

# Разработка и тестирование Ansible ролей и плейбуков

# План

- Локальная разработка при помощи Vagrant
- Тестирование ролей при помощи Molecule и Testinfra
- Подключение Travis CI для автоматического прогона тестов

# Развитие проекта `infra`

В прошлых ДЗ вы создали инфраструктурный репозиторий `infra` на GitHub. Убедитесь что данный проект находится у вас на локальной машине.

**Если у вас нет репозитория `infra` на GitHub, выполните сначала предыдущие ДЗ.**

# Проект *infra* и проверка ДЗ

Создайте новую ветку в вашем локальном репозитории для выполнения данного ДЗ. Т.к. это чертовое задание, посвященное работе с **Ansible**, то ветку можно назвать **ansible-4**.

Проверка данного ДЗ будет производиться через Pull Request ветки с ДЗ к ветке мастер и добавление в Reviewers пользователей **Artemmkin** и **Nklya**.

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

# Локальная разработка с Vagrant

# Установка Vagrant

1. Установите VirtualBox на вашу локальную машину. VirtualBox - один из провайдеров, которым Vagrant может управлять для создания VMs. Мы будем использовать данный провайдер для локального запуска VM.
2. 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**.

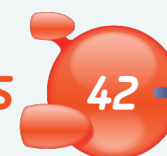
Мы создадим инфраструктуру, которую мы создавали до этого в GCE при помощи Terraform, на своей локальной машине, используя Vagrant.

# .gitignore

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

*infra/.gitignore*

```
# Vagrant  
.vagrant/  
*.logs
```





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

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
```

```
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:

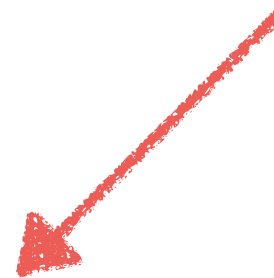
```
$ vagrant box list  
google/gce          (google, 0.1.0)  
ubuntu/precise64    (virtualbox, 20170427.0.0)  
ubuntu/trusty64     (virtualbox, 20170619.0.0)  
ubuntu/xenial64     (virtualbox, 20170922.0.0)
```

Проверим статус VMs:

VMs запущены

```
$ vagrant status  
Current machine states:
```

```
dbserver            running (virtualbox)  
appserver           running (virtualbox)
```



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 10.10.10.20
```

```
PING 10.10.10.20 (10.10.10.20) 56(84) bytes of data.
```

```
64 bytes from 10.10.10.20: icmp_seq=1 ttl=64 time=0.025 ms
```

```
64 bytes from 10.10.10.20: icmp_seq=2 ttl=64 time=0.021 ms
```

```
--- 10.10.10.20 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.

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



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 = "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 используем команду `provision`.

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



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/artemkin/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

# pre\_tasks

Pre\_tasks похожи на обычные задачи, однако выполняют перед задачами, определенными в роли. Воспользуемся этим для установки нужной для работы Ansible версии Python. Определим pre\_task в плейбуке `ansible/db.yml`, а также в `ansible/app.yml`.

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/db.yml (ссылка на [gist](#))*

---

```
- name: Configure MongoDB
  hosts: db
  gather_facts: false
  become: true
```

**pre\_tasks:**

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

**roles:**

```
- db
```



Аналогично измените плейбук `ansible/app.db`, т.к. для хоста `appserver` мы используем тот же Vagrant бокс, что и для `dbserver`



# 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/artemkin/hw133/ansible/site.retry
```

# Доработка роли db

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

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

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`.



*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 задачи 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
```

Видим, что провижининг выполнен 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 '^]'.
```



IP адрес dbserver



Порт для проверки

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 `racker_app.yml` плейбука, необходимую для настройки хоста приложения. Создадим новый файл для задач `ruby.yml` внутри роли app и скопируем в него задачи из плейбука `racker_app.yml`.

*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-dev
  - build-essential**tags:** ruby
- name: Install Ruby bundler  
gem:
  - name: bundler
  - state: present
  - user\_install: no**tags:** ruby

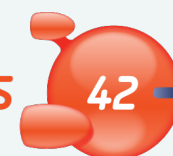
Вынесем настройки puma сервера также в отдельный файл для задач в рамках роли. Создадим файл `app/tasks/puma.yml` и скопируем в него задачи из `app/tasks/main.yml`, относящиеся к настройке Puma сервера и запуску приложения.

*`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 задачи 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



# Провижиным appserver

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

*ansibe/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 соответствует 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/artemkin/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, "failed": true, "msg": "Unable to find  
'db_config.j2' in expected paths."}
```

```
RUNNING HANDLER [app : reload puma] *****  
to retry, use: --limit @/Users/artemkin/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
```



Рассмотрим задачи, определенные в файле `puma.yml`. Первым делом, заменим модуль для копирования `unit` файла с `copy` на `template`, чтобы иметь возможность параметризовать `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

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 `puta.service` расширение `.j2`, чтобы обозначить данный файл как шаблон. Заменим в созданном шаблоне все упоминания `appuser` на переменную `deploy_user` (см. след слайд)



*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
```

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/deploy.yml, который применяется для группы хостов app и который нам тоже нужно не забыть параметризовать.

*ansible/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/Artemmkin/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.

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]
```

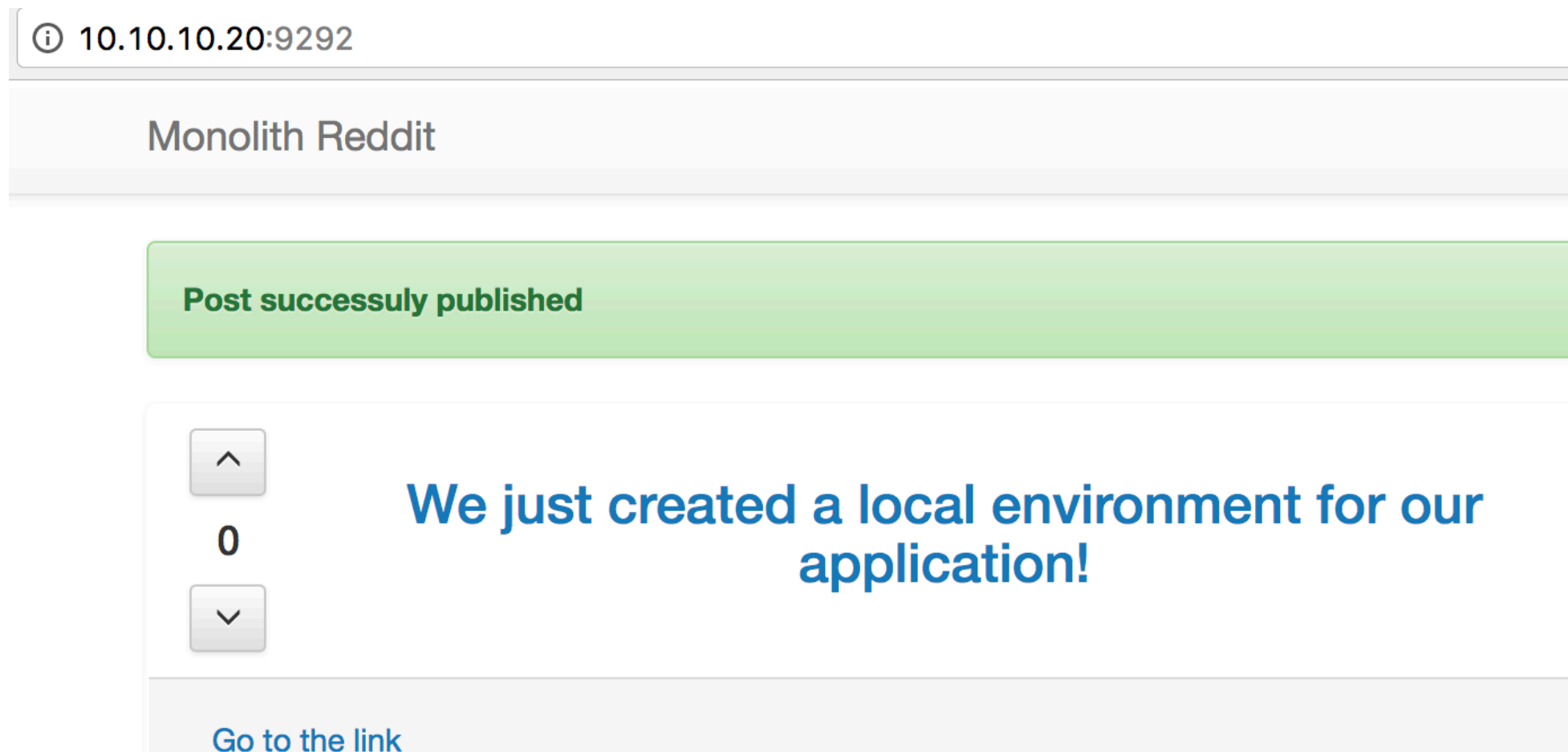
```
RUNNING HANDLER [restart puma] *****  
changed: [appserver]
```

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



# Проверим работу приложения

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





# Удалим машины

Выполните в директории ansible команду по удалению созданных машин.

```
$ vagrant destroy -f
```

# Тестирование роли

# Установка

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](#).

Создайте файл requirements.txt в директории ansible с содержимым указанным в данном [gist](#). Если у вас уже есть этот файл, то вам необходимо лишь поправить этот файл соответственным образом.

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

```
# проверим, что установились нужные версии
```

```
$ molecule --version  
molecule, version 2.1.0
```

```
$ ansible --version  
ansible 2.3.2.0
```

Мы устанавливаем версию ansible < 2.4, т.к. наблюдались проблемы при работе с molecule у последней версии ansible

# 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/artemkin/hw133/ansible/roles/db/molecule/default successfully.
```

Указываем Vagrant  
как драйвер для  
создания VMs



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 = File('/etc/mongod.conf')
```

```
    assert config_file.contains('bindIp: 0.0.0.0')
```

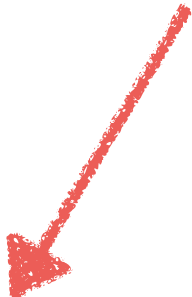
```
    assert config_file.is_file
```

# 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



Также можем при необходимости дебага подключиться по SSH внутрь VM:

```
$ molecule list
```

Instance Name	Driver Name	Provisioner Name	Created	Converged
-----	-----	-----	-----	-----
<b>instance</b>	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

Для применения роли нам необходимо вызвать нашу роль в плейбуке и указать в этом плейбуке к каким хостам применять данную конфигурацию. Molecule init генерирует плейбук, содержащий два сценария: один для установки python 2.X, а второй для применения нашей роли. Данный плейбук можно посмотреть по пути `db/molecule/default/playbook.yml`.

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

# playbook.yml

---

- name: Converge  
hosts: all  
gather\_facts: False  
tasks:
  - name: Install python for Ansible  
raw: test -e /usr/bin/python || (apt -y update && apt install -y python-minimal)  
become: True  
changed\_when: False
- name: Converge  
become: true  
hosts: all  
vars:
  - mongo\_bind\_ip: 0.0.0.0roles:
  - 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 racker\_db.yml и racker\_app.yml и убедитесь, что все работает как прежде (используйте теги для запуска только нужных задач).

# Задание со звездочкой

- Вынести роль db в отдельный репозиторий: удалить роль из репозитория infra и сделать подключение роли через requirements.txt
- Подключить Travis CI для созданного репозитория с ролью db для автоматического прогона тестов в **GCE** (нужно использовать соответствующий драйвер в Molecule). Пример, как это может выглядеть, можно посмотреть [здесь](#). Примерные шаги по настройке Travis CI указаны в данном [gist](#).

# Задание со звездочкой

- Настроить оповещения в слак чат, который использовали в ДЗ 2