



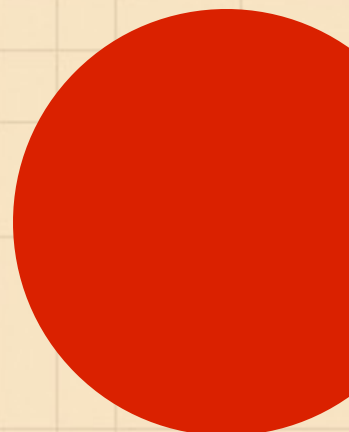

NEW
PRO
LAB


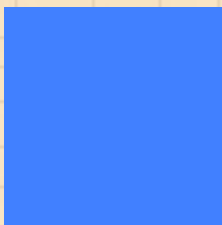
Превращаем код в проект или Немного о CI/CD

Николай Марков
@enchantner

NEWPROLAB.COM

DevOps

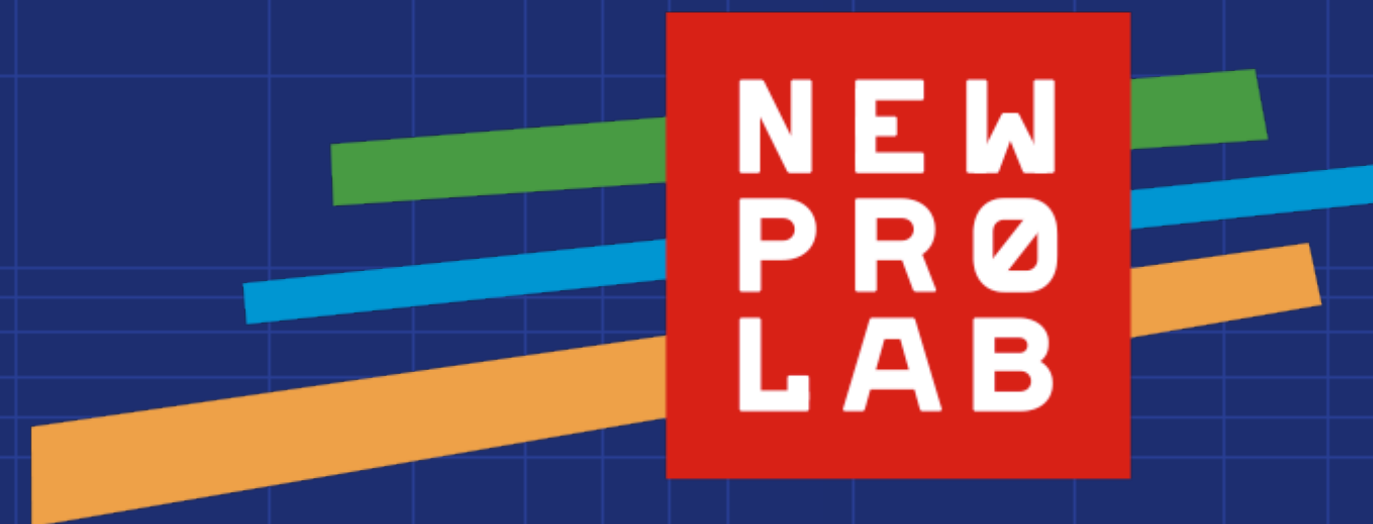


- Набор практик, а не профессия
 - Сопровождение кода, а не техподдержка
 - Выпуск релизов, а не копирование кода на бой
 - Репозитории артефактов и пакетов, а не “python script.py”
 - Системы управления конфигурацией, а не scp
 - Мониторинг, а не “авось 16 гигов хватит”
 - Согласование с эксплуатацией, а не “автоматический запуск тестов”
- 
- 

Continuous Integration/Continuous Delivery

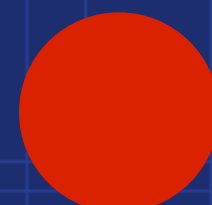
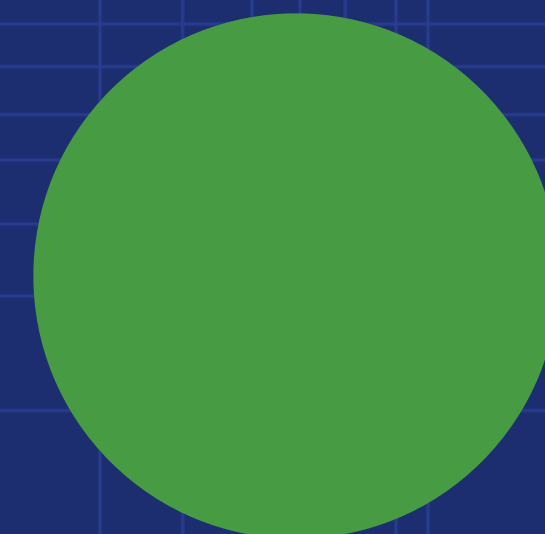
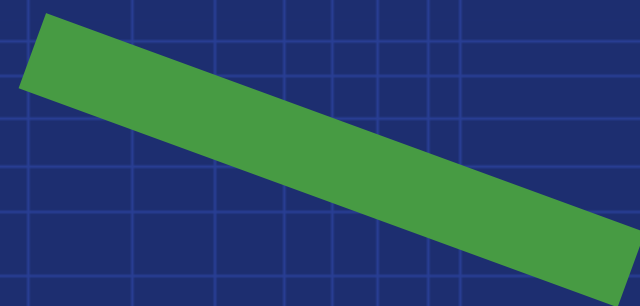
- Использование ветвлений и тегов в контроле версий
- Разделение на стабильные и нестабильные ветки
- Автоматизированное тестирование
- “Ночные” сборки
- Релиз “в любое время”







Python проект


шаг первый - окружение и зависимости




Первое правило клуба



**Лучше ВООБЩЕ НИКОГДА не делать
"sudo pip install". Да, даже если в доке
так написано.**



**Исключение - если это внутри контейнера и вы четко
знаете, что делаете**



Virtualenv

- Создаем окружение
 - **python3 -m venv my_venv**
- Заходим в него:
 - **source my_venv/bin/activate** (на Windows: **my_venv\scripts\activate.bat**)
- Выходим:
 - **deactivate**
- Еще удобно через virtualenvwrapper:
 - **mkvirtualenv my_venv**
 - **workon my_venv**
 - **lsvirtualenv**
 - **rmvirtualenv my_venv**
- Лично я всегда первым делом обновляю установщики:
 - **pip install -U pip wheel setuptools**

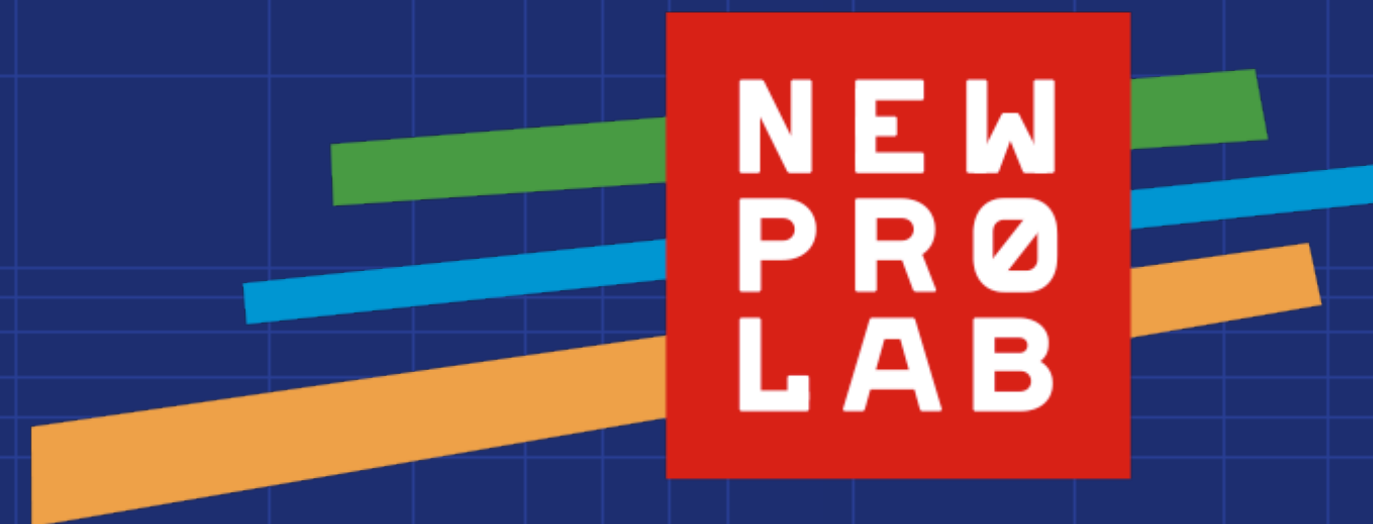
Conda

- Создаем окружение
 - **conda create --name my_venv**
- Заходим в него:
 - **source activate my_venv** или **activate myenv** (зависит от системы)
- Выходим:
 - **deactivate** или **source deactivate**
- Ставим пакеты через **conda install** вместо **pip install**:
 - **conda install requests**
- Удаляем окружение
 - **conda remove --name my_venv --all**

Зависимости проекта

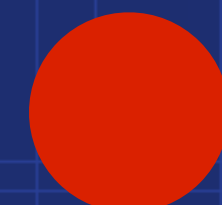
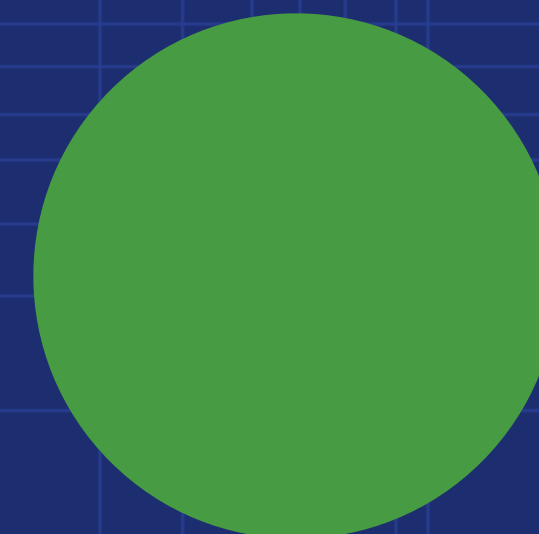
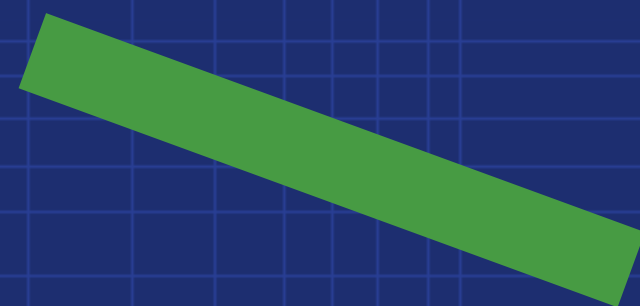


- **pip freeze > requirements.txt**
- еще есть **pip list**
- Посмотрим глазами на содержимое. На некоторых системах эта команда создает ненужную запись о несуществующем пакете "*pkg-resources==0.0.0*" - удалим ее, если она присутствует.




Python проект

шаг второй - репозиторий



Github всем, посоны



This repository


Search


Pull requests


Issues


Marketplace

Explore







 torvalds / linux

Watch

5,955


Star


52,237


Fork

19,406


<> Code


 Pull requests 182


 Projects 0


 Insights


Linux kernel source tree

 721,379 commits

 1 branch

 534 releases

 ∞ contributors

 GPL-2.0

Branch: master


New pull request











Create new file

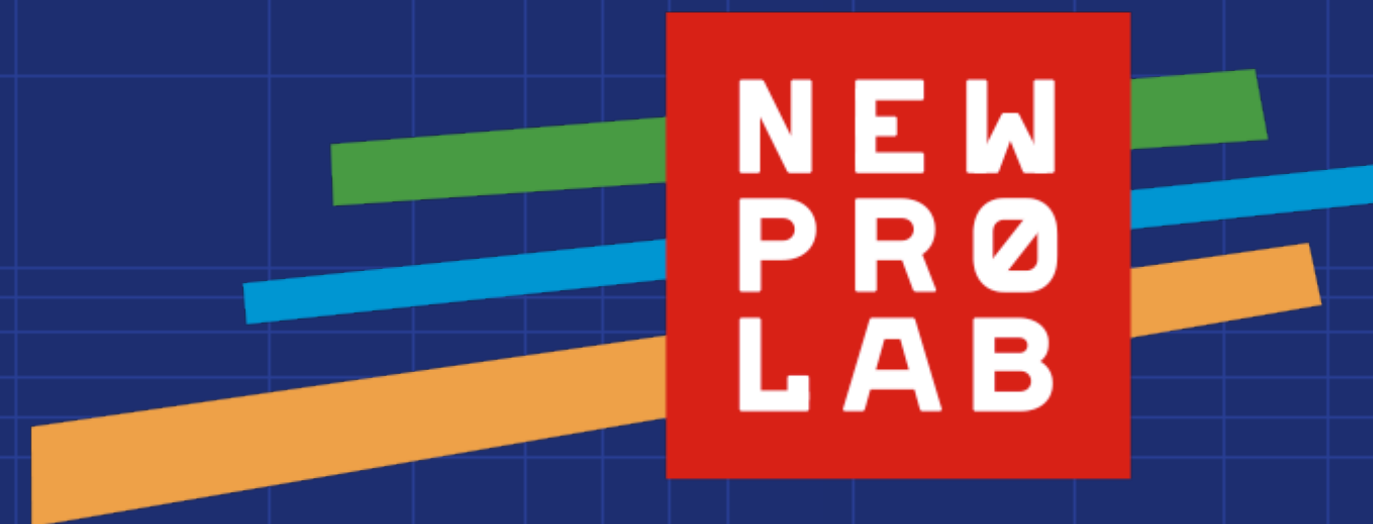
Upload files

Find file

Clone or download

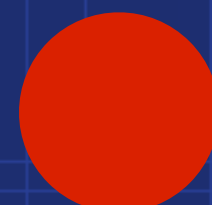
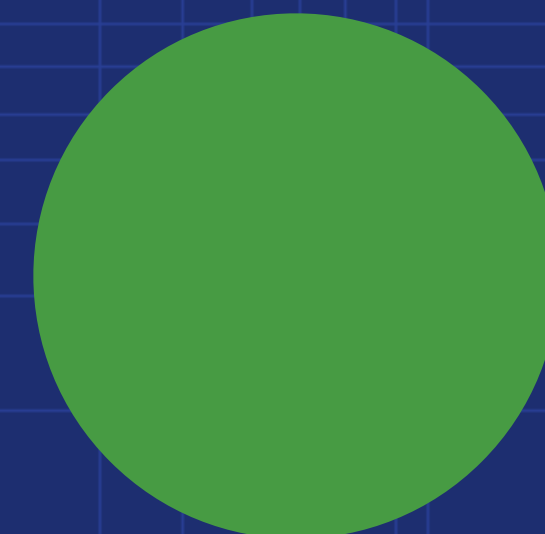
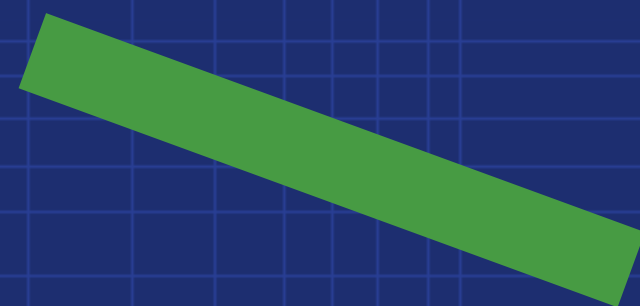
 torvalds Merge tag 'pwm/for-4.15-rc1' of git://git.kernel.org/pub/scm/linux/ke... Latest commit 5a78775 9 hours ago

 Documentation	Merge tag 'pwm/for-4.15-rc1' of git://git.kernel.org/pub/scm/linux/ke...	9 hours ago
 arch	Merge tag 'for-linus-20171120' of git://git.infradead.org/linux-mtd	10 hours ago
 block	Merge branch 'work.iov_iter' of git://git.kernel.org/pub/scm/linux/ke...	6 days ago
 certs	License cleanup: add SPDX GPL-2.0 license identifier to files with no...	21 days ago
 crypto	kmemcheck: stop using GFP_NOTRACK and SLAB_NOTRACK	8 days ago
 drivers	Merge tag 'pwm/for-4.15-rc1' of git://git.kernel.org/pub/scm/linux/ke...	9 hours ago
 firmware	License cleanup: add SPDX GPL-2.0 license identifier to files with no...	21 days ago
 fs	Merge tag 'xfs-4.15-merge-3' of git://git.kernel.org/pub/scm/fs/xfs/x...	10 hours ago
 include	Merge tag 'for-linus-20171120' of git://git.infradead.org/linux-mtd	10 hours ago
 init	EXPERT Kconfig menu: fix broken EXPERT menu	6 days ago



Python проект

шаг третий - аргументы и конфиги





Argparse

```
import argparse
import os.path


def build_parser():
    parser = argparse.ArgumentParser()
    parser.add_argument(
        '-c', '--config', dest='config', action='store', type=str,
        help='path to custom config',
        default=os.path.join(os.path.dirname(__file__), "config.yaml")
    )
    return parser

def main():
    parser = build_parser()
    params, other_params = parser.parse_known_args()
    conf = load_config(params.config)
    ...
```

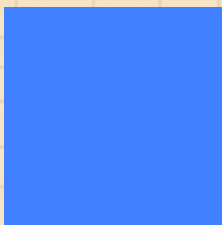

Конфигурационные файлы

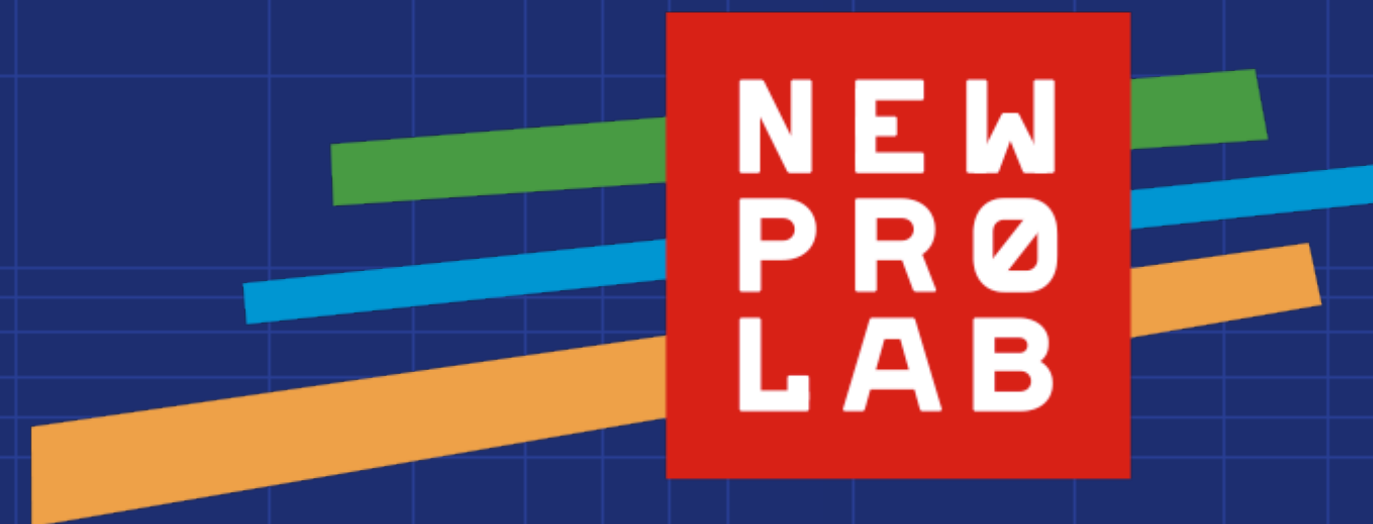


- В конфигурационном файле должны быть все настройки программы, которые мы хотим менять без модификации ее кода
- Формат конфига может быть любым, я бы предложил взять YAML, JSON или INI (про INI можно почитать вот тут - <https://docs.python.org/3/library/configparser.html>)



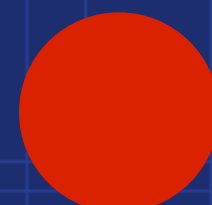
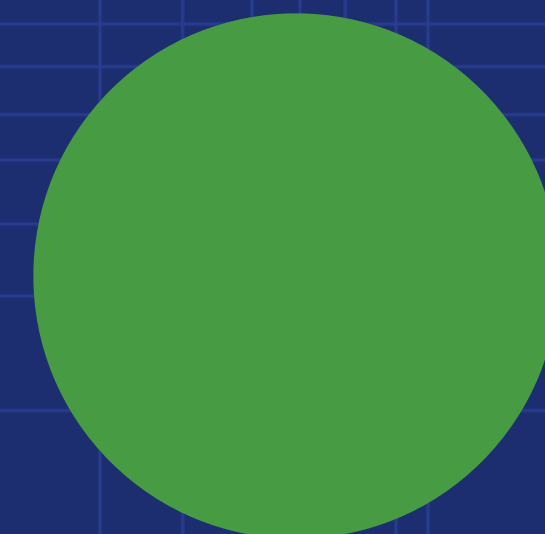
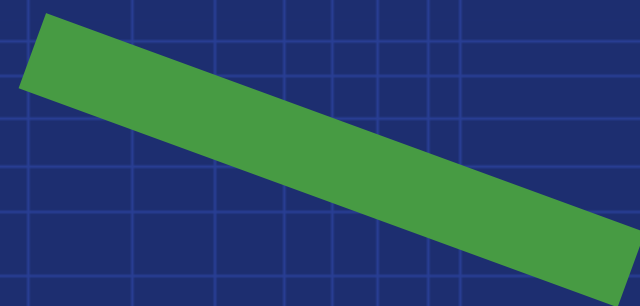
<https://12factor.net/>





Python проект

шаг четвертый - пакетирование



Что такое Python пакет?

- Помните эту команду? `pip install -U pip wheel setuptools`
- Пакеты бывают (в общем случае) двух типов:
 - Архивы (чаще всего формата `.tar.gz`)
 - Бинарные дистрибутивы (формата `.whl`)
- Главный файл пакета - `setup.py`
- Интерпретатор по умолчанию ищет пакеты в `/usr/lib/python3.6`, либо же в `venv/lib/python3.6/site-packages`
- Можно передавать дополнительные пути для поиска с помощью переменной окружения `PYTHONPATH` или добавляя их в `sys.path`

Структурируем схему пакета



my_package <- это папка с нашим проектом



|—— **MANIFEST.in** <- до этого мы сейчас дойдем

|—— **my_package** <- это папка с именем нашего модуля, то, что будет в "*import my_package*"

| |—— **cli.py** <- это базовый файл с нашим кодом

| |—— **config.yaml** <- это файл конфигурации

| |—— **__init__.py** <- это чаще всего пустой файл, который превращает папку в модуль питона

|—— **requirements.txt** <- это наш файл с зависимостями

|—— **setup.py** <- до этого мы сейчас дойдем



setup.py

```
import os
import os.path

from setuptools import find_packages
from setuptools import setup

def find_requires():
    dir_path = os.path.dirname(os.path.realpath(__file__))
    requirements = []
    with open('{0}/requirements.txt'.format(dir_path), 'r') as reqs:
        requirements = reqs.readlines()
    return requirements

if __name__ == "__main__":
    setup(
        name="my_package",
        version="0.0.1",
        description='my cool package',
        packages=find_packages(),
        install_requires=find_requires(),
        include_package_data=True,
        entry_points={
            'console_scripts': [
                'my_command = my_package.cli:main',
            ],
        },
    )
```

MANIFEST.in (включить не-питоновые файлы в проект)

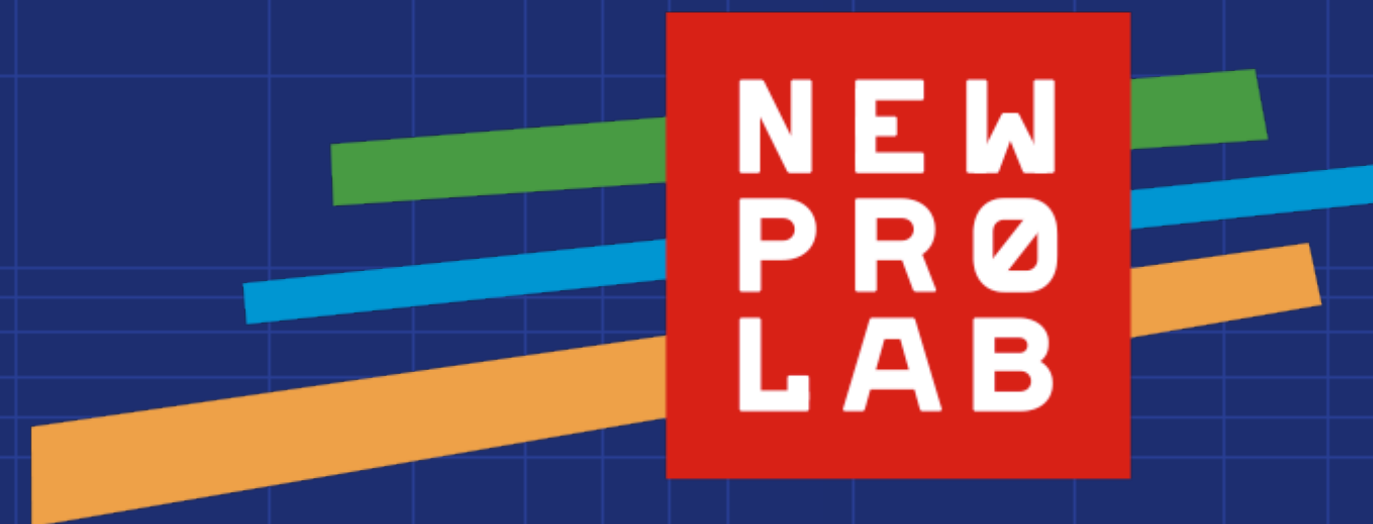
```
include *requirements.txt  
recursive-include my_package *
```


Проверяем, что tar.gz собирается

- `python setup.py sdist`
- Создает папки **dist** и **%имя проекта%.egg-info**, вторую можно смело удалить (там метаданные)
- Созданный в папке **dist** архив и будем собранным пакетом Python
- <https://packaging.python.org/tutorials/distributing-packages/>

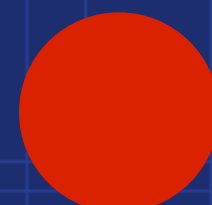
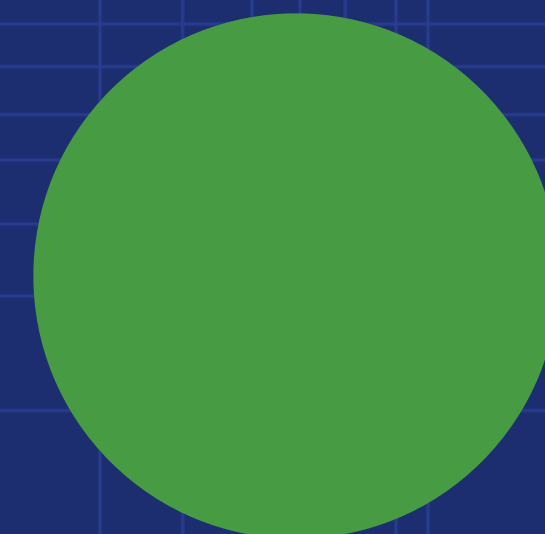
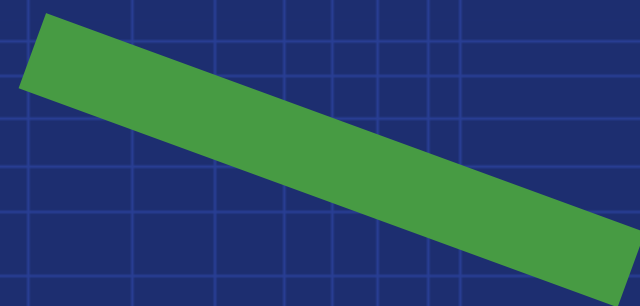
Проверяем, что .whl собирается

- **python setup.py bdist_wheel** (есть еще опция *--universal*, когда проект совместим с Python 2)
- Создает те же папки плюс **build**, где будет примерная структура пакета после инсталляции
- В папке **dist** появится файл **.whl**, который будет бинарным пакетом Python
- Во многих случаях собранный бинарный пакет будет устанавливаться только на ту же ОС, где собирался



Python проект

шаг пятый - документация



Sphinx (правильный)

- `pip install sphinx sphinx-argparse`
- Чаще всего заготовки лежат в папке **docs** в корне проекта
- в папке docs: **sphinx-quickstart**
- лучше задать значения для: "Project name", "Author name(s)", "Project version", "autodoc: automatically insert docstrings from modules" (y), "viewcode: include links to the source code of documented Python objects" (y)
- в файле **conf.py** в **extensions** добавим '**sphinxarg.ext**', а еще наверху раскомментируем и поправим одну точку на две:

```
import os
import sys
sys.path.insert(0, os.path.abspath('..'))
```


Sphinx (правильный)

- Добавим логотип для красоты:
 - **html_logo = '../logo.png'**
- Добавим в **index.rst** после **:caption: Contents:** строки **"quickstart"** и **"develop"** - это дерево нашей документации
- Это значит, нам надо будет создать два файла - **quickstart.rst** и **develop.rst** в том же каталоге

quickstart.rst

Quickstart

=====

```
.. contents:: :local:

.. argparse::
   :module: my_package.cli
   :func: build_parser
   :prog: my_command
```


develop.rst

Reference

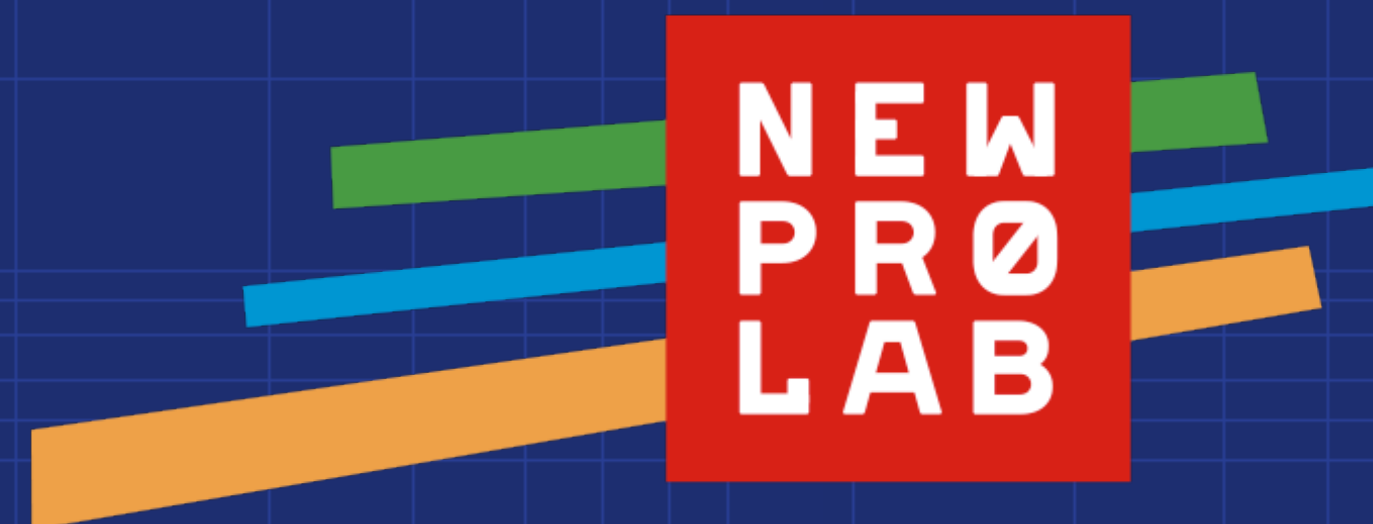
=====

```
.. contents:: :local:

.. automodule:: my_package.cli
   :inherited-members:
```

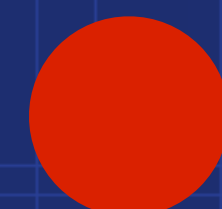
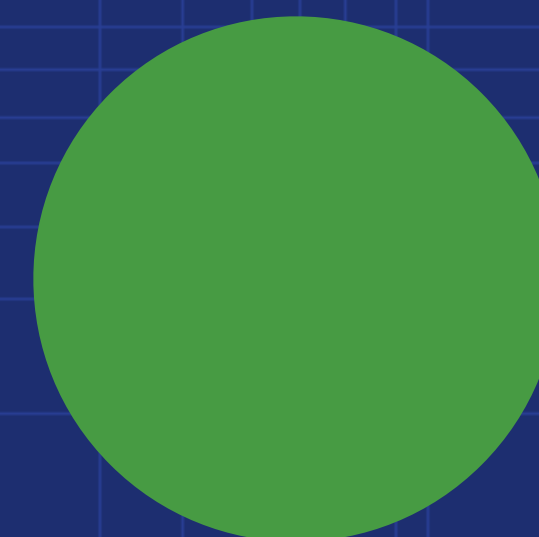
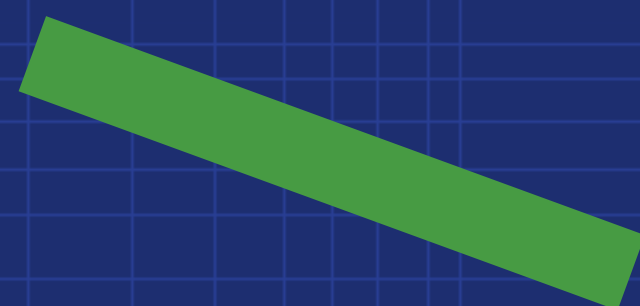

Собираем

make html или **make.bat html** - в папке **_build/html**
создастся куча файлов, главный - **index.html**



Python проект

шаг шестой - режим разработки



Сделаем вид, что пакет стоит в системе

- Используем "**pip install --editable .**" для отладочного режима
- Более старый вариант (не рекомендуется) - "**python setup.py develop**"
- Гораздо меньше ошибок возникает, если всегда использовать путь от имени пакета
- Любые изменения в файлах сразу же станут видны внутри пакета
- Точнее, не сразу, а после перезапуска интерпретатора и/или **reload()** модуля

Особенности и хитрости

- Лично я всегда использую импорты от имени пакета:
 - **from my_package.config import config**
- С отладочным режимом легко проморгать добавление файлов в **MANIFEST.in** - всегда проверяйте руками
- Файл **setup.py** в идеале должен лежать в корне проекта, иначе нельзя будет ставить с помощью **pip** напрямую из **git**. Но можно и не париться и хранить несколько проектов в одном репозитории.

Локальный репозиторий

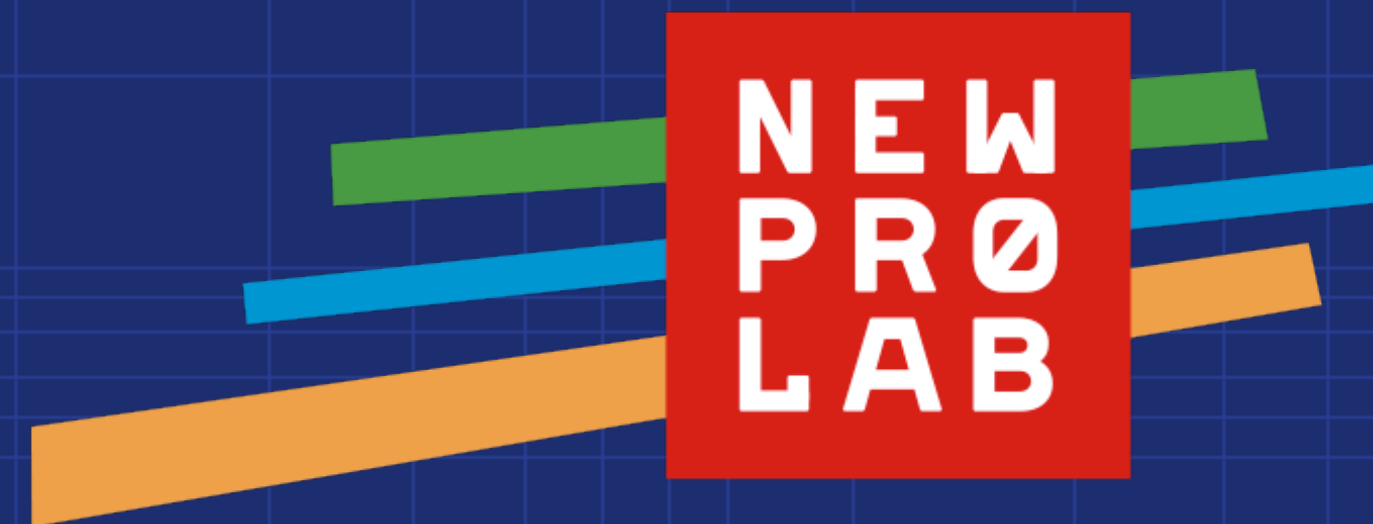
- Пакеты обычно ставятся с **pypi.python.org**, однако можно поднять и локальный репозиторий: <https://pypi.python.org/pypi/pypiserver>
- Потребуется поправить два файла: **~/.pypirc** и **~/.pip/pip.conf**

```
[distutils]
index-servers =
    myrepo

[myrepo]
repository: http://myrepo.example.com:8088
username: kaPaTE|\b
password: 123456
```

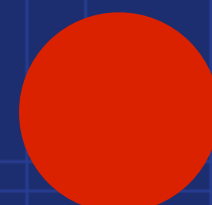
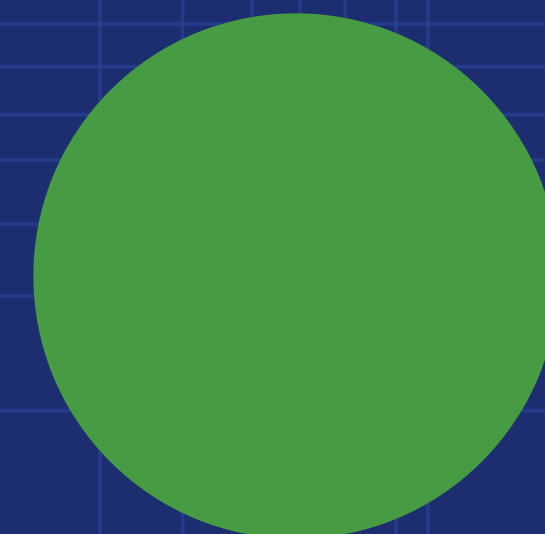
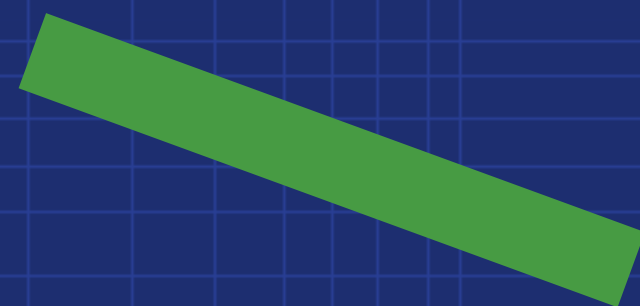
```
[global]
extra-index-url = http://myrepo.example.com:8088/simple/
trusted-host = myrepo.example.com
```

Есть еще **Artifactory** и
прочие хранилища





Python проект

шаг седьмой - заливка и CI






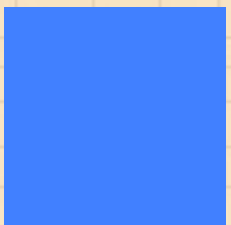
.gitignore



- Файл находится в корне проекта/репозитория. Если его там нет - создайте его
- Просто пишем пути к файлам, которые не хотим отслеживать

Jenkins - инструмент для CI



- **sudo apt install jenkins** или **yum install jenkins** или **brew install jenkins**
 - при первом запуске потребуется пароль из *~/.jenkins/secrets/initialAdminPassword*
 - не забудьте ввести e-mail - там адский баг
 - для примера дополнительно установим GitHub плагин
- 
- 

Jenkins - инструмент для CI

- Ваша нода с Jenkins должна быть видна извне, чтобы публичный GitHub мог запускать джобы
- На GitHub лучше всего создать так называемый Deployment Key (сгенерировать через **ssh-keygen**, например) и прописать путь к вебхуку Jenkins'а, выбрав соответствующий плагин
- При создании джобы в Jenkins надо указать репозиторий и поставить галочку **"GitHub hook trigger for GITScm polling"**

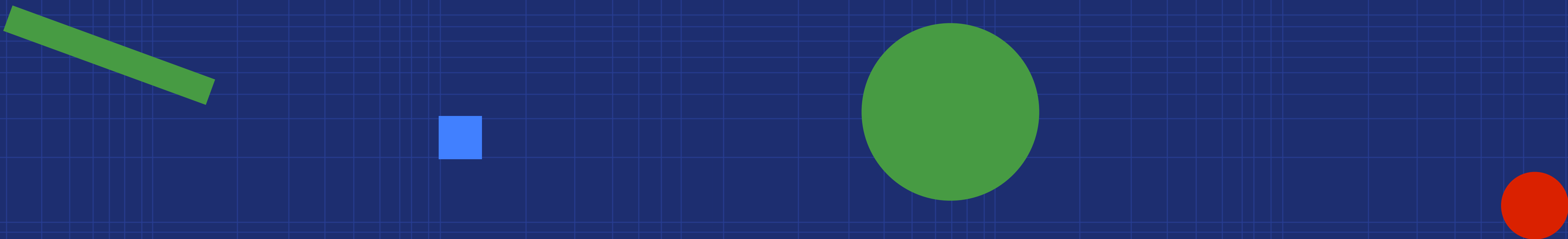
<http://bit.ly/2jT2X9f>





NEW
PRO
LAB

Python проект

шаг восьмой - управление конфигурацией



Система управления конфигурацией



Не совсем корректно называть ее “штукой для инсталляции всяких вещей распределенно”. Это больше “система, которая приводит окружение к конкретному состоянию на основе декларативного описания конфигурации”

Ansible

Есть еще **Puppet** и **Chef**, но они написаны на Ruby, так что любой питонист должен их ненавидеть. Или нет.

```
---
- name: Wait for port 22 to be ready
  become: no
  delegate_to: 127.0.0.1
  wait_for: host="{{ inventory_hostname }}" port=22 state=started timeout=60 delay=15
  tags:
    - always

- name: Updating apt cache
  apt: update-cache=yes
  tags:
    - always

- name: Installing default system packages
  apt: name={{ item }}
  with_items: "{{ DEFAULT_SYSTEM_PACKAGES }}"
  tags:
    - always
```


Создаем пользователя

```
---
- name: Adding user
  user: name={{ user_name }} generate_ssh_key=yes ssh_key_bits=2048 ssh_key_file=.ssh/id_rsa home=/home/{{ user_name }} shell=/bin/bash

- name: Enabling remote login for user - copying key
  command: cp /home/{{ AWS_REMOTE_USER }}/.ssh/authorized_keys /home/{{ user_name }}/.ssh
  when: remote_login

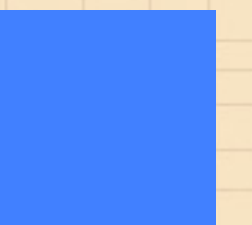
- name: Enabling remote login for user - fixing permissions
  file: path=/home/{{ user_name }}/.ssh/authorized_keys owner={{ user_name }} group={{ user_name }}
  when: remote_login

- name: Creating bash profile
  file: path=/home/{{ user_name }}/.bash_profile state=touch owner={{ user_name }} group={{ user_name }}

- name: Fixing locale
  lineinfile: "dest=/home/{{ user_name }}/.bash_profile state=present line='export LC_ALL=en_US.UTF-8'"

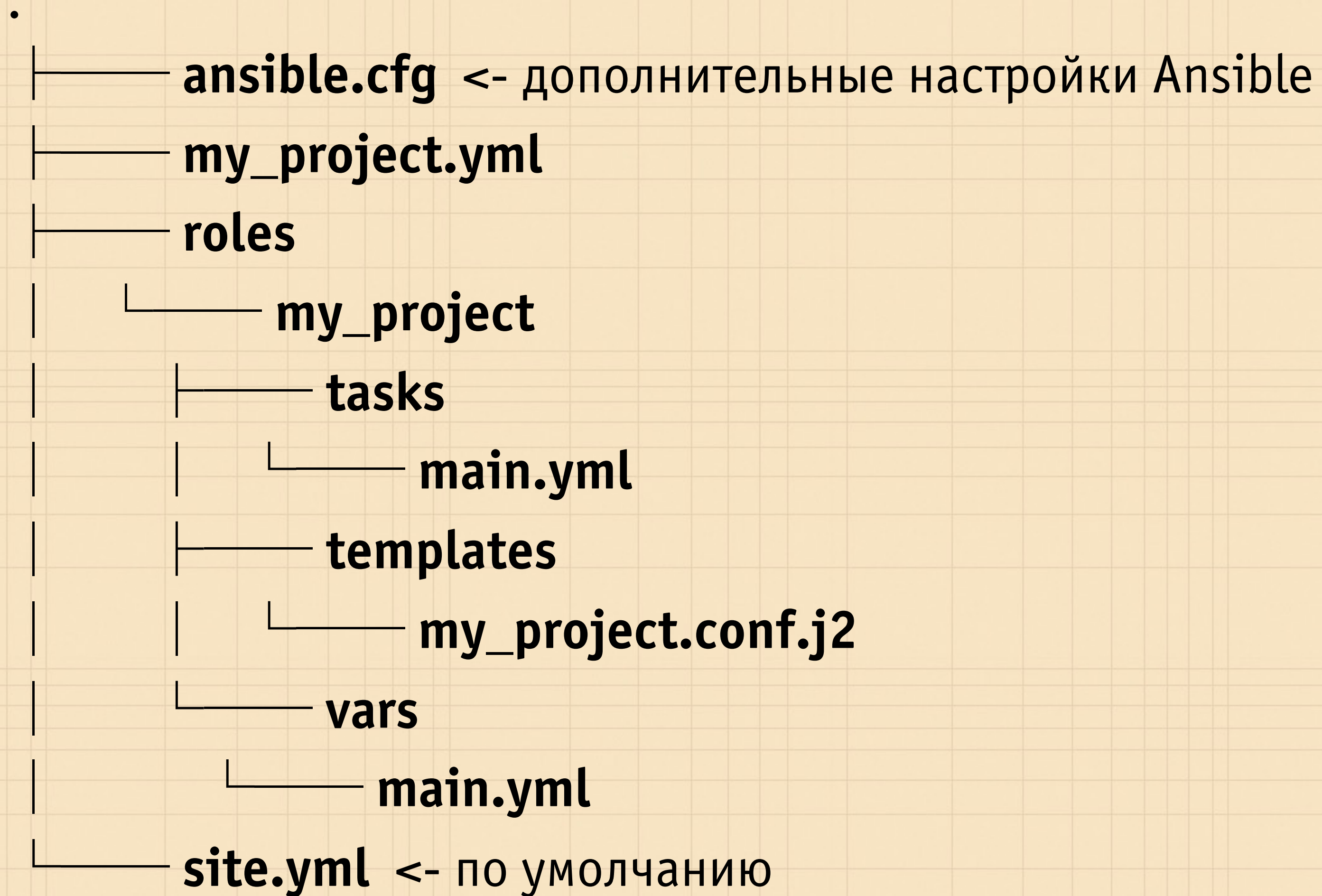
- name: Setting up passwordless sudo
  lineinfile: "dest=/etc/sudoers state=present line='%{{ user_name }}          ALL = (ALL) NOPASSWD: ALL'"
  when: passwordless
```

Создаем virtualenv



```
-  
- name: Checking if already exists  
  stat: path=/home/{{ AWS_REMOTE_NEW_USER }}/.virtualenvs/{{ virtualenv_name }}  
  register: venv_stat  
  become_user: "{{ AWS_REMOTE_NEW_USER }}"  
  
- name: Creating folder for virtualenvs  
  file: path=/home/{{ AWS_REMOTE_NEW_USER }}/.virtualenvs state=directory  
  when: not venv_stat.stat.exists  
  become_user: "{{ AWS_REMOTE_NEW_USER }}"  
  
- name: Creating virtualenv (Python 3)  
  shell: pyvenv-3.6 /home/{{ AWS_REMOTE_NEW_USER }}/.virtualenvs/{{ virtualenv_name }}  
  become_user: "{{ AWS_REMOTE_NEW_USER }}"  
  when: "{{ not venv_stat.stat.exists and py3 is defined }}"  
  
- name: Installing default Python packages in virtualenv  
  pip: name={{ item }} state="latest" executable=/home/{{ AWS_REMOTE_NEW_USER }}/.virtualenvs/{{ virtualenv_name }}/bin/pip  
  with_items: "{{ DEFAULT_PYTHON_PACKAGES }}"  
  become_user: "{{ AWS_REMOTE_NEW_USER }}"
```

Схема библиотеки



Инвентарь (Inventory)

https://docs.ansible.com/ansible/latest/intro_inventory.html

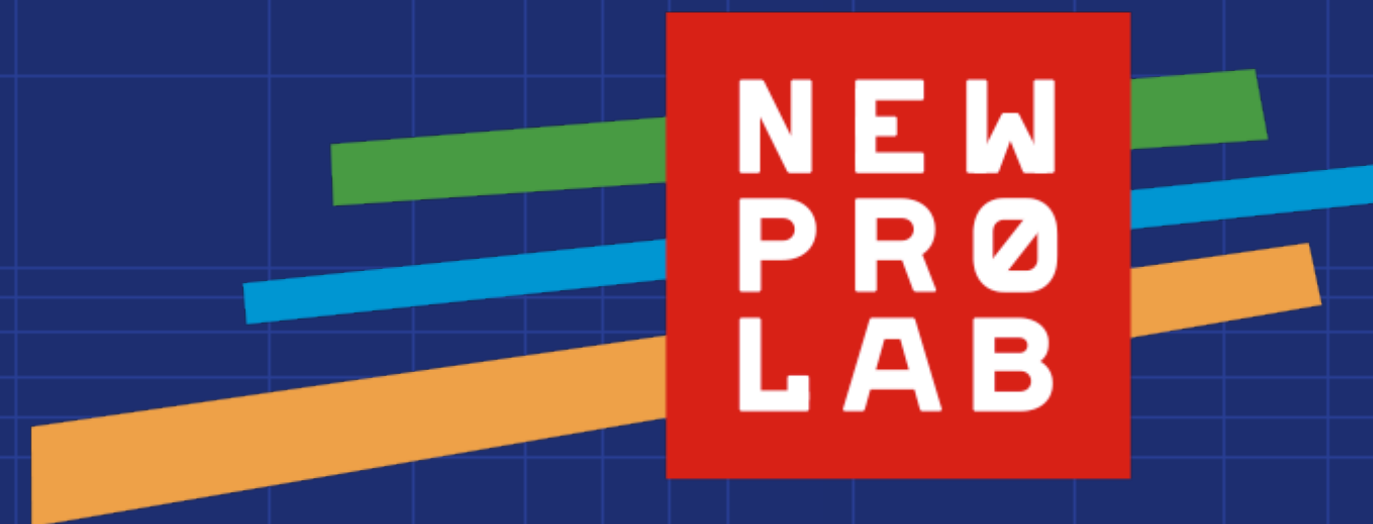
Проверка жизнеспособности

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

Запуск распределенно

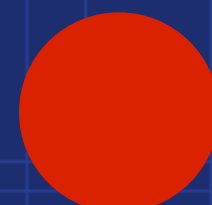
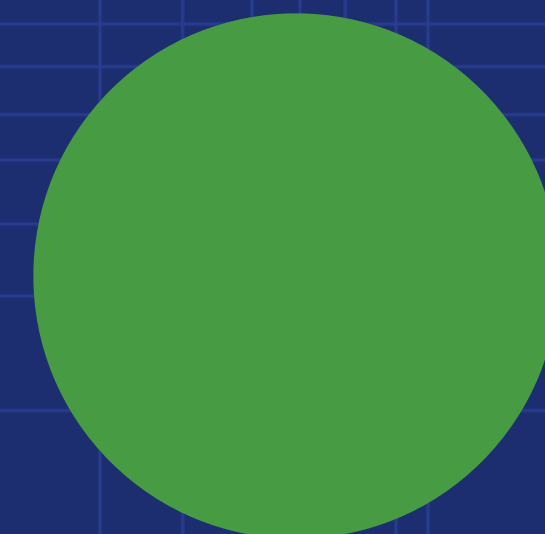
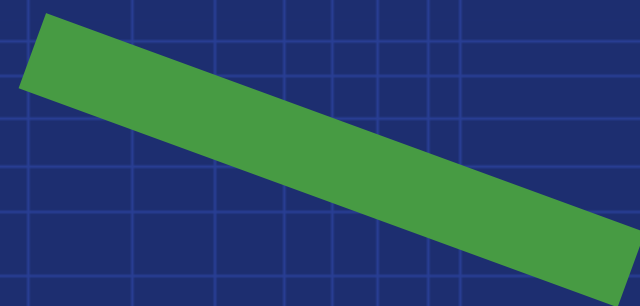
```
ansible -i ./inventory.ini remote -b --become-user=root all \
-m shell -a 'apt-get install nginx'
```

```
ansible-playbook -i ./inventory.ini my_project.yml
```



Python проект

шаг девятый - контейнеризация




Docker / LXC

- Никак не связан с виртуализацией
- Механизм изоляции ресурсов для дерева процессов на основе фичи ядра Linux, которая называется cgroups
- Средненько работает на Windows и MacOS, но ситуация улучшается
- Две основные базовые сущности - image и container
- Docker на Go, LXC(LXD) на C
- **sudo apt install docker-ce**
- **docker-compose** - распределенные инфраструктуры попроще
- **Kubernetes** - сложные распределенные инфраструктуры

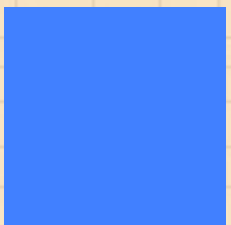
Dockerfile



```
FROM ubuntu:17.04
MAINTAINER Nikolay Markov "enchantner@gmail.com"
RUN apt update -y && \
    apt install -y python3-pip python3-dev
COPY requirements.txt /app/requirements.txt
WORKDIR /app
RUN pip3 install -r requirements.txt
COPY . /app
ENTRYPOINT [ "python3", "hello.py" ]
```



https://www.youtube.com/watch?v=u_iAXzy3xBA



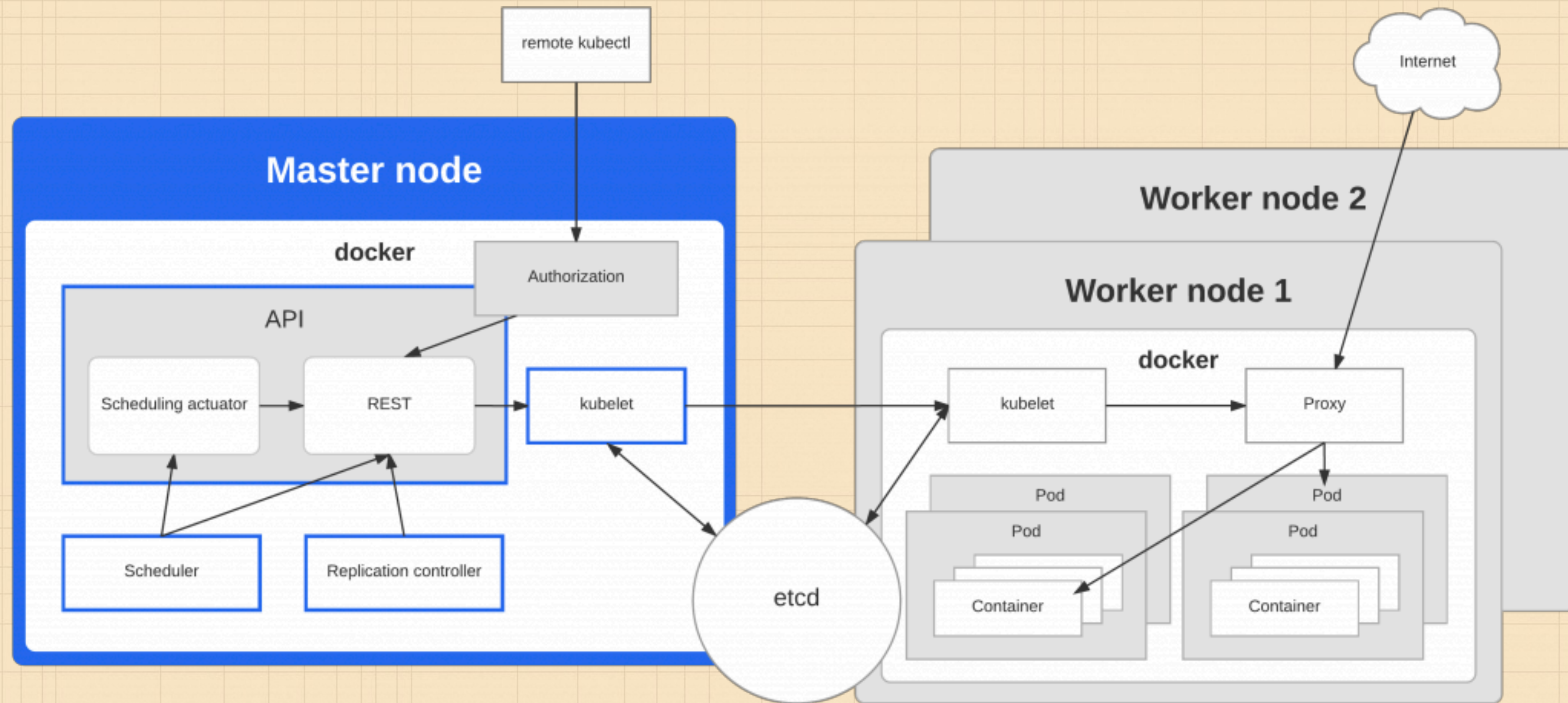
Плохое

<https://habrahabr.ru/post/332450/>

Хорошее

<https://habrahabr.ru/company/southbridge/blog/323554/>

Kubernetes + Jenkins Pipeline



<https://habrahabr.ru/company/flant/blog/352036/>