Ansible: работа с ролями и окружениями

Проект infra и проверка ДЗ

Создайте новую ветку в вашем инфраструктурном репозитории для выполнения данного ДЗ. Т.к. это третье задание, посвященое работе с Ansible, то ветку назовите `ansible-3`.

Проверка данного ДЗ будет производиться через Pull Request ветки с ДЗ к ветке `master`. После того, как один из преподавателей сделает approve пул реквеста, ветку с ДЗ можно смерджить.

План

- Переносим созданные плейбуки в раздельные роли
- Описываем два окружения
- Используем коммьюнити роль *nginx*
- Используем Ansible Vault для наших окружений

Роли

Организация нашего конфигурационного кода стала выглядеть лучше, после того как мы ввели несколько плейбуков, но у нас все равно есть проблемы.

Во-первых, наши шаблоны и файлы хранятся в одних и тех же директориях для всех плейбуков. Как результат, становится сложно понять, что к чему относится, особенно если у нас возрастет количество плейбуков.

Во-вторых, т.к. переменных в плейбуке может быть большое количество, их не очень удобно определять в самом плейбуке. Нам определенно хотелось бы вынести их в отдельный файл.

Роли

Ну и самая большая проблема состоит в том, что текущую конфигурацию сложно версионировать и тяжело подстраивать для различных окружений.

Плейбуки не подходят как формат для распространения и переиспользования кода (нет версии, зависимостей и метаданных, зато много хардкода)

Роли

Роли представляют собой основной механизм группировки и переиспользования конфигурационного кода в Ansible.

Роли позволяют сгруппировать в единое целое описание конфигурации отдельных сервисов и компонент системы (таски, хендлеры, файлы, шаблоны, переменные). Роли можно затем переиспользовать при настройке окружений, тем самым избежав дублирования кода.

Ролями можно также делиться и брать у сообщества (community) в Ansible Galaxy.

Ansible Galaxy

Ansible Galaxy - это централизованное место, где хранится информация о ролях, созданных сообществом (community roles).

Ansible имеет специальную команду для работы с Galaxy. Получить справку по этой команде можно на сайте Galaxy или использовав команду:

```
$ ansible-galaxy -h
```

ansible-galaxy init

Также команда `ansible-galaxy init` позволяет нам создать структуру роли в соответсвии с принятым на Galaxy форматом.

Мы не будем делиться созданными нами ролями на Galaxy, однако используем эту команду для создания заготовки ролей.

В директории `ansible` создайте директорию `roles` и выполните в ней следующие команды для создания заготовки ролей для конфигурации нашего приложения и БД:

```
$ ansible-galaxy init app
$ ansible-galaxy init db
```

Посмотрим структуру созданных заготовок и выделим непонятные части...

Структура роли

```
$ ansible-galaxy init db
- db was created successfully

$ tree db

db
├── README.md
├── defaults            # <-- Директория для переменных по умолчанию
│   └── main.yml
├── handlers
│   └── main.yml
├── meta                # <-- Информация о роли, создателе и зависимостях
│   └── main.yml
├── tasks               # <-- Директория для тасков
│   └── main.yml
├── tests
│   ├── inventory
│   └── test.yml
└── vars                # <-- Директория для переменных, которые не должны
    └── main.yml        #     переопределяться пользователем

6 directories, 8 files
```

Роль для базы данных

Создадим роль для конфигурации MongoDB:

1. Создадим в папке `roles` папку `db`
2. Выполним команду `ansible-galaxy init` в папке `roles/db`
3. Скопируем секцию `tasks` в сценарии плейбука `ansible/db.yml` и вставим ее в файл в директории `tasks` роли `db`

Файл `ansible/roles/db/tasks/main.yml`:

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

Роль для базы данных

В директорию шаблоннов роли `ansble/roles/db/templates` скопируем шаблонизированный конфиг для MongoDB из директории `ansible/templates`.

Особенностью ролей также является, что модули `template` и `copy`, которые используются в тасках роли, будут по умолчанию проверять наличие шаблонов и файлов в директориях роли `templates` и `files` соответственно.

Поэтому укажем в таске только имя шаблона в качестве источника.

Роль для базы данных

Файл ansible/roles/db/tasks/main.yml:

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

Не забудем определить уже используемый хендлер в директории handlers роли.

Файл ansible/roles/db/handlers/main.yml:

```
# handlers file for db
- name: restart mongod
  service: name=mongod state=restarted
```

Роль для базы данных

Определим используемые в шаблоне переменные в секции переменных по умолчанию (файл ansible/roles/db/defaults/main.yml):

```
# defaults file for db
mongo_port: 27017
mongo_bind_ip: 127.0.0.1
```

Роль для приложения

Создадим роль для управления конфигурацией инстанса приложения:

1. Создадим в папке `roles` папку `app`
2. Выполним команду `ansible-galaxy init` в папке `roles/app`
3. Скопируем секцию `tasks` в сценарии плейбука `ansible/app.yml` и вставим ее в файл для тасков роли `app`.

Не забудем при этом заменить `src` в модулях `copy` и `template` для указания только имени файлов.

Роль для приложения

Пример файла `ansible/roles/app/tasks/main.yml`:

```
# tasks file for app
- 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
```

Роль для приложения

1. Создадим директорию для шаблонов `templates` и директорию для файлов `files` в директории роли `ansible/roles/app` (если не использовали ранее команду `ansible-galaxy init`).
2. Скопируйте файл `db_config.j2` из директории `ansible/templates` в директорию `ansible/roles/app/templates/`
3. Файл `ansible/files/puma.service` скопируем в `ansible/roles/app/files/`.

Роль для приложения

1. Опишем используемый хендлер в соответствующей директории роли *app*

   Пример `ansible/roles/app/handlers/main.yml`:

```
# handlers file for app
- name: reload puma
  systemd: name=puma state=restarted
```

2. Не забудем также определить переменную по умолчанию для адреса подключения к MongoDB в файле `ansible/roles/app/defaults/main.yml`:

```
# defaults file for app
db_host: 127.0.0.1
```

Вызов ролей

Используем роли в созданных ранее плейбуках:

Удалим определение тасков и хендлеров в плейбуке `ansible/app.yml` и заменим на вызов роли:

```
- name: Configure App
  hosts: app
  become: true

  vars:
   db_host: 10.132.0.2

  roles:
    - app
```

Вызов ролей

Аналогичную операцию проделаем с файлом `ansible/db.yml`:

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

  vars:
    mongo_bind_ip: 0.0.0.0

  roles:
    - db
```

Проверка ролей

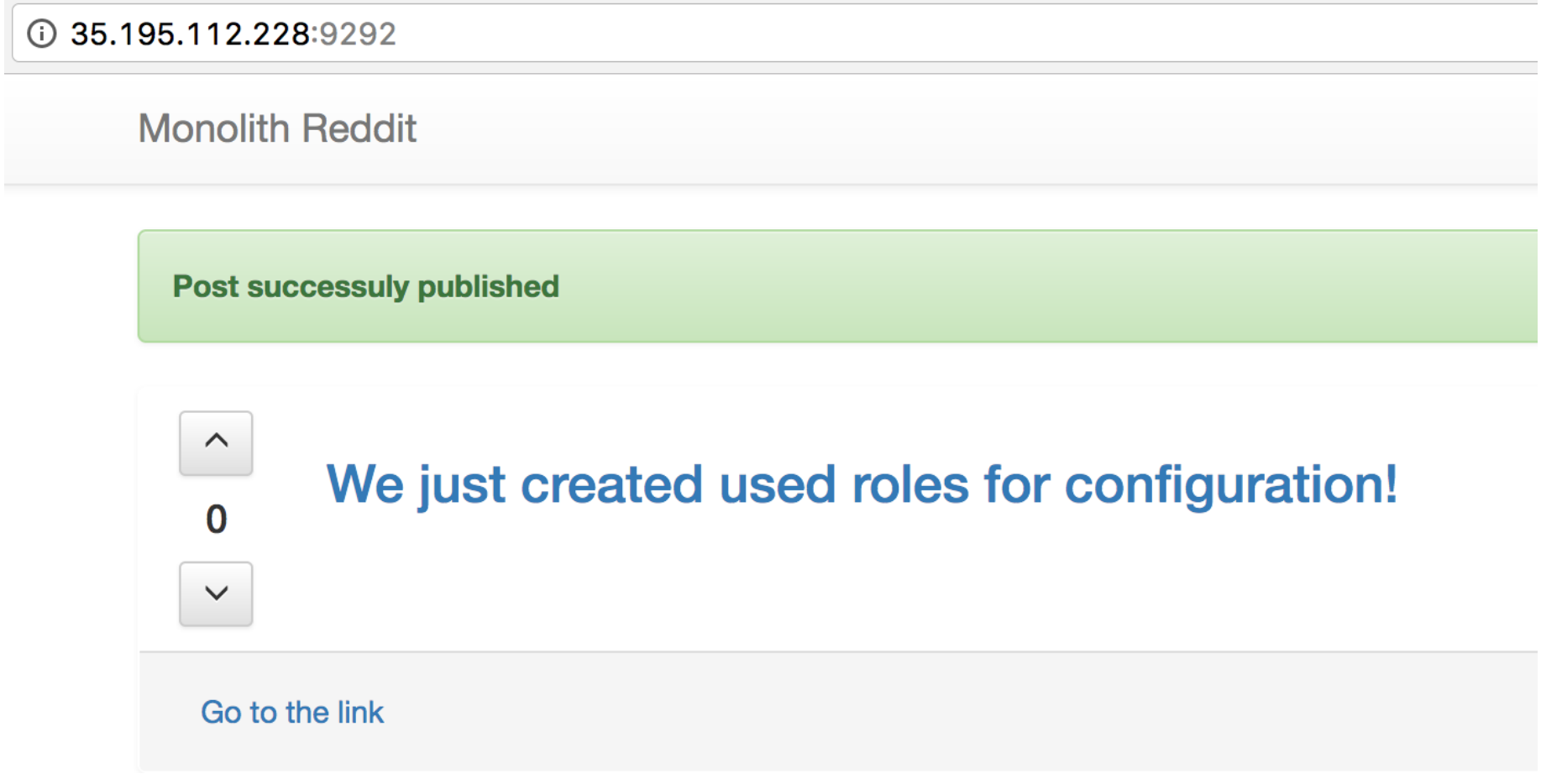
Для проверки роли пересоздадим инфраструктуру окружения *stage*, используя команды:

```
$ terraform destroy
$ terraform apply -auto-approve=false
```

Перед проверкой не забудьте изменить внешние IP адреса инстансов в инвентори файле `ansible/inventory` и переменную `db_host` в плейбуке `app.yml`:

```
$ ansible-playbook site.yml --check
$ ansible-playbook site.yml
```

Проверим работу приложения



Окружения

Обычно инфраструктура состоит из нескольких окружений. Эти окружения могут иметь небольшие отличия в настройках инфраструктуры и конфигурации управляемых хостов.

С помощью Terraform мы уже описали инфраструктуру для тестового и боевого окружения (production).

Теперь используем Ansible для управления каждым из них...

Создадим директорию `environments` в директории `ansible` для определения настроек окружения. В директории `ansible/environments` создадим две директории для наших окружений `stage` и `prod`.

Inventory File

Так как мы управляем разными хостами на разных окружениях, то нам необходим свой инвентори-файл для каждого из окружений.

Скопируем инвентори файл `ansible/inventory` в каждую из директорий окружения `environtents/prod` и `environments/stage`.

Сам файл `ansible/inventory` при этом удалим.

Окружение по умолчанию

Теперь, когда у нас два инвентори файла, то чтобы управлять хостами окружения нам необходимо явно передавать команде, какой инвентори мы хотим использовать.

Например, чтобы задеплоить приложение на *prod* окружении мы должны теперь написать:

```
$ ansible-playbook -i environments/prod/inventory deploy.yml
```

Таким образом сразу видно, с каким окружением мы работаем. В нашем случае, мы также определим окружение по умолчанию (*stage*), что упростит команду для тестового окружения.

Окружение по умолчанию

Определим окружение по умолчанию в конфиге Ansible (файл `ansible/ansible.cfg`):

```
[defaults]
inventory = ./environments/stage/inventory  # Inventory по-умолчанию задается здесь
remote_user = appuser
private_key_file = ~/.ssh/appuser
host_key_checking = False
```

Переменные групп хостов

Параметризация конфигурации ролей за счет переменных дает нам возможность изменять настройки конфигурации, задавая нужные значения переменных.

Ansible позволяет задавать переменные для групп хостов, определенных в инвентори файле. Воспользуемся этим для управления настройками окружений.

Переменные групп хостов

Директория `group_vars`, созданная в директории плейбука или инвентори файла, позволяет создавать файлы (имена, которых должны соответствовать названиям групп в инвентори файле) для определения переменных для группы хостов.

Создадим директорию `group_vars` в директориях наших окружений `environments/prod` и `environments/stage`.

Конфигурация Stage

Зададим настройки окружения stage, используя групповые переменные:

1. Создадим файлы `stage/group_vars/app` для определения переменных для группы хостов `app`, описанных в инвентори файле `stage/inventory`
2. Скопируем в этот файл переменные, определенные в плейбуке `ansible/app.yml`.
3. Также **удалим** определение переменных из самого плейбука `ansible/app.yml`.

Пример файла `ansible/environments/stage/group_vars/app`:

```
db_host: 10.132.0.2
```

Конфигурация Stage

Аналогичным образом определим переменные для группы хостов БД на окружении *stage*:

1. Создадим файл `stage/group_vars/db` и скопируем в него содержимое переменные из плейбука `ansible/db.yml`
2. Секцию определения переменных из самого плейбука `ansible/db.yml` удалим.

Пример файла `ansible/environments/stage/group_vars/db`:

```
mongo_bind_ip: 0.0.0.0
```

Конфигурация Stage

Как мы помним, по умолчанию Ansible создает группу *all* для всех хостов указанных в инвентори файле. Создадим файл с переменными для этой группы. Таким образом переменные в этом файле будут доступны всем хостам окружения.

Создайте файл `ansible/environments/stage/group_vars/all` со следующим содержимым:

```
env: stage
```

Конфигурация Prod

Конфигурация окружения *prod* будет идентичной, за исключением переменной `env`, определенной для группы *all*.

1. Для настройки окружения *prod* скопируйте файлы `app`, `db`, `all` из директории `stage/group_vars` в директорию `prod/group_vars`.
2. В файле `prod/group_vars/all` измените значение `env` переменной на **prod**:

Должно получиться так:

```
env: prod
```

Вывод информации об окружении

Для хостов из каждого окружения мы определили переменную env, которая содержит название окружения.

Теперь настроим вывод информации об окружении, с которым мы работаем, при применении плейбуков.

Определим переменную по умолчанию env в используемых ролях...

Вывод информации об окружении

Для роли *app* в файле `ansible/roles/app/defaults/main.yml`:

```
# defaults file for app
db_host: 127.0.0.1
env: local
```

Для роли *db* в файле `ansible/roles/db/defaults/main.yml`:

```
# defaults file for db
mongo_port: 27017
mongo_bind_ip: 127.0.0.1
env: local
```

Вывод информации об окружении

Будем выводить информацию о том, в каком окружении находится конфигурируемый хост. Воспользуемся модулем debug для вывода значения переменной. Добавим следующий таск в начало наших ролей.

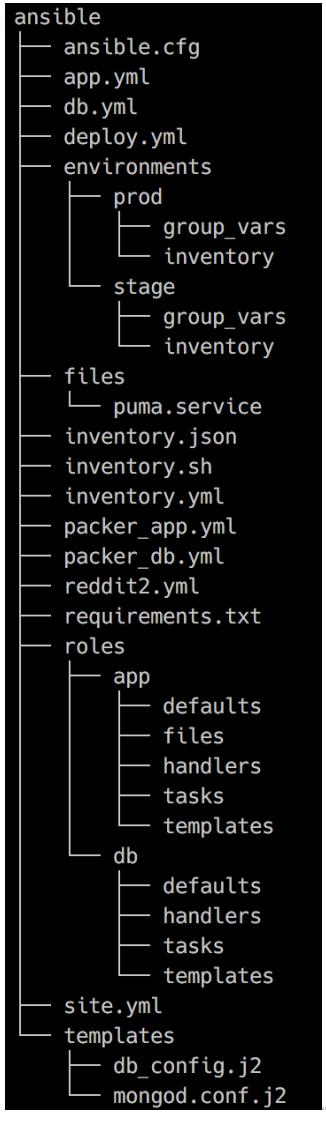
Для роли *app* (файл ansible/roles/app/tasks/main.yml):

```
# tasks file for app
- name: Show info about the env this host belongs to
  debug:
    msg: "This host is in {{ env }} environment!!!"
```

Добавим такой же таск в роль *db* (файл ansible/roles/db/tasks/main.yml):

```
# tasks file for db
- name: Show info about the env this host belongs to
  debug: msg="This host is in {{ env }} environment!!!"
```

Организуем плейбуки



К данному моменту директория `ansible` нашего репозитория выглядит примерно следующим образом.

В корне директории довольно много файлов. Часть из них это остатки от предыдущих ДЗ, а часть - это плейбуки, которые мы используем для вызова ролей.

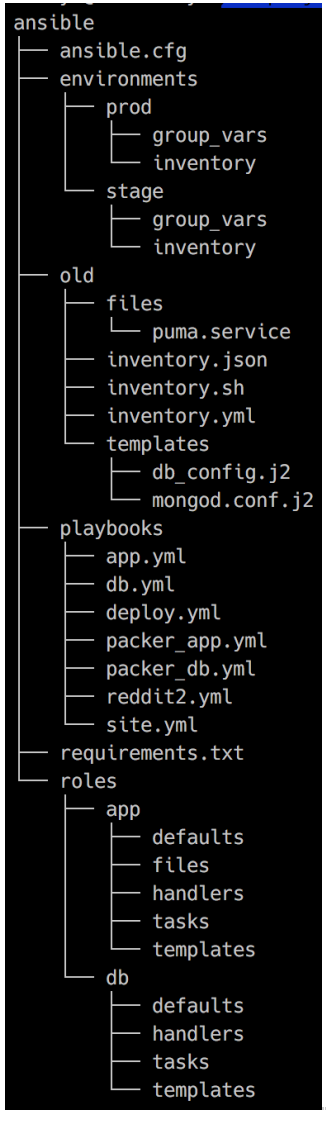
Организуем плейбуки

Перенесем все плейбуки в отдельную директорию согласно best practices:

- Создадим директорию `ansible/playbooks` и перенесем туда все наши плейбуки, в том числе из прошлого ДЗ.
- В директории `ansible` у нас остались еще файлы из прошлых ДЗ, которые нам не особо нужны. Создадим директорию `ansible/old` и перенесем туда все, что не относится к текущей конфигурации.

> После этого стоит проверить шаблоны `app` и `db`, используемые Packer.

Организуем плейбуки



В итоге получаем примерно такую, как на картинке, красивую структуру директорий.

В корне папки `ansible` из файлов остаются только `ansible.cfg` и `requirements.txt`

Улучшим файл ansible.cfg

Заодно улучшим наш ansible.cfg. Для этого приведем его к такому виду:

```
[defaults]
inventory = ./environments/stage/inventory
remote_user = appuser
private_key_file = ~/.ssh/appuser
# Отключим проверку SSH Host-keys (поскольку они всегда разные для новых инстансов)
host_key_checking = False
# Отключим создание *.retry-файлов (они нечасто нужны, но мешаются под руками)
retry_files_enabled = False
# # Явно укажем расположение ролей (можно задать несколько путей через ; )
roles_path = ./roles

[diff]
# Включим обязательный вывод diff при наличии изменений и вывод 5 строк контекста
always = True
context = 5
```

Проверка работы с окружениями

Для проверки пересоздадим инфраструктуру окружения *stage*, используя команды:

```
$ terraform destroy
$ terraform apply -auto-approve=false
```

Теперь запустим Ansible...

Перед проверкой не забудьте изменить внешние IP-адреса инстансов в инвентори файле `ansible/environments/stage/inventory` и переменную `db_host` в `stage/group_vars/app`

Настройка stage окружения

Если все сделано правильно, то получим примерно такой вывод команды `ansible-playbook`:

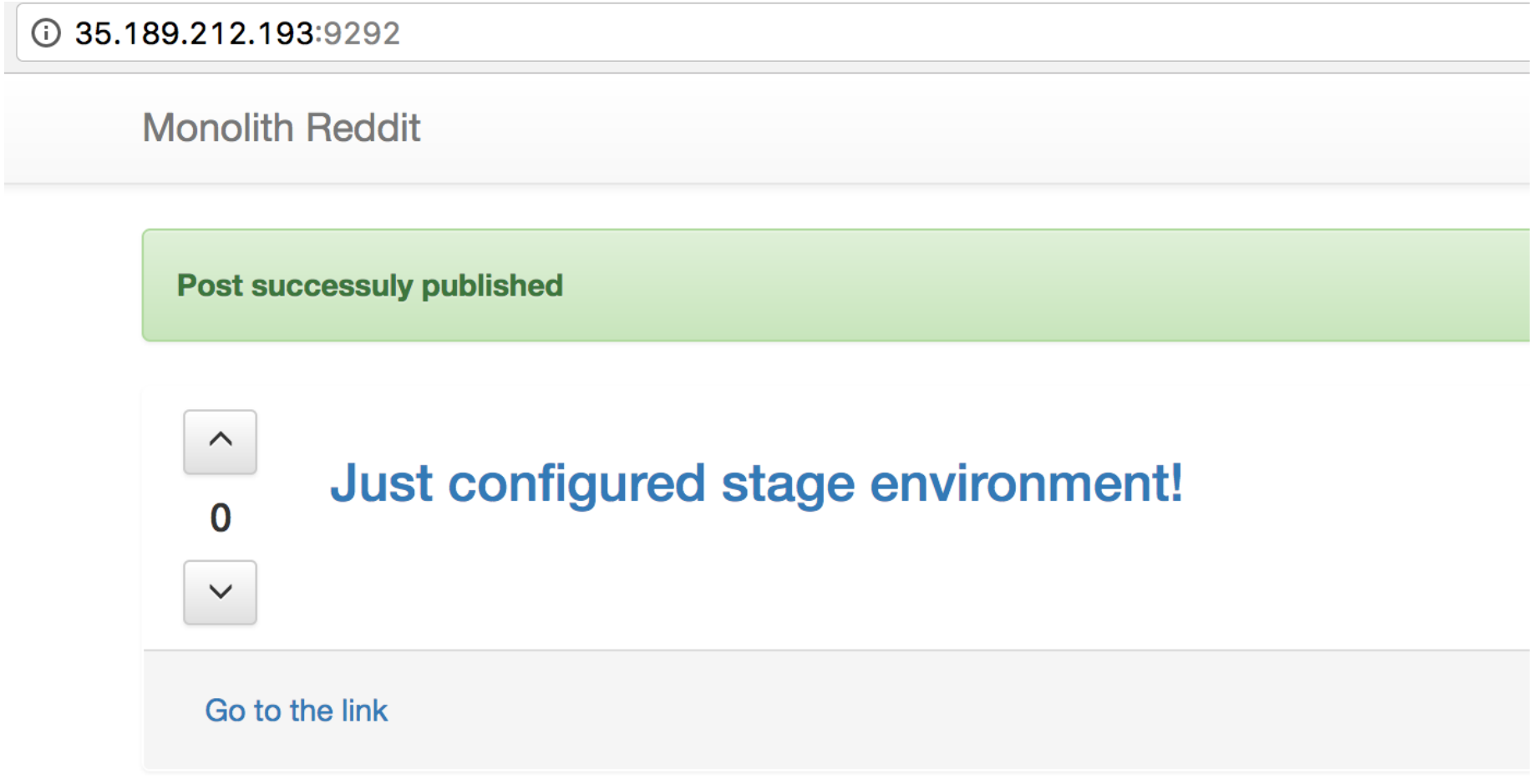
```
$ ansible-playbook playbooks/site.yml --check
$ ansible-playbook playbooks/site.yml

PLAY [Configure MongoDB]
********************************************************************************

TASK [Gathering Facts]
********************************************************************************
ok: [dbserver]

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

Проверим работу приложения



Настройка Prod окружения

Для проверки настройки *prod* окружения сначала удалим инфраструктуру окружения *stage*. Затем поднимем инфраструктуру для *prod* окружения.

> Перед проверкой не забудьте изменить внешние IP-адреса инстансов в инвентори файле `ansible/environments/prod/inventory` и переменную `db_host` в `prod/group_vars/app`

Настройка Prod окружения

Если все сделано правильно, то получим примерно такой вывод команды `ansible-playbook`:

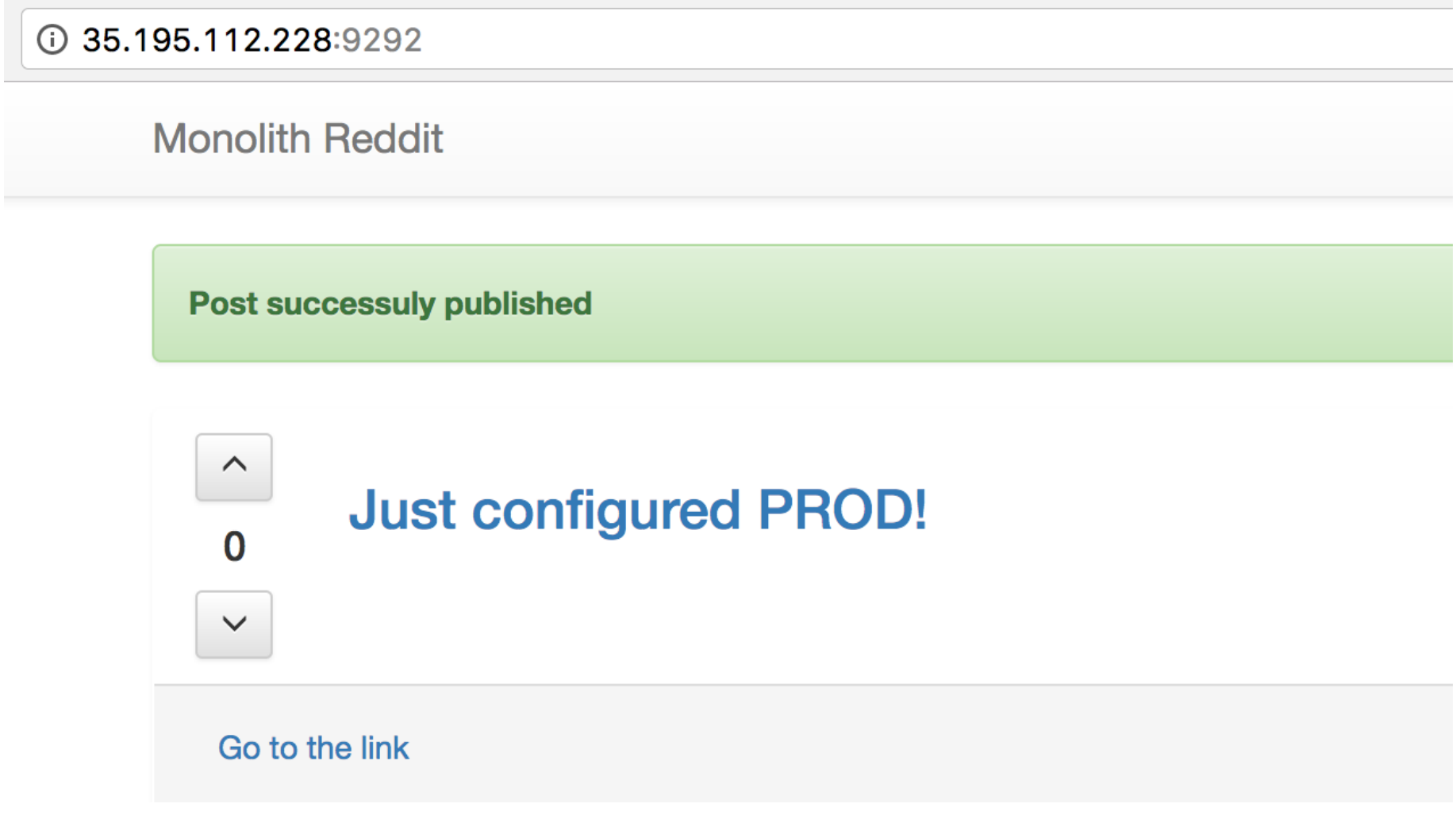
```
$ ansible-playbook -i environments/prod/inventory playbooks/site.yml --check
$ ansible-playbook -i environments/prod/inventory playbooks/site.yml

PLAY [Configure MongoDB]
********************************************************************************

TASK [Gathering Facts]
********************************************************************************
ok: [dbserver]

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

Проверим работу приложения



Работа с Community-ролями

Как мы уже говорили, коммьюнити-роли в основном находятся на портале Ansible Galaxy и работа с ними производится с помощью утилиты `ansible-galaxy` и файла `requirements.yml`

Используем роль `jdauphant.nginx` и настроим обратное проксирование для нашего приложения с помощью **nginx**.

Работа с Community-ролями

Хорошей практикой является разделение зависимостей ролей (`requirements.yml`) по окружениям.

1. Создадим файлы `environments/stage/requirements.yml` и `environments/prod/requirements.yml`
2. Добавим в них запись вида:

```
- src: jdauphant.nginx
  version: v2.21.1
```

3. Установим роль:

```
ansible-galaxy install -r environments/stage/requirements.yml
```

4. Комьюнити-роли *не стоит коммитить в свой репозиторий*, для этого добавим в `.gitignore` запись: `jdauphant.nginx`

Работа с Community-ролями

Рассмотрим документацию роли.

Как мы видим, для минимальной настройки проксирования необходимо добавить следующие переменные:

```
nginx_sites:
  default:
    - listen 80
    - server_name "reddit"
    - location / {
        proxy_pass http://127.0.0.1:порт_приложения;
      }
```

Добавим эти переменные в `stage/group_vars/app` и `prod/group_vars/app`

Самостоятельное задание

- Добавьте в конфигурацию Terraform открытие 80 порта для инстанса приложения
- Добавьте вызов роли `jdauphant.nginx` в плейбук `app.yml`
- Примените плейбук `site.yml` для окружения *stage* и проверьте, что приложение теперь доступно на 80 порту

Работа с Ansible Vault

Для безопасной работы с приватными данными (пароли, приватные ключи и т.д.) используется механизм Ansible Vault. Данные сохраняются в зашифрованных файлах, которые при выполнении плейбука автоматически расшифровываются. Таким образом, приватные данные можно хранить в системе контроля версий.

Для шифрования используется мастер-пароль (aka **vault key**). Его нужно передавать команде `ansible-playbook` при запуске, либо указать файл с ключом в `ansible.cfg`. **Не допускайте хранения этого ключ-файла в Git!** Используйте для разных окружений разный **vault key**.

Работа с Ansible Vault

Подготовим плейбук для создания пользователей, пароль пользователей будем хранить в зашифрованном виде в файле `credentials.yml`:

1. Создайте файл `vault.key` со произвольной строкой ключа
2. Изменим файл `ansible.cfg`, добавим опцию `vault_password_file` в секцию `[defaults]`

```
[defaults]
...
vault_password_file = vault.key
```

! Обязательно добавьте в `.gitignore` файл `vault.key`. А еще лучше - храните его out-of-tree, аналогично ключам SSH (например, в папке `~/.ansible/vault.key`)

Работа с Ansible Vault

Добавим плейбук для создания пользователей - файл `ansible/playbooks/users.yml` (ссылка на gist):

```
- name: Create users
  hosts: all
  become: true

  vars_files:
    - "{{ inventory_dir }}/credentials.yml"

  tasks:
    - name: create users
      user:
        name: "{{ item.key }}"
        password: "{{ item.value.password|password_hash('sha512',
65534|random(seed=inventory_hostname)|string) }}"
        groups: "{{ item.value.groups | default(omit) }}"
      with_dict: "{{ credentials.users }}"
```

Работа с Ansible Vault

Создадим файл с данными пользователей для каждого окружения (ссылка на gist)

Файл для *prod* (`ansible/environments/prod/credentials.yml`):

```
credentials:
  users:
    admin:
      password: admin123
      groups: sudo
```

Файл для *stage* (`ansible/environments/stage/credentials.yml`):

```
credentials:
  users:
    admin:
      password: qwerty123
      groups: sudo
    qauser:
      password: test123
```

Работа с Ansible Vault

1. Зашифруем файлы используя `vault.key` (используем одинаковый для всех окружений):

```
$ ansible-vault encrypt environments/prod/credentials.yml
$ ansible-vault encrypt environments/stage/credentials.yml
```

2. Проверьте содержимое файлов, убедитесь что они зашифрованы
3. Добавьте вызов плейбука в файл `site.yml` и выполните его для `stage` окружения:

```
$ ansible-playbook site.yml –check
$ ansible-playbook site.yml
```

Работа с Ansible Vault

Теперь проверьте, что пользователи созданы в системе (внимание, по умолчанию, вход по паролю на инстансах GCE отключен)

P.S.

Для редактирования переменных нужно использовать команду `ansible-vault edit <file>`

А для расшифровки: `ansible-vault decrypt <file>`

Задание со ⭐: Работа с динамическим инвентори

В прошлом ДЗ было задание со ⭐ про работу с динамическим инвентори.

Настройте использование динамического инвентори для окружений `stage` и `prod`.

В коде Ansible это должно быть закоммичено.

> ! Если вы используете `gce.py`, убедитесь в том, что вы НЕ закоммитите ключи сервисного пользователя (файл *.json)!

Задание с ⭐⭐: Настройка TravisCI

В предыдущих ДЗ вы уже использовали TravisCI. Теперь настройте его для контроля состояния вашего инфраструктурного репозитория.

Необходимо, чтобы для коммитов в **master** и **PR** выполнялись как минимум эти действия:

- `packer validate` для всех шаблонов
- `terraform validate` и `tflint` для окружений `stage` и `prod`
- `ansible-lint` для плейбуков Ansible
- в `README.md` добавлен бейдж с статусом билда

Задание с ⭐⭐: Настройка TravisCI

- Перед выполнением сборки TravisCI может копировать `.example`-файлы в нормальные и создавать заглушки требуемых файлов.
- Из секции `before_install` нельзя удалять секцию наших тестов `otus-homeworks`.

> Для отладки прохождения тестов советуем воспользоваться `trytravis`.

Проверка ДЗ

- Результаты вашей работы находятся в ветке **ansible-3** 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-3` 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, наличие пустой строки в конце файлов и верифицирует шаблоны пакера.