репозитории»: каждый прогон может принести новое поведение и новые ошибки, а плейбук, который «только приводит конфигурацию», становится ещё...

▸ **ПОЧЕМУ ИМЕННО ТАК: state: present описывает версию словами «любая, лишь бы стояла», state: latest — «самая свежая из доступных». Второе это не настройка, а решение об обновлении: его принимает тот, кто знает...**

▸ **ЧЕГО НЕ ДЕЛАТЬ: не ставьте state: latest «чтобы не отставать» на все пакеты сразу (name: "*"). Такой прогон может обновить ядро, библиотеки и базу данных за один заход, а откатывать придётся из снимка.**

▸ **РЕШЕНИЕ ВТОРОЕ: ПЕРЕЗАГРУЗКА ПАРКА. Перезагрузка по одной машине медленная, и рано или поздно кто-то напишет один play на все машины. Так делать нельзя: одновременно перезагруженный парк — это одновременно...**

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: правило не зависит от инструмента: пока не обновлена и не проверена первая машина, вторая не трогается.**

▸ **РЕШЕНИЕ ТРЕТЬЕ: ПАРОЛИ И ДОСТУПЫ. В наших файлах стоит учебный пароль Console47, а соль хэша — прямо в плейбуке. В рабочем проекте так нельзя по двум причинам: пароль виден в репозитории, а смена пароля...**

```
deploy_password: "{{ vault_deploy_password }}"
```

• а сам хэш считают там же, где и остальные секреты.

• Второй момент: /etc/sudoers.d и членство в группе sudo — это не «настройка», а выдача прав.

# Если что-то пошло не так

▸ **ПРОБЛЕМА: No package matching 'apache2' is available**

```
fatal: [db1]: FAILED! => {"changed": false,
"failures": ["No package matching 'apache2' is available"],
"msg": "Failed to install some of the specified packages"}
```

- Причина: задача ставит apache2 на все машины парка, а на Rocky Linux этот пакет называется httpd.
- Модуль честно сообщает, что пакета в подключённых репозиториях нет.
- Порядок проверки:
  - Какое семейство ОС у машины:
  - ansible db1 -m ansible.builtin.setup -a "filter=ansible_os_family"
  - Что лежит в переменной на этой машине:
  - ansible db1 -m ansible.builtin.debug -a "var=web_package"
  - Есть ли пакет в репозиториях Rocky:
  - ansible db1 -b -m ansible.builtin.command -a "dnf info httpd"
- Лечится это не правкой задачи и не вторым when, а переменной по семейству: apache2 в vars/Debian.yml, httpd в vars/RedHat.yml.
- Если на db1 подставился apache2 — файл vars/RedHat.yml не прочитан.

▸ **ПРОБЛЕМА: Missing sudo password**

# Проверка результата

- Практика выполнена, если выполняются все пункты.
- 1.
- Синтаксис всех файлов практики в порядке:

```
ansible-playbook --syntax-check admin_tasks.yml
ansible-playbook --syntax-check 02_users.yml (и остальные)
При успехе команда молчит.
```

- 2.
- Повторный прогон итогового плейбука не меняет ничего:

```
ansible-playbook admin_tasks.yml
web1 : ok=14 changed=0 ...
web2 : ok=14 changed=0 ...
db1 : ok=13 changed=0 ...
```

- 3.
- Пользователь выкладки есть, лишней учётной записи нет:

```
ansible all -b -m ansible.builtin.command -a "id deploy"
```

- 4.

# Что дальше

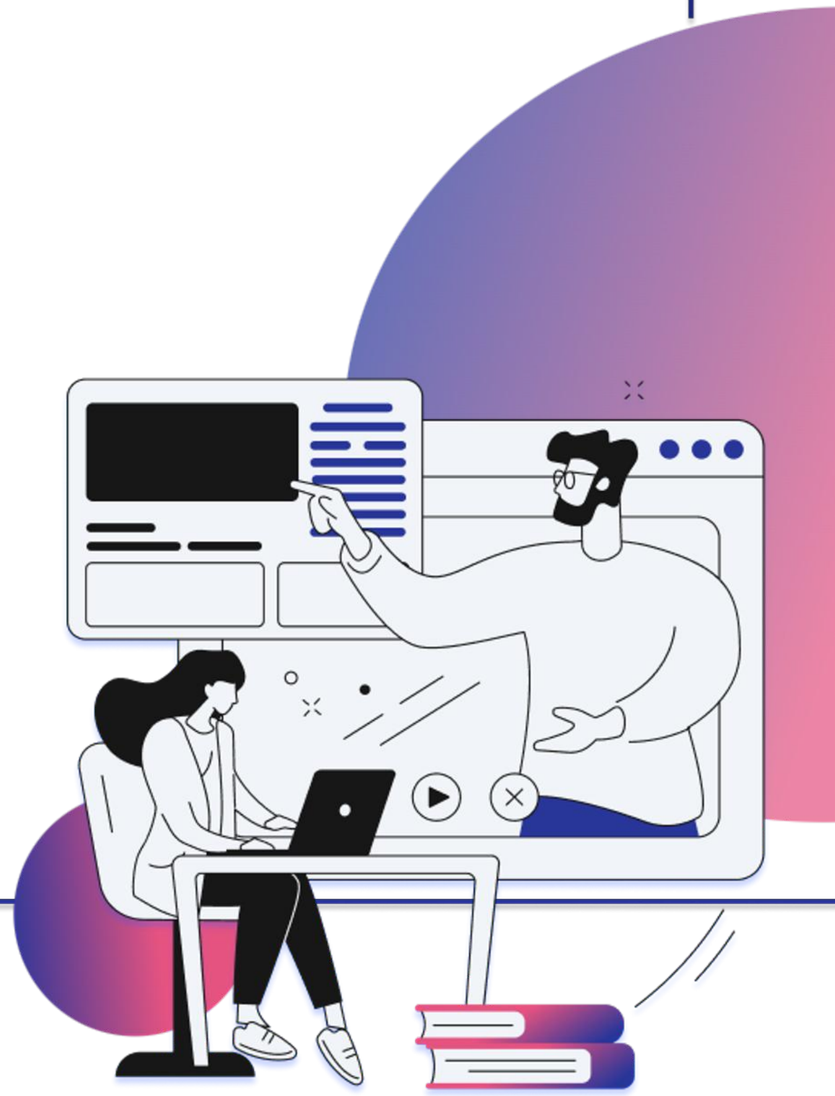
- Далее в курсе:
  - практика 10 — роли: задачи из admin_tasks.yml переезжают в роль
  - web_admin, а переменные по семействам — в vars/main.yml роли;
  - практика 11 — регулярные выражения и точечная правка: строка
  - cron и параметр ядра правятся не шаблоном, а lineinfile;
  - практика 12 — Jinja2 для настоящих конфигов: наш шаблон
  - engineer-nopass.j2 станет одним из многих;
  - практика 14 — сложные проекты: serial вместо --limit при
  - перезагрузке парка, include_tasks вместо копирования задач;
  - практика 16 — качество автоматизации: ansible-lint проверит
  - в том числе и то, что записано в этой практике.
- ПЕРЕД ПЕРЕХОДОМ уберите за собой:
  - снимок ВМ делать не нужно: состояние машин — часть работы;
  - если вы включали перезагрузку, проверьте, что правая часть
  - отчёта снова changed=0 (перезагрузка сбрасывает uptime,
  - но не описание состояния).

Учебный центр АйТи Клауд

# Практика 10. Роли и итоговая работа

**ЭТА ПРАКТИКА — ИТОГ ТРЁХДНЕВНОГО ТРЕКА.**

современные технологии простым языком



# Обзор практики

- ЭТА ПРАКТИКА — ИТОГ ТРЁХДНЕВНОГО ТРЕКА.
- Всё, что делалось по частям в практиках 4–9, собирается здесь в одно: плейбук практики 7 становится ролью web_admin, а роль — проектом, который запускается одной командой на четырёх машинах.

## Что мы сделаем

- Разберём, во что превращается плейбук, когда его копируют под каждый случай, и как роль убирает эти копии.
- Разберём структуру роли и разницу между defaults и vars.
- Создадим роль командой ansible-galaxy role init, перенесём в неё плейбук практики 7 и разберём способы вызова роли.
- Соберём итоговый проект, прогоним его на web1, web2 и db1, добьёмся changed=0 и проверим работу по критериям итоговой работы трёхдневного трека.

# Что понадобится

- СТЕНД ИЗ ПРАКТИК 1–9.
- Четыре машины собраны, вход по ключу работает, ansible.cfg и inventory.ini лежат в ~/ansible-lab:

```
control 10.0.2.5 — управляющая машина, здесь мы всё выполняем
web1 10.0.2.11 — Ubuntu 26.04 LTS
web2 10.0.2.12 — Ubuntu 26.04 LTS
db1 10.0.2.13 — Rocky Linux 10
```

- ЧТО ДОЛЖНО БЫТЬ ГОТОВО ЗАРАНЕЕ:
  - Ansible установлен в ~/ansible-venv (практика 1);
  - коллекции ansible.posix и community.general установлены: роль
  - ставит community.general.ufw и ansible.posix.firewalld;
  - в инвентаре есть группы web, db, linux, debian и redhat
  - (практика 2);
  - плейбук практики 7 мог уже прогоняться: apache2, httpd
  - и открытые порты роли не мешают.
- ФАЙЛЫ ИЗ КАТАЛОГА КУРСА лежат в files/10_roles — это готовый проект целиком; каждый файл напечатан в шаге, где разбирается.
- ▸ **ВРЕМЯ: четыре академических часа.**

# ЧАСТЬ 1. ОТ ПЛЕЙБУКА К РОЛИ

-

# ШАГ 1. ЗАЧЕМ РОЛИ: ВОСЕМЬ ЗАДАЧ, КОТОРЫЕ КОПИРУЮТ

- Плейбук практики 7 работает на двух семействах ОС.
- Сокращённо он выглядит так (целиком — files/07_conditions_loops/02_two_families.yml):
  - name: Веб-сервер на Ubuntu и Rocky Linux двумя задачами с when
  - hosts: web:db
  - tasks:
  - name: Установить apache2 на Ubuntu
  - ansible.builtin.apt:
  - name: apache2
  - when: ansible_os_family == "Debian"
  - name: Установить httpd на Rocky Linux
  - ansible.builtin.dnf:
  - name: httpd
  - when: ansible_os_family == "RedHat"
  - ...и ещё шесть задач: страница, служба, межсетевой экран.
- Теперь нужно развернуть сайт ещё на двух машинах — файл копируют: cp 02_two_families.yml 05_new_web.yml, потом правят hosts.
- Через месяц файлов пять, и они разошлись: одна задача живёт в пяти местах, страница где-то кладётся модулем copy, а где-то template.

# ШАГ 2. СТРУКТУРА РОЛИ: ЧТО КЛАДЁТСЯ КУДА

- Имена подкаталогов роли — это интерфейс: Ansible ищет задачи только в tasks/main.yml, шаблоны — только в templates/, файлы — только в files/.

```
roles/web_admin/
├── tasks/main.yml задачи роли, выполняются по порядку
├── handlers/main.yml обработчики: перезапуск службы
├── defaults/main.yml переменные по умолчанию
├── vars/main.yml переменные, которые роль фиксирует
├── files/ файлы роли для модуля copy (style.css)
├── templates/index.j2 шаблоны Jinja2 для модуля template
├── meta/main.yml автор, лицензия, платформы, зависимости
```

- Что кладётся куда:
  - tasks/main.yml — единственное место, откуда Ansible берёт
  - задачи роли: нет файла — роль вызовется и ничего не сделает,
  - без единой ошибки;
  - handlers/main.yml — обработчики: без него notify из задачи
  - не сработает и прогон закончится ошибкой;
  - defaults/main.yml и vars/main.yml — переменные роли (шаг 3);

# ШАГ 3. DEFAULTS ПРОТИВ VARS: ЧТО ПЕРЕОПРЕДЕЛЯЮТ, А ЧТО ФИКСИРУЮТ

- У роли два файла переменных, и различаются они приоритетом:
  - defaults/main.yml — значения, которые ПРЕДПОЛАГАЕТСЯ
  - ПЕРЕОПРЕДЕЛЯТЬ. Приоритет самый низкий: play, group_vars,
  - host_vars, параметры роли и --extra-vars сильнее;
  - vars/main.yml — значения, которые роль ФИКСИРУЕТ. Приоритет
  - выше, чем у play, group_vars и host_vars; сильнее их только
  - параметры роли и --extra-vars.
- У нас в defaults лежат порт сайта, каталог сайта и заголовок страницы — то, что вызывающий вправе поменять.
- В vars лежат имена пакетов и служб: apache2 на Ubuntu и httpd на Rocky Linux — это не «значение по умолчанию», а факт операционной системы.

▸ **ПРОВЕРКА: поставьте в group_vars/web.yml строку web_admin_index_file: web.html — роль положит страницу в файл web.html, потому что defaults переопределяются. Так и проверяют, что роль берёт значение именно из...**

▸ **ЧЕГО НЕ ДЕЛАТЬ: не храните в vars роли секреты: значение из vars нельзя переопределить ни group_vars, ни host_vars, поэтому секрет придётся править в самой роли, общей для всех машин. Секреты живут в Vault...**

# ШАГ 4. СОЗДАЁМ РОЛЬ: ANSIBLE-GALAXY ROLE INIT

- Скелет роли создаёт сам Ansible:

```
cd ~/ansible-lab/10_roles
ansible-galaxy role init roles/web_admin
- Role roles/web_admin was created successfully
```

- Появилось: README.md, defaults/, files/, handlers/, meta/, tasks/, templates/, tests/ и vars/ — каталоги с файлами-заглушками.
- ЧТО ЗАПОЛНЯЕТСЯ, А ЧТО УДАЛЯЕТСЯ:
  - tasks, handlers, defaults, vars и meta — заполняем своим
  - содержимым;
  - templates/ и files/ — пустые каталоги: кладём index.j2
  - и style.css;
  - README.md — переписываем: заготовка Galaxy состоит
  - из разделов-заглушек;
  - tests/ — удаляем (rm -rf tests): каталог остался от времён,
  - когда роли проверяли запуском tests/test.yml в Travis CI.
  - Сегодня для этого есть molecule (практика 16).

▸ **ВАЖНО: путь roles/web_admin выбран не случайно. Ansible ищет роли в каталоге roles рядом с плейбуком, поэтому site.yml сможет написать просто role: web_admin. Список мест поиска меняется переменной roles_path...**

# ШАГ 5. ПЕРЕНОСИМ ПЛЕЙБУК ПРАКТИКИ 7 В РОЛЬ: ЗАДАЧИ

- Файл roles/web_admin/tasks/main.yml целиком:

```
---
# Задачи роли web_admin: веб-сервер и учебная страница сайта.
# Порядок повторяет плейбук практики 7: пакет, страница, служба,
# межсетевой экран. Имена пакетов, служб и межсетевых экранов
# берутся из vars/main.yml по факту ansible_os_family.
- name: Проверить, что семейство ОС поддерживается ролью
ansible.builtin.assert:
that:
```

- Что изменилось по сравнению с плейбуком практики 7:
  - имена пакетов и служб ушли в vars/main.yml и собираются
  - задачей set_fact по факту ansible_os_family: в роли нет
  - ни одного when по семейству ОС;
  - страница кладётся модулем template, а не copy: на каждой
  - машине она своя;
  - в начале роли стоит ansible.builtin.assert: на незнакомом

# ШАГ 6. ШАБЛОН И ФАЙЛЫ РОЛИ

- Страница сайта — шаблон, оформление — файл.
- Файл roles/web_admin/templates/index.j2:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="utf-8">
<title>{{ web_admin_site_title }} — {{ inventory_hostname }}</title>
<link rel="stylesheet" href="{{ web_admin_style_file }}">
</head>
<body>
```

- Файл style.css из каталога files/ роли:

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: страница из шаблона — видимая проверка автоматизации: имя машины, адрес, семейство ОС и владелец подставляются Jinja2 на управляющей машине, поэтому index.j2 шаблон. В style.css подстановок...**

▸ **ВАЖНО: строка {{ ansible_managed }} — метка «Ansible managed», её подставляет сам Ansible. Не заменяйте её датой и временем: почему — разобрано в секции «Если что-то пошло не так».**

# ШАГ 7. ПЕРЕМЕННЫЕ РОЛИ: DEFAULTS И VARS

- Значения по умолчанию — файл roles/web_admin/defaults/main.yml:

```
---
# defaults/main.yml – значения по умолчанию роли web_admin.
# Приоритет у них самый низкий: play, group_vars, host_vars
# и --extra-vars их перекрывают.
# Разворачивать ли страницу сайта: для группы web значение
# переопределено в group_vars/web.yml.
web_admin_deploy_site: false
# Что написано на странице.
```

- Значения, которые роль фиксирует, — roles/web_admin/vars/main.yml:

▸ **ПРОВЕРКА: после шага 11 посмотрите, какие переменные дошли до узла:**

# ШАГ 8. ОБРАБОТЧИКИ В РОЛИ: LISTEN И FLUSH_HANDLERS

- Файл roles/web_admin/handlers/main.yml:

```
---
# handlers/main.yml — обработчики роли web_admin.
#
# Задачи роли уведомляют не обработчик по имени, а тему listen:
# у одной темы может быть несколько обработчиков, и тогда
# по одному notify срабатывают оба, в порядке объявления.
#
# Имя обработчика начинается с имени роли: имена обработчиков
```

- **ПОЧЕМУ ИМЕННО ТАК: имена обработчиков, в отличие от имён задач, общие для всего play. Если две роли объявят обработчик с именем «Перезапустить веб-сервер», notify из первой роли может запустить обработчик...**
- В выводе имя роли печатается дважды — один раз как контекст роли, второй раз как часть имени обработчика:
- **ВАЖНО: обработчик срабатывает только тогда, когда задача действительно что-то изменила. Если шаблон не менялся, notify не отправляется, перезапуска нет.**
- ЧТО ДЕЛАЕТ FLUSH_HANDLERS.
- Обработчики ждут конца play; если службу нужно перезапустить раньше, в tasks ставят ansible.builtin.meta: flush_handlers.

# ШАГ 9. ВЫЗОВ РОЛИ: ЧЕТЫРЕ СПОСОБА И РАЗНИЦА СТАТИКИ

- **СПОСОБ 1: Ключ roles в play — то, что написано в нашем site.yml:**

```
roles:
- role: web_admin
```

- **СПОСОБ 2: Роль с параметрами: тот же ключ, запись через role и словарь vars. Параметры роли сильнее её defaults и vars:**

```
roles:
- role: web_admin
vars:
web_admin_deploy_site: true
```

- **СПОСОБ 3: when на роли: роль выполнится только на части машин:**

```
roles:
- role: web_admin
```

- **СПОСОБ 4: ansible.builtin.include_role — вызов из задач:**
- **ЧЕМ СТАТИКА ОТЛИЧАЕТСЯ ОТ ДИНАМИКИ: Ключ roles, import_role и import_tasks разбираются при чтении плейбука: Ansible видит все задачи роли заранее, поэтому работают --list-tasks и теги внутри роли. include_role...**

# ШАГ 10. META/MAIN.YML И README РОЛИ

- Файл roles/web_admin/meta/main.yml:

```
---
# meta/main.yml — описание роли. Читают его двое: ansible-galaxy
# (когда роль публикуют или устанавливают) и сам Ansible (когда
# роль вызывают — ради раздела dependencies).
galaxy_info:
role_name: web_admin
author: engineer
description: Веб-сервер и учебная страница сайта на Ubuntu и Rocky Linux
```

- Файл roles/web_admin/README.md:

▸ **ВАЖНО: зависимость — это не «вызвать роль перед своей», а «моя роль невозможна без той». Если перед вашей ролью просто нужно что-то выполнить, зависимость лучше не использовать: она сработает и тогда, когда...**

# ЧАСТЬ 2. ИТОГОВАЯ РАБОТА ТРЕКА «3 ДНЯ»

-

# ШАГ 11. СОБИРАЕМ ПРОЕКТ: ИНВЕНТАРЬ, GROUP_VARS, SITE.YML

- Проект — это роль плюс всё, что нужно для её запуска:

```
mkdir -p ~/ansible-lab/10_roles
cp -r files/10_roles/. ~/ansible-lab/10_roles/
cd ~/ansible-lab/10_roles
```

- Инвентарь — файл inventory.ini:

```
# Инвентарь учебного проекта практики 10. Формат INI.
#
# Группы по назначению (web, db) и группы по семейству ОС
# (debian, redhat) — разные вещи. Группа linux собирает
# все управляемые узлы стенда.
```

- Настройки — файл ansible.cfg:
- Переменные всего проекта — group_vars/all.yml:
- Переменные группы web — group_vars/web.yml:
- И точка входа — site.yml:
- И описание проекта — файл files/10_roles/README.md:

▸ **ПРОВЕРКА: посмотрите, что получилось, до запуска:**

# ШАГ 12. ПЕРВЫЙ ПРОГОН: РАЗБИРАЕМ ВЫВОД

```
ansible-playbook site.yml
PLAY [Веб-сервер и сайт на узлах стенда]
TASK [web_admin : Установить веб-сервер]
changed: [web1]
changed: [web2]
changed: [db1]
TASK [web_admin : Разложить главную страницу из шаблона]
changed: [web1]
```

- Вывод сокращён; звёздочки-разделители после имени задачи опущены: звёздочки запрещены правилами оформления текстовых практик.
- Счётчики ok и skipped здесь не приводим: ok зависит от того, считать ли задачи внутри пропущенного блока по одной, а skipped — от того же.
- Что именно пропущено, видно по выводу и сказано ниже словами.
- Числа changed тоже зависят от состояния машин: у нас к началу практики apache2, httpd и правила межсетевого экрана уже на месте (их поставила практика 9), поэтому часть задач отвечает ok, а не changed.
- Обработчики в первом прогоне выполнялись, поэтому попали и в ok, и в changed.
- Как складываются ok и skipped, подробно разобрано в практике 9, шаг 10.

▸ **ЧТО ЗДЕСЬ ЧИТАТЬ: задачи помечены именем роли (web_admin : ...) — префикс Ansible добавляет сам; на web1 и web2 пропускается ветка Rocky Linux, а на db1 — ветка Ubuntu (правила ufw) и весь блок со страницей...**

▸ **ПРОВЕРКА: если apache2 остался на узле от практики 9, у задачи установки пакета changed не будет — и это нормально.**

# ШАГ 13. ВТОРОЙ ПРОГОН: CHANGED=0

- Запустите тот же плейбук ещё раз:

```
ansible-playbook site.yml
PLAY RECAP
db1 : changed=0 failed=0
web1 : changed=0 failed=0
web2 : changed=0 failed=0
```

- changed=0 на всех трёх машинах — главный признак того, что роль описывает состояние, а не выполняет действия.
- Обработчиков во втором прогоне нет: шаблон не изменился, значит уведомлять службу не о чем.
- Счётчики ok и skipped не приводим по той же причине, что и в первом прогоне: пропущенные ветки перечислены словами в шаге 12.

▸ **ПРОВЕРКА: если changed не ноль, найдите в выводе задачу со строкой changed: [имя]: она и есть причина.**

# ШАГ 14. ПРОВЕРКА ПО КРИТЕРИЯМ ИТОГОВОЙ РАБОТЫ

- Сайт отдаётся с обеих Ubuntu-машин:

```
curl -s http://10.0.2.11 | grep '<h1>'
curl -s http://10.0.2.12 | grep '<h1>'
<h1>Веб-узел учебного стенда Ansible</h1>
```

- Службы включены в автозапуск, порты открыты средствами Ansible:

```
ssh engineer@web1 'systemctl is-enabled apache2'
enabled
ssh engineer@db1 'systemctl is-enabled httpd'
enabled
ssh engineer@web1 'sudo ufw status | grep 80'
```

- КРИТЕРИИ ИТОГОВОЙ РАБОТЫ (раздел 6 программы трека «3 дня»):
  - роль создана и структурирована: есть tasks, handlers, defaults,
  - templates и meta;
  - сайт работает на управляемых Ubuntu-узлах: curl отдаёт
  - страницу из шаблона;
  - служба включена в автозапуск: systemctl is-enabled apache2

# ШАГ 15. ГДЕ РОЛЬ ПЕРЕИСПОЛЬЗУЕМА, А ГДЕ ВСЁ ЕЩЁ «ПОД СТЕНД»

- Переиспользуемая часть роли:
  - задачи: они не знают ни имён машин, ни адресов;
  - шаблон index.j2: он работает на любой машине парка;
  - переменные: имена пакетов и служб взяты из фактов узла;
  - обработчики: они не привязаны к конкретной службе.
- Часть, которая всё ещё «под стенд»: заголовок страницы говорит про учебный стенд, владелец машины — про engineer, а в group_vars/web.yml зашито, что сайт нужен именно группе web нашего инвентаря.
- Роль открывает порт 22, потому что так устроен стенд, и делает сразу две работы — веб-сервер и межсетевой экран.
- ▸ **ЧЕГО НЕ ДЕЛАТЬ: не дописывайте роль под конкретную машину — ни условия inventory_hostname == "web1", ни отдельного значения в vars для одного узла. Как только роль узнала имя хоста, её нельзя применить ко...**

# Если что-то пошло не так

▸ **ПРОБЛЕМА: ERROR! the role 'web_admin' was not found in ...**

```
ERROR! the role 'web_admin' was not found in /home/engineer/ansible-lab/10_roles/roles:/home/engineer/.ansi...
```

• Причин две: роль лежит не там, где Ansible её ищет — места поиска перечислены в самом сообщении, — или каталог роли назван иначе, чем написано в play.

• Проверяйте по порядку: сначала куда Ansible смотрит и что лежит в roles/, потом имя роли в site.yml — оно должно совпадать с именем каталога символ в символ.

```
ls -d roles/*/
ansible-config dump | grep -i roles_path
```

▸ **ПРОБЛЕМА: ERROR! 'roles' is not a valid attribute for a Task**

```
ERROR! 'roles' is not a valid attribute for a Task
The offending line appears to be:
tasks:
- roles:
^ here
```

• Ключ roles — ключ play, а не ключ задачи: его вставили внутрь блока tasks или на уровень ниже, чем нужно.

• Проверяйте по порядку: сначала уровень ключа (roles стоит рядом с hosts, become и gather_facts, а не внутри tasks), потом файл целиком.

▸ **ПРОБЛЕМА: переменная роли не переопределяется**

• Симптом: в group_vars/web.yml стоит web_admin_index_file: web.html, а роль кладёт страницу в index.html — и ошибок при этом нет.

# Проверка результата

- Итоговая работа выполнена, если верны все пять пунктов.
- 1.
- Файлы и структура роли на месте:

```
cd ~/ansible-lab/10_roles
find . -type f | wc -l
14
```

- 2.
- Синтаксис в порядке, и роль видит три машины:

```
ansible-playbook --syntax-check site.yml
ansible-playbook --list-hosts site.yml
hosts (3): web1, web2, db1
```

- 3.
- Сайт отдаётся с обеих Ubuntu-машин:

```
curl -s http://10.0.2.11 | grep '<h1>'
curl -s http://10.0.2.12 | grep '<h1>'
```

- 4.

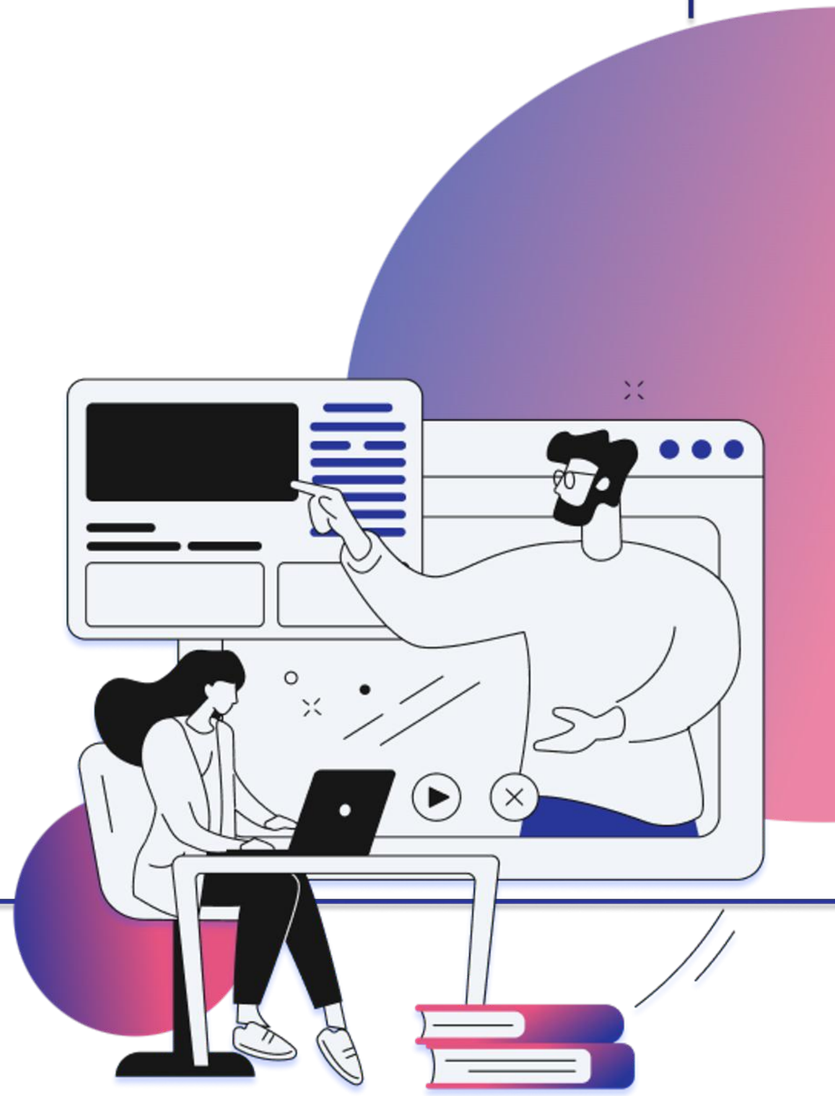
# Что дальше

- Далее в курсе:
  - практика 11 — регулярные выражения и точечная правка: правка
  - готовых конфигов там, где шаблон избыточен;
  - практика 12 — Jinja2 для настоящих конфигов: шаблоны nginx
  - и sshd_config, циклы, фильтры и validate; index.j2 из нашей
  - роли — самый простой случай того, что разбирается там;
  - практика 13 — хранилище: диски, LVM, файловые системы;
  - практика 14 — сложные проекты: import_role и include_role,
  - delegate_to, serial, tags на весь проект;
  - практика 15 — Galaxy и коллекции: geerlingguy.nginx вместо
  - своей роли, requirements.yml, argument_specs для web_admin;
  - практика 16 — качество и динамическая инвентаризация:
  - ansible-lint и molecule на роль web_admin; итоговая работа
  - трека «5 дней».
- ПЕРЕД ПЕРЕХОДОМ уберите за собой:
  - убирать нечего: проект — часть курса, он понадобится

Учебный центр АйТи Клауд

# Практика 11. Регулярные выражения и точечная правка

**ЭТА ПРАКТИКА О ТОМ, КАК ПРАВИТЬ ЧУЖОЙ ФАЙЛ ТОЧНО, А НЕ «ПОХОЖЕ».**



современные технологии простым языком

# Обзор практики

- ЭТА ПРАКТИКА О ТОМ, КАК ПРАВИТЬ ЧУЖОЙ ФАЙЛ ТОЧНО, А НЕ «ПОХОЖЕ».
- МЫ РАЗБЕРЁМ РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ В ОБЪЁМЕ, КОТОРЫЙ НУЖЕН LINEFILE, BLOCKINFILE И REPLACE, И ЗАКОНЧИМ ПРАВИЛОМ ВЫБОРА МЕЖДУ ПРАВКОЙ СТРОКИ И ШАБЛОНОМ.

## Что мы сделаем

- Сначала — синтаксис регулярных выражений: классы символов, якоря, квантификаторы, группы, обратные ссылки и жадность.
- Возьмём ровно то, что встречается в параметрах lineinfile и replace, и ничего сверх.
- Затем разберём, как YAML обрабатывает обратные слеши: почему '\d+' и "\\d+" дают одно и то же, а "\d+" — ошибка разбора плейбука.
- Дальше — правка конфигураций: три параметра sshd одной задачей через loop по списку словарей, backrefs, управляемый блок в /etc/hosts с маркерами, замена серверов времени в chrony.conf, уборка старых копий...
- Закончим правилом, которое важнее синтаксиса: правишь существующий чужой файл — lineinfile; создаёшь свой — template.

# Что понадобится

```
control 10.0.2.5 — управляющая машина, Ubuntu 26.04 LTS
web1 10.0.2.11 — Ubuntu 26.04 LTS
web2 10.0.2.12 — Ubuntu 26.04 LTS
db1 10.0.2.13 — Rocky Linux 10, на нём есть chrony
```

- **ЧТО ДОЛЖНО БЫТЬ СДЕЛАНО РАНЬШЕ: стенд собран, вход по ключу работает и снимок машин сделан (практика 1); в /home/engineer/ansible-lab лежат inventory.ini и ansible.cfg, команды выполняются из него (практика...**
- ФАЙЛЫ ИЗ РЕПОЗИТОРИЯ лежат в files/11_regex_configs: образцы conf/sshd_config.in, conf/chrony.conf.in и conf/resolv.conf.in — это конфигурации, которые мы правим
- плейбуки 01_regexp.yml, 02_backrefs.yml, 03_blockinfile_markers.yml, 04_replace.yml, 05_find_age.yml и 06_validate.yml — по шагам 7, 9, 10, 11, 12 и 13.
- README.md каталога объясняет назначение каждого файла.
- Копировать надо содержимое каталога, а не сам каталог, иначе файлы лягут в ~/ansible-lab/11_regex_configs, а плейбуки ищут conf/ рядом с собой:

```
scp -r files/11_regex_configs/. engineer@10.0.2.5:/home/engineer/ansible-lab/
```

- Если репозиторий уже лежит на управляющей машине, то же самое делается на месте, из каталога ~/ansible-lab:

```
cd ~/ansible-lab
cp -r /путь/к/репозиторию/files/11_regex_configs/. .
```

- **СКОЛЬКО ЭТО ЗАЙМЁТ: около двух академических часов. Без Rocky Linux шаг 11 разбирается по тексту без запуска, остальное работает на Ubuntu.**

# ЧАСТЬ 1. РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ И YAML

-

# ШАГ 1. ЗАЧЕМ РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ В ANSIBLE

- Регулярное выражение — образец текста, а не сам текст.
- Ansible передаёт выражения в Python, поэтому синтаксис здесь питоновский: тот же, что в модуле re.
- У grep и sed диалект другой (шаг 5).
- Мест, где выражения встречаются, немного: lineinfile (regexp, search_string, insertafter, insertbefore), blockinfile, replace (regexp, after, before), find и assert (проверки вида is search).
- Возьмём строку из образца conf/sshd_config.in:
  `#PermitRootLogin prohibit-password`
- Надо включить этот параметр, не завися от того, что написано после имени: ^ — начало строки, #? — знак комментария есть или его нет, PermitRootLogin — имя параметра, \s — пробельный символ после имени.

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: почти вся правка чужих конфигураций — это «включить параметр, который уже есть, но закомментирован».**

▸ **ЧЕГО НЕ ДЕЛАТЬ: не проверяйте выражение сразу плейбуком на живом узле. lineinfile и replace при промахе не падают: они молча дописывают строку или молча ничего не делают. Проверка на копии файла занимает...**

# ШАГ 2. КЛАССЫ СИМВОЛОВ, ЯКОРЯ, ЭКРАНИРОВАНИЕ

- Что значит каждый знак:

```
. любой символ, кроме перевода строки;
\. именно точка, а не «любой символ»;
\d цифра, то же самое, что [0-9];
\w буква, цифра или подчёркивание;
\s пробел или табуляция;
\S любой символ, кроме пробельного;
[abc] один символ из набора: a, b или c;
[^abc] один символ, кроме a, b и c;
```

- Точка, ^, $ и обратный слеш — метасимволы: если они нужны как обычные знаки, их экранируют.

▸ **ГДЕ ЧТО ИЩЕТСЯ: В lineinfile выражение применяется к каждой строке файла по отдельности, поэтому ^ и $ означают начало и конец строки. В replace выражение применяется ко всему файлу целиком, но с включённым...**

▸ **ЭКРАНИРОВАНИЕ НА ЖИВОМ ПРИМЕРЕ: В образце chrony.conf.in имена серверов времени содержат точки:**

- Если написать '^server 0.centos', точка совпадёт с любым символом, и под выражение попадёт заодно '0xcentos' или '0-centos'.
- Правильно экранировать каждую точку:

▸ **ПРОВЕРКА: посмотрите на строку посимвольно до того, как писать выражение: cat -A показывает невидимые символы и пробелы в конце строки — самую частую причину промаха якоря $.**

# ШАГ 3. КВАНТИФИКАТОРЫ, ГРУППЫ, ЖАДНОСТЬ

- Квантификатор говорит, сколько раз повторить предыдущий знак:
  - ноль раз или больше;
  - + один раз или больше;
  - ? ноль раз или один раз;
  - {2,} два раза или больше;
  - {2,4} от двух до четырёх раз.
- Квантификатор относится только к одному предыдущему знаку: \s* — это «сколько угодно пробелов».
- ▸ **ВАЖНО: выражение, которое начинается с { или [, надо брать в кавычки, иначе YAML решит, что начался словарь или список. Запись regexp: '{2,}' не разберётся, а запись regexp: '^{2,}$' разберётся.**
- Скобки не просто группируют: то, что попало в скобки, запоминается, и на это можно сослаться по номеру открытой скобки слева направо — \1, \2.
- Так строка собирается из того, что уже есть в файле:

```
regexp: '^(#?)\s*(LoginGraceTime)\s+\S+$'
line: '\2 30'
```

- Первая группа ловит необязательную решётку, вторая — имя параметра.
- Имя берётся из файла, а не повторяется в плейбуке второй раз.
- ▸ **ЖАДНОСТЬ: Квантификаторы * и + жадные: захватывают как можно больше, и '.*' съест всю строку до конца, включая комментарий. Ленивая версия '.*?' берёт как можно меньше. Вместо .* в середине выражения почти...**
- ▸ **ВАЖНО: обратные ссылки пишутся в одинарных кавычках: line: '\2 30'. В двойных кавычках YAML попытается прочитать \2 как escape-последова- тельность, не найдёт её и остановит разбор плейбука.**

# ШАГ 4. YAML И ОБРАТНЫЕ СЛЕШИ: ОДИНАРНЫЕ ПРОТИВ ДВОЙНЫХ КАВЫЧЕК

- YAML обрабатывает строки по-разному, и это ловит почти всех, кто начинает писать выражения в плейбуках.
- ОДИНАРНЫЕ КАВЫЧКИ — строка берётся как есть, обратный слеш остаётся собой.
- Единственное исключение: одинарная кавычка внутри строки записывается двумя одинарными.
- ДВОЙНЫЕ КАВЫЧКИ — строка обрабатывается как в языках программирования: \n превращается в перевод строки, \t — в табуляцию, \\ — в один обратный слеш.
- Последовательность, которой в этом списке нет, например \d, считается ошибкой разбора.
- Отсюда три записи одного и того же выражения:

```
regexp: '^\s*MaxAuthTries\s+\d+$' в модуль уйдёт ^\s*MaxAuthTries\s+\d+$
regexp: "^\\s*MaxAuthTries\\s+\\d+$" то же самое, но слеши удвоены
regexp: "^\s*MaxAuthTries\s+\d+$" ошибка разбора YAML
```

- То есть '\d+' и "\\d+" — одна и та же строка из четырёх символов, а "\d+" — не строка, а сообщение об ошибке.

▸ **ПОЧЕМУ ИМЕННО ТАК: в курсе принято писать параметры regexp, search_string, insertafter, insertbefore, line, replace и validate в одинарных кавычках. Двойные остаются там, где внутрь подставляется переменная:...**

▸ **ПРОВЕРКА: разобрать плейбук можно и без Ansible, тем же способом, что в практике 3:**

```
python3 -c "import yaml; yaml.safe_load(open('01_regexp.yml'))"
```

# ШАГ 5. ОБРАЗЦЫ КОНФИГОВ И ТРЕНИРОВКА НА КОПИИ

- Править будем три файла.
- Вот они целиком, ровно в том виде, в котором лежат в files/11_regex_configs/conf.
- files/11_regex_configs/conf/sshd_config.in:

```
# Образец drop-in файла для sshd.
# Копируется на узел как /etc/ssh/sshd_config.d/20-sshd-hardening.conf
# задачами ansible.builtin.copy из 01_regexp.yml и 06_validate.yml.
# Все параметры закомментированы: так выглядит файл до первого прогона.
# Содержимое совпадает с напечатанным в 11_regex_configs.txt.
#PermitRootLogin prohibit-password
#PasswordAuthentication yes
#MaxAuthTries 6
```

- files/11_regex_configs/conf/chrony.conf.in:
- files/11_regex_configs/conf/resolv.conf.in:
- Скопируйте образцы в /tmp (cp conf/chrony.conf.in /tmp/chrony.conf и так далее для двух других файлов) и проверьте выражения командой grep -nE.
- Эти команды ничего на узлах не меняют:

▸ **ВАЖНО: у grep -E свой диалект: \s он понимает, а \d — нет, вместо цифры он прочитает букву d. Ansible обрабатывает выражения модулем re, и там \d работает как обычно: проверяя на копии, заменяйте \d на [0-9]....**

# ЧАСТЬ 2. ПРАВКА КОНФИГУРАЦИЙ: LINEFILE, BLOCKINFILE, REPLACE

-

# ШАГ 6. ПАРАМЕТРЫ LINEFILE: REGEXP, SEARCH_STRING, INSERTAFTER, ABSENT

- lineinfile — модуль одной строки.
- Он умеет следующее:

```
path файл, который правим. Обязателен;
line строка, которая должна оказаться в файле;
regexp по какому образцу искать существующую строку;
search_string буквальная подстрока для поиска вместо regexp;
state present (по умолчанию) или absent;
insertafter вставить после последнего совпадения выражения;
insertbefore вставить перед последним совпадением выражения;
firstmatch true – insertafter и insertbefore работают
```

- REGEXP ПРОТИВ SEARCH_STRING. regexp — образец, search_string — буквальная подстрока.
- Если в строке есть знаки, которые в выражении что-то значат (точки, скобки, решётки), search_string избавляет от экранирования.
- Обе задачи — с search_string и со state: absent — напечатаны целиком в плейбуке шага 7.
- Главное про search_string: подстрока должна оставаться в строке и после замены.
- Для строки 'Banner none' подходит подстрока 'Banner', а 'Banner none' не подходит: на втором прогоне модуль не нашёл бы строку и дописал её в конец ещё раз.
- Параметры regexp, search_string и backrefs несовместимы: в одной задаче бывает только один из них.

# ШАГ 7. ПЛЕЙБУК 01_REGEXP.YML: ТРИ ПАРАМЕТРА SSHD ОДНОЙ ЗАДАЧЕЙ

- Полный текст совпадает с файлом files/11_regex_configs/01_regexp.yml:

```
---
# 01_regexp.yml — шаги 7 и 8 практики 11.
# Три параметра sshd одной задачей: loop по списку словарей {regexp, line}.
# Файл /etc/ssh/sshd_config.d/20-sshd-hardening.conf создаётся из образца
# conf/sshd_config.in, в котором параметры закомментированы.
# ansible-playbook 01_regexp.yml
# Второй прогон обязан дать changed=0.
- name: "Закрыть sshd: вход root, вход по паролю, число попыток"
```

- Два решения здесь требуют объяснения. force: false у задачи copy — образец кладётся на место только тогда, когда файла ещё нет: иначе плейбук вернёт закомментированные строки и второй прогон никогда не даст...
- ▸ **ВАЖНО: файл 20-sshd-hardening.conf — тот же, что создаёт практика 8: там в нём уже стоят PermitRootLogin no и PasswordAuthentication no. copy с force: false его не перезаписывает, поэтому на стенде после...**
- Запуск:
- Обработчик сработал один раз, хотя notify стоит у пяти задач: он выполняется не на каждый notify, а один раз в конце блока (практика 4).
- ▸ **ВАЖНО: строка AllowGroups sudo ограничивает вход списком групп, и ошибка в ней закрывает SSH всем, кто в эти группы не входит. Оставьте текущую сессию открытой, проверьте вход во втором терминале, и только...**

# ШАГ 8. ИДЕМПОТЕНТНОСТЬ: CHANGED=0 И ЧТО ДЕЛАЕТ НЕТОЧНЫЙ REGEXP

- Запустим тот же плейбук второй раз: PLAY RECAP покажет ok=6 changed=0 на каждой машине — описание состояния совпало с состоянием машины.
- Теперь посмотрим, что делает неточное выражение.
- Заменим в списке параметров выражение для пароля на такое:

```
regexp: '^#?\s*PasswordAuthentication\s+yes'
```

- Оно совпадает с исходной строкой '#PasswordAuthentication yes' в образце, но не совпадает со строкой 'PasswordAuthentication no', которую сам же модуль и записал.
- Результат второго прогона:

```
changed: [web1] => (item=PasswordAuthentication no)
ssh engineer@web1 'grep -n PasswordAuthentication /etc/ssh/sshd_config.d/20-sshd-hardening.conf'
9:PasswordAuthentication no
13:PasswordAuthentication no
```

- Строка добавилась второй раз, и ошибки при этом не было: задача успешна, файл на месте, служба перезапущена.

▸ **ПОЧЕМУ ИМЕННО ТАК: выражение обязано совпадать и с исходным видом строки, и с тем, который получится после замены. Это правило сформулировано в справке модуля одной фразой: при изменении строки выражение...**

▸ **ЧЕГО НЕ ДЕЛАТЬ: не правьте конфигурацию задачей shell с sed -i 's/PermitRootLogin yes/PermitRootLogin no/' по файлу /etc/ssh/sshd_config. У способа четыре изъяна, и все они видны на повторном прогоне....**

# ШАГ 9. ПЛЕЙБУК 02_BACKREFS.YML: ГРУППЫ И ОБРАТНЫЕ ССЫЛКИ

- backrefs: true меняет поведение модуля в двух местах: в line можно ссылаться на группы выражения (\1, \2, \g<1>), а при несовпадении файл не меняется вообще — line не дописывается, задача завершается со...
- Второе свойство важнее первого: lineinfile при промахе дописывает строку в конец, и это правильно, когда задача называется «строка должна быть в файле».
- Когда задача называется «поправить то, что уже есть», дописывание в конец — не то, чего ждут.
- Полный текст совпадает с файлом files/11_regex_configs/02_backrefs.yml:

```
---
# 02_backrefs.yml — шаг 9 практики 11.
# backrefs: true — в line работают группы выражения (\1, \2),
# а при несовпадении файл не меняется: строка не дописывается.
# ansible-playbook 02_backrefs.yml
# Второй прогон обязан дать changed=0.
- name: Править строки, сохраняя то, что в них уже написано
hosts: web
```

- Запуск: задача с совпадением даёт changed, задача с промахом — ok, и в выводе видно changed=False: если строка обязана появиться, backrefs не подходит.

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: backrefs — способ переписать существующую строку, ничего не потеряв: у параметра меняют только значение, а имя, отступы и комментарий в конце строки остаются на месте, потому что их забрали...**

▸ **ЧЕГО НЕ ДЕЛАТЬ: не используйте backrefs как «lineinfile, который ничего не делает при промахе».**

# ШАГ 10. ПЛЕЙБУК 03_BLOCKINFILE_MARKERS.YML: МАРКЕРЫ И УДАЛЕНИЕ БЛОКА

- lineinfile управляет строкой, blockinfile — блоком строк между маркерами:

```
# BEGIN ANSIBLE MANAGED BLOCK
... строки блока ...
# END ANSIBLE MANAGED BLOCK
marker шаблон строки маркера, внутри обязательна {mark};
marker_begin что подставить вместо {mark} в начале, по умолчанию BEGIN;
marker_end что подставить вместо {mark} в конце, по умолчанию END;
block строки блока: значением строкой или через lookup;
state present (по умолчанию) или absent — удалить блок
```

- Полный текст совпадает с файлом files/11_regex_configs/03_blockinfile_markers.yml:
- Запуск.
- Номера строк у вас будут другими: они зависят от того, что уже лежит в /etc/hosts.
- Важно, что маркеров четыре и они парные.
- Второй блок лежит в том же файле, но с другими маркерами: blockinfile нашёл свои границы по строкам 'НАЧАЛО LAB FQDN' и 'КОНЕЦ LAB FQDN' и не тронул первый блок.
- Справка модуля говорит об этом прямо: если в одном файле больше одного управляемого блока, маркер у каждой задачи обязан быть своим.

▸ **ПОЧЕМУ ИМЕННО ТАК: blockinfile — компромисс между lineinfile и template. template переписывает файл целиком и требует, чтобы весь файл был наш. lineinfile правит по одной строке и не умеет сказать «эти пять...**

# ШАГ 11. ПЛЕЙБУК 04_REPLACE.YML: CHRONY.CONF И RESOLV.CONF

- replace не ищет строку целиком, а заменяет все совпадения выражения во всём файле:

```
path файл, в котором идёт замена. Обязателен;
regexp что искать. Обязателен;
replace на что заменить; если параметр не задан, найденное
удаляется целиком;
backup true – сохранить копию файла до замены;
validate проверить результат до подмены файла;
after искать только после строки, найденной по этому выражению;
before искать только до строки, найденной по этому выражению.
```

- Отличие от lineinfile: lineinfile следит за тем, чтобы в файле была нужная строка, replace — за тем, чтобы не было ненужного образца.
- Первый описывает состояние, второй чаще применяют как разовую уборку.
- Главная опасность replace — молчание: если выражение не совпало, задача завершается успешно и ничего не меняет.
- Ниже — ключевые задачи из файла files/11_regex_configs/04_replace.yml; полный текст — в самом файле.
- Первый play работает на db1: образец conf/chrony.conf.in кладётся на место задачей copy, а перед ней задача command с grep -c решает, наш ли это уже файл, — иначе второй прогон вернул бы образец и снова нашёл...
  - name: Заменить все строки внешнего пула, сохранив номер сервера
  - ansible.builtin.replace:

# ШАГ 12. ПЛЕЙБУК 05_FIND_AGE.YML: FIND С PATTERNS, AGE И SIZE

- find ищет объекты по условиям и возвращает список:

```
paths где искать: путь или список путей. Обязателен;
patterns образцы имён вроде '*.bak' — это маски оболочки,
а не регулярные выражения; use_regex: true включает
в patterns регулярные выражения;
age отбор по возрасту: +7d старше недели, -1w младше недели;
size отбор по размеру: +10M больше 10 МБ;
recurse true — искать и во вложенных каталогах; file_type задаёт,
что искать: file, directory, link или any.
```

- В register попадают поля matched (сколько нашлось), files (список с путями) и examined (сколько просмотрено).
- Задача «удалить старое» делается в два шага: сначала find, потом file со state: absent в цикле по найденному.
- Ключевые задачи из files/11_regex_configs/05_find_age.yml:
  - name: Найти копии старше недели
  - ansible.builtin.find:
  - paths: /etc
  - patterns: "{{ backup_patterns }}"

# ШАГ 13. ПЛЕЙБУК 06_VALIDATE.YML: ПРОВЕРКА ДО ПОДМЕНЫ ФАЙЛА

- Ansible не пишет изменённый текст сразу на место: он готовит его во временном файле и только потом подменяет им рабочий конфиг.
- Полный текст совпадает с файлом files/11_regex_configs/06_validate.yml:

```
---
# 06_validate.yml — шаг 13 практики 11.
# validate: конфигурация проверяется во временном файле до подмены.
# Сначала задача с опечаткой падает, потом записывается верная строка.
# Ошибка перехвачена нарочно (ignore_errors) и сразу проверена assert:
# так видно, что validate отклонил файл и файл на узле не изменился.
# ansible-playbook 06_validate.yml
# Второй прогон обязан дать changed=0 у последней задачи записи.
```

- Запуск.
- Первая задача падает нарочно, и текст этой ошибки разобран в разделе «Если что-то пошло не так».
- Итог прогона:

▸ **ЧЕГО НЕ ДЕЛАТЬ: не убирайте validate «чтобы плейбук прошёл» и не прячьте его ошибку через ignore_errors в рабочем плейбуке. Без проверки опечатка применяется, sshd не стартует, и вы теряете машину, потому что...**

# ШАГ 14. КОГДА РЕГУЛЯРНОЕ ВЫРАЖЕНИЕ — ПЛОХОЙ ИНСТРУМЕНТ

- Регулярное выражение — код, у которого нет ни типов, ни тестов.
- Ошибка в нём тихая: задача проходит успешно, а файл остаётся не тем, каким должен быть.
- Правило выбора в курсе звучит так:
  ПРАВИШЬ СУЩЕСТВУЮЩИЙ ЧУЖОЙ ФАЙЛ — LINEFILE;
  СОЗДАЁШЬ СВОЙ ФАЙЛ — TEMPLATE.
- Три вопроса: файл целиком наш? (тогда template собирает его из переменных); файл чужой и принадлежит пакету? (тогда lineinfile, blockinfile или replace); правка разовая или постоянная? (разовая уборка —...

▸ **ВАЖНО: признак того, что выражение перестало быть инструментом правки и стало программой, — длина и вложенные группы. Если выражение длиннее строки, которую вы хотите получить, пишите шаблон: в практике 12...**

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: правку модулем lineinfile видно в отчёте и в системе контроля версий, правку шаблоном — как разницу файлов, а правку через sed внутри shell не видно никак.**

# Если что-то пошло не так

▸ **ПРОБЛЕМА: строка добавляется в файл при каждом прогоне**

```
ssh engineer@web1 'grep -n PasswordAuthentication /etc/ssh/sshd_config.d/20-sshd-hardening.conf'
9:PasswordAuthentication no
13:PasswordAuthentication no
```

• Модуль не нашёл строку по выражению и дописал новую: regexp не совпа- дает с тем, что реально лежит в файле, а задача при этом успешна.

• Порядок проверки: cat -A покажет пробелы и табуляции как есть; не якорь- те лишнего, потому что '^\s*Password\s+no$' не совпадёт со строкой, в конце которой есть пробел

• для точки пишут \.; дубликаты уберите тем же модулем со state: absent; проверьте повторным прогоном (changed=0).

▸ **ЧЕГО НЕ ДЕЛАТЬ: не чините такие дубликаты командой shell: sed -i. У неё нет ни validate, ни резервной копии, ни проверки повторного прогона, поэтому ошибка выражения не проявляется вообще: sed -i меняет файл...**

▸ **ПРОБЛЕМА: backrefs ничего не изменил, хотя задача прошла успешно**

```
TASK [Попробовать выражение, которого в файле нет] *****************
ok: [web1]
```

• Не поломка, а описанное поведение: при backrefs: true несовпадение оставляет файл без изменений.

• Порядок проверки: есть ли параметр в файле (grep -n ChallengeResponseAuthentication ...); совпадает ли выражение со строкой — проверяйте на копии файла, регистр важен

• нужен ли \s* в начале для отступа; если строка обязана появиться, backrefs здесь лишний.

▸ **ПРОБЛЕМА: found unknown escape character 'd'**

```
ERROR! We were unable to read either as JSON nor YAML, these are the errors we got from each:
```

# Проверка результата

- Практика пройдена, если выполняются все пункты.
- Команды выполняются с управляющей машины, из каталога /home/engineer/ansible-lab.
- 1.
- На web1 и web2 настройки sshd действуют (проверка идёт командой

```
sshd -T, а не чтением файла):
ssh engineer@web1 'sudo sshd -T | grep -E "permitrootlogin|passwordauthentication|maxauthtries"'
permitrootlogin no
passwordauthentication no
maxauthtries 3
```

- 2.
- В файле нет дубликатов и нет удалённой строки, а в /etc/hosts два

```
управляемых блока (grep -c 'ANSIBLE MANAGED BLOCK' даёт 2):
ssh engineer@web1 'cat -n /etc/ssh/sshd_config.d/20-sshd-hardening.conf'
7 PermitRootLogin no
```

- 3.
- На db1 chrony переведён на новые серверы, служба работает:

# Что дальше

- Далее в курсе:
  - практика 12 — Jinja2 для настоящих конфигов: sshd_config, chrony.conf
  - и resolv.conf собираются из шаблонов целиком, а выражения нужны
  - только там, где шаблон не подходит;
  - практика 13 — хранилище: разметка диска и монтирование; права
  - и владельцы на точке монтирования — та же тема состояний;
  - практика 16 — качество автоматизации: ansible-lint и molecule
  - разбираются там на своём проекте files/16_quality_inventory
  - и роли web_admin; тот же порядок — сначала линтер, потом
  - повторный прогон с changed=0 — применим и к шести плейбукам
  - этой практики.
- ПЕРЕД ПЕРЕХОДОМ уберите учебную копию резолверов: команда уборки приведена в разделе «Проверка результата».

Учебный центр АйТи Клауд

# Практика 12. Jinja2 для настоящих конфигов

**ЭТА ПРАКТИКА — ПРЯМОЕ ПРОДОЛЖЕНИЕ ШЕСТОЙ.**



современные технологии простым языком

# Обзор практики

- ЭТА ПРАКТИКА — ПРЯМОЕ ПРОДОЛЖЕНИЕ ШЕСТОЙ.
- Там мы собрали из шаблона страницу сайта и motd.
- Здесь шаблоны становятся рабочим инструментом: из них собираются конфигурации nginx и sshd, resolv.conf и chrony.conf — файлы, которые читает служба, а не человек.
- Ошибка в таком файле означает не некрасивую страницу, а упавший веб-сервер или потерянный доступ по SSH.
- Поэтому здесь же разбираются проверка конфигурации до подстановки, управление пробелами и границы применимости lookup.

## Что мы сделаем

- Соберём из переменных четыре настоящие конфигурации: виртуальный хост nginx, upstream с циклом по бэкендам, sshd_config циклом по словарю параметров, motd, resolv.conf и chrony.conf.
- Каждую проверим до подстановки: nginx -t и sshd -t -f.

# Что понадобится

- Стенд из практики 1, все четыре машины включены:
  `control 10.0.2.5 — управляющая машина, отсюда все команды`
  `web1 10.0.2.11 — Ubuntu 26.04 LTS, на ней nginx на порту 8080`
  `web2 10.0.2.12 — Ubuntu 26.04 LTS`
  `db1 10.0.2.13 — Rocky Linux 10`
- ЧТО ДОЛЖНО БЫТЬ СДЕЛАНО РАНЬШЕ:
  - инвентарь inventory.ini и ansible.cfg из практики 2, вход по ключу;
  - практика 6: настраиваемый факт /etc/ansible/facts.d/site.fact
  - на узлах (из него motd берёт владельца машины);
  - практика 11: drop-in файл /etc/ssh/sshd_config.d/20-sshd-hardening.conf
  - на web1. Это не тот файл, который собирает эта практика: шаблон
  - sshd_config.j2 пишет основной /etc/ssh/sshd_config. Drop-in из практик
  - 8 и 11 остаётся на месте и продолжает действовать — строка Include
  - в нашем шаблоне стоит первой;
  - на control в /etc/hosts добавлено имя сайта web1.ansible-lab.local;
  - практики 9 и 10: веб-сервер поставлен и запущен на всём парке —

# ЧАСТЬ 1. ШАБЛОНЫ ДЛЯ НАСТОЯЩИХ КОНФИГОВ

-

# ШАГ 1. ОТ СТРАНИЦЫ К КОНФИГУ: ЧТО МЕНЯЕТСЯ ПОСЛЕ ПРАКТИКИ 6

- У конфигурации службы цена ошибки выше, чем у страницы сайта: кривая вёрстка — неприятно, нерабочий sshd_config — машина, на которую не зайти.
- Отсюда три привычки, которых не было в практике 6:
  - ВСЕ ЗНАЧЕНИЯ — ИЗ ПЕРЕМЕННЫХ: шаблон — форма, данные снаружи;
  - ПРОБЕЛЫ КОНТРОЛИРУЮТСЯ ЯВНО: строка с {% for %} оставляет
  - в готовом файле свой отступ;
  - КОНФИГУРАЦИЯ ПРОВЕРЯЕТСЯ ДО ПОДСТАНОВКИ: за это отвечает
  - параметр validate модуля template.

▸ **ПОЧЕМУ ИМЕННО ТАК: конфигурация — это данные, а не код. Пока значения лежат в переменных, одна форма годится и для web1, и для стенда.**

# ШАГ 2. ПЕРЕМЕННЫЕ И ВИРТУАЛЬНЫЙ ХОСТ

- Содержимое files/12_jinja_configs/vars/nginx.yml:

```
---
# Переменные виртуального хоста nginx.
# Файл подключается плейбуком 01_nginx.yml через vars_files.
# Порт, который слушает виртуальный хост. 80 занят apache2:
# его поставили практики 9 и 10, и два веб-сервера на одном
# порту не уживаются. Поэтому nginx слушает 8080 — так же,
# как это сделано в практике 15.
nginx_port: 8080
```

- db1 стоит в списке с enabled: false: вывести машину из балансировки — значит поправить одно значение в переменных, а не шаблон.
- Содержимое files/12_jinja_configs/templates/nginx_vhost.j2:

▸ **ВАЖНО: В КОНФИГЕ NGINX ЗНАК $ ПРИНАДЛЕЖИТ NGINX, А {{ }} — JINJA2. Строка proxy_set_header Host $host остаётся как есть: Jinja2 ищет только {{ }} и {% %}, а путаница начинается там, где $ ставят перед {{ }}.**

▸ **ПРОВЕРКА: соберите шаблон на управляющей машине, не трогая web1:**

# ШАГ 3. UPSTREAM: ЦИКЛ ПО БЭКЕНДАМ

- Содержимое files/12_jinja_configs/templates/nginx_upstream.j2:

```
# {{ ansible_managed }}
# Файл собран из шаблона nginx_upstream.j2.
# Бэкенды: {{ nginx_backends | map(attribute='name') | join(', ') }}
# Метки сайтов приходят строкой YAML: шаблон превращает её в данные.
{% set site_tags = nginx_site_tags_raw | from_yaml %}
{% set live_backends = nginx_backends | selectattr('enabled') | list %}
map $host $site_tag {
{% for site, tag in site_tags.items() %}
```

- Здесь четыре приёма: map(attribute='name') | join(', ') собирает имена в строку; from_yaml превращает строку nginx_site_tags_raw в словарь, по которому идёт цикл по парам
- selectattr('enabled') вместе с list отбирает включённые бэкенды (web1 и web2, но не db1); default(1) и default(2) подставляют значения, которых нет в данных.
- ▸ **ПОЧЕМУ ИМЕННО ТАК: без | list фильтр length получил бы не список, а объект-генератор. Для строки в формате JSON есть фильтр from_json.**

# ШАГ 4. ПЛЕЙБУК 01_nginx.yml И ПРОВЕРКА КОНФИГУРАЦИИ

- Содержимое files/12_jinja_configs/01_nginx.yml:

```
---
# Практика 12, шаги 3–5: виртуальный хост nginx из шаблонов.
# Запуск: ansible-playbook 01_nginx.yml --diff
# Повторный прогон должен дать changed=0.
#
# nginx слушает 8080, а не 80: порт 80 на web1 и web2 занят apache2,
# который поставили практики 9 и 10. Пакет nginx ставит ещё и свой
# виртуальный хост по умолчанию, а тот слушает 80, поэтому плейбук
```

- Две задачи про виртуальный хост по умолчанию стоят здесь не для красоты.
- Пакет nginx ставит свой виртуальный хост, и на Ubuntu он лежит ссылкой /etc/nginx/sites-enabled/default, а на Rocky — файлом /etc/nginx/conf.d/default.conf.
- Оба слушают 80 — тот самый порт, который занят apache2 из практик 9 и 10.
- Пока чужой хост на месте, nginx не запустится: «bind() to 0.0.0.0:80 failed (98: Address already in use)».
- Наш виртуальный хост слушает 8080, поэтому хост по умолчанию убирается, а не правится.
- Первый прогон меняет семь задач — установку пакета, снятие чужого виртуального хоста, правило межсетевого экрана для 8080, каталог, страницу, upstream и свой виртуальный хост — и один раз запускает обработчик.
- Второй обязан дать changed=0.

# ШАГ 5. SSHD_CONFIG: ЦИКЛ ПО СЛОВАРЮ ПАРАМЕТРОВ

- Содержимое files/12_jinja_configs/vars/sshd.yml:

```
---
# Параметры sshd_config. Словарь: имя параметра — значение.
# Шаблон печатает его циклом по sshd_params.items(), поэтому порядок
# строк в файле совпадает с порядком ключей здесь. YAML порядок ключей
# сохраняет, и это важно: sshd_config читается сверху вниз, а при
# повторе одного параметра побеждает первый.
sshd_params:
Port: 22
```

- Содержимое files/12_jinja_configs/templates/sshd_config.j2:
- Цикл идёт по sshd_params.items() и печатает ключ и значение; фильтр format выравнивает ключ по ширине 30 символов, а строки Include, Subsystem и AllowUsers стоят вне цикла.
- ЧЕГО НЕ ДЕЛАТЬ: не заменяйте sshd_config целиком на удалённой машине без второго сеанса SSH и способа отката: наш шаблон не содержит ни HostKey, ни AcceptEnv.

# ШАГ 6. ПЛЕЙБУК 02_sshd.yml: validate: sshd -t -f %s

- Содержимое files/12_jinja_configs/02_sshd.yml:

```
---
# Практика 12, шаги 6–7: sshd_config на web1 (Ubuntu) и db1 (Rocky).
# Запуск: ansible-playbook 02_sshd.yml --diff
# Перед запуском откройте второй сеанс SSH к узлам: шаблон
# заменяет файл целиком.
- name: "Настроить sshd из шаблона со словарём параметров"
hosts: linux
become: true
```

- validate здесь проверяет полный конфигурационный файл: модуль записывает результат шаблона во временный файл, подставляет его путь вместо %s, запускает sshd -t и только при успехе меняет файл на узле.

▸ **ПРОВЕРКА: обработчик один, а имя службы разное — ssh на Ubuntu и sshd на Rocky (значение sshd_service из файла переменных):**

▸ **ЧЕГО НЕ ДЕЛАТЬ: не проверяйте sshd_config глазами как текст: файл может быть верным и не давать результата, если параметр перекрыт drop-in файлом выше по тексту.**

# ШАГ 7. MOTD ИЗ ФАКТОВ И СПИСКА МАШИН

- Содержимое files/12_jinja_configs/templates/motd.j2:

```
{#- Шаблон /etc/motd. Собирается на каждой машине группы linux. -#}
{{ ansible_managed }}
================================================================
{{ inventory_hostname | upper }} — учебный стенд курса Ansible
{{ ansible_distribution }} {{ ansible_distribution_version }} ({{ ansible_os_family }})
================================================================
Машина: {{ inventory_hostname }} ({{ ansible_default_ipv4.address }})
Ядер: {{ ansible_processor_vcpus }}
```

- Новые приёмы: адреса соседних машин берутся из hostvars; тест is defined плюс default закрывает случай, когда факта нет; loop.last не ставит запятую после последней машины.
- ЧЕГО НЕ ДЕЛАТЬ: не ставьте в конфиг значения, которые меняются сами (дату сборки, время работы машины): файл будет считаться изменённым на каждом прогоне.

# ШАГ 8. ПЛЕЙБУК 03_motd.yml: MOTD, RESOLV.CONF, CHRONY

- Содержимое files/12_jinja_configs/03_motd.yml:

```
---
# Практика 12, шаги 8–9: motd, resolv.conf и chrony.conf из шаблонов.
# Запуск: ansible-playbook 03_motd.yml --diff
- name: "Разложить /etc/motd на всех машинах группы linux"
hosts: linux
become: true
tasks:
- name: "Собрать /etc/motd из шаблона motd.j2"
```

- Три play: первый раскладывает motd на три узла группы linux; второй настраивает resolv.conf; третий ограничен группой redhat, потому что на Rocky Linux chrony есть по умолчанию.
- **ВАЖНО: ПЕРЕД ВТОРЫМ PLAY ПОСМОТРИТЕ СВОЙ /etc/resolv.conf. Адрес 10.0.2.3 — это DNS виртуальной сети стенда: если у вас другие адреса, поправьте resolv_nameservers, иначе перестанут разрешаться имена.**

# ЧАСТЬ 2. ЧТО ОТЛИЧАЕТ РАБОТУ ОТ УЧЕБНОГО ПРИМЕРА

-

# ШАГ 9. RESOLV.CONF И CHRONY.CONF ИЗ ПЕРЕМЕННЫХ

- Содержимое files/12_jinja_configs/templates/resolv.conf.j2:

```
# {{ ansible_managed }}
# Файл собран из шаблона resolv.conf.j2.
# Значения — переменные resolv_*, заданные в плейбуке 03_motd.yml.
{% for server in resolv_nameservers %}
nameserver {{ server }}
{% endfor %}
{% if resolv_search is defined and resolv_search | length > 0 %}
search {{ resolv_search | join(' ') }}
```

- Содержимое files/12_jinja_configs/templates/chrony.conf.j2:
- Два условия соединены союзом and: resolv_search должна быть определена И непуста — так не появится строка search без значений.
- Фильтр lower приводит семейство ОС к нужному регистру.
- Пустой список chrony_servers означает, что блок server не печатается: условие стоит снаружи цикла.

# ШАГ 10. TRIM_BLOCKS И LSTRIP_BLOCKS: ПУСТЫЕ СТРОКИ В КОНФИГЕ

- Это шаг, ради которого стоит прочитать практику целиком.
- Возьмём маленький шаблон с циклом и отступами:

```
upstream demo {
{% for backend in nginx_backends %}
server {{ backend.address }};
{% endfor %}
}
```

▸ **ВАРИАНТ 1: trim_blocks: false, lstrip_blocks: false — значения по умолчанию самого Jinja2:**

```
upstream demo {
server 10.0.2.11:8080;
server 10.0.2.12:8080;
```

- Пустые строки появились потому, что перевод строки после {% for %} и {% endfor %} остался в выводе.

▸ **ВАРИАНТ 2: trim_blocks: true, lstrip_blocks: false — значения по умолчанию модуля template:**

- Пустых строк нет, но отступ первой строки поехал, а последняя строка — четыре пробела без текста: отступы перед {% for %} и {% endfor %} остались в выводе и сложились с отступами тела цикла.

▸ **ВАРИАНТ 3: trim_blocks: true и lstrip_blocks: true — то, что стоит во всех задачах этой практики — даёт ровно тот файл, который нужен:**

▸ **ПОЧЕМУ ИМЕННО ТАК: trim_blocks убирает перевод строки после блока {% %}, lstrip_blocks — пробелы в начале строки перед блоком. Первый в модуле template включён по умолчанию, второй — нет, и именно из-за него в...**

# ШАГ 11. ФИЛЬТРЫ И ТЕСТЫ, КОТОРЫЕ НУЖНЫ В КОНФИГАХ

- Фильтры в шаблонах встречаются постоянно: важно понимать, что фильтр берёт слева и что отдаёт справа.
- Ниже — фрагмент плейбука для проверки фильтров.
  - name: "Показать работу фильтров и тестов"
  - hosts: web1
  - gather_facts: true
  - vars_files:
  - vars/nginx.yml
  - vars/sshd.yml
  - vars:
  - site_defaults:
  - title: "учебный стенд"
  - owner: "engineer"
  - site_override:
  - owner: "course"
  - none_value: null
  - log_path: "/var/log/nginx/access.log"

# ШАГ 12. LOOKUP: FILE, TEMPLATE, ENV

- Третий источник данных — файлы и окружение управляющей машины: текст объявления, тег сборки, имя окружения в CI.
- Содержимое files/12_jinja_configs/data/notice.txt:

```
ВНИМАНИЕ: на стенде идут учебные работы.
Конфигурации машин управляются Ansible: правки руками будут перезаписаны.
Если что-то сломалось — обратитесь к преподавателю, а не правьте файл.
```

- Содержимое files/12_jinja_configs/templates/notice.j2:

```
{#- Шаблон рендерит lookup('template', ...) на управляющей машине. -#}
Объявление для {{ inventory_hostname }} ({{ ansible_os_family }})
Стенд: {{ site_title | default('учебный стенд курса Ansible') }}
Источник: {{ lab_notice_source | basename }}
Строк в тексте: {{ lab_notice_text | default('') | length }}
```

- Содержимое files/12_jinja_configs/04_lookup.yml:
- lookup('file', путь) читает файл с управляющей машины и отдаёт его содержимое строкой
- lookup('env', 'ИМЯ') читает переменную окружения процесса, из которого запущен ansible-playbook (удобно в CI)
- lookup('template', путь) рендерит шаблон на управляющей машине и отдаёт готовый текст, но с переменными той машины, для которой выполняется задача.
- Файл на узле не создаётся.

# ШАГ 13. SLURP: ПРОЧИТАТЬ ФАЙЛ С УЗЛА И СРАВНИТЬ С ШАБЛОНОМ

- Последний вопрос практики: файл на узле точно такой, каким его сделал Ansible?
- Содержимое files/12_jinja_configs/05_slurp.yml:

```
---
# Практика 12, шаг 13: прочитать файл с узла и сравнить с шаблоном.
# Запуск: ansible-playbook 05_slurp.yml
# Плейбук ничего не меняет: changed=0 должен быть и в первый раз.
- name: "Прочитать /etc/motd с узла и сравнить с шаблоном"
hosts: web1
become: true
tasks:
```

- slurp читает файл на узле и отдаёт его в base64, а b64decode возвращает текст; lookup('template', ...) собирает ожидаемое содержимое тем же шаблоном motd.j2; assert сравнивает два текста, а trim снимает вопрос...
- ЧЕГО НЕ ДЕЛАТЬ: не сравнивайте файлы контрольными суммами, если хотите понять, чем они отличаются: md5 скажет «разные», но не покажет где.

▸ **ПРОВЕРКА: прогоните плейбук, испортьте файл на узле и прогоните снова:**

- Первый прогон даёт changed=0, второй падает на assert: конфигурацией на узле управляет Ansible, а не человек.

# ШАГ 14. JINJA2 ВНЕ ШАБЛОНОВ: WHEN, ИМЯ ЗАДАЧИ, LOOP, VARS

- В WHEN — БЕЗ ДВОЙНЫХ ФИГУРНЫХ СКОБОК:

```
when: nginx_backends | selectattr('enabled') | list | length > 0
when: ansible_os_family == "Debian" and nginx_port | int == 8080
```

- В ИМЕНИ ЗАДАЧИ — МОЖНО И НУЖНО: имя печатается в отчёте, и по нему потом ищут, что сломалось:
  - name: "Развернуть виртуальный хост {{ nginx_server_name }}"
- В LOOP — ЧЕРЕЗ ВЫРАЖЕНИЕ, С ФИЛЬТРОМ И LIST: без | list в loop попадёт объект-генератор, и задача упадёт.
  - name: "Создать каталоги площадок"
  - ansible.builtin.file:
  - path: "/var/www/{{ item.name }}"
  - state: directory
  - mode: "0755"
  - loop: "{{ nginx_backends | selectattr('enabled') | list }}"
  - loop_control:
  - label: "{{ item.name }}"
- В VARS — КАК АЛЬТЕРНАТИВА ВЕТВЛЕНИЮ:

```
vars:
```

# ШАГ 15. ШАБЛОН КАК КОД: ГДЕ ХРАНИТЬ И ЧТО НЕ ПОПАДЁТ В ФАЙЛ

- Шаблон — такой же код, как плейбук: он живёт в репозитории, у него есть версии, его читают на ревью.
- КОММЕНТАРИИ.
- Внутри шаблона работают комментарии Jinja2:

```
{#- Шаблон /etc/motd. Собирается на каждой машине группы linux. -#}
```

- В готовый файл такой комментарий не попадает: он для того, кто читает шаблон.
- А обычный комментарий «#» попадает, и это правильно: тот, кто откроет готовый конфиг на узле, увидит, что файл собран Ansible.

▸ **ГДЕ ХРАНИТЬ: шаблоны лежат в templates/ рядом с плейбуком или в роли, данные для lookup — в data/, переменные — в vars/. ЧТО НЕ ДОЛЖНО ПОПАДАТЬ В ШАБЛОН: значения, которые отличаются от машины к машине;...**

▸ **ОГЛАВЛЕНИЕ КАТАЛОГА: files/12_jinja_configs/README.md перечисляет все шаблоны, переменные и плейбуки с указанием шага практики, порядок их запуска и то, на что обратить внимание.**

# Если что-то пошло не так

▸ **ПРОБЛЕМА: nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in use)**

```
nginx: [emerg] bind() to 0.0.0.0:80 failed (98: Address already in use)
nginx: [emerg] still could not bind()
fatal: [web1]: FAILED! => {"changed": false,
"msg": "Unable to restart service nginx: Job for nginx.service failed"}
```

- Порт 80 занят apache2 из практик 9 и 10, а nginx пытается занять его своим виртуальным хостом по умолчанию: на Ubuntu это ссылка /etc/nginx/sites-enabled/default, на Rocky — файл /etc/nginx/conf.d/default.conf.
- Наш виртуальный хост слушает 8080, а чужой убирают задачи «Убрать виртуальный хост по умолчанию» в 01_nginx.yml.
- Порядок проверки:
  - Посмотреть, кто держит порт 80:
  - ssh engineer@web1 'sudo ss -ltnp | grep :80'
  - Проверить, что чужой виртуальный хост снят:
  - ls -l /etc/nginx/sites-enabled/
  - Проверить, что в vars/nginx.yml стоит nginx_port: 8080, а в готовом
  - файле на узле — строка listen 8080.
- То же сообщение видно в journalctl -u nginx: пакет nginx запускает службу сразу после установки, ещё до того, как плейбук уберёт чужой виртуальный хост.

▸ **ПРОБЛЕМА: AnsibleUndefinedVariable: 'nginx_root' is undefined**

# Проверка результата

- Практика выполнена, если выполняются все пункты.
- 1.
- Виртуальный хост собран из шаблона: nginx -t на web1 отвечает
  «configuration file /etc/nginx/nginx.conf test is successful».
- 2.
- Сайт отвечает по имени, и именно на порту 8080:

```
curl -s http://web1.ansible-lab.local:8080/
<h1>web1</h1>
```

  Порт 80 nginx не слушает: там apache2 из практик 9 и 10. Поэтому по адресу http://10.0.2.11/ ответит страница практики 10, а не страница практики 12. Порт 8080 открыт в межсетевом экране самим плейбуком: без этого правила ufw не пропустил бы запрос с control.

- 3.
- Путь /api/ уходит в upstream, а приложения на стенде нет:

```
curl -s -o /dev/null -w '%{http_code}\n' http://web1.ansible-lab.local:8080/api/
```

- 4. sshd_config разложен на два семейства:

# Что дальше

- Далее в курсе:
  - практика 13 — хранилище: диски, LVM, файловые системы;
  - практика 14 — сложные проекты: шаблоны переедут в роли;
  - практика 15 — Galaxy и системные роли RHEL: сравним свой
  - шаблон nginx с ролью geerlingguy.nginx — она встанет на тот же
  - порт 8080 и рядом с apache2 на 80.
- УБЕРИТЕ ЗА СОБОЙ только временные файлы проверок и учебное объявление: конфигурации нужны следующим практикам. nginx на 8080 останавливать не нужно: он не мешает apache2 на 80.

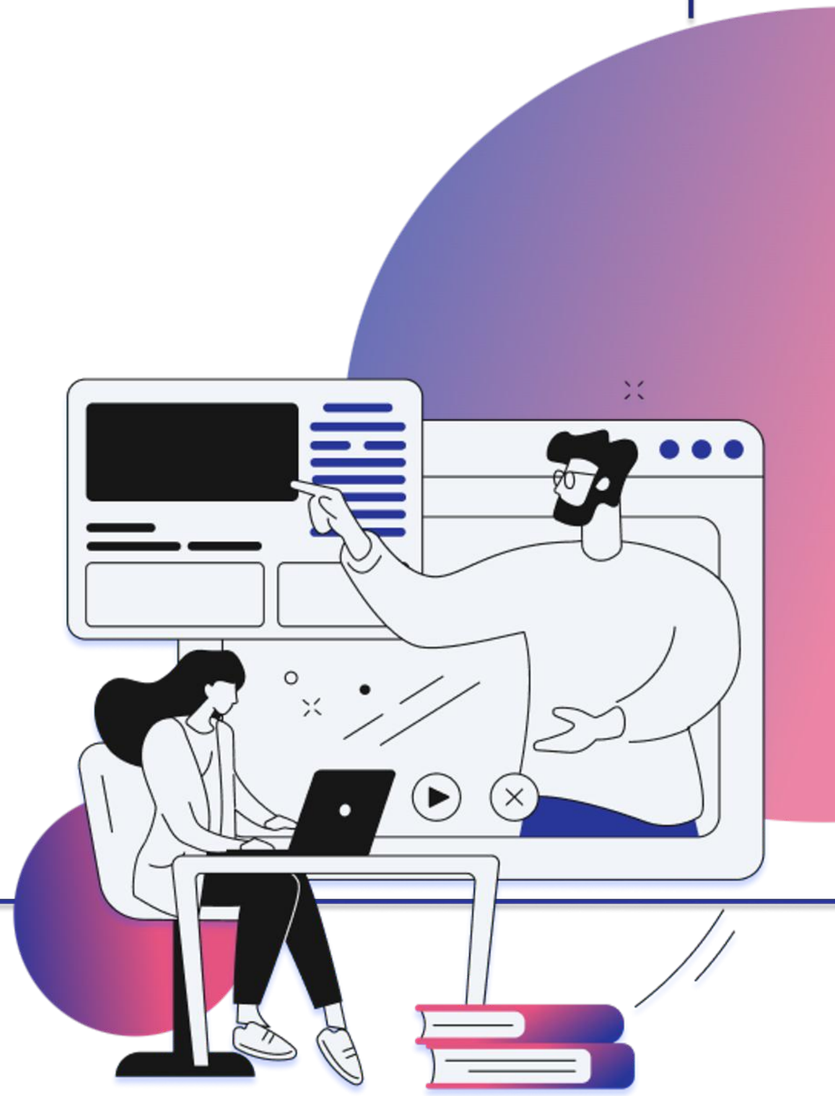
```
rm -f /tmp/vhost.test /tmp/up.test
ansible linux -b -m ansible.builtin.file \
-a "path=/etc/lab-notice state=absent"
```

Учебный центр АйТи Клауд

# Практика 13. Хранилище: диски, LVM, файловые системы

**ЭТА ПРАКТИКА О ДИСКАХ.**

современные технологии простым языком



# Обзор практики

- ЭТА ПРАКТИКА О ДИСКАХ: МЫ РАЗМЕТИМ ВТОРОЙ ДИСК СТЕНДА, СОБЕРЁМ ИЗ НЕГО ГРУППУ ТОМОВ И ЛОГИЧЕСКИЙ ТОМ, СОЗДАДИМ ФАЙЛОВУЮ СИСТЕМУ, СМОНТИРУЕМ ЕЁ В /srv/data И РАСШИРИМ БЕЗ РАЗМОНТИРОВАНИЯ. ГЛАВНОЕ УПРАЖНЕНИЕ —...

# Что мы сделаем

- Посмотрим на второй диск и убедимся, что он есть и что на нём нет ничего нужного.
- Разметим его таблицей GPT: один раздел на весь диск.
- Соберём из раздела группу томов data и логический том webdata, создадим на нём файловую систему — ext4 на Ubuntu, xfs на Rocky Linux — и смонтируем её в /srv/data с записью в /etc/fstab.
- Расширим том и файловую систему, не размонтируя её.
- Соберём всё в один плейбук и добьёмся того, чтобы повторный прогон дал changed=0.
  - шаг 1 — что такое физический том, группа томов и логический том;
  - шаг 2 — файлы практики и переменные по семействам ОС;
  - шаг 3 — осмотр дисков: lsblk, blkid, findmnt, df -h;
  - шаг 4 — разметка второго диска модулем community.general.parted;
  - шаг 5 — группа томов data и логический том webdata;
  - шаг 6 — файловая система: ext4 на Ubuntu, xfs на Rocky Linux;
  - шаг 7 — монтирование тома и запись в /etc/fstab;
  - шаг 8 — расширение тома и файловой системы без размонтирования;
  - шаг 9 — весь путь целиком одним плейбуком;
  - шаг 10 — главное упражнение практики: повторный прогон с changed=0;
  - шаг 11 — что отличает работу от учебного примера.

# Что понадобится

▸ **СТЕНД ИЗ ПРАКТИК 1, 2 И 9: Три управляемые машины:**

```
web1 10.0.2.11 — Ubuntu 26.04 LTS
web2 10.0.2.12 — Ubuntu 26.04 LTS
db1 10.0.2.13 — Rocky Linux 10
```

• Второй диск 10 ГБ есть на всех трёх узлах.

• На управляющей машине control (10.0.2.5) второго диска нет: там размечать нечего.

▸ **ВНИМАНИЕ, БЕЗОПАСНОСТЬ: Практика размечает диск и создаёт файловую систему — это операции, которые уничтожают данные:**

– работать только на учебном стенде. На рабочем сервере и на своём
– ноутбуке такой плейбук запускать нельзя;
– работать только со вторым диском /dev/sdb. Корневая система лежит
– на /dev/sda, и её трогать не нужно;
– модуль community.general.parted с другим значением label
– переписывает таблицу разделов целиком: разделы и данные на диске
– пропадают. Поэтому диск задан переменной, а перед разметкой стоит
– проверка «диск есть, и с него ничего не смонтировано»;
– если вы не уверены, какой диск видит узел, остановитесь и посмотрите

# ЧАСТЬ 1. ДИСК, LVM, ФАЙЛОВАЯ СИСТЕМА, МОНТИРОВАНИЕ

CLOUD
УЧЕБНЫЙ ЦЕНТР

-

# ШАГ 1. ФИЗИЧЕСКИЙ ТОМ, ГРУППА ТОМОВ И ЛОГИЧЕСКИЙ ТОМ

- Диск — это набор секторов.
- Чтобы хранить на нём файлы, нужны три слоя: таблица разделов (какие области диска кому принадлежат), файловая система (как внутри области лежат файлы) и точка монтирования (где файловая система видна в дереве...
- LVM добавляет между разделом и файловой системой ещё один слой:
  - ФИЗИЧЕСКИЙ ТОМ (PV) — раздел или диск, отданный под LVM: в практике
  - /dev/sdb1. Его создаёт pvcreate, и это делает модуль lvg;
  - ГРУППА ТОМОВ (VG) — пул места из физических томов: в практике
  - группа data. Место в ней нарезается экстентами по 4 МиБ;
  - ЛОГИЧЕСКИЙ ТОМ (LV) — «раздел внутри пула», его получает файловая
  - система: в практике webdata.
  - /dev/sdb1 группа томов data логический том webdata
  - физический -> пул 10 ГБ, из них -> /dev/mapper/data-webdata
  - том 6 ГБ отдано тому |
  - v
  - ext4 или xfs на /srv/data
- У логического тома два имени, и оба ведут к одному устройству: /dev/mapper/data-webdata и /dev/data/webdata.
- В плейбуках курса используется первое: оно одинаково на обоих семействах ОС.

# ШАГ 2. ФАЙЛЫ ПРАКТИКИ И ПЕРЕМЕННЫЕ ПО СЕМЕЙСТВАМ

- Копируем файлы практики в рабочий каталог курса.
- Каталог vars копируется целиком — плейбуки ссылаются на него по имени:

```
cd ~/ansible-lab
cp -r /путь/к/репозиторию/files/13_storage_lvm/. .
ls
01_inspect.yml 02_parted.yml 03_lvm.yml 04_filesystem.yml
05_mount.yml 06_resize.yml 07_full_storage.yml vars
```

- Различия между Ubuntu и Rocky Linux собраны в два файла переменных, как в практике 9.
- Файл для Ubuntu — files/13_storage_lvm/vars/Debian.yml:

```
---
# Переменные семейства Debian (Ubuntu 26.04): web1 и web2.
# Файл читается задачей ansible.builtin.include_vars в pre_tasks —
```

- Файл для Rocky Linux — files/13_storage_lvm/vars/RedHat.yml.
- Имена переменных те же, значения другие; ниже — отличающиеся строки (полностью файл лежит в каталоге курса):
- Файл выбирается по факту ansible_os_family, значение которого — Debian или RedHat.
- Подключается он задачей include_vars в pre_tasks:

# ШАГ 3. ОСМОТР ДИСКОВ: LSBLK, BLKID, FINDMNT, DF -H

- Перед разметкой нужно посмотреть на диск: у команд разметки нет отмены.
- Смотреть надо четыре вещи: какие устройства есть и какого они размера (lsblk), есть ли на диске файловая система (blkid), не смонтировано ли с него что-то (lsblk, findmnt) и сколько места на корне (df -h).
- Плейбук files/13_storage_lvm/01_inspect.yml:

```
---
# 01_inspect.yml — практика 13, шаг 3: осмотр дисков до любых изменений.
# Плейбук только читает состояние: здесь нет ни одной задачи,
# которая что-то меняет на узле.
# Запуск из каталога ~/ansible-lab:
# ansible-playbook 01_inspect.yml
- name: "Осмотр дисков до разметки"
hosts: linux
```

- Запускаем:
- Задача с lsblk отчитывается ok, а не changed: параметр changed_when: false говорит Ansible, что команда ничего не меняет.
- Это тема шага 10.
- Вывод задачи «Вывести дерево блочных устройств» (сокращён):

▸ **ВАЖНО: blkid на чистом диске отвечает rc=2 и пустым stdout. Это не сбой, а ответ «файловой системы нет»: поэтому у задачи failed_when: false. Если бы в stdout был TYPE="ext4", разметку надо остановить.**

# ШАГ 4. РАЗМЕТКА ВТОРОГО ДИСКА: COMMUNITY.GENERAL.PARTED

▸ **ЧЕГО НЕ ДЕЛАТЬ: не запускайте этот плейбук на рабочей машине и на своём компьютере. Модуль создаёт таблицу разделов на устройстве из параметра device: если там окажется диск с данными, данные пропадут молча. И...**

• Плейбук files/13_storage_lvm/02_parted.yml:

```
---
# 02_parted.yml — практика 13, шаг 4: разметка второго диска.
#
# ОПАСНО: плейбук размечает диск. Работать только на учебном стенде
# и только со вторым диском. Запуск такого плейбука на машине с данными
# уничтожит эти данные: модуль с другим label переписывает таблицу
# разделов целиком.
#
```

• РАЗБОР ПАРАМЕТРОВ:

– device — диск, с которым работаем; number — номер раздела, без него
– модуль не станет ничего делать;
– label — тип таблицы разделов. Берём gpt вместо старой msdos:
– нет предела в четыре раздела, у разделов есть имена; name задаёт
– имя раздела, для GPT его принято указывать;

# ШАГ 5. ГРУППА ТОМОВ DATA И ЛОГИЧЕСКИЙ ТОМ WEBDATA

- Файл files/13_storage_lvm/03_lvm.yml:

```
---
# 03_lvm.yml — практика 13, шаг 5: группа томов data и логический том webdata.
# Запуск из каталога ~/ansible-lab:
# ansible-playbook 03_lvm.yml
- name: "LVM: группа томов и логический том"
hosts: linux
become: true
pre_tasks:
```

- РАЗБОР:
  - community.general.lvg описывает группу томов: vg — имя, pvs —
  - список физических томов. Модуль сам выполнит pvcreate для нового
  - раздела и vgcreate для группы, а если группа уже есть, но список
  - pvs вырос, — vgextend;
  - community.general.lvol описывает логический том: vg — группа,
  - lv — имя, size — размер, state: present — «том должен быть»;

# ШАГ 6. ФАЙЛОВАЯ СИСТЕМА: EXT4 НА UBUNTU, XFS НА ROCKY

- Файл files/13_storage_lvm/04_filesystem.yml:

```
---
# 04_filesystem.yml — практика 13, шаг 6: файловая система на логическом томе.
# Тип файловой системы берётся из переменной по семейству ОС:
# ext4 на Ubuntu, xfs на Rocky Linux.
#
# Запуск из каталога ~/ansible-lab:
# ansible-playbook 04_filesystem.yml
- name: "Файловая система на логическом томе"
```

- **ПОЧЕМУ РАЗНЫЕ ФАЙЛОВЫЕ СИСТЕМЫ: ext4 — привычная файловая система Ubuntu, xfs — файловая система по умолчанию в семействе RedHat: на Rocky Linux ставить на данные что-то другое означает идти против поддержки....**
- ЧТО УМЕЮТ ОБЕ, А ЧТО НЕТ:
  - ext4 растёт и уменьшается; уменьшение — только на размонтированной
  - файловой системе, командой resize2fs;
  - xfs растёт только смонтированным (xfs_growfs), а уменьшить xfs
  - нельзя вообще: только пересобрать файловую систему, потеряв данные;
  - у самого модуля community.general.filesystem есть параметр resizefs:

# ШАГ 7. МОНТИРОВАНИЕ: ANSIBLE.POSIX.MOUNT И /etc/FSTAB

- Файл files/13_storage_lvm/05_mount.yml:

```
---
# 05_mount.yml — практика 13, шаг 7: монтирование и запись в /etc/fstab.
# state: mounted означает «строка в fstab есть и том смонтирован сейчас».
#
# Запуск из каталога ~/ansible-lab:
# ansible-playbook 05_mount.yml
- name: "Монтирование логического тома"
hosts: linux
```

- ДВА СОСТОЯНИЯ, КОТОРЫЕ ЛЕГКО ПЕРЕПУТАТЬ:
  - state: present — строка есть в /etc/fstab, но сейчас том может быть
  - не смонтирован. Так готовят запись заранее, например перед
  - перезагрузкой;
  - state: mounted — строка в fstab есть И том смонтирован сейчас.
  - Это то, что нужно в практике: состояние «работает»;
  - state: unmounted — строка в fstab остаётся, а том размонтирован;

# ЧАСТЬ 2. РАСШИРЕНИЕ И ИДЕМПОТЕНТНОСТЬ: РАБОТА, А НЕ УЧЕБНЫЙ ПРИМЕР

-

# ШАГ 8. РАСШИРЕНИЕ ТОМА И ФАЙЛОВОЙ СИСТЕМЫ БЕЗ РАЗМОНТИРОВАНИЯ

• Файл files/13_storage_lvm/06_resize.yml:

```
---
# 06_resize.yml — практика 13, шаг 8: расширение логического тома
# и файловой системы без размонтирования.
#
# resizefs: true заставляет модуль расширить и файловую систему:
# ext4 — через resize2fs, xfs — через xfs_growfs. Разница инструментов
# от нас скрыта, но знать её нужно: xfs растёт только смонтированным.
#
```

▸ **ДВА ИНСТРУМЕНТА, КОТОРЫЕ СКРЫВАЕТ МОДУЛЬ: Модуль lvol видит, что том стал больше, и, из-за resizefs: true, расширяет файловую систему:**

– ext4 растёт командой resize2fs, и она работает на смонтированной
– файловой системе — это и называется расширением на ходу;
– xfs растёт командой xfs_growfs, и ей передают точку монтирования,
– а не устройство. Поэтому xfs расширяется только смонтированным.

• В плейбуке разницы не видно, а в выводе видна: ext4 показывает в df -h 7.8G, а xfs — 8.0G: так считают сами инструменты.

▸ **ВАЖНО: файловая система растёт только до размера устройства. Если сначала не расширить том, resize2fs и xfs_growfs ответят, что система уже занимает всё устройство. Порядок один: свободное место — расширить...**

# ШАГ 9. ВЕСЬ ПУТЬ ЦЕЛИКОМ: 07_FULL_STORAGE.YML

- Все задачи собраны в один плейбук files/13_storage_lvm/07_full_storage.yml.
- Он начинается так:

```
---
# 07_full_storage.yml — практика 13, шаг 9: весь путь целиком.
# Диск — раздел — группа томов — логический том — файловая система —
# монтирование, одним плейбуком.
#
# Описание собрано так, чтобы повторный прогон дал changed=0:
# раздел описывается без размеров, том — точным числом, файловая
# система — без force, а все читающие команды помечены
```

- Задачи шагов 4–7 перенесены в файл дословно, отличаются только две:
  - размер тома берётся из storage_lv_size_extended: плейбук описывает
  - КОНЕЧНОЕ состояние (8G), а шаг 5 — начальное (6G);
  - читающие команды помечены changed_when: false, а partprobe
  - выполняется только при изменении таблицы разделов.
- Первый запуск:

# ШАГ 10. ГЛАВНОЕ УПРАЖНЕНИЕ: ПОВТОРНЫЙ ПРОГОН С CHANGED=0

- Запустите итоговый плейбук второй раз, ничего не меняя:

```
ansible-playbook 07_full_storage.yml
PLAY RECAP
db1 : ok=16 changed=0 unreachable=0 failed=0 skipped=1
web1 : ok=16 changed=0 unreachable=0 failed=0 skipped=1
web2 : ok=16 changed=0 unreachable=0 failed=0 skipped=1
```

- changed=0 означает, что описание совпало с состоянием машин.
- Для диска это самый сложный случай в курсе: у раздела нет имени, и модуль сравнивает написанное с тем, что ядро сообщает о диске.
- Сравнивает он не всё — именно там и теряется идемпотентность.
- ШЕСТЬ ПРИЧИН, ПО КОТОРЫМ CHANGED БЫВАЕТ НЕ НУЛЕВЫМ:
  - ЧИТАЮЩИЕ КОМАНДЫ БЕЗ CHANGED_WHEN. Задача с ansible.builtin.command
  - — это всегда «я выполнил команду», то есть changed. lsblk, blkid,
  - findmnt, df и mount -a ничего не меняют, поэтому у них стоит
  - changed_when: false. Без него changed=0 не получится.
  - FILESYSTEM С FORCE: TRUE. Модуль создаёт файловую систему заново
  - на каждом прогоне: changed всегда, а данные уничтожены. Правильно —

# ШАГ 11. ЧТО ОТЛИЧАЕТ РАБОТУ ОТ УЧЕБНОГО ПРИМЕРА

- ПЕРВОЕ: ЦЕНА ОШИБКИ. Ошибка в имени диска в стенде лечится снимком ВМ, в работе тот же прогон уничтожает данные. Поэтому перед разметкой смотрят не только lsblk, но и lsscsi, lsof, pvs и vgs.
- ВТОРОЕ: ИМЕНА УСТРОЙСТВ. /dev/sdb — не имя, а адрес: в fstab в работе пишут UUID=... или LABEL=..., а не /dev/sdb1. Мы монтируем /dev/mapper/data-webdata: имена LVM устойчивы к порядку дисков.
- ТРЕТЬЕ: МЕСТО НЕ РАСТЁТ САМО. За свободным местом следят мониторингом (vgs, df -h), к дискам с данными в fstab добавляют nofail. И помните про асимметрию: уменьшить xfs нельзя вообще, а ext4 — только в окно...

# Если что-то пошло не так

▸ **ПРОБЛЕМА: Device /dev/sdb1 not found.**

```
fatal: [web1]: FAILED! => {"changed": false, "msg": "Device /dev/sdb1 not found."}
```

- Так отвечает community.general.filesystem, когда устройства нет.
- Причина почти всегда одна: ядро ещё не увидело раздел.
- Проверяйте по порядку:
  - Есть ли раздел: lsblk /dev/sdb
  - Есть ли файл устройства: ls -l /dev/sdb1
  - Перечитать таблицу: partprobe /dev/sdb
  - Сверить имя раздела с файлом переменных:
  - grep -n storage_partition ~/ansible-lab/vars/Debian.yml
- Пункт 4 выполняется на управляющей машине: переменные storage_* подключает include_vars внутри плейбука, в ad-hoc команде их нет — debug с var=storage_partition ответит VARIABLE IS NOT DEFINED!.
- Если файл появился после partprobe, вернитесь к шагу 4: в задаче разметки для этого стоит until по lsblk.

▸ **ПРОБЛЕМА: '/dev/sdb1' is already used as ext4, use force=true to overwrite**

```
fatal: [web1]: FAILED! => {"changed": false, "msg": "'/dev/sdb1' is already used as ext4, use force=true to...
```

- Причина: на устройстве уже есть файловая система другого типа — например, storage_fstype поменяли с ext4 на xfs.
- Проверка: blkid на разделе и на томе, сравнение вывода со значением storage_fstype.

# Проверка результата

- Практика выполнена, если выполняются все пункты.
- 1.
- Синтаксис всех файлов практики в порядке:

```
ansible-playbook --syntax-check 07_full_storage.yml
ansible-playbook --syntax-check 02_parted.yml (и остальные)
При успехе команда молчит.
```

- 2.
- Повторный прогон итогового плейбука не меняет ничего:

```
ansible-playbook 07_full_storage.yml
web1 : ok=16 changed=0 unreachable=0 failed=0 skipped=1
web2 : ok=16 changed=0 unreachable=0 failed=0 skipped=1
db1 : ok=16 changed=0 unreachable=0 failed=0 skipped=1
```

- 3.
- Диск размечен, том и файловая система на месте:

```
ansible all -b -m ansible.builtin.command -a "vgs"
```

- 4.

# Что дальше

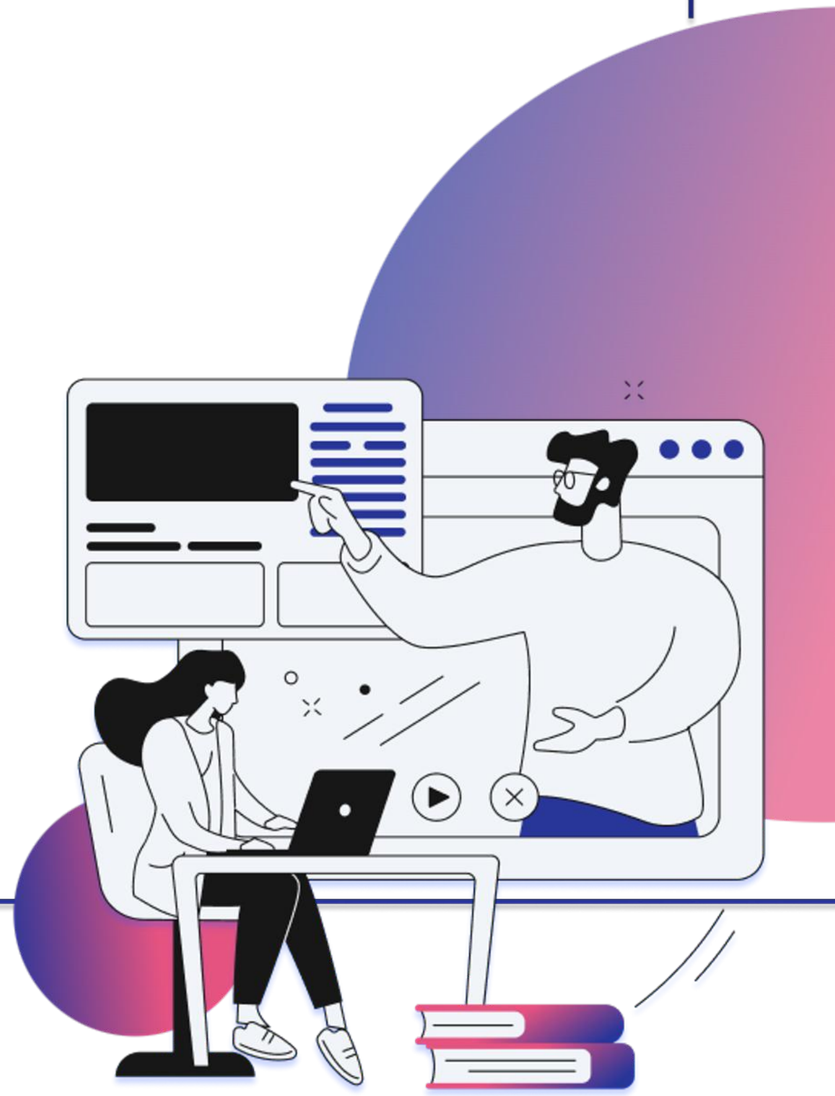
- Далее в курсе:
  - практика 14 — сложные проекты: те же задачи переезжают в роль,
  - появляются include_tasks и serial вместо --limit;
  - практика 15 — Galaxy: чужие роли вместо своих — geerlingguy.nginx
  - на веб-узлах и rhel-system-roles.timesync на db1 — и сравнение
  - с тем, что мы написали сами;
  - практика 16 — качество: ansible-lint и molecule разбираются там
  - на своём проекте files/16_quality_inventory и роли web_admin;
  - тот же порядок — линтер, потом повторный прогон с changed=0 —
  - применим и к плейбукам этой практики.
- ПЕРЕД ПЕРЕХОДОМ уберите за собой: снимок ВМ не нужен, размеченный том — часть работы.
- Проверьте, что /srv/data смонтирован на всех трёх узлах.

Учебный центр АйТи Клауд

# Практика 14. Управление автоматизацией в сложных проектах

**ЭТА ПРАКТИКА — ПРО УПРАВЛЕНИЕ ПАРКОМ, А НЕ ПРО НАСТРОЙКУ ОДНОЙ МАШИНЫ.**

современные технологии простым языком



# Обзор практики

- ЭТА ПРАКТИКА — ПРО УПРАВЛЕНИЕ ПАРКОМ, А НЕ ПРО НАСТРОЙКУ ОДНОЙ МАШИНЫ.
- Роль web_admin из практики 10 станет проектом: точка входа, три плейбука, два окружения, отчёт о парке на управляющей машине и обновление боевого парка по одной машине.
- Разберём, чем статическое включение отличается от динамического и почему это видно на тегах.

## Что мы сделаем

- Соберём проект из каталогов — site.yml, playbooks/, inventory/, group_vars/, tasks/ и roles/ — и разберём, почему «всё в одном файле» перестаёт работать.
- Разберём import_playbook, import_tasks и include_tasks, import_role и include_role: что Ansible разбирает при загрузке, а что во время прогона, и как это отражается на тегах, условиях и выводе --list-tasks.
- Поставим опыт: один и тот же файл задач включим двумя способами и посмотрим на разницу.
- Научим проект трём вещам, которых не было в практике 10: собирать отчёт о парке в один файл на управляющей машине (delegate_to и run_once), обновлять машины партиями (serial) и жить в двух окружениях одним и...

# Что понадобится

- СТЕНД ИЗ ПРАКТИК 1–13.
- Четыре машины собраны, вход по ключу работает, Ansible установлен в ~/ansible-venv:

```
control 10.0.2.5 — управляющая машина, здесь мы всё выполняем
web1 10.0.2.11 — Ubuntu 26.04 LTS
web2 10.0.2.12 — Ubuntu 26.04 LTS
db1 10.0.2.13 — Rocky Linux 10
```

- ЧТО ДОЛЖНО БЫТЬ ГОТОВО ЗАРАНЕЕ:
  - коллекции ansible.posix и community.general установлены: роль
  - web_admin ставит community.general.ufw и ansible.posix.firewalld;
  - проект практики 10 прогонялся хотя бы раз: роль web_admin
  - раскладывает сайт на web1 и web2 и открывает порт 80;
  - на корневом разделе машин больше 500 МБ свободного места.
- ГОТОВЫЙ ПРОЕКТ лежит в files/14_complex_projects/ и напечатан здесь файл за файлом.
- Роль web_admin напечатана в практике 10: в проект она скопирована из files/10_roles/.
- **ВРЕМЯ: три академических часа. Скопируем проект в каталог курса:**

```
mkdir -p ~/ansible-lab/14_complex_projects
```

# ЧАСТЬ 1. СТРУКТУРА ПРОЕКТА: СТАТИКА И ДИНАМИКА

-

# ШАГ 1. ИЗ ЧЕГО СОСТОИТ ПРОЕКТ И ПОЧЕМУ НЕ «ВСЁ В ОДНОМ ФАЙЛЕ»

- В практике 10 проект состоял из одного play и роли.
- Задач теперь больше: веб-часть, сервер баз данных, отчёт, два окружения.
- В одном файле это три сотни строк, где работы переплетены.

```
14_complex_projects/
├── ansible.cfg настройки Ansible для проекта
├── site.yml точка входа: только import_playbook
├── inventory/
│   ├── dev.ini окружение dev: группа dev
│   └── prod.ini окружение prod: группа prod
├── group_vars/
│   ├── all/
```

- Каталога reports/ в проекте нет: его создаёт задача из report.yml при первом прогоне.
- Настройки проекта — файл ansible.cfg:

▸ **ПОЧЕМУ ИМЕННО ТАК: инвентаря по умолчанию нет, потому что у проекта два равноправных окружения: забытый ключ -i молча настроил бы dev там, где имели в виду prod. Параметр roles_path пригодится, когда плейбук...**

# ШАГ 2. SITE.YML: ТОЧКА ВХОДА НА IMPORT_PLAYBOOK

• В точке входа нет задач: только включения плейбуков.

```
---
# site.yml — точка входа проекта практики 14: только включения
# плейбуков, ни одной задачи. Порядок включений — порядок работы.
# Окружение выбирается ключом -i, сам файл от него не зависит.
# Ключ import_playbook записан коротко, как в документации Ansible:
# это ключ плейбука, а не модуль. Полное имя —
# ansible.builtin.import_playbook, принимаются оба варианта.
- name: Настроить веб-часть стенда
```

• Ansible читает указанный файл при разборе точки входа и подставляет его plays на это место: порядок plays остаётся тем, в котором стоят включения.

▸ **ПОЧЕМУ ИМЕННО ТАК: плейбук нельзя включить динамически. Для файла задач есть include_tasks, для роли — include_role, а динамического включения плейбука в Ansible нет: только import_playbook.**

▸ **ВАЖНО: import_playbook — не задача: его нельзя поставить внутрь tasks, завернуть в loop или пометить тегом. Теги, serial и условия задаются в самих включаемых плейбуках.**

▸ **ПРОВЕРКА: до запуска убедимся, что файлы разбираются:**

• Команда молчит.

• Вывод Ansible в этой практике напечатан сокращённо, звёздочки после имени задачи и символ табуляции опущены.

# ШАГ 3. IMPORT_TASKS И INCLUDE_TASKS: ОДИН ФАЙЛ ДВУМЯ СПОСОБАМИ

- Общие проверки машины лежат в отдельном файле tasks/common_checks.yml:

```
---
# tasks/common_checks.yml — общие проверки машины.
#
# Один и тот же файл подключён в проекте тремя способами:
# * playbooks/webservers.yml — статически, ключом import_tasks;
# * playbooks/db.yml — динамически, ключом include_tasks;
# * playbooks/report.yml — статически, перед сбором отчёта.
#
```

- Задачи ничего не меняют: они читают факты и останавливают прогон, если машина не готова, поэтому проверки не портят changed=0.
- Динамическое включение этого файла — в playbooks/db.yml:
- Путь начинается с ../, потому что плейбуки лежат в playbooks/, а файл задач — в tasks/.
- Кроме включения, в db.yml есть шапка play (hosts, db, tags) и две простые задачи: каталог резервных копий и файл-описание.
- Статическое включение того же файла написано в webservers.yml (шаг 4).
- ▸ **ТЕПЕРЬ ОПЫТ: Напишите динамическое включение без apply — так, как его пишут по привычке, — и посмотрите на список задач:**
- В play №1 файл разобран заранее: все пять задач стоят своими именами.

# ШАГ 4. IMPORT_ROLE И INCLUDE_ROLE: ТО ЖЕ РАЗЛИЧИЕ ДЛЯ РОЛЕЙ

• Веб-часть проекта — файл playbooks/webservers.yml целиком:

```
---
# playbooks/webservers.yml — веб-часть проекта практики 14.
# Здесь всё сделано статически: роль подключена ключом roles, общие
# проверки — ключом import_tasks. Поэтому задачи видны в --list-tasks,
# а тег на включении достаётся всем включённым задачам.
# serial берётся из group_vars окружения: у dev это "100%", у prod — 1.
- name: "Веб-серверы: сайт, проверки и безопасное обновление"
hosts: web
```

• Ключ roles — тоже статическое включение: задачи роли появляются в play до начала прогона, и тег web с play достаётся им всем.

• Динамический вызов той же роли выглядит так:

• Тег и условие на include_role относятся к самому вызову, а не к задачам роли.

• Свои особенности: include_role умеет tasks_from — вызвать, например, tasks/checks.yml внутри роли (приём практики 15); параметры роли через vars сильнее и defaults, и vars роли: выше них только --extra-vars...

▸ **ПОЧЕМУ ИМЕННО ТАК: роль вызывается ключом roles, потому что вызов у нас один. Когда ролей станет несколько, удобнее станет import_role: он позволяет задать параметры роли в вызове.**

▸ **ПРОВЕРКА: убедимся, что роль нашлась:**

# ШАГ 5. ТЕГИ НА ВЕСЬ ПРОЕКТ: WEB, DB, REPORT, CHECKS, ALWAYS

– web, db, report — теги plays в трёх плейбуках: веб-часть, сервер
– баз данных, отчёт;
– checks — тег файла проверок и обоих включений: одни проверки,
– без настройки машин;
– always — второй тег файла проверок: проверки идут в любом прогоне.
– ansible-playbook --list-tags site.yml -i inventory/dev.ini
– play #1 (web): Веб-серверы: сайт, проверки и безопасное обновление
– TASK TAGS: [always, checks, web]
– play #2 (db): Сервер баз данных
– TASK TAGS: [always, checks, db]
– play #3 (linux): Собрать отчёт о парке машин
– TASK TAGS: [always, checks, report]
– ansible-playbook site.yml -i inventory/dev.ini --tags db
– ansible-playbook site.yml -i inventory/dev.ini --skip-tags report
– ansible-playbook site.yml -i inventory/dev.ini --tags all,never

• Первая команда настроит только db1, вторая сделает всё, кроме отчёта.

# ЧАСТЬ 2. ОКРУЖЕНИЯ, ОТЧЁТ И ОБНОВЛЕНИЕ ПАРКА

-

# ШАГ 6. DELEGATE_TO И RUN_ONCE: ОТЧЁТ О ПАРКЕ В ОДИН ФАЙЛ

- Отчёт о парке просят постоянно: версии ОС, ядра и свободное место по всем машинам одной страницей.
- Собираем его на управляющей машине.

```
---
# playbooks/report.yml — отчёт о парке машин.
#
# Отчёт собирается не на узлах, а на управляющей машине: файл должен
# быть один. Ключ delegate_to: localhost выполняет задачу на control,
# а run_once — один раз, хотя play идёт по всем машинам парка.
#
# serial в этом play не задан намеренно: run_once выполняется один раз
```

- Без delegate_to: localhost задача copy создала бы reports/fleet-report.txt на web1, web2 и db1 — три файла вместо одного.
- RUN_ONCE выполняет задачу один раз, хотя play идёт по всем машинам, и применяет результат ко всей партии.
- HOSTVARS нужен потому, что текст отчёта собирает один хост, а говорит он обо всём парке.

▸ **ПОЧЕМУ ИМЕННО ТАК: можно было собрать отчёт циклом из задач lineinfile по строке на машину. Минусов два: строка дописывается, а устаревшую удалять некому, и порядок строк случайный; шаблон в copy отдаёт ровно...**

▸ **ВАЖНО: если делегировать задачу адресу, которого нет в инвентаре, машина в инвентарь не попадёт: переменные у неё будут только из группы all. Если такая машина нужна по-настоящему, её добавляют add_host.**

▸ **ПРОВЕРКА: отчёт лежит на управляющей машине:**

# ШАГ 7. DELEGATE_FACTS: ЧЬИ ЭТО ФАКТЫ И ЧЬЯ ПЕРЕМЕННАЯ

- Делегирование не меняет того, чья это задача.
- Формулировка из документации Ansible: покупки принадлежат вам, даже если домой их привёз кто-то другой.
- Отсюда два следствия.

▸ **ПЕРВОЕ: аргументы задачи (dest, content) подставляются из переменных того хоста, по которому идёт play, а служебные параметры подключения — интерпретатор Python, connection, become, shell — берутся у того...**

```
delegate_to: localhost
vars:
source_host: "{{ hostvars[inventory_hostname]['ansible_host'] }}"
```

- inventory_hostname остаётся именем исходной машины, а одноимённые переменные могут относиться уже к машине, которой задача делегирована.

▸ **ВТОРОЕ: факты, собранные делегированной задачей, Ansible записывает исходной машине play, а не той, на которой задача выполнилась. Чтобы факты достались делегированной машине, ставят delegate_facts: true:**

– name: Собрать факты с машин парка в один play
– ansible.builtin.setup:
– delegate_to: "{{ item }}"
– delegate_facts: true
– loop: "{{ groups['linux'] }}"

- Так делают, когда play идёт по одной группе машин, а факты нужны о другой.
- В нашем отчёте delegate_facts не нужен: строку отчёта каждая машина считает сама, обычной задачей set_fact.

# ШАГ 8. SERIAL И MAX_FAIL_PERCENTAGE: ОБНОВЛЯЕМ ПО ОДНОЙ МАШИНЕ

- В веб-части проекта стоит строка, которой не было в прошлых практиках:

```
serial: "{{ update_serial | default(1) }}"
max_fail_percentage: 0
```

- serial задаёт размер партии.
- Ansible выполняет play целиком на первой партии машин и только потом переходит к следующей: при serial: 1 будет две партии — сначала web1 (все задачи play), потом web2.

```
serial: 1 одна машина за заход
serial: "50%" половина машин парка за заход
serial: [1, "20%"] разные размеры партий по порядку
```

- Проценты считаются от общего числа машин, которые попали в play.
- Если число не делится нацело, остаток попадает в последнюю партию, а размер партии никогда не бывает меньше единицы.
- ▸ **ВАЖНО: serial не отменяет параллельность внутри партии. Он задаёт только размер партии; машины внутри неё обрабатываются одновременно, и число одновременных подключений ограничено параметром forks. Отсюда:...**
- Зачем это в работе.
- Обновить десять веб-серверов «всеми сразу» — значит одновременно перезапустить службу на всех десяти, и часть запросов уйдёт в никуда.
- Партиями делают иначе: одна-две машины выведены из работы, обновлены, возвращены.
- max_fail_percentage — страховка при обновлении партиями.
- Значение считается внутри партии: если в партии отказало больше машин, чем указано в процентах, Ansible останавливает play и не трогает остальные партии.

# ШАГ 9. ANY_ERRORS_FATAL, STRATEGY: FREE И THROTTLE

- ANY_ERRORS_FATAL.
- По умолчанию отказ на одной машине останавливает работу только на ней: остальные продолжают. any_errors_fatal меняет это: задача доработает на всех машинах текущей партии, после чего play останавливается...
- Ставят его там, где половина сделанной работы хуже, чем несделанная: обновление схемы базы, переключение балансировщика, смена сертификата.
- Ключ работает и на уровне play, и на уровне блока.
- STRATEGY: FREE.
- По умолчанию работает линейная стратегия: все машины проходят первую задачу, потом все вторую. free (strategy: free в play) отпускает каждую машину по её маршруту: машина, закончившая первую задачу, сразу...
- Приёмы, которые рассчитывают на «партия прошла play целиком» (serial), с free теряют смысл.
- THROTTLE ограничивает число машин, которые выполняют конкретную задачу одновременно.
- Ставится на задачу или на блок:
  - name: Обратиться к внешней системе, которая считает запросы
  - ansible.builtin.uri:
  - url: "https://api.example.com/register/{{ inventory_hostname }}"
  - method: POST
  - throttle: 1
- Ключевое слово — ограничивает: увеличить параллельность throttle не может.
- Чтобы он подействовал, его значение должно быть меньше forks и размера партии.

# ШАГ 10. ДВА ОКРУЖЕНИЯ ОДНИМ И ТЕМ ЖЕ SITE.YML

- Теперь инвентарей два — по одному на окружение.
- Инвентарь окружения dev, файл inventory/dev.ini:

```
# inventory/dev.ini — окружение dev. Формат INI.
# Группа dev нужна, чтобы Ansible нашёл файл group_vars/dev.yml:
# машины попадают в неё через группу linux, а не перечислением.
[web]
web1 ansible_host=10.0.2.11
web2 ansible_host=10.0.2.12
[db]
db1 ansible_host=10.0.2.13
```

- Второй инвентарь, inventory/prod.ini, отличается группой окружения и комментарием; остальные секции те же:
- Имена групп dev и prod совпадают с именами файлов в group_vars: так Ansible и понимает, какой файл переменных читать.
- В рабочем проекте prod был бы другим парком машин, часто — динамическим (практика 16).
- Переменные окружения dev — файл group_vars/dev.yml:
- Переменные окружения prod — файл group_vars/prod.yml:
- Общие переменные проекта — файл group_vars/all/vars.yml:

# ШАГ 11. СЕКРЕТЫ ОКРУЖЕНИЯ: VAULT.YML РЯДОМ С GROUP_VARS

- Секреты живут там же, где переменные, — в group_vars, и открытым текстом в репозиторий не попадают:

```
---
# ОБРАЗЕЦ файла секретов проекта. Это не зашифрованный файл: в курс он
# положен открытым текстом, потому что шифрование проверяется на стенде,
# а непроверенный шифротекст студент своим паролем не откроет.
#
# Как из образца сделать рабочий файл:
# cp group_vars/all/vault.yml.example group_vars/all/vault.yml
# # впишите свои значения, затем зашифруйте файл:
```

- Это файл group_vars/all/vault.yml.example.
- Секреты бывают общие для проекта (group_vars/all/vault.yml) и отдельные для окружения (group_vars/dev/vault.yml и group_vars/prod/vault.yml): пароли боевой и пробной баз разные, и доступ к ним у разных людей.
- Запуск проекта с секретами требует пароля Vault:

▸ **ПОЧЕМУ ИМЕННО ТАК: у Ansible Vault нет понятия «окружение»: шифрование это свойство файла, а не окружения. Значит, окружение секретов задаётся тем же способом, что и окружение переменных: именами групп и...**

▸ **ЧЕГО НЕ ДЕЛАТЬ: не держите пароль Vault в репозитории: файл .vault_pass и его копии попадают в .gitignore и в репозиторий не кладутся никогда. И не переименовывайте vault.yml.example в vault.yml, не зашифровав...**

# ШАГ 12. --LIMIT ВМЕСТЕ С DELEGATE_TO, ADD_HOST И META

- ПЕРВЫЙ приём: --limit вместе с delegate_to.
- Ключ --limit сокращает список машин, по которым идёт play:

```
ansible-playbook site.yml -i inventory/prod.ini --tags report --limit web1
```

- В файле отчёта будет одна строка, а остальные машины получат значение по умолчанию — «данных нет»: их факты не собирались, потому что play по ним не шёл. delegate_to меняет место выполнения задачи, а не список...
- **ВТОРОЙ: add_host добавляет машину в инвентарь во время прогона. Нужен там, где машина появляется в процессе: создалась в облаке, поднялась из шаблона. Добавленная машина действует только для следующих plays:**
  - name: Добавить машину в инвентарь текущего прогона
  - ansible.builtin.add_host:
  - name: web3
  - groups: web
- **ВАЖНО: add_host всегда выполняется на управляющей машине — это одна из задач, которую делегировать нельзя, как и debug с include. В рабочем проекте здесь же указывают ansible_host, если адрес новой машины не...**
- **ТРЕТИЙ: meta: end_play и meta: end_host — это не модули, а команды самому Ansible:**
  - name: Остановить play для всех машин
  - ansible.builtin.meta: end_play
  - name: Вывести одну машину из работы
  - ansible.builtin.meta: end_host
- end_play заканчивает текущий play для всех машин (следующие plays из site.yml выполнятся), end_host — только для той машины, на которой сработал.

# ШАГ 13. ИТОГ ПРАКТИКИ: ОДНА КОМАНДА НА ЧЕТЫРЕ МАШИНЫ

- Проект собран.
- Запускаем его целиком на боевом окружении:

```
cd ~/ansible-lab/14_complex_projects
ansible-playbook site.yml -i inventory/prod.ini
PLAY [Веб-серверы: сайт, проверки и безопасное обновление]
TASK [web_admin : Разложить главную страницу из шаблона] changed: [web1]
...остальные задачи роли: установка, оформление, служба, межсетевой
экран; в задачах отчёта видно имя хоста со стрелкой — web1 -> localhost
RUNNING HANDLER [web_admin : web_admin: перезапустить веб-сервер]
changed: [web1]
```

- **ЧТО ЗДЕСЬ ЧИТАТЬ: строка PLAY для веб-части напечатана дважды — serial: 1 в prod разбил две машины на две партии; в задачах отчёта видно web1 -> localhost: задача идёт по машине web1, а выполняется на...**
- **ЧТО ВИДНО В ЧИСЛАХ: В ok входят все задачи, которые машина выполнила, включая обработчики; они же попадают и в changed. Разница между web1 и web2 (28 против 25) — это три задачи отчёта: они выполняются один...**
- Отчёт на управляющей машине:
- Повторный прогон ничего не меняет:
- changed=0 на всех трёх машинах: проект описывает состояние, а не выполняет действия.
- Отчёт тоже не изменился — в нём нет ни даты, ни времени.

# Если что-то пошло не так

▸ **ПРОБЛЕМА: тег на include_tasks не тегирует задачи внутри файла**

• Симптом: с --tags checks проверки идут на web1, а на db1 не идут, и ошибки при этом нет:

```
TASK [Выполнить общие проверки машины]
included: /home/engineer/ansible-lab/14_complex_projects/tasks/common_checks.yml for db1
```

• Причина: тег динамического включения относится к самому включению, а задачи внутри файла остаются без тегов.

• Порядок проверки:

```
grep -n -A6 'include_tasks' playbooks/db.yml
ansible-playbook --list-tasks site.yml -i inventory/dev.ini --tags checks
```

• Если в play №2 видна только строка «Выполнить общие проверки машины», apply написан не там: он должен стоять внутри словаря включения.

▸ **ПРОБЛЕМА: делегированная задача берёт не ту переменную**

```
fatal: [web1 -> localhost]: FAILED! => {"changed": false, "msg": "Destination /fleet-report.txt does not ex...
```

• Причина: при делегировании часть переменных берётся у машины, которой задача делегирована, а у адреса не из инвентаря переменные только из группы all.

• Порядок проверки:

```
grep -n -A4 'delegate_to' playbooks/report.yml
ansible-playbook site.yml -i inventory/dev.ini --tags report \
--limit web1 -e report_dir=/tmp
```

# Проверка результата

- 1.
- Проект разбирается в обоих окружениях:

```
cd ~/ansible-lab/14_complex_projects
ansible-playbook --syntax-check site.yml -i inventory/dev.ini
ansible-playbook --syntax-check site.yml -i inventory/prod.ini
```

- 2.
- Теги видны и работают:

```
ansible-playbook --list-tags site.yml -i inventory/prod.ini
...в списке есть always, checks, db, report, web
```

- 3.
- Статика и динамика различимы: в play веб-серверов задачи

```
из common_checks.yml перечислены поимённо, в play сервера баз
данных — только включение (после добавления apply — и там).
```

- 4.
- Два окружения отличаются только переменными:

```
ansible-inventory -i inventory/prod.ini --host web1 | grep title
```

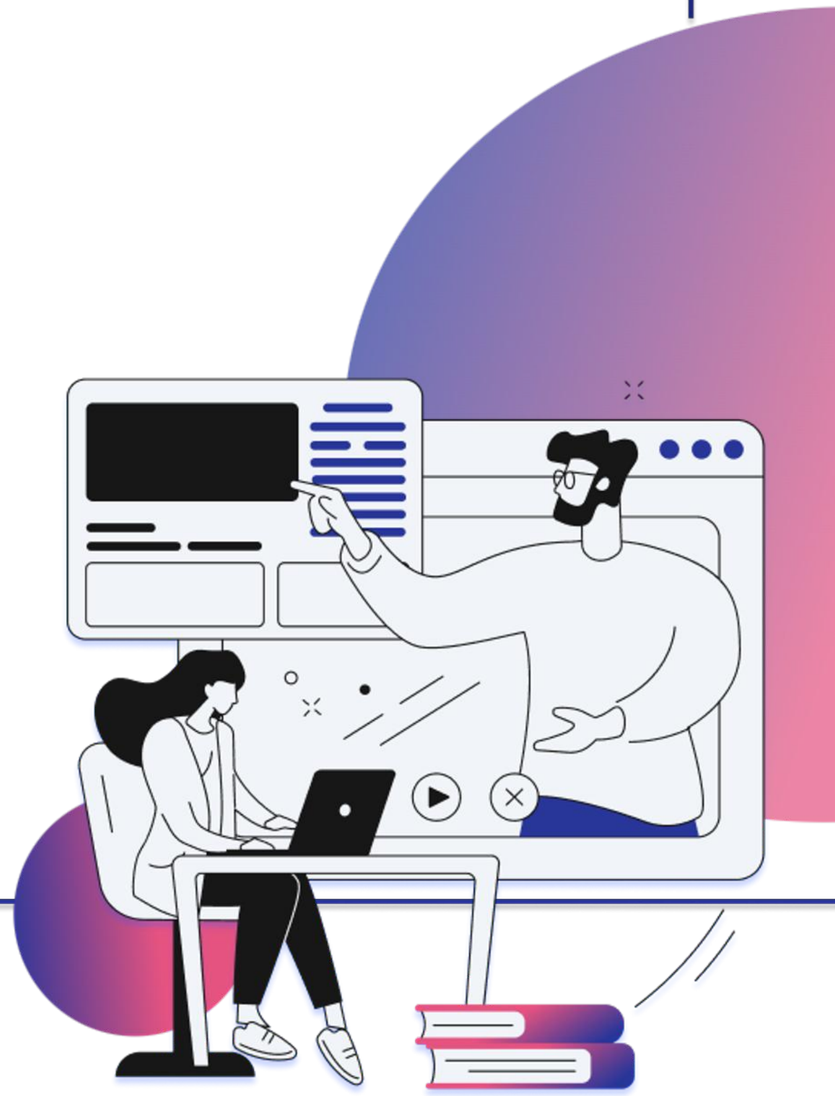
# Что дальше

- Далее в курсе:
  - практика 15 — Galaxy, коллекции и системные роли RHEL: своя роль
  - web_admin заменяется на geerlingguy.nginx, требования проекта —
  - в requirements.yml, для роли пишется meta/argument_specs.yml;
  - практика 16 — качество автоматизации и динамическая инвентаризация:
  - ansible-lint и molecule на проект из этой практики, а inventory
  - заменяется внешним инвентарём, который отдаёт список машин сам;
  - приложение app_1_commands.txt — шпаргалка по командам, включая
  - --list-tasks, --list-tags, --list-hosts и --limit.
- ПЕРЕД ПЕРЕХОДОМ уберите за собой: убирать нечего, проект понадобится в практиках 15 и 16.
- Если вы шифровали vault.yml, проверьте пароль: git status.

Учебный центр АйТи Клауд

# Практика 15. Galaxy, коллекции и системные роли RHEL

**ЭТА ПРАКТИКА ПРО ГОТОВОЕ.**



современные технологии простым языком

# Обзор практики

- **ЭТА ПРАКТИКА ПРО ГОТОВОЕ: Свою роль web_admin мы написали руками в практике 10; здесь мы возьмём роль из Galaxy, поставим её вместе с коллекциями одним файлом requirements.yml и разберём, где Ansible ищет роли...**

## Что мы сделаем

- Разберём порядок поиска ролей и коллекций: roles_path, каталог roles/ рядом с плейбуком и переменную ANSIBLE_ROLES_PATH.
- Соберём requirements.yml с ролями и коллекциями и закрепим в нём версии, установим по нему роль geerlingguy.nginx и запустим её так, чтобы она не спорила с apache2 за порт 80.
- Сравним готовую роль со своей web_admin.
- Сделаем свой скелет роли и создадим по нему заготовку.
- Разберём коллекции: установку, список, проверку версий, --force, пространства имён и FQCN.
- Поставим на db1 системную роль timesync из пакета rhel-system-roles.
- Опишем аргументы своей роли в meta/argument_specs.yml и напишем для неё README.md.

# Что понадобится

- СТЕНД ИЗ ПРАКТИК 1–14.
- Четыре машины, вход по ключу, каталог ansible-lab:

```
control 10.0.2.5 — управляющая машина, здесь мы всё выполняем
web1 10.0.2.11 — Ubuntu 26.04 LTS
web2 10.0.2.12 — Ubuntu 26.04 LTS
db1 10.0.2.13 — Rocky Linux 10
```

- ЧТО ДОЛЖНО БЫТЬ ГОТОВО ЗАРАНЕЕ:
  - Ansible 2.19 в ~/ansible-venv (практика 1), окружение активировано:
  - команда ansible-galaxy входит в тот же пакет;
  - проект практики 10 собран в ~/ansible-lab/10_roles: оттуда мы
  - возьмём инвентарь, готовую роль web_admin, плейбук site.yml
  - и каталог group_vars;
  - есть доступ в интернет: ansible-galaxy скачивает роли и коллекции
  - с galaxy.ansible.com;
  - снимок машин сделан: мы поставим два новых веб-сервера и заменим
  - конфигурацию chrony на db1.

# ЧАСТЬ 1. ГОТОВЫЕ РОЛИ И СВОЙ СКЕЛЕТ

-

# ШАГ 1. ГДЕ ANSIBLE ИЩЕТ РОЛИ

- Ansible не ищет роль по всему диску.
- Места поиска перечислены в документации в таком порядке:
  - внутри коллекций, если коллекции используются;
  - каталог roles/ в том же каталоге, где лежит плейбук;
  - каталоги из параметра roles_path — слева направо;
  - каталог, в котором лежит сам файл плейбука.
- Значение roles_path по умолчанию:
  `~/.ansible/roles:/usr/share/ansible/roles:/etc/ansible/roles`
- Именно поэтому роль, поставленная командой ansible-galaxy role install без всяких настроек, потом находится: она попадает в ~/.ansible/roles.
- Настроить путь можно параметром roles_path в ansible.cfg или переменной окружения ANSIBLE_ROLES_PATH: файл в текущем каталоге сильнее ~/.ansible.cfg, а переменная окружения сильнее файла.
- ▸ **ВАЖНО: значение из ansible.cfg ЗАМЕНЯЕТ список по умолчанию, а не дополняет его. Если написать roles_path = ./roles, каталог ~/.ansible/roles перестанет участвовать в поиске, и роли, поставленные раньше без...**
- ▸ **ПРОВЕРКА: посмотрите, что Ansible видит сейчас, до правки ansible.cfg:**

```
ansible-config dump | grep -i roles_path
DEFAULT_ROLES_PATH(default) = ['/home/engineer/.ansible/roles', '/usr/share/ansible/roles', '/etc/ansible/r...
```

# ШАГ 2. ПРОЕКТ ПРАКТИКИ: ANSIBLE.CFG И ИНВЕНТАРЬ

- Собираем отдельный проект, чтобы не ломать проект практики 10:

```
mkdir -p ~/ansible-lab/15_galaxy/roles
cp -r ~/ansible-lab/10_roles/roles/web_admin ~/ansible-lab/15_galaxy/roles/
cp -r files/15_galaxy_collections/. ~/ansible-lab/15_galaxy/
cp ~/ansible-lab/10_roles/inventory.ini ~/ansible-lab/15_galaxy/
cp ~/ansible-lab/10_roles/site.yml ~/ansible-lab/15_galaxy/
cp -r ~/ansible-lab/10_roles/group_vars ~/ansible-lab/15_galaxy/
cd ~/ansible-lab/15_galaxy
```

- Порядок команд важен: сначала копируется роль из практики 10, потом поверх неё ложится каталог из files/ — он добавляет в роль два файла, meta/argument_specs.yml и README.md.
- site.yml и group_vars тоже берутся из проекта практики 10, и это не «на всякий случай».
- Роль без плейбука не запустить, а проверка аргументов в шаге 13 делается прогоном site.yml: мы передадим роли значение через -e и посмотрим, что скажет проверка. group_vars нужен, чтобы play работал с теми же...
- Своей копии этих двух файлов в каталоге курса нет: site.yml и group_vars уже напечатаны и разобраны в практике 10.
- site.yml вызывает нашу роль web_admin: apache2 на web1 и web2 и httpd на db1, то есть порт 80.
- С nginx из практики 12, который слушает 8080, он не спорит.
- Настройки проекта — файл ansible.cfg из files/15_galaxy_collections (пояснения в его комментариях повторяют то, что сказано в шаге 1):

```
[defaults]
```

# ШАГ 3. REQUIREMENTS.YML: РОЛИ И КОЛЛЕКЦИИ ОДНИМ ФАЙЛОМ

- Файл requirements.yml описывает, что нужно скачать с Galaxy.
- Он один на проект: ключ roles — роли, ключ collections — коллекции.

```
roles:
# Готовая роль веб-сервера nginx. Для ролей ansible-galaxy понимает
# только точную версию: диапазоны он для ролей не поддерживает.
- name: geerlingguy.nginx
version: "3.1.4"
collections:
# ansible.posix — модули firewalld, sysctl, mount. Нужна своей роли
# web_admin и практике 13. Запись ниже читается так: «не ниже 1.5.4
```

- Что здесь важно понять:
  - роль и коллекция описываются одинаково: name плюс version;
  - поле source задаёт другой сервер, если коллекция лежит не на
  - galaxy.ansible.com, а, например, на Automation Hub;
  - ставить оба типа вместе умеет команда без уточнения типа:
  - ansible-galaxy install -r requirements.yml. Команды

# ШАГ 4. УСТАНОВКА: ANSIBLE-GALAXY INSTALL -R

```
ansible-galaxy install -r requirements.yml
Starting galaxy role install process
- extracting geerlingguy.nginx to /home/engineer/ansible-lab/15_galaxy/galaxy_roles/geerlingguy.nginx
- geerlingguy.nginx (3.1.4) was installed successfully
Starting galaxy collection install process
Installing 'ansible.posix:1.7.0' to '/home/engineer/ansible-lab/15_galaxy/collections/ansible_collections/a...
ansible.posix:1.7.0 was installed successfully
Installing 'community.general:11.4.0' to '/home/engineer/ansible-lab/15_galaxy/collections/ansible_collecti...
```

- Номера версий у вас будут свои: galaxy ставит самую свежую версию, которая удовлетворяет диапазону.
- Важно другое — куда всё легло: роль в первый каталог roles_path (galaxy_roles/geerlingguy.nginx), коллекции в первый каталог collections_path (collections/ansible_collections/...).
- Предупреждение о несуществующем пути — не ошибка.
- На управляющей машине нет каталога /usr/share/ansible/roles: он есть только там, где роли ставит пакет операционной системы, например на Rocky Linux.
- У своей роли версия unknown: её никто не ставил с Galaxy, у неё нет файла meta/.galaxy_install_info.

▸ **ЕСЛИ ИНТЕРНЕТА НЕТ: Установка закончится ошибкой вида «failed to download the file» или «sorry, geerlingguy.nginx was not found on https://galaxy.ansible.com». Файлы практики при этом остаются рабочим...**

▸ **ПРОВЕРКА: повторный запуск той же команды ничего не скачивает:**

# ШАГ 5. ГОТОВАЯ РОЛЬ: GEERLINGGUY.NGINX И ПОРТ 8080

- В практике 10 мы написали свою роль веб-сервера.
- Теперь возьмём готовую: geerlingguy.nginx.
- Она ставит nginx и ведёт его конфигурацию; подробности — в README роли, который лежит рядом с её задачами в galaxy_roles/geerlingguy.nginx.
- Плейбук из files/15_galaxy_collections/01_galaxy_role.yml:
  - name: nginx из Galaxy на машинах группы web
  - hosts: web
  - become: true
  - gather_facts: true
  - vars:
  - # Документированные переменные роли geerlingguy.nginx.
  - # nginx_remove_default_vhost убирает виртуальный хост, который
  - # пакет nginx ставит сам: тот слушает порт 80 и мешает apache2.
  - nginx_remove_default_vhost: true
  - nginx_service_state: started
  - nginx_service_enabled: true
  - # Свой виртуальный хост: слушает 8080 и отдаёт ту же страницу,

# ШАГ 6. СВОЯ РОЛЬ ИЛИ ГОТОВАЯ: ПРАВИЛО ВЫБОРА

- Теперь у нас два веб-сервера и две роли.
- Сравним их по одному признаку: сколько работы делает каждая и кто за неё отвечает.
  - своя роль web_admin: восемь задач, шаблон страницы, обработчик.
  - Пишем сами, потому что нужен именно наш сайт, наш владелец
  - машины на странице и наши имена переменных;
  - готовая geerlingguy.nginx: ставит nginx, ведёт конфигурацию,
  - умеет виртуальные хосты и upstream. Пишем не мы, но отвечает
  - ОС, разбор ошибок.
- Правило выбора, которым пользуются в работе:
  - БЕРЁМ ГОТОВОЕ, если роль делает ровно то, что нужно, и её кто-то
  - принимает исправления. У готовой роли есть версии, README
  - и история изменений — значит, её поведение предсказуемо;
  - ПИШЕМ САМИ, если нужен контроль над поведением: свои шаблоны,
  - свои права, свой порядок задач, своя логика по семействам ОС.
  - Готовая роль с десятками переменных ради двух задач — лишняя
  - зависимость и лишний повод разбираться в чужой логике.

# ШАГ 7. СВОЙ СКЕЛЕТ РОЛИ: --ROLE-SKELETON

- В практике 10 роль создавалась командой ansible-galaxy role init с готовым набором каталогов.
- Этот набор можно заменить своим: в команде есть ключ --role-skeleton.
- В скелете лежат те же файлы, что появятся в новой роли, но с расширением .j2: такие файлы Ansible рендерит как шаблоны Jinja2 и отбрасывает расширение.
- Заготовка лежит в files/15_galaxy_collections/skeleton.
- Вот её описание роли — файл skeleton/meta/main.yml.j2:

```
galaxy_info:
role_name: "{{ role_name }}"
author: "{{ author }}"
description: "{{ description }}"
company: "{{ company }}"
license: "{{ license }}"
min_ansible_version: "{{ min_ansible_version }}"
platforms:
```

- Строковые значения взяты в кавычки намеренно: описание роли вполне может содержать двоеточие с пробелом, а без кавычек YAML на этом ломается.
- Имена переменных role_name, author, license и остальные подставляет сама команда role init.
- Второй файл заготовки — skeleton/README.md.j2: шапка README с подстановками role_name, description, author и license, а ниже разделы Requirements, Role Variables, Dependencies, Example Playbook, License и...

# ЧАСТЬ 2. КОЛЛЕКЦИИ, СИСТЕМНЫЕ РОЛИ И ДОКУМЕНТИРОВАНИЕ

-

# ШАГ 8. КОЛЛЕКЦИИ: УСТАНОВКА, СПИСОК, ПРОВЕРКА, --FORCE

- Коллекция — это набор модулей, плагинов и ролей в одном пакете.
- Пространство имён (namespace) стоит в имени первым: ansible.posix, community.general, ansible.netcommon.
- Первое слово — кто отвечает за содержимое, второе — что внутри.

```
ansible-galaxy collection install community.general
ansible-galaxy collection install 'community.general:>=8.0.0,<12.0.0'
ansible-galaxy collection install -r requirements.yml
ansible-galaxy collection install community.general --force
ansible-galaxy collection install community.general --upgrade
```

- Двоеточие с диапазоном в командной строке берётся в кавычки: знаки больше и меньше оболочка понимает по-своему.

```
ansible-galaxy collection list
# /home/engineer/ansible-lab/15_galaxy/collections/ansible_collections
Collection Version
```

- Проверка целостности установленного — команда verify: она сравнивает контрольные суммы файлов на диске с тем, что записано в MANIFEST.json коллекции, и при необходимости сверяется с сервером.
- Ключ --offline проверяет только локальные файлы, без обращения к Galaxy.
- Коллекцию, поставленную из каталога или из git, так не проверить: для неё в MANIFEST.json нет данных о сервере.
- ЧТО ЗНАЧИТ --FORCE.

# ШАГ 9. FQCN: ПОЧЕМУ ПОЛНОЕ ИМЯ НАДЁЖНЕЕ КОРОТКОГО

- Модуль можно назвать коротко, а можно полным именем (Fully Qualified Collection Name) — с пространством имён и коллекцией:
  `ufw короткое имя`
  `community.general.ufw полное имя`
- Короткое имя работает, пока коллекция ровно одна.
- Как только в проекте оказываются две коллекции с одинаковым модулем, Ansible выбирает первую по своему порядку поиска, и плейбук начинает вести себя по-разному на разных машинах.
- Ошибки при этом нет: задача выполняется, просто не тем модулем.
- Полное имя однозначно: коллекция названа, значит вопрос «чей модуль» не возникает ни у Ansible, ни у читателя.
- В курсе полные имена используются везде — в том числе в своей роли web_admin, где стоят community.general.ufw и ansible.posix.firewalld.

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: в большом проекте коллекций десятки, и часть имён совпадает. Правило «в плейбуках и ролях — только FQCN» проверяется линтером (практика 16) и снимает целый класс загадочных расхождений...**

# ШАГ 10. REQUIREMENTS.MIN.YML: МИНИМАЛЬНЫЙ НАБОР

- Второй файл требований описывает не «что нам интересно», а «без чего не живёт стенд».
- Это удобно для изолированного контура и для проверки в CI, где лишние загрузки — потерянное время.

```
collections:
- name: ansible.posix
version: ">=1.5.4,<2.0.0"
- name: community.general
version: ">=8.0.0"
ansible-galaxy collection install -r requirements.min.yml
```

- Готовых ролей из Galaxy в этом файле нет: роль nginx — учебный пример, а не зависимость стенда.
- И самого ansible-core здесь быть не может: ansible-galaxy умеет ставить только роли и коллекции, а движок ставится pip в виртуальное окружение (практика 1).
- В больших проектах встречается и третье имя — collections/requirements.yml: файл рядом с каталогом коллекций описывает только коллекции.
- Смысл тот же: один файл и закреплённые версии, только без ролей.

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: два файла требований — обычная практика. requirements.yml описывает всё, что нужно проекту, requirements.min.yml или requirements-prod.yml — узкий список для конкретного контура. Одинаковые...**

# ШАГ 11. ПРОВЕРКА УСТАНОВЛЕННОГО: 03_COLLECTIONS_CHECK.YML

- Смотреть на список коллекций глазами каждый раз неудобно, поэтому проверим его плейбуком.
- Файл files/15_galaxy_collections/03_collections_check.yml:
  - name: Проверить установленные коллекции
  - hosts: localhost
  - gather_facts: false
  - connection: local
  - vars:
  - required_collections:
  - ansible.posix
  - community.general
  - tasks:
  - name: Получить список коллекций в формате JSON
  - ansible.builtin.command:
  - argv:
  - ansible-galaxy
  - collection

# ШАГ 12. СИСТЕМНЫЕ РОЛИ RHEL: ПАКЕТ RHEL-SYSTEM-ROLES

- До сих пор мы брали роли с Galaxy — то есть у сообщества.
- У системных ролей RHEL другой источник: их выпускает вендор, они входят в состав операционной системы и ставятся пакетом.
- На Rocky Linux пакет называется rhel-system-roles.
- Ставим пакет на Rocky (на Ubuntu его нет и не будет: он собран только для семейства RedHat):

```
ssh engineer@db1
sudo dnf install -y rhel-system-roles
```

- Роли из пакета ложатся в системный каталог, который мы уже перечислили в roles_path:

```
ls -d /usr/share/ansible/roles/rhel-system-roles.* | head -3
/usr/share/ansible/roles/rhel-system-roles.certificate
/usr/share/ansible/roles/rhel-system-roles.selinux
/usr/share/ansible/roles/rhel-system-roles.timesync
ansible-galaxy collection list | grep -i rhel
```

- Вторая команда нужна для второго варианта поставки: те же роли вендор отдаёт коллекцией redhat.rhel_system_roles на Automation Hub.
- Если коллекция установлена, роль вызывается полным именем — redhat.rhel_system_roles.timesync.
- Проверьте на своей машине, какой из двух вариантов доступен, и используйте его: имена у них разные, и угадывать тут нечего.
- Плейбук из files/15_galaxy_collections/02_system_role_timesync.yml:

# ШАГ 13. ДОКУМЕНТИРОВАНИЕ СВОЕЙ РОЛИ

- Роль без документации — это роль, которую нельзя передать.
- Два файла закрывают вопрос: meta/argument_specs.yml описывает аргументы так, что их понимает сам Ansible, а README.md — человек.
- Файл roles/web_admin/meta/argument_specs.yml из files/15_galaxy_collections (начало и два аргумента из десяти; остальные описаны так же):

```
# meta/argument_specs.yml — описание аргументов роли web_admin.
#
# Файл читает сам Ansible: перед первой задачей роли он подставляет
# задачу проверки аргументов и сверяет значения с этим описанием.
# Неверный тип или неизвестный аргумент останавливают прогон
# с понятным текстом.
#
# Значения по умолчанию здесь ПОВТОРЯЮТ defaults/main.yml: само
```

- Что происходит при ошибке в аргументах:
  - неверный тип значения — роль останавливается до первой задачи
  - с текстом Validation of arguments failed;
  - неизвестный аргумент, переданный роли, — тоже остановка: в тексте
  - будет список Supported parameters include, то есть перечень

# Если что-то пошло не так

▸ **ПРОБЛЕМА: ERROR! the role 'geerlingguy.nginx' was not found in ...**

```
ERROR! the role 'geerlingguy.nginx' was not found in /home/engineer/ansible-lab/15_galaxy/roles:/home/engin...
```

- Причин две: роль не установлена или лежит не там, где Ansible её ищет.
- Места поиска перечислены в самом сообщении — читайте его буквально, начиная с первого пути.
- Проверяйте по порядку: сначала есть ли роль на диске, потом попадает ли её каталог в список.

```
ls -d galaxy_roles/*/ roles/*/ 2>/dev/null
ansible-config dump | grep -i roles_path
ansible-galaxy role list
```

- Три частые причины: роль ставили без -r requirements.yml, то есть в ~/.ansible/roles, а roles_path в ansible.cfg этот каталог больше не содержит
- в requirements.yml стоит другое имя (geerlingguy.nginx против nginx); роль поставили в каталог, которого нет в roles_path.

▸ **ПРОБЛЕМА: конфликт версий коллекций**

```
ERROR! Failed to resolve the requested dependencies map. Could not guarantee that all requirements are comp...
```

- Так Ansible отвечает, когда требования противоречат друг другу: например, одна коллекция требует community.general не ниже 12.0.0, а диапазон в requirements.yml ограничен версией ниже 12.0.0.
- Ни одно решение не удовлетворяет обоим условиям сразу, поэтому установка прекращается — и это правильное поведение.
- Проверяйте по порядку: сначала запустите установку с -vvv и посмотрите, какие версии требовались; потом ослабьте диапазон в requirements.yml; в последнюю очередь ставьте конфликтующую коллекцию отдельно.

```
ansible-galaxy collection install -r requirements.yml -vvv
```

# Проверка результата

- Практика выполнена, если верны все шесть пунктов.
- 1.
- Требования установлены и видны:

```
ansible-galaxy role list | grep nginx
- geerlingguy.nginx, 3.1.4
ansible-galaxy collection list | grep community.general
community.general 11.4.0
Номера версий у вас свои; важно, что роль и коллекции нашлись
в каталогах проекта, а не в ~/.ansible.
```

- 2.
- Все три плейбука практики проходят проверку синтаксиса:

```
ansible-playbook --syntax-check 01_galaxy_role.yml
ansible-playbook --syntax-check 02_system_role_timesync.yml
```

- 3.
- Готовая роль из Galaxy работает рядом со своей:
- 4.

# Что дальше

- Далее в курсе:
  - практика 16 — качество автоматизации и динамическая инвентаризация:
  - ansible-lint проверит ту же роль web_admin по правилу fqcn,
  - molecule прогонит её на машинах стенда (драйвер delegated —
  - это не песочница, машины настоящие), а аргументы, описанные
  - здесь, станут частью проверок;
  - приложение 1 — шпаргалка по командам: там же собраны ключи
  - ansible-galaxy, разобранные в этой практике.
- ПЕРЕД ПЕРЕХОДОМ уберите за собой:
  - пробную роль из скелета: rm -rf ~/ansible-lab/15_galaxy/my_role;
  - nginx на web1 и web2 останавливать не нужно: он слушает 8080,
  - apache2 на 80 работает как раньше. Если nginx больше не нужен,
  - уберите его одной командой:
  - ssh engineer@web1 'sudo systemctl disable --now nginx';
  - на db1 конфигурация chrony заменена системной ролью. Вернуть
  - до практики, и это единственный надёжный способ отката;

Учебный центр АйТи Клауд

# Практика 16. Качество автоматизации и инвентаризация

**ЭТА ПРАКТИКА — ИТОГОВАЯ РАБОТА ТРЕКА «5 ДНЕЙ».**

современные технологии простым языком



# Обзор практики

- ЭТА ПРАКТИКА — ИТОГОВАЯ РАБОТА ТРЕКА «5 ДНЕЙ».
- Мы проверим роль web_admin инструментами: ansible-lint найдёт ошибки в описаниях, molecule проверит идемпотентность и результат, а внешний инвентарь заменит файл со списком машин, который мы вели руками с...

## Что мы сделаем

- Поставим ansible-lint и molecule в то же окружение, где живёт Ansible, и разберём вывод линтера: код правила, файл, строку, серьёзность нарушения.
- Соберём черновик плейбука с типичными ошибками и разберём девять правил, которые на нём сработают.
- Настроим .ansible-lint: что выключаем совсем, что оставляем предупреждением и почему содержательные правила отключать нельзя.
- Напишем сценарий molecule с драйвером delegated, прогоним converge, idempotence и verify.
- Затем заменим статический инвентарь внешним: напишем inventory/stand.py, разберём формат ответа (_meta.hostvars, группы, hosts, children) и включим плагин в ansible.cfg.
- Закончим прогоном итоговой работы по критериям трека.

# Что понадобится

- СТЕНД ИЗ ПРАКТИК 1–15.
- Четыре машины собраны, вход по ключу работает:

```
control 10.0.2.5 — управляющая машина, здесь мы всё выполняем
web1 10.0.2.11 — Ubuntu 26.04 LTS
web2 10.0.2.12 — Ubuntu 26.04 LTS
db1 10.0.2.13 — Rocky Linux 10
```

- Учётная запись engineer, ключ /home/engineer/.ssh/id_ed25519, учебный пароль Console47 (он опубликован потому, что стенд изолирован).
- ЧТО ДОЛЖНО БЫТЬ ГОТОВО ЗАРАНЕЕ:
  - интернет: ansible-lint и molecule ставятся из PyPI;
  - SSH с control на web1 и web2: molecule будет работать на них
  - по-настоящему, а не в песочнице.
- ФАЙЛЫ ИЗ КАТАЛОГА КУРСА лежат в files/16_quality_inventory — это готовый проект целиком; каждый файл напечатан в шаге, где разбирается.
- Роль web_admin — та же, что в files/10_roles/roles/ web_admin: её файлы напечатаны в практике 10 и здесь не повторяются.
- ЕДИНСТВЕННЫЙ ИСПОЛНЯЕМЫЙ СКРИПТ КУРСА — внешний инвентарь files/16_quality_inventory/inventory/stand.py: по общему правилу готовых скриптов в files/ нет, но здесь скрипт и есть предмет изучения.
- Ему нужен бит исполнения, и мы его выставим.
- ▸ **ВРЕМЯ: два академических часа.**

# ЧАСТЬ 1. ЛИНТЕР, БЫСТРЫЕ ПРОВЕРКИ И MOLECULE

-

# ШАГ 1. СТАВИМ ANSIBLE-LINT И MOLECULE

- Оба инструмента — программы на Python, и ставятся они в то же виртуальное окружение, что и Ansible:

```
source ~/ansible-venv/bin/activate
pip install ansible-lint molecule
ansible-lint --version
ansible-lint 25.8.1 using ansible-core:2.19.0 ansible-compat:25.6.0
molecule --version
molecule 25.6.1 using python 3.13
```

- Курс рассчитан на ansible-lint 25.x и molecule 25.x; мелкие версии выходят каждые несколько недель, поэтому номера у вас будут другие.

▸ **ПОЧЕМУ ИМЕННО ТАК: линтер и molecule проверяют тот же Ansible, который запускает плейбуки: в другом окружении они увидят другую версию ansible-core и другой разбор YAML.**

▸ **ЧЕГО НЕ ДЕЛАТЬ: не ставьте molecule с драйвером docker или podman: в стенде нет Docker. И не запускайте pip install от root: пакеты уйдут в системный Python, а в Ubuntu 26.04 это запрещено (PEP 668).**

# ШАГ 2. КОПИРУЕМ ПРОЕКТ И ДЕЛАЕМ ПЕРВЫЙ ПРОГОН

- Скопируем проект и выставим бит исполнения внешнему инвентарю: без него Ansible его не запустит (разбор — в конце практики).

```
mkdir -p ~/ansible-lab/16_quality_inventory
cp -r /путь/к/репозиторию/files/16_quality_inventory/. \
~/ansible-lab/16_quality_inventory/
cd ~/ansible-lab/16_quality_inventory
chmod +x inventory/stand.py
```

- ВСЕ КОМАНДЫ ДАЛЬШЕ выполняются из каталога проекта ~/ansible-lab/16_quality_inventory: ansible-lint, Ansible и molecule ищут свои настройки в текущем каталоге.
- Единственное исключение — сравнение с каталогом практики 10: оно помечено ниже, и после него мы возвращаемся в проект.

```
cd ~/ansible-lab/16_quality_inventory
ansible-lint
Passed: 0 failure(s), 0 warning(s) on 18 files.
```

- 18 файлов — это весь каталог проекта: роль, сценарий molecule, инвентарь и настройки.
- Чисто потому, что в проекте уже лежит файл .ansible-lint (шаг 4): правило name[casing] в нём выключено.
- Пустой вывод — плохой учебный материал, поэтому посмотрим, как выглядит вывод с нарушениями.
- Роль там та же, а настроек нет: в каталоге ~/ansible-lab/10_roles файла .ansible-lint нет, и выше, в ~/ansible-lab, линтер его тоже не найдёт.
- Здесь файлов 14, а не 18: в каталоге практики 10 нет ни сценария molecule, ни инвентаря.

# ШАГ 3. ДЕВЯТЬ ПРАВИЛ НА ЧЕРНОВИКЕ

- Правила удобнее разбирать на маленьком файле, чем на роли.
- Черновик создаётся в корне проекта — рядом с .ansible-lint, поэтому сразу видно, какие правила проект погасил:

```
cat > draft_lint.yml <<'EOF'
# draft_lint.yml — черновик для разбора вывода ansible-lint.
- hosts: web
tasks:
- name: Установить пакет mc
apt:
name: mc
state: present
```

- Черновик лежит в проекте, поэтому линтер проверяет его вместе с остальными файлами — запуск без имени файла:
- 19 файлов — это 18 файлов проекта (шаг 2) и сам черновик.
- В отчёте девять строк, а нарушений в черновике десять: строки key-order[task] в отчёте нет — правило выключено в skip_list проекта (шаг 4), хотя модуль ansible.builtin.dnf в черновике записан раньше name.
- Значит, десять нарушений — девять разных правил: no-changed-when попался дважды.
- Из девяти показанных строк восемь — нарушения, а jinja[spacing] взят из warn_list: линтер печатает его в общем списке, но прогон из-за него не валится (в отчёте это видно как 1 warning(s)).
- NAME[PLAY] — у play нет имени: в выводе вместо «PLAY [Веб-сервер и сайт]» появится «PLAY [web]».

# ШАГ 4. .ANSIBLE-LINT: SKIP_LIST, WARN_LIST, EXCLUDE_PATHS

- Настройки линтера лежат в файле .ansible-lint в корне проекта.
- Вот он целиком — files/16_quality_inventory/.ansible-lint:

```
---
# .ansible-lint — настройки линтера для проекта практики 16.
# Линтер ищет файл в текущем каталоге и выше: запускать из корня.
# Профиль не задаём: профиль — правила «для публикации в Galaxy».
profile: null
# Пути, которые линтер не открывает: кэш и служебный файл molecule.
exclude_paths:
- .cache/
```

- Три ключа делают три разные вещи: EXCLUDE_PATHS — линтер не открывает файлы (так убирают чужой код); SKIP_LIST — правило не работает; WARN_LIST — правило работает, нарушение видно, но прогон не валится.

▸ **ПОЧЕМУ КОСМЕТИКУ МОЖНО ОТКЛЮЧИТЬ: Name[casing] и key-order[task] — про то, как записан текст, а не про то, что делает роль: имена обработчиков у нас начинаются с имени роли (web_admin: ...), а в черновике шага...**

▸ **ПОЧЕМУ СОДЕРЖАТЕЛЬНЫЕ НЕЛЬЗЯ: Command-instead-of-module и no-changed-when ловят потерю идемпотентности: плейбук с такой задачей формально работает, а на втором прогоне сообщает changed=1 там, где ничего не...**

▸ **ВАЖНО: списки из файла и из командной строки складываются, а не заменяют друг друга: флаг --skip-list добавит правило к перечисленным.**

▸ **ЧЕГО НЕ ДЕЛАТЬ: не пишите skip_list: [all] и не отключайте правило из-за одной задачи — правьте задачу.**

# ШАГ 5. БЫСТРЫЕ ПРОВЕРКИ: SYNTAX-CHECK, CHECK, DIFF, LIST-TASKS

- Линтер проверяет описание, но не запускает его.
- Перед прогоном полезны четыре команды ansible-playbook.

```
ansible-playbook --syntax-check site.yml
playbook: site.yml
ansible-playbook --list-tasks site.yml
tasks:
web_admin : Установить веб-сервер
```

- РЕЖИМ ПРОВЕРКИ И РАЗЛИЧИЯ:

```
ansible-playbook --check --diff site.yml
--- before: /var/www/html/index.html
+++ after: /var/www/html/index.html
```

- ЧЕГО --CHECK НЕ ГАРАНТИРУЕТ: он не доказывает, что изменений не будет.
- Задачи, которые режим проверки не поддерживают, будут пропущены со словом skipped, а не проверены.
- Часть модулей в режиме проверки видит не то состояние, что при обычном прогоне, поэтому changed может появиться там, где ничего не изменится, и наоборот.
- Вывод --check полезен как быстрый фильтр, но не заменяет ни прогон, ни тест.

# ШАГ 6. MOLECULE: ЗАЧЕМ И КАК УСТРОЕН СЦЕНАРИЙ

- Линтер проверяет, как роль написана, но ничего не знает о том, работает ли она.
- Для этого нужен прогон, и не один: ПЕРВЫЙ проверяет, что роль выполняется и делает обещанное (служба запущена, страница на месте, порт отвечает), ВТОРОЙ — идемпотентность: правильно написанная роль во второй...
- Это тот самый changed=0, которого мы добивались вручную в практиках 4–13.
- Molecule — обвязка для таких прогонов.
- СЦЕНАРИЙ — это каталог molecule/default внутри проекта: molecule.yml (что за машины и чем их проверять), converge.yml (что применяем), prepare.yml (что сделать до применения) и verify.yml (что считаем успехом).
- Действия molecule — это глаголы, и они же шаги последовательности:

```
molecule create подготовить машины сценария
molecule converge применить converge.yml
molecule idempotence применить ещё раз и проверить changed=0
molecule verify выполнить verify.yml
molecule destroy убрать за собой машины
molecule test всё перечисленное подряд
```

# ШАГ 7. MOLECULE.YML: ДРАЙВЕР DELEGATED

- Драйвер — то, что создаёт машины для сценария.
- Драйвер docker поднял бы контейнер, драйвер delegated не создаёт ничего: он работает на хостах, которые уже есть.
- В стенде курса нет Docker — это первая причина.
- Вторая важнее: роль управляет службами через ansible.builtin.systemd_service, включает ufw и firewalld, а контейнер без systemd — не та машина, где эту роль имеет смысл проверять.
- Delegated работает на настоящих web1 и web2 из стенда.
- Файл files/16_quality_inventory/molecule/default/molecule.yml:

```
---
# molecule/default/molecule.yml — сценарий проверки роли web_admin.
# Запуск из каталога проекта: molecule test
# Драйвер delegated: molecule ничего не создаёт, а работает
# на хостах, которые уже есть. В стенде курса нет Docker.
driver:
name: delegated
options:
```

- ▸ **ВАЖНО: в режиме delegated машины настоящие. Molecule на них всё меняет по-настоящему: ставит пакеты, включает ufw, перезапускает службы. Это не песочница, а тот же стенд, на котором вы работали все пятнадцать...**
- ▸ **ЧЕГО НЕ ДЕЛАТЬ: не запускайте сценарий с managed: true — molecule попытается выполнить create и destroy, которых для delegated нет.**

# ШАГ 8. CONVERGE.YML И PREPARE.YML

- CONVERGE.YML — что применяем.
- Это обычный плейбук, и он делает то же, что site.yml из практики 10:

```
---
# converge.yml — то, что molecule применяет к машинам сценария:
# обычный плейбук, как site.yml. Переменные берутся из group_vars
# инвентаря, который строит molecule.
- name: Проверить роль web_admin на узлах стенда
hosts: all
become: true
gather_facts: true
```

- Хост all — это машины из platforms, то есть web1 и web2: плейбук не знает про конкретные имена и остаётся переносимым.
- PREPARE.YML — что сделать до применения.
- Файл files/16_quality_inventory/molecule/default/prepare.yml:

▸ **ПОЧЕМУ ИМЕННО ТАК: в сценарии с delegated никто не мешает указать в platforms имя чужой машины. Проверка адреса и семейства превращает эту ошибку в понятное сообщение на первом шаге.**

# ШАГ 9. ПРОГОН: CONVERGE И IDEMPOTENCE

```
cd ~/ansible-lab/16_quality_inventory
molecule converge
TASK [web_admin : Установить веб-сервер] ***
changed: [web1]
PLAY RECAP ***
web1 : ok=12 changed=6 unreachable=0 failed=0 skipped=0
web2 : ok=12 changed=6 unreachable=0 failed=0 skipped=0
```

- Шесть изменений на каждой машине: пакеты, страница, служба, порт.
- Теперь то, ради чего molecule и берут:
  `molecule idempotence`
- ЧТО ИМЕННО ПРОВЕРЯЕТ IDEMPOTENCE.
- Molecule запускает converge.yml второй раз и смотрит на итог: если хотя бы одна задача сообщила changed, проверка падает и molecule показывает, какая именно.
- Changed=0 означает, что роль описывает состояние, а не выполняет действия: пакет уже стоит, страница уже такая, служба уже запущена, менять нечего.
- Именно поэтому задачи command и shell без changed_when (шаг 3) ломают эту проверку всегда.
- ▸ **ПОЧЕМУ ИМЕННО ТАК: руками идемпотентность проверяют так же — второй прогон и взгляд на changed=0. Molecule делает это за вас и делает каждый раз: забыть проверку невозможно, она часть последовательности test....**

# ШАГ 10. VERIFY.YML: ПРОВЕРКА РЕЗУЛЬТАТА ЧЕРЕЗ ASSERT

- Converge показывает, что задачи выполнились.
- Проверка результата — отдельный файл: ansible.builtin.assert с условиями, которые должны быть истинны после работы роли.
- Файл files/16_quality_inventory/ molecule/default/verify.yml:

```
---
# verify.yml — проверка результата после converge: служба запущена,
# файл сайта на месте, порт 80 отвечает. Каждая проверка —
# ansible.builtin.assert: если условие ложно, play падает с fail_msg.
- name: Проверить, что роль web_admin сделала обещанное
hosts: all
become: false
gather_facts: false
```

- ▸ **ЧТО ЗДЕСЬ ВАЖНО: сначала задача СОБИРАЕТ ФАКТЫ (service_facts), потом assert их проверяет: спрашивать систему он не умеет. Список that читается как «и», а порт проверяется запросом по сети: служба может быть...**
- Целиком сценарий запускается командой molecule test: разбор, применение, повторный прогон, проверки, уборка.
- ▸ **ЧЕГО НЕ ДЕЛАТЬ: не превращайте verify.yml в пересказ задач роли. Проверять надо то, что видно снаружи: служба, файл, порт, ответ по сети. Проверка «задача установки пакета сказала ok» ничего не добавляет:...**

# ЧАСТЬ 2. ДИНАМИЧЕСКАЯ ИНВЕНТАРИЗАЦИЯ И ИТОГОВАЯ РАБОТА

-

# ШАГ 11. ЗАЧЕМ НУЖЕН ДИНАМИЧЕСКИЙ ИНВЕНТАРЬ

- С практики 2 у нас есть файл inventory.ini со списком машин.
- Он удобен, пока машины не меняются.
- В реальной работе парк меняется сам:
  - машины создаёт и удаляет облако, и список известен только ему;
  - у каждой машины свои адреса, зоны, группы безопасности и теги,
  - и появляются они вместе с машиной, а не в тот день, когда кто-то
  - вспомнил поправить файл;
  - файл в репозитории устаревает молча: Ansible не знает, что
  - в списке нет двух машин, и просто не работает с ними.
- Динамическая инвентаризация — это когда список машин Ansible не читает, а спрашивает.
- Отвечает на вопрос «плагин инвентаря»: программа, которая умеет получить список из своего источника.
- В курсе взят самый наглядный источник — собственный скрипт: он делает то же, что облачные плагины, только вместо API облака читает словарь в Python.

# ШАГ 12. INVENTORY/STAND.PY: ВНЕШНИЙ ИНВЕНТАРЬ

- Внешний инвентарь подключается плагином ansible.builtin.script: Ansible запускает файл как программу и читает JSON из её вывода, а язык программирования ему безразличен.
- Файл files/16_quality_inventory/ inventory/stand.py:

```
#!/usr/bin/env python3
"""Внешний инвентарь учебного стенда: плагин ansible.builtin.script.
Ansible вызывает файл двумя способами:
inventory/stand.py --list — отдай весь инвентарь;
inventory/stand.py --host web1 — отдай переменные одного хоста.
Ответ на --list — один объект JSON: ключи, кроме служебного _meta, —
имена групп (у группы бывают hosts, vars и children), а в _meta.hostvars
лежат переменные хостов: так Ansible получает их одним запросом.
```

- ▸ **ПОЧЕМУ ИМЕННО ТАК: функций две, и это не формальность. Inventory() только собирает данные и ничего не печатает — её можно вызвать из другого скрипта и проверить, что получилось. Main() только печатает. Если...**
- ▸ **ВАЖНО: скрипт печатает в stdout ровно один объект JSON. Любая отладочная печать попадёт в тот же поток, и Ansible получит не разбираемый текст. Проверить скрипт можно без Ansible:**
- Если JSON не разбирается, искать причину в Ansible бессмысленно.

# ШАГ 13. ФОРМАТ ОТВЕТА: _META, ГРУППЫ, HOSTS, CHILDREN

- Формат ответа — часть контракта с Ansible, и в нём четыре вида записей.
- КЛЮЧИ ВЕРХНЕГО УРОВНЯ — ИМЕНА ГРУПП: web, db, linux, debian, redhat.
- Совпадать с прежним инвентарём они обязаны: плейбуки написаны как hosts: linux или hosts: web, и группа с другим именем означает «no hosts matched».
- **У ГРУППЫ ТРИ ВОЗМОЖНЫХ ПОЛЯ: HOSTS (список машин группы), CHILDREN (группы внутри группы: так linux собирает web и db, a debian и redhat — по семейству ОС) и VARS (переменные группы; у нас их нет).**
- _META.HOSTVARS — переменные хостов, единственное место, куда попадают ansible_host, ansible_user и ansible_ssh_private_key_file.
- Без _meta Ansible не знает переменных хостов и вынужден спрашивать их отдельно: запускать скрипт с ключом --host для каждой машины.
- На трёх машинах это незаметно, на трёхстах — триста запусков вместо одного.
- Поэтому _meta первое, что проверяют, когда инвентарь «работает не так»: машины находятся, а переменные пустые.
- **ЧЕГО НЕ ДЕЛАТЬ: не возвращайте пароль в hostvars и не храните его в скрипте. Динамический инвентарь — обычный файл в репозитории, его читают все. Подключение у нас по ключу, а секреты живут в Vault (практика...**

# ШАГ 14. ПОДКЛЮЧАЕМ ПЛАГИН: ANSIBLE.CFG И ANSIBLE-INVENTORY

- Чтобы Ansible взял инвентарь из скрипта, в ansible.cfg проекта достаточно строки inventory.
- Плагин ansible.builtin.script включён в ansible-core по умолчанию, но мы перечислим плагины явно: так видно, откуда берутся машины.
- Файл files/16_quality_inventory/ansible.cfg:

```
# ansible.cfg — настройки Ansible для проекта практики 16.
# Команды выполняются из каталога проекта: этот файл побеждает
# остальные три места, где Ansible ищет настройки.
[defaults]
# Инвентарь — не файл, а внешний скрипт: Ansible читает JSON
# из его вывода. Подробно — в шагах 11–15.
inventory = inventory/stand.py
host_key_checking = False
```

- ▸ **ВАЖНО: ЧИТАЕМЫЙ ВЫВОД В ANSIBLE-CORE 2.19 ВКЛЮЧАЕТСЯ ИНАЧЕ. Invalid callback for stdout specified: yaml». Нужны две строки, и обе у нас есть: stdout_callback = default и callback_result_format = yaml.**
- ПРОВЕРКА — то же, что в практике 2, но теперь по внешнему инвентарю:
- Три машины: db1, web1, web2 — как в статическом инвентаре.
- ▸ **ЧЕГО НЕ ДЕЛАТЬ: не перечисляйте в enable_plugins только script — тогда перестанет работать статический инвентарь из шага 15.**

# ШАГ 15. СТАТИЧЕСКИЙ ИНВЕНТАРЬ ДЛЯ СРАВНЕНИЯ

- Тот же стенд, записанный руками: files/16_quality_inventory/ inventory/stand.ini.

```
# inventory/stand.ini — тот же стенд статически, для сравнения
# с внешним инвентарём (шаг 15): группы и адреса те же.
[web]
web1 ansible_host=10.0.2.11
web2 ansible_host=10.0.2.12
[db]
db1 ansible_host=10.0.2.13
[linux:children]
```

- ПРОВЕРКА — два инвентаря должны дать одно дерево:
- Дерево групп совпадает: web, db, linux, debian, redhat.
- ▸ **ВАЖНО: параметр -i в командной строке перекрывает настройку inventory из ansible.cfg: в проверках выше мы указываем -i, чтобы сравнить два источника, а не тот, что выбран в проекте.**

# ШАГ 16. КЭШ ИНВЕНТАРЯ И ОБЛАЧНЫЕ ПЛАГИНЫ

- Кэш инвентаря выключен по умолчанию, и не зря: смысл у него один — не обращаться к источнику лишний раз.
- Для скрипта на Python это экономия на спичках, для плагина, который ходит в API облака, — заметная.
- Включается он тремя настройками в секции [inventory], и все три у нас в ansible.cfg:
  - CACHE = TRUE — кэш включён;
  - CACHE_PLUGIN = ANSIBLE.BUILTIN.JSONFILE — кэш лежит в файлах рядом
  - с проектом (есть и другие плагины кэша: redis, memcached);
  - CACHE_TIMEOUT = 3600 — сколько секунд считать кэш свежим. Ноль
  - означает «кэш бессмертен», и это почти всегда ошибка.

▸ **КЭШ РАБОТАЕТ НЕ У ВСЕХ ПЛАГИНОВ: включение кэша в ansible.cfg само ничего не кэширует. Проверить свой плагин можно по справке:**

```
ansible-doc -t inventory ansible.builtin.script
```

- У плагина script опции кэша нет: он запускает наш скрипт каждый раз.
- А у плагинов, которые ходят в облако, кэш есть — и там он и помогает, и мешает: список машин в кэше остаётся таким, каким был при первом обращении.

▸ **ГДЕ ТО ЖЕ САМОЕ В РЕАЛЬНОЙ РАБОТЕ: Механизм один, источники разные:**

  - AWS — плагин aws_ec2 из коллекции amazon.aws: машины берутся
  - из API, теги становятся группами, регион и зона — переменными;
  - Azure — плагин azure_rm из коллекции azure.azcollection;

# ШАГ 17. ИТОГОВАЯ РАБОТА ТРЕКА «5 ДНЕЙ»

- Плейбук проекта — files/16_quality_inventory/site.yml:

```
---
# site.yml — точка входа проекта. Инвентарь берётся из ansible.cfg:
# inventory = inventory/stand.py, то есть список машин приходит
# от внешнего скрипта, а не из файла.
- name: Веб-сервер и сайт на узлах стенда
hosts: linux
become: true
gather_facts: true
```

- Прогон по внешнему инвентарю (переменные стоят в play: файл один):
- Второй прогон — проверка идемпотентности:
- КРИТЕРИИ ИТОГОВОЙ РАБОТЫ из ПРОГРАММА_5_дней.md, раздел 6:
  - РОЛЬ ПРОХОДИТ ЛИНТЕР — ansible-lint не выдаёт ошибок уровня fatal;
  - ПРОВЕРКА ИДЕМПОТЕНТНОСТИ — molecule idempotence, changed=0;
  - ПРОВЕРКА РЕЗУЛЬТАТА — molecule verify выполняет assert-проверки;
  - ВНЕШНИЙ ИНВЕНТАРЬ РАБОТАЕТ — ansible-inventory --graph показывает

# Если что-то пошло не так

▸ **ПРОБЛЕМА: ansible-lint валит прогон на файлах, которые мы сами набрали, и на чужом коде.**

```
Failed: 8 failure(s), 1 warning(s) on 19 files.
```

• Это нормальное поведение линтера, а не поломка: так выглядит прогон с черновиком шага 3 (18 файлов проекта и сам черновик).

• Порядок разбора:

– Посмотрите, к какому файлу относится нарушение: черновик
– draft_lint.yml (шаг 3) набран с ошибками нарочно.
– Отделите свой код от чужого: нарушения в чужой роли или в
– коллекции исправить нельзя — такие каталоги убирают
– в exclude_paths, а не в skip_list.
– Содержательные правила (command-instead-of-module,
– no-changed-when, risky-file-permissions) чинят; косметические
– (name[casing], key-order) можно выключить осознанно. Линтер
– должен быть запущен из корня проекта.

▸ **ПРОБЛЕМА: Driver delegated not found**

```
CRITICAL Driver delegated not found
```

• Драйвера с таким именем molecule не нашла.

# Проверка результата

- Работа сделана, если выполняются все пункты.
- 1.
- Инструменты стоят в том же окружении, что Ansible, и линтер

```
в каталоге проекта не находит нарушений:
cd ~/ansible-lab/16_quality_inventory
ansible-lint --version
ansible-lint 25.8.1 using ansible-core:2.19.0 ...
ansible-lint
Passed: 0 failure(s), 0 warning(s) on 18 files.
```

- 2.
- Роль идемпотентна: второй прогон ничего не меняет.

```
molecule idempotence
INFO Idempotence completed successfully.
```

- 3.
- Проверки результата проходят и падают, когда результат сломан:
- 4.

# Что дальше

- Это последняя практика трека «5 дней»: материал курса закончен.
- Дальше — приложения, к которым возвращаются в работе:
  - приложение 1 — шпаргалка по командам курса (ansible, ansible-doc,
  - ansible-vault, ansible-galaxy, ansible-lint, molecule);
  - приложение 2 — частые ошибки: блоки ПРОБЛЕМА из всех практик
  - и порядок диагностики «с чего начать, если плейбук не работает»;
  - приложение 3 — Windows и сетевое оборудование: чего в курсе нет.
- ПЕРЕД ПЕРЕХОДОМ уберите за собой мусор прогонов:

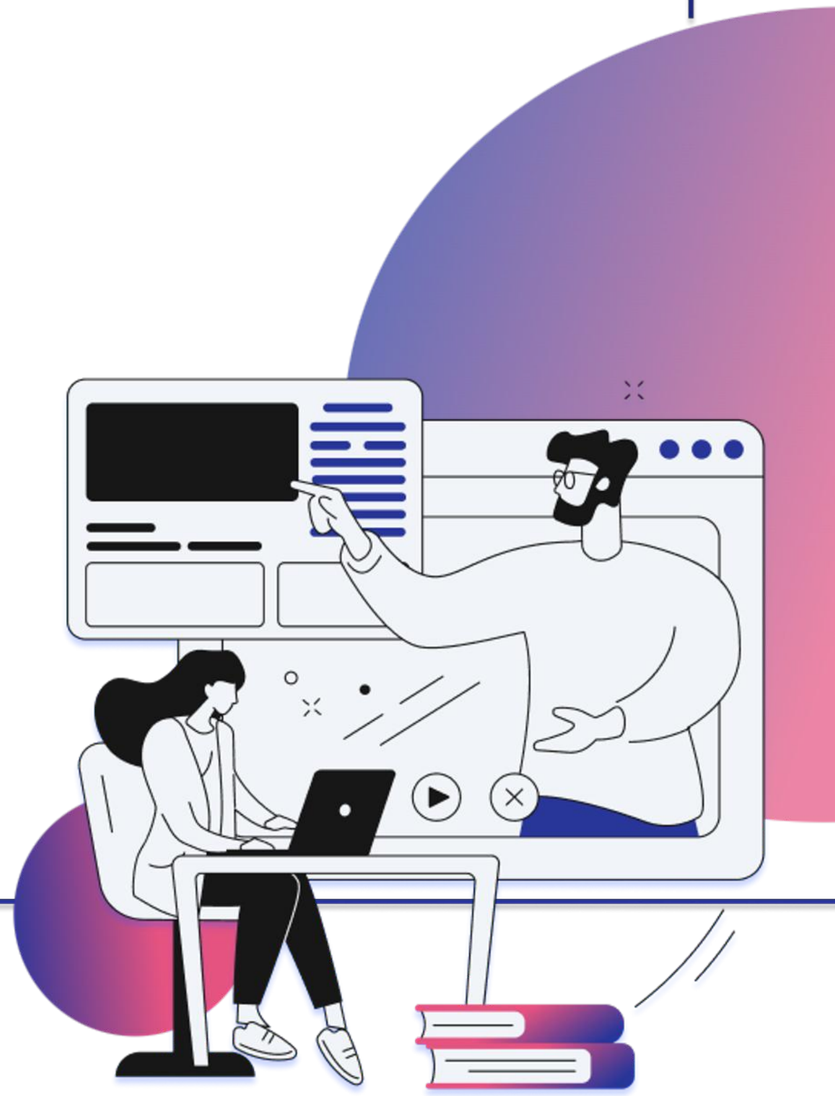
```
cd ~/ansible-lab/16_quality_inventory
rm -rf .cache/ molecule/default/.molecule/
```

Учебный центр АйТи Клауд

# Приложение 1. Шпаргалка по командам Ansible

**ЭТО КАРМАННЫЙ СПРАВОЧНИК К КУРСУ: КОМАНДЫ, КЛЮЧИ И МОДУЛИ ПО ТЕМАМ ПРАКТИК 1-16.**

современные технологии простым языком



# Обзор практики

- ЭТО КАРМАННЫЙ СПРАВОЧНИК К КУРСУ: КОМАНДЫ, КЛЮЧИ И МОДУЛИ ПО ТЕМАМ ПРАКТИК 1-16. ГДЕ КЛЮЧ ИЛИ ЗНАЧЕНИЕ В ПРАКТИКАХ НЕ ВСТРЕЧАЮТСЯ, ЭТО СКАЗАНО ПРЯМО, ЧТОБЫ ШПАРГАЛКА НЕ ВЫДАВАЛА ЛИШНЕГО ЗА КУРС. Стенд: control...

# 1. УСТАНОВКА И ОКРУЖЕНИЕ

- Ansible ставится только на control: в репозитории Ubuntu версия отстаёт, установка в системный Python запрещена (PEP 668).

```
sudo apt update
sudo apt install -y python3-venv
python3 -m venv ~/ansible-venv
source ~/ansible-venv/bin/activate
pip install ansible
ansible --version
```

- Окружение активируют в каждом сеансе терминала, иначе «ansible: command not found».
- В выводе --version важны: ansible [core 2.19.0], config file (взятый файл; None — файла нет) и ansible python module location — путь должен вести в ~/ansible-venv.

```
ansible-config dump | grep -i forks
ansible-config dump --only-changed
```

- dump печатает настройки с пометкой (default) или именем файла, из которого значение взято; --only-changed оставляет только отличия; list — полный перечень настроек с описаниями.
- ЧЕТЫРЕ МЕСТА ПОИСКА ansible.cfg, читается только первый найденный:
  - ANSIBLE_CONFIG — переменная окружения, сильнее всех;
  - ./ansible.cfg — файл в текущем каталоге;
  - ~/.ansible.cfg — файл в домашнем каталоге пользователя;

# 2. ИНВЕНТАРЬ

- Инвентарь — машины, группы и переменные.
- В курсе это файл /home/engineer/ansible-lab/inventory.ini, формат INI:

```
[web]
web1 ansible_host=10.0.2.11
web2 ansible_host=10.0.2.12
[db]
db1 ansible_host=10.0.2.13
[linux:children]
web
db
```

- [linux:children] — группа из групп.
- Группа all есть всегда, ungrouped собирает машины без группы; адрес машины — переменная ansible_host.
- Тот же инвентарь бывает в формате YAML (all → children → web → hosts → web1), в курсе применяется INI.
- ПЕРЕМЕННЫЕ ПОДКЛЮЧЕНИЯ лежат в group_vars и host_vars рядом с инвентарём или плейбуком:
- Переменные хоста сильнее групповых; группа all слабее любой другой.
- --graph печатает дерево групп, --list — весь инвентарь в JSON, --host — переменные одной машины, --yaml (не из практик) — то же в YAML.

# 3. AD-HOC КОМАНДЫ

```
ansible <образец машин> -m <модуль> -a "<параметры>" [ключи]
```

- Ключи:
  - -m — модуль; короткие имена (ping, apt) в ad-hoc разрешены;
  - -a — параметры модуля одной строкой в кавычках;
  - -b — become, повышение прав до root через sudo;
  - -K, --ask-become-pass — спросить пароль sudo; на стенде sudo без
  - пароля, поэтому в практиках ключ не применяется;
  - -u engineer — пользователь подключения (задан в ansible.cfg);
  - -i — инвентарь; -f — машин параллельно (forks, умолчание 5);
  - --limit — сузить выбор машин; -v, -vv, -vvv — подробность вывода;
  - -e, --check, --diff — работают и здесь (в практиках — у playbook).
- Примеры на модулях курса:

```
ansible all -m ansible.builtin.ping
ansible linux -m ansible.builtin.command -a "hostname"
ansible web1 -m ansible.builtin.shell -a 'echo $HOME'
ansible web -m ansible.builtin.copy -a "src=/home/engineer/ansible-lab/yaml/01_scalars.yml dest=/tmp/01_sca...
```

# 4. ПЛЕЙБУКИ

```
ansible-playbook site.yml
ansible-playbook site.yml -e web_port=8082
ansible-playbook 03_vars_files.yml -e @extra_vars.yml
```

- Проверки до запуска:

```
ansible-playbook --syntax-check site.yml
ansible-playbook --list-tasks site.yml
ansible-playbook --list-hosts site.yml
ansible-playbook --list-tags site.yml -i inventory/dev.ini
ansible-playbook --check --diff site.yml
```

- Часть плейбука и пароли:
- Ключи, которых в практиках нет, — для справки:
- КОЛОНКИ PLAY RECAP (сводка печатается последней, по каждой машине):
  - ok — задачи завершились успешно; changed — частный случай успеха;
  - changed — задачи изменили состояние машины, это итог прогона;
  - unreachable — к машине не удалось подключиться; failed — задача
  - упала, и play на этой машине остановлен;

# 5. ПЕРЕМЕННЫЕ

- Где определяют переменные: vars в play, vars_files, group_vars/all.yml, group_vars/<группа>.yml, host_vars/<машина>.yml, include_vars, set_fact, register, -e и --extra-vars (например -e @extra_vars.yml).
- ПОРЯДОК ПРИОРИТЕТА от самого слабого к самому сильному:
  - defaults роли; переменные инвентаря и group_vars/all.yml;
  - group_vars/<группа>.yml; host_vars/<машина>.yml;
  - факты о машине и кэшированные set_fact;
  - vars в play; vars_prompt; vars_files; vars роли;
  - vars на блоке задач и на задаче; include_vars;
  - set_fact и register; параметры роли; --extra-vars (сильнее всех).
- Правило: чем ближе переменная к задаче и чем позже появилась, тем сильнее.

```
ansible web1 -m ansible.builtin.debug -a "var=web_package"
- name: Показать значения переменных
ansible.builtin.debug:
msg:
- "машина: {{ inventory_hostname }}"
- "web_port: {{ web_port }}"
- name: Проверить, что семейство ОС поддерживается ролью
```

# 6. ШАБЛОНЫ И ФАКТЫ

– name: Разложить главную страницу из шаблона
– ansible.builtin.template:
– src: index.j2
– dest: "{{ web_admin_site_root }}/{{ web_admin_index_file }}"
– owner: root
– mode: "0644"
– backup: true
– validate: "nginx -t"
– trim_blocks: true
– lstrip_blocks: true
– src ищется в каталоге templates рядом с плейбуком или ролью;
– mode пишется строкой в кавычках: "0644", "0755";
– backup: true оставляет копию прежнего файла с отметкой времени;
– validate — проверка над временным файлом до подмены, место обозначается %s: sshd -t -f %s, visudo -cf %s; у nginx без %s;
– trim_blocks убирает перевод строки после {% %}, lstrip_blocks —
– пробелы перед блоком в начале строки.

# 7. VAULT

```
ansible-vault create secrets_demo.yml
ansible-vault view group_vars/all/vault.yml
ansible-vault edit group_vars/all/vault.yml
ansible-vault encrypt group_vars/all/vault.yml
ansible-vault decrypt group_vars/all/vault.yml
ansible-vault rekey group_vars/all/vault.yml
ansible-vault encrypt_string --name vault_demo_token 'MyStrongPassword'
```

- Файл начинается строкой $ANSIBLE_VAULT;1.1;AES256; семь действий — create, decrypt, edit, encrypt, encrypt_string, rekey, view — печатает ansible-vault --help.
- Пароль передают тремя способами:

```
ansible-playbook 02_register.yml --ask-vault-pass
```

▸ **ВАЖНО: файл с паролем (~/.vault_pass, права 600) в репозиторий не попадает и рядом с зашифрованным файлом не лежит. Имя переменной пишут с префиксом vault_: vault_db_password, а обычная переменная ссылается на...**

# 8. РОЛИ И GALAXY

- Структура роли, которую создаёт ansible-galaxy role init:

```
roles/web_admin/
defaults/main.yml значения, которые предполагается менять
vars/main.yml значения, которые роль фиксирует
tasks/, handlers/, meta/, templates/, files/, README.md
```

- defaults слабее group_vars и host_vars, vars — сильнее их; секреты в vars не хранят.
- Каталог tests удаляют: его место заняла molecule.

```
ansible-galaxy role init roles/web_admin
ansible-galaxy role init --role-skeleton=./skeleton my_role
ansible-galaxy install -r requirements.yml
ansible-galaxy role list
```

# 9. ХРАНИЛИЩЕ: ДИСКИ, LVM, МОНТИРОВАНИЕ

- Практика 13: на web1, web2 и db1 — второй диск /dev/sdb (10 ГБ), раздел, группа томов data, логический том webdata, файловая система (ext4 на Ubuntu, xfs на Rocky) и точка монтирования /srv/data.

```
community.general.parted device, number, label: gpt, name, flags: [lvm]
community.general.lvg vg: data, pvs: [/dev/sdb1]
community.general.lvol vg, lv, size, shrink: false, state: present
community.general.filesystem fstype, dev: /dev/mapper/data-webdata
ansible.posix.mount path, src, fstype, opts, dump, passno
```

- Расширение тома и файловой системы без размонтирования — тот же lvol с resizefs: true: ext4 растёт через resize2fs, xfs — через xfs_growfs и только смонтированным.
- Порядок: место, раздел, pvresize, lvextend.

```
ansible all -b -m ansible.builtin.command -a "lsblk -o NAME,SIZE,TYPE,FSTYPE /dev/sdb"
ansible all -b -m ansible.builtin.command -a "parted /dev/sdb unit MiB print"
ansible all -b -m ansible.builtin.command -a "vgs"
```

- **ВАЖНО: разметку запускают только на учебном стенде и только со вторым диском. force: true у filesystem означает mkfs на каждом прогоне: данные пропадут, а changed=0 не наступит. fstab проверяют до...**

# 10. КАЧЕСТВО: LINT, MOLECULE, ИНВЕНТАРЬ

```
ansible-lint
ansible-lint draft_lint.yml
```

- Линтер ищет файл .ansible-lint в текущем каталоге и выше, поэтому запускать его нужно из корня проекта. exclude_paths — файлы не открываются, skip_list — правило выключено, warn_list — нарушение видно, но...
- MOLECULE: сценарий — каталог molecule/default с файлами molecule.yml, converge.yml, prepare.yml, verify.yml.

```
molecule create
molecule converge
molecule idempotence
molecule verify
molecule destroy
molecule test
```

- Драйвер delegated ничего не создаёт: в стенде нет Docker, роль проверяется на настоящих web1 и web2 (managed: false). idempotence применяет converge.yml второй раз и проверяет changed=0.
- ВНЕШНИЙ ИНВЕНТАРЬ (практика 16): скрипт inventory/stand.py печатает JSON; ключ _meta.hostvars хранит переменные хостов для Ansible.

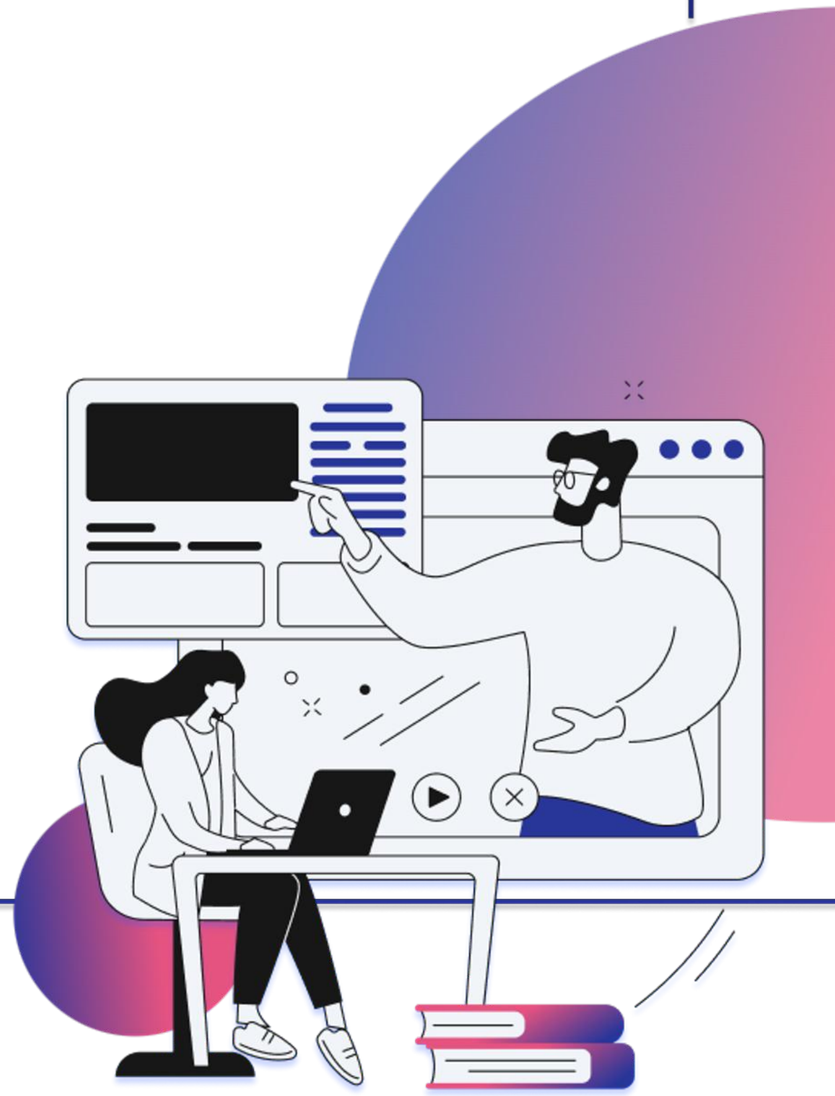
# 11. СВОДНАЯ ТАБЛИЦА МОДУЛЕЙ КУРСА

```
-----------------------------------------------------------------------
ansible.builtin.ping — проверить подключение (1, 3, 4)
ansible.builtin.command — программа без оболочки (3, 8, 12, 13)
ansible.builtin.shell — команда через /bin/sh -c (3, 4, 8, 12, 16)
ansible.builtin.copy — положить готовый файл (3, 4, 8, 10, 12)
ansible.builtin.file — каталог, права, отсутствие (3, 7, 8, 10, 12)
ansible.builtin.template — разложить шаблон Jinja2 (6, 8, 10, 12)
ansible.builtin.lineinfile, ansible.builtin.blockinfile, ansible.builtin.replace — правка текста (8, 11)
```

# 12. ЧТО СМОТРЕТЬ ПРИ ОШИБКЕ

- Смотрим в таком порядке:
  - PLAY RECAP целиком: какая колонка ненулевая;
  - строка TASK выше отчёта: задача и машина, на которой она упала;
  - ansible-playbook --syntax-check: ошибка разбора YAML до запуска;
  - ansible-playbook -vvv и --limit: параметры задачи, подставленные
  - значения и обмен по SSH;
  - ansible all -m ansible.builtin.ping: доступность узла и Python;
  - ansible-inventory --graph и --host: тот ли инвентарь и переменные;
  - ansible-config dump --only-changed: прочитан ли ansible.cfg;
  - ansible-doc <модуль>: параметры именно вашей версии.
- Разбор частых ошибок — причина и порядок проверки — в приложении app_2_troubleshooting.txt.

Учебный центр АйТи Клауд



# Приложение 2. Частые ошибки и что с ними делать

**ЭТО СПРАВОЧНИК, А НЕ УЧЕБНИК.**

современные технологии простым языком

# Обзор практики

- **ЭТО СПРАВОЧНИК, А НЕ УЧЕБНИК: Здесь собраны ошибки, которые студенты курса видят на стенде: из секций «Если что-то пошло не так» всех шестнадцати практик и из шагов, где ошибка заметна только в выводе. Найдите...**

# КАК ЧИТАТЬ ЭТО ПРИЛОЖЕНИЕ

- Блок устроен одинаково: точный текст ошибки, причина и порядок проверки с командами.
- Текст ошибки приведён как есть, на английском: искать нужно именно его.
- Пометка «Практика N» указывает, где ошибка разобрана полностью.
- Курс собран на ansible-core 2.19: в другой версии текст ошибки может отличаться, а причина и проверки — те же.

# 1. ПОДКЛЮЧЕНИЕ И SSH

▸ **ПРОБЛЕМА: Permission denied (publickey).**

• Ключ не принят: его нет на узле или у файла ключа неверные права.

• Порядок: права ключа (chmod 600), потом authorized_keys.

• Практика 1.

```
ls -l ~/.ssh/id_ed25519
ssh -i ~/.ssh/id_ed25519 engineer@10.0.2.11 hostname
```

▸ **ПРОБЛЕМА: Host key verification failed.**

• Узел предъявил чужой ключ или он не совпадает с known_hosts: бывает после пересоздания ВМ.

• Порядок: host_key_checking, потом узел.

• Практика 2.

```
ansible-config dump --only-changed | grep -i host_key
ssh-keygen -F web1
```

▸ **ПРОБЛЕМА: Missing sudo password (в оболочке — sudo: a password is required)**

• NOPASSWD не применён: файл в /etc/sudoers.d потерял права 0440 или содержит опечатку.

• Порядок: sudo -n на узле, потом visudo.

• Практики 1 и 9.

# 2. ИНВЕНТАРЬ И КОНФИГУРАЦИЯ

▸ **ПРОБЛЕМА: [WARNING]: Unable to parse ... as an inventory source**

• Ansible не нашёл инвентарь и остался с одной машиной localhost: прогон может ничего не сделать.

• Порядок: pwd, потом cfg.

• Практики 1 и 2.

```
pwd ; ls -l inventory.ini
ansible --version | grep 'config file'
```

▸ **ПРОБЛЕМА: [WARNING]: Could not match supplied host pattern, ignoring: web3**

• Шаблон не совпал ни с чем: Ansible предупреждает и продолжает работу с теми, кто совпал.

• Опасен случай, когда не совпало ничего.

• Практика 2.

```
ansible-inventory --graph
ansible 'web3' --list-hosts
```

▸ **ПРОБЛЕМА: одна и та же машина записана в инвентаре несколько раз**

• Ошибки нет: повторная строка не создаёт вторую машину, а перезаписывает её переменные; побеждает последняя строка.

• Порядок: дубликаты.

• Практика 2.

# 3. YAML И ОТСТУПЫ

▸ **ПРОБЛЕМА: yaml.scanner.ScannerError: mapping values are not allowed here**

• Разборщик остановился на двоеточии и не ожидал значения словаря: это про двоеточие, а не про отступ.

• Порядок: отступ, кавычки.

• Практики 3 и 4.

```
ansible-playbook --syntax-check 05_broken.yml
sed -n '8,12p' 05_broken.yml
```

▸ **ПРОБЛЕМА: while scanning a quoted scalar / found unexpected end of stream**

• Кавычка не закрыта, и YAML читает остаток файла как строку; номер строки указывает на начало значения.

• Порядок: сосчитать кавычки.

• Практики 6 и 11.

```
sed -n '17,20p' 01_facts.yml
grep -c '"' 01_facts.yml
```

▸ **ПРОБЛЕМА: ERROR! Syntax Error while loading YAML: did not find expected key**

• Структура не сходится: отступ не кратен двум или в отступе таб, который глазом не отличается от пробела.

• Порядок: отступы, табы.

• Практики 3 и 4.

# 4. AD-HOC И МОДУЛИ

▸ **ПРОБЛЕМА: ERROR! Missing target hosts**

• Перед этим видно Could not match supplied host pattern, ignoring: webb1.

• Опечатка в имени или машины нет в инвентаре.

• Порядок: граф, имя.

• Практика 3.

```
ansible-inventory --graph
ansible-inventory --host web1
```

▸ **ПРОБЛЕМА: missing required arguments: dest (команда ad-hoc набрана без -a)**

• Модуль получил пустой набор параметров: забыт ключ -a.

• Порядок: обязательные параметры модуля, потом команда с ключом -a.

• Практика 3.

```
ansible-doc -s ansible.builtin.copy
ansible web -m ansible.builtin.copy -a "src=/tmp/a dest=/tmp/b"
```

▸ **ПРОБЛЕМА: модуль command отказался работать с | или >**

• Оболочки в цепочке нет: знак конвейера ушёл в программу как имя файла, а с перенаправлением ошибки нет, но файл не создан.

• Практика 3.

# 5. ПЛЕЙБУКИ

▸ **ПРОБЛЕМА: ERROR! 'host' is not a valid attribute for a Play**

• Тот же класс — 'notife' ... for a Task и 'ignore_errors' ... for a Play: ключ назван неверно или стоит не на своём уровне.

• Порядок: строка из ошибки, потом отступ.

• Практики 4 и 7.

```
sed -n '1,12p' 04_web_full.yml
ansible-playbook --syntax-check 04_web_full.yml
```

▸ **ПРОБЛЕМА: ERROR! The requested handler 'Restart Apache' was not found**

• Дальше: ...in either the main handlers list nor in the listening handlers list. notify вызвал обработчик, которого нет в этом play.

• Порядок: имена посимвольно, тот же play, отступ.

• Практики 4, 7 и 10.

```
grep -n 'notify' -A2 04_web_full.yml
grep -n 'listen' handlers/main.yml
```

▸ **ПРОБЛЕМА: ERROR! 'roles' is not a valid attribute for a Task**

• Ansible показывает место: tasks:, затем - roles: с пометкой ^ here. roles — ключ play.

• Порядок: уровень ключа, перенос на уровень play.

• Практики 10 и 14.

# 6. ПЕРЕМЕННЫЕ И ФАКТЫ

▸ **ПРОБЛЕМА: VARIABLE IS NOT DEFINED! / 'dict object' has no attribute**

• Первое печатает debug, когда переменной нет (задача остаётся зелёной), в 2.19 — заглушкой << error 1 - 'lab_owner' is undefined >>.

• Второе — обращение к несуществующему ключу факта, обычно из-за опечатки (ansible_distibution).

• Порядок: имя, потом vars_files и setup.

• Практики 5 и 6.

```
ansible web1 -m ansible.builtin.debug -a "var=lab_owner"
ansible web1 -m ansible.builtin.setup -a "filter=ansible_distribution*"
```

▸ **ПРОБЛЕМА: переменная переопределилась не там, где ожидалось**

• Ошибки нет: одно имя задано в двух местах. vars роли сильнее group_vars и play; --extra-vars сильнее всех.

• Порядок: найти объявления.

• Практика 5.

```
grep -rn 'web_port' ~/ansible-lab --include='*.yml'
ansible web1 -m ansible.builtin.debug -a "var=web_port"
```

▸ **ПРОБЛЕМА: факт не появился: ansible_local пуст или путь слишком короткий**

• Файлов .fact не найдено: проверьте каталог /etc/ansible/facts.d, расширение .fact, секцию [general] в INI и сбор фактов в этом play.

• Для INI путь на уровень длиннее: ansible_local.site.general.owner.

# 7. VAULT

▸ **ПРОБЛЕМА: ERROR! Attempting to decrypt but no vault secrets found**

• Рядом: ERROR!

• Decryption failed (no vault secrets were found that could decrypt): пароля нет или он не подошёл, и Ansible намеренно не различает эти случаи.

• Порядок: первая строка файла и способ пароля.

• Практика 5.

```
head -1 group_vars/all/vault.yml
ansible-config dump --only-changed | grep -i vault
```

▸ **ПРОБЛЕМА: ERROR! input is already encrypted**

• Файл уже зашифрован; это защита от потери данных.

• Порядок: первая строка файла, для смены пароля — rekey, а не encrypt.

• Практика 5.

```
head -1 secrets_demo.yml
ansible-vault rekey secrets_demo.yml
```

▸ **ПРОБЛЕМА: пароль Vault лежит в репозитории**

• Ошибок нет, но если .vault_pass попал в git рядом с зашифрованным файлом, шифрование не защищает ничего.

• Порядок: .gitignore.

# 8. ШАБЛОНЫ JINJA2

▸ **ПРОБЛЕМА: AnsibleUndefinedVariable: 'nginx_root' is undefined**

• Переменной нет: файл переменных не подключён, опечатка или она задана для другой группы.

• Порядок: vars_files, потом debug.

• Практика 12.

```
ansible web1 -m ansible.builtin.debug -a "var=nginx_root"
grep -n -A4 'vars_files' 01_nginx.yml
```

▸ **ПРОБЛЕМА: jinja2.exceptions.TemplateSyntaxError: unexpected '}'**

• В задаче это template error while templating string.

• Порядок: строка из сообщения, потом пары скобок.

• Практики 6 и 12.

```
grep -n '{{' templates/nginx_vhost.j2
ansible localhost -m ansible.builtin.template -a "src=templates/nginx_vhost.j2 dest=/tmp/vhost.test"
```

▸ **ПРОБЛЕМА: template error while templating string: expected token**

• Дальше: got ':' и String: server {{ backend.address:8080 }};.

• Внутри {{ }} записали конструкцию из конфига.

• Порядок: строка, данные.

# 9. ФАЙЛЫ И ПРАВА

▸ **ПРОБЛЕМА: Destination directory /var/www/html/site does not exist**

• copy не создаёт каталоги, а права меняет только root.

• Рядом: Destination directory is not writable by the current user и chmod: Operation not permitted.

• Порядок: каталог, file, become.

• Практики 6 и 8.

```
ls -ld /var/www/html/site ; grep -n -B3 'copy:' 01_copy.yml
ansible web1 -b -m ansible.builtin.command -a "lsattr /var/www/html/site/index.html"
```

▸ **ПРОБЛЕМА: mode must be in octal or symbolic notation**

• Значение не является ни восьмеричным числом, ни символической записью.

• Порядок: mode: "0644", арифметика, пробелы, цикл.

• Практики 3 и 8.

```
grep -rn 'mode:' *.yml
grep -n 'file_mode' group_vars/all.yml
```

▸ **ПРОБЛЕМА: lineinfile вставляет строку повторно при каждом прогоне**

• В файле две строки PasswordAuthentication no: regexp не совпал с тем, что лежит в файле.

• Порядок: cat -A, якоря, экранирование точки, state: absent.

# 10. РОЛИ, GALAXY, КОЛЛЕКЦИИ

▸ **ПРОБЛЕМА: ERROR! the role 'web_admin' was not found in ...**

• Дальше перечислены места поиска (.../10_roles/roles, ~/.ansible/roles, /usr/share/ansible/roles).

• Роль ищут рядом с файлом, где написан play.

• Порядок: что лежит в roles/, потом имя.

• Практики 10 и 14.

```
ls -d roles/*/
ansible-config dump | grep -i roles_path
```

▸ **ПРОБЛЕМА: переменная роли не переопределяется**

• web_admin_index_file: web.html в group_vars не действует: переменная объявлена в vars/main.yml, а vars сильнее group_vars.

• Практика 10.

```
grep -rn 'web_admin_index_file' roles/web_admin/
ansible web1 -m ansible.builtin.debug -a "var=web_admin_index_file"
```

▸ **ПРОБЛЕМА: роль «не видит» файл: Could not find or access 'index.j2'**

• В сообщении есть блок Searched in: с путём .../templates/index.j2.

• У template это templates/, у copy — files/.

• Порядок: имя.

# 11. ХРАНИЛИЩЕ: ДИСКИ, LVM, ФАЙЛОВЫЕ СИСТЕМЫ

▸ **ПРОБЛЕМА: Device /dev/sdb1 not found.**

• Так отвечает community.general.filesystem: ядро не увидело раздел.

• Рядом — '/dev/sdb1' is already used as ext4, use force=true to overwrite и unknown filesystem type 'xfs'; force: true уничтожит данные.

• Порядок: lsblk, partprobe, blkid.

• Практика 13.

```
lsblk /dev/sdb ; ls -l /dev/sdb1 ; partprobe /dev/sdb
blkid /dev/sdb1 /dev/mapper/data-webdata
```

▸ **ПРОБЛЕМА: Error: Partition(s) on /dev/sdb are being used.**

• Диск занят: смонтирован, активны тома LVM или swap; при umount видно target is busy.

• Порядок: findmnt, lsblk -f, fuser -vm.

• Практика 13.

```
findmnt --source /dev/sdb1 ; lsblk -f /dev/sdb
fuser -vm /srv/data
```

▸ **ПРОБЛЕМА: Sorry, no shrinking of webdata without force=true.**

• Рядом: /dev/mapper/data-webdata needs to be mounted for XFS operations: xfs_growfs работает с точкой монтирования.

• Размер в задаче меньше текущего; force: true обрежет ФС.

# 12. КАЧЕСТВО И ИНВЕНТАРИЗАЦИЯ

▸ **ПРОБЛЕМА: ansible-lint валит прогон на файлах, которые мы сами набрали**

• Итог Failed: 10 failure(s) — это норма.

• Нарушения: risky-octal, command-instead-of-module, no-changed-when, name[missing].

• Чужой код убирают в exclude_paths.

• Практики 3 и 16.

```
ansible-lint
ansible-lint --version
```

▸ **ПРОБЛЕМА: CRITICAL Driver delegated not found**

• Драйвера нет в molecule.yml или molecule не установлена полностью.

• Рядом: molecule не находит роль без ANSIBLE_ROLES_PATH.

• Порядок: molecule.yml, потом molecule drivers.

• Практика 16.

```
grep -A2 '^driver:' molecule/default/molecule.yml
molecule drivers
```

▸ **ПРОБЛЕМА: внешний инвентарь вернул не тот JSON — забыт _meta**

• Рядом: кэш не сбрасывается и Unable to parse .../inventory/stand.py as an inventory source (нет права на исполнение).

# ПОРЯДОК ДИАГНОСТИКИ: С ЧЕГО НАЧАТЬ, ЕСЛИ ПЛЕЙБУК НЕ РАБОТАЕТ

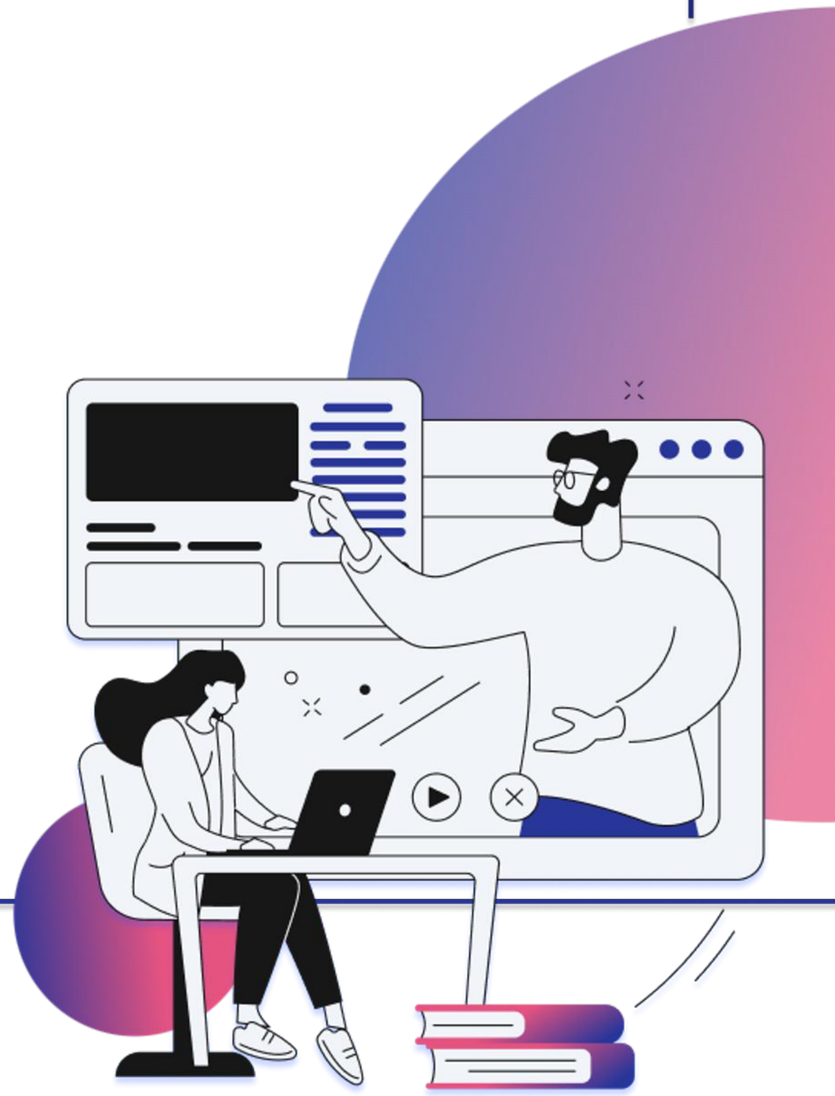
- Прогон упал, а причина непонятна.
- Порядок ниже экономит время: каждый шаг отсекает целый класс причин.
- Первый шаг не пропускайте.
  - 1. ПРОВЕРЬТЕ ПОДКЛЮЧЕНИЕ AD-HOC. Одна команда по одной машине
  - отвечает на вопрос, работают ли SSH и Python на узле:
  - ansible web1 -m ansible.builtin.ping
  - UNREACHABLE — это не про плейбук: разбирайтесь с подключением,
  - пока не увидите pong.
  - 2. ПРОВЕРЬТЕ ИНВЕНТАРЬ И СПИСКИ. Убедитесь, что задача попадает
  - туда, куда вы думаете, и что роли нашлись:
  - ansible-inventory --graph
  - ansible-playbook site.yml --list-hosts
  - ansible-playbook site.yml --list-tasks
  - 3. ПРОВЕРЬТЕ СИНТАКСИС. Ошибки YAML и опечатки в ключах видно
  - до прогона; при успехе команда молчит:
  - ansible-playbook --syntax-check site.yml

Учебный центр АйТи Клауд

# Приложение 3. Windows и сетевое оборудование

**ЭТО ПРИЛОЖЕНИЕ — ОБЗОРНОЕ.**

современные технологии простым языком



# Обзор практики

- ЭТО ПРИЛОЖЕНИЕ — ОБЗОРНОЕ.
- Практики в нём нет и быть не может: в стенде курса четыре машины Linux.
- Здесь разобрано, как то же самое делается для Windows и для сетевого оборудования, чем эти два случая отличаются от Linux и где смотреть дальше.
- Материал нужен, чтобы не было удивления: «Ansible, оказывается, умеет и это», — и чтобы вы понимали границы применимости того, что разобрано в практиках 1–16.

# ПОЧЕМУ ЭТОТ МАТЕРИАЛ ВЫНЕСЕН В ПРИЛОЖЕНИЕ

- Три причины, и все три практические.
  - WINDOWS — НЕ ЦЕЛЕВАЯ СИСТЕМА УПРАВЛЕНИЯ в России. Основной
  - парк — Linux и отечественные сборки. Управление Windows через
  - Ansible требует WinRM, сертификатов и, в большинстве случаев,
  - контроллера домена — это отдельный стенд и отдельная
  - компетенция. Держать её в основном курсе нет смысла.
  - СЕТЕВОЕ ОБОРУДОВАНИЕ требует своего стенда: коммутатор или
  - его эмуляция (EVE-NG, GNS3, ContainerLab). Задачи там устроены
  - принципиально иначе — не «поставить пакет», а «добавить строку
  - в конфигурацию».
  - И ТО И ДРУГОЕ — ПРО ОДНО И ТО ЖЕ. Механизм Ansible остаётся
  - подключения и набор модулей. Понимая практики 1–16, вы
  - разберётесь и здесь — нужно знать, с чего начать.

▸ **ЗАЧЕМ ЭТО В РАБОТЕ: в вакансиях и в проектах Ansible для сетевого оборудования встречается постоянно, а Windows — почти всегда как «остаток», который надо не сломать. Оба случая полезно хотя бы узнавать.**

# ЧАСТЬ 1. WINDOWS

-

# ШАГ 1. КОГДА ANSIBLE ДЛЯ WINDOWS ВСЁ ЖЕ НУЖЕН

- Windows-задачи встречаются в трёх ситуациях:
  - смешанный парк: часть служб работает на Windows (1С, MS SQL,
  - IIS, файловые ресурсы), и настраивать их руками не хочется;
  - разовые работы: раскатать обновления, изменить реестр,
  - раздать сертификат, создать учётные записи;
  - аудит: собрать состояние служб, установленных программ,
  - обновлений на парке Windows-машин.

▸ **ЧЕГО ЖДАТЬ НЕ СТОИТ: Ansible не заменяет групповые политики и не пытается. Управление доменом, политики безопасности и обновления в крупной среде удобнее и надёжнее делать средствами самого домена. Ansible...**

▸ **ВАЖНО: в курсе нет Windows-машины, и все команды приложения проверить не на чем. Это описание, а не практика. Если вы будете его повторять — поднимите отдельную тестовую машину, а не рабочую.**

# ШАГ 2. ЧТО НУЖНО НА УПРАВЛЯЕМОЙ МАШИНЕ WINDOWS

- Агент в Ansible не нужен и здесь, но нужен транспорт.
- Роль агента играет служба WinRM — встроенная в Windows служба удалённого управления, которая принимает команды PowerShell по сети.
- Условия, без которых ничего не заработает:
  - Windows Server 2016 и новее либо Windows 10/11 (для узлов);
  - включён WinRM, и он принимает подключения по HTTPS (порт 5986)
  - или по HTTP (порт 5985, только в изолированной сети);
  - включена проверка подлинности (NTLM или Kerberos);
  - на управляющей машине установлены библиотеки Python для WinRM.
- Для настройки WinRM в составе коллекции `ansible.windows` есть готовый скрипт `ConfigureRemotingForAnsible.ps1`.
- Его запускают на управляемой машине от имени администратора:

```
# в PowerShell с правами администратора
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
.\ConfigureRemotingForAnsible.ps1
```

- ЧТО ДЕЛАЕТ ЭТОТ СКРИПТ:
  - включает WinRM и настраивает слушателя на HTTPS;
  - создаёт самоподписанный сертификат, если своего нет;

# ШАГ 3. ЧТО НУЖНО НА УПРАВЛЯЮЩЕЙ МАШИНЕ

- В виртуальное окружение, созданное в практике 1, ставится библиотека `pywinrm`:
```
source ~/ansible-venv/bin/activate
pip install pywinrm
```
- Если подключение идёт через Kerberos (обычная ситуация в домене), нужны ещё библиотеки:
```
pip install pywinrm[kerberos]
```
- И коллекция модулей для Windows:
```
ansible-galaxy collection install ansible.windows
```

▸ **ЧТО ЭТО ЗА КОЛЛЕКЦИЯ: необязательная для ядра часть. В пакете Ansible community она уже есть, при установке только `ansible-core` — нет. Именно поэтому в практике 1 мы ставили полный пакет.**

▸ **Этого пакета в актуальных дистрибутивах нет, и он всё равно не попал бы в наше виртуальное окружение — Ansible искал бы библиотеку в системном Python. Правильный путь один: `pip install` в то окружение, где...**

# ШАГ 4. ИНВЕНТАРЬ ДЛЯ WINDOWS

- Windows-машины описываются в том же инвентаре, что и Linux, — и это важное свойство Ansible: парк остаётся одним.

```
[windows]
win1 ansible_host=10.0.2.21
win2 ansible_host=10.0.2.22
[windows:vars]
ansible_user=engineer
ansible_connection=winrm
ansible_port=5986
ansible_winrm_transport=ntlm
```

- Разберём переменные:
  - `ansible_connection=winrm` — вместо SSH используется WinRM.
  - Это ключевая настройка: без неё Ansible попробует SSH и получит
  - отказ;
  - `ansible_port=5986` — HTTPS-слушатель WinRM. Порт 5985 — HTTP,
  - и в учебном изолированном стенде его иногда используют, чтобы
  - не возиться с сертификатом;

# ШАГ 5. МОДУЛИ ДЛЯ WINDOWS

- Модули Windows отделены от Linux-модулей и живут в коллекциях `ansible.windows` и `community.windows`.
- Названия начинаются с `win_` — по ним их и видно в `ansible-doc`.
- Часть модулей, которые встречаются чаще всего:
  - `ansible.windows.win_ping` — проверка подключения;
  - `ansible.windows.win_command` — запуск программы без оболочки,
  - `win_shell` — через PowerShell (аналог `command` и `shell`);
  - `ansible.windows.win_copy` — копирование файлов;
  - `ansible.windows.win_template` — шаблон Jinja2 на управляемой
  - машине; синтаксис шаблонов тот же, что в практике 6;
  - `ansible.windows.win_file` — файлы и каталоги;
  - `ansible.windows.win_package` — установка пакетов (`msi`, `exe`);
  - `ansible.windows.win_service` — службы Windows;
  - `ansible.windows.win_user` и `win_group` — локальные
  - пользователи и группы;
  - `ansible.windows.win_regedit` — реестр;
  - `ansible.windows.win_feature` — роли и компоненты Windows;

# ШАГ 6. ЧЕМ ЗАДАЧИ WINDOWS ОТЛИЧАЮТСЯ ОТ LINUX

- Различий шесть, и каждое приводит к ошибкам, если о нём не знать.
- 1.
- ТРАНСПОРТ — WinRM, а не SSH.
- Значит, другая настройка сети,
  сертификаты и порты (5985/5986 вместо 22). Диагностика тоже
  другая: сначала проверяют WinRM на узле, а не SSH.
- 2.
- ОБОЛОЧКА — PowerShell, а не bash.
- Синтаксис команд, переменные
  (`$env:PATH`), конвейеры и код возврата устроены иначе.
  Поэтому модули Windows существуют отдельно, а не «те же,
  но с другим именем».
- 3.
- ПУТИ — с обратным слешем: `C:\Program Files\...`.
- В YAML
  обратный слешь надо экранировать или брать строку в одинарные

# ШАГ 7. ЧЕГО НЕ ДЕЛАТЬ В WINDOWS-ЗАДАЧАХ

– НЕ ПИШИТЕ ПАРОЛЬ В ИНВЕНТАРЕ. Даже учебный: привычка останется,
– а пароль уедет в систему контроля версий.
– НЕ ОТКЛЮЧАЙТЕ ПРОВЕРКУ СЕРТИФИКАТА В РАБОЧЕЙ СРЕДЕ.
– `ansible_winrm_server_cert_validation=ignore` — только стенд.
– НЕ ЗАМЕНЯЙТЕ МОДУЛИ КОМАНДАМИ POWERSHELL. `win_shell` с длинным
– скриптом невозможно проверить на идемпотентность: он выполняется
– каждый раз и каждый раз сообщает `changed`.
– НЕ УПРАВЛЯЙТЕ КОНТРОЛЛЕРОМ ДОМЕНА ЧЕРЕЗ ANSIBLE БЕЗ НЕОБХОДИМОСТИ.
– Для этого есть штатные средства, и они безопаснее.
– НЕ ЗАБЫВАЙТЕ ПРО ЛИЦЕНЗИИ И ПЕРЕЗАГРУЗКИ. `win_updates`
– и установка компонентов часто требуют перезагрузки; в Ansible
– это делается модулем `ansible.windows.win_reboot` — по смыслу
– как `ansible.builtin.reboot` из практики 9.

# ЧАСТЬ 2. СЕТЕВОЕ ОБОРУДОВАНИЕ

-

# ШАГ 8. ПОЧЕМУ СЕТЕВЫЕ УСТРОЙСТВА — ЭТО ДРУГОЙ СЛУЧАЙ

- На сетевом коммутаторе нет Python.
- И агента поставить нельзя, и модуль на устройство не скопировать.
- Модель «подключились, скопировали модуль, выполнили, удалили» здесь не работает.
- Поэтому в Ansible для сетевого оборудования всё иначе:
  - МОДУЛЬ ВЫПОЛНЯЕТСЯ НА УПРАВЛЯЮЩЕЙ МАШИНЕ, а не на устройстве.
  - Он подключается к устройству и передаёт ему команды;
  - СОЕДИНЕНИЕ — не SSH в обычном смысле, а `network_cli`:
  - интерактивная командная строка устройства, в которую Ansible
  - пишет команды и читает ответы. Для устройств с API используется
  - `httpapi`, для NETCONF — `netconf`;
  - КОНФИГУРАЦИЯ — ЭТО СТРОКИ, а не файлы. Их нельзя «положить
  - целиком»: сначала смотрим текущее состояние, потом добавляем
  - или убираем строки;
  - ФАКТОВ В ПРИВЫЧНОМ ВИДЕ НЕТ. `gather_facts` не работает:
  - вместо него отдельный модуль фактов для каждой платформы
  - (`cisco.ios.ios_facts`, `arista.eos.eos_facts`).

# ШАГ 9. КОЛЛЕКЦИИ И СОЕДИНЕНИЯ

- Базовые возможности даёт коллекция `ansible.netcommon`: именно она содержит соединения `network_cli`, `httpapi`, `netconf` и модули `cli_command`, `cli_config`, работающие с любой платформой.
- Модули для конкретных вендоров — в отдельных коллекциях:
  - `cisco.ios` — Cisco IOS и IOS-XE;
  - `cisco.nxos` — Cisco NX-OS;
  - `arista.eos` — Arista EOS;
  - `juniper.junos` — Juniper Junos;
  - `vyos.vyos` — VyOS;
  - `community.network` — старые платформы, поддержка сообществом.
- Установка:

```
ansible-galaxy collection install ansible.netcommon
ansible-galaxy collection install cisco.ios
```

▸ **ПОЧЕМУ ИМЕННО ТАК: `ansible.netcommon.cli_command` подходит, когда нужно выполнить произвольную команду, а модуль вендора — когда нужна проверка версии, откат, работа с кандидатом конфигурации. Вендорские...**

# ШАГ 10. ИНВЕНТАРЬ ДЛЯ СЕТЕВЫХ УСТРОЙСТВ

- Устройства описываются в том же инвентаре, но с другим набором переменных:

```
[switches]
sw1 ansible_host=10.0.2.31
sw2 ansible_host=10.0.2.32
[switches:vars]
ansible_connection=network_cli
ansible_network_os=cisco.ios.ios
ansible_user=engineer
ansible_password={{ vault_switch_password }}
```

- Что здесь важно:
  - `ansible_connection=network_cli` — иначе Ansible полезет по SSH
  - и попытается запустить Python;
  - `ansible_network_os=cisco.ios.ios` — платформа. Значение
  - совпадает с именем коллекции; без него модуль не поймёт,
  - с чем разговаривает;
  - `ansible_become_method=enable` — переход в привилегированный

# ШАГ 11. ЧТО ОБЫЧНО ДЕЛАЮТ НА СЕТЕВОМ ОБОРУДОВАНИИ

- Четыре задачи покрывают почти всю повседневную работу.
- 1.
- СНЯТЬ РЕЗЕРВНУЮ КОПИЮ КОНФИГУРАЦИИ.
- Это первое, что делают,

```
прежде чем что-то менять:
- name: Сохранить конфигурацию коммутаторов
cisco.ios.ios_config:
backup: true
backup_options:
dir: "{{ playbook_dir }}/backup"
filename: "{{ inventory_hostname }}.cfg"
```

- 2.
- ПРОЧИТАТЬ СОСТОЯНИЕ: версии, интерфейсы, таблица маршрутизации,

```
соседи. Модули фактов возвращают данные, из которых можно
```

- 3.
- ИЗМЕНИТЬ КОНФИГУРАЦИЮ.

# Если что-то пошло не так

▸ **ПРОБЛЕМА: win_ping отвечает UNREACHABLE, порт 5986 недоступен**

```
win1 | UNREACHABLE! => {"msg": "winrm connection error: ..."}
```

• Проверяйте по порядку:
  – Слушает ли WinRM на узле:
  – # на управляемой машине Windows
  – winrm enumerate winrm/config/listener
  – Открыт ли порт. С управляющей машины:
  – nc -z -w3 10.0.2.21 5986
  – Настроен ли WinRM скриптом ConfigureRemotingForAnsible.ps1
  – и запущен ли он от администратора.

• Самая частая причина: порт 5986 есть, а слушателя нет — WinRM включён, но не настроен.

▸ **ПРОБЛЕМА: SSL: CERTIFICATE_VERIFY_FAILED**

```
win1 | UNREACHABLE! => {"msg": "ssl: ... certificate verify failed"}
```

• Узел предъявил самоподписанный сертификат.

• На стенде это решается переменной:

```
ansible_winrm_server_cert_validation=ignore
```

# ЧТО ДАЛЬШЕ И ГДЕ СМОТРЕТЬ

- Если предстоит работать с Windows:
  - документация коллекции `ansible.windows` — список модулей
  - с примерами;
  - `ansible-doc ansible.windows.win_service` — справка по любому
  - модулю, как в практике 3;
  - практика 2 — переменные подключения; практика 5 — где хранить
  - пароль; практика 7 — как разделить задачи по семейству ОС.
- Если предстоит работать с сетевым оборудованием:
  - документация `ansible.netcommon` — соединения и общие модули;
  - `ansible-doc cisco.ios.ios_config` — примеры настройки;
  - практика 4 — идемпотентность; практика 14 — `serial`, чтобы
  - менять устройства по одному; практика 15 — коллекции.
- ЧЕГО В ЭТОМ КУРСЕ НЕТ И НЕ БУДЕТ:
  - практики с Windows и с коммутатором: для них нужен отдельный
  - стенд и отдельная программа;
  - разбора Active Directory, DHCP, DNS и других служб Windows;