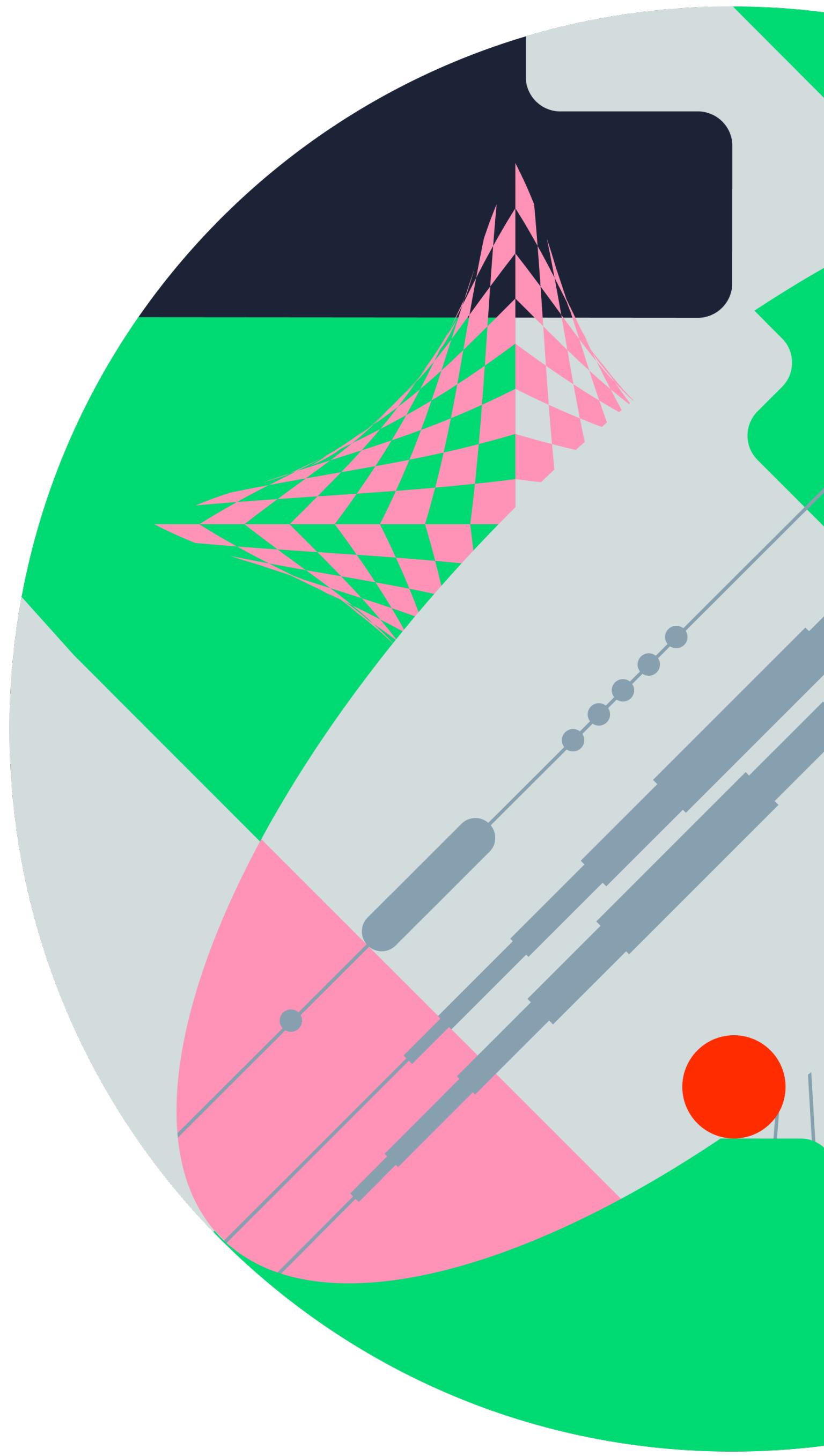
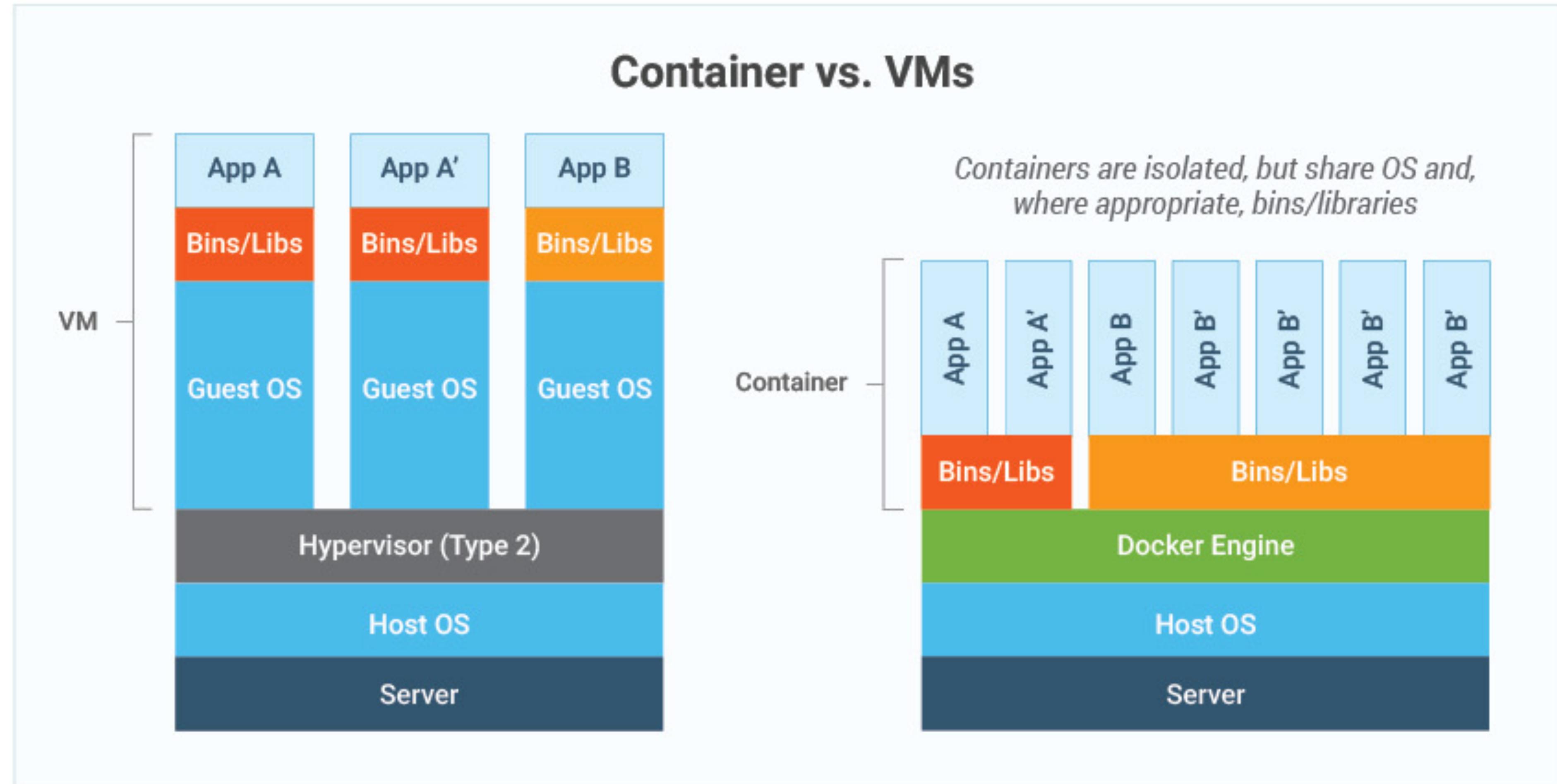


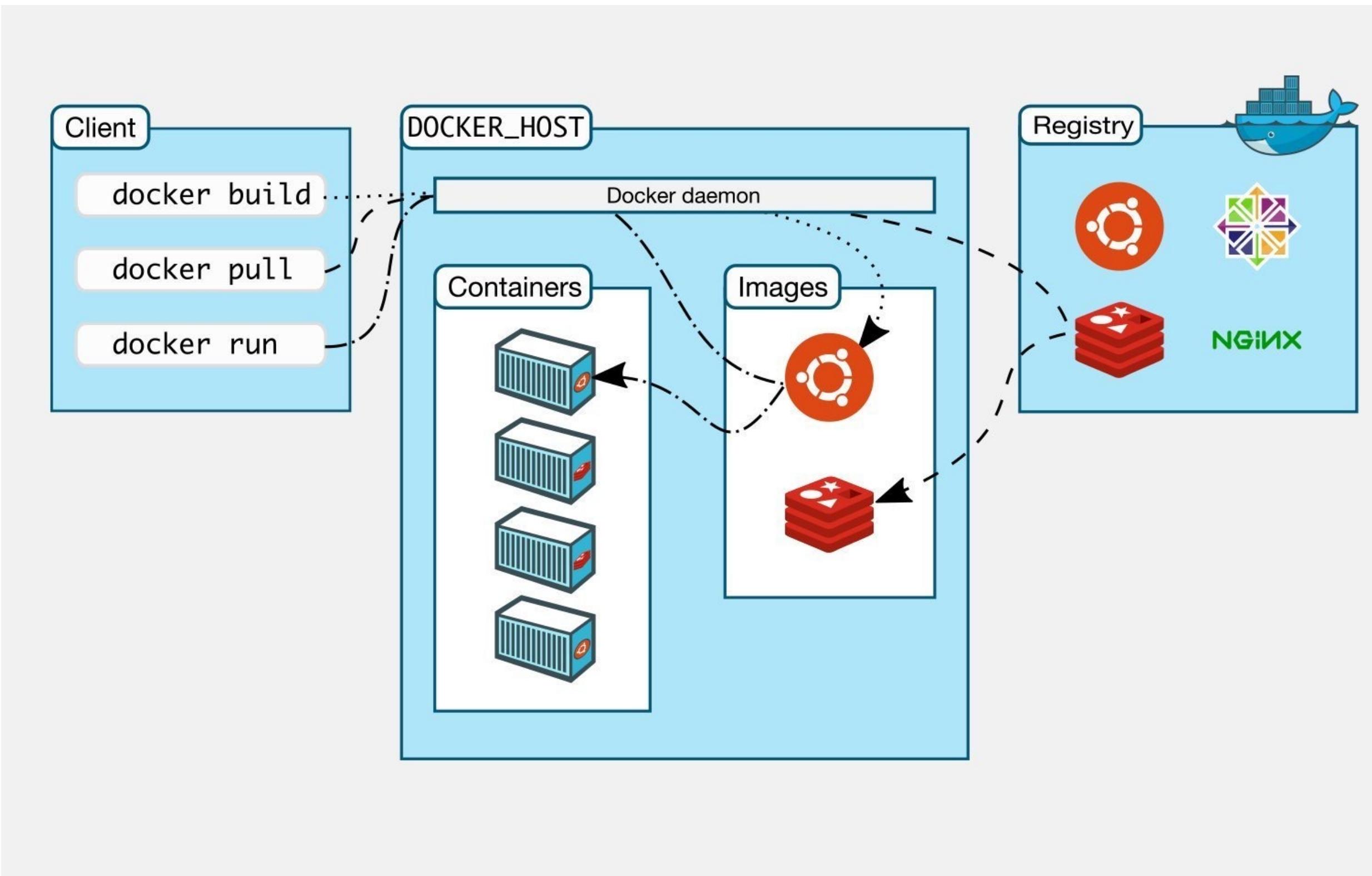
# Docker, Docker Compose, CI/CD



# Контейнеризация и виртуализация



# Основные понятия



● **Docker engine** – клиент-серверное приложение, состоящее из:

- Сервера с демоном **dockerd**, который создает образы, контейнеры, сети и тома, а также управляет ими
- **API** для отправки команд демону
- **CLI** для взаимодействия с сервером через API



# Управление образами

- Docker image – шаблон для создания контейнера, состоящий из ОС, приложений и их зависимостей

```
1 docker build -t <image_name> . //Собрать образ из Dockerfile  
2 docker images //Список образов  
3 docker pull <image_name> //Обновить образ из реестра  
4 docker rmi <image_name> //Удалить образ  
5 docker image prune //Удалить все неиспользуемые образы
```



# Управление контейнерами: команды

**Docker container** – изолированная среда для исполнения экземпляра приложения, построенного на основе Docker image

- 1 docker ps //Показать все контейнеры
- 2 docker run <image\_name> //Запустить контейнер из образа
- 3 docker stop //Остановить контейнер
- 4 docker start //Запустить контейнер
- 5 docker exec -it <container\_id> bash //Выполнить команду в запущенном контейнере
- 6 docker rm <container\_id> //Удалить остановленный контейнер
- 7 docker restart <container\_id> //Перезапустить контейнер
- 8 docker logs -f <container\_id> //Посмотреть логи контейнера
- 9 docker stats <container\_id> //Информация по используемым ресурсам



# Управление контейнерами: тома

- **Docker volume** – файловая система за пределами контейнера (на хост-машине) для постоянного хранения данных

```
1 docker volume create <volume_name> . //Создать том volume_name  
2 docker volume ls //Список томов  
3 docker volume inspect <volume_name> //Информация по тому  
4 docker rm <volume_name> //Удалить том  
5 docker volume prune //Удалить все неиспользуемые тома
```



# Сети

- Docker network – механизм для связи контейнеров между собой и с внешними системами

Виды сетей по изоляции:

- **none**. Полная сетевая изоляция, без доступа до внешней сети или других контейнеров
- **host**. Нет сетевой изоляции, использует напрямую сетевой стек хостовой системы
- **bridge**. Тип сети по умолчанию
- **macvlan**. MAC-адреса для контейнеров

```
1 docker network create <network_name> . //Создать сеть network_name
2 docker network ls //Список сетей
3 docker network dis/connect <network_name> <container_id> //Дис/коннект контейнера к сети
4 docker network inspect <network_name> //Информация о сети
5 docker network rm <network_name> //Удалить сеть
6 docker run --network none/host <image_name> //Указать сеть явно (только эти значения)
```



# Ключи и флаги

```
1 docker run -p 8080:80 nginx //Пробросить порт
2 docker run -v /data:/app/data myapp //Подключить том
3 docker run --env-file .env myapp //Пробросить переменные окружения
4 docker run --restart=on_failure nginx //Политика перезапуска контейнера
5 docker build --no-cache -t myapp . //Собрать образ без использования кеша
6 docker <cmd_name> --help //В любой непонятной ситуации
```



# Пишем свой Dockerfile: hello world

Dockerfile – файл с инструкциями для создания Docker image

```
1 FROM golang:alpine #image:tag
2 WORKDIR /cmd
3
4 COPY go.mod go.sum ./
5 COPY main.go ./
6
7 RUN go mod download
8
9
10 # app - название приложения
11 RUN go build -o app .
12
13 # Не открывает порт сама по себе
14 EXPOSE 8080
15
16 # Точка входа (исполняемый файл)
17 ENTRYPOINT ["/app/cmd"]
18
19 # Аргументы по умолчанию для ENTRYPOINT
20 CMD ["--mode=production"]
```

- Образ
- Зависимости
- Приложение

## Минусы:

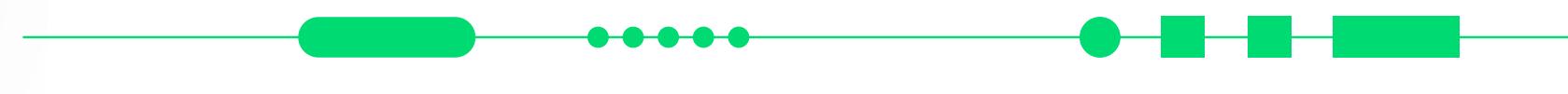
- Размер образа (SDK, кэш, исходники)
- Большая поверхность атаки
- Пересборка всех слоев при любых изменениях

# Пишем свой Dockerfile: multistage сборка

```
1 # Этап сборки
2 FROM golang:1.23-alpine AS builder
3 WORKDIR /cmd
4 COPY .
5 RUN go mod download && CGO_ENABLED=0 go build -o app
6
7 # Этап запуска (минимальный образ)
8 FROM alpine:3.21
9 WORKDIR /cmd
10 COPY --from=builder /app/cmd . # Перенос только бинарника
11 CMD ["../app"]
```

## А ещё быстрее?

- Копировать только нужное
- .dockerignore
- Объединять команды в один слой
- Удалять кеш и временные файлы



# Docker compose

**Docker compose** – инструмент для управления мультиконтейнерными приложениями

Пример `docker-compose.yml`:

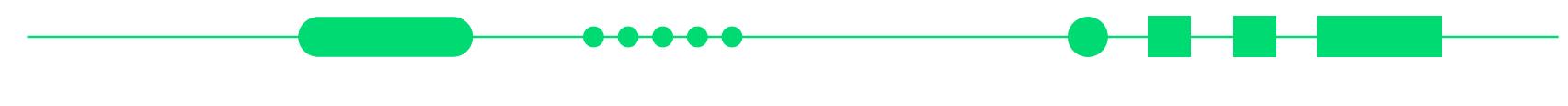
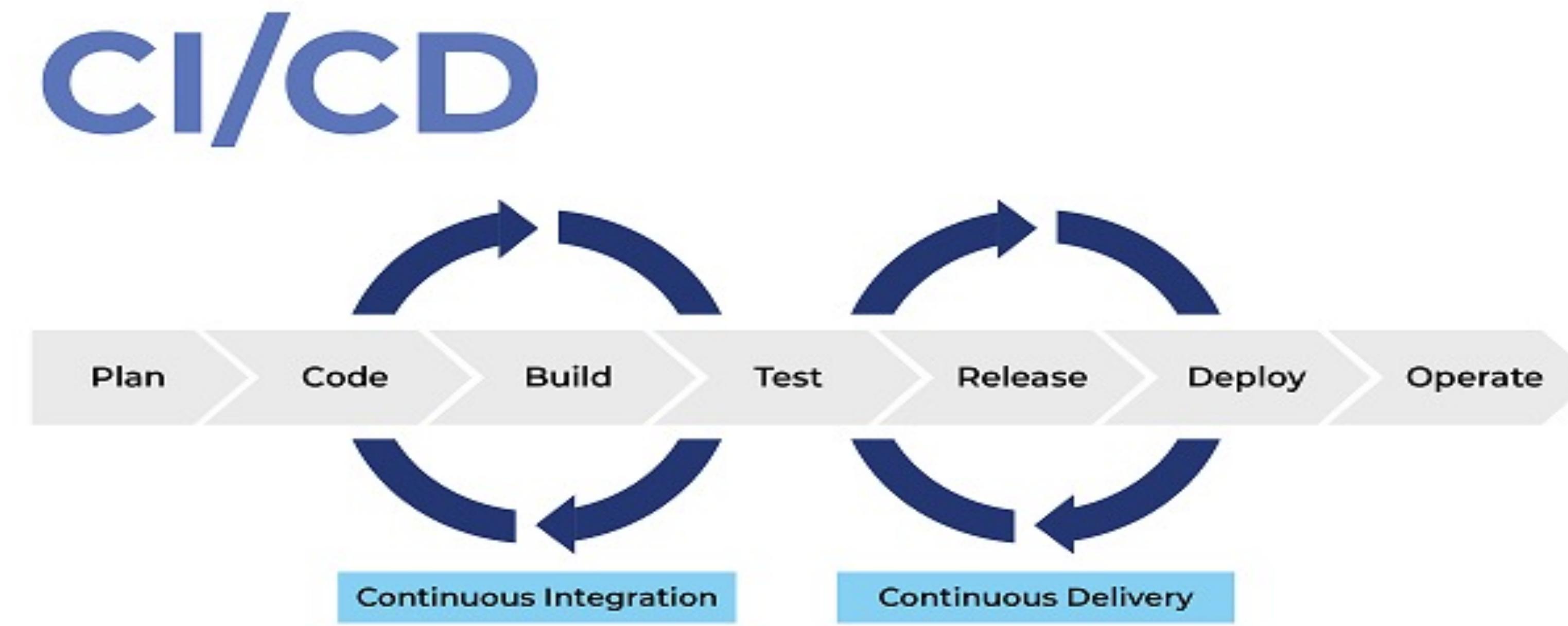
```
1 services:
2   app:
3     image: dockertest:local
4     container_name: app_container
5     restart: always
6     build:
7       context: .
8       dockerfile: Dockerfile
9     ports:
10      - ${APP_PORT}:${APP_PORT}
11   volumes:
12     - ./configs:/configs
13     - ./templates:/templates
14   depends_on:
15     - db
16
17 db:
18   image: postgres:${POSTGRES_VERSION}
19   restart: always
20   container_name: postgres_container
21   ports:
22     - ${POSTGRES_PORT}:${POSTGRES_PORT}
23   environment:
24     - POSTGRES_DB=${POSTGRES_DB}
25     - POSTGRES_USER=${POSTGRES_USER}
26     - POSTGRES_PASSWORD=${POSTGRES_PASSWORD}
```

```
1 docker-compose up -d // Запуск в фоновом режиме
2 docker-compose down // Удаляет контейнеры, сети и тома
3 docker-compose build // Сборка
4 docker-compose logs -f <service_name> // Просмотр логов
5 docker-compose start | stop | restart <service_name> // Управление сервисом
6 docker-compose down && docker-compose build --no-cache && docker-compose up //It works
```



# CI/CD

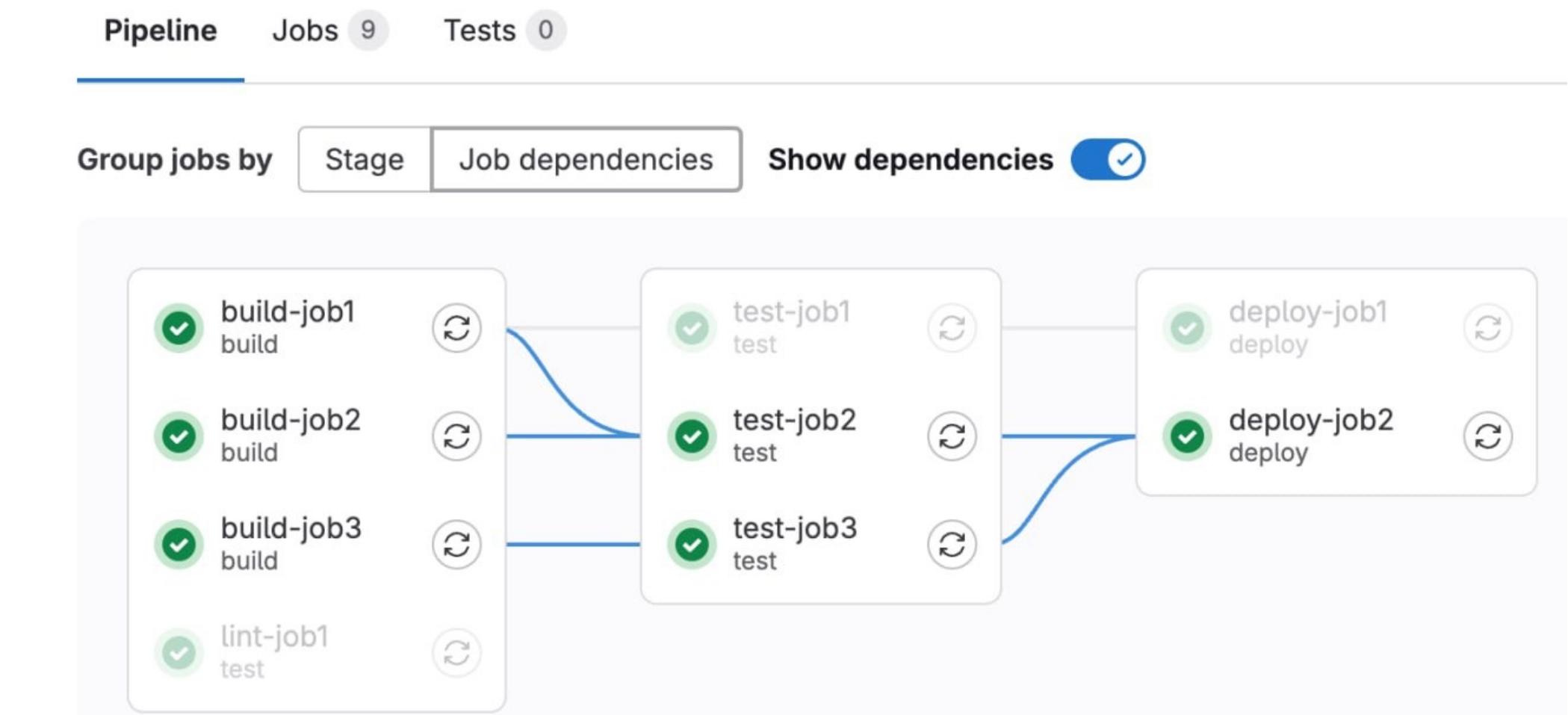
- **Continuous integration (CI)** – Непрерывная интеграция
- **Continuous delivery (CD)** – Непрерывная доставка
- **CI/CD Pipeline** – автоматизированная последовательность действий для интеграции, тестирования и развертывания кода



# CI/CD pipeline

```
1 stages:
2   - build
3   - test
4   - deploy
5
6 build-job:
7   stage: build
8   script:
9     - go build -o app ./cmd/
10  artifacts:
11    paths:
12      - app # Для последующих стадий
13
14 test-job:
15   stage: test
16   script:
17     - go test -v -race ./...
18     - go vet ./...
19
20 deploy-prod:
21   stage: deploy
22   only:
23     - main
24   dependencies:
25     - build-job
26   script:
27     - scp -P $SSH_PORT app deploy@$SERVER_IP:/app/ #gitlab variables
28     - ssh -p $SSH_PORT deploy@$SERVER_IP "sudo systemctl restart app"
```

- Стадии сборки
- Джобы привязаны к стадии. Исполняются параллельно или по порядку (зависимости)
- Artifacts, script, before\_script, when, ...
- Gitlab variables



# CI/CD pipeline: docker

```
1 stages:
2   - build
3   - tests
4   - cleanup
5
6 unit tests:
7   stage: unit tests
8   needs: []
9   image: golang:1.22-alpine
10  coverage: /total:\s+(statements)\s+\d+.\d+%
11  script:
12    - cp configs/.env.template configs/.env
13    - cp configs/Docker.env.template configs/Docker.env
14    - go mod download
15    - go test -v ./...
16
17 build:
18   stage: build
19   needs: []
20   script:
21     - cp configs/Docker.env.template configs/.env
22     - docker rm -f test-container || true
23     - docker build -t tests:latest -f build/integrationTests/Dockerfile .
24
25 tests:
26   stage: tests
27   script:
28     - cp configs/Docker.env.template configs/.env
29     - docker compose -f deployments/docker-compose-dev.yml down
30     - docker compose -f deployments/docker-compose-dev.yml up -d
31     - docker run --network=deployments_service-network --name test-container
32   tests:latest
33
34 cleanup:
35   stage: cleanup
36   when: always
37   script:
38     - cp configs/.env.template configs/.env
39     - docker compose -f deployments/docker-compose-dev.yml down || true
40     - docker rm -f test-container || true
41     - docker image prune -fa
42     - docker volume prune -f
43     - docker network prune -f
```

## Когда?

- Нужна production like среда
- Оркестрация
- Масштабирование
- «Сложные» зависимости

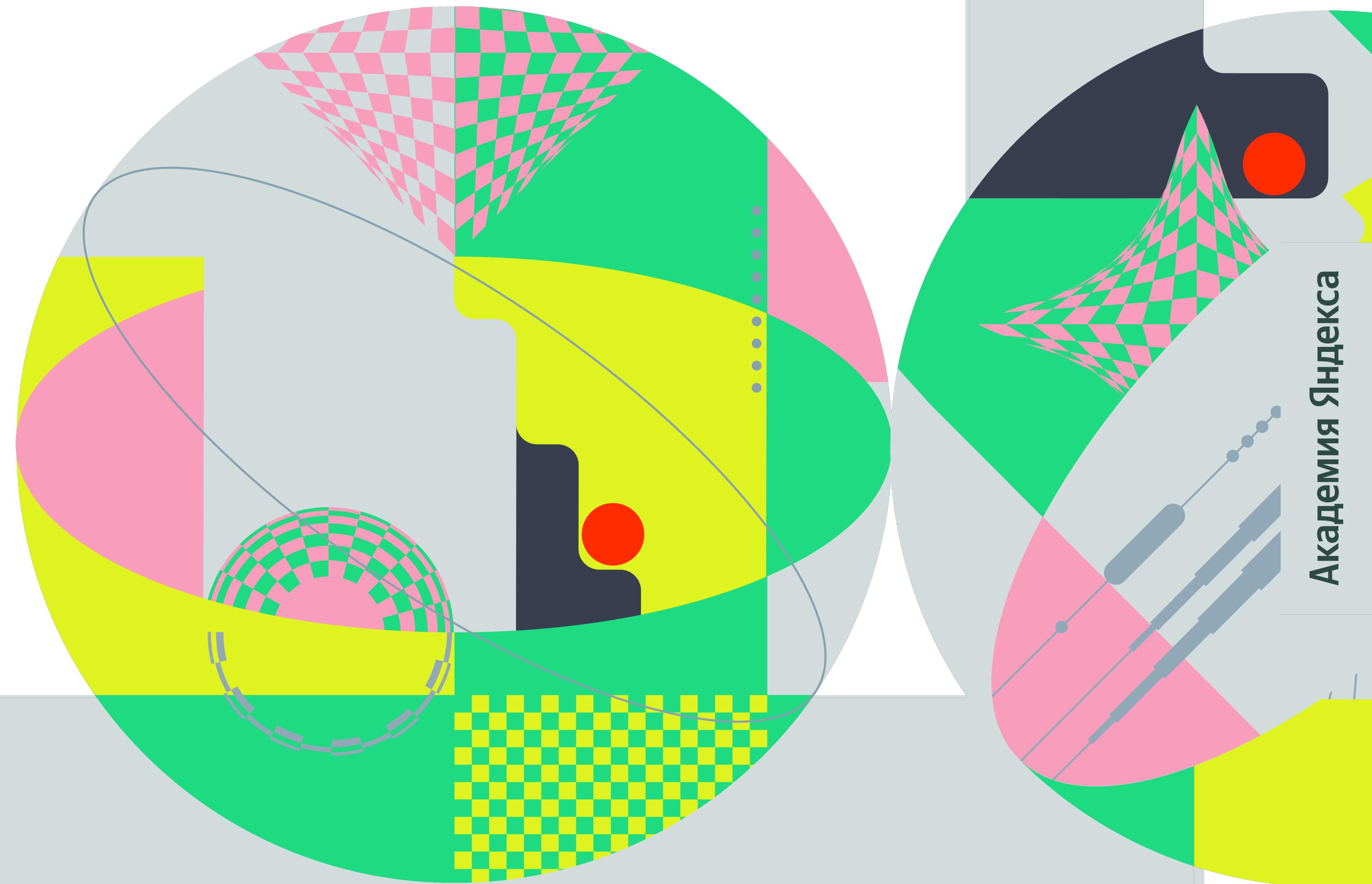


# CI/CD pipeline: docker

Критерий	Jenkins	GitHub Actions	Gitlab CI
Использование	Плагин	Нативно для Github	Нативно для Gitlab
Безопасность	Зависит от плагинов	Базовая	Есть DevSecOps
Масштабируемость	Плагин для k8s	Через Actions	Auto DevOps
Матричные сборки	Плагин	Через YAML	Из коробки



# Спасибо за внимание!



Академия Яндекса

[academy.yandex.ru/lyceum](http://academy.yandex.ru/lyceum)

ask@yandexlyceum.ru