

Системой контроля версий выбрали git, разместили код на github.

Первая задача размещение и запуск проекта из репозитория на серверах.

Чтобы процесс настройки не повторять в случае переезда или аварии, решил использовать Ansible для описания процессов конфигурации серверов. Имел с ним дело в прошлом командном проекте.

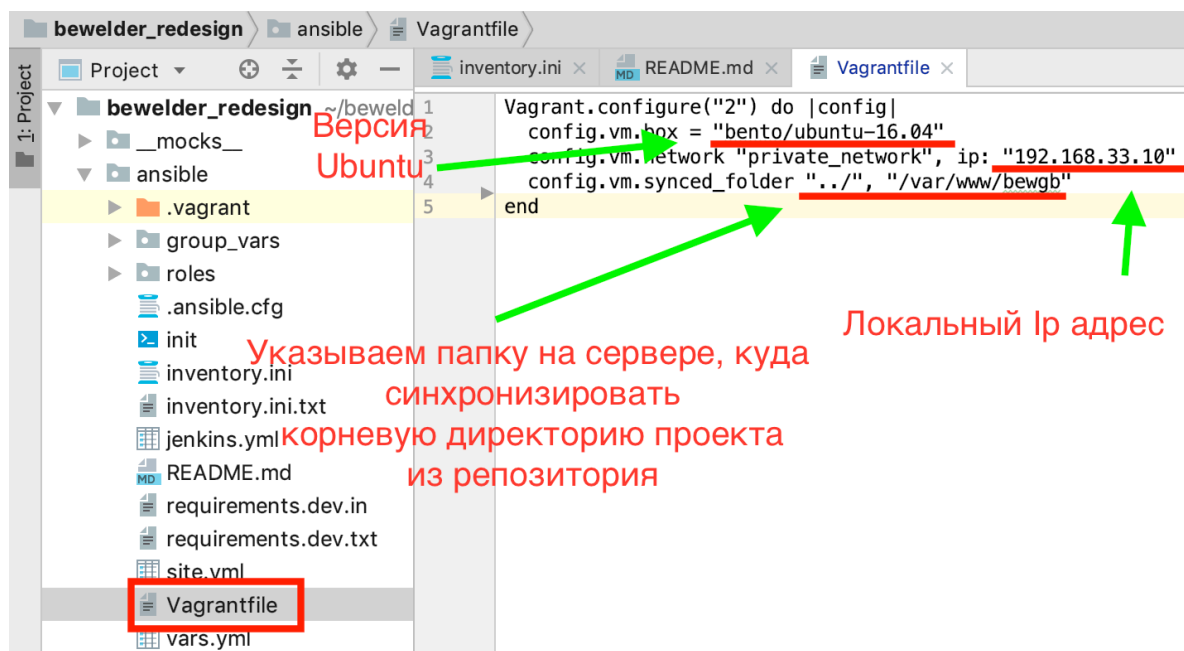
А для тестирования этих сборок на локальной машине, решил использовать Vagrant, так как тоже имею с ним дело для быстрого создания виртуальных серверов при разработке, как правило Linux Ubuntu.

В качестве системы интеграции был выбран Jenkins. Раньше с ним не работал, выбрал по критерию популярности и простоты использования.

В целом с системным администрированием знаком лишь в теории, читал статьи книги, но участвовал в прошлом командном проекте в этой же роли. Чувствую пробел в этой области, и решил воспользоваться случаем и попрактиковаться.

Vagrant

Настройка состоит лишь из одного файла



Запустив в консоли команду

`vagrant up`

Собирается локальный сервер Ubuntu 16

Доступ

```
ssh vagrant@192.168.33.10 -p vagrant
```

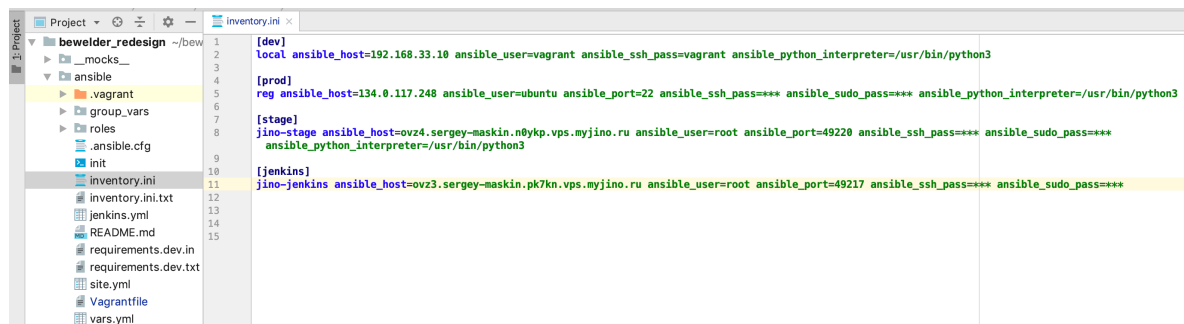
Либо

```
vagrant ssh
```

Локальный сервер с Ubuntu готов. Следующий шаг приступаем к его настройке.

Ansible

Позволяет удаленно выполнять различные задачи на сервере.
Первый файл который нужно знать inventory.ini:

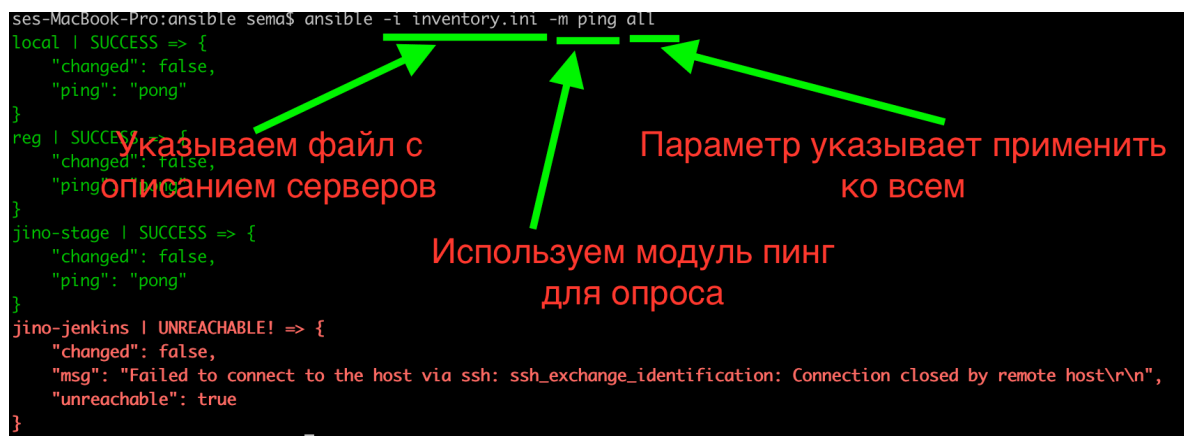


В квадратных скобках группы серверов, у нас dev, prod, stage, jenkins. В нашем случае в каждой группе по одной машине. В начале название local, reg(по имени тостера), jino-stage, jino-jenkins. Далее имя домена или айпи, пароли, порты, ключи и так далее, много различных параметров. Это необходимо Ansible для подключения по ssh.

Пример простой команды:

```
ansible -i inventory.ini -m ping all
```

```
ses-MacBook-Pro:ansible sema$ ansible -i inventory.ini -m ping all
local | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
reg | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
jino-stage | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
jino-jenkins | UNREACHABLE! => {
  "changed": false,
  "msg": "Failed to connect to the host via ssh: ssh_exchange_identification: Connection closed by remote host\r\n",
  "unreachable": true
}
```



В результате Ansible пингует наши сервера. Имена и доступы указаны в файле inventory.ini. Он добавлен в .gitignore и храниться только на локальной машине, так как там напрямую написаны пароли. Ansible позволяет шифровать файлы с помощью утилиты ansible-vault. Так же можно отказаться от паролей и сделать все доступы по ssh ключам. Это тему здесь не рассматриваю.

Имеется множество модулей. Документация тут https://docs.ansible.com/ansible/latest/modules/modules_by_category.html

Следующая важная утилита ansible-playbook. Позволяет выполнять уже наборы задач.

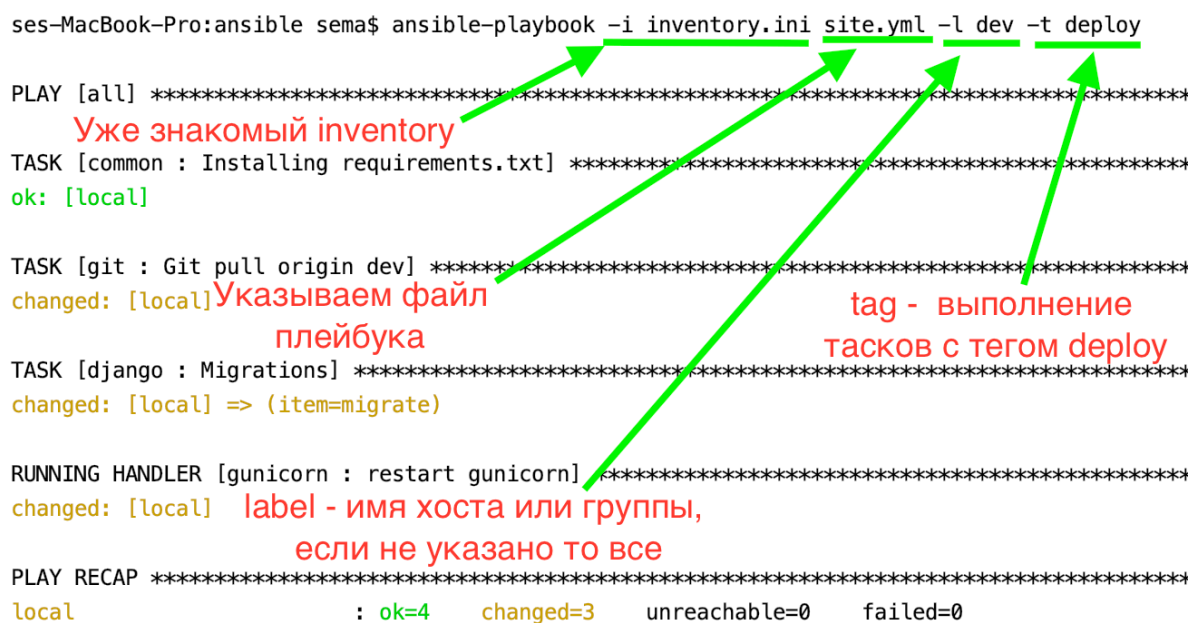
Пример простой команды:

```
ansible-playbook -i inventory.ini playbook.yml -l dev -t deploy
```

```
ses-MacBook-Pro:ansible sema$ ansible-playbook -i inventory.ini site.yml -l dev -t deploy
PLAY [all] *****
TASK [common : Installing requirements.txt] *****
ok: [local]

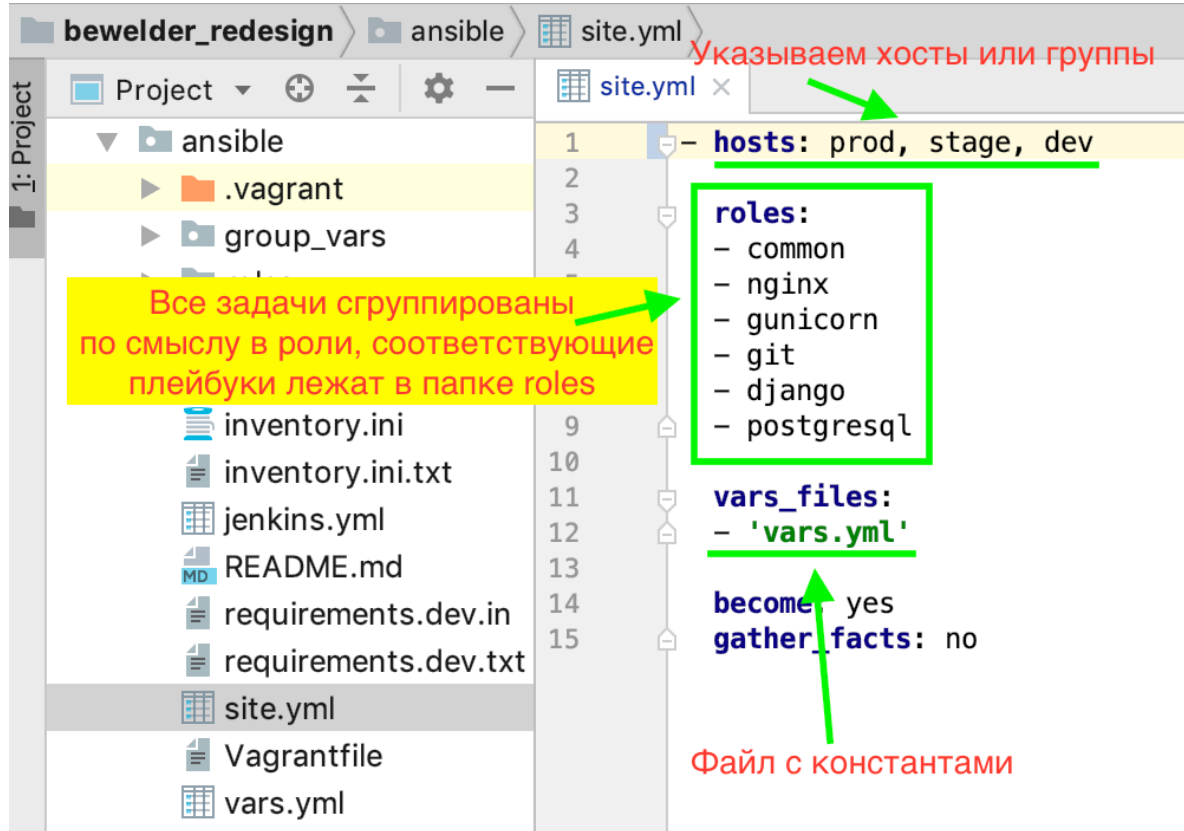
TASK [git : Git pull origin dev] *****
changed: [local]
TASK [django : Migrations] *****
changed: [local] => (item=migrate)

RUNNING HANDLER [unicorn : restart unicorn] *****
changed: [local]
PLAY RECAP *****
local                : ok=4    changed=3    unreachable=0    failed=0
```

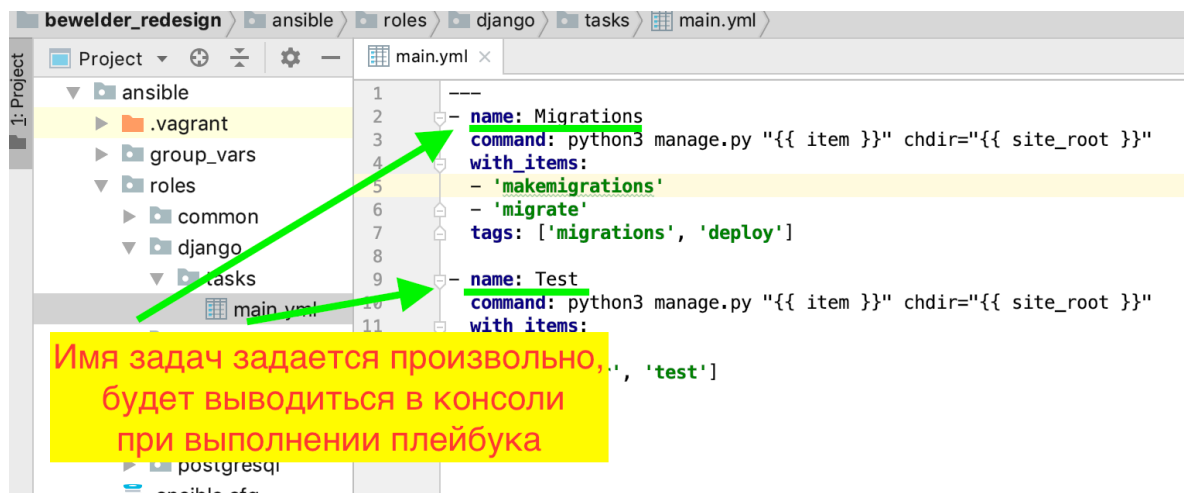


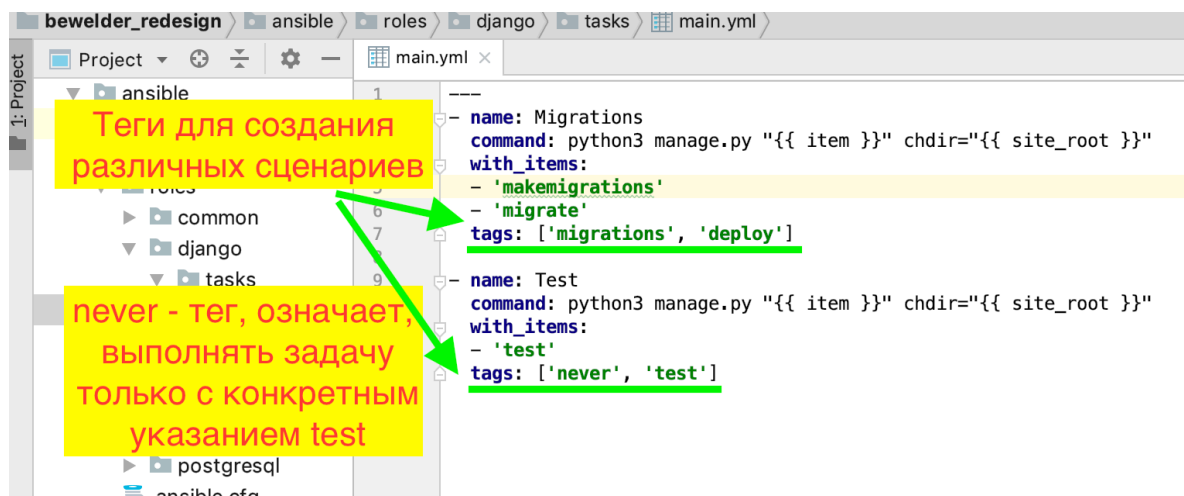
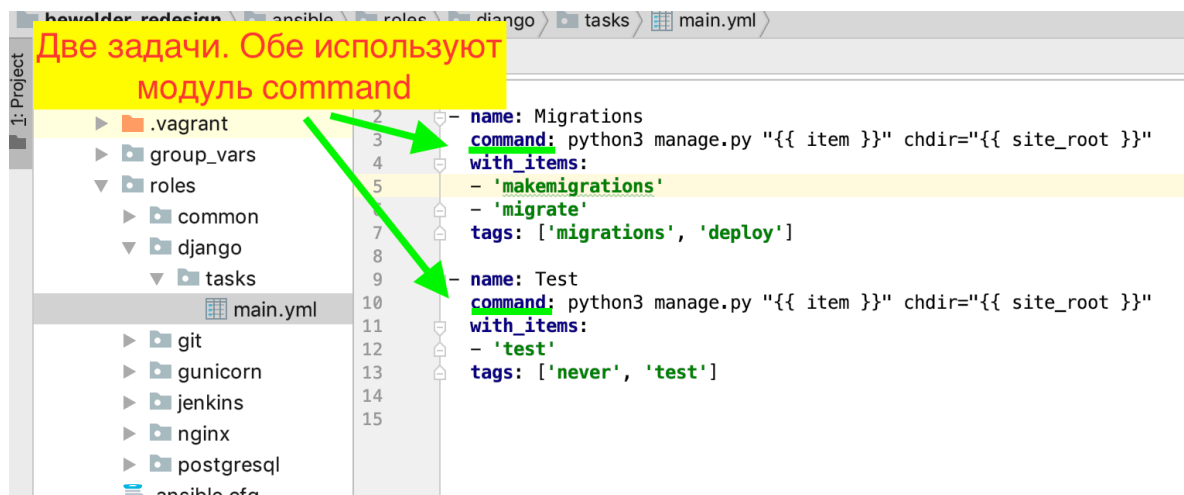
В коносли видим, успешное выполнение задач. Устанавливаются пакеты pip из requirements, обновляется ветка репозитория, выполняются миграции данных, и перезагружается gunicorn.

Следующий файл который нужно знать site.yml



Пример роли django:





Таким образом получаем такие полезные команды как например:

```
ansible-playbook -i inventory.ini site.yml -l dev
```

Для полной сборки дев версии проекта из ветки dev на наш локальный сервер, собранный с помощью Vagrant на первом шаге.

Или, например:

```
ansible-playbook -i inventory.ini site.yml -l prod -t deploy
```

Для обновления продакшн сервера из ветки master.

Или, например:

```
ansible-playbook -i inventory.ini site.yml -l stage -t 'deploy, test'
```

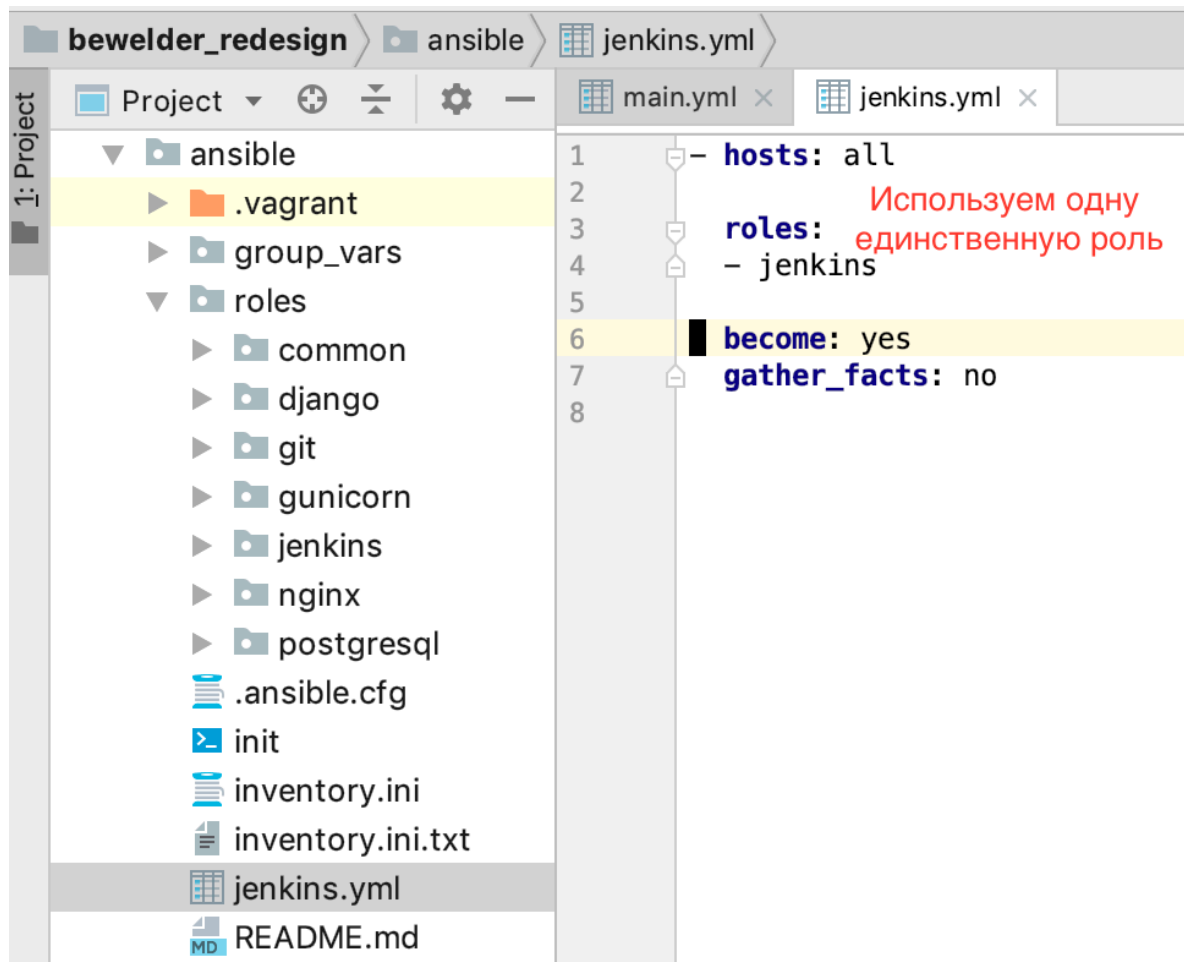
Для обновления stage сервера из ветки dev и ее тестирования.

Таким образом, получили все необходимые сервера. Осталось перенести управление на отдельный сервер с доступом через веб браузер. Для этого воспользуемся jenkins.

Jenkins

Используется для непрерывной интеграции программного обеспечения. Для сборки самого сервера используем скрипт.

```
ansible-playbook -i inventory.ini jenkins.yml
```



При успешной установке на сервере станет доступна веб версия Jenkins.



Welcome to Jenkins!

Sign in

☐

Keep me signed in

Jenkins предоставляет возможность создавать различные задачи, используя множество плагинов. Документация <https://plugins.jenkins.io/>

Jenkins умеет работать с плейбуками Ansible. Установил соответствующий плагин.

Пример создания задачи:

[Создать Item](#)

Жмем создать

[Пользователи](#)[История сборок](#)[Настроить Jenkins](#)[My Views](#)[Lockable Resources](#)[Credentials](#)[New View](#)

Очередь сборок



Очередь сборок пуста

Состояние сборщиков



1 В ожидании

2 В ожидании

General

Расширенные настройки проекта

Управление исходным кодом

Триггеры сборки

Матрица конфигураций

Среда сборки

Сборка

Послесборочные операции

Описание

[Plain text] [Предпросмотр](#)

☐ GitHub project

☐ This build requires lockable resources

☐ Throttle builds

☐ Удалять устаревшие сборки

☐ Это - параметризованная сборка

☐ Приостановить сборки

☐ Разрешить параллельный запуск задачи

?

?

?

?

?

Расширенные настройки проекта

Исходный код будем брать из гит, понадобятся как раз ansible конфиги

Расширенные...

Управление исходным кодом

Нет

☒ Git

Repositories

Repository URL

https://github.com/testpass1982/bewelder_redesign.git

Credentials

- none -

+

 Add

Расширенные...

Add Repository

Branches to build

Branch Specifier (blank for 'any')

*/deploy

Add Branch

Просмотрщик репозитория

(Автоматически)

Additional Behaviours

Добавить

Subversion

?

Триггеры сборки

☐ Trigger builds remotely (e.g., from scripts)

☐ Build after other projects are built

☐ Запускать периодически

☐ GitHub hook trigger for GITScm polling

☐ Опрашивать SCM об изменениях

?

?

?

?

?

Сборка

Invoke Ansible Playbook

Playbook path:

Inventory:
☐ Do not specify Inventory
☐ File or host list
☒ Inline content
Dynamic inventory: ☐
Content:

```
[stage]
jino-stage ansible_host=ovz4.sergey-maskin.n0ykp.vps.myjino.ru ansible_user=root ansible_port=49220
ansible_ssh_pass=          ansible_sudo_pass=          ansible_python_interpreter=/usr/bin/python3
```

Host subset:

Credentials:

Vault Credentials:

☐ become
☒ sudo
sudo user:

Tags to run:

Tags to skip:

Task to start at:

Number of parallel processes:

Disable the host SSH key check: ☒

Unbuffered stdout: ☒

Colorized stdout: ☐

Extra Variables:

Additional parameters:

Послесборочные операции

Создаем таким образом необходимые задачи:

Jenkins

Задачи продакшна

Задачи стейджа

Name	Последний успех	Последняя неудача	Последняя продолжительность
prod-build	Н/Д	Н/Д	Н/Д
prod-deploy	3 часа 22 минут - #4	Н/Д	1 минута 16 секунд
stage-build	20 часов - #38	Н/Д	24 минут
stage-deploy	5 часа 1 минута - #7	Н/Д	6 минут 33 секунд
stage-test	Н/Д	20 часов - #4	24 минут

Очередь сборок

Очередь сборок пуста

Состояние сборщиков

1 В ожидании

2 В ожидании

В итоге получаем build и deploy задачи для прода и стейжа. А также задачу для теста стейжа.

Пример выполнения задачи stage-deploy:

Вывод на консоль

Started by upstream project "stage-deploy" build number 7
originally caused by:
Started by user Administrator
Building in workspace /var/lib/jenkins/workspace/stage-deploy/default

Обновление ветки репозитория

```
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url https://github.com/testpass1982/bewelder_redesign.git # timeout=10
Fetching upstream changes from https://github.com/testpass1982/bewelder_redesign.git
> git --version # timeout=10
> git fetch --tags --progress https://github.com/testpass1982/bewelder_redesign.git +refs/heads/*:refs/remotes/origin/*
Checking out Revision d669ea8bfa25c312a64dcedf62c5793e8b3f74 (refs/remotes/origin/deploy)
> git config core.sparsecheckout # timeout=10
> git checkout -f d669ea8bfa25c312a64dcedf62c5793e8b3f74
Commit message: "add stage group config. add jenkins and stage urls for prod nginx proxy"
> git rev-list --no-walk d669ea8bfa25c312a64dcedf62c5793e8b3f74 # timeout=10
[default] $ ansible-playbook ansible/site.yml -i /tmp/inventory8095768937714152651.ini -t deploy -s -U root -l stage
[DEPRECATION WARNING] The command line option --become is deprecated. Use --become-command instead. This feature will be removed in
version 2.9. Deprecation warnings can be disabled by setting deprecation_warnings=False in ansible.cfg.
```

Запуск плейбука

```
PLAY [all] *****
TASK [common : Installing requirements.txt] *****
ok: [jino-stage]

TASK [git : Git pull origin dev] *****
changed: [jino-stage]

TASK [django : Migrations] *****
changed: [jino-stage] => (item=makemigrations)
changed: [jino-stage] => (item=migrate)

RUNNING HANDLER [unicorn : restart unicorn] *****
changed: [jino-stage]

PLAY RECAP *****
jino-stage : ok=4 changed=3 unreachable=0 failed=0

Finished: SUCCESS
```

Процесс выполнения плейбука

Статус сборки

В результате, появляется возможность контролировать процессы доставки кода, автоматизировать задачи тестирования, и многое другое.

Прошу не считать данное описание как руководство. В реальном проекте понадобится помощь специалиста в данной области.

Первоочередные задачи

- Переделать все доступы к серверам через ssh ключи
- Настроить https для подакшн сервера