Блок 1: Основы и Архитектура (Core Concepts & Architecture)

Базируется на Task 4.1, Section 4 и видео 1-5, 16-18.

1. Что такое **Pipeline** и из каких двух основных сущностей он состоит?
2. В каком файле должна храниться конфигурация пайплайна и где он должен лежать?
3. В чем принципиальная разница между **CI** (Continuous Integration), **CD** (Continuous Delivery) и **CD** (Continuous Deployment)?
4. Что такое **GitLab Runner**? Может ли пайплайн работать без раннера?
5. В чем разница между **Shared Runner**, **Group Runner** и **Project (Specific) Runner**? В каких случаях какой лучше использовать?
6. Какие бывают **Executors** у раннеров? В чем разница между `Shell Executor` и `Docker Executor`?
7. Что происходит, если для задачи (Job) не найден подходящий Runner (по тегам или доступности)?
8. Как GitLab понимает, какой именно раннер должен взять конкретную задачу? (Подсказка: Tags).
9. Где выполняется код пайплайна: на сервере GitLab или на машине с Runner?
10. Почему важно, чтобы файл конфигурации `.gitlab-ci.yml` лежал в репозитории вместе с кодом (Pipeline as Code)?

Блок 2: Структура .gitlab-ci.yml (Jobs, Stages, Workflow)

Базируется на Task 4.1, 4.3 и видео 6-13.

11. Как работает ключевое слово `stages`? В каком порядке выполняются задачи внутри **одной** стадии?
12. Что произойдет со стадией `deploy`, если задача в стадии `test` упадет с ошибкой?
13. Как запустить задачу из следующей стадии, не дожидаясь завершения всех задач текущей стадии? (Подсказка: `needs`).
14. В чем разница между ключевыми словами `script`, `before_script` и `after_script`? Выполнится ли `after_script`, если основной скрипт упал?
15. Как с помощью `rules` (или `only`/`except`) сделать так, чтобы задача запускалась **только** при создании Merge Request?
16. Как настроить задачу, чтобы она запускалась только вручную (по кнопке)? Какое ключевое слово за это отвечает?
17. Что делает ключевое слово `allow_failure: true`? В каких случаях его безопасно применять?
18. Как с помощью `workflow` предотвратить запуск дублирующих пайплайнов (например, один на push в ветку, второй на merge request)?

19. Можно ли запустить скрипт, лежащий в отдельном файле (например, `ci/script.sh`), внутри `.gitlab-ci.yml`? Как?
20. Что такое `image` в контексте Job? Можно ли использовать разные образы для разных задач в одном пайплайне?

Блок 3: Данные, Переменные и Артефакты (Artifacts, Cache, Variables)

Базируется на Task 4.2, 4.3 и видео 14-15, 33, 40.

21. **Классический вопрос собеседований:** В чем принципиальная разница между **Artifacts** и **Cache**?
22. Если мне нужно передать скомпилированный бинарный файл из стадии `build` в стадию `deploy`, что я должен использовать: кэш или артефакты?
23. Если я хочу ускорить установку зависимостей (`node_modules`, `.venv`), что я должен использовать: кэш или артефакты?
24. Как долго хранятся артефакты по умолчанию и как изменить это время (`expire_in`)?
25. Где задаются переменные окружения (Variables)? Назови минимум 3 места (в файле, в UI...).
26. Какой приоритет у переменных? Переменная в UI GitLab перезапишет переменную в `.gitlab-ci.yml` или наоборот?
27. Что такое **Masked** переменная? От чего она защищает?
28. Что такое **Protected** переменная? Будет ли она доступна в пайплайне, запущенном на ветке `feature/login`?
29. Как передать версию билда (например, из `git tag` или `CI_PIPELINE_ID`) в код самого приложения или в Docker-образ?
30. Какие **Predefined Variables** (предопределенные переменные) ты знаешь? Как получить имя текущей ветки или хэш коммита?

Блок 4: Docker и Образы (Docker integration)

Базируется на Task 4.2, 4.4 и видео 19, 34, 37, 39.

31. Что такое **Docker-in-Docker (dind)** и зачем нужен сервис `docker:dind` при сборке образов?
32. Как авторизоваться в приватном Docker Registry (например, GitLab Container Registry) внутри пайплайна? Какая переменная содержит токен автоматически?
33. Почему плохая практика использовать тег `latest` для деплоя в Production?
34. Как настроить **Dynamic Versioning** для Docker-образа (чтобы тег образа совпадал с git-тегом или номером коммита)?
35. Можно ли использовать `docker-compose` внутри CI/CD пайплайна? Что для этого нужно установить в image?

36. Как использовать собственный Docker-образ в качестве среды выполнения для тестов?
37. В чем преимущество принципа "Build Once, Deploy Many" (Собрать один раз, деплоить везде) перед пересборкой образа для каждого окружения?

Блок 5: Стратегии Деплоя и Окружения (Environments & CD Strategies)

Базируется на Task 4.3, 4.4, 4.5 и видео 35-36, 42-45.

38. Что такое **Environment** в GitLab CI? Что дает использование ключевого слова `environment` (url, history)?
39. Как реализовать **Staging** окружение, которое деплоится автоматически при мерже в `main`, и **Production**, который деплоится только вручную?
40. Как сделать "Rollback" (откат) на предыдущую версию, используя интерфейс Environments в GitLab?
41. Что такое **Canary Deployment**? Можно ли его реализовать средствами GitLab CI?
42. Что такое **Blue-Green Deployment**?
43. Зачем нужно делать "Stop Job" (настройка `on_stop`) для динамических окружений (Review Apps)?
44. Как защитить Production-окружение от случайного деплоя джуниором? (Подсказка: Protected Environments).
45. Почему в GitFlow деплой на Production часто привязывают к созданию **Git Tag**, а не просто к коммиту в `main`?

Блок 6: Продвинутые практики (Monorepo, Optimization, Templates)

Базируется на видео 40, 44, 46-55.

46. В чем сложность CI/CD для **Monorepo** (монорепозитория) по сравнению с **Polyrepo**?
47. Как использовать ключевое слово `changes` в `rules`, чтобы пересобирать только те микросервисы, код которых изменился?
48. Что такое **Parent-Child Pipelines** (родительские и дочерние пайплайны) и как они помогают в монорепозиториях?
49. Как избежать дублирования кода в `.gitlab-ci.yml`? (Подсказка: `extends`, `yaml anchors` или `include`).
50. Как подключить (include) шаблон пайплайна, который лежит в другом репозитории или является стандартным шаблоном GitLab?
51. Как ускорить пайплайн с помощью распараллеливания тестов (`parallel`)?
52. В чем плюсы выноса логики деплоя в отдельные скрипты или шаблоны?

Блок 7: Kubernetes & Integration

Базируется на видео 24-27, 56-60.

53. Как пайплайн GitLab CI получает доступ к кластеру Kubernetes? (Где хранится `KUBECONFIG`?).
54. Какой Docker-образ обычно используется для выполнения команд `kubectl` или `helm`?
55. Как безопасно передать секреты (пароли БД, API ключи) в Kubernetes при деплое через CI/CD?
56. Нужно ли устанавливать GitLab Runner прямо внутрь кластера Kubernetes? Какие это дает преимущества?
57. Как обновить версию образа в Deployment манифесте Kubernetes при CI/CD (используя `sed`, `envsubst` или `helm`)?

Блок 8: DevSecOps (Безопасность)

Базируется на Task 4.6 и видео 41.

58. Что такое **SAST** (Static Application Security Testing)? На каком этапе пайплайна его лучше запускать?
59. Что такое **DAST** (Dynamic Application Security Testing)? Требуется ли для него запущенное приложение?
60. Что такое **Container Scanning**? Что он ищет внутри Docker-образов?
61. Что такое **Secret Detection**? Почему это критически важно проверять на каждом коммите?
62. Как заблокировать Мерж Реквест, если сканер безопасности нашел критическую уязвимость?
63. Что такое **Defect Dojo** и зачем интегрировать его с пайплайном?

Блок 9: Ситуационные задачи (Troubleshooting & Best Practices)

Проверка понимания логики.

64. Пайплайн завис в статусе "Pending". Раннеры зарегистрированы и активны (зеленые). В чем может быть причина? (Проверь теги, проверь "Run untagged jobs").
65. У вас есть job `build` и job `test`. Job `test` требует файл, созданный в `build`. Вы запускаете пайплайн, но `test` падает с ошибкой "File not found". `build` прошел успешно. В чем причина? (Забыли `artifacts`).
66. Вы запушили секретный токен AWS в репозиторий в файле `.gitlab-ci.yml`. Что нужно сделать немедленно? (Просто удалить коммит недостаточно).
67. Разработчик жалуется, что пайплайн идет слишком долго (20 минут). Какие шаги по оптимизации вы предпримете в первую очередь?

68. Вы хотите, чтобы тесты запускались на каждом коммите, а деплой только при появлении тега `v*`. Как это описать в `rules`?
69. Почему считается плохой практикой делать `git pull` на боевом сервере внутри CI пайплайна вместо деплоя Docker-контейнера?
70. Как вы будете дебажить упавший Job, если по логам непонятно, что произошло? (Можно ли подключиться к раннеру?).
71. Зачем разделять `build` и `deploy` на разные стадии? Почему нельзя сделать все в одном скрипте?
72. У вас есть ветка `main` (Protected). Как разрешить пайплайну пушить в эту ветку (например, для автоматического обновления версии в файле), если "Push" запрещен для всех? (Использование Deploy Tokens или Project Access Tokens).
73. Как настроить уведомления (Email, Slack) о падении пайплайна?
74. Что делать, если `docker build` занимает много времени из-за скачивания базовых слоев? (Docker layer caching).
75. Вы настроили GitLab Runner на своем локальном ноутбуке для тестов. Что произойдет, когда вы закроете крышку ноутбука?

Вот список **топ-20 самых популярных вопросов на собеседованиях** по GitLab CI/CD.

Эти вопросы встречаются чаще всего, так как они проверяют не просто знание синтаксиса, а понимание архитектуры и процессов. Ответы сформированы на основе ваших материалов.

Топ-3 "Вечных" вопроса (Спрашивают почти всегда)

1. В чем разница между Continuous Delivery и Continuous Deployment?

- **Суть вопроса:** Понимаете ли вы бизнес-процесс релиза.

2. Чем отличаются Artifacts (Артефакты) от Cache (Кэша)?

- **Суть вопроса:** Это самая частая ошибка новичков, влияющая на производительность и логику.

3. Что такое GitLab Runner и зачем он нужен?

- **Суть вопроса:** Понимание архитектуры (код не выполняется "в воздухе").

⚙️ Технические вопросы (Hard Skills)

4. Где должен лежать файл конфигурации CI/CD и как он называется?

5. Что произойдет, если одна из команд в `script` завершится с ошибкой?

6. Как передать файл, созданный в стадии `build`, в стадию `deploy`?

7. Как сделать так, чтобы деплой на Production запускался только по кнопке?

8. В чем разница между `stages` (глобально) и `stage` (в job)?

9. Как запустить задачу только для определенной ветки (например, `main`)?

10. Зачем нужно указывать `image` в задаче?

🛡️ Безопасность и DevOps-практики

11. Как безопасно хранить пароли и ключи API в GitLab CI?

12. Что такое "Канареечный релиз" (Canary Deployment)?

13. Что такое Blue-Green Deployment?

14. Зачем нужен SAST (Static Application Security Testing)?

15. Что такое "Ад слияния" (Merge Hell) и как CI помогает его избежать?

🧠 Ситуационные вопросы (на уровень Middle)

16. Пайплайн идет слишком долго. Как можно его ускорить?

17. Как вы реализуете версионирование Docker-образов? Почему нельзя использовать `latest`?

18. Как сделать так, чтобы при падении тестов (`test` stage) деплой (`deploy` stage) не начинался?

19. Что такое DAST и чем он отличается от SAST?

20. Как реализовать DevSecOps подход в пайплайне?