Теория к сдаче

1. **docker build** – `docker build -t <image_name>:<tag> <path_to_dockerfile>` — используется для создания образа Docker из Dockerfile

   EX: `docker build -t myimage:1.0 .` - создание образа с именем "myimage" и тегом "1.0" на основе Dockerfile, находящегося в текущем каталоге.

2. **docker tag --** `docker tag <existing_image_name>:<tag> <new_image_name>:<new_tag>` — используется для создания новой метки (тега) для существующего образа Docker.

   EX: `docker tag myimage:1.0 myimage:latest` — создание новой метку latest для существующего образа myimage с тегом 1.0. Теперь у нас есть две метки (1.0 и latest), указывающие на один и тот же образ.

3. **docker run --** `docker run [flags] <image_name> [command]` – создание и запуск контейнера на основе определенного образа.

   [command] - опциональная команда, которая будет выполнена внутри контейнера при его запуске. Если команда не указана, будет выполнена команда, заданная внутри образа по умолчанию.

   EX: Запуск контейнера с указанием порта хоста и порта контейнера `docker run -p 8080:80 myimage`

4. **docker start** указал идентификатор и остановленный контейнер запустился

5. **docker stop** указал идентификатор и запущенный контейнер остановился

6. **docker pause** указал идентификатор и запущенный контейнер стал на паузу

7. **docker unpause** указал идентификатор и контейнер на паузе опять дышит

8. **docker restart**

9. **docker ps –** показывает все запущенные контейнеры

10. **docker logs –** используется для просмотра логов, сгенерированных контейнером

11. **docker images**

12. **docker network**

    В контексте Docker, сеть представляет собой виртуальное средство связи, которое позволяет контейнерам Docker общаться друг с другом и с внешними ресурсами, такими как хостовая машина или другие сетевые сервисы.

    Сеть Docker может быть создана для группировки и изоляции контейнеров, чтобы они могли обмениваться данными и взаимодействовать друг с другом. Когда контейнеры находятся в одной сети, они могут использовать DNS для обнаружения других контейнеров по их именам и общаться между собой с помощью сетевых протоколов.

    Сеть Docker также может предоставлять контейнерам доступ к внешним ресурсам или сетевым сервисам, таким как базы данных, веб-серверы или другие контейнеры. Контейнеры, подключенные к одной сети, могут общаться с любыми другими контейнерами в этой сети, независимо от того, находятся ли они на одной хостовой машине или на разных.

```
Usage:  docker network COMMAND
Commands:
  connect <network> <container> Connect a container to a network
  create <network> -- Create a network
  disconnect <network> <container>  Disconnect a container from a network
  inspect <network> -- Display detailed information on one or more networks
  ls            List networks
  prune         Remove all unused networks
  rm            Remove one or more networks. All containers connected to that network will be disconnected from it.
```

13. **docker volume**

Volume в контексте Docker - это механизм для сохранения и управления данными, используемыми контейнерами Docker. Volume представляет собой отдельное хранилище данных, которое может быть присоединено к одному или нескольким контейнерам.

Основная цель использования Docker volume состоит в том, чтобы обеспечить постоянное хранение данных, которые нужны контейнеру, даже после его удаления или перезапуска. Это позволяет сохранить состояние приложения или обмениваться данными между контейнерами.

EX: `docker run -v myvolume:/path/to/mount myimage`
Создание volume с именем myvolume, а затем контейнер запускается с присоединением этого volume к пути /path/to/mount внутри контейнера.

```
Usage:  docker volume COMMAND
Commands:
  create      Create a volume
  inspect     Display detailed information on one or more volumes
  ls          List volumes
  prune       Remove unused local volumes
  rm          Remove one or more volumes
```

14. **docker inspect < object name > —** используется для получения подробной информации о Docker-объекте, таком как контейнер, образ, сеть или том.

Она предоставляет доступ к метаданным и настройкам объекта в формате JSON.

15. Dockerfile

Dockerfile — это конфигурационный файл, в котором описаны инструкции, которые будут применены при сборке Docker-образа и запуске контейнера. Dockerfile создается в корневой директории проекта и не имеет расширения. Инструкции пишутся капсом, а их значения отделяются пробелом.

Приведем пример несложного Dockerfile, и на его примере разберем логику взаимодействия инструкций между собой.

```
FROM python:latest
RUN apt-get update && apt-get install python3-pip -y && pip install --upgrade pip && pip install pipenv
RUN mkdir -p /usr/src/app/
WORKDIR /usr/src/app/
COPY . /usr/src/app/
EXPOSE 5000
RUN pip install --no-cache-dir -r requirements.txt
CMD ["python", "web_interface.py"]
```

Dockerfile имеет следующую логику заполнения:

1. Первой инструкцией всегда идёт FROM с указанием родительского образа. Например, FROM python:latest.
2. Инструкция RUN может принимать конвейер команд Linux, чтобы не создавать лишние слои. Например, RUN apt-get update && apt-get install python3-pip -y && pip install --upgrade pip && pip install pipenv.
3. Инструкция WORKDIR устанавливает рабочий каталог контейнера. Например, WORKDIR /usr/src/app/. Последующие команды RUN, CMD, ENTRYPOINT наследуют привязку WORKDIR.
4. COPY в Dockerfile используется для копирования файлов и директорий из локальной файловой системы в образ Docker, который вы создаете. COPY <source> <destination>
5. Инструкция EXPOSE, которая выражает намерение открыть заданный порт. Инструкция сама по себе не открывает порт без применения команды docker run с ключом -P. Если нужно «повесить» контейнер на определённый внешний порт с переадресацией во внутренний порт контейнера — применяется ключ -p с указанием внутреннего и внешнего порта через «:». Например, docker run -p 5000:8080
6. Завершающей инструкцией всегда идёт CMD. Например, CMD ["python", "web_interface.py"]. CMD наследует привязку к WORKDIR, поэтому web_interface.py будет запущен из папки /usr/src/app/.

Инструкция	Описание	Пример использования
FROM	Задает базовый образ. Все последующие инструкции создают слои поверх родительского образа.	FROM python:latest FROM debian:wheezy
RUN	Выполняет команду внутри контейнера и сохраняет результат.	RUN mkdir /usr/src/app/ RUN apt-get update && apt-get insta python3-pip -y
COPY	Копирует файлы и папки из текущей директории, где находится пользователь в указанную директорию в контейнере	COPY . /usr/src/app/
ADD	Копирует файлы и папки из текущей позиции пользователя, скачивает файлы по URL и работает с tar-архивами.	ADD https://1cloud.ru/archive/api_config.i /usr/src/app/
CMD	Выполняет команду с указанными аргументами во время запуска контейнера.	CMD ["python", "web_interface.py"]

Инструкция	Описание	Пример использования
ENTRYPOINT	Похожа на CMD, но при запуске контейнера не переопределяется в отличие от CMD.	ENTRYPOINT ["python", "web_interface.py"]
ENV	Задает переменные среды внутри образа, на которые могут ссылаться другие инструкции.	ENV ADMIN="ivan"
ARG	Задает переменные, значение которых передается докером во время сборки образа.	ARG maintainer=ivan
WORKDIR	Устанавливает рабочую директорию контейнера.	WORKDIR /usr/src/app/
VOLUME	Создает и подключает постоянный том хранения данных.	VOLUME /data_cont_1
EXPOSE	Указывает планируемый рабочий порт у контейнера. Инструкция сама по себе не открывает порт. Чтобы использовался указанный в EXPOSE порт — нужно	EXPOSE 5000

Инструкция	Описание	Пример использования
	указать docker run -P при запуске контейнера.	
LABEL	Добавляет метаданные в образ.	LABEL maintainer="katkov_ivan"

16. **Docker Compose** — инструмент, разработанный для помощи в определении и совместном использовании многоконтейнерных приложений. Используется файл .yml для определения сервисов**.** Пример такого прикола:

```yaml
version: '3'
services:
  web:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - 80:80
    volumes:
      - ./app:/app
    depends_on:
      - db
  db:
    image: mysql:5.7
    environment:
      - MYSQL_ROOT_PASSWORD=secret
      - MYSQL_DATABASE=myapp
      - MYSQL_USER=user
      - MYSQL_PASSWORD=password
    volumes:
      - dbdata:/var/lib/mysql

volumes:
  dbdata:
```

В этом примере у нас есть два сервиса: web и db.

Сервис web собирается с использованием Dockerfile, который находится в текущем контексте (текущая директория) и имеет имя Dockerfile. Он проксирует порт 80 хостовой машины на порт 80 контейнера. Каталог ./app с локальной файловой системы монтируется внутрь контейнера по пути /app. Также web зависит от сервиса db.

Сервис db использует образ MySQL версии 5.7. Он устанавливает несколько переменных среды, включая пароль для root пользователя и создает базу данных и пользователя. Данные MySQL сохраняются в томе dbdata.

Под "сервисом" в контексте Docker Compose подразумевается отдельный контейнер Docker, который выполняет определенную функцию в вашем приложении. Каждый сервис в docker-compose.yml представляет собой отдельный экземпляр контейнера Docker, который может содержать свои настройки, зависимости, порты и другие параметры.