



Методическое пособие
по выполнению домашнего задания курса
"Администратор Linux. Professional"

Развертывание веб приложения

Содержание

1. Введение	3
2. Цели домашнего задания	4
3. Описание домашнего задания	4
4. Пошаговая инструкция выполнения домашнего задания	5
5. Критерий оценивания	23
6. Рекомендуемые источники	23

1. Введение

За последние десятилетия, средняя скорость разработки увеличивается с большой прогрессией от года к году, потому что IT продукты приносят огромные деньги бизнесу практически в любой сфере. Высокая интенсивность породила большое количество методологий, ускоряющих разработку. Также выросли требования к инфраструктуре. Появилась необходимость динамически обновлять, откатывать, масштабировать и резервировать нашу инфраструктуру. В нынешних реалиях, одной из ключевых методологий в построении информационных систем, является - Infrastructure as Code (Инфраструктура как код, IaC). Она подразумевает подход для управления и описания инфраструктуры ЦОД через конфигурационные файлы, а не через ручное редактирование конфигураций на серверах или интерактивное взаимодействие. Этот подход может включать в себя как декларативный способ описания инфраструктуры, так и через скрипты.

IaC несёт в себе следующие плюсы:

Скорость и уменьшение затрат

Масштабируемость и стандартизация

Безопасность и документация

Восстановление в аварийных ситуациях

P.S. Подробнее про IaC, вы можете почитать в нашей статье - <https://habr.com/ru/company/otus/blog/574278/>

В данной работе, мы применим знания полученные в прошлых уроках и опишем базовую инфраструктуру с помощью конфигураций и манифестов

2. Цели домашнего задания

Получить практические навыки в настройке инфраструктуры с помощью манифестов и конфигураций. Отточить навыки использования ansible/vagrant/docker.

3. Описание домашнего задания

Варианты стенда:

nginx + php-fpm (laravel/wordpress) + python (flask/django) +
js(react/angular);

nginx + java (tomcat/jetty/netty) + go + ruby;

можно свои комбинации.

Реализации на выбор:

на хостовой системе через конфиги в /etc;

деплой через docker-compose.

Для усложнения можно попросить проекты у коллег с курсов по разработке

К сдаче принимается:

vagrant стэнд с проброшенными на локалхост портами

каждый порт на свой сайт

через нжинкс Формат сдачи ДЗ - vagrant + ansible

4. Пошаговая инструкция выполнения домашнего задания

В этом методическом указании мы рассмотрим вариант стенда nginx + php-fpm (wordpress) + python (django) + js(node.js) с деплоем через docker-compose

Настройка окружения в docker-compose

Для развёртки wordpress необходима база данных, выберем mysql:

```
database:
  image: mysql:8.0 # используем готовый образ mysql от разработчиков
  container_name: database
  restart: unless-stopped
  environment:
    MYSQL_DATABASE: ${DB_NAME} # Имя и пароль базы данных будут задаваться
    в отдельном .env файле
    MYSQL_ROOT_PASSWORD: ${DB_ROOT_PASSWORD}
  volumes:
    - ./dbdata:/var/lib/mysql # Чтобы данные базы не пропали при
    остановке/удалении контейнера, будем сохранять их на хост-машине
  command: '--default-authentication-plugin=mysql_native_password'
```

env:

```
# Переменные которые будут использоваться для создания и подключения БД
DB_NAME=wordpress
DB_ROOT_PASSWORD=dbpassword
# Переменные необходимые python приложению
MYSITE_SECRET_KEY=put_your_django_app_secret_key_here
DEBUG=True
```

Для того чтобы объединить наши приложения, создадим сеть и будем добавлять каждый контейнер в неё:

```
database:
  image: mysql:8.0
  container_name: database
  restart: unless-stopped
  environment:
    MYSQL_DATABASE: ${DB_NAME}
    MYSQL_ROOT_PASSWORD: ${DB_ROOT_PASSWORD}
```

```
volumes:
  - ./dbdata:/var/lib/mysql
command: '--default-authentication-plugin=mysql_native_password'
networks:
  - app-network
```

networks:

```
app-network:
  driver: bridge
```

Контейнер wordpress:

```
wordpress:
  image: wordpress:5.1.1-fpm-alpine # официальный образ от разработчиков
  container_name: wordpress
  restart: unless-stopped

# на странице образа в docker hub написано, какие можно задать переменные
# контейнеру https://hub.docker.com/_/wordpress

  environment:
    WORDPRESS_DB_HOST: database
    WORDPRESS_DB_NAME: "${DB_NAME}" # Также импортируем переменные из .env
    WORDPRESS_DB_USER: root
    WORDPRESS_DB_PASSWORD: "${DB_ROOT_PASSWORD}"

  volumes:
    - ./wordpress:/var/www/html # сохраняем приложение на хост машине

  networks:
    - app-network

  depends_on:
    - database # контейнер wordpress дождется запуска БД
```

Контейнер nginx:

```
nginx:
  image: nginx:1.15.12-alpine
  container_name: nginx
  restart: unless-stopped

# Т.к. все запросы к приложениям будут проходить через nginx, пробросим под
# каждое приложение по порту.

  ports:
    - 8083:8083
```

```

- 8081:8081
- 8082:8082

volumes:

# будет использоваться php-fpm, необходимо смонтировать статические файлы
wordpress :

- ./wordpress:/var/www/html
- ./nginx-conf:/etc/nginx/conf.d # монтируем конфиг

networks:

- app-network

depends_on: # nginx будет запускаться после всех приложений
- wordpress
- app
- node

```

Рассмотрим сервер nginx для wordpress:

```

# Данный сервер отвечает за проксирование на wordpress через fastcgi

server {

# Wordpress будет отображаться на 8083 порту хоста

    listen 8083;
    listen [::]:8083;
    server_name localhost;
    index index.php index.html index.htm;

# Задаем корень корень проекта, куда мы смонтировали статику wordpress

    root /var/www/html;
    location ~ /\.well-known/acme-challenge {
        allow all;
        root /var/www/html;
    }
    location / {
        try_files $uri $uri/ /index.php$?args;
    }

# Само fastcgi проксирование в контейнер с wordpress по 9000 порту

    location ~ \.php$ {
        try_files $uri =404;
    }
}

```

```

        fastcgi_split_path_info ^(.+\.(php))(/.+)$;
        fastcgi_pass wordpress:9000;
        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME
$document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }

    location = /favicon.ico {
        log_not_found off; access_log off;
    }

    location ~* \.(css|gif|ico|jpeg|jpg|js|png)$ {
        expires max;
        log_not_found off;
    }
}

```

Сервер nginx для django:

```

upstream django {
    server app:8000;
}

server {
    # Django будет отображаться на 8081 порту хоста
    listen 8081;
    listen [::]:8081;
    server_name localhost;
    location / {
        try_files $uri @proxy_to_app;
    }
    # тут используем обычное проксирование в контейнер django
    location @proxy_to_app {
        proxy_pass http://django;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
    }
}

```



```

    proxy_set_header Connection "upgrade";
    proxy_redirect off;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Host $server_name;
}
}

```

Сервер nginx для node.js:

Node.js будет отображаться на 8082 порту хоста

```

server {
    listen 8082;
    listen [::]:8082;
    server_name localhost;
    location / {
        proxy_pass http://node:3000;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_redirect off;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Host $server_name;
    }
}

```

Описание контейнера django:

```

app:
    build: ./python # для нашего приложения нужны зависимости, поэтому
собираем свой образ
    container_name: app
    restart: always
    env_file:
        - .env # импортируем в контейнер переменные из .env

```

command:

```
"gunicorn --workers=2 --bind=0.0.0.0:8000 mysite.wsgi:application" #  
команда для запуска django проекта, приложение будет работать на 8000 порту  
контейнера
```

networks:

- app-network

Исходный код Django приложения:

requirements.txt:

Django==3.1

gunicorn==20.0.4

pytz==2020.1

manage.py:

```
#!/usr/bin/env python
```

```
"""Django's command-line utility for administrative tasks."""
```

```
import os
```

```
import sys
```

```
def main():
```

```
    """Run administrative tasks."""
```

```
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'mysite.settings')
```

```
    try:
```

```
        from django.core.management import execute_from_command_line
```

```
    except ImportError as exc:
```

```
        raise ImportError(
```

```
            "Couldn't import Django. Are you sure it's installed and "
```

```
            "available on your PYTHONPATH environment variable? Did you "
```

```
            "forget to activate a virtual environment?"
```

```
        ) from exc
```

```
    execute_from_command_line(sys.argv)
```

```
if __name__ == '__main__':
```

```
    main()
```

wsgi.py:

```
import os
```

```
from django.core.wsgi import get_wsgi_application
```

```
os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'mysite.settings')
application = get_wsgi_application()
```

urls.py:

```
from django.contrib import admin
from django.urls import path
```

```
urlpatterns = [
    path('admin/', admin.site.urls),
]
```

settings.py:

```
import os
import ast
from pathlib import Path

BASE_DIR = Path(__file__).resolve(strict=True).parent.parent
SECRET_KEY = os.getenv('MYSITE_SECRET_KEY', '')
DEBUG = ast.literal_eval(os.getenv('DEBUG', 'True'))
ALLOWED_HOSTS = []
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

```
MIDDLEWARE = [
```

```

'django.middleware.security.SecurityMiddleware',
'django.contrib.sessions.middleware.SessionMiddleware',
'django.middleware.common.CommonMiddleware',
'django.middleware.csrf.CsrfViewMiddleware',
'django.contrib.auth.middleware.AuthenticationMiddleware',
'django.contrib.messages.middleware.MessageMiddleware',
'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'mysite.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    ],
]

WSGI_APPLICATION = 'mysite.wsgi.application'

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

AUTH_PASSWORD_VALIDATORS = [
    {

```

```

        'NAME':
'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME':
'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True

STATIC_URL = '/static/'

```

Описание контейнера node.js:

```

node:
  image: node:16.13.2-alpine3.15
  container_name: node
  working_dir: /opt/server # переназначим рабочую директорию для удобства
  volumes:
    - ./node:/opt/server # пробрасываем приложение в директорию контейнера
  command: node test.js # запуск приложения
  networks:
    - app-network

```

```

Cam dockerfile:
FROM python:3.8.3
ENV APP_ROOT /src

```

```
ENV CONFIG_ROOT /config
RUN mkdir ${CONFIG_ROOT}
COPY requirements.txt ${CONFIG_ROOT}/requirements.txt
RUN pip install -r ${CONFIG_ROOT}/requirements.txt
RUN mkdir ${APP_ROOT}
WORKDIR ${APP_ROOT}
ADD . ${APP_ROOT}
```

Исходный код node.js приложения:

```
test.js:

const http = require('http');
const hostname = '0.0.0.0';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello from node js server');
});

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

На этом настройка docker-compose закончена, приступим к написанию playbook:

```
hosts: DynamicWeb # имя хоста, который мы создадим Vagrant`om
become: yes # Установка Docker через sudo
gather_facts: false
tasks: # Перечисляем задачи которые выполнит наш playbook
  - name: Install docker packages # устанавливаем пакеты необходимые для
установки докера
    become: yes
    apt:
      name: "{{ item }}"
      state: present
      update_cache: yes
```

```

with_items:
    - apt-transport-https
    - ca-certificates
    - curl
    - software-properties-common
tags:
    - docker
- name: Add Docker's official GPG key
  become: yes
  apt_key:
    url: https://download.docker.com/linux/ubuntu/gpg
    state: present
  tags:
    - docker

- name: Verify that we have the key with the fingerprint
  become: yes
  apt_key:
    id: 0EBFCD88
    state: present
  tags:
    - docker

- name: Set up the stable repository # добавляем репозиторий докера
  become: yes
  apt_repository:
    repo: deb [arch=amd64] https://download.docker.com/linux/ubuntu xenial
stable
    state: present
    update_cache: yes
  tags:
    - docker

- name: Update apt packages
  become: yes
  apt:
    update_cache: yes

```

```

tags:
  - docker
- name: Install docker # установка докера
  become: yes
  apt:
    name: docker-ce
    state: present
    update_cache: yes
  tags:
    - docker
- name: Add remote "vagrant" user to "docker" group
  become: yes
  user:
    name: vagrant
    group: "docker"
    append: yes
  tags:
- docker - name: Install docker-compose
  become: yes
  get_url:
    url :
https://github.com/docker/compose/releases/download/1.25.1-rc1/docker-compose
-Linux-x86_64
    dest: /usr/local/bin/docker-compose
    mode: 0777
- name: Copy project # Копируем проект с хост машины в созданную через
vagrant
  copy: src=project dest=/home/vagrant
- name: reset ssh connection # чтобы применились права на использование
docker, необходимо перелогиниться
  meta: reset_connection
- name: Run container
  shell:
    cmd: "docker-compose -f docker-compose.yml up -d"
    chdir: /home/vagrant/project

```


Ну и соответственно Vagrantfile:

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

  config.vm.provision "ansible" do |ansible|
    ansible.playbook = "prov.yml"
  end

  config.vm.define "DynamicWeb" do |vmconfig|
    vmconfig.vm.box = 'bento/ubuntu-20.04'
    vmconfig.vm.hostname = 'DynamicWeb'

    vmconfig.vm.network "forwarded_port", guest: 8083, host: 8083
    vmconfig.vm.network "forwarded_port", guest: 8081, host: 8081
    vmconfig.vm.network "forwarded_port", guest: 8082, host: 8082
    vmconfig.vm.provider "virtualbox" do |vbx|
      vbx.memory = "2048"
      vbx.cpus = "2"
      vbx.customize ["modifyvm", :id, '--audio', 'none']
    end
  end

end
```

Вам необходимо собрать перечисленные выше конфигурации и получить структуру проекта приведенную ниже. Только в этом случае выполнится корректная сборка стенда.

```
someuser@mac:~/dynamicweb$ tree
```

```
.
├── project
│   ├── docker-compose.yml
│   ├── nginx-conf
│   │   └── nginx.conf
│   ├── node
│   │   └── test.js
│   ├── python
│   │   └── Dockerfile
```

```
|  |  └─ manage.py
|  |  └─ mysite
|  |  |  └─ asgi.py
|  |  |  └─ __init__.py
|  |  |  └─ settings.py
|  |  |  └─ urls.py
|  |  |  └─ wsgi.py
|  |  └─ requirements.txt
|  └─ README.md
|  └─ screens
|      └─ 8081.png
|      └─ 8082.png
|      └─ 8083.png
└─ prov.yml
└─ Vagrantfile
```

6 directories, 17 files

Конфигурирование выполнено, развернём стенды:

```
someuser@mac:~/Документы/dynamicweb$ vagrant up
Bringing machine 'DynamicWeb' up with 'virtualbox' provider...
==> DynamicWeb: Importing base box 'bento/ubuntu-20.04'...
==> DynamicWeb: Matching MAC address for NAT networking...
==> DynamicWeb: Checking if box 'bento/ubuntu-20.04' version '202112.19.0' is
up to date...
==> DynamicWeb: Setting the name of the VM:
dynamicweb_DynamicWeb_1644131475981_21322
==> DynamicWeb: Clearing any previously set network interfaces...
==> DynamicWeb: Preparing network interfaces based on configuration...
    DynamicWeb: Adapter 1: nat
==> DynamicWeb: Forwarding ports...
    DynamicWeb: 8083 (guest) => 8083 (host) (adapter 1)
    DynamicWeb: 8081 (guest) => 8081 (host) (adapter 1)
    DynamicWeb: 8082 (guest) => 8082 (host) (adapter 1)
    DynamicWeb: 22 (guest) => 2222 (host) (adapter 1)
==> DynamicWeb: Running 'pre-boot' VM customizations...
==> DynamicWeb: Booting VM...
```

```

==> DynamicWeb: Waiting for machine to boot. This may take a few minutes...
DynamicWeb: SSH address: 127.0.0.1:2222
DynamicWeb: SSH username: vagrant
DynamicWeb: SSH auth method: private key
DynamicWeb:
DynamicWeb: Vagrant insecure key detected. Vagrant will automatically
replace
DynamicWeb: this with a newly generated keypair for better security.
DynamicWeb:
DynamicWeb: Inserting generated public key within guest...
DynamicWeb: Removing insecure key from the guest if it's present...
DynamicWeb: Key inserted! Disconnecting and reconnecting using new SSH
key...
==> DynamicWeb: Machine booted and ready!
==> DynamicWeb: Checking for guest additions in VM...
==> DynamicWeb: Setting hostname...
==> DynamicWeb: Mounting shared folders...
DynamicWeb: /vagrant => /home/someuser/Документы/dynamicweb
==> DynamicWeb: Running provisioner: ansible...
DynamicWeb: Running ansible-playbook...
/usr/lib/python3/dist-packages/requests/__init__.py:89:
RequestsDependencyWarning: urllib3 (1.26.7) or chardet (3.0.4) doesn't match
a supported version!
    warnings.warn("urllib3 ({{}}) or chardet ({{}}) doesn't match a supported "

PLAY [DynamicWeb]
*****

TASK [Install docker packages]
*****

[DEPRECATION WARNING]: Invoking "apt" only once while using a loop via
squash_actions is deprecated. Instead of using a loop to supply multiple
items
and specifying `name: "{{ item }}"`, please use `name:
['apt-transport-https',
'ca-certificates', 'curl', 'software-properties-common']` and remove the
loop.
This feature will be removed in version 2.11. Deprecation warnings can be
disabled by setting deprecation_warnings=False in ansible.cfg.

```

```
changed: [DynamicWeb] => (item=['apt-transport-https', 'ca-certificates',  
'curl', 'software-properties-common'])
```

```
TASK [Add Docker's official GPG key]  
*****
```

```
changed: [DynamicWeb]
```

```
TASK [Verify that we have the key with the fingerprint]  
*****
```

```
ok: [DynamicWeb]
```

```
TASK [Set up the stable repository]  
*****
```

```
changed: [DynamicWeb]
```

```
TASK [Update apt packages]  
*****
```

```
ok: [DynamicWeb]
```

```
TASK [Install docker]  
*****
```

```
changed: [DynamicWeb]
```

```
TASK [Add remote "vagrant" user to "docker" group]  
*****
```

```
changed: [DynamicWeb]
```

```
TASK [Install docker-compose]  
*****
```

```
changed: [DynamicWeb]
```

```
TASK [Copy project]  
*****
```

```
changed: [DynamicWeb]
```

```
TASK [Run container]  
*****
```

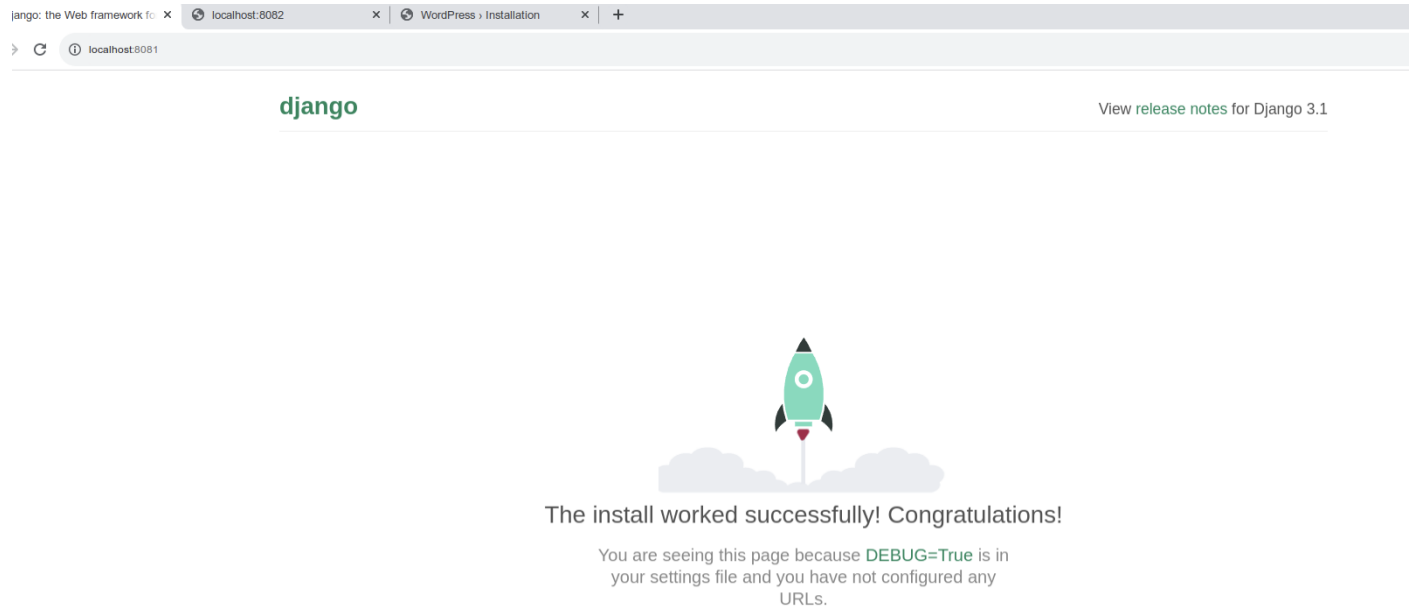
```
0changed: [DynamicWeb]
```

PLAY RECAP

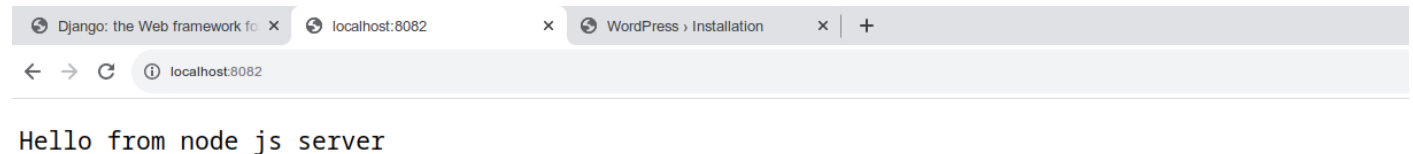
DynamicWeb : ok=10 changed=8 unreachable=0 failed=0
skipped=0 rescued=0 ignored=0

Проверка:

<http://localhost:8081/>




<http://localhost:8082/>



<http://localhost:8083/>

Web framework fo x localhost:8082 x WordPress › Installation x +

localhost8083/wp-admin/install.php



English (United States)

Afrikaans

العربية

العربية المغربية

অসমীয়া

Azərbaycan dili

گۆنئی آذربایجان

Беларуская мова

Български

বাংলা

བོད་ཡིག

Bosanski

Català

Cebuano

Čeština

Cymraeg

Dansk

Deutsch (Österreich)

Deutsch (Schweiz)

Deutsch (Schweiz, Du)

Deutsch

Deutsch (Sie)

~

5. Критерий оценивания

Критерий оценивания:

Статус "Принято" ставится при выполнении следующих условий:

Ссылка на репозиторий GitHub.

Vagrantfile с шагами установки необходимых компонентов

Настройка виртуальных машин происходит с помощью Ansible.

Приложения доступны с разных портов

Vagrant порты проброшены на локалхост.

Описать в README.md последовательность действий для выполнения задания, сопровождая небольшими комментариями.

Описать проверку правильности выполненной работы.

6. Рекомендуемые источники

- Документация Vagrant - (<https://www.vagrantup.com/docs>)
- Документация Ansible - (<https://docs.ansible.com/ansible/latest/collections/index.html>)
- Документация Docker - (<https://docs.docker.com/compose/gettingstarted/>)
- Документация Nginx - (<https://nginx.org/ru/docs/>)
- Docker Hub - (<https://hub.docker.com/>)