**Контейнеры**

Контейнеры представляют собой средства инкапсуляции приложения с его зависимостями. контейнер содержит изолированный экземпляр операционной системы (ОС), который можно использовать для запуска приложений.

Преимущества:

* Устранение ошибок, обусловленных изменениями рабочей среды. Теперь нельзя будет сказать “это работает на моем компьютере”.
* Легче чем виртуалка. Можно запускать одновременно десятки контейнеров, и имитировать возможность распределенной системы.

**Состояния контейнера:**

* создан (created) – контейнер инициализирован командой *docker create*, но его работа пока еще не началась.
* перезапуск (restarting)
* активен или работает (running)
* приостановлен (paused)
* остановлен (exited)

**docker run –** команда запускающая контейнер.



Опции:

* -h – задать хост
* --name – задать имя контейнера
* --rm – если такой контейнер уже существует, то удалить его и создать новый. (по умолчанию если контейнер с таким именем уже существует, будет запущен существующий).
* -d – запустить контейнер в фоновом режиме. (Без этого флага мы попадем в терминал контейнера, и после закрытия этого терминала контейнер остановится.)
* -v – запуск контейнера с volume.
* -p – пробросить порты между из контейнера на хостовую машину.  
  
* -P – механизм Docker должен автоматически выбрать свободный порт для перенаправления с хоста в контейнер. Чтобы определить номера портов, назначенные Докером, можно выполнить команду **docker port <container>**

После выполнения команды, докер загрузит образ Debian (если мы его еще не загружали ранее), и выполнит в нем команду echo.

**Образ Docker –** файл или шаблон для запуска контейнера. Он включает в себя исполняемые файлы, зависимости, конфигурацию и другие компоненты. Образы являются неизменяемыми и версионируются. После изменения образа создается его новая версия. Это позволяет управлять версиями приложений, и легко воспроизводить окружение в любой момент времени.

**docker ps –** посмотреть список запущенных контейнеров. В списке отображается ID контейнера; команда, выполняемая внутри контейнера; пробрасываемые порты; docker-образ; время, прошедшее с момента запуска контейнера; статус контейнера; имя контейнера (удобно для чтения).

**docker inspect <containerName | containerId> -** детальная информация о контейнере.

**docker logs <container>** - вывести логи контейнера.

**docker stop <container> -** остановить контейнер.

**docker rm <container> -** удалить контейнер.

**docker rm -v –** удалить контейнер и связанные с ним volumes.

**docker diff**

**docker build –** запустить контейнер с образом, описанным в Dockerfile.

**Контекст создания образа –** набор локальных файлов и каталогов, к которым можно обращаться из инструкций ADD или COPY в Dockerfile (может быть и пустым). Указывается как аргумент команды build.

Эти файлы и каталоги передаются в демон Docker как часть процесса создания. Чтобы не получить задержек, не добавляйте в контекст большие каталоги с кучей файлов.

В качестве контекста можно указывать git-репозиторий.



По умолчанию Docker ищет файл Dockerfile в корневом каталоге контекста.

-**f –** указать расположение Dockerfile в контексте.



**.dockerignore –** файл работает аналогично .gitignore. Все файлы, указанные в .dockerignore будут проигнорированы при передаче контекста.

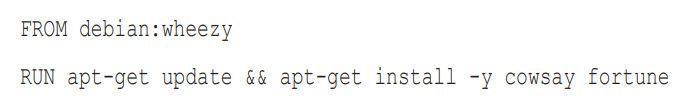
**Dockerfile –** текстовый файл, содержащий набор операций, которые используются для создания Docker-образа.

Инструкции:

**FROM образ –** определяет базовый образ ОС. Обязательна для Dockerfile.

**RUN –** определяет команду, выполняемую в командной оболочке **во время сборки** образа. Может быть использована для установки зависимостей, выполнения сценариев, копирования файлов.

Каждая инструкция RUN создает новый слой в образе, поэтому рекомендуется объединять их в одну строку с помощью &&.



**ENTRY-POINT –** позволяет определить выполняемый файл ли команду, которай будет вызываться при запуске контейнера. В эту выполняемую программу передаются как аргументы любые инструкции *CMD* и аргументы команды *docker run*, записанные после имени образа.



**CMD –** Определяет аргументы по умолчанию для ENTRYPOINT или указывает команду, которая будет выполняться при запуске контейнера, если не определен ENTRYPOINT. CMD может быть переопределен при запуске контейнера с помощью *docker run* с аргументами.

**COPY –** скопировать файл из файловой системы хоста (вашего компьютера) в цифровую систему образа. Первый аргумент определяет файл на вашем компьютере, а второй – целевой путь. Можно использовать шаблонные символы для копирования нескольких фалов или каталогов. Нельзя копировать файлы, находящиеся вне контекста создания.

**ADD –** как и COPY может копировать файл(директорию) из контекста в образ. Помимо этого может автоматически распаковывать архивы и копировать файлы из удаленных URL.

**ENV –** определяет переменные среды внутри образа. На эти переменные можно ссылаться в следующих инструкциях Dockerfile, или во время работы контейнера.



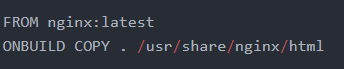
**EXPOSE –** используется для указания портов, которые контейнер будет слушать во время его выполнения. Это необходимо для того, чтобы внешние сервисы или другие контейнеры могли установить соединение с приложением, работающим внутри контейнера.

EXPOSE не выполняет фактического проброса портов, для этого необходимо использовать опцию **-p** при запуске контейнера с помощью *docker run*.

**MAINTAINER –** информация об авторе образа (на случай если заходите загрузить образ в реестр). Эта информация будет доступна в выводе команды *docker inspect*.



**ONBUILD –** используется для определения команд, которые будут выполнены автоматически наследующим образом, во время его сборки. Например, вы можете использовать ONBUILD, чтобы автоматически установить зависимости или скопировать файлы в определенные директории.



ONBUILD не выполняет никаких действий при обработке текущего Dockerfile.

**USER –** используется для указания пользователя, под которым будут выполняться команды внутри контейнера. Может указываться имя пользователя или идентификатор UID. По умолчанию выполняется под root.  
Таким образом мы можем настроить права пользователю и избежать выполнения всех команд под root.

**VOLUME –** указать путь, куда будет смонтирован volume. Если такой файл или каталог уже существует в образе, то он копируется в том при запуске контейнера.

Фактическое монтирование выполняется при запуске контейнера используя опцию **-v** или **-mount** при использовании команды docker run.

**WORKDIR –** определяет рабочий каталог для всех последующих инструкций. То есть все команды далее будут выполнены в этом каталоге. Эту инструкцию можно использовать несколько раз. При использовании относительного пути он будет определятся относительно текущего WORKDIR.

**#** - комментарий

В некоторых инструкциях (RUN, CMD, ENTRYPOINT) допускается использование как формата командной оболочки так и формата *exec*.

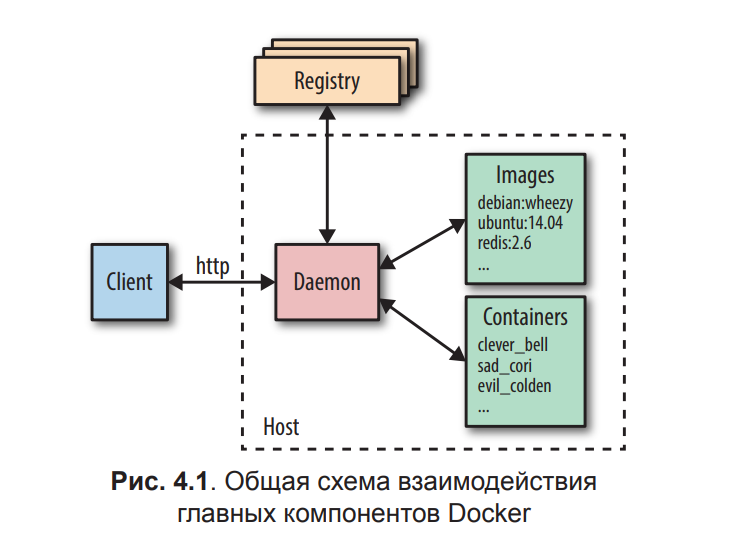
Формат **командной оболочки** – строка произвольной формы, передаваемае интерпретатору /bin/sh.

Формат **exec** принимает JSON-массив, предполагая, что первый элемент массива является именем исполняемого фала, а остальные элементы – параметры. например, *["executable", "param1", "param2"]*.

Используйте формат exec чтобы избежать случайного искажения строк командной оболочкой, или когда образ не содержит bin/sh.

**Архитектура Docker**

* **Docker daemon (**демон Docker**) –** ответственный за создание, запуск и контроль работы контейнеров, и за создание и хранение образов. Запускается командой **docker daemon**, обычно его запускает операционная система.
* **Клиент Docker –** используется для взаимодействия с демоном по протоколу HTTP. Можно без затруднений организовать соединение с удаленными демонами Docker
* **Реестры Docker –** используются для хранения и распространения образов.



**Файловая система**

Docker использует Union File System. Такие файловые системы позволяют подключать несколько файловых систем с наложением друг на друга, причем для пользователя они будут выглядеть как одна файловая система.

Docker поддерживает несколько различных реализаций: UnionFS, включая AUFS, Overlay, devicemapper, BTRFS и ZFS. Реализацию можно посмотреть с помощью команды **docker info** в заголовке **«Storage Driver»**. Файловую систему можно заменить, но это рекомендуется только в тех случаях, когда вы точно знаете, что делаете.

Образы Докера состоят из нескольких уровней. Каждый уровень представляет собой защищенную от записи файловую систему. Для каждой инструкции в Dockerfile создается свой уровень, который размещается поверх предыдущих уровней. Во время преобразования образа в контейнер (командой docker run или docker create) механизм Docker выбирает нужный образ и добавляет на самом верхнем уровне файловую систему с возможностью записи (одновременно с этим инициализируются разнообразные параметры настройки, такие как IP-адрес, имя, идентификатор и ограничения ресурсов).

**docker history <образ> -** позволяет увидеть набор уровней, формирующих образ.

Поскольку ненужные уровни значительно увеличивают размеры образов (а для файловой системы AUFS установлен строгий лимит, равный 127 уровням), во многих файлах Dockerfile можно обнаружить попытку свести к минимуму количество уровней посредством записи нескольких команд Unix в одной инструкции RUN.

Для **ускорения создания образов** Docker выполняет кэширование каждого уровня. Это позволяет переиспользовать уже созданные и неизмененные слои образа. Инструкции берутся из кэша при соблюдении условий:

* В кэше была обнаружена предыдущая инструкция;
* В кэше имеется уровень, который имеет в точности ту же инструкцию и предшествующий родительский уровень.

docker build **--no-cache –** запретить кэширование.

**ТОМА (VOLUMES)**

**volumes –** это файлы или каталоги, которые монтируются непосредственно на хост (наш комп).

Тома используются для:

* Сохранение данных между запусками контейнера. Полезно для сохранения состояния БД и файлов конфигурации приложения при перезапуске контейнера.
* Обмен данными между контейнерами. Несколько контейнеров могут иметь доступ к общим данным.
* Получение в контейнере доступа к фалам с хоста.

Запустить контейнер с volume:



**docker volume create <volumeName> -** создать именованный volume. Мы можем управлять именованными томами и использовать их в контейнерах.

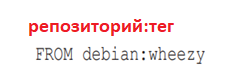


**Работа с реестрами**

Есть официальный реестр Docker Hub, в котором хранятся различные образы докера. А есть и частные реестры. Вы можете использовать образы из реестров в качестве каркаса для своих реестров.

Образы хранятся в следующей иерархии:

* **Реестр –** сервис отвечающий за хранение и распространение образов. По умолчанию используется Docker Hub.
* **Репозиторий –** набор взаимосвязанных образов (обычно представляющих различные версии одного приложения или сервиса)
* **Тег –** алфавитно-цифровой идентификатор, присваиваемый образам внутри репозитория.



Если не указывать тег явно, то будет использован тег **latest.** Но лучше использовать явный тег.

**docker pull <образ> -** загрузить образ из репозитория на локальную машину.

**docker push –** загрузить свой образ на Docker Hub. Предварительно нужно создать учетную запись в реестре.

**DOCKER COMPOSE**

**Docker Compose –** инструмент для создания и выполнения приложений, скомпонованных из нескольких Docker-контейнеров. Используется в основном при разработке и тестировании.

**Оркестровка и управление кластером –** при развертывании большого количества контейнеров важно наличие инструментов для контроля и управления всей системой в целом. Каждый новый контейнер должен быть размещен на некотором хосте, его нужно контролировать и обновлять. Система должна правильно реагировать на сбои или изменения нагрузки, перемещая, запуская или останавливая контейнеры.

Основные решения: Kubernetes от Google, Marathon, Fleet от CoreOS (Операционная система, специализирующаяся на запуске контейнеров), Docker Swarm – собственный инструмент Docker.