Что такое DockerFile?

Docker позволяет вам делиться с другими средой, в которой ваш код запускался и помогает в её простом воссоздании на других машинах.

Dockerfile *- это обычный конфигурационный файл, описывающий пошаговое создание среды вашего приложения. В этом файле подробно описывается, какие команды будут выполнены, какие образы задействованы, и какие настройки будут применены. А движок Docker-а при запуске уже распарсит этот файл (именуемый как* ***Dockerfile****), и создаст из него соответствующий образ (Image), который был описан.*

Для начала, создадим файл cli.php в корне проекта с содержимым:

```
<?php
$n = $i = 5;

while ($i--) {
   echo str_repeat(' ', $i).str_repeat('* ', $n - $i)."\n";
}
```

И файл под названием Dockerfile, с содержимым:

```
FROM php:7.2-cli
COPY cli.php /cli.php
RUN chmod +x /cli.php
CMD php /cli.php
```

Имена команд в Dockerfile (выделенные красным) - это синтаксис разметки *Dockerfile*. Эти команды означают:

- **FROM** – Выбор образа(Image), к примеру, *ubuntu:18.10*, в нашем коде используется образ *php:7.2-cli*, потому весь код будет запускаться внутри образа с предустановленным php 7.2-cli.

- **COPY** - Копирует файл с основной системы в контейнер (копируем файл *cli.php* внутрь контейнера, с одноимённым названием)
- **RUN** - Выполнение shell-команды из терминала контейнера (в текущем случае, присвоим права на выполнение скрипта */cli.php*
- **CMD** - Выполняет эту команду каждый раз, при новом запуске контейнера

Для просмотра полного списка команд можете перейти по ссылке

Для создания образа из *Dockerfile* нужно выполнить:
`docker build <DOCKERFILE_PATH> --tag <IMAGE_NAME>`
<DOCKERFILE_PATH> - путь к файлу *Dockerfile* (`.` - текущая директория),
<IMAGE_NAME> - имя, под которым образ будет создан
Выполним:

```
docker build . --tag pyramid
```

*При том, что имя файла **Dockerfile** при указывании пути упускается, нужно указывать только директорию, в которой этот файл находится (а `.` означает, что файл находится в той директории, из которой была запущена консоль)*

После того, как команда выполнилась, мы можем обращаться к образу по его имени, которое было указано в **<IMAGE_NAME>**, проверим список образов: `docker images`

Теперь, запустим контейнер из нашего образа командой `docker run pyramid`

Shell скрипт был успешно скопирован, и выполнен благодаря указанному в *Dockerfile* параметру **CMD**.

Нам бы хотелось, чтобы можно было удобно изменять количество строк, из скольки состоит пирамида. Для этого, отредактируем файл *cli.php*, и изменим, чтобы количество аргументов принималось из командной строки.
Отредактируем вторую строку на:

```
$n = $i = $argv[1] ?? 5; //а было $n = $i = 5
// это значит, что мы принимаем аргумент из консоли, а если он не получен, то используем по-умолчанию 5
```

После чего, пересоберём образ: `docker build . --tag pyramid`
И запустим контейнер: `docker run pyramid php /cli.php 9`, получив вывод пирамиды в 9 строк

Почему это работает?
Когда контейнер запускается, вы можете переопределить команду записанную в *Dockerfile* в поле **CMD**.
Наша оригинальная **CMD** команда, записанная в Dockerfile `php /cli.php` - будет переопределена новой `php /cli.php 9`.
Но, было бы неплохо передавать этот аргумент самому контейнеру, вместо переписывания всей команды.
Перепишем так, чтобы вместо команды `php /cli.php 7` можно было передавать просто аргумент-число.
Для этого, дополним **Dockerfile**:

```
FROM php:7.2-cli
COPY cli.php /cli.php
RUN chmod +x /cli.php
ENTRYPOINT ["php", "/cli.php"]
## аргумент, который передаётся в командную строку
CMD  ["9"]
```

Мы немного поменяли формат записи. В таком случае, *CMD* будет добавлена к тому, что выполнится в *ENTRYPOINT*.
`["php", "/cli.php"]` на самом деле запускается, как `php /cli.php`. И, учитывая то, что **CMD** будет добавлена после выполнения текущей, то итоговая команда будет выглядеть как: `php /cli.php 9` - и пользователь сможет переопределить

этот аргумент, передавая его в командную строку, во время запуска контейнера.
Теперь, заново пересоберём образ

```
docker build . --tag pyramid
```

И запустим контейнер с желаемым аргументом

```
docker run pyramid 3
```