ИСПОЛЬЗОВАНИЕ **DOCKER COMPOSE** ДЛЯ **РАЗРАБОТКИ** И **ТЕСТИРОВАНИЯ**

Романов Алексей
IT Enduro

tg: @romanowalex
GitHub: romanow

HEISENBUG

ПЛАН ДОКЛАДА

- Какие проблемы мы хотим решить?
- Что такое Docker Compose, из чего он состоит и какую задачу решает.
- Собираем, запускаем и смотрим что получилось.
- Как организуем работу с профилями?
- Разберем что делать, когда необходимо ожидать завершения запуска другого контейнера.
- А что с ресурсами?
- Почему не используем Minikube?
- Поговорим про то, как отлаживать приложения внутри контейнера.
- Рассмотрим, как использовать это в локальной разработке.

ДОКЛАДЧИК

Романов Алексей

- 11+ лет Java/Kotlin.
- 6+ лет в роли TeamLead/Архитектора.
- 5 лет опыт DevOps.
- 8 лет преподаю в Бауманском курс «Распределенные Системы Обработки Информации» aka Микросервисы.
- Работаю Software Архитектор.
- Мы с вами встречались на других конференциях :)

КАКИЕ ПРОБЛЕМЫ МЫ ХОТИМ РЕШИТЬ?

- Получить локальную сборку сервисов, близкую к Dev или Stage стенду.
- Проверить изменения в части микросервисов.

- Возможность проверить свою задачу в интеграции с другими сервисами.
- Не ждать это 100500 лет пока задеплоится на Dev.
- Иметь возможность дебажить эти сервисы.

Мы все положим в Docker! – говорили они, – вся разработка сразу станет легкой и приятной!

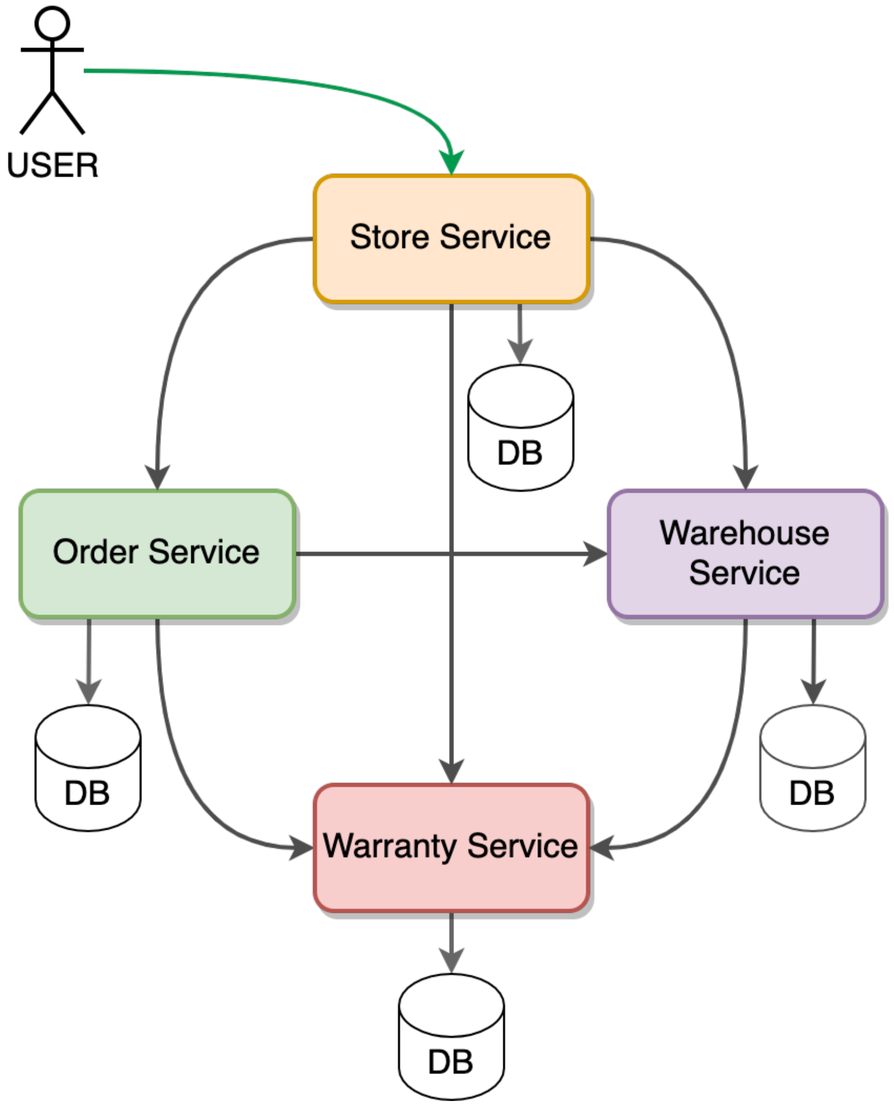
ЧТО ТАКОЕ DOCKER COMPOSE?

Docker Compose используется для одновременного управления несколькими контейнерами, входящими в состав приложения.

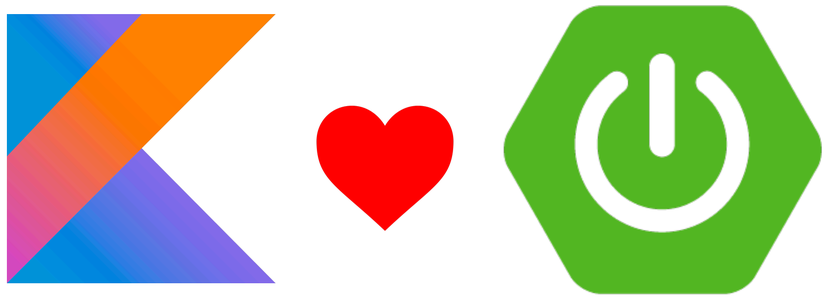
Этот инструмент предлагает те же возможности, что и Docker, но позволяет работать с более сложными приложениями.

```
1.  version: "3.8"
2.  services:
3.    postgres:
4.      image: library/postgres:13
5.      volumes:
6.        - db-data:/var/lib/postgresql/data
7.      ports:
8.        - "5432:5432"
9.
10.   store-service:
11.     image: romanowalex/store-service:v1.0
12.     environment:
13.       SPRING_PROFILES_ACTIVE: docker
14.     ports:
15.       - "8480:8480"
16.     depends_on:
17.       - postgres
```

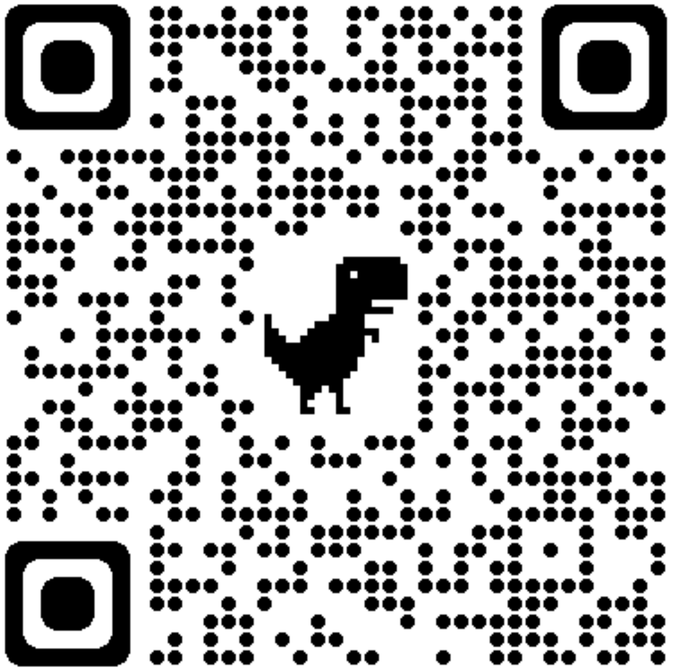
ЧТО БУДЕМ СОБИРАТЬ?



- Получить список заказов пользователя.
- Информация по конкретному заказу.
- Выполнить покупку.
- Вернуть заказ.
- Запрос гарантии по заказу.



ЧТО БУДЕМ СОБИРАТЬ?

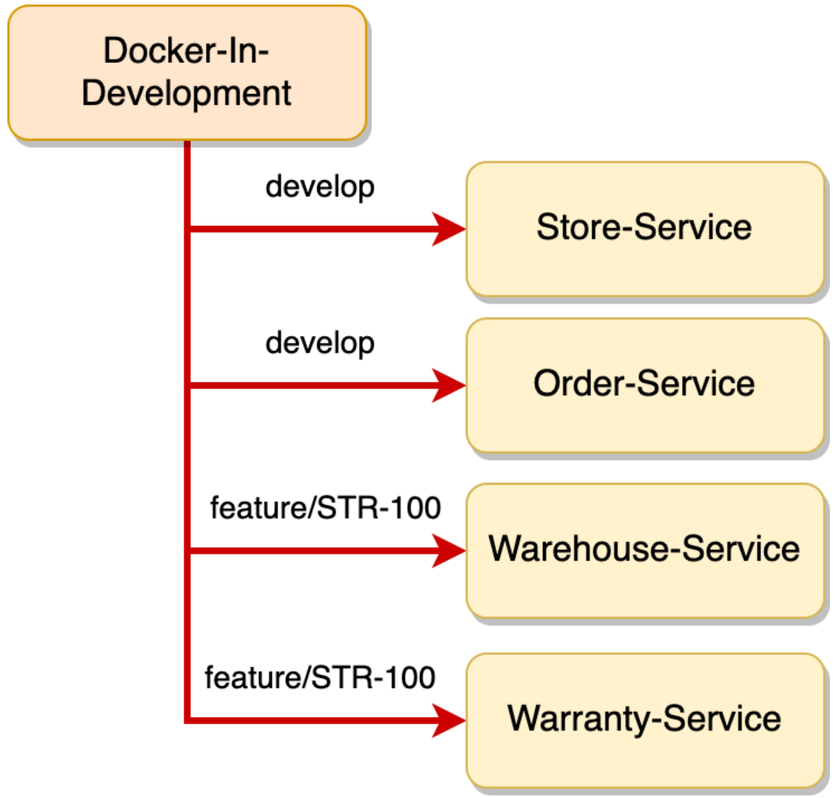


https://github.com/Romanow/docker-in-development

КАК БУДЕМ СОБИРАТЬ?

ЧТОБЫ ВСЕХ ОТЫСКАТЬ,
ВОЕДИНО СОЗВАТЬ
~~И ЕДИНОЮ ЧЕРНОЮ~~
~~ВОЛЕЙ СКОВАТЬ~~

GIT MODULES

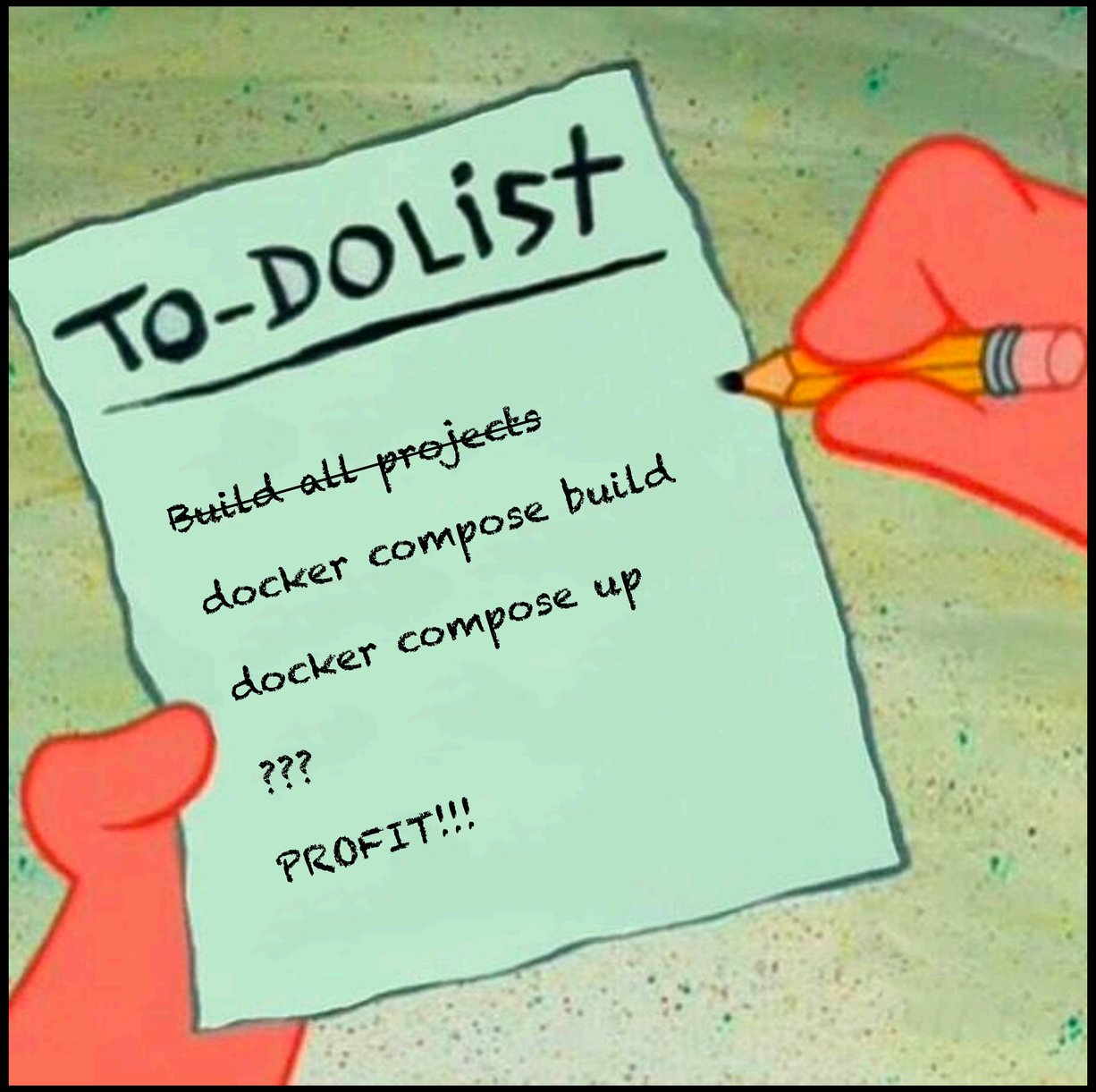



```
1.  [submodule "modules/store-service"]
2.     path = modules/store-service
3.     url = git@github.com:Romanow/store-service.git
4.     branch = develop
5.
6.  [submodule "modules/order-service"]
7.     path = modules/order-service
8.     url = git@github.com:Romanow/order-service.git
9.     branch = develop
10.
11. [submodule "modules/warehouse-service"]
12.    path = modules/warehouse-service
13.    url = git@github.com:Romanow/warehouse-service.git
14.    branch = feature/STR-100
15.
16. [submodule "modules/warranty-service"]
17.    path = modules/warranty-service
18.    url = git@github.com:Romanow/warranty-service.git
19.    branch = feature/STR-100
```

TO-DOList
Build all projects
docker compose build
docker compose up
???
PROFIT!!!

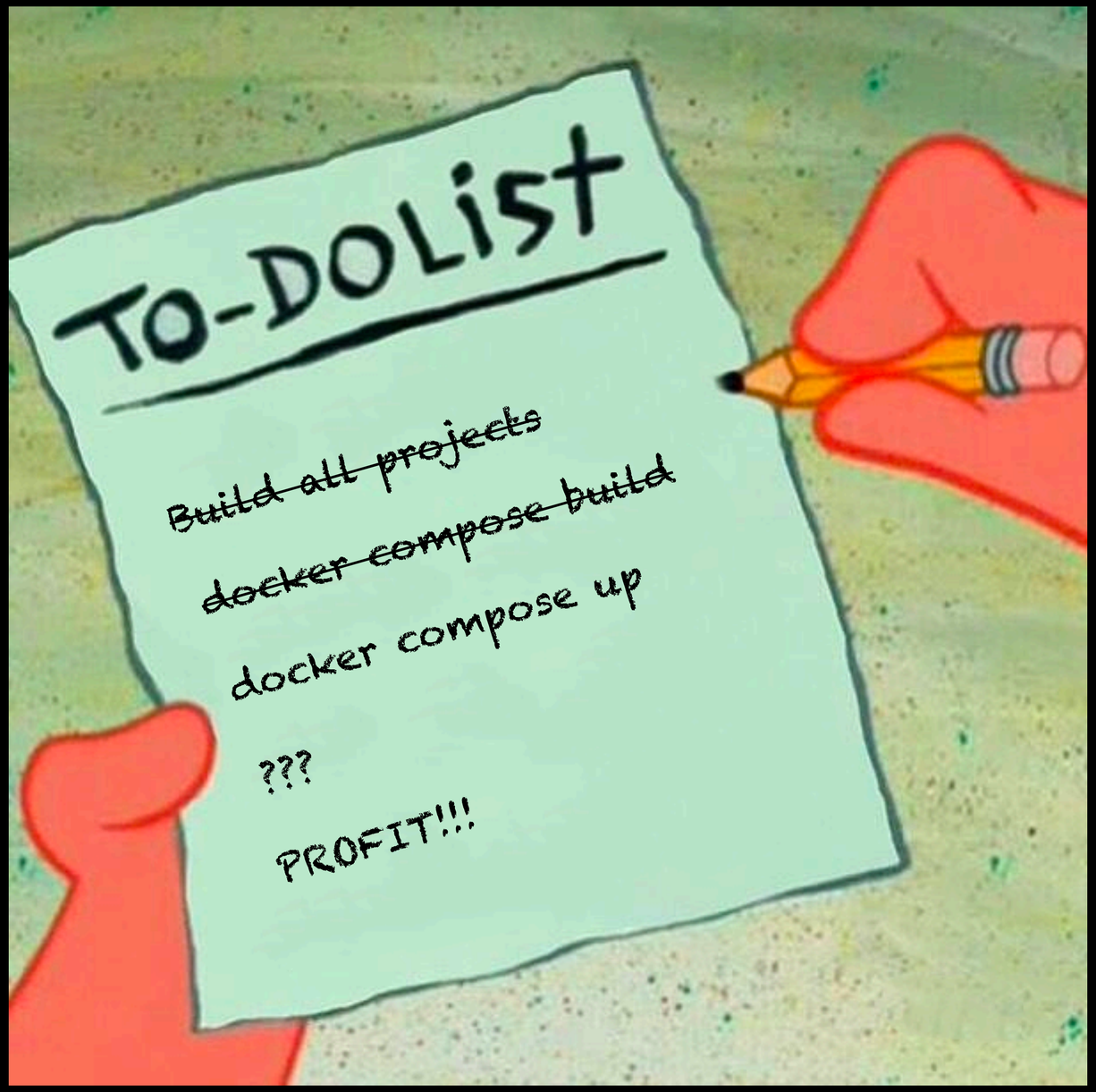
СОБИРАЕМ ПРОЕКТ

```
1. #!/usr/bin/env bash
2.
3. for dir in ./modules/*; do
4.     gradlew_exist="$dir"/gradlew
5.     if [[ ! -f $gradlew_exist ]]; then
6.       continue
7.     fi
8.
9.   printf "Building module ‘%s'" "$(basename "$dir")"
10.  "$dir"/gradlew clean build -p "$dir"
11. done
```

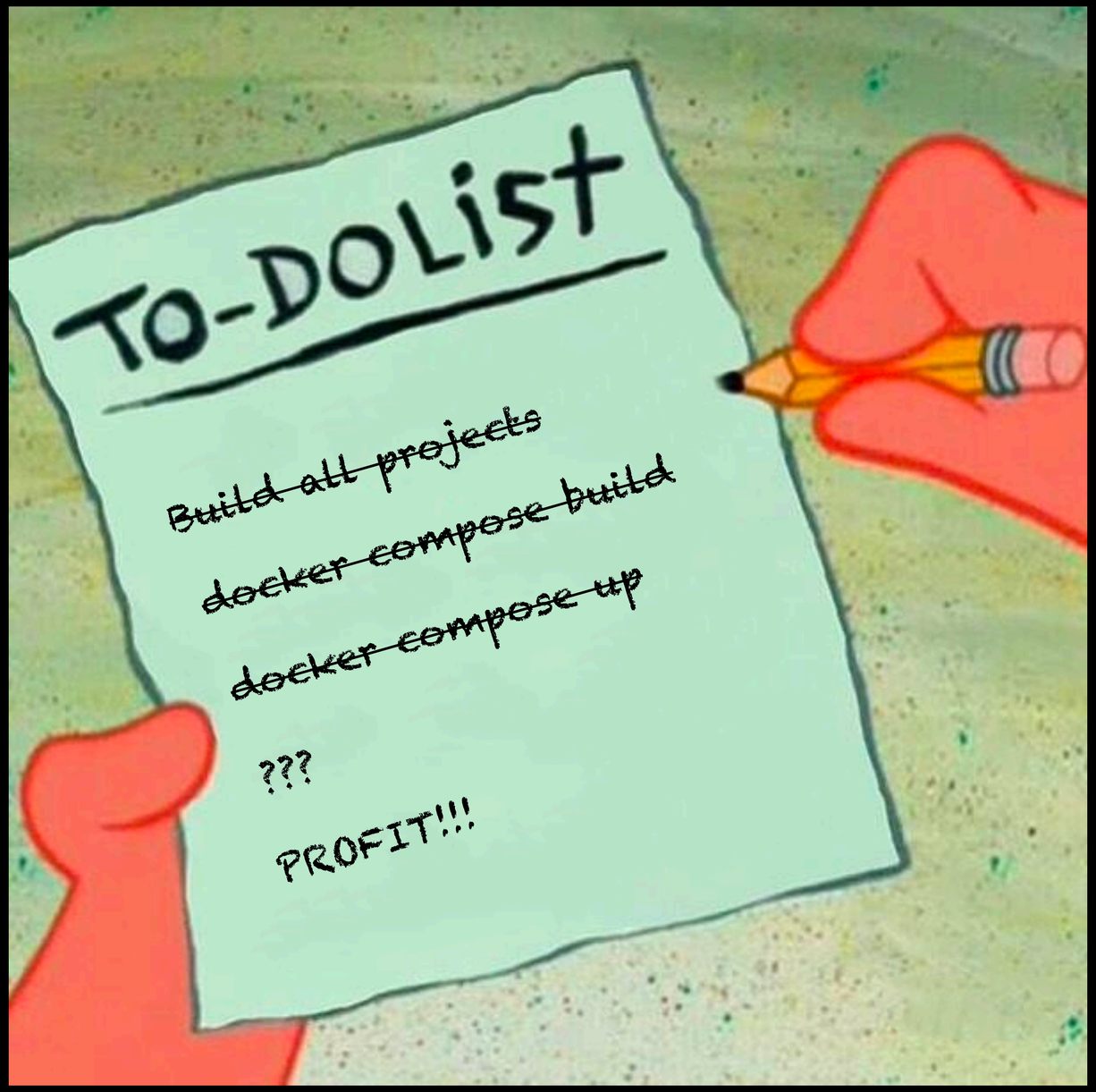

TO-DOLIST
~~Build all projects~~
docker compose build
docker compose up
???
PROFIT!!!

ЗАВОРАЧИВАЕМ В DOCKER

```
1. store-service:
2.   build: ./modules/store-service
3.   image: romanowalex/store-service:v1.0
4.   container_name: store-service
5.   environment:
6.     SPRING_PROFILES_ACTIVE: docker
7.   ports:
8.     - "8480:8480"
9.   depends_on:
10.    - postgres
```


TO-DOLIST
~~Build all projects~~
~~docker compose build~~
docker compose up
???
PROFIT!!!

RUN DOCKER
RUN
imgflip.com


TO-DOLIST
~~Build all projects~~
~~docker compose build~~
~~docker compose up~~
???
PROFIT!!!

РАБОТА С ПРОФИЛЯМИ

Рецепт для Java + Spring Boot, но идея работает и для других языков и фреймворков.

- Базовые настройки храним в *application.yml*.
- Специфичные для окружения настройки в *application-<env>.yml*.
  Например: *application-docker.yml*.
- Пароли и другие sensentive данные, адреса и т.п. можно передавать через OS environment. Например для *application-dev.yml:*

```
spring:
  datasource:
    username: program
    password: ${DATABASE_PASSWORD}
    url: ${DATABASE_CONNECTION_URL}
    driver-class-name: org.postgresql.Driver
```

- Все специфичные настройки окружения описываем в *README.md*.

ПОРЯДОК ЗАПУСКА СЕРВИСОВ

Control startup and shutdown order in Compose

Estimated reading time: 3 minutes

You can control the order of service startup and shutdown with the depends_on option. Compose always starts and stops containers in dependency order, where dependencies are determined by `depends_on`, `links`, `volumes_from`, and `network_mode: "service:..."`.

However, for startup Compose does not wait until a container is “ready” (whatever that means for your particular application) - only until it’s running. There’s a good reason for this.

There are several things to be aware of when using `depends_on`:

- `depends_on` does not wait for `db` and `redis` to be “ready” before starting `web` - only until they have been started. If you need to wait for a service to be ready, see Controlling startup order for more on this problem and strategies for solving it.
- The `depends_on` option is ignored when deploying a stack in swarm mode with a version 3 Compose file.

ПОРЯДОК ЗАПУСКА СЕРВИСОВ

```
1.  FROM openjdk:11-jdk
2.
3.  EXPOSE 8480
4.  WORKDIR application
5.
6.  ADD https://github.com/ufoscout/.../2.9.0/wait /application/wait.sh
7.  RUN chmod a+x /application/wait.sh
8.
9.  COPY --from=builder application/spring-boot-loader/ ./
10. COPY --from=builder application/dependencies/ ./
11. COPY --from=builder application/snapshot-dependencies/ ./
12. COPY --from=builder application/application/ ./
13.
14. ENTRYPOINT /application/wait.sh && \
15.     java org.springframework.boot.loader.JarLauncher
```

ПОРЯДОК ЗАПУСКА СЕРВИСОВ

```
1. store-service:
2.   build: ./modules/store-service
3.   image: romanowalex/store-service:v1.0-docker-wait
4.   container_name: store-service
5.   environment:
6.     SPRING_PROFILES_ACTIVE: docker
7.     WAIT_HOSTS: order-service:8380
8.     WAIT_TIMEOUT: 240
9.   ports:
10.    - "8480:8480"
11.  depends_on:
12.    - postgres
```

ПОТРЕБЛЕНИЕ РЕСУРСОВ

- Мы рассматриваем Docker Compose 3.x
- В docker compose мы не можем задавать ресурсы (можем только в swarm).
- Используем Postgres, Kafka и N приложений на Java / Spring.
- Для docker выделено 10Gb и 4 CPU.

Количество сервисов	Потребление памяти	Потребление CPU	Как это все работает
5	40 %	20 %	Быстро
10	60 %	30 %	Быстро
20	100 %	60 %	Очень медленно
25	100 %	70 %	Для запуска не хватает памяти

ПОЧЕМУ НЕ ИСПОЛЬЗУЕМ MINIKUBE

Самый очевидный и распространенный аналог: Minikube.

Плюсы:

- Используем helm чарты, что и для реального деплоя на стенды.
- Получаем локальный стенд с возможностью масштабирования сервисов.
- Можно задавать лимиты CPU и памяти.

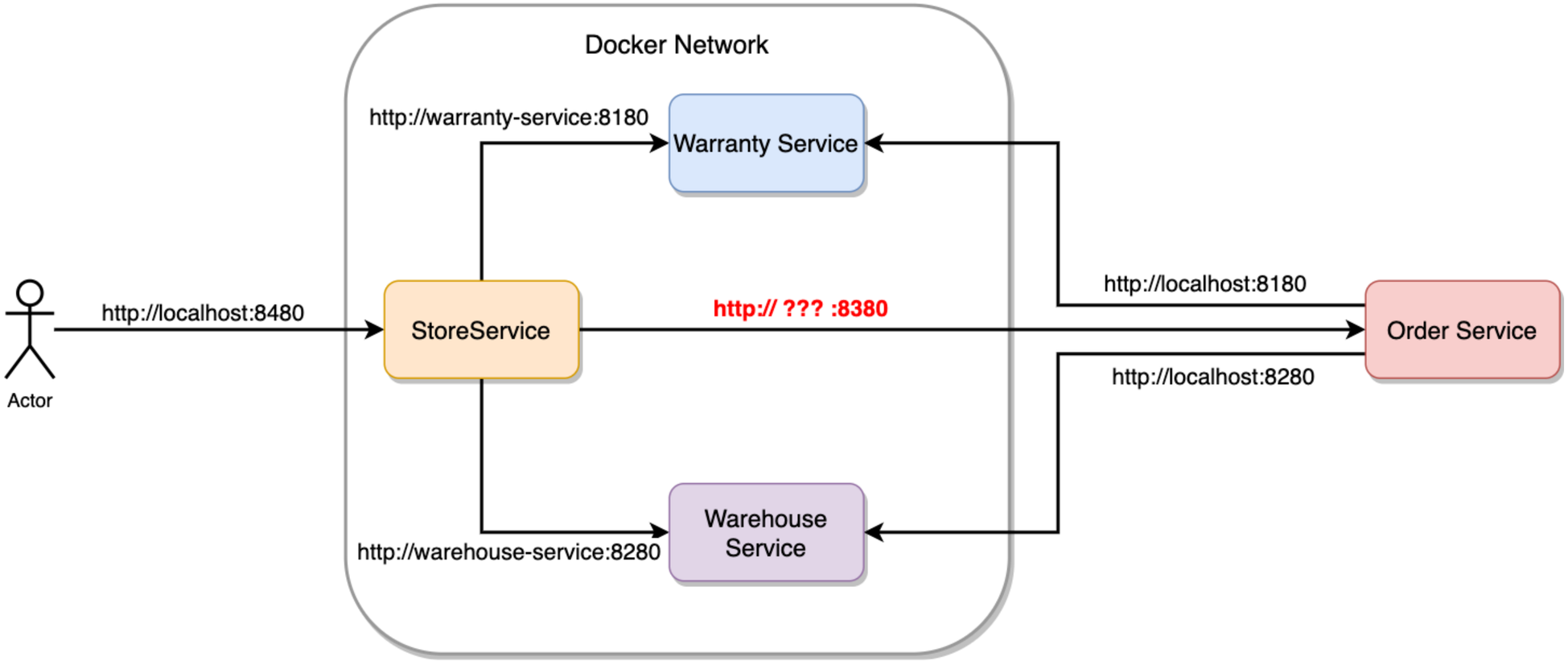
Минусы:

- Писать манифесты и helm чарты сложно.
- При запуске Minikube через docker не работает ingress на Windows и Mac.
- Если запускаем на VM, то требуется на 30% больше ресурсов.

ОТЛАДКА ВНУТРИ КОНТЕЙНЕРА

```
1.  FROM openjdk:11-jdk
2.
3.  EXPOSE 8480
4.  EXPOSE 5005
5.
6.  WORKDIR application
7.  COPY --from=builder application/spring-boot-loader/ ./
8.  COPY --from=builder application/dependencies/ ./
9.  COPY --from=builder application/snapshot-dependencies/ ./
10. COPY --from=builder application/application/ ./
11.
12. ENTRYPOINT ["java",
13.     "-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=*:5005",
14.     "org.springframework.boot.loader.JarLauncher"]
```

ЛОКАЛЬНАЯ РАЗРАБОТКА



ЛОКАЛЬНАЯ РАЗРАБОТКА

- В настройках адреса сервисов прописываем как `services.order-url: ${ORDER_URL:http://order-service:8020}`
- В Docker Compose в блоке environment `ORDER_URL: host.docker.internal:8380`
- В самом приложении создаем профиль *application-docker-to-local.properties*, где прописываем адреса на localhost.

ВЫВОДЫ

- Это некоторое промежуточное решение между «у меня все работает локально» и «все работает на тестовом контуре k8s».
- Docker Compose удобный инструмент, но есть некоторые сложности, которые приходится решать «костылями».
- У разработчиков больше нет отмазки «как я весь контур подниму, чтобы это протестировать?!».
- Можно делать debug прямо внутри контейнера.
- Можно использовать для локальной разработки.

ВОПРОСЫ

ССЫЛКИ

@romanowalex

https://github.com/Romanow

romanowalex

https://github.com/Romanow/docker-in-development