Lesson 31

22.07.2021

```
static int div(int a, int b) {
        try {
            return a / b;
        } catch (RuntimeException rex) {
            return -1;
        } catch (ArithmeticException aex) {
            return 0;
        } finally {
            System.out.println("DONE");
        }
    }
```

```
public class Ex2 {
    public static void main(String[] args) {
        Robot r = new Robot();
        r.printName(1);
        r.printName(1.0);
    }
}

class Robot{
    public void printName(double d){
        System.out.println("I am a Robot");
    }
}
```

```
public class Ex3 {
    public static void main(String[] args) {
        Integer i = new Integer(null);
        String s = new String(null);
    }
}
```

```
class Ex4
{
    void method(int i)
    {

    }
}

class Ex4_1 extends Ex4
{
    @Override
    void method(Integer i)
    {

    }
}
```

@Аннотации

Аннотации представляют из себя дескрипторы, включаемые в текст программы, и используются для хранения метаданных программного кода, необходимых на разных этапах жизненного цикла программы. Информация, хранимая в аннотациях, может использоваться соответствующими обработчиками для создания необходимых вспомогательных файлов или для маркировки классов, полей и т.д.

@Retention

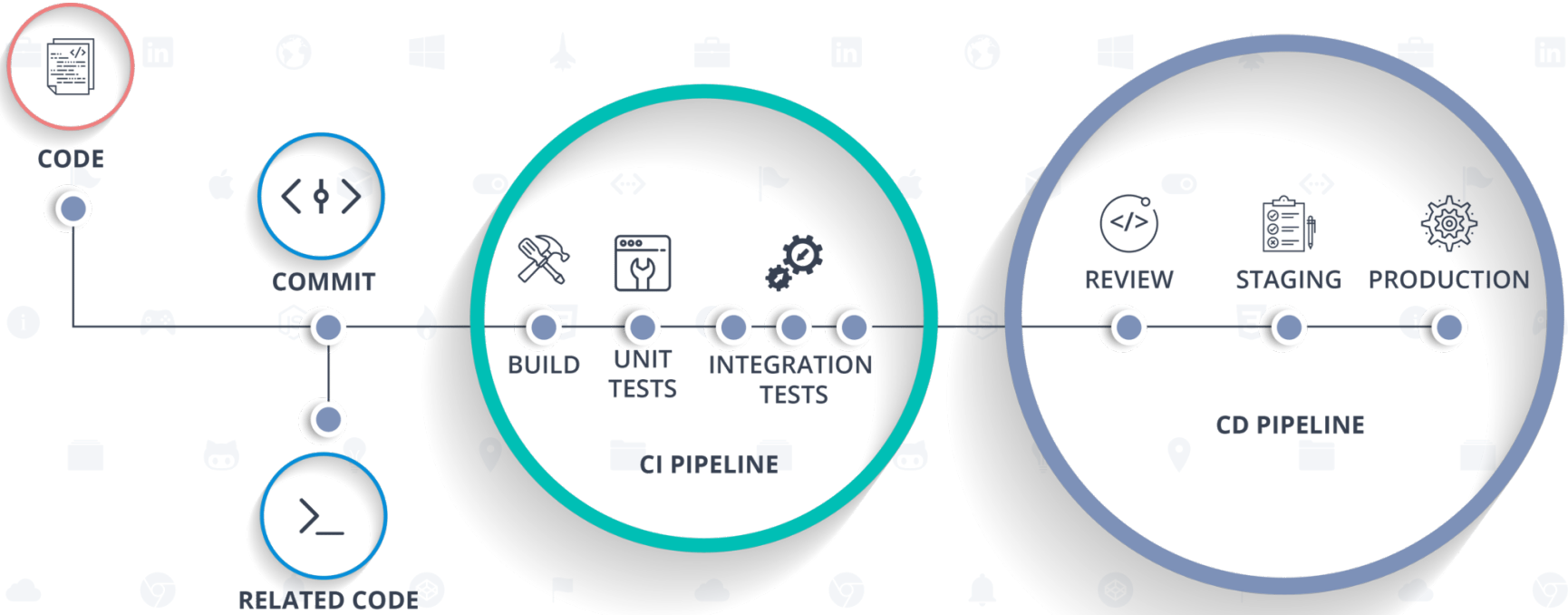
Аннотация @Retention позволяет определить жизненный цикл аннотации : будет она присутствовать только в исходном коде, в скомпилированном файле, или она будет также видна и в процессе выполнения. Выбор нужного типа аннотации **@Retention** зависит от того, как будет использоваться данная аннотацию. Например, генерировать что-то побочное из исходных кодов, или в процессе выполнения "стучаться" к классу через reflection.

RetentionPolicy.**SOURCE**	аннотация используется на этапе компиляции и должна отбрасываться компилятором
RetentionPolicy.**CLASS**	аннтоация будет записана в class-файл компилятором, но не должна быть доступна во время выполнения (runtime)
RetentionPolicy.**RUNTIME**	аннотация будет записана в class-файл и доступна во время выполнения через reflection

@Target

Параметр @Target указывает, что именно должно быть помечено аннотацией. Это может быть поле, метод, тип и т.д. Для этого следует использовать параметры к аннотации.

@Target(ElementType.PACKAGE)	только для пакетов
@Target(ElementType.TYPE)	только для классов
@Target(ElementType.CONSTRUCTOR)	только для конструкторов
@Target(ElementType.METHOD)	только для методов
@Target(ElementType.FIELD)	только для атрибутов(переменных) класса
@Target(ElementType.PARAMATER)	только для параметров метода
@Target(ElementType.LOCAL_VARIABLE)	только для локальных переменных


CODE
COMMIT
RELATED CODE
BUILD
UNIT TESTS
INTEGRATION TESTS
CI PIPELINE
REVIEW
STAGING
PRODUCTION
CD PIPELINE

Что такое непрерывная интеграция

Непрерывная интеграция (Continuous Integration) в разработке программ это автоматизированный процесс сборки и тестирования кода в разделяемом репозитории. Когда делаются новые коммиты, они изолируются, собираются и тестируются на соответствие определенным стандартам прежде чем вольются в основную кодовую базу.

Непрерывная интеграция позволяет быстро выявлять поломки, ошибки или баги, при этом все перечисленное не попадает в кодовую базу, а исправляется как можно скорее. Команды разработчиков, да и все, желающие стать разработчиками, должны разбираться в том, как работают системы непрерывной интеграции.

Обычно непрерывная интеграция сопряжена с непрерывной доставкой (Continuous Delivery), поэтому этапы непрерывной автоматизированной доставки исполняемого кода в продакшен часто обозначают CI/CD.

Преимущества непрерывной интеграции

Непрерывная интеграция обеспечивает множество преимуществ вашей компании, среди которых:

- Действительно раннее обнаружение проблем и исправление их до слияния кода.
- Более короткие и менее напряженные интеграции.
- Благодаря улучшению видимости повышается эффективность коммуникации.
- На поиск багов уходит меньше времени.
- Вам больше не нужно ждать, пока код протестируется.
- Повышается эффективность быстрой доставки ПО.
- Делается возможным непрерывный фидбэк по изменениям, что со временем может улучшить продукт.

GitLab

Bamboo

Docker — программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации. Позволяет «упаковать» приложение со всем его окружением и зависимостями в контейнер, который может быть перенесён на любую Linux-систему, а также предоставляет среду по управлению контейнерами.

Зачем нам нужен Docker?

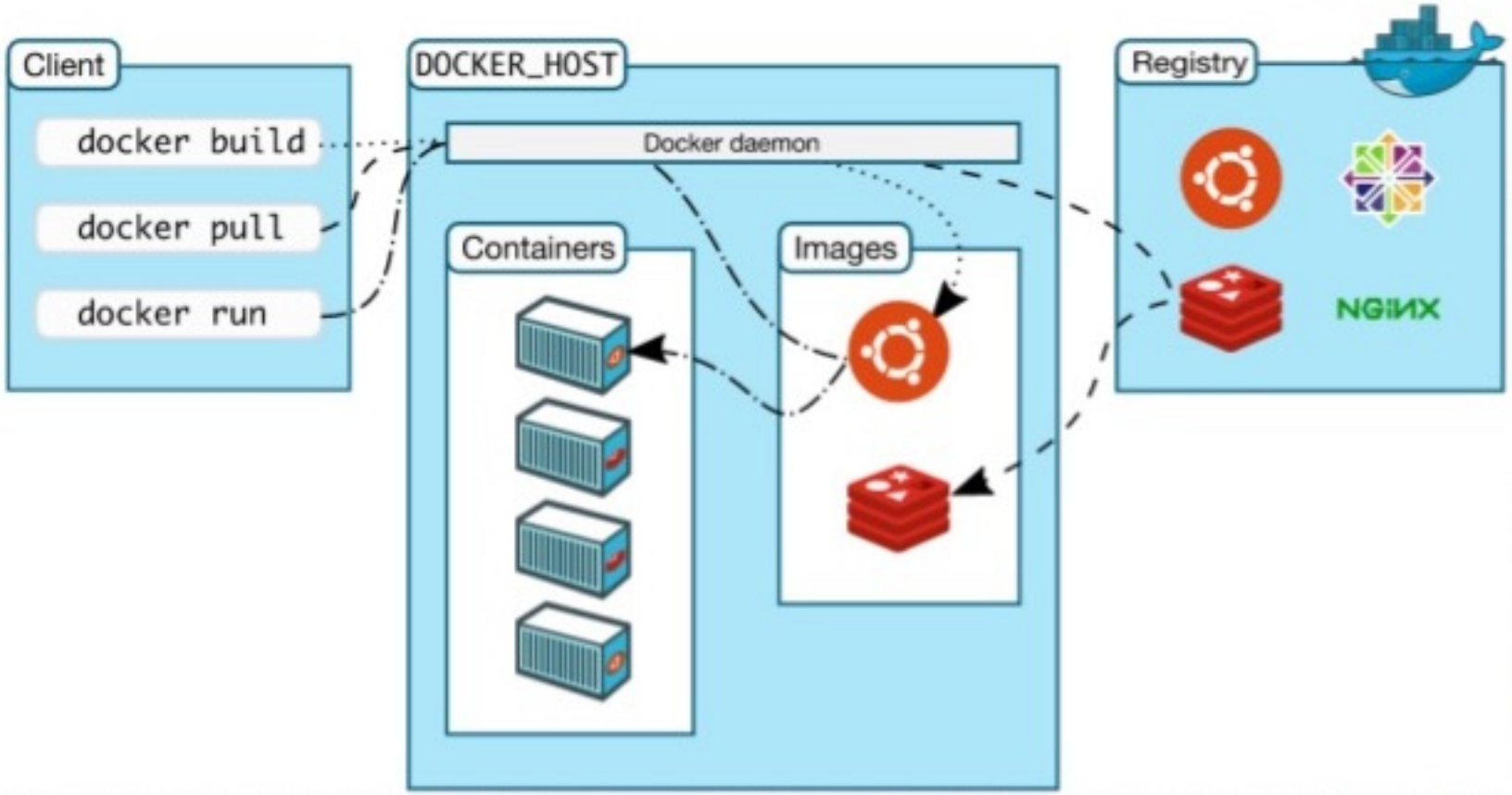
- Быстрый процесс разработки. Нет нужды в установке сторонних программ вроде PostgreSQL, Redis, Elasticsearch. Они могут быть запущены в контейнерах.

- Удобная инкапсуляция приложений. Вы можете предоставить свое приложение как единое целое.

- Одинаковое поведение на локальном компьютере и тестовом, производственном сервере.

- Простой и понятный мониторинг.

- Легко масштабируется. Если вы сделали свое приложение правильно, то оно будет готово к масштабированию не только в Docker.


Client
docker build
docker pull
docker run
DOCKER_HOST
Docker daemon
Containers
Images
Registry
NGINX

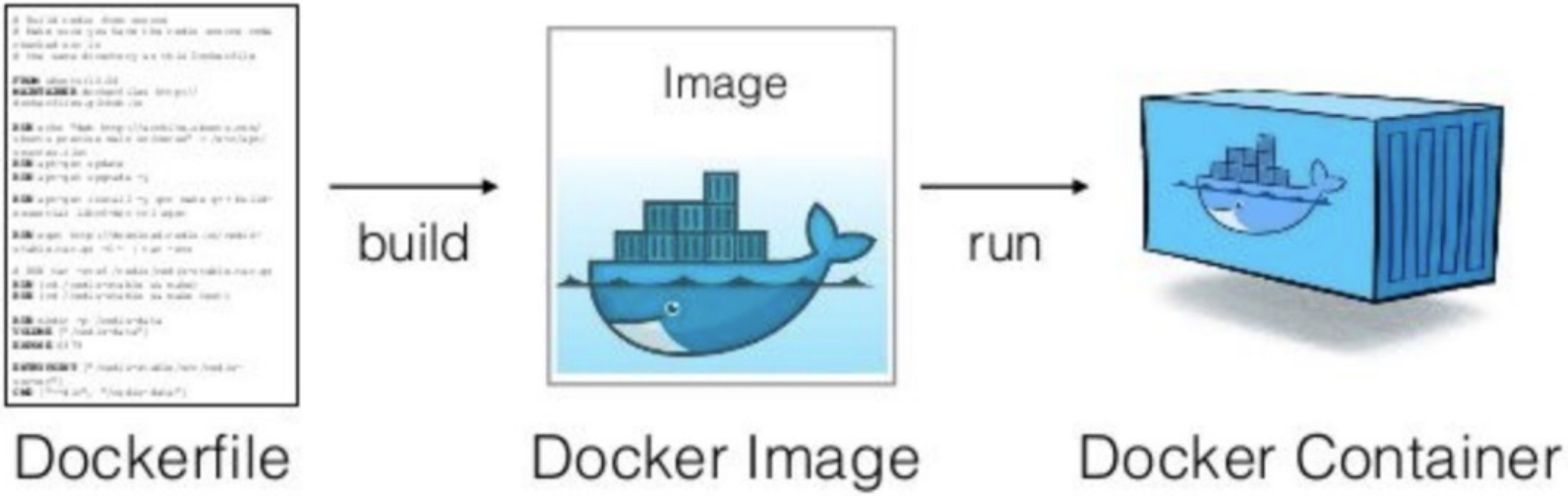
Основные компоненты Docker:

Контейнеры – изолированные при помощи технологий операционной системы пользовательские окружения, в которых выполняются приложения. Разработчики проекта Docker исповедует принцип: один контейнер – это одно приложение.

Образы – доступные только для чтения шаблоны приложений. Поверх существующих образов могут добавляться новые уровни, которые совместно представляют файловую систему, изменяя или дополняя предыдущий уровень. Обычно новый образ создается либо при помощи сохранения уже запущенного контейнера в новый образ поверх существующего, либо при помощи специальных инструкций для утилиты **dockerfile**. Для разделения различных уровней контейнера на уровне файловой системы могут использоваться **AUFS, btrfs, vfs и Device Mapper**.

Реестры (registry), содержащие репозитории (**repository**) образов, – сетевые хранилища образов. Могут быть как приватными, так и общедоступными. Самым известным реестром является Docker Hub.

Image vs. Container



Dockerfile – это скрипт, содержащий набор команд *dockerfile* и операционной системы (то есть команд Linux). Прежде чем создать свой dockerfile, давайте познакомимся с этими командами. Вот наиболее важные из них:

FROM — Указывает базовый образ для создания нового образа. **Эта команда должна быть первой в dockerfile**.

MAINTAINER — Необязательная команда, указывает имя владельца образа.

RUN — Используется для выполнения команды в ходе сборки образа.

ADD — Скопировать файл из файловой системы хоста в новый образ. Можно указать URL файла, в этом случае Docker загрузит его в заданную директорию.

ENV — Определяет переменную среды.

CMD — Команда которая будет запущена при создании нового контейнера на основе образа.

ENTRYPOINT — Задает команду по умолчанию, которая будет выполнена при запуске контейнера.

USER — Задает пользователя или UID для создаваемого на основе образа контейнера.

VOLUME — Монтирует директорию хоста в контейнер.

EXPOSE — Указывает какие порты будут слушаться в контейнере.

```
FROM ubuntu:latest
RUN apt-get update
RUN apt-get install --no-install-recommends --no-install-suggests -y curl
ENV SITE_URL https://google.com/
WORKDIR /data
VOLUME /data
CMD sh -c "curl -L $SITE_URL > /data/results"
```

Образы создаются из Dockerfile с помощью команды ***docker build***.

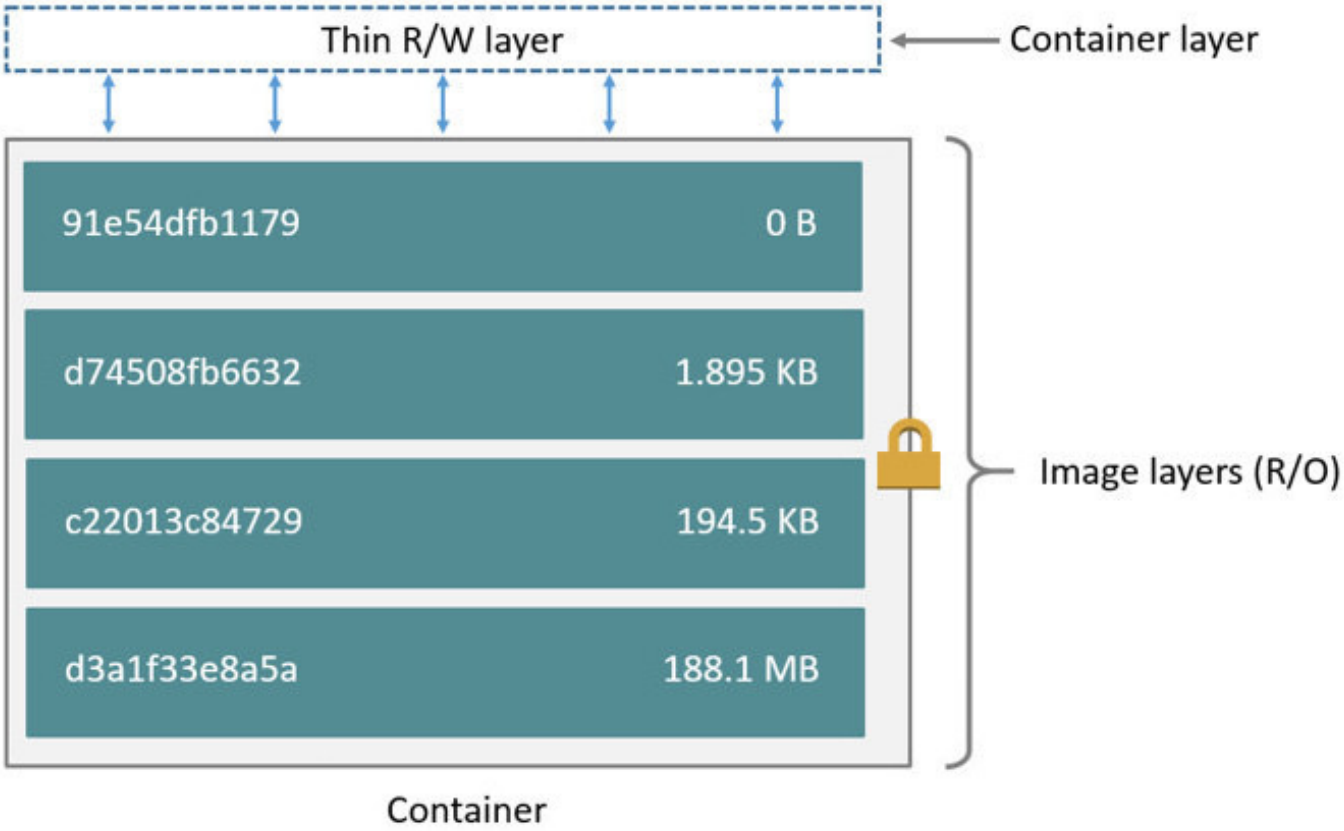
Образы хранятся в Docker-реестрах, как например Docker Hub и их можно скачать с помощью команды ***docker pull***

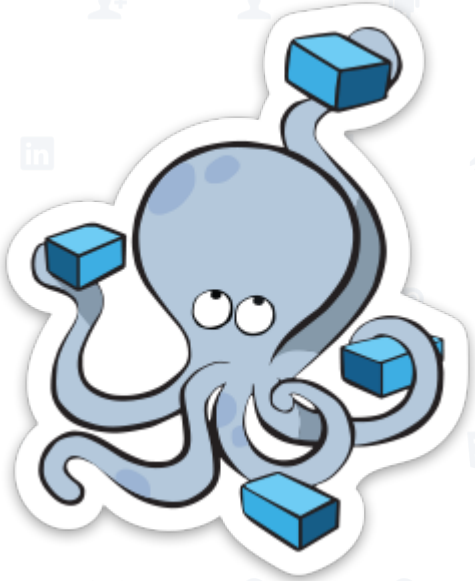
Чтобы просмотреть скачанные Docker-образы, выполните ***docker images***

Образы в Docker так устроены, что они состоят из нескольких слоев. Каждая инструкция из Dockerfile создает новый слой образа.

Каждый слой представляет собой набор отличий (diff) от предыдущего слоя.
Чтобы просмотреть все слоя образа, выполните команду ***docker history***

Контейнеры создаются из образов с помощью команды ***docker run***, а выполнив команду ***docker ps*** можно узнать какие контейнеры в данный момент запущены.





kubernetes

Программное обеспечение для автоматизации развёртывания, масштабирования контейнеризированных приложений и управления ими