Пишем роли ansible не ломая прод — особенности check_mode или как правильно его готовить

В данном опусе я попытаюсь рассказать как можно писать роли и плэйбуки 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 сделает 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.

Игнорируем игнорирование

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 плэйбуков, 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 }}"
```

Никаких холостых прогонов

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
```

Хороший shell - он как модуль

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.

Учимся правильно падать

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
```

Учимся правильно изменяться

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
```

Выводы

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

Revision #1
Created 4 April 2024 13:37:14 by Антон Сергеевич Абраменко
Updated 4 April 2024 13:40:09 by Антон Сергеевич Абраменко