

Web программирование

Курс для бакалавриата

Александр Менщиков,
к.т.н., доцент ИТМО

Очереди, вебсокеты и Docker

Александр Менщиков,
к.т.н., доцент ИТМО

Содержание лекции

1. Очереди задач
2. Вебсокеты в FastAPI
3. Docker, Dockerfile и Docker Compose

Очереди задач

Зачем нужны фоновые задачи?

Фоновые задачи – задачи, которые будут выполняться после получения ответа.

```
@app.get("/")
async def root():
    return HTMLResponse("<form action='/send-notification' ...")

@app.post("/send-notification/")
async def send_notification(email: str = Form(), message: str = Form()):
    await sleep(5)
    return {"message": "Notification sent"}
```

Фоновые задачи

```
from fastapi import BackgroundTasks, FastAPI
from asyncio import sleep
from datetime import datetime

app = FastAPI()
```

```
async def write_notification(email: str, message=""):
    await sleep(5)
    with open("log.txt", mode="a+") as email_file:
        content = f"notification for {email}: {message} saved. Timestamp: {datetime.now()}\n"
        email_file.write(content)
```

```
@app.get("/send-notification/{email}")
async def send_notification(email: str, background_tasks: BackgroundTasks):
    background_tasks.add_task(write_notification, email, message="some notification")
    return {"message": "Notification sent in the background"}
```

```
notification saved. Timestamp: 2024-09-30 16:55:05.888212
notification saved. Timestamp: 2024-09-30 16:55:06.294398
notification saved. Timestamp: 2024-09-30 16:55:06.650997
notification saved. Timestamp: 2024-09-30 16:55:21.922824
notification saved. Timestamp: 2024-09-30 16:55:22.069551
notification saved. Timestamp: 2024-09-30 16:55:22.208785
notification saved. Timestamp: 2024-09-30 16:55:22.364338
notification saved. Timestamp: 2024-09-30 16:55:22.505399
notification saved. Timestamp: 2024-09-30 16:55:22.647180
```

Celery

Celery — это асинхронный менеджер задач, позволяющий выполнять длительные операции в фоновом режиме без блокировки основного потока веб-приложения.

- Асинхронная обработка: Запуск задач в фоновом режиме
- Управление очередями: Распределение задач по очередям для обработки их на разных воркерах
- Повтор выполнения задач: Повторный запуск задач при возникновении ошибок
- Отложенное выполнение: Планирование выполнения задач в будущем

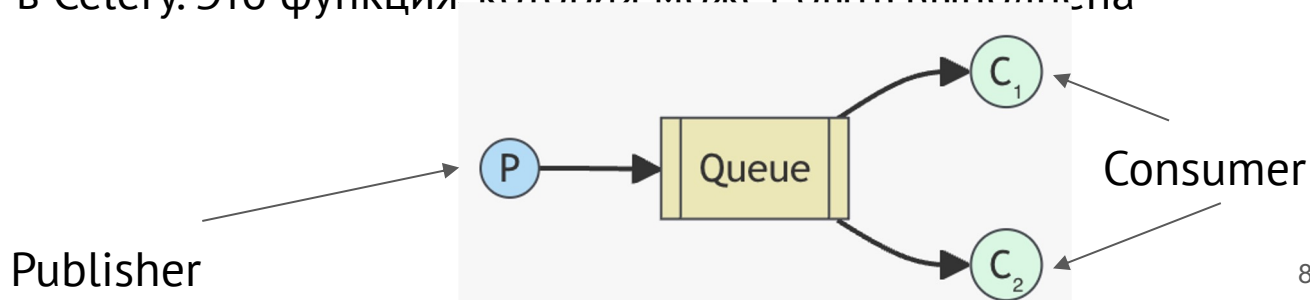
Термины

Брокер – система очередей, которая хранит задачи. Примеры: Redis, RabbitMQ

Приложение – веб-сервер, который обрабатывает HTTP-запросы и может отправлять задачи в очередь

Воркер – отдельный процесс Celery, который выполняет задачи. Воркеры получают задачи от брокера и обрабатывают их

Таск – единица работы в Celery. Это функция которая может быть выполнена асинхронно воркером



Создаем приложение

```
from fastapi import FastAPI
from datetime import datetime
import time
```

```
app = FastAPI()
```

```
@app.get("/send-notification/{email}")
async def send_notification(email: str):
    task = write_notification.delay(email, message="some notification")
    return {"task_id": task.id, "message": "Notification sent in the background"}
```

Подключение Celery и создание задачи

```
# pip install
```

```
celery
```

```
redis
```

```
flower
```

```
from celery import Celery
```

```
from celery.result import AsyncResult
```

```
# Initialize Celery
```

```
celery = Celery(__name__)
```

```
celery.conf.broker_url = "redis://localhost:6379/0"
```

```
celery.conf.result_backend = "redis://localhost:6379/0"
```

```
@celery.task(name="write_notification")
```

```
def write_notification(email: str, message: str = ""):
```

```
    time.sleep(5)
```

```
    with open("log.txt", mode="a+") as email_file:
```

```
        content = f"notification for {email}: {message} saved. Timestamp: {datetime.now()}\n"
```

```
        email_file.write(content)
```

```
    return f"Notification sent to {email}"
```

Запускаем

- 1. Redis
- 2. Приложение: `uvicorn main:app --reload`
- 3. Воркера: `celery -A main.celery worker --loglevel=info`
- 4. Мониторинг: `celery -A main.celery flower`

Show

15

 tasks

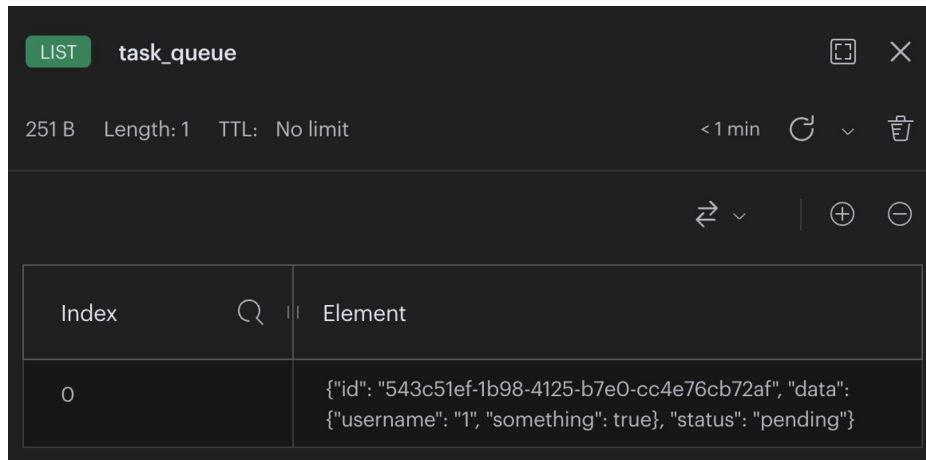
Search:

Name	UUID	State	args	kwargs	Result	Received
write_notification	ae705201-d7ed-45cc-a0d1-bd57ebe7d2d1	STARTED	('a@a.ru',)	{'message': 'some notification'}		2024-09-30 14:38:49.054
write_notification	b7dc3597-3b6a-4dbd-ac56-f20c2a686b6d	SUCCESS	('a@a.ru',)	{'message': 'some notification'}	'Notification sent to a@a.ru'	2024-09-30 14:38:48.826
write_notification	e8da087b-f4c5-4fb0-bd10-16084b85ef5b	SUCCESS	('a@a.ru',)	{'message': 'some notification'}	'Notification sent to a@a.ru'	2024-09-30 14:38:48.581

Напишем собственную реализацию функции добавления

```
# Функция для добавления задачи в очередь
def enqueue_task(task_data: Dict[str, Any]):
    task_id = str(uuid.uuid4())
    task = {
        "id": task_id,
        "data": task_data,
        "status": "pending"
    }

    redis_client.rpush(Queue_NAME, json.dumps(task))
    redis_client.set(f"task:{task_id}", json.dumps(task))
    return task_id
```



LIST task_queue

251 B Length: 1 TTL: No limit < 1 min

Index	Element
0	{"id": "543c51ef-1b98-4125-b7e0-cc4e76cb72af", "data": {"username": "1", "something": true}, "status": "pending"}

Напишем собственную реализацию воркера

Функция воркера

```
def worker():
```

```
    print("Worker started")
```

```
    while True:
```

```
        # Получаем задачу из очереди
```

```
        _, task_json = redis_client.blpop(Queue_NAME)
```

```
        task = json.loads(task_json)
```

```
        # Обновляем статус задачи
```

```
        task["status"] = "processing"
```

```
        task["started_at"] = datetime.now().isoformat()
```

```
        redis_client.set(f"task:{task['id']}", json.dumps(task))
```

```
        # Выполняем задачу
```

```
        process_task(task["data"])
```

Функция для выполнения задачи

```
def process_task(task_data):
```

```
    # Здесь выполняется задача
```

```
    time.sleep(5)
```

```
    print(f"Task processed: {task_data}")
```

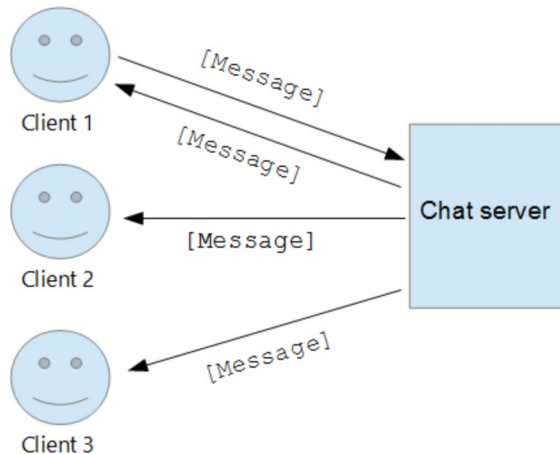
Вебсокеты и FastAPI

Вебсокеты

https://developer.mozilla.org/ru/docs/Web/API/WebSockets_API

<https://fastapi.tiangolo.com/advanced/websockets/>

- HTML страница с JS – /
- Websocket endpoint – /ws



JavaScript

```
<script>
var ws = new WebSocket("ws://localhost:8000/ws");
var form = document.getElementById('form');
var input = document.getElementById('messageText');
var messages = document.getElementById('messages');

ws.onmessage = function (event) {
    var messageItem = document.createElement('div');
    messageItem.textContent = event.data;
    messages.appendChild(messageItem);
};

form.onsubmit = function (event) {
    event.preventDefault();
    ws.send(input.value);
    input.value = '';
};
</script>
```

- **ws.onmessage** –
вызывается при
получении сообщения
- **form.onsubmit** –
вызывается при
нажатии кнопки submit
(отправки формы)
 - **ws.send** – отправка
сообщения

Чат

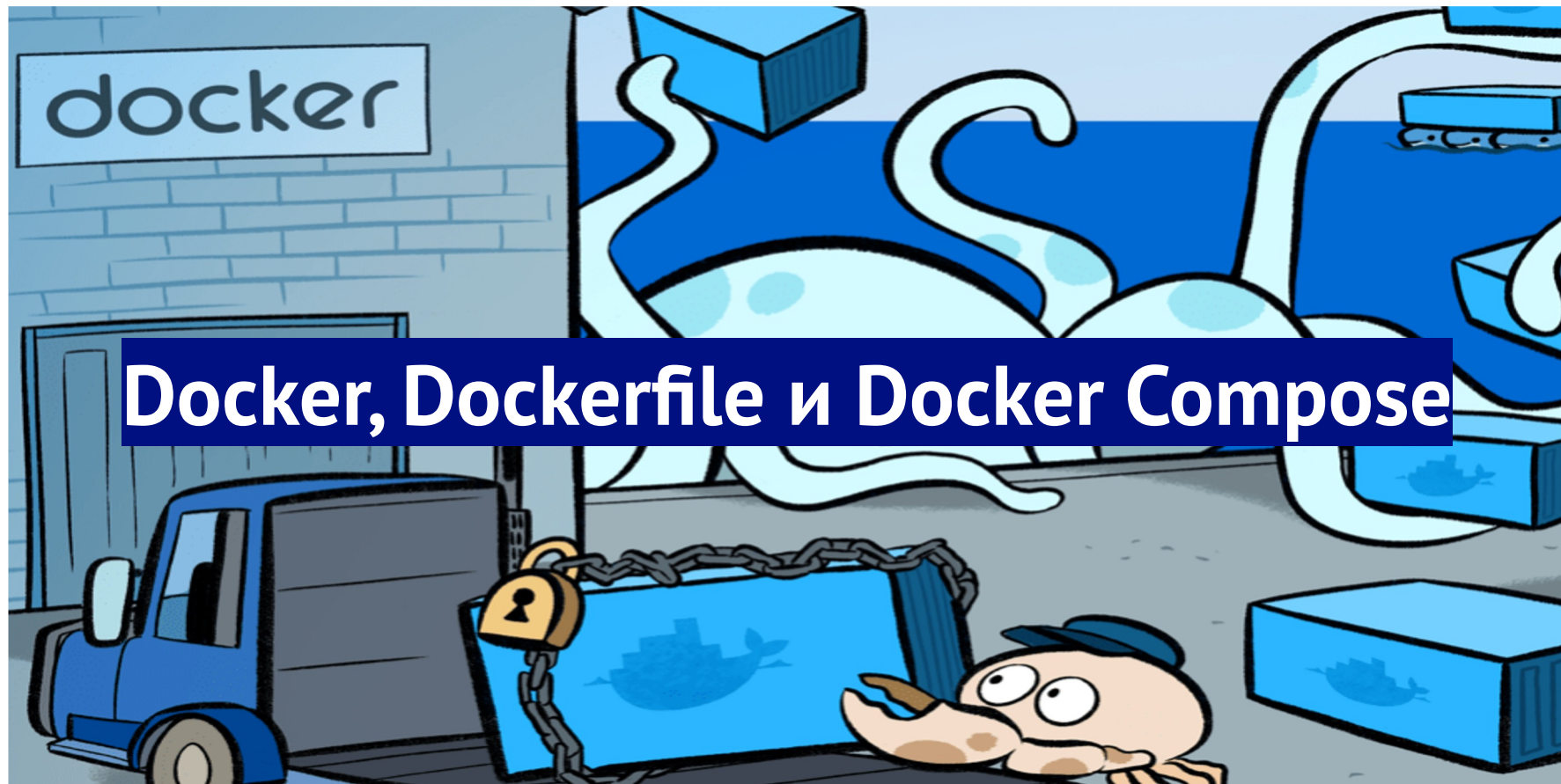
```
from fastapi import FastAPI, WebSocket, WebSocketDisconnect  
active_connections = []
```

```
@app.websocket("/ws")  
async def websocket_endpoint(websocket: WebSocket):  
    await websocket.accept()  
    active_connections.append(websocket)  
    try:  
        while True:  
            data = await websocket.receive_text()  
            for connection in active_connections:  
                await connection.send_text(data)  
    except WebSocketDisconnect:  
        active_connections.remove(websocket)
```

- Храним все подключения в массиве active_connections
- При ошибке WebSocketDisconnect исключаем клиента
- При успешном подключении висим в бесконечном цикле отправки сообщений

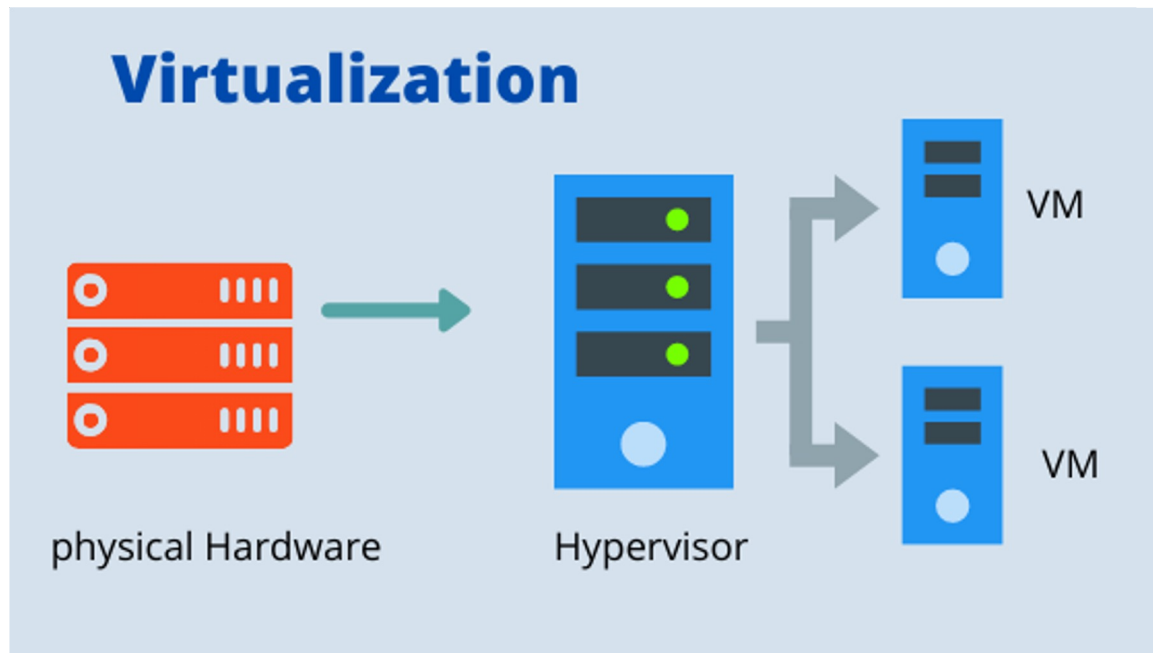
Ключевые функции WebSocket

1. Установка соединения
 - a. JS onopen
 - b. Python accept
2. Отправка и получение сообщений
 - a. JS send
 - b. Python receive
3. Поддержка различных типов сообщений
 - a. text
 - b. binary
4. Обработка дисконнектов
 - a. JS onclose
 - b. Python WebSocketDisconnect
5. Обработка ошибок
 - a. JS onerror
6. Закрытие соединения
 - a. JS close



Docker, Dockerfile и Docker Compose

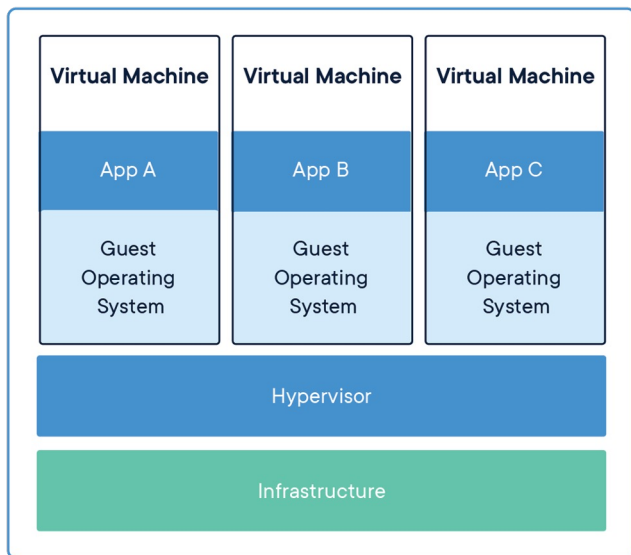
Виртуализация



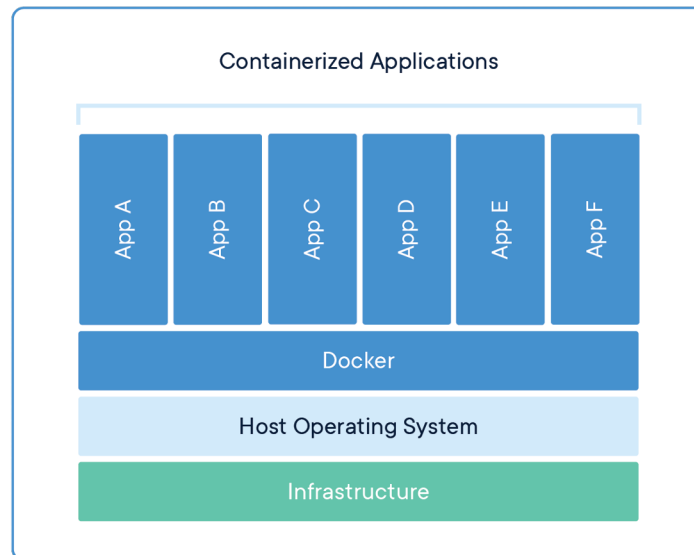
Плюсы виртуализации

- Resource utilisation
- Security
- Isolation
- Scalability
- Portability
- Easy backups
- Abstraction from hardware

Виртуализация и контейнеризация



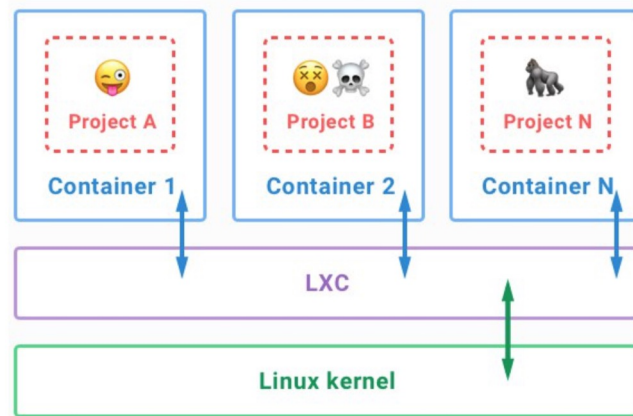
Virtual machines



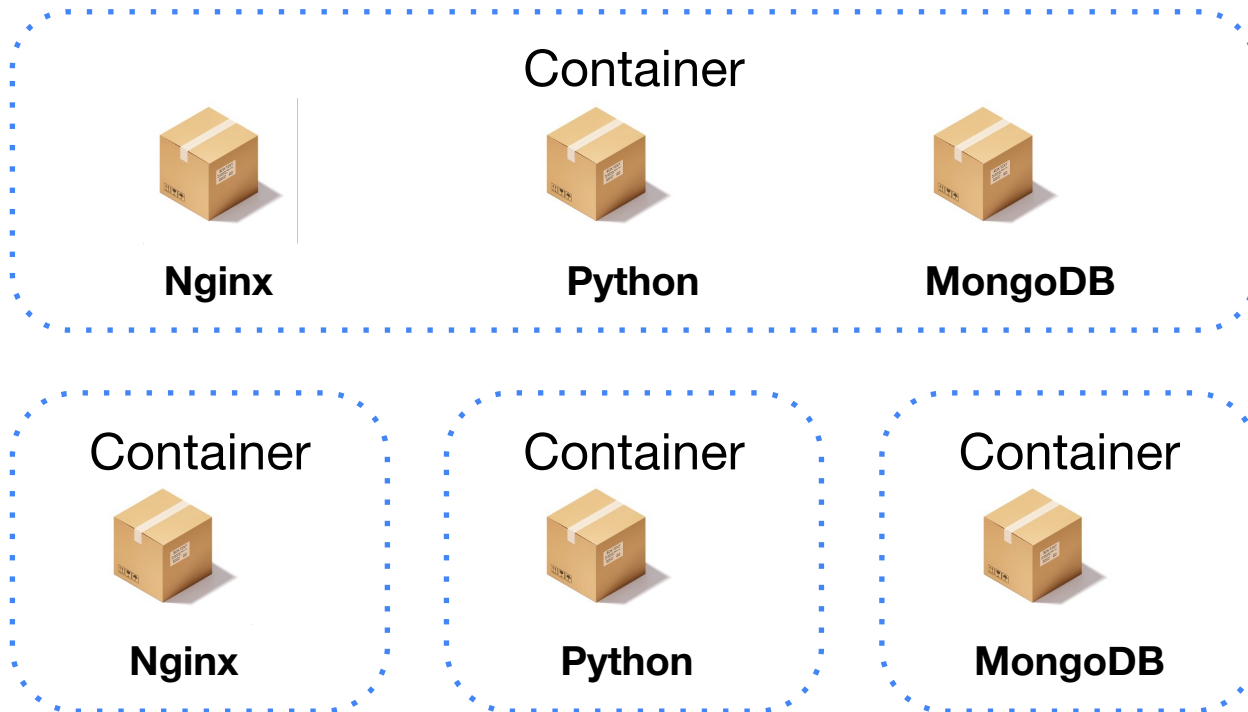
Containers

Containers != virtualization

- 👍 Выше производительность
- 👍 Выше portability и scalability
- 👍 Ниже потребление ресурсов
- ⚠️ Ограниченная поддержка функций ОС
- ⚠️ Shared Kernel



Контейнеры уровня машины и уровня приложения

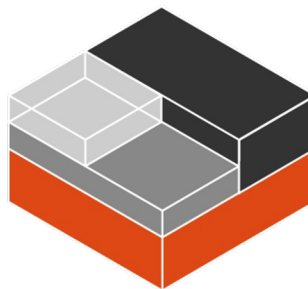


LXD и Docker

Docker hosts application containers



LXD hosts machine containers



Установка

<https://docs.docker.com/get-docker/>

<https://docs.docker.com/engine/install/ubuntu/>

1. Set up Docker's `apt` repository.

```
# Add Docker's official GPG key:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc

# Add the repository to Apt sources:
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

2. Install the Docker packages.

Latest Specific version

To install the latest version, run:

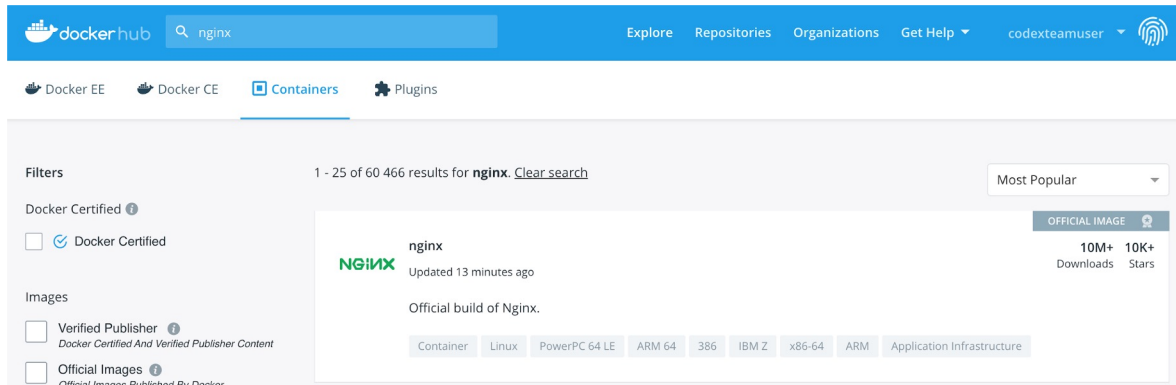
```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin dock
```

3. Verify that the Docker Engine installation is successful by running the `hello-world` image.

```
$ sudo docker run hello-world
```

Элементы

- Image
- Container
- Network
- Volume



docker/whalesay ☆

By [docker](#) • Updated 5 years ago

An image for use in the Docker demo tutorial

Container

<https://hub.docker.com/r/docker/whalesay/>

Hello world

```
> docker pull docker/whalesay
```

```
> docker run docker/whalesay cowsay WAD
```

Images

```
> docker image ls
```

```
> docker image inspect docker/whalesay
```

```
root@scw-recurring-tesla:~# docker image ls
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
root_flask	latest	10b38c58159b	About an hour ago	210MB
root_flask-simple	latest	8695c3e97616	About an hour ago	210MB
codexteamuser/hawk-collector	prod	b921deac96e6	3 days ago	20.3MB
consul	latest	197999eb696c	11 days ago	116MB
ubuntu	latest	1d622ef86b13	11 days ago	73.9MB
nginx	latest	602e111c06b6	12 days ago	127MB
python	3.8.1-slim-buster	b99890b7a7dc	2 months ago	193MB
docker/whalesay	latest	6b362a9f73eb	4 years ago	247MB

Containers

```
> docker run --rm -it ubuntu
```

```
> docker run -d ubuntu bash -c "while true; do sleep 3; done"
```

```
> docker ps
```

```
> docker ps --format "{{.ID}}\t{{.Image}}\t{{.Names}}"
```

```
d4021aab2350    ubuntu    cool_saha
```

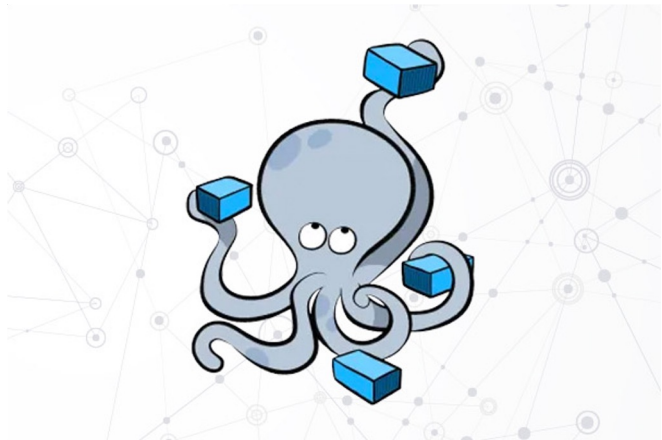
```
> docker exec -it d4021aab2350 bash
```

```
root@d4021aab2350:/# ps uaxf
```

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
root	66	13.3	0.1	4112	3400	pts/0	Ss	12:53	0:00	bash
root	74	0.0	0.1	5884	2908	pts/0	R+	12:53	0:00	_ ps uaxf
root	1	0.2	0.1	3980	3156	?	Ss	12:50	0:00	bash -c while true; do sleep 3; done
root	73	0.0	0.0	2512	588	?	S	12:53	0:00	sleep 3

Docker-compose

```
services:
  nginx:
    image: nginx
    ports:
      - "127.0.0.1:8000:80"
    volumes:
      - ./index.html:/usr/share/nginx/html/index.html
```



```
> docker compose -f 1-nginx.yml up
```

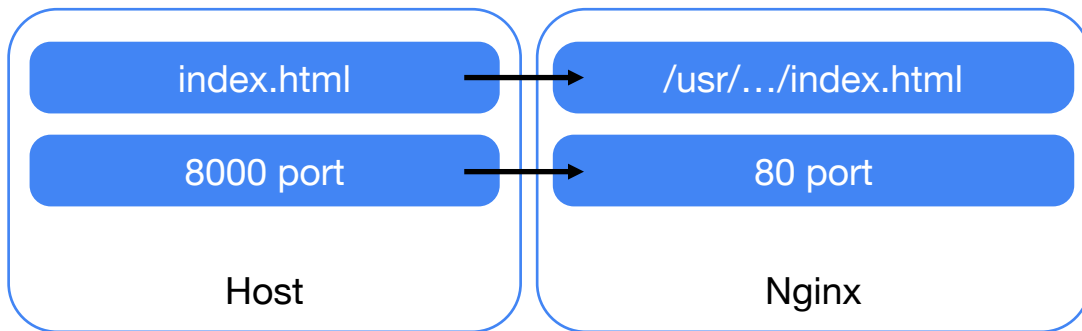
Compose CLI

- > `docker compose ps`
- > `docker compose build`
- > `docker compose up`
- > `docker compose down`
- > `docker compose up -d`
- > `docker compose -f file.yml up`

Docker compose port mapping

```
root@scw-recurring-tesla:~# docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
15432721d0f6	nginx	"nginx -g 'daemon of...'"	About a minute ago	Up 1 second	0.0.0.0:80->80/tcp	root_nginx_1

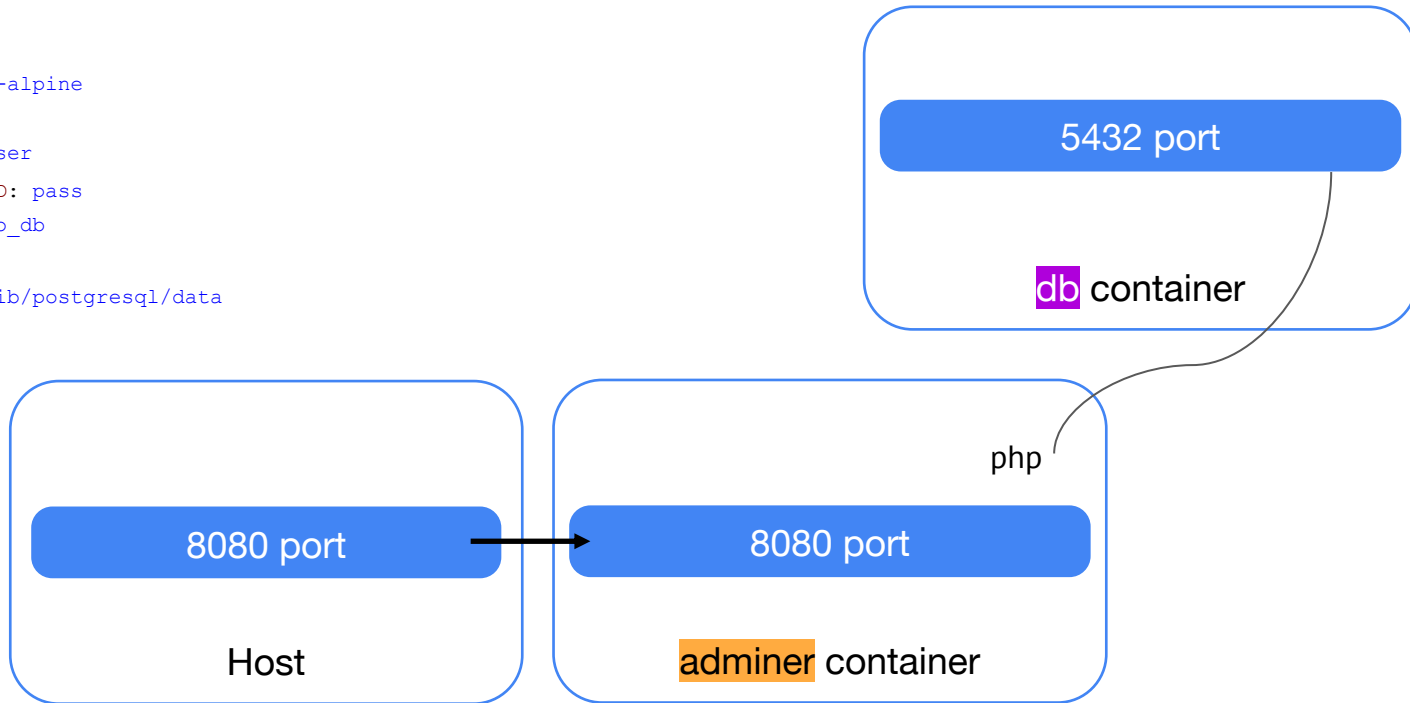


Docker compose Network

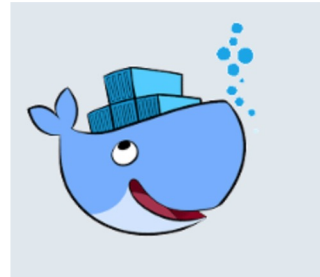
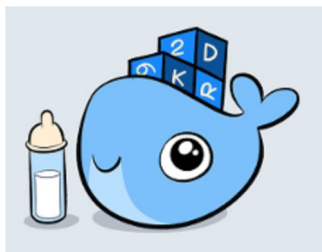
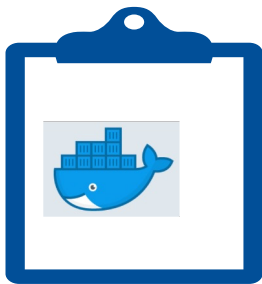
```
services:
  db:
    image: postgres:13-alpine
    environment:
      POSTGRES_USER: user
      POSTGRES_PASSWORD: pass
      POSTGRES_DB: demo_db
    volumes:
      - db-data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  adminer:
    image: adminer
    ports:
      - "8080:8080"
    depends_on:
      - db

volumes:
  db-data:
```



Dockerfile



```
FROM python:3.9-slim
WORKDIR /app
COPY . .
RUN pip install fastapi uvicorn
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

Dockerfile

```
services:
  web:
    build: .
    ports:
      - "8000:8000"
docker-compose.yml
```

```
from fastapi import FastAPI
app = FastAPI()
@app.get("/")
def read_root():
    return {"message": "Hello, FastAPI!"}
main.py
```

MongoDB and FastAPI

```
services:
  mongodb:
    image: mongo:4.2
    volumes:
      - ./data:/data/db

  web:
    build: .
    depends_on:
      - mongodb
    ports:
      - "8000:8000"
```

```
FROM python:3.9-slim
```

```
WORKDIR /app
```

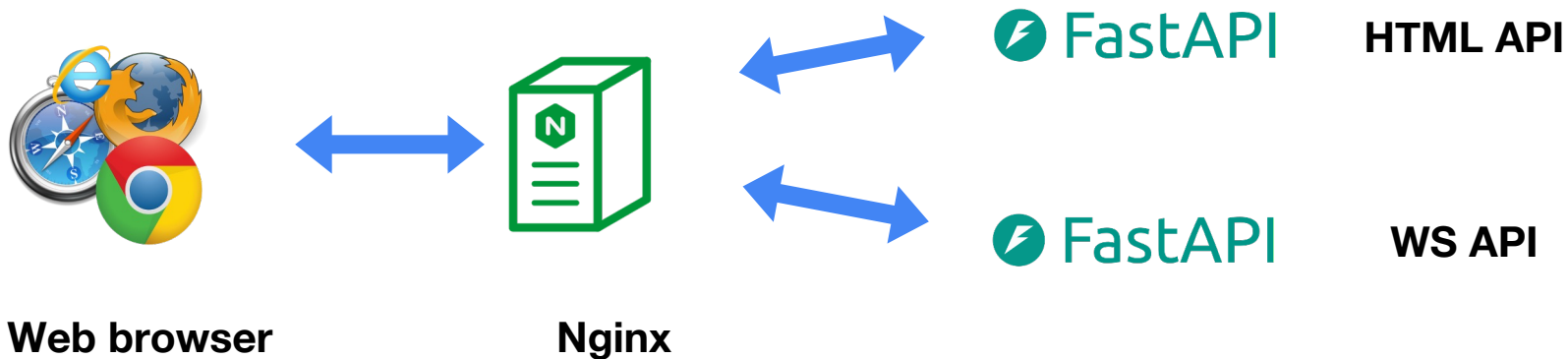
```
COPY . .
```

```
RUN pip install fastapi uvicorn
```

```
mongoengine
```

```
CMD ["uvicorn", "main:app", "--host",
"0.0.0.0", "--port", "8000"]
```

Docker Nginx FastAPI Websocket



```
nginx:
  image: nginx:alpine
  volumes:
    - ./nginx/nginx.conf:/etc/nginx/nginx.conf:ro
  ports:
    - "8080:80"
  depends_on:
    - html_app
    - ws_app
```

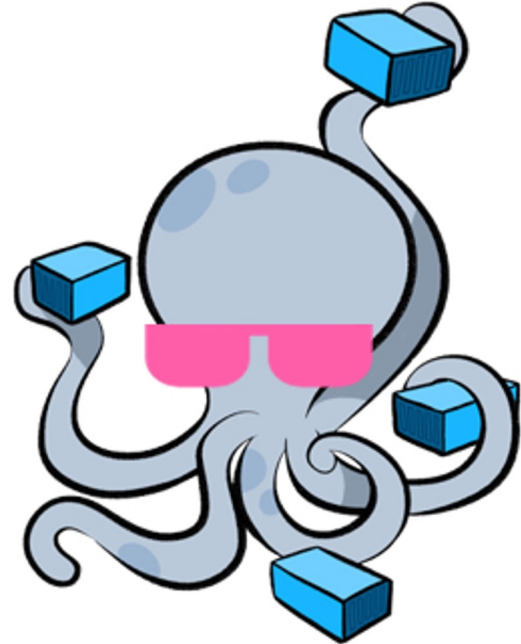
```
html_app:
  build: ./html_app
  container_name: fastapi_html
  command: uvicorn main:app --host 0.0.0.0 --port 8000

ws_app:
  build: ./ws_app
  container_name: fastapi_ws
  command: uvicorn main:app --host 0.0.0.0 --port 8000
```

Advanced техники

<https://github.com/docker/awesome-compose>

- scale
- .dockerignore
- multi staged build



Coding samples

https://github.com/cs-itmo/webdev_2024



Полезные ссылки

- <https://fastapi.tiangolo.com/tutorial/background-tasks/>
- <https://docs.celeryq.dev/>
- <https://redis.io/docs/latest/commands/rpush/>
- <https://redis.io/docs/latest/commands/blpop/>
- https://developer.mozilla.org/ru/docs/Web/API/WebSockets_API
- <https://fastapi.tiangolo.com/advanced/websockets/>
- <https://docs.docker.com/get-docker/>
- <https://docs.docker.com/engine/install/ubuntu/>
- <https://github.com/docker/awesome-compose>

Вопросы?