

Каскадный деплой в GitOps: два подхода на FluxCD

Промоушн между окружениями, который не даёт
одному коммиту разъехаться по всем кластерам

Дмитрий Рыбалка · Mindbox · DevOps 2026

01 Открытие

git push

```
1  apiVersion: kustomize.config.k8s.io/v1beta1
2  kind: Kustomization
3
- namespace: a2p-ingress
4
5  resources:
```

Одна удалённая строка. Один коммит в master.

01 Открытие

30

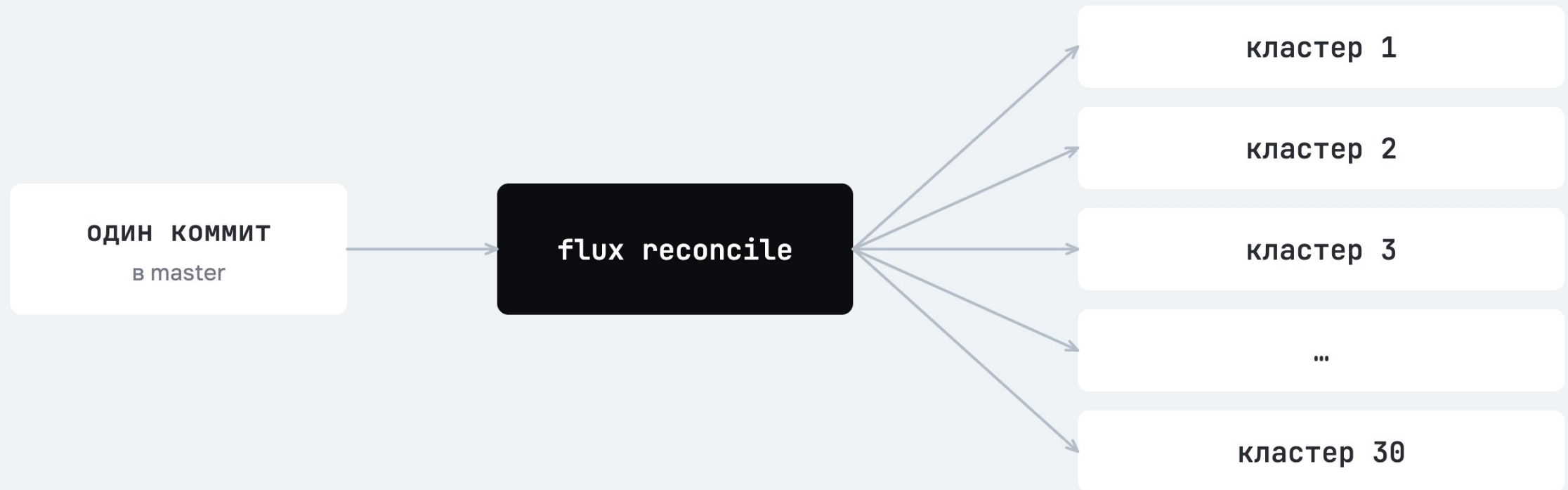
кластеров

за 5 минут. Инженер не успел дойти до лифта.

01 Открытие

Blast radius

В каждом кластере namespace удалён и создан с нуля: около 40 микросервисов, настройки Kafka и Postgres.



«Flux отработал корректно»

01 Открытие

Главная мысль

**В GitOps опасен не сам reconciliation.
Опасно его автоматическое распространение
без gate.**

01 Открытие

Кто я



Дмитрий Рыбалка

Principal SRE · Mindbox

Опыт: SberHealth · SberMarket · Lamoda



5+ лет

с FluxCD



x40

рост инфраструктуры —
встретил лицом

AOT

Эксперт

совет экспертов AOT



Амбассадор

Kubernetes

Говорю о том, что **сам** строю, внедряю и поддерживаю.

01 Открытие

План

- 01** **Модель gate**
что это и по какой оси работает

- 02** **Что уже есть во Flux**
примитивы в коробке и чего в ней нет

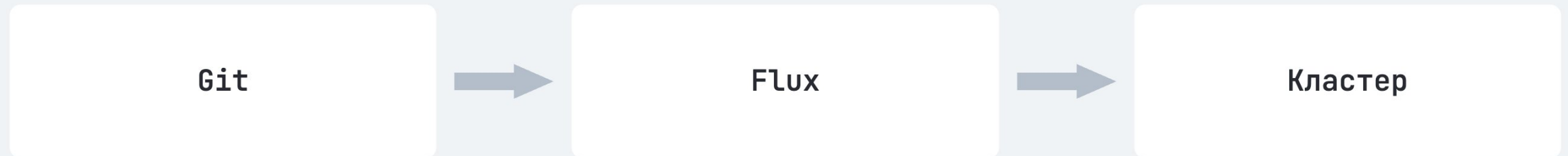
- 03** **Развилка**
где живёт state machine промоушна

- 04** **Два подхода**
промоушн в CI и промоушн в контроллере

- 05** **К чему пришли мы**
спойлер: не туда, куда собирались

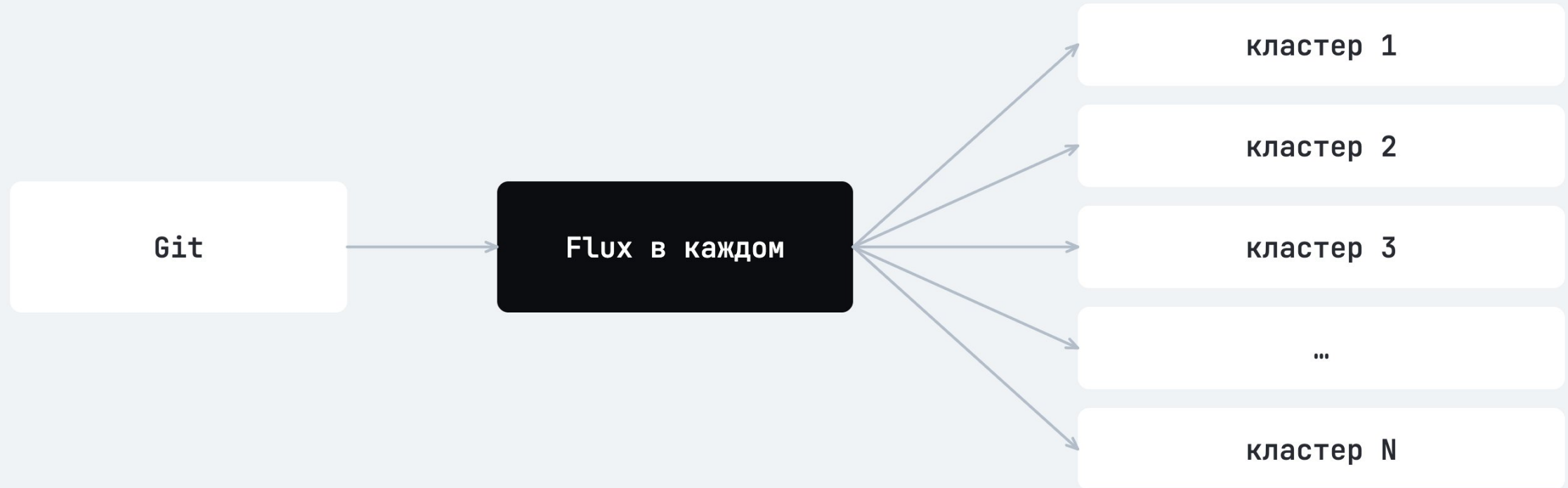
02 Где появляется blast radius

GitOps в презентациях



02 Где появляется blast radius

Вот как в жизни



02 Где появляется blast radius

Точка связности

Общий репозиторий

Все окружения описаны в одном месте

Общие базовые манифесты

Один k8s-base на всех

Общие overlay

Правка в базе прилетает во все окружения сразу

Все применяют одновременно



время → все применили в одном окне реконсиляции, без очереди и порядка

02 Где появляется blast radius

**Нет ни одного места,
где можно сказать: **СТОП****

Два разных слова

reconciliation

Приведение состояния к желаемому внутри кластера.
Базовое требование к GitOps-инструменту.

propagation

Распространение изменения между окружениями.
В принципах GitOps его нет, Flux его не делает.

OpenGitOps, принцип 4: Continuously Reconciled

03 Модель gate

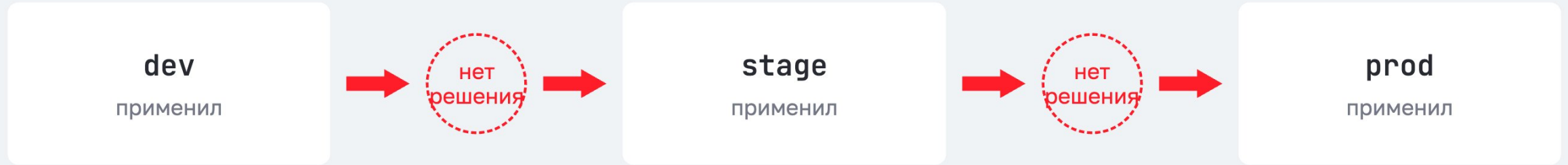
Promotion gate

Дальше — просто gate. Это главное слово доклада.

Проверка между окружениями.
Прошла — изменение едет дальше. **Не прошла — стоит.**



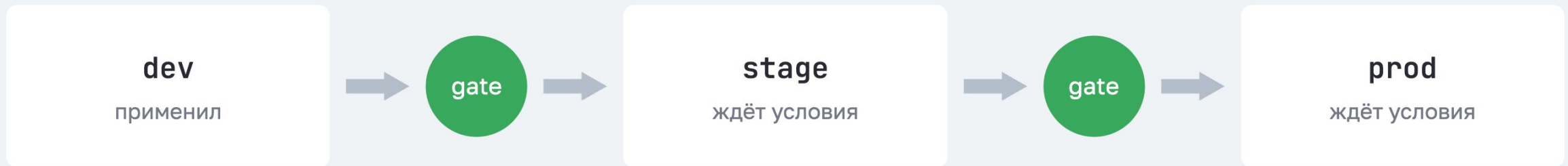
Каскад без gate



Для нашей задачи это ещё не управляемый каскад. Это **последовательный rollout**: решения между стадиями нет.

03 Модель gate

Каскад с gate



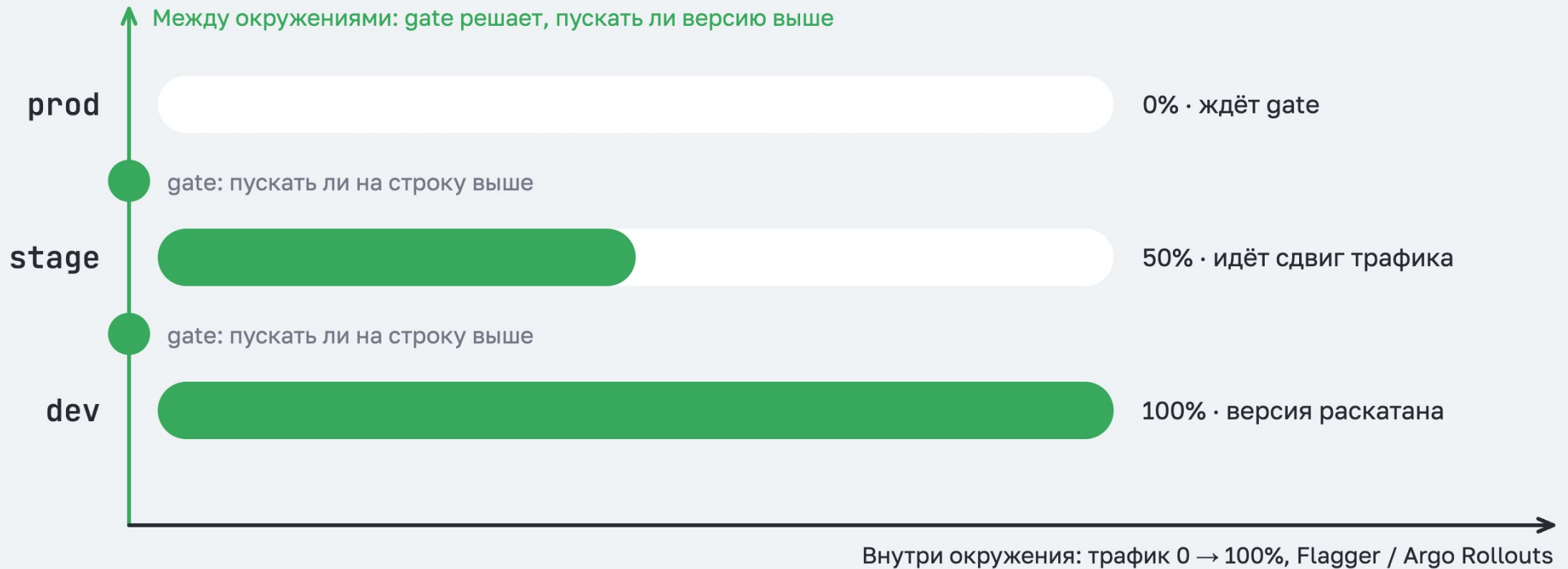
Изменение едет дальше, только когда gate открыт.

А это не то же самое, что Flagger или Argo Rollouts?

03 Модель gate

Две оси

Не альтернативы — они складываются: Flagger двигает трафик внутри строки, gate пускает на строку выше.



03 Модель gate

Ready=True, а приложение не работает



Релиз зелёный

Новое правило Ingress
для Kibana, Ready=True



nginx не жалуется

Конфиг принят, ошибок
в логах нет



Kibana работает

Поды живы



200 → 4XX

Коды ответа сменились

Заметили по алерту: резкий рост 4XX-х **в первые 2 минуты** после релиза.

03 Модель gate

Четыре уровня сигнала

Каждый следующий имеет смысл, только если предыдущий зелёный.

01 Доехало и применилось?
delivery

02 Запустилось?
workload

03 Работает ли сервис?
SLI

04 Работает ли бизнес?
ломается позже, замечают снаружи

Уровни 1 и 2: доехало и запустилось

Доехало и применилось?

- Источник Ready
- Ревизия совпала с желаемой
- Не висит suspend

Запустилось?

`wait: true`

- Реплики доступны, подов в Pending нет
 - Migration Job из релиза дождались
- Рестарты wait не видит

Уровни 1–2 отвечают только на вопрос: **доехало и запустилось**. Как сервис работает — они не говорят.

Уровень 3: работает ли сервис

Ошибки

По классам кодов, не суммарно

Задержка

По перцентилям, не среднее

Насыщение

Очередь, пул соединений, воркеры

Трафик

И он не нулевой

История с Kibana ловится здесь: Ready зелёный, а **доля 4xx выросла** сразу после релиза.

Уровень 4: работает ли бизнес

Способ её найти: вспомните последний инцидент, о котором сообщил человек, а не мониторинг.

**Метрика, которая ломается раньше технических и которую замечает
не инженер, а пользователь.**

03 Модель gate

Четыре правила

Не зависят от стека. Зависят от soak time.

Soak time — сколько версия стоит на окружении, **прежде чем ехать дальше.**

База и абсолют

1

Регрессия против baseline до релиза + абсолютная граница SLO

Длиннее медленной обратной связи

2

Не короче прогрева кэша, крона, батча, ретраев

Смотреть на соседей

3

За это время плохой релиз часто ломает сначала зависимый сервис

Fail closed

4

Сигнала за soak time нет — gate не открывается

03 Модель gate

Главная мысль

**В GitOps опасен не сам reconciliation.
Опасно его автоматическое распространение
без gate.**

**Flux даёт примитивы для gate.
Модель промоушна он не даёт**

Что вы и так знаете

suspend

```
flux suspend kustomization app
```

Ручной стоп-кран.

status.conditions[Ready]

```
lastAppliedRevision: main@sha1:9f2c...
```

Сюда смотрит gate: предыдущий этап Ready — можно ехать дальше.

dependsOn

```
dependsOn: [ infrastructure ]
```

Штатный каскад, но только в пределах одного кластера. Между кластерами его нет.

wait + healthChecks

```
wait: true · timeout: 5m
```

kstatus закрывает уровни «доехало» и «запустилось».

Но все это видно только внутри кластера!

04 Что уже есть во Flux

Commit status

Как вынести Ready из кластера наружу.

01

kustomize-controller
реконсилировал

02

событие с ревизией →
notification-controller

03

commit status в GitLab
/ GitHub

04

внешняя автоматика
читает статус через API и
делает промоушн — без
доступа в кластер

**Движок промоушна не обязан жить в кластере.
Ему достаточно уметь читать статус коммита.**

04 Что уже есть во Flux

Чего Flux не даёт

Эти четыре пробела закрывают инструменты дальше.

Порядок

Между окружениями и кластерами

Запись о промоушне

«Эта версия промоутнута туда-то»

Approve

Как объект, а не как кнопка в пайплайне

Решение по метрикам

Автоматическое, без человека

Промоушн между окружениями у вас — это пайплайн в CI?

Поднимите руку, если да.

05 Развилка

Где живёт state промоушна



подход 1

В пайплайне CI

Пайплайн знает порядок стадий и ждёт статус. Кластер только применяет и отдаёт факт.

подход 2

В контроллере

Kargo держит Freight, стадии и проверки как объекты. CI только собирает артефакт.

Как версия доезжает до кластера

Gitless: версия — это тег OCI-артефакта

CI кладёт OCI-артефакт в registry.

Промоушн — новый тег. Git не меняется.

Git-классика: версия — это коммит

Каждый промоушн — коммит.

– лишние коммиты и теги в master,
история раздувается

06 Подход 1: промоушн в CI · сторонний проект · CI + gitless, доступа в кластеры нет

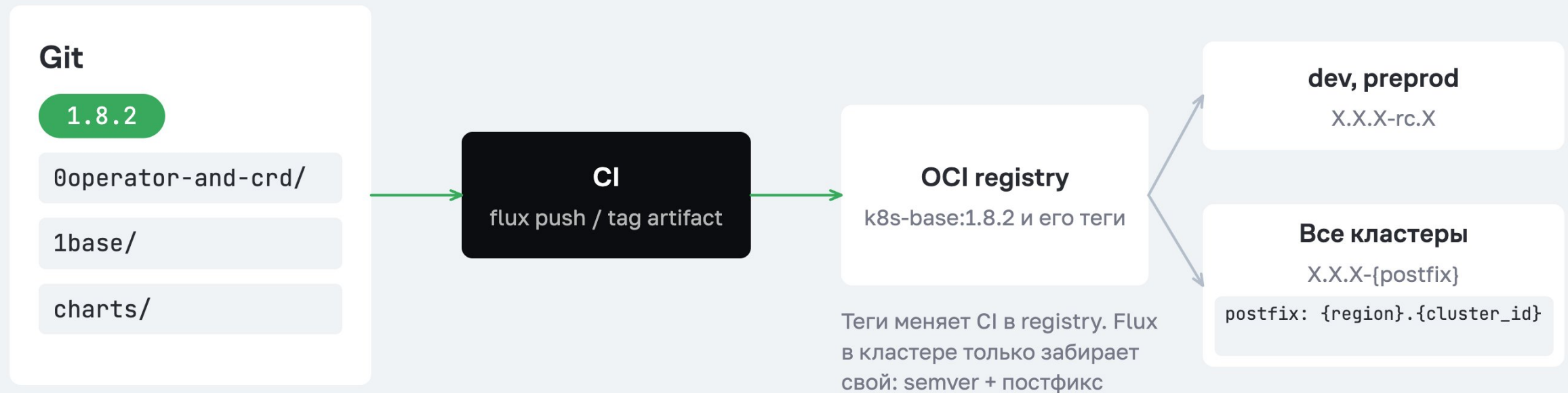
Подход 1: промоушн в CI

Порядок стадий знает пайплайн. Промоушн — **новый тег** на тот же артефакт. Кластер только применяет и отдаёт факт состояния релиза.

06 Подход 1: промоушн в CI · сторонний проект · CI + gitless, доступа в кластеры нет

Схема

Движок снаружи. Flux внутри только применяет.



06 Подход 1: промоушн в CI · сторонний проект · CI + gitless, доступа в кластеры нет

flux push / tag artifact

Публикация артефакта: X.X.X — неизменяемый тег

```
flux push artifact \  
oci://${OCI_HOST}/k8s-base:${CI_COMMIT_TAG} \  
--path=./manifests \  
--source=${CI_PROJECT_URL} \  
--revision="${CI_COMMIT_SHA}"
```

Промоушн: новый тег в OCI registry

```
flux tag artifact oci://${OCI_HOST}/k8s-base:${CI_COMMIT_TAG} \  
--tag ${CI_COMMIT_TAG}-${PREFIX} \  
--creds ${OCI_USERNAME}:${OCI_PASSWORD}
```

revision уезжает в метаданные артефакта — без неё commit status с OCI не работает.

06 Подход 1: промоушн в CI · сторонний проект · CI + gitless, доступа в кластеры нет

Что внутри

Между стадиями ничего не пересобираем: по каскаду едет один и тот же OCI-артефакт, с одним digest.

Внутри — готовые манифесты.

Изменение зафиксировано в артефакте и стало неизменяемым.

06 Подход 1: промоушн в CI · сторонний проект · CI + gitless, доступа в кластеры нет

OCIRepository: semver и маска

Источник: semver + маска кластера

```
kind: OCIRepository
spec:
  url: oci://{OCI_HOST}/k8s-base
  ref:
    semver: "≥ 1.0.0-0" # pre-release тоже
    semverFilter: <маска кластера>
---
kind: Kustomization
spec:
  sourceRef: { kind: OCIRepository, name: k8s-base }
  path: ./
```

```
# dev, preprod
"*.rc.*"
# все кластеры
"*.~{region}\.({cluster_id})?"
```

Что выберет Flux

# теги в registry	кто заберёт
1.8.2-rc.3	dev, preprod
1.8.2	# неизменяемый
1.8.2-ru.01	ru.01
1.8.1-ru.01	# предыдущий

Flux берёт **наибольшую версию** под маской своего кластера.

06 Подход 1: промоушн в CI · сторонний проект · CI + gitless, доступа в кластеры нет

Кто может выкатить

Создавать теги окружений может только **CI**. Не человек.

CI

push и создание тегов

Люди

только чтение

Flux в кластерах

только pull

06 Подход 1: промоушн в CI · сторонний проект · CI + gitless, доступа в кластеры нет

Release queue: очередь в пайплайне

N+1 ждёт N

Новый пайплайн промоушна стартует только после того, как предыдущий прошёл до конца

Тег не выпускается

Пока N не завершён, тега N+1 в registry нет — Flux нечего подхватить

Порядок гарантирует **CI**. Semver и маска гарантируют, что Flux возьмёт наибольшую из выпущенных версий.

06 Подход 1: промоушн в CI · сторонний проект · CI + gitless, доступа в кластеры нет

Где gate: пайплайн в GitLab

Перевесить тег — одна команда. Вопрос в том, кто её разрешает.



Gate стоит дважды: **перед каскадом** и после каждого кластера внутри него.

06 Подход 1: промоушн в CI · сторонний проект · CI + gitless, доступа в кластеры нет

Реестр статусов: где CI узнаёт, что кластер здоров

The screenshot shows a dark-themed monitoring dashboard. At the top left, there's a header 'ЛЕНТА СОБЫТИЙ' (Event Stream) and a main title 'События' (Events). Below the title is a subtitle: 'Релизы, плановые работы и аварии на одной временной оси. Период и площадка — общие для всех вкладок.' (Releases, planned work and incidents on the same time axis. Period and platform — common for all tabs). To the right of the subtitle are filters: a radio button for 'Авто' (Auto) and a button for 'выкл' (off), and a time range selector with options '24ч', '7д' (selected), '30д', and 'Custom'. Below these are tabs for 'Все 3' (All 3), 'Работы 0' (Jobs 0), 'Релизы 3' (Releases 3), and 'Аварии и инциденты 0' (Incidents and incidents 0). Further down is a 'ПЛОЩАДКА' (Platform) section with dropdowns for 'Страна: все' (Country: all), 'ЦОД: все' (Data center: all), and 'ru-dev' (selected), followed by a 'Сбросить' (Reset) button. The main content area displays a list of events. Two events are visible, both with a green status dot and the text 'SUCCEEDED'. Each event includes a timestamp, a container icon, the name 'k8s-base', and a version number (1.33.8 and 1.33.7). To the right of each event are two buttons: 'country: ru' and 'cluster: ru-dev'.

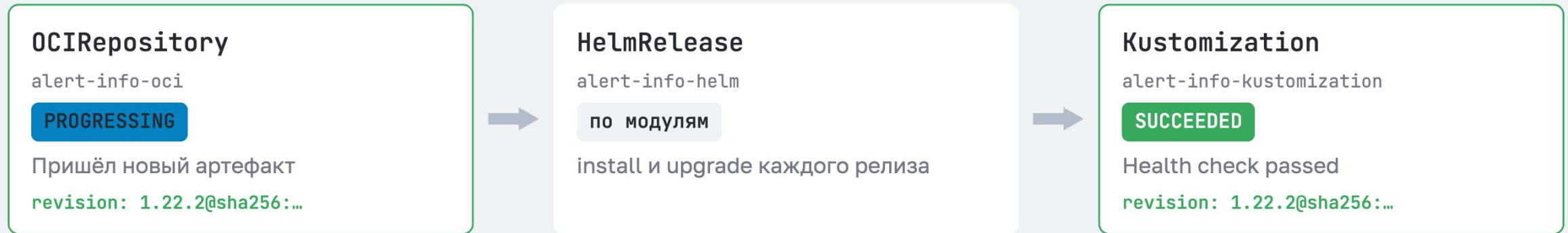
Статус	Время	Объект	Версия	Страна	Кластер	Результат
●	24 янв 15:51	k8s-base	1.33.8	ru	ru-dev	SUCCEEDED
●	24 янв 11:16	k8s-base	1.33.7	ru	ru-dev	SUCCEEDED

Доступа в кластеры нет, поэтому кластеры сами отправляют статус сюда, а gate читает его отсюда.

06 Подход 1: промоушн в CI · сторонний проект · CI + gitless, доступа в кластеры нет

Как Flux заполняет реестр статусов

По одному Alert на этап. События одной версии реестр склеивает **по revision**.



```

kind: Alert
metadata: { name: alert-info-kustomization }
spec:
  providerRef: { name: grafana-oncall } # type: generic
  eventSources:
  - kind: Kustomization
    name: platform-release
  inclusionList: ["Health check passed.*"]
  eventMetadata: # ключ «версия × кластер»
    cluster_name: ${cluster_name}
    env: ${cluster_env}
  
```

FAILED Ошибка
alert-error-kustomization

Габли

Namespace в eventSources — поимённо, иначе новый модуль **молча остаётся без событий**.

06 Подход 1: промоушн в CI · сторонний проект · CI + gitless, доступа в кластеры нет

Полный контур



06 Подход 1: промоушн в CI · сторонний проект · CI + gitless, доступа в кластеры нет

SLO-снимок: что происходит после релиза

01

Релиз применён: Flux закончил reconcile

02

Окно: hook ждёт, метрики новой версии копятя

03

Снимок: применяем оценку, Keptn снимает метрики

04

Проверка: CEL и healthChecks → Ready → статус «применён» уходит наружу

Весь путь — **до 20–25 минут**. Это верхняя граница на все проверки, а не фиксированная пауза.

Для Helm-чарта: post-upgrade hook создаёт манифест снимка, Kyverno удаляет его по TTL — [label cleanup.kyverno.io/ttl](https://labelcleanup.kyverno.io/ttl).

06 Подход 1: промоушн в CI · сторонний проект · CI + gitless, доступа в кластеры нет

Пример кодом: метрики и порог SLO

Keptn: чем мерить

```
apiVersion: metrics.keptn.sh/v1
kind: KeptnMetricsProvider
metadata: { name: prometheus, namespace: elastic }
spec: { type: prometheus, targetServer: http://prometheus.monitoring:9090 }
---
kind: KeptnMetric
metadata: { name: error-rate, namespace: elastic }
spec:
  provider: { name: prometheus }
  query: nginx_ingress_controller_requests:burnrate15m{slo="kibana"}
  fetchIntervalSeconds: 30
---
kind: KeptnMetric
metadata: { name: p99-latency, namespace: elastic }
spec:
  provider: { name: prometheus }
  query: histogram_quantile(0.99, sum by (le) (rate(
    nginx_ingress_controller_request_duration_seconds_bucket
    {ingress="kibana"}[5m])))
```

Keptn: что считать нормой и сама оценка

```
apiVersion: lifecycle.keptn.sh/v1
kind: KeptnEvaluationDefinition
metadata:
  name: kibana-slo
  namespace: elastic
spec:
  objectives:
    - keptnMetricRef: { name: error-rate, namespace: elastic }
      evaluationTarget: "<0.01" # меньше 1% ошибок
    - keptnMetricRef: { name: p99-latency, namespace: elastic }
      evaluationTarget: "<0.5" # p99 меньше 500 мс
---
kind: KeptnEvaluation # лежит в /slo/check.yaml
metadata: { name: kibana-check, namespace: elastic }
spec:
  evaluationDefinition: kibana-slo
  retries: 3
  retryInterval: 30s
```

06 Подход 1: промоушн в CI · сторонний проект · CI + gitless, доступа в кластеры нет

Пример кодом: оценка и Flux

Helm hook: ждём окно, потом создаём оценку

```
apiVersion: batch/v1
kind: Job
metadata:
  name: kibana-slo-gate
  annotations: { helm.sh/hook: post-upgrade }
  labels: { cleanup.kyverno.io/ttl: 2h } # Kyverno удалит
spec:
  template:
    spec:
      serviceAccountName: kibana-slo-gate # нужен RBAC, см. ниже
      restartPolicy: Never
      containers:
        - name: gate
          image: <ваш образ с kubectl>
          command: [sh, -c]
          args: ["sleep 15m && kubectl apply -f /slo/check.yaml"]
```

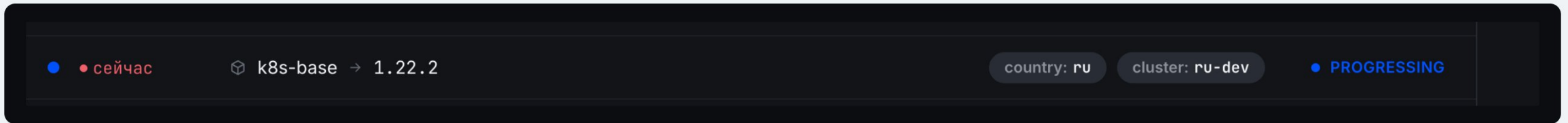
Нужен свой ServiceAccount с Role на create keptnevaluations — шаблонами чарта, не хуком.

Flux ждёт оценку

```
apiVersion: kustomize.toolkit.fluxcd.io/v1
kind: Kustomization
metadata: { name: elastic, namespace: flux-system }
spec:
  path: ./infra/elastic
  wait: true
  timeout: 25m # окно 15m + оценка
  healthChecks:
    - apiVersion: lifecycle.keptn.sh/v1
      kind: KeptnEvaluation
      name: kibana-check
      namespace: elastic
  # у HelmRelease тоже spec.timeout: 25m
```

06 Подход 1: промоушн в CI · сторонний проект · CI + gitless, доступа в кластеры нет

Реестр: PROGRESSING

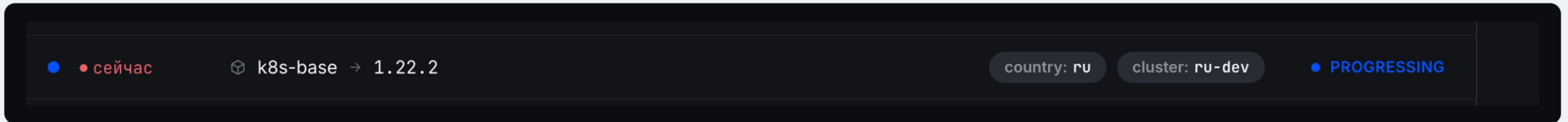


Статус живёт у пары «версия × кластер»:
k8s-base **1.22.2** в ru-dev — пока PROGRESSING.

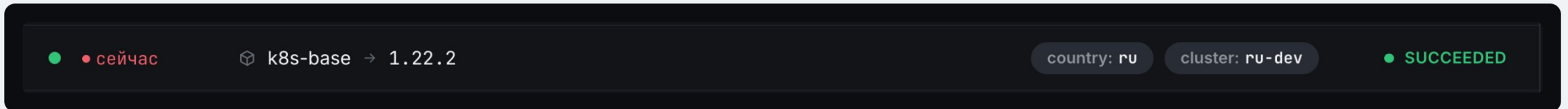
06 Подход 1: промоушн в CI · сторонний проект · CI + gitless, доступа в кластеры нет

Реестр: SUCCEEDED

было



стало



Поменялись два слова — это и есть **разрешение ехать дальше.**

06 Подход 1: промоушн в CI · сторонний проект · CI + gitless, доступа в кластеры нет

Job ожидания

```
32 07:59:45 INFO - ru-dev: Progressing, ждём
33 07:59:45 INFO - ru-preprod: события версии 1.22.2 ещё не приходили
34 08:00:05 INFO - ru-dev: Progressing, ждём
35 08:00:05 INFO - ru-preprod: события версии 1.22.2 ещё не приходили
```

Опрос раз в 20 секунд: 07:59:45 → 08:00:05.

Джоба не завершится, пока не увидит нужный статус **по каждому кластеру**.

06 Подход 1: промоушн в CI · сторонний проект · CI + gitless, доступа в кластеры нет

Два разных «нет»

Progressing

Сигнал есть, применение идёт, ждём

События ещё не приходили

Сигнала нет вообще. Версия до кластера не доехала, или сломалось что-то по дороге

Gate ждёт в обоих случаях. Но чинить их надо по-разному — это **fail closed** из четырёх правил в живом виде.

06 Подход 1: промоушн в CI · сторонний проект · CI + gitless, доступа в кластеры нет

Две метки времени

```
38 08:00:45 INFO - ru-dev: k8s-base 1.22.2 применён успешно (2026-08-31T07:58:38.000Z)
39 08:00:45 INFO - ru-preprod: Progressing, ждём
```

08:00:45 — gate это увидел

Когда джоба ожидания прочитала статус

07:58:38 — проверка завершилась

Метка из события: Flux закончил reconcile

Между ними около двух минут, и это не задержка деплоя. Это **окно снятия SLO-снимка**.

Потолок на все проверки — 20–25 минут. Здесь уложились в две.

06 Подход 1: промоушн в CI · сторонний проект · CI + gitless, доступа в кластеры нет

Итог подхода 1

Контроллера промоушна в кластере **нет ни одного**.

Артефакт

OCI, X.X.X неизменяем

Flux

Только применяет

Компонент здоровья

SLO-снимок за окно

Job ожидания

Ждёт статус — это и
есть gate

06 Подход 1: промоушн в CI

Главная мысль

**В GitOps опасен не сам reconciliation.
Опасно его автоматическое распространение
без gate.**

07 Подход 2: промоушн в контроллере · Mindbox · Kargo + Git, доступ в кластеры есть

Подход 2: промоушн в контроллере

Kargo хранит состояние промоушна и verification и **решает**, может ли Freight перейти на следующую Stage. CI только собирает.

07 Подход 2: промоушн в контроллере · Mindbox · Kargo + Git, доступ в кластеры есть

Kargo не привязан к Argo CD

Имя — от команды авторов Argo.

Argo CD для Kargo — только один из способов доставить релиз и узнать его статус.



Что Kargo берёт у Argo CD

- `argocd-update` — синхронизировать Application
- `argocd-wait` — дождаться Healthy и Synced
- health check Application — статус релиза



Как это работает с Flux

- `git-commit`, `git-push` — Flux подхватит коммит
- `oci-push` — перевесить тег, как в подходе 1
- `http` — дернуть Receiver, не ждать интервал
- `verification` — статус релиза через AnalysisTemplate

Под Argo CD заточены **только отдельные шаги**: `argocd-update`, `argocd-wait`. Остальные с Argo CD не связаны, поэтому Kargo работает и поверх Flux.

07 Подход 2: промоушн в контроллере · Mindbox · Kargo + Git, доступ в кластеры есть

Что добавляет Kargo

Доставку Kargo не заменяет — он оркестрирует то, что уже есть.

Имя

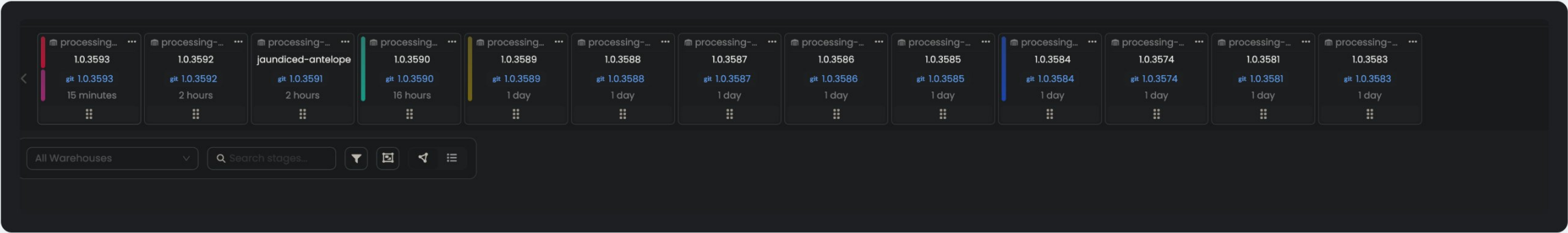
У промоушна появляется сущность

Состояние

Видно, где какая версия сейчас

История

Промоушн остаётся записью



Лента Freight: каждая версия — отдельный Freight, цветная метка — стадия, где он сейчас. 1.0.3593 — в staging и beta, 1.0.3590 — в sigma, 1.0.3589 — в omega, 1.0.3584 — в stable.

07 Подход 2: промоушн в контроллере · Mindbox · Kargo + Git, доступ в кластеры есть

Сущности Kargo

Project · ProjectConfig границы: неймспейс, права, политики продвижения



Каждый переезд Freight между Stage записан как **Promotion**: кто, что, куда, когда и с каким результатом.

07 Подход 2: промоушн в контроллере · Mindbox · Kargo + Git, доступ в кластеры есть

Как это выглядит у нас



processing-orders: Warehouse → omega и staging → beta → sigma → stable. У каждой стадии своя версия и время последнего промоушна.

07 Подход 2: промоушн в контроллере · Mindbox · Kargo + Git, доступ в кластеры есть

Промоушн и проверка — это записи

Promotions · стадия omega

Promotions

Verifications

Live Manifest

Freight History

Settings

Date	Name	Created By	Freight
✔ Sep 17th 2026 12:20:12	omega.01m2qe33bte7tzvdd5er3mzk3.b7332dc	system	1.0.3589 (b7332dc)
✔ Sep 16th 2026 17:40:18	omega.01m2ne0gdxhjk7j729tneb4cc.9ea531a	system	1.0.3581 (9ea531a)

Verifications · та же стадия

Promotions

Verifications

Live Manifest

Freight History

Settings

Date	Duration	AnalysisRun	Freight
Sep 17th 2026 12:20:13	a few seconds		
Sep 16th 2026 17:40:21	a few seconds		

07 Подход 2: промоушн в контроллере · Mindbox · Kargo + Git, доступ в кластеры есть

Шаги — плагины. Вариант А: git-commit → PR

Kargo не диктует, как двигать версию: шаги промоушна собираются как конструктор.

```
steps:
- uses: git-clone          # клонировать GitOps-репозиторий
- uses: yaml-update        # поднять версию в values
- uses: git-commit
- uses: git-push           # в отдельную ветку
- uses: git-open-pr
- uses: git-wait-for-pr    # ждём мержа
```

Kargo становится ещё одним автором коммитов в вашем репозитории.

07 Подход 2: промоушн в контроллере · Mindbox · Kargo + Git, доступ в кластеры есть

Три варианта Kargo: git, GitLab или gitless

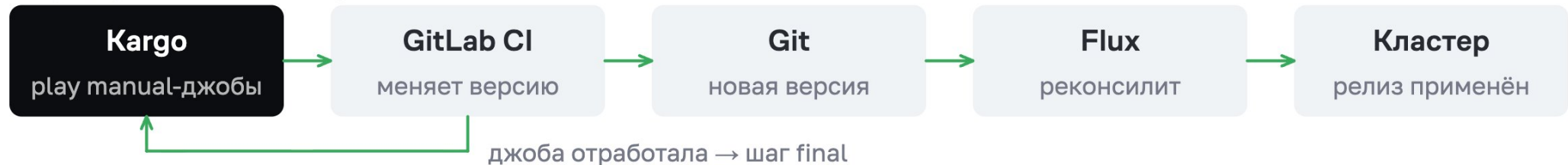
A

Kargo сам пишет в git



B

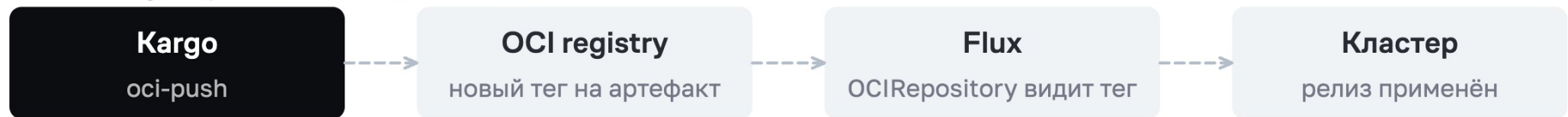
Kargo дёргает GitLab — так у нас сейчас



Потом verification: AnalysisTemplate по метрикам — и только тогда Freight едет дальше.

C

Gitless: Kargo перевешивает тег, Git не меняется



07 Подход 2: промоушн в контроллере · Mindbox · Kargo + Git, доступ в кластеры есть

Вариант В: реальный gitlab-play-job

```
kind: ClusterPromotionTask
metadata: { name: gitlab-play-job }
spec:
  vars: [apiURL, projectId, jobName, targetTag]
  steps:
    - uses: set-freight-alias # Freight = тег релиза
    - uses: http # live: что сейчас в environment
    - uses: http # project: ссылки для UI
    - uses: http # find-pipeline, find-job по тегу
    - uses: set-metadata # ссылки на коммиты и джобу
    - uses: http # play или retry manual-джобы
    - uses: http # wait: опрос раз в 30 с, до 2 ч
    - uses: http # final: deployment = success
    - uses: set-metadata # кто, какая джоба, тег
```

0 шагов Argo CD

Всего 12 шагов: 9 × http, 2 × set-metadata, 1 × set-freight-alias

- идемпотентно: если в environment уже этот тег, джобу не трогаем
- GitLab deployment = джоба отработала; здоровье проверяет verification
- токен — sharedSecret, в манифесте его нет

07 Подход 2: промоушн в контроллере · Mindbox · Kargo + Git, доступ в кластеры есть

С чем ещё работает проверка

Провайдеры AnalysisTemplate из документации Kargo. VictoriaMetrics — через тот же провайдер prometheus.

Мониторинг



Prometheus

Prometheus

наш пример — дальше



VictoriaMetrics

VictoriaMetrics

через провайдер prometheus



DATADOG

Datadog



Amazon CloudWatch



New Relic



influxdb™

InfluxDB



graphite

Graphite

Skywalking

Apache SkyWalking

Свои системы



Job

Kubernetes Job: любой контейнер — интеграционные тесты, нагрузка



Web

HTTP-запрос во внутренний сервис, разбор JSON-ответа

07 Подход 2: промоушн в контроллере · Mindbox · Kargo + Git, доступ в кластеры есть

AnalysisTemplate: проверка через Prometheus

```
kind: AnalysisTemplate  # CRD из Argo Rollouts
spec:
  args: [{ name: app }]
  metrics:
    - name: error-rate
      initialDelay: 2m      # дать метрикам наполниться
      interval: 1m
      count: 10             # окно ≈ 10 минут
      failureLimit: 1
      successCondition: result[0] < 0.01
      provider:
        prometheus:
          address: http://prometheus.monitoring:9090
          query: |
            sum(rate(http_requests_total{app="{{args.app}}",code=~"5.."}[5m]))
            / sum(rate(http_requests_total{app="{{args.app}}"}[5m]))
```

```
kind: Stage
spec:
  verification:
    analysisTemplates:
      - name: error-rate
    args:
      - name: app
        value: processing-orders
```

→ $\text{count} \times \text{interval} + \text{initialDelay}$ — окно проверки, здесь около 12 минут

→ failureLimit — сколько промахов терпим

→ successCondition — выражение по результату запроса

07 Подход 2: промоушн в контроллере · Mindbox · Kargo + Git, доступ в кластеры есть

Метрик нет: что будет с релизом

Один случай — **правильно**, три — **неверно**.

Prometheus лёг

СТОИТ

Правильно. Но сам не откатится — нужен алерт

Нет ни одной 5xx

СТОИТ

Запрос пустой. Лечится:
or vector(0)

Нет трафика: 0 / 0

откат

Лечится вторым условием:
решает человек

«Нет метрик — значит, ОК»

пропуск

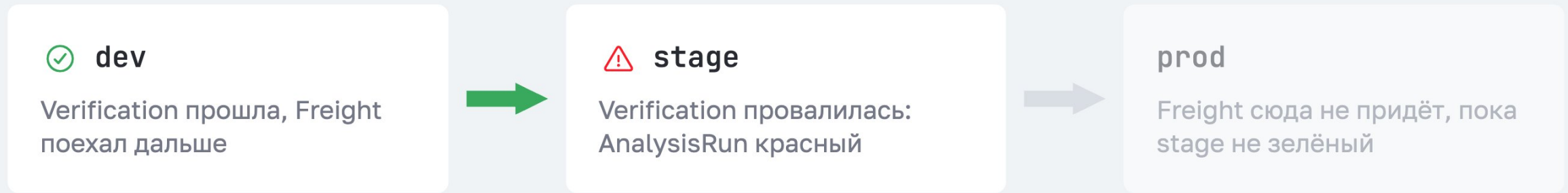
Самое опасное: gate молча пропускает релиз

```
successCondition: result[0] < 0.01
failureCondition: result[0] ≥ 0.01 # NaN → Inconclusive
query: |
  (sum(rate(..{code=~"5.."}[5m])) or vector(0))
  / sum(rate(..[5m]))
```

Нет метрик — релиз стоит и
зовёт человека.

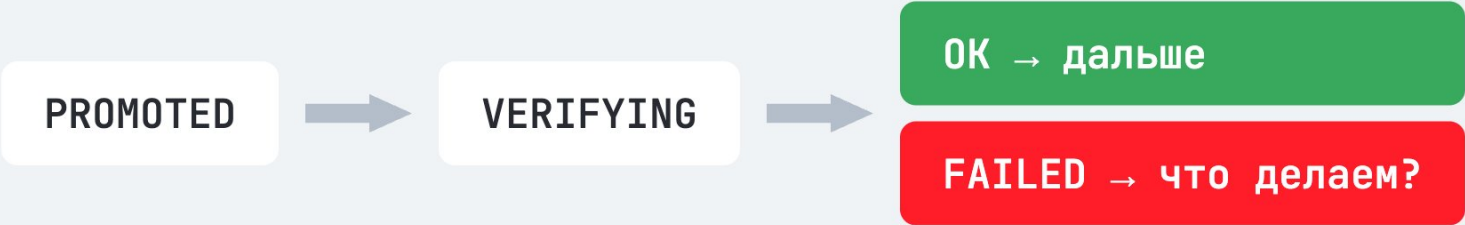
07 Подход 2: промоушн в контроллере · Mindbox · Kargo + Git, доступ в кластеры есть

Verification блокирует downstream



Провалилась проверка — каскад встал сам, без чьего-то решения.

Релиз не прошёл проверку



	Подход 1 · CI + OCI (GitLess)	Подход 2 · Kargo
STOP	Job ожидания не видит SUCCEEDED — следующий тег не выпускается	Verification красная — Freight не едет на следующую стадию
RETRY	Перезапустить job ожидания: статус пересчитается	Reverify: проверка запускается заново
ROLLBACK	CI-джоба удаляет тег, Flux сам встаёт на предыдущий тег под маской	autoRollback: Kargo сам промоутит последний проверенный Freight
SUPERSEDE	N+1 ждёт в очереди, пока пайплайн N не дойдёт до конца	Новые промоушены ждут, пока идёт verification. Ручной approve пропускает проверку

08 Сравнение

Сравним подходы

Критерий	Подход 1 · промоушн в CI	Подход 2 · промоушн в Kargo
Где живёт state machine	Пайплайн GitLab CI	Контроллер: Stage, Freight, Promotion
Release unit	OCI-артефакт, тег X.X.X	Freight: набор артефактов
Source of truth	OCI registry: какие теги есть под маской	Kargo — что куда промоутнуто, Git — желаемое состояние
Способ доставки	OCIRepository: semver + маска	Любой: git-commit, GitLab-джоба, oci-push
Audit trail	История тегов и пайплайнов	Объекты Promotion и Verification
Основной риск	State machine пишем сами: очередь, retry, rollback, потерянные события, аудит	Второй control plane: свои CRD, кредиты, эксплуатация

09 А к чему пришли мы

Цена второго контура

Что приходится поддерживать, чтобы промоушн через Kargo работал.
Не настроить один раз, а обновлять, чинить и объяснять новым людям.

Сущности и CRD

- Project, ProjectConfig
- Warehouse, Stage, Freight
- AnalysisTemplate
- CRD Argo Rollouts

Компоненты · в нашей реализации

- Контроллер в каждом кластере
- sync-cronjob с RBAC
- Отдельный UI для approve

Доступы

- GitLab-креды
- ServiceAccount для Grafana через Crossplane

**Для инфраструктуры Kargo не понадобился.
Для продуктов — внедрили.**

09 А к чему пришли мы

Где мы сейчас

Инфраструктура

GitLab CI, очередь needs.
Автопроверок нет

staging



beta



sigma



stable

Продукты

Kargo, processing-orders

staging



beta



sigma



stable

Стадии одни и те же. Решает «ехать дальше» в инфраструктуре человек, в продуктах — verification.

10 Финал

Три строки

GitOps

даёт скорость

Promotion

даёт управляемость

Observability

даёт основание для решения

Два вопроса — два решения

Сложность promotion flow определяет
где держать state machine

Линейный flow, одна команда

State machine живёт в CI. Так у нас в инфраструктуре

Ветвления, разные owners, автопроверки, policies

Контроллер окупается. Так у продуктов в Kargo

Сеть и trust boundary определяют
направление feedback

Можем читать централизованно

Pull: gate сам читает метрики и статус

Не можем

Push: кластер сам публикует статус наружу

Главная мысль

**В GitOps опасен не сам reconciliation.
Опасно его автоматическое распространение
без gate.**

Первый шаг

Выделите один сервис и сделайте каскад **dev → stage**. Без production.

Чек-лист

Сигналы, правила soak time и грабли

Контакты

Дмитрий Рыбалка · Mindbox

Обсудим ↓ Фидбек →

- 01 Gate без контроллера: реестр статусов и SLO-снимок
- 02 Kargo: когда второй контур окупается
- 03 Сигналы для gate: четыре уровня и soak time
- 04 Изолированные кластеры: commit status или свой реестр

Бэкап

Слайды под темы обсуждения. Показываю, только если тему выбрал зал.

Чек-лист gate

Сигналы

- ☐ Доехало: ревизия совпала с желаемой, suspend: false
- ☐ Запустилось: wait: true, рестарты смотрим метриками
- ☐ Сервис: ошибки по классам кодов, p99, насыщение, трафик не нулевой
- ☐ Бизнес: метрика из последнего инцидента, о котором сообщил человек

Правила soak time

- ☐ Сравниваем с базой: метрики до релиза
- ☐ Окно длиннее самой медленной обратной связи: кэш, кроны, батчи
- ☐ Смотрим на зависимые сервисы
- ☐ Fail closed: сигнала за soak time нет — gate не открывается

Грабли

- ☐ Метрики снимаем после окна, а не сразу после релиза
- ☐ timeout у Kustomization и HelmRelease больше окна
- ☐ Пустой вектор и NaN: or vector(0) и failureCondition
- ☐ CRD Argo Rollouts стоят до старта Kargo
- ☐ Ручной approve в обход проверки — только по правам

Сигналы для gate: четыре уровня и soak time

01 Доехало

Ревизия совпала,
реконсильция свежая,
нет suspend

02 Запустилось

Реплики, рестарты,
Pending, миграции

03 Работает сервис

Ошибки по классам,
перцентили,
насыщение, трафик
не ноль

04 Работает бизнес

Метрика, которую
первым замечает
пользователь

Где задаётся окно	Как
Flux + Helm	post-upgrade hook ждёт окно, создаёт KeptnEvaluation; Flux ждёт её в healthChecks
Kargo verification	AnalysisTemplate: initialDelay + count × interval
Kargo между стадиями	requiredSoakTime: Freight должен простоять на стадии N минут
Сторонний проект	до 20–25 минут на все проверки после релиза — потолок

Изолированные кластеры: commit status или свой реестр

	Commit status из Flux	Свой реестр статусов
Что нужно	Provider и Alert в notification-controller	Свой компонент и хранилище
Ограничения	Только Kustomization; с OCI — начиная с Flux 2.5	Нет, модель данных своя
Доступы	Токен GitLab в каждом кластере	Кластер только пушит статус наружу
Гранулярность	Привязан к коммиту	Версия × кластер, живёт дольше коммита

Один GitLab, свои кластеры, промоушн по коммитам — берите **штатный путь**.
Свой реестр — когда не выполняется хотя бы одно из этого.

Hotfix в обход gate

Подход 1 · GitLab CI

Лейбл hotfix на пайплайне: ожидание gate пропускается, CI сразу вешает теги на все кластеры.

Подход 2 · Kargo

Manual approve Freight для нужной стадии: verification и soak time на промежуточных стадиях пропускаются. Дальше обычный promote.

```
kargo approve --project <project> \  
  --freight <freight-id> --stage stable  
kargo promote --project <project> \  
  --freight-alias <alias> --stage stable
```

Обход — осознанное действие: он требует прав и **остаётся в истории**.

11 Бэкап · Flux подробно

Два источника

Это не два разных GitOps. Это два способа привязать окружение к версии.

GitRepository

ref: branch, tag, semver, commit

OCIRepository

ref: tag, semver, digest

Дайджест даёт то, что в Git дёшево не получить:
одни и те же байты в staging и в проде.

Что такое промоушн технически

merge в ветку

Ссылка окружения переезжает на новый коммит

перевесить тег

Ссылка окружения переезжает на новый артефакт

Сама операция тривиальна.

Вся сложность в том, кто имеет право её сделать и по какому сигналу.

11 Бэкап · Flux подробно

HelmRelease

remediation

Автооткат релиза при неудаче

test

Helm-тесты после установки

Всё перечисленное отвечает на вопрос «установилось ли».
На вопрос **«работает ли оно хорошо»** не отвечает ничего.

11 Бэкап · Flux подробно

Provider, Alert, Receiver

Через Receiver с кластером разговаривает CI.

Наружу

Alert → Provider: результат реконсиляции уезжает во внешнюю систему

Внутри

Receiver: «подтянись сейчас», не дожидаясь интервала

Как это выглядит на практике

```
kind: Provider
spec:
  type: gitlab
  address: https://gitlab.example.com/infra/fluxcd
  secretRef: { name: gitlab-token }
---
kind: Alert
spec:
  providerRef: { name: gitlab }
  eventSources:
    - kind: Kustomization
      name: '*'
```

Движение только наружу: кластер пушит статус в GitLab, снаружи в кластер никто не ходит.