

Проект - варианты заданий и правила выполнения

Общие требования

Список будет пополняться :)

Dockerfile:

- Минимальная версия докера Docker version 19.03.13, build 4484c46d9d
- Базовый образ ubuntu:22.04
- Не использовать Expose
- При установке любых пакетов и программ (в том числе в requirements) ВСЕГДА указывать версии
- Ограничить установку зависимостей apt одной строкой (один RUN)
- Если настройка одной части приложения состоит из нескольких команд → необходимо разместить их в одном слое (в одном RUN)

Docker-compose:

- Минимальная версия docker compose version 1.27.4, build 40524192
- Все должно собираться по команде docker-compose build без sudo
- Не использовать тип сети HOST
- Не отрывать лишних (непредусмотренных заданием) портов
- Не использовать порты хост-машины $\leftarrow 1024$

Варианты заданий

<https://docs.google.com/spreadsheets/d/1Tk7py40c2guXg82FIYvp5BF9DYnMFcatnOCnKyLXH4/edit#gid=0>

Подсказки по условиям вашего задания - ниже.

Расшифровка условий задания

- Построение тестов
 - Проверка на соответствие стилю кодирования / бьютификация - подключаем проверку стиля кодирования (технологии ниже) и встраиваем ее в цепочку запуска
 - Проверка на pep8 - используем <https://pypi.org/project/pep8/>
 - Форматирование Python (yapf) - <https://github.com/google/yapf>
 - Форматирование Python (black) - <https://github.com/psf/black>
 - Форматирование Python (flake8) - <https://github.com/PyCQA/flake8>
 - Форматирование HTML, JS (prettier) - <https://prettier.io>
 - Форматирование Python (pindent) - <http://svn.python.org/projects/python/trunk/Tools/scripts/pindent.py>

- Форматирование Python (pybetter) - <https://github.com/lensvol/pybetter>
- Форматирование Python (ssort) - <https://github.com/bwhmather/ssort>
- Форматирование Python (usort) - <https://github.com/facebook/usort>
- Проверка бьютификатором для HTML - <https://pypi.org/project/html-linter/>
- Причесывание бьютификатором HTML - <https://pypi.org/project/css-html-prettify/>
- Проверка бьютификатором для JS - <https://pypi.org/project/pyjslint/>
- Статический анализ - подключаем статическую проверку (через pylint) и встраиваем ее в цепочку запуска
 - **Анализ по 10 существующим критериям** - выберите по 10 уникальных критериев проверки, настройте запуск на них и допустите все десять ошибок в коде проекта:)
 - **Создание своего критерия и проверка только по нему** - проверяем на наличие переменных, название которых совпадает с вашим именем
- Интеграционные тесты - пишем интеграционные тесты (через requests) и встраиваем их в цепочку запуска
 - Проверка на коды возврата
 - Проверка на заголовки
 - Проверка на загрузку файла
- Selenium - пишем selenium тесты и встраиваем их в цепочку запуска (нельзя использовать для selenium отдельный контейнер, реализуйте тесты в рамках контейнера tester)
 - Система для тестирования - <https://dev.digital.etu.ru/trajectories-test/>
 - Авторизуйтесь по своему логину/паролю в ЛК ЛЭТИ (укажите через .env-файл)
 - По сути заданий - см. лекцию 2024-03-01
- Docker
 - Внешний SSH доступ в контейнеры - организуем доступ через протокол SSH контейнер одним из следующих способов: или по ключу в каталоге с проектом, или генерируем пароль для доступа и сообщаем его при сборке / запуске, или генерируем новую пару ключей и выводим их в файлы. Порт для SSH должен быть доступен снаружи docker-compose конфигурации.
 - В app - по публичному ключу (существующему)
 - В tester - по публичному ключу (существующему)
 - В app и tester - по публичному ключу (существующему)
 - В app - по паролю
 - В tester - по паролю
 - В app и tester - по паролю
 - В app - по сгенерированной в процессе сборки паре ключей (ключи выводим в файл)
 - В tester - по сгенерированной в процессе сборки паре ключей (ключи выводим в файл)
 - В app и tester - по сгенерированной в процессе сборки паре ключей (ключи выводим в файл)
 - Вывод логов работы tester - задание о том, куда и как выводить логи тестирования в контейнере tester
 - **Каждый этап тестирования - в docker log (stdout + stderr) и в отдельный файл оба потока по каждому виду тестирования** Совместно выводим логи тестирования (stdout + stderr) так, чтобы их видел и docker logs, и они собирались в файле.
 - **Каждый этап тестирования - в docker log (stdout + stderr) и в общие файлы (отдельно - для stdout, отдельно - для stderr)** - Совместно выводим логи тестирования (stdout + stderr) так, чтобы их видел docker logs,

но при этом в один файл сохраняем stdout логов, в другой - stderr.

- **Каждый этап тестирования - в docker log (stdout + stderr) + добавить к записям лога timestamp** - помимо вывода в docker log нужно также сделать, чтобы перед каждой записью в логе стоял timestamp (или текущее дата и время)
- Docker-compose
 - Передача параметров в конфигурацию через .env, какие параметры передаем - нужно сделать как пример env файла, так и смаппить (А кое где и написать скрипты настройки) параметры на нужное поведение
 - **Порт для веб-сервера** - публичный порт, на котором слушает веб-сервер
 - **Список этапов тестирования для запуска** - список шагов из пункта "Построение тестов", которые будут запущены. Если не задано, запускаем все этапы. Если задано - то только указанные.
 - **Публичный SSHключ для доступа в контейнер(ы)** - это отдельный ключ, не связанный с заданием "Внешний SSH доступ в контейнеры" из предыдущего раздела.
 - **Ключ отладки для Flask** - флаг отладочной работы (debug) для Flask приложения
 - Ограничения ресурсов - ограничения ресурсов для контейнеров в docker-compose.yml
 - **ОЗУ** - ограничьте доступную каждому из контейнеров ОЗУ до объема $100 + \text{НОМЕР_ВАРИАНТА} * 10 \text{ МБ}$
 - **Ядра процессора** - ограничьте доступные в каждом контейнере количество ядер ЦПУ до $(1 + \text{НОМЕР_ВАРИАНТА} \% 2)$ (остаток от деления номера вашего варианта на два)
 - **Максимальное Количество процессов** - ограничьте до количества НОМЕР_ВАРИАНТА

Варианты средней сложности

Вам необходимо реализовать docker-compose конфигурацию из двух узлов (не больше и не меньше):

- app - контейнер с существующим демонстрационным веб-приложением (https://github.com/moevm/devops-examples/tree/main/EXAMPLE_APP).
 - Устанавливать приложение необходимо скачивая репозиторий <https://github.com/moevm/devops-examples> и копируя файлы из него при сборке вашего контейнера:)
 - Чтобы все заработало, вам придется потратить время и поразбираться - из коробки может не работать.
 - Возможно, вам для выполнения заданий потребуются фиксы в исходник - делайте для них патчи (<https://man7.org/linux/man-pages/man1/patch.1.html>)
 - Корнем дерева процессов выступает запущенное веб-приложение
- tester - контейнер для запуска **всех** тестов (состав и особенности тестов задаются в таблице вариантов)
 - Корнем дерева процессов выступает стандартный python http сервер (python -m http.server 3000)
 - Этот сервер должен быть запущен в каталоге контейнера, где будет происходить работа тестовых скриптов
 - Тестовые скрипты запускаются через docker exec

Оба контейнера должны использовать написанные вами образы, собираемые из локальных Dockerfile. Шаблоны для имен Dockerfile:

- Dockerfile_app
- Dockerfile_tester

Помимо Dockerfile, вам также необходимо сделать файл README.md, содержащий примеры команд для запуска тестов и проверки всей конфигурации. Это сильно ускорит проверку:)

Параметры конфигурации задаются в таблице вариантов + общие требования (http://se.moevm.info/doku.php/courses:devops:project#общие_требования).

Варианты высокой сложности

1. Автоматизация тестирования курсовых по Android

Идея - разработать набор github actions, которые будут по состоянию репозитория проверять (базово) соответствие этапам выполнения работы и генерировать / отображать статус в readme.

Подробности об этапах:

https://se.moevm.info/doku.php/staff:courses:application_development_for_mobile_platforms:course_work:topics

Проверяем:

1. Макет и УС (есть вики страница, файл макета загружен в репо)
2. UI на заглушках (если задан нужный тег- проверяем наличие исходников андроид проекта, его собираемость через github actions, .gitignore , название пакета)
3. Юнит-тесты (тег, сборка и запуск)
4. App is ready (тег, требования, запуск, запуск стресстестов)
5. Оценка сложности UI вашего приложения (вики станица и ее содержимое)
6. Пояснительная записка (наличие файлов)
7. Интеграционные тесты (тег, сборка и запуск)

Используем в качестве технологий github actions. Сдаем отдельным репо.

2. Проверка корректности учебных работ на языке Python (командная строка)

Идея - автоматизировать процесс проверки лабораторных и курсовых работ.

Для проверки кода - pylint, для тестирования работы в командной строке <https://github.com/cucumber/aruba>, для профилирования работы - valgrind.

Формат выполнения github actions. Сдаем отдельным репо.

Этапы проверки

- Успешная проверка на явные синтаксические проблемы через линтер(не запустится, нет комментариев, неменяемые имена переменных....)
- Успешное тестирование работы на заранее известных примерах аргументов командной

строки (== приложение не падает с заранее известными аргументами)

- Стресс-тестирование аргументов командной строки (проверка, что если подавать почти рандомные аргументы, приложение не сломается. Аргументы необходимо генерировать рандомно, но в соответствии с описанием ожидаемой структуры)
- Стресс-тестирование stdin
- Профилирование работы по памяти и времени (valgrind + time)

3. Проверка корректности учебных работ на языке C (командная строка)

Аналогично теме 2, но компилируем в gcc (и проверяем что все ок с компиляцией) + другие линтеры.

Правила оценивания

Оценка за проект складывается из следующих критериев (пропорционально степени выполнения вашего варианта задания):

- (0-10 баллов) Требования к организации тестов из вашего варианта
- (0-10 баллов) Требования к работе в Docker
- (0-10 баллов) Требования к работе в составе Docker-Compose

Подробнее о процедуре проверки

https://docs.google.com/document/d/1iqn_jtQ7yP_jgMv1a-jw3qriWTVHDyAeEPAnQT51tl8/edit

Правила работы в репозитории

Все работы сдаются только и исключительно в виде PR в репозиторий курса. Подробно:

<https://github.com/moevm/devops-1h2024/blob/main/README.md>

По работе в selenium

Если у вас сложности:

- Примеры скриптов и тестов
https://github.com/moevm/devops-examples/tree/main/selenium_tests
- Курс <https://stepik.org/course/575>
- Старая, но актуальная лекция про Selenium <https://www.youtube.com/watch?v=gLj6BMml69I>

From:
<http://se.moevm.info/> - **se.moevm.info**

Permanent link:
<http://se.moevm.info/doku.php/courses:devops:project?rev=1708446625>

Last update: **2024/02/20 16:30**



