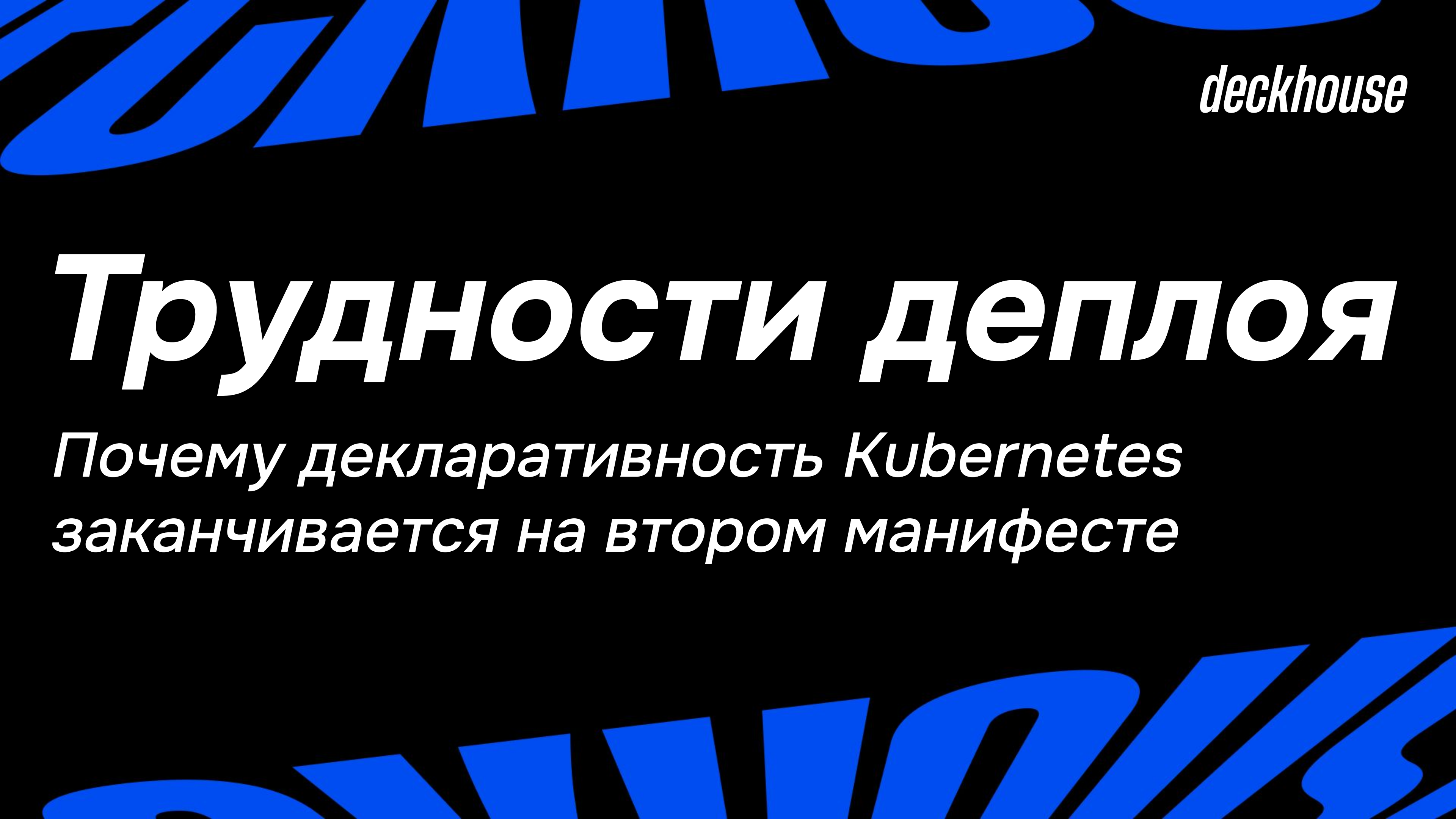




deckhouse

Трудности деплоя

*Почему декларативность Kubernetes
заканчивается на втором манифесте*



Илья Леси́ков

Разработчик Nelm, werf



**Флант—
разработчик
линейки**

**продуктов
Deckhouse**

17+

лет в Open Source:
Kubernetes, Linux,
DevOps

№ 1

контрибьютор
в проекты CNCF
из России

500+

сотрудников,
более половины —
инженеры

>260

компаний-
пользователей

С 2017

в production — более 1 300
кластеров

В топе

вендоров ИТ-решений
для банков*
и промышленности**



Реестр
российского ПО



Лицензии ФСТЭК и ФСБ России,
сертификат ФСТЭК на Deckhouse Platform

* Рейтинг [«Крупнейшие ИТ-вендоры в банках»](#), TAdviser, 2024

** Рейтинг [«Крупнейшие ИТ-вендоры в промышленности»](#), TAdviser, 2024

Деплой с kubectl

Как деплоится Kubeflow (CNCF):

```
while ! kustomize build example \  
  | kubectl apply --server-side --force-conflicts -f -; do  
  echo "Retrying to apply resources"  
  sleep 20  
done
```

Зачем выкат в цикле?

kubect!

- Не ждёт готовности ресурсов
- То есть не обеспечивает порядок выката
- Не может катить CRD и CR сразу
- *Может докатить удачно после нескольких перезапусков*

Выкат в цикле — решение?

Нет, ведь:

- Войдёт в бесконечный цикл при невалидном манифесте
- Не удалит ресурс со старым именем при переименовании
- Не удалит ресурс при удалении манифеста из рендера

Есть ли что лучше?

Сравним инструменты для выката:

- Helm (v4 с SSA)
- Argo CD
- Flux
- Nelm
- Nelm Operator

Nelm и Nelm Operator

- Nelm – форк и альтернатива Helm
- Nelm – движок развёртывания werf и Deckhouse
- werf – сборка и деплой десятков тысяч проектов
- Nelm Operator – GitOps-оператор на основе Nelm

Мой опыт

- Последние 5 лет разрабатываю werf, Nelm, Nelm Operator
- Специализируюсь на выкате в Kubernetes
- В прошлом Release Engineer/DevOps со специализацией в Kubernetes

Что рассмотрим

По несколько проблем
в категориях:

- Порядок выката
- Отслеживание ресурсов
- Жизненный цикл ресурсов

Порядок выката

Порядок выката в `kubectl apply`

- Порядок манифестов в файле определяет порядок выката
- Других возможностей нет

Когда kubectl apply не хватает

Распространённые случаи:

- Выкатить CRD и CR вместе
- Выкатить Deployment только после *выполнения* Job
- Выкатить релиз после двух других релизов

Задание порядка выката

Основные способы:

- Сортировка по Kind
- Выкат стадиями
- Выкат графом

Выкат CRD: ***Helm***

Что не так с деплоем CRD
из директории [crds/](#)

- CRD в кластере не обновятся
- При повторных релизах добавленные в директорию CRD не выкатятся

Выкат CRD: ***Helm***

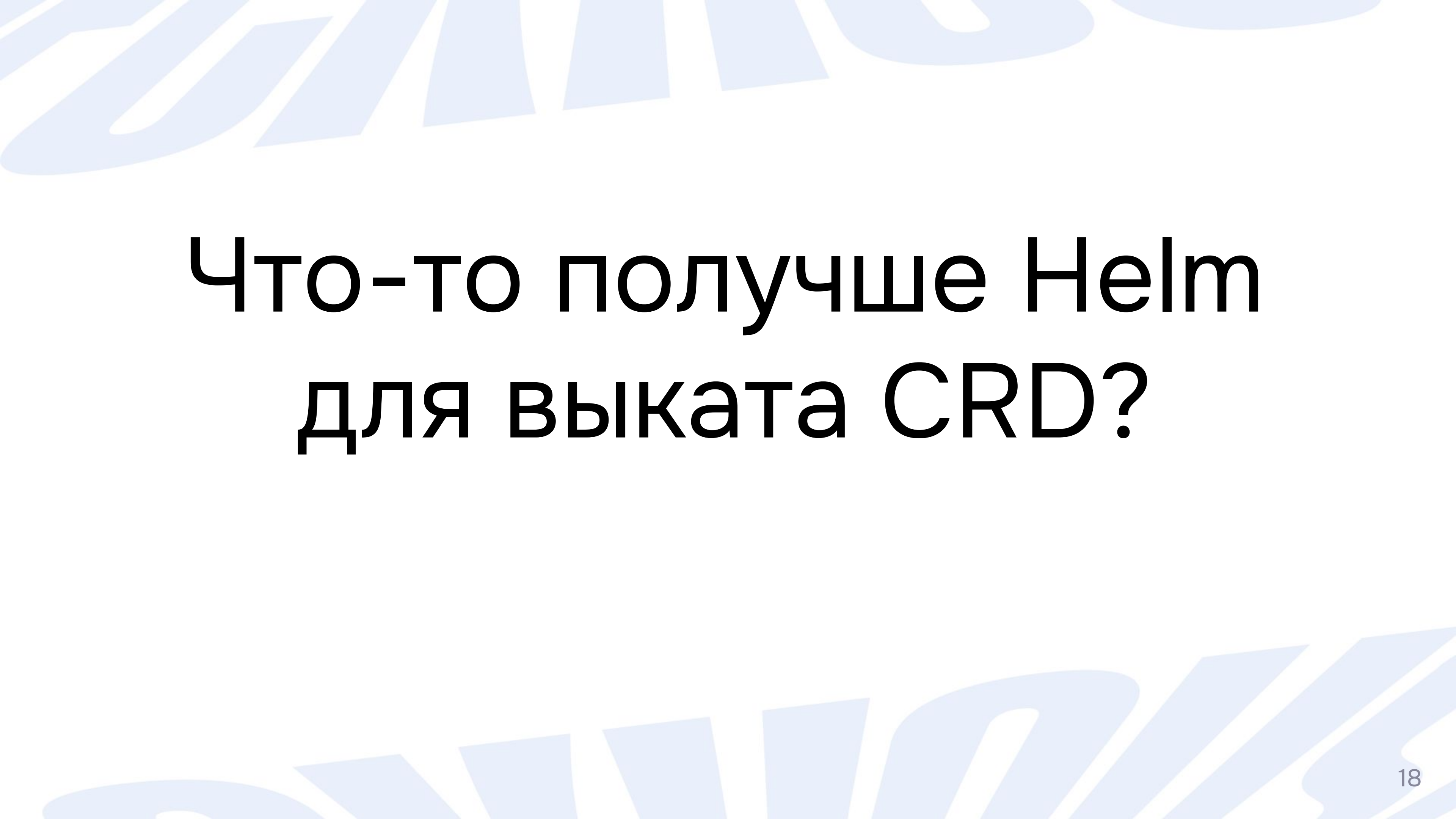
Что не так с деплоем CRD
из [templates/](#)?

- CRD не влезают в Secret релиза

Выкат CRD: **Helm**

Как выкручиваются разработчики Helm-чартов?

- Самый популярный чарт на artifacthub.io:
`kube-prometheus-stack`
- Не используют нативные способы выката CRD
- Сохраняют CRD в ConfigMap и запускают Job, внутри которой выполняются `kubectl get configmap` и `kubectl apply`



Что-то получше Helm
для выката CRD?

Выкат CRD: *Flux*

- Катит CRD отдельной ранней стадией
- Не полагается на Helm для выката CRD
- Проблем нет

Выкат CRD: *Argo CD*

- Катит CRD до их CR, даже если задано иное
- Проблем нет

Выкат CRD: *Nelm* и *Nelm Operator*

- Катит CRD отдельной ранней стадией
- Проблем нет

Выкат CRD

<i>kubectl</i>	Крайне проблемный
<i>Helm</i>	Проблемный
<i>Argo CD</i>	Полный
<i>Flux</i>	Полный
<i>Nelm</i>	Полный
<i>Nelm Operator</i>	Полный

Выкатить и *дождаться*
ГОТОВНОСТИ одного ресурса
перед другим?

Выкат Job с Deployment

- Сначала применить Job
- Дождаться *выполнения* Job
- Применить Deployment
- А иногда Job нужно выкатить после Deployment

Выкат Job: kubectl apply

- Выкатывает в порядке объявления манифестов в файле
- Готовности Job не ждёт

Выкат Job: *Helm* и *Flux*

Порядок задаётся через объявление Job'ы хуком:

```
kind: Job
```

```
metadata:
```

```
  annotations:
```

```
    helm.sh/hook: pre-install,pre-upgrade
```

```
    helm.sh/weight: "0"
```

Выкат Job: Helm и Flux

- Job катится отдельной ранней стадией
- Ожидается готовность Job

Выкат Job: *Helm* и *Flux*

Проблемы:

- Плохо подходит для выката StatefulSet > Job > Deployment
- Т. к. незэфемерные ресурсы-хуки плохо переименовываются и не удаляются

Выкат Job: *Argo CD*

Порядок задаётся через объявление Job'ы хуком:

```
kind: Job
```

```
metadata:
```

```
  annotations:
```

```
    helm.sh/hook: pre-install,pre-upgrade
```

```
    helm.sh/weight: "0"
```

Выкат Job: Argo CD

...или через волны:

kind: Job

metadata:

annotations:

argocd.argoproj.io/sync-wave: "-1"

Выкат Job: *Argo CD*

Выкат волнами:

- Дождидается готовности Job
- Подходит для деплоя неэфемерных ресурсов
- Проблем нет

Выкат Job: *Nelm* и *Nelm Operator*

Через хуки:

kind: Job

metadata:

annotations:

helm.sh/hook: pre-install,pre-upgrade

helm.sh/weight: "0"

Выкат Job: **Nelm** и **Nelm Operator**

Через веса (аналог волн):

kind: Job

metadata:

annotations:

werf.io/weight: "-1"

Выкат Job: *Nelm* и *Nelm Operator*

...но лучше через прямые зависимости:

```
kind: Deployment
```

```
metadata:
```

```
  annotations:
```

```
    werf.io/deploy-dependency-1:
```

```
      "state=ready,kind=Job,name=migrate"
```

Выкат Job: *Nelm* и *Nelm Operator*

Прямые зависимости

- Выкат графом — максимальная параллелизация
- Не нужно синхронизировать стадии между сабчартами
- Можно зависеть от внешних ресурсов

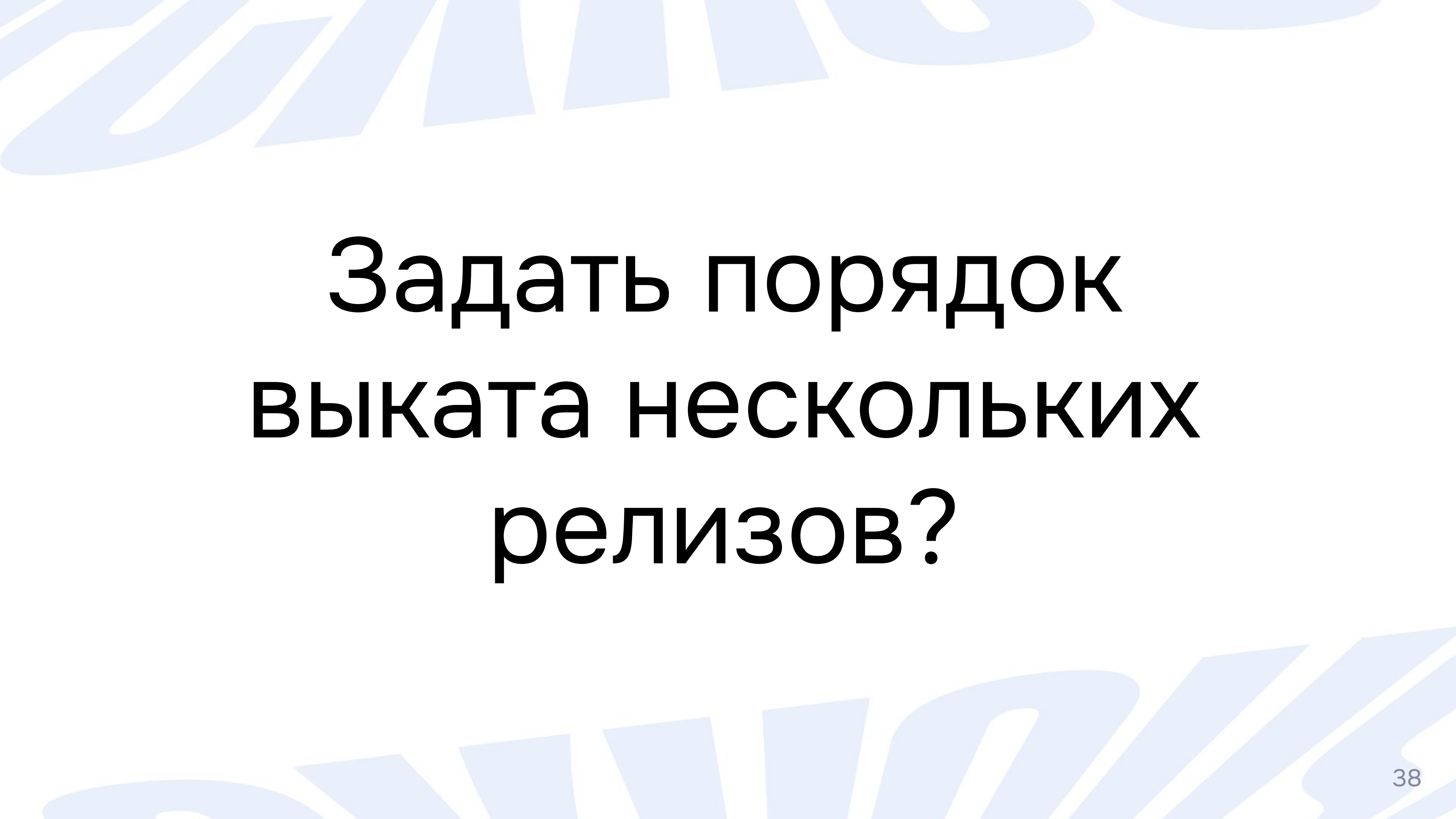
Выкат Job: *Nelm* и *Nelm Operator*

Швейцарский нож для задания порядка

- Можно зависеть от готовности, присутствия или отсутствия
- Можно объявить зависимость по имени чарта, `kind`, `apiVersion`, `name`, `namespace`
- Везде поддерживается regex

Задание порядка выката

<i>kubectl</i>	Вручную
<i>Helm</i>	Хуки
<i>Argo CD</i>	Стадии
<i>Flux</i>	Хуки
<i>Nelm</i>	Хуки, стадии, зависимости
<i>Nelm Operator</i>	Хуки, стадии, зависимости



**Задать порядок
выката нескольких
релизов?**

Несколько релизов: *kubectl*

- Нет оркестрации нескольких релизов
- Решается портянками на shell

Несколько релизов: Helm

- Нет оркестрации нескольких релизов
- Решается сторонним ПО: helmwave, helmfile

Несколько релизов: *Argo CD*

App of Apps:

```
kind: Application
```

```
metadata:
```

```
  annotations:
```

```
    argocd.argoproj.io/sync-wave: "-1"
```

```
---
```

```
kind: Application
```

```
metadata:
```

```
  annotations:
```

```
    argocd.argoproj.io/sync-wave: "0"
```

Несколько релизов: *Argo CD*

ApplicationSet:

```
kind: ApplicationSet
```

```
spec:
```

```
  strategy:
```

```
    rollingSync:
```

```
      steps:
```

```
        - matchExpressions:
```

```
          - { key: tier, operator: In, values: ["1"] }
```

```
        - matchExpressions:
```

```
          - { key: tier, operator: In, values: ["2"] }
```

Несколько релизов: *Flux*

Граф релизов через dependsOn:

```
kind: HelmRelease
spec:
  dependsOn:
    - name: another-release
```

Несколько релизов: *Nelm*

- Нет оркестрации нескольких релизов
- Экспериментально: nelmwave (спасибо разработчику nelmwave)

Несколько релизов: *Nelm Operator*

- Нет оркестрации нескольких релизов
- В планах выкат графом через аналог dependsOn у Flux

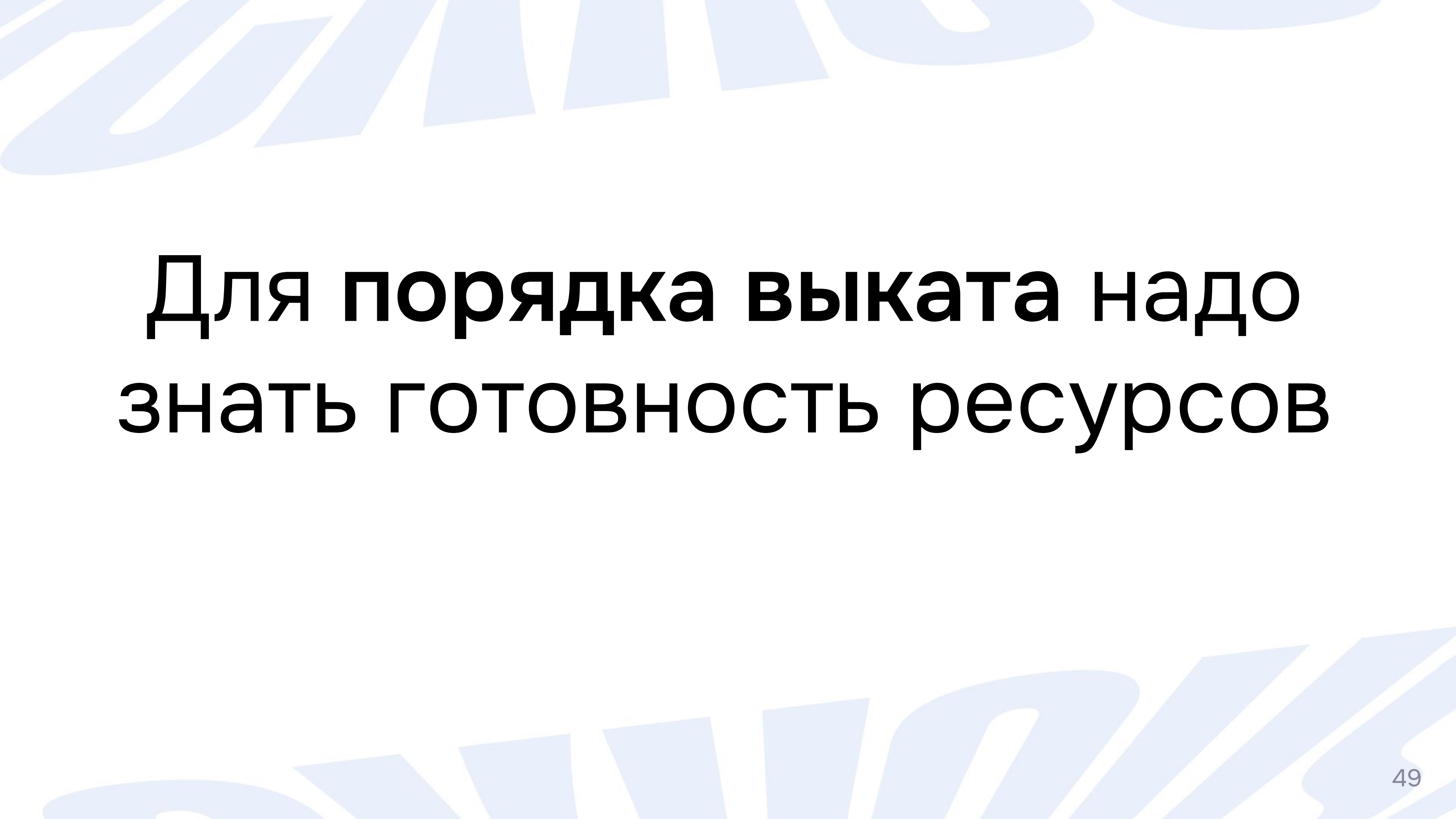
Порядок выката релизов

<i>kubectl</i>	Нет
<i>Helm</i>	Стороннее ПО
<i>Argo CD</i>	Есть
<i>Flux</i>	Есть
<i>Nelm</i>	Стороннее ПО
<i>Nelm Operator</i>	Нет

Итог: порядок выката

	Выкат CRD	Порядок выката ресурсов	Порядок выката релизов
<i>kubectl</i>	Крайне проблемный	Вручную	Нет
<i>Helm</i>	Проблемный	Хуки	Стороннее ПО
<i>Argo CD</i>	Полный	Стадии	Есть
<i>Flux</i>	Полный	Хуки	Есть
<i>Nelm</i>	Полный	Хуки, стадии, зависимости	Стороннее ПО
<i>Nelm Operator</i>	Полный	Хуки, стадии, зависимости	Нет

Отслеживание ГОТОВНОСТИ



**Для порядка выката надо
знать готовность ресурсов**

**Для статуса релиза надо
знать ГОТОВНОСТЬ ресурсов**

Индикаторы готовности ресурсов

- У Job это **Complete**
- У CRD – **Established** (нет)
- У всех кастомных ресурсов по-разному
- По итогу зеленый релиз – что-то готово, а что-то нет

Готовность стандартных ресурсов

- `kubectl apply` не отслеживает готовность
- Helm, Argo CD, Flux, Nelm и Nelm Operator отслеживают без проблем

Правила готовности для CR «из коробки»

<i>kubectl</i>	Нет
<i>Helm</i>	Нет
<i>Argo CD</i>	Большая библиотека правил для популярных CR
<i>Flux</i>	Нет
<i>Nelm</i>	Небольшая библиотека правил для популярных CR
<i>Nelm Operator</i>	Небольшая библиотека правил для популярных CR

Кастомные правила: *Helm, Nelm/Operator*

Нет возможности добавить кастомные правила
отслеживания для ресурсов

Кастомные правила: *Flux*

Можно писать правила на CEL:

```
healthCheckExprs:  
  - current: status.conditions.filter  
(e, e.type == 'Ready').all(e, e.status == 'True')  
  
  failed: status.conditions.filter  
(e, e.type == 'Ready').all(e, e.status == 'False')
```

Кастомные правила: *Argo CD*

Можно писать правила на Lua:

```
if condition.type == "Ready" and  
condition.status == "False" then  
    hs.status = "Degraded"  
    return hs  
end
```

Угадывание ГОТОВНОСТИ: *Helm, Flux*

- Используют библиотеку kstatus
- Ищут condition **Reconciling, Stalled, Ready**

Угадывание готовности: *Nelm/Operator*

- Эвристика с большим количеством правил:
 - Ищут condition *Reconciling, Stalled, Ready*
 - Ищут десятки других полей в status
- Определяют готовность ~50 % неизвестных CR

Угадывание готовности: *Nelm/Operator*

- Детектируют не только готовность, но и ошибки выката
- Последние пару лет не приходили с false positives
- Эвристика выключается поресурсно

Итог: готовность ресурсов

	Стандартные ресурсы	Библиотека правил для CR	Кастомные правила	Угадывание готовности
<i>kubectl</i>	Нет	Нет	Нет	Нет
<i>Helm</i>	Да	Нет	Нет	Ограниченное
<i>Argo CD</i>	Да	Большая	Lua	Нет
<i>Flux</i>	Да	Нет	CEL	Ограниченное
<i>Nelm</i>	Да	Малая	Нет	Продвинутое
<i>Nelm Operator</i>	Да	Малая	Нет	Продвинутое

Жизненный цикл ресурсов

Обновление immutable- ресурсов

Immutable в Kubernetes

- Ресурсы в Kubernetes часто имеют immutable-поля
- Обновить можно только пересозданием
- Пересоздание = удалить + создать

Эфемерные иммутабельные ресурсы

Эфемерные иммутабельные ресурсы

Поменяем `image: ubuntu` на `alpine` в выкаченной Job:

```
kind: Job
spec:
  template:
    spec:
      containers:
      - image: alpine
```

Эфемерные иммутабельные ресурсы

Получим ошибку, что поле нельзя менять:

```
The Job "migrate" is invalid: spec.template:  
Invalid value: ... field is immutable
```

Эфемерные ресурсы: *kubectl*

- Нельзя переименовать ресурс (получится два ресурса)
- Можно удалить вручную перед `kubectl apply`

Эфемерные ресурсы: *Helm, Flux*

- В шаблоне переименовывать при каждом релизе
- Объявить ресурс пересоздающимся хуком:

```
kind: Job
```

```
metadata:
```

```
  annotations:
```

```
    helm.sh/hook: pre-install,pre-upgrade
```

```
    helm.sh/hook-delete-policy: before-hook-creation
```

Эфемерные ресурсы: *Argo CD*

- То же, что и в Helm/Flux
- Всегда пересоздавать ресурс:

`metadata:`

`annotations:`

`argocd.argoproj.io/sync-options: Replace=true,Force=true`

Эфемерные ресурсы: *Nelm/Operator*

- То же, что и в Helm/Flux/Argo
- Пересоздавать только при необходимости:

`metadata:`

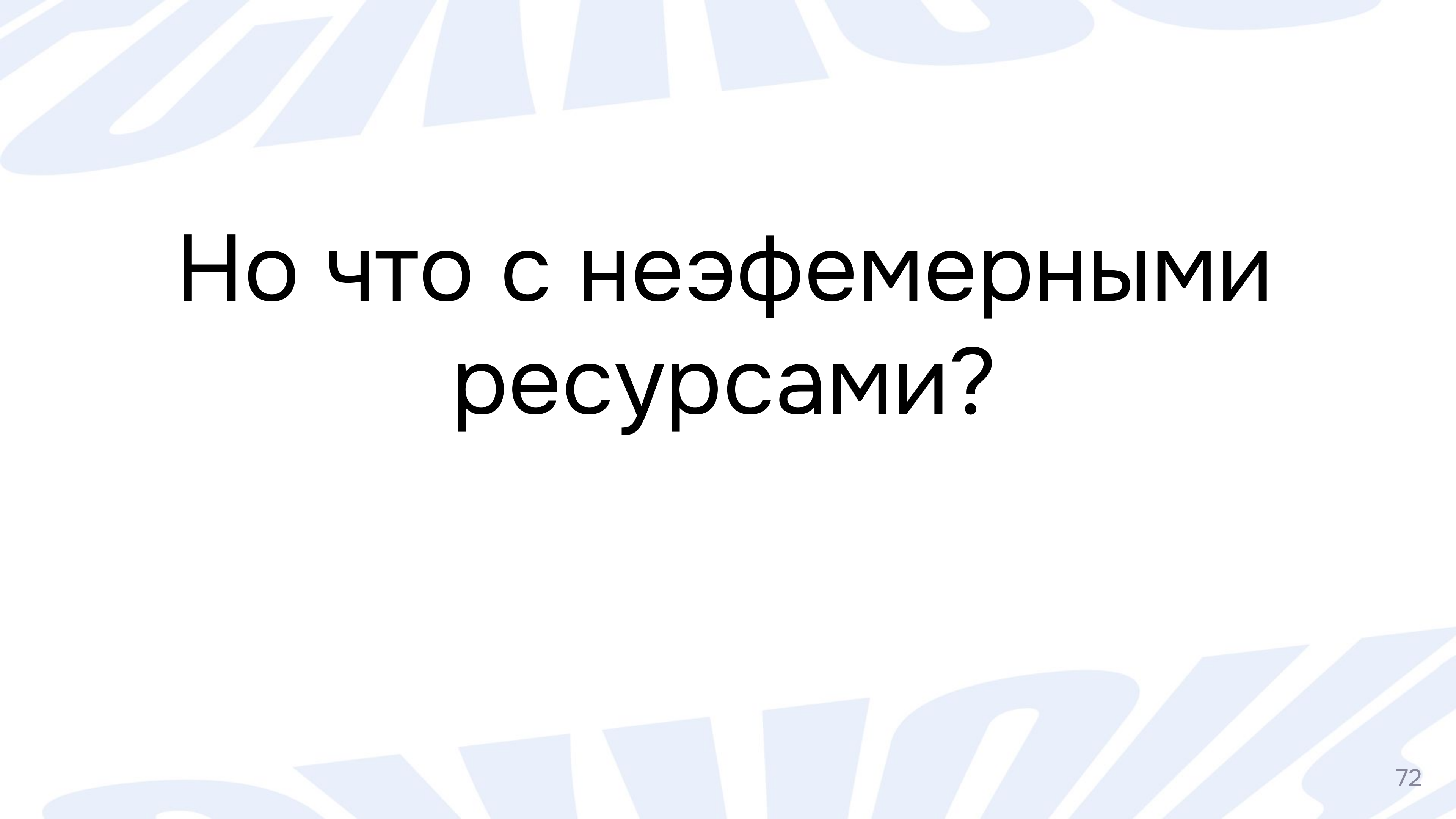
`annotations:`

`werf.io/delete-policy: before-creation-if-immutable`

Эфемерные ресурсы: *Nelm/Operator*

Аннотация `werf.io/delete-policy`:

- Поддерживает то же, что и `helm.sh/hook-delete-policy`
- Применяется на любые ресурсы, не только хуки
- Пересоздавать только при immutable-ошибке — `before-creation-if-immutable`



Но что с неэфемерными
ресурсами?

Неэфемерные иммутабельные ресурсы

Поменяем **image: ubuntu** на **alpine** в выкаченном StatefulSet:

```
kind: StatefulSet
spec:
  template:
    spec:
      containers:
        - image: alpine
```

Неэфемерные иммутабельные ресурсы

Получим ошибку, что поле нельзя менять:

```
The StatefulSet "db" is invalid: ... field is  
immutable
```

Неэфемерные ресурсы: *kubectl*, *Helm*, *Flux*

- Удалить вручную перед выкатом
- Других вариантов нет

Неэфемерные ресурсы: *Argo CD*

- Удалить вручную перед выкатом
- Навесить аннотацию, выкатить, снять аннотацию:

`metadata:`

`annotations:`

`argocd.argoproj.io/sync-options: Replace=true, Force=true`

Неэфемерные ресурсы: *Nelm/Operator*

- То же, что и в Argo CD
- Пересоздавать только при необходимости:

metadata:

annotations:

werf.io/delete-policy: before-creation-if-immutable

Обновление иммутабельных ресурсов

	Эфемерные ресурсы	Неэфемерные ресурсы
<i>kubectl</i>	Вручную	Вручную
<i>Helm</i>	Автоматически	Вручную
<i>Argo CD</i>	Автоматически	Вручную
<i>Flux</i>	Автоматически	Вручную
<i>Nelm</i>	Автоматически	Автоматически
<i>Nelm Operator</i>	Автоматически	Автоматически

Сравнительная таблица

	<i>kubectl</i>	<i>Helm</i>	<i>Argo CD</i>	<i>Flux</i>	<i>Nelm</i>
Выкат CRD	—	+	+	+	+
Порядок выката ресурсов	+	+	+	+	+
Порядок выката релизов	—	+	+	+	—
Готовность стандартных ресурсов	—	+	+	+	+
Правила готовности из коробки	—	—	+	—	+
Кастомные правила готовности	—	—	+	+	—
Угадывание готовности	—	+	—	+	+
Иммутабельные ресурсы	—	+	+	+	+

Nelm и Nelm Operator

Новые фишки из коробки:

- Планирование релиза как в terraform
- Локальная валидация (Kubecconform+)
- Патчинг ресурсов (gojq)
- Секреты в Git
- TypeScript как альтернатива Helm-шаблонам (Deno)

deckhouse

Спасибо за ВНИМАНИЕ

✈ @werf_ru

🐙 github.com/werf/nelm

🐙 github.com/werf/nelm-operator

🐙 github.com/werf/werf



Deckhouse
Platform Open