

Ansible

- [Список используемых источников](#)
- [Molecule: Тестирование ролей](#)
- [Использование переменных внутри других переменных](#)
- [Расстановка отступов](#)
- [Использование нескольких строк в переменных](#)
- [Использование фильтров для управления данными](#)
- [Ansible galaxy](#)
 - [Управление зависимостями](#)
- [Инвентарь](#)
 - [Динамический инвентарь](#)
- [Пишем роли ansible не ломая прод — особенности check_mode или как правильно его готовить](#)
- [Переменные среды](#)
- [Callback plugins](#)
- [Отладка](#)
 - [Задачи по отладке](#)
- [Jinja](#)
 - [Макросы](#)
- [Коллекции Ansible](#)
- [Иерархия переменных в ansible](#)
- [Структуры сущностей ansible](#)

?????? ??????????????

????????

- [Good practices for ansible](#)
- [Ansible naming conventions](#)
- [Looking up secrets from Hashicorp Vault](#)
- [Galaxy Installation with Ansible](#)
- [How to use Ansible Galaxy](#)
- [Ansible role argument specification](#)
- [Пишем роли ansible не ломая прод — особенности check_mode или как правильно его готовить](#)

Molecule: ?????????????? ??????

?????? ?????????????????? ???????????:

- [Ansible Molecule project](#)
- [Разработка и тестирование Ansible-ролей с использованием Molecule и Podman](#)
- [Тестирование Ansible с использованием Molecule с Ansible в качестве верификатора](#)
- [Ansible Testing Using Molecule with Ansible as Verifier](#)
- [Тестирование ролей Ansible с помощью Molecule в Ubuntu 18.04](#)
- [Инструкция: как тестировать ansible-роли и узнавать о проблемах до продакшена](#)
- [Testing your Ansible roles with Molecule](#)
- [Testing Ansible roles and playbooks with Molecule](#)
- [Ansible Collections: Role Tests with Molecule](#)

???????????????? ???? ?????

?????? ?????????? ??????????

???????????? "vars"

```
set_fact:
```

```
variable: '{{ vars['my_' + variablename + '_variable'] }}'
```

????????????? ???? ??????

Yaml

```
app:
  config1:
    base:
      {{ service1.company.backend | to_nice_yaml(indent=2) | trim | indent(6) }}
  config2:
    node:
      {{ service1.company.addr | to_nice_yaml(indent=2) | trim | indent(6) }}
```

???????????????? ???? ??????
????? ? ????????????

???????????????? ?????? ?????????? **join**

Этот способ позволяет использовать стандартный метод строки - **join**.

```
ansible -m debug -a msg="{{ '-'|join((var1, var2, var3)) }}" localhost
```

??????? ?????????????????? ????????????????

- [Understanding multi line strings in YAML and Ansible \(Part I - YAML\)](#)

Ansible galaxy

Ansible galaxy

???????????? ???? ??????????

???????????? ???? ??????????

```
ansible-galaxy collection install <FQCN>
```

????????? ? ?????????? ??????????

```
ansible-galaxy collection install --collections-path collections <FQCN>
```

<FQCN> - Имя коллекции вместе с её полным путём (fully-qualified collection name)

??????????

Инвентарь

???????????????? ???? ?????

???????? ???? ????????????????? ???? ?????????

1. [Динамические инвентории Ansible](#)
2. [Working with dynamic inventory](#)

????? ???? ansible ?? ?????

???? — ??????????????

check_mode ??? ???

????????????? ??? ????????????

В данном опусе я попытаюсь рассказать как можно писать роли и плейбуки такими, чтобы они отыгрывали без падений при запуске с флагом `--check`. Зачем вот это всё: вы в команде адептов инфраструктуры как код и не только раскатываете свою инфраструктуру, но и обновляете ее и хотите быстро восстанавливать, а это значит что вы должны обеспечивать одну из ключевых концепций ansible - идемпотентность повторных запусков. Для этого вы вынуждены не только писать роли, применяя их в лабораторных условиях, но и применения их на реальной инфраструктуре, о том как дебажить при этом ямлы написано например [здесь](#). Применять код который делает множество изменений в продакшн - такое себе удовольствие не для слабонервных, поэтому и мы будем пускать его предварительно с флагом `--check` - в холостую, желая посмотреть те изменения которые будут внесены в целевую инфраструктуру. То есть, если вы собираетесь реконфигурировать сервис в одном конфиге и перезапустить-перегрузить его, то вы должны увидеть только 2 изменения (changed) по итогам запуска.

Итак, как этого добиться.

????????????? ??????????????????

Конечно же можно заигнорить таски с ошибками выставив `ignore_errors: true`, но это путь к непредсказуемому поведению на целевом хосте и пропуску не только единичных задач, но и блоков и плейбуков, в рамках холостого запуска. К чему это приведет - это приведет к потенциальной необработанной ошибке. Как это исправить - определить логическую переменную и использовать ее в качестве флага игнорирования:

```
- set_fact:
  your_vars_ignore: true
- debug:
  var: your_vars_ignore
  ignore_errors: "{{ your_vars_ignore }}"
```

Использовать в качестве таковых переменных встроенные переменные `ansible`, например `ansible_check_mode`, то есть при запуске с флагом `--check` игнорировать ошибки - поможет вам в тех случаях когда нецелесообразно вычислять промежуточную переменную и выполнение игнорируемого кода предсказуемо.

```
- debug:
  msg: "no errors in check mode"
  ignore_errors: "{{ ansible_check_mode }}"
```

???????? ?????????? ??????????

С другой стороны даже такое игнорирование часто не обеспечивает полноту выполнения холостого запуска кода. В тех случаях когда блок кода не вносит изменения на целевую систему, например определение переменных, скачивание на контроллер архивов для их распаковки, стоит игнорировать `check_mode`:

```
---
- set_fact:
  always_var: "this var is defined always"
  check_mode: false
```

????????? shell - ?? ??? ????????

Мы все частенько пренебрегаем использованием модулей `command` и `shell` не искав найдя нужного модуля `ansible` или используя специфичные утилиты командной строки. Так вот использование этих модулей с флагом `check` будет практически всегда выдавать изменение. Этим можно и нужно гибко [управлять](#).

????????? ?????????????? ??????????

`Ansible` позволяет определить, что означает “сбой” в каждой задаче, используя `failed_when` условие. Как и во всех условных обозначениях в `Ansible`, списки из нескольких `failed_when` условий объединены неявным `and`, что означает сбой задачи только при выполнении всех условий. Если вы хотите вызвать сбой при выполнении любого из условий, вы должны определить условия явным `or`.

Вы можете проверить наличие сбоя, выполнив поиск слова или фразы в выходных данных команды или на основе кода возврата:

```

- name: Fail task when the command error output prints FAILED
  ansible.builtin.command: /usr/bin/example-command -x -y -z
  register: command_result
  failed_when: "'FAILED' in command_result.stderr"

- name: Fail task when both files are identical
  ansible.builtin.raw: diff foo/file1 bar/file2
  register: diff_cmd
  failed_when: diff_cmd.rc == 0 or diff_cmd.rc >= 2

```

?????? ?????????? ????????????

Ansible позволяет определить, когда конкретная задача “изменила” удаленный узел, используя `changed_when` условие. Это позволяет вам определить, на основе кодов возврата или выходных данных, следует ли сообщать об изменении в статистике Ansible и следует ли запускать обработчик или нет. Как и во всех условных обозначениях в Ansible, списки из нескольких `changed_when` условий объединены неявным `and`.

```

---
- name: Report 'changed' when the return code is not equal to 2
  ansible.builtin.shell: /usr/bin/billybass --mode="take me to the river"
  register: bass_result
  changed_when: "bass_result.rc != 2"

- name: This will never report 'changed' status
  ansible.builtin.shell: wall 'beep'
  changed_when: False

- name: Combine multiple conditions to override 'changed' result
  ansible.builtin.command: /bin/fake_command
  register: result
  ignore_errors: True
  changed_when:
    - '"ERROR" in result.stderr'
    - result.rc == 2

```

??????

Использование описанных выше способов позволяет выполнить следующие требования к скриптам конфигурирования:

- при повторном запуске скрипта состояние целевой системы не будет изменено
- при запуске скрипта с флагом `--check` выполняются все задачи
- запуск скрипта с флагом `check` не приводит к изменению на целевой инфраструктуре

?????????? ?????

- ANSIBLE_MAX_DIFF_SIZE=104857600 - ?????????? ?????? ?????????? ??????? ??????
???????

Callback plugins

?????? ?????????????????? ????????????????

- [Creating an Alerting Callback Plugin in Ansible - Part I\(ansible stdout_compact logger\)](#)

???????

Отладка

?????? ?? ????????

?????? ?????????????????? ????????????????

- [Debugging tasks](#)

Jinja

????????? Ansible

Коллекцией (collection) называется распространяемый как единое целое набор компонентов, расширяющих возможности Ansible.

В состав коллекции могут входить следующие компоненты:

- роли (roles);
- расширения (plugins);
- модули (modules);
- наборы сценариев (playbooks).

????????? ????????????

Пример команды для создания структуры каталогов коллекции:

```
ansible-galaxy collection init my_space.my_collection
```

Эта команда создаст в текущем каталоге подкаталог `my_space.my_collection/` и разместит в нем файлы коллекции `my_space.my_collection`.

В результате выполнения команды будет создана структура файлов и каталогов:

```
my_space/my_collection/  
├─ docs/  
├─ galaxy.yml  
├─ meta/  
│ └─ runtime.yml  
├─ plugins/  
│ └─ README.md  
├─ README.md  
└─ roles/
```

В составе коллекции могут быть следующие файлы и каталоги:

- `meta/`
Каталог с файлами, содержащими служебную информацию о коллекции, используемую утилитой `ansible-galaxy`.
- `plugins/`
Опциональный каталог, содержащий расширения, необходимые для работы коллекции.

- `roles/`

Каталог с подкаталогами ролей. Подробности приведены в [описании ролей](#).

- `tests/`

Каталог с файлами автоматических тестов для коллекции.

- `CHANGELOG.md`

Файл с историей изменений между версиями.

- `LICENSE`

Файл лицензии, устанавливающей правила использования и распространения коллекции.

- `README.md`

Описание коллекции, включающее в себя:

- предназначение коллекции;
- требования к окружению;
- список поддерживаемых операционных систем;
- порядок описания коллекции в файле зависимостей Ansible и способ установки;
- список ролей;
- порядок запуска автоматических тестов;
- тип лицензии, под которой распространяется код коллекции;
- информация об авторских правах.

- `galaxy.yml`

Сведения о коллекции, используемые утилитой `ansible-galaxy`, включая номер ее актуальной версии.

Пример заполнения `galaxy.yml`

```
---
namespace: astra
name: ald_pro
version: 1.0.0
description: Collection for ALD Pro deployment
readme: README.md
authors:
  - LLC "RusBITech-Astra"
dependencies:
  "freeipa.ansible_freeipa": "1.10.0"
  "ansible.utils": "3.0.0"
tags:
  - astra
  - ald_pro
repository: https://hub.astra-automation.ru/ui/repo/published/astra/ald_pro/
documentation: https://hub.astra-automation.ru/ui/repo/published/astra/ald_pro/docs/
```

Здесь:

- `astra` – название пространства имен;
- `ald_pro` – название коллекции;
- `version` – номер версии коллекции;

- `description` – краткое описание коллекции;
- `readme` – путь к файлу `README`, содержащему подробное описание коллекции;
- `authors` – список авторов коллекции;
- `dependencies` – зависимости, необходимые для использования коллекции;
- `tags` – список тегов, по которым можно найти коллекцию среди множества других;
- `repository` – ссылка на репозиторий с исходным кодом коллекции;
- `documentation` – ссылка на документацию коллекции.
- `requirements_ansible.yml`
Зависимости Ansible, необходимые для использования коллекции.

Особенности коллекций, доступных на портале [Automation Hub](#), приведены в разделе [Automation Hub](#).

????

Роли используют для группирования задач по развертыванию и настройке сложных систем.

Пример создания структуры каталогов для роли:

```
ansible-galaxy init httpd
```

Здесь `httpd` – название каталога, в котором будет создана структура роли.

В результате выполнения команды будет создана следующая структура файлов и каталогов:

```
httpd/  
├─ defaults/  
│  └─ main.yml  
├─ files/  
├─ handlers/  
│  └─ main.yml  
├─ meta/  
│  └─ main.yml  
├─ README.md  
├─ tasks/  
│  └─ main.yml  
├─ templates/  
├─ tests/  
│  └─ inventory  
│     └─ test.yml  
└─ vars/  
   └─ main.yml
```

В составе роли могут быть следующие файлы и каталоги:

- `defaults/`
Каталог с файлами, содержащими значения по умолчанию для переменных роли.
- `files/`
Каталог с вспомогательными файлами, используемыми ролью, например, шаблоны HTML-страниц, изображения и так далее. При выполнении роли файлы из этого каталога копируются на управляемые узлы.
- `handlers/`
Каталог с файлами обработчиков, выполняемых при использовании роли.
- `meta/`
Каталог с файлами, содержащими служебную информацию о роли, используемую утилитой `ansible-galaxy`.
- `tasks/`
Каталог с файлами, выполняющимися при использовании роли.
- `templates/`
Каталог с шаблонами формата Jinja 2.
- `vars/`
Каталог с файлами, содержащими список переменных роли. Каждая роль содержит список переменных, позволяющих управлять настройками соответствующего ПО.
- `LICENSE`
Файл лицензии, под которой распространяется код роли.
- `README.md`
Описание роли, включающее в себя:
 - Предназначение роли.
 - Требования к окружению.
 - Список переменных роли.
Переменные роли делятся на обязательные и опциональные. Для опциональных переменных, значения которых не заданы, будут использованы значения по умолчанию.
 - Предупреждение
Значения обязательных переменных должны быть явно заданы в наборе сценариев или используемом им файле с переменными.
 - Примеры использования.
 - Порядок запуска автоматических тестов.
 - Тип лицензии, под которой распространяется код роли.
 - Информация об авторских правах.
 - Список поддерживаемых операционных систем.

Сценарий может включать список подключаемых ролей как в следующем примере:

```
- hosts: server.example.com
  become: true
  gather_facts: true
  roles:
    - astra.postgresql.postgresql
    - astra.rubackup.rubackup_server
```

????????? ????????????

По умолчанию утилита `ansible-galaxy` использует <https://galaxy.ansible.com/> в качестве реестра коллекций.

Чтобы загрузить коллекцию, используйте следующую команду:

```
ansible-galaxy collection download my_namespace.my_collection
```

Здесь:

- `my_namespace` – название пространства имен;
- `my_collection` – название коллекции.

В результате выполнения этой команды утилита `ansible-galaxy` загрузит указанную коллекцию и ее зависимости и создаст файл `requirements.yml`, который можно использовать для установки этой коллекции на узлах без доступа к серверу Galaxy.

По умолчанию все коллекции загружаются в каталог `collections/`, который создается в текущем каталоге. Чтобы загрузить коллекции в другой каталог, укажите путь к нему в значении аргумента `-p`.

Если необходимо загрузить определенную версию коллекции, выполните следующую команду:

```
ansible-galaxy collection download my_namespace.my_collection:1.0.0
```

Здесь `1.0.0` – версия коллекции.

Если необходимо установить несколько коллекций, воспользуйтесь одним из следующих способов:

- перечислите необходимые коллекции в командной строке, например:

```
ansible-galaxy collection download my_namespace1.my_collection1:1.0.0 my_namespace2.my_coll
```

- укажите необходимые коллекции файле `requirements.yml`, например:

```
---
collections:
  - name: my_namespace1.my_collection1
  - name: my_namespace2.my_collection2
  version: ">=1.2.0"
```

??????????

Чтобы установить коллекцию, используйте следующую команду:

```
ansible-galaxy collection install my_namespace.my_collection
```

По умолчанию все коллекции устанавливаются в домашнем каталоге пользователя, конкретно в `~/.ansible/collections/ansible_collections/`.

?????????? ????????????

Если вы загрузили архив коллекции, для ее установки используйте следующую команду:

```
ansible-galaxy collection install my_namespace-my_collection-1.0.0.tar.gz -p ./collections
```

Здесь `-p ./collections` – путь к каталогу для установки коллекции.

Если коллекция разрабатывается локально, чтобы установить ее, используйте следующую команду:

```
ansible-galaxy collection install /path/to/collection -p ./collections
```

Здесь `/path/to/collection` – путь к каталогу, который содержит файлы разрабатываемой коллекции.

В результате выполнения команды Ansible соберет коллекцию на основе файлов `MANIFEST.json` или `galaxy.yml`.

Если в одном каталоге находятся несколько коллекций, они могут быть установлены одной командой, например:

```
ansible-galaxy collection install /path/to/directory -p ./collections
```

Здесь `/path/to/directory` – путь к каталогу, коллекции из которого необходимо установить.

?????????? ?????????????? ??????? ??????????????

По умолчанию утилита `ansible-galaxy` устанавливает последнюю доступную версию коллекции.

Чтобы установить определенную версию коллекции, укажите ее после названия коллекции с использованием идентификатора диапазона `==`. Например, чтобы установить версию `1.0.1` коллекции `my_namespace.my_collection`, используйте следующую команду:

```
ansible-galaxy collection install my_namespace.my_collection:==1.0.1
```

Если вы хотите установить версию в определенном диапазоне, используйте несколько идентификаторов версий, разделенных запятой. Например, чтобы установить версию, которая больше или равна `1.0.0` и меньше `2.0.0`, используйте команду:

```
ansible-galaxy collection install 'my_namespace.my_collection:>=1.0.0,<2.0.0'
```

Примечание

По умолчанию утилита `ansible-galaxy` игнорирует предварительные версии, например, бета-версии. Если необходимо установить такую версию, укажите ее с помощью идентификатора диапазона `==`, например:

```
ansible-galaxy collection install my_namespace.my_collection:==1.0.0-beta.1
```

Доступные идентификаторы диапазона:

- `*` - самая последняя версия (по умолчанию);
- `!=` - не равна указанной версии;
- `==` - равна указанной версии;
- `>=` - больше или равна указанной версии;
- `>` - больше указанной версии;
- `<=` - меньше или равна указанной версии;
- `<` - меньше указанной версии.

????????? ? ?????????????????????? ??????

requirements.yml

Файл `requirements.yml` позволяет устанавливать несколько коллекций.

Для каждой коллекции можно указать следующую информацию:

- `name` - название;
- `version` - версия (можно указать диапазон версий);
- `source` - URL источника, из которого будет загружена коллекция;
- `type` - тип источника:
 - `dir` - локальный каталог;
 - `file` - локальный файл в формате `tar.gz`;

- galaxy – реестр коллекций;
- git – репозиторий Git;
- subdirs – подкаталог внутри каталога, указанного в поле source;
- url – URL-адрес.

Например, для установки коллекции из репозитория Git, добавьте в requirements.yml следующие данные:

```
collections:  
  - name: https://github.com/organization/repo_name.git  
    type: git
```

В файле requirements.yml можно указать роли, используя ключ roles, например:

```
roles:  
  - name: geerlingguy.java  
    version: "1.9.6"  
  
collections:  
  - name: geerlingguy.php_roles  
    version: ">=0.9.3"  
    source: https://galaxy.ansible.com
```

Чтобы установить роли и коллекции, указанные в файле requirements.yml, используйте следующую команду:

```
ansible-galaxy install -r requirements.yml
```

Если необходимо установить только коллекции, используйте следующую команду:

```
ansible-galaxy collection install -r requirements.yml
```

Если необходимо установить только роли, используйте следующую команду:

```
ansible-galaxy role install -r requirements.yml
```

Важно

Если в аргументах указан путь к целевому каталогу для установки, ansible-galaxy установит только роли и пропустит коллекции.

?????????? ?? ?????????????? Git

Установка коллекций из репозитория Git подходит для тестирования и работы с неофициальными версиями.

Чтобы установить коллекцию из репозитория Git с помощью командной строки, вместо названия коллекции используйте URI репозитория. Перед URI необходимо добавить префикс `git+`, если вы не используете SSH-аутентификацию. Вы можете указать ветку, коммит или тег с помощью синтаксиса, где нужное значение отделяется запятой от URI репозитория.

Например, если необходимо установить коллекцию из репозитория, используя последнюю версию в ветке `devel`, используйте следующую команду:

```
ansible-galaxy collection install git+https://github.com/organization/repo_name.git,devel
```

Если необходимо установить коллекцию из частного репозитория GitHub, используйте следующую команду:

```
ansible-galaxy collection install git@github.com:organization/repo_name.git
```

Если необходимо установить коллекцию из локального репозитория Git, используйте следующую команду:

```
ansible-galaxy collection install git+file:///home/user/path/to/repo_name.git
```

Важно

Встраивание учетных данных в URI небезопасно. Используйте надежные варианты аутентификации, чтобы ваши учетные данные не были раскрыты в журналах или где-либо еще.

При установке коллекции из репозитория Ansible использует метаданные коллекции, содержащиеся в файлах `galaxy.yml` или `MANIFEST.json`, чтобы корректно собрать коллекцию.

По умолчанию Ansible ищет файлы метаданных в двух местах репозитория:

- в корневом каталоге;
- в каждом каталоге на один уровень ниже от корня.

Если в корневом каталоге репозитория существует файл `galaxy.yml` или `MANIFEST.json`, Ansible использует метаданные коллекции из этого файла для установки отдельной коллекции.

Пример структуры репозитория с одной коллекцией:

```
├─ galaxy.yml
├─ plugins/
│  ├─ lookup/
│  ├─ modules/
│  └─ module_utils/
└─ README.md
```

Если в одном или нескольких каталогах по пути к репозиторию (на один уровень ниже) существует файл `galaxy.yml` или `MANIFEST.json`, Ansible устанавливает каждый каталог с файлом метаданных как коллекцию.

Пример репозитория с двумя коллекциями:

```
├─ collection1/
│  ├─ docs/
│  ├─ galaxy.yml
│  └─ plugins/
│     ├─ inventory/
│     └─ modules/
└─ collection2/
   ├─ docs/
   ├─ galaxy.yml
   ├─ plugins/
   │  └─ filter/
   │  └─ modules/
   └─ roles/
```

Если структура репозитория отличается, или вам нужна только определенная коллекция, можно указать путь к каталогу с метаданными через символ `#` в URI репозитория. Например, чтобы установить только `collection2` из примера выше, используйте команду:

```
ansible-galaxy collection install git+https://github.com/organization/repo_name.git#/collection2
```

????????? ??????? ????????????

Чтобы получить список установленных коллекций, выполните следующую команду:

```
ansible-galaxy collection list
```

Запустите команду с аргументом `-vvv`, чтобы отобразить более подробную информацию о коллекциях.

Чтобы вывод команды содержал данные только об определенной коллекции, укажите ее полное название, например:

```
ansible-galaxy collection list my_namespace.my_collection
```

Чтобы искать коллекции, размещенные вне каталога по умолчанию, используйте аргумент `-p`. Если необходимо указать несколько путей, разделите их с помощью `:`, например:

```
ansible-galaxy collection list -p '/opt/ansible/collections:/etc/ansible/collections'
```

????????? ????????????

После установки вы можете проверить, соответствует ли содержимое установленной коллекции содержимому коллекции на сервере. Эта функция предполагает, что коллекция установлена по одному из настроенных путей и существует на одном из настроенных серверов Galaxy.

Чтобы выполнить проверку, используйте следующую команду:

```
ansible-galaxy collection verify my_namespace.my_collection
```

Если команда проверки коллекции выполнена успешно, то в выводе не будет никаких сообщений. Если коллекция была изменена, измененные файлы будут перечислены под названием коллекции.

Пример вывода:

```
Collection my_namespace.my_collection contains modified content in the following files:  
my_namespace.my_collection  
  plugins/inventory/my_inventory.py  
  plugins/modules/my_module.py
```

Используйте аргумент `-vvv` для отображения дополнительной информации, такой как версия и путь к установленной коллекции, URL-адрес удаленной коллекции, используемой для проверки, и результат успешной проверки.

Если у вас установлена не самая последняя версия коллекции, вам следует указать конкретную версию для проверки. Если версия не указана, установленная коллекция проверяется на соответствие последней версии, доступной на сервере.

Также можно указать коллекции для проверки в файле `requirements.yml`. Зависимости в этом файле не включаются в процесс проверки и должны быть проверены отдельно с помощью команды:

```
ansible-galaxy collection verify -r requirements.yml
```

Проверка файлов формата `tar.gz` не поддерживается. Если `requirements.yml` содержит пути к таким файлам или URL для установки, можно использовать аргумент `--ignore-errors`, чтобы гарантировать обработку всех коллекций.

????????? ????????????

Если вам больше не нужна коллекция, удалите каталог с ней из своей файловой системы:

```
rm -rf ~/.ansible/collections/ansible_collections/my_namespace/my_collection
```

???????????????? ???? ????????

После установки вы можете ссылаться на содержимое коллекции по полному квалифицированному имени ([FQCN](#)). FQCN состоит из следующих компонентов:

- namespace – пространство имен;
- collection name – название коллекции;
- resource name – название ресурса, например, модуля, роли, набора сценариев или любого другого контента внутри коллекции.

Пример FQCN:

```
astra.ald_pro.controller
```

???????????????? ???? ??????? ?????

Ключевое слово `collections` позволяет определить список коллекций, в которых роль или набор сценариев должны искать неуказанные названия модулей и действий. Таким образом, вы можете использовать ключевое слово, а затем ссылаться на модули и расширения действий по их сокращенным названиям в рамках этой роли или сценария.

Важно

Если в наборе сценариев используется как ключевое слово `collections`, так и одна или несколько ролей, то роли не используют коллекции, определенные в наборе сценариев. Это значит, что внутри роли нельзя ссылаться на коллекцию, заданную в наборе сценариев, и ее необходимо указывать явно. Это одна из причин, по которой рекомендуется всегда использовать FQCN.

???????????????? ? ?????

В рамках роли можно управлять тем, какие коллекции будет искать Ansible для выполнения задач внутри роли. Для этого в файле `meta/main.yml` используйте слово `collections`. Ansible будет использовать список коллекций, определенный внутри роли, даже если в наборе сценариев, который вызывает роль, определены другие коллекции в отдельной записи с ключевым словом `collections`. Роли, определенные внутри коллекции, всегда неявно ищут сначала собственную коллекцию, поэтому вам не нужно использовать ключевое слово

`collections` для доступа к модулям, действиям или другим ролям, содержащимся в той же коллекции.

Пример заполнения файла роли:

```
collections:
  - my_namespace.first_collection
  - my_namespace.second_collection
  - other_namespace.other_collection
```

????????????? ? ?????????? ????????????

В наборе сценариев можно управлять коллекциями, которые Ansible использует для поиска модулей и расширений действий. Однако любые роли, которые вы вызываете в наборе сценариев, определяют собственный порядок поиска коллекций, они не наследуют настройки вызывающего набора сценариев. Это верно даже в том случае, если роль не определяет собственное ключевое слово для коллекций.

Пример набора сценариев:

```
- name: Run a play using the collections keyword
hosts: all
collections:
  - my_namespace.my_collection

tasks:

  - name: Import a role
    ansible.builtin.import_role:
      name: role1

  - name: Run a module not specifying FQCN
    my_module:
      option1: value

  - name: Run a debug task
    ansible.builtin.debug:
      msg: '{{ lookup("my_namespace.my_collection.lookup1", "param1")| my_namespace.my_collect
```

Ключевое слово `collections` создает упорядоченный путь поиска для расширений без пространства имен и ссылок на роли. Оно не устанавливает содержимое и не изменяет поведение Ansible в отношении загрузки расширений или ролей. Обратите внимание, что для недействующих или модульных расширений, например, для поисковиков, фильтров и тестов, все равно требуется указывать FQCN.

При использовании ключевого слова `collections` не обязательно добавлять `ansible.builtin` в список поиска. Если его не включить, по умолчанию доступно следующее содержимое:

- стандартные модули и расширения Ansible, доступные через `ansible-base` или `ansible-core`;
- поддержка старых путей расширений третьих сторон.

Примечание

Предпочтительнее использовать FQCN модуля или расширения вместо использования ключевого слова `collections`.

???????????????? ???? ???? ???? ??

??????????

Вы можете распространять наборы сценариев в своей коллекции и вызывать их следующими способами:

- из командной строки:

```
ansible-playbook my_namespace.my_collection.playbook1.yml -i ./myinventory
```

- из другого набора сценариев:

```
- name: Import a playbook
  ansible.builtin.import_playbook: my_namespace.my_collection.playbookX
```

При создании наборов сценариев внутри коллекций рекомендуется использовать универсальные настройки для `hosts`:

- Глобальное выполнение на всех узлах. При необходимости можно ограничить выполнение с помощью аргумента `--limit` или пользовательского инвентаря:

```
- hosts: all
```

- Ограничение на управляющий узел (`localhost`), используется для задач, которые выполняются на управляющем узле:

```
- hosts: localhost
```

- Гибкое указание целевых узлов через переменную:

```
- hosts: '{{ target | default("webservers") }}'
```

Здесь `target` – переменная, которую можно задать через параметр `-e 'target=host1,host2'`. Если переменная не передана, используется группа узлов

webservers из инвентаря.

Примечание

- Названия наборов сценариев, как и другие ресурсы коллекции, имеют ограниченный набор допустимых символов. Названия могут содержать только строчные алфавитно-цифровые символы, а также `_` и должны начинаться с буквы. Символ `-` не допустим в названиях наборов сценариев. Наборы сценариев с названиями, содержащими недопустимые символы, не могут быть адресованы – это ограничение импортера Python, который используется для загрузки ресурсов коллекции.
- Все расширения должны находиться в каталогах, специфичных для коллекции.

?????? ?????????????? ??????????????

- [Коллекции Ansible](#)

????????? ?????????????? ? ansible

1. command line values (for example, -u my_user, these are not variables)
2. role defaults (defined in role/defaults/main.yml)
3. inventory file or script group vars
4. inventory group_vars/all
5. playbook group_vars/all
6. inventory group_vars/*
7. playbook group_vars/*
8. inventory file or script host vars
9. inventory host_vars/*
10. playbook host_vars/*
11. host facts / cached set_facts
12. play vars
13. play vars_prompt
14. play vars_files
15. role vars (defined in role/vars/main.yml)
16. block vars (only for tasks in block)
17. task vars (only for the task)
18. include_vars
19. set_facts / registered vars
20. role (and include_role) params
21. include params
22. extra vars (always win precedence)

????????? ?????????? ansible

?????? ?????????????????? ????????????????

- [Sample Ansible setup](#)
- [Ansible Inventory Project](#)