1. Общая информация

В этом разделе описана технология контейнеризации Docker: архитектура, принципы работы, практическое применение. Контейнеризация приложений это упаковка приложения в отдельный контейнер (программную среду) со всеми необходимыми библиотеками, связями, зависимостями. Контейнеризация приложений очень популярна в разработке программного обеспечения и практических приложениях: анализе данных, развертывании инфраструктуры распределенных систем, эксплуатации. Контейнеризация помогает сделать приложения более безопасными, облегчает их развёртывание в продуктивной среде, и улучшает масштабирование. Технология контейнеризации переживает фазу бурного роста и считается одним из основных направлений развития в индустрии разработки программного обеспечения.

Технология Docker предназначена для разработки, установки и запуска в специальных сущностях - контейнерах. С использованием инструментов Docker контейнеризуются программные продукты, информационные системы, отдельные приложения или масштабные системы со сложной архитектурой, состоящей из множества сервисов, в соответствии с концепцией микросервисной архитектуры.

Технология Docker настолько распространена, что стала фактически стандартом де-факто в контейнеризации, часто используется термин «докеризация».

Технология контейнеризации идеологически похожа на виртуализацию, но есть существенные различия. Виртуальные машины обеспечивают полный уровень изоляции гостевых операционных систем, однако на это расходуется много ресурсов. Принцип работы Docker отличается от принципов работы виртуальных машин тем, что виртуальная машина взаимодействует напрямую с аппаратным обеспечением, а Docker работает с низкоуровневыми инструментами основной операционной системы, позволяя экономить ресурсы и выполнять задачи быстрее. Необходимо отметить, что на практике распространено применение контейнеров docker в виртуальных машинах.

2. Терминология Docker

Docker Платформа (Docker Platform) - это программный комплекс, который упаковывает приложения в контейнеры, запускает контейнеры в аппаратных средах (серверах), управляет логикой работы контейнеров, обеспечивает работу пользователя в системе. Платформа Docker позволяет помещать в контейнеры код и его зависимости (используемые внешние библиотеки, переменные среды окружения, служебные файлы, параметры). При таком подходе упрощается запуск, перенос, воспроизведение, масштабирование систем.

Docker «Движок» (Docker Engine) - это клиент-серверное приложение, обеспечивающее весь цикл работы с технологией Docker. Docker Engine состоит из двух продуктов:

- Docker Community Edition - это бесплатное ПО, основанное на инструментах open-source,
- Docker Enterprise – платное программное обеспечение, предназначенное для использования производственными компаниями в больших коммерческих проектах.

Docker Клиент (Docker Client) – основной инструмент пользователя при работе с Docker. Взаимодействие осуществляется с использованием командной строки Docker CLI (Docker Command Line Interface). В Docker CLI пользователь вводит команды, начинающиеся с ключевого слова «docker», эти команды обрабатываются Docker Клиентом и с использованием API Docker отправляются Docker Демону.

Docker Образ (Docker Image) – набор данных, содержащий:

- образ базовой операционной системы (файловая система, системные настройки, драйверы устройств),
- прикладное программное обеспечение для развертывания в базовой операционной системе, с настройками,
- библиотеки, служебные файлы.

Docker образ используется для создания Docker Контейнера (Docker Container). Различают базовые и дочерние образы:

- Base images (базовые образы) не имеют родительского образа. Обычно это образы с операционной системой, такие как ubuntu, busybox или debian.
- Child images (дочерние образы) построены на базовых образах и обладают дополнительной функциональностью.

Существуют официальные и пользовательские образы (любые из них могут быть базовыми и дочерними):

- Официальные образы официально поддерживаются компанией Docker. Обычно в их названии одно слово (например, python, ubuntu).
- Пользовательские образы создаются пользователями и построены на базовых образах. Формат имени пользовательского образа «имя пользователя»/«имя образа».

Docker Контейнер (Docker Container) - запускаемый экземпляр Docker Образа. В контейнере запускаются приложения со всеми требуемыми взаимосвязями. Контейнер разделяет имеющиеся ресурсы на уровне ядра операционной системы с другими контейнерами, работает изолировано от других контейнеров в своей операционной системе (домашняя ОС, hosted OS).

Docker Демон (Docker Daemon) - сервис, запущенный в фоновом режиме, предназначенный для управления образами, контейнерами, сетями и томами. Взаимодействие с Docker Демоном осуществляется через запросы к API Docker.

Docker Реестр, Docker хаб (Docker Hub) – место хранения образов Docker, облачное хранилище. Многие провайдеры услуг хостинга предоставляют возможность хранить образы Docker в своих репозиториях (например, Amazon, Yandex). Самым популярным и наиболее используемым хранилищем является официальный репозиторий Docker Hub (https://hub.docker.com/), используемый при работе с Docker по умолчанию. Обычно в репозиториях хранятся разные версии одних и тех же образов, обладающих одинаковыми именами и разными тегами (идентификаторами образов), разделяемые двоеточием. Например,

- «python» - официальный репозиторий Python на Docker Hub,
- «python:3.7-slim» - версия образа с тегом «3.7-slim» в репозитории Python.

В реестр можно отправить как целый репозиторий, так и отдельный образ.

Файл Dockerfile – текстовый файл, содержащий упорядоченный перечень команд, необходимых при создании (building) Docker образа (Docker image). Этот файл содержит описание базового образа, который будет представлять собой исходный слой образа. Популярные официальные базовые образы:

alpine	легкая ОС linux, оптимальна для простых задач или обучения, для большинства задач требует дополнительной установки пакетов	https://hub.docker.com/_/alpine
ubuntu	ОС linux с большим набором библиотек и утилит	https://hub.docker.com/_/ubuntu
nginx	самый популярный и очень функциональный web сервер	https://hub.docker.com/_/nginx
python	ОС linux с установленным программным обеспечением для работы с Python	https://hub.docker.com/_/python/

В образ контейнера поверх базового образа можно добавлять дополнительные слои в соответствии с инструкциями из Dockerfile. Если Dockerfile описывает образ, который планируется использовать для решения задач машинного обучения, то в нём могут быть инструкции для включения библиотек NumPy, Pandas и Scikit-learn. В образе может содержаться, поверх всех остальных, ещё один «тонкий» слой, содержащий программу, которую планируется запускать в контейнере.

Пример Dockerfile

```
FROM alpine                    # определяет базовый образ
RUN apk add python;            # добавляет к базовому образу python
COPY test.py /apps             # копирует test.py в директорию /apps образа
CMD ["python", "/apps/test.py"] # запускает выполнение файла test.py
```

Docker Том (Docker Volume) – специальный уровень Docker контейнера, который позволяет передавать данные в Docker контейнер. Docker Том это наиболее предпочтительный механизм постоянного хранения данных, используемых или создаваемых приложениями.

Сетевые механизмы Docker (Docker Networking) организуют связь между контейнерами Docker. Соединённые с помощью Docker Сети (Docker Network) контейнеры могут выполняться на одном и том же хосте или на разных хостах, взаимодействуя между собой как отдельные независимые сервисы.

Docker Compose - инструмент экосистемы Docker, упрощающий работу с многоконтейнерными приложениями. Docker Compose выполняет инструкции, описанные в файле docker-compose.yml.

Kubernetes - технология, автоматизирующая развёртывание и масштабирование контейнеризированных приложений, а также управление ими. В настоящее время Kubernetes - лидер рынка средств для оркестрации контейнеров, опережает по популярности Docker Swarm (аналогичный инструмент экосистемы Docker).

3. Установка docker

Установка Docker для различных операционных систем подробно описана здесь https://docs.docker.com/engine/install/

Несмотря на то, что для Docker созданы инструменты для разных операционных систем, использование Windows – не очень распространённая практика, работать с такими образами сложнее. Поэтому рекомендуется работать с Docker в Unix системах.

4. Базовые команды

docker --version	Посмотреть используемую версию docker
docker container hello-world	Проверка работы docker («hello, world»)
docker ps –a	Посмотреть информацию о контейнерах
docker rm <Container ID>	Удалить контейнер
docker stop <Container ID>	Остановить docker контейнер
docker network	Работа с docker сетями
docker image pull alpine	Загрузить docker образ (docker image) операционной системы alpine
docker image ls	Посмотреть имеющиеся в локальной системе пользователя docker образы (docker image)
docker container ls –a	Посмотреть все имеющиеся docker контейнеры (docker container)
docker container run alpine ls –l	Запустить команду «ls -l» в docker контейнере с docker образом операционной системы alpine
docker container run –it alpine sh	Запустить docker контейнер с docker образом с операционной системой alpine в интерактивном режиме и открыть терминал sh
docker container start <ContainerId>	Запустить docker контейнер с идентификатором <ContainerId>
docker container exec <ContainerId> <Command>	Выполнить команду <Command> в docker контейнере с идентификатором <ContainerId>
docker container diff <ContainerId>	Посмотреть изменения, сделанные в docker контейнере с идентификатором <ContainerId>
docker run -d -P --name <имя контейнера> <имя образа>	Запустить образ в контейнере в фоновом режиме (флаг –d, detached mode), все внутренние порты контейнера сделать внешними открытыми и случайными (флаг – P), внутренние (т.е. относящиеся к приложению в контейнере) и внешние (т.е. те, через которые контейнер общается в внешней средой) порты связываются.
Docker ps –a –q	Вывести идентификаторы всех существующих контейнеров
docker rm $(docker ps –a –q –f status=exited)	Удалить все контейнеры, находящиеся в статусе Exited
docker port <Container ID>	Посмотреть порты, относящиеся к контейнеру
docker image build –t image_name:v1 .	Создать новый образ с тэгом image_name:v1. Последний параметр (точка, «.») указывает на то, что действия происходят в текущей директории (в которой должен находится Dockerfile)
docker container commit <ContainerId>	Создать новый образ на основе измененного контейнера
docker image tag <ContainerId> <Image Name>	Создать тег для образа
docker image history <ContainerId>	Посмотреть историю образа
docker image inspect <ImageName>	Просмотр детализированной информации об образе
docker image inspect –format “{{ json .Os }}” alpine	Просмотр детализированной информации об образе
docker image inspect –format “{{ json	Просмотр детализированной информации об образе

.RootFS.Layers }}" alpine	
docker push <Image name>	Загрузить образ в docker репозиторий
docker network create <имя сети>	Создать сеть docker для обмена сообщениями между контейнерами
docker run -n <имя сети> <имя контейнера>	Запустить контейнер с привязкой к сети

5. Сборка образа

Новый Docker образ создается командой docker build с использованием инструкций в Dockerfile, при этом подразумевается, что Dockerfile находится в текущей рабочей директории. Если Dockerfile находится в другом месте, то его на расположение нужно указать с использованием флага -f. В файлах Dockerfile содержатся следующие инструкции по созданию образа:

Команда	Назначение	Пример использования
FROM	задаёт базовый (родительский) образ	FROM ubuntu
LABEL	описывает метаданные, например сведения о том, кто создал и поддерживает образ	LABEL maintainer="yuchernyshov@usurt.ru "
ENV	устанавливает постоянные переменные среды	ENV PATH="/home/user"
RUN	выполняет команду и создаёт слой образа, используется для установки в контейнер пакетов	RUN pip install numpy
COPY	копирует в контейнер файлы и папки	COPY . /apps
ADD	копирует файлы и папки в контейнер, может распаковывать tar-файлы (архивы)	ADD test.py /apps
CMD	описывает команду с аргументами, которая должна выполниться при запуске контейнера, может быть лишь одна инструкция CMD	CMD ["python", "test.py"]
WORKDIR	задаёт рабочую директорию для следующей за ней инструкции	WORKDIR /apps
ARG	задаёт переменные для передачи Docker во время сборки образа	ARG my_var=1
ENTRYPOINT	предоставляет команду с аргументами для вызова во время выполнения контейнера, аргументы не переопределяются.	ENTRYPOINT ["python", "./script.py"]
EXPOSE	указывает на необходимость открыть порт	EXPOSE 8000
VOLUME	создаёт точку монтирования для работы с постоянным хранилищем	VOLUME /my_volume

6. Практический пример сборки образа и запуска контейнера

При практической работе с Docker необходимо учитывать, что если у пользователя недостаточно прав для выполнения инструкций docker, то эти инструкции необходимо выполнять с правами суперпользователя (superuser), например

sudo docker ps –a

Альтернатива этому – создание группы пользователей docker и наделение пользователей этой группы соответствующими правами. Далее команда sudo опускается.

1. Создание docker-контейнера, запускающего простой python скрипт

   docker container run alpine python

По сообщениям системы видим, что python в образе alpine по умолчанию отсутствует.

2. Создаем файл Dockerfile с содержимым

   FROM alpine
   RUN apk add python
   COPY . /apps
   WORKDIR /apps
   CMD [“python”, “test.py”]
3. Создаем файл test.py, который будет выполняться в контейнере, например

   a = [i2 for i in range(1,11)]**
   print(a)
4. Создаем Docker образ

   docker image build –t test_python:0.1 .
5. Запускаем контейнер с использованием созданного Docker образа

   docker container run test_python:0.1
6. Проверяем статус запущенного контейнера

   docker container ls -a

7. docker-compose

Docker Compose — это инструментальное средство, входящее в состав Docker. Оно предназначено для решения задач, связанных с развёртыванием проектов, состоящих из нескольких независимых совместно работающих приложений.

Установка docker-compose описана здесь: https://docs.docker.com/compose/install/

Команды docker-compose

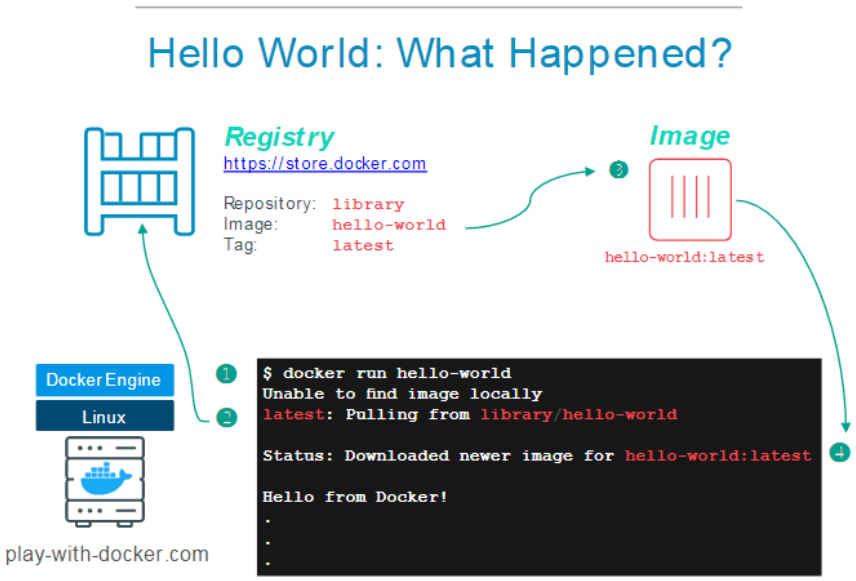
docker-compose build	создать все необходимые docker-образы
docker-compose up	запустить все docker-контейнеры
docker-compose down	остановить все docker-контейнеры
docker-compose logs -f [service name]	посмотреть log-файлы
docker-compose ps	посмотреть все работающие контейнеры
docker-compose exec [service name] [command]	выполнить команду в определенном контейнере-сервисе
docker-compose images	посмотреть все доступные docker-образы

8. Приложения

В этом разделе приведены схемы, иллюстрирующие работу Docker (источник - https://training.play-with-docker.com/)

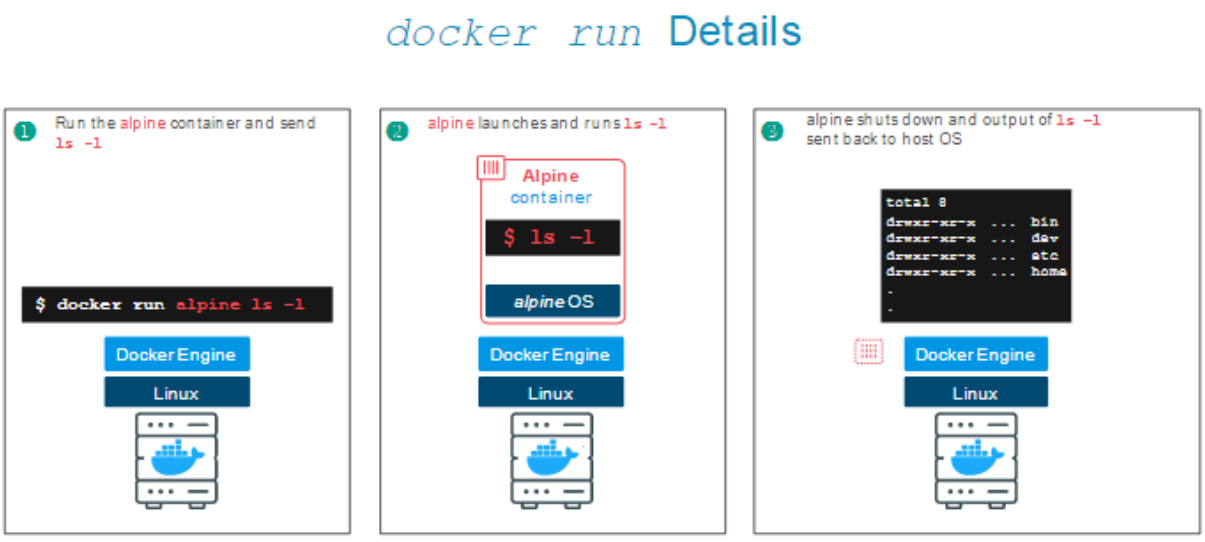
A. Docker “Hello, World”

При выполнении команды **docker run hello-world** выполняется запрос docker-образа hello-world:latest в реестре Docker, этот образ загружается на локальный Docker Engine, запускается в контейнере, в результате выполнения выводится сообщение «Hello from Docker!»



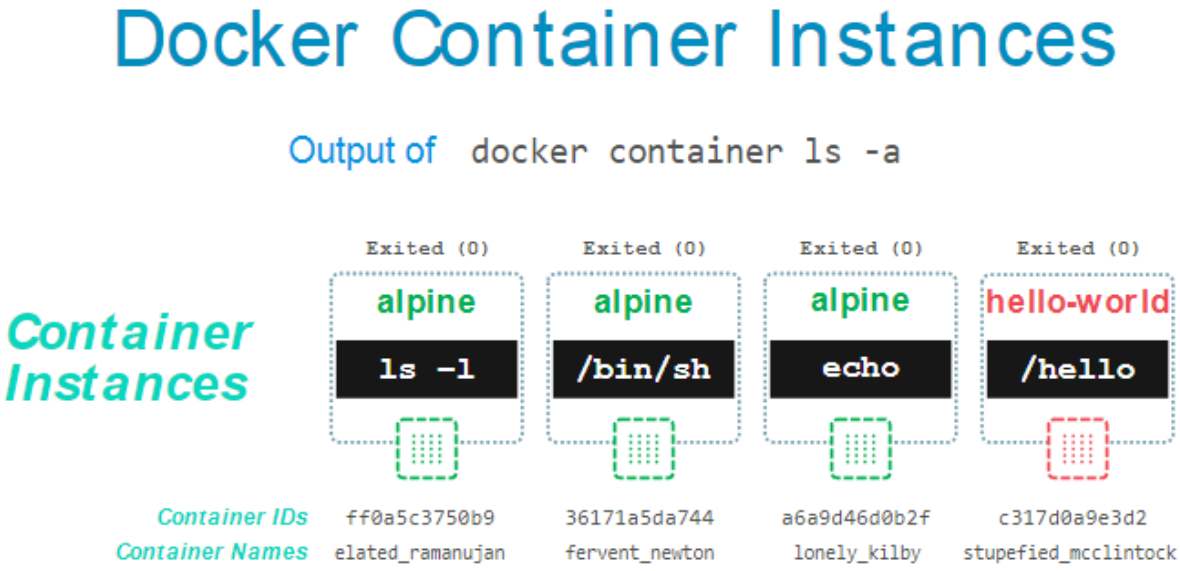
B. Выполнение команды docker run

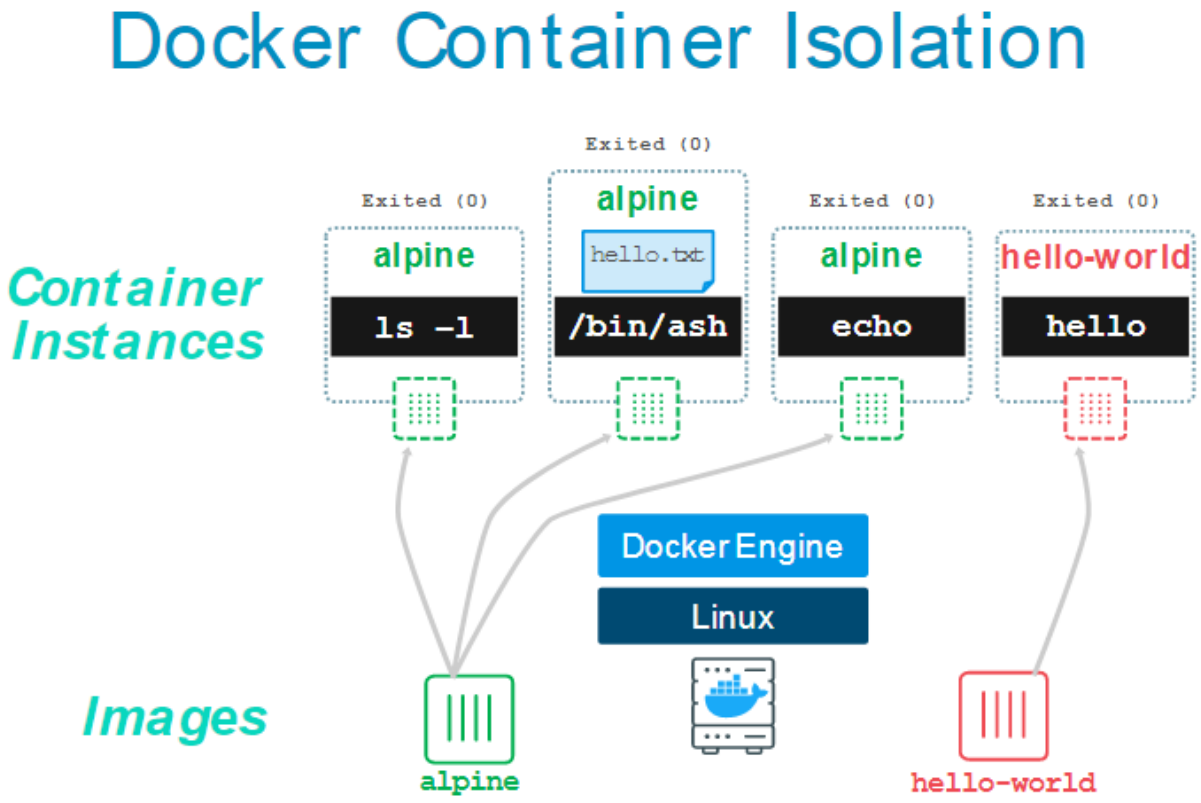
При выполнении команды **docker run <парамеры> <имя образа> Docker Engine** запускает контейнер с указанным в команде образом и параметрами.



C. Docker контейнеры

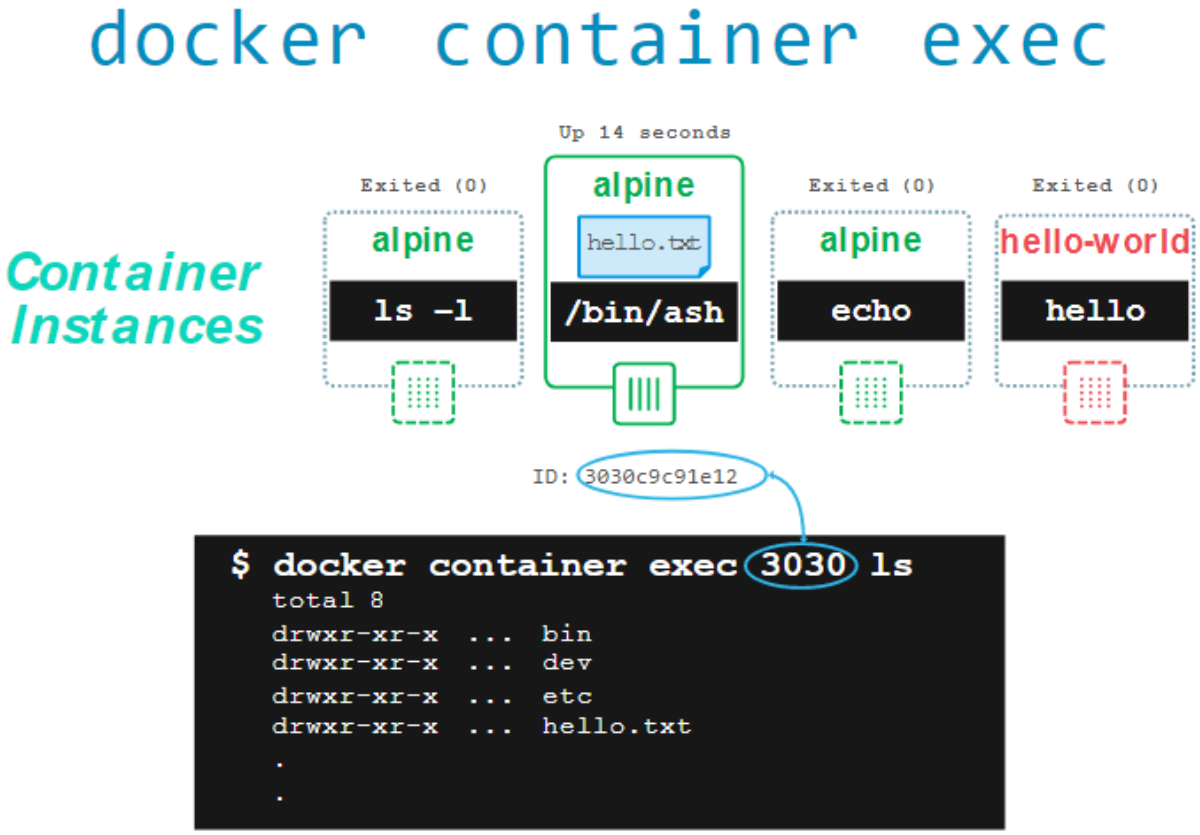
Docker контейнеры, запущенные на выполнение командой docker run, представляют собой раздельные изолированные сущности. У каждого контейнера есть свой идентификатор. Информацию о имеющихся контейнерах можно посмотреть командой **docker container ls -a** (флаг –a указывает на то, что надо посмотреть все контейнеры, а не только выполняющиеся)



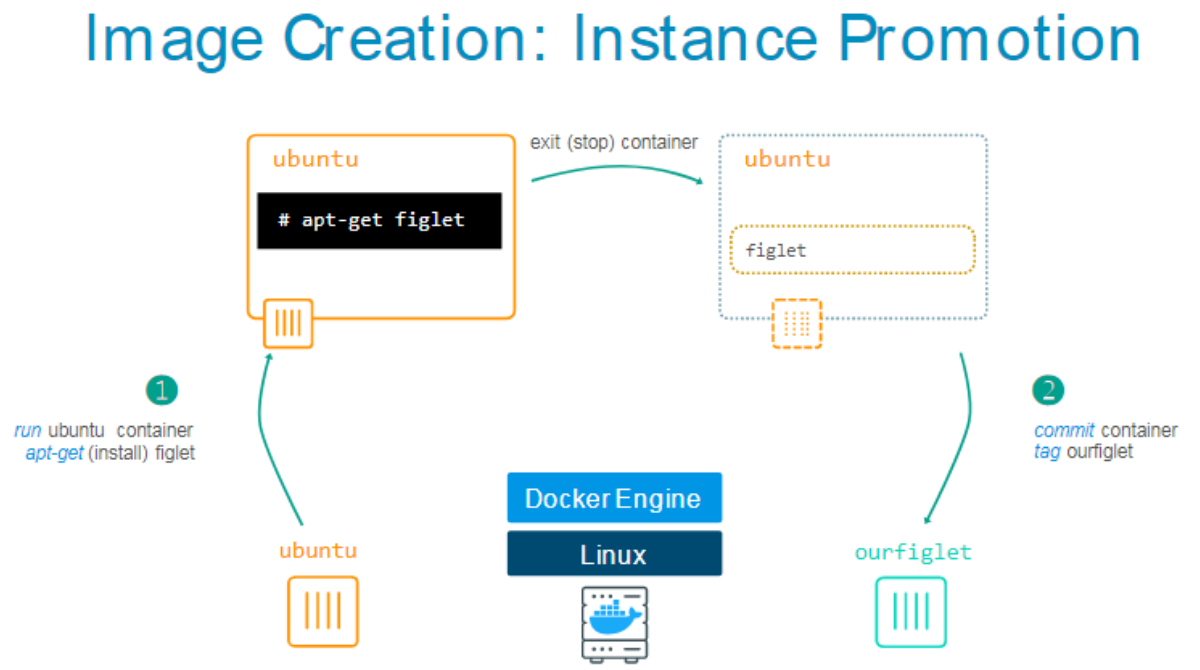


D. Выполнение команд в контейнере

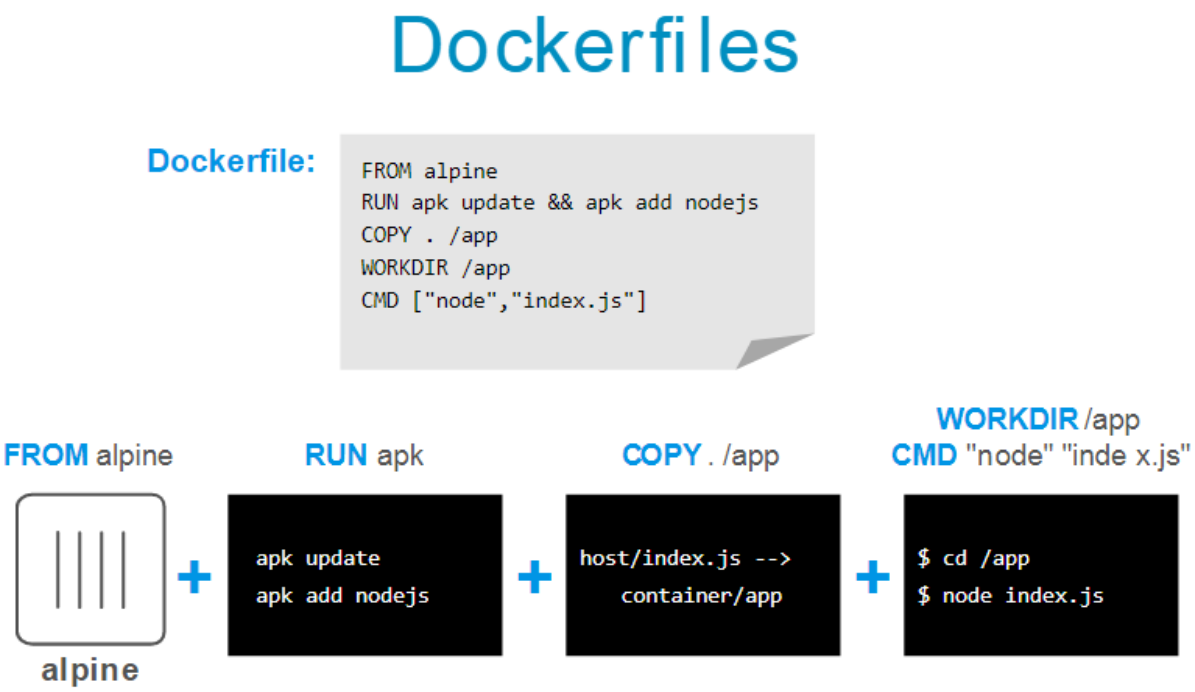
В отдельном контейнере можно выполнять команды, контейнеры идентифицируются номером.



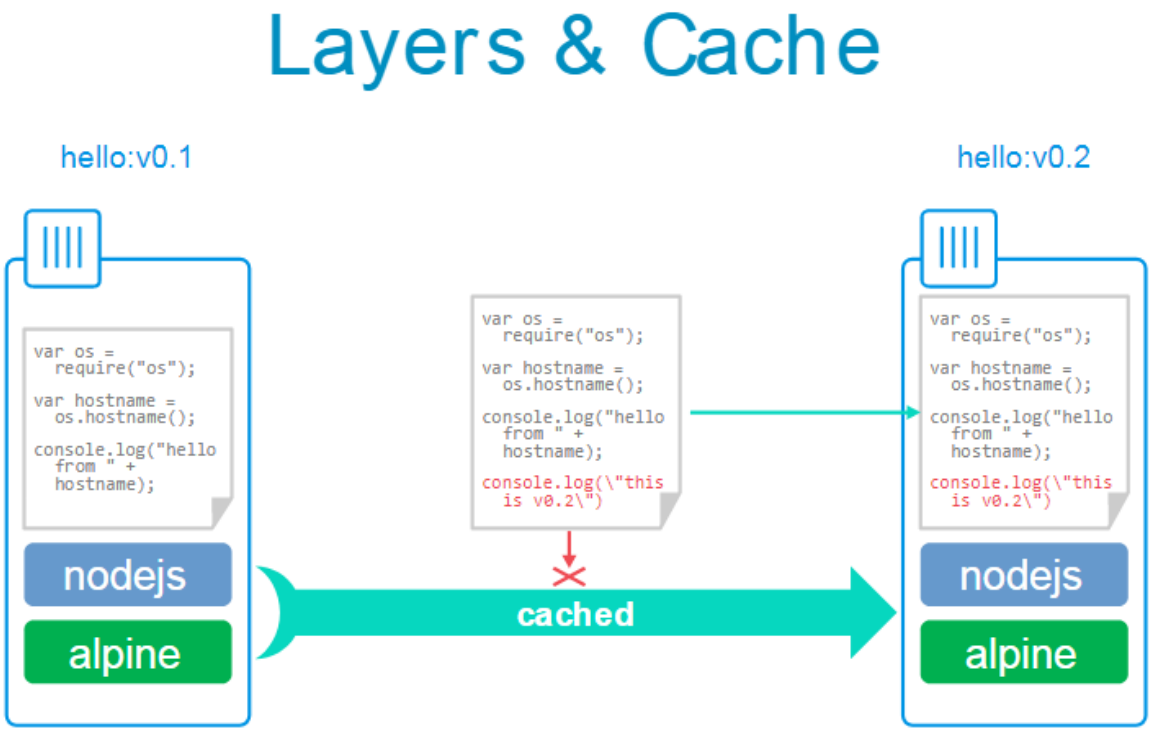
E. Запуск и остановка docker контейнера



F. Dockerfile



G. Слои docker образа



H. Создание нового docker образа