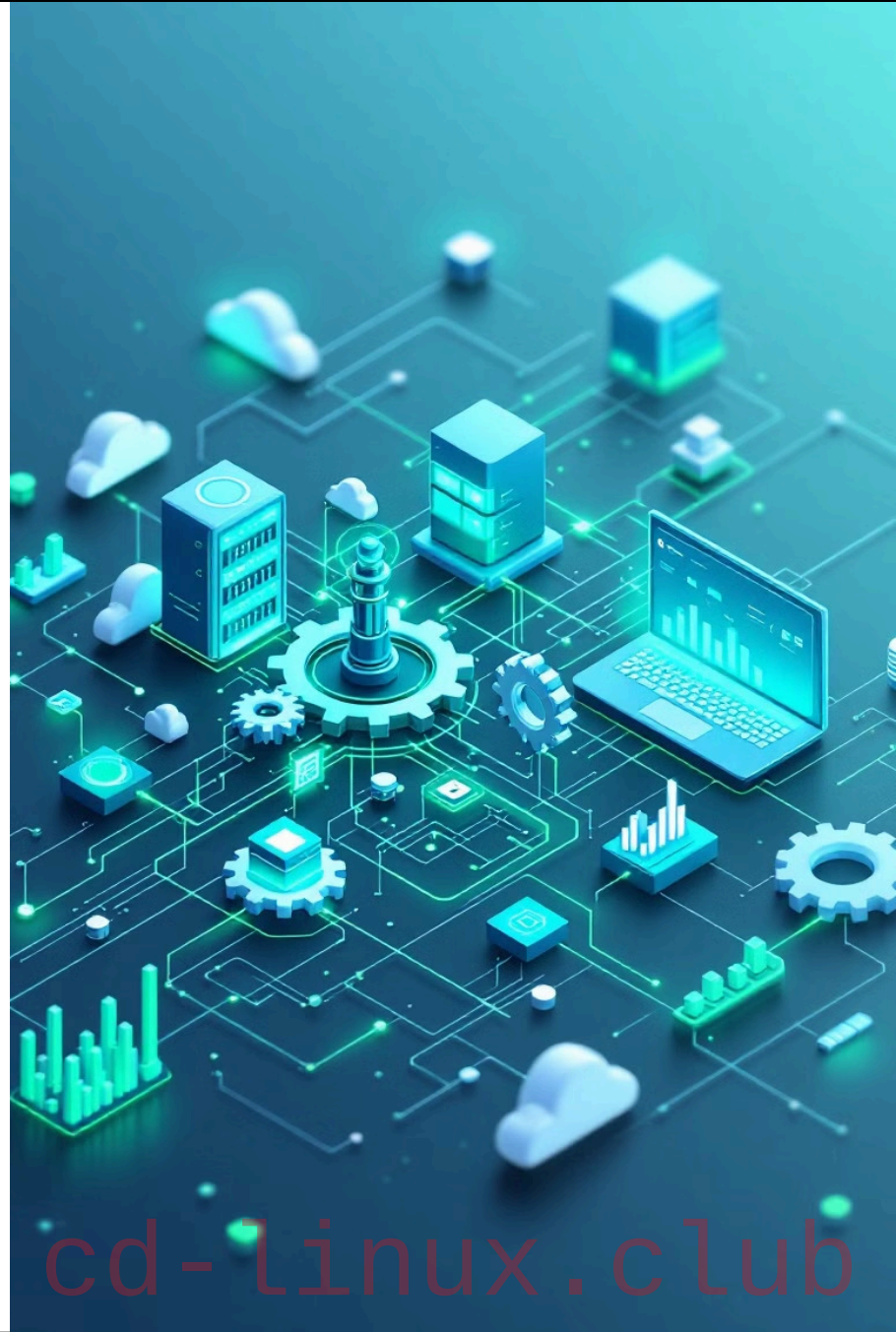
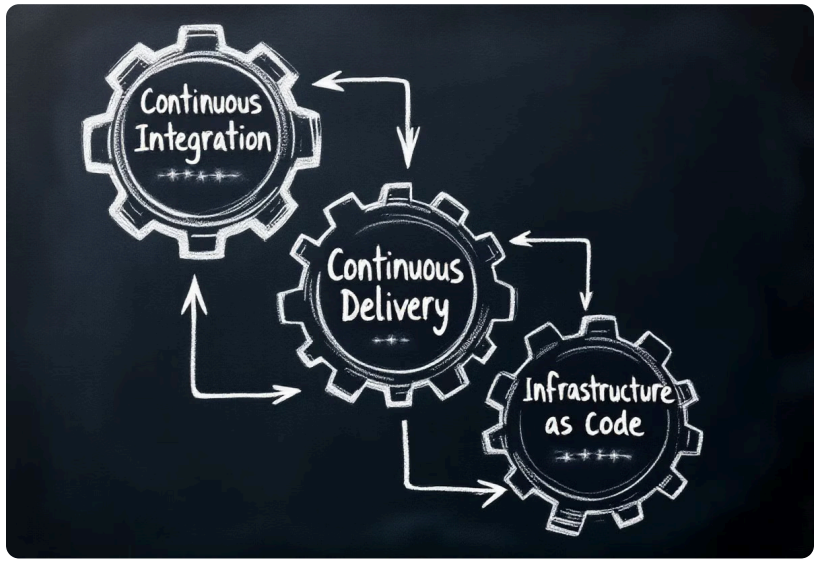
CI/CD. Часть 4

Сегодня мы рассмотрим практические аспекты создания и настройки CI/CD процессов. Погрузимся в детали разработки пайплайнов, интеграции с VSC и оптимизации сборок через параллельное исполнение.

by Nikita Bezmen



План урока



Разработка pipeline CI/CD

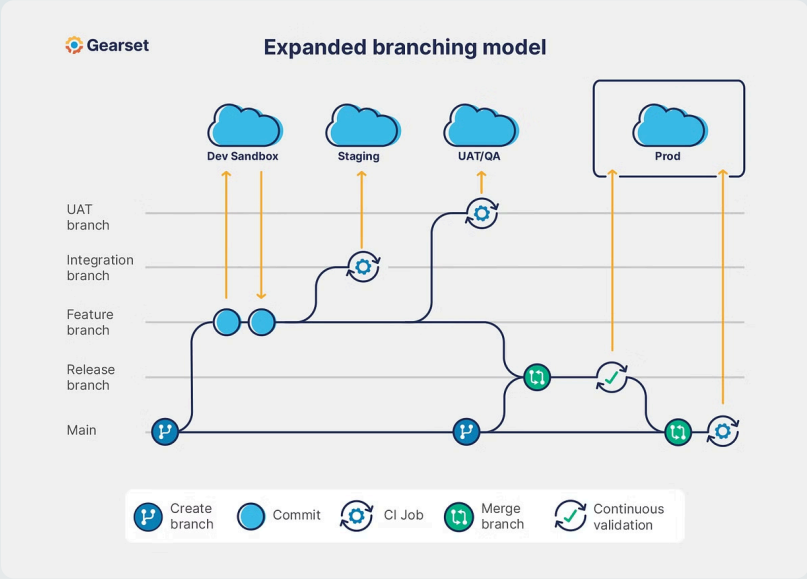
Создание автоматизированных цепочек для интеграции, тестирования и доставки кода

Интеграция с VSC

Использование Visual Studio Code в сочетании с инструментами DevOps

Параллельное исполнение сборки

Ускорение процессов сборки за счет параллельных вычислений



Разработка pipeline CI/CD

Планирование

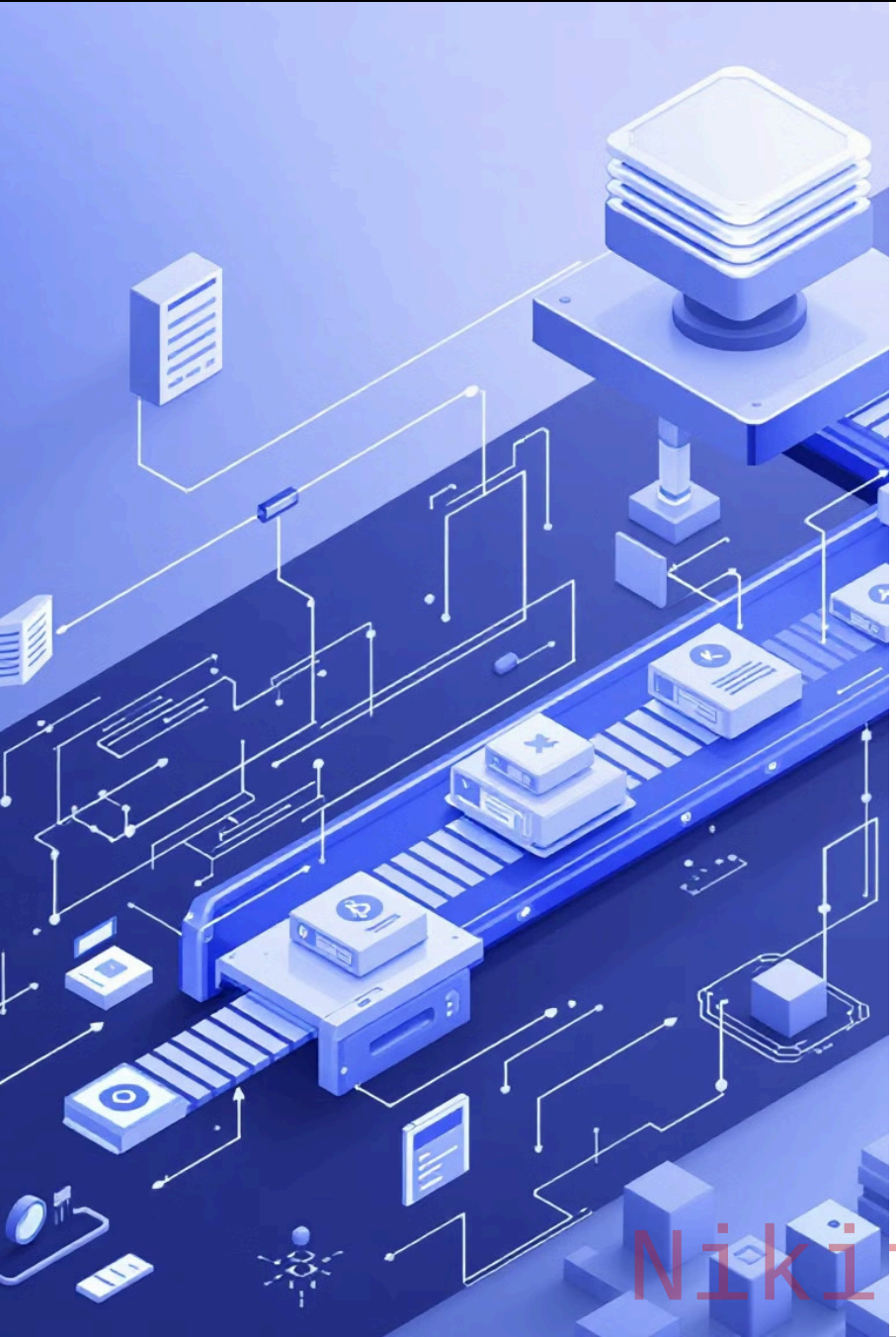
Определение этапов и их последовательности в вашем пайплайне

Выбор инструментов

Подбор подходящих CI/CD инструментов под ваш проект

Конфигурация

Настройка автоматизации для каждого этапа пайплайна



Что такое CI/CD пайплайн?

Автоматизированный процесс

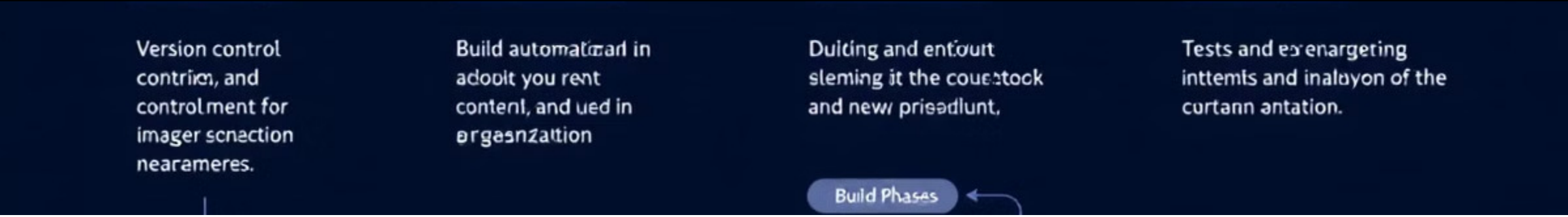
Связанная цепочка действий, которая запускается при изменении кода

Интеграция и доставка

Автоматически выполняет тестирование, сборку и развертывание приложения

Эффективность

Позволяет ускорить процесс разработки и снизить количество ошибок



Шаги в пайплайне CI/CD

Контроль версий

Управление исходным кодом через системы вроде Git

Автоматическая сборка

Сборка приложения после каждого коммита

- mvn clean install
- mvn package
- mvn deploy

Автоматическое тестирование

Запуск модульных, интеграционных и функциональных тестов

Тестирование и CI-сервер

Уровни тестирования

- Модульные тесты
- Интеграционные тесты
- Функциональные тесты
- Нагрузочные тесты

CI-серверы

- Jenkins
- Travis CI
- GitLab CI
- CircleCI
- GitHub Actions

Настройка пайплайна в GitLab CI/CD

Создание файла конфигурации

Создайте .gitlab-ci.yml в корне проекта

Определение этапов

Настройте этапы сборки, тестирования и деплоя

Настройка среды

Укажите Docker-образы для каждого этапа

Создание скриптов

Напишите скрипты для выполнения задач

Настройка условий и оповещений

Добавьте условия запуска и создайте оповещения

Пример .gitlab-ci.yml файла

```
stages:
  - build
  - test
  - deploy

build_job:
  stage: build
  script:
    - echo "Building the project..."
    - mvn clean package

test_job:
  stage: test
  script:
    - echo "Running tests..."
    - mvn test

deploy_job:
  stage: deploy
  script:
    - echo "Deploying to production..."
    - mvn deploy
```

Настройка окружений

Тестовое окружение

Для проверки кода перед релизом

Стейджинг

Имитирует продакшн-среду для финальной проверки

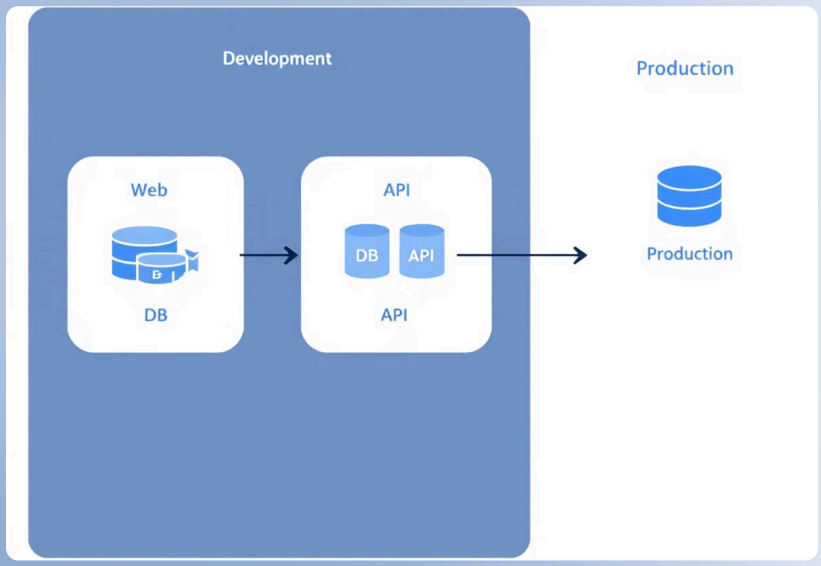
Откат

Возможность возврата к стабильной версии

Продакшн

Для работающего приложения с реальными пользователями

Настройка Docker Compose для сред



Создание файла конфигурации

Создайте docker-compose.yml в корне проекта

Определение сервисов

Укажите необходимые сервисы, образы и порты

Настройка окружений

Создайте отдельные конфигурации для тестовой и продакшн среды

Задайте параметры окружений

Настройте переменные среды, базы данных и другие параметры

```yaml
version: '3'

services:
  app:
    image: my-image-name
    ports:
      - "8000:8000"
    environment:
      ENVIRONMENT: ${ENVIRONMENT}
      DATABASE_URL: ${DATABASE_URL}

environments:
  test:
    <<: *default
    environment:
      - ENVIRONMENT=test
      - DATABASE_URL=postgres://testuser:testpassword@db/testdb

  production:
    <<: *default
    environment:
      - ENVIRONMENT=production
      - DATABASE_URL=postgres://produser:prodpassword@db/proddb
```

Пример docker-compose.yml

Запуск тестового окружения

docker-compose up -d --build --environment=test

Запуск продакшн окружения

docker-compose up -d --build --environment=production

Настройка мониторинга

Сбор метрик

Отслеживание ключевых показателей работы приложения

Проактивное обнаружение

Выявление проблем до их влияния на пользователей

Алертинг

Оповещение команды о потенциальных проблемах

Анализ производительности

Выявление узких мест и оптимизация работы

Настройка мониторинга с Prometheus

Установка инструментов

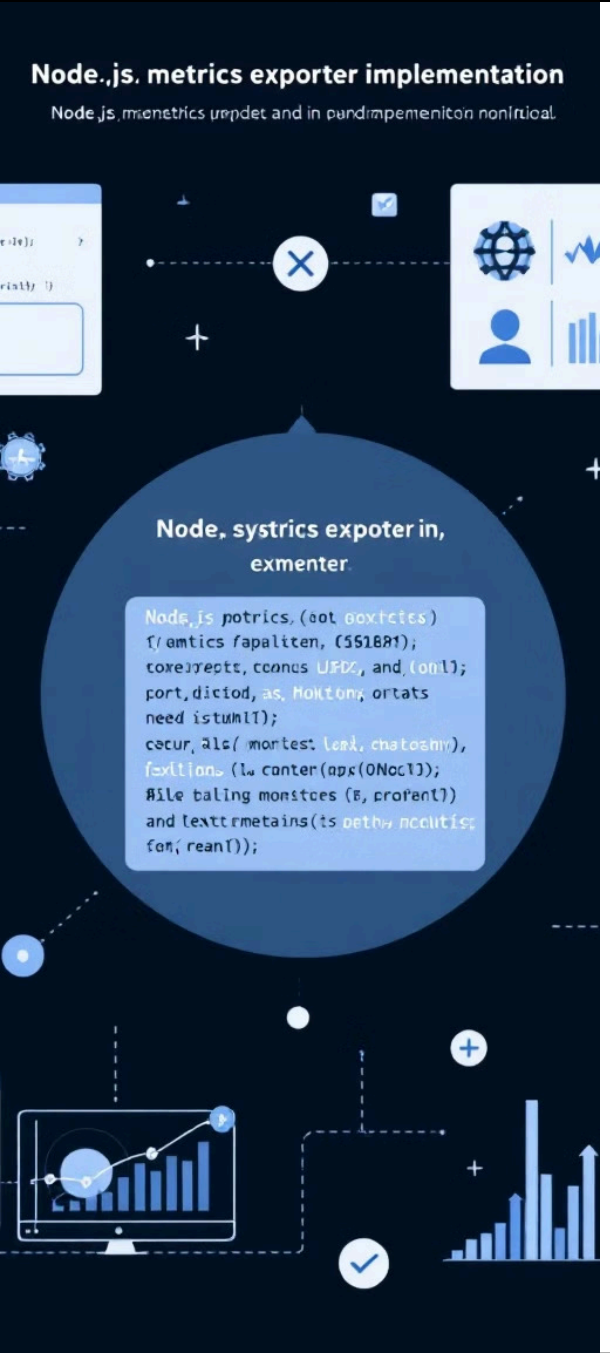
Разверните Prometheus и Grafana на сервере

Настройка конфигурации

```
global:
  scrape_interval: 15s
  evaluation_interval: 15s
scrape_configs:
  - job_name: 'my_http_server'
    metrics_path: /metrics
    static_configs:
      - targets:
['my_http_server:9090']
```

Подключение экспортеров

Создайте экспортеры метрик для вашего приложения



Настройка экспортера метрик

```
const Prometheus = require('prom-client');

const collectDefaultMetrics = () => {
  Prometheus.collectDefaultMetrics();
};

const metricsMiddleware = (req, res) => {
  res.set('Content-Type', Prometheus.register.contentType);
  res.end(Prometheus.register.metrics());
};

module.exports = {
  collectDefaultMetrics,
  metricsMiddleware,
};
```

Визуализация и алерты в Grafana

Создание дашбордов

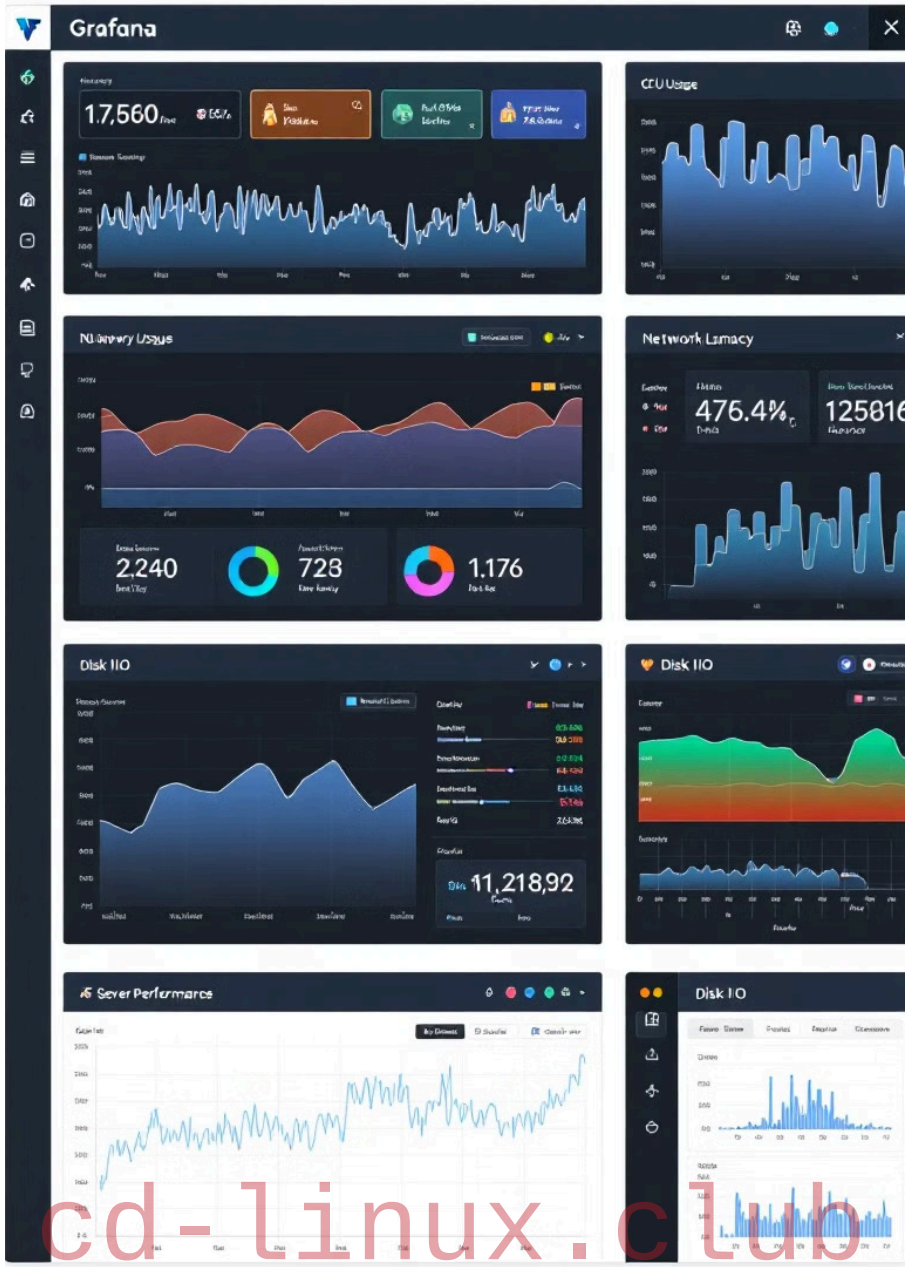
Визуализация ключевых метрик производительности

Настройка Alertmanager

Конфигурация системы оповещений

Интеграция со Slack

Настройка оповещений в рабочие каналы



Система уведомлений

Обнаружение проблемы

Мониторинг выявляет отклонение от нормы

Создание алерта

Формирование структурированного оповещения

Отправка уведомления

Доставка через настроенные каналы связи

Реакция команды

Расследование и устранение проблемы

Настройка уведомлений в Slack

Создание бота

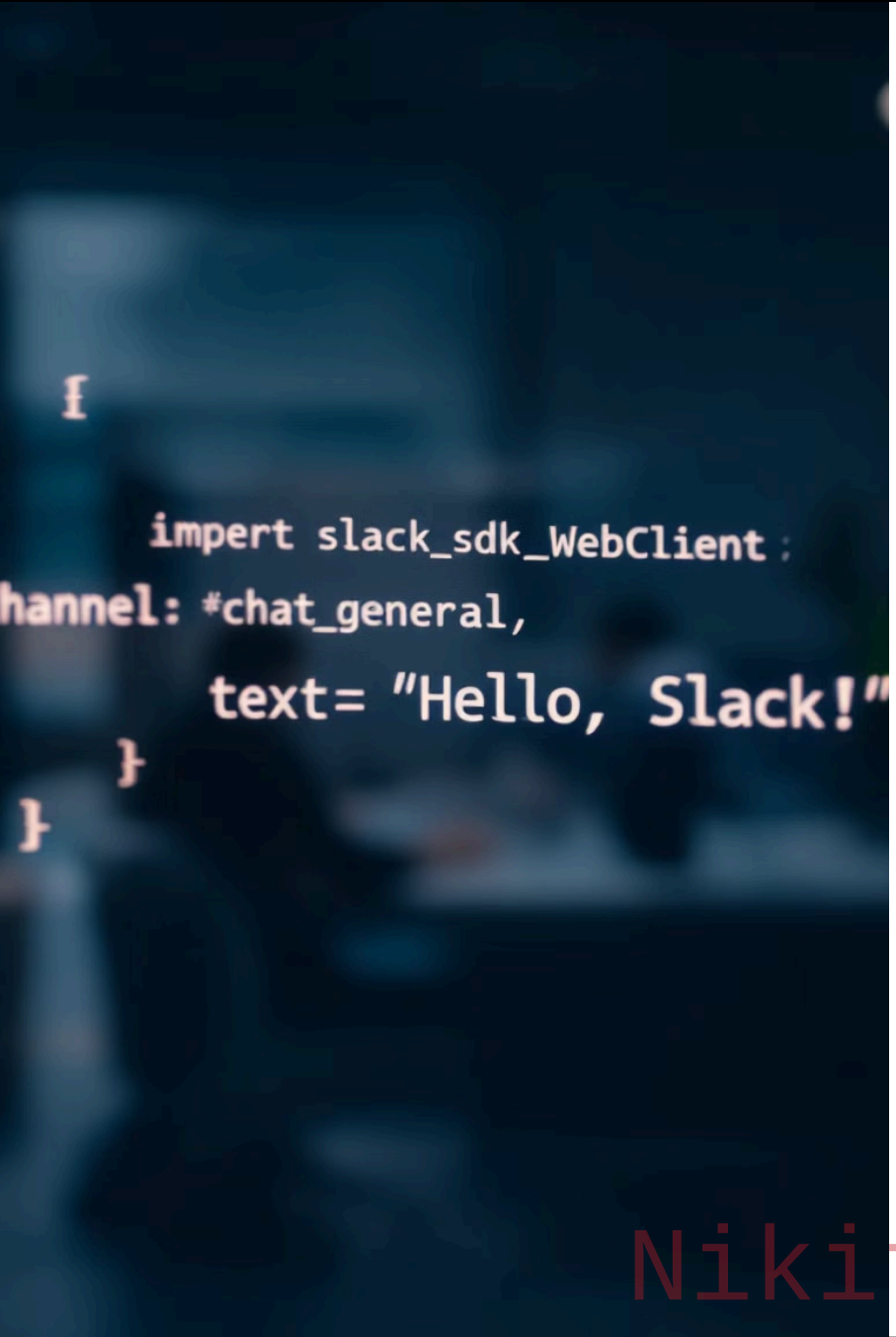
Зарегистрируйте нового Slack-бота и получите токен доступа

Добавление токена

```
variables:
  SLACK_WEBHOOK: ""
  SLACK_CHANNEL: "#"
  SLACK_BOT_TOKEN: ""
```

Настройка канала

Определите канал для получения уведомлений о состоянии пайплайна



Скрипт для отправки уведомлений

```
import os
import slack

def send_notification(message):
    client = slack.WebClient(token=os.environ['SLACK_BOT_TOKEN'])
    response = client.chat_postMessage(
        channel=os.environ['SLACK_CHANNEL'],
        text=message
    )
    return response

# Пример отправки уведомления
send_notification("Произошла ошибка во время сборки проекта")
```

Интеграция скрипта в пайплайн

1

Обнаружение ошибки

Пайплайн выявляет проблему при сборке или тестировании

2

Вызов скрипта уведомлений

Активация Python-скрипта с параметрами ошибки

3

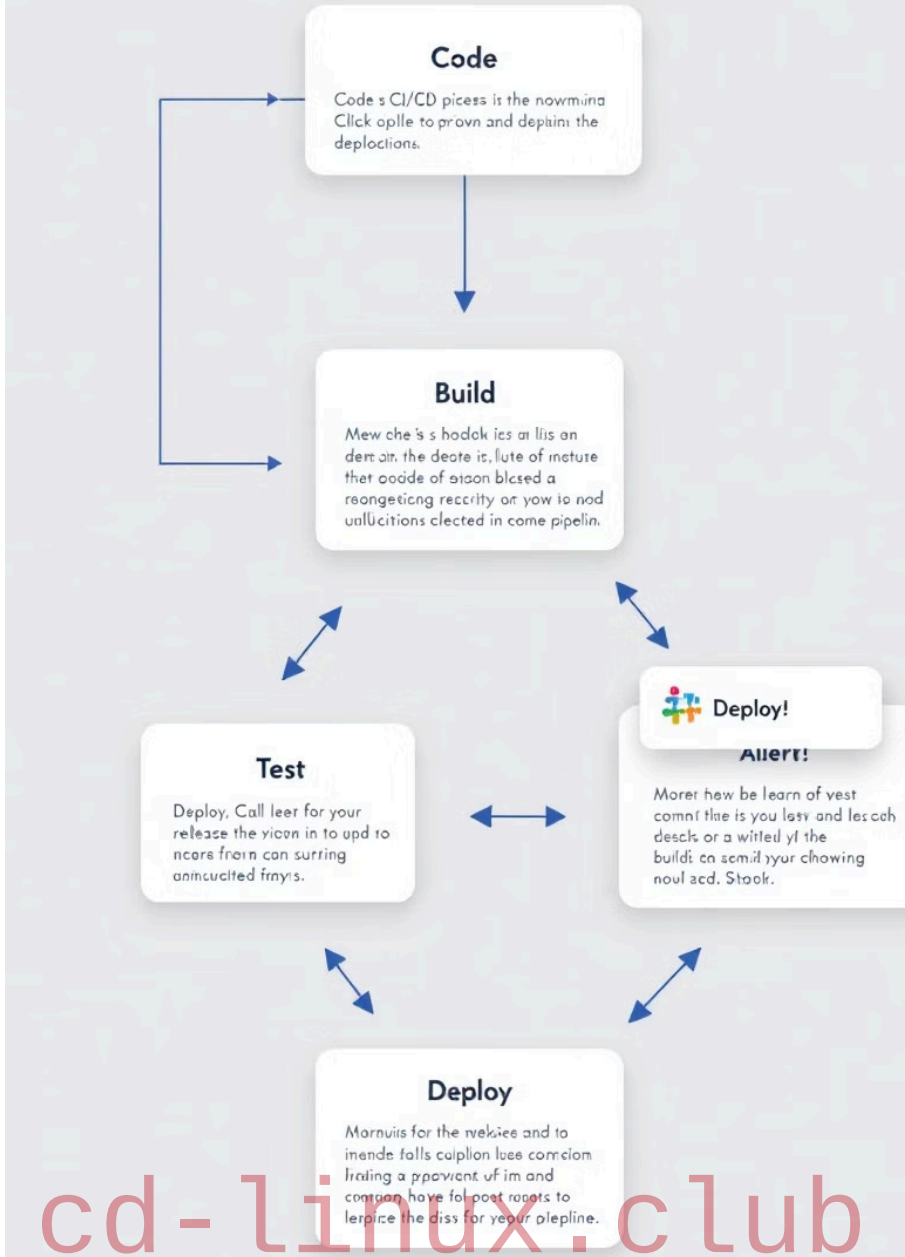
Отправка в Slack

Формирование и доставка структурированного сообщения

4

Реакция команды

Разработчики получают уведомление и исправляют проблему



Тестирование CI/CD пайплайна

Модульное тестирование

Проверка отдельных компонентов пайплайна

Интеграционное тестирование

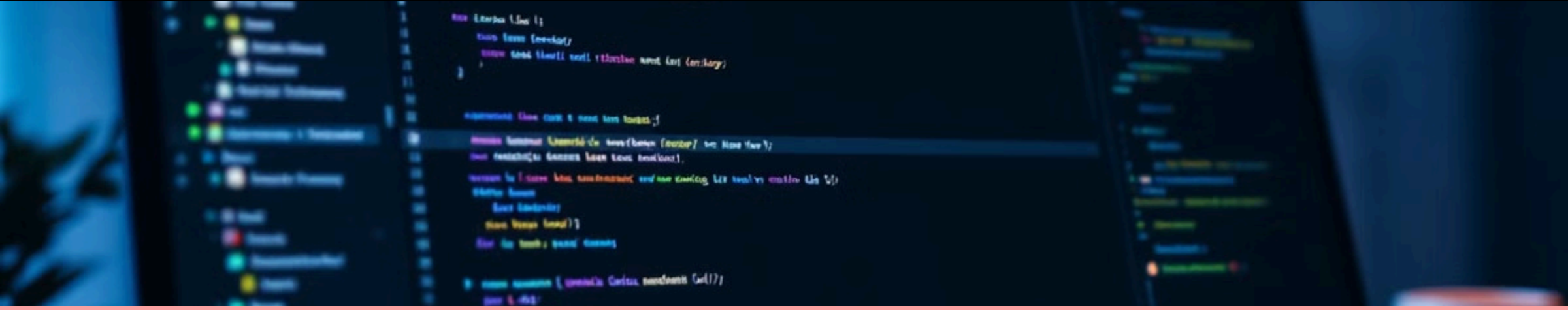
Проверка взаимодействия компонентов

Стресс-тестирование

Проверка под нагрузкой множественных сборок

End-to-End тестирование

Проверка всего процесса от коммита до деплоя



Интеграция с VSC

Среда разработки

VSC - мощный редактор кода с поддержкой расширений

DevOps интеграция

Возможность работать с инструментами DevOps прямо из редактора

Автоматизация процессов

Запуск команд и скриптов без переключения между программами

Способы интеграции VSC

Расширения

Установка расширений для различных языков и инструментов

Командная строка

Запуск команд прямо из встроенного терминала VSC

Встроенные средства

Использование встроенных инструментов для Git, GitHub, Azure

Использование расширений VSC

Установка расширения

Откройте Market Place и найдите нужное расширение (например, Python)

Создание файла

Создайте новый файл с соответствующим расширением (.py)

Написание кода

Напишите код, используя подсказки и автодополнение

Сохранение и запуск

Сохраните (Ctrl+S) и запустите код (F5)

Работа с терминалом VSC

Открытие терминала

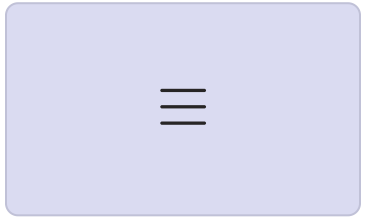
Ctrl+Shift+` или через меню "Вид" -> "Терминал"

В терминале можно выполнять любые команды OS и инструментов

Примеры команд

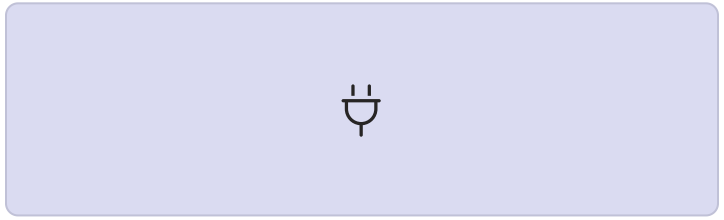
- npm install - установка зависимостей
- git commit - сохранение изменений
- docker build - сборка контейнера
- kubectl apply - деплой в Kubernetes

Встроенные средства интеграции



Открыть боковое меню

Нажмите Ctrl+B для доступа к боковой панели

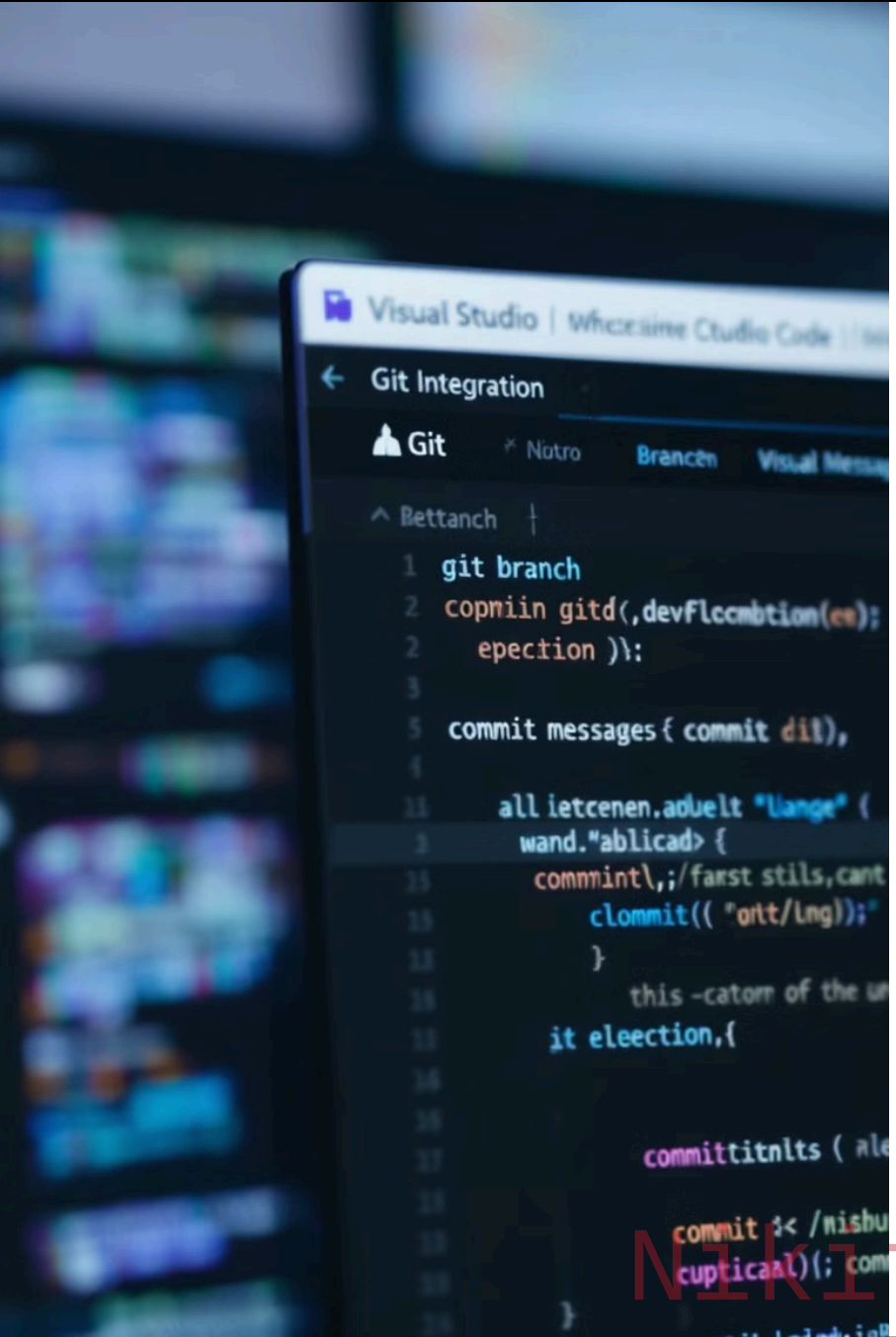


Выбрать инструмент

Нажмите на нужный инструмент (Git, GitHub)

Ввести учетные данные

Авторизуйтесь в сервисе при необходимости



Интеграция с Git

Создание репозитория

Инициализация нового или клонирование существующего Git-репозитория

Управление ветками

Создание и переключение между ветками разработки

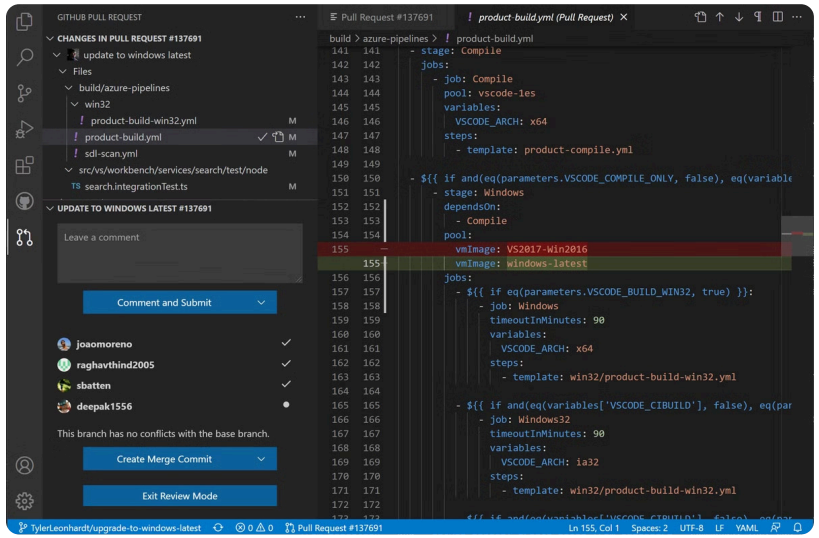
Коммиты

Фиксация изменений и отправка в удаленный репозиторий

История

Просмотр истории изменений и сравнение версий

Работа с Git в VSC



Клонирование репозитория

git clone https://github.com/YOUR-USERNAME/YOUR-REPOSITORY

Отправка коммитов

git checkout master
git add README
git commit -m "Added shared comment to readme"
git push shared master

Просмотр истории

git log – покажет историю коммитов в хронологическом порядке

Вывод команды git log

Команда git log выводит список коммитов с их хеш-суммами, автором, датой и сообщением. Это помогает отслеживать историю изменений проекта и находить нужные коммиты.

```
$ git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    Change version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    Remove unnecessary test

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700

    Initial commit
```

Особенности вывода git log

Обратный хронологический порядок

Последние коммиты отображаются вверху списка

SHA-1 контрольные суммы

Уникальные идентификаторы для каждого коммита

Информация об авторе

Имя и email создателя коммита

Дата и время

Когда был создан коммит

Управление ветками в Git

1

git branch

Создание, просмотр и удаление веток

git checkout

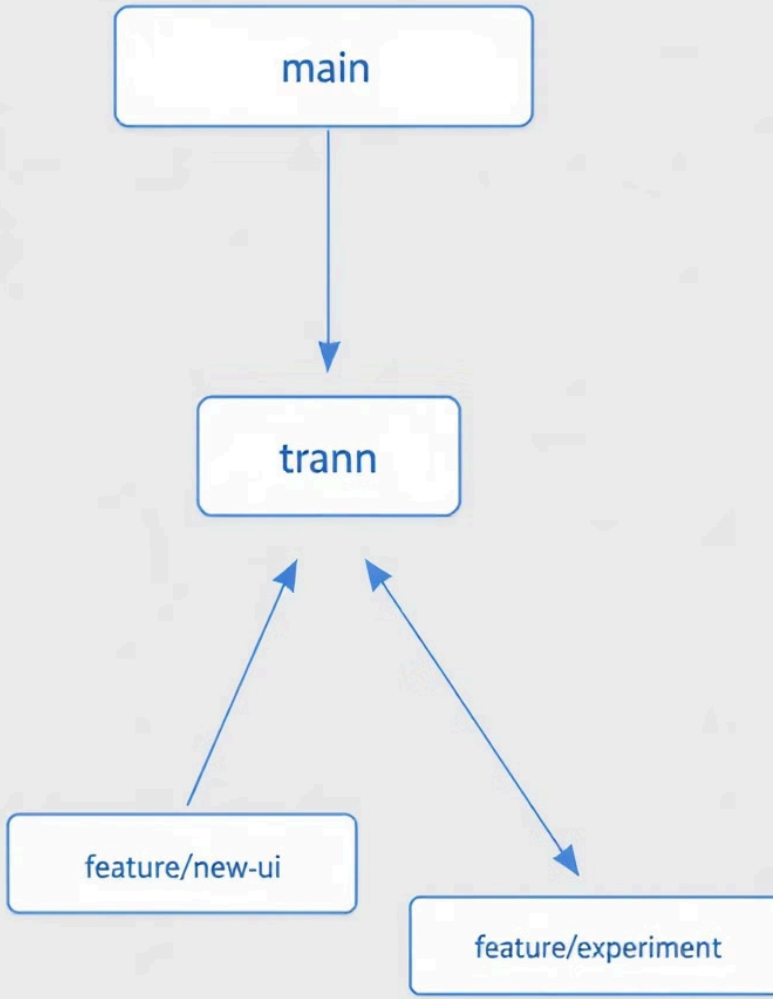
Переключение между ветками

git merge

Слияние изменений из разных веток

git mergetool

Решение конфликтов при слиянии веток



Интеграция с Docker

Платформа для контейнеризации

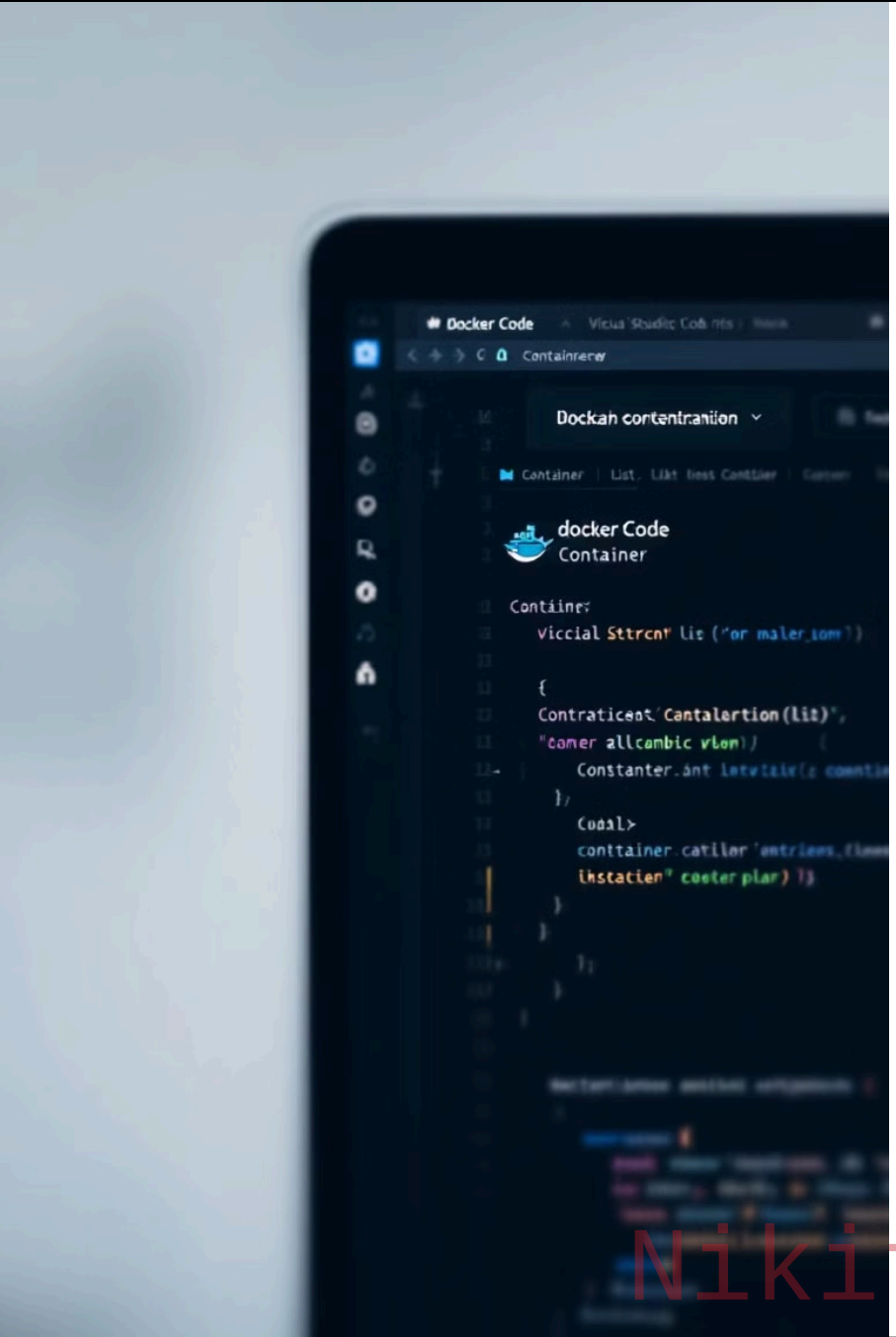
Docker упрощает создание, доставку и запуск приложений

Интеграция с VSC

Расширение Docker для VSC позволяет управлять контейнерами

Единая среда

Работа с Docker без переключения между окнами



Возможности Docker в VSC

Создание контейнеров

Создание новых Docker-контейнеров из VSC

Управление контейнерами

Запуск и остановка контейнеров через интерфейс VSC

Настройка параметров

Изменение параметров контейнеров

Доступ к контейнерам

Подключение к контейнерам через терминал VSC

Создание Docker-контейнера в VSC

Установите расширение Docker

Найдите и установите расширение Docker из маркетплейса VSC

Создайте Dockerfile

Через команду "Docker: Add Docker Files to Workspace"

Настройте Dockerfile

Отредактируйте файл под требования вашего приложения

Создайте docker-compose.yml

Для управления несколькими сервисами вашего приложения

Управление Docker-контейнерами

Запуск контейнеров

- Через команду "Docker: Compose Up"
- С помощью кнопки "Start" в панели Docker
- Командой docker run в терминале

```
docker run -it --name my_container ubuntu:latest
```

Остановка контейнеров

- Через команду "Docker: Compose Down"
- С помощью кнопки "Stop" в панели Docker
- Командой docker stop в терминале

```
docker stop my_container
```

Остановка контейнеров из VSC

Через панель команд

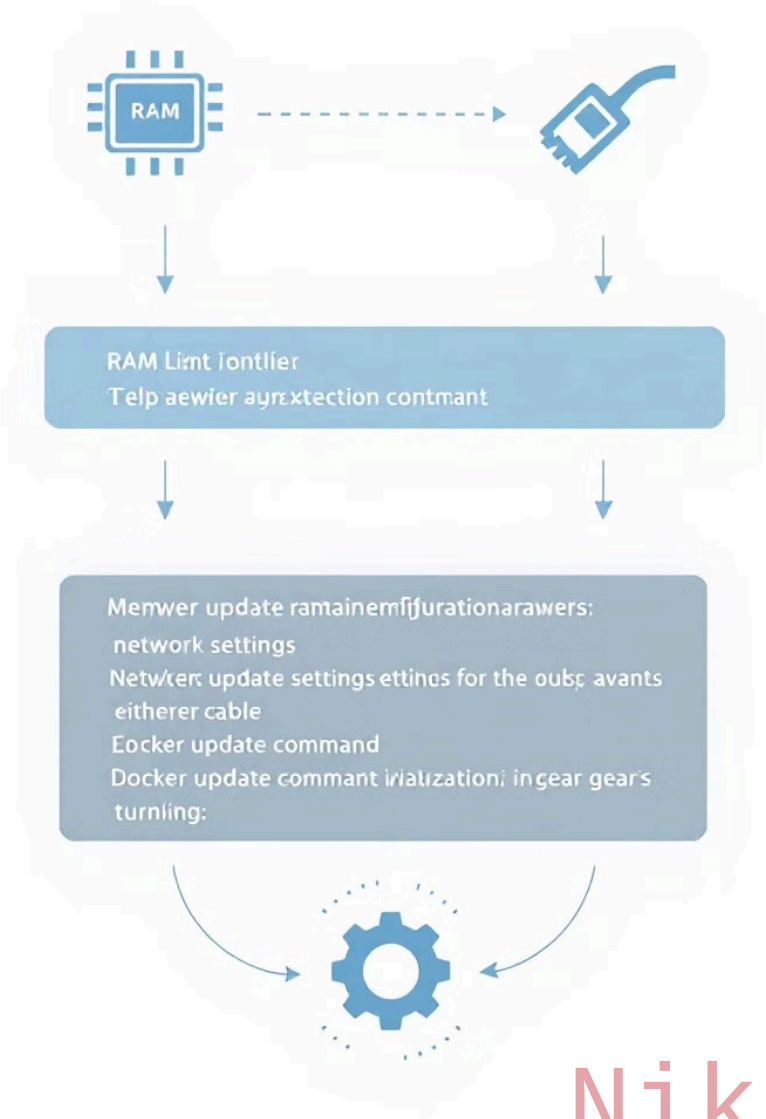
Нажмите Ctrl+Shift+P и введите "Docker: Compose Down"

Через панель Docker

Выберите контейнер в боковой панели и нажмите кнопку "Stop"

Через терминал

Выполните команду docker stop my_container



Изменение параметров контейнеров

Изменение доступной памяти

Настройка лимитов оперативной памяти для контейнера

Изменение монтирования томов

Настройка точек монтирования для хранения данных

Изменение сетевых настроек

Настройка сетевых интерфейсов и портов

Настройка памяти контейнера

Через команду run

```
docker run -m 2g myimage
docker run -m 2g --memory-swap 1g myimage
```

Флаг -m задает объем памяти, а --memory-swap - размер файла подкачки

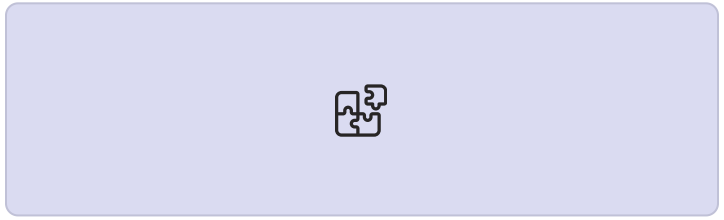
Через docker-compose.yml

```
version: '3'
services:
  myservice:
    image: myimage
    resources:
      limits:
        memory: 2g
        memory-swap: 1g
```

Параметры памяти в docker-compose

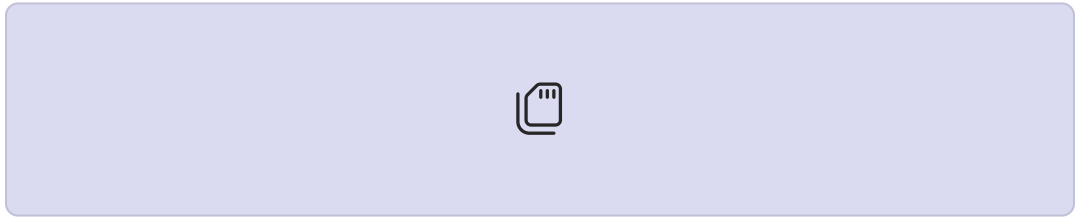
Создайте docker-compose.yml

Основной файл конфигурации для Docker Compose



Определите сервисы

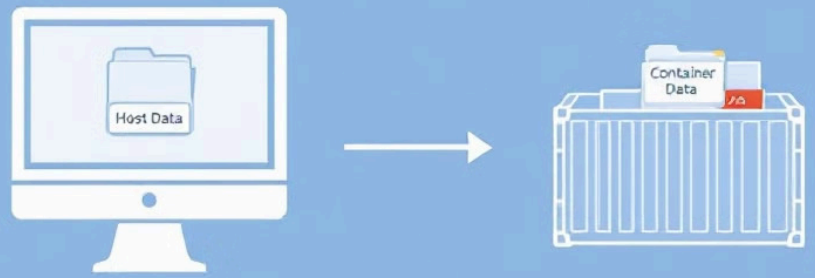
Опишите компоненты вашего приложения



Укажите лимиты ресурсов

Настройте ограничения памяти в разделе resources

Docker Volume Mounting



Монтирование томов Docker

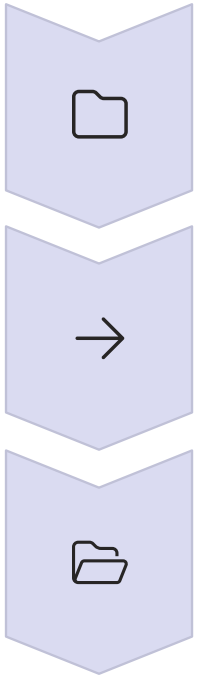
Опции монтирования

- -v : - монтирование с хоста
- -v : - монтирование Docker Volume
- -v :: - с дополнительными параметрами

Преимущества томов

- Сохранение данных между перезапусками
- Возможность резервного копирования
- Легкое масштабирование
- Безопасное хранение конфиденциальных данных

Пример монтирования тома



Локальная директория

/home/user/app

Команда Docker

docker run -v /home/user/app:/app my_image

Директория в контейнере

/app

Настройка сети Docker

Опции сетевых настроек

- --network - выбор сети
- -p : - проброс портов
- --hostname - имя хоста внутри контейнера
- --dns - настройка DNS-серверов

Типы сетей

- bridge - стандартная сеть (по умолчанию)
- host - использование сети хоста
- none - без сетевого доступа
- overlay - для мульти-хост коммуникаций

Пример настройки сети

Создание сети

docker network create my_network

Запуск контейнера в сети

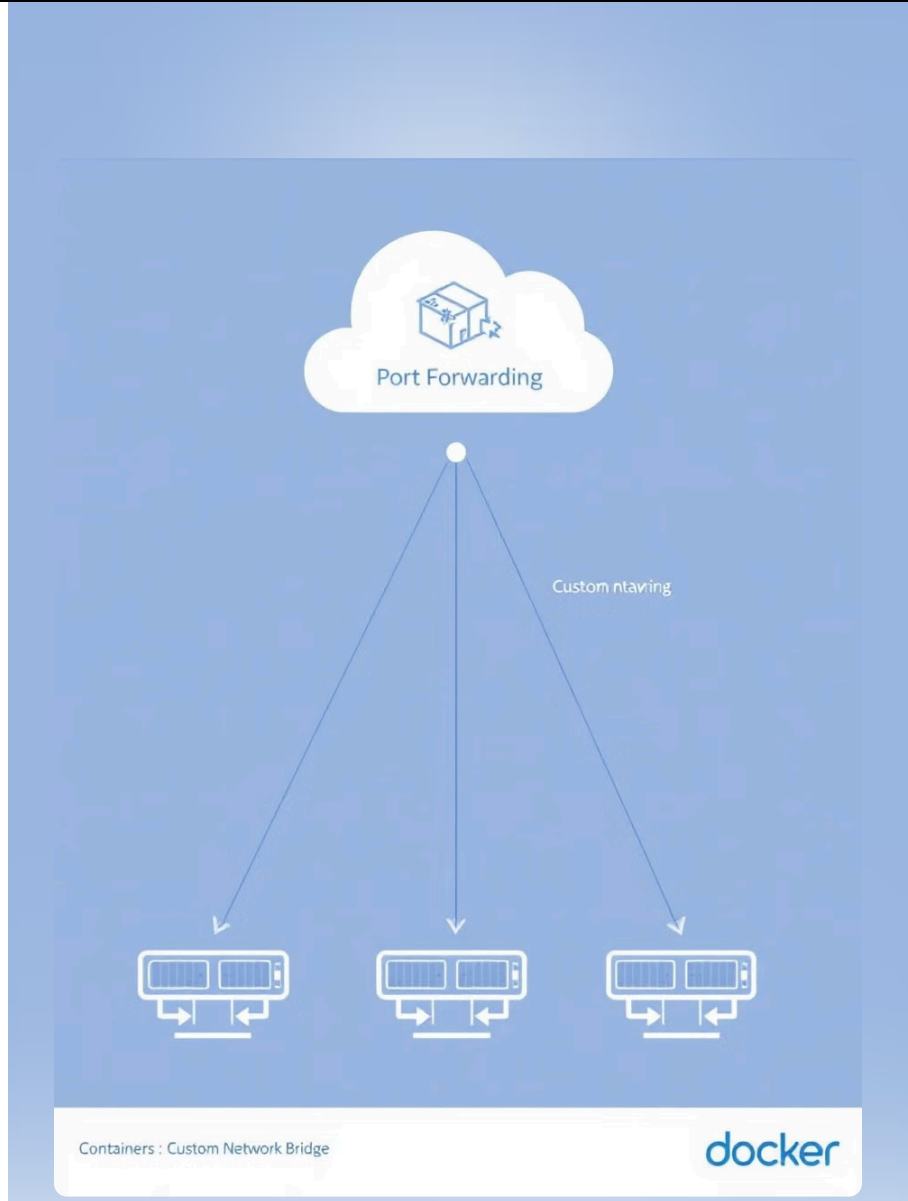
docker run --network my_network -p 8080:80 my_image

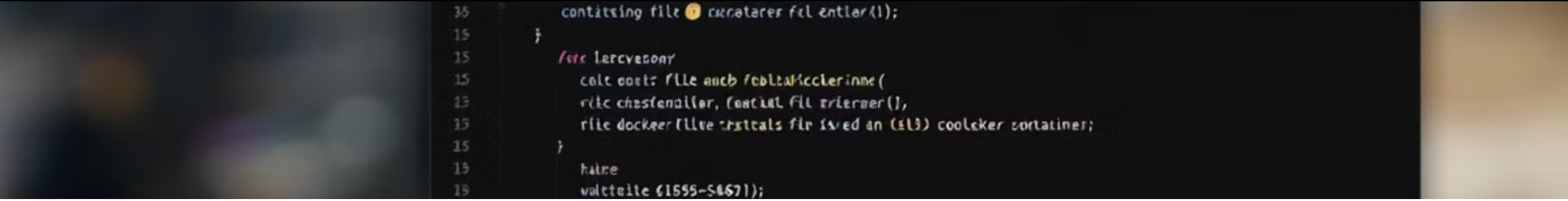
Проверка подключения

Теперь приложение доступно по адресу http://localhost:8080

Проверка сети

docker network inspect my_network





Подключение к контейнеру из VSC

1

Установите расширение Docker

Добавьте расширение Docker из маркетплейса VSC

Запустите контейнер

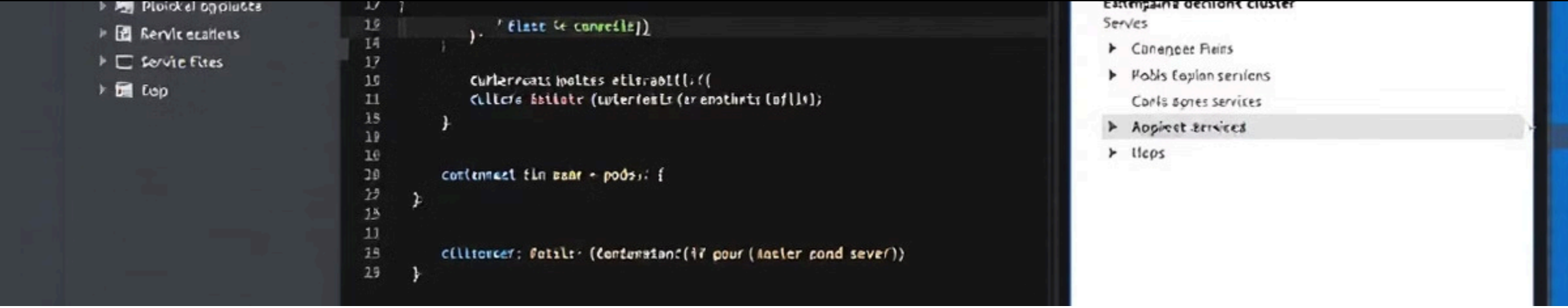
Используя команду docker run или docker-compose up

Откройте панель Docker

Найдите запущенный контейнер в списке

Подключитесь к контейнеру

Нажмите "Attach Shell" рядом с нужным контейнером



Интеграция с Kubernetes

Создание кластера

Настройка и запуск нового кластера Kubernetes

Деплоймент приложений

Развертывание контейнеров на кластере

Масштабирование

Увеличение или уменьшение количества экземпляров приложения

Мониторинг

Отслеживание состояния приложений в реальном времени

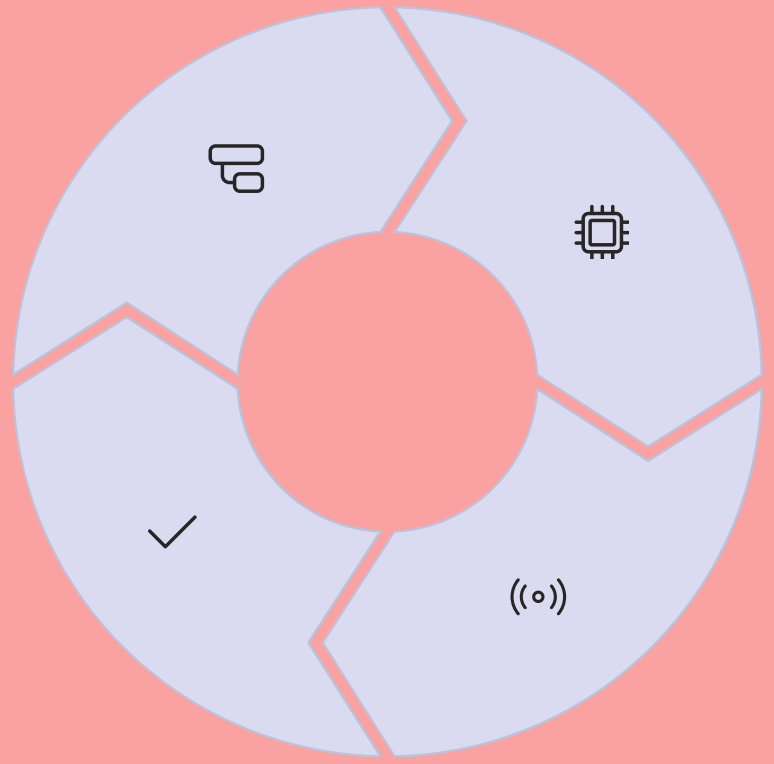
Параллельное исполнение сборки

Разделение задач

Сборка разбивается на независимые подзадачи

Распределение по ядрам

Задачи распределяются между доступными процессорами



Объединение результатов

Сборка финального продукта из параллельно созданных частей

Параллельное выполнение

Одновременная компиляция нескольких файлов

Что такое параллельная сборка?

Определение

Процесс сборки ПО, при котором задачи выполняются одновременно на нескольких ядрах процессора

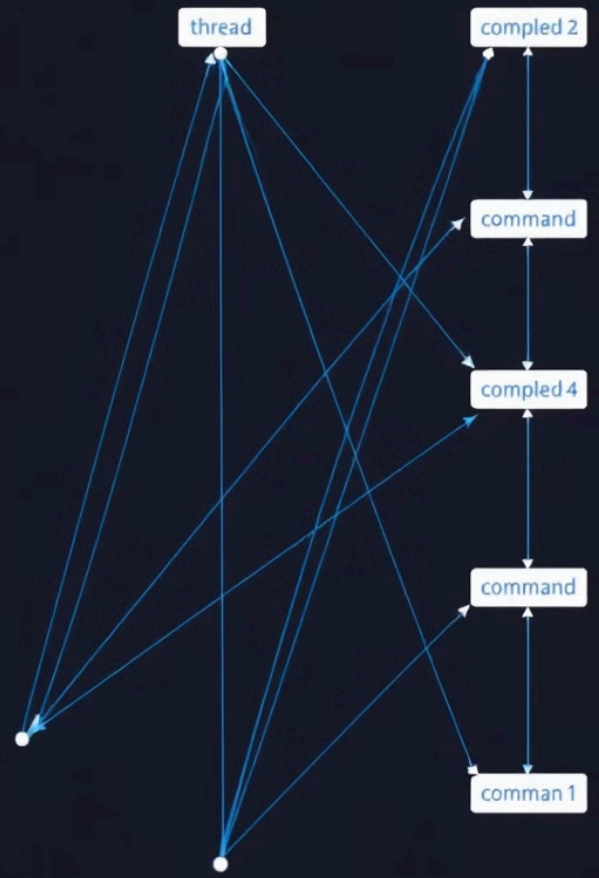
Задачи сборки делятся на независимые потоки для параллельного выполнения

Преимущества

- Значительное ускорение процесса сборки
- Лучшее использование ресурсов компьютера
- Снижение времени ожидания для разработчиков
- Быстрая обратная связь о качестве кода

Инструменты параллельной сборки

Современные инструменты сборки поддерживают параллельное выполнение задач для ускорения процесса. Каждый имеет свои особенности настройки и оптимизации многопоточной работы.



Параллельная сборка в Make

Базовый синтаксис

Используйте параметр -j с указанием числа потоков

```
all:
    make -j8 build
```

Оптимальное количество потоков

Обычно равно количеству ядер процессора + 1

Можно использовать автоопределение: $(nproc)

```
all:
    make -j$(nproc) build
```

Мониторинг процесса

Добавьте флаг -l для ограничения нагрузки

```
make -j8 -l5.5 build
```

Параллельная сборка в CMake

```
set(CMAKE_BUILD_TYPE Release)
set(CMAKE_CXX_FLAGS "-Wall")
set(CMAKE_CXX_FLAGS_RELEASE "-O3")
add_executable(myapp main.cpp)
add_custom_target(build
    COMMAND ${CMAKE_COMMAND} --build ${CMAKE_BINARY_DIR} --target myapp -- -j8
    DEPENDS myapp)
```

В этом примере CMake настроен на использование 8 потоков для сборки проекта. Команда add_custom_target создает цель build, которая запускает многопоточную сборку приложения myapp.

Проверка усвоения материала

1 **Что такое CI/CD пайплайн?**

Дайте определение и перечислите основные этапы.

2 **Как настроить оповещения в Slack?**

Опишите последовательность действий и необходимые компоненты.

3 **Что такое Docker Compose?**

Для чего используется и как настраивается?

4 **Какие типы интеграции доступны в VSC?**

Перечислите и кратко опишите каждый.

5 **Как изменить параметры Docker-контейнера?**

Опишите способы изменения памяти, томов и сетевых настроек.

6 **Что такое параллельное исполнение сборки?**

Почему оно важно и какие инструменты его поддерживают?

7 **Как запустить параллельную сборку в Make?**

Приведите пример команды с оптимальными параметрами.

8 **Какие расширения VSC полезны для DevOps?**

Назовите минимум три и опишите их функциональность.

9 **Чем отличается тестовое окружение от продакшн?**

Перечислите ключевые различия и способы настройки.

10 **Как настроить мониторинг в CI/CD пайплайне?**

Опишите инструменты и базовую конфигурацию.