Docker — это платформа для создания, доставки и запуска приложений в контейнерах. Контейнеры содержат приложения и их зависимости, ведут себя подобно виртуальным машинам, но при этом используют ядро операционной системы хоста. Docker улучшает переносимость запускаемых приложений, обеспечивает изоляцию их от хоста и других запущенных приложений.

Архитектура системы Docker:

Docker Engine - это приложение, работающее по клиент-серверной архитектуре. Оно устанавливается на хост-машине. В состав Docker Engine входят три компонента:

* Сервер: Служба Docker под названием dockerd. Обеспечивает обслуживание, исполнение Docker контейнеров, управление образами, сетями и так далее.
* Rest API: Используется для управления dockerd.
* Интерфейс командной строки (CLI): Клиентское приложение, которое используется для отправки команд в dockerd по Rest API. Клиент Docker может взаимодействовать с несколькими серверами.

Реестр Docker - сервер, где хранятся образы Docker. Docker Hub - место, где по умолчанию хранятся образы, его официальный публичный реестр. Также имеется возможность запустить собственный, независимый, частный реестр и наполнять его самостоятельно.

Объекты Docker – единицы, используемые при работе с Docker:

* Образы - шаблоны, доступные только для чтения, с инструкциями по созданию контейнера Docker. Образ Docker можно взять из Docker Hub и использовать как есть, либо добавить дополнительные инструкции с помощью Dockerfile и создать новый модифицированный образ. Также имеется возможность cоздавать собственные образы операционных систем с нуля.
* Контейнеры - создаются запуска образов Docker. Все приложения и их окружение работают внутри контейнера. Запуск, остановка, удаление и иные операции с контейнерами могут осуществляться с помощью Rest API или CLI.
* Тома – хранят данные, генерируемые Docker и используемые контейнерами. Они полностью управляются Docker через CLI или Docker API. Тома работают как с контейнерами Windows, так и с контейнерами Linux. Вместо того чтобы хранить данные в записываемом слое контейнера, либо монтировать папки из ОС хоста, зачастую лучше использовать для этого тома. Содержимое тома существует вне жизненного цикла контейнера, поэтому использование тома не увеличивает размер контейнера, облегчает миграцию и резервное копирование.
* Сети – используются для общения изолированных контейнеров с внешним миром, хостом и другими контейнерами в зависимости от конфигурации.

Процесс создания образов Docker:

* Создание Docker образа начинается с Dockerfile — текстового файла с инструкциями для построения образа.
* Dockerfile содержит команды для установки зависимостей, копирования файлов и других операций.
* После написания Dockerfile выполняется команда docker build, которая создает образ на основе инструкций из файла.

Dockerfile:

* Dockerfile — это текстовый документ, содержащий все команды, которые пользователь может вызвать в командной строке для сборки образа.
* Docker выполняет инструкции в Dockerfile по порядку.
* Dockerfile должен начинаться с инструкции FROM. Инструкция FROM указывает родительский образ, из которого производится сборка.

Контекст создания образа:

* Набор файлов, необходимых для сборки образа.
* Может быть локальной директорией, tar файлом, git репозиторием (в том числе удалённым), либо Dockerfile, содержащим пути до файлов, переданным напрямую.

Уровни образа:

* Образ Docker состоит из слоев. Каждая инструкция в Dockerfile создает новый слой.
* Базовый слой — это минимальная операционная система (например, Alpine или Ubuntu).
* Промежуточные слои добавляются для установки программного обеспечения и зависимостей.
* Слой приложения содержит код приложения и конфигурацию.

UnionFS:

* UnionFS —файловая система, объединяющая несколько файловых систем-ветвей с определённым приоритетом в одну. Это позволяет файлам и каталогам изолированных файловых систем прозрачно перекрываться, модификации и запись при этом будут выполняться над определяемой приоритетом конкретной файловой системой.
* В Docker подобные UnionFS файловые системы используется для создания слоев образов, оптимизации хранения и уменьшения дублирования данных.

Влияние кэширования на процесс создания образов:

* Кэширование позволяет Docker повторно использовать слои из предыдущих сборок.
* Если слой образа не изменился, Docker использует его из кэша.
* Изменения в Dockerfile будут инвалидировать кэш для слоев, следующих за измененным слоем, что вызовет их повторное исполнение.