Практическое пособие Ansible

Это практическое пособие познакомит вас с Ansible. Вам понадобится виртуальная или реальная машина, которая будет выступать в роли узла для Ansible. Окружение для Vagrant идет в комплекте с этим пособием.

Ansible — это программное решение для удаленного управления конфигурациями. Оно позволяет настраивать удаленные машины. Главное его отличие от других подобных систем в том, что Ansible использует существующую инфраструктуру SSH, в то время как другие (chef, puppet, и пр.) требуют установки специального PKI-окружения.

Пособие покрывает такие темы:

1.Установка Ansible и Vagrant
2.Файл инвенторизации
3.Модули shell, copy, сбор фактов, переменные
4.Запуск на группу хостов
5.Плейбуки
6.Пример: поднимаем кластер, устанавливаем и настраиваем Apache и балансировщик нагрузок HAproxy
7.Обработка ошибок, откат
8.Шаблоны конфигурации
9.Роли

Ansible использует так называемый push mode: конфигурация «проталкивается» (push) с главной машины. Другие CM-системы обычно поступают наоборот – узлы «тянут» (pull) конфигурацию с главной машины.

Этот режим интересен потому что вам не нужно иметь публично доступную главную машину для удаленной настройки узлов; это узлы должны быть доступны (позже мы увидим, что скрытые узлы также могут получать конфигурацию).

Что нужно для Ansible

Необходимы следующие Python-модули

- python-yaml
- python-jinja2

На Debian/Ubuntu запустите:

```
sudo apt-get install python-yaml python-jinja2 python-paramiko python-crypto
```

У вас также должна быть пара ключей в ~/.ssh.

Установка Ansible

Из исходников

Ветка devel всегда стабильна, так что используем ее. Возможно, вам нужно будет установить git (sudo apt-get install git на Debian/Ubuntu).

```
git clone git://github.com/ansible/ansible.git
cd ./ansible
```

Теперь можно загрузить окружение Ansible.

```
source ./hacking/env-setup
```

Из deb пакета

```
sudo apt-get install make fakeroot cdbs python-support
git clone git://github.com/ansible/ansible.git
cd ./ansible
make deb
sudo dpkg -i ../ansible_1.1_all.deb (version may vary)
```

В этом пособии предполагается, что вы использовали именно этот способ.

Установка Vagrant

Vagrant позволяет с легкостью создавать виртуальные машины и запускать их на VirtualBox. Vagrantfile идет в комплекте с пособием.

Чтобы запустить Vagrant вам нужно установить:

- VirtualBox
- Ruby (скорее всего уже установлено на вашей системе)
- Vagrant 1.1+ (см. http://docs.vagrantup.com/v2/installation/index.html).

Теперь инициализируйте виртуальную машину с помощью следующей команды. Имейте ввиду, что вам не нужно скачивать какой-либо "box" вручную. Это пособие уже содержит готовый Vagrantfile, он содержит все, что нужно для работы.

```
vagrant up
```

и налейте себе кофе (если вы используете vagrant-hostmaster, то вам нужно будет ввести root-пароль). Если что-то пошло не так, загляните в туториал по Vagrant'y.

Добавление SSH-ключей на виртуальной машине

Чтобы продолжить, вам нужно добавить свои ключи в authorized_keys root'a на виртуальной машине. Это не обязательно (Ansible может использовать sudo и авторизацию по паролю), но так будет намного проще.

Ansible идеально подходит для этой задачи, поэтому используем его. Однако, я не буду пока ничего объяснять. Просто доверьтесь мне.

```
ansible-playbook -c paramiko -i step-00/hosts step-00/setup.yml --ask-pass --
sudo
```

В качестве пароля введите vagrant. Если возникнут ошибки "Connections refused", то проверьте настройки фаервола.

Теперь добавьте свои ключи в ssh-agent (ssh-add).

Inventory

Теперь нам нужно подготовить файл inventory. Место по умолчанию это /etc/ansible/hosts.
Но вы можете настроить Ansible так, чтобы использовался другой путь. Для этого используется переменная окружения (ANSIBLE_HOSTS) или флаг -i.

Мы создали такой файл inventory:

```
host0.example.org ansible_ssh_host=192.168.33.10 ansible_ssh_user=root
host1.example.org ansible_ssh_host=192.168.33.11 ansible_ssh_user=root
host2.example.org ansible_ssh_host=192.168.33.12 ansible_ssh_user=root
```

ansible_ssh_host это специальная переменная, которая содержит IP-адрес узла, к которому будет создаваться соединение. В данном случае она не обязательна, если вы используете gem vagrant-hostmaster. Также, вам нужно будет менять IP-адреса если вы устанавливали и настраивали свою виртуальную машину с другими адресами.

ansible_ssh_user это еще одна специальная переменная которая говорит Ansible'y подключаться под указанным аккаунтом (юзером). По умолчанию Ansible использует ваш текущий аккаунт, или другое значение по умолчанию, указанное в ~/.ansible.cfg (remote_user).

Проверка

Теперь когда Ansible установлен, давайте проверим, что все работает:

```
ansible -m ping all -i step-01/hosts
```

Здесь Ansible попытается запустить модуль ping (подробнее о модулях позже) на каждом хосте. Вывод должен быть примерно таким:

```
host0.example.org | success >> {
    "changed": false,
    "ping": "pong"
}

host1.example.org | success >> {
    "changed": false,
    "ping": "pong"
}

host2.example.org | success >> {
    "changed": false,
    "ping": "pong"
}
```

Отлично! Все три хоста живы и здоровы, и Ansible может общаться с ними.

Общение с узлами

Теперь мы готовы. Давайте поиграем с уже знакомой нам командой из прошлого раздела: ansible. Эта команда – одна из трех команд, которую Ansible использует для взаимодействия с узлами.

Сделаем что-нибудь полезное

В прошлой команде -m ping означал «используй модуль ping». Это один из множества модулей, доступных в Ansible. Модуль ping очень прост, он не требует никаких аргументов. Модули, требующие аргументов, могут получить их через -a. Давайте взглянем на несколько модулей.

Модуль shell

Этот модуль позволяет запускать shell-команды на удаленном узле:

```
ansible -i step-02/hosts -m shell -a 'uname -a' host0.example.org
```

Вывод должен быть вроде:

```
host0.example.org | success | rc=0 >>
Linux host0.example.org 3.2.0-23-generic-pae #36-Ubuntu SMP Tue Apr 10 22:19:09
UTC 2012 i686 i686 i386 GNU/Linux
```

Легко!

Модуль copy

Модуль copy позволяет копировать файл из управляющей машины на удаленный узел. Представим, что нам нужно скопировать наш /etc/motd в /tmp узла:

```
ansible -i step-02/hosts -m copy -a 'src=/etc/motd dest=/tmp/' host0.example.org
```

Вывод:

```
host0.example.org | success >> {
    "changed": true,
    "dest": "/tmp/motd",
    "group": "root",
    "md5sum": "d41d8cd98f00b204e9800998ecf8427e",
    "mode": "0644",
    "owner": "root",
    "size": 0,
    "src": "/root/.ansible/tmp/ansible-1362910475.9-246937081757218/motd",
    "state": "file"
}
```

Ansible (точнее, модуль copy, запущенный на узле) ответил кучей полезной информации в формате JSON. Позже мы увидим, как это можно использовать.

У Ansible есть огромный
список модулей, который покрывает практически все, что можно делать в системе. Если вы не нашли подходящего модуля, то написание своего модуля – довольно простая задача (и не обязательно писать его на Python, главное, чтобы он понимал JSON).

Много хостов, одна команда

Все что было выше – замечательно, но нам нужно управлять множеством хостов. Давайте попробуем. Допустим, мы хотим собрать факты про узел и, например, хотим узнать какая версия Ubuntu установлена на узлах. Это довольно легко:

```
ansible -i step-02/hosts -m shell -a 'grep DISTRIB_RELEASE /etc/lsb-release' all
```

all означает «все хосты в файле inventory». Вывод будет примерно таким:

```
host1.example.org | success | rc=0 >>
DISTRIB_RELEASE=12.04

host2.example.org | success | rc=0 >>
DISTRIB_RELEASE=12.04

host0.example.org | success | rc=0 >>
DISTRIB_RELEASE=12.04
```

Больше фактов

Легко и просто. Однако, если нам нужно больше информации (IP-адреса, размеры ОЗУ, и пр.), такой подход может быстро оказаться неудобным. Решение – использовать модуль setup. Он специализируется на сборе фактов с узлов.

Попробуйте:

```
ansible -i step-02/hosts -m setup host0.example.org
```

ответ:

```
"ansible_facts": {
    "ansible_all_ipv4_addresses": [
        "192.168.0.60"
    ],
    "ansible_all_ipv6_addresses": [],
    "ansible_architecture": "x86_64",
    "ansible_bios_date": "01/01/2007",
    "ansible_bios_version": "Bochs"
    },
    ---snip---
    "ansible_virtualization_role": "guest",
    "ansible_virtualization_type": "kvm"
},
"changed": false,
"verbose_override": true
```

Вывод был сокращен для простоты, но вы можете узнать много интересного из этой информации. Вы также можете фильтровать ключи, если вас интересует что-то конкретное.

Например, вам нужно узнать, сколько памяти доступно на всех хостах. Это легко: запустите ansible -i step-02/hosts -m setup -a 'filter=ansible_memtotal_mb' all:

```
host2.example.org | success >> {
    "ansible_facts": {
        "ansible_memtotal_mb": 187
    },
    "changed": false,
    "verbose_override": true
}

host1.example.org | success >> {
    "ansible_facts": {
        "ansible_memtotal_mb": 187
    },
    "changed": false,
    "verbose_override": true
}

host0.example.org | success >> {
    "ansible_facts": {
        "ansible_memtotal_mb": 187
    },
    "changed": false,
    "verbose_override": true
}
```

Заметьте, что узлы ответили не в том порядке, в котором они отвечали выше. Ansible общается с хостами параллельно!

Кстати, при использовании модуля setup можно указывать * в выражении filter=. Как в shell.

Выбор хостов

Мы видели, что all означает «все хосты», но в Ansible есть
куча иных способов выбирать хосты:

- host0.example.org:host1.example.org будет запущен на host0.example.org и на host1.example.org
- host*.example.org будет запущен на всех хостах, названия которых начинается с 'host' и заканчивается на '.example.org' (тоже как в shell)

Группировка хостов

Хосты в inventory можно группировать. Например, можно создать группу debian, группу web-servers, группу production и так далее.

```
[debian]
host0.example.org
host1.example.org
host2.example.org
```

Можно даже сократить:

```
[debian]
host[0-2].example.org
```

Если хотите задавать дочерние группы, используйте [groupname:children] и добавьте дочерние группы в него. Например, у нас есть разные дистрибутивы Линукса, их можно организовать следующим образом:

```
[ubuntu]
host0.example.org

[debian]
host[1-2].example.org

[linux:children]
ubuntu
debian
```

Установка переменных

Вы можете добавлять переменные для хостов в нескольких местах: в файле inventory, файлах переменных хостов, файлах переменных групп и др.

Обычно я задаю все переменные в файлах переменных групп/хостов (подробнее об этом позже). Однако, зачастую я использую переменные напрямую в файле inventory, например, ansible_ssh_host, которая задает IP-адрес хоста. По умолчанию Ansible резолвит имена хостов при соединении по SSH. Но когда вы инициализируете хост, он, возможно, еще не имеет IP-адреса. ansible_ssh_host будет полезен в таком случае.

При использовании команды ansible-playbook (а не обычной команды ansible), переменные можно задавать с помощью флага --extra-vars (или -e). О команде ansible-playbook мы поговорим в следующем шаге.

ansible_ssh_port, как вы могли догадаться, используется, чтобы задать порт соединения по SSH.

```
[ubuntu]
host0.example.org ansible_ssh_host=192.168.0.12 ansible_ssh_port=2222
```

Ansible ищет дополнительные переменные в файлах переменных групп и хостов. Он будет искать эти файлы в директориях group_vars и host_vars, внутри директории, где расположен главный файл inventory.

Ansible будет искать файлы по имени. Например, при использовании упомянутого ранее файле inventory, Ansible будет искать переменные host0.example.org в файлах:

- group_vars/linux
- group_vars/ubuntu
- host_vars/host0.example.org

Если этих файлов не существует – ничего не произойдет, но если они существуют – они будут использованы.

Теперь, когда мы познакомились с модулями, инвентаризацией и переменными, давайте, наконец, узнаем о настоящей мощи Ansible с плейбуками.

Плейбуки Ansible

Концепция плейбуков очень проста: это просто набор команд Ansible (задач, tasks), похожих на те, что мы выполняли с утилитой ansible. Эти задачи направлены на конкретные наборы узлов/групп.

Пример с Apache (a.k.a. "Hello World!" в Ansible)

Продолжаем с допущением, что ваш файл inventory выглядит так (назовем его hosts):

```
[web]
host1.example.org
```

и все хосты — это системы на основе Debian.

Заметка: помните, что вы можете (и в нашем упражнении мы делаем это) использовать ansible_ssh_host чтобы задать реальный IP-адрес хоста. Вы также можете изменять inventory и использовать реальный hostname. В любом случае, используйте машину, с которой безопасно экспериментировать. На реальных хостах мы также добавляем ansible_ssh_user=root чтобы избежать потенциальных проблем с разными конфигурациями по умолчанию.

Давайте соберем плейбук, который установит Apache на машины группы web.

```
- hosts: web
  tasks:
    - name: Installs apache web server
      apt: pkg=apache2 state=installed update_cache=true
```

Нам всего лишь нужно сказать, что мы хотим сделать, используя правильные модули Ansible. Здесь мы используем модуль apt, который может устанавливать пакеты Debian. Мы также просим этот модуль обновить кэш.

Нам нужно имя для этой задачи. Это не обязательно, но желательно для вашего же удобства.

Ну, в целом было довольно легко! Теперь можно запустить плейбук (назовем его apache.yml):

```
ansible-playbook -i step-04/hosts -l host1.example.org step-04/apache.yml
```

Здесь step-04/hosts это файл inventory, -l ограничивает запуск хостом host1.example.org,
а apache.yml это наш плейбук.

При запуске команды будет вывод подобный этому:

```
PLAY [web] ********************

GATHERING FACTS ********************
ok: [host1.example.org]

TASK: [Installs apache web server] *********************
changed: [host1.example.org]

PLAY RECAP ********************
host1.example.org              : ok=2    changed=1    unreachable=0    failed=0
```

Примечание: возможно, вы заметите проходящую мимо корову, если у вас установлен cowsay :-) Если она вам не нравится, можно отключить ее так: export ANSIBLE_NOCOWS="1".

Давайте проанализируем вывод строчка за строчкой.

```
PLAY [web] *********************
```

Ansible говорит нам, что play выполняется в группе web. Play — это набор инструкций Ansible, связанных с хостом. Если бы у нас был другой -host: blah в плейбуке, он бы тоже вывелся (но после того, как первый play завершен).

```
GATHERING FACTS *********************
ok: [host1.example.org]
```

Помните, когда мы использовали модуль setup? Перед каждым воспроизведением Ansible запускает его на каждом хосте и собирает факты. Если это не требуется (скажем, потому что вам не нужна никакая информация о хосте) можно добавить gather_facts: no под строкой хоста (на том же уровне, где находится tasks:).

```
TASK: [Installs apache web server] *********************
changed: [host1.example.org]
```

Теперь самое главное: наша первая и единственная задача запущена, и, так как там сказано changed, мы знаем, что она изменила что-то на хосте host1.example.org.

```
PLAY RECAP *********************
host1.example.org              : ok=2    changed=1    unreachable=0    failed=0
```

Наконец, Ansible выводит выжимку того, что произошло: две задачи были выполнены, и одна из них изменила что-то на хосте (это была наша задача apache; модуль setup ничего не меняет).

Давайте запустим это еще раз и посмотрим, что произойдет:

```
$ ansible-playbook -i step-04/hosts -l host1.example.org step-04/apache.yml

PLAY [web] *********************

GATHERING FACTS *********************
ok: [host1.example.org]

TASK: [Installs apache web server] *********************
ok: [host1.example.org]

PLAY RECAP *********************
host1.example.org              : ok=2    changed=0    unreachable=0    failed=0
```

Теперь changed равен '0'. Это совершенно нормально и является одной из главных особенностей Ansible: плейбук будет делать что-то, только если есть что делать. Это называется идемпотентностью. Это значит что можно запускать плейбук сколько угодно раз, но в итоге мы будем иметь машину в одном и том же состоянии (ну, только если вы не будете безумствовать с модулем shell, но тут Ansible уже не сможет ничего поделать).

Улучшаем набор apache

Мы установили apache, давайте теперь настроим virtualhost.

Улучшение плейбука

Нам нужен лишь один виртуальный хост на сервере, но мы хотим сменить дефолтный на что-то более конкретное. Поэтому нам придется удалить текущий virtualhost, отправить наш virtualhost, активировать его и перезапустить apache.

Давайте создадим директиорию под названием files и добавим нашу конфигурацию для host1.example.org, назовем ее awesome-app:

```
<VirtualHost *:80>
```

```
  DocumentRoot /var/www/awesome-app

  Options -Indexes

  ErrorLog /var/log/apache2/error.log
  TransferLog /var/log/apache2/access.log
</VirtualHost>
```

Теперь небольшое обнуление плейбука и все готово:

```
- hosts: web
  tasks:
    - name: Installs apache web server
      apt: pkg=apache2 state=installed update_cache=true

    - name: Push default virtual host configuration
      copy: src=files/awesome-app dest=/etc/apache2/sites-available/ mode=0640

    - name: Deactivates the default virtualhost
      command: a2dissite default

    - name: Deactivates the default ssl virtualhost
      command: a2dissite default-ssl

    - name: Activates our virtualhost
      command: a2ensite awesome-app
      notify:
        - restart apache

  handlers:
    - name: restart apache
      service: name=apache2 state=restarted
```

Поехали:

```
$ ansible-playbook -i step-05/hosts -l host1.example.org step-05/apache.yml

PLAY [web] ********************

GATHERING FACTS ********************
ok: [host1.example.org]

TASK: [Installs apache web server] *********************
ok: [host1.example.org]

TASK: [Push default virtual host configuration] *********************
changed: [host1.example.org]

TASK: [Deactivates the default virtualhost] *********************
changed: [host1.example.org]

TASK: [Deactivates the default ssl virtualhost] *********************
changed: [host1.example.org]

TASK: [Activates our virtualhost] *********************
changed: [host1.example.org]

NOTIFIED: [restart apache] *********************
changed: [host1.example.org]

PLAY RECAP *********************
host1.example.org              : ok=7    changed=5    unreachable=0    failed=0
```

Круто! Ну, если задуматься, мы немного опережаем события. Не нужно ли проверить корректность конфигурации перед тем, как перезапускать apache? Чтобы не нарушать работоспособность сервиса в случае если конфигурация содержит ошибку.

Перезапуск в случае ошибки конфигурации

Мы установили apache, изменили virtualhost и перезапустили сервер. Но что, если мы хотим перезапускать сервер только когда конфигурация корректна?

Откатываемся, если есть проблемы

Ansible содержит классную особенность: он остановит всю обработку, если что-то пошло не так. Мы используем эту особенность чтобы остановить плейбук, когда конфигурация не валидна.

Давайте изменим файл конфигурации виртуального хоста awesome-app и сломаем его:

```
  <VirtualHost *:80>
    RocumentDoot /var/www/awesome-app

    Options -Indexes

    ErrorLog /var/log/apache2/error.log
    TransferLog /var/log/apache2/access.log
  </VirtualHost>
```

Как я сказал, если задача не может исполниться, обработка останавливается. Так что нужно удостовериться в валидности конфигурации перед перезапуском сервера. Мы также начнем с добавления виртуального хоста до удаления дефолтного виртуального хоста, так что последующий перезапуск (возможно, сделанный напрямую на сервере) не сломает apache.

Нужно было сделать это в самом начале. Так как мы уже запускали этот плейбук, дефолтный виртуальный хост уже деактивирован. Не проблема: этот плейбук можно использовать на других невинных хостах, так что давайте защитим их.

```
- hosts: web
  tasks:
    - name: Installs apache web server
      apt: pkg=apache2 state=installed update_cache=true

    - name: Push future default virtual host configuration
      copy: src=files/awesome-app dest=/etc/apache2/sites-available/ mode=0640

    - name: Activates our virtualhost
      command: a2ensite awesome-app

    - name: Check that our config is valid
      command: apache2ctl configtest

    - name: Deactivates the default virtualhost
      command: a2dissite default

    - name: Deactivates the default ssl virtualhost
      command: a2dissite default-ssl

    notify:
        - restart apache

  handlers:
    - name: restart apache
      service: name=apache2 state=restarted
```

Поехали:

```
$ ansible-playbook -i step-06/hosts -l host1.example.org step-06/apache.yml

PLAY [web] ********************

GATHERING FACTS ********************
ok: [host1.example.org]

TASK: [Installs apache web server] *********************
ok: [host1.example.org]

TASK: [Push future default virtual host configuration] *********************
changed: [host1.example.org]

TASK: [Activates our virtualhost] *********************
changed: [host1.example.org]

TASK: [Check that our config is valid] *********************
failed: [host1.example.org] => {"changed": true, "cmd": ["apache2ctl",
"configtest"], "delta": "0:00:00.045046", "end": "2013-03-08 16:09:32.002063",
"rc": 1, "start": "2013-03-08 16:09:31.957017"}
stderr: Syntax error on line 2 of /etc/apache2/sites-enabled/awesome-app:
Invalid command 'RocumentDoot', perhaps misspelled or defined by a module not
included in the server configuration
stdout: Action 'configtest' failed.
The Apache error log may have more information.

FATAL: all hosts have already failed -- aborting

PLAY RECAP ********************
host1.example.org              : ok=4    changed=2    unreachable=0    failed=1
```

Как вы заметили, apache2ctl возвращает код ошибки 1. Ansible видит это и останавливает работу. Отлично!

Ммм, хотя нет, не отлично… Наш виртуальный хост все равно был добавлен. При любой последующей попытке перезапуска Apache будет ругаться на конфигурацию и выключаться. Так что нам нужен способ отлавливать ошибки и возвращаться к рабочему состоянию.

(прим. переводичка: хабраюзер @clickfreak в комментариях советует взлянуть на специальную фичу Ansible 2.x).

Использование условий

Мы установили Apache, добавили виртуальный хост и перезапустили сервер. Но мы хотим вернуться к рабочему состоянию, если что-то пошло не так.

Возврат при проблемах

Здесь нет никакой магии. Прошлая ошибка – не вина Ansible. Это не система резервного копирования, и она не умеет отказывать все к прошлым состояниям. Безопасность плейбуков – ваша ответственность. Ansible просто не знает, как отменить эффект a2ensite awesome-app.

Как было сказано ранее, если задача не может исполниться – обработка останавливается… но мы можем принять ошибку (и нам нужно это делать). Так мы и поступим: продолжим обработку в случае ошибки, но только чтобы вернуть все к рабочему состоянию.

```
- hosts: web
  tasks:
    - name: Installs apache web server
      apt: pkg=apache2 state=installed update_cache=true

    - name: Push future default virtual host configuration
      copy: src=files/awesome-app dest=/etc/apache2/sites-available/ mode=0640
```

```
    - name: Activates our virtualhost
      command: a2ensite awesome-app

    - name: Check that our config is valid
      command: apache2ctl configtest
      register: result
      ignore_errors: True

    - name: Rolling back - Restoring old default virtualhost
      command: a2ensite default
      when: result|failed

    - name: Rolling back - Removing our virtualhost
      command: a2dissite awesome-app
      when: result|failed

    - name: Rolling back - Ending playbook
      fail: msg="Configuration file is not valid. Please check that before re-
running the playbook."
      when: result|failed

    - name: Deactivates the default virtualhost
      command: a2dissite default

    - name: Deactivates the default ssl virtualhost
      command: a2dissite default-ssl

    notify:
        - restart apache

  handlers:
    - name: restart apache
      service: name=apache2 state=restarted
```

Ключевое слово register записывает вывод команды apache2ctl configtest (exit status, stdout, stderr, ...), и when: result|failed проверяет, содержит ли переменная (result) статус failed.

Поехали:

```
$ ansible-playbook -i step-07/hosts -l host1.example.org step-07/apache.yml

PLAY [web] ********************

GATHERING FACTS ********************
ok: [host1.example.org]

TASK: [Installs apache web server] *********************
ok: [host1.example.org]

TASK: [Push future default virtual host configuration] *********************
ok: [host1.example.org]

TASK: [Activates our virtualhost] *********************
changed: [host1.example.org]

TASK: [Check that our config is valid] *********************
failed: [host1.example.org] => {"changed": true, "cmd": ["apache2ctl",
"configtest"], "delta": "0:00:00.051874", "end": "2013-03-10 10:50:17.714105",
"rc": 1, "start": "2013-03-10 10:50:17.662231"}
stderr: Syntax error on line 2 of /etc/apache2/sites-enabled/awesome-app:
```

```
Invalid command 'RocumentDoot', perhaps misspelled or defined by a module not
included in the server configuration
stdout: Action 'configtest' failed.
The Apache error log may have more information.
...ignoring

TASK: [Rolling back - Restoring old default virtualhost] *********************
changed: [host1.example.org]

TASK: [Rolling back - Removing our virtualhost] *********************
changed: [host1.example.org]

TASK: [Rolling back - Ending playbook] *********************
failed: [host1.example.org] => {"failed": true}
msg: Configuration file is not valid. Please check that before re-running the
playbook.

FATAL: all hosts have already failed -- aborting

PLAY RECAP ********************
host1.example.org              : ok=7    changed=4    unreachable=0    failed=1
```

Кажется, все работает как нужно. Давайте попробуем перезапустить apache:

```
$ ansible -i step-07/hosts -m service -a 'name=apache2 state=restarted'
host1.example.org
host1.example.org | success >> {
    "changed": true,
    "name": "apache2",
    "state": "started"
}
```

Теперь наш Apache защищен от ошибок конфигурации. Помните, переменными можно пользоваться практически везде, так что этот плейбук можно использовать для apache и в других случаях. Напишите один раз и пользуйтесь везде.

Деплоим сайт с помощью Git

Мы установили Apache, добавили виртуальный хост и безопасно перезапустили сервер. Теперь давайте используем модуль git чтобы сделать деплой приложения.

Модуль git

Ну, честно говоря, тут все будет просто, ничего нового. Модуль git это просто еще один модуль. Но давайте попробуем что-нибудь интересное. А позже это пригодится, когда мы будем работать с ansible-pull.

Виртуальный хост задан, но нам нужно внести пару изменений чтобы закончить деплой. Мы деплоим приложение на PHP, так что нужно установить пакет libapache2-mod-php5. Также нужно установить сам git, так как, очевидно, модуль git требует его наличия.

Можно сделать так:

```
    ...
    - name: Installs apache web server
      apt: pkg=apache2 state=installed update_cache=true

    - name: Installs php5 module
      apt: pkg=libapache2-mod-php5 state=installed

    - name: Installs git
      apt: pkg=git state=installed
```

```
    ...
```

но в Ansible есть способ лучше. Он может проходить по набору элементов и использовать каждый в определенном действии, вот так:

```
- hosts: web
  tasks:
    - name: Updates apt cache
      apt: update_cache=true

    - name: Installs necessary packages
      apt: pkg={{ item }} state=latest
      with_items:
        - apache2
        - libapache2-mod-php5
        - git

    - name: Push future default virtual host configuration
      copy: src=files/awesome-app dest=/etc/apache2/sites-available/ mode=0640

    - name: Activates our virtualhost
      command: a2ensite awesome-app

    - name: Check that our config is valid
      command: apache2ctl configtest
      register: result
      ignore_errors: True

    - name: Rolling back - Restoring old default virtualhost
      command: a2ensite default
      when: result|failed

    - name: Rolling back - Removing out virtualhost
      command: a2dissite awesome-app
      when: result|failed

    - name: Rolling back - Ending playbook
      fail: msg="Configuration file is not valid. Please check that before re-
running the playbook."
      when: result|failed

    - name: Deploy our awesome application
      git: repo=https://github.com/leucos/ansible-tuto-demosite.git
dest=/var/www/awesome-app
      tags: deploy

    - name: Deactivates the default virtualhost
      command: a2dissite default

    - name: Deactivates the default ssl virtualhost
      command: a2dissite default-ssl
      notify:
        - restart apache

  handlers:
    - name: restart apache
      service: name=apache2 state=restarted
```

Поехали:

```
$ ansible-playbook -i step-08/hosts -l host1.example.org step-08/apache.yml

PLAY [web] ********************
```

```
GATHERING FACTS ********************
ok: [host1.example.org]

TASK: [Updates apt cache] *********************
ok: [host1.example.org]

TASK: [Installs necessary packages] *********************
changed: [host1.example.org] => (item=apache2,libapache2-mod-php5,git)

TASK: [Push future default virtual host configuration] *********************
changed: [host1.example.org]

TASK: [Activates our virtualhost] *********************
changed: [host1.example.org]

TASK: [Check that our config is valid] *********************
changed: [host1.example.org]

TASK: [Rolling back - Restoring old default virtualhost] *********************
skipping: [host1.example.org]

TASK: [Rolling back - Removing out virtualhost] *********************
skipping: [host1.example.org]

TASK: [Rolling back - Ending playbook] *********************
skipping: [host1.example.org]

TASK: [Deploy our awesome application] *********************
changed: [host1.example.org]

TASK: [Deactivates the default virtualhost] *********************
changed: [host1.example.org]

TASK: [Deactivates the default ssl virtualhost] *********************
changed: [host1.example.org]

NOTIFIED: [restart apache] *********************
changed: [host1.example.org]

PLAY RECAP *********************
host1.example.org              : ok=10   changed=8    unreachable=0    failed=0
```

Теперь можно перейти на http://192.168.33.11 и увидеть котенка и имя сервера.

Строка tags: deploy позволяет запустить определнную порцию плейбука. Допустим, вы запушили новую версию сайта. Вы хотите ускорить процесс и запустить только ту часть, которая ответственна за деплой. Это можно сделать с помощью тегов. Естественно, "deploy" — это просто строка, можно задавать любую. Давайте посмотрим, как это можно использовать:

```
$ ansible-playbook -i step-08/hosts -l host1.example.org step-08/apache.yml -t deploy
X11 forwarding request failed on channel 0

PLAY [web] *****

GATHERING FACTS *****
ok: [host1.example.org]

TASK: [Deploy our awesome application] *****
changed: [host1.example.org]

PLAY RECAP *****
host1.example.org: ok=2 changed=1 unreachable=0 failed=0
```

Добавляем еще один веб-сервер

У нас есть один веб-сервер. Мы хотим два.

Обновление inventory

Мы ожидаем наплыва трафика, так что давайте добавим еще один веб-сервер и балансировщик, который мы настроим в следующем шаге. Давайте закончим с inventory:

```
[web]
host1.example.org ansible_ssh_host=192.168.33.11 ansible_ssh_user=root
host2.example.org ansible_ssh_host=192.168.33.12 ansible_ssh_user=root

[haproxy]
host0.example.org ansible_ssh_host=192.168.33.10 ansible_ssh_user=root
```

Помните, здесь мы указываем ansible_ssh_host потому что хост имеет не тот IP, что ожидается. Можно добавить эти хосты к себе в /etc/hosts или использовать реальные имена (что вы и будете делать в обычной ситуации).

Сборка второго веб-сервера

Мы не зря напрягались перед этим. Деплой второго сервера очень прост:

```
$ ansible-playbook -i step-09/hosts step-09/apache.yml

PLAY [web] *********************

GATHERING FACTS *********************
ok: [host2.example.org]
ok: [host1.example.org]

TASK: [Updates apt cache] *********************
ok: [host1.example.org]
ok: [host2.example.org]

TASK: [Installs necessary packages] *********************
ok: [host1.example.org] => (item=apache2,libapache2-mod-php5,git)
changed: [host2.example.org] => (item=apache2,libapache2-mod-php5,git)

TASK: [Push future default virtual host configuration] *********************
ok: [host1.example.org]
changed: [host2.example.org]

TASK: [Activates our virtualhost] *********************
changed: [host2.example.org]
changed: [host1.example.org]

TASK: [Check that our config is valid] *********************
changed: [host2.example.org]
changed: [host1.example.org]

TASK: [Rolling back - Restoring old default virtualhost] *********************
skipping: [host1.example.org]
skipping: [host2.example.org]

TASK: [Rolling back - Removing out virtualhost] *********************
skipping: [host1.example.org]
skipping: [host2.example.org]
```

```
TASK: [Rolling back - Ending playbook] *********************
skipping: [host1.example.org]
skipping: [host2.example.org]

TASK: [Deploy our awesome application] *********************
ok: [host1.example.org]
changed: [host2.example.org]

TASK: [Deactivates the default virtualhost] ********************
changed: [host1.example.org]
changed: [host2.example.org]

TASK: [Deactivates the default ssl virtualhost] *********************
changed: [host2.example.org]
changed: [host1.example.org]

NOTIFIED: [restart apache] *********************
changed: [host1.example.org]
changed: [host2.example.org]

PLAY RECAP *********************
host1.example.org              : ok=10   changed=5    unreachable=0    failed=0
host2.example.org              : ok=10   changed=8    unreachable=0    failed=0
```

Все, что нужно, это удалить -l host1.example.org из командной строки. Помните, -l позволяет ограничить хосты для запуска. Теперь ограничения не требуется, и запуск произойдет на всех машинах группы web.

Если бы в группе web были другие машины, и нам нужно было бы запустить плейбук только на некоторых из них, можно было бы использовать, например, такое: -l firsthost:secondhost:....

Теперь у нас есть чудесная ферма веб-серверов, давайте превратим ее в кластер с помощью балансировщика нагрузок.

Шаблоны

Мы будем использовать haproxy в качестве балансировщика. Установка такая же, как с apache. Но конфигурация немного сложнее, потому что нам нужно указать список всех веб-серверов в конфигурации haproxy. Как это сделать?

Шаблон конфигурации HAProxy

Ansible использует Jinja2, систему шаблонов для Python. Внутри Jinja2-шаблона можно использовать любую переменную, которая определена Ansible'ом.

Например, если нужно вывести на экран inventory_name хоста, для которого собран шаблон, то можно просто написать {{ inventory_hostname }} в Jinja2-шаблоне. Или, если нужно вывести IP-адрес первого ethernet-интерфейса (о котором Ansible знает благодаря модулю setup), то можно написать {{ ansible_eth1['ipv4']['address'] }}.

Jinja2 также поддерживает условия, циклы и прочее.

Давайте создадим директорию templates/ с Jinja-шаблоном внутри. Назовем его haproxy.cfg.j2. Расширение .j2 даем прото для удобства, оно не обязательно.

```
global
    daemon
    maxconn 256

defaults
    mode http
    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000ms
```

```
listen cluster
    bind {{ ansible_eth1['ipv4']['address'] }}:80
    mode http
    stats enable
    balance roundrobin
{% for backend in groups['web'] %}
    server {{ hostvars[backend]['ansible_hostname'] }} {{ hostvars[backend]
['ansible_eth1']['ipv4']['address'] }} check port 80
{% endfor %}
    option httpchk HEAD /index.php HTTP/1.0
```

Тут есть несколько новых для нас деталей.

Во-первых, {{ ansible_eth1['ipv4']['address'] }} заменится на IP балансировщика нагрузки на eth1.

Дальше у нас есть цикл. Он используется для генерации списка бэкенд-серверов. Каждый шаг цикла соответствует одному хосту из группы [web], и каждый такой хост будет записан в переменную backend. С помощью фактов хоста для каждого из хостов будет сгенерирована строка. Факты всех хостов доступны через переменную hostvars, поэтому достать переменные (например, имя хоста или IP, как в нашем случае) из других хостов очень легко.

Можно было написать список хостов вручную, у нас их всего два. Но мы надеемся, что популярность заставит нас заводить сотни серверов. Так что, при добавлении или изменении серверов нам нужно лишь обновить группу [web].

HAProxy playbook

Самое сложное позади. Написать плейбук для устновки и конфигурации HAproxy очень легко:

```
- hosts: haproxy
  tasks:
    - name: Installs haproxy load balancer
      apt: pkg=haproxy state=installed update_cache=yes

    - name: Pushes configuration
      template: src=templates/haproxy.cfg.j2 dest=/etc/haproxy/haproxy.cfg
mode=0640 owner=root group=root
      notify:
        - restart haproxy

    - name: Sets default starting flag to 1
      lineinfile: dest=/etc/default/haproxy regexp="^ENABLED" line="ENABLED=1"
      notify:
        - restart haproxy

  handlers:
    - name: restart haproxy
      service: name=haproxy state=restarted
```

Выглядит знакомо, правда? Новый модуль тут только один: template. У него такие же аргументы, как у copy. А еще мы ограничили этот плейбук группой haproxy.

А теперь… попробуем. В нашем inventory содержатся только необходимые для кластера хосты, поэтому нам не нужно делать дополнительных ограничений и можно даже запустить оба плейбука. Ну, на самом деле, нам нужно запускать их одновременно, так как haproxy-плейбуку нужны факты из двух веб-серверов. Чуть позже мы узнаем, как избежать этого.

```
$ ansible-playbook -i step-10/hosts step-10/apache.yml step-10/haproxy.yml

PLAY [web] ********************

GATHERING FACTS ********************
ok: [host1.example.org]
ok: [host2.example.org]
```

```
TASK: [Updates apt cache] *********************
ok: [host1.example.org]
ok: [host2.example.org]

TASK: [Installs necessary packages] ***********************
ok: [host1.example.org] => (item=apache2,libapache2-mod-php5,git)
ok: [host2.example.org] => (item=apache2,libapache2-mod-php5,git)

TASK: [Push future default virtual host configuration] ***********************
ok: [host2.example.org]
ok: [host1.example.org]

TASK: [Activates our virtualhost] ***********************
changed: [host1.example.org]
changed: [host2.example.org]

TASK: [Check that our config is valid] ***********************
changed: [host1.example.org]
changed: [host2.example.org]

TASK: [Rolling back - Restoring old default virtualhost] ***********************
skipping: [host1.example.org]
skipping: [host2.example.org]

TASK: [Rolling back - Removing out virtualhost] ***********************
skipping: [host1.example.org]
skipping: [host2.example.org]

TASK: [Rolling back - Ending playbook] ***********************
skipping: [host1.example.org]
skipping: [host2.example.org]

TASK: [Deploy our awesome application] ***********************
ok: [host2.example.org]
ok: [host1.example.org]

TASK: [Deactivates the default virtualhost] ***********************
changed: [host1.example.org]
changed: [host2.example.org]

TASK: [Deactivates the default ssl virtualhost] ***********************
changed: [host2.example.org]
changed: [host1.example.org]

NOTIFIED: [restart apache] ***********************
changed: [host2.example.org]
changed: [host1.example.org]

PLAY RECAP ***********************
host1.example.org              : ok=10   changed=5    unreachable=0    failed=0
host2.example.org              : ok=10   changed=5    unreachable=0    failed=0

PLAY [haproxy] ***********************

GATHERING FACTS ***********************
ok: [host0.example.org]

TASK: [Installs haproxy load balancer] ***********************
changed: [host0.example.org]

TASK: [Pushes configuration] ***********************
changed: [host0.example.org]
```

```
TASK: [Sets default starting flag to 1] *********************
changed: [host0.example.org]

NOTIFIED: [restart haproxy] *********************
changed: [host0.example.org]

PLAY RECAP *********************
host0.example.org              : ok=5    changed=4    unreachable=0    failed=0
```

Вроде все хорошо. Зайдите на http://192.168.33.10/ и оцените результат. Кластер задеплоен! Можно даже посмотреть на статистику HAproxy: http://192.168.33.10/haproxy?stats.

Снова переменные

Итак, мы установили балансировщик нагрузки, и он работает нормально. Мы берем переменные из фактов и используем их для генерации конфигурации.

Ansible также поддерживает другие виды переменных. Мы уже видели ansible_ssh_host в файле inventory, но теперь используем переменные, которые заданы в файлах host_vars и group_vars.

Тонкая настройка конфигурации HAProxy

Обычно HAProxy проверяет, живы ли бэкенды. Если бэкенд не откликается, то он удаляется из пула, и HAProxy больше не шлет ему запросы.

У бэкэндов может быть указан вес (от 0 до 256). Чем выше вес, тем больше запросов сервер получит по сравнению с другими серверами. Это полезно, когда узлы отличаются по мощности и нужно направить трафик в соответствии с этим.

Мы используем переменные для настройки этих параметров.

Group-переменные

Интервал проверки haproxy будет задан в файле group_vars. Таким образом, все экземпляры haproxy унаследуют это.

Нужно создать файл group_vars/haproxy внутри директории inventory. Название файла должно совпадать с названием группы, для которой задаются переменные. Если бы мы задавали переменные для группы web, то назвали бы файл group_vars/web.

```
haproxy_check_interval: 3000
haproxy_stats_socket: /tmp/sock
```

Название переменной может быть любым. Естественно, рекомендуется давать осмысленные названия, но каких-то специальных правил нет. Можно делать даже комплексные переменные (то есть Python dict) вот так:

```
haproxy:
    check_interval: 3000
    stats_socket: /tmp/sock
```

Это дело вкуса. Такой подход позволяет делать логическую группировку. Мы пока будем использовать простые переменные.

Переменные хоста

С переменными хоста такая же история, но файлы живут в директори host_vars. Давайте зададим вес бэкенда в host_vars/host1.example.com:

```
haproxy_backend_weight: 100
```

и для host_vars/host2.example.com:

```
haproxy_backend_weight: 150
```

Если бы мы задали haproxy_backend_weight в group_vars/web, то он бы использовался по-умолчанию: переменные из файла host_vars имеют приоритет перед переменными из group_vars.

Обновляем шаблон

Теперь необходимо обновить шаблон, чтобы он использовал эти переменные.

```
global
    daemon
    maxconn 256
{% if haproxy_stats_socket %}
    stats socket {{ haproxy_stats_socket }}
{% endif %}

defaults
    mode http
    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000ms

listen cluster
    bind {{ ansible_eth1['ipv4']['address'] }}:80
    mode http
    stats enable
    balance roundrobin
{% for backend in groups['web'] %}
    server {{ hostvars[backend]['ansible_hostname'] }} {{ hostvars[backend]
['ansible_eth1']['ipv4']['address'] }} check inter {{ haproxy_check_interval }}
weight {{ hostvars[backend]['haproxy_backend_weight'] }} port 80
{% endfor %}
    option httpchk HEAD /index.php HTTP/1.0
```

Заметили блок {% if ...? Этот блок будет отработан, если условие верно. Так что, если мы где-нибудь зададим haproxy_stats_socket для балансировщика нагрузки (можно даже добавить --extra-vars="haproxy_stats_sockets=/tmp/sock" при вызове из командной строки), то блок будет добавлен в сгенерированный конфигурационный файл.

Имейте ввиду, что такой метод очень плох с точки зрения безопасности!

Поехали:

```
ansible-playbook -i step-11/hosts step-11/haproxy.yml
```

Мы можем, но не обязаны запускать плейбук apache, потому что ничего не изменилось. Но пришлось добавить небольшой трюк. Вот обновленный плейбук haproxy:

```
- hosts: web
- hosts: haproxy
  tasks:
    - name: Installs haproxy load balancer
      apt: pkg=haproxy state=installed update_cache=yes

    - name: Pushes configuration
      template: src=templates/haproxy.cfg.j2 dest=/etc/haproxy/haproxy.cfg
mode=0640 owner=root group=root
      notify:
```

```
      - restart haproxy

  - name: Sets default starting flag to 1
    lineinfile: dest=/etc/default/haproxy regexp="^ENABLED" line="ENABLED=1"
    notify:
      - restart haproxy

handlers:
  - name: restart haproxy
    service: name=haproxy state=restarted
```

Видите? Мы добавили пустой блок для веб-хостов в самом начале. В нем ничего не происходит. Но факт его наличия заставит Ansible собрать факты для группы web. Это необходимо, потому что плейбук haproxy использует факты из этой группы. Если не сделать этого, то Ansible будет ругаться, что ключа ansible_eth1 не существует.

Мигрируем к ролям!

Теперь, когда все плейбуки готовы, давайте все отрефакторим! Мы переделаем все через роли. Роли — это просто еще один способ организации файлов, но у них есть несколько интересных возможностей. Не будем вдаваться в детали, все они описаны в документации Ansible. Моя любимая фича — это зависимости ролей: роль B может зависеть от другой роли A. Поэтому при применении роли B, автоматически будет применена роль A.

Структура ролей

Роли добавляют немного «магии» в Ansible: они предполагают особую организацию файлов. Роли полагается структурировать определенным образом, хотя вы можете делать это как угодно вам. Тем не менее, если придерживаться соглашений, вам будет гораздо легче создавать модульные плейбуки. Содержать код в порядке будет гораздо легче. Рубисты называют это "convention over configuration".

Структура файлов для ролей такая:

```
roles
  |
  |_some_role
       |
       |_files
       |   |
       |   |_file1
       |   |_...
       |
       |_templates
       |   |
       |   |_template1.j2
       |   |_...
       |
       |_tasks
       |   |
       |   |_main.yml
       |   |_some_other_file.yml
       |   |_ ...
       |
       |_handlers
       |    |
       |    |_main.yml
       |    |_some_other_file.yml
       |    |_ ...
       |
       |_vars
       |    |
       |    |_main.yml
       |    |_some_other_file.yml
```

```
        |    |_ ...
        |
        |_meta
             |
             |_main.yml
             |_some_other_file.yml
             |_ ...
```

Довольно просто.

Файлы main.yml не обязательны. Но если они присутствуют, то роли добавят их к отработке автоматически. Эти файлы можно использовать для добавления других тасков и хэндлеров.

Обратите внимание на директории vars и meta. vars нужна для случаев, когда есть куча переменных, связанных с ролью. Но мне лично не нравится задавать переменные в ролях и сценариях напрямую. Я считаю, что переменные должны быть частью конфигурации, а сценарии — это структура. Иными словами, я считаю сценарии фабриками, а данные — параметрами для фабрик. Поэтому я предпочитаю видеть «данные» (например, переменные) вне ролей и сценариев. Тогда мне легче шарить роли и не раскрывать слишком много информации о внутренностях серверов. Но это дело личных предпочтений. Ansible предоставляет вам выбор.

В директории meta находятся зависимости, но об этом поговорим в следующий раз. Сценарии лежат в директории roles.

Создаем роль Apache

Теперь у нас достаточно знаний, чтобы создать роль для apache на основе нашего плейбука.

Несколько простых шагов:

- создать директорию ролей и структуру роли apache
- вынести хэндлер apache в файл roles/apache/handlers/main.yml
- перенести конфигурационный файл apache awesome-app в roles/apache/files/
- создать плейбук для роли

Задаем структуру

Все просто:

```
mkdir -p step-12/roles/apache/{tasks,handlers,files}
```

Теперь копируем таски из apache.yml в main.yml. Файл выглядит так:

```
- name: Updates apt cache
  apt: update_cache=true

- name: Installs necessary packages
  apt: pkg={{ item }} state=latest
  with_items:
    - apache2
    - libapache2-mod-php5
    - git

...

- name: Deactivates the default ssl virtualhost
  command: a2dissite default-ssl
  notify:
    - restart apache
```

Это не полный текст файла, а просто иллюстрация. Файл в точности повторяет содержание apache.yml между tasks: и handlers:.

Мы также убрали обращения к директориям files/ и templates/ в тасках. Так как используется стандартная структура ролей, Ansible сам знает, в какие директории смотреть.

Выносим хэндлер

Нужно создать файл step-12/roles/apache/handlers/main.yml:

```
- name: restart apache
  service: name=apache2 state=restarted
```

Переносим файл конфигурации

Еще проще:

```
cp step-11/files/awesome-app step-12/roles/apache/files/
```

Роль apache работает. Но нам нужен способ запустить ее.

Создаем плейбук роли

Давайте создадим плейбук верхнего уровня для связывания хостов и групп хостов с ролями. Назовем файл site.yml, так как нам нужна общая конфигурация сайта. Заодно добавим туда haproxy:

```
- hosts: web
  roles:
    - { role: apache }

- hosts: haproxy
  roles:
    - { role: haproxy }
```

Совсем не сложно. Теперь давайте создадим роль haproxy:

```
mkdir -p step-12/roles/haproxy/{tasks,handlers,templates}
cp step-11/templates/haproxy.cfg.j2 step-12/roles/haproxy/templates/
```

потом извлечем хэндлер и удалим упоминание templates/.

Попробуем?:

```
ansible-playbook -i step-12/hosts step-12/site.yml
```

Если все хорошо, то мы увидим "PLAY RECAP":

```
host0.example.org          : ok=5    changed=2    unreachable=0 failed=0
host1.example.org          : ok=10   changed=5    unreachable=0 failed=0
host2.example.org          : ok=10   changed=5    unreachable=0 failed=0
```

Вы наверное заметили, что запуск всех ролей в site.yml занимает много времени. Что если нужно сделать изменения только для веб-серверов? Легко! Используем limit-флаг:

```
ansible-playbook -i step-12/hosts -l web step-12/site.yml
```

На этом миграция на роли закончена.