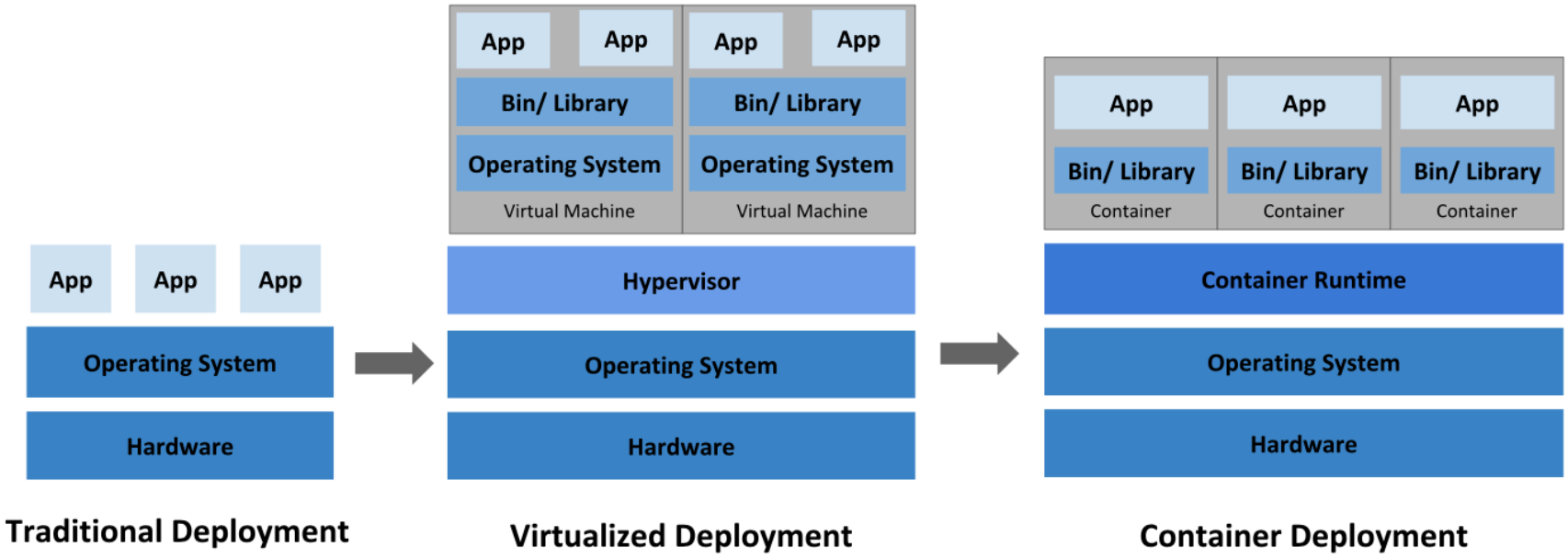
DOCKER

Филиппова Ольга, весна 2021

Мотивация

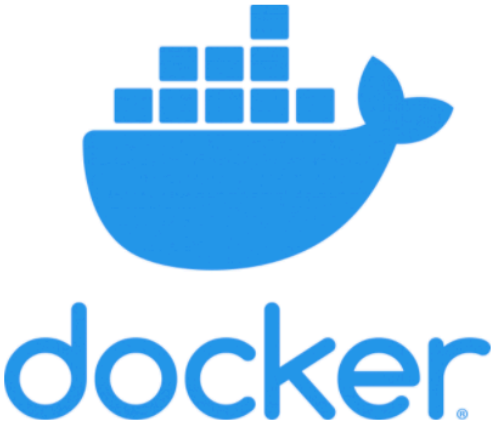
Немного истории



Преимущества Docker

- Компактность, простота и эффективность создания образа контейнера по сравнению с использованием образа виртуальной машины;
- Идентичная окружающая среда при разработке, тестировании и релизе: на ноутбуке работает так же, как и в облаке;
- Разделение задач между Dev и Ops: создавайте образы контейнеров приложений во время сборки/релиза, а не во время развертывания;
- Незаменим для микросервисной архитектуры: независимые приложения упаковываются в контейнеры, которые можно динамически развертывать и управлять;
- Безопасность.

Основные понятия



Docker is an open platform for developing, shipping, and running applications.

- Образ контейнера (container image или docker image) - это стандартная единица программного обеспечения, в которую упаковано приложение со всеми необходимыми для его работы зависимостями — кодом приложения, средой запуска, системными инструментами, библиотеками и настройками;
- Образ контейнера становится контейнером в тот момент, когда мы его запускаем (например, выполняем `docker run`).

Пример типичной папки

```
.
├── Dockerfile    <- как создать docker image
├── Pipfile       <- pipenv
├── Pipfile.lock  <- pipenv
└── app.py        <- само приложение
```

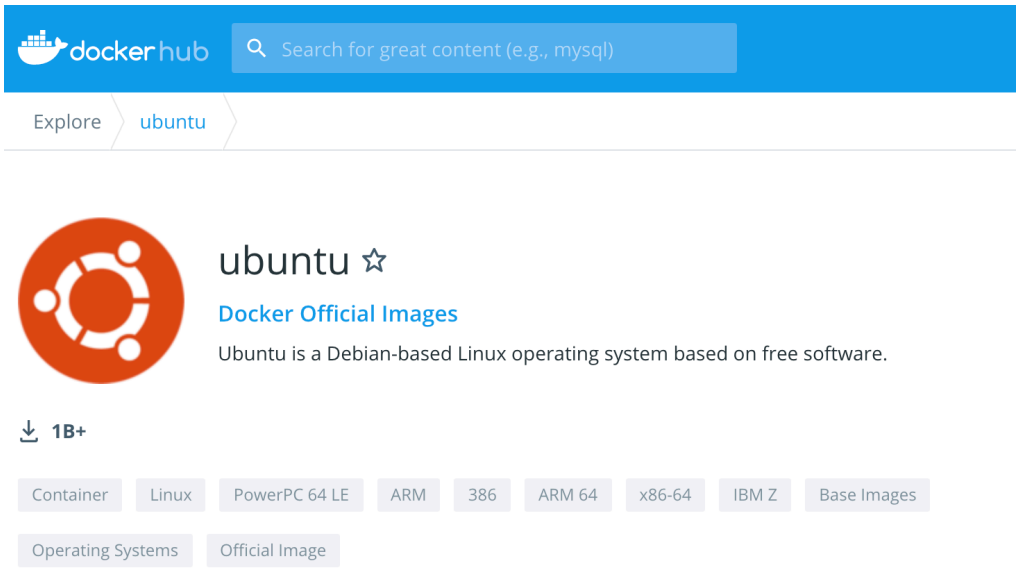
Dockerfile

скрипт с инструкциями для создания image.

```
1  FROM ubuntu:18.04
2  COPY . /app
3  RUN make /app
4  CMD python /app/app.py
```

Dockerhub

Реестр докер образов, который содержит как образы загруженные сторонними разработчиками, так и образы, выпущенные разработчиками docker



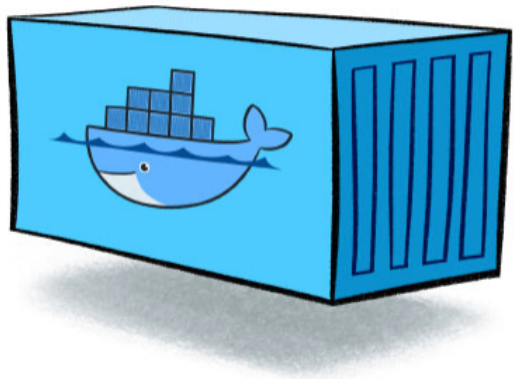
Основные команды image

Command	Description
docker image build	Build an image from a Dockerfile
docker image pull	Pull an image or a repository from a registry
docker image push	Push an image or a repository to a registry
docker image rm	Remove one or more images

Основные команды container

Command	Description
docker container stop	Stop one or more running containers
docker container rm	Remove one or more containers
docker container run	Run a command in a new container
docker container start	Start one or more stopped containers
docker container exec	Run a command in a running container

Делаем контейнер доступным окружающему миру



Ports

Порты задаем аргументом в команде docker run

```
docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Name, shorthand	Default	Description
`--publish , -p`		Publish a container's port(s) to the host

```
1  docker run -p 3000:3000 image-name
```

Container Volumes

Это инструмент, который предоставляет возможность соединять какую-то папку внутри контейнера с папкой на host machine.

Есть два типа Container Volumes

	Named Volumes	Bind Mounts
Host Location	Docker chooses	You control
Mount Example (using `-v`)	my-volume:/usr/local/data	/path/to/data:/usr/local/data
Populates new volume with container contents	Yes	No
Supports Volume Drivers	Yes	No

Container Volumes

создаем новый Named Volume

```
1  docker volume create new-volume
```

Передаем его в качестве аргумента команде docker run

```
1  docker run -v new-volume:/etc/data image-name
```

Env

```
docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]
```

Name, shorthand	Default	Description
`--env , -e`		Set environment variables

```
1  docker run MySQL_example
2      -e MYSQL_ROOT_PASSWORD=secret `
3      -e MYSQL_DATABASE=example
```

Dockerfile best practices

- Create ephemeral containers
- Understand build context
- Don’t install unnecessary packages
- Decouple applications
- Minimize the number of layers

https://docs.docker.com/develop/develop-images/dockerfile_best-practices

Взаимодействие нескольких контейнеров

Docker Compose

is a tool for defining and running multi-container Docker applications

Using Compose is basically a three-step process:

1. Define your app's environment with a Dockerfile so it can be reproduced anywhere.
2. Define the services that make up your app in docker-compose.yml so they can be run together in an isolated environment.
3. Run docker-compose up and Compose starts and runs your entire app.

Docker Compose

docker-compose.yml example

```yaml
version: "3.9"
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
    environment:
      FLASK_ENV: development
  redis:
    image: "redis:alpine"
```

Networking

Remember that containers, by default, run in isolation and don't know anything about other processes or containers on the same machine. So, how do we allow one container to talk to another? The answer is **networking**. Now, you don't have to be a network engineer (hooray!). Simply remember this rule...

> If two containers are on the same network, they can talk to each other. If they aren't, they can't.

Networking

in Docker Compose

By default Compose sets up a single network for your app. Each container for a service joins the default network and is both reachable by other containers on that network, and discoverable by them at a hostname identical to the container name.

Deploy проекта

Deploy проекта с ипользованием GitHub

1. push проекта на гит
2. на сервере заберем код с гита
3. на сервере с помощью Docker Compose собираем наши контейнеры и поднимаем

Deploy проекта с ипользованием dockerhub

1. создаем image для каждого приложения и делаем push на dockerhub
2. на серве с помощью Docker Compose собираем наши контейнеры и поднимаем (в docker-compose.yml прописываем какие images необходимо скачать с dockerhub)

Полезные ссылки

туториалы:

https://docs.docker.com/get-started/

https://www.youtube.com/watch?v=QF4ZF857m44

https://www.youtube.com/playlist?list=PLQJ7ptkRY-xbR0ka2TUxJkXna40XWu92m

Семинар

устанавливаем докер

установите Docker Desktop для своей системы по инструкции вот отсюда
https://docs.docker.com/desktop/
чтобы проверить что все установилось набираем docker ps

поднимаем образ jupyter notebook

гуглим docker hub jupyter notebook
копируем команду docker pull jupyter/datascience-notebook

команда смотреть образы:
docker images

поднимаем образ командой
docker run -p 8888:8888 <image_id>
расстраиваемся что нельзя сохранить результат своего труда и решаем добавить volume

заходим в контейнер:
docker exec - it <container_id> /bin/bash
для этого, перед тем как погасить контейнер, смотрим какой каталог замапить с папкой на хосте <path>

гасим контейнер командой
docker container stop <container_id>
добавляем volume командой
docker run -p 8888:8888 -v “$(pwd)”:<path> <image_id>

радуемся что файл который мы создали появился локально

пытаемся импортировать lightgbm понимаем что его там нет и решаем создать свой image с lightgbm

для этого сначала создаем dockerfile

FROM jupyter/datascience-notebook
RUN pip install lightgbm

теперь надо создать из него новый образ, сразу дадим ему имя, чтобы

не путаться в айдишниках
docker build . -t jupyter_lgbm

теперь все как раньше, поднимаем контейнер из образа
docker run -p 8888:8888 -v “$(pwd)”:<path> jupyter_lgbm

заходим, проверяем что теперь lightgbm стоит