Быстрый, масштабируемый CI для фронтендера

Эмиль Янгиров
trash_js

Эмиль Янгиров

Frontend Developer / FrontOps

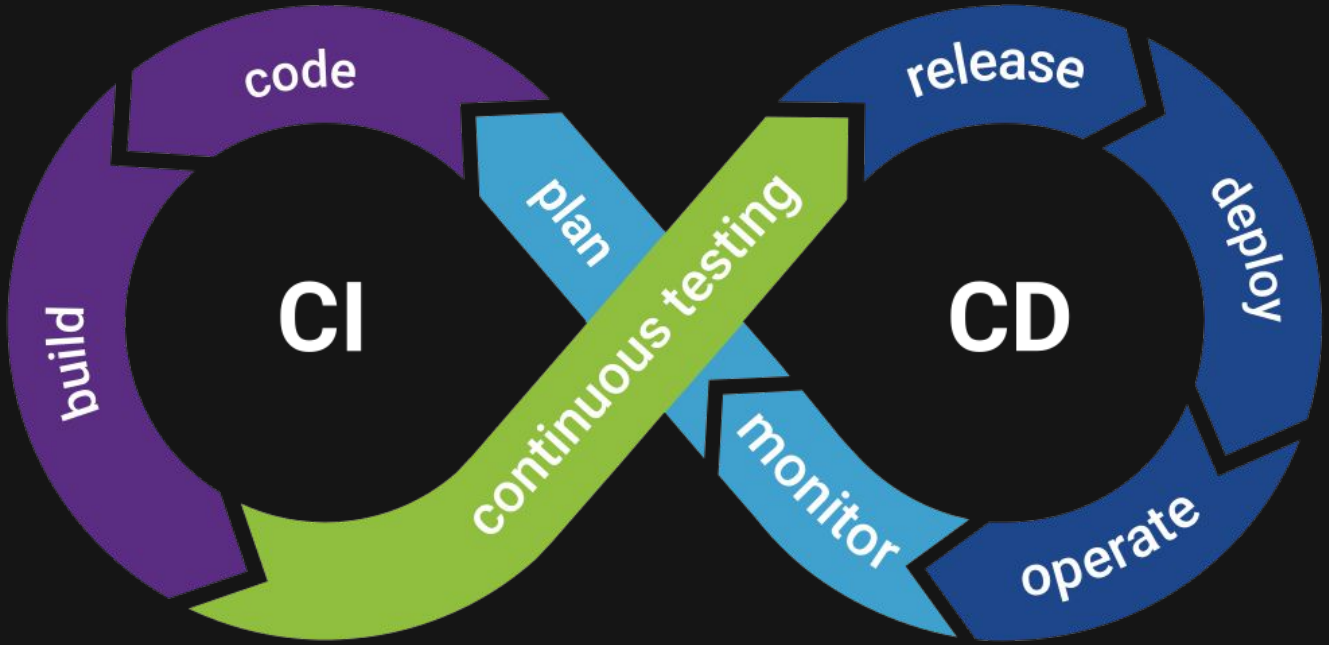
Core Team @ ЗВУК

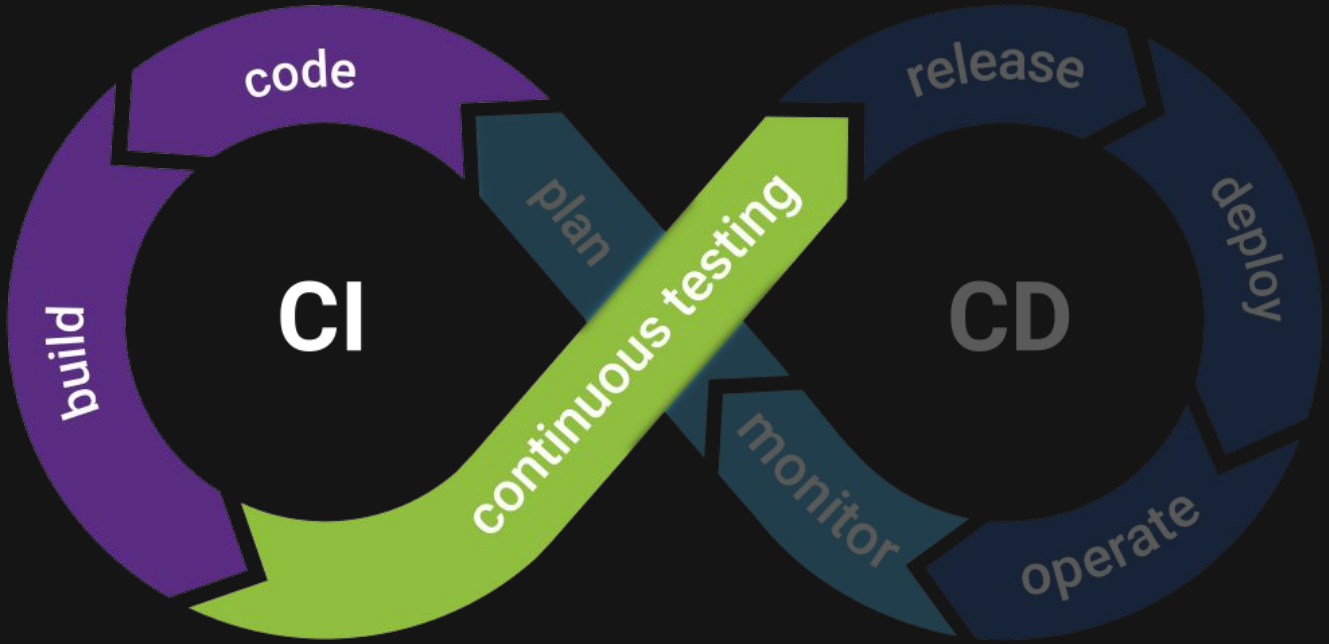
trash_js

Пишу о фронтенд-экосистеме и FrontOps

О чём будет доклад

- 📌 Проблемы и теория
- 📌 Ускорение CI
- 📌 Масштабирование конфигов CI
- 📌 Полезности и инструменты



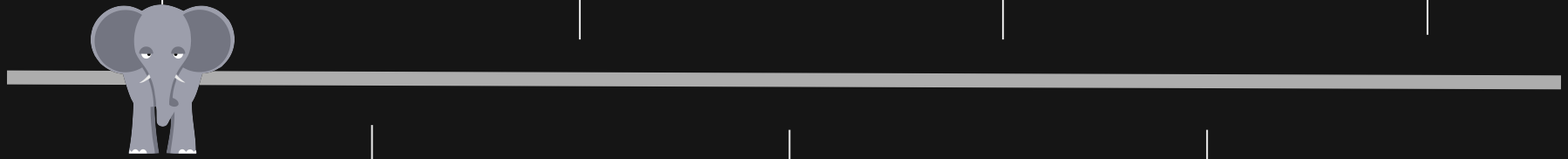


Проблема

Собираем

Масштабируем

Выводы



Теория

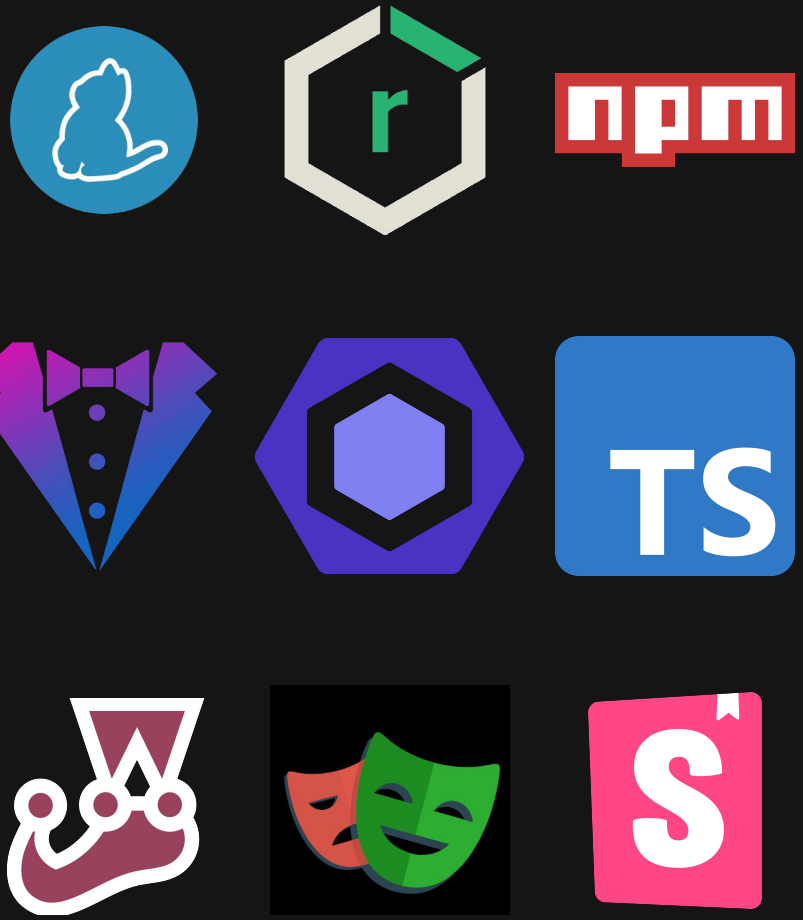
Оптимизируем

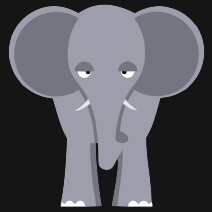
Советы

Типовой пайплайн CI/CD

Типовой пайплайн CI/CD

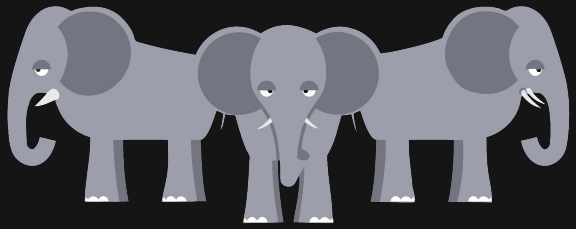
Типовой пайплайн CI/CD



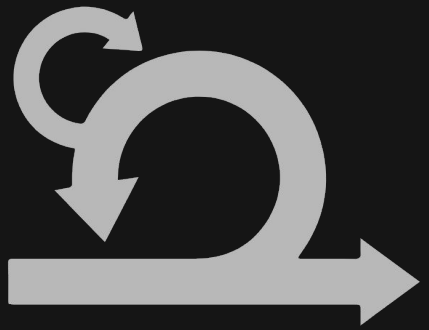


🗂️ Проект

```
lint:
 script:
   - yarn install
   - yarn eslint .
```



Копируем

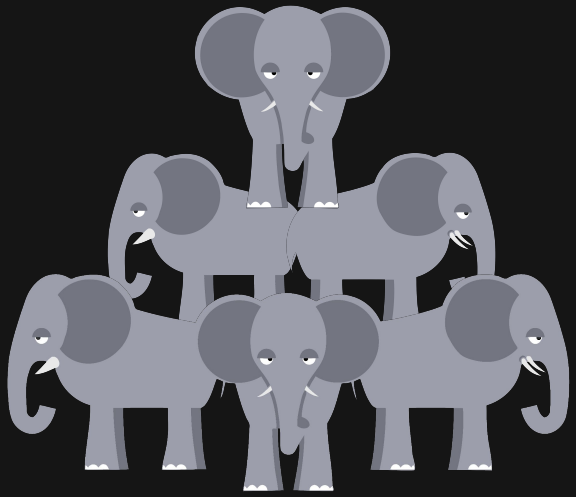


🗂️ Проект 1

```
lint:
 script:
   - yarn install
   - yarn eslint .
```

🗂️ Проект 2

```
lint:
 script:
   - yarn install
   - yarn lint:eslint
   - yarn lint:prettier
```



🗂️ Проект 1

```
lint:
 script:
   - yarn install
   - yarn eslint
```

🗂️ Проект 2

```
lint:
 script:
   - yarn install
   - yarn lint:eslint
   - yarn lint:prettier
```

🗂️ Проект 3

```
lint:
 script:
   - yarn install
   - yarn build
   - yarn lint:code
   - yarn lint:format
   - yarn lint:styles
```

😔 Если нужно сделать изменения?

- После обновления версии линтера, поменялся API для CLI
- Добавить этап проверки безопасности (DevSecOps)
- Изменить настройки деплоя

🫠 Проблемы

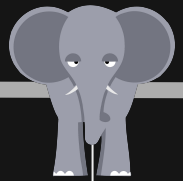
- Нужно менять в каждом проекте
- Конфигурация CI/CD начинает отличаться
- Затрудняется поддержка
- Отвлекает разработчиков

Проблема

Собираем

Масштабируем

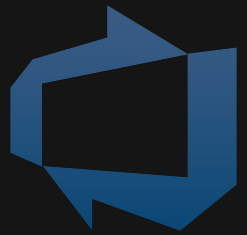
Выводы



Теория

Оптимизируем

Советы



⚠ Disclaimer

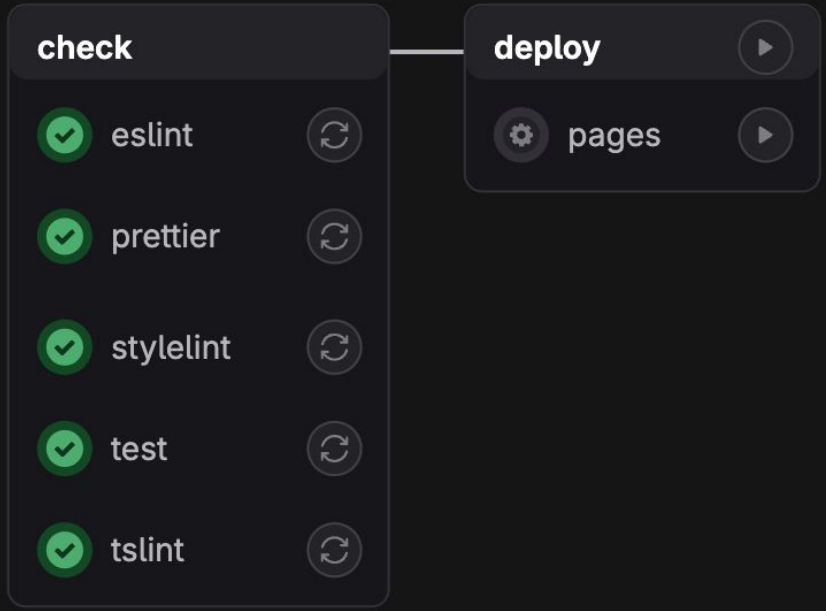
🗂️ .gitlab-ci.yml

Что такое пайплайн?

Runner: Машина выполняющая джобы пайплайна

Stages: Логические блоки, выполняются последовательно

Jobs: Задачи **внутри стейджей**, могут выполняться параллельно



https://docs.gitlab.com/ee/ci/yaml/

🤟🏻 Синтаксис

Stages

Stage — этап в пайплайне, определяющий порядок выполнения задач.

Job — задача, выполняющая операцию **внутри этапа**.

```
stages:
  - lint
  - build
  - deploy
```

https://docs.gitlab.com/ee/ci/yaml/#stages

Jobs

```yaml
eslint:
  stage: lint
  script:
    - yarn lint:code

test:
  stage: lint
  script:
    - yarn test:coverage
```

```yaml
build stage:
  stage: build
  script:
    - yarn build:prod

deploy stage:
  stage: deploy
  script:
    - make deploy
  only:
    - master
```

https://docs.gitlab.com/ee/ci/yaml/#job-keywords

Downstream Pipelines

```
qa-test:
 trigger:
   project: qa/web-autotests
   branch: master
rules:
 - changes:
     - pages/**/*
```

https://docs.gitlab.com/ee/ci/pipelines/downstream_pipelines.html

Artifacts

```yaml
test:
 stage: test
 rules:
   - if: $CI_PIPELINE_SOURCE == "merge_request_event"
 script:
   - yarn test:coverage
 artifacts:
   paths:
     - coverage/
   expire_in: 1 week
```

https://docs.gitlab.com/ee/ci/yaml/#artifacts

Rules

```yaml
test:
  stage: test
  rules:
    - if: $CI_COMMIT_BRANCH == "master"
    - if: $CI_COMMIT_BRANCH == "develop"
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"
  script:
    - yarn test
```

https://docs.gitlab.com/ee/ci/yaml/#rules

Predefined variables

- **CI_COMMIT_TAG** – тег, для которого запускается пайплайн
- **CI_COMMIT_BRANCH** – имя ветки, из которой был сделан коммит
- **CI_PIPELINE_SOURCE** – источник пайплайна (web, schedule, MR, etc)

https://docs.gitlab.com/ee/ci/variables/predefined_variables.html

Includes

```yaml
include:
  - project: 'web-ci/ci'
    ref: master
    file:
      - 'lint.yml'
```

```yaml
include:
  - project: 'web-ci/ci'
    ref: master
    file:
      - 'lint.yml'
    rules:
      - if: $CI_COMMIT_BRANCH == 'master'
```

https://docs.gitlab.com/ee/ci/yaml/#include

Anchors

```
.lint: &install
 before_script:
   - yarn install

eslint:
 <<: *install
 stage: lint
 script:
   - yarn lint:code
```

https://docs.gitlab.com/ee/ci/yaml/yaml_optimization.html#yaml-anchors-for-scripts

Anchors

```
.lint: &install
 before_script:
   - yarn install

eslint:
 <<: *install
 stage: lint
 script:
   - yarn lint:code
```

Reference

```
.lint:
  script:
   - yarn install

eslint:
 stage: lint
 script:
  - !reference
    [.lint, script]
  - yarn lint:code
```

https://docs.gitlab.com/ee/ci/yaml/yaml_optimization.html#reference-tags

Anchors

```
.lint: &install
 before_script:
   - yarn install

eslint:
 <<: *install
 stage: lint
 script:
   - yarn lint:code
```

Reference

```
.lint:
  script:
    - yarn install

eslint:
 stage: lint
 script:
   - !reference
     [.lint, script]
   - yarn lint:code
```

Extends

```
.lint:
 stage: lint
 before_script:
   - yarn install

eslint:
 extends: .lint
 script:
   - yarn lint:code
```

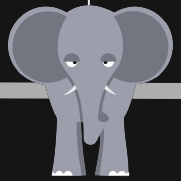
https://docs.gitlab.com/ee/ci/yaml/#extends

Проблема

Собираем

Масштабируем

Выводы



Теория

Оптимизируем

Советы

Собираем всё вместе

⚙️ Чего мы хотим?

- **Отдельный репозиторий** для конфигов CI
- Подключение к проектам через **include**

```yaml
include:
 - project: 'web-ci/ci'
   ref: master
   file:
     - 'common.yml' # основные настройки пайплайна
     - 'lint.yml' # линтинг
     - 'test.yml' # тесты
```

🗂️ Нотация в package.json

```
"scripts": {
 "lint": "...",
 "lint:code": "eslint ...",
 "lint:format": "prettier ...",
 "lint:styles": "stylelint ...",
 "lint:types": "tsc —noEmit",
  ...
}
```

🖖 Разделяем на джобы

Долгое время выполнения

Все линтеры запускаются последовательно, замедляя обратную связь.

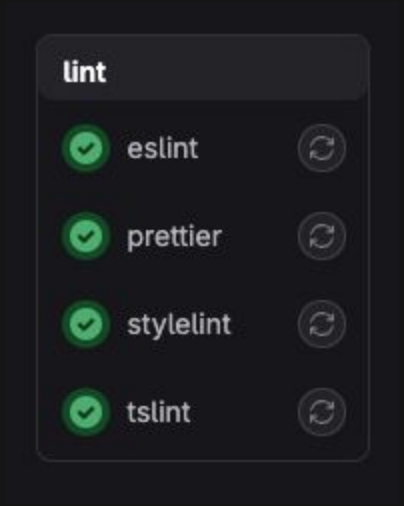
Трудно найти ошибки

В одной джобе сложно выявить, какой линтер провалился.

```
lint:
  script:
    - yarn install
    - yarn stylelint
    - yarn eslint
    - yarn prettier
    - yarn tsc --noEmit
```

lint.yml


```yaml
.lint-settings:
  stage: lint
  before_script:
    - yarn install
  rules:
    - if: $CI_PIPELINE_SOURCE == "merge_request_event"

eslint:
  extends: .lint-settings
  script:
    - yarn lint:code

stylelint:
  extends: .lint-settings
  script:
    - yarn lint:styles

prettier: ...

tslint: ...
```



Где проблема?

lint.yml

- Долгое время пайплайна
- Бойлерплейт код
- Потеря ресурсов

```
.lint-settings:
 stage: lint
 before_script:
  - yarn install
 rules:
  - if: $CI_PIPELINE_SOURCE ==
    "merge_request_event"

eslint:
 extends: .lint-settings
 script:
   - yarn lint:code

stylelint:
 extends: .lint-settings
 script:
   - yarn lint:styles

prettier: ...

tslint: ...
```

⚙️ Cache

Кэш сохраняет временные файлы между заданиями и пайплайнами для ускорения работы.

```
cache:
 key:
   files:
     - yarn.lock
 paths:
   - .yarn
   - node_modules
 policy: pull
```

https://docs.gitlab.com/ee/ci/caching/

⚙️ Cache VS Artifacts

	Cache 🏎️	**Artifacts** 📦
Назначение	Ускорение CI/CD за счет повторного использования зависимостей.	Передача файлов между job’ами внутри одного pipeline.
Где хранится	На раннере или в S3. Доступ только внутри раннера.	В GitLab (S3, локально). Доступ через UI.
Срок хранения	Неограничен, нужно очищать вручную.	Настраивается через expire_in.
Использование	node_modules/, vendor/, target/ и т. д.	Билды, тестовые отчеты, скомпилированные артефакты.

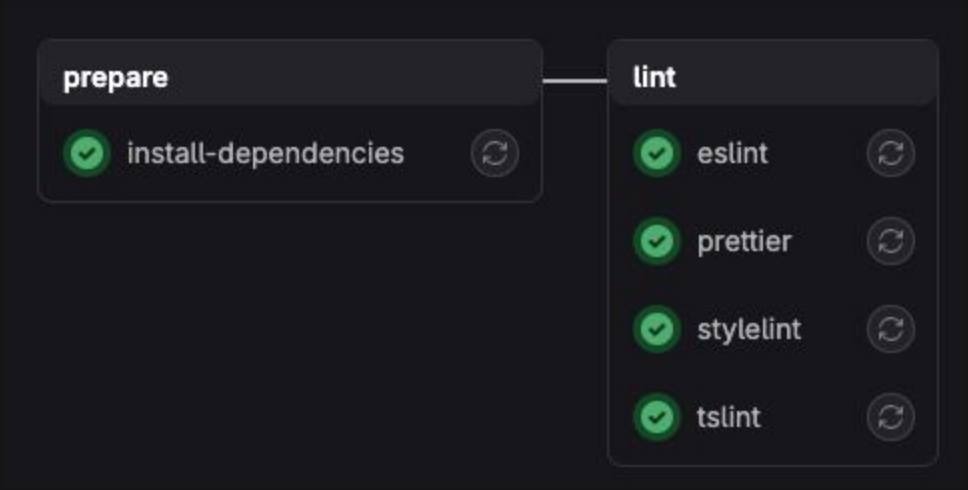
https://docs.gitlab.com/ee/ci/caching/

⚙️ Cache Policy

- **pull** только использует уже существующий кэш.
- **push** только обновляет кэш, не используя текущий.
- **pull-push** использует текущий кэш и при необходимости обновляет его.

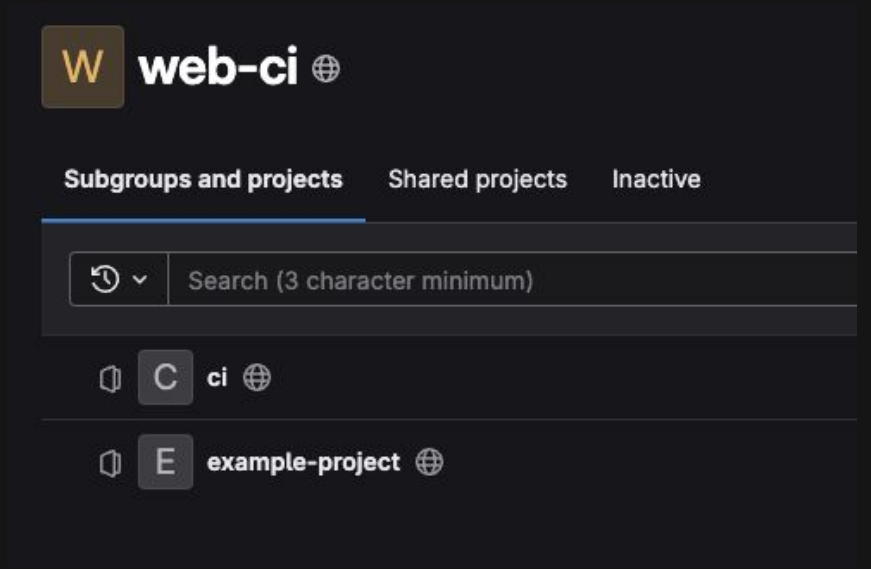
```yaml
cache: &global-cache
  key:
    files: yarn.lock
  paths:
    - .yarn
    - node_modules
  policy: pull

install-dependencies:
  stage: prepare
  script:
    - yarn install --immutable
  cache:
    - <<: *global-cache
      policy: pull-push
```

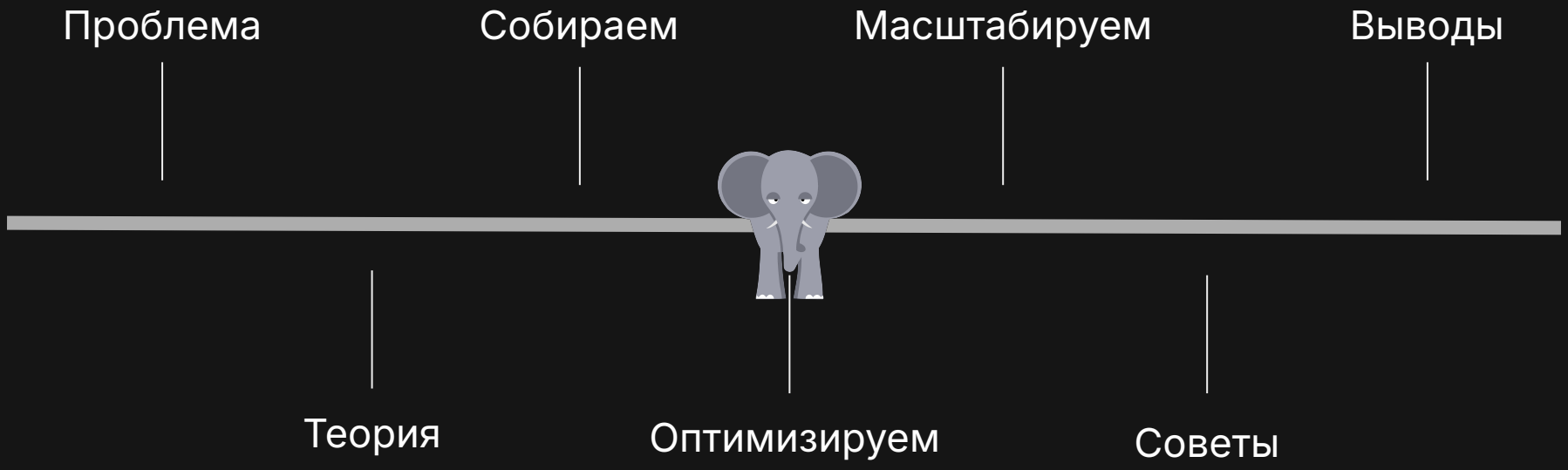
https://docs.gitlab.com/ee/ci/yaml/#cachepolicy



⚙️ Подключение к проекту

```yaml
include:
  - project: 'web-ci/ci'
    ref: master
    file:
      - 'common.yml'
      - 'lint.yml'
```





Где еще проблема?

```
eslint . \
  --cache
  --cache-strategy content
```

https://eslint.org/docs/latest/use/command-line-interface#caching

Стратегии кэширования

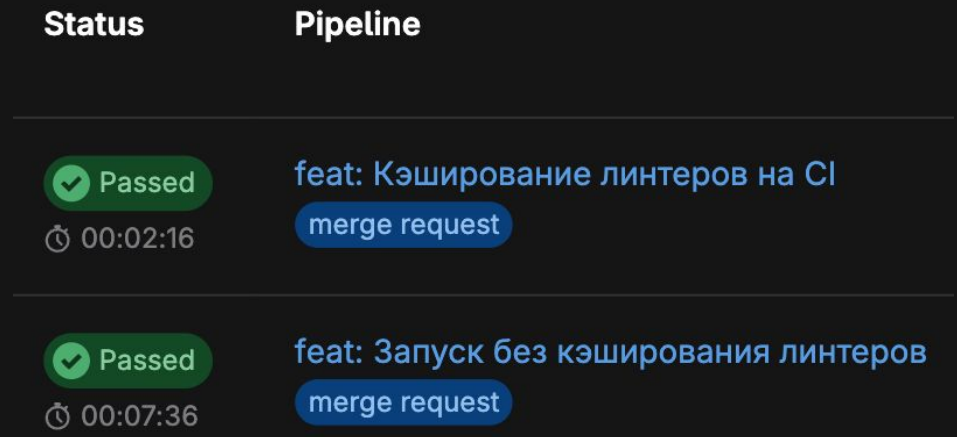
Стратегия кэширования определяет, как система будет отслеживать изменения и когда повторно использовать кэш для ускорения выполнения задач в пайплайне.

- **Metadata**: Кэшируется на основе метаданных файлов (время изменения, размер). Используется **по умолчанию**.
- **Content**: Кэшируется на основе содержимого файлов.

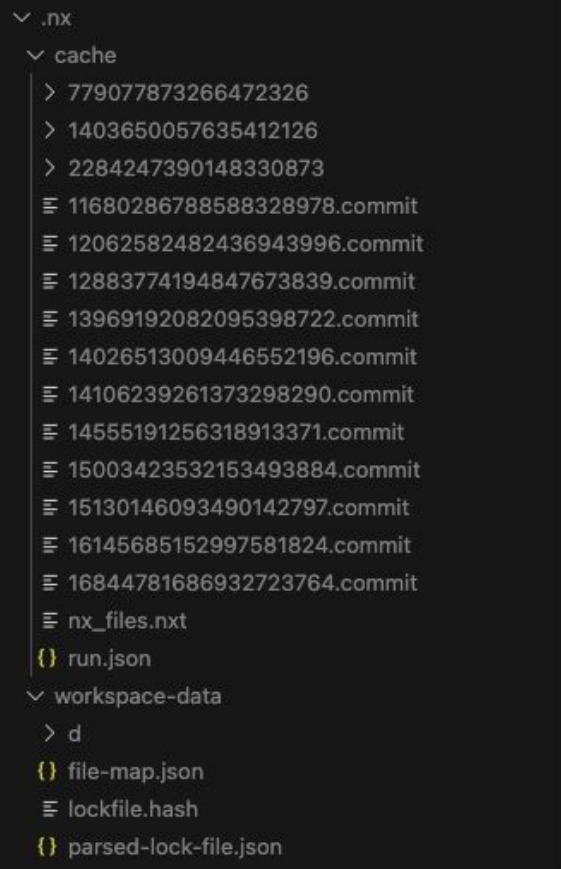
https://eslint.org/docs/latest/use/command-line-interface#--cache-strategy

lint.yml

```yaml
eslint:
 extends: .lint-settings
 script:
   - yarn lint:code
 cache:
   - !reference [cache]
   - key: eslint-cache
     paths:
       - .eslintcache
     policy: pull-push
```



Еще немного о кэше



https://nx.dev/concepts/how-caching-works

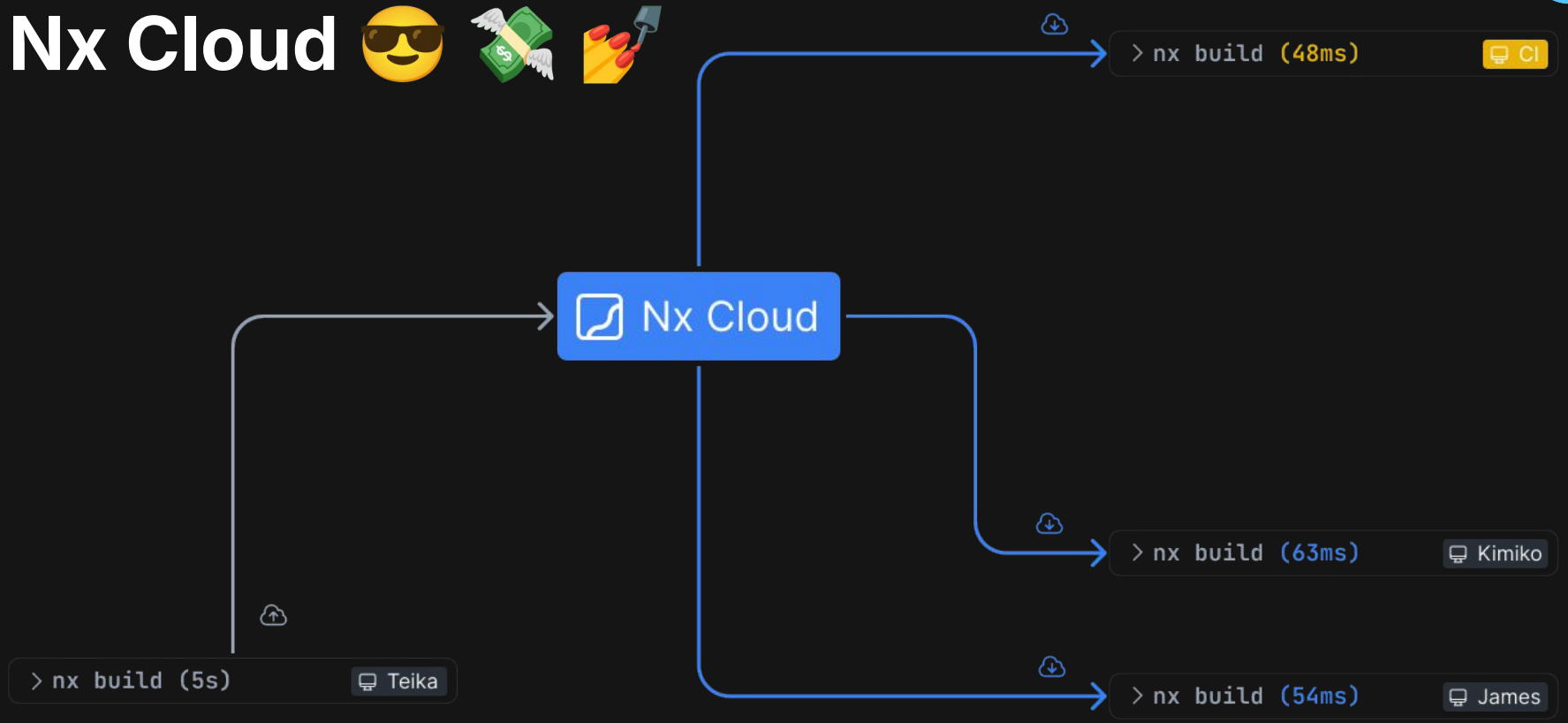
NX: Unknown Local Cache Error

```
NX Invalid Cache Directory for Task "myapp:build"

The local cache artifact in ".nx/cache/nx/7865247804590281 95" was not generated on this machine.
As a result, the cache's content integrity cannot be confirmed, which may make cache restoration pc
If your machine ID has changed since the artifact was cached, run "nx reset" to fix this issue.
Read about the error and how to address it here: https://nx.dev/troubleshooting/unknown-local-cache
```

```
NX Unrecognized Cache Artifacts

Nx found unrecognized artifacts in the cache directory and will not be able to use them.
Nx can only restore artifacts it has metadata about.
Read about this warning and how to address it here: https://nx.dev/troubleshooting/unknown-local-ca
```

https://nx.dev/troubleshooting/unknown-local-cache

Nx Cloud 😎 💸 🖌️



https://nx.dev/ci/features/remote-cache

🥰 Open Source

All Custom Runners

Runner	Storage
nx-remotecache-azure	Azure Blob Storage
@pellegrims/nx-remotecache-s3	S3 Storage
nx-remotecache-minio	MinIO Storage
@vercel/remote-nx	Vercel Cache
nx-remotecache-redis	Redis Cache

https://github.com/NiklasPor/nx-remotecache-custom

Legacy Cache

Nx 21 переходит на кеш с БД вместо файлов.

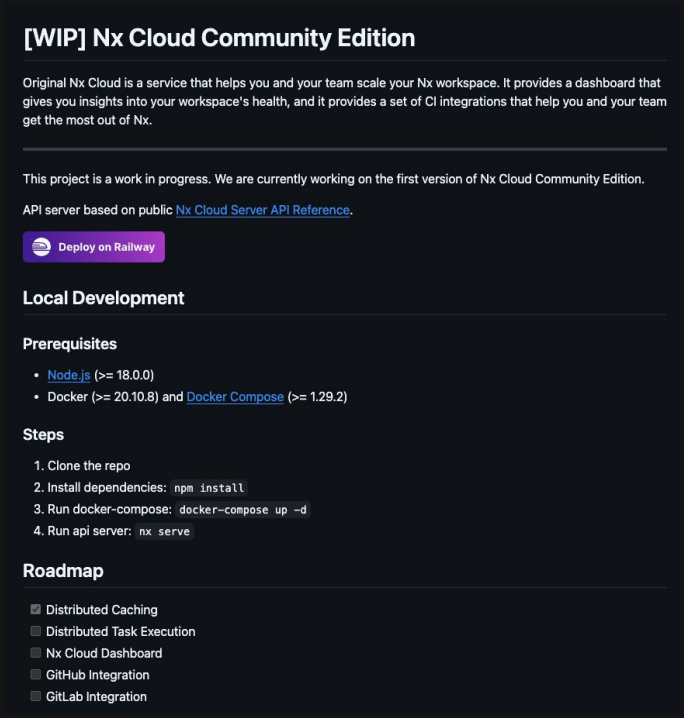
Legacy файловый кеш доступен в Nx 20 через **useLegacyCache: true**.

tasksRunnerOptions: кастомные таск-раннеры не работают с новым кэшом.

NX_REJECT_UNKNOWN_LOCAL_CACHE больше не действует, для кэша лучше использовать *Nx Cloud* или *Nx Powerpack*.

https://nx.dev/deprecated/legacy-cache

Community Nx Cloud ❤



[WIP] Nx Cloud Community Edition

Original Nx Cloud is a service that helps you and your team scale your Nx workspace. It provides a dashboard that gives you insights into your workspace's health, and it provides a set of CI integrations that help you and your team get the most out of Nx.

This project is a work in progress. We are currently working on the first version of Nx Cloud Community Edition.

API server based on public Nx Cloud Server API Reference.

Deploy on Railway

Local Development

Prerequisites

- Node.js (>= 18.0.0)
- Docker (>= 20.10.8) and Docker Compose (>= 1.29.2)

Steps

1. Clone the repo
2. Install dependencies: `npm install`
3. Run docker-compose: `docker-compose up -d`
4. Run api server: `nx serve`

Roadmap

- [x] Distributed Caching
- [ ] Distributed Task Execution
- [ ] Nx Cloud Dashboard
- [ ] GitHub Integration
- [ ] GitLab Integration

https://github.com/IKatsuba/nx-cloud

Status	Pipeline
Passed 00:08:04	feat: Добавление NX-кэша в CI merge request
Passed 00:18:31	feat: Без NX-кэша в CI merge request

Добавляем тесты

test.yml

```yaml
.test:
  stage: test
  script:
    - yarn test

.coverage:
  stage: test
  script:
    - yarn test:coverage
  artifacts:
    paths:
      - coverage/
    reports:
      coverage_report:
        coverage_format: cobertura
        path: coverage/cobertura-coverage.xml
```

Оптимизация тестов на CI

- **--onlyChanged:** Запуск только изменённых тестов
- **--changedSince:** Проверка изменений относительно основной ветки
- **--findRelatedTests:** Поиск связанных тестов
- **Настройка параллельности выполнения**
  - **--runInBand:** Выполняет тесты последовательно
  - **--maxWorkers=50%:** Ограничивает использование воркеров до 50% доступных CPU
  - **--workerThreads:** Включает использование потоков воркеров для повышения производительности

https://vitest.dev/guide/improving-performance

Кастомизация CI

```yaml
include:
 - project: 'web-ci/ci'
   ref: master
   file:
     - 'common.yml'
     - 'lint.yml'
     - 'test.yml'
```

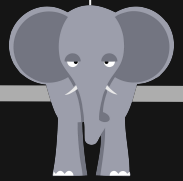
```yaml
test:
 extends: .coverage

prettier:
 when: never
```

Проблема

Собираем

Масштабируем

Выводы



Теория

Оптимизируем

Советы

⚙️ Список пайплайнов

- common.yml – основные настройки пайплайна
- lint.yml – линтинг ESLint, Stylelint, Prettier и типов TypeScript
- test.yml – запуск тестов

⚙️ Список пайплайнов

- common.yml – основные настройки пайплайна
- lint.yml – линтинг ESLint, Stylelint, Prettier и типов TypeScript
- test.yml – запуск тестов
- gitlab-pages.yml – деплой на Gitlab Pages
- publish-packages.yml – джобы для публикации пакетов

⚙️ Список пайплайнов

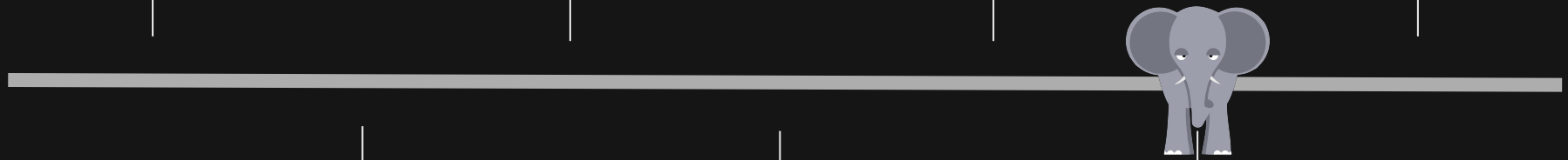
- common.yml – основные настройки пайплайна
- lint.yml – линтинг ESLint, Stylelint, Prettier и типов TypeScript
- test.yml – запуск тестов
- gitlab-pages.yml – деплой на Gitlab Pages
- publish-packages.yml – джобы для публикации пакетов
- devsecops.yml – пайплайн запуска проверок DevSecOps
- sonarqube.yml – запуск проверки SonarQube

Проблема

Собираем

Масштабируем

Выводы



Теория

Оптимизируем

Советы

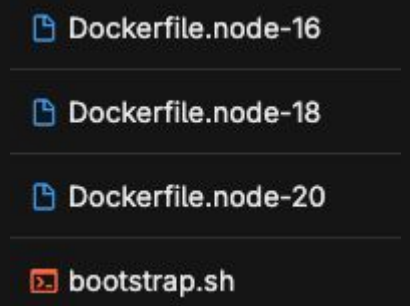
💡 Tips & Tricks

Базовый docker-образ

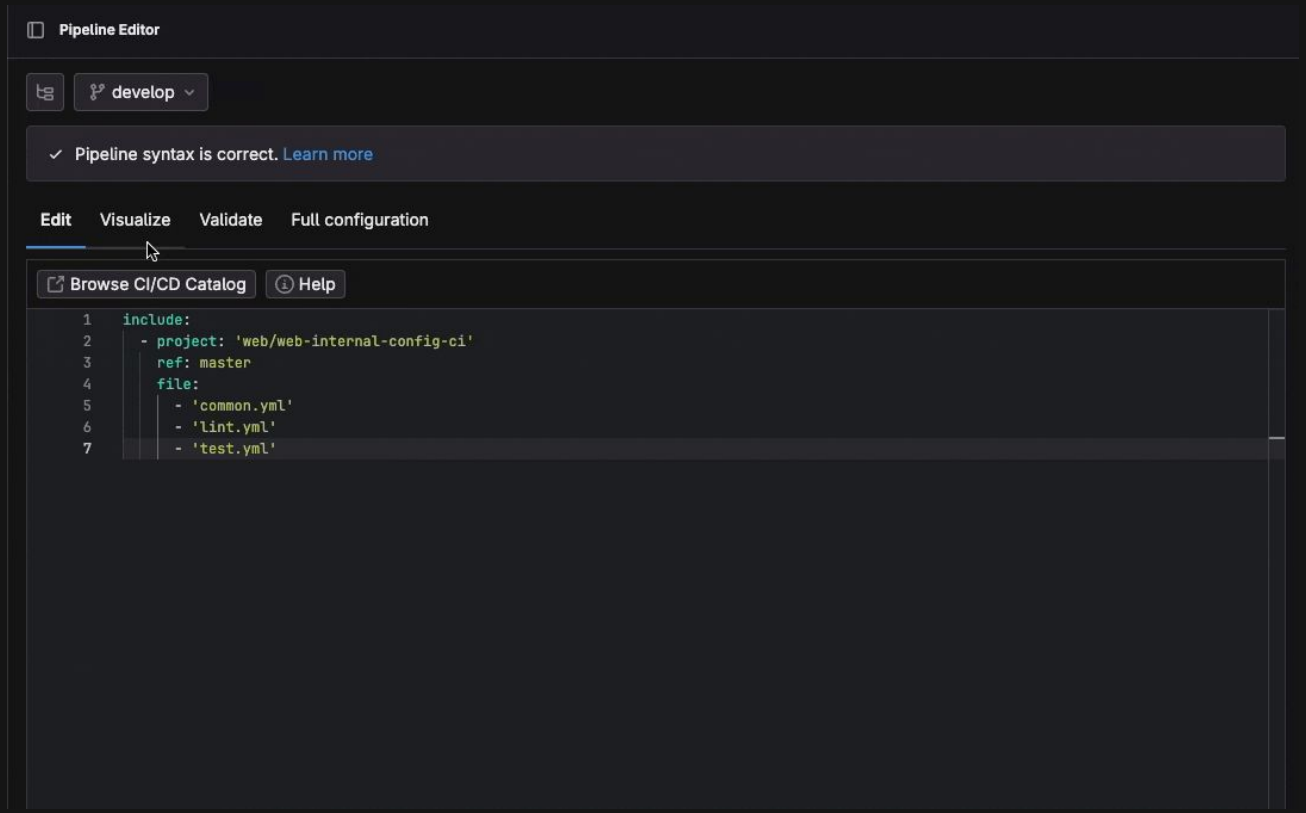
Стандартные образы могут не содержать всех необходимых зависимостей.

Можно использовать один образ в качестве основы для всех CI/CD джоб.

```
# Dockerfile
FROM node:20.15.1-alpine

RUN apk add sharp
RUN rm -rf /var/lib/apt/lists/
RUN apk update
```



GitLab Pipeline Editor

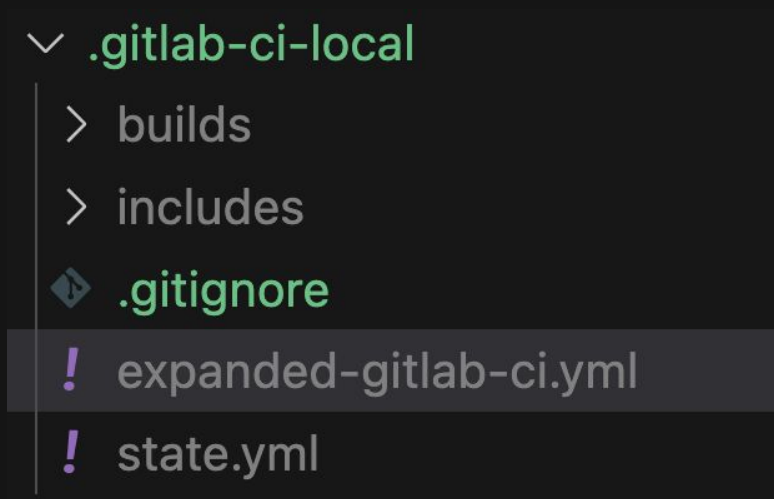


gitlab-ci-local

Инструмент для локального запуска и отладки GitLab CI пайплайнов.

- Экономит время, быстрая итерация
- Поддержка артефактов, переменных
- Локальная отладка

```
brew install gitlab-ci-local
gitlab-ci-local .
```



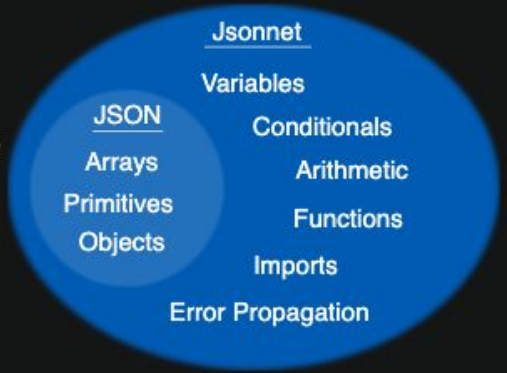
https://github.com/firecow/gitlab-ci-local

A configuration language
for app and tool developers

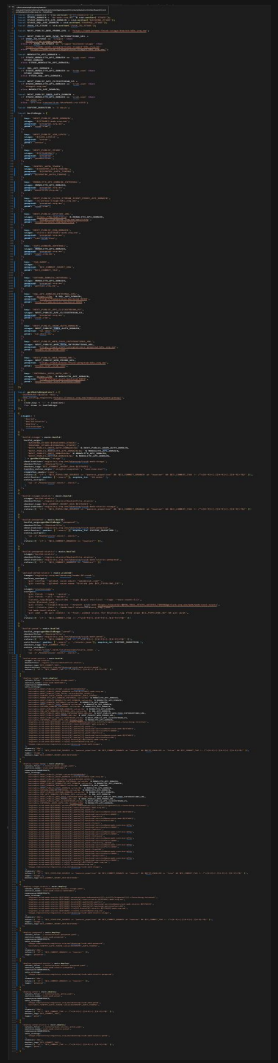
- Generate config data
- Side-effect free
- Organize, simplify, unify
- Manage sprawling config

A simple extension of JSON

- Open source (Apache 2.0)
- Familiar syntax
- Reformatter, linter
- Editor & IDE integrations
- Formally specified



https://jsonnet.org/



500+ строк

⬇️⬇️⬇️

50 строк

Переменные

```
local LOG_LEVEL = std.extVar('LOG_LEVEL');
local buildArgs = [
 {
   key: 'NEXT_PUBLIC_LOG_LEVEL',
   stage: LOG_LEVEL,
   preprod: 'info',
   prod: 'error',
 }
 {
   key: 'NEXT_PUBLIC_CDN',
   default: 'cdn.domain.com',
 }
];
```

Функции

```
local getBuildArgs(env) = [
 item.key + '=' + std.get(item, env, item.default)
 for item in buildArgs
];
```

Результат

```
{
 stage: getBuildArgs("stage"),
 preprod: getBuildArgs("preprod"),
 prod: getBuildArgs("prod")
}
```

```
brew jsonnet
jsonnet test.jsonnet \
  > generated-config.yml
```

Jsonnet

```
{
  "preprod": [
     "NEXT_PUBLIC_LOG_LEVEL=info",
     "NEXT_PUBLIC_CDN=cdn.domain.com"
  ],
  "prod": [
     "NEXT_PUBLIC_LOG_LEVEL=error",
     "NEXT_PUBLIC_CDN=cdn.domain.com"
  ],
  "stage": [
     "NEXT_PUBLIC_LOG_LEVEL=info",
     "NEXT_PUBLIC_CDN=cdn.domain.com"
  ]
}
```

Почему Jsonnet, а не просто JS?

- Jsonnet не требует Node.js или зависимостей
- Установка и выполнение в одну строку

```
brew jsonnet
jsonnet test.jsonnet > generated-config.yml
```

Реальный пример с Jsonnet

Библиотека для упрощения создания типовых пайплайнов на Jsonnet, используемых для сборки и деплоя сервисов в CI/CD.

```
local web = import 'web-ci/web.libsonnet';

{
 local projectName = '$CI_PROJECT_NAME',
 local env = 'stage',

 'build-stage': web.build(env, projectName, buildArgs),
 'deploy-stage': web.deploy(env, projectName),
}
```

```
/* Функция возвращает аргументы билда в зависимости от окружения */
local getBuildArgs(env, args=[]) = [
 item.key + '=' + if std.objectHas(item, env) then item[env] else item.value
 for item in args
];

/* Функция возвращает правила запуска пайплайна в зависимости от окружения */
local getRules(env) = {
 stage: [{ 'if': '$CI_PIPELINE_SOURCE == "web"' }],
 preprod: [{ 'if': '$CI_COMMIT_BRANCH == "master"' }],
 prod: [{ 'if': '$CI_COMMIT_TAG =~ /^v[0-9]+[.][0-9]+[.][0-9]+?$/' }],
}[env];

/* Функция возвращает докер-тег образа, в зависимости от окружения */
local getTag(env) = {
 stage: '$CI_COMMIT_SHORT_SHA',
 preprod: '$CI_COMMIT_SHORT_SHA',
 prod: '$CI_COMMIT_TAG',
}[env];

/* Функция возвращает стандартизированное название докер-образа */
local getDestination(suffix=null) =
    'registry.company/web/' + std.join('-', ['$CI_PROJECT_NAME', suffix]);
```

```
/* Функция возвращает аргументы билда в зависимости от окружения */
local getBuildArgs(env, args=[]) = [
 item.key + '=' + if std.objectHas(item, env) then item[env] else item.value
 for item in args
];

/* Функция возвращает правила запуска пайплайна в зависимости от окружения */
local getRules(env) = {
 stage: [{ 'if': '$CI_PIPELINE_SOURCE == "web"' }],
 preprod: [{ 'if': '$CI_COMMIT_BRANCH == "master"' }],
 prod: [{ 'if': '$CI_COMMIT_TAG =~ /^v[0-9]+[.][0-9]+[.][0-9]+?$/' }],
}[env];

/* Функция возвращает докер-тег образа, в зависимости от окружения */
local getTag(env) = {
 stage: '$CI_COMMIT_SHORT_SHA',
 preprod: '$CI_COMMIT_SHORT_SHA',
 prod: '$CI_COMMIT_TAG',
}[env];

/* Функция возвращает стандартизированное название докер-образа */
local getDestination(suffix=null) =
    'registry.company/web/' + std.join('-', ['$CI_PROJECT_NAME', suffix]);
```

```
/* Функция возвращает аргументы билда в зависимости от окружения */
local getBuildArgs(env, args=[]) = [
 item.key + '=' + if std.objectHas(item, env) then item[env] else item.value
 for item in args
];

/* Функция возвращает правила запуска пайплайна в зависимости от окружения */
local getRules(env) = {
 stage: [{ 'if': '$CI_PIPELINE_SOURCE == "web"' }],
 preprod: [{ 'if': '$CI_COMMIT_BRANCH == "master"' }],
 prod: [{ 'if': '$CI_COMMIT_TAG =~ /^v[0-9]+[.][0-9]+[.][0-9]+?$/' }],
}[env];

/* Функция возвращает докер-тег образа, в зависимости от окружения */
local getTag(env) = {
 stage: '$CI_COMMIT_SHORT_SHA',
 preprod: '$CI_COMMIT_SHORT_SHA',
 prod: '$CI_COMMIT_TAG',
}[env];

/* Функция возвращает стандартизированное название докер-образа */
local getDestination(suffix=null) =
    'registry.company/web/' + std.join('-', ['$CI_PROJECT_NAME', suffix]);
```

```
/* Функция возвращает аргументы билда в зависимости от окружения */
local getBuildArgs(env, args=[]) = [
 item.key + '=' + if std.objectHas(item, env) then item[env] else item.value
 for item in args
];

/* Функция возвращает правила запуска пайплайна в зависимости от окружения */
local getRules(env) = {
 stage: [{ 'if': '$CI_PIPELINE_SOURCE == "web"' }],
 preprod: [{ 'if': '$CI_COMMIT_BRANCH == "master"' }],
 prod: [{ 'if': '$CI_COMMIT_TAG =~ /^v[0-9]+[.][0-9]+[.][0-9]+?$/' }],
}[env];

/* Функция возвращает докер-тег образа, в зависимости от окружения */
local getTag(env) = {
 stage: '$CI_COMMIT_SHORT_SHA',
 preprod: '$CI_COMMIT_SHORT_SHA',
 prod: '$CI_COMMIT_TAG',
}[env];

/* Функция возвращает стандартизированное название докер-образа */
local getDestination(suffix=null) =
    'registry.company/web/' + std.join('-', ['$CI_PROJECT_NAME', suffix]);
```

```
{
  // Функции-хелперы
  get_build_args(env, variables):: getBuildArgs(env, variables),
  get_rules(env):: getRules(env),
  get_tag(env):: getTag(env),
  get_destination(suffix):: getDestination(suffix),

  // Стандартный флоу деплоя
  stages():: [
    'build',
    'deploy'
  ],

  // Функция сборки сервиса
  build(
    env,
    buildArgs=[],
    projectName='$CI_PROJECT_NAME',
    customFields={},
  )::
    someFunction(),

  // Функция деплоя сервиса
  deploy(
    env,
    projectName='$CI_PROJECT_NAME',
    customFields={},
  )::
    someFunction(),
}
```

```
{
 // Функции-хелперы
 get_build_args(env, variables):: getBuildArgs(env, variables),
 get_rules(env):: getRules(env),
 get_tag(env):: getTag(env),
 get_destination(suffix):: getDestination(suffix),

 // Стандартный флоу деплоя
 stages():: [
   'build',
   'deploy'
 ],

 // Функция сборки сервиса
 build(
   env,
   buildArgs=[],
   projectName='$CI_PROJECT_NAME',
   customFields={},
 )::
   someFunction(),

 // Функция деплоя сервиса
 deploy(
   env,
   projectName='$CI_PROJECT_NAME',
   customFields={},
 )::
   someFunction(),
}
```

```
{
 // Функции-хелперы
 get_build_args(env, variables):: getBuildArgs(env, variables),
 get_rules(env):: getRules(env),
 get_tag(env):: getTag(env),
 get_destination(suffix):: getDestination(suffix),

 // Стандартный флоу деплоя
 stages():: [
   'build',
   'deploy'
 ],

 // Функция сборки сервиса
 build(
   env,
   buildArgs=[],
   projectName='$CI_PROJECT_NAME',
   customFields={},
 )::
   someFunction(),

 // Функция деплоя сервиса
 deploy(
   env,
   projectName='$CI_PROJECT_NAME',
   customFields={},
 )::
   someFunction(),
}
```

API библиотеки на Jsonnet

```
local web = import 'web-ci/web.libsonnet';

{
 local projectName = '$CI_PROJECT_NAME',
 local env = 'stage',

 'build-stage': web.build(env, projectName, buildArgs),
 'deploy-stage': web.deploy(env, projectName),
}
```

Проблема

Теория

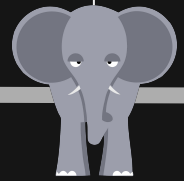
Собираем

Оптимизируем

Масштабируем

Советы

Выводы



Выводы

- Единый подход к CI упрощает жизнь
- Избавились от боли, связанной с разными конфигурациями CI
- Сэкономили время и нервы на настройке и поддержке
- Не нужно разбираться в тонкостях CI/CD
- Фокусируемся на фичах, а не инфре


JS
TS
ЗВУК

Демо

gitlab.com/web-ci

Телеграм

trash_js

❤ Оценить доклад

Вопросы?