Разработка и тестирование Ansible ролей и плейбуков

Проект infra и проверка ДЗ

Создайте новую ветку в вашем инфраструктурном репозитории для выполнения данного ДЗ. Т.к. это четвертое задание, посвященое работе с Ansible, то ветку назовите **ansible-4**.

Проверка данного ДЗ будет производиться через Pull Request ветки с ДЗ к ветке мастер.

После того, как один из преподавателей сделает approve пул реквеста, ветку с ДЗ можно смерджить.

План

- Локальная разработка при помощи Vagrant, доработка ролей для провижининга в Vagrant
- Тестирование ролей при помощи Molecule и Testinfra
- Переключение сбора образов пакером на использование ролей
- * Подключение Travis CI для автоматического прогона тестов

Локальная разработка с Vagrant

Установка Vagrant

- Установите VirtualBox на вашу локальную машину. VritualBox - один из провайдеров, которым Vagrant может управлять для создания VMs. Мы будем использовать данный провайдер для локального запуска VM.

- Установим сам Vagrant, скачав нужный пакет для вашей ОС. Чтобы проверить установку можно воспользоваться командой:

```
$ vagrant -v
```

Опишем локальную инфраструктуру

Описание характеристик VMs, которые мы хотим создать, должно содержаться в файле с названием **Vagrantfile**.

Мы создадим инфраструктуру, которую мы создавали до этого в GCE при помощи Terraform, на своей локальной машине, используя Vagrant.

.gitignore

Перед началом работы с Vagrant добавим следующие строки в наш .gitignore файл, чтобы не комитить информацию о создаваемых Vagrant машинах и логах

```
.gitignore
... <- предыдущие записи
# Vagrant & molecule
.vagrant/
*.log
*.pyc
.molecule
.cache
.pytest_cache
```

Vagrantfile

В директории **ansible** создайте файл **Vagrantfile** с определением **двух** VM:

ansible/Vagrantfile (**ссылка на gist**)

```
Vagrant.configure("2") do |config|

  config.vm.provider :virtualbox do |v|
    v.memory = 512
  end

  config.vm.define "dbserver" do |db|
    db.vm.box = "ubuntu/xenial64"
    db.vm.hostname = "dbserver"
    db.vm.network :private_network, ip: "10.10.10.10"
  end
  config.vm.define "appserver" do |app|
    app.vm.box = "ubuntu/xenial64"
    app.vm.hostname = "appserver"
    app.vm.network :private_network, ip: "10.10.10.20"
  end
end
```

количество памяти, выделяемое провайдером под VMs

Имя VM

название бокса (образа VM)

IP адрес внутреннего интерфейса

Создание VM

Создадим виртуалки, описанные в Vagrantfile. Выполните следующую команду, в директории **ansible**, где находится Vagrantfile:

```
$ vagrant up
Bringing machine 'dbserver' up with 'virtualbox' provider...
Bringing machine 'appserver' up with 'virtualbox' provider...
==> dbserver: Box 'ubuntu/xenial64' could not be found. Attempting to find and install...
    dbserver: Box Provider: virtualbox
    dbserver: Box Version: >= 0
==> dbserver: Loading metadata for box 'ubuntu/xenial64'
    dbserver: URL: https://atlas.hashicorp.com/ubuntu/xenial64
==> dbserver: Adding box 'ubuntu/xenial64' (v20170922.0.0) for provider: virtualbox
    dbserver: Downloading: https://vagrantcloud.com/ubuntu/boxes/xenial64/versions/
20170922.0.0/providers/virtualbox.box
```

Если у вас еще нет указанного бокса (образа VM) на локальной машине, то Vagrant попытается его скачать с Vagrant Cloud - главного хранилища Vagrant боксов, откуда Vagrant скачивает образы по умолчанию.

Проверка работы VMs

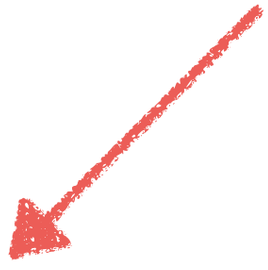
Проверим, что бокс скачался на нашу локальную машину:

```
$ vagrant box list
google/gce       (google, 0.1.0)
ubuntu/precise64 (virtualbox, 20170427.0.0)
ubuntu/trusty64  (virtualbox, 20170619.0.0)
ubuntu/xenial64  (virtualbox, 20170922.0.0)
```

Проверим статус VMs:

```
$ vagrant status
Current machine states:
dbserver                  running (virtualbox)
appserver                 running (virtualbox)
```

VMs запущены



Проверка работы VMs

Проверим SSH доступ к VM с названием **appserver** и проверим пинг хоста **dbserver** по адресу, который мы указали в Vagrantfile

```
$ vagrant ssh appserver
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-96-generic x86_64)

Last login: Tue Sep 26 14:12:35 2017 from 10.0.2.2
ubuntu@appserver:~$ ping -c 2 10.10.10.10
PING 10.10.10.10 (10.10.10.10) 56(84) bytes of data.
64 bytes from 10.10.10.10: icmp_seq=1 ttl=64 time=0.025 ms
64 bytes from 10.10.10.10: icmp_seq=2 ttl=64 time=0.021 ms

--- 10.10.10.10 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 999ms
rtt min/avg/max/mdev = 0.021/0.023/0.025/0.002 ms

ubuntu@appserver:~$ exit
logout
Connection to 127.0.0.1 closed.
```

Доработка ролей

Провижининг

Vagrant поддерживает большое количество провижинеров, которые позволяют автоматизировать процесс конфигурации созданных VMs с использованием популярных инструментов управления конфигурацией и обычных скриптов на bash.

Мы будем использовать **Ansible** провижинер для проверки работы наших ролей и плейбуков.

Добавим провижинер в

Начнем с доработки **db** роли.
Добавим провижининг в определение хоста **dbserver**:

ansible/Vagrantfile (**ссылка на gist**)

```
config.vm.define "dbserver" do |db|
  db.vm.box = "ubuntu/xenial64"
  db.vm.hostname = "dbserver"
  db.vm.network :private_network, ip: "10.10.10.10"

  db.vm.provision "ansible" do |ansible|
    ansible.playbook = "playbooks/site.yml"
    ansible.groups = {
    "db" => ["dbserver"],
    "db:vars" => {"mongo_bind_ip" => "0.0.0.0"}
    }
  end
end
```

Определение провижинера

Какой плейбук запускать

Определение группы хостов и переменных

Запуск провижинера

Провижининг происходит автоматически при запуске новой машины. Если же мы хотим применить провижининг на уже запущенной машине, то необходимо использовать команду provision.

Если мы хотим применить команду для конкретного хоста, то нам также нужно передать его имя в качестве аргумента.

Запуск провижинера

Применение конфигурации провалилось :(

```
$ vagrant provision dbserver

==> dbserver: Running provisioner: ansible...
    dbserver: Running ansible-playbook...

PLAY [Configure MongoDB] *******************************************************
TASK [Gathering Facts] *********************************************************
fatal: [dbserver]: FAILED! => {"changed": false, "failed": true, "module_stderr": "Shared connection to
127.0.0.1 closed.\r\n", "module_stdout": "/bin/sh: 1:
/usr/bin/python: not found\r\n", "msg": "MODULE FAILURE", "rc": 0}
        to retry, use: --limit @/Users/user/hw133/ansible/site.retry

PLAY RECAP *********************************************************************
dbserver                   : ok=0    changed=0    unreachable=0    failed=1

Ansible failed to complete successfully. Any error output should be
visible above. Please fix these errors and try again.
```

Из ошибки видим, что Ansible не может найти питон нужной ему версии 2.X

Добавим python

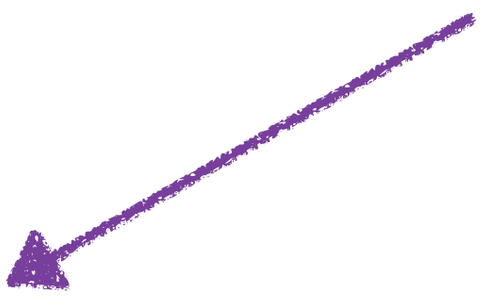
С одной стороны мы можем добавить установку Python в плейбуки **ansible/playbooks/db.yml** и в **ansible/playbooks/app.yml**. С другой стороны, это будет дублирование кода. И это действие нам все равно нужно выполнять для всех хостов.

Поэтому добавим еще один плейбук **base.yml**, в котором опишем установку python. И добавим его в **site.yml**. Таким образом, он будет выполняться для всех хостов в начале выполнения.

Заодно удалим плейбук **users.yml** из **site.yml**

ansible/site.yml

```
---
- include: base.yml
- include: users.yml
- include: db.yml
- include: app.yml
- include: deploy.yml
```



Добавим python

Используем raw модуль, который позволяет запускать команды по SSH и не требует наличия python на управляемом хосте. Отменим также сбор фактов ансиблом, т.к. данный процесс требует установленного python и выполняется перед началом применения конфигурации

ansible/playbooks/base.yml (**ссылка на gist**)

```
---
- name: Check && install python
  hosts: all
  become: true
  gather_facts: False

  tasks:
    - name: Install python for Ansible
      raw: test -e /usr/bin/python II (apt -y update && apt install -y python-minimal)
      changed_when: False
```

Проверка провижининга

Повторим попытку провижининга хоста **dbserver**:

```
$ vagrant provision dbserver

==> dbserver: Running provisioner: ansible...
TASK [db : Show info about the env this host belongs to] ************************
ok: [dbserver] => {
    "msg": "This host is in local environment!!!"
}

TASK [db : Change mongo config file] *******************************************
changed: [dbserver]

RUNNING HANDLER [db : restart mongod] ******************************************
fatal: [dbserver]: FAILED! => {"changed": false, "failed": true, "msg": "Could not find
the requested service mongod: host"}
        to retry, use: --limit @/Users/user/hw133/ansible/site.retry
```

Доработка роли db

На предыдущем слайде мы видим, что провижининг заработал, т.е. Ansible прогнал таски роли: положил конфиг монги и попытался ее перезапустить, но... не нашел установленной монги.

Помним, что установка MongoDB у нас производилась в отдельном плейбуке **packer_db.yml**, который использовался в качестве провижинера в Packer. Включим этот плейбук в нашу роль, чтобы роль **db** позволяла управлять всем жизненным циклом нашей БД, включая ее установку.

Доработка роли db

Изменим роль **db**, добавив файл тасков **db/tasks/install_mongo.yml** для установки MongoDB. Добавим к каждому таску тег **install**, пометив его как шаг установки.

Скопируйте таски из файла **packer_db.yml** или данного gist и вставьте их в файл **db/tasks/install_mongo.yml**

Несколько файлов тасков

Поскольку наши роли начинают включать в себя все больше тасков, то мы начинаем группировать их по разным файлам. Мы уже вынесли таски установки MongoDB в отдельный файл роли, аналогично поступим для тасков управления конфигурацией.

Вынесем таски управления конфигом монги в отдельный файл **config_mongo.yml**.

Доработка роли db

db/tasks/config_mongo.yml (ссылка на gist)

```
---
- name: Change mongo config file
  template:
    src: templates/mongod.conf.j2
    dest: /etc/mongod.conf
    mode: 0644
  notify: restart mongod
```

main.yml

В файле **main.yml** роли будем вызывать таски в нужном нам порядке:

db/tasks/main.yml (ссылка на gist)

```
---
# tasks file for db

- name: Show info about the env this host belongs to
  debug:
    msg: "This host is in {{ env }} environment!!!"

- include: install_mongo.yml
- include: config_mongo.yml
```

Проверим работу роли

Применим роль для локальной машины **dbserver**:

```
$ vagrant provision dbserver

==> dbserver: Running provisioner: ansible...

TASK [db : Add APT key] ********************************************************
changed: [dbserver]

TASK [db : Add APT repository] *************************************************
changed: [dbserver]

TASK [db : Install mongodb package] ********************************************
changed: [dbserver]

PLAY RECAP *********************************************************************
dbserver                   : ok=7    changed=4    unreachable=0    failed=0
```


Проверим работу роли

Видим, что провижининг выполнился успешно. Проверим доступность порта монги для хоста **appserver**, используя команду telnet:

```
$ vagrant ssh appserver
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-96-generic x86_64)

Last login: Tue Sep 26 14:13:40 2017 from 10.0.2.2

ubuntu@appserver:~$ telnet 10.10.10.10 27017
Trying 10.10.10.10...
Connected to 10.10.10.10.
Escape character is '^]'.
```

Порт для проверки

IP адрес dbserver

Подключение удалось, значит порт доступен для хоста appserver и конфигурация роли верна.

Доработка роли app

Аналогично роли **db** мы включим в нашу роль **app** конфигурацию из **packer_app.yml** плейбука, необходимую для настройки хоста приложения.

Создадим новый файл для тасков **ruby.yml** внутри роли app и скопируем в него таски из плейбука **packer_app.yml**

Доработка роли app

app/tasks/ruby.yml (**ссылка на gist**)

```
---
- name: Install ruby and rubygems and required packages
  apt: "name={{ item }} state=present"
  with_items:
    - ruby-full
    - ruby-bundler
    - build-essential
  tags: ruby
```

Доработка роли app

Вынесем настройки puma сервера также в отдельный файл для тасков в рамках роли. Создадим файл **app/tasks/puma.yml** и скопируем в него таски из **app/tasks/main.yml**, относящиеся к настройке Puma сервера и запуску приложения.

app/tasks/puma.yml (ссылка на gist)

```
---
- name: Add unit file for Puma
  copy:
    src: puma.service
    dest: /etc/systemd/system/puma.service
  notify: reload puma

- name: Add config for DB connection
  template:
    src: db_config.j2
    dest: /home/appuser/db_config
    owner: appuser
    group: appuser

- name: enable puma
  systemd: name=puma enabled=yes
```

main.yml

В файле **main.yml** роли будем вызывать таски в нужном нам порядке:

app/tasks/main.yml (**ссылка на gist**)

```
---
# tasks file for app

- name: Show info about the env this host belongs to
  debug:
    msg: "This host is in {{ env }} environment!!!"

- include: ruby.yml
- include: puma.yml
```

Провижиним appserver

Аналогично **dbserver** определим Ansible провижинер для хоста **appserver** в Vagrantfile:

ansible/Vagrantfile (**ссылка на gist**)

```
...
...
config.vm.define "appserver" do |app|
    app.vm.box = "ubuntu/xenial64"
    app.vm.hostname = "appserver"
    app.vm.network :private_network, ip: "10.10.10.20"

    app.vm.provision "ansible" do |ansible|
      ansible.playbook = "site.yml"
      ansible.groups = {
      "app" => ["appserver"],
      "app:vars" => { "db_host" => "10.10.10.10"}
      }
    end
  end
```

inventory

Вы могли задаться вопросом, почему мы нигде не указываем инвентори файл, а указываем странные опции, вроде:

```
ansible.groups = {
"app" => ["appserver"],
"app:vars" => { "db_host" => "10.10.10.10"}
}
```

Дело в том, что Vagrant динамически генерирует инвентори файл для провижининга в соответствии с конфигурацией в **Vagrantfile**.

То есть передавая опции выше, у нас будет создаваться группа [**app**], в которой будет один хост **appserver** (что соответсвует создаваемой VM). Далее мы определяем переменные для данной группы app.

inventory

Вы можете посмотреть, какой инвентори файл Vagrant сгенерировал при провижининге **dbserver**.

```
$ cat .vagrant/provisioners/ansible/inventory/vagrant_ansible_inventory
# Generated by Vagrant
...
dbserver ansible_ssh_host=127.0.0.1 ansible_ssh_port=2201 ansible_ssh_user='ubuntu'
ansible_ssh_private_key_file='/Users/user/hw133/ansible/.vagrant/machines/dbserver/
virtualbox/private_key'

[db]
dbserver

[db:vars]
mongo_bind_ip=0.0.0.0
```

Проверка роли

Применим провижининг для хоста **appserver**:

```
$ vagrant provision appserver

==> appserver: Running provisioner: ansible...

TASK [app : Install Ruby bundler] ***********************************************
changed: [appserver]

TASK [app : Add unit file for Puma] *********************************************
changed: [appserver]

TASK [app : Add config for DB connection] ***************************************
fatal: [appserver]: FAILED! => {"changed": false, "checksum": "dfbe4b5cf3ec32d91d20045e2ee7f7b26c60ef34",
"msg": "Destination directory /home/appuser does not exist"}

RUNNING HANDLER [app : reload puma] **********************************************
        to retry, use: --limit @/Users/user/hw133/ansible/site.retry

PLAY RECAP **********************************************************************
appserver                  : ok=5    changed=3    unreachable=0    failed=1

Ansible failed to complete successfully. Any error output should be
visible above. Please fix these errors and try again.
```

Доработка роли app

На предыдущем слайде мы можем видеть, что наш провижининг работает и наша роль произвела некоторые настройки как, например, установка **ruby**, **bundler**, **unit** файла для Puma.

Но Ansible не удалось создать файл с настройками подключения к БД, потому что данный файл он пытается создать в домашней директории пользователя **appuser**, которого у нас нет.

У нас есть два варианта решения проблемы: 1) создать пользователя, как часть роли; 2) параметризировать нашу конфигурацию, чтобы мы могли использовать ее для пользователя другого, чем **appuser**.

Мы пойдем по второму пути.

Параметризации роли

В нашей роли мы захардкодили пути установки конфигов и деплоя приложения в домашнюю директорию пользователя **appuser**. Параметризуем имя пользователя, чтобы дать возможность использовать роль для иного пользователя.

Определим переменную по умолчанию внутри нашей роли:

app/defaults/main.yml
```
---
# defaults file for app

db_host: 127.0.0.1
env: local
deploy_user: appuser
```

puma.yml

Рассмотрим таски, определенные в файле **puma.yml**. Первым делом, заменим модуль для копирования **unit** файла с **copy** на **template**, чтобы иметь возможность параметризировать unit файл:

app/tasks/puma.yml (ссылка на gist)

```
---
- name: Add unit file for Puma
  template:
    src: puma.service.j2
    dest: /etc/systemd/system/puma.service
  notify: reload puma
```

puma.yml

Далее параметризуем сам unit файл. Переместим его из директории **app/files** в директорию **app/templates**, т.к. мы поменяли используемый для копирования модуль и добавим к файлу **puma.service** расширение **.j2**, чтобы обозначить данный файл как шаблон.

Заменим в созданном шаблоне все упоминания **appuser** на переменную **deploy_user** (см. след слайд)

puma.yml

app/templates/puma.service.j2 (**ссылка на gist**)

```
[Unit]
Description=Puma HTTP Server
After=network.target

[Service]
Type=simple
EnvironmentFile=/home/{{ deploy_user }}/db_config
User={{ deploy_user }}
WorkingDirectory=/home/{{ deploy_user }}/reddit
ExecStart=/bin/bash -lc 'puma'
Restart=always

[Install]
WantedBy=multi-user.target
```

puma.yml

Снова обратимся к **app/tasks/puma.yml** и параметризуем оставшуюся конфигурацию

app/tasks/puma.yml (**ссылка на gist**)

```
---
- name: Add unit file for Puma
  template:
    src: puma.service.j2
    dest: /etc/systemd/system/puma.service
  notify: reload puma

- name: Add config for DB connection
  template:
    src: db_config.j2
    dest: "/home/{{ deploy_user }}/db_config"
    owner: "{{ deploy_user }}"
    group: "{{ deploy_user }}"

- name: enable puma
  systemd: name=puma enabled=yes
```

deploy.yml

Для провижининга хоста **appserver** мы использовали плейбук **site.yml**.

Данный плейбук, помимо плейбука **app.yml**, также вызывает плейбук **ansible/playbooks/deploy.yml**, который применяется для группы хостов **app** и который нам тоже нужно не забыть параметризировать.

deploy.yml

ansible/playbooks/deploy.yml (**ссылка на gist**)

```
---
- name: Deploy App
  hosts: app
  vars:
    deploy_user: appuser

  tasks:
  - name: Fetch the latest version of application code
    git:
      repo: 'https://github.com/express42/reddit.git'
      dest: "/home/{{ deploy_user }}/reddit"
      version: monolith
    notify: restart puma

  - name: bundle install
    bundler:
      state: present
      chdir: "/home/{{ deploy_user }}/reddit"

 handlers:
 - name: restart puma
   become: true
   systemd: name=puma state=restarted
```

Переопределение переменных

Мы ввели дополнительную переменную для пользователя, запускающего приложение и параметризировали нашу конфигурацию. Теперь при вызове плейбуков для **appserver** переопределим дефолтное значение переменной пользователя на имя пользователя используемое нашим боксом по умолчанию, т.е. **ubuntu**.

Используем при этом переменные **extra_vars**, имеющие самый высокий приоритет по сравнению со всеми остальными.

Переопределение переменных

Добавим extra_vars переменные в блок определения провижинера в **Vagrantfile**

ansible/Vagrantfile (**ссылка на gist**)

```
...
app.vm.provision "ansible" do |ansible|
      ansible.playbook = "site.yml"
      ansible.groups = {
      "app" => ["appserver"],
      "app:vars" => { "db_host" => "10.10.10.10"}
      }
      ansible.extra_vars = {
        "deploy_user" => "ubuntu"
      }
end
```

Проверка роли

Применим провижининг для хоста **appserver**:

```
$ vagrant provision appserver

==> appserver: Running provisioner: ansible...

TASK [Fetch the latest version of application code] ****************************
changed: [appserver]

TASK [bundle install] **********************************************************
changed: [appserver]

RUNNING HANDLER [restart puma] *************************************************
changed: [appserver]

PLAY RECAP *********************************************************************
appserver                  : ok=11   changed=5    unreachable=0    failed=0
```

Проверка роли

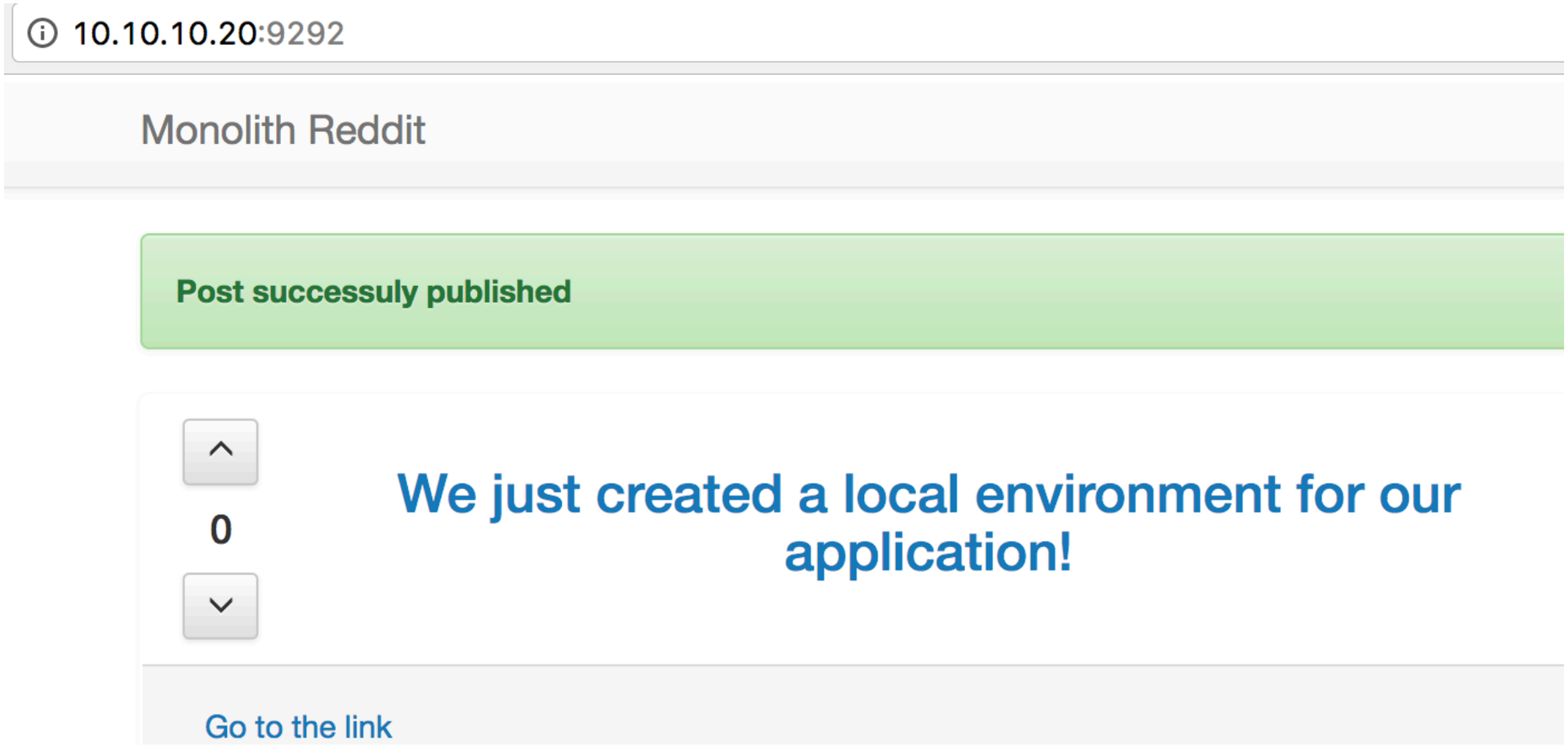
```
TASK [Fetch the latest version of application code] ****************************
fatal: [appserver]: FAILED! => {"changed": false, "cmd": "/usr/bin/git clone --origin origin https://github.com/Otus-DevOps-2017-11/reddit.git /home/ubuntu/
e dir '/home/ubuntu/reddit': Permission denied", "rc": 128, "stderr": "fatal: could not create work tree dir '/home/ubuntu/reddit': Permission denied\n", "s
 dir '/home/ubuntu/reddit': Permission denied"], "stdout": "", "stdout_lines": []}

PLAY RECAP *********************************************************************
appserver                  : ok=24   changed=16   unreachable=0    failed=1
```

В **случае**, если у вас **vagrant provision** падает с ошибкой из-за невозможности записать в директорию **/home/ubuntu**, проверьте под каким пользователем Vagrant выполняет плейбуки.

При необходимости измените имя пользователя в extra_vars.

Проверим работу приложения

Откроем страницу в браузере по адресу хоста appserver, т.е. 10.10.10.20:9292



Проверим конфигурацию

Выполните в директории ansible команду по удалению созданных машин:

$ vagrant destroy -f

Теперь создадим окружение и проверим что все корректно настроится и задеплоится:

$ vagrant up

Проверьте отсутсвие ошибок и что приложение корректно работает.
Удалите окружение:

$ vagrant destroy -f

Задание со *

Как мы видим из лога, **nginx** у нас также настраивается для **appserver** в процессе провижининга. Но если мы попробуем открыть адрес 10.10.10.20, то приложения там не будет.

- Дополните конфигурацию Vagrant для корректной работы проксирования приложения с помощью nginx

Тестирование роли

Установка зависимостей

Для локального тестирования Ansible ролей будем использовать Molecule для создания машин и проверки конфигурации и Testinfra для написания тестов.

Сначала установим все необходимые компоненты для тестирования: Molecule, Ansible, Testinfra на локальную машину используя **pip**.

Установку данных модулей рекомендуется выполнять в созданной через virtualenv среде работы с питоном. Инструкции по установке virtualenv и virtualenvwrapper можно посмотреть здесь.

P.S. Коммитить в репозиторий созданный virtualenv НЕ нужно!!!

Установка зависимостей

Добавьте в файл **requirements.txt** в директории **ansible** следующие записи:

ansible/requirements.txt

```
ansible>=2.4
molecule>=2.6
testinfra>=1.10
python-vagrant>=0.5.15
```

```
$ pip install -r requirements.txt
```

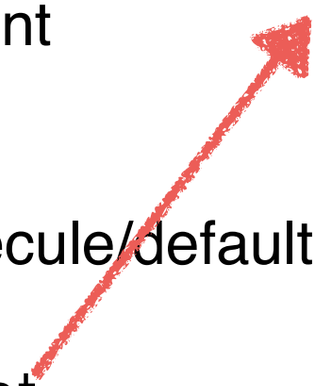
```
$ molecule --version
molecule, version 2.7.0
```

```
$ ansible --version
ansible 2.4.x.x
```

Тестирование db роли

Используем команду molecule init для создания заготовки тестов для роли db. Выполните команду ниже в директории с ролью ansible/roles/db:

```
$ molecule init scenario --scenario-name default -r db -d vagrant

--> Initializing new scenario default...
Initialized scenario in /Users/user/hw133/ansible/roles/db/molecule/default successfully.
```



Указываем Vagrant как драйвер для создания VMs

Тестирование db роли

Добавим несколько тестов, используя модули Testinfra, для проверки конфигурации, настраиваемой ролью **db**:

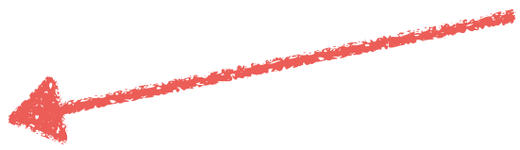
db/molecule/default/tests/test_default.py (ссылка на gist)

```
...
# check if MongoDB is enabled and running
def test_mongo_running_and_enabled(host):
    mongo = host.service("mongod")
    assert mongo.is_running
    assert mongo.is_enabled

# check if configuration file contains the required line
def test_config_file(File):
    config_file = host.file('/etc/mongod.conf')
    assert config_file.contains('bindIp: 0.0.0.0')
    assert config_file.is_file
```

Создание тестовой машины

Описание тестовой машины, которая создается Molecule для тестов содержится в файле **db/molecule/default/molecule.yml**

```
---
...
driver:
  name: vagrant
  provider:
    name: virtualbox
lint:
  name: yamllint
platforms:
  - name: instance
    box: ubuntu/xenial64
provisioner:
  name: ansible
  lint:
    name: ansible-lint
```



По умолчанию используется нужный нам бокс

Создание тестовой машины

Создадим VM для проверки роли. В директории **ansible/roles/db** выполните команду:

```
$ molecule create
--> Test matrix
--> Action: 'create'
    PLAY [Create] ******************************************************************
    TASK [Create molecule instance(s)] *********************************************
    changed: [localhost] => (item={'box': u'ubuntu/xenial64', 'name': u'instance'})
```

Посмотрим список созданных инстансов, которыми управляет Molecule:

```
$ molecule list
Instance Name    Driver Name    Provisioner Name    Created    Converged
---------------  -------------  ------------------  ---------  -----------
instance         Vagrant        Ansible             True       False
```

Тестирование роли

Также можем при необходимости дебага подключиться по SSH внутрь VM:

```
$ molecule list
Instance Name    Driver Name    Provisioner Name      Created    Converged
---------------  -------------  ------------------  ---------  -----------
instance         Vagrant        Ansible                True       False

$ molecule login -h instance

Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-96-generic x86_64)

Last login: Tue Sep 26 19:48:15 2017 from 10.0.2.2

ubuntu@instance:~$ exit
```

playbook.yml

Molecule init генерирует плейбук для применения нашей роли. Данный плейбук можно посмотреть по пути **db/molecule/default/playbook.yml**.

Поскольку таски нашей роли требуют выполнения из-под суперпользователя, то добавим **become** в определение сценария этого плейбука.

Дополнительно зададим еще переменную mongo_bind_ip.

playbook.yml

```
---
- name: Converge
  become: true
  hosts: all
  vars:
    mongo_bind_ip: 0.0.0.0
  roles:
    - role: db
```

Применим конфигурацию

Применим **playbook.yml**, в котором вызывается наша роль к созданному хосту:

```
$ molecule converge
    ...
    TASK [db : Change mongo config file] ********************************************
    changed: [instance]

    RUNNING HANDLER [db : restart mongod] ***********************************************
    changed: [instance]

    PLAY RECAP *********************************************************************
    instance                   : ok=9    changed=6    unreachable=0    failed=0
```

Прогоним тесты

```
$ molecule verify
collected 2 items

    tests/test_default.py ..

====== 2 passed, 3 warnings in 4.10 seconds ======
Verifier completed successfully.
```

Самостоятельно

- Напишите тест к роли **db** для проверки того, что БД слушает по нужному порту (**27017**). Используйте для этого один из модулей Testinfra

- Используйте роли **db** и **app** в плейбуках **packer_db.yml** и **packer_app.yml** и убедитесь, что все работает как прежде (используйте теги для запуска только нужных тасков, теги указываются в шаблоне пакера*).
  * Шаблоны пакера находятся в директории **packer** и называются **app.json** и **db.json**.

- Билд пакера падает потому что не видит роли? Попробуйте решить эту проблему **самостоятельно**. Если не получается, фрагмент решения можно подсмотреть здесь.

Задание со *

- Вынести роль **db** в отдельный репозиторий: удалить роль из репозитория **infra** и сделать подключение роли через **requirements.yml** обоих окружений;

- Подключить TravisCI для созданного репозитория с ролью **db** для автоматического прогона тестов в **GCE** (нужно использовать соответсвующий драйвер в molecule). Пример, как это может выглядеть, можно посмотреть здесь.
  Примерные шаги по настройке TravisCI указаны в данном gist.
  У роли **должен** быть бейдж со статусом билда;

- Настроить оповещения о билде в слак чат, который использовали в предыдущих ДЗ;

Проверка ДЗ

- Результаты вашей работы находятся в ветке **ansible-4** вашего инфраструктурного репозитория.
- В README внесите описание того, что сделано.
- Создайте Pull Request к ветке master (**описание PR нужно заполнять**);
- В ревьюверы можно никого не добавлять;
- Добавьте "Labels" **Ansible** и **ansible-4** к вашему Pull Request;
- После того, как один из преподавателей сделает approve пул реквеста, ветку с ДЗ можно смерджить и закрыть PR.

- P.S. TravisCI проверяет наличие файлов ansible и наличие пустой строки в конце файлов. И верифицирует шаблоны пакера.