DevSecOps Interview Prep

Базовая демо-книга для подготовки

к интервью и hands-on заданиям

Практический фокус: CI/CD, supply chain, secrets, контейнеры, Kubernetes, IaC, облако, IAM.

Собрано на основе инженерных практик, публичных материалов, карьерных порталов и типовых assessment-паттернов.

Подходит как для структурирования знаний, так и для подготовки к техническому интервью и hands-on заданиям.

white2hack (c)

Релиз: 1 марта 2026

Версия: v1.0

Базовая демо-версия для учебных и ознакомительных целей

Вводные сведения и дисклеймер

DevSecOps Interview Prep — Базовая демо-книга для подготовки к интервью и hands-on заданиям. Автор: white2hack (c). Дата релиза: 5 марта 2026. Версия: v1.0.

Настоящий материал является обзорным и учебным. Он не является индивидуальной консультацией, не образует договорных отношений, не гарантирует прохождение интервью, получение оффера, соответствие внутренним стандартам конкретной компании и не заменяет инженерную проверку применимости решений в вашей среде.

Автор не несёт ответственности за прямой, косвенный, случайный или иной ущерб, возникший вследствие использования или неверного толкования представленных подходов, конфигураций, команд, политик, примеров кода, архитектурных схем и рекомендаций. Любое применение примеров должно сопровождаться собственным тестированием, threat modeling, review прав доступа и проверкой совместимости со средой.

Важно: Английский язык остаётся основным языком индустрии и документации. Русскоязычные переводы терминов в книге даны как вспомогательный слой для более точного понимания и устного объяснения темы на интервью.

Предисловие

Эта книга собрана как компактный, но содержательный материал по DevSecOps на русском языке. Её задача — не перегрузить читателя академической теорией, а дать рабочий язык для интервью и опорные практические схемы: как строится secure CI/CD, как работают supply chain controls, где именно нужны политики, какие компромиссы принимаются и как отвечать зрелым инженерным языком.

Структура намеренно построена «от модели к практике»: сначала фиксируем принципы и зрелость, затем разбираем pipeline, секреты, контейнеры, Kubernetes, IaC, облако и IAM, а после переходим к показательным кейсам в духе hands-on assessment. В конце вынесены глоссарий, корректные русские переводы ключевых терминов и короткий список важнейших книг по теме.

Как пользоваться книгой: Сначала прочитайте главы 1-2, чтобы выстроить язык ответа и общую рамку мышления. Затем отработайте главы 3-6 вместе с примерами кода: проговаривайте, что и зачем делает каждый контроль. Финально разберите три кейса и попробуйте ответить на них устно в формате design walkthrough.

Содержание

Глава 1. Что такое DevSecOps на практике

DevSecOps как операционная модель

DevSecOps — это не набор сканеров, а способ встроить инженерные контроли безопасности в ежедневный поток поставки изменений. Его цель — не просто «найти уязвимость», а сделать так, чтобы безопасный путь был самым простым, быстрым и воспроизводимым.

Сильный ответ на интервью обычно показывает системное мышление: кто владеет риском, где именно вставляются контроли, что считается сигналом, какой SLA/ SLO у пайплайна, как минимизируется обход политик и что происходит, если средство безопасности упало.

Ключевой сдвиг в DevSecOps — переход от ручных исключений и heroic effort к детерминированным guardrails. Всё, что можно формализовать, должно жить в коде, политике, шаблоне репозитория или reusable pipeline component.

Четыре уровня зрелости

1) Точечные сканеры без ownership: отчёты есть, результата нет.

2) Интеграция в CI: сканирование связано с pull request и quality gates.

3) Security-as-code: политики, подписи артефактов, federation identity, unit-тесты для guardrails.

4) Platform model: разработчик получает безопасную paved road по умолчанию.

Зрелость определяется не количеством тулов, а тем, как быстро команда может выпускать изменения без деградации риска. Хороший сигнал — низкая доля ручных исключений, быстрый remediation loop и прозрачная телеметрия по ложноположительным срабатываниям.

Что проверяют на интервью

Обычно DevSecOps-собеседование проверяет сразу несколько слоёв: SDLC, CI/CD, secrets, контейнеры, Kubernetes, облако, IaC, IAM, supply chain, monitoring, incident response и способность объяснять trade-off между безопасностью, скоростью и надёжностью.

Важно уметь мыслить не только «каким сканером найти проблему», но и «каким механизмом сделать повторное появление класса ошибок менее вероятным». Это и отличает senior/lead подход от purely tool-based ответа.

Короткая ментальная карта DevSecOps

Разработка -> Pull Request -> CI Build -> Security Gates -> Artifact Registry

-> Deploy Staging -> Runtime Telemetry -> Controlled Release -> Prod

Безопасность не отдельный этап в конце, а слой контролей на каждом переходе.

Что проговорить вслух на интервью: Где именно находится контроль: pre-commit, PR, build, deploy или runtime. Что именно блокируется жёстко, а что идёт в advisory mode. Какой риск снимает контроль и что происходит при его обходе или отказе.

Глава 2. SDLC, supply chain и secure-by-default delivery

Где ставить контроли

DevSecOps-контроли хорошо работают, когда распределены по этапам: pre-commit, pull request, build, package, deploy и runtime. На ранних этапах дешевле всего ловить policy violations, секреты и грубые ошибки IaC. На build-этапе — проверять зависимости, provenance и повторяемость сборки. На deploy и runtime — валидировать, что в среду попал именно доверенный артефакт и что поведение системы соответствует ожиданиям.

Практический принцип: не пытайтесь одним монолитным quality gate решать всё сразу. Разделяйте fast feedback и deeper analysis. То, что блокирует разработчика на каждом коммите, должно быть быстрым, понятным и с минимальным уровнем шума.

Минимальный pipeline без

яопасности

Для типичного web-сервиса минимальный зрелый контур выглядит так: secret scanning, dependency scanning/SCA, SAST для критичных правил, build reproducibility, container scan, SBOM generation, image signing, provenance/attestation, environment-specific deploy policy, runtime logging и оповещение на drift или policy bypass.

Полезно разделять обязательные контроли и advisory-контроли. Например, hard fail для явных секретов, критичных CVE в exploitable path, неподписанных образов и forbidden Kubernetes fields; soft fail или warn-only для stylistic замечаний, если они не несут реального риска.

Как не сломать velocity

Скорость разработки ломается не из-за безопасности как таковой, а из-за плохой инженерной упаковки безопасности. Основные причины: ложные срабатывания, отсутствие baseline, отсутствие ownership, нерепродуцируемые правила и блокировки без remediation guidance.

Хороший DevSecOps-инженер проектирует controls так, чтобы у разработчика была понятная причина блокировки, ссылка на playbook и предсказуемый путь исправления. Там, где возможно, используйте automated fixes, secure templates и pull-request comments с контекстом, а не просто «pipeline failed».

Пример компактного GitHub Actions pipeline с security gates

```yaml
name: secure-ci
on:
  pull_request:
  push:
    branches: [ main ]

permissions:
  contents: read
  id-token: write
  security-events: write

jobs:
  scan-build-sign:
    runs-on: ubuntu-latest
    steps:
```

```
- uses: actions/checkout@v4
- name: Secret scan
  run: gitleaks detect --source . --no-banner
- name: Dependency scan
  run: trivy fs --exit-code 1 --severity HIGH,CRITICAL .
- name: Build image
  run: docker build -t app:${{ github.sha }} .
- name: Generate SBOM
  run: syft packages dir:. -o spdx-json > sbom.json
- name: Scan image
  run: trivy image --exit-code 1 app:${{ github.sha }}
- name: Sign image
  run: cosign sign --yes registry.example/app:${{ github.sha }}
```

Чек-лист быстрого triage при инциденте в pipeline

```
# 1. Остановить новые релизы
# 2. Отозвать токены/секреты
# 3. Понять, какие артефакты успели выйти
# 4. Проверить provenance и журнал запуска job
# 5. Изолировать runner и собрать форензику
# 6. Ретроспективно проверить все подозрительные deploy
```

Что проговорить вслух на интервью: Где именно находится контроль: pre-commit, PR, build, deploy или runtime. Что именно блокируется жёстко, а что идёт в advisory mode. Какой риск снимает контроль и что происходит при его обходе или отказе.

Глава 3. CI/CD hardening, секреты и доверенная сборка

Безопасность runner/agent слоя

Runner часто является самым недооценённым активом в DevSecOps. Если он shared, долгоживущий, привилегированный и имеет доступ одновременно к исходному коду, секретам и продовым артефактам, то он превращается в идеальную точку для supply chain compromise.

Базовые меры: ephemeral runners, жёсткая сегментация между trust zones, запрет на привилегированные контейнеры без исключительного бизнес-обоснования, минимальные permissions у pipeline token, отдельные execution pools для untrusted и trusted workloads, обязательный аудит использования self-hosted runners.

Секреты: от хранения к полной модели обращения

DevSecOps-зрелость определяется не тем, где лежат секреты, а тем, каков их жизненный цикл. Надо понимать, кто их выдаёт, как происходит rotation, как контролируется scope, как отслеживается использование, как секреты попадают в runtime и можно ли заменить их на federation identity без статических ключей.

Оптимальный путь — уменьшать число долгоживущих секретов. Там, где возможно, используйте OIDC federation между CI-системой и облаком, short-lived credentials, secret injection в runtime через vault/provider integration и обязательный secret scanning на уровне репозитория и PR.

Trusted build и provenance

Сильный ответ на интервью включает не только «мы сканируем image», но и «мы знаем, где и как он был собран». Именно provenance и attestation позволяют отделять доверенные артефакты от артефактов неизвестного происхождения.

Практически это означает: фиксированные builder images, version-pinned actions/plugins, контролируемые build inputs, изоляция сети при сборке там, где это реально, SBOM, подписи и проверка подписи на admission/deploy этапе.

GitHub OIDC к AWS без статического access key

```yaml
permissions:
  id-token: write
  contents: read

steps:
  - uses: actions/checkout@v4
  - name: Configure AWS credentials via OIDC
    uses: aws-actions/configure-aws-credentials@v4
    with:
      role-to-assume: arn:aws:iam::123456789012:role/gha-deploy-role
      aws-region: eu-central-1
  - name: Verify identity
    run: aws sts get-caller-identity
```

Пример жёстких правил для GitLab protected branch pipeline

```yaml
stages: [lint, scan, build, deploy]
variables:
  DOCKER_BUILDKIT: '1'
scan:
  stage: scan
  script:
    - gitleaks detect --source . --no-banner
    - trivy fs --exit-code 1 --severity HIGH,CRITICAL .
  only:
    - merge_requests
    - main
build:
```

```
stage: build
script:
 - docker build --pull --no-cache -t registry.local/app:$CI_COMMIT_SHA .
rules:
 - if: '$CI_COMMIT_BRANCH == "main"'
```

Что проговорить вслух на интервью: Где именно находится контроль: pre-commit, PR, build, deploy или runtime. Что именно блокируется жёстко, а что идёт в advisory mode. Какой риск снимает контроль и что происходит при его обходе или отказе.

Глава 4. Контейнеры, Kubernetes и runtime-защита

Контейнерная безопасность не заканчивается на image scan

Типичная ошибка кандидата — сводить container security к сканеру образов. На практике приходится контролировать базовые образы, содержимое слоёв, права процесса, capabilities, filesystem permissions, user namespace, источники пакетов, уровень привилегий в runtime и сетевую поверхность.

Сильная инженерная стратегия строится на слоях: curated base images, минимизация пакетов, non-root, read-only root filesystem там, где возможно, drop capabilities, регулярная пересборка образов, отдельные сканы filesystem и container image, policy checks на манифесты и runtime telemetry.

Kubernetes: guardrails, а не ручное ревью всего подряд

В Kubernetes важнее не найти один плохой манифест, а выстроить платформенные ограничители: admission policies, namespace segmentation, RBAC, network policies, pod security standards, secret management, image provenance enforcement и audit logging.

На интервью часто ценится подход warn -> audit -> enforce. Сначала команда видит нарушения и понимает impact, затем политики тестируются

в staging, и только после этого включается жёсткое применение в production.

Runtime и detection

Даже идеально просканированный образ может вести себя опасно после деплоя. Поэтому DevSecOps должен уметь отвечать на вопросы: какие системные вызовы считаются подозрительными, как фиксируются попытки запуска shell, обращения к kube-api, чтение сервисных токенов, попытки lateral movement и anomalous egress.

Практически это может реализовываться через Falco, eBPF-based detection, cloud-native runtime sensors и события admission/runtime в SIEM. Но главное — не конкретный бренд, а связка detection + ownership + response playbook.

Пример более безопасного Dockerfile

```
FROM python:3.12-slim

ENV PYTHONDONTWRITEBYTECODE=1   PYTHONUNBUFFERED=1

RUN addgroup --system app && adduser --system --ingroup app app

WORKDIR /app

COPY requirements.txt .

RUN pip install --no-cache-dir -r requirements.txt

COPY . .

USER app

EXPOSE 8080

CMD ["python", "app.py"]
```

Kyverno policy: запрет latest и обязательный non-root

```
apiVersion: kyverno.io/v1
kind: ClusterPolicy

metadata:
  name: require-secure-images

spec:

  validationFailureAction: Enforce
```

```yaml
rules:
  - name: disallow-latest-tag
    match:
      any:
        - resources:
            kinds: [Pod]
    validate:
      message: 'Использование тега latest запрещено.'
      pattern:
        spec:
          containers:
            - image: '!*:latest'
  - name: require-non-root
    match:
      any:
        - resources:
            kinds: [Pod]
    validate:
      message: 'Контейнер должен запускаться не от root.'
      pattern:
        spec:
          securityContext:
            runAsNonRoot: true
```

Что проговорить вслух на интервью: Где именно находится контроль: pre-commit, PR, build, deploy или runtime. Что именно блокируется жёстко, а что идёт в advisory mode.

Какой риск снимает контроль и что происходит при его обходе или отказе.

Глава 5. IaC, Policy as Code и контроль конфигурационного дрейфа

IaC как контрольная плоскость безопасности

Infrastructure as Code даёт не только скорость, но и шанс сделать инфраструктуру проверяемой, версионируемой и поддающейся policy enforcement. Однако IaC сам по себе не гарантирует безопасность: если модули не проходят ревью, state хранится небезопасно, а изменения идут в обход pull request, риски лишь принимают более «элегантную» форму.

На интервью стоит показывать понимание полного контура: модульный дизайн, pre-commit/lint/plan/apply workflow, scan policies, защищённый backend для state, контроль drift, правила change approval и разграничение ролей между platform, security и delivery-командами.

Какие проверки реально полезны

Базовый минимум: format/lint, validate, policy scanning, поиск жёстко заданных секретов, проверка небезопасных defaults, анализ сетевой экспозиции, encryption-at-rest, IAM blast radius, logging/retention и проверка защищённости storage buckets/state backends.

Полезно отделять «нарушение архитектурного стандарта» от «эксплуатируемого риска». Например, публичный S3 без явной бизнес-потребности — hard fail. Незначительное stylistic расхождение в теге ресурса — advisory.

PaC и drift management

Policy as Code работает лучше всего там, где есть чёткий контракт: какие поля, состояния и отношения между объектами считаются допустимыми. Хорошая политика сопровождается тестами, примерами допустимых/недопустимых конфигураций и понятным remediation guidance.

Drift detection критичен, потому что даже идеальный Terraform-план ничего не гарантирует, если ресурсы меняются вручную в консоли. Зрелая модель включает schedule-based drift checks, alerting, ограничение прямого доступа и, где возможно, auto-remediation.

Terraform: S3 backend для state с шифрованием и блокировками

```
terraform {

  backend "s3" {

    bucket         = "tf-state-prod"
    key            = "network/prod.tfstate"
    region         = "eu-central-1"
    encrypt        = true
    dynamodb_table = "tf-locks"
  }

}
```

OPA/Rego: запрет публичных security group

```
package terraform.security

deny[msg] {
  input.resource_type == "aws_security_group_rule"
  input.change.after.cidr_blocks[_] == "0.0.0.0/0"
  input.change.after.from_port == 22
  msg := "SSH из 0.0.0.0/0 запрещён"

}
```

Что проговорить вслух на интервью: Где именно находится контроль: pre-commit, PR, build, deploy или runtime. Что именно блокируется жёстко, а что идёт в advisory mode. Какой риск снимает контроль и что происходит при его обходе или отказе.

Глава 6. Облако, IAM и наблюдаемость для DevSecOps

IAM как граница blast radius

Большая часть серьёзных инцидентов в DevSecOps связана не с экзотическими zero-day, а с обычной комбинацией: избыточные права, слабый контроль секретов, отсутствие сегментации окружений и отсутствие ясности, кто именно выполняет действие в облаке. Поэтому IAM — один из центральных навыков DevSecOps-инженера.

Хорошая практика: раздельные роли для build, deploy и runtime; временные креденшелы; отказ от shared static keys; environment scoping; read/write split; permission boundaries или аналоги; обязательный аудит AssumeRole / Workload Identity событий.

Что смотреть в логах и телеметрии

Зрелый DevSecOps не ограничивается «pipeline прошёл/не прошёл». Нужна телеметрия по bypass-попыткам, использованию break-glass доступа, скачиванию артефактов, запуску нестандартных jobs, аномальным cloud API-вызовам и расхождениям между заявленным и фактическим состоянием среды.

Наблюдаемость должна отвечать минимум на три вопроса: что произошло, кто это сделал и какой был потенциальный радиус ущерба. Поэтому важны не только сырые логи, но и нормальные correlation identifiers, сохранность аудита и понятные playbooks для triage.

Как говорить о trade-offs

На собеседовании почти всегда полезно проговаривать trade-off. Например: «Полная блокировка всех HIGH CVE на любом PR слишком шумная; я бы начал с exploitable critical/high на internet-facing сервисах и добавил baseline/expiry для технического долга». Такой ответ показывает зрелость, а не догматизм.

Ещё один хороший паттерн — формулировать controls через риск и обратимость: что можно enforced прямо сейчас, что лучше сначала перевести в audit mode, а что требует platform-level investment, чтобы не разрушить скорость доставки.

Минимальный набор вопросов при ревью облачной роли

```
# Какие действия реально нужны workload?
# Можно ли заменить static credential на OIDC / workload identity?
# Есть ли разделение dev/stage/prod?
# Что произойдет, если этот токен утечёт?
# Логируются ли AssumeRole / STS / service account exchanges?
# Есть ли срок жизни, rotation и scope restrictions?
```

Что проговорить вслух на интервью: Где именно находится контроль: pre-commit, PR, build, deploy или runtime. Что именно блокируется жёстко, а что идёт в advisory mode. Какой риск снимает контроль и что происходит при его обходе или отказе.

Глава 7. Показательные hands-on кейсы для интервью

Ниже — три демонстрационных кейса в стиле практических заданий, которые часто дают на интервью для ролей уровня DevSecOps Engineer / Senior. Они построены так, чтобы проверять **не только знание инструмента, но и порядок мышления:** как кандидат локализует риск, как принимает решение о блокировке/допуске и как проектирует устойчивый механизм, а не разовое исправление.

Кейс 1. Компрометация CI/CD через утёкший токен и небезопасный runner

Сценарий: во внутреннем GitLab или GitHub Actions пайплайне найден токен облачного сервиса с избыточными правами. Одновременно выяснилось, что shared runner запускает привилегированные контейнеры и может собирать проекты из недоверенных merge request.

Задача кандидата: предложить быстрый containment, безопасный редизайн пайплайна и план долгосрочного hardening.

Фрагмент небезопасного pipeline, который должен насторожить

```
name: deploy
on: [push]

jobs:

 release:

  runs-on: self-hosted

  steps:
   - uses: actions/checkout@v4
   - run: docker run --privileged -v /var/run/docker.sock:/var/run/docker.sock app-builder
   - run: echo "$AWS_SECRET_ACCESS_KEY" > /tmp/key.txt
   - run: ./deploy.sh production
```

- Немедленно остановить релизы и отозвать/перевыпустить секреты, которые могли быть скомпрометированы.
- Изолировать runner для форензики и проверить журналы запусков, историю артефактов и успешных deploy за период риска.

- Перевести доступ в облако на OIDC federation, разнести execution pools по trust zones и убрать privileged execution как default.
- Добавить provenance, подпись артефактов и верификацию подписи перед deploy.

Что отличает сильный ответ: Кандидат разделяет срочные действия и долгосрочный redesign. Говорит про ownership, trust boundaries и telemetry, а не только про один сканер. Предлагает enforceable механизмы: policy, signing, short-lived credentials, templates, gated promotion, runtime visibility.

Кейс 2. Kubernetes admission hardening, подпись образов и политики запуска

Сценарий: команда деплоит сервисы в общий кластер Kubernetes. Разработчики периодически запускают образы с тегом latest, контейнеры от root и манифесты без ограничений ресурсов. Нужно ввести guardrails без полного торможения релизов.

Задача кандидата: предложить набор admission-политик, верификацию образов, сканирование и порядок rollout.

Пример проверки подписи образа на admission-этапе

```yaml
apiVersion: kyverno.io/v1
kind: ClusterPolicy

metadata:

  name: verify-signed-images

spec:
  validationFailureAction: Enforce

  rules:

    - name: require-signature

      match:

        any:

          - resources:

              kinds: [Pod]

      verifyImages:

        - imageReferences:
```

```
    - "registry.example.com/*"
  attestors:
   - entries:
      - keys:
          publicKeys: |-
           -----BEGIN PUBLIC KEY-----
           ...
           -----END PUBLIC KEY-----
```

- Сначала включить audit mode и собрать статистику нарушений по командам и типам сервисов.
- Параллельно выдать secure templates: non-root, pinned tags, resource limits, readOnlyRootFilesystem.
- После стабилизации включить enforce для production и internet-facing workloads, а затем расширять покрытие.

Что отличает сильный ответ: Кандидат понимает, что image scan без image trust — неполная защита. Обсуждает curated registries, подпись образов, verify policy, staged rollout политик и контроль исключений. Связывает admission guardrails с соседними слоями защиты: Pod Security, NetworkPolicy, RBAC и runtime detection.

Кейс 3. Terraform, облачные привилегии и дрейф инфраструктуры

Сценарий: ревью показало, что часть AWS-ресурсов создана вручную, а часть — через Terraform. В state-файле найдены чувствительные данные, IAM-политики избыточны, а S3-бакеты для логов не имеют достаточных ограничений.

Задача кандидата: спроектировать remediation-план: state hygiene, policy checks, контроль drift, least privilege и безопасный delivery IaC.

Набросок pre-merge проверок для Terraform

```
terraform fmt -check
terraform validate
checkov -d . --quiet
```

```
trivy config .

terraform plan -out=tfplan

conftest test tfplan.json
```

- Вынести state в защищённый backend с шифрованием, блокировками и ограничением доступа.
- Собрать инвентаризацию ресурсов, появившихся вне Terraform, и определить стратегию import/replace.
- Добавить policy checks на публичную экспозицию, IAM privilege escalation paths, encryption, logging и tagging standards.
- Ввести периодический drift detection и запретить ad-hoc изменения в консоли там, где это возможно организационно.

Что отличает сильный ответ: Кандидат говорит не только про tfsec/Checkov, но и про state backend, apply-roles, change approval и blast radius. Понимает, как связать remediation с процессом и ownership, а не только с разовым запуском сканера. Разделяет legacy cleanup и будущий secure-by-default delivery.

Глава 8. Русско-английский глоссарий

Термин	Определение
CI/CD	Непрерывная интеграция и доставка/развёртывание; автоматизированный конвейер сборки, тестирования и выпуска изменений.
Runner / Agent	Исполнитель задач пайплайна. Именно на нём запускаются сборка, тесты, сканирование и публикация артефактов.
Artifact	Результат сборки: образ контейнера, пакет, бинарный файл, отчёт сканирования, SBOM и т. д.
SBOM	Software Bill of Materials — машинно-читаемый перечень компонентов и зависимостей внутри артефакта.
SAST	Статический анализ исходного кода без запуска приложения.
DAST	Динамический анализ работающего приложения с точки зрения внешнего поведения.
SCA	Анализ сторонних зависимостей, лицензий и известных уязвимостей в пакетах и контейнерных слоях.
IaC	Infrastructure as Code — описание инфраструктуры кодом, например Terraform, CloudFormation, Pulumi.
PaC	Policy as Code — выражение правил контроля в виде кода и их автоматическое применение.

Термин	Определение
OIDC	OpenID Connect; часто используется для федерации identity между CI-системой и облаком без долгоживущих ключей.
Provenance	Доказуемое происхождение артефакта: где, кем, из какого кода и при каких условиях он был собран.
Attestation	Подписанное утверждение о свойствах артефакта: результатах тестов, SBOM, среде сборки, уровне доверия.
Admission Controller	Механизм, который проверяет и/или модифицирует объект до его создания в Kubernetes.
NetworkPolicy	Политика сетевого взаимодействия между pod/namespace в Kubernetes.
Runtime security	Контроль поведения приложения и контейнера во время исполнения.
Drift	Расхождение между желаемым состоянием инфраструктуры и фактическим состоянием в среде.
Secret sprawl	Бесконтрольное размножение секретов по репозиториям, пайплайнам и локальным машинам.
Break-glass	Аварийный механизм временного расширенного доступа, применяемый под аудит и строгий контроль.
Blast radius	Масштаб потенциального ущерба при компрометации учётной записи, runner или пайплайна.
Shift-left / Shift-right	Сдвиг проверок влево — на ранние

Термин	Определение
	этапы; вправо — в прод и эксплуатацию.

Глава 9. Перевод ключевых терминов

Ниже даны литературные русскоязычные эквиваленты англоязычных терминов. На практике на интервью и в документации чаще используется английский оригинал, но грамотный русский перевод помогает точнее объяснить смысл и не терять техническое содержание.

English term	Корректный перевод
Software supply chain security	Безопасность цепочки поставки программного обеспечения
Least privilege	Принцип наименьших привилегий
Workload identity	Идентичность рабочей нагрузки
Ephemeral environment	Эфемерная среда
Drift detection	Выявление дрейфа конфигурации
Admission control	Контроль допуска
Policy as Code	Политики как код
Security as Code	Безопасность как код
Break-glass access	Аварийный доступ повышенных привилегий
Blast radius	Радиус поражения / зона потенциального ущерба

English term	Корректный перевод
Artifact provenance	Подтверждённое происхождение артефакта
Attestation	Криптографическое подтверждение свойств артефакта
Shift-left	Смещение проверок в ранние этапы цикла разработки
Shift-right	Смещение проверок в эксплуатацию и пострелизный контур
Guardrails	Инженерные ограничители / защитные рамки

Глава 10. Ключевые книги по DevSecOps

Ниже — компактный список наиболее полезных книг, которые дают хороший фундамент и помогают говорить о DevSecOps не как о наборе разрозненных сканеров\тулов, а как о цельной инженерной системе. Внутри каждой записи указан линк на Amazon Book для быстрого перехода *(покупайте литературу, которая помогает вам зарабатывать, поддерживайте авторов, делайте вклад в индустрию)*

Learning DevSecOps: A Practical Guide to Processes and Tools

Авторы: Steve Suehring. Издатель: O'Reilly Media. Год: 2024.

Хорошая стартовая книга для системной сборки DevSecOps-функции: объясняет культуру, процесс и опорный стек инструментов. Полезна тем, кто хочет понимать не только отдельные тулзы, но и общую модель доставки.

Amazon Book link

Security as Code: DevSecOps Patterns with AWS

Авторы: BK Sarthak Das, Virginia Chu. Издатель: O'Reilly Media. Год: 2023.

Практическая книга про security-as-code на облачном и контейнерном стеке. Даёт хорошие паттерны для CI/CD, Kubernetes, облака и автоматизации политик.

Amazon Book link

Policy as Code: Improving Cloud Native Security

Авторы: Jimmy Ray. Издатель: O'Reilly Media. Год: 2024.

Сильная современная книга по Policy as Code: где применять политики, как их тестировать и как не превратить governance в тормоз для разработки.

Amazon Book link

Cloud Native Security

Авторы: Chris Binnie, Rory McCune. Издатель: Wiley. Год: 2021.

Фундаментальная книга по защите контейнеров, оркестрации и облачно-нативной инфраструктуры. Хороший баланс между архитектурой, hardening-практиками и runtime-рисками.

Amazon Book link

Identity Security for Software Development: Building with Identity, Secrets, and Credentials

Авторы: John Walsh, Uzi Ailon, Matt Barker. Издатель: O'Reilly Media. Год: 2025.

Очень полезная книга про workload identity, машино-машинную аутентификацию, секреты и сервисные учётные данные. Закрывает часть темы, которую часто недооценивают даже сильные DevOps-кандидаты.

Amazon Book link

Accelerating DevSecOps on AWS: Create secure CI/CD pipelines using Chaos and AIOps

Авторы: Nikit Swaraj. Издатель: Packt. Год: 2022.

Практический маршрут от сборки пайплайна до наблюдаемости и эксплуатационной зрелости на AWS. Подходит для обсуждения end-to-end delivery pipeline.

Amazon Book link

Securing the CI/CD Pipeline: Best Practices for DevSecOps

Авторы: Sai Sravan Cherukuri. Издатель: Independently published. Год: 2024.

Узко сфокусированная книга именно по защите CI/CD-контура: runners, secrets, approval gates, trusted builders и контроль артефактов.

Amazon Book link