

Сборка проекта

+ Docker

Неделя 9

Представим ситуацию

Ассистент по КПО скачал ИДЗ и пытается запустить его на своем компьютере. В ИДЗ студент реализовал API с использованием Spring Web и базы данных MySQL. Само приложение запускается на порту 8080.

Перед тем, как запустить программу, ассистенту придется установить себе MySQL, убедиться, что БД настроена корректно, а также, что порт 8080 не занят. Вывод: ассистент несет убытки по части памяти своего ПК, а также по времени.



Решение?

Пожалеем ассистента и постараемся решить нашу проблему. Но как?

Было бы здорово, если бы была бы возможность отправить ассистенту готовую «коробку» с приложением, которую бы осталось только запустить. Даже без того, чтобы смотреть в код!

What are the benefits of software containers?



Everything contained
in a single package



Runs anywhere: laptop,
data center or cloud



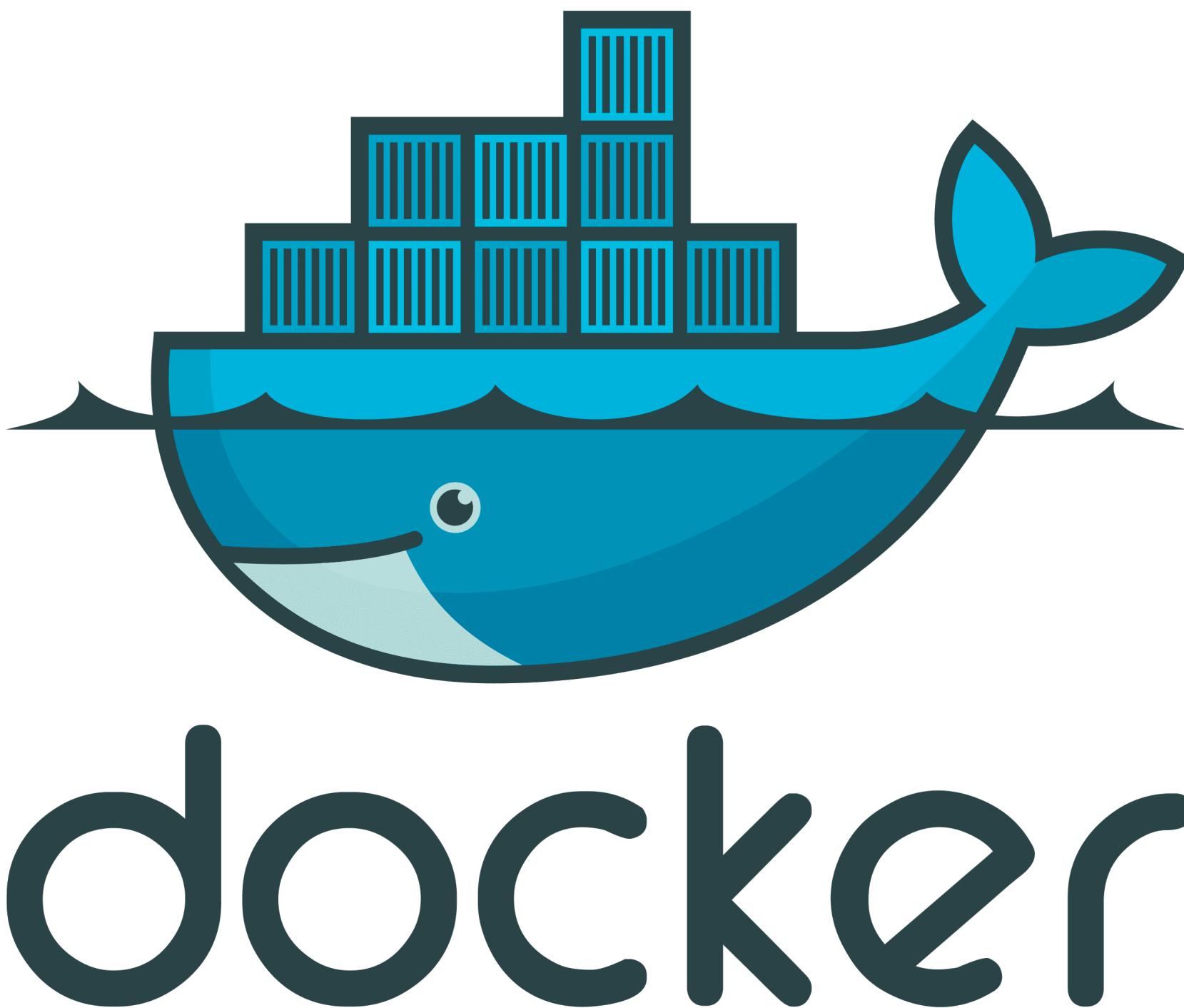
Applications are isolated
on the operating system



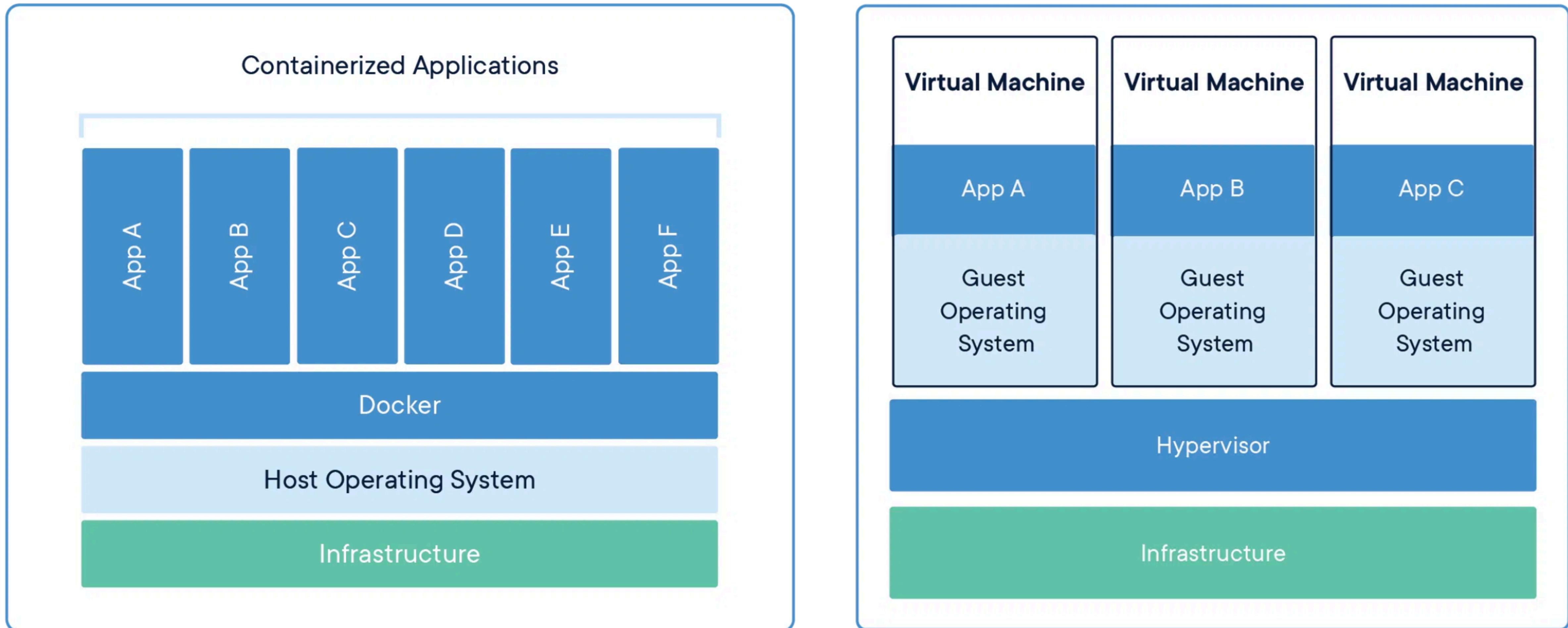
Uses fewer resources than
a virtual machine

Что такое Docker?

Docker – платформа, которая позволяет создавать контейнеры и управлять ими. Позволяет «упаковать» приложение со всем окружением и зависимостями.



Docker-котейнеры vs. виртуальные машины

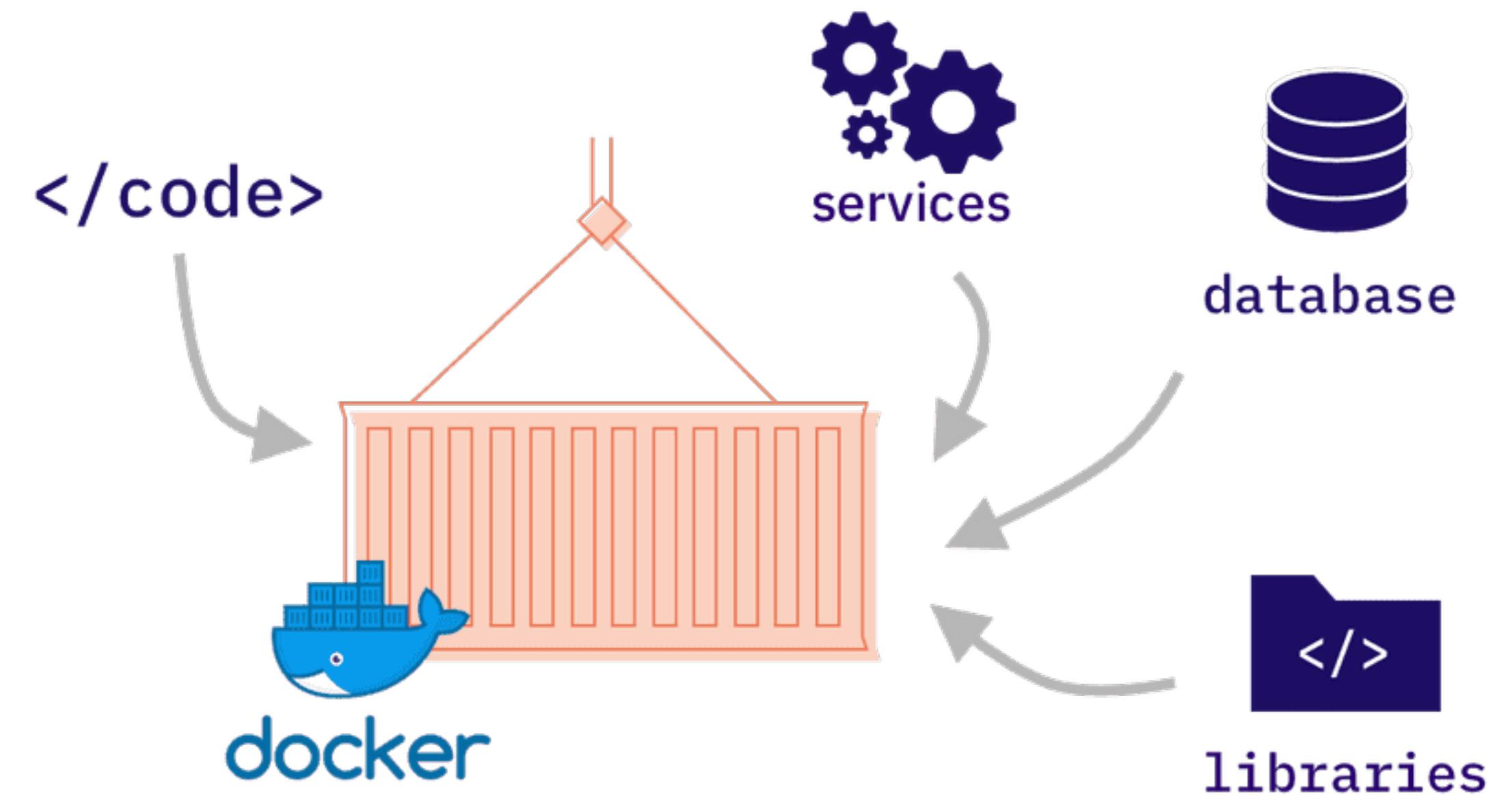


Docker-контейнеры и docker-образы

Парочка неформальных определений:

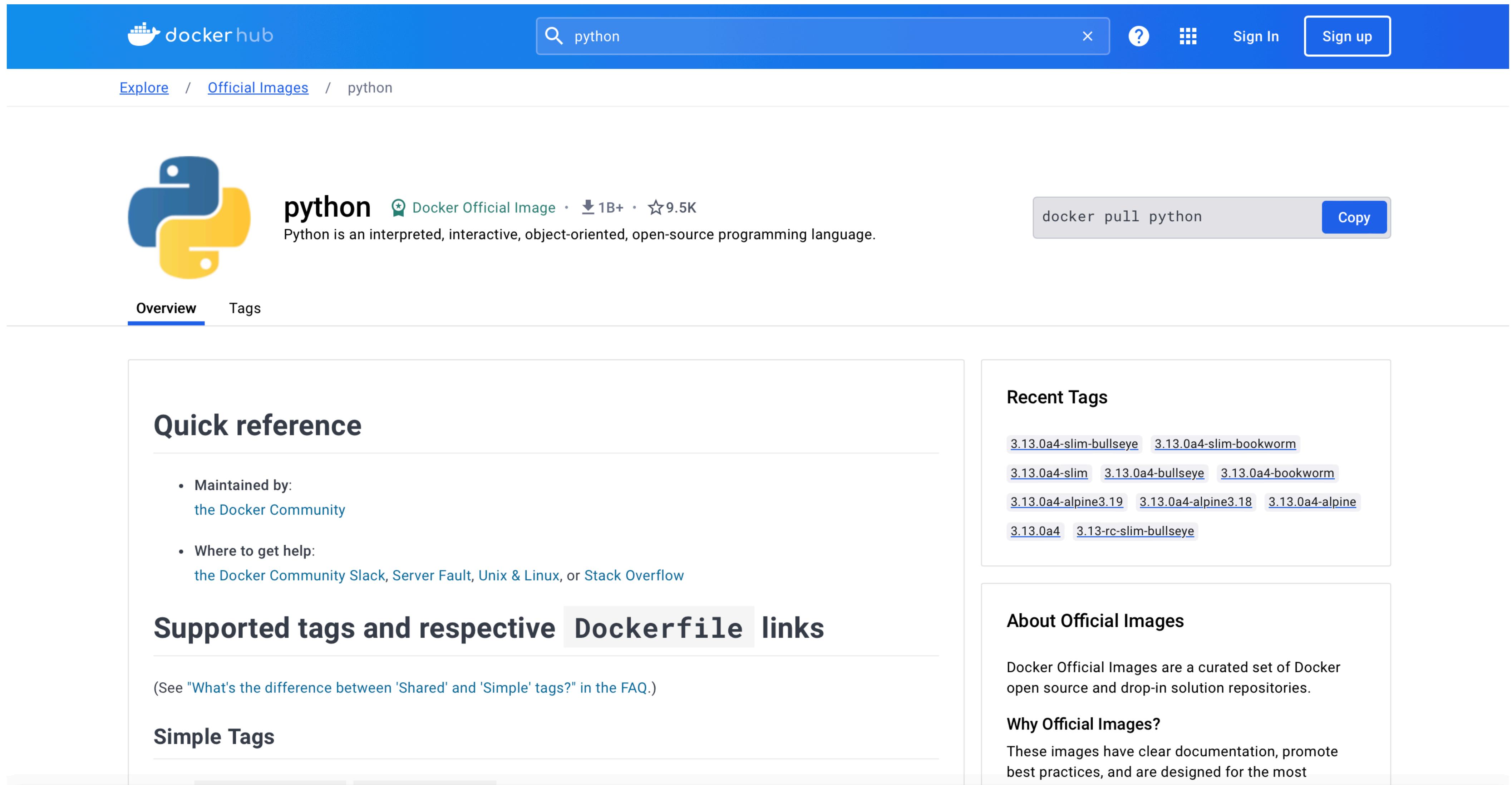
Контейнер – среда, в которую упаковано приложение. Что-то в виде коробки, которая никак не связана с внешним миром.

Образ – «заготовка», на основе которой можно запустить контейнер (можно подтягивать из интернета или использовать свой)



Запускаем простой контейнер

* На основе образа из интернета (DockerHub)



The screenshot shows the Docker Hub interface for the official Python image. At the top, there's a search bar with "python" and navigation links for "Explore", "Official Images", and "python". Below the header, the Python logo is displayed, followed by the text "python" and "Docker Official Image". It shows statistics: 1B+ downloads and 9.5K stars. A brief description states: "Python is an interpreted, interactive, object-oriented, open-source programming language." To the right, there's a button to "docker pull python" with a "Copy" option. The page is divided into several sections: "Quick reference" (with links to maintainers and help), "Supported tags and respective Dockerfile links" (with a note about shared vs simple tags), "Simple Tags" (listing tags like 3.13.0a4-slim-bullseye, 3.13.0a4-slim-bookworm, etc.), "Recent Tags" (listing recent tags such as 3.13.0a4, 3.13-rc-slim-bullseye, etc.), and "About Official Images" (explaining what official images are).

python Docker Official Image • 1B+ • 9.5K

Python is an interpreted, interactive, object-oriented, open-source programming language.

Overview Tags

Quick reference

- Maintained by:
the Docker Community
- Where to get help:
the Docker Community Slack, Server Fault, Unix & Linux, or Stack Overflow

Supported tags and respective Dockerfile links

(See "What's the difference between 'Shared' and 'Simple' tags?" in the FAQ.)

Simple Tags

Recent Tags

3.13.0a4-slim-bullseye 3.13.0a4-slim-bookworm
3.13.0a4-slim 3.13.0a4-bullseye 3.13.0a4-bookworm
3.13.0a4-alpine3.19 3.13.0a4-alpine3.18 3.13.0a4-alpine
3.13.0a4 3.13-rc-slim-bullseye

About Official Images

Docker Official Images are a curated set of Docker open source and drop-in solution repositories.

Why Official Images?

These images have clear documentation, promote best practices, and are designed for the most

Запускаем простой контейнер

Сейчас все это можно сделать в том числе через графический интерфейс.

Шаг 1. Нужно скачать образ для запуска.

```
docker pull python
```

```
lucyrez@MacBook-Air-Lucy ~ % docker pull python
Using default tag: latest
latest: Pulling from library/python
c2964e85ea54: Pull complete
d3436c315a5d: Pull complete
[603ae72c83b1: Pull complete
bcabfc6c415b: Pull complete
f22e038e21dd: Pull complete
3ae45c5f75be: Pull complete
2107f7596de6: Pull complete
13322359dbd4: Pull complete
Digest: sha256:e83d1f4d0c735c7a54fc9dae3cca8c58473e3b3de08fcb7ba3d342ee75cf09d
Status: Downloaded newer image for python:latest
docker.io/library/python:latest
```

Запускаем простой контейнер

Шаг 2. После скачивания можно посмотреть доступные для запуска образы и запустить образ в контейнере.

```
docker images
```

```
docker run python * вот здесь процесс быстро завершился
```

```
docker run -it python
```

```
lucyrez@MacBook-Air-Lucy ~ % docker images
[REPOSITORY      TAG          IMAGE ID      CREATED      SIZE
python          latest        d8be44680b2e   4 weeks ago   1.02GB
[lucyrez@MacBook-Air-Lucy ~ % docker run python
[lucyrez@MacBook-Air-Lucy ~ % docker run -it python
Python 3.12.2 (main, Feb 13 2024, 08:24:27) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
[>>>
```

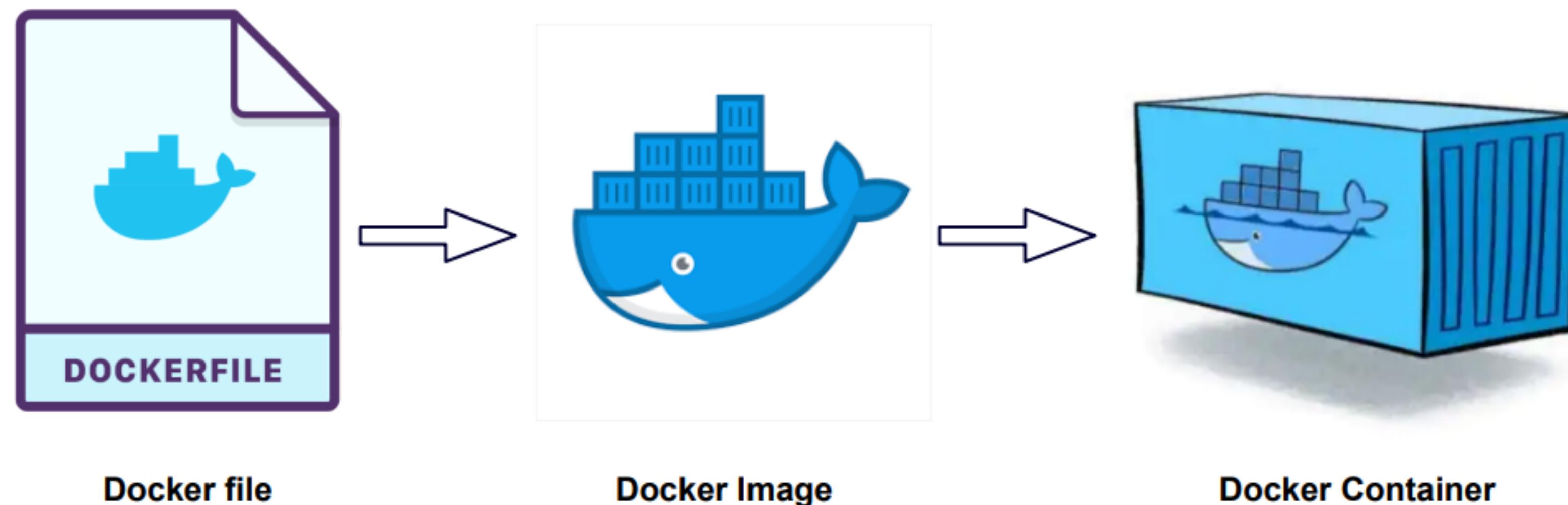
Поздравляем! Теперь у вас запущен Python внутри контейнера

Как запустить контейнер для своего приложения?

Помним о том, что для запуска контейнера нужен образ. Но откуда его взять, если в интернете вашего приложения точно нет? Ответ – нужно создать образ вручную.

Делается это с помощью Docker-файлов.

На картинке показан весь процесс: из Docker-файла можно получить образ, который можно будет запустить в контейнере. Осталось разобраться как сделать docker-файл для вашего приложения (см. след. слайд)



Docker-файл: что это такое? Как написать?

Dockerfile – файл, который пошагово объясняет Docker-у как именно нужно собрать ваш образ.

Dockerfile позволяет создать НЕ контейнер, а именно образ для контейнера.

Для того, чтобы начать работу, создадим файл с названием Dockerfile в корне проекта. Название должно быть именно таким (без расширения)

Пример Dockerfile

Здесь приложение на NodeJS, которое будет запускаться на порту 3000 в docker-контейнере.

```
FROM node
WORKDIR /app
COPY . .
RUN npm install
EXPOSE 3000
CMD [ "node", "app.js" ]
```

Пример Dockerfile (для нашего проекта)

Тут мы видим пример того, как может выглядеть Dockerfile для проекта на java/kotlin.

```
1 FROM openjdk:11
2 ADD /target/SpringWebDemo-1.0-SNAPSHOT.jar music-api.jar
3 CMD ["java", "-jar", "music-api.jar"]
4
```

Проект на Java невозможно запустить без JDK.

Поэтому, в первой строчке мы говорим, что образ должен быть основан на JDK, придется скачать его из интернета.

Секундочку, а что такое jar?

Что вообще происходит?

Лирическое отступление: сборка проекта

Обычно вы всегда запускали проект через IDE.

Но что, если у человека, который хочет пользоваться вашим приложением нет IDE? Или вы не хотите показывать ему ваш исходный код?

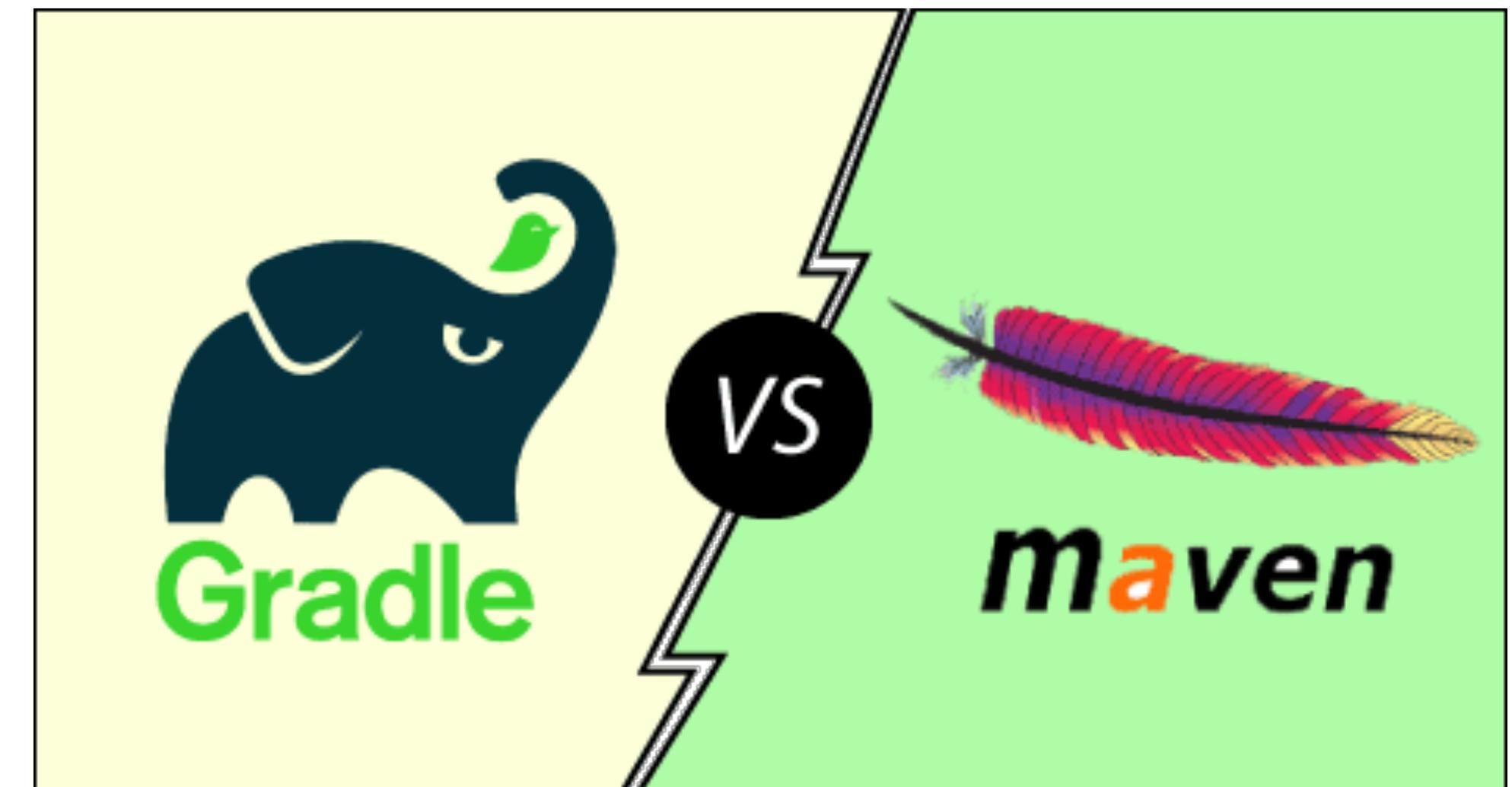
В таком случае, ваш проект нужно собрать. Или, как еще говорят на исконно русском – «сбилдить».

Системы сборки: Maven и Gradle

Все это инструменты для автоматизации сборки проектов.

Преимущества Gradle:

- Он использует DSL, а не XML. Можно даже писать на Kotlin
- Более гибкий
- Эффективнее Maven-а: демонстрацию можно увидеть здесь:
<https://gradle.org/maven-vs-gradle/>



Жизненный цикл Maven

clean – удаляются все скомпилированные файлы из каталога target

validate – идет проверка, вся ли информация доступна для сборки проекта

compile – комбинируются файлы с исходным кодом

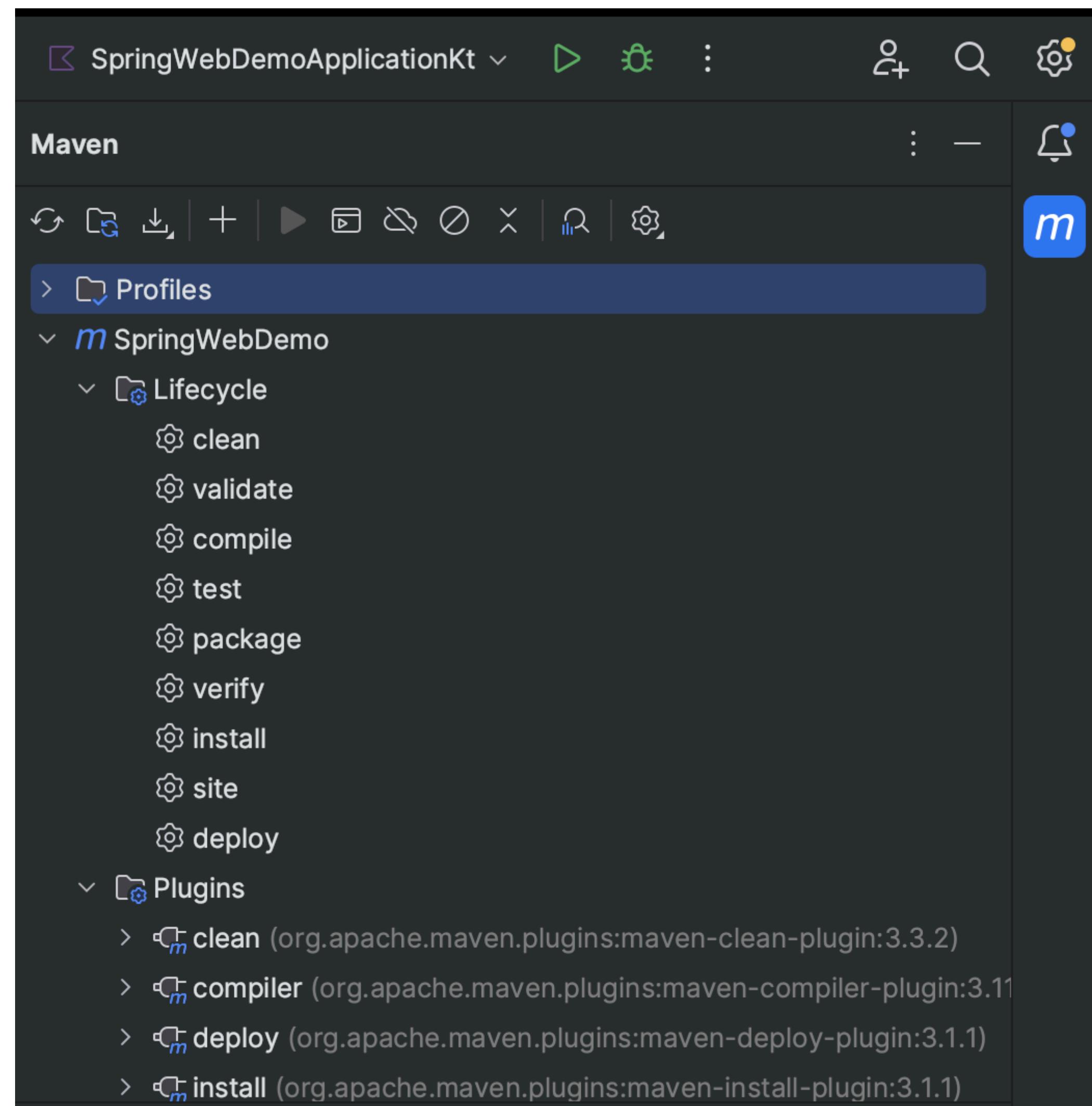
test – запускаются тесты

package – скомпилированные файлы упаковываются в jar/war архив

verify – проверки для подтверждения готовности упакованного файла

install – пакет помещается в локальный maven-репозиторий

deploy – собранный архив котируется в удаленный maven-репозиторий

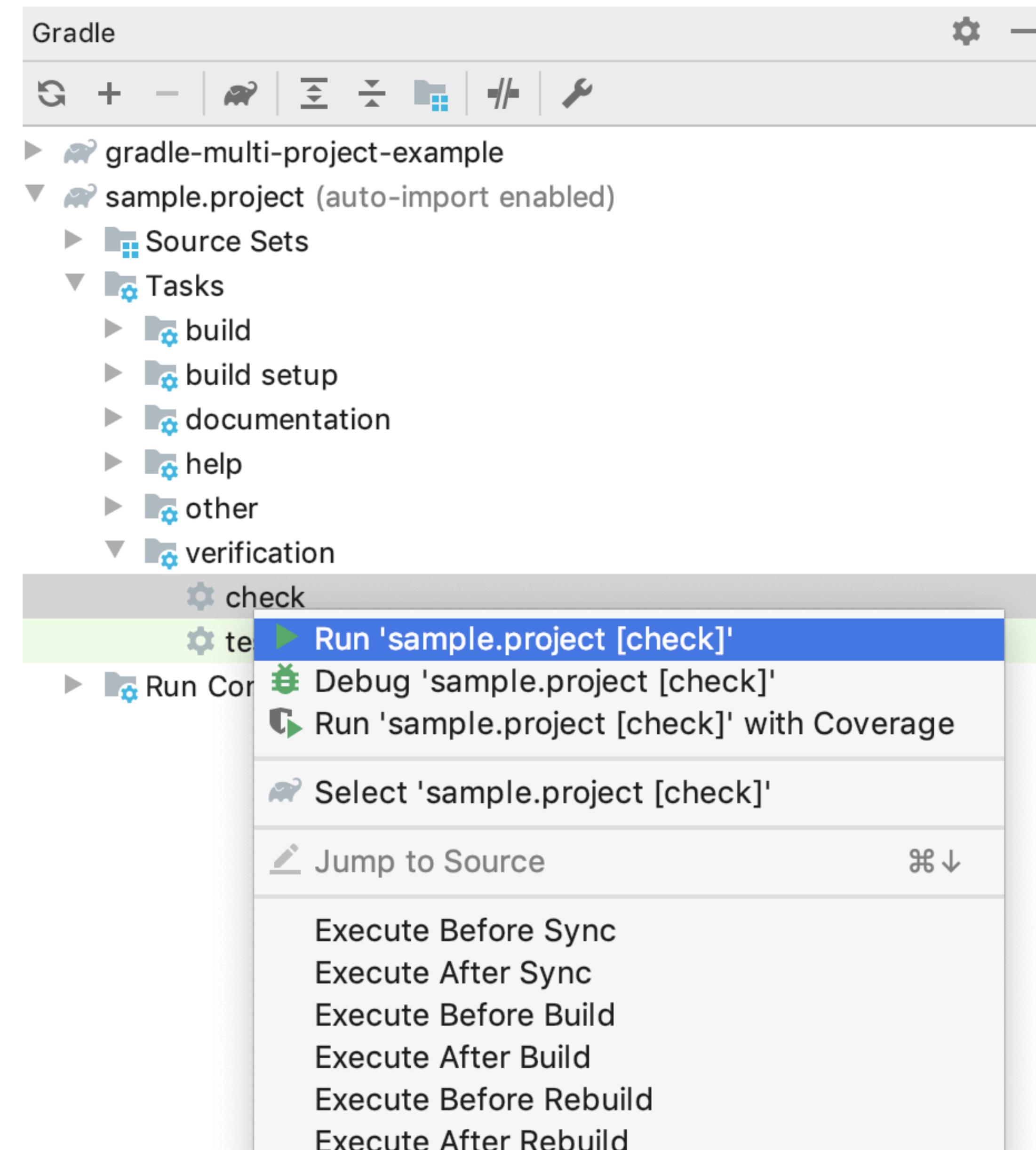


Task в Gradle

Идея аналогичная с Maven.

Это задачи, связанные со сборкой, которые можно запустить.

Для сборки приложения нужно запустить задачу build



Jar и War

Формат архива, который содержит исполняемый код приложения. Такой архив можно запустить, если у вас установлен JDK. И работать с этим можно прямо из консоли.

Теперь для запуска не нужны файлы с исходным кодом, а тем более IDE.

```
java -jar <путь_по_которому_расположен_jar>
```

Вернемся к Docker-у

Теперь должно быть понятней, что происходит на 2 и 3 строках.

```
1 >> FROM openjdk:11
2      ADD /target/SpringWebDemo-1.0-SNAPSHOT.jar music-api.jar
3      CMD ["java", "-jar", "music-api.jar"]
4
```

2 – добавляем в образ («коробочку») архив jar с исполняемым кодом (тут мы его копируем и переименовываем в music-api.jar). Этот архив будет лежать в корне.

3 – запускаем команду `java -jar music-api.jar`

* Тут скачается 11 версия JDK, нам нужна как минимум 17

** Перед сборкой образа нужно убедиться, что jar архив уже есть по указанному пути

Собираем образ

```
Deploying '<unknown> Dockerfile: Dockerfile'...
Building image...
Preparing build context archive...
[=====] 474/474 files
Done

Sending build context to Docker daemon...
[=====] 51,29MB
Done

Step 1/3 : FROM openjdk:11
11: Pulling from library/openjdk
114ba63dd73a: Pull complete
Digest: sha256:99bac5bf83633e3c7399aed725c8415e7b569b54e03e4599e580fc9cdb7c21ab
Status: Downloaded newer image for openjdk:11
    --> c8fb8288d1c8
Step 2/3 : ADD /target/SpringWebDemo-0.0.1-SNAPSHOT.jar music-api.jar
    --> 4edadacf2c5
Step 3/3 : CMD ["java", "-jar", "music-api.jar"]
    --> Running in 3c697d353d96
    --> Removed intermediate container 3c697d353d96
    --> 243bffa8dda0

Successfully built 243bffa8dda0
'<unknown> Dockerfile: Dockerfile' has been deployed successfully.
```

Запускаем контейнер

Пробуем запустить контейнер по ID образа.

Вроде работает! Но рано радоваться...

```
[lucyrez@MacBook-Air-Lucy target % docker images
[REPOSITORY      TAG          IMAGE ID      CREATED        SIZE
<none>          <none>        d2af05979072  29 seconds ago  558MB
python           latest        d8be44680b2e  4 weeks ago   1.02GB
openjdk          17           4717374ea615  22 months ago  501MB
lucyrez@MacBook-Air-Lucy target % docker run d2af05979072
[
  .
  /\\ \ / ----| - - - - - ( ) - - - - - \ \ \ \ \ \
  ( ( )\ __| | - | | | - \ - ' | \ \ \ \ \
  \ \ \ \ \ \ ) | | | | | | ( | | ) ) ) ) )
  ' | | | | .--| - | | | | | \ --, | / / / /
[ ======| _|=====| _|=/ / / / /
:: Spring Boot ::                      (v3.2.3)

2024-03-10T15:21:52.688Z  INFO 1 --- [           main] c.h.s.SpringWebDemoApplicationKt : Starting SpringWebDemoApplication
Kt v0.0.1-SNAPSHOT using Java 17.0.2 with PID 1 (/music-api.jar started by root in /)
[2024-03-10T15:21:52.690Z  INFO 1 --- [           main] c.h.s.SpringWebDemoApplicationKt : No active profile set, falling ba
ck to 1 default profile: "default"
2024-03-10T15:21:53.238Z  INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA rep
ositories in DEFAULT mode.
2024-03-10T15:21:53.264Z  INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository s
canning in 22 ms. Found 2 JPA repository interfaces.
2024-03-10T15:21:53.465Z  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080
  (http)
2024-03-10T15:21:53.470Z  INFO 1 --- [           main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2024-03-10T15:21:53.470Z  INFO 1 --- [           main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache
```

Запускаем контейнер

В логах мы увидим, что приложение в контейнере упало.

При внимательном осмотре можно обнаружить, что это связано с БД.

(Тут нужно вспомнить про то, что контейнер изолирован от вашего компьютера, поэтому он не видит вашу локальную БД).

Вывод: нужно запустить БД внутри контейнера.

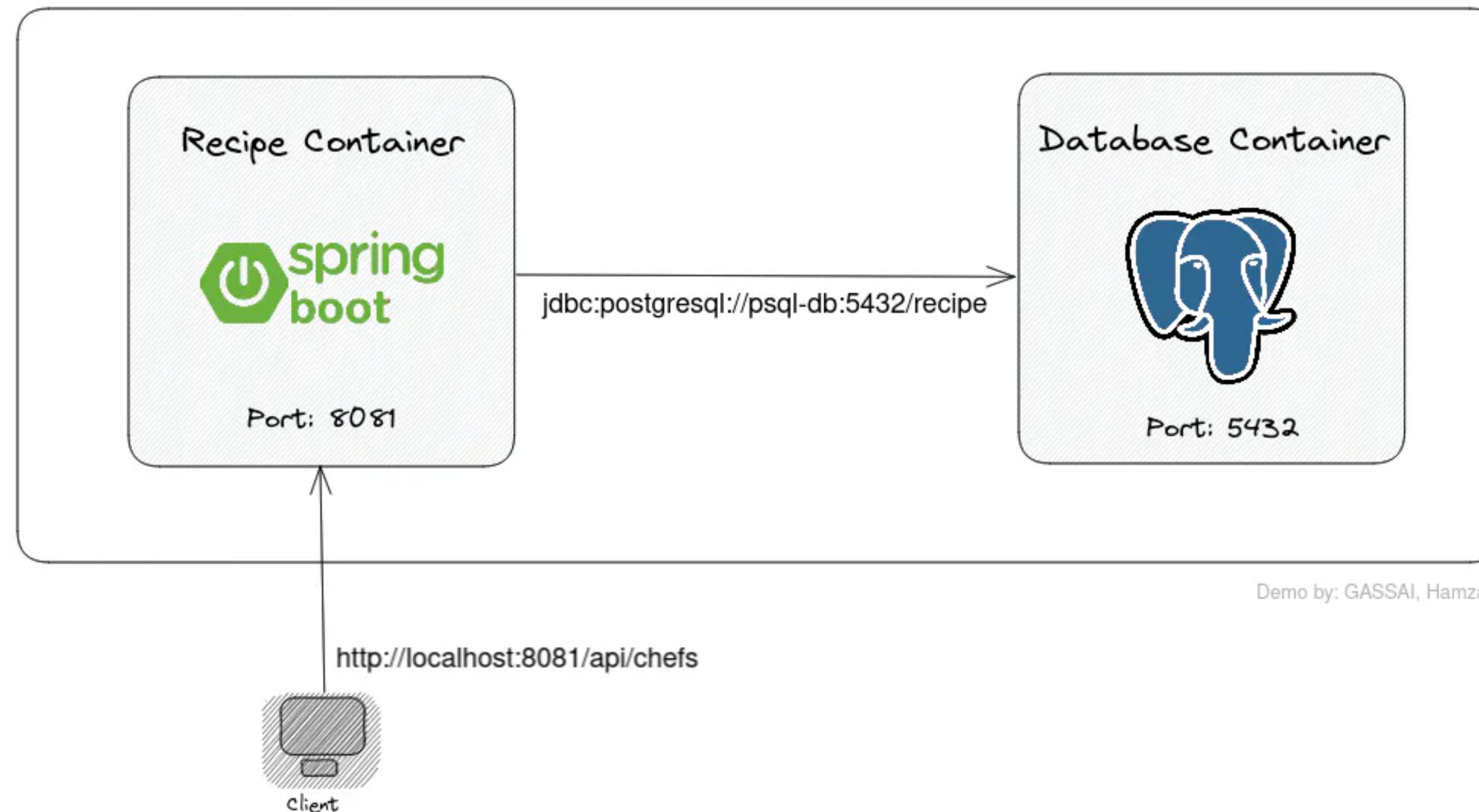
```
Caused by: org.hibernate.HibernateException: Unable to determine Dialect without JDBC metadata (please set 'jakarta.persistence.jdbc.url' for common cases or 'hibernate.dialect' when a custom Dialect implementation must be provided)
    at org.hibernate.engine.jdbc.dialect.internal.DialectFactoryImpl.determineDialect(DialectFactoryImpl.java:191) ~[hibernate-core-6.4.4.Final.jar!/:6.4.4.Final]
    at org.hibernate.engine.jdbc.dialect.internal.DialectFactoryImpl.buildDialect(DialectFactoryImpl.java:87) ~[hibernate-core-6.4.4.Final.jar!/:6.4.4.Final]
    at org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator.getJdbcEnvironmentWithDefaults(JdbcEnvironmentInitiator.java:143) ~[hibernate-core-6.4.4.Final.jar!/:6.4.4.Final]
    at org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator.getJdbcEnvironmentUsingJdbcMetadata(JdbcEnvironmentInitiator.java:348) ~[hibernate-core-6.4.4.Final.jar!/:6.4.4.Final]
    at org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator.initiateService(JdbcEnvironmentInitiator.java:107) ~[hibernate-core-6.4.4.Final.jar!/:6.4.4.Final]
    at org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator.initiateService(JdbcEnvironmentInitiator.java:68) ~[hibernate-core-6.4.4.Final.jar!/:6.4.4.Final]
    at org.hibernate.boot.registry.internal.StandardServiceRegistryImpl.initiateService(StandardServiceRegistryImpl.java:130) ~[hibernate-core-6.4.4.Final.jar!/:6.4.4.Final]
    at org.hibernate.service.internal.AbstractServiceRegistryImpl.createService(AbstractServiceRegistryImpl.java:263) ~[hibernate-core-6.4.4.Final.jar!/:6.4.4.Final]
    ... 38 common frames omitted
```

lucyrez@MacBook-Air-Lucy target %

Решение: docker-compose

Файл docker-compose позволяет объединить несколько контейнеров в один «стек» и связать их между собой.

Мы реализуем подобную схему, где API и БД будут находиться в разных контейнерах, но общаться между собой.



Пишем docker-compose

Шаг 1: Создаем файл docker-compose.yml в корне проекта (на одном уровне с Dockerfile)

Шаг 2: Описываем в файле 2 контейнера – базу данных и веб-приложение.

Шаг 3: Не забываем сменить настройки подключения к БД в application.properties/application.yml (после изменений придется пересобрать jar)

Описываем сервисы

При описания контейнера БД указываем образ СУБД для скачивания. Не забываем открыть порты у контейнеров.

В описании контейнера API указываем “build: .” вместо image, чтобы собрать образ из Dockerfile.

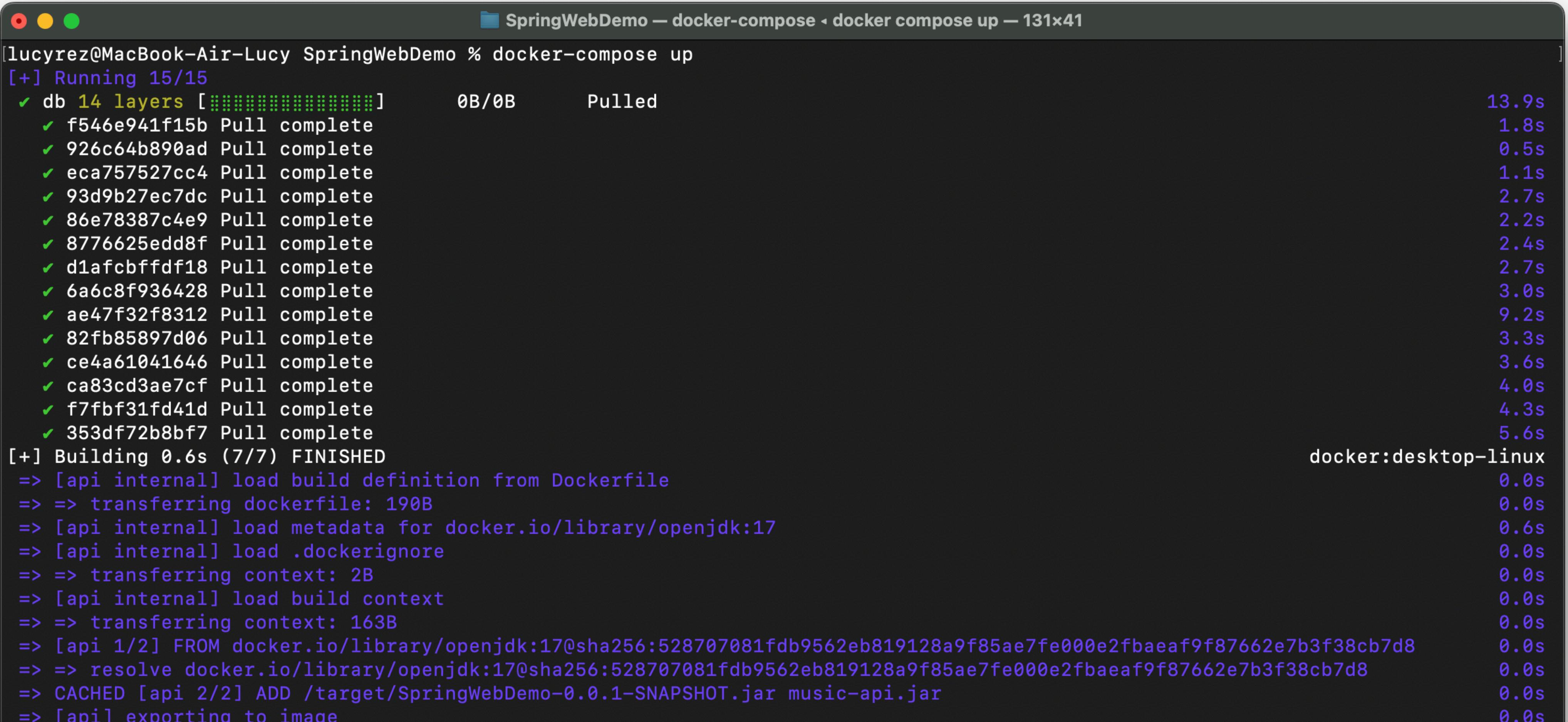
```
services:  
  db:  
    ports:  
      - 5432:5432  
    container_name: postgresDB  
    image: postgres  
    environment:  
      # Here you can add URL, Authentication configuration
```

```
api:  
  build: .  
  container_name: musicAPI  
  ports:  
    - 8080:8080  
  environment:  
    # Here you can add DB configuration
```

Запускаем docker-compose

После написания docker-compose, можем поднять стек контейнеров одной командой:

```
docker-compose up
```



The screenshot shows a terminal window titled "SpringWebDemo – docker-compose ▾ docker compose up – 131x41". The command entered is "docker-compose up". The output details the pulling of 14 layers for the "db" service, which are all completed. It then moves on to building the "api" service, which is finished in 0.6s. The build process involves transferring Dockerfiles, metadata, context, and finally building the image from the openjdk:17 base image, adding the target JAR file, and exporting the image.

```
[lucyrez@MacBook-Air-Lucy SpringWebDemo % docker-compose up
[+] Running 15/15
✓ db 14 layers [████████████████]    0B/0B      Pulled
  ✓ f546e941f15b Pull complete          13.9s
  ✓ 926c64b890ad Pull complete          1.8s
  ✓ eca757527cc4 Pull complete          0.5s
  ✓ 93d9b27ec7dc Pull complete          1.1s
  ✓ 86e78387c4e9 Pull complete          2.7s
  ✓ 8776625edd8f Pull complete          2.2s
  ✓ d1afcbffdf18 Pull complete          2.4s
  ✓ 6a6c8f936428 Pull complete          2.7s
  ✓ ae47f32f8312 Pull complete          3.0s
  ✓ 82fb85897d06 Pull complete          9.2s
  ✓ ce4a61041646 Pull complete          3.3s
  ✓ ca83cd3ae7cf Pull complete          3.6s
  ✓ f7fbf31fd41d Pull complete          4.0s
  ✓ 353df72b8bf7 Pull complete          4.3s
  ✓
[+] Building 0.6s (7/7) FINISHED
=> [api internal] load build definition from Dockerfile      docker:desktop-linux
=> => transferring dockerfile: 190B                           0.0s
=> [api internal] load metadata for docker.io/library/openjdk:17   0.0s
=> [api internal] load .dockerignore                         0.6s
=> => transferring context: 2B                            0.0s
=> [api internal] load build context                      0.0s
=> => transferring context: 163B                          0.0s
=> [api 1/2] FROM docker.io/library/openjdk:17@sha256:528707081fdb9562eb819128a9f85ae7fe000e2fbaeaf9f87662e7b3f38cb7d8 0.0s
=> => resolve docker.io/library/openjdk:17@sha256:528707081fdb9562eb819128a9f85ae7fe000e2fbaeaf9f87662e7b3f38cb7d8 0.0s
=> CACHED [api 2/2] ADD /target/SpringWebDemo-0.0.1-SNAPSHOT.jar music-api.jar                                0.0s
=> [api] exporting to image                                 0.0s
```

Итог: что нужно для запуска приложения

Для запуска приложения «ассистенту» теперь нужно три* вещи:

1. Получить .jar архив от студента
2. Вместе с ним получить Dockerfile и docker-compose.yml файлы
3. Запустить контейнеры

* К сожалению, ассистентам все равно придется смотреть исходный код. Жизнь несправедливая штука:(

Задание

Это простое задание на откачивание навыка сборки проекта и его запуска в docker-контейнере.

Вам потребуется запустить [вот этот проект](#) (с прошлого семинара), используя docker-compose. БД будет выступать в качестве отдельного контейнера.

Требования к БД: имя пользователя – testUser, пароль – 229, название контейнера – DBContainer.

Требования к Web-приложению: слушает запросы на порту 80, название контейнера – WebContainer.

(желательно, чтобы данные для аутентификации в БД передавались через переменные среды в docker-compose)

Дополнительные материалы

- 1) Контейнеризация Spring Boot приложения с PostgreSQL: <https://potatoes.hashnode.dev/how-to-dockerize-a-spring-boot-app-with-postgresql>
- 2) Gradle vs. Maven: <https://habr.com/ru/companies/otus/articles/593903/>
- 3) Разница между JAR и WAR: <https://itsobes.ru/JavaSobes/v-chiom-raznitsa-mezhdu-jar-i-war/>