How to Kill a Computer

in a Funny Way

by Vyacheslav Davydov | (•) (•)
v.davydov@qsoft.ru

KEEP CALM

AND

~~DESTROY YOUR LAPTOP~~

Содержание

Предисловие

По настроению и вдохновению буду всё это дело стараться держать в актуальном состоянии, следи за обновлением CHANGELOG.md. Если у тебя какие-то идеи/предложения/исправления/пожелания, то шли их мне на почту **v.davydov@qsoft.ru**, с удовольствием присмотрюсь к ним!

Есть вероятность, что могут возникнуть проблемы, если у тебя мак или винда. В таком случае… переходи на линукс) Ладно, конечно же тоже пиши на почту, будем разбираться вместе.

Ссылка на папку с необходимыми данными: **https://my.qsoft.ru/bitrix/tools/disk/focus.php?folderId=216092&action=openFolderList&ncc=1**

Помни, ты всё делаешь на свой страх и риск! Если у тебя уже есть худо-бедно поднятые сервисы и ты не готов к тому, что всё может поломаться, тогда поднятие сервисов лучше отложить до более подходящего момента!

Что необходимо?

1. docker
2. docker compose
3. утилита make
4. пару часов времени
5. хорошее настроение!!!

Какие контейнеры поднимаются?

- api-gateway
- attribute-service
- a-plus-service
- cart-service
- clickhouse-analytics-service
- docs
- feed-service
- front-service
- idm-service
- idm-service-consumer
- import-prepare
- import-deploy
- master-service
- media-service
- mindbox-service
- mindbox-service-consumer
- mobile-service
- notification-service
- notification-service-consumer
- order-service
- order-service-consumer
- redirect-service — поднимается, но не работает, надо дорабатывать
- retail-service
- saga-service
- search-service
- search-service-consumer
- search-service-elastic-worker
- search-service-create-indexing-worker
- search-service-broken-indexes-cleaner

- site-service
- stock-service
- user-service
- user-service-consumer
- wms-service
- autoheal — для перезапуска unhealthy или упавших контейнеров
- clickhouse
- elasticsearch
- kibana
- minio
- mongo
- nginx
- postgres
- rabbitmq
- кластерный Redis
  - redis-node-0
  - redis-node-1
  - redis-node-2
  - redis-node-3
  - redis-node-4
  - redis-node-5
- *список будет дополняться...*

Поехали!

1. Формируем структуру проекта. Чтобы ничего не поломать, лучше делать это в новой папке. Итак:

   1.1. Создаём папку «*farm-new-site*».

   1.2. Разархивируем в неё папку «_» из вложения.

   1.3. Заходим в настройки IDE и исключаем из workspace следующие папки:

   - ✓ ./_/api
   - ✓ ./_/data
   - ✓ ./_/db
   - ✓ ./_/dependencies
   - ✓ ./_/minio

   Делаем это для того, чтобы эти файлы лишний раз не анализировались и чтобы на них не тратились ресурсы. Если у тебя «**VS Code**» тогда просто скопируй из вложения папку «.*vscode*» с настройками в корень проекта.

   1.4. Заходим в настройки IDE и правим рабочую директорию для «*eslint*» («*front-service*») — делаем её равной «*./services/front-service*». Если у тебя «**VS Code**» тогда просто скопируй из вложения папку «.*vscode*» с настройками в корень проекта.

   1.5. Внутри «*farm-new-site*» создаём папку «*logs*» — в этой папке будет в удобный доступ к логам всех Laravel сервисов (через символьные ссылки).

   1.6. Внутри «*farm-new-site*» создаём папку «*packages*». Клонируем из репозитория следующие пакеты:

   - ✓ event-dispatcher-package
   - ✓ firebase-client-package
   - ✓ foundation-package
   - ✓ mindbox-sdk-package
   - ✓ mindbox-service-client-package
   - ✓ redis-ref-cleaner-package
   - ✓ service-dto-package
   - ✓ stock-service-client-packag
   - ✓ user-service-client-package
   - ✓ *список будет дополняться...*

   1.7. Внутри «*farm-new-site*» создаём папку «*services*». Клонируем из репозитория следующие сервисы:

- ✓ attribute-service
- ✓ a-plus-service
- ✓ cart-service
- ✓ feed-service
- ✓ idm-service
- ✓ master-service
- ✓ media-service
- ✓ mindbox-service
- ✓ mobile-service
- ✓ notification-service
- ✓ order-service
- ✓ retail-service
- ✓ saga-service
- ✓ search-service
- ✓ site-service
- ✓ user-service
- ✓ wms-service
- ✓ api-gateway
- ✓ clickhouse-analytics-service
- ✓ import-service
- ✓ stock-service
- ✓ front-service
- ✓ redirect-service
- ✓ *список будет дополняться…*

Обращаю внимание, что после клонирования менять докер файлы или конфиги для рэдиса/эластики не нужно!

1.8. Внутрь папки «*farm-new-site*» клонируем документацию из репозитория (**https://gitlab.qsoft.ru/farm-new-site/docs**).

1.9. В папку «*farm-new-site*» копируем «*Makefile*» и «*docker-compose.yml*» из вложения. По желанию можно ещё скопировать файл «*.editorconfig*» — для форматирования файлов. Только чтобы он заработал, необходимо, чтобы твоя IDE его поддерживала. Для «**VS Code**» нужно установить расширение **EditorConfig for VS Code**.

1.10. В итоге должна получиться такая структура:

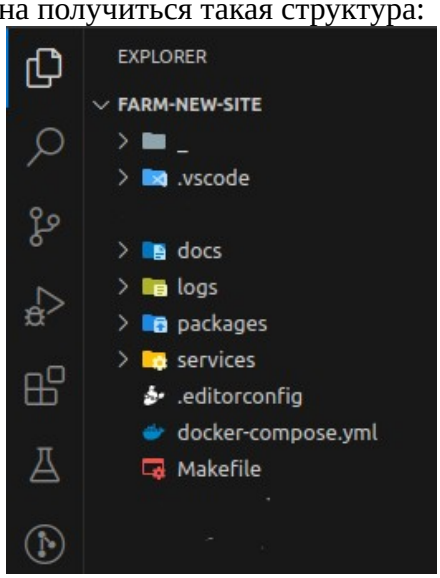

Рис. 1: Структура папок

1.11. Приступаем непосредственно к разворачиванию проекта. Контейнеров много, так что ресурсов понадобится тоже немало, на всякий случай закрой важные программы и сохрани все документы, чтобы ничего не потерять. А вообще по хорошему, желательно в принципе закрыть все программы. Готов? Точно? Обратного пути не будет… Ну хорошо, как скажешь. Открывай в IDE папку «*farm-new-site*», переходи в консоль и запускай команду «*make init*».

1.12. Компьютер минут 15 минимум будет готов взлететь, иногда поглядывай, что там нет никаких ошибок. А пока процесс идёт, начинай ознакамливаться с разделами «*Пару слов о волшебной папке « »*, «*Docker и docker compose*» «*Работа с Makefile*» и «*Тонкости при локальной работе*».

Пару слов о волшебной папке «_»

Что полезного есть в данной папке:

1. **api** — тут присутствуют некоторые API запросы. По идее должны импортироваться в Postman, но это не факт — вообще это выгрузки из дополнения «**Thunder Client**» для «**VS Code**».
2. **data** — системная папка. В ней будут хранится данные контейнеров, чтобы при их удалении ничего не пропадало (например, чтобы не удалялись БД в «*postgres*», очереди в «*rabbit*» и т. д. при удалении контейнера). Подробнее об этом в «*Docker и docker compose*».
3. **dependencies** — чтобы проект быстро инициализировался при первом запуске, для твоего удобства выгружены зависимости «*composer*», «*node*» и «*go*», которые при инициализации просто копируются в необходимый сервис
4. **db** — чтобы проект не был девственно чистым, для твоего удобства подготовлены некоторые выгрузки БД, чтобы сайт корректно завёлся. Также в данной папке присутствует импорт, который нужно будет прогнать для деплоя цен, остатков, а также для актуализации локальной эластики.
5. **deployment** — «*nginx*» + докер файлы для сервисов. Deployment сделан максимально приближенным к боевому! Из отличий:
   - в nginx убрано кэширование, также подправлены заголовки «*Host*», убрано использование SSL
   - в сервисах исходный код не копируется в контейнер, а монтируется через «*volumes*» в «*docker-compose.yml*», чтобы при разработке изменения в файлах на хосте сразу синхронизировались с контейнером, иначе пришлось бы каждый раз при внесении изменений в код перезапуск контейнер
   - из-за того, что на этапе билда контейнеров в них по сути отсутствует исходный код, то из докер файла сервисов пришлось выкинуть установку зависимостей («*composer*», «*npm*» и «*pkg*») — подробнее в «*Тонкости при локальной работе*»
6. **docker_images** — образы для докера. Почти все образы можно без проблем скачать с docker hub, есть только 1 проблемный — **unit-php**. Это типо наш внутренний образ компании, но доступа на его скачивание у простых смертных нет, именно по этой причине падал билд контейнеров. Его я собрал отдельно и выгрузил, чтобы потом его импортировать в докер (docker load). С остальными образами решил поступить также — всё-таки удобнее и правильнее, когда они уже есть и нет нужны рассчитывать на стороннего посредника в лице docker hub — вдруг с ним что-то произойдёт или разработчики какого-либо образа решат его удалить оттуда.
7. **env** — будет очень неожиданно, но в этой папке содержатся env значения для сервисов. Есть один момент, подробнее в «*Тонкости при локальной работе*».

8. **minio** — аналогично с БД — выгрузил некоторые файлы для наполнения сайта. Хочу обратить внимание, что сайт будет смотреть на локальный минио, а не на боевой, как это сейчас реализовано на тестовых хостах!
9. **secrets** — это просто секретные файлы, которые монтируются в контейнер. Подробнее в «*Docker и docker compose*».

Docker и docker compose

Как уже упоминалось, проекты собираются из докер файлов, лежащих в папке «*_/deployment*», так что теперь нет нужды локально вносить изменения в докер файл сервиса, а затем следить за тем, чтобы случайно его не закоммитить. Сами образы тоже максимально приближены к боевым. В принципе про докер файлы мне больше сказать нечего — момент с установкой зависимостей мы обсудим в «*Тонкости при локальной работе*».

Так что переходим к «*docker-compose.yml*». На очевидных моментах останавливаться не буду, с ними можно ознакомиться самостоятельно. Расскажу немного про интересные штучки дрючки:

- **deploy → resources → limit** — можно задавать лимиты по ресурсам для контейнеров. Хочу обратить внимание, что здесь это именно лимит, а не ресурсы, которые резервируются. Т.е. лимит «memory: 1024M» НЕ означает, что под контейнер зарезервировалось 1Гб оперативы. Это значит, что контейнер не сможет скушать более 1Гб. В целом я выставил средние значения, по идее сильно играться ими не стоит. Если вдруг комп прям не тянет, тогда ок, можно уменьшить ресурсы — начать следует с «*api-gateway*», «*nginx*», «*redis*», затем «*php-services*», «*front-service*». Эластику, рэбит и постгрес менять прям если остальное не помогло, т. к. они капризные и могут не завестись. Но тут тоже надо иметь в виду, что при ограничениях сервисы будут работать медленнее и из-за этого сайт не будет функционировать так, как ожидается. Например, попытка регистрации может не пройти, т. к. «*idm-service*» будет очень долго отвечать «*user-service*» и запрос будет обрываться по таймауту. В общем тут надо быть аккуратнее. В целом ограничивать можно память, а также CPU. С процессором там немного хитрая система, работает она немного по-особенному — если у тебя 4 ядра, то тогда получается, что total cpus у тебя равен 4. И далее каждому сервису, ты можешь выдавать определённую частичку, например можно задать лимит «cpus: 0.05». Т.е. параметр cpus отвечает не за ядра, а за процессорное время, которое выделяется под сервис. Подробнее можно прочитать здесь: «**Runtime options with Memory, CPUs, and GPUs**»
- **deploy → replicas** — отвечает за кол-во копий контейнеров. Данная опция будет полезна для индексаций — сейчас по-умолчанию сделаны 3 активных региона (16,63,77), однако у «*search-service-elastic-worker*» «*replicas=1*». Если необходимо, чтобы индексация шла быстрее, тогда можно увеличить replicas и тогда она будет идти параллельно. Но тут тоже надо быть осторожными — может не выдержать и отвалиться сама эластика
- **restart** — автоматический перезапуск контейнеров, если вдруг они упали. Есть опция «*always*», но тогда они будут перезапускаться, например при включении компьютера. Нам же больше подходит опция «*unless-stopped*» — в этом случае, если мы сами остановили контейнер или демон докера, то контейнеры перезапускаться не будут, они сделают рестарт только если упадут сами, например из-за какого-то исключения в коде

- **healthcheck** — проверка «здоровья» контейнера. Именно по работе «*healthcheck*» предлагаю ознакомиться со статьёй «**How To Successfully Implement A Healthcheck In Docker Compose**». Также дополнительно хочу отметить, что у нас есть контейнер «**autoheal**», который мониторит наши контейнеры и перезапускает их, если они отмечены как «*unhealthy*»

- **depends_on** — список зависимостей сервиса. При этом, если зависимость помечена как «*service_started*», тогда главный сервис запустится сразу после запуска зависимости. Но зачастую такое поведение не очень корректное — например, консьюмеру нужен рэбит. Если он запустится сразу после старта рэбита, то консьюмер упадёт с ошибкой, т.к рэбиту нужно около 30 сек чтобы инициализироваться и быть готовым к работе. Поэтому более подходящим вариантом будет использование «*service_healthy*» — главный сервис ожидает до тех пор, пока зависимость не будет помечена как «*healthy*» и только после этого уже запускается

- **secrets** — данная секция используется для монтирования в контейнер секретных файлов. Работает это по типу env, только вместо пары ключ/значение монтируется именно файл в необходимую нам директорию. Например, для «*idm-service*» монтируются секретные ключи, а для «*notification-service*» — секретный ключ для доступа к API Google Firebase. Также монтировать можно не только именно секретные файлы, а просто файлы, которые не хранятся в репе, но необходимы для работы. Например, для эластики монтируется файл «*synonyms.txt*», а для PHP сервисов — «*composer.phar*», чтобы он был необходимой для сервиса версии (о работе с композером подробнее в «*Тонкости при локальной работе*»)

- **volumes** — монтирование директорий. При помощи данной секции монтируем исходный код, чтобы во время разработки при модификации файлов изменения сразу синхронизировались с файловой системой контейнера. Также при помощи этой штуки мы храним данные некоторых контейнеров в «*_/data*», что позволяет спокойно удалять или ребилдить контейнеры не переживая за то, что потеряется какая-либо важная информация. Т.е. в нашем случае контейнер — это просто сущность для выполнения каких-либо действий. Эта сущность может оперировать какими-либо данными, но мы сознательно отделяем эти 2 вещи и не храним данные непосредственно в контейнере, т. к. это не есть хорошо

- **command** — некоторые контейнеры (например PHP-консьюмеры или Go/Node сервисы) имеют данную секцию. По сути это просто команда, которая начинает выполняться при запуске контейнера

- **user** — имя пользователя, от которого работает контейнер. Например, это нужно для «*composer*» («*user: root*»), иначе он не может получить доступ к файловой системе, и как следствие установить зависимости

- **hostname** — необходимо для рэбита («*hostname: localhost*»). Если не указать, тогда при удалении контейнера все данные рэбита потеряются. Точнее не так, они не потеряются, просто рэбит их не возьмёт — проблема в том, что рэбит на каждую новую сессию генерирует рандомное имя хоста и получается так, что при перезапуске

он «думает», что он на новом хосте и не подтягивает старые данные. Приходится явно указывать «*hostname*», чтобы каждый раз при запуске он был один и тот же.

Работа с Makefile

Наверное самый полезный инструмент, который здорово будет упрощать жизнь. Команды:

- **help** (или просто make) — вывод списка всех команд и их краткого описания
- **motivation** — более 400 отборных цитат для поднятия настроения и мотивации на работу. Обязательно начинай свой день с нескольких цитаток и всё у тебя будет чики брики!!!
- **start** — запуск запуск сервиса/сервисов. Имейте в виду, что перед запуском команда очищает все логи в Laravel сервисах, чтобы случайно не захламить диск (например, очень много логов на «**search-service**», особенно при индексации). Контейнеры запускаются с ожиданием зависимостей.
  - *make start* — запуск всех контейнеров, перечисленных в «*docker-compose.yml*» (кроме тех, у которых «*scale=0*», например импорт, очистка битых индексов и кибана, подробнее в «*Тонкости при локальной работе*»)
  - *make start c=idm-service* — запуск конкретного контейнера (с зависимостями)
  - *make start c='idm-service master-service'* — запуск нескольких контейнеров (с зависимостями). Здесь важно обернуть список сервисов в строку, а также отделять их друг от друга пробелом
- **stop** — остановка сервиса/сервисов. Обращаю внимание, что остановка происходит не через «*docker compose stop*», а через «*docker compose down*», т.о. не надо пугаться — это корректно, что удаляются все контейнеры + общая сеть
  - *make stop* — остановка всех контейнеров, перечисленных в «*docker-compose.yml*»
  - *make stop c=idm-service* — остановка конкретного контейнера (без зависимостей, они продолжают работать)
  - *make stop c='idm-service master-service'* — остановка нескольких контейнеров (без зависимостей, они продолжают работать). Здесь важно обернуть список сервисов в строку, а также отделять их друг от друга пробелом
- **restart** — перезапуск сервиса/сервисов. Выполняется команда «*stop*», а потом «*start*»
  - *make restart* — перезапуск всех контейнеров, перечисленных в docker-compose.yml
  - *make restart c=idm-service* — перезапуск конкретного контейнера (зависимости при этом не перезапускаются)
  - *make restart c='idm-service master-service'* — перезапуск нескольких контейнеров (зависимости при этом не перезапускаются). Здесь важно обернуть список сервисов в строку, а также отделять их друг от друга пробелом
- **build** — билд сервиса/сервисов. На всякий случай сначала происходит загрузка всех докер образов из «*_/docker_images*» и только потом уже билд

    - *make build* — билд всех контейнеров, перечисленных в docker-compose.yml
    - *make build c=idm-service* — билд конкретного контейнера
    - *make build c='idm-service master-service'* — билд нескольких контейнеров. Здесь важно обернуть список сервисов в строку, а также отделять их друг от друга пробелом
- **rebuild** — билд сервиса/сервисов. Последовательное выполнение команд «*stop*», «*build*», «*start*».
    - *make rebuild* — билд всех контейнеров, перечисленных в docker-compose.yml
    - *make rebuild c=idm-service* — билд конкретного контейнера
    - *make rebuild c='idm-service master-service'* — билд нескольких контейнеров. Здесь важно обернуть список сервисов в строку, а также отделять их друг от друга пробелом
- **start-laravel-services** — запуск всех Laravel сервисов (вместе с зависимостями)
- **stop-laravel-services** — остановка всех Laravel сервисов (без зависимостей)
- **restart-laravel-services** — перезапуск всех Laravel сервисов (без зависимостей). Последовательное выполнение команд «*stop-laravel-services*», «*start-laravel-services*»
- **start-go-services** — запуск всех Go сервисов (вместе с зависимостями)
- **stop-go-services** — остановка всех Go сервисов (без зависимостей)
- **restart-go-services** — перезапуск всех Go сервисов (без зависимостей). Последовательное выполнение команд «*stop-go-services*», «*start-go-services*»
- **start-backend** — запуск бэкенда: запуск Laravel сервисов, Go сервисов, «*redirect-service*», а также их зависимостей
- **stop-backend** — остановка бэкенда
- **restart-backend** — перезапуск бэкенда. Последовательное выполнение команд stop-backend, start-backend
- **composer** — установка composer зависимостей в Laravel сервисе/сервисах. composer монтируется в «*/tmp/composer.phar*» (через секреты docker-compose) и затем он доступен для вызова изнутри контейнера. Подробнее в «*Тонкости при локальной работе*»
    - *make composer* — установка зависимостей во всех Laravel сервисах
    - *make composer c=idm-service* — установка зависимостей в конкретном Laravel сервисе
    - *make composer c='idm-service master-service'* — установка зависимостей в нескольких Laravel сервисах. Здесь важно обернуть список сервисов в строку, а также отделять их друг от друга пробелом

- **migrate** — проведение миграций в Laravel сервисе/сервисах. На «*a-plus-service*» будет выкидываться ошибка — это нормально, у нас нет своей БД для этого сервиса
  - *make migrate* — проведение миграций во всех Laravel сервисах
  - *make migrate c=idm-service* — проведение миграций в конкретном Laravel сервисе
  - *make migrate c='idm-service master-service'* — проведение миграций в нескольких Laravel сервисах. Здесь важно обернуть список сервисов в строку, а также отделять их друг от друга пробелом
- **optimize** — сброс оптимизации Laravel сервиса/сервисов (выполнение команды «*php artisan optimize:clear*»).
  - *make optimize* — сброс оптимизации во всех Laravel сервисах
  - *make optimize c=idm-service* — сброс оптимизации в конкретном Laravel сервисе
  - *make optimize c='idm-service master-service'* — сброс оптимизации в нескольких Laravel сервисах. Здесь важно обернуть список сервисов в строку, а также отделять их друг от друга пробелом
- **queues** — привязка очередей в Laravel сервисе/сервисах (выполнение команды «*php artisan event-dispatcher:queue-binding*»). В некоторых сервисах могут вылетать ошибки — это корректно, очереди используются не во всех сервисах
  - *make queues* — привязка очередей во всех Laravel сервисах
  - *make queues c=idm-service* — привязка очередей в конкретном Laravel сервисе
  - *make queues c='idm-service master-service'* — привязка очередей в нескольких Laravel сервисах. Здесь важно обернуть список сервисов в строку, а также отделять их друг от друга пробелом
- **master** — переключение на ветку «*master*» и получение изменений из удалённого репозитория. Работает для всех сервисов, пакетов и доки. Обращаю внимание, что команда не делает stash, т.о. если есть какие-то локальные изменения, которые будут конфликтовать во время актуализации, то они так и останутся неразрешёнными и переключения на «*master*» в этом конфликтном сервисе не произойдёт. Также для этой команды важна структура директорий — если ты вдруг захочешь, например, документацию, переместить в папку с пакетами, то команда сломается
  - *make master* — переключение на ветку master и получение изменений из удалённого репозитория абсолютно везде
  - *make master c=packages/firebase-client-package* — переключение на ветку master и получение изменений из удалённого репозитория в конкретном сервисе/пакете. Обрати внимание, что т. к. это универсальная команда, то свой отсчёт она ведёт от корневой директории «*farm-new-site*», т.о. мало будет просто указать название сервиса/пакета, необходимо также явно указывать папки, относительно корневой

- *make master c='services/idm-service docs'* — переключение на ветку «*master*» и получение изменений из удалённого репозитория в конкретных сервисах/пакетах. Обрати внимание, что т. к. это универсальная команда, то свой отсчёт она ведёт от корневой директории «*farm-new-site*», т.о. мало будет просто указать название сервиса/пакета, необходимо также явно указывать папки, относительно корневой

- **clear-logs** — очистка логов в Laravel сервисах (файл «*storage/logs/laravel.log*»)
- **clear-front-cache** — очистка Nuxt кэша (команда «*npx nuxi cleanup*»)
- **clear-laravel-cache** — очистка пользовательского кэша BO BCEX Laravel сервисах («*php artisan cache:clear*»)
- **clear-go-cache** — очистка Go кэша, используемого при билде приложения. Go — компилируемый язык, перед работой ему нужно сбилдить приложение. Однако билд на Go реализован по хитрому — он кэширует некоторые части программы и если они не изменялись, то берёт из кэша (чем-то напоминает докер). Директория для кэша внутри контейнера Go — «*/root/.cache*». Эту директорию мы тоже монтируем в «*docker-compose.yml*», т.о. кэш сборки Go приложения хранится в «*_/data/go*». Если не прокидывать, тогда получается, что при каждом запуске контейнера Go будет билдить приложение каждый раз с чистого листа, это может растянуться надолго (у меня занимало около 5 минут). Так что дабы не разнести экран монитора, нужно обязательно использовать кэш, т. к. тогда сборка занимает до 10 сек. Проблема только в том, Go не подчищает за собой, поэтому этот кэш может раздуваться, следовательно нам нужно ручками иногда его подчищать
- **clear-broken-indexes** — очистка индексов без алиаса в эластике
- **kibana** — запуск кибаны для эластики
- **import-prepare** — запуск этапа импорта «*prepare*». Подробнее в «*Тонкости при локальной работе*»
- **import-deploy** — запуск этапа импорта «*deploy*». Подробнее в «*Тонкости при локальной работе*»
- **db-dump** — формирование дампа БД при помощи «*pg_dump*». Дампы выгружаются в папку «*_/db*»
  - *make db-dump* — формирование дампа БД всех сервисов
  - *make db-dump d=idm_data* — формирование дампа БД. В данном примере на выходе будет файл дампа «*_/db/idm_data.tar*». Обращаю внимание, что в качестве названия нужно указывать название БД, а не название сервиса
  - *make db-dump d='idm-service master-service'* — формирование дампов нескольких БД. На выходе в данном примере будут файлы дампов «*_/db/idm_data.tar*» и «*_/db/master_data.tar*». Обращаю внимание, что в качестве названия нужно указывать название БД, а не название сервиса

- **db-restore** — восстановление БД из дампа при помощи «*pg_restore*». Чтобы команда отработала корректно, необходимо файлы с дампами поместить в «*_/db*»
  - *make db-restore* — восстановление БД из дампа во всех сервисах
  - *make db-restore d=idm_data* — восстановление БД из дампа. В данном примере на необходимо наличие файла «*_/db/idm_data.tar*». Обращаю внимание, что в качестве названия нужно указывать название БД, а не название сервиса
  - *make db-restore d='idm_data master_data'* — формирование дампов нескольких БД. В данном примере нам необходимо наличие файлов «*_/db/idm_data.tar*» и «*_/db/master_data.tar*». Обращаю внимание, что в качестве названия нужно указывать название БД, а не название сервиса
- **db-vacuum** — высвобождение свободного места на диске. Специфика работы постгреса заключается в том, что при удалении записей из таблиц, они на самом деле физически не удаляются. Т.о. место на диске он не высвобождает и для системы оно по по-прежнему является занятым. Чтобы это поправить, необходимо ручками это дело подчищать. Подробнее можно почитать здесь: **VACUUM**.
- **dep-dump** — формирование дампа зависимостей сервисов — просто копирование папок «*vendor*», «*node_mobules*» и «*pkg*» в папку «*_/dependencies*»
- **dep-restore** — восстановление зависимостей сервисов — просто копирование папок «*vendor*», «*node_mobules*» и «*pkg*» из папки «*_/dependencies*» в папки с соответствующими сервисами
- **init** — инициализация проекта. Что происходит:
  - остановка всех запущенных контейнеров
  - билд контейнеров
  - переключение на ветку «*master*» и получение актуальных изменений из удалённого репозитория
  - восстановление «*composer*», «*node*» и«*go*» зависимостей в сервисах (их перенос из папки «*_/dependencies*»)
  - очистка кэша Go
  - очистка кэша Nuxt
  - очистка Laravel логов
  - нормальный старт всех сервисов со всеми зависимостями
  - привязка очередей
  - восстановление БД из дампа
  - высвобождение свободного места («*vacuum*») БД
- **refresh** — переустановка проекта с «чистого» листа. Что происходит:

- остановка всех запущенных контейнеров
- билд контейнеров
- переключение на ветку «*master*» и получение актуальных изменений из удалённого репозитория
- установка «*composer*» зависимостей
- очистка кэша Go
- очистка кэша Nuxt
- очистка Laravel логов
- нормальный старт всех сервисов со всеми зависимостями
- привязка очередей
- восстановление БД из дампа
- высвобождение свободного места («*vacuum*») БД

Данной командой удобно пользоваться перед началом работы над задачей — можно быть уверенным, что всё будет актуализировано от мастера

Тонкости при локальной работе

- env значения подключаются в «*docker-compose*» из файлов в папке «*_/env*» — сначала подключается файл «*_common.env*», а затем уже все остальные. Важно учитывать то, что при изменении env, контейнер необходимо перезапускать, иначе изменения не применятся.
- Сайт смотрит на локальный минио, поэтому если, например, нужно для коллекции или товара добавить фоточку, то загружать файлик надо именно в локальный минио
- чтобы запустить индексацию без запуска полного импорта, можно тригернуть событие окончания индексации через RabbitMQ. Для этого необходимо прейти на вкладку «*Exchanges*», затем в таблице выбрать «*application*» и опубликовать следующее сообщение:
  - Routing key: *direct.search-service.update-elastic-products*
  - Payload: {"data": *{"productIds":null,"isDelta":false,"packageName":"21567_20240511031305"},"from": "external"}*

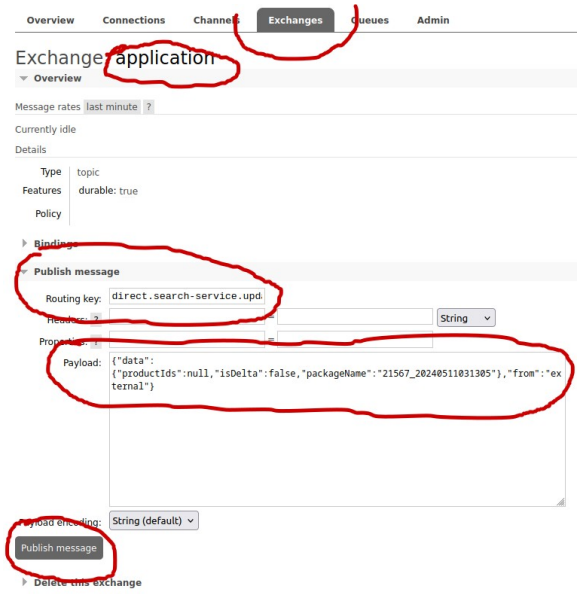

Рис. 2: Ручной запуск полной индексации

- чтобы индексации шла параллельно, нужно увеличить кол-во экземпляров «*search-service-elastic-worker*» через replicas (подробнее «*Docker и docker compose*»)
- если вдруг необходимо работать с composer, то обращаться к нему следует изнутри контейнера таким образом: «*php /tmp/composer.phar*». Пару примеров:
  - *docker exec -it notification-service php /tmp/composer.phar require farma-packages/firebase-client-package*
  - *docker exec -it idm-service bash && php /tmp/composer.phar i*

- документация поднимается через «*npm run watch*», т.о. можно править код и изменения сразу будут применяться
- фронт поднимата по команде «*npm i && npm run dev*» (см. «*docker-compose.yml*»). Т.о. изменения в коде применяются сразу, однако если нужно установить какую-либо новую зависимость, то придётся делать рестарт контейнера
- «*stock-service*», «*import-service*» и «*clickhouse-analytics-service*» — сервисы на Go => при каждом изменении кода их необходимо перезапускать. Эти сервисы запускаются через команду «*go run*». а исходный код монтируется через «*volumes*» в docker-compose.yml. А вот с «*api-gateway*» так не прокатит — несмотря на то, что это тоже гошный сервис, его необходимо ребилдить, т. к. исходный код вставляется в непосредсвенно в образ при его билде
- импорт — прогоняется 2мя отдельными командами: «*make import-prepare*» и «*make import-deploy*». При их выполнении поднимается контейнер, производятся необходимые манипуляции и затем контейнер гасится. Что хочу здесь отметить:
  - на dev окружении импорт надо закидывать в бакет «*import*» папки «*test*» (а на бою папка называется «*first*») — хадкод, тут ничего не поделаешь
  - на dev окружении импорты после этапа «*prepare*» не перемещаются в папку «*processed*» — опять же, хардкод, видимо это сделано для ускорения отладки кода, чтобы не заморачиваться в постоянной загрузкой файла в минио
- чтобы запустить генерацию фидов без импорта и индексации, можно тригернуть событие окончания индексации через RabbitMQ. Для этого необходимо прейти на вкладку «*Exchanges*», затем в таблице выбрать «*application*» и опубликовать следующее сообщение:
  - Routing key: *direct.clickhouse-analytics-service.indexing-completed*
  - Payload: *{"data":{"status":"success","isDelta":false,"packageName":"666"}}*
- *список будет дополняться...*

После установки

1. Открываем файл «*services/api-gateway/flexible/settings/config.json*». Изменяем 4ю строку — «*cache_ttl*» ставим «*0s*». Благодаря этому при разработке не будет приколюх с кэшированием на шлюзе (**HTTP Server Settings**). Делаем ребилд сервиса
2. Все сервисы будут крутиться в контейнерах, т.о. нет нужны локально что-либо дополнительно устанавливать. Однако, если ты пользуешь «**VS Code**», то возможно будут проблемы с подсветкой синтаксиса, а также с «проваливанием» внутрь функций. Для решения этой проблемы локально придётся установить «**Go**», «**Node.js и NPM**». Для PHP можно использовать следующее расширения:
    - «**PHP Intelephense**» — простое, лёгкое, но не всегда во все функции можно «проваливаться»
    - «**PHP DEVSENSE**» — помощнее предыдущего, но была проблема, что слишком много оперативы жрал

Для докера можно установить такое расширение — «**Docker**». Помогает хотя бы чисто визуально понимать что происходит с контейнерами, также легко можно их перезапускать, смотреть логи и т.д.

3. Настройка Minio. Переходим в минио («*Доступы*»). Создаём бакеты:
    - feed-data
    - import
    - master-data
    - media-data
    - mindbox-data
    - retail-data
    - site-data
    - test — этот бакет нужен для импорта синонимов, к сожалению, на импорте на dev окружении название этого бакета захардкожено и нельзя поменять

Для всех бакетов выставляем публичный доступ.

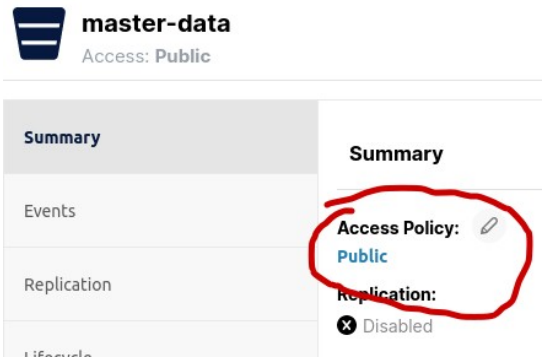

Рис. 3: Настройка доступа к бакетам

Далее загружаем в бакеты файлы из сооветствующих директорий «*_/minio*». Quick tip — выбираем «*Upload folder*» и далее в диалоговом окне прям заходим в эту папку — тогда минио загрузит файлы как и полагается.

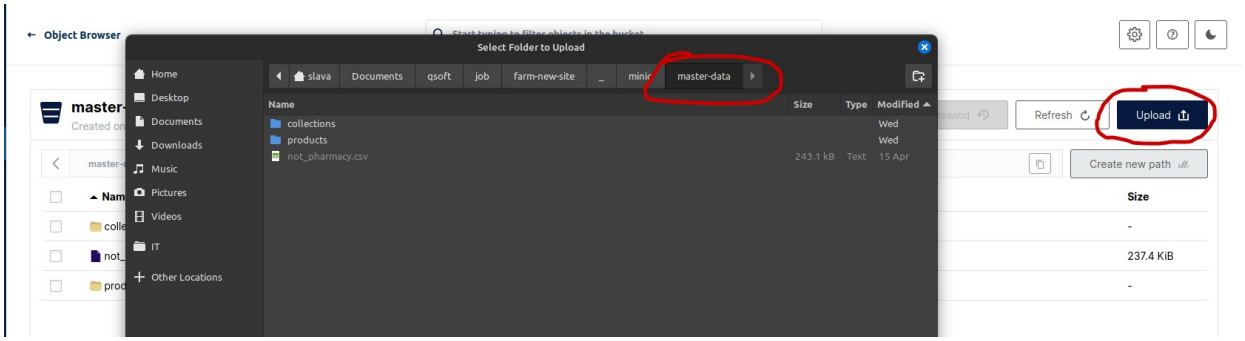

Рис. 4: Небольшая подсказка при загрузке папок в Minio

4. Настройка GUI клиента для подключения к Redis («*Доступы*»). Например, можно использовать программу «**Another Redis Desktop Manager**».

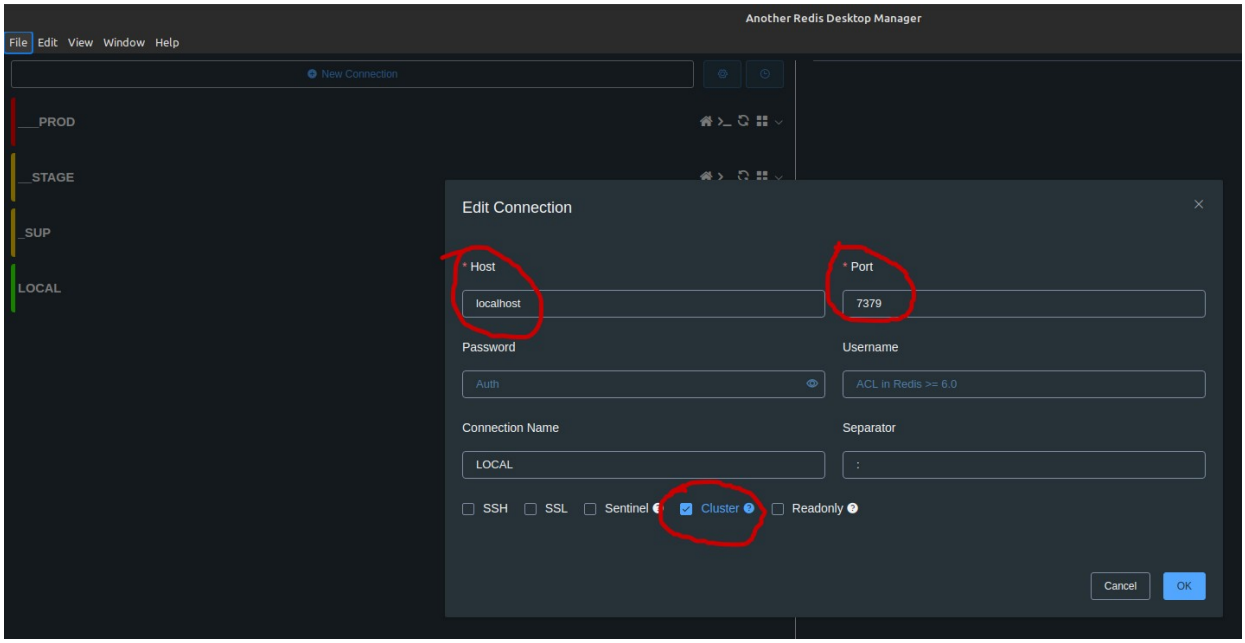

Рис. 5: Настройка GUI клиента для подключения к Redis

5. Настройка GUI клиента для подключения к Elasticsearch («*Доступы*»). Например, можно использовать программу «**Elasticvue**».

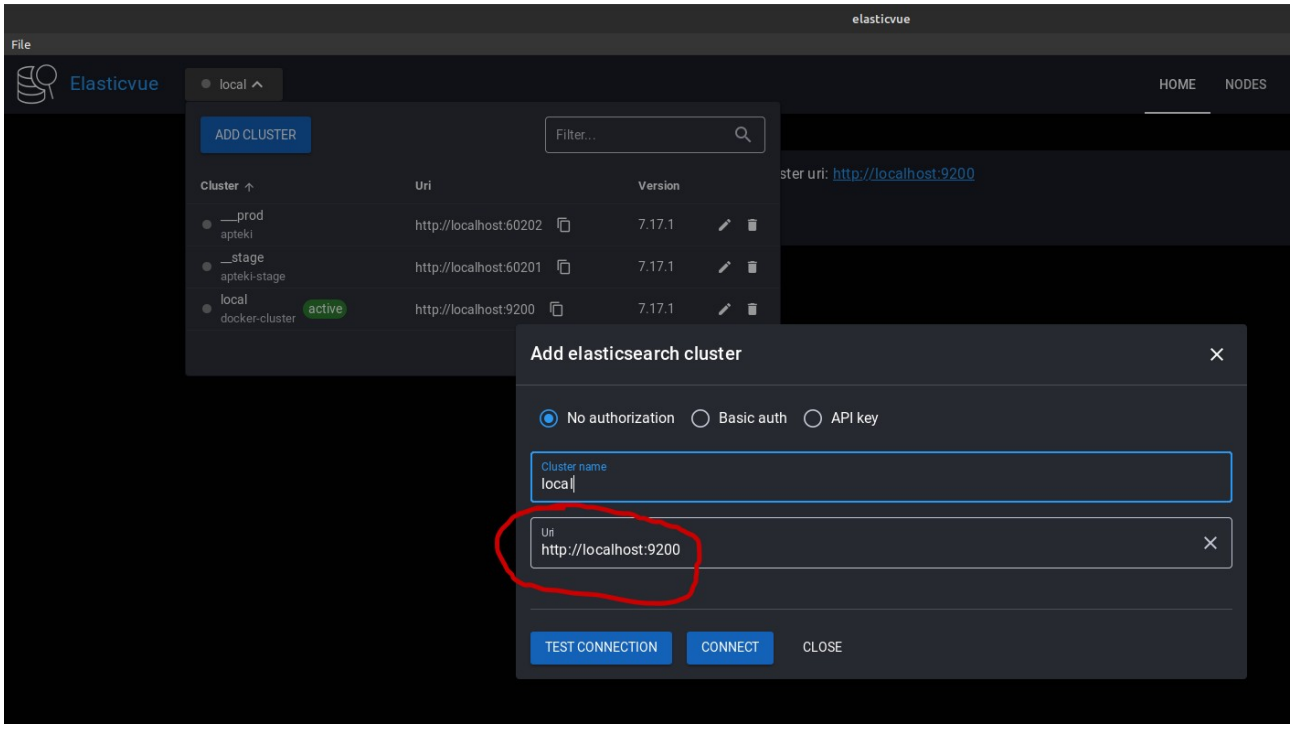

Рис. 6: Настройка GUI клиента для подключения к Elasticsearch

Далее необходимо создать индексы с произвольным названием по кол-ву активных регионов (по-умолчанию, активно 3 региона («*retail_data.regions*»).

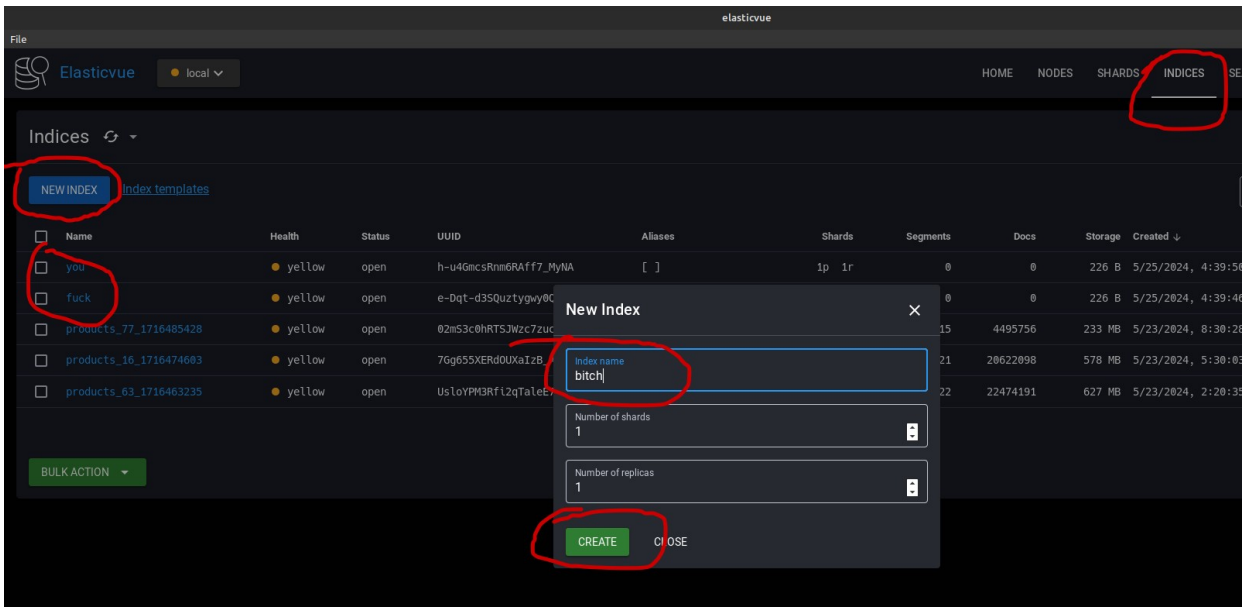

Рис. 7: Создание индексов в Elasticsearch

После этого, индексам необходимо проставить алиасы вида «*products_<код региона>*». По-умолчанию, нужно проставить такие:

- products_16
- products_63
- products_77

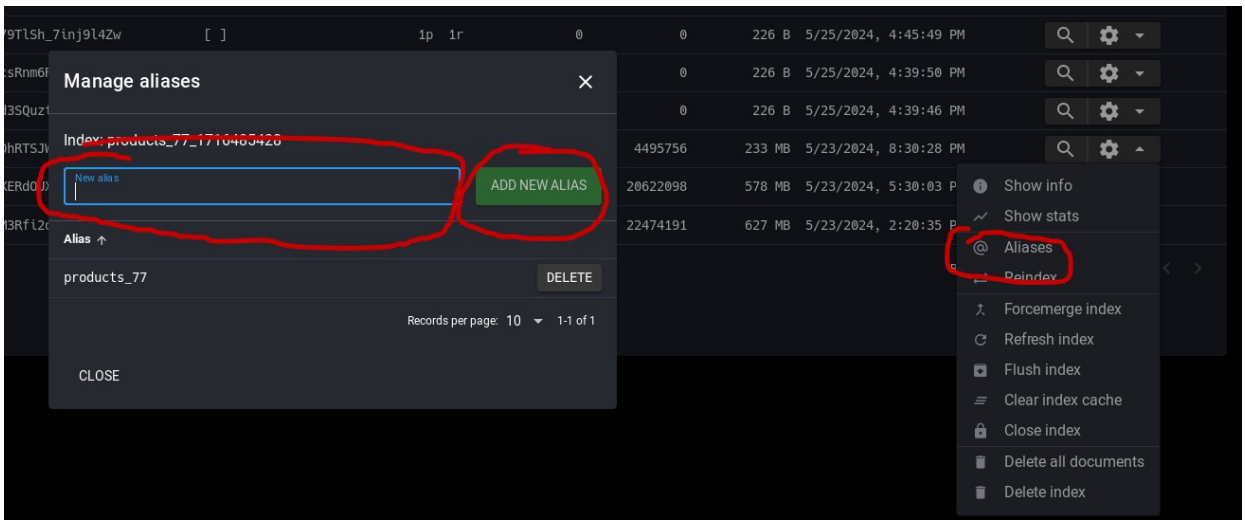

Рис. 8: Добавление алиасов для индексов

После этого убеждаемся, что алиасы корректно установились.

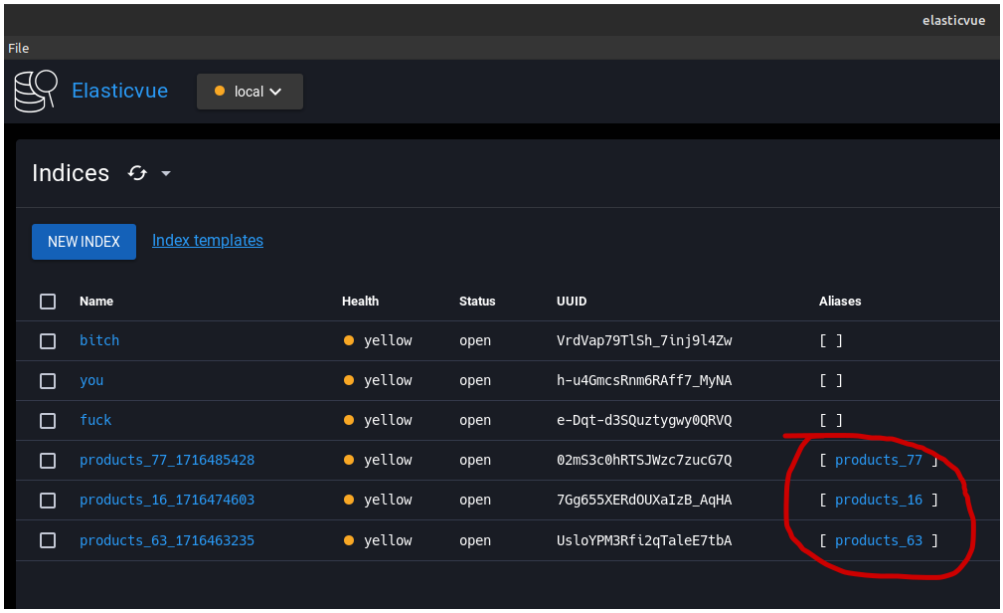

Рис. 9: Проверка корректности установки алиасов

Для чего мы это делаем? Дело всё в том, что при запуске индексации создаётся новый индекс вида «*products_<код региона>_<timestamp>*». После того как индексация завершается индексы переключаются — алиас переносится со старого индекса на новый. Индексы без алиасов затем чистятся командой «*php artisan elastic:remove-broken*» (у нас локально для этого используется отдельный контейнер и команда «*make clear-broken-indexes*»). Косяк в том, что если старый индекс не будет обнаружен, то индексация упадёт («*services/search-service/app/Client/ElasticClient.php->replaceIndex*»).

6. *список будет дополняться…*

Красава, поздравляю!
Ты успешно справился(-ась), наслаждайся работой!
А ну быстро сотвори сейчас что-то крутое и прекрасное, тебе ясно?

Доступы

http://localhost:3000	фронт сайта
http://localhost:15672	RabbitMQ Логин: guest Пароль: guest
http://localhost:8900/buckets	Minio Логин: sail Пароль: password
http://localhost:3030/swagger	Дока
http://localhost:56011	Кибана
mongodb://localhost/?directConnection=true	MongoDB
postgresql://localhost:5432	Postgres Логин: postgres Пароль: postgres
http://localhost:9200	эластика
localhost:7379	redis Логин/пароль не нужен, главное подключаться как к кластерному
localhost:8123	clickhouse логин/пароль пустые

Потенциальные проблемы

1. **Занят порт** — на хосте отключить/удалить службу, которая оккупировала этот порт или же поменять порты в «*docker-compose.yml*». Но если изменять «*docker-compose.yml*», то стоит помнить, что возможно что-то сломается в других сервисах, например надо будет перепроверить env (например, порт для подключения к БД).
2. **Waiting for control socket to be removed…** — так и не смог понять из-за чего появляется эта проблема, помогает просто перезапуск сервиса.