

Общая схема запуска

Файл `docker-compose.yml` описывает многоконтейнерное окружение приложения для отслеживания сна: веб-сервис Django, базу данных, брокер задач, систему мониторинга, логирование и сопутствующие сервисы. Команда `docker-compose up` из каталога проекта создаёт и запускает все описанные в файле сервисы, автоматически поднимая их зависимости и агрегируя их логи в одном выводе терминала. При первом запуске при необходимости выполняется сборка образов и загрузка недостающих образов из реестра, а при последующих запусках переиспользуются уже созданные контейнеры и тома данных.

Архитектура контейнеров

- В `docker-compose.yml` логично выделены несколько групп сервисов:
 - Основное веб-приложение Django (Gunicorn + Nginx), использующее общий том для статических файлов и конфигурации.
 - Инфраструктурные компоненты: PostgreSQL как основная БД, Redis как брокер для Celery и кеш, Celery worker и Celery beat для фоновых задач.
 - Подсистема мониторинга и логирования: Prometheus для сбора метрик, Grafana для их визуализации, Loki и Promtail для централизованного сбора логов.
 - Хранилища и вспомогательные сервисы: ClickHouse (аналитические данные), MinIO (объектное хранилище), Qdrant (векторная БД), Ollama (LLM-модели) и т.п., для которых, описаны отдельные тома для долговременного хранения данных.

Между контейнерами определены внутренние Docker-сети: как минимум одна «backend»-сеть для взаимодействия между приложением и БД/брокером, а также, при необходимости, публичная сеть для Nginx, публикующего HTTP-порт наружу. Для долговременного хранения данных в `docker-compose.yml` привязаны именованные тома к PostgreSQL, ClickHouse, MinIO, Loki, Prometheus и Qdrant, что позволяет пересоздавать контейнеры без потери пользовательских и аналитических данных.

Container CPU usage

48.67% / 2400% (24 CPUs available)

Container memory usage

3.80GB / 15.2GB

Show charts

Q Search

Only show running containers

<input type="checkbox"/>	Name	Container ID	Image	Port(s)	CPU (%)	Memory usage...	Memory (%)	Disk read...	Net...	PIDS	Last s	Actions
<input type="checkbox"/>	clickhouse	1aacc9f484bd	clickhouse:172.17.0.2	Show all ports (2)	15.98%	435.7MB / 15.57%	2.73%	561MB / 11.1MB	5.4KB	727	2 min	
<input type="checkbox"/>	langfuse-server-1	28c389ad33b	langfuse:3001-3000		0%	592.9MB / 15.57%	3.72%	269MB / 8.19KB	115KB	35	2 min	
<input type="checkbox"/>	grafana	8bcb5b5ced81	grafana:3000-3000		0.68%	101.9MB / 15.57%	0.64%	254MB / 328KB	52.5KB	29	2 min	
<input type="checkbox"/>	minio	72979768a5f	minio:9000	Show all ports (2)	10.78%	99.36MB / 15.57%	0.62%	145MB / 86KB	8KB	26	2 min	
<input type="checkbox"/>	prometheus	a7c6c4381cb4	prom/prom:9090-9090		0%	35.69MB / 15.57%	0.22%	107MB / 12.5MB	1.04M	25	2 min	
<input type="checkbox"/>	loki	a638a9613f79	grafana/loki:3100-3100		2.01%	51.55MB / 15.57%	0.32%	95.7MB / 209KB	45.6KB	26	2 min	
<input type="checkbox"/>	promtail	f0ffc67ee382	grafana/promtail:9090-9090		2.07%	25.29MB / 15.57%	0.16%	79.6MB / 36.9KB	6.84KB	17	2 min	
<input type="checkbox"/>	celery-1	6dd335ab0cbe	alexcocom:alexcocom		0.37%	1.91GB / 15.57%	12.3%	54.1MB / 4.1KB	63.3KB	26	2 min	
<input type="checkbox"/>	qdrant_sleep	6a02419315e5	qdrant/qdrant:6333-6333		0.55%	122.5MB / 15.57%	0.77%	51.5MB / 0B	10.7KB	104	2 min	
<input type="checkbox"/>	db-1	4e7f34c1405d	postgres:15432-5432		9.01%	49.86MB / 15.57%	0.31%	47.5MB / 598KB	24.8KB	7	2 min	
<input type="checkbox"/>	web-1	34aac9919dc1	alexcocom:8000-8000		0.02%	181.5MB / 15.57%	1.14%	46.8MB / 0B	47KB	26	2 min	
<input type="checkbox"/>	celery-beat-1	6facd43093b0	alexcocom:alexcocom		0%	170.4MB / 15.57%	1.07%	38.3MB / 0B	4.73KB	25	2 min	
<input type="checkbox"/>	ollama	5dd33ab1140	ollama:ollama		0%	23.85MB / 15.57%	0.15%	28.1MB / 0B	3.97KB	12	2 min	
<input type="checkbox"/>	nginx-1	1e4c1310477	nginx:1.25-80-80		0%	25.54MB / 15.57%	0.16%	15.1MB / 4.1KB	2.32KB	25	2 min	
<input type="checkbox"/>	node-exporter	b9b3edcf3e23	prom/node:9100-9100		0.02%	9.88MB / 15.57%	0.06%	15.1MB / 0B	26.6KB	5	2 min	
<input type="checkbox"/>	redis-1	590c2b5ac0be	redis:7.2-6379-6379		0.31%	6.79MB / 15.57%	0.04%	14.4MB / 0B	73.3KB	6	2 min	

Permissions to install update

Рисунок 2 – развернутый докер

Управление окружением

Управление конфигурацией основано на файле `.env` в корне проекта, который создаётся пользователем на основе шаблона `.env.example`. Docker Compose по умолчанию подхватывает `.env` из текущего каталога и использует его переменные для подстановки значений в `docker-compose.yml` (механизм `variable interpolation`), что позволяет не хранить секреты и специфичные настройки окружения прямо в YAML-файле. Дополнительно отдельные сервисы могут использовать директиву `env_file`: и/или секцию `environment`: для передачи своих переменных окружения, а при необходимости можно запускать окружение с альтернативным набором переменных через флаг `--env-file`, указав другой путь к файлу окружения.

Типичные группы переменных окружения включают: параметры доступа к БД (хост, порт, имя БД, логин, пароль), секретный ключ Django и настройки DEBUG, параметры подключения к Redis и брокеру Celery, credenшелы для MinIO и внешних сервисов, а также базовые URL и учётные записи доступа к Prometheus/Grafana/Loki. Такое разделение позволяет легко переключаться между локальной разработкой, тестовой и продуктивной средой, изменяя только содержимое `.env`, не модифицируя сам `docker-compose.yml`.

Развёртывание с нуля и время

Развёртывание окружения «с нуля» включает установку Docker и Docker Compose, подготовку `.env` и первый запуск. После клонирования репозитория и создания файла `.env` по шаблону демонстрационный запуск выполняется

через `docker-compose up -d --build`, что в одном шаге собирает образы (если указаны секции `build:`), загружает недостающие образы из реестра и создаёт/запускает все сервисы в фоне.

Общее время первого развёртывания определяется скоростью сети и производительностью диска, так как на этом этапе скачиваются образы баз данных, брокеров, систем мониторинга и самого веб-приложения, а также выполняется их начальная инициализация. После завершения первого цикла `docker-compose up` повторные запуски выполняются заметно быстрее, поскольку образы уже присутствуют локально, а данные хранятся в именованных томах, и Compose лишь пересоздаёт/запускает контейнеры на основе существующего состояния.

```
PS C:\Users\dmitr\Education\SII\Web-sleep-app\sleepproject> docker-compose up -d
[+] Running 17/17
✔Network sleepproject_default          Created           0.1s
✔Container node-exporter               Started           1.6s
✔Container sleepproject-db-1           Healthy          12.6s
✔Container minio                       Healthy          7.6s
✔Container prometheus                 Started          1.7s
✔Container loki                       Started           1.8s
✔Container sleepproject-redis-1        Started           2.3s
✔Container ollama                     Started           2.3s
✔Container promtail                   Started           1.8s
✔Container clickhouse                 Started           2.0s
✔Container qdrant_sleep                Started           1.7s
✔Container grafana                    Started           2.1s
✔Container sleepproject-celery-beat-1   Started           2.8s
✔Container sleepproject-celery-1        Started           2.6s
✔Container sleepproject-langfuse-server-1 Started          12.7s
✔Container sleepproject-web-1          Started           2.8s
✔Container sleepproject-nginx-1        Started           3.2s
PS C:\Users\dmitr\Education\SII\Web-sleep-app\sleepproject>
```

Рисунок 2 – запуск контейнеров