нетология

Работа с Roles

Алексей
Метляков

Алексей Метляков

DevOps Engineer

OpenWay

Алексей Метляков

План занятия

1. Что такое Role?
2. Для чего нужна Role?
3. Из чего состоит Role?
4. Где хранить Role?
5. Как создать Role?
6. Как тестировать Role?
7. Molecule
8. Хранение и использование Role
9. Итоги
10. Домашнее задание

Что такое Role?

Role - группа **tasks**, которая нацелена на выполнение действий, приводящих к единому результату.

- **Role** - выполняет список действий
- Список может состоять из одного действия
- Действия должны приводить к одному общему результату (например, установка и настройка nginx)
- Если в рамках действий **role** необходимо получить отдельный результат, стоит написать для этого отдельную **role**
- Если ваш **playbook** использует **roles**, то использование **tasks** нежелательно, при этом можно использовать **pre_tasks** и **post_tasks**
- Если ваш **playbook** использует **roles**, то для расширения функционала стоит писать **role**, даже если она состоит из атомарного действия

Для чего нужна Role?

Использование **role** помогает достигнуть таких целей, как:

- Повышение читаемости кода
- Создание верхнеуровневых **playbook**
- Решение вопроса версионирования
- Повышение универсализации

Из чего состоит Role?

Стандартная общая структура директории с **role** состоит из:

- **Defaults** - директория, которая содержит в себе **yaml** файл с переменными по умолчанию, необходимыми для выполнения **role**
- **Handlers** - директория с **yaml**, в котором содержатся все **handlers** от **role**
- **Meta** - содержит информацию о самой роли для обеспечения работоспособности с **ansible-galaxy**
- **Tasks** - директория содержит в себе **yaml** файлы с **plays** и **tasks**
- **Templates** - директория с шаблонами конфигурационных файлов (в случае, если они нужны)
- **Tests** - тестовый **inventory** и **playbook**, которые, обычно, позволяют запустить роль на localhost
- **Vars** - директория, которая содержит yaml файлы с остальными переменными

Из чего состоит Role?

- В каждой из этих директорий, **ansible** ищет **main.yml**
- В некоторых случаях, можно добавлять свои **.yml** файлы с наполнением и подключать их в **main.yml**
- Роль можно дополнить своим собственным **module** или другим **plugin**, для этого его нужно добавить в директорию **library**

Где хранить Role?

- Директория **roles** в плейбуке
- По пути **/etc/ansible/roles**
- По пути **/usr/share/ansible/roles**
- Кастомный путь

Для кастомного пути нужно:

- Переопределять путь в **ansible.cfg** в параметре **DEFAULT_ROLE_PATH**
- Создать переменную окружения **ANSIBLE_ROLES_PATH**

Как создать Role?

- Для начала нужно определить: какую цель преследует исполнение будущей роли?
- Поискать уже готовое решение на galaxy.ansible.com
- Создать стандартную структуру директорий и файлов при помощи **ansible-galaxy**
- Создать необходимые **tasks**, **handlers**
- Определить все необходимые переменные в **defaults** и **vars**
- Создать готовый тестовый **playbook** в **tests**
- Заполнить **meta** всю информацию о роли, наиболее полно описать её в **README.md**

Как тестировать Role?

В данном виде тестирование Role превращается в достаточно сложную задачу, так как нужно:

- Провести проверку синтаксиса
- Подготовить тестовое окружение
- Провести проверку на работоспособность
- Провести проверку на идемпотентность
- Исправить ошибки на каждом из этих этапов и повторять весь сценарий, пока не будет получен положительный результат

Molecule

Данный фреймворк позволяет избавить нас от рутины и заниматься только созданием и исправлением ошибок. Он умеет:

- Создавать новые **roles**
- Создавать **scenarios** тестирования
- Тестировать **roles** против разного окружения
- Поддерживает docker, podman, delegated как драйверы подключений

Официальная документация тут

Как правильно создать Role?

- Создать стандартную структуру директорий и файлов при помощи **molecule**
- Создать необходимые **tasks**, **handlers**
- Определить все необходимые переменные в **defaults** и **vars**
- Создать готовый тестовый **playbook** в **tests**, заполнить файлы **molecule** для проведения тестирования
- Заполнить **meta** всю информацию о роли, наиболее полно описать её в **README.md**

Структура директорий

После инициализации новой роли мы получаем следующие директории и файлы:

- Стандартный набор директорий и файлов для **role**
- molecule - набор **scenarios** для тестирования

Внутри любого **scenario** находятся следующие файлы:

- **molecule.yml** - основной файл для **molecule**
- **converge.yml** - **playbook**, который **molecule** будет использовать для запуска тестов.
- **verify.yml** - дополнительные тесты после исполнения **role**

Структура molecule.yml

Внутри файла находятся следующие директивы:

- **dependency** - перечисление зависимостей роли
- **driver** - указание параметров выбранного **drirver**
- **platform** - перечисление хостов для выбранного **driver**
- **provisioner** - указание поставщика для **molecule**
- **verifier** - выбор **framework** для проведения проверок

Туда же можно добавить:

- **lint** - конфигурирование **linter** для тестирования
- **scenario** - перечисление сценариев тестирования

Как правильно тестировать Role?

В данном виде тестирование **role** упрощается, так как нужно:

- Подготовить тестовое окружение
- Запустить сценарий проверки через **molecule**
- Исправить ошибки на каждом из этих этапов и повторять весь сценарий, пока не будет получен положительный результат

Как правильно тестировать Role?

Тестирование условно можно разделить на три вида:

- Использование полного сценария тестирования
- Использование собственных сценариев тестирования
- Использование отдельных частей сценария самостоятельно

Как правильно тестировать Role?

molecule test - запускает полный сценарий тестирования role

Полный сценарий включает в себя:

- **lint** - прогон линтеров
- **destroy** - удаление старых инстансов с прошлого запуска
- **dependency** - производит установку ansible-зависимостей, если есть
- **syntax** - проверка синтаксиса с помощью `ansible-playbook --syntax-check`
- **create** - создание инстансов для тестирования
- **prepare** - подготовка инстансов, если это необходимо

Как правильно тестировать Role?

molecule test - запускает полный сценарий тестирования role

Полный сценарий включает в себя:

- **converge** - запуск тестируемого плейбука
- **idempotence** - проверка на идемпотентность при помощи повторного запуска
- **side_effects** - действия, которые не относятся к role, но необходимые для тестирования
- **verify** - запуск тестов с помощью указанного фреймворка тестирования
- **cleanup** - очистка внешней инфраструктуры от результатов тестирования
- **destroy** - уничтожение инстансов для тестирования

Как правильно тестировать Role?

- Каждую из указанных частей сценария можно вызвать отдельно, но нужно держать в уме, что у каждой из них могут быть зависимости.
- Для того, чтобы понимать, что каждая из **tasks** будет запускать, в случае отдельного вызова - необходимо пользоваться конструкцией **molecule matrix <task_name>.**
- Перед тем, как уничтожать инстансы, к ним можно подключиться и в ручном режиме проверить все изменения в системе.
- Оставить возможность подключения можно и с полным сценарием тестирования, воспользовавшись параметром **--destroy=never.**

Как правильно тестировать Role?

Очерёдность сценариев можно переопределить через директиву **scenario.** Формат записи в **molecule.yml** будет выглядеть так:

```
---
scenario:
  <task>_sequence:
    - list
    - of
    - tasks
...
---
#Example of redefined of test scenario
scenario:
  test_sequence:
    - create
    - converge
    - idempotence
    - destroy
...
```

Список основных команд molecule

- molecule init role --driver-name <driver> <rolename>
- molecule init scenario --driver-name <driver> <scenarioname>
- molecule test
- molecule test --destroy=never
- molecule matrix <taskname>
- molecule matrix -s <scenarioname> <taskname>
- molecule <taskname>

Хранение и использование role

Традиционно, для версионирования и хранения **role** используется **git**. Для определения текущей версии используют семантический подход:

- **major** - добавление feature ломает совместимость с предыдущей версией
- **minor** - добавление feature не ломает совместимость
- **patch** - исправление багов, рефакторинг и прочие схожие

Номер версии необходимо выставлять при помощи **git tag** или интерфейсов вашего инструмента для **git**.

Хранение и использование role

Чтобы использовать **role** в **playbook** - необходимо объявить её в файле **requirements.yml** и использовать команду **ansible-galaxy** для последующего скачивания нужной версии роли. Список команд:

- ansible-galaxy install -r -requirements.yml
- ansible-galaxy install -r -requirements.yml
- ansible-galaxy install -r -requirements.yml --force
- ansible-galaxy install <rolename>

Итоги

- **Role** необходимо использовать в том случае, если снижается текущая читаемость **playbook**
- Разделение **playbook** на отдельные **role** позволяет жить каждой логической единице независимыми жизненными циклами и обрастать дополнительным функционалом
- Дополнительный функционал можно принудительно не использовать, оставаясь на старой версии **role**
- Существует удобный **framework** для тестирования - **molecule**
- Множество готовых **role** находятся в открытом доступе с лицензией **BSD**
- Собственные **role** можно хранить в частных хранилищах **git** и использовать напрямую оттуда

Домашнее задание

Давайте посмотрим ваше домашнее задание.

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

нетология

Задавайте вопросы и пишите отзыв о лекции!

Алексей Метляков

Алексей Метляков