

Космические скорости GitLab CI: от 20 минут к 40 секундам за месяц

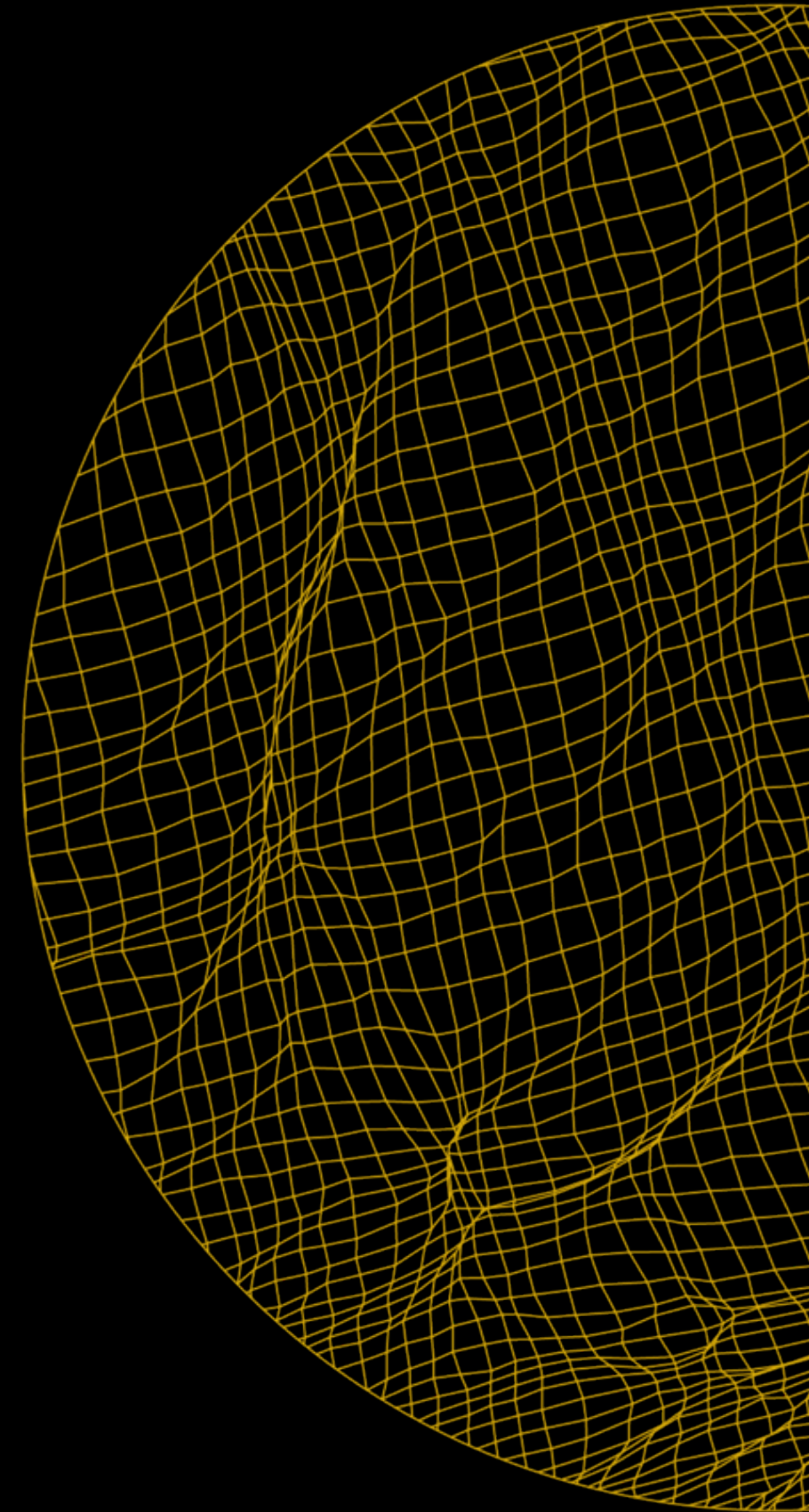


Иван
Малюгин

🏐 билайн



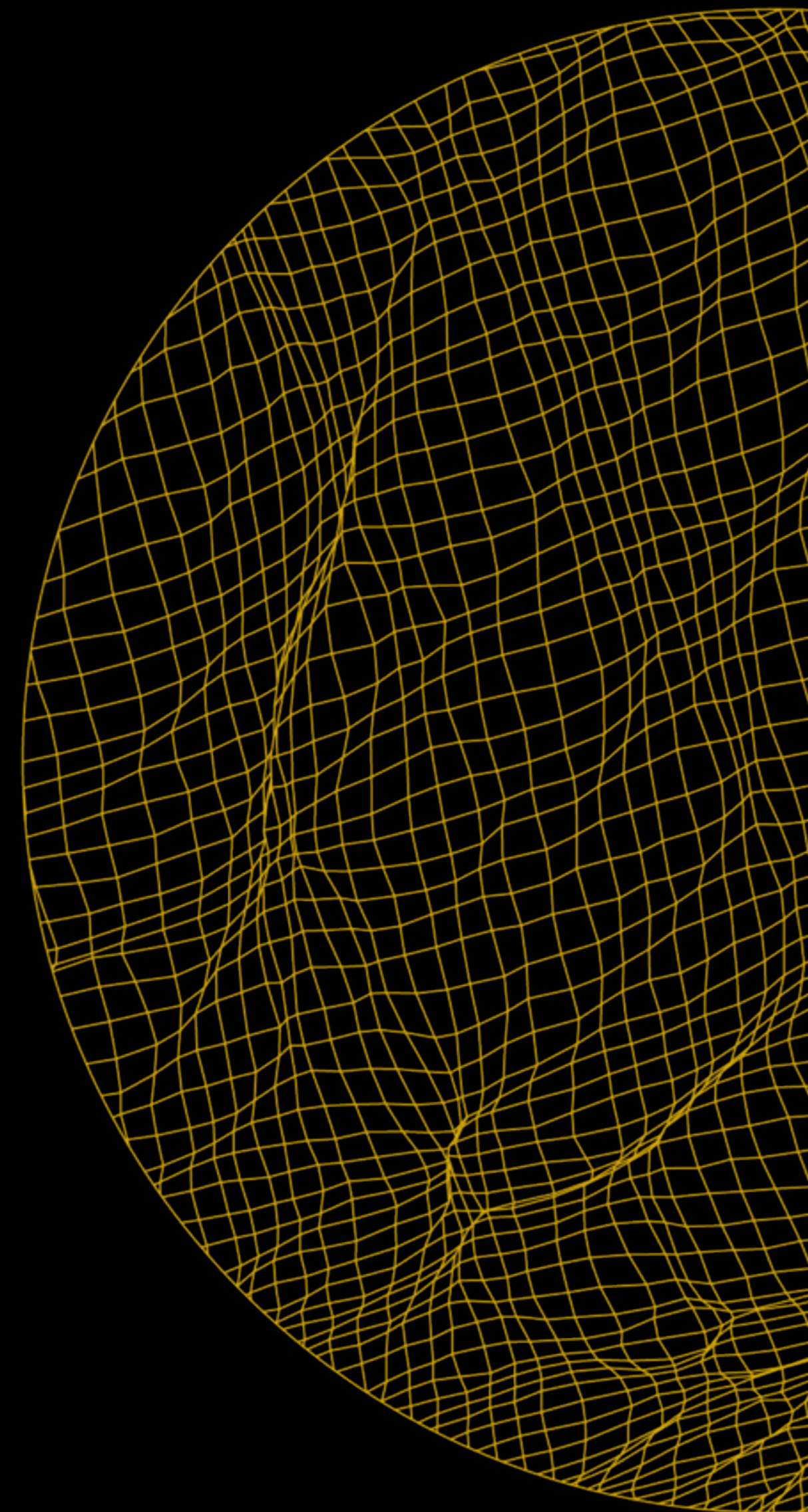
DIY-ракета



Источники:

- https://www.reddit.com/r/rocketry/comments/15b9ska/building_my_first_diy_rocket
- <https://tavika.ru/2013/01/handemade-rocket.html>

DIY-ракета



Источники:

- https://www.reddit.com/r/rocketry/comments/15b9ska/building_my_first_diy_rocket
- <https://tavika.ru/2013/01/handemade-rocket.html>

DIY-ракета

Ожидание

-50/70%



Реальность

-20%



Источники:

- https://www.reddit.com/r/rocketry/comments/15b9ska/building_my_first_diy_rocket
- <https://tavika.ru/2013/01/handemade-rocket.html>

DIY-ракета

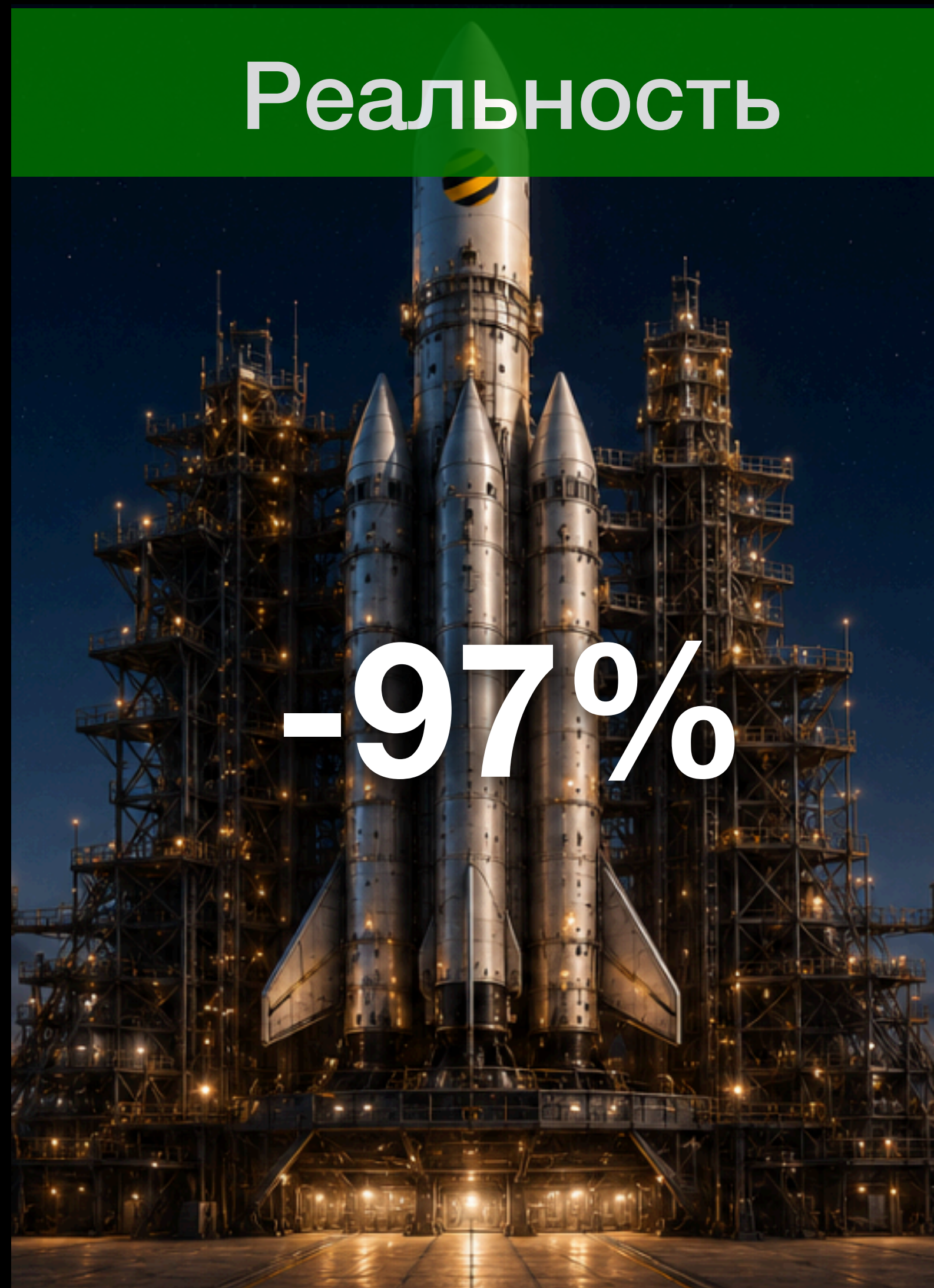
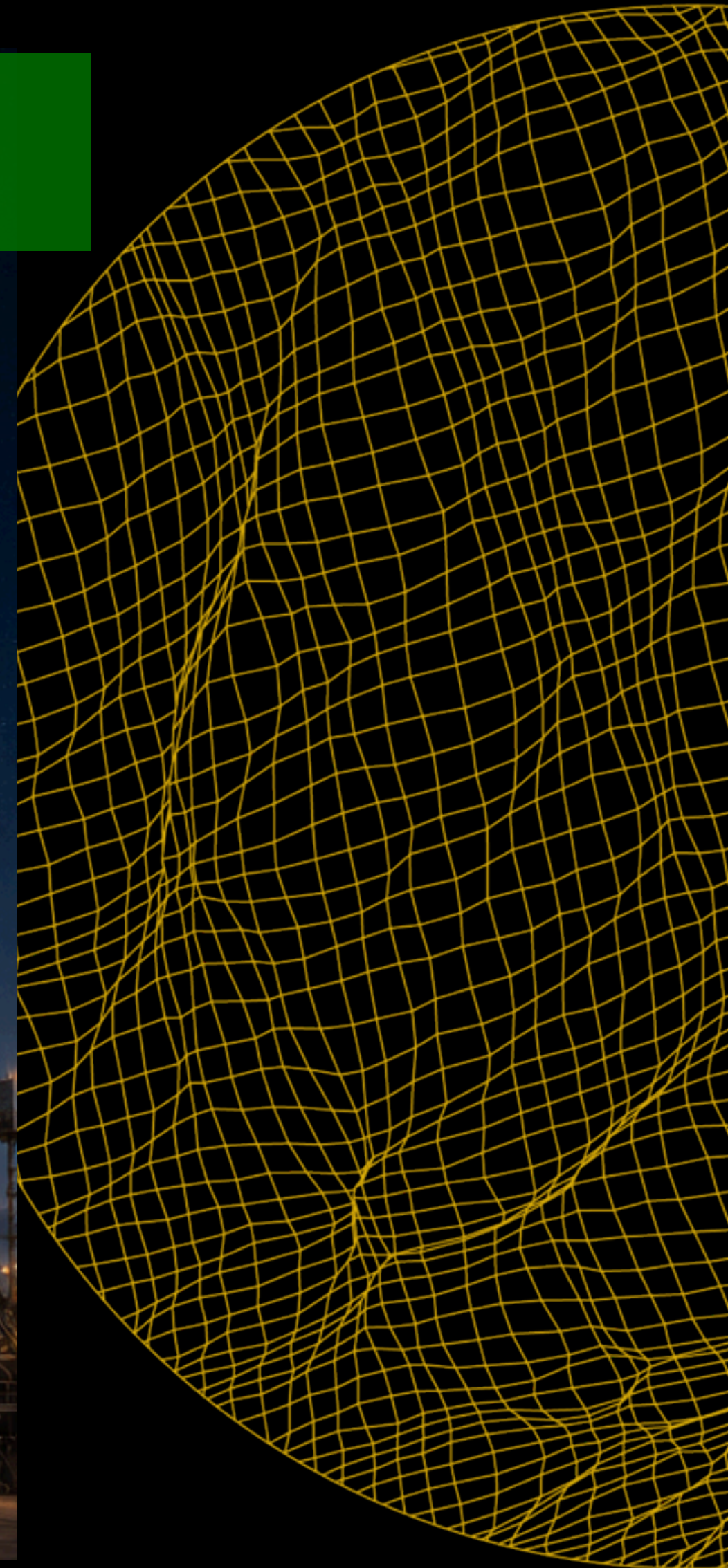
Ожидание

-50/70%



Реальность

-97%



Источники:

- https://www.reddit.com/r/rocketry/comments/15b9ska/building_my_first_diy_rocket
- <https://tavika.ru/2013/01/handemade-rocket.html>

Зачем мне это, я не DevOps?

- как микросервисы, так и статические билды

Зачем мне это, я не DevOps?

- как микросервисы, так и статические билды
- специфика сборки фронтов

Зачем мне это, я не DevOps?

- как микросервисы, так и статические билды
- специфика сборки фронтов
- растущий тренд на веб-мастеров

Зачем мне это, если я не DevOps?

- как микросервисы, так и статические билды
- специфика сборки фронтов
- растущий тренд на веб-мастеров
- поделиться со своим DevOps

Минорная правка

packages/platform-gui/src/pages/Editor/connected/NavButtons/LibraryButton.tsx

+1 -1



View file @ 665a8ca2

```
...    @@ -9,7 +9,7 @@ import ComponentsIcon from './assets/components.svg';
9      export default defineProps({
10       isInverse: true,
11       isGrayScale: true,
12       - title: 'Библиотека компонент',
12       + title: 'Библиотека компонентов',
13       slug: NavTabs.BLOCKS,
14       children: <ComponentsIcon />,
15     })(NavButton);
```

packages/platform-gui/src/pages/Editor/connected/NavTabs/index.jsx

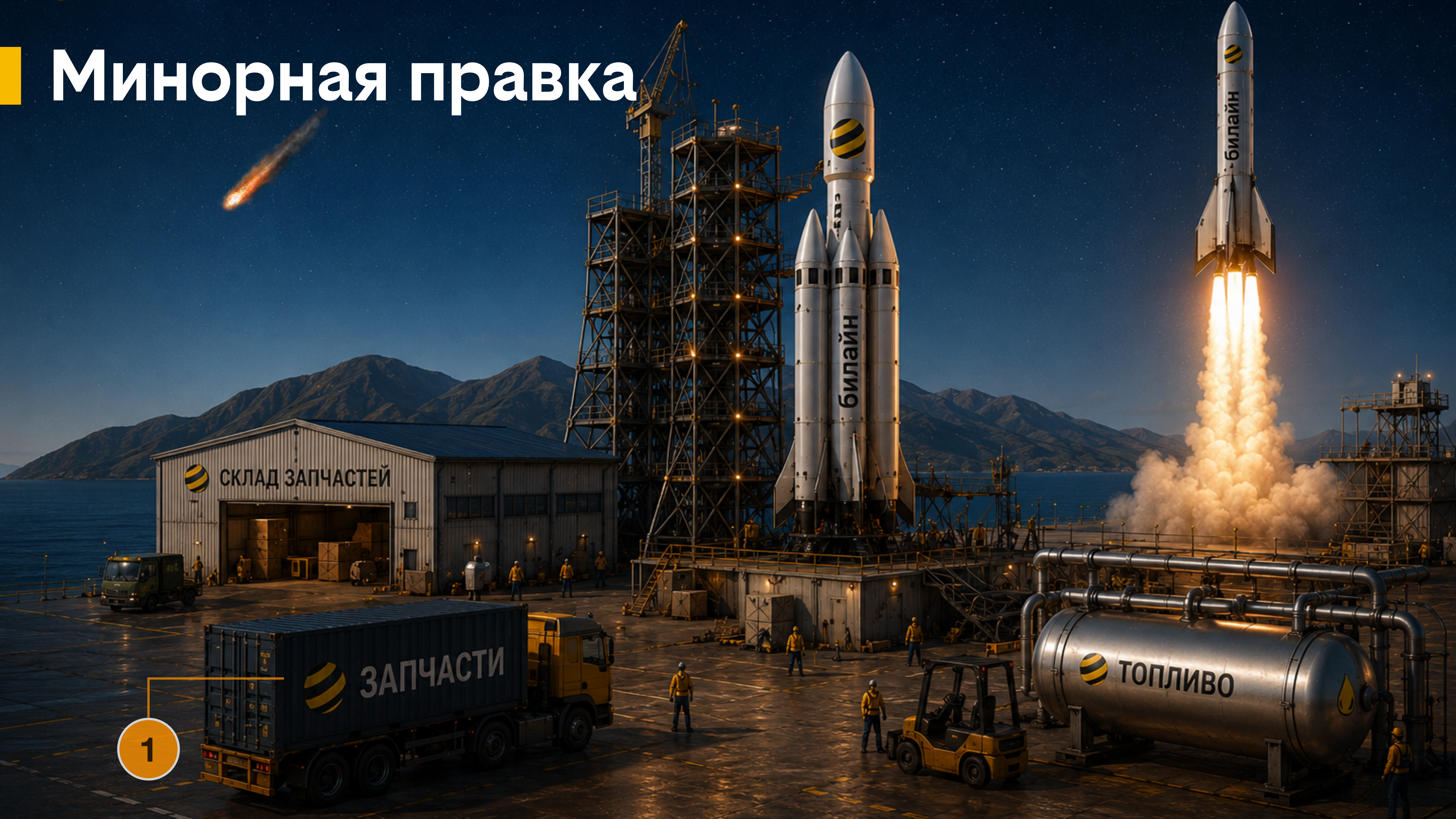
+1 -1



View file @ 665a8ca2

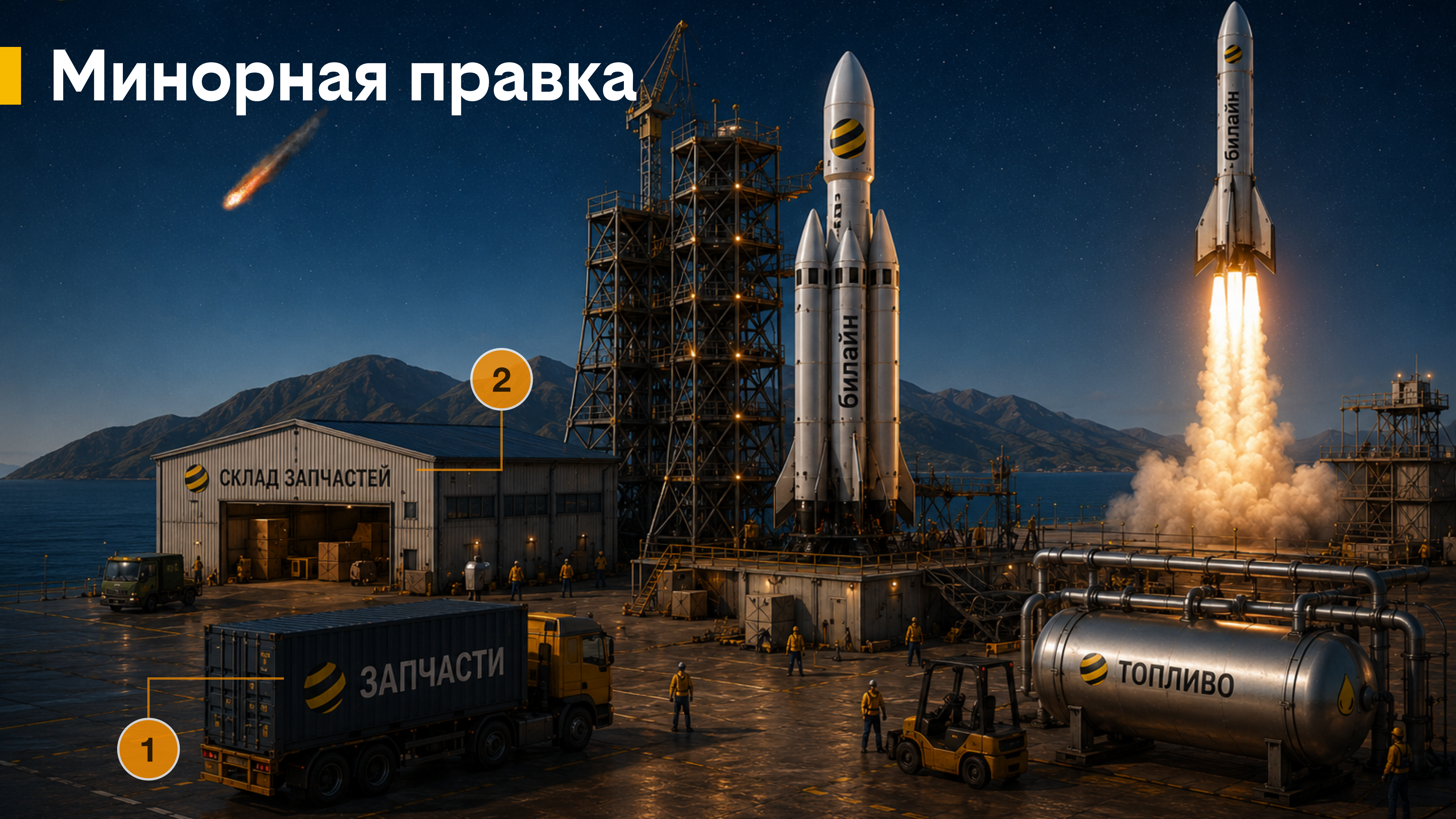
```
...    @@ -7,7 +7,7 @@ import { NavTab } from '../components';
7      import { LibraryPanel, NodesTreePanel, PropsPanel } from '../panels';
8
9      export const LibraryTab = defineProps({
10       - title: 'Библиотека компонент',
10       + title: 'Библиотека компонентов',
11       slug: NavTabs.BLOCKS,
12       renderChildren: () => <LibraryPanel />,
13     })(NavTab);
...
...
```


Минорная правка

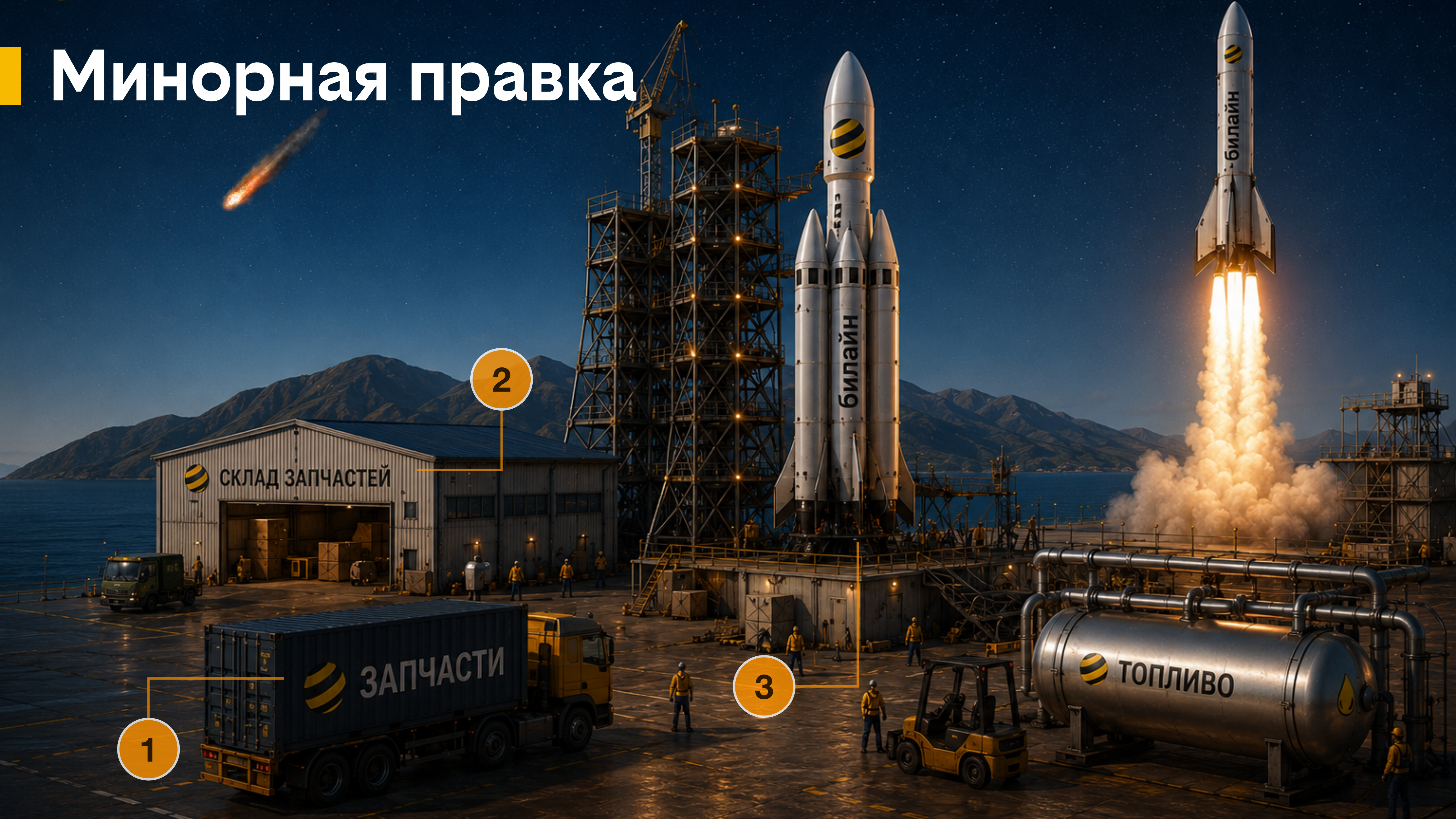


1

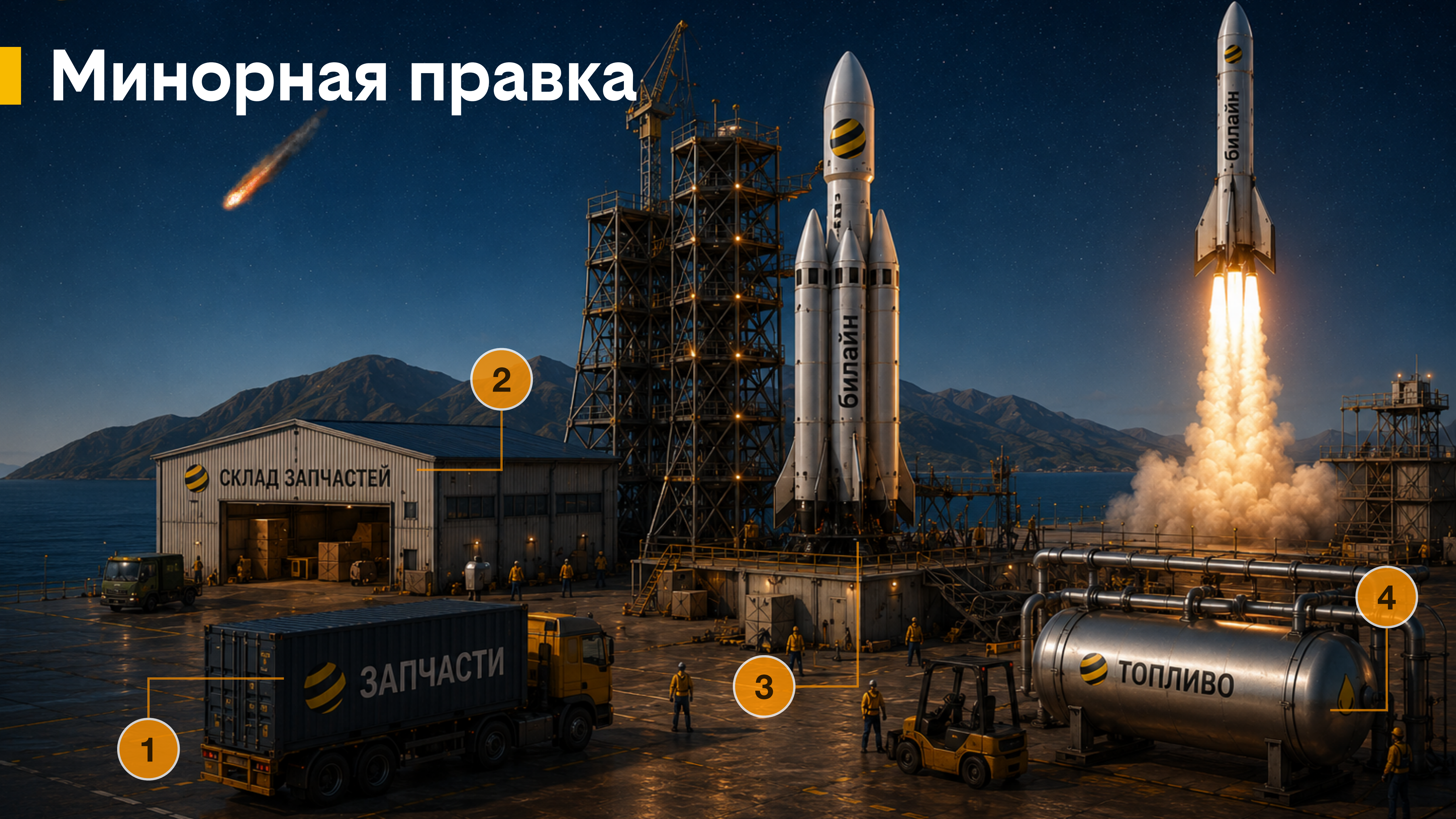
Минорная правка



Минорная правка



Минорная правка



Минорная правка



Минорная правка



Очень медленный пайплайн

<div>✓ Passed</div> <div>00:18:37 6 months ago</div>	Merge branch 'feature/LCP-1708' into 'dev' #2715826 dev branch			
<div>✓ Passed</div> <div>00:21:13 6 months ago</div>	Merge branch 'feature/LCP-1629' into 'dev' #2710258 dev branch			
<div>✓ Passed</div> <div>00:24:44 6 months ago</div>	Fix typo #2704254 dev branch			
<div>✓ Passed</div> <div>00:24:47 6 months ago</div>	Fix error while publishing, add proper err... #2701125 dev branch			
<div>✓ Passed</div> <div>00:18:33 6 months ago</div>	Merge branch 'main' into 'release' #2696184 release branch			
<div>✓ Passed</div> <div>00:18:18 6 months ago</div>	CSP-4468: fix security issue with react #2696181 main branch			

Очень медленный пайплайн

Fix typo

✓ Passed Created 6 months ago by **Малюгин Иван Александрович**, finished 6 months ago

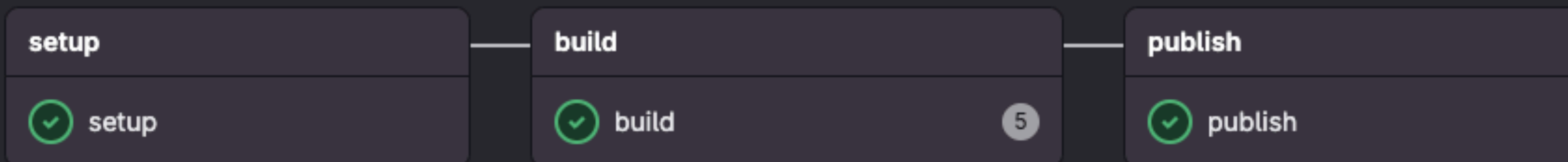
For commit `665a8ca2` 

1 related merge request: !256 Dev to load

branch  8 jobs ⌚ 24 minutes 44 seconds, queued for 1 seconds

Pipeline Jobs 8 Tests 0

Group jobs by Stage Job dependencies



Куда пропадает время

Итого: 23 мин

setup

8 мин

build

14 мин

publish

1 мин

Куда пропадает время

Итого: 23 мин

setup

8 мин

pod start 10 сек

install 7 мин

pack 40 сек

build

14 мин

pod start 10 сек

unpack 20 сек

buildah 13 мин

webpack 12 сек

publish

1 мин

■ Проблема всех фронтов #1



node_modules

- большой проект: 384 000 файлов, 2700 МБ
- средний проект: 86 000 файлов, 640 МБ
- минимальный проект: 35 000 файлов, 190 МБ

Беспокойные `node_modules`

- скачиваются в кэш
- копируются в проект
- копируются в контейнер
- упаковываются в артефакты
- распаковываются из артефактов

Стандартные решения

- использовать yarn cache dir
- накидывать ресурсы в раннер
- упаковывать в tar при передаче
- объединять install & build

Стандартные решения

Итого: ~~-5 мин~~
23 мин

setup

8 мин

pod start 10 сек

install ~~7 мин~~ 2 мин

pack 40 сек

build

14 мин

pod start 10 сек

unpack 20 сек

buildah 13 мин

webpack 12 сек

publish

1 мин

Стандартные решения

Итого: 18 мин

setup

3 мин

pod start 10 сек

install 2 мин

pack 40 сек

build

14 мин

pod start 10 сек

unpack 20 сек

buildah 13 мин

webpack 12 сек

publish

1 мин

node_modules

- ~~скачиваем в кэш~~
- копируем в проект
- копируем в контейнер
- упаковываем артефакты
- распаковываем артефакты

Тем временем локально

- скачиваем в кэш
- копируем в проект
- копируем в контейнер
- упаковываем артефакты
- распаковываем артефакты

Тем временем локально

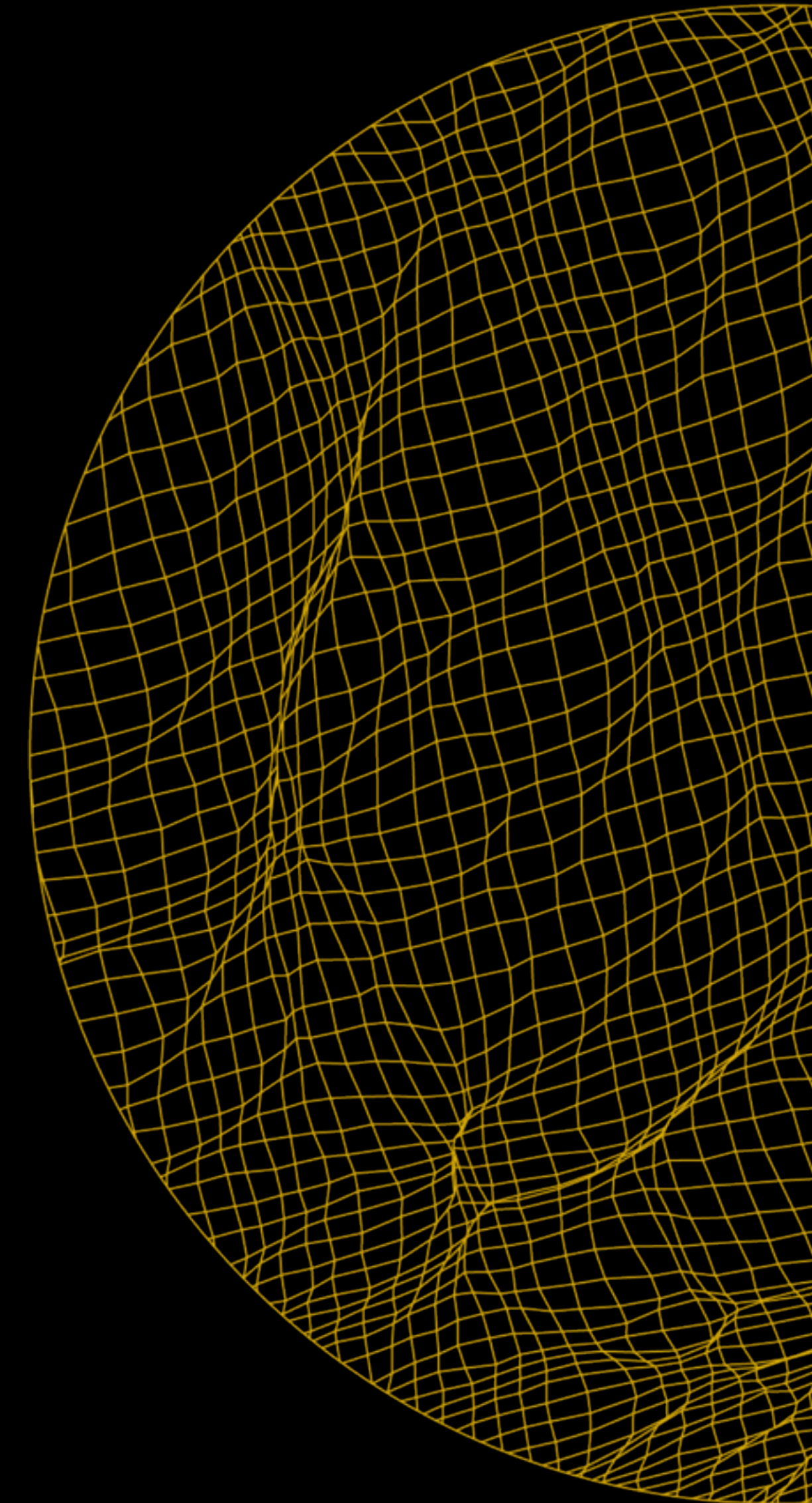
Итого: <1 мин

install ■ 1-20 сек

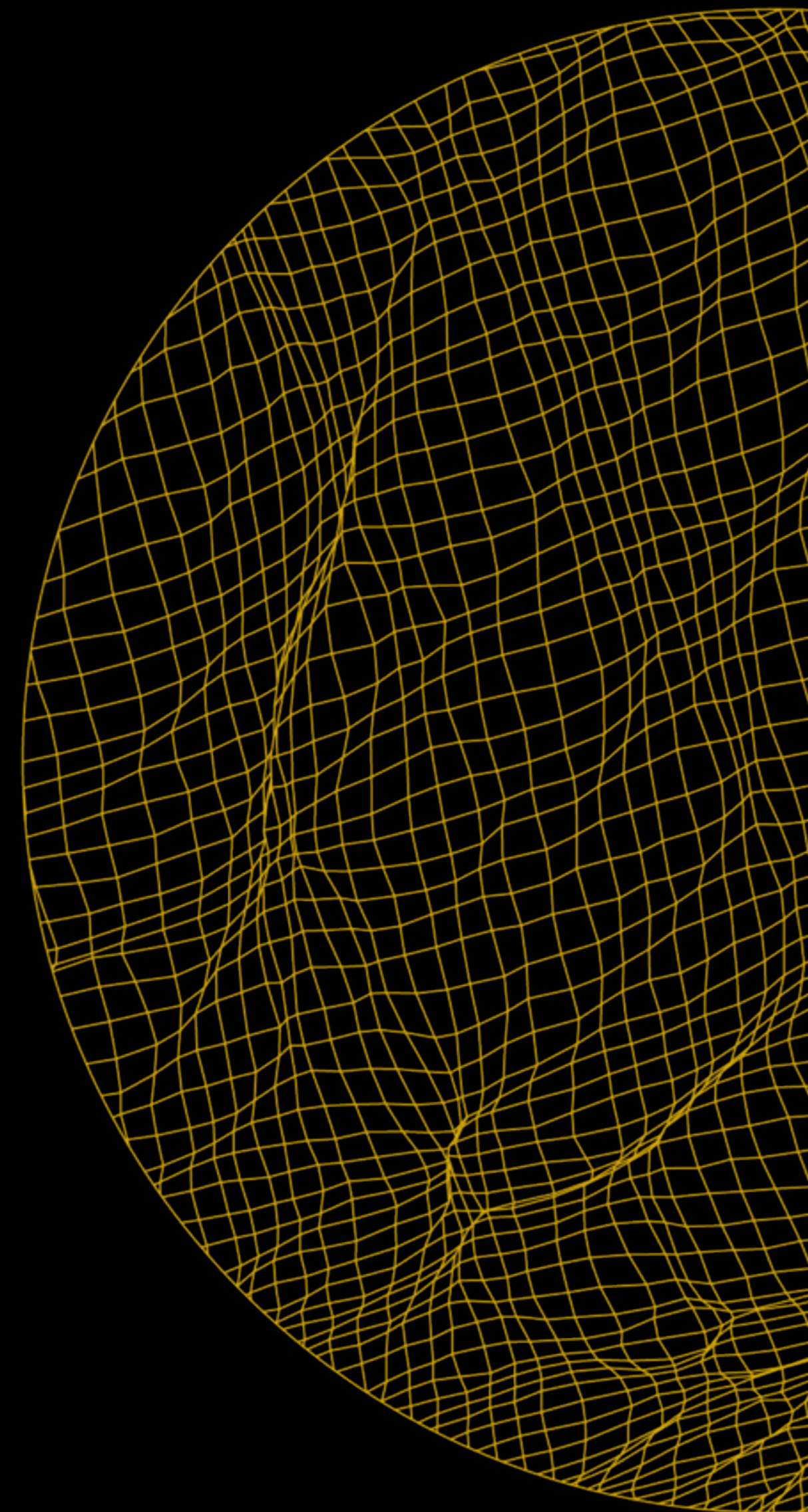
build ■ 12 сек

publish ■ 30 сек

Свой сервис взамен gitlab



Последний шанс, gitlab



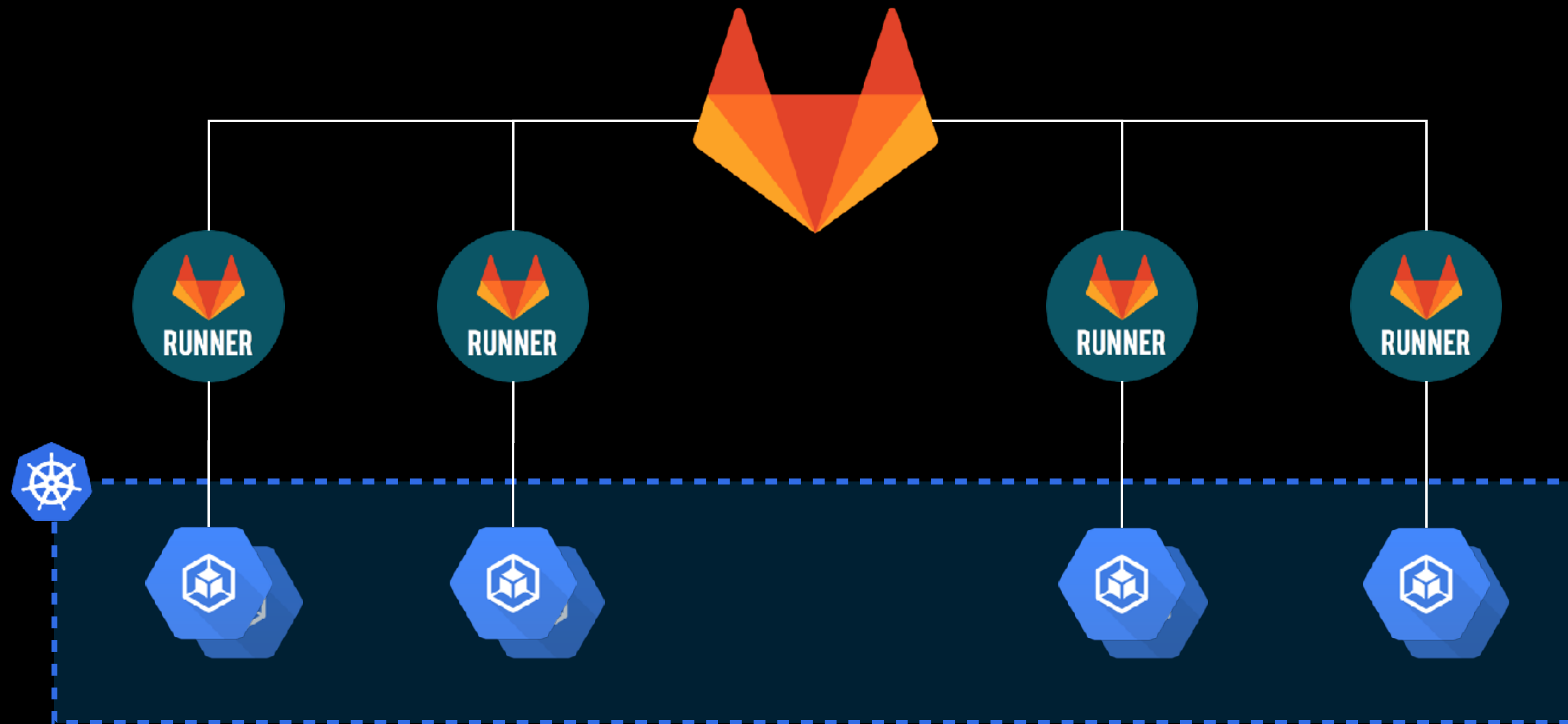
Часть I.

Рабочие пространства

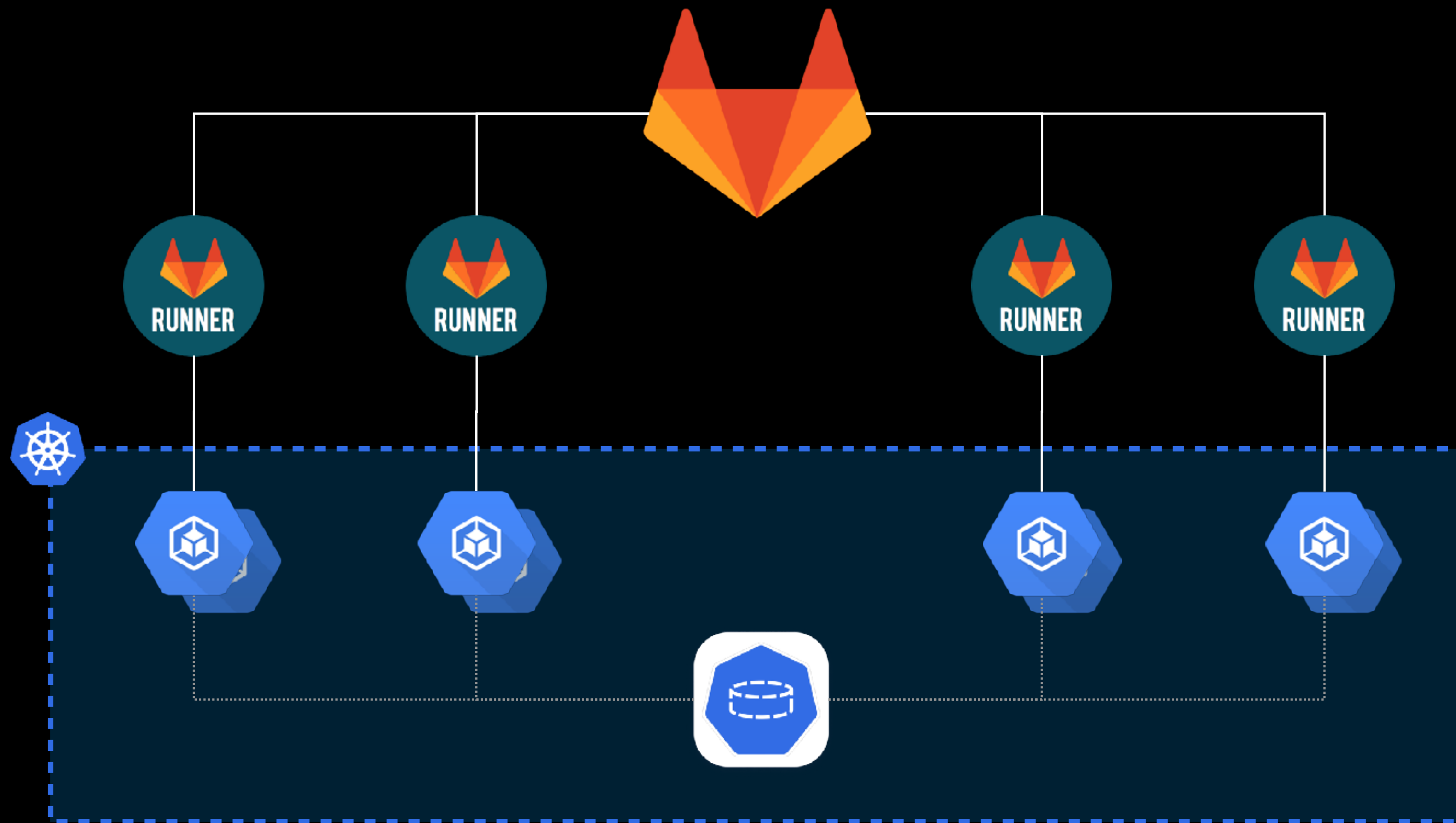
Переиспользуем node_modules



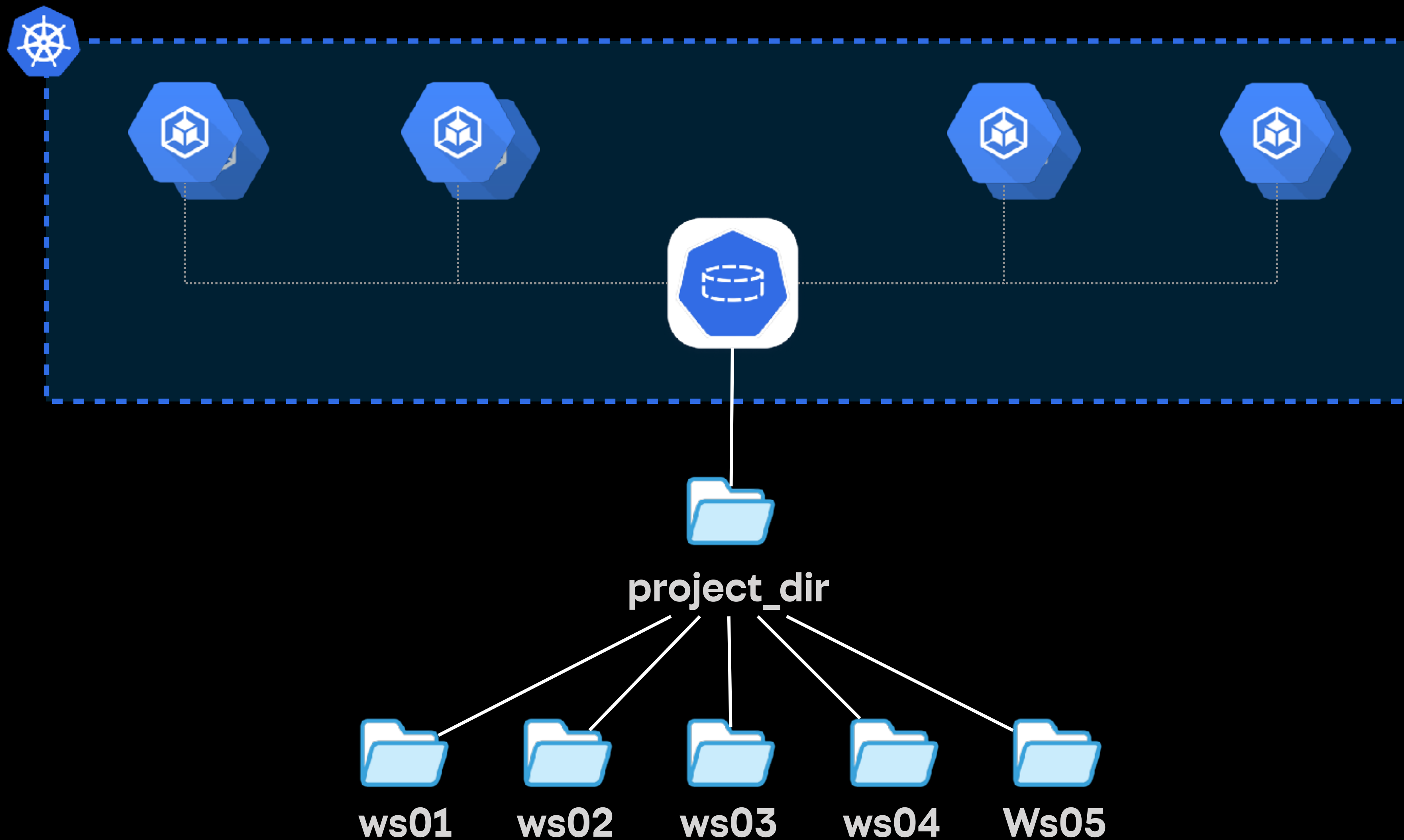
Как устроены gitlab-runner'ы



Как устроены gitlab-runner'ы



Переиспользуем workspace-папки



Из чего состоит решение

- фиксированные рабочие папки (workspaces)

Из чего состоит решение

- фиксированные рабочие папки (workspaces)
- lockfile для фиксации кто занял проект и когда

Из чего состоит решение

- фиксированные рабочие папки (workspaces)
- lockfile для фиксации кто занял проект и когда
- набор bash-скриптов ws-acquire, ws-release, ws-init

Что дало решение

-3,5 мин
Итого: 18 мин

setup

3 мин

pod start

10 сек

1-20 сек

install

2 мин

pack

40 сек

build

14 мин

pod start

10 сек

unpack

20 сек

buildah

13 мин

webpack

12 сек

publish

-20 сек

1 мин

Что дало решение

Итого: 14,5 мин

setup 20 сек

pod start 10 сек

install 1-20 сек

build 13 мин

pod start 10 сек

buildah 13 мин

webpack 12 сек

publish 40 сек

Вау эффект

- срезали 2 - 3 минуты со всех пайплайнов
- по другим проектам — 40 - 50%

Совсем немного минусов

- кончатся папки

Совсем немного минусов

- кончатся папки
- временные папки

Совсем немного минусов

- кончатся папки
- временные папки
- долго удалять

Совсем немного минусов

- кончатся папки
- временные папки
- долго удалять
- lock при падении job'ов

Совсем немного минусов

- кончатся папки
- временные папки
- долго удалять
- lock при падении job'ов
- артефакты нестабильных сборок

Совсем немного минусов

- кончатся папки
- временные папки
- долго удалять
- lock при падении job'ов
- артефакты нестабильных сборок
- большие изменения в `node_modules`

Совсем немного минусов

- кончатся папки
- временные папки
- долго удалять
- lock при падении job'ов
- артефакты нестабильных сборок
- большие изменения в `node_modules`
- потребление диска

Совсем немного минусов

- кончатся папки
- временные папки
- долго удалять
- lock при падении job'ов
- артефакты нестабильных сборок
- большие изменения в `node_modules`
- потребление диска
- нагрузка на диск

Совсем немного минусов

- кончатся папки
- временные папки
- долго удалять
- lock при падении job'ов
- артефакты нестабильных сборок
- большие изменения в `node_modules`
- потребление диска
- нагрузка на диск
- много кода в коммунальных скриптах

Совсем немного минусов

- кончатся папки
- временные папки
- долго удалять
- lock при падении job'ов
- артефакты нестабильных сборок
- большие изменения в `node_modules`
- потребление диска
- нагрузка на диск
- много кода в коммунальных скриптах
- сложнее отладка

Совсем немного минусов

- кончатся папки
- временные папки
- долго удалять
- lock при падении job'ов
- артефакты нестабильных сборок
- большие изменения в node_modules
- потребление диска
- нагрузка на диск
- много кода в коммунальных скриптах
- сложнее отладка
- документация

Совсем немного минусов

- кончатся папки
- временные папки
- долго удалять
- lock при падении job'ов
- артефакты нестабильных сборок
- большие изменения в `node_modules`
- потребление диска
- нагрузка на диск
- много кода в коммунальных скриптах
- сложнее отладка
- документация
- растёт энтропия

■ Всё в порядке



Внедряем во все проекты



Проблема #2

Итого: 14,5 мин

setup

20 сек

pod start 10 сек

install 1-20 сек

build

pod start 10 сек

buildah

13 мин

13 мин

webpack

12 сек

publish

40 сек

Часть II

Боли контейнеризации

Сборка контейнера





01. Основы контейнерной сборки

- Процесс сборки декларативно задается через Dockerfile

```
# Dockerfile
```

```
FROM <base-image>
```

```
ENV <key>=<value>
```

```
COPY <src> <dst>
```

```
RUN <command>
```




01. Основы контейнерной сборки

- Процесс сборки декларативно задается через Dockerfile
- Каждая RUN/COPY-создает новый неизменяемый слой

```
# Dockerfile
FROM node:24

ENV NODE_ENV=production
COPY . .
RUN yarn install
```




01. Основы контейнерной сборки

- Процесс сборки декларативно задается через Dockerfile
- Каждая RUN/COPY-создает новый неизменяемый слой
- Файлы вне слоёв OverlayFS нельзя использовать напрямую — их необходимо копировать.

```
# Dockerfile
FROM node:24

ENV NODE_ENV=production
COPY . .
RUN yarn install
```


Текущий процесс



project_dir



node_modules



project_dir



node_modules

Текущий процесс



■ Не копируем node_modules

- принимаем как аксиому
- workspace-решение больше не подходит
- установку переносим в контейнер
- нужен multi-stage build



02. Dockerfile multi-stage build

- Dockerfile может содержать несколько блоков FROM (стейдж)

```
# Dockerfile
FROM node:24
...

FROM base
...
```




02. Dockerfile multi-stage build

- Dockerfile может содержать несколько блоков FROM (стейдж)
- Стейджам можно давать имена при помощи ключевого слова "as"

```
# Dockerfile
FROM node:24 as base
...

FROM base as build
...
```




02. Dockerfile multi-stage build

- Dockerfile может содержать несколько блоков FROM (стейдж)
- Стейджам можно давать имена при помощи ключевого слова "as"
- Имя стейджа можно использовать в последующих FROM или в операции COPY --from

```
# Dockerfile
FROM node:24 as base
...
FROM base as build
...
FROM node:24
COPY --from=build
```


Хорошие практики

node:alpine

yarn install (dev)

Хорошие практики

node:alpine

yarn install (dev)

yarn build

Хорошие практики

node:alpine

yarn install (dev)

yarn build

node:alpine

yarn install --production

Хорошие практики

node:alpine

yarn install (dev)

yarn build

node:alpine

yarn install --production

final-image

Хорошие практики

node:alpine

yarn install (dev)

yarn build

node:alpine

yarn install --production

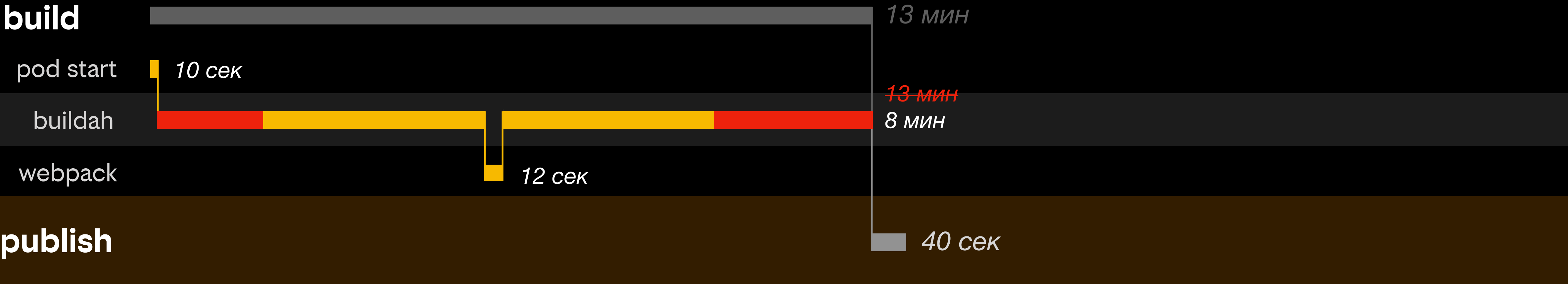
final-image

Посмотреть код:



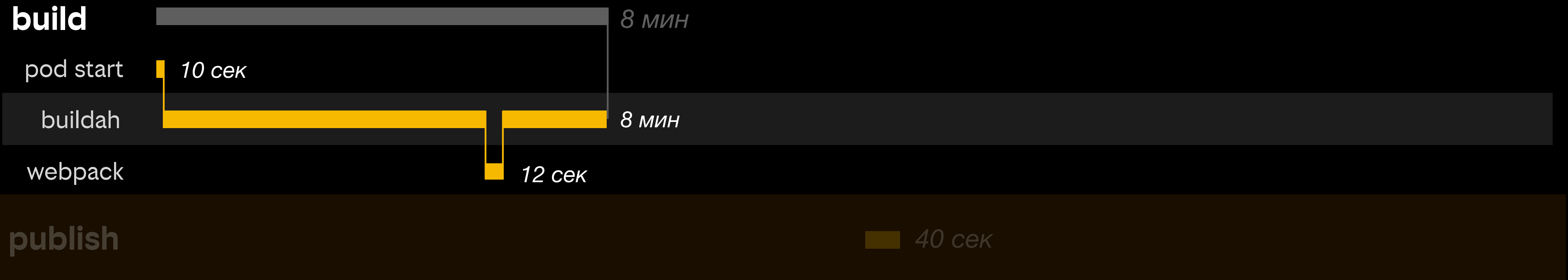
Что из этого вышло

-5 мин
Итого: 14,5 мин

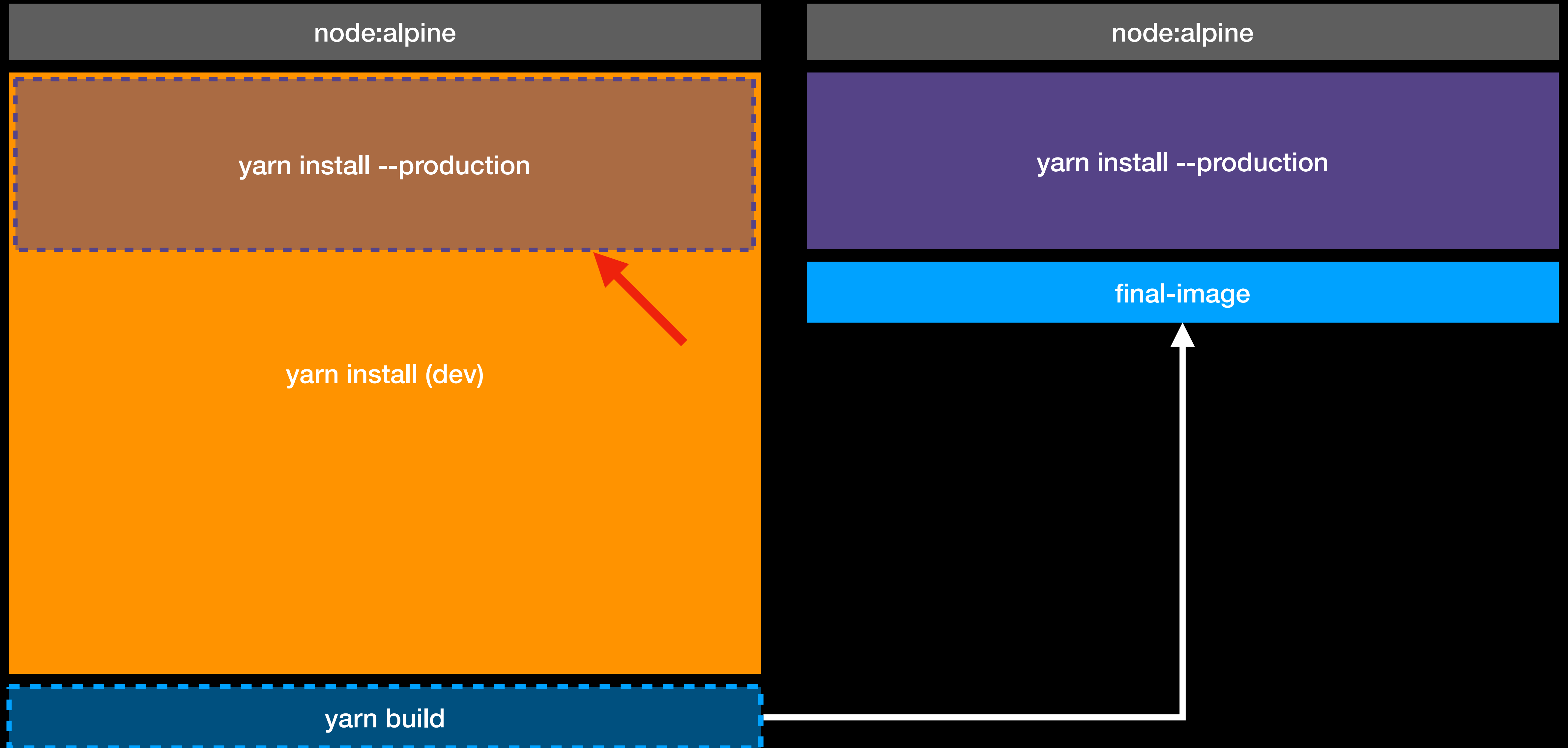


Что из этого вышло

Итого: 8,5 мин



Проблема



Лучшие практики

node:alpine

yarn install --production

yarn install (dev)

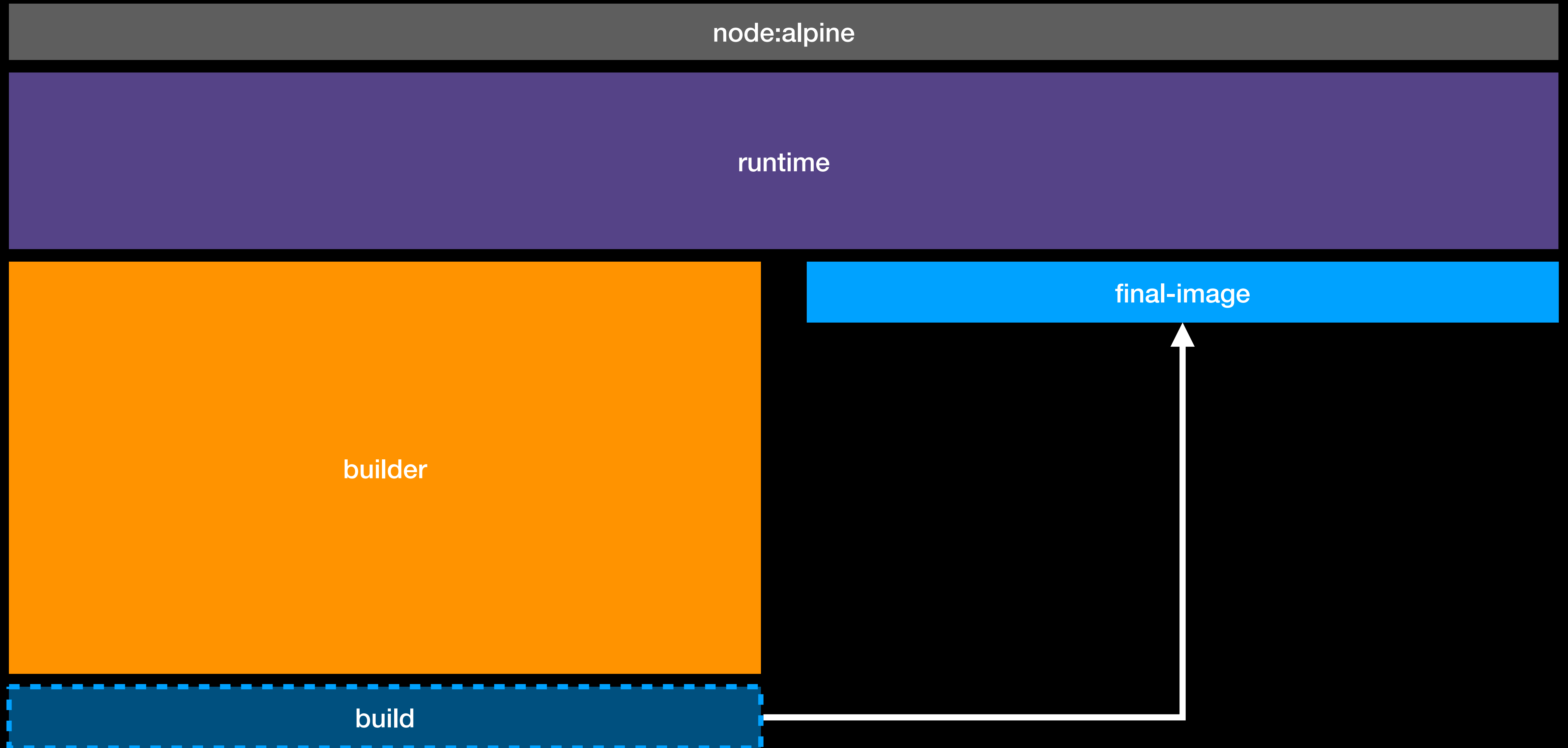
yarn build

node:alpine

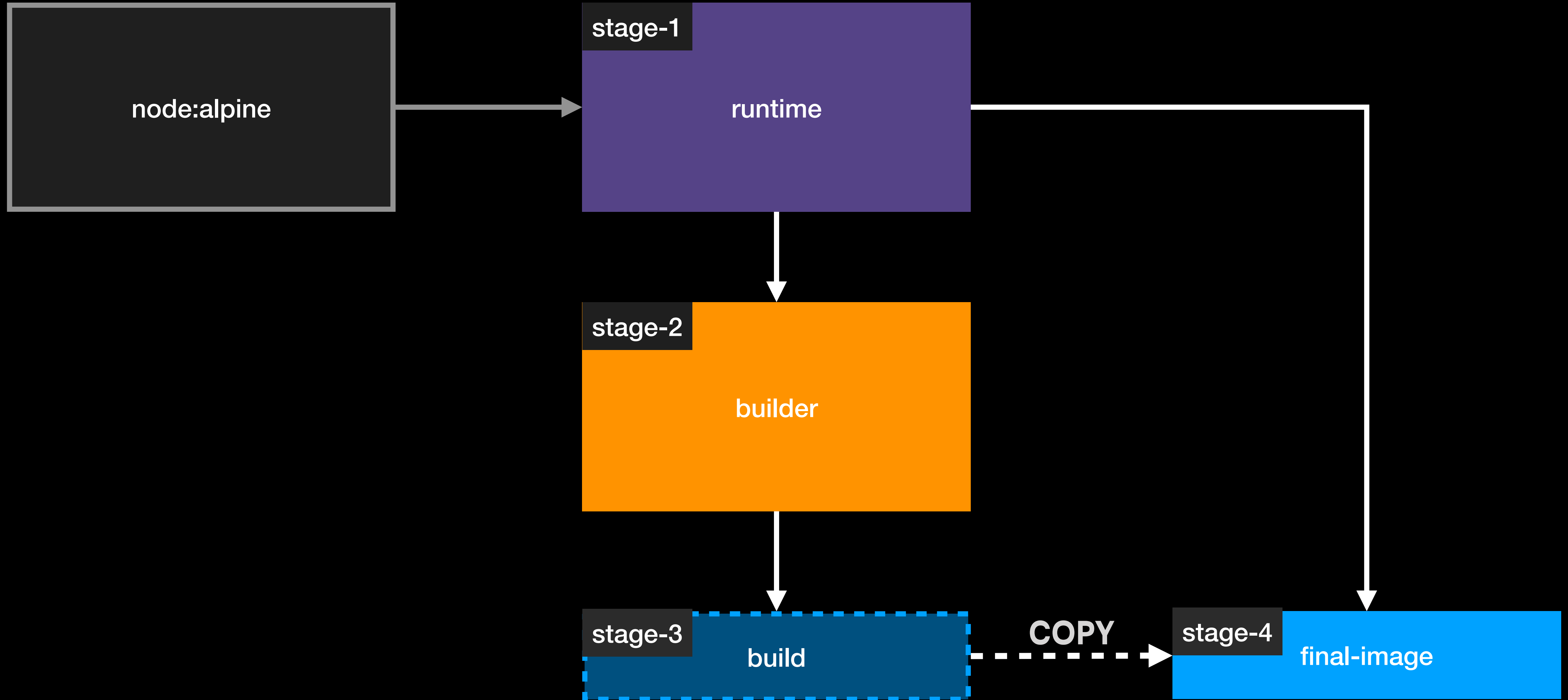
yarn install --production

final-image

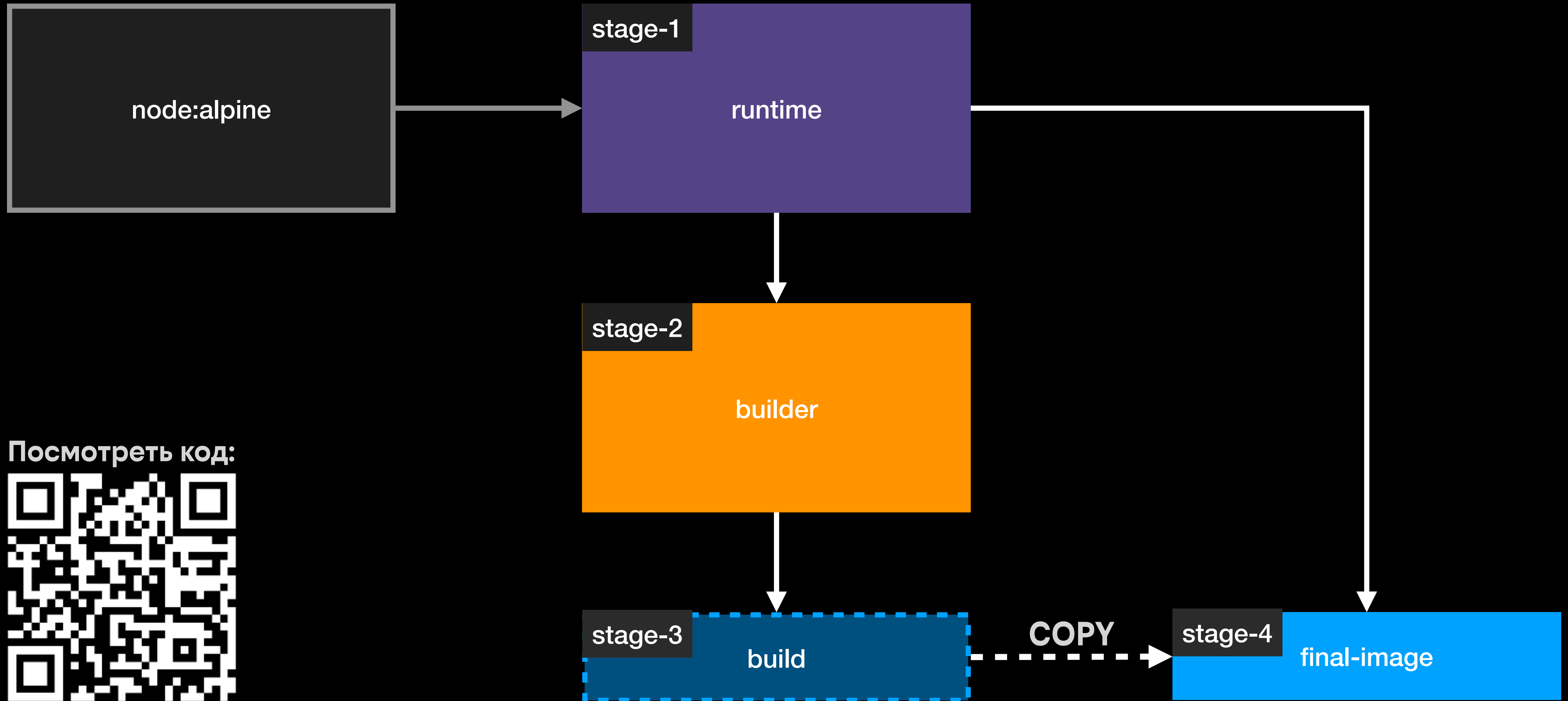
Переиспользуем runtime



Multi-stage build



Multi-stage build



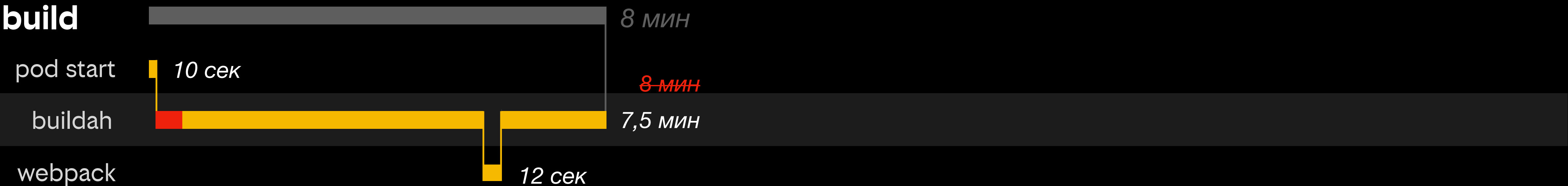
Посмотреть код:





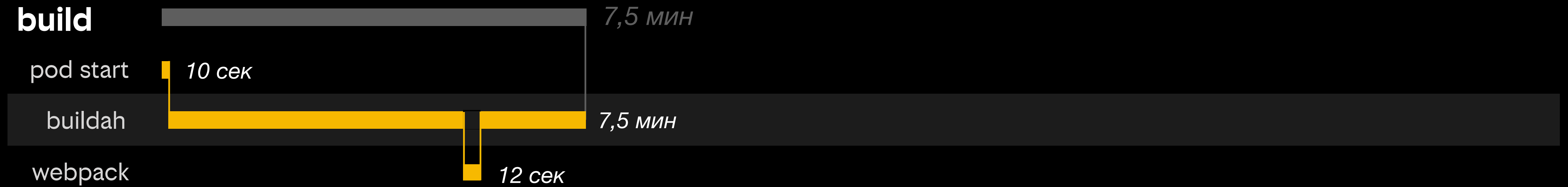
Незначительное улучшение

Итого: ~~8 мин~~ **-30 сек**
8,5 мин



Незначительное улучшение

Итого: 8 мин



Результат



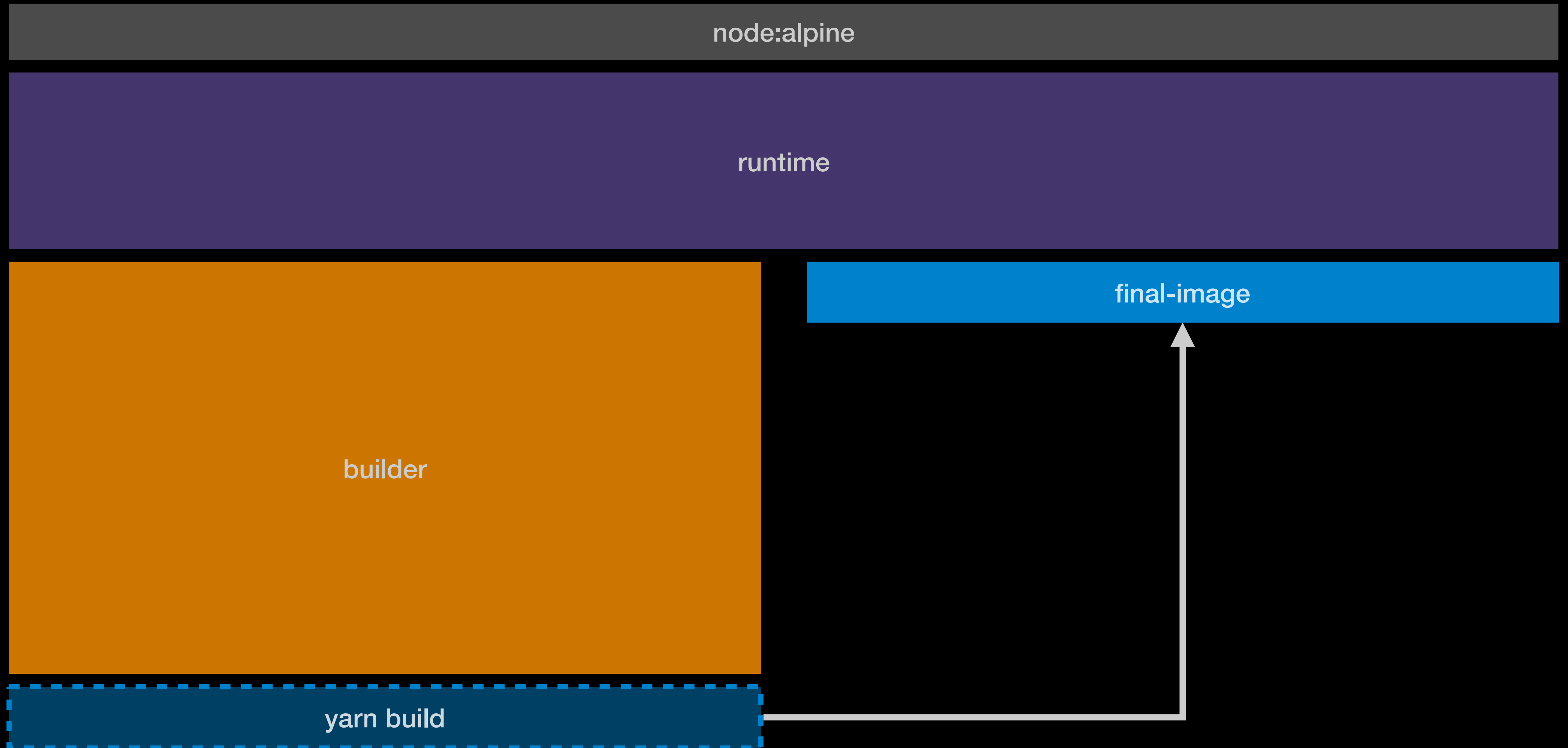
Часть III

Рабочие контейнеры

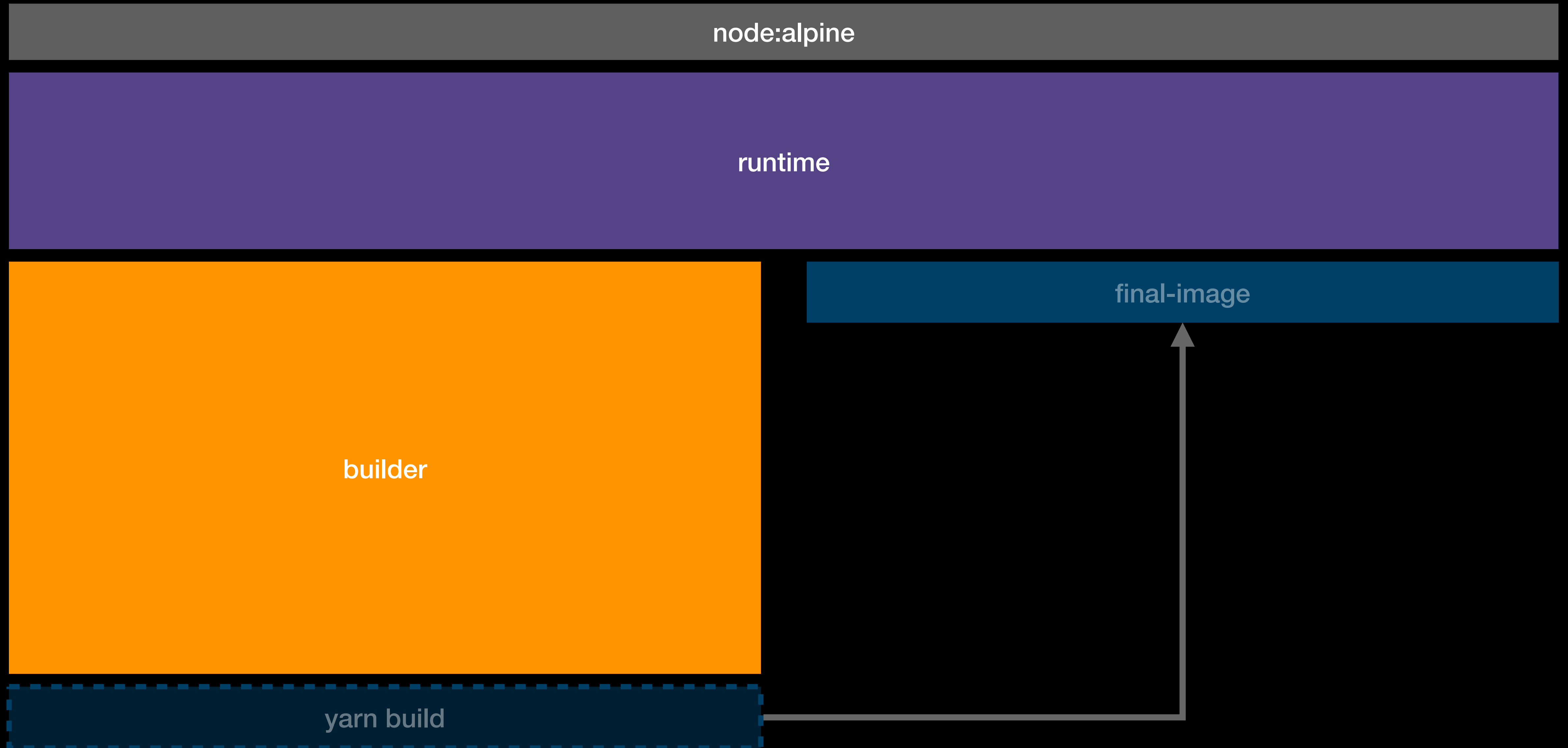
Адаптация ws-директорий



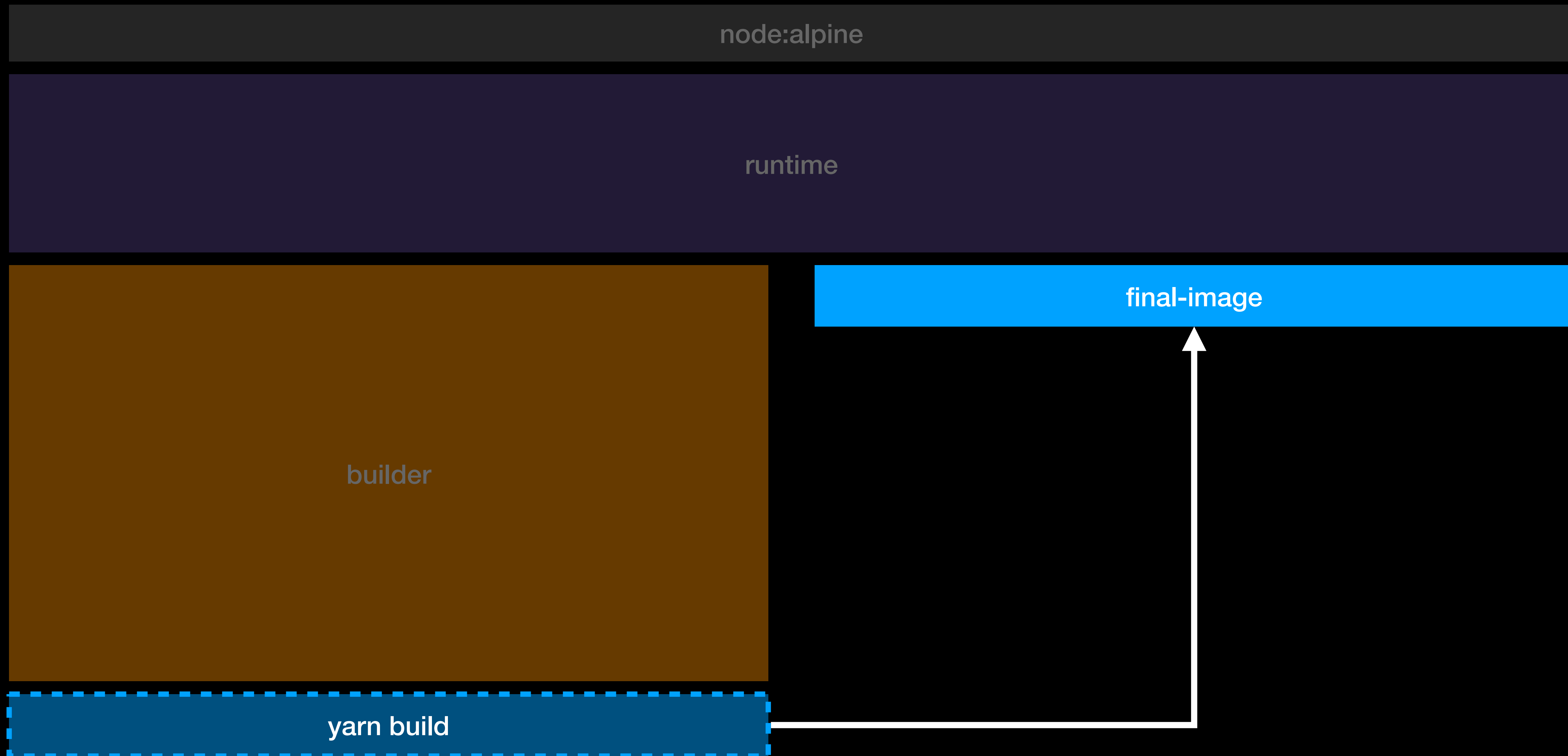
Разделяем пайплайн на две части



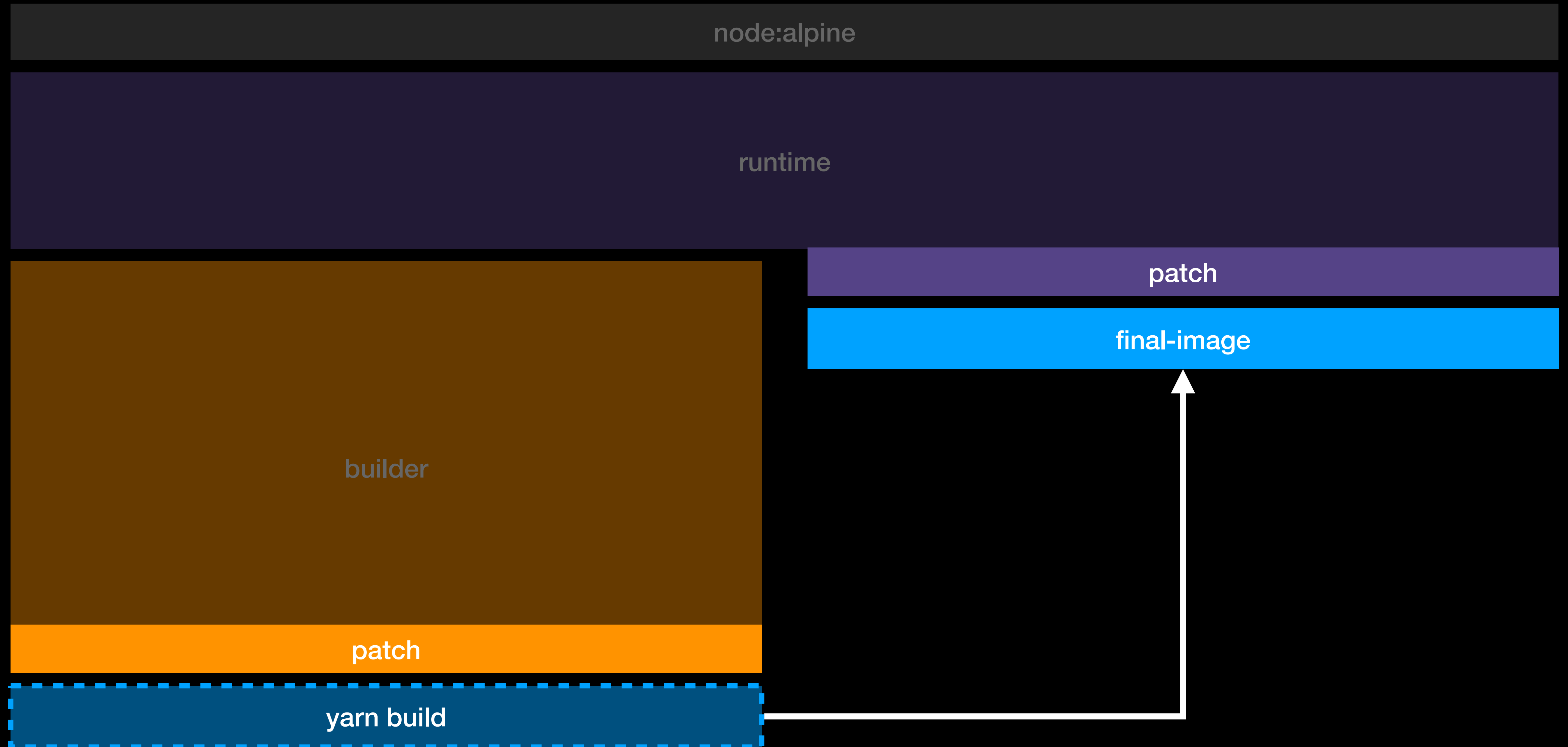
Тяжелый prebuild



Легкий build



Обновление зависимостей





03. Динамические multi-stage сборки

```
# Dockerfile
```

```
FROM node:24 as base
```

```
...
```

```
FROM base as build
```

```
...
```

```
FROM node:24
```

```
COPY --from=build
```




03. Динамические multi-stage сборки

- Все популярные сборщики позволяют при вызове указать финальный образ через --target

```
# Dockerfile
FROM node:24 as base
...

FROM base as build
...

FROM node:24

COPY --from=build
```




03. Динамические multi-stage сборки

- Все популярные сборщики позволяют при вызове указать финальный образ через `--target`
- Кроме того, название базового образа в `FROM` можно задавать динамическим аргументом `ARG`

```
# Dockerfile  
  
ARG IMAGE  
FROM $IMAGE as base  
...  
  
FROM base as build  
...  
  
FROM node:24  
  
COPY --from=build
```



03. Динамические multi-stage сборки

- Все популярные сборщики позволяют при вызове указать финальный образ через `--target`
- Кроме того, название базового образа в `FROM` можно задавать динамическим аргументом `ARG`
- `ARG` задается при вызове сборки, а также принимает значение по умолчанию

```
# Dockerfile
ARG IMAGE=node:24
FROM $IMAGE as base
...

FROM base as build
...

FROM node:24

COPY --from=build
```




03. Динамические multi-stage сборки

- Все популярные сборщики позволяют при вызове указать финальный образ через `--target`
- Кроме того, название базового образа в `FROM` можно задавать динамическим аргументом `ARG`
- `ARG` задается при вызове сборки, а также принимает значение по умолчанию
- Сборщик проанализирует `Dockerfile` и выполнит только нужные шаги, опустив все лишнее

```
# Dockerfile

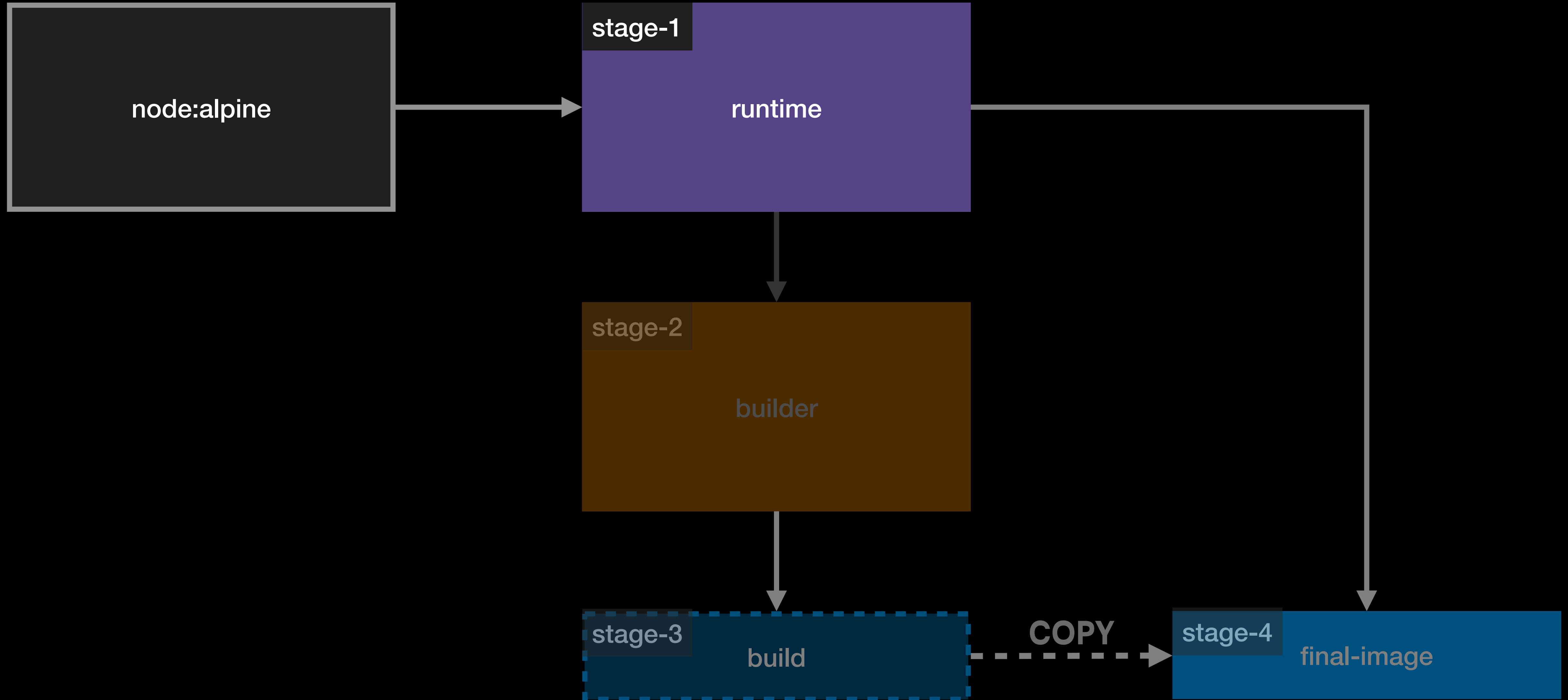
ARG IMAGE=node:24
FROM $IMAGE as base
...

FROM base as build
...

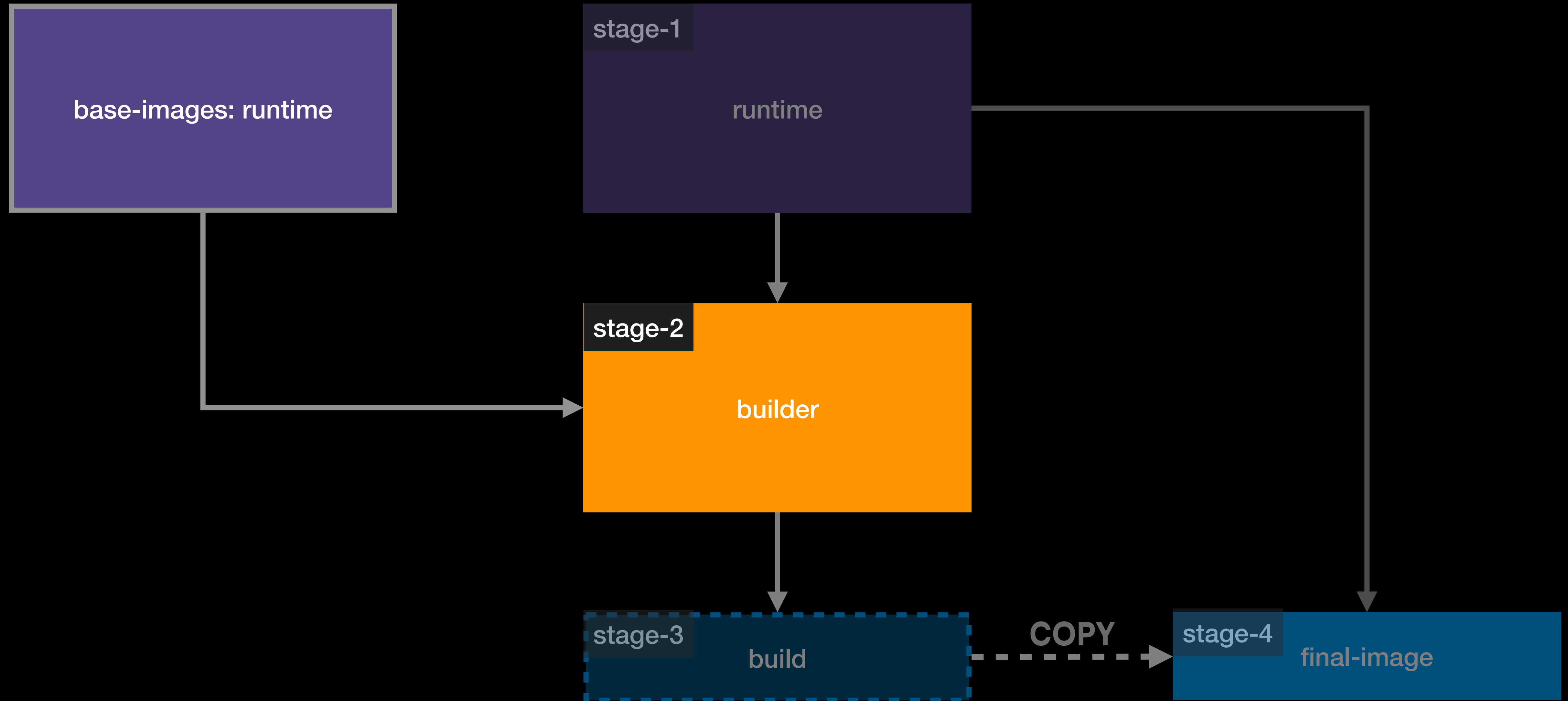
FROM node:24

COPY --from=build
```

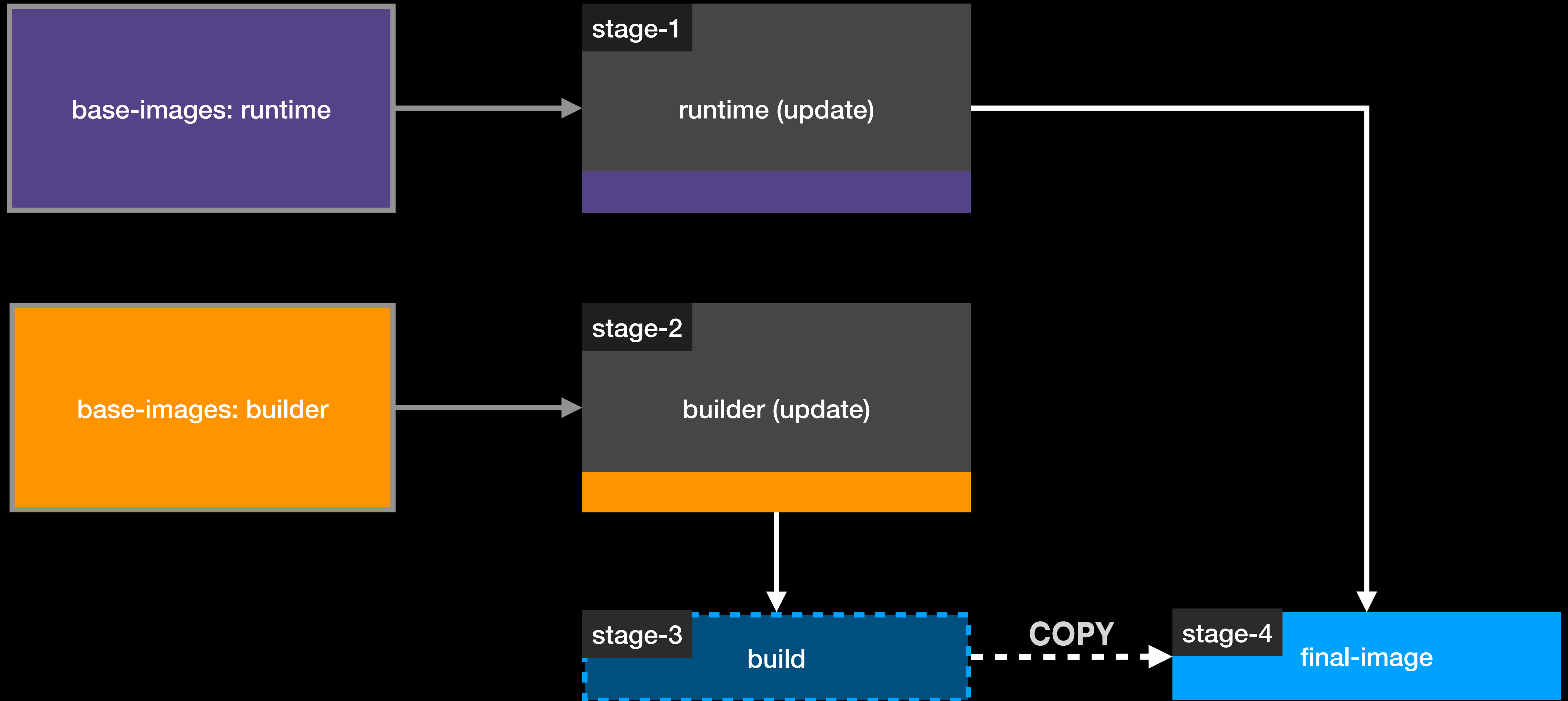
Multi-stage prebuild - runtime



Multi-stage prebuild - builder



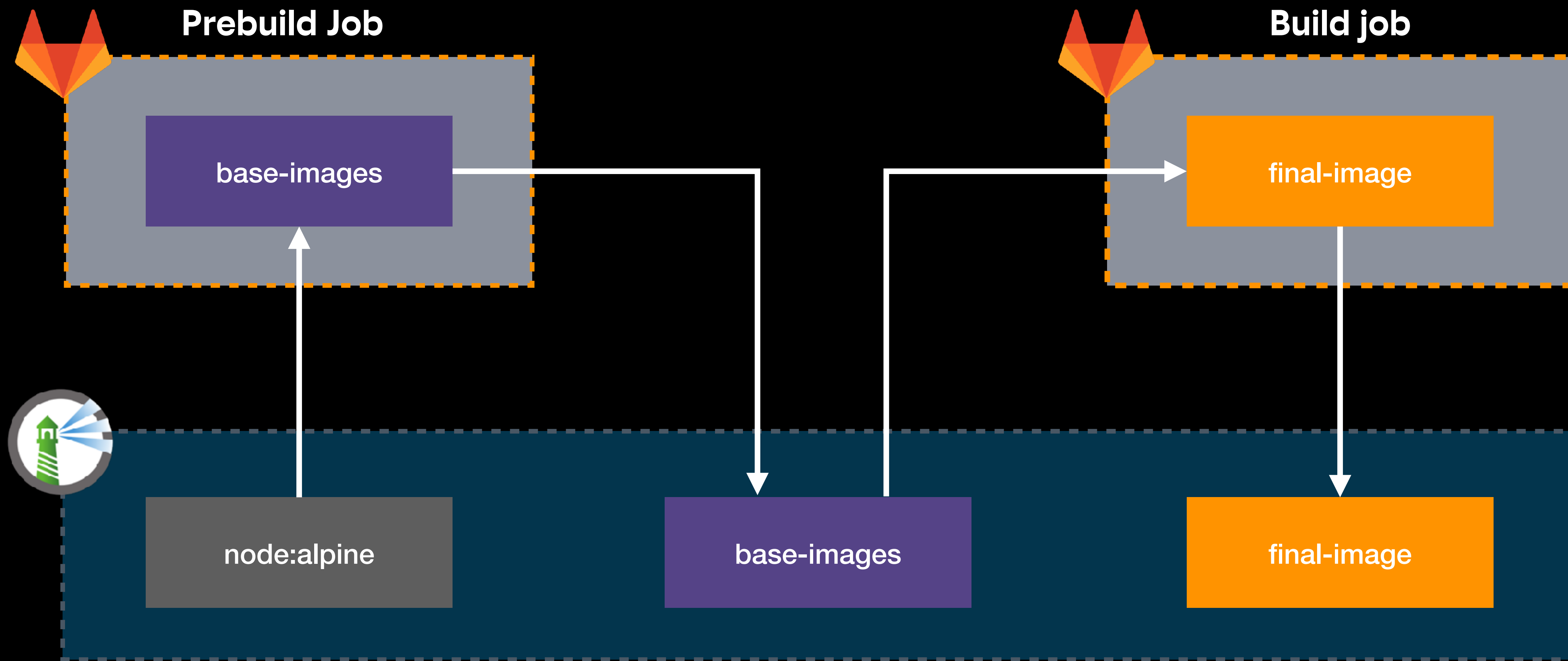
Multi-stage build



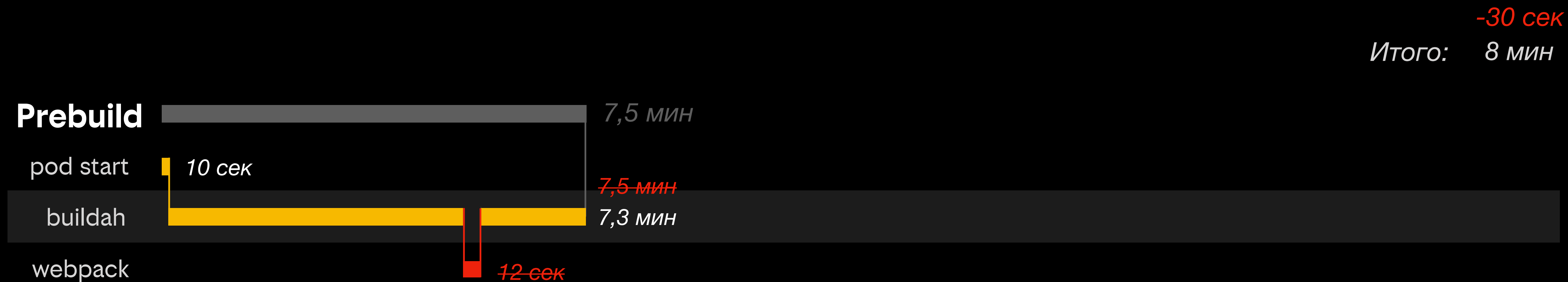
Переиспользуем базовые слои



Вместе с хранилищем



Отдельный prebuild



Gitlab rules для prebuild

rules:

- if: \$CI_COMMIT_BRANCH != "dev"
when: never
- changes:
 - .gitlab-ci.yml
 - Dockerfile
 - yarn.lock
- when: manual

Gitlab rules для prebuild

rules:

- if: \$CI_COMMIT_BRANCH != "dev"
when: never
- changes:
 - .gitlab-ci.yml
 - Dockerfile
 - yarn.lock
- when: manual

Код полностью:



Структура хранения

node: 24-alpine

final-image

final-image

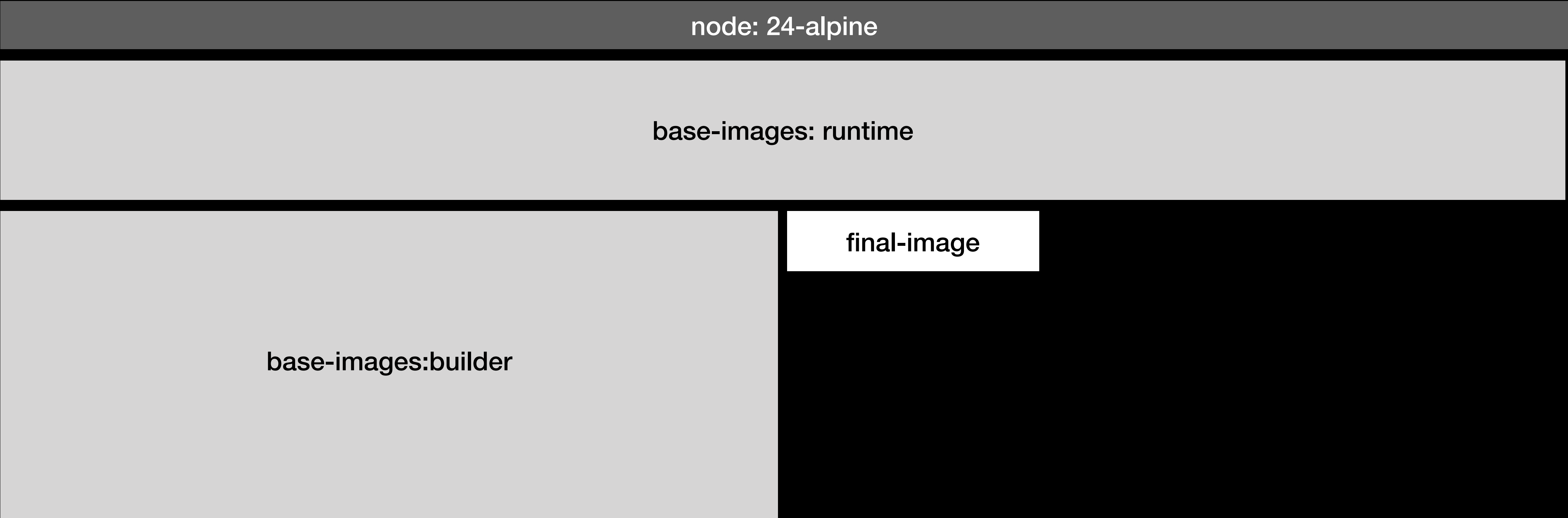
final-image

final-image

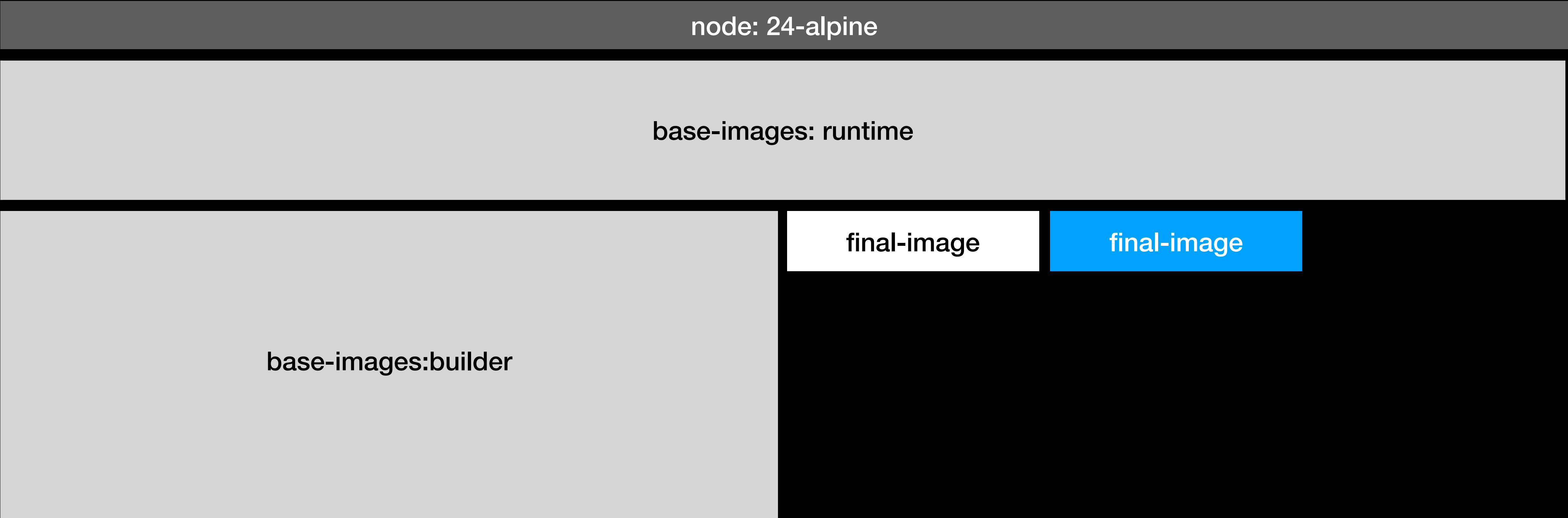
final-image

...

Новая структура хранения

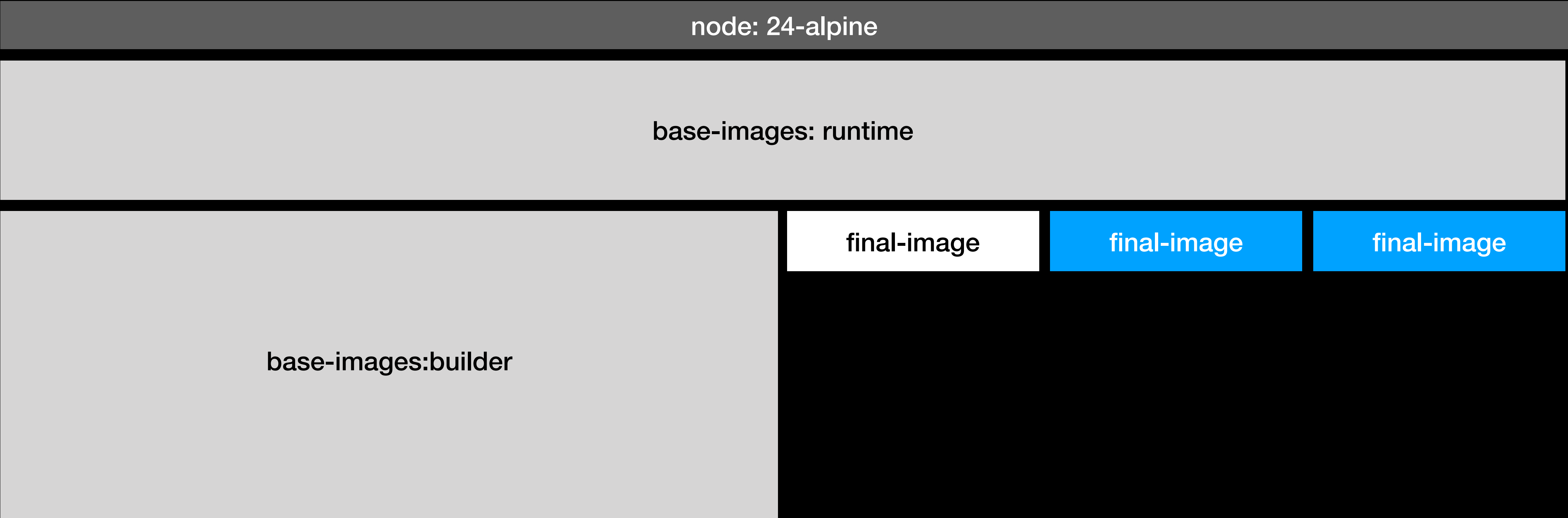


Новая структура хранения

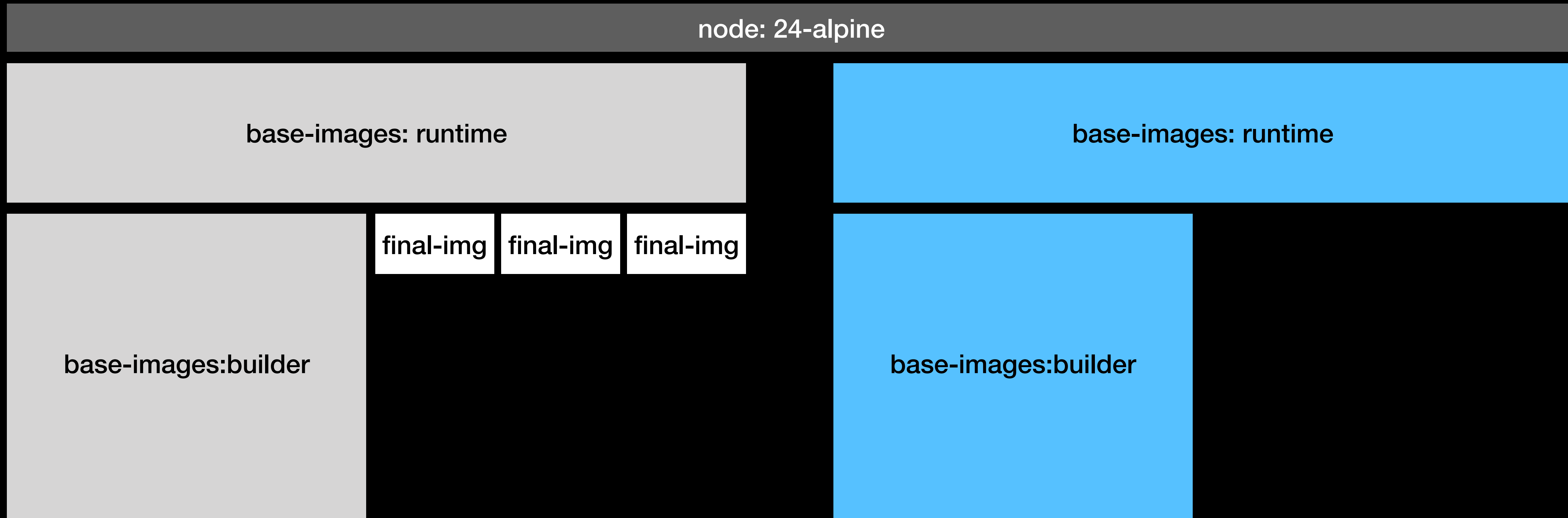




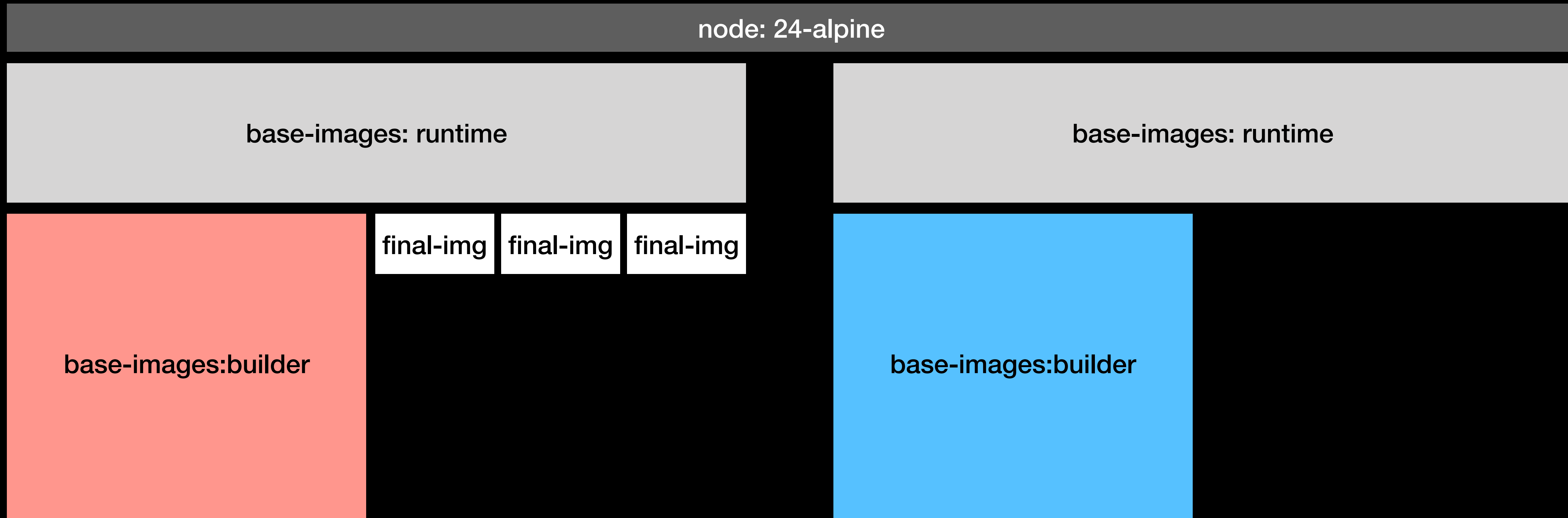
Новая структура хранения



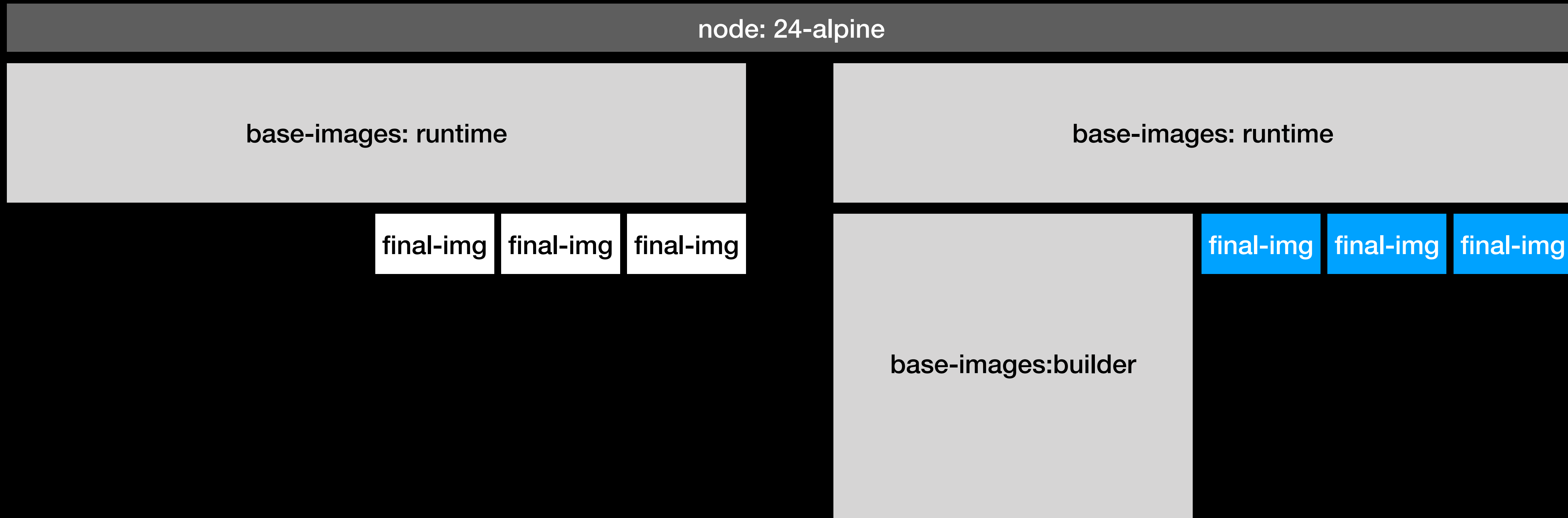
Новая структура хранения



Новая структура хранения



Новая структура хранения

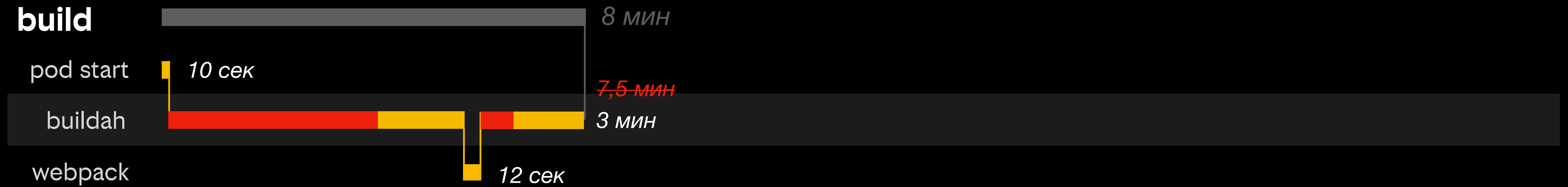


Заказываем готовые ступени



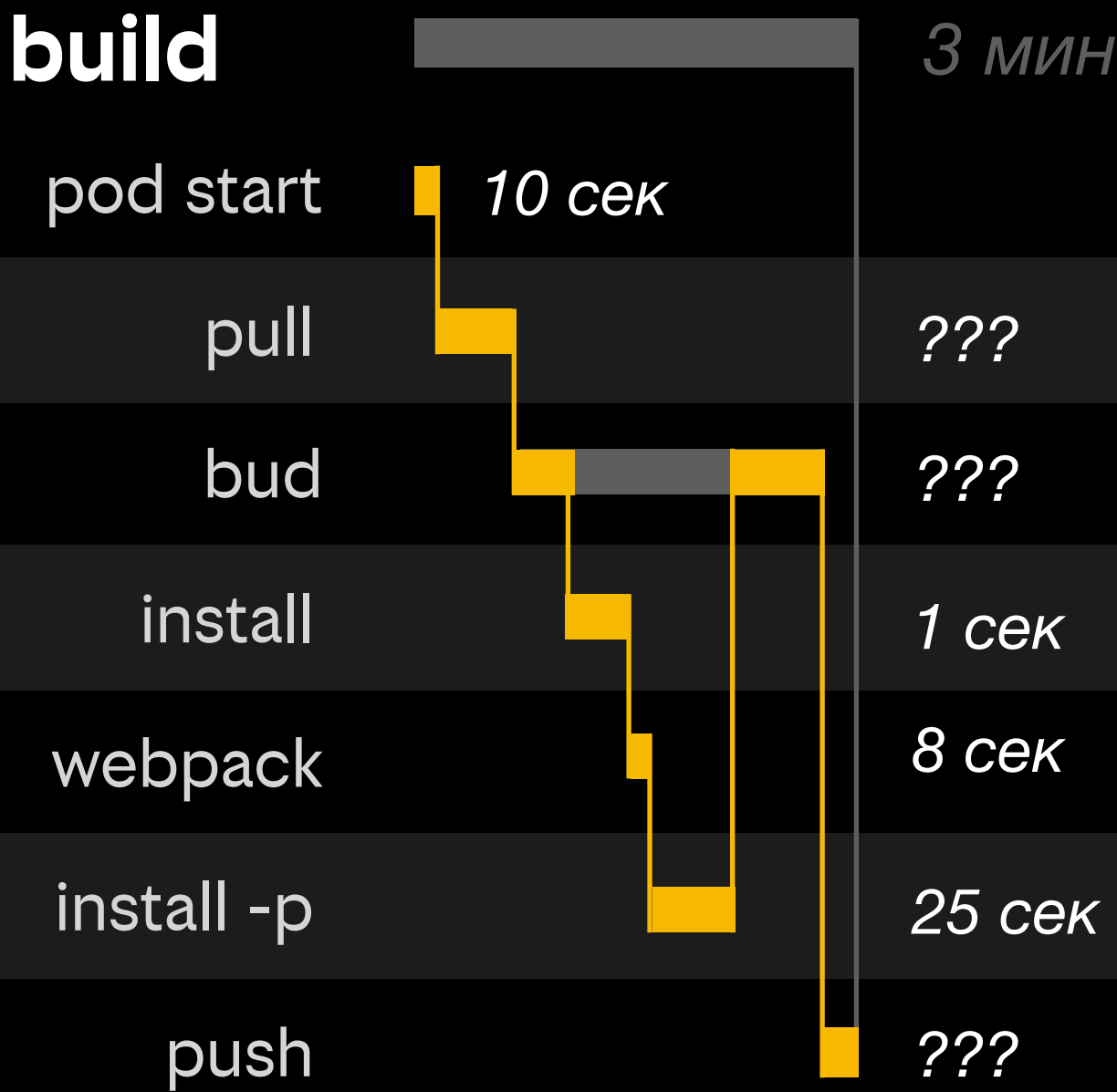
Значительно быстрее

-4,5 мин
Итого: 7,5 мин



На что уходит время

Итого: 3 мин



Секция script

✓ 30 Executing "step_script" stage of the job script

03:52

Пользовательские секции

```
✓ 449 [CMD] buildah push harbor.v.domain/path/to/final-image:dev-1.0.0-1939 00:06
450 Getting image source signatures
451 Copying blob sha256:bbcf595291f444fdc85e6a19f992cefd9158c8bd5178c71ea64552884a44c984
452 Copying blob sha256:dc92897b8f79f2f6820a0f6a9da9fa7a585fc4d23e8281897182dfc47bb21b06
453 Copying blob sha256:850cb46c010c71770c94c642211be93ea36abe1984271cd668507b290ba935a8
454 Copying blob sha256:b9d89e45ac894b6871726a0f9a0e301f7c6e9030a080c55cec4a19d47597854e
455 Copying blob sha256:31ad4a471c6852bfec14d757cc75a566c82a9769f12c3918cf6bf52cc0eeb3d0
456 Copying blob sha256:5c0b409296cdcda25f42f4b532ccd8d6cbfbb5177c706006fe9efc8d9d50f1d7
457 Copying blob sha256:b8d950a864888eb8b38ea4f2b9a1ab0cf830a1e3d0160cad9b8013a0b496f0e
458 Copying blob sha256:4ab2bf570dcc8064edbce340740416dd2c789da9606268e9cbb8b30f208eb86e
459 Copying blob sha256:5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
460 Copying blob sha256:2c13818c852f576b0debf76865aa7e88bdc8e40ba014dd15d230d32a555e1b7c
461 Copying blob sha256:dab0216fbc46c540abd372363c894f4e3532cc9597a08022091f5bc6e6406480
462 Copying blob sha256:bcbe0980bd63a43e03aceaebd79b5e216c05ca4971b729d441464ac4a5a2bb83
463 Copying config sha256:4ee5d5b2278f71c63493b27054ee84df117b2f1932398198a694794e3aabf9dc
464 Writing manifest to image destination
```

Пользовательские секции

```
echo -e "section_start:$(date +%s):my-section-id\r\e[0K" >&2;  
...  
echo -e "section_end:$(date +%s):my-section-id\r\e[0K" >&2;
```


Схлопывать по умолчанию

```
echo -e "section_start:$(date +%s):my-section-id[collapsed=true]\r\e[0K" >&2;  
...  
echo -e "section_end:$(date +%s):my-section-id\r\e[0K" >&2;
```

Схлопывать по умолчанию

```
echo -e "section_start:$(date +%s):my-section-id[collapsed=true]\r\e[0K" >&2;  
...  
echo -e "section_end:$(date +%s):my-section-id\r\e[0K" >&2;
```

```
> 74 [CMD] RUN_ENV=ci yarn install --prefer-offline --frozen-lockfile --production  
122 Searching for hook in /home/host/app
```

00:51

Удобные helper-методы для секций

```
ci-cmd *
```

```
ci-ccmd *
```

```
ci-section "Скрипт между вызовами будем обернут в секцию"
```

```
...
```

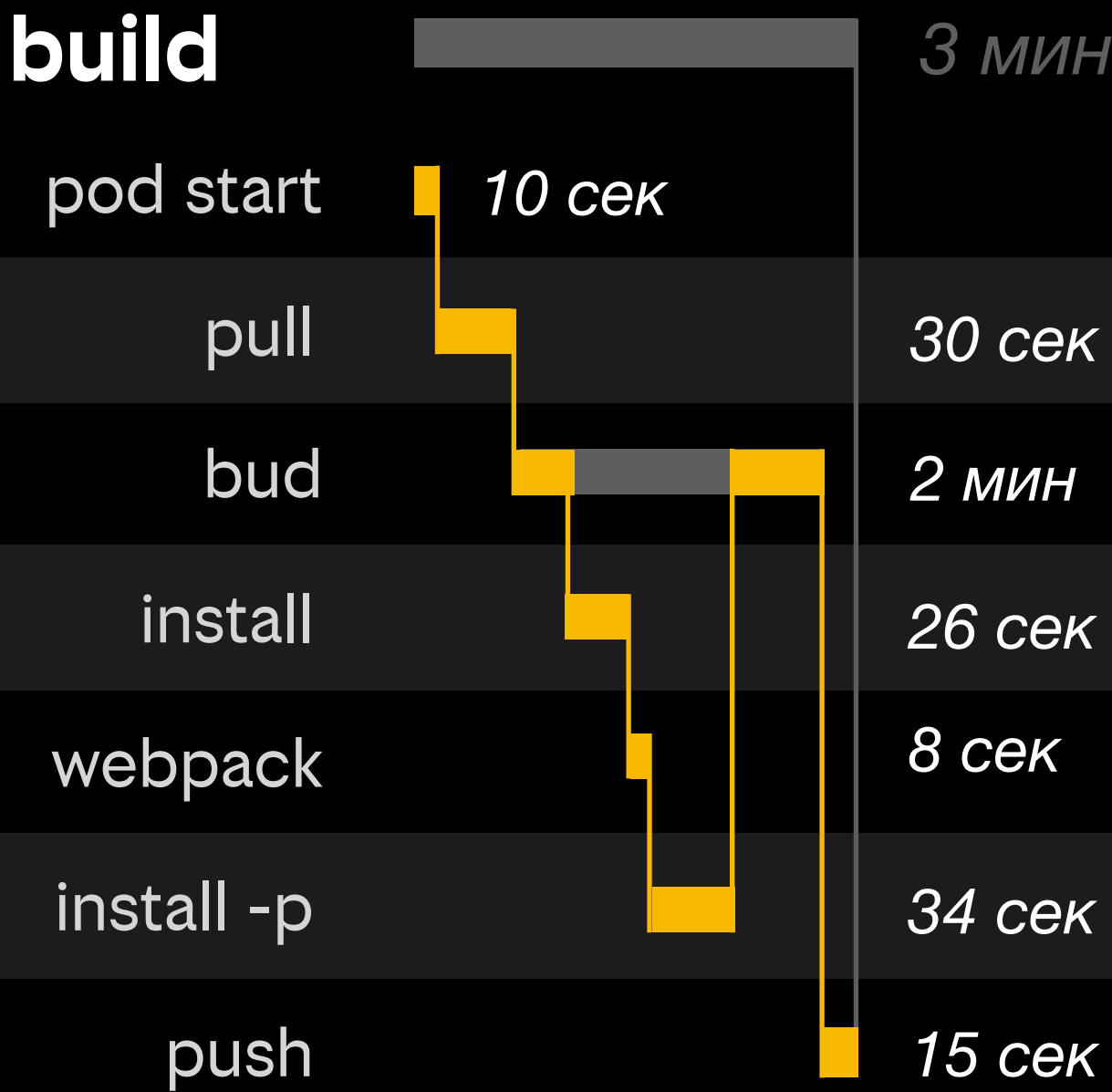
```
ci-endsection
```

Посмотреть код:



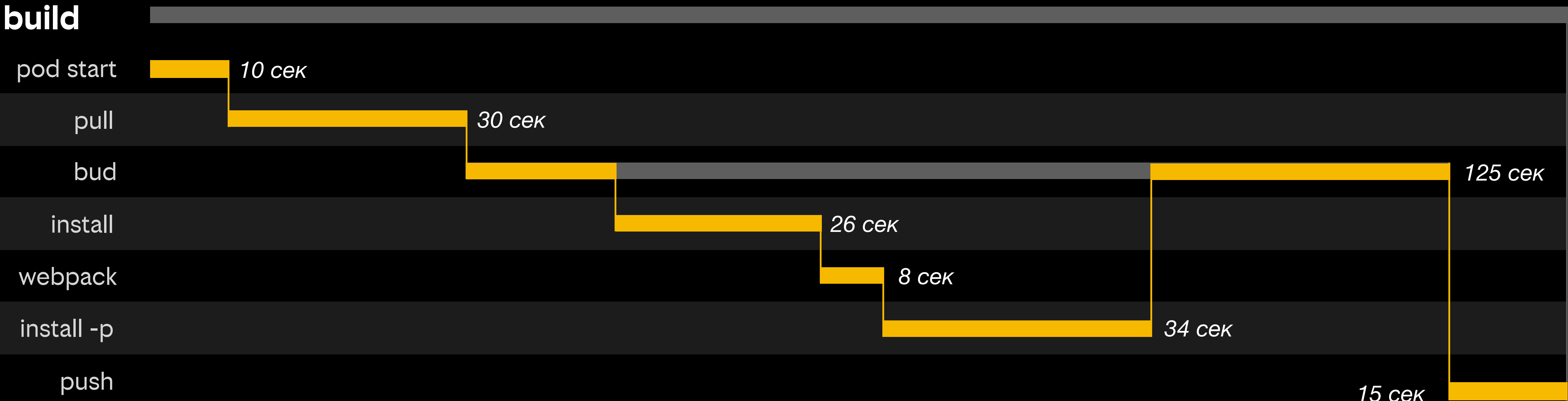
На что уходит время

Итого: 3 мин



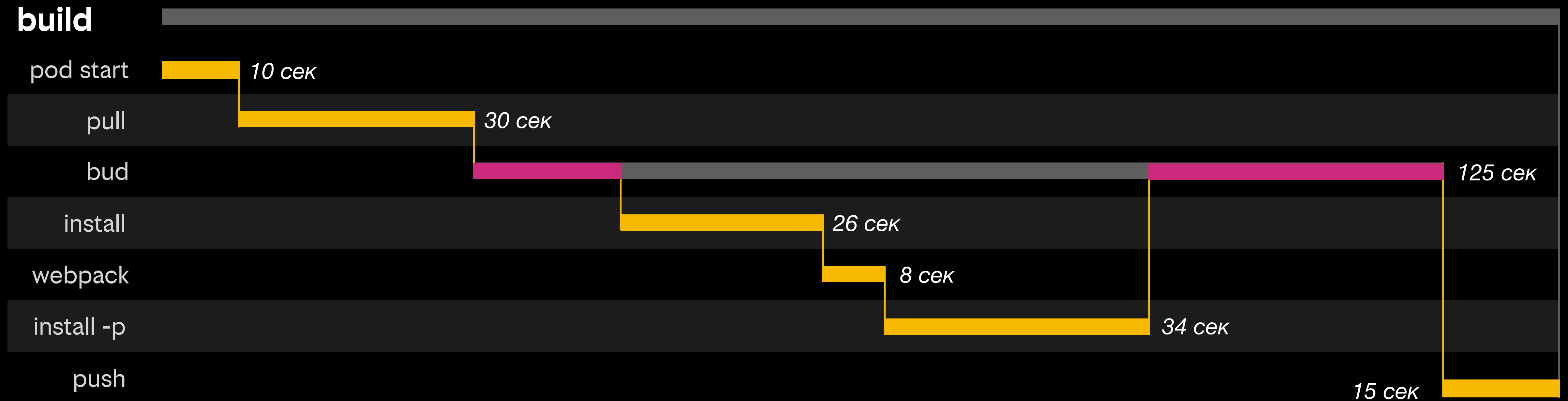
На что уходит время

Итого: 180 сек



На что уходит время

Итого: 180 сек



Образы собираются медленно

- скачивание и распаковка базового образа
- упаковка готового образа
- побочные расходы сборки
- абстракция над файловой системы

Открыли настройки - buildah info

```
{
  ...
  "store": {
    "GraphDriverName": "overlay",
    "GraphOptions": [
      "overlay.imagestore=/var/lib/shared",
      "overlay.mount_program=/usr/bin/fuse-overlayfs",
      "overlay.mountopt=nodev,fsync=0"
    ],
    "GraphRoot": "/var/lib/containers/storage",
    "GraphStatus": {
      "Backing Filesystem": "extfs",
      "Native Overlay Diff": "false",
      "Supports d_type": "true",
      "Using metacopy": "false"
    },
    "RunRoot": "/run/containers/storage"
  }
}
```





04. Введение в OverlayFS

- Через год после выхода докера в Linux Kernel появился нативный механизм наложения файловых системам — OverlayFS



04. Введение в OverlayFS

- Через год после выхода докера в Linux Kernel появился нативный механизм наложения файловых системам — OverlayFS
- OverlayFS не реализует хранение и работает поверх стандартных файловых систем



04. Введение в OverlayFS

- Через год после выхода докера в Linux Kernel появился нативный механизм наложения файловых системам — OverlayFS
- OverlayFS не реализует хранение и работает поверх стандартных файловых систем
- Fuse-overlayfs — это userspace-реализация overlayfs



04. Введение в OverlayFS

- Через год после выхода докера в Linux Kernel появился нативный механизм наложения файловых системам — OverlayFS
- OverlayFS не реализует хранение и работает поверх стандартных файловых систем
- Fuse-overlayfs — это userspace-реализация overlayfs
- В отличие от нативного overlayfs, fuse-overlayfs позволяет имитировать overlayfs без root-привилегий



04. Введение в OverlayFS

- Через год после выхода докера в Linux Kernel появился нативный механизм наложения файловых системам — OverlayFS
- OverlayFS не реализует хранение и работает поверх стандартных файловых систем
- Fuse-overlayfs — это userspace-реализация overlayfs
- В отличие от нативного overlayfs, fuse-overlayfs позволяет имитировать overlayfs без root-привилегий
- Привелигированный режим в нативном overlayfs требуется для выполнения операции монтирования (mount)

Гипотеза: fuse-overlayfs медленный

- Медленный diff ("Native Overlay Diff": false)
- Работа с метаданными значительно медленнее нативной

Как перейти на нативный режим?

- Запускать сборочный контейнер в привилегированном режиме

Как перейти на нативный режим?

- Запускать сборочный контейнер в привилегированном режиме
- Overlayfs использует особенный относительно безопасный вид монтирования

Как перейти на нативный режим?

- Запускать сборочный контейнер в привилегированном режиме
- Overlayfs использует особенный относительно безопасный вид монтирования
- В 2018 — форки ядра

Как перейти на нативный режим?

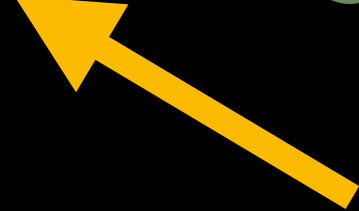
- Запускать сборочный контейнер в привилегированном режиме
- Overlayfs использует особенный относительно безопасный вид монтирования
- В 2018 — форки ядра
- В 2021, linux kernel 5.11 — экспериментальный rootless overlayfs

Как перейти на нативный режим?

- Запускать сборочный контейнер в привилегированном режиме
- Overlayfs использует особенный относительно безопасный вид монтирования
- В 2018 — форки ядра
- В 2021, linux kernel 5.11 — экспериментальный rootless overlayfs
- Linux kernel 5.15 — стабильный rootless overlayfs

Снова смотрим buildah info

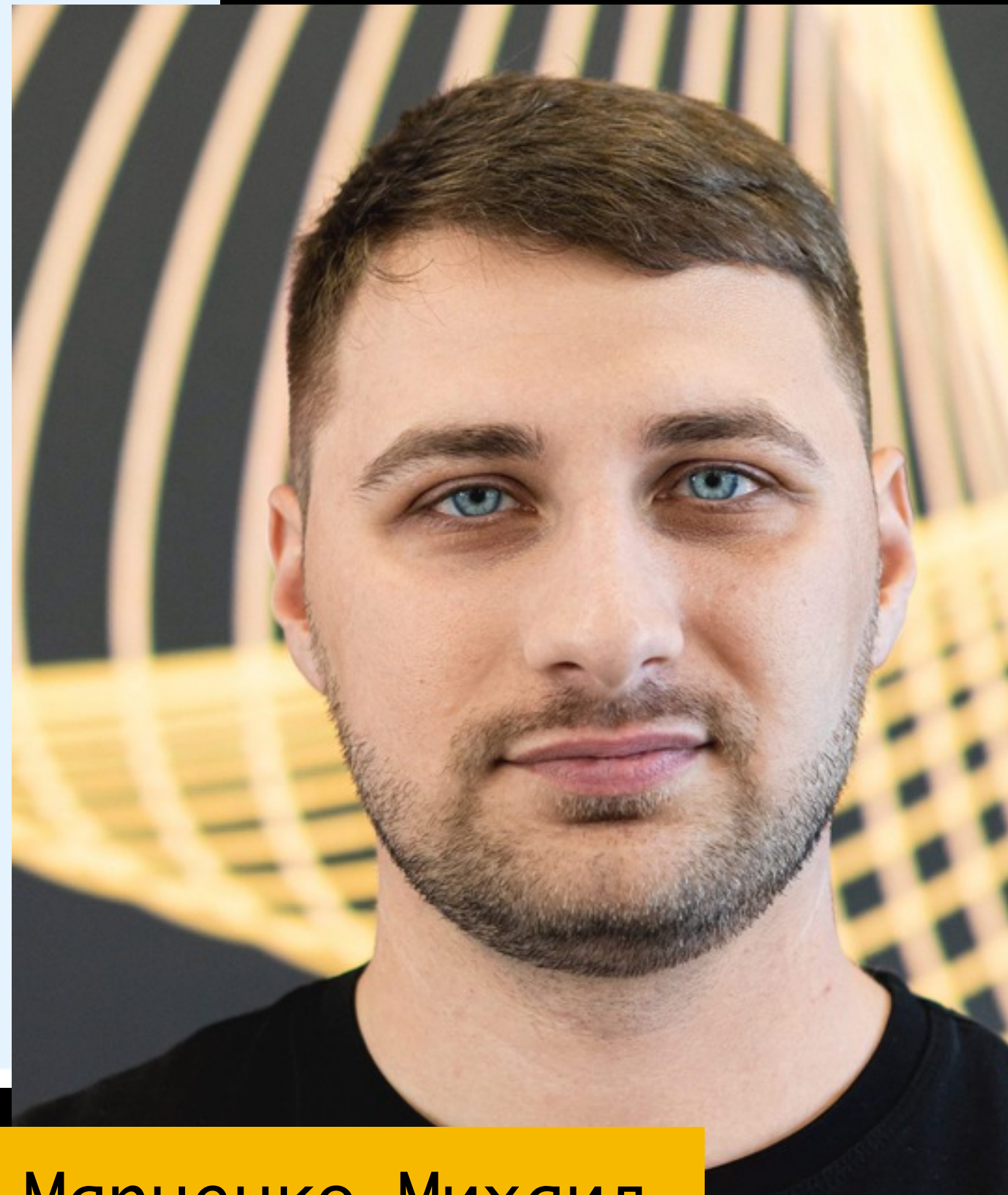
```
{  
  "host": {  
    ...  
    "kernel": "5.4.0-159-generic",  
    "os": "linux"  
  },  
  "store": {  
    ...  
  }  
}
```



Связался с коллегами из инфры

МА

Морозкин Артём



Марченко Михаил

ΦΦ

Фаюршин Фархад

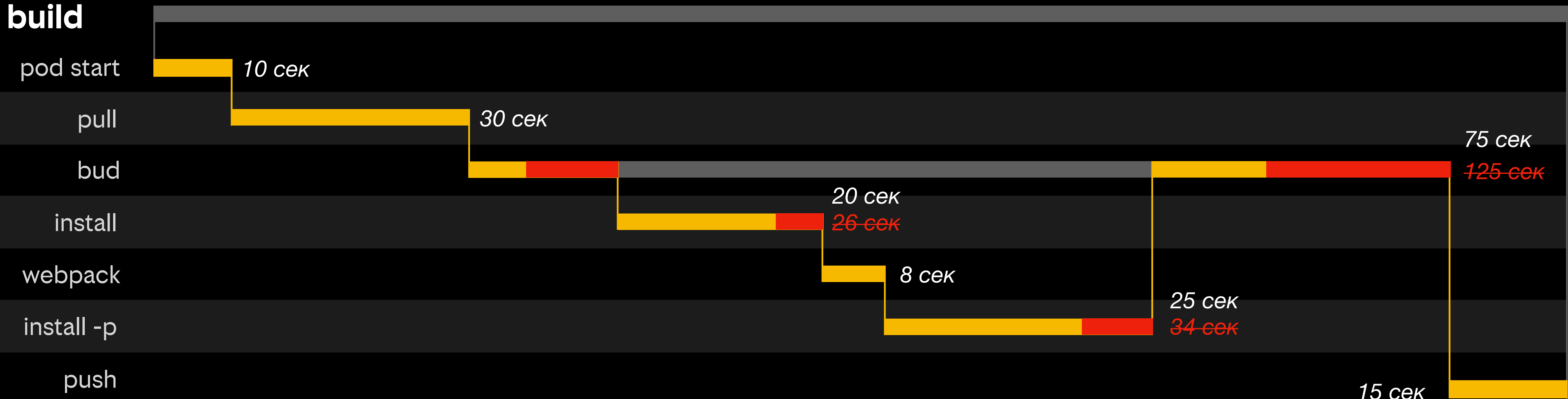
Связался с коллегами из инфры

- Обновили kernel на одном из раннеров
- Помогли настроить на него селектор



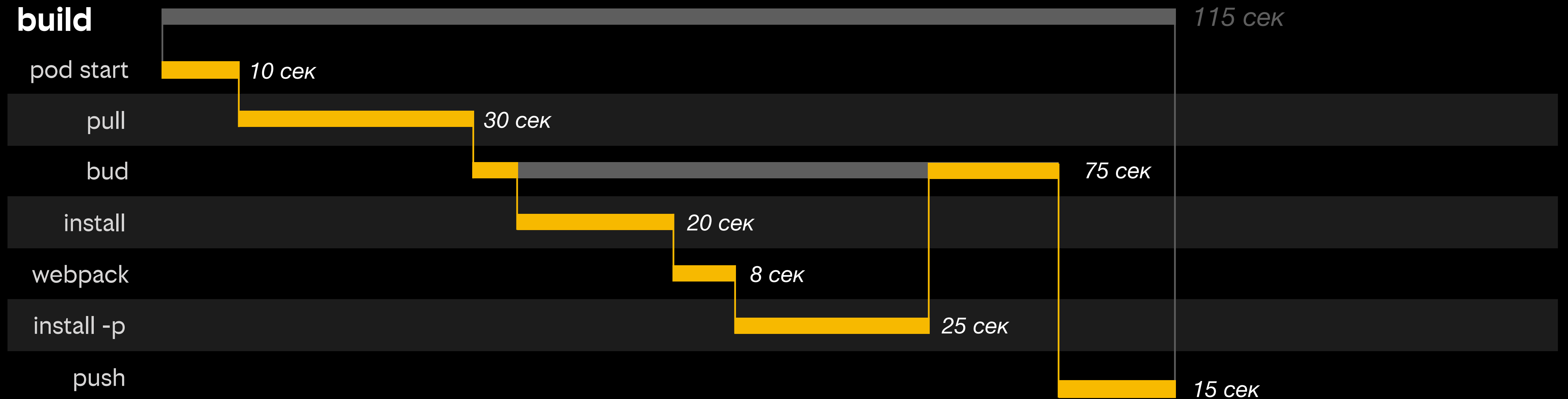
"Native Overlay Diff": true

Итого: ~~-50 сек~~
180 сек



"Native Overlay Diff": true

Итого: 130 сек



Всё ещё долго

- нейросеть настойчиво предлагала собирать в памяти
- любезно выделили tmpfs на 2GB

Сборка в памяти

- нейросеть настойчиво предлагала собирать в памяти
- любезно выделили tmpfs на 2GB
- build ускорился с 1:50 до 1:20
- prebuild упал - 2Gb не хватило, 4Gb едва хватало, заложили 8Gb в запас

Сборка в памяти

- нейросеть настойчиво предлагала собирать в памяти
- любезно выделили tmpfs на 2GB
- build ускорился с 1:50 до 1:20
- prebuild упал - 2Gb не хватило, 4Gb едва хватало, заложили 8Gb в запас
- prebuild ускорился с 5:30 до 4:50

Размер не главное, но:

	Builder	Runtime	Final
До начала работы	-	-	575
Сборка базовых образов	900	541	576 +1

■ Как узнать, что внутри образа?

● Dive

■ Как узнать, что внутри образа?

- Dive
- Archive utility

Посмотреть код:



Как узнать, что внутри образа?

```

✓  📁 builder
  ✓  📁 blobs
    ✓  📁 sha256
      >  📁 6bb471c7406ae9a3bf9187b352ef8892e3d5ab002b4e0d5e4fdb8a8130d2bd8d 2
      >  📁 29df493baa13de438d6d2ece3a8333032e0b7b9b9d8cce4ee82194da255f61e1 2
      >  📁 95989682e74292629524b57cb982468b957141664c34f2caba6c680933cf800a 2
      ✓  📁 accc174778a252474d74133b886f9e146ef927b52ab53b6fce98a8bc6f5e0d47 2
        >  📁 bin
        >  📁 etc
        >  📁 lib
        >  📁 opt
        >  📁 usr
        >  📁 var
      >  📁 e7edabbd05646dcc420073507ce99c710be00e369d3b7f51e3e94b5e3501a2ca 2
      📄 5f70bf18a086007016e948b04aed3b82103a36bea41755b6cddfaf10ace3c6ef
      📄 6bb471c7406ae9a3bf9187b352ef8892e3d5ab002b4e0d5e4fdb8a8130d2bd8d
      📄 29df493baa13de438d6d2ece3a8333032e0b7b9b9d8cce4ee82194da255f61e1
      ≡ 62c6e9a26412caeebf605e19ddd0dd512446fec3e113366aaa0f236a0bee7c6d
      ≡ 89c5bb8e68b484275c1bfd86db4ddea795410f9df3b292b892d411d5d993a7f4
```

Yarn cache - 1050мб (до gzip)

```

  ✓  📁 home
    ✓  📁 node
      > 📁 .config
      > 📁 .yarn
      > 📁 app
    ✓  📁 cache
      ✓  📁 yarn
        ✓  📁 v6
          📁 .tmp
          > 📁 npm-@aashutoshrathi-word-wrap-1.2.6-bd9154aec9983f77b3a034ecaa015c2e4201f6cf-integrity
          > 📁 npm-@adobe-css-tools-4.4.4-2856c55443d3d461693f32d2b96fb6ea92e1ffa9-integrity
          > 📁 npm-@ahooksjs-use-request-2.8.15-daa32a8395ba75e8deb9f4fde4e221a4a8f525db-integrity
          > 📁 npm-@babel-cli-7.19.3-55914ed388e658e0b924b3a95da1296267e278e2-integrity
          > 📁 npm-@babel-code-frame-7.26.2-4b5fab97d33338eff916235055f0ebc21e573a85-integrity
          > 📁 npm-@babel-code-frame-7.27.1-200f715e66d52a23b221a9435534a91cc13ad5be-integrity
          > 📁 npm-@babel-compat-data-7.28.5-a8a4962e1567121ac0b3b487f52107443b455c7f-integrity
          > 📁 npm-@babel-core-7.28.5-4c81b35e51e1b734f510c99b07dfbc7bbbb48f7e-integrity
          > 📁 npm-@babel-eslint-parser-7.23.3-7bf0db1c53b54da0c8a12627373554a0828479ca-integrity
          > 📁 npm-@babel-generator-7.28.5-712722d5e50f44d07bc7ac9fe84438742dd61298-integrity
```


Перенесли кэш в builder

	Builder	Runtime	Final
До начала работы	-	-	575
Сборка базовых образов	900	541	576 +1
Cache перемещен в builder	871 -29	271 -270	331 -245

Папка .git весит 200мб

- GIT_DEPTH = 1

Папка .git весит 200мб

- GIT_DEPTH = 1
- git lfs

Папка .git весит 200мб

- GIT_DEPTH = 1
- git lfs
- дублируются тестовые mock'и и бинарники

Удалена папка .git/lfs/objects

	Builder	Runtime	Final
До начала работы	-	-	575
Сборка базовых образов	900	541	576 +1
Cache перемещен в builder	871 -29	271 -270	331 -245
Удалены LFS-объекты из .git	706 -165	271	331

GIT_LFS_SKIP_SMUDGE=1

	Builder	Runtime	Final
До начала работы	-	-	575
Сборка базовых образов	900	541	576 +1
Cache перемещен в builder	871 -29	271 -270	331 -245
Удалены LFS-объекты из .git	706 -165	271	331
Удалены LFS-объекты	541 -165	236 -35	296 -35

Анализ слоев `node_modules`

- runtime-модули задублированы в `builder`



05. OverlayFS: Copy up

- Нижние слои (lower) доступны только для чтения, поэтому даже **при удалении** файла, его содержимое не удаляется, а только отмечается в верхнем (upper) слое как удаленное



05. OverlayFS: Copy up

- Нижние слои (lower) доступны только для чтения, поэтому даже **при удалении** файла, его содержимое не удаляется, а только отмечается в верхнем (upper) слое как удаленное
- Если файл уже существует, то перед первым изменением файла выполняется операция **copy-up**: файл со всеми метаданными копируется в верхний слой



05. OverlayFS: Copy up

- Нижние слои (lower) доступны только для чтения, поэтому даже **при удалении** файла, его содержимое не удаляется, а только отмечается в верхнем (upper) слое как удаленное
- Если файл уже существует, то перед первым изменением файла выполняется операция **copy-up**: файл со всеми метаданными копируется в верхний слой
- Даже если изменение касается только метаданных, происходит полное копирование файла. Данное поведение можно изменить по средством флага **metacopy=on**

Убрали дублирование в node_modules

- включили опцию metacopy=on

Зачистили node_modules

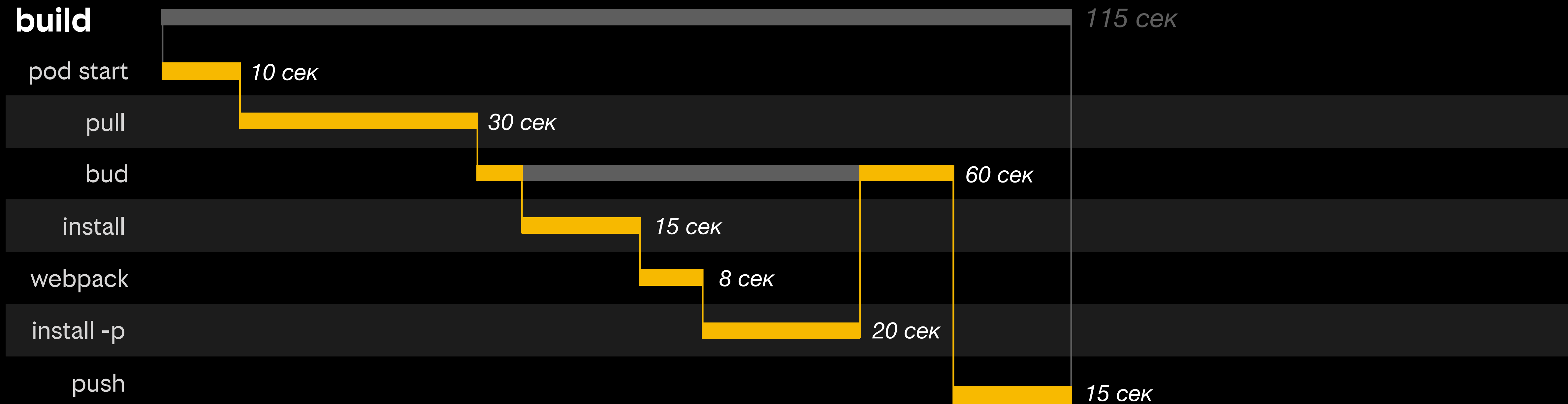
- включили опцию metacore=on
- перенесли некоторые не продуктовые модули в dev-зависимости (docusaurus)
- сбросили yarn.lock (чистая установка)

Произвели чистку зависимостей

	Builder	Runtime	Final
До начала работы	-	-	575
Сборка базовых образов	900	541	576 +1
Cache перемещен в builder	871 -29	271 -270	331 -245
Удалены LFS-объекты из .git	706 -165	271	331
Удалены LFS-объекты	541 -165	236 -35	296 -35
Зачистка модулей и metacore=on	434 -107	165 -71	225 -71

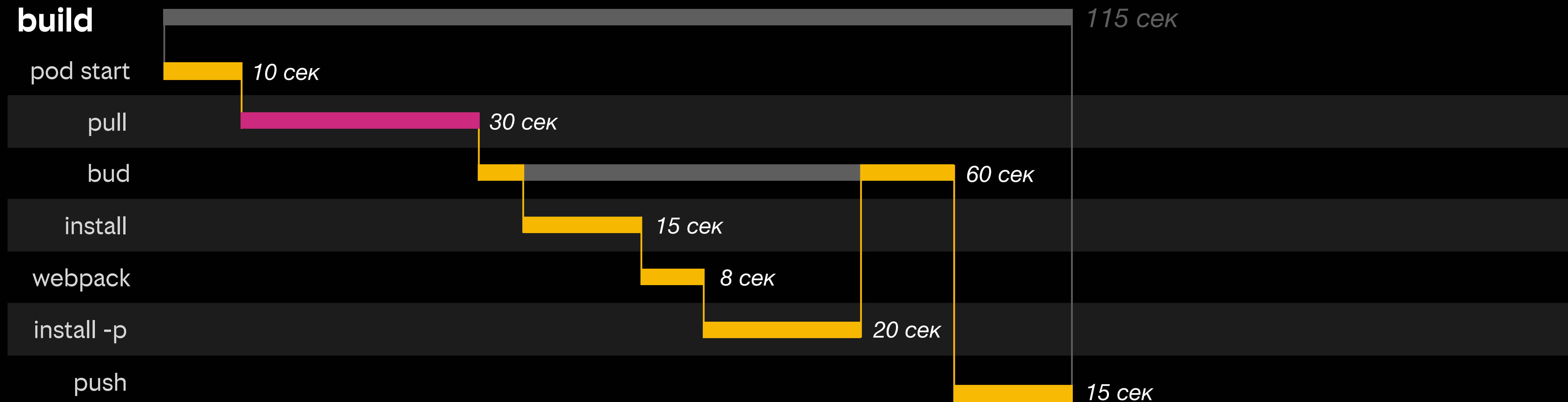
Сборка стала чуть быстрее

Итого: 115 сек



Долгий pull

Итого: 115 сек



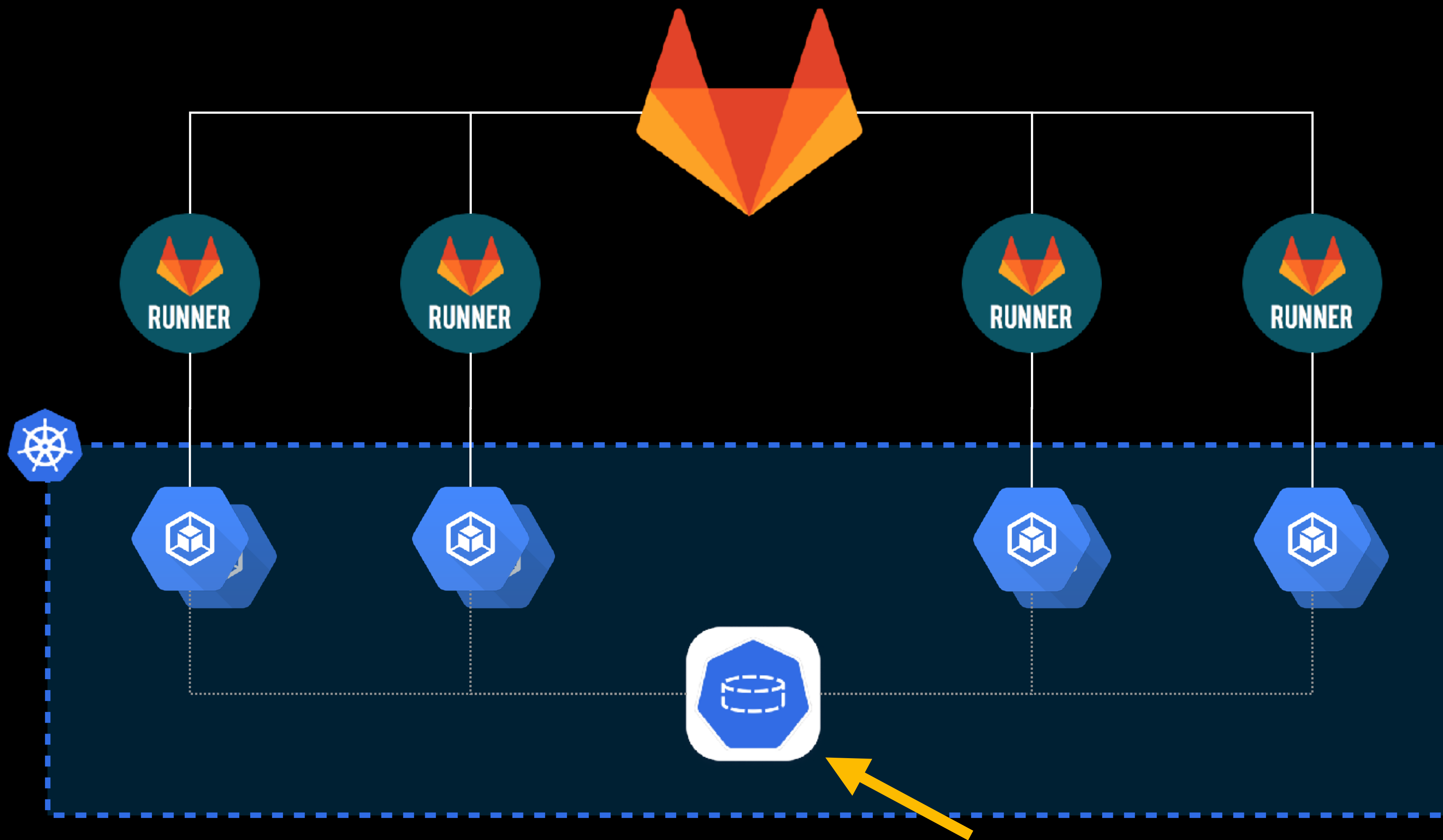
Часть IV

Инверсия контейнеров

Как не доставлять ступени



Как не доставлять базовые образы



■ Как не доставлять базовые образы

- хранить образы на rvc
- без архива

storage.conf

```
[storage]
```

```
driver = "overlay"
```

```
graphroot = "/var/lib/containers/storage"
```

```
runroot = "/var/lib/containers/run"
```

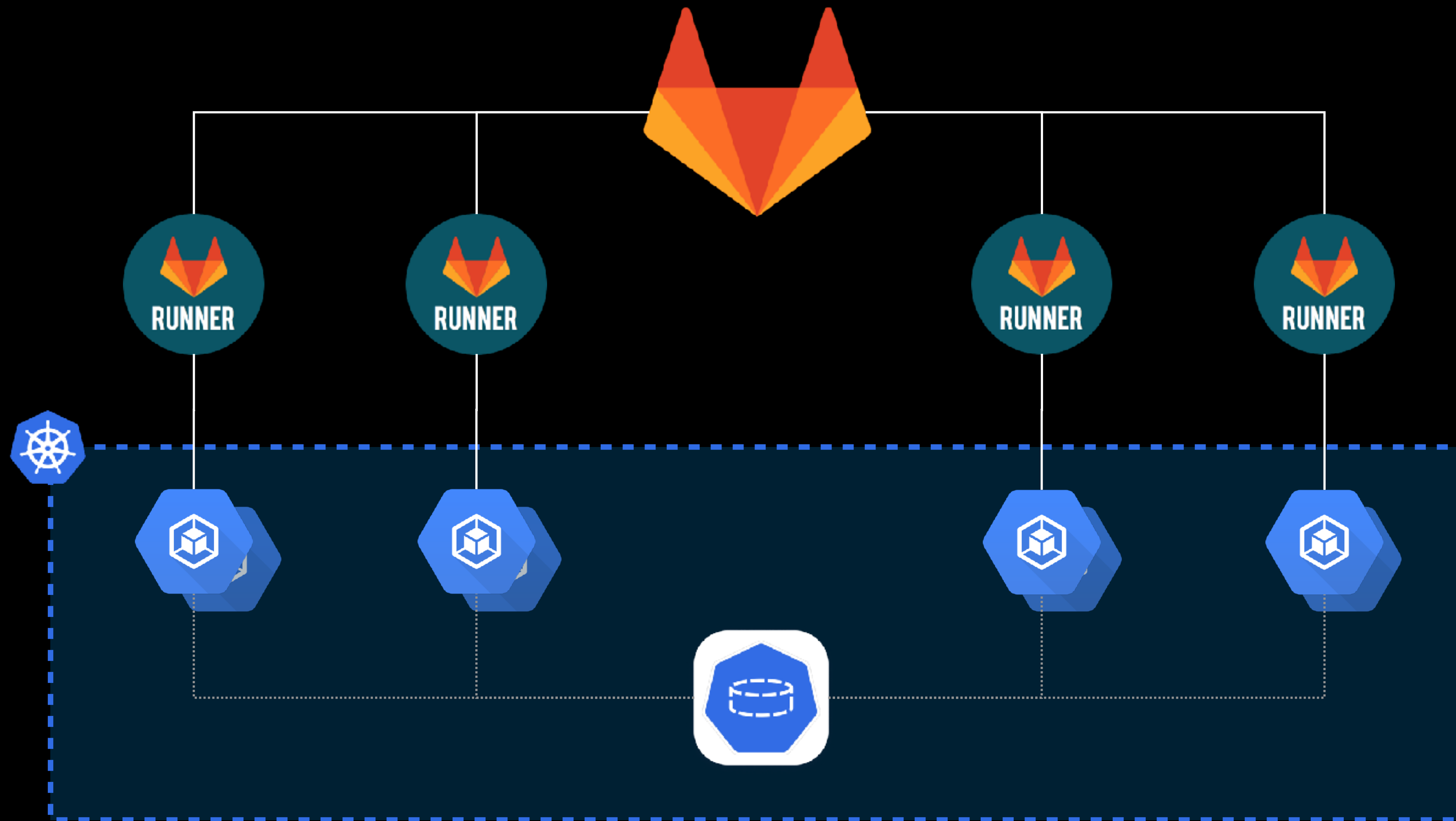
```
[storage.options]
```

```
additionalimagestores = ["/pvc/lib/containers/additional"]
```

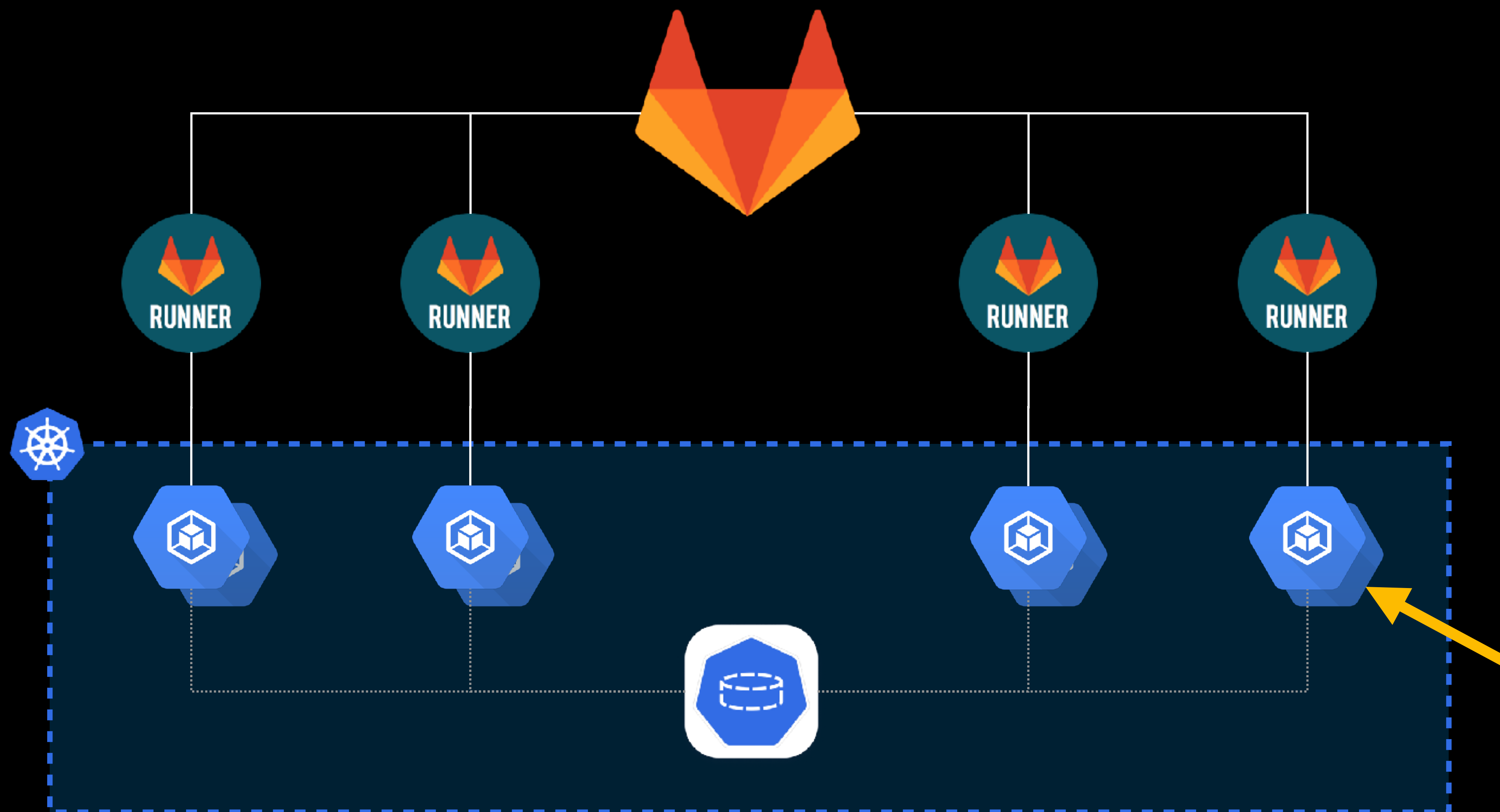

OverlayFS не работает с NFS

- Ошибка: DT_UNKNOWN
- Отсутствует поддержка d_type

Другие варианты?

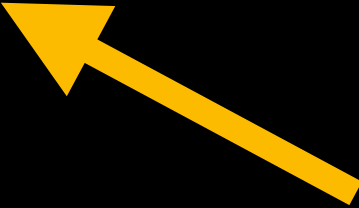


Сборочный образ



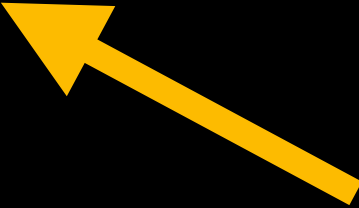
Сборочный образ

```
# gitlab-ci.yml
build:
  stage: build
  image: harbor.domain/common/container-builder:214-main
  script:
    - ...
```



Сборочный образ: builder

```
# gitlab-ci.yml
build:
  stage: build
  image: harbor.domain/project/base-images:builder
  script:
    - ...
```



Добавляем buildah в builder

node: 24-alpine

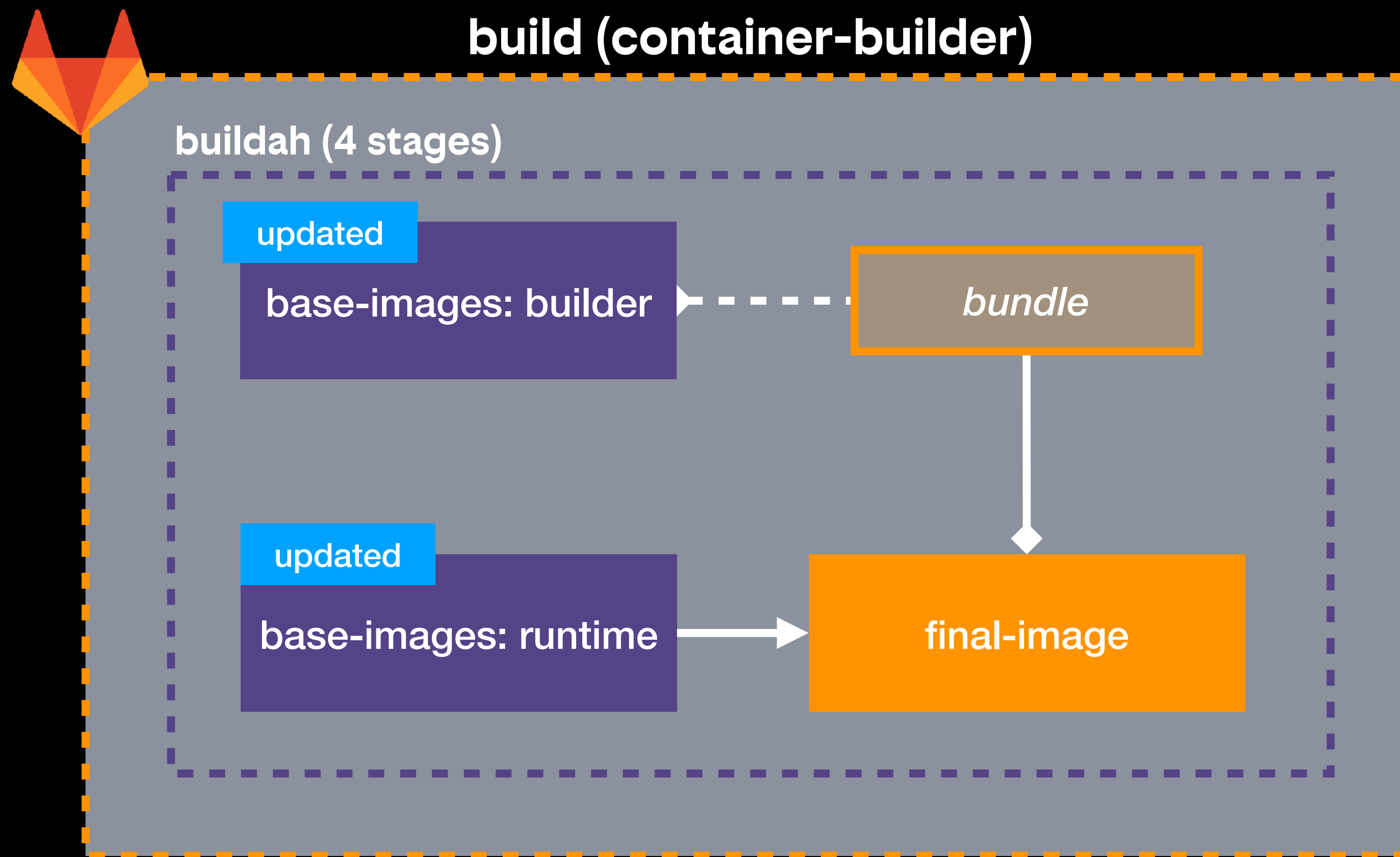
base-images: runtime

final-image

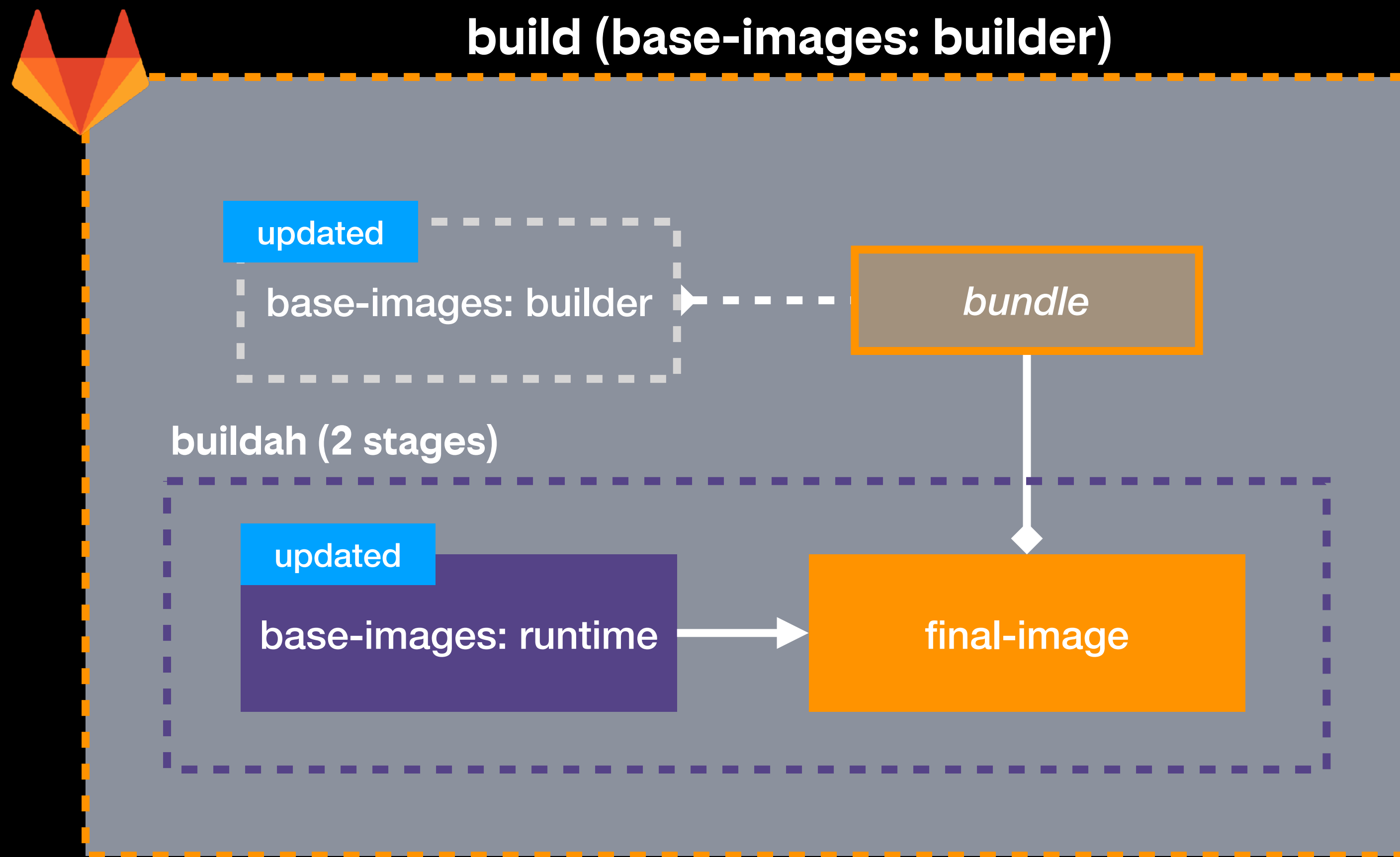
base-images:builder

buildah (+35mb)

Меньше виртуализации



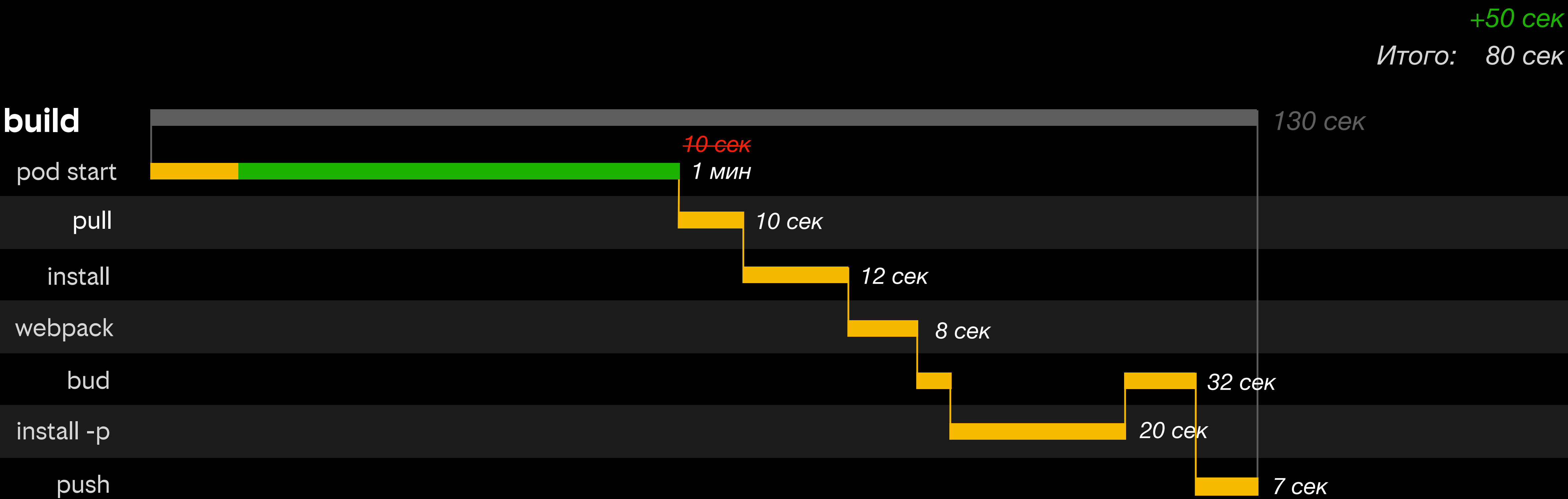
Меньше виртуализации



Портативный космодром



Если base-image поменялся



Прогрев образов

```
prewarm_runner:  
  stage: prewarm  
  image:  
    name: harbor.domain/project/base-images:builder  
    pull_policy: always  
  script:  
    - echo "Worker hot and ready"
```

Прогрев образов

```
prewarm_runner:  
  stage: prewarm  
  image:  
    name: harbor.domain/project/base-images:builder  
    pull_policy: always  
  script:  
    - echo "Worker hot and ready"  
parallel:  
  matrix:  
    - WORKER: ["01", "02", "03", "04", "05"]
```

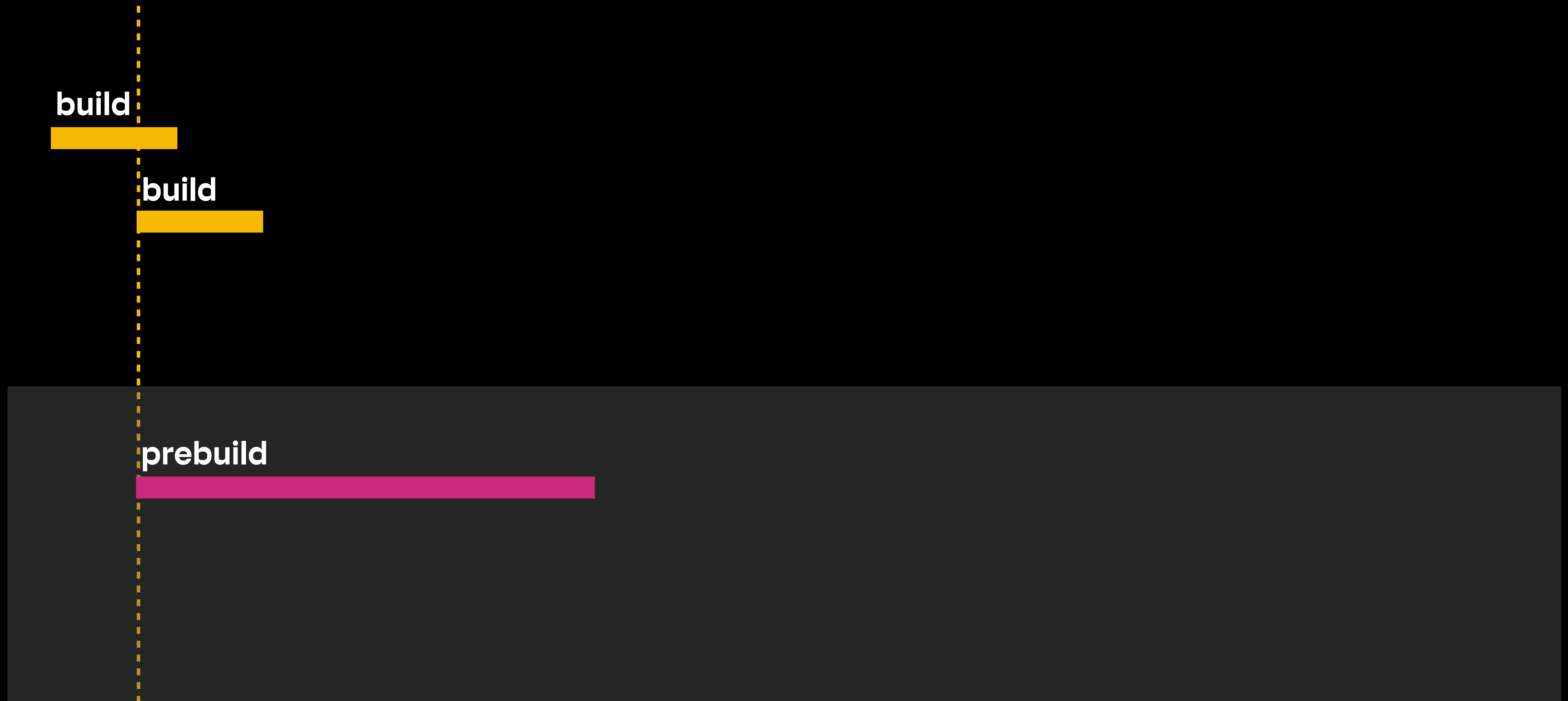

Прогрев образов

```
prewarm_runner:
  stage: prewarm
  image:
    name: harbor.domain/project/base-images:builder
    pull_policy: always
  script:
    - echo "Worker hot and ready"
  parallel:
    matrix:
      - WORKER: ["01", "02", "03", "04", "05"]
  variables:
    KUBERNETES_NODE_SELECTOR_HOSTNAME:
      "kubernetes.io/hostname=community-gitlab-worker$WORKER"
```

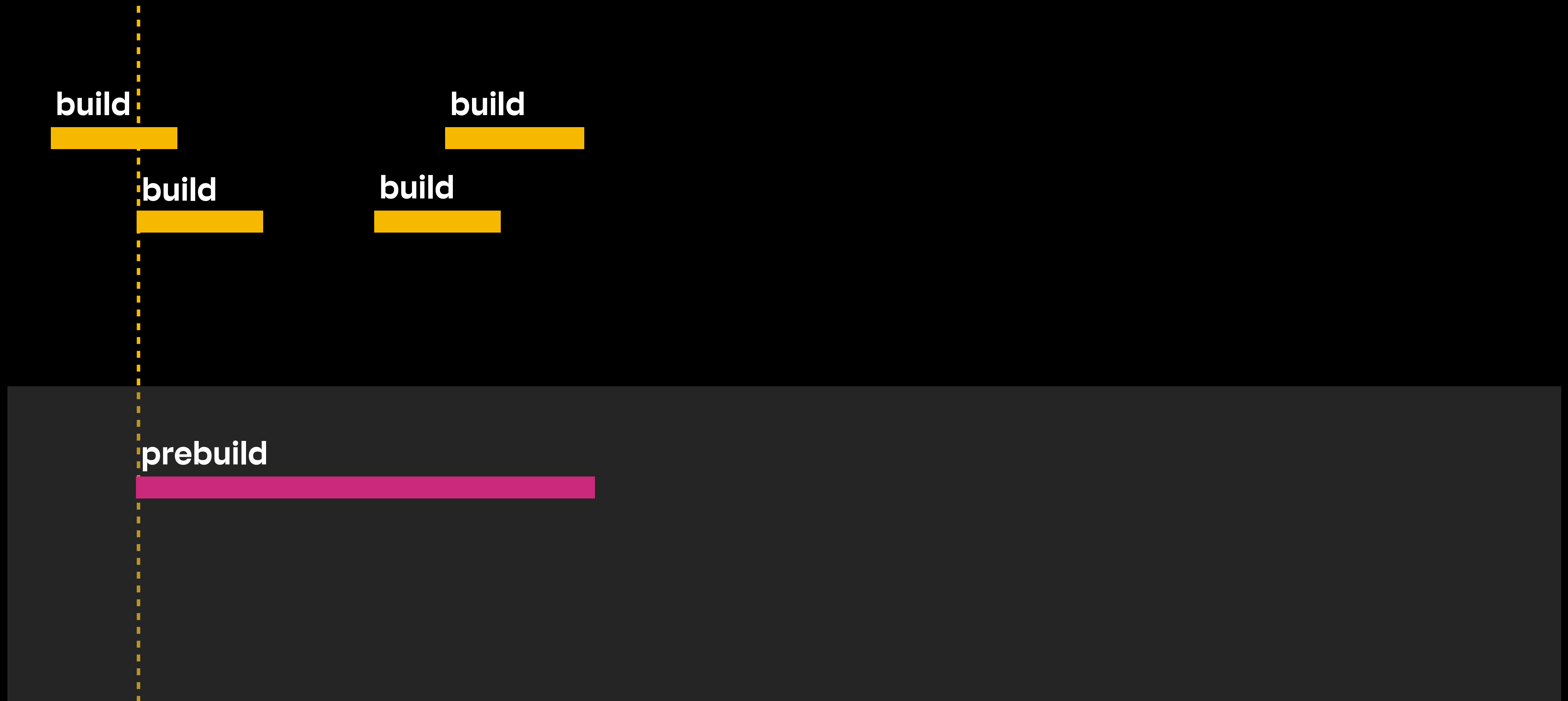
Таймлайн запуска job'ов

build

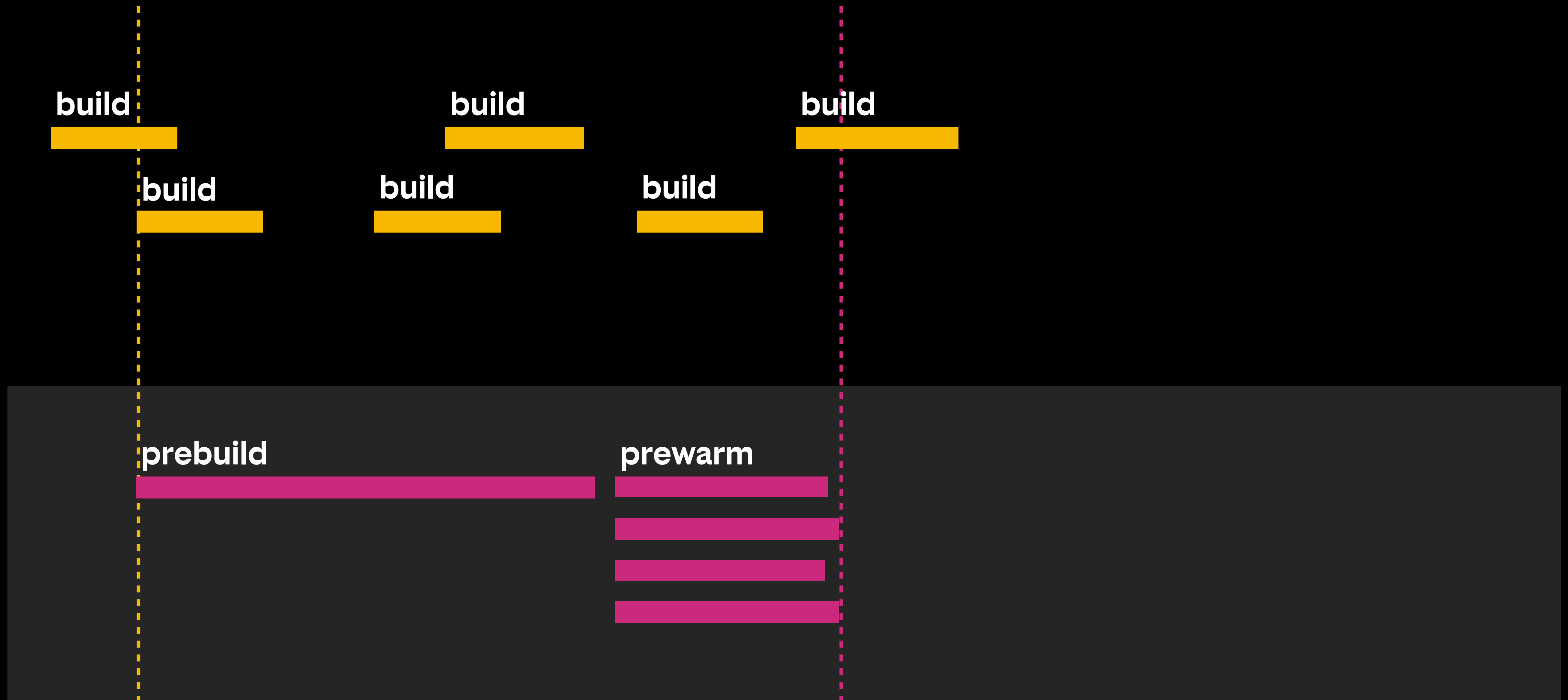
Таймлайн запуска job'ов



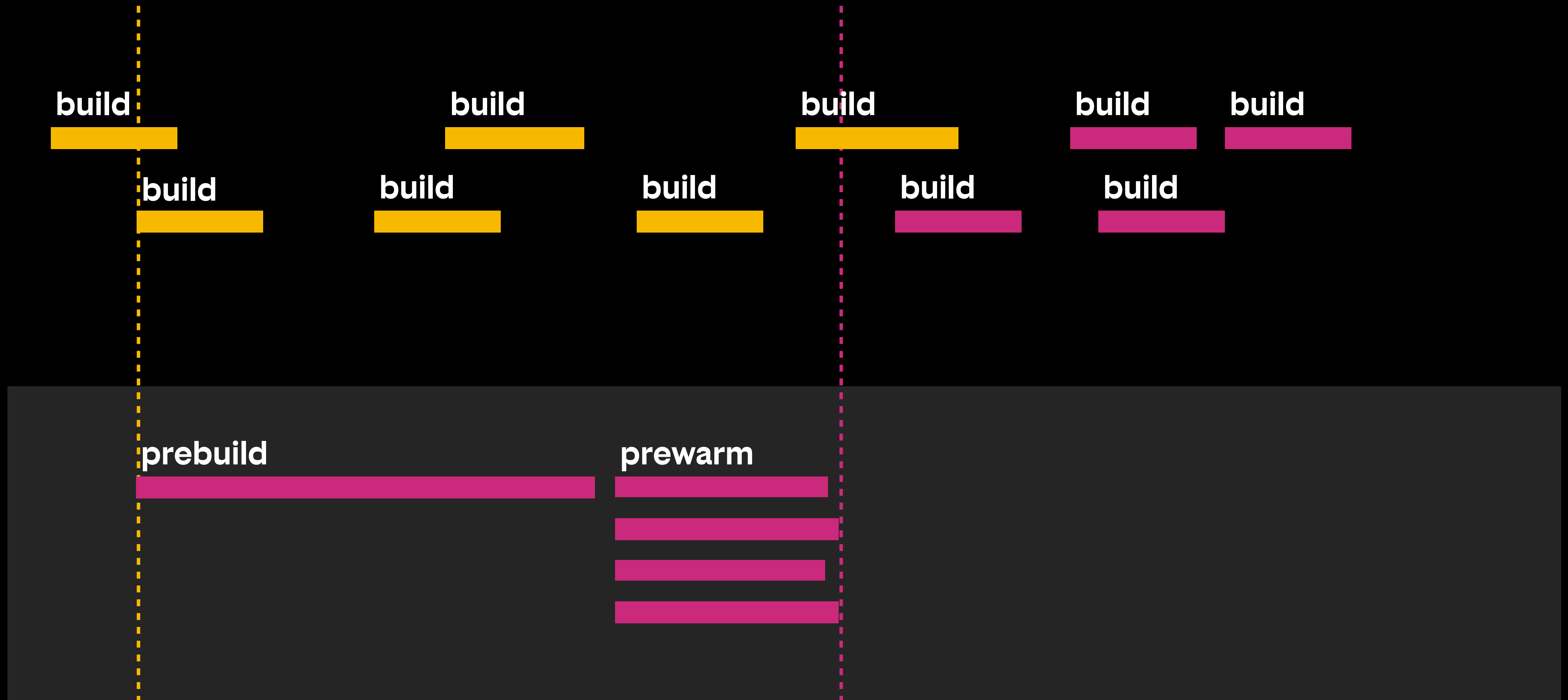
Таймлайн запуска job'ов



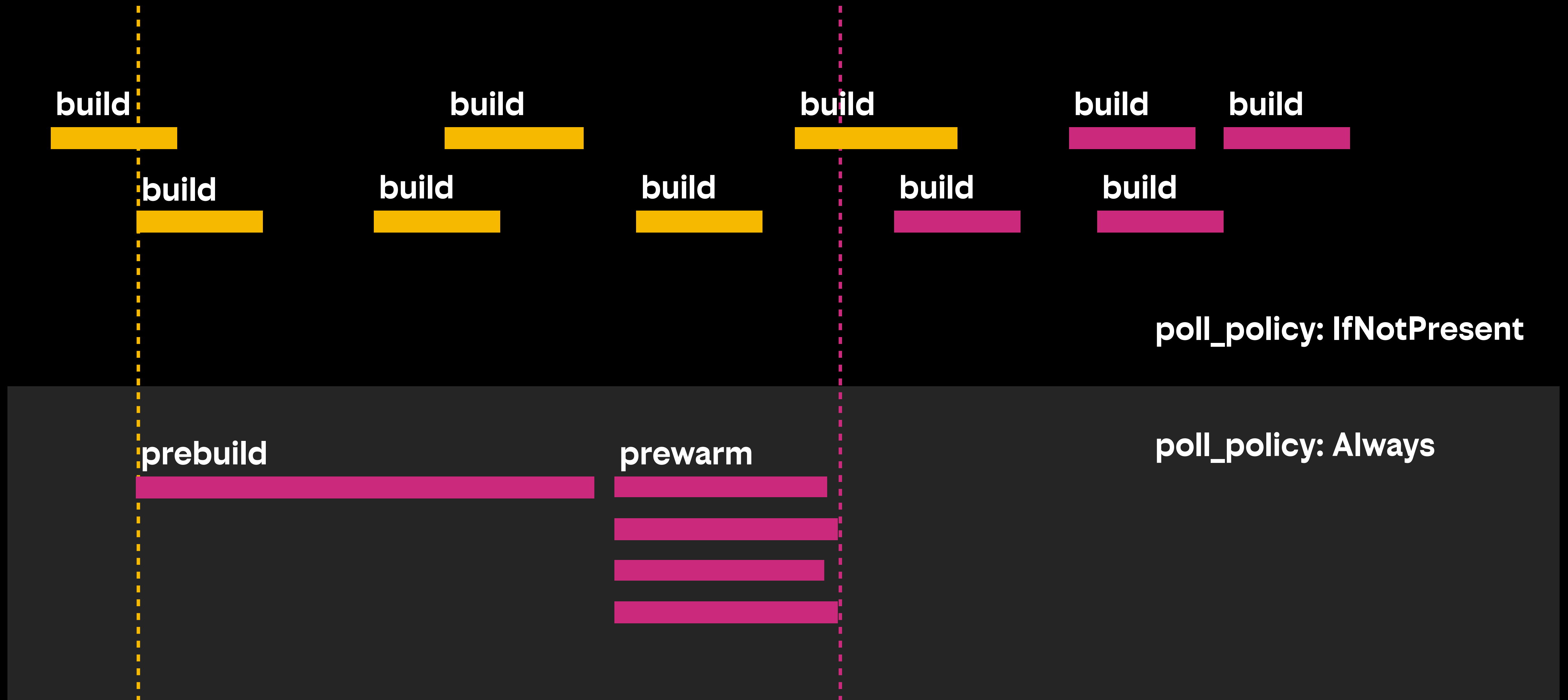
Таймлайн запуска job'ов



Таймлайн запуска job'ов



Pull policy





Нужен base-images: runtime



Копируем внутрь builder-a

node: 24-alpine

base-images: runtime

base-images: builder

node: 24-alpine

base-images: runtime



Упаковываем с ракетой



storage.conf

```
[storage]
```

```
driver = "overlay"
```

```
graphroot = "/var/lib/containers/storage"
```

```
runroot = "/var/lib/containers/run"
```

```
[storage.options]
```

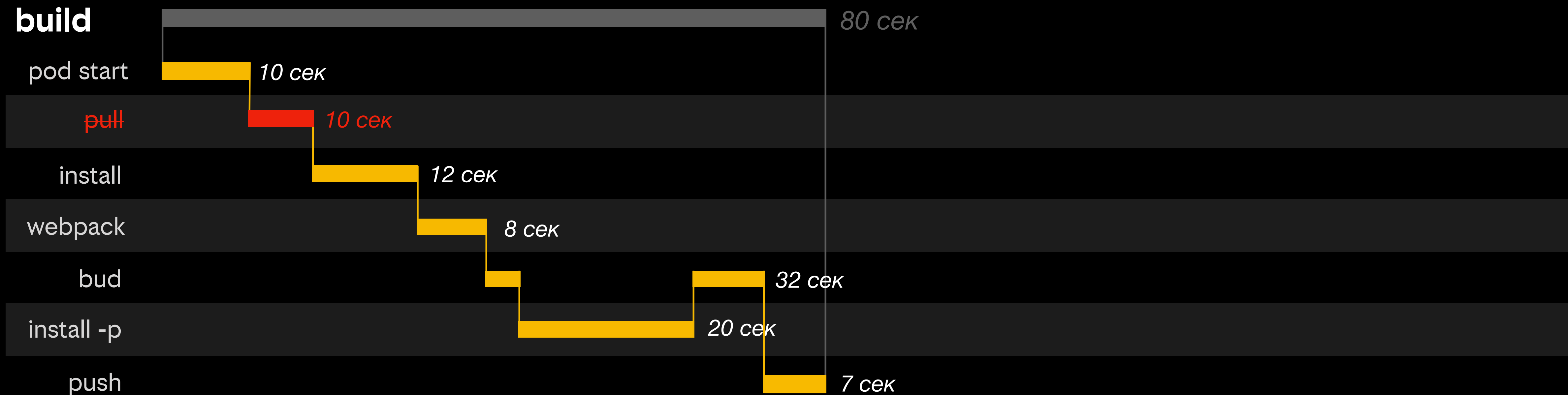
```
additionalimagestores = ["/var/lib/containers/additional"]
```

Добавили runtime в builder

	Builder	Runtime	Final
До начала работы	-	-	575
Сборка базовых образов	900	541	576 +1
Cache перемещен в builder	871 -29	271 -270	331 -245
Удалены LFS-объекты из .git	706 -165	271	331
Удалены LFS-объекты	541 -165	236 -35	296 -35
Зачистка модулей	434 -107	165 -71	225 -71
Добавлен базовый образ	604 +170	165	225

Ожидания: pull больше не нужен

-10 сек
Итого: 80 сек



Реальность: стали падать Job'ы

```
252 [1/3] STEP 12/14: RUN ci-bin 'ci-postinstall' 'ci-ccmd yarn prisma generate'
253 ci-ccmd yarn prisma generate
254 [1/3] STEP 13/14: USER node
255 [1/3] STEP 14/14: RUN ci-ccmd RUN_ENV=ci FORCE_COLOR=1 yarn install --production --prefer-offline --frozen-lockfile
> 256 [CMD] RUN_ENV=ci FORCE_COLOR=1 yarn install --production --prefer-offline --frozen-lockfile
✓ 407 Cleaning up project directory and file based variables
408 WARNING: Event retrieved from the cluster: Node is not ready
409 ERROR: Job failed: pod "runner-7hmmd8hxp-project-7091-concurrent-0-apdyhr4x" status is "Failed"
```

Два дня отладки спустя

- начал подозревать проблему в раннерах

Два дня отладки спустя

- начал подозревать проблему в раннерах
- обратился в поддержку - почитать метрики



Смирнов Сергей



YOU DESTROYED THE FABRIC OF SPACETIME

Два дня отладки спустя

- начал подозревать проблему в раннерах
- обратился в поддержку - почитать метрики
- каждая ошибка перезагружала коммунальный worker

Два дня отладки спустя

- начал подозревать проблему в раннерах
- обратился в поддержку - почитать метрики
- каждая ошибка перезагружала коммунальный worker
- за 2 дня, вызвал около 40 перезагрузок

Два дня отладки спустя

- начал подозревать проблему в раннерах
- обратился в поддержку - почитать метрики
- каждая ошибка перезагружала коммунальный worker
- за 2 дня, вызвал около 40 перезагрузок
- 20 перезагрузок спустя - ко мне постучали



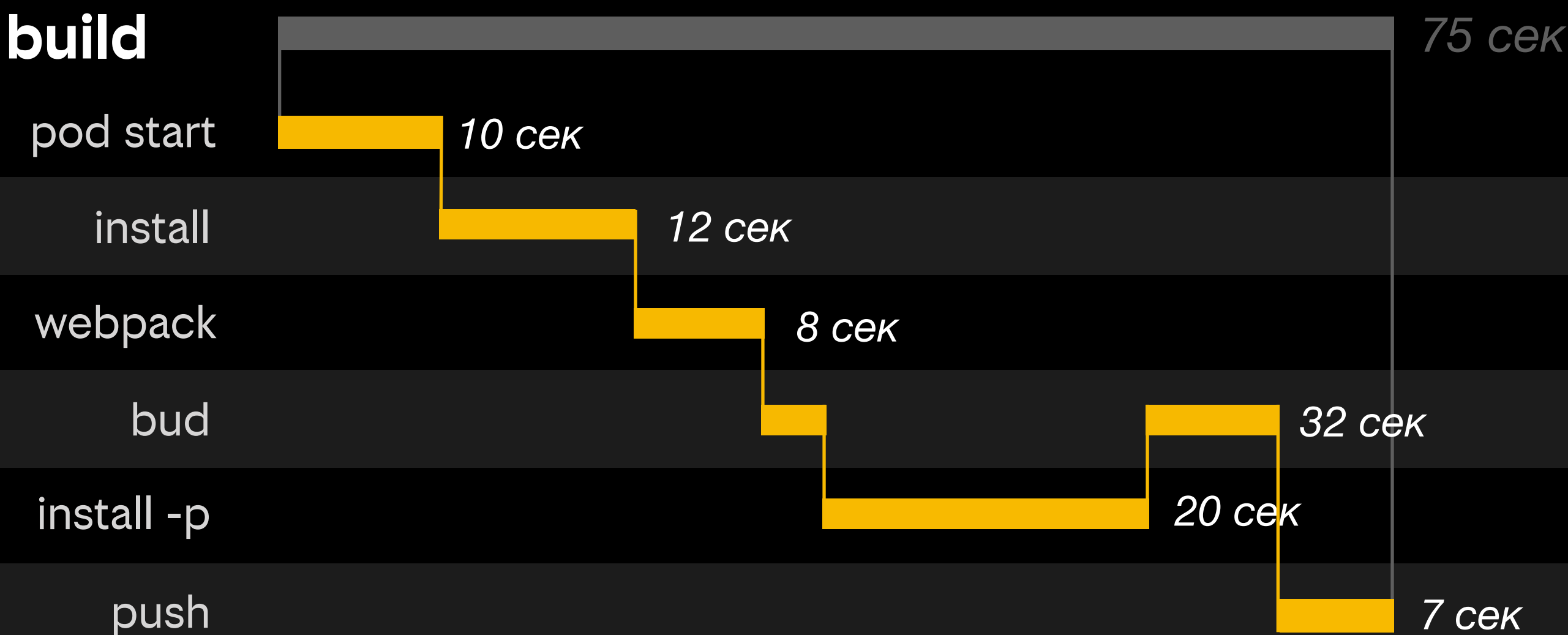
Смирнов Сергей

Виноват metacopy=on

- Базовый образ в дополнительном хранилище
- Сборка в основном хранилище

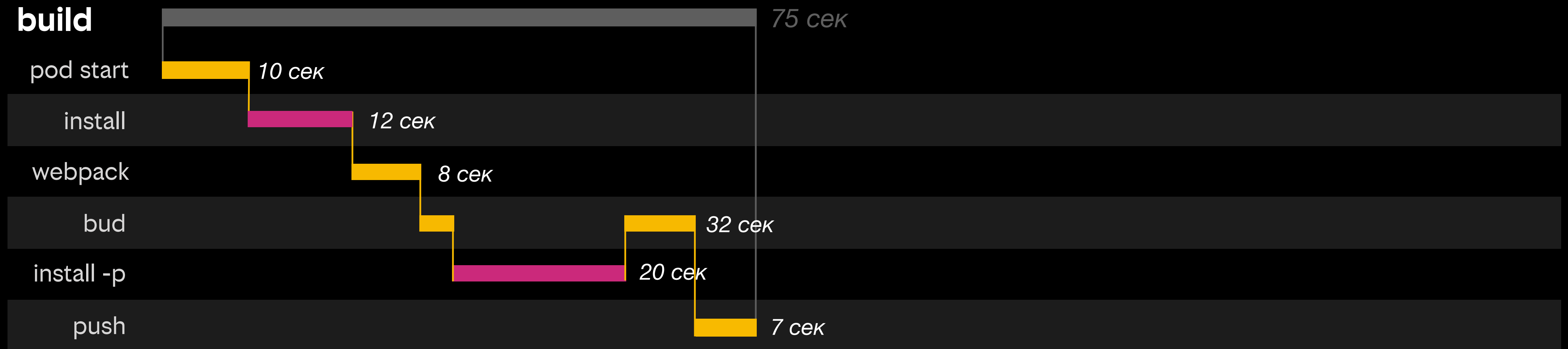
Build на базовом образе

Итого: 70 сек



Почему install - не секунда

Итого: 70 сек



Локальный запуск

```
yarn install v1.22.22
```

```
[1/4] 🔍 Resolving packages...
```

```
warning Resolution field "babel-watch@7.7.3" is incompatible with requested version "babel-watch@^7.8.1"
```

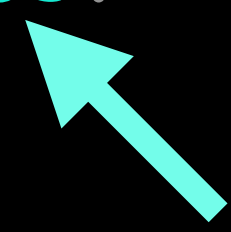
```
warning Resolution field "@beef/conf@2.0.0-alpha.325" is incompatible with requested version "@beef/conf@^1.3.3-alpha.198"
```

```
warning Resolution field "postcss-preset-env@10.2.0" is incompatible with requested version "postcss-preset-env@^10.2.1"
```

```
warning Resolution field "babel-watch@7.7.3" is incompatible with requested version "babel-watch@^7.8.1"
```

```
success Already up-to-date.
```

```
✨ Done in 0.95s.
```

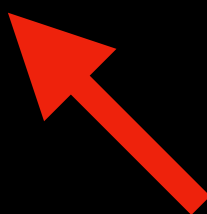


Запуск на стенде

```
[CMD] RUN_ENV=ci FORCE_COLOR=1 yarn install --production --prefer-offline --frozen-lockfile
yarn install v1.22.22
[1/4] Resolving packages...
warning Resolution field "babel-watch@7.7.3" is incompatible with requested version "babel-
watch@^7.8.1"
warning Resolution field "@beef/conf@2.0.0-alpha.325" is incompatible with requested version
"@beef/conf@^1.3.3-alpha.198"
warning Resolution field "postcss-preset-env@10.2.0" is incompatible with requested version
"postcss-preset-env@^10.2.1"
warning Resolution field "babel-watch@7.7.3" is incompatible with requested version "babel-
[2/4] Fetching packages...
[3/4] Linking dependencies...
warning "postcss-preset-env > @csstools/postcss-cascade-layers@5.0.2" has unmet peer dependency
"postcss@^8.4".
warning "postcss-preset-env > @csstools/postcss-color-function@4.0.12" has unmet peer dependency

... ещё 200 строк

[4/4] Building fresh packages...
✨ Done in 24.96s.
```





07. Порядок и особенности выполнения yarn install



07. Порядок и особенности выполнения `yarn install`

- При выполнении команды `yarn install`, происходит сопоставление `node_modules/.yarn-integrity` с текущими `yarn.lock` и `package.json`



07. Порядок и особенности выполнения `yarn install`

- При выполнении команды `yarn install`, происходит сопоставление `node_modules/.yarn-integrity` с текущими `yarn.lock` и `package.json`
- В случае обнаружения несовпадений или использовании флагов `--check-files` / `--force` выполняется полная валидация зависимостей: каждый модуль из `node_modules` **сопоставляется** с сохраненной в `yarn cache` версией



07. Порядок и особенности выполнения `yarn install`

- При выполнении команды `yarn install`, происходит сопоставление `node_modules/.yarn-integrity` с текущими `yarn.lock` и `package.json`
- В случае обнаружения несовпадений или использовании флагов `--check-files` / `--force` выполняется полная валидация зависимостей: каждый модуль из `node_modules` **сопоставляется** с сохраненной в `yarn cache` версией
- Если в кэше пусто, то скачивается и распаковывается `tarball`



07. Порядок и особенности выполнения `yarn install`

- При выполнении команды `yarn install`, происходит сопоставление `node_modules/.yarn-integrity` с текущими `yarn.lock` и `package.json`
- В случае обнаружения несовпадений или использования флагов `--check-files` / `--force` выполняется полная валидация зависимостей: каждый модуль из `node_modules` **сопоставляется** с сохраненной в `yarn cache` версией
- Если в кэше пусто, то скачивается и распаковывается `tarball`
- Каждый файл пакета сравнивается с соответствующим файлом в кэше по двум критериям: размеру (**size**) и времени изменения (**mtime**)



07. Порядок и особенности выполнения `yarn install`

- При выполнении команды `yarn install`, происходит сопоставление `node_modules/.yarn-integrity` с текущими `yarn.lock` и `package.json`
- В случае обнаружения несовпадений или использования флагов `--check-files` / `--force` выполняется полная валидация зависимостей: каждый модуль из `node_modules` **сопоставляется** с сохраненной в `yarn cache` версией
- Если в кэше пусто, то скачивается и распаковывается `tarball`
- Каждый файл пакета сравнивается с соответствующим файлом в кэше по двум критериям: размеру (**size**) и времени изменения (**mtime**)
- В случае **несовпадения** файл переписывается из кэша в `node_modules` проекта

■ Не совпадают `mtime` и `size`. Почему?

■ Внешний кэш, установка в контейнере

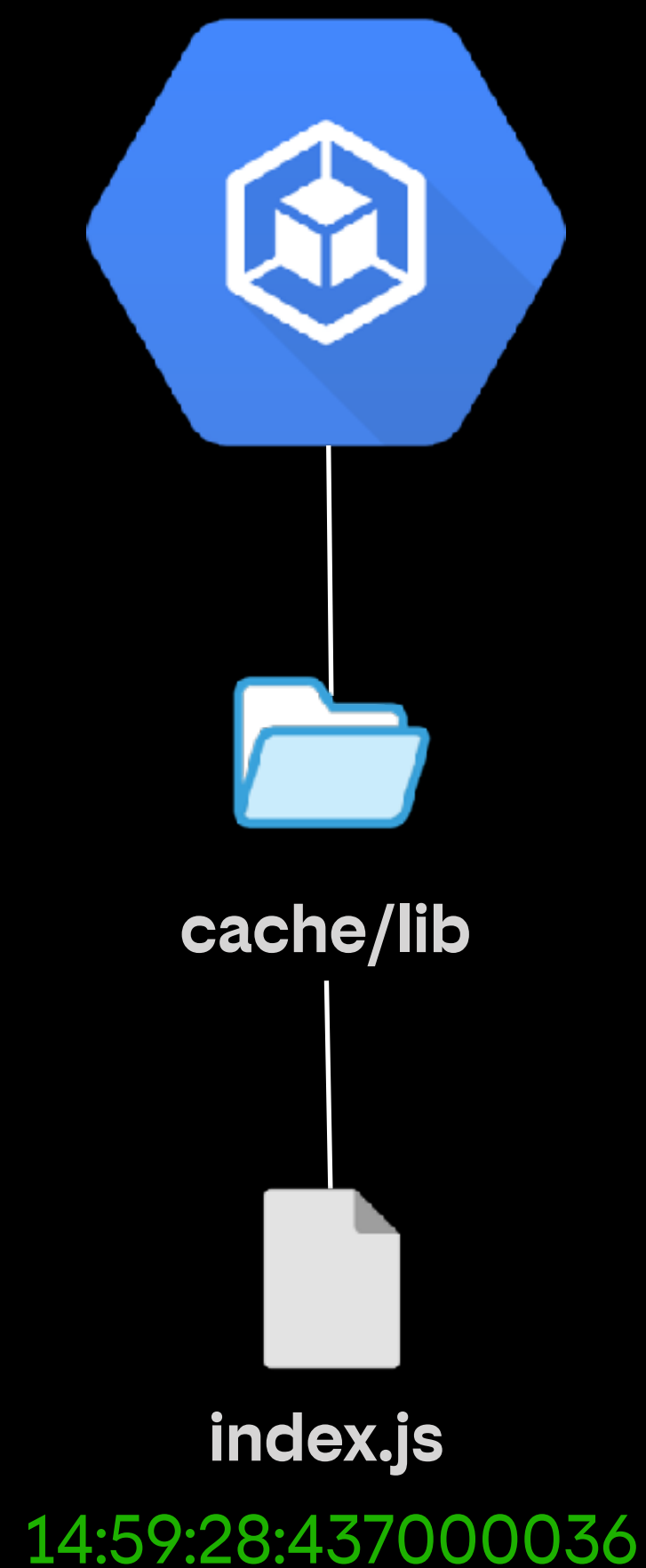
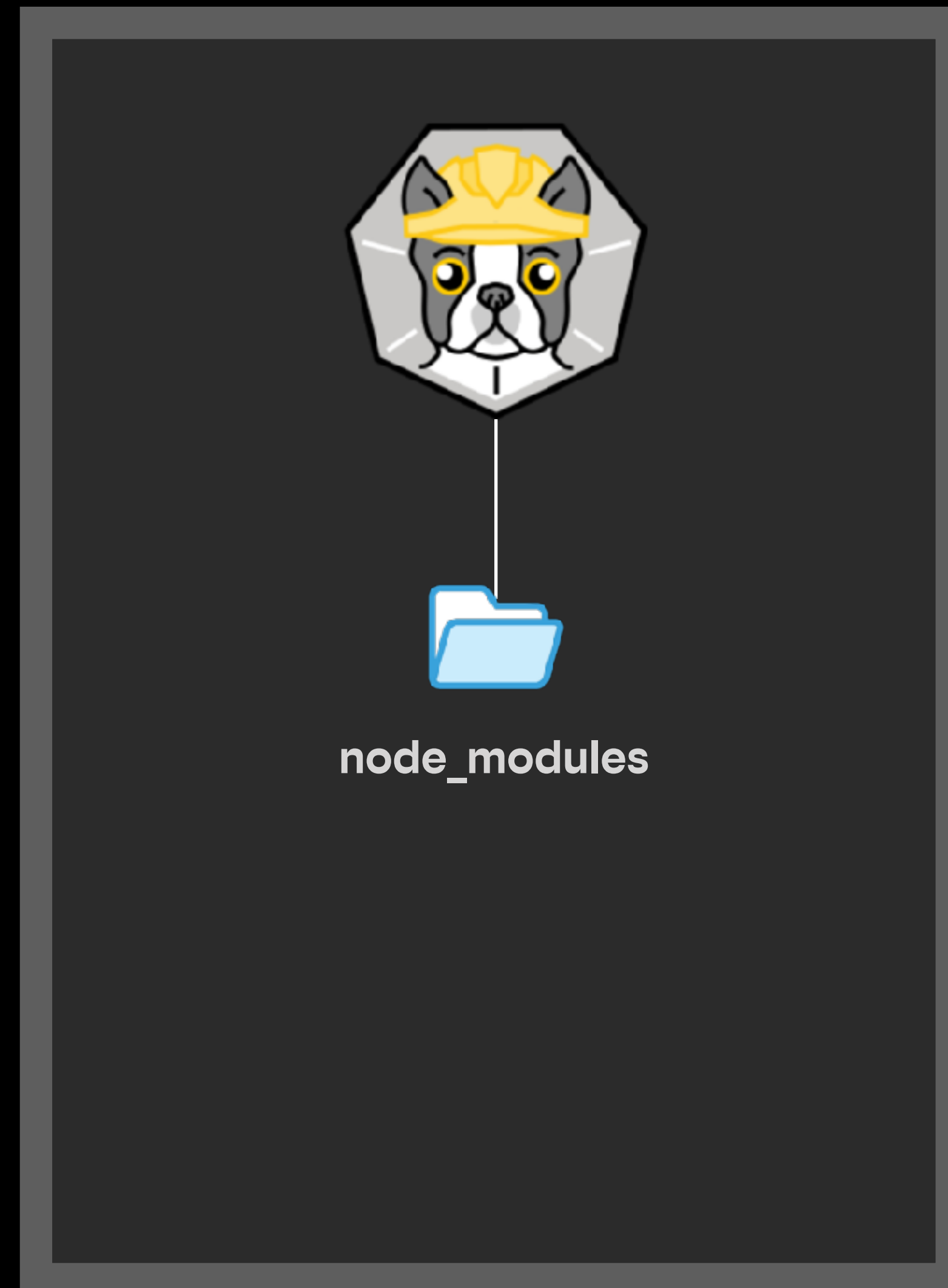


cache

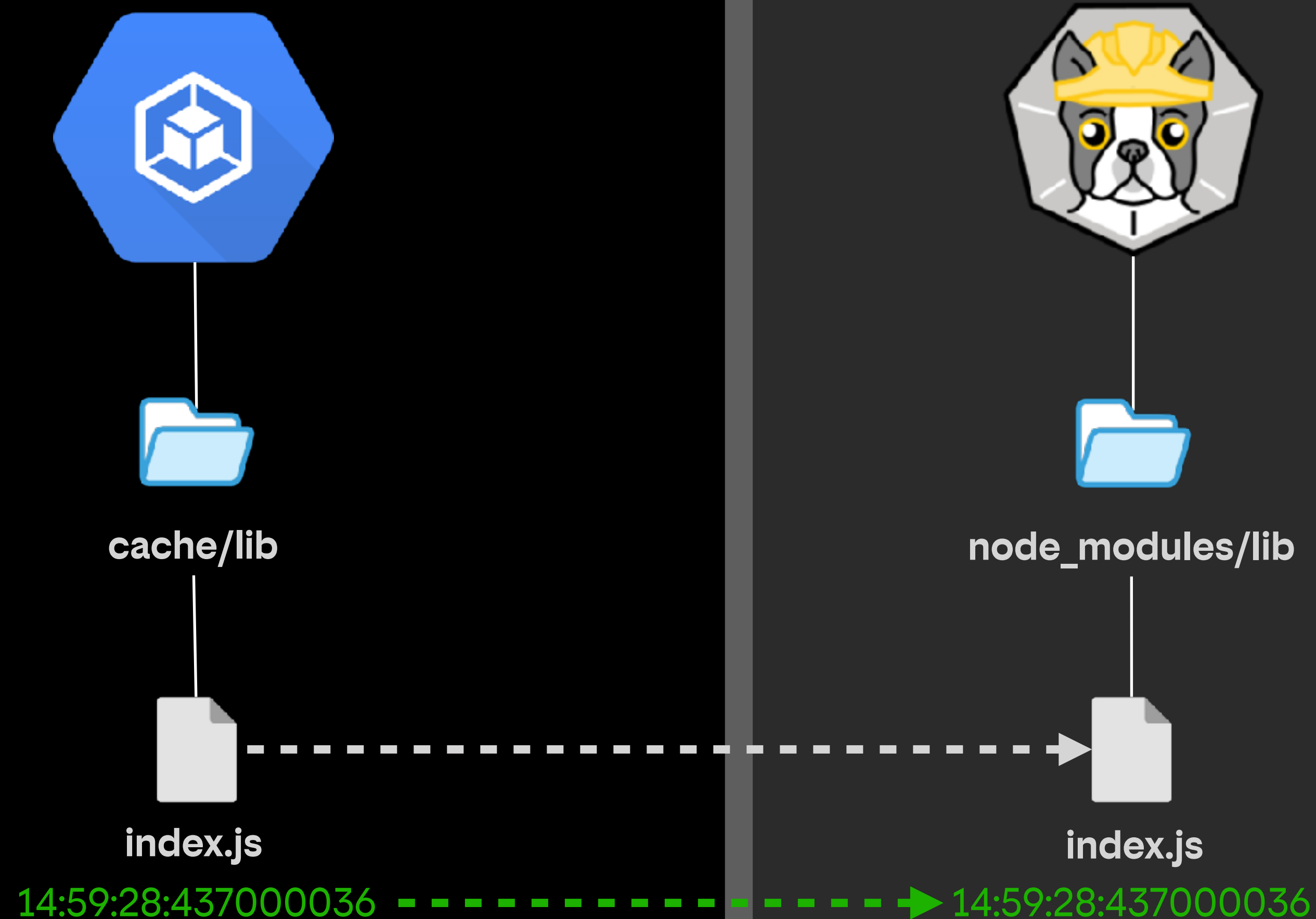


node_modules

Библиотека скачивается в кэш

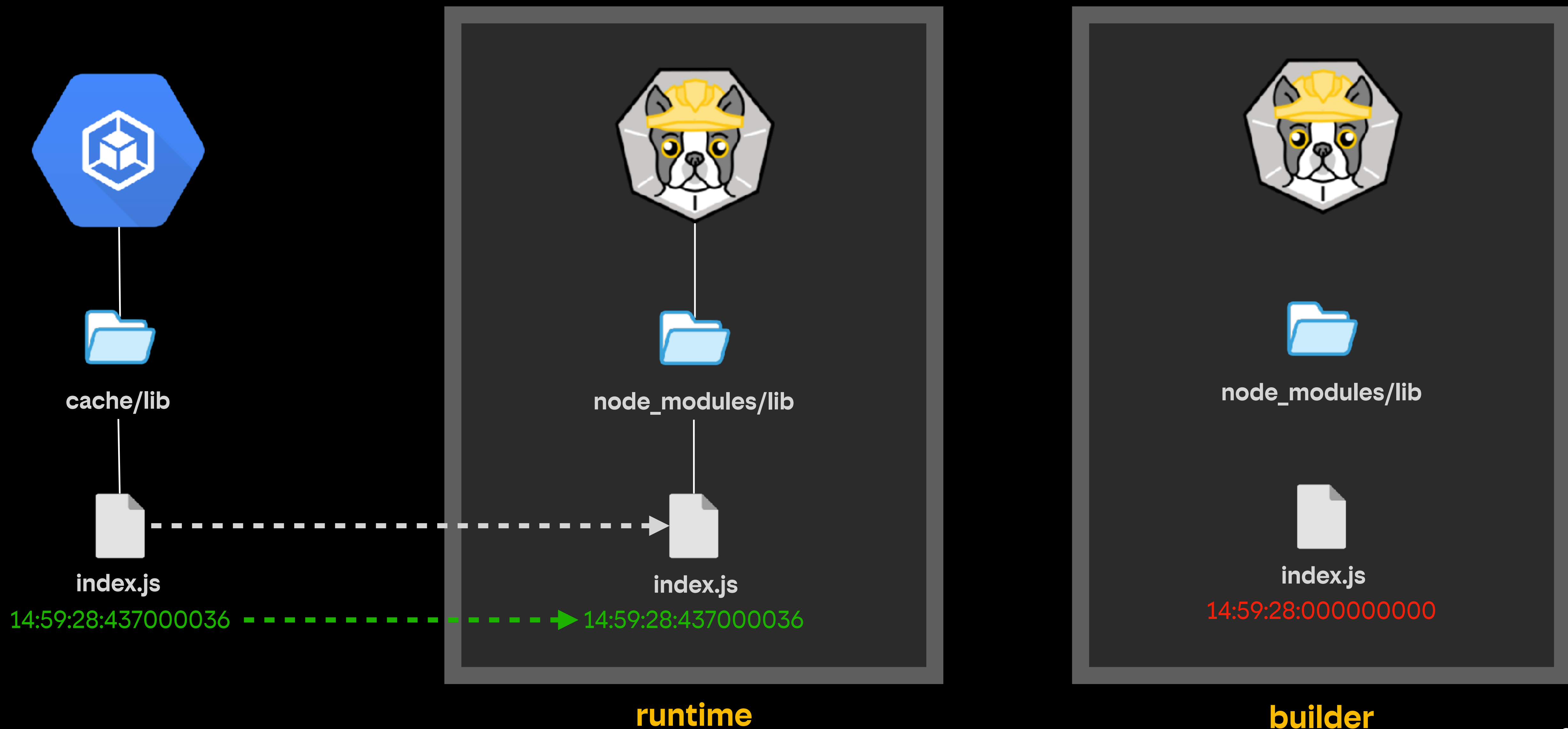


Библиотека копируются из кэша

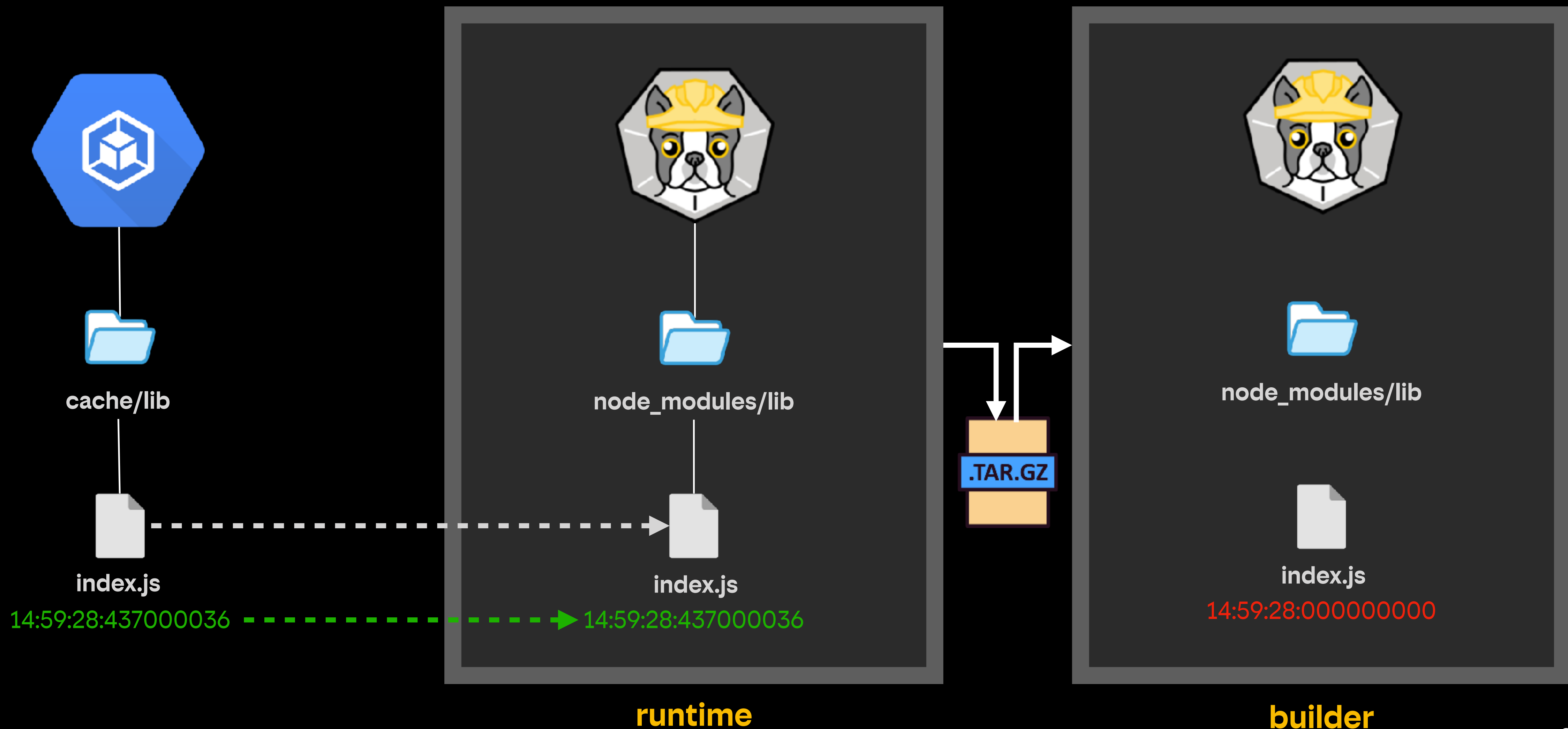


runtime

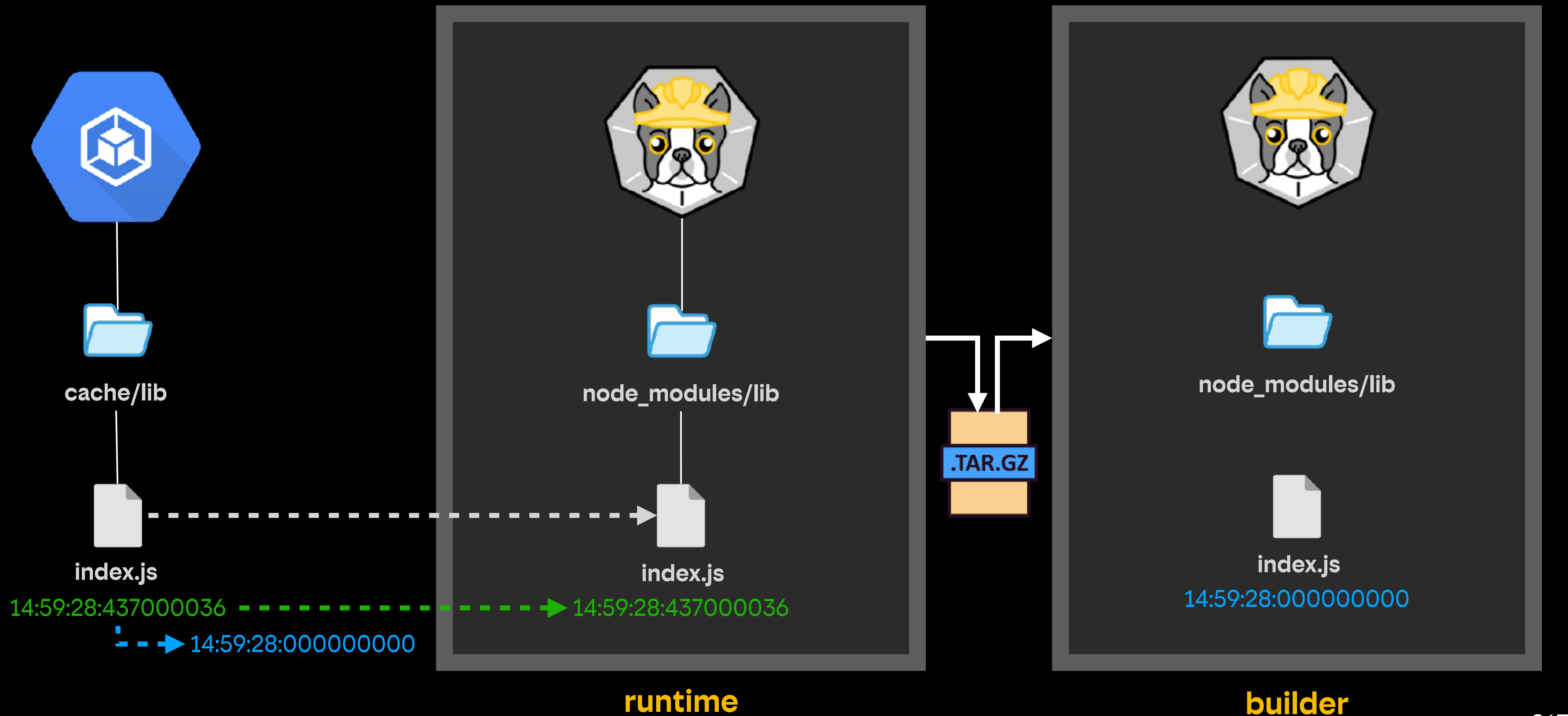
Новый build на базе runtime



Новый build на базе runtime



Простой фикс



Выравниваем кэш

- Пофайловый touch с округлением - 2 минуты

Выравниваем кэш

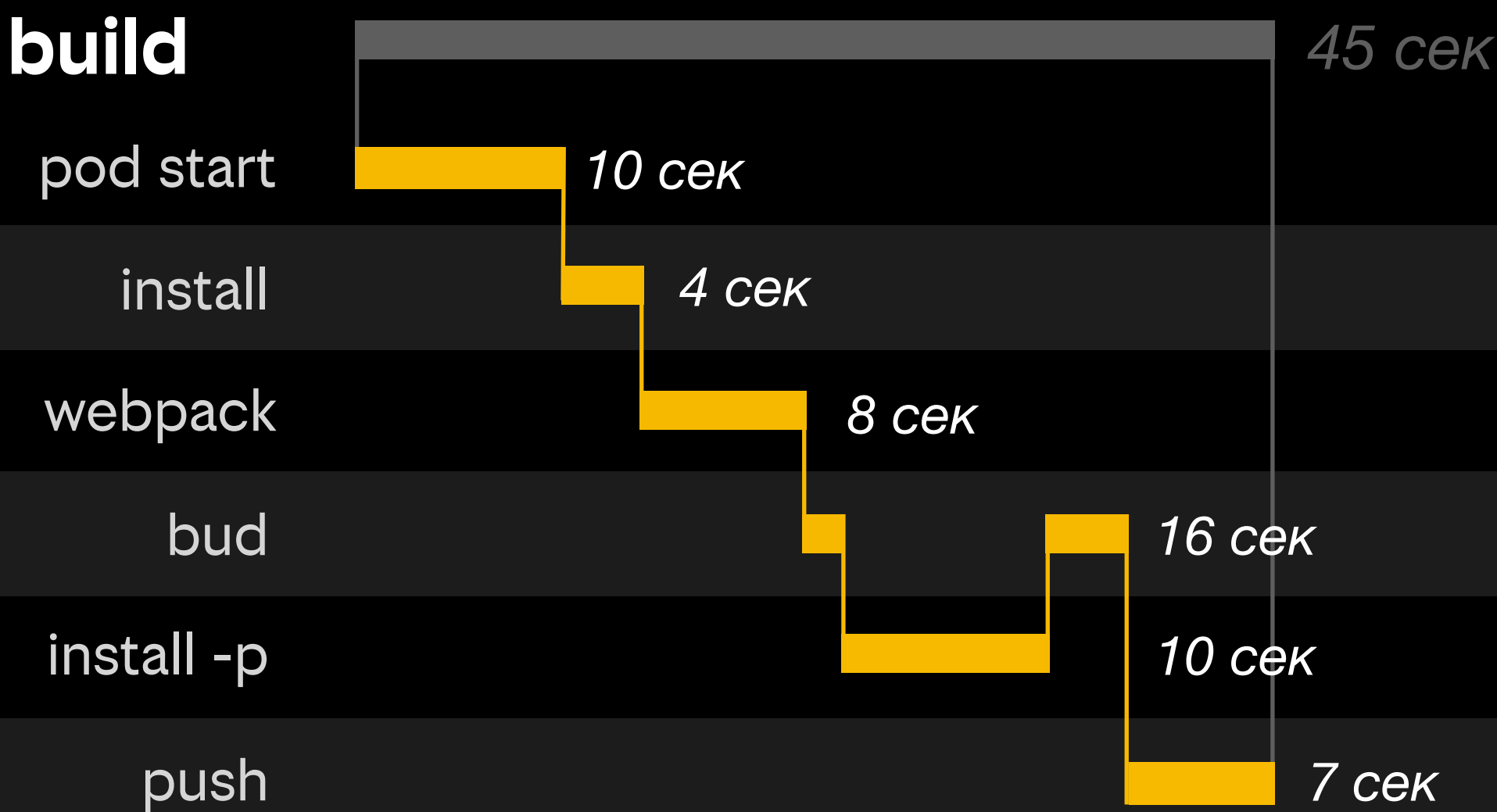
- Пофайловый touch с округлением - 2 минуты
- Массовый поиск и массовый touch - 2 минуты

Выравниваем кэш

- Пофайловый touch с округлением - 2 минуты
- Массовый поиск и массовый touch - 2 минуты
- Tape Archive - 15 секунд

Финальный тайминг

Итого: 45 сек



Финальный размер

	Builder	Runtime	Final
До начала работы	-	-	575
Сборка базовых образов	900	541	576 +1
Cache перемещен в builder	871 -29	271 -270	331 -245
Удалены LFS-объекты из .git	706 -165	271	331
Удалены LFS-объекты	541 -165	236 -35	296 -35
Зачистка модулей	434 -107	165 -71	225 -71
Добавлен базовый образ	604 +170	165	225
Исправлен кэш и удалены npm tarball	507 -97	165	200 -25

Часть V

Ограника решения

Заглянем в кишки

ВХОД ОКИСЛИТЕЛЯ
Жидкий кислород (LCx)

ТУРБОНАСОС ОКИСЛИТЕЛЯ
Повышает давление и подаёт кислород в камеру сгорания

ТУРБИНА
Приводит в движение турбокомпрессоры

ГЛАВНЫЙ КЛАПАН ОКИСЛИТЕЛЯ
Регулирует поток кислорода в камеру сгорания

ОХЛАЖДАЮЩАЯ РУБАШКА
Жидкое топливо циркулирует вокруг камеры сгорания, охлаждая стенки

ВХОД ГОРЮЧЕГО
Керосин (RP-1)

ТУРБОНАСОС ГОРЮЧЕГО
Повышает давление и подаёт горючее в камеру сгорания

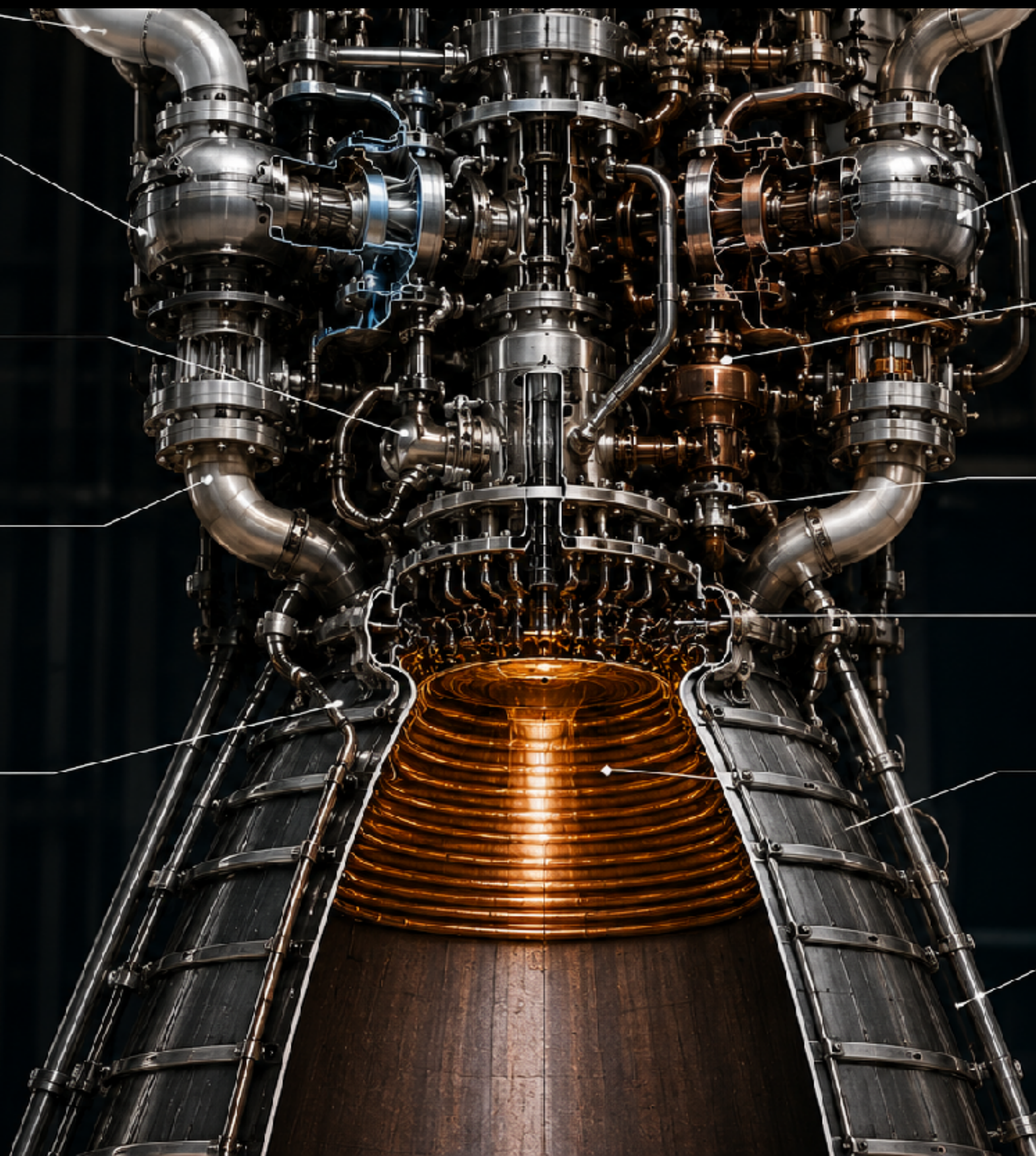
ГАЗОГЕНЕРАТОР
Сжигает часть топлива с кислородом для привода турбины

ГЛАВНЫЙ КЛАПАН ГОРЮЧЕГО
Регулирует поток горючего в камеру сгорания

ИНЖЕКТОРНАЯ ГОЛОВКА
Распыление компонентов топлива и смешивание перед сгоранием

КАМЕРА СГОРАНИЯ
Здесь происходит сгорание топлива и формирование высокотемпературных газов

СОПЛО
Расширяет газы, увеличивая скорость истечения и создавая тягу



Как писать чистый код



Иннополис

17-18 апреля



ДО ЧИСТОГО И ПОНЯТНОГО КОДА
ВАС ОТДЕЛЯЕТ ВСЕГО ОДИН ПРИНЦИП



билайн

Начальник отдела

Малюгин Иван

*Чистый код — это код, написанный с заботой о том,
чтобы его было легко читать и сопровождать*



Как писать чистый код



Иннополис

17-18 апреля



ДО ЧИСТОГО И ПОНЯТНОГО КОДА
ВАС ОТДЕЛЯЕТ ВСЕГО ОДИН ПРИНЦИП



билайн

Начальник отдела

Малюгин Иван

Шакальное
видео

Чистый код — это код, написанный с заботой о том,
чтобы его было легко читать и сопровождать



■ Как писать чистый код ~ доносить суть

- от общего к частному

■ Как писать чистый код ~ доносить суть

- от общего к частному
 - без лишних деталей

Как писать чистый код ~ доносить суть

- от общего к частному
 - без лишних деталей
 - за минимальное число итераций

Как писать чистый код ~ доносить суть

- от общего к частному
 - без лишних деталей
 - за минимальное число итераций
 - целыми порциями

Как писать чистый код ~ доносить суть

- от общего к частному
 - без лишних деталей
 - за минимальное число итераций
 - целыми порциями
- на понятном языке

■ Единая точка входа скриптов

`/usr/local/bin/ci-tools`

■ Единая точка входа скриптов

ci-tools

■ От общего к частному

	buildah
	setup
	init
ci-tools	pipeline
	copy
	install
	util

■ Минимальное число итераций

ci-tools

buildah
setup
init

init
bud
pull
push

pipeline

rename
trigger

copy
install

buildah

util

normalize-ts

Минимальное число итераций

ci-tools

buildah
setup
init

pipeline

copy
install
util

init
bud
pull
push

rename
trigger

buildah
normalize-ts

Код ci-tools:



Структура скриптов

 /usr/local

 bin

 ci-tools

 lib

 ci-tools

 setup.sh

 init.sh

 buildah.sh

 copy.sh

 util.sh

 pipeline.sh

 install.sh

ci-tools buildah *

 /usr/local

 bin

 ci-tools

 lib

 ci-tools

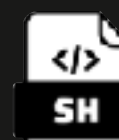
 setup.sh

 init.sh

 buildah.sh

 copy.sh

 util.sh

 pipeline.sh

 install.sh

 ci-buildah

 init.sh

 bud.sh

 pull.sh

 push.sh

 restore.sh

 backup.sh

Вспомогательные методы

 /usr/local

 bin

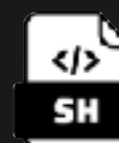
 ci-tools

 lib

 ci-tools

 setup.sh

 init.sh

 buildah.sh

 copy.sh

 util.sh

 pipeline.sh

 install.sh

 ci-buildah

 init.sh

 bud.sh

 pull.sh

 push.sh

 restore.sh

 backup.sh

 ci-helpers

 core.sh

 log.sh

 alases.sh

 gates.sh

Где хранить скрипты

- в репозитории проекта

Где хранить скрипты

- в репозитории проекта
- в коммунальном базовом образе

Где хранить скрипты

- в репозитории проекта
- в коммунальном базовом образе
- в коммунальной репозитории

Для установки скриптов нужно

- `git clone`
- `mv` в `/usr/*`
- `chown/chmod`
- подключить `ci-helpers`

Можно ли одной командой?

```
before_script:  
  - ci-tools setup
```

Где хранить скрипты

- Скрипты - в коммнуальном репозитории
- Установщик - в коммунальном базовом образе

ТОЛЬКО КОД ИНИЦИАЛИЗАЦИИ

 /usr/local

 bin

 ci-tools

 lib

 ci-tools

 setup.sh

 init.sh

 buildah.sh

 copy.sh

 util.sh

 pipeline.sh

 install.sh

 ci-buildah

 init.sh

 bud.sh

 pull.sh

 push.sh

 restore.sh

 backup.sh

 ci-helpers

 core.sh

 log.sh

 alases.sh

 gates.sh

ТОЛЬКО КОД ИНИЦИАЛИЗАЦИИ

 /usr/local

 bin

 ci-tools

 lib

 ci-tools

 setup.sh

 init.sh

 buildah.sh

 copy.sh

 util.sh

 pipeline.sh

 install.sh

 ci-buildah

 init.sh

 bud.sh

 pull.sh

 push.sh

 restore.sh

 backup.sh

 ci-helpers

 core.sh

 log.sh

 alases.sh

 gates.sh

Только код инициализации

 /usr/local

 bin

 ci-tools

 lib

 ci-tools

 setup.sh

 init.sh

 buildah.sh

 copy.sh

 util.sh

 pipeline.sh

 install.sh

 /usr/bin

 ci-tools

 ci-buildah

 init.sh

 bud.sh

 pull.sh

 push.sh

 restore.sh

 backup.sh

 ci-helpers

 core.sh

 log.sh

 alases.sh

 gates.sh

Порядок самоинициализации

- `/usr/bin/ci-tools setup` - скачивает скрипты в `/usr/local`
- `/usr/local/bin/ci-tools setup`

Вспомогательные методы в gitlab-ci

before_script:

- ci-tools setup

script:

- ci-tools version builder
- ci-buildah init
- ci-cmd yarn install



08. Особенности исполнения sh-скриптов



08. Особенности исполнения sh-скриптов

- При запуске shell-скрипта дочерний процесс получает **часть контекста родителя**: экспортированные переменные окружения, аргументы вызова, текущую директорию и стандартные дескрипторы. Внутреннее состояние shell (массивы, локальные переменные, алиасы) автоматически не передаётся.



08. Особенности исполнения sh-скриптов

- При запуске shell-скрипта дочерний процесс получает **часть контекста родителя**: экспортированные переменные окружения, аргументы вызова, текущую директорию и стандартные дескрипторы. Внутреннее состояние shell (массивы, локальные переменные, алиасы) автоматически не передаётся.
- Дочерний процесс выполняется в собственном контексте исполнения: изменения переменных, директорий и настроек внутри него не изменяют состояние родительского shell.



08. Особенности исполнения sh-скриптов

- При запуске shell-скрипта дочерний процесс получает **часть контекста родителя**: экспортированные переменные окружения, аргументы вызова, текущую директорию и стандартные дескрипторы. Внутреннее состояние shell (массивы, локальные переменные, алиасы) автоматически не передаётся.
- Дочерний процесс выполняется в собственном контексте исполнения: изменения переменных, директорий и настроек внутри него не изменяют состояние родительского shell.
- Ограничение изоляции дочернего процесса можно обойти через `source` или `eval`, выполняющих скрипт в контексте текущего shell, изменяя его состояние.



08. Особенности исполнения sh-скриптов

- При запуске shell-скрипта дочерний процесс получает **часть контекста родителя**: экспортированные переменные окружения, аргументы вызова, текущую директорию и стандартные дескрипторы. Внутреннее состояние shell (массивы, локальные переменные, алиасы) автоматически не передаётся.
- Дочерний процесс выполняется в собственном контексте исполнения: изменения переменных, директорий и настроек внутри него не изменяют состояние родительского shell.
- Ограничение изоляции дочернего процесса можно обойти через `source` или `eval`, выполняющих скрипт в контексте текущего shell, изменяя его состояние.
- В GitLab CI скрипты одной Job выполняются в рамках одного shell-контекста, поэтому состояние между командами сохраняется.



08. Особенности исполнения sh-скриптов

- При запуске shell-скрипта дочерний процесс получает **часть контекста родителя**: экспортированные переменные окружения, аргументы вызова, текущую директорию и стандартные дескрипторы. Внутреннее состояние shell (массивы, локальные переменные, алиасы) автоматически не передаётся.
- Дочерний процесс выполняется в собственном контексте исполнения: изменения переменных, директорий и настроек внутри него не изменяют состояние родительского shell.
- Ограничение изоляции дочернего процесса можно обойти через `source` или `eval`, выполняющих скрипт в контексте текущего shell, изменяя его состояние.
- В GitLab CI скрипты одной Job выполняются в рамках одного shell-контекста, поэтому состояние между командами сохраняется.
- В Dockerfile каждая инструкция RUN запускается в отдельном shell-контексте, поэтому состояние одного вызова не переносится в следующий.

Вспомогательные методы в gitlab-ci

before_script:

- eval "\$(ci-tools setup)"

script:

- ci-tools version builder
- ci-buildah init
- ci-cmd yarn install

Если вызвать без eval

```
$ ci-tools setup --branch=feature/LCP-2059
```

```
Installing ci-tools from branch feature/LCP-2059 of https://git.domain/b2c-pipeline/ci\_templates.git
```

```
Cloning into 'ci'...
```

```
Setup helpers
```

```
# Error: ci-tools setup must be piped to eval
```

```
# Run:
```

```
#   eval $(ci-tools setup --branch=feature/LCP-2059) instead
```

```
source /usr/local/lib/ci-helpers/core.sh
```

```
source /usr/local/lib/ci-helpers/aliases.sh
```

```
source /usr/local/lib/ci-helpers/gates.sh
```

```
source /usr/local/lib/ci-helpers/log.sh
```

Что видит eval

```
$ eval $(ci-tools setup --branch=feature/LCP-2059)
Installing ci-tools from branch feature/LCP-2059 of https://git.domain/b2c-pipeline/ci\_templates.git
Cloning into 'ci'...
Setup helpers
# Error: ci-tools setup must be piped to eval
# Run:
#   eval $(ci-tools setup --branch=feature/LCP-2059) instead
source /usr/local/lib/ci-helpers/core.sh
source /usr/local/lib/ci-helpers/aliases.sh
source /usr/local/lib/ci-helpers/gates.sh
source /usr/local/lib/ci-helpers/log.sh
```

Что видит eval

```
$ eval $(ci-tools setup --branch=feature/LCP-2059)
```

```
Installing ci-tools from branch feature/LCP-2059 of https://git.domain/b2c-pipeline/ci\_templates.git
```

```
Cloning into 'ci'...
```

```
Setup helpers
```

```
# Error: ci-tools setup must be piped to eval
```

```
# Run:
```

```
# eval $(ci-tools setup --branch=feature/LCP-2059) instead
```

```
source /usr/local/lib/ci-helpers/core.sh
```

```
source /usr/local/lib/ci-helpers/aliases.sh
```

```
source /usr/local/lib/ci-helpers/gates.sh
```

```
source /usr/local/lib/ci-helpers/log.sh
```

Код setup.sh



Жизненный цикл сборки

Скрипт	Фаза	Запуск из
RUN_ENV=ci yarn install --frozen-lockfile --prefer-offline --production	<i>prebuild</i>	<i>Dockerfile</i>
RUN_ENV=ci yarn install --frozen-lockfile --prefer-offline	<i>prebuild</i>	<i>Dockerfile</i>
RUN_ENV=ci yarn install --frozen-lockfile --prefer-offline	<i>build</i>	<i>gitlab-ci</i>
NODE_ENV=production yarn workspace @scope/pkg build	<i>build</i>	<i>gitlab-ci</i>
RUN_ENV=ci yarn install --frozen-lockfile --prefer-offline --production	<i>build</i>	<i>Dockerfile</i>
RUN_ENV=ci yarn prisma generate	<i>build-final</i>	<i>Dockerfile</i>
yarn prisma migrate deploy && node ./build/main.js	<i>runtime</i>	<i>service</i>
RUN_ENV=ci yarn prisma generate	<i>code quality</i>	<i>gitlab-ci</i>

Жизненный цикл сборки

Этап	Скрипт	Фаза
install --runtime	RUN_ENV=ci yarn install --frozen-lockfile --prefer-offline --production	<i>prebuild</i>
install	RUN_ENV=ci yarn install --frozen-lockfile --prefer-offline	<i>prebuild</i>
install	RUN_ENV=ci yarn install --frozen-lockfile --prefer-offline	<i>build</i>
build	NODE_ENV=production yarn workspace @scope/pkg build	<i>build</i>
install --runtime	RUN_ENV=ci yarn install --frozen-lockfile --prefer-offline --production	<i>build</i>
prepare	RUN_ENV=ci yarn prisma generate	<i>build-final</i>
deploy	yarn prisma migrate deploy && node ./build/main.js	<i>runtime</i>
prepare	RUN_ENV=ci yarn prisma generate	<i>code quality</i>

Жизненный цикл сборки

Этап

install --runtime

install

build

prepare

deploy

Методы жизненного цикла

Этап

ci-lifecycle install --runtime

ci-lifecycle install

ci-lifecycle build

ci-lifecycle prepare

ci-lifecycle deploy

По аналогии с ci-tools

 /usr/local

 bin

 ci-tools

 lib

 ci-tools

 setup.sh

 init.sh

 buildah.sh

 copy.sh

 util.sh

 pipeline.sh

 install.sh

Методы жизненного цикла

 /usr/local

 bin

 ci-tools

 ci-lifecycle

 lib

 ci-tools

 setup.sh

 init.sh

 buildah.sh

 copy.sh

 util.sh

 pipeline.sh

 install.sh

 /home/node/app

 ci-lifecycle

 install.sh

 build.sh

 prepare.sh

 deploy.sh

Коммунальные vs локальные

 /usr/local

 bin

 **ci-tools**

 **ci-lifecycle**

 lib

 **ci-tools**

 **<tools>.sh**

 **ci-lifecycle**

 **<hooks>.sh**

 /home/node/app

 **ci-tools**

 **<tools>.sh**

 **ci-lifecycle**

 **<hooks>.sh**

Что скрывает ci-tools setup

- `ci-tools setup` (bootstrap) - скачивает коммунальные скрипты
- `ci-tools setup` - инициализирует helper-ы
- `ci-tools init` - подготовка параметров и директорий
- `ci-lifecycle init` - git checkout в проекте
- `ci-lifecycle install` - установка зависимостей

Понижение пользователя

```
RUN ci-tools setup --from=/home/host/ci
```

```
RUN ci-tools copy --chown=node:node . .
```

```
USER node
```

```
RUN ci-lifecycle install --runtime
```

```
USER root
```

```
RUN ci-tools copy --exclude=.yarn-tarball.tgz ../cache ../cache
```


Понижение пользователя

```
RUN ci-tools setup --from=/home/host/ci
```

```
RUN ci-tools copy --chown=node:node . .
```

```
RUN ci-lifecycle install --runtime
```

```
RUN ci-tools copy --exclude=.yarn-tarball.tgz ../cache ../cache
```

Понижение пользователя

```
RUN ci-tools setup --from=/home/host/ci
```

```
RUN ci-tools copy --chown=node:node . .
```

```
RUN ci-lifecycle install --runtime
```

```
RUN ci-tools copy --exclude=.yarn-tarball.tgz ../cache ../cache
```

Код ci-node.sh



Скрываем ENV

Было

```
ENV BUILDAH_FORMAT=docker
```

```
ENV BUILDAH_ISOLATION=chroot
```

```
RUN apk add --no-cache buildah netavark
```

Стало

```
RUN ci-env install buildah
```


Скрываем ENV

- Пишем переменные в .env файл

■ Скрываем ENV

- Пишем переменные в .env файл
- Считываем их через source

Скрываем ENV

- Пишем переменные в .env файл
- Считываем их через source

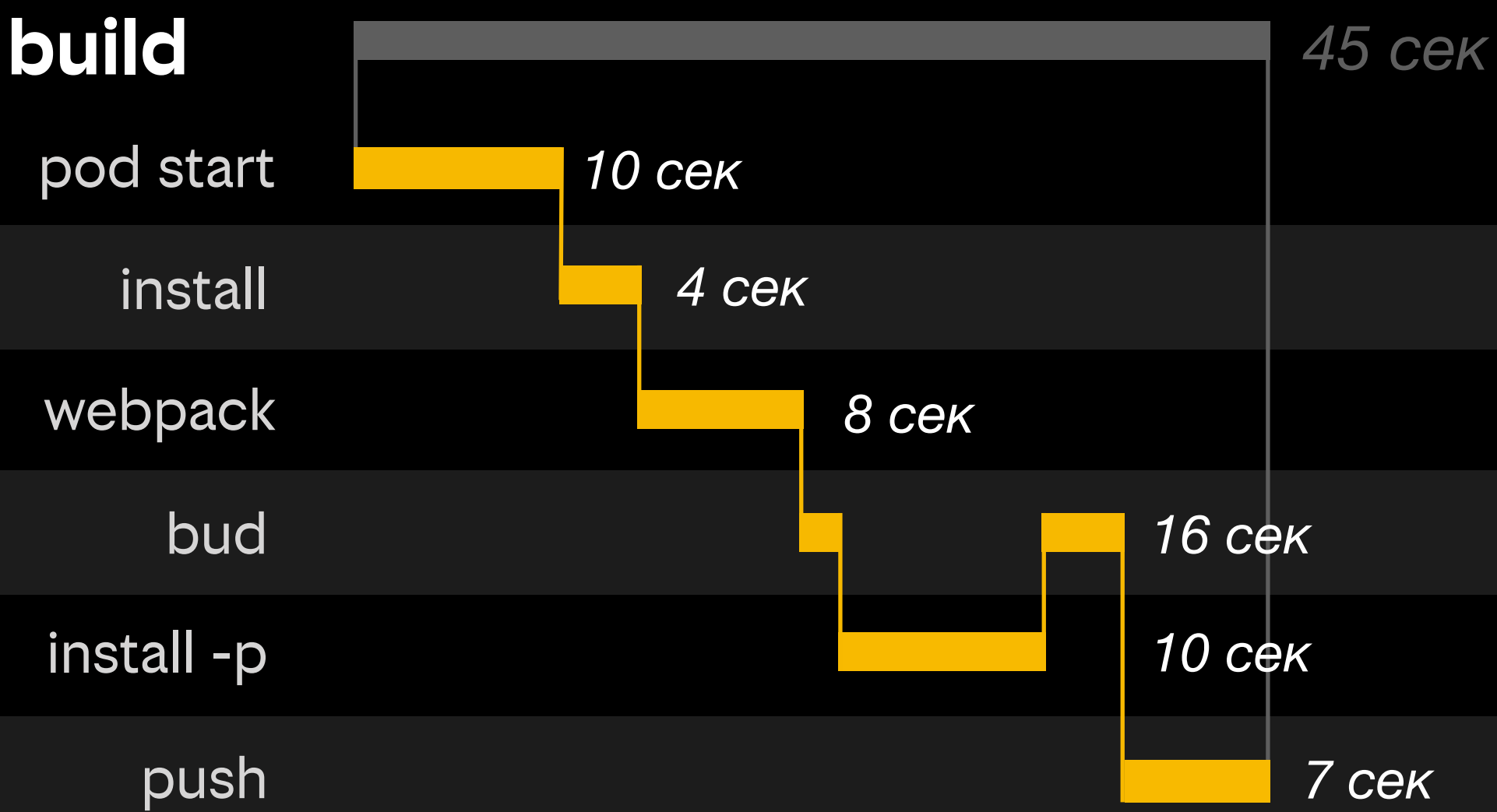
Код ci-env.sh



Итоги

Финальный тайминг

Итого: 45 сек



Финальный Dockerfile

```
ARG FROM=harbor.domain/common/b2c/node:24-runtime
```

```
# -----  
# Минимальная основа для запуска nodejs-приложения  
FROM $FROM AS runtime
```

```
RUN ci-tools setup --from=/home/host/ci  
RUN ci-tools copy --chown=node:node . .  
RUN ci-lifecycle install --runtime
```

```
# -----  
# Репозиторий + модули для запуска сборки  
FROM $FROM AS builder
```

```
COPY --chown=node:node . .  
RUN ci-lifecycle install  
RUN ci-tools copy --exclude=.yarn-tarball.tgz  
    ../cache ../cache  
RUN ci-tools install buildah  
RUN ci-tools buildah restore --additional  
VOLUME /var/lib/containers/storage
```

```
# -----  
# Сборка финального backend-образа  
FROM runtime as backend-runtime
```

```
RUN ci-lifecycle prepare  
USER node
```

```
ENTRYPOINT ["/bin/sh", "-c"]  
CMD ["ci-lifecycle deploy"]
```

```
# -----  
# Сборка финального nginx-fe-образа  
FROM docker.io/library/nginx:latest as nginx-runtime
```

```
COPY "packages/$APPLICATION/build/" /usr/share/nginx/$APPLICATION  
COPY nginx.conf /etc/nginx/nginx.conf
```


Финальный .gitlab-ci.yml (без rules/vars)

```
default:
  image: harbor.domain/project/base-images:builder
  before_script:
    - eval "$(ci-tools setup)"
stages:
  - build
  - publish
  - prebuild_runner
  - prewarm_runner

build:
  stage: build
  script:
    - ci-tools version
    - ci-tools pipeline rename "Building $VERSION" &
    - ci-lifecycle build
    - ci-buildah init
    - ci-buildah bud --target $BUILD_TARGET --from=base-images:runtime
    - ci-buildah push

publish:
  stage: publish
  needs: []
  script:
    - yarn ci-publish
```

Gitlab prebuild job

```
prebuild_runner:
  stage: prebuild
  image:
    name: harbor.domain/common/b2c/node:24-builder
    pull_policy: always
  allow_failure: true
  before_script:
    - eval "$(ci-tools setup --skip-install --from ref:3.62.0)"
  script:
    - ci-tools version builder
    - ci-tools pipeline rename "Rebuilding base image" &
    - ci-buildah init
    - ci-buildah bud --target runtime -t base-images:runtime
    - ci-tools util normalize-mtime ../cache &
    - ci-buildah backup; wait
    - ci-buildah bud --target builder -t base-images:builder --from=base-images:runtime
    - ci-buildah push base-images:builder
    - ci-buildah push base-images:runtime
  rules:
    - if: $CI_COMMIT_BRANCH != "dev"
      when: never
    - changes:
        - .gitlab-ci.yml
        - Dockerfile
        - yarn.lock
    - when: manual
```

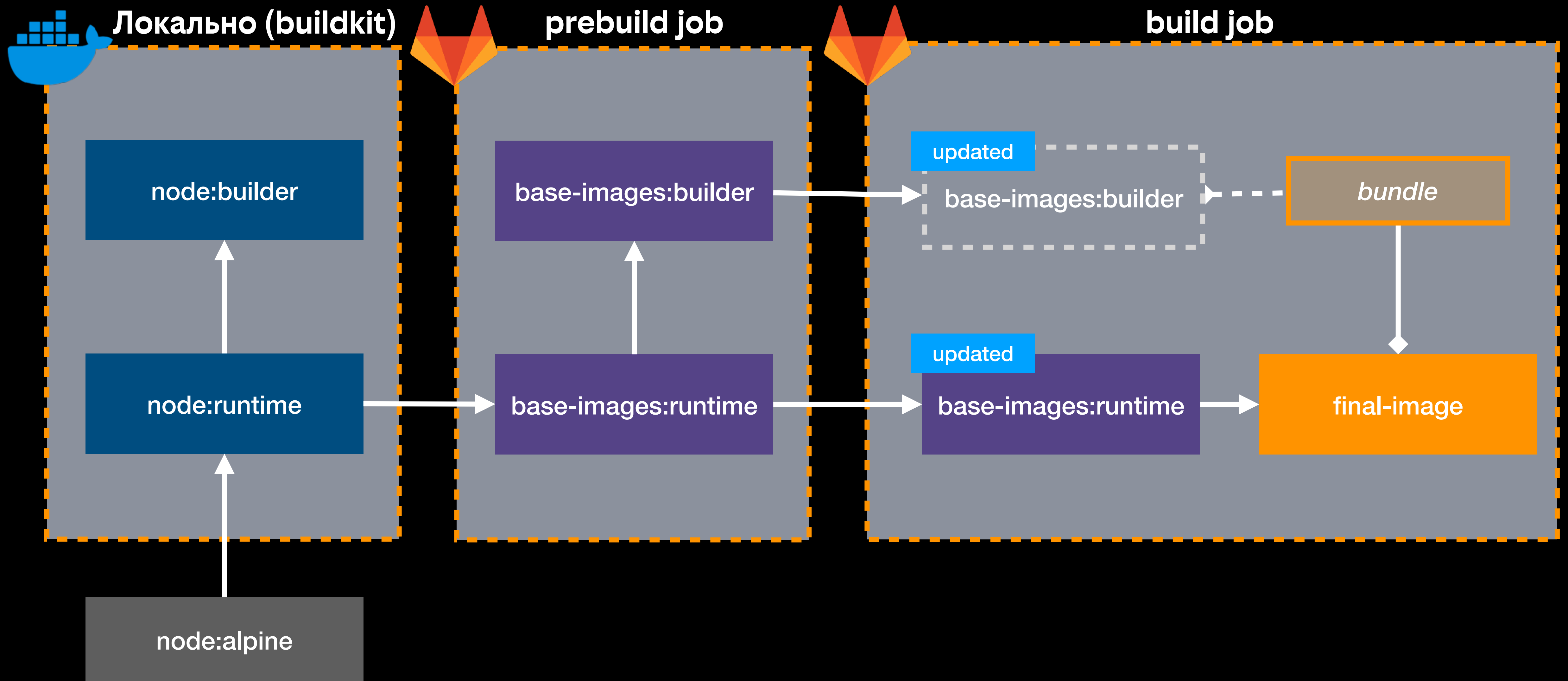
Gitlab prewarm job

```
prewarm_runner:  
  stage: prewarm  
  image:  
    name: harbor.domain/project/base-images:builder  
    pull_policy: always  
  allow_failure: true  
  variables:  
    KUBERNETES_NODE_SELECTOR_HOSTNAME: "kubernetes.io/hostname=community-gitlab-worker$WORKER"  
  script:  
    - echo "Worker hot and ready"  
  parallel:  
    matrix:  
      - WORKER: ["01", "02", "03", "04", "05"]
```

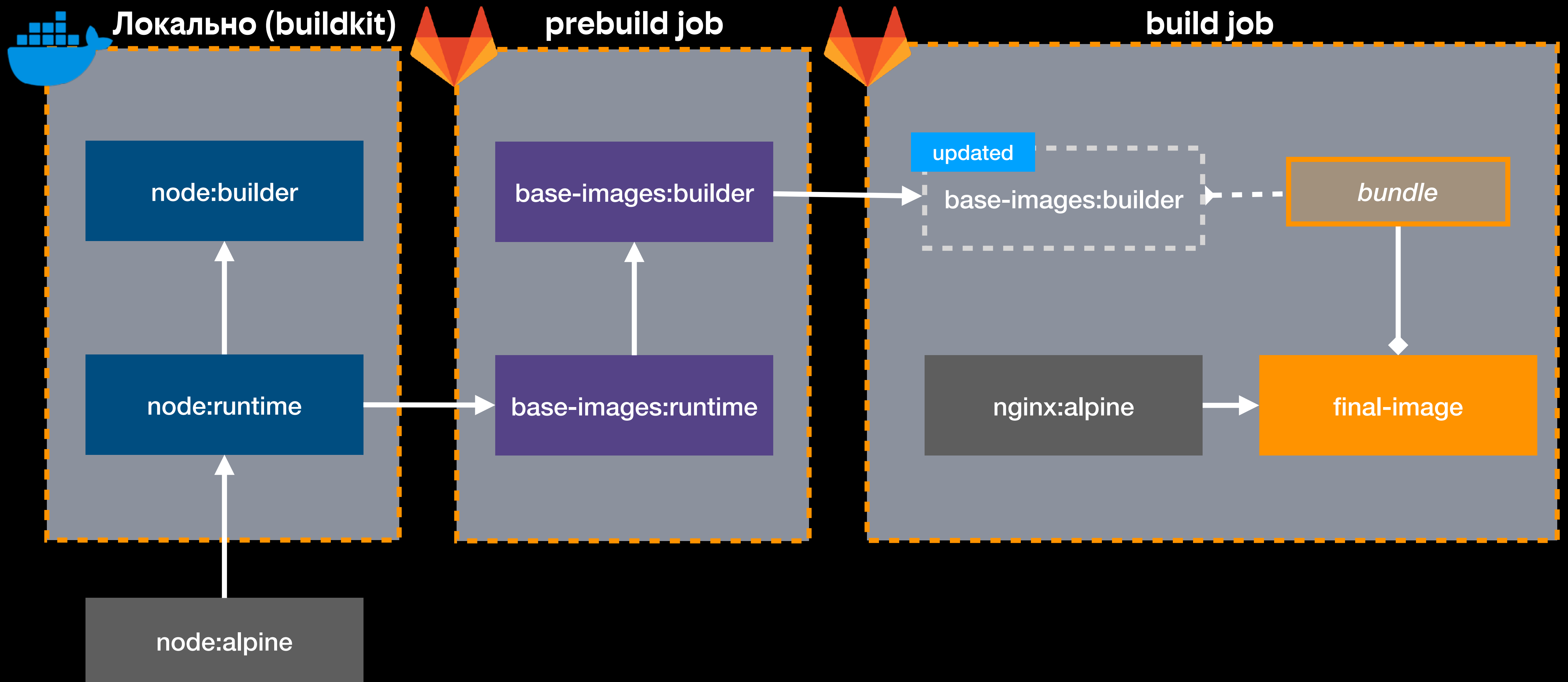
Финальный код:



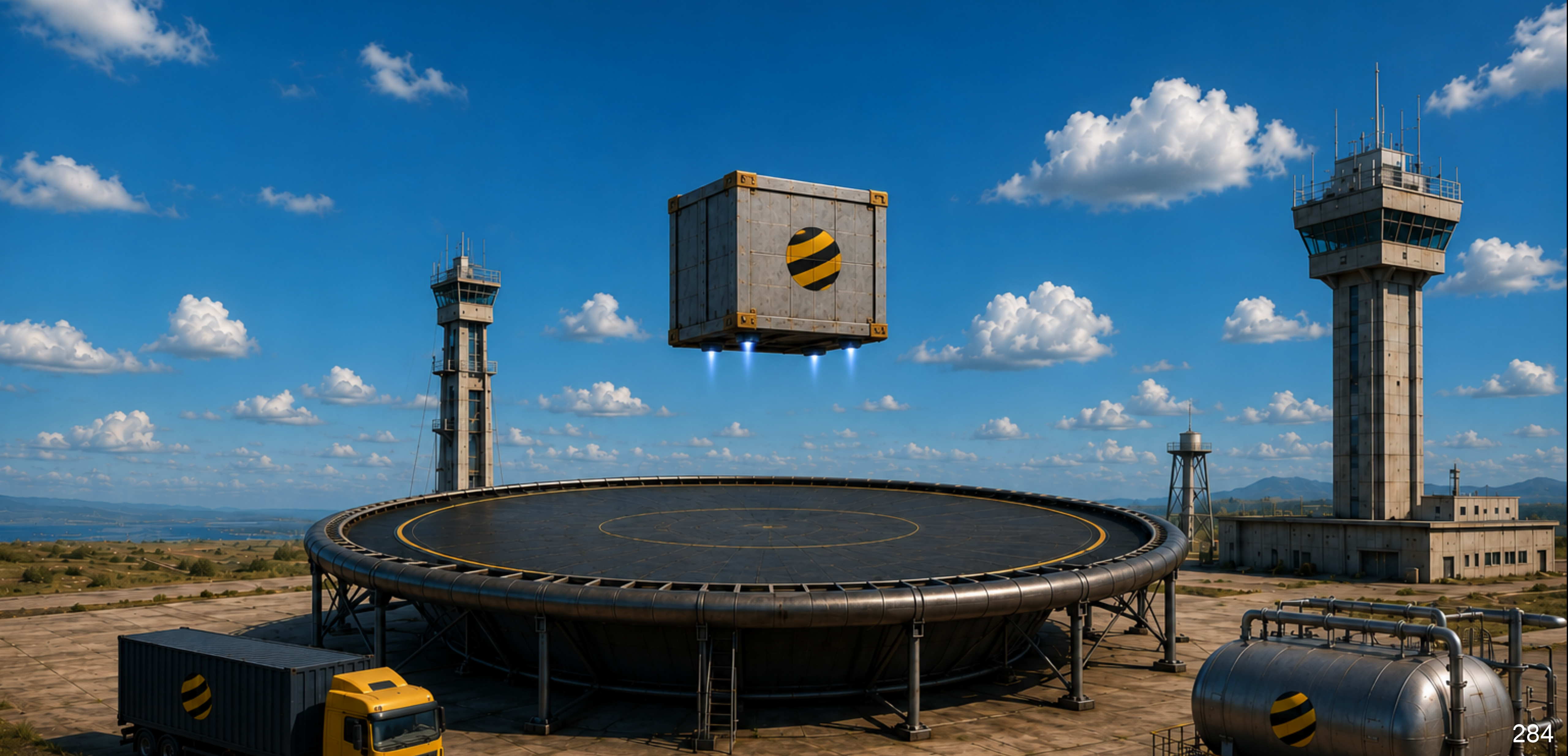
Весь процесс сборки сервиса



Весь процесс сборки фронта



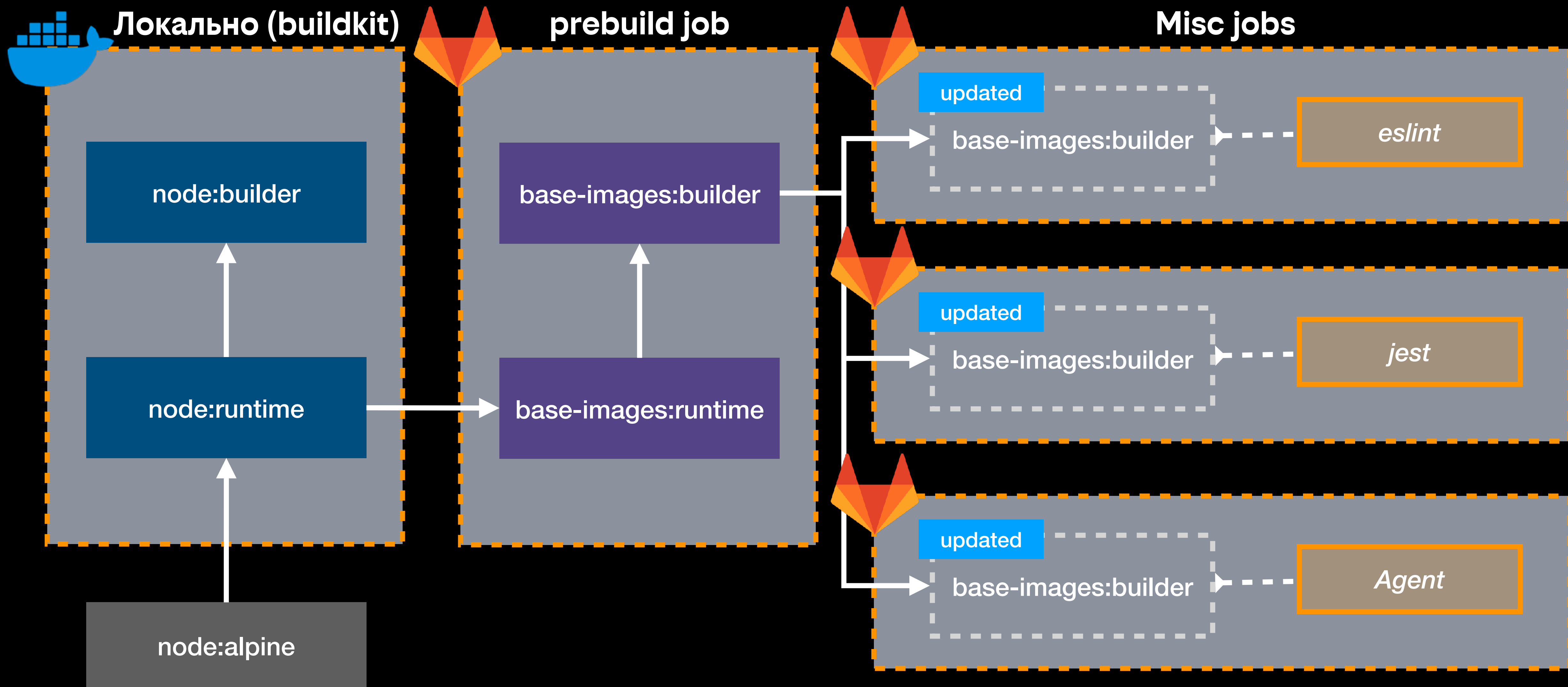
Любые задачи: 5-10 секунд



Любые задачи: 5-10 секунд

- тесты
- линтеры
- публикации
- агенты

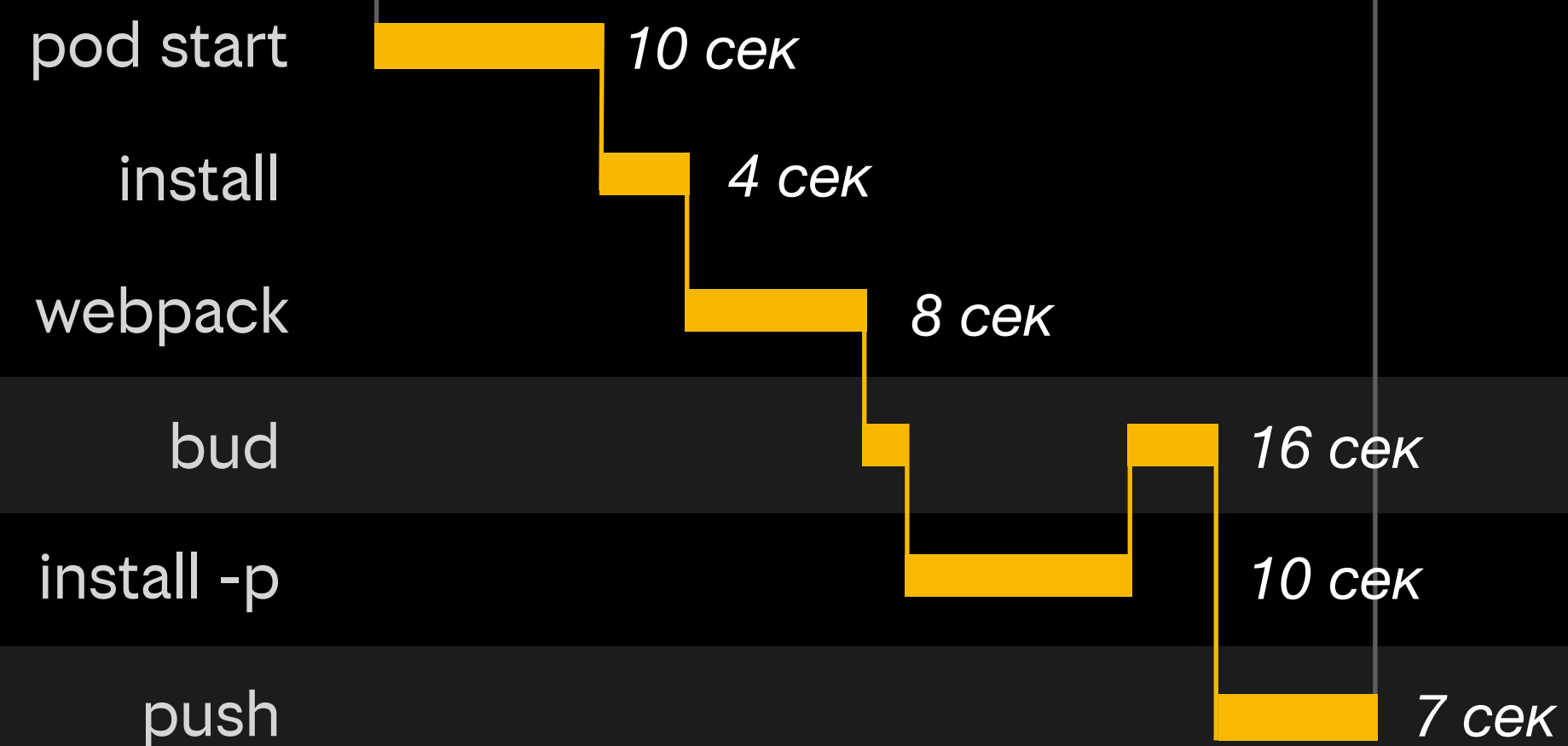
Любые задачи



Что с публикацией?

Итого: 45 сек

build



publish



Можно ли еще быстрее?

- а нужно ли?

Можно ли еще быстрее?

- а нужно ли?
- секунды - не минуты
- пользовательские оптимизации в ci-lifecycle
- горячие k8s поды
- job runtime-синхронизация

Плюсы

- экономим хранилище (до -90%)

Плюсы

- экономим хранилище (до -90%)
- экономим время (до -97%)

Плюсы

- экономим хранилище (до -90%)
- экономим время (до -97%)
- ускорился даже deploy

Плюсы

- экономим хранилище (до -90%)
- экономим время (до -97%)
- ускорился даже deploy
- в любых сборках

Плюсы

- экономим хранилище (до -90%)
- экономим время (до -97%)
- ускорился даже deploy
- в любых сборках
- расширяемость
- переиспользуемость
- масштабируемость

Сравним минусы с ws-директориями

- кончатся папки
- временные папки
- долго удалять
- lock при падении job'ов
- артефакты нестабильных сборок
- большие изменения в node_modules
- потребление диска
- нагрузка на диск
- много кода в коммунальных скриптах
- сложнее отладка
- документация
- растёт энтропия

Сравним минусы с ws-директориями

- ~~кончатся папки~~
- временные папки
- долго удалять
- lock при падении job'ов
- артефакты нестабильных сборок
- большие изменения в node_modules
- потребление диска
- нагрузка на диск
- много кода в коммунальных скриптах
- сложнее отладка
- документация
- растёт энтропия

Сравним минусы с ws-директориями

- ~~кончатся папки~~
- ~~временные папки~~
- долго удалять
- lock при падении job'ов
- артефакты нестабильных сборок
- большие изменения в node_modules
- потребление диска
- нагрузка на диск
- много кода в коммунальных скриптах
- сложнее отладка
- документация
- растёт энтропия

Сравним минусы с ws-директориями

- ~~кончатся папки~~
- ~~временные папки~~
- ~~долго удалять~~
- lock при падении job'ов
- артефакты нестабильных сборок
- большие изменения в node_modules
- потребление диска
- нагрузка на диск
- много кода в коммунальных скриптах
- сложнее отладка
- документация
- растёт энтропия

Сравним минусы с ws-директориями

- ~~кончатся папки~~
- ~~временные папки~~
- ~~долго удалять~~
- ~~lock при падении job'ов~~
- артефакты нестабильных сборок
- большие изменения в node_modules
- потребление диска
- нагрузка на диск
- много кода в коммунальных скриптах
- сложнее отладка
- документация
- растёт энтропия

Сравним минусы с ws-директориями

- ~~кончатся папки~~
- ~~временные папки~~
- ~~долго удалять~~
- ~~lock при падении job'ов~~
- ~~артефакты нестабильных сборок~~
- ~~большие изменения в node_modules~~
- потребление диска
- нагрузка на диск
- много кода в коммунальных скриптах
- сложнее отладка
- документация
- растёт энтропия

Сравним минусы с ws-директориями

- 
- ~~кончатся папки~~
 - ~~временные папки~~
 - ~~долго удалять~~
 - ~~lock при падении job'ов~~
 - ~~артефакты нестабильных сборок~~
 - ~~большие изменения в node_modules~~
 - потребление диска
 - нагрузка на диск
 - много кода в коммунальных скриптах
 - сложнее отладка
 - документация
 - растет энтропия

Сравним минусы с ws-директориями

- ~~кончатся папки~~
- ~~временные папки~~
- ~~долго удалять~~
- ~~lock при падении job'ов~~
- ~~артефакты нестабильных сборок~~
- ~~большие изменения в node_modules~~
- потребление диска
- нагрузка на диск
- много кода в коммунальных скриптах
- сложнее отладка
- документация
- растёт энтропия

Сравним минусы с ws-директориями

- ~~кончатся папки~~
- ~~временные папки~~
- ~~долго удалять~~
- ~~lock при падении job'ов~~
- ~~артефакты нестабильных сборок~~
- ~~большие изменения в node_modules~~
- ~~потребление диска~~
- ~~нагрузка на диск~~
- много кода в коммунальных скриптах
- сложнее отладка
- документация
- растёт энтропия

Сравним минусы с ws-директориями

- ~~кончатся папки~~
- ~~временные папки~~
- ~~долго удалять~~
- ~~lock при падении job'ов~~
- ~~артефакты нестабильных сборок~~
- ~~большие изменения в node_modules~~
- ~~потребление диска~~
- ~~нагрузка на диск~~
- ~~много кода в коммунальных скриптах~~
- сложнее отладка
- документация
- растёт энтропия

Сравним минусы с ws-директориями

кончатся папки

временные папки

долго удалять

lock при падении job'ов

артефакты нестабильных сборок

большие изменения в node_modules

потребление диска

нагрузка на диск

много кода в коммунальных скриптах

сложнее отладка

документация

растет энтропия

Сравним минусы с ws-директориями

кончатся папки

временные папки

долго удалять

lock при падении job'ов

артефакты нестабильных сборок

большие изменения в node_modules

потребление диска

нагрузка на диск

много кода в коммунальных скриптах

сложнее отладка

документация

растет энтропия

Сравним минусы с ws-директориями

кончатся папки

временные папки

долго удалять

lock при падении job'ов

артефакты нестабильных сборок

большие изменения в node_modules

потребление диска

нагрузка на диск

много кода в коммунальных скриптах

сложнее отладка

документация

растет энтропия

■ Все в порядке. Правда



Минусы

- незначительное изменение package.json: +20-50 сек

Минусы

- незначительное изменение package.json: +20-50 сек
- нестандартная модель работы containerd/kubelet

Минусы

- незначительное изменение package.json: +20-50 сек
- нестандартная модель работы containerd/kubelet
- не clean install

Минусы

- незначительное изменение package.json: +20-50 сек
- нестандартная модель работы containerd/kubelet
- не clean install
- не ускоряем полезную нагрузку (конкретные инструменты)

Закралась мысль

- причина ускорения в tmpfs

Закралась мысль

- причина ускорения в tmpfs
- отключил tmpfs

Закралась мысль

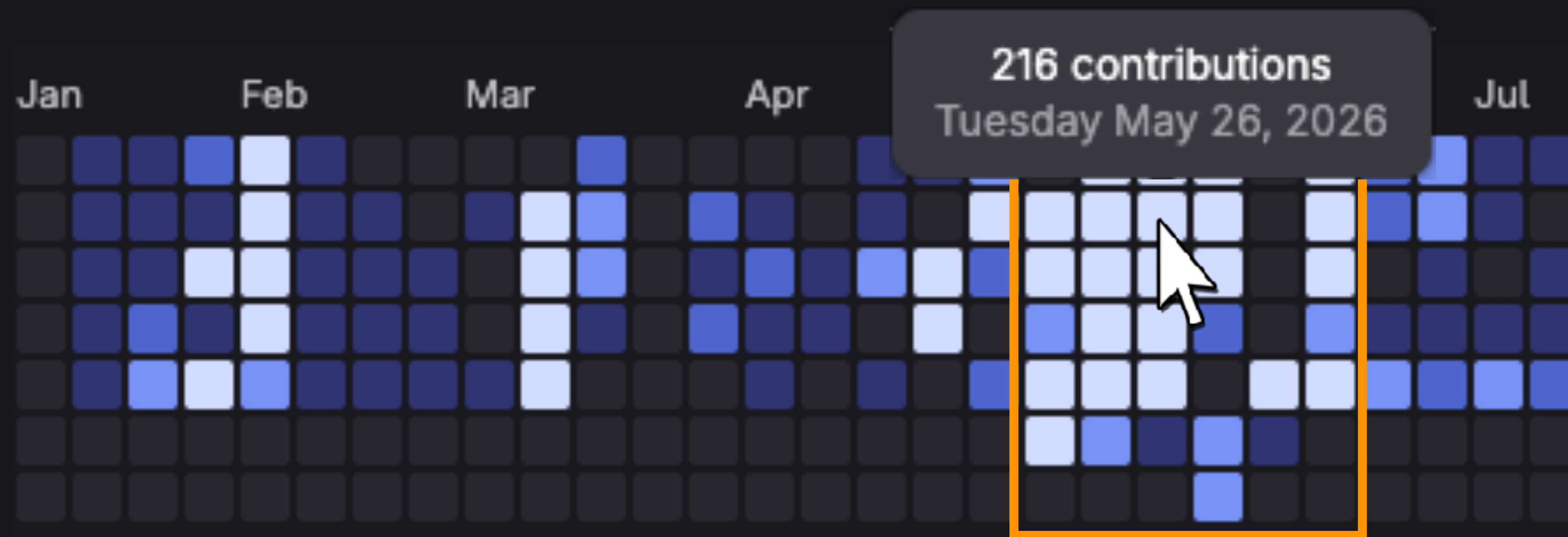
- причина ускорения в tmpfs
- отключил tmpfs
- prebuild замедлился на 10%

Закралась мысль

- причина ускорения в tmpfs
- отключил tmpfs
- prebuild замедлился на 10%
- build ускорился на 10%

**Космические скорости GitLab CI:
от 20 минут к 40 секундам за месяц**

Космические скорости GitLab CI: от 20 минут к 40 секундам **за месяц**



Космические скорости GitLab CI: от 20 минут к 40 секундам за месяц

- 1600 коммитов в три репозитория

Космические скорости GitLab CI: от 20 минут к 40 секундам за месяц

- 1600 коммитов в три репозитория
- из них 1500 "wip" - `git add . && git commit -m "wip" -n && git push`

Космические скорости GitLab CI: от 20 минут к 40 секундам за месяц

- 1600 коммитов в три репозитория
- из них 1500 "wip" - `git add . && git commit -m "wip" -n && git push`
- счетчик сборки 2-летнего проекта с 800 поднялся до 1800

Что бы я сделал иначе

- shellcheck
- CI Lint for GitLab (webstorm plugin)

Мораль

- перепроверяйте принятые решения

Мораль

- перепроверяйте принятые решения
- баланс важнее паттернов и практик

Мораль

- перепроверяйте принятые решения
- баланс важнее паттернов и практик
- можно писать хороший код в любом стеке

Мораль

- перепроверяйте принятые решения
- баланс важнее паттернов и практик
- можно писать хороший код в любом стеке
- никто не знает ваш стек лучше вас

Мораль

- перепроверяйте принятые решения
- баланс важнее паттернов и практик
- можно писать хороший код в любом стеке
- никто не знает ваш стек лучше вас
- копайте и расширяйте кругозор

Мораль

- перепроверяйте принятые решения
- баланс важнее паттернов и практик
- можно писать хороший код в любом стеке
- никто не знает ваш стек лучше вас
- копайте и расширяйте кругозор
- серебряные пули - ...

Спасибо за внимание!



Иван
Малюгин

 билайн

Основная часть
исходного кода

