Формат Практика 3 — CI/DevSecOps: конвейер безопасной доставки ПО

Цель: автоматизировать сборку, тесты и проверки безопасности (Shift-Left) до слияния в основную ветку. Основной стек: Git, Docker, GitHub Actions/GitLab CI, Gitleaks (Secret Scanning), Semgrep (SAST), Trivy (SCA/Container Scan), SBOM (CycloneDX).

Дорожная карта и оценка времени

1. Базовый CI pipeline: сборка Docker-образа и запуск тестов внутри контейнера.
2. Интеграция quality gates: перенос pre-commit проверок в CI.
3. DevSecOps:
    - Secret Scanning (Gitleaks)
    - SAST (Semgrep, профиль OWASP Top 10)
    - SCA/Container Scan (Trivy, политика fail on HIGH/CRITICAL)
4. Отчетность: публикация SARIF/JSON и SBOM как артефактов; интеграция Code Scanning (GH) при наличии.
5. Защита ветки: Required status checks и запрет merge при провале CI.
6. Демонстрация сценариев падения и исправления.
7. Документирование: ADR-0002 «Стратегия CI и инструменты DevSecOps».

Оценка времени: базовый поток 3–4 часа; бонусы +60 минут.

Цели и результаты

- Освоите настройку CI под reproducible build и изолированные тесты в контейнере.
- Внедрите quality gates через pre-commit в CI.
- Настроите DevSecOps проверки: Gitleaks, Semgrep, Trivy (FS/Image).
- Сгенерируете и опубликуете артефакты: SARIF, логи, SBOM.
- Защитите ветку main правилами и required checks.
- Оформите ADR-0002 с обоснованием выбора инструментов и политик.

Артефакты:

- CI конфиг: .github/workflows/ci.yml или .gitlab-ci.yml
- Отчёты: gitleaks.sarif/json, semgrep.sarif, trivy-fs.sarif, trivy-image.sarif, sbom.cdx.json
- REPORT.md, docs/adr/ADR-0002.md

Пререквизиты

- Репозиторий из Практик 1–2 с Dockerfile и базовыми тестами (например, pytest).
- pre-commit настроен локально; есть конфиг .pre-commit-config.yaml.
- Токены/секреты в CI не требуются для базовой конфигурации.

Подсказка: если у вас docker compose, сохраните также вариант команд compose; в шаблоне используем docker build/run для универсальности.

Раздел 1. Базовый CI pipeline (Build + Test в контейнере)

Вариант A — GitHub Actions (.github/workflows/ci.yml)

```
name: CI

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

permissions:
  contents: read
  security-events: write  # для загрузки SARIF

concurrency:
  group: ci-${{ github.ref }}
  cancel-in-progress: true

jobs:
  build-test:
    name: Build and Test (Docker)
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Set up Buildx
        uses: docker/setup-buildx-action@v3

      - name: Build image
        uses: docker/build-push-action@v5
        with:
          context: .
          file: ./Dockerfile
          load: true          # загрузить образ локально для последующего
запуска
          tags: app:ci

      - name: Run tests in container
        run: |
          docker run --rm -e PYTHONUNBUFFERED=1 app:ci pytest -q
```

Вариант B — GitLab CI (.gitlab-ci.yml)

```
stages: [prechecks, build, test, security, report]

variables:
  DOCKER_TLS_CERTDIR: ""
```

```
docker-build:
  stage: build
  image: docker:27
  services:
    - docker:27-dind
  script:
    - docker build -t $CI_REGISTRY_IMAGE:ci .
  artifacts:
    expire_in: 1 week

pytest:
  stage: test
  image: docker:27
  services:
    - docker:27-dind
  needs: ["docker-build"]
  script:
    - docker run --rm $CI_REGISTRY_IMAGE:ci pytest -q
```

Раздел 2. Интеграция quality gates (pre-commit в CI)

GitHub Actions (добавьте job)

```
  precommit:
    name: pre-commit checks
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: '3.12'
      - name: Install pre-commit
        run: |
          python -m pip install --upgrade pip
          pip install pre-commit
      - name: Run pre-commit
        run: |
          pre-commit run -a --show-diff-on-failure
```

GitLab CI

```
precommit:
  stage: prechecks
  image: python:3.12-slim
  script:
    - python -m pip install --upgrade pip
    - pip install pre-commit
    - pre-commit run -a --show-diff-on-failure
```

Важно: пайплайн должен падать, если форматирование/линт не соответствует стандартам.

Раздел 3. DevSecOps проверки

3.1 Secret Scanning — Gitleaks

GitHub Actions

```
  gitleaks:
    name: Gitleaks (secrets)
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Install gitleaks
        run: |
          curl -sSfL
https://raw.githubusercontent.com/gitleaks/gitleaks/master/install.sh |
bash -s v8.18.4
      - name: Run gitleaks
        run: |
          ./gitleaks detect --redact --report-format sarif --report-path
gitleaks.sarif
      - name: Upload artifact
        uses: actions/upload-artifact@v4
        with:
          name: gitleaks-sarif
          path: gitleaks.sarif
      - name: Upload SARIF to Code Scanning
        uses: github/codeql-action/upload-sarif@v3
        with:
          sarif_file: gitleaks.sarif
```

GitLab CI

```
gitleaks:
  stage: security
  image: zricethezav/gitleaks:latest
  script:
    - gitleaks detect --redact --report-format sarif --report-path
gitleaks.sarif
  artifacts:
    when: always
    paths: [gitleaks.sarif]
    expire_in: 1 week
```

3.2 SAST — Semgrep (OWASP Top 10)

GitHub Actions

```
  semgrep:
    name: Semgrep (SAST)
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: '3.12'
      - name: Install semgrep
        run: |
          pip install semgrep==1.*
      - name: Run semgrep
        run: |
          semgrep ci --config p/owasp-top-ten --sarif --output
semgrep.sarif
      - name: Upload artifact
        uses: actions/upload-artifact@v4
        with:
          name: semgrep-sarif
          path: semgrep.sarif
      - name: Upload SARIF to Code Scanning
        uses: github/codeql-action/upload-sarif@v3
        with:
          sarif_file: semgrep.sarif
```

GitLab CI

```
semgrep:
  stage: security
  image: returntocorp/semgrep:latest
  script:
    - semgrep ci --config p/owasp-top-ten --sarif --output semgrep.sarif
  artifacts:
    when: always
    paths: [semgrep.sarif]
    expire_in: 1 week
```

3.3 SCA/Container Scan — Trivy

Политика: провал CI при HIGH/CRITICAL уязвимостях.

GitHub Actions (FS + Image)

```
  trivy-fs:
    name: Trivy FS (deps)
```

```
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Trivy FS scan
        uses: aquasecurity/trivy-action@0.20.0
        with:
          scan-type: 'fs'
          format: 'sarif'
          output: 'trivy-fs.sarif'
          ignore-unfixed: true
          severity: 'HIGH,CRITICAL'
          exit-code: '1'
      - uses: actions/upload-artifact@v4
        with:
          name: trivy-fs-sarif
          path: trivy-fs.sarif
      - uses: github/codeql-action/upload-sarif@v3
        with:
          sarif_file: trivy-fs.sarif

  trivy-image:
    name: Trivy Image (container)
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: docker/setup-buildx-action@v3
      - uses: docker/build-push-action@v5
        with:
          context: .
          file: ./Dockerfile
          load: true
          tags: app:ci
      - name: Trivy Image scan
        uses: aquasecurity/trivy-action@0.20.0
        with:
          image-ref: 'app:ci'
          format: 'sarif'
          output: 'trivy-image.sarif'
          ignore-unfixed: true
          severity: 'HIGH,CRITICAL'
          exit-code: '1'
      - uses: actions/upload-artifact@v4
        with:
          name: trivy-image-sarif
          path: trivy-image.sarif
      - uses: github/codeql-action/upload-sarif@v3
        with:
          sarif_file: trivy-image.sarif
```

GitLab CI (FS + Image)

```
trivy-fs:
  stage: security
  image: aquasecurity/trivy:latest
  script:
    - trivy fs --exit-code 1 --format sarif --output trivy-fs.sarif --
ignore-unfixed --severity HIGH,CRITICAL .
  artifacts:
    when: always
    paths: [trivy-fs.sarif]
    expire_in: 1 week

trivy-image:
  stage: security
  image: docker:27
  services: [docker:27-dind]
  script:
    - docker build -t app:ci .
    - apk add --no-cache curl
    - curl -sSfL
https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/install.
sh | sh -s -- -b /usr/local/bin
    - trivy image --exit-code 1 --format sarif --output trivy-image.sarif
--ignore-unfixed --severity HIGH,CRITICAL app:ci
  artifacts:
    when: always
    paths: [trivy-image.sarif]
    expire_in: 1 week
```

Раздел 4. Отчётность и артефакты (SARIF, SBOM)

- Публикация артефактов: используйте actions/upload-artifact (GH) или artifacts (GL).
- Загрузка SARIF в Code Scanning (GH): github/codeql-action/upload-sarif.
- SBOM (CycloneDX):

SBOM — варианты

```
# Вариант 1: Trivy CLI (FS)
trivy sbom --format cyclonedx --output sbom.cdx.json .

# Вариант 2: Syft по образу (рекомендуется после сборки)
syft app:ci -o cyclonedx-json=sbom.cdx.json
```

Совет: SBOM всегда прикладывайте как артефакт; по возможности проверьте его в стороннем валидаторе.

Примеры job для SBOM

GitHub Actions

```
sbom:
  name: SBOM (CycloneDX)
  runs-on: ubuntu-latest
  steps:
    - uses: actions/checkout@v4
    - name: Install Trivy
      run: |
        curl -sfL
https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/inst
all.sh | sh -s -- -b /usr/local/bin
    - name: Generate SBOM (FS)
      run: trivy sbom --format cyclonedx --output sbom.cdx.json .
    - name: Upload SBOM
      uses: actions/upload-artifact@v4
      with:
        name: sbom-cyclonedx
        path: sbom.cdx.json
```

GitLab CI

```
sbom:
  stage: report
  image: aquasecurity/trivy:latest
  script:
    - trivy sbom --format cyclonedx --output sbom.cdx.json .
  artifacts:
    when: always
    paths: [sbom.cdx.json]
    expire_in: 1 week
```

Раздел 5. Защита ветки и демонстрация

- Настройте Branch Protection Rules (GitHub) или Protected Branches (GitLab):
  - Запрет прямых пушей в main.
  - Требовать успешный проход всех CI-проверок.
  - Требовать PR/MR и минимум 1 review (по политике команды).
- Демонстрационные сценарии:
  1. Добавьте уязвимую зависимость → Trivy FS/Image должен «краснить» пайплайн.
  2. Захардкодьте секрет → Gitleaks/ Semgrep должен «краснить» пайплайн.
  3. Исправьте и добейтесь «зелёного» статуса и merge.

Задания (Tasks)

1. Добавить в репозиторий CI конфигурацию (GH Actions или GitLab CI) с job’ами Build/Test, pre-commit, Gitleaks, Semgrep, Trivy (FS и Image), публикация артефактов.

2. Настроить политику fail-on для Trivy: HIGH,CRITICAL.
3. Сгенерировать SBOM (CycloneDX) и приложить как артефакт.
4. Включить Code Scanning (GH) и загрузку SARIF (при использовании GitHub).
5. Настроить защиту ветки main и required status checks.
6. Подготовить REPORT.md с логами/скринами, ссылками на CI и анализом результатов.
7. Оформить ADR-0002 «Стратегия CI и инструменты DevSecOps».

Формат сдачи

- Ссылка на репозиторий (доступ на чтение преподавателю/жюри).
- Ветка: feature/practice3 → PR/MR → merge в main после «зелёного» CI.
- Обязательные файлы/директории:
  - .github/workflows/ci.yml или .gitlab-ci.yml
  - REPORT.md
  - docs/adr/ADR-0002.md
- Артефакты/доказательства:
  - Ссылки на успешные/провальные прогонки CI с логами.
  - Скачанные отчёты (SARIF/JSON) и SBOM.
  - Скриншоты настроек Branch Protection/Required checks.

Критерии оценивания (Rubric)

+3 Репозиторий собирается, тесты стабильно проходят в контейнере (лог CI).

+3 Полный набор DevSecOps проверок: Gitleaks, Semgrep, Trivy (FS, Image), политики применены (HIGH/CRITICAL → fail).

+2 Отчётность: SARIF/JSON загружены/опубликованы, SBOM сформирован и приложен.

+2 Защита ветки включена, Required checks настроены (скрин/лог).

+2 ADR-0002 оформлен (контекст, решение, последствия +/–, критерии пересмотра, security considerations).

+1 (Бонус) Интеграция GitHub Code Scanning или GitLab Security Dashboard.

+1 (Бонус) Автогенерация SBOM по образу (Syft) и его проверка внешним валидатором.

–2..–4 За нерепродуцируемость/отсутствие логов/нечитаемые скриншоты.

–2 Секреты в репозитории, отсутствие сканирования секретов.

Troubleshooting

- Docker Buildx/кеш: обновите docker/setup-buildx-action; при flake отключите кэширование временно.
- Таймауты/Rate limits Trivy: используйте локальный DB кэш (по умолчанию), повторный запуск ускорится.
- False positives Semgrep/Gitleaks: документируйте исключения (baseline/ignore), не отключайте правила без ADR.

- GitLab DIND права: `DOCKER_TLS_CERTDIR=""`, совместимые версии docker:service.
- SARIF не виден в GH: проверьте `security-events: write` и путь `sarif_file`.
- pre-commit не находит хуки: убедитесь в наличии .pre-commit-config.yaml в корне.

Дополнительные материалы и чек-листы

Шаблон REPORT.md

```
# REPORT — Практика 3: CI/DevSecOps

## Репозиторий и ветка
- Repo: <url>
- PR/MR: <url>

## CI и результаты
- Логи успешной прогонки: <url/скрин>
- Логи провальных сценариев: <url/скрин>

## Артефакты
- Gitleaks: gitleaks.sarif
- Semgrep: semgrep.sarif
- Trivy FS: trivy-fs.sarif
- Trivy Image: trivy-image.sarif
- SBOM: sbom.cdx.json

## Выводы
- Ключевые находки и их устранение
- План улучшений
```

Шаблон ADR-0002

```
# ADR-0002: Стратегия CI и инструменты DevSecOps

- Дата: YYYY-MM-DD
- Статус: Proposed | Accepted | Superseded

## Контекст
Кратко: цель CI, требования безопасности, ограничения инфраструктуры.

## Решение
CI-провайдер (GH/GL), этапы пайплайна, инструменты (Gitleaks, Semgrep,
Trivy), политики (fail on HIGH/CRITICAL), публикация артефактов.

## Последствия
Плюсы/Минусы, затраты (TCO), обслуживание, риски (false positives),
критерии пересмотра.

## Security Considerations
Credentials, секреты, доступы, supply chain, SBOM.
```

Self-check (перед сдачей)

- CI успешно собирает образ и запускает тесты в контейнере.
- Pre-commit проверки проходят локально и в CI.
- Gitleaks/Semgrep/Trivy выполняются; политики применяются.
- Артефакты (SARIF/SBOM) доступны для скачивания.
- Защита ветки и required checks включены.
- REPORT.md и ADR-0002 оформлены и добавлены в репозиторий.

Чек-лист демонстрации

- Показать провальный PR/MR из-за уязвимой зависимости/секрета.
- Показать «зелёный» PR/MR после исправлений.
- Показать артефакты (SARIF/SBOM) и настройки защиты ветки.

README-вставка (бейдж статуса CI для GitHub)

```
![CI](https://github.com/<org>/<repo>/actions/workflows/ci.yml/badge.svg)
```

Замените / на ваш namespace. Для GitLab используйте встроенные статус-бейджи проекта.