Docker и remote

Дмитрий Халанский

5 октября 2021 г.

1. Запуск других ОС в своей
   - Виртуальные машины
   - docker
2. git
   - remote
   - submodules
3. Что ещё поизучать?
   - Текстовые редакторы
   - Языковые инструменты
   - Makefile
   - Регулярные выражения

Виртуальные машины

- Виртуальные машины — программы, которые симулируют поведение другого компьютера.
- Зачем:
  - Работаете на одной ОС, хотите потестировать, как код работает на другой ОС...
  - ... или вообще на другой архитектуре (например, на ARM).
  - Нужна программа, которая работает только на другой ОС.
  - Хотите потрогать вирус, не давая ему доступ к своим личным файлам.
- Примеры: Android Emulator, iOS Simulator, VirtualBox, QEMU, DosBox.

Термины

Host (“хозяин”) — аппарат, в котором запущена виртуальная машина.

Guest (“гость”) — аппарат, симулирующийся виртуальной машиной.

Input capture (“захват ввода”) — неосторожный щелчок мышью внутри окна виртуальной машины иногда её “захватывает”; иногда и клавиатура тоже захвачена целиком, так что сочетания клавиш в host-системе перестают работать.

Snapshot (“слепок”) — сохранённое состояние виртуальной машины.

QEMU

QEMU — популярный механизм виртуализации.

- Производительный;
- Бесплатный для любых целей;
- Поддерживает много всяких гостевых и хостовых платформ;
- Нужно читать документацию, чтобы разобраться.

Лайфхак: в Линуксе можно запускать в терминале с флагом `-curses`.
См. `spice-vdagent` для более хорошей интеграции хоста с гостем.

VirtualBox

VirtualBox — разработанный Oracle механизм виртуализации.

- Настройка щёлканьем мышью;
- Бесплатная для личного использования;
- Есть специальные драйверы для Windows-гостя;
- Поддерживает x86 и x86_64 как гостей и хостов.
- Если не для личного использования, тщательно читайте лицензию.

KVM

KVM (Kernel-based Virtual Machine) — встроенный в Linux механизм виртуализации. QEMU обычно работает поверх него.

1. Запуск других ОС в своей
   - Виртуальные машины
   - docker
2. git
   - remote
   - submodules
3. Что ещё поизучать?
   - Текстовые редакторы
   - Языковые инструменты
   - Makefile
   - Регулярные выражения

Docker

Проблемы с поставкой программ:

- Программы зависят от библиотек. Иногда от конкретных версий.
- Иногда программы опираются на особенности ОС...
- ... или размещения файлов.

Как сделать установщик для программы так, чтобы она точно заработала или хотя бы сообщила, что не так?

- Строгие разработческие практики: отслеживание используемых зависимостей и предположений.
- Регулярная проверка работоспособности в разных конфигурациях.
- Автоматический анализ программ на предмет неявных зависимостей.

Docker

Проблемы с поставкой программ:

- Программы зависят от библиотек. Иногда от конкретных версий.
- Иногда программы опираются на особенности ОС...
- ... или размещения файлов.

Как сделать установщик для программы так, чтобы она точно заработала или хотя бы сообщила, что не так?

- Строгие разработческие практики: отслеживание используемых зависимостей и предположений.
- Регулярная проверка работоспособности в разных конфигурациях.
- Автоматический анализ программ на предмет неявных зависимостей.
- Или можно скопировать состояние компьютера, на котором программа точно идёт, и рассылать его. Это Docker.

Зачем нужен Docker

- Контейнеры не нужно долго настраивать, как виртуальные машины. Альтернатива: nix-shell.
- Контейнер можно удалить или создать одной командой. Альтернатива: nix-shell.
- Не надо мучиться при поисках/сборке библиотек. Альтернатива: пакетный менеджер.
- Можно не перенастраивать сервер, если сменились требования: обновил Docker-образ — всё заработало как надо. Альтернативы для этого: Puppet, Ansible, Chef или решения, встроенные в механизмы оркестрации типа Kubernetes.

Контейнеры

Container (контейнер) — легковесный аналог виртуальной машины.
`docker container` — команды для управления контейнерами.
`docker container ls --all` их перечисляет.
На каждый запуск сервиса Docker создаёт новый контейнер.
Пример: `docker container run -it archlinux bash` — запустить (`run`) новый контейнер, основанный на образе Arch Linux, с "сервисом" `bash` в терминале (`-t`), к которому мы тут же подключаемся интерактивно (`-i`).
Изменения, внесённые в *контейнере*, не сохраняются в *образе*. По соглашению, в контейнере не должно происходить интересных изменений, так как в каждом контейнере обычно запущен один сервис.

Образы

Image (образ) — состояние docker-контейнера. `docker image` — команды для управления образами. `docker image ls` их перечисляет. Существует коллекция docker-образов на любой вкус: https://hub.docker.com/. У образа есть название и тег. `docker pull название:тег` выкачивает образ локально.

Популярные образы

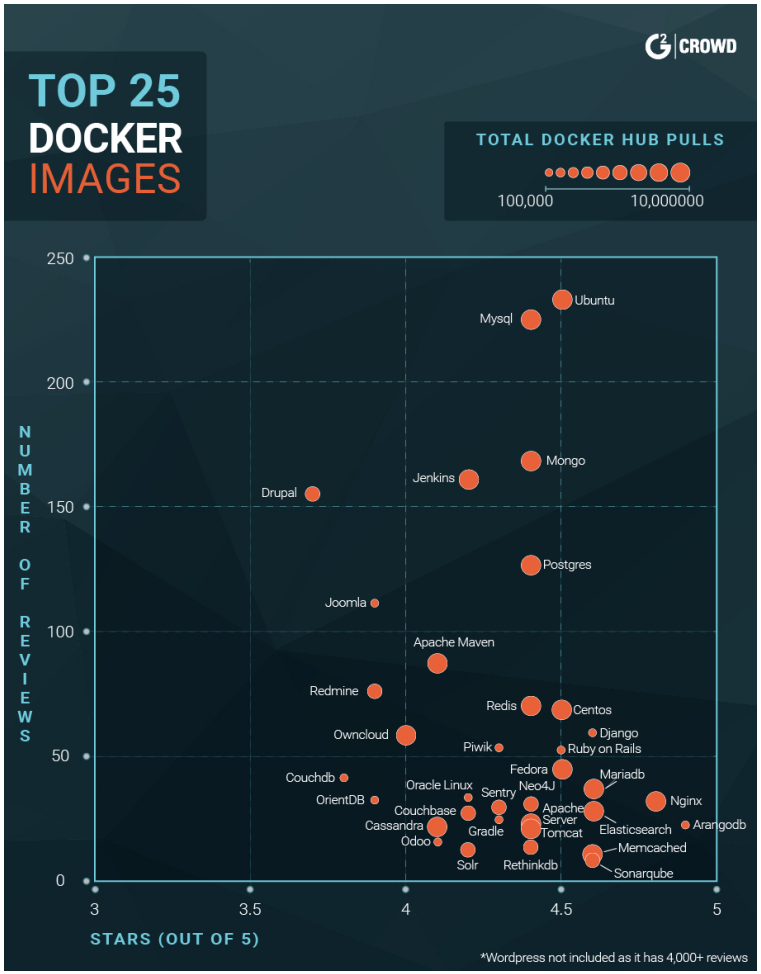

Рис.: https://learn.g2.com/best-docker-containers-repository

Полезные команды

- `docker container diff C` — перечислить изменения, внесённые контейнером *C*.
- `docker container commit C` — создать новый образ на основе состояния контейнера *C*.
- `docker container attach C` — подключить свой терминал к контейнеру *C*.
- `docker container exec C cmd` — выполнить команду `cmd` в контейнере *C*. Её stdin и stdout подключены к вызывающей консоли.
- `docker container create I cmd` — создать контейнер на основе образа *I*, выполняющий команду `cmd`.
- `docker run -it I` — интерактивно запустить команду по умолчанию в новом контейнере на основе *I*. Мораль №1: есть команда по умолчанию; мораль №2: в некоторых командах можно опускать `container` или `image`.

Сложный пример

```
docker run \
  --name some-nginx \
  -v /some/content:/usr/share/nginx/html:ro \
  -p 8080:80 \
  -d \
  nginx
```

Запустить новый контейнер на основе образа `nginx`, назвать его `some-nginx`, дать ему доступ на чтение (`ro`) к директории `/some/content/`, которую внутри контейнера будет видно по пути `/usr/share/nginx/html`, пробросить порт 8080 на хостовой машине на порт 80 на гостевой; запускать в фоне (`-d`).
Запустится веб-сервер nginx.
("\" ровно в конце строки в shell, C и других языках означает, что следующая строка считается продолжением данной)

Мусор

Каждый вызов `docker run` создаёт новый контейнер. Если их воспринимать как черновики, то место на диске можно сберечь, если пользоваться командой `docker container prune`, уничтожающей остановленные контейнеры.

Dockerfile

- Чтобы создать нетривиальный docker-образ, не стоит собирать его с нуля или вносить изменения руками и делать `docker commit`: так будет сложно понять, как воссоздать образ заново в случае потери или смены требований. Лучше воспользоваться `Dockerfile` — файлом, описывающим программно, как создать образ. См. https://docs.docker.com/engine/reference/builder/.
- Выполнение каждой команды в Dockerfile сгенерирует новый *слой* — набор изменений относительно предыдущего слоя. Если на шаге n произошла ошибка (или шаг n в Dockerfile изменился), шаги $[1; n-1]$ не будут выполняться заново.
- На самом деле, при создании контейнер заводит новый слой относительно своего образа, а `docker commit` превращает схлопывает этот слой и образ в новый образ.

Пример Dockerfile

```
FROM ubuntu:18.04
COPY poems /root/poems
ENV EDITOR vim
RUN apt-get update && apt-get install -y \
    git \
    vim \
    && rm -rf /var/lib/apt/lists/*
CMD $EDITOR /root/poems
```

- В качестве основы взять образ `ubuntu:18.04`;
- Скопировать файл `poems` по пути `/root/poems` в образе;
- Записать `vim` в переменную `EDITOR`;
- Установить `git` и `vim`;
- Назначить в качестве команды по умолчанию `vim /root/poems`.

1. Запуск других ОС в своей
   - Виртуальные машины
   - docker
2. git
   - remote
   - submodules
3. Что ещё поизучать?
   - Текстовые редакторы
   - Языковые инструменты
   - Makefile
   - Регулярные выражения

remote

Remote — другая копия репозитория, о которой знает данная.
Сведения о remote записываются в `.git/config`. Remote задаётся именем (оно не значимо) и URL.
См. `git-config(1)`, ищите там ключ `remote.<name>.url` и те, что под ним.

fetch

`git fetch` обновляет сведения о ветках и тегах в remote-репозиториях и выкачивает объекты, доступные из них.
Список веток хранится в `.git/refs/remotes/имя/`.
Чтобы увидеть не только ветки в локальном репозитории, но и в remote, нужно выполнить `git branch --all`.

Upstream (tracking)-ветка

См. `git-config(1)`, ключи:

- `branch.<name>.remote` — в каком репозитории хранится “главная” версия данной ветки;
- `branch.<name>.merge` — как называется соответствующая ветка в том репозитории.

Назначение веток:

- `git checkout X`, если ветки X нет в локальном репозитории, но есть в каком-то remote, создаст новую ветку X и назначит ей upstream.
- `git push -u Y X` назначит ветке X upstream-ветку X из remote Y.

pull и push

- `git pull` делает `git fetch`, а затем `git merge Y/X`, где Y/X — это upstream-ветка для данной.
- `git pull --rebase` делает `git fetch`, а затем `git rebase Y/X`.
- `git push` отсылает изменения в upstream и делает `git merge`, но на стороне upstream.
- `git push --force` отсылает изменения в upstream и делает `git reset` на стороне upstream.
- `git push --force-with-lease` отсылает изменения в upstream, сверяет состояние upstream с тем, которое выкачано локально, и делает `git reset` только в том случае, если upstream не менялся.

1. Запуск других ОС в своей
   - Виртуальные машины
   - docker
2. git
   - remote
   - submodules
3. Что ещё поизучать?
   - Текстовые редакторы
   - Языковые инструменты
   - Makefile
   - Регулярные выражения

git submodules

Submodules — механизм git, позволяющий отсылаться к содержимому других репозиториев (с другими проектами) из данного. Например, можно положить как submodule в свой репозиторий какую-нибудь библиотеку, исходники которой нужны для сборки.
Репозитории для submodule хранятся в `.git/modules`; описания submodule — в `.git/config`. В index и в tree-объектах хранится запись о директории submodule, где в качестве хэша указан хэш коммита в репозитории данного submodule. Список этих хэшей можно увидеть командой `git submodule`.
По умолчанию `git clone` не выкачивает submodule, из-за чего сборка может ломаться. Тогда можно вызвать `git submodule init` или сразу выкачивать с `git clone --recurse-submodules`.
См. `git-submodule(1)`.

1. Запуск других ОС в своей
   - Виртуальные машины
   - docker
2. git
   - remote
   - submodules
3. Что ещё поизучать?
   - Текстовые редакторы
   - Языковые инструменты
   - Makefile
   - Регулярные выражения

vim

- Сложно изучить. Если хочется:
  - Команда `vimtutor` обучает основам `vim` за вечер.
  - Книга “Practical Vim” доучивает остальному.
- Идеален, когда в редактировании текста много однотипной рутины: её можно автоматизировать.
- Есть плагины на любой вкус.
- Плагины пишутся на особом и неприятном языке vimscript.
- Многие программы поддерживают vim mode.
- vim или его младший брат vi есть почти на каждом сервере.

kak

Идейное продолжение `vim`.

- Изучить легче, чем `vim`.
- Понять *полностью* возможно своими силами.
- Во всём лучше, чем `vim`.
- Нет плагинов на любой вкус.
- Плагины пишутся на shell-скриптах, что не сильно лучше vimscript.
- Ни в каких других программах нет “`kak` mode”.

emacs

- На самом деле не редактор, а среда для исполнения программ на особом диалекте языка LISP, в которую встроен убогий редактор по умолчанию. Можно туда установить vim.
- Медленный на больших файлах.
- Медленный в принципе.
- Встроен тетрис.
- Тут можно жить: настроить веб-браузер, почтовый клиент, vim, IRC, программу для TODO, и будет единая среда, в которой всё на одном языке и легко настраивается.

1. Запуск других ОС в своей
   - Виртуальные машины
   - docker
2. git
   - remote
   - submodules
3. Что ещё поизучать?
   - Текстовые редакторы
   - Языковые инструменты
   - Makefile
   - Регулярные выражения

Отдельные инструменты

Linter Программа, сообщающая о стилистических проблемах и предупреждающая о возможных ошибках.

- Haskell `hlint`;
- shell `shellcheck`;
- Python Тысячи их; пусть будет `pylint`;
- Rust clippy;
- C++ `clang-tidy`...

Formatter Программа, автоматически выравнивающая код согласно установленному стилю. Не тратьте ни минуты на подсчёт пробелов!

- Haskell `brittany`;
- Python Тысячи их; пусть будет `YAPF`;
- Rust `rustfmt`;
- C++ `clang-format`...

Интеграция в редактор

Language Server Protocol Протокол, по которому редакторы могут общаться с “языковыми серверами” для подсвета ошибок в коде, вывода типов, форматирования и прочего. Поддерживается vim, emacs, kak, VS Code, IntelliJ IDEA, да и вообще всем. Серверы разного качества есть для всех значимых языков. Список серверов: https://microsoft.github.io/language-server-protocol/implementors/servers/.

Tree sitter Библиотека для обхода редакторами синтаксических деревьев.

IDE

IDE (Integrated Development Environment) — комбайнер, в котором всё легко:

- Легко проводить некоторые из возможных преобразований кода.
- Легко работать с проектами, включающими код на многих языках.
- Легко рисовать графические интерфейсы и настраивать базы данных.
- Легко устанавливать плагины.
- Легко породить плохо структурированный код, который можно читать только в IDE.
- Легко напороться на баг в IDE, из-за которого перестанет работать вообще всё.
- Легко перестать понимать, как проект собирается (и почему иногда не собирается).

Мораль: опытный программист пусть пользуется чем хочет, а остальным придётся в какой-то момент разобраться, как всё работает под капотом.

1. Запуск других ОС в своей
   - Виртуальные машины
   - docker
2. git
   - remote
   - submodules
3. Что ещё поизучать?
   - Текстовые редакторы
   - Языковые инструменты
   - Makefile
   - Регулярные выражения

make

`make` — древняя, но актуальная система сборки проектов. Команда `make` смотрит в содержимое файла `Makefile` в той же директории, и вызывает команды, которые нужны, чтобы собрать проект, трогая при этом как можно меньше файлов.
`Makefile` — набор правил, каждое из которых состоит из трёх частей:

- Чтобы собрать файл X...
- зависящий от файлов Y_1, Y_2, Y_3...
- надо выполнить команды C_1, C_2, C_3.

Если файл Y_2 изменился (что видно по времени его изменения в файловой системе), то `make` пересоберёт X.
`make` активно пользуются в быту. Это удобнее, чем скрипты, тем, что ничего лишнего не исполняется и всё работает быстро.
Туториал.

1. Запуск других ОС в своей
   - Виртуальные машины
   - docker
2. git
   - remote
   - submodules
3. Что ещё поизучать?
   - Текстовые редакторы
   - Языковые инструменты
   - Makefile
   - Регулярные выражения

Регулярные выражения

Вспомним glob-ы (пример: `abc*.tx?`).
Существует более мощная конструкция для схожих целей — *регулярные выражение* (*regular expressions*, regex). Их надо знать каждому.
Примеры использования:

- `grep` — (get `regular` expression `pattern`) — утилита, которая ищет во входе подстроки, подходящие под регулярное выражение.
- `git grep` — утилита, которая ищет в файлах репозитория строки, подходящие под регулярное выражение.

И тем, и тем можно пользоваться с флагом `-F`, чтобы паттерн был не регулярным выражением, а обычной строкой. `-i` — "искать, игнорируя регистр".
Существует много синтаксисов ("диалектов") регулярных выражений.
Следует знать хотя бы Perl-style и POSIX.