perplexity

Задание: Обратный прокси NGINX для 4 Hello World приложений с Docker Compose и акцентом на DevOps best practices

Цель

Развернуть 4 минимальных Hello World-приложения (Python, Go, Java, Node.js) в контейнерах Docker и настроить NGINX (также в контейнере) как обратный прокси с HTTPS. Использовать переменные окружения (env), Docker secrets, volume, отдельные сети (network), запускать контейнеры не под root, добавить healthcheck, логи, корректно использовать build, CMD и ENTRYPOINT[1] [2] [3] [4].

1. Структура проекта

```
project-root/
│
├── python_app/
│   ├── app.py
│   ├── Dockerfile
│   └── .env
├── node_app/
│   ├── index.js
│   ├── Dockerfile
│   └── .env
├── go_app/
│   ├── main.go
│   ├── Dockerfile
│   └── .env
├── java_app/
│   ├── DemoApplication.java
│   ├── Dockerfile
│   └── .env
├── nginx/
│   ├── nginx.conf
│   ├── Dockerfile
│   ├── ssl/
│   │   ├── selfsigned.crt
│   │   └── selfsigned.key
│   └── .env
├── secrets/
│   └── app_secret.txt
├── data/
│   └── (папки для volume)
├── .env
└── docker-compose.yml
```

- `.env` файлы хранят переменные окружения для сервисов и Compose[1] [2] [3].
- `secrets/` — для Docker secrets[4].

2. Пример .env файла

project-root/.env

```
PYTHON_PORT=8001
NODE_PORT=8002
GO_PORT=8003
JAVA_PORT=8004
SECRET_KEY=supersecret
UID=1000
GID=1000
```

- UID/GID — для запуска контейнеров не под root[5].

3. Использование Docker secrets

docker-compose.yml

```
secrets:
  app_secret:
    file: ./secrets/app_secret.txt

services:
  python_app:
    ...
    secrets:
      - app_secret
    environment:
      - SECRET_FILE=/run/secrets/app_secret
```

- Внутри контейнера секрет доступен как файл `/run/secrets/app_secret`[4] [6].

4. Volumes

- Для хранения данных и логов используйте volumes:

```
services:
  python_app:
    ...
    volumes:
      - python_data:/app/data
      - python_logs:/app/logs
volumes:
  python_data:
  python_logs:
```

- Для NGINX: volume для ssl и логов[7] [8] [9].

5. Сети (networks)

```
networks:
  backend:
  frontend:

services:
  python_app:
    networks:
      - backend
  nginx:
    networks:
      - backend
      - frontend
```

- Изоляция сервисов и прокси[10] [11].

6. User (UID/GID)

- В Dockerfile используйте:

```
RUN addgroup --gid ${GID} appgroup && adduser --uid ${UID} --ingroup appgroup --disabled-
USER appuser
```

- В Compose:

```
user: "${UID}:${GID}"
```

- UID/GID задаются через .env[12] [5].

7. Healthcheck

- В Dockerfile:

```
HEALTHCHECK --interval=30s --timeout=10s --retries=3 \
  CMD curl -f http://localhost:${PYTHON_PORT}/ || exit 1
```

- В Compose:

```
healthcheck:
  test: ["CMD", "curl", "-f", "http://localhost:8001/"]
  interval: 30s
  timeout: 10s
  retries: 3
```

- Для каждого сервиса свой healthcheck[13] [14].

8. Логирование

- Логи приложений пишите в volume, логи NGINX — в отдельный volume[15] [16] [17].
- Для просмотра:
  - `docker compose logs -f`
  - `docker compose logs python_app`

9. Docker build

- Для пересборки:
  - `docker compose build`
- Используйте build-args для передачи переменных на этапе сборки[18] [19].

10. CMD vs ENTRYPOINT

- ENTRYPOINT — основной исполняемый файл, CMD — параметры по умолчанию[20] [21].
- Пример:

```
ENTRYPOINT ["python", "app.py"]
CMD ["--debug"]
```

- Можно переопределять параметры при запуске.

11. Пример docker-compose.yml (фрагмент)

```
version: '3.8'

services:
  python_app:
    build:
      context: ./python_app
      args:
        UID: ${UID}
        GID: ${GID}
    env_file: ./python_app/.env
    secrets:
      - app_secret
    volumes:
      - python_data:/app/data
      - python_logs:/app/logs
    networks:
      - backend
    user: "${UID}:${GID}"
    healthcheck:
```

```
      test: ["CMD", "curl", "-f", "http://localhost:${PYTHON_PORT}/"]
      interval: 30s
      timeout: 10s
      retries: 3
    logging:
      driver: "json-file"
      options:
        max-size: "10m"
        max-file: "3"
  nginx:
    build: ./nginx
    ports:
      - "443:443"
    volumes:
      - ./nginx/ssl:/etc/nginx/ssl
      - nginx_logs:/var/log/nginx
    networks:
      - backend
      - frontend
    depends_on:
      - python_app
      - node_app
      - go_app
      - java_app
    user: "101:101"
    healthcheck:
      test: ["CMD", "curl", "-f", "https://localhost/"]
      interval: 30s
      timeout: 10s
      retries: 3

volumes:
  python_data:
  python_logs:
  nginx_logs:

networks:
  backend:
  frontend:

secrets:
  app_secret:
    file: ./secrets/app_secret.txt
```

12. Чек-лист

Этап	Проверка
	Используются .env и переменные окружения
	Используются Docker secrets
	Все сервисы используют volumes
	Сервисы разделены по сетям

Этап	Проверка
	Контейнеры не запускаются под root
	Для всех сервисов прописан healthcheck
	Логи пишутся в volume и доступны через docker compose logs
	Используются build, CMD, ENTRYPOINT

Полезные ссылки

- Описание .env и переменных: [1] [2] [3]
- Docker secrets: [4] [6]
- Volumes: [7] [8] [9]
- Networks: [10] [11]
- User: [12] [5]
- Healthcheck: [13] [14]
- Логи: [15] [16] [17]
- Build: [18] [19]
- CMD/ENTRYPOINT: [20] [21]

Теперь вы умеете строить production-like инфраструктуру с Docker Compose, используя все лучшие практики DevOps для безопасности, мониторинга и масштабируемости!

⁂

1. https://www.warp.dev/terminus/docker-compose-env-file
2. https://docs.docker.com/compose/how-tos/environment-variables/variable-interpolation/
3. https://collabnix.com/how-to-set-environment-variables-in-docker-compose/
4. https://www.bitdoze.com/docker-compose-secrets/
5. https://blog.giovannidemizio.eu/2021/05/24/how-to-set-user-and-group-in-docker-compose/
6. https://stackoverflow.com/questions/42139605/how-do-you-manage-secret-values-with-docker-compose-v3-1
7. https://docs.docker.com/engine/storage/volumes/
8. https://kinsta.com/blog/docker-compose-volumes/
9. https://docs.docker.com/reference/compose-file/volumes/
10. https://www.netmaker.io/resources/docker-compose-network
11. https://www.warp.dev/terminus/docker-compose-networks
12. https://stackoverflow.com/questions/48727548/how-to-configure-docker-compose-yml-to-up-a-container-as-root
13. https://cyberpanel.net/blog/docker-compose-healthcheck
14. https://signoz.io/guides/how-to-view-docker-compose-healthcheck-logs/

15. https://spacelift.io/blog/docker-compose-logs
16. https://signoz.io/guides/docker-compose-logs/
17. https://kodekloud.com/blog/docker-compose-logs/
18. https://spacelift.io/blog/docker-compose
19. https://docs.docker.com/reference/cli/docker/compose/build/
20. https://spacelift.io/blog/docker-entrypoint-vs-cmd
21. https://refine.dev/blog/docker-entrypoint/