Превращение из Operations в Solution providers

Константин Дипеж

Сообщество DeusOps

vk.com/deusops

deusops

Обо мне

Константин Брюханов

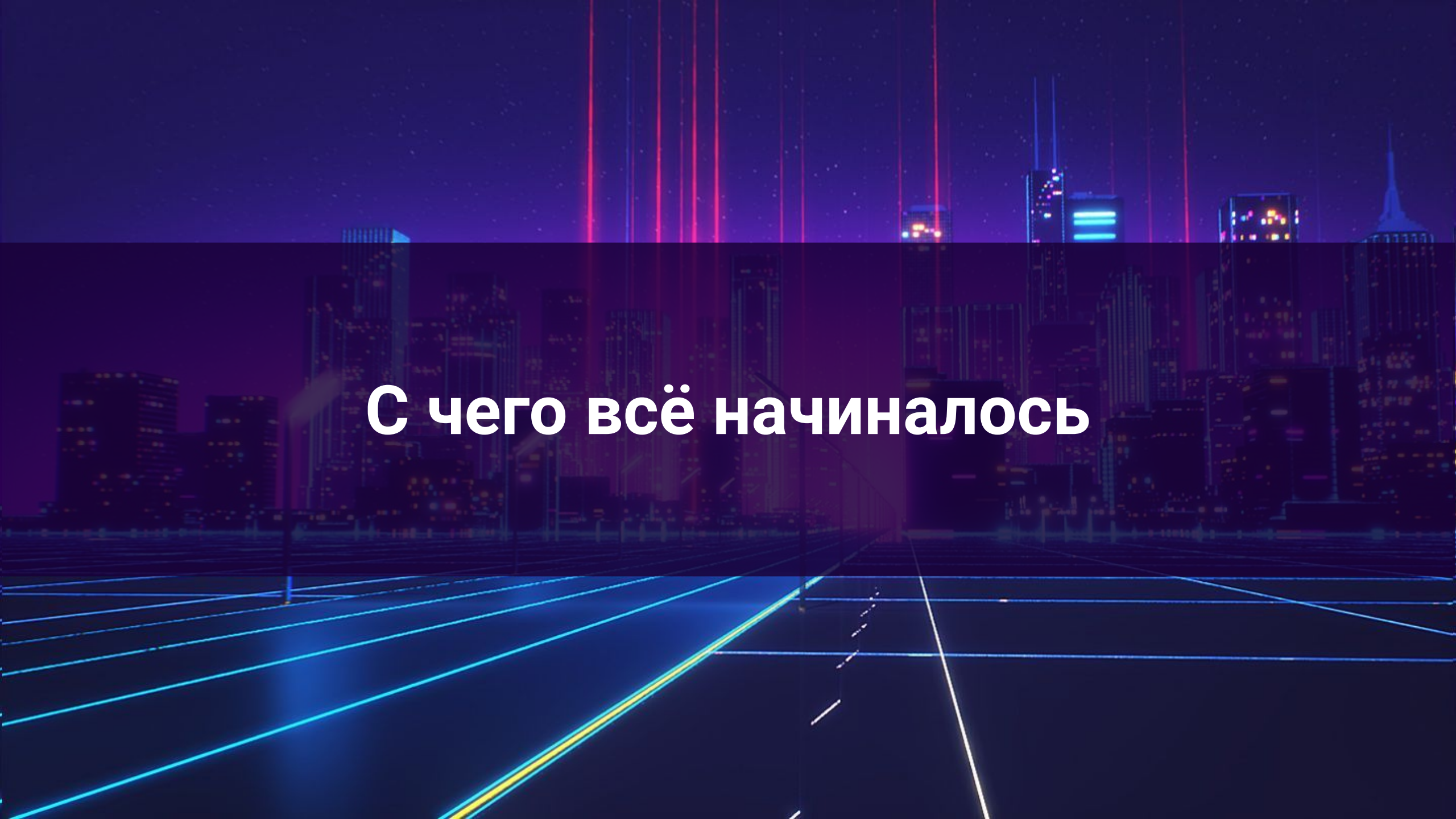
- Основатель и генеральный директор ООО Деусопс
- Получаю PhD с диссертацией о ML для задач DevOps
- Создал и преподаю дисциплины “**Инфраструктура как код**” и “**Технологии сборки и развертывания ПО**” в университете **ИТМО**
- Создатель **DeusOps**
- Ведущий одноименного подкаста

Обо мне

DEUS
OPS

DeusOps это:

- Сообщество учащихся и работающих DevOps-специалистов
- Youtube-канал с живыми воркшопами и разборами рабочих задач
- Стримы и подкасты с приглашенными гостями
- Платформа для Continuous Education


С чего всё начиналось

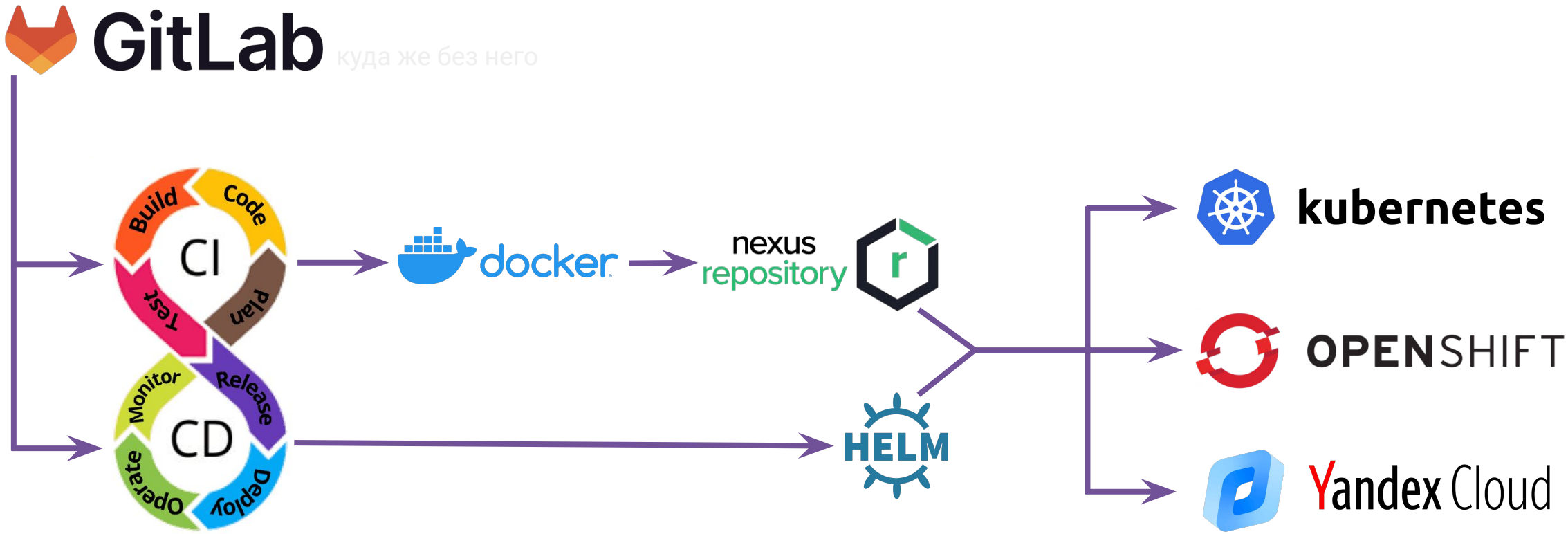
С чего всё начиналось

Мы пришли в новую классную компанию... точно не банк

- Много опытных специалистов
- Тут есть девопсы!
- Большое количество разрабатываемых сервисов
- Всё делается с нуля
- Крайне амбициозные сроки

С чего всё начиналось

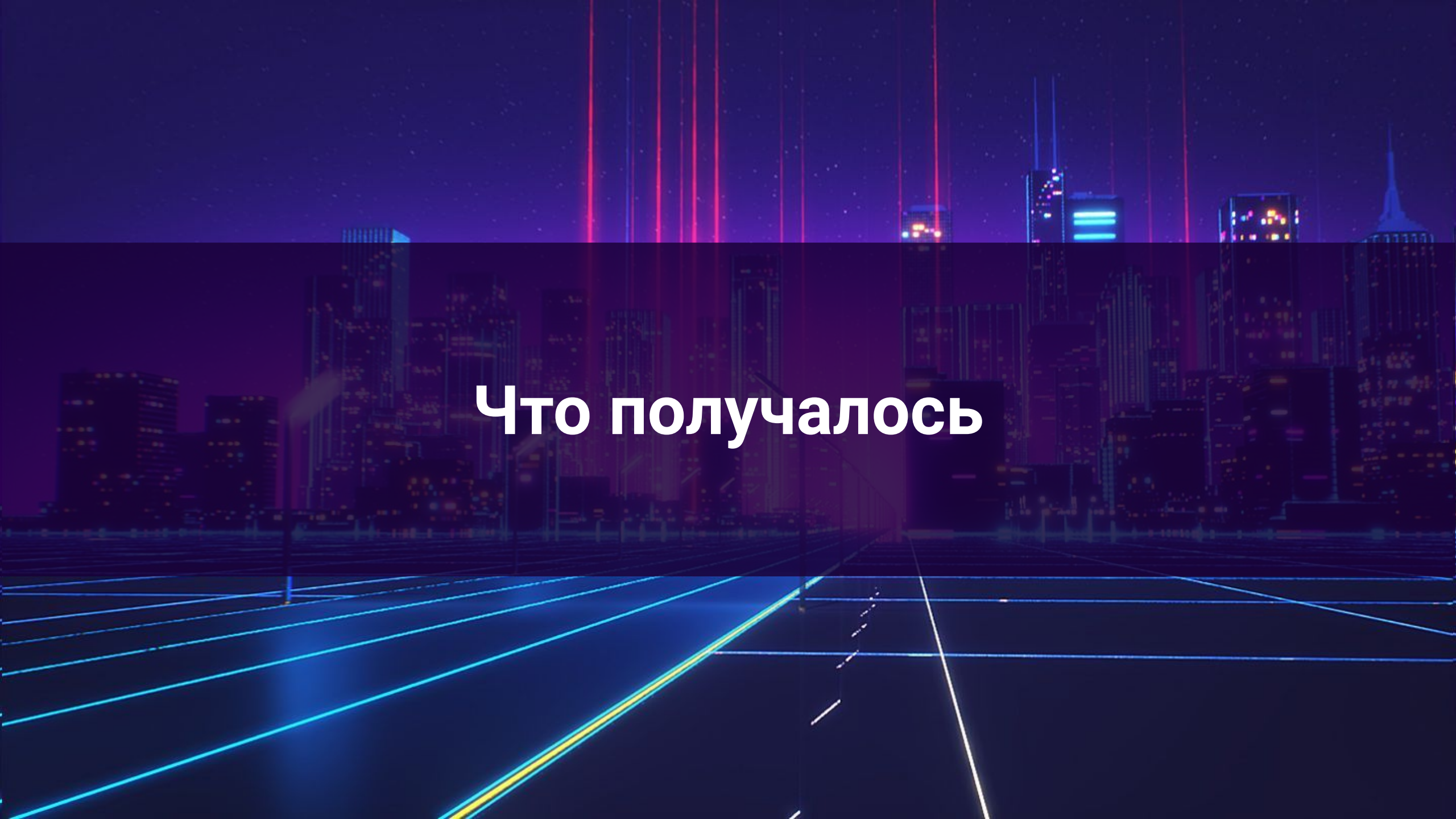
Целевые инструменты на проекте



С чего всё начиналось

Что же мы придумали

- Монорепные монолиты распиливались на микросервисы
- Проекты делились на гитлаб-сабгруппы
- Микросервисы разбивались на разные репозитории
- В каждом проекте хранилось всё, что нужно для запуска
    - gitlab-ci.yml
    - Dockerfile
    - helm-chart


Что получалось

Что получалось

Количество отдевопсеных проектов

проект 

Что получалось

Количество отдевопсеных проектов

проект

проект

проект

Что получалось

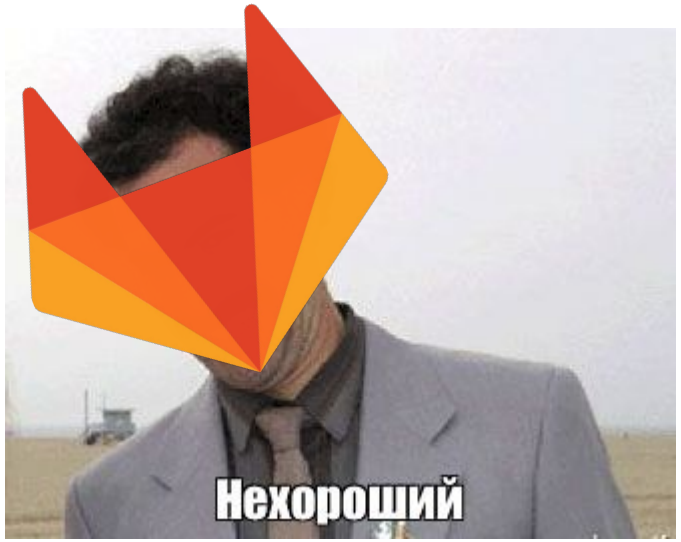
Количество отдевопсеных проектов

проект

проект

получили новое требование – правим все докерфайлы

проект



проект

проект

Что получалось

Количество отдевопсеных проектов

проект

проект

получили новое требование – правим все докерфайлы

проект

проект

НЕ ОЧЕНЬ!!

проект

проект

нашли баг в ci - правим все gitlab-ci.yml

проект

проект

проект

Что получалось

Какие проблемы?

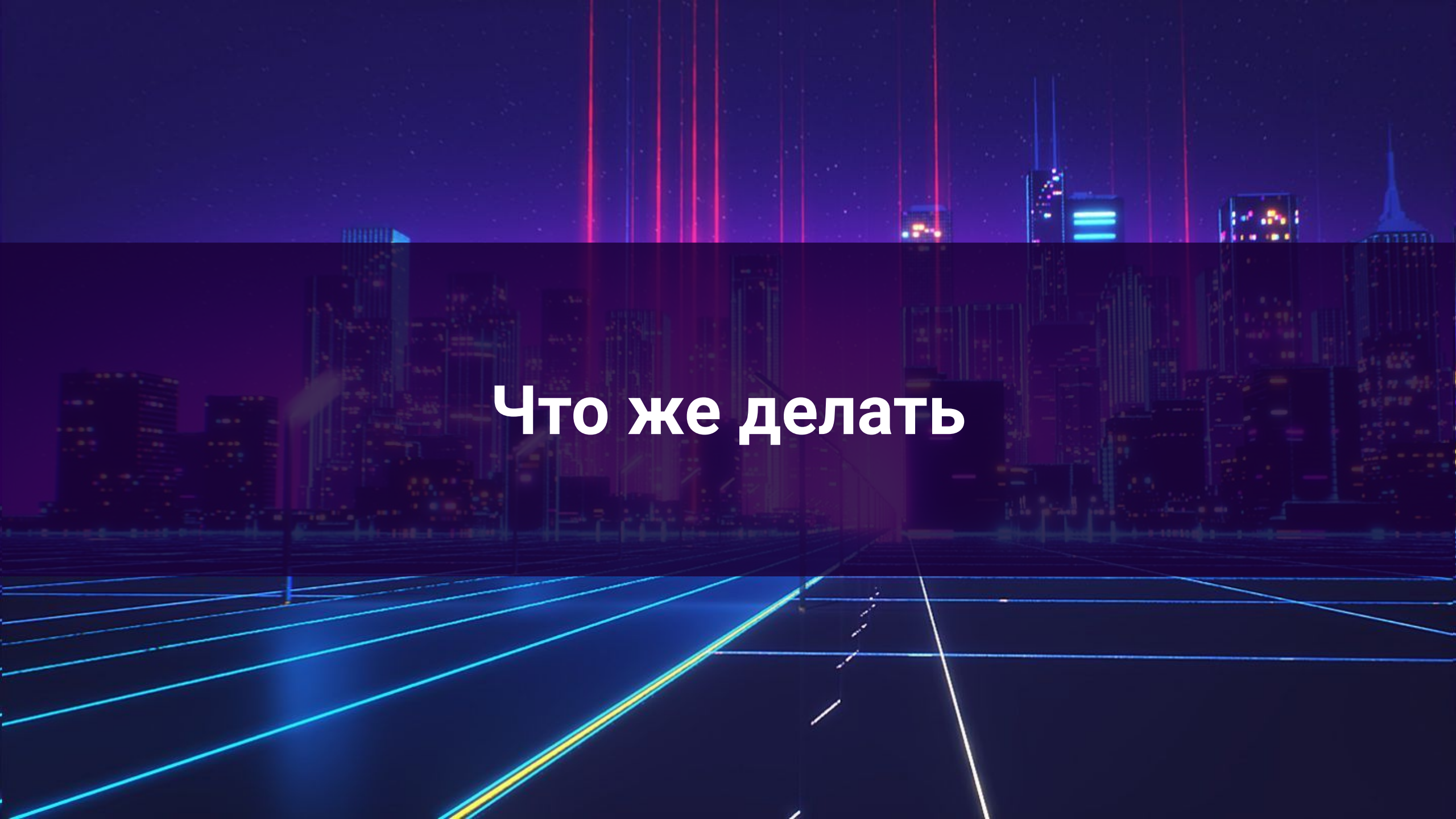
- Невозможность одной правкой заменить ошибку во всех проектах
- Необходимость вносить правки в растущем количестве сущностей
- Необходимость поддерживать версии devops-штук во всех проектах
- Каждый новый проект увеличивает энтропию
- Это мы еще до helm-чартов с их ингрессами не дошли…

Что получалось

Баланс изменился..

сопровождение и фиксы девопсовских штук в старых проектах

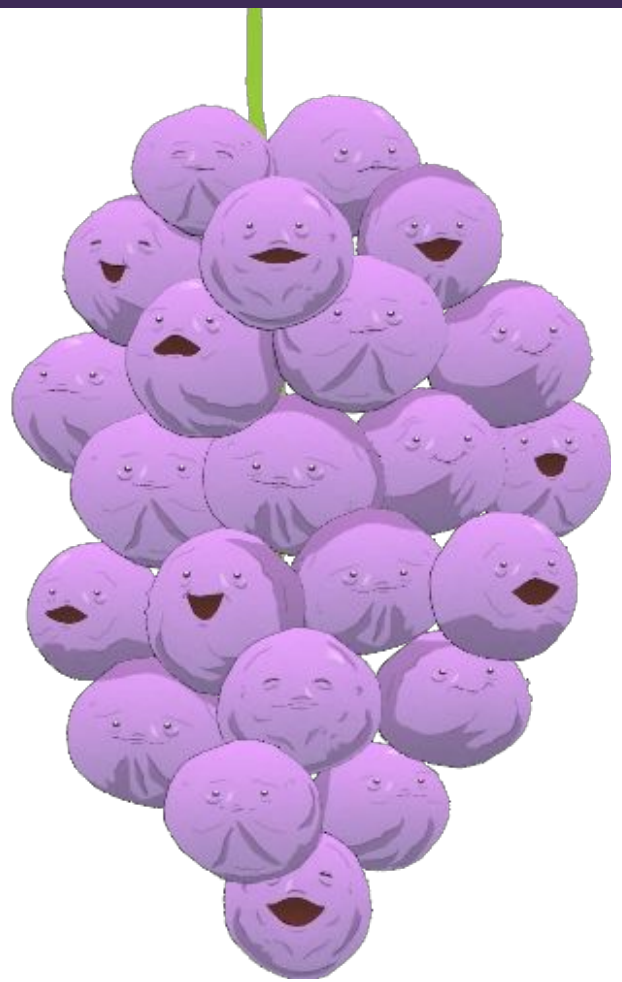
Отдевопсить новый проект

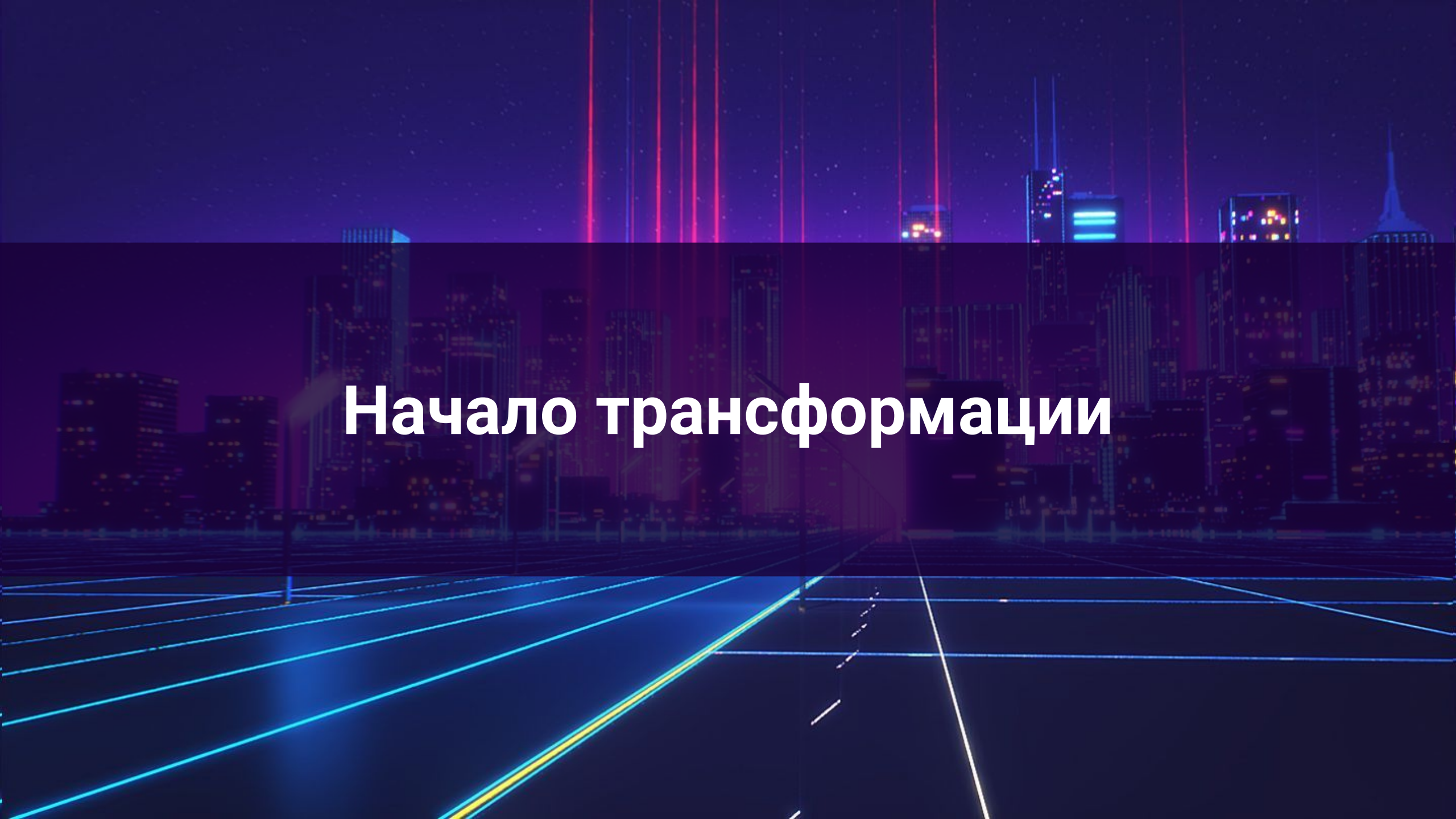

Что же делать

Что же делать

Принимаем вспоминашки

- Вспоминаем про “переиспользование кода”
- И что наш gitlab-ci, докерфайлы и хелм-чарты – это код
- И что всем этим пользуются “заказчики” в виде команд
- И о практике вынесения ansible-ролей в отдельные репозитории
- ???
- PROFIT




Начало трансформации

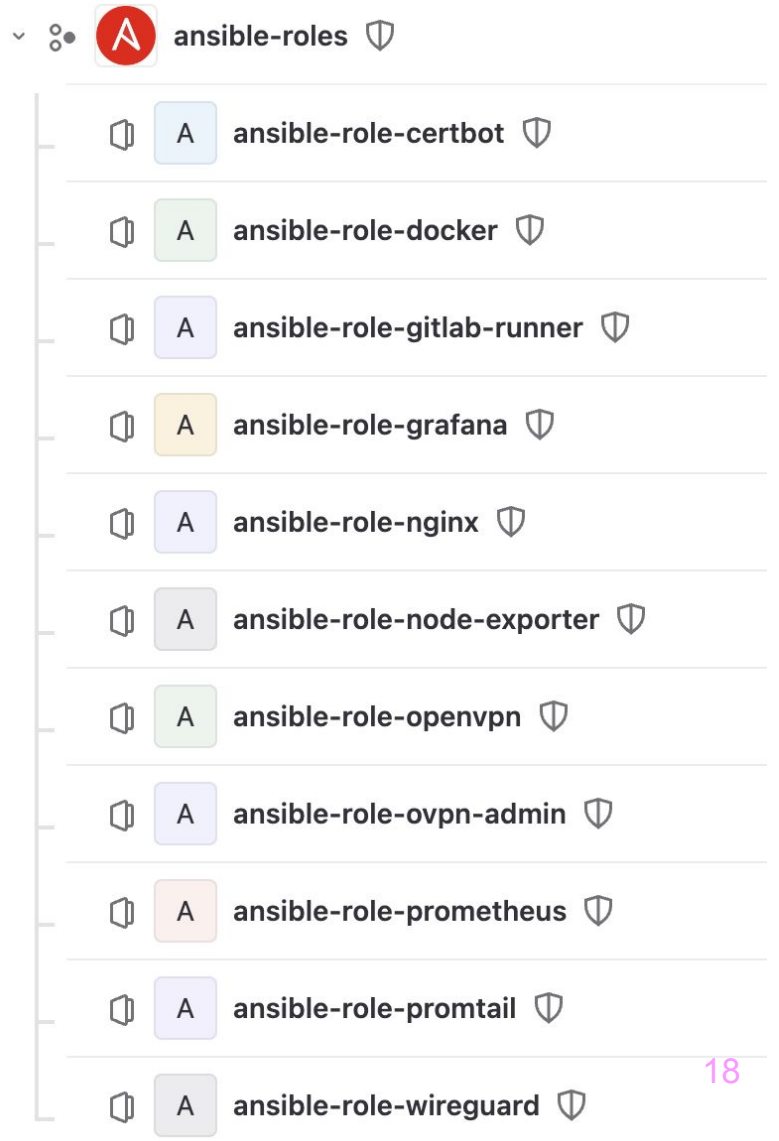
Начало трансформации

Вспоминаем про DRY

Мы с вами постоянно говорим о том, что нужно отделять общее от частного, и мы уже давно умеем выносить **ansible-роли** отдельно от проектов и подключать их, настраивая параметры через переопределяемые переменные

И terraform-модули мы разрабатываем так же…

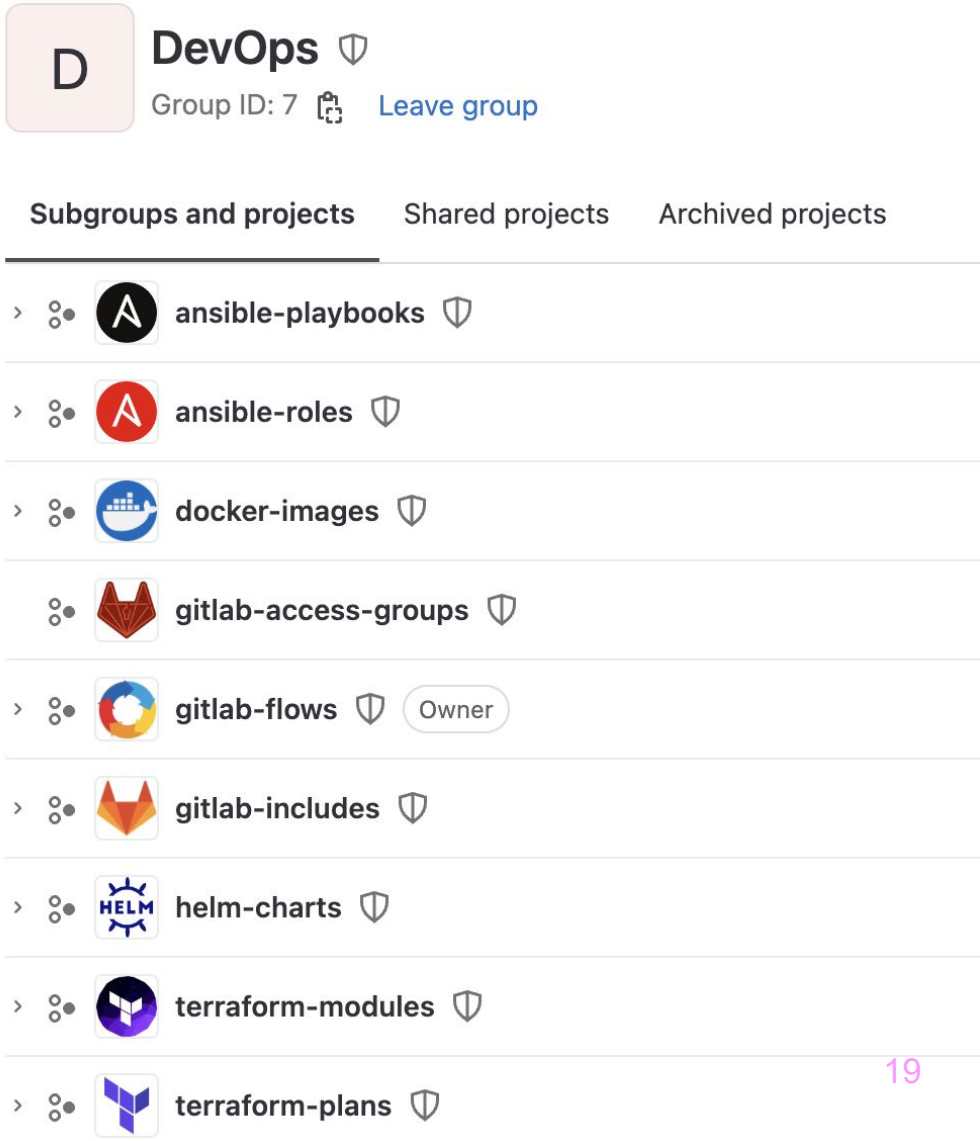
Неужели, это всё, на что мы способны? Конечно же нет!



Начало трансформации

Инкапсулируемся

- Создали глобальную группу в гитлабе – **DevOps**
- В ней мы разместили подгруппы для каждого типа devops-ингридиентов – которые мы убираем из проектов
- Давайте о них по порядку

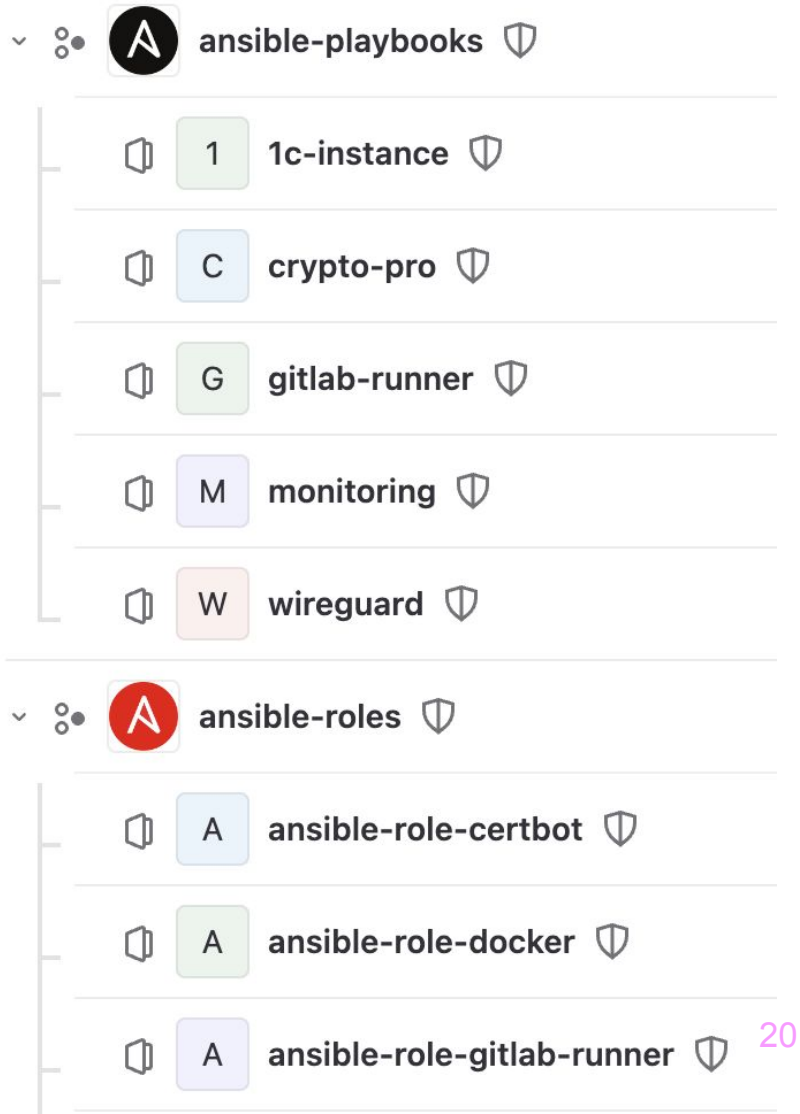


Начало трансформации

Ansible

Пока что, всё понятно:

- Подгруппа для проектов с ansible-плейбуками – тут мы храним конфигурации для определенных наборов ролей под инвентари, а также зависимости с указанием этих ролей
- Подгруппа для проектов с ansible-ролями – атомарные конфигурируемые роли

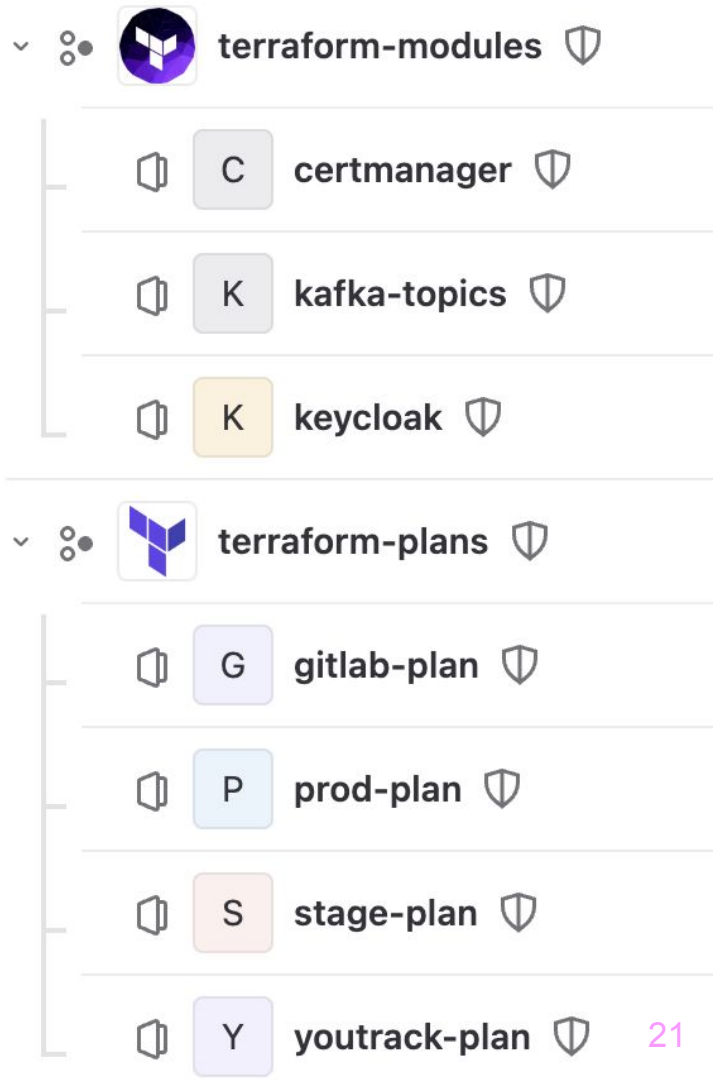


Начало трансформации

Terraform

После предыдущего слайда и тут всё понятно:

- Подгруппа для проектов с terraform-манифестами – мы эти штуки называем “планы”. Здесь лежат описания контуров либо площадок, с указанием модулей необходимых для работы
- Подгруппа для проектов с terraform-модулями – атомарные конфигурируемые модули



Начало трансформации

Неинфраструктурные сущности

- **Еще одна подгруппа – это helm-чарты**

  Мы поняли, что нет никакого смысла хранить чарты в репозиториях проектов, ведь почти всегда для однотипных проектов – чарт будет идентичен, и мы лишь конфигурируем его через **values**
- **Ну и подгруппа для docker-образов**

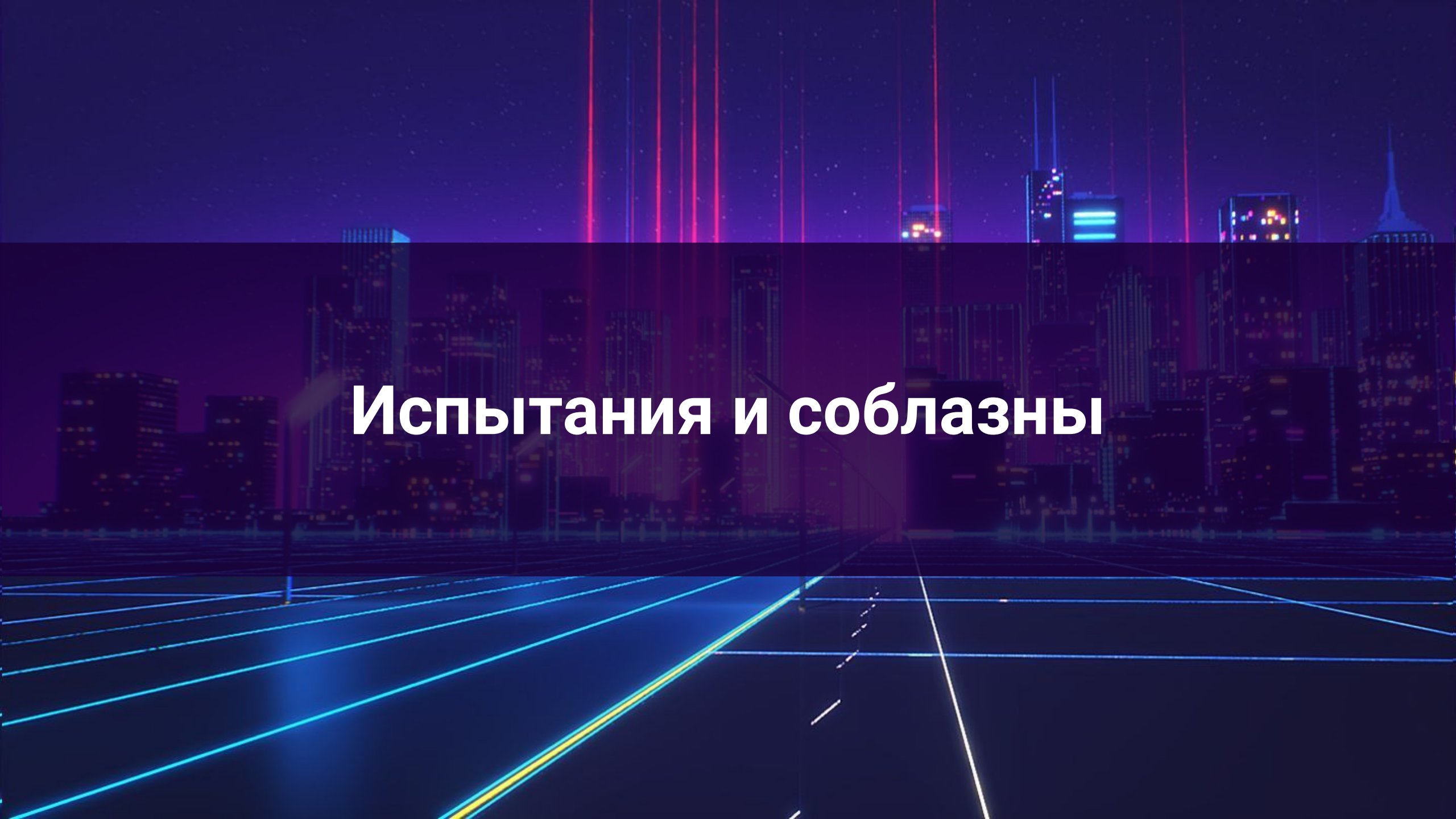
  Началось всё с базовых образов, но мы быстро поняли, что однотипные приложения в целом требуют один и тот же образ, с возможностью конфигурирования через **ARG**

Начало трансформации

Святой грааль пайплайнов

Важнейшей задачей стало убрать из репозиториев последнюю сущность – описания пайплайнов.

Это делалось в несколько этапов. Мы ошибались и учились.



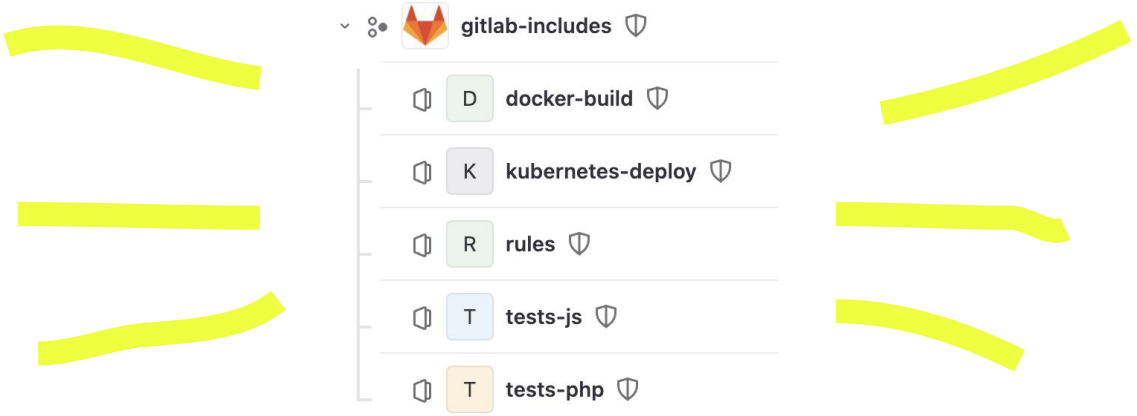
Испытания и соблазны

Испытания и соблазны

Что же делать с пайплайнами

Первое, что приходит на ум – вынести повторяющиеся части в отдельные куски кода

- Сможем их вызывать в нужном месте!
- Сможем их тюнить, если нигде не накосячим с переменными!
- Даже сможем их инклудить из других репозиториев!
- О, мы их тоже уберем в наш devops-репозиторий и будем там версионировать!



Испытания и соблазны

Главный соблазн

- Нужно быть очень осторожными и не упарываться по инклудам
- Относитесь к gitlab-инклудам, как к ansible-ролям – делайте атомарные работающие куски кода, которые можно тюнить переопределяя дефолтные переменные
- Никогда-никогда-никогда не создавайте Include Hell… и бегите если попали в него

Испытания и соблазны

Пример gitlab-includes

Примерное содержание одного из файлов с вынесенными атомарными gitlab-задачами

docker-build.yml

```
.docker_login: &docker-login
    - echo "$REGISTRY_PASSWORD" | docker login
      -u "$REGISTRY_USER" --password-stdin $REGISTRY

.docker_build: &docker-build
    - if [[ "${CI_COMMIT_REF_SLUG}" == "main" ]]; then
        IMAGE_TAG="latest"; fi
    - docker build
        --build-arg AUTOMATION_CLID=${AUTOMATION_CLID}
        --build-arg AUTOMATION_PT=${AUTOMATION_PT}
        -f ${DOCKERDIR}/${IMAGE_ID}/${DOCKERFILE}
        -t ${CI_REGISTRY_IMAGE}/${IMAGE_ID}:${IMAGE_TAG} .

.docker_push: &docker-push
    - *docker-login
    - if [[ ! -z "${CI_COMMIT_TAG}" ]]; then
        IMAGE_TAG="${CI_COMMIT_TAG}"; fi
    - if [[ "${CI_COMMIT_REF_SLUG}" == "main" ]]; then
        IMAGE_TAG="latest"; fi
    - docker push ${CI_REGISTRY_IMAGE}/${IMAGE_ID}:${IMAGE_TAG}
```

Испытания и соблазны

Пример gitlab-includes

Примерное содержание одного из файлов с вынесенными атомарными gitlab-задачами

docker-build.yml

```
.docker_build_dind:
  stage: build
  image: docker:stable
  services:
    - docker:dind
  variables:
    DOCKERDIR: "${CI_PROJECT_DIR}/dockerfiles"
    DOCKERFILE: "Dockerfile"
    REGISTRY: ${CI_REGISTRY}
    REGISTRY_USER: ${CI_REGISTRY_USER}
    REGISTRY_PASSWORD: ${CI_REGISTRY_PASSWORD}
    IMAGE_TAG: ${CI_COMMIT_REF_SLUG}
    IMAGE_PATH: ''
  script:
    - export IMAGE_ID=${CI_JOB_NAME#* *}
    - *docker-define-image-tag
    - *docker-login
    - *docker-build
    - *docker-push
  tags:
    - docker
```

Испытания и соблазны

Что же делать с пайплайнами

Мы вынесли повторяющиеся части пайплайна. Что же у нас остаётся?

- Сам пайплайн в gitlab-ci.yml в проекте. Теперь он выглядит как набор инклудов и скелет из джобов, которые вызываются. Кроме того, их можно тюнить через variables!
- **Gitlab CI Predefined variables** для конкретного проекта – помогают в тюнинге
- А еще есть **project variables** и **group variables** – помогают убрать частные настройки проекта и секреты

Испытания и соблазны

Пример gitlab-ci.yml

Примерное содержание файла с проектным пайплайном

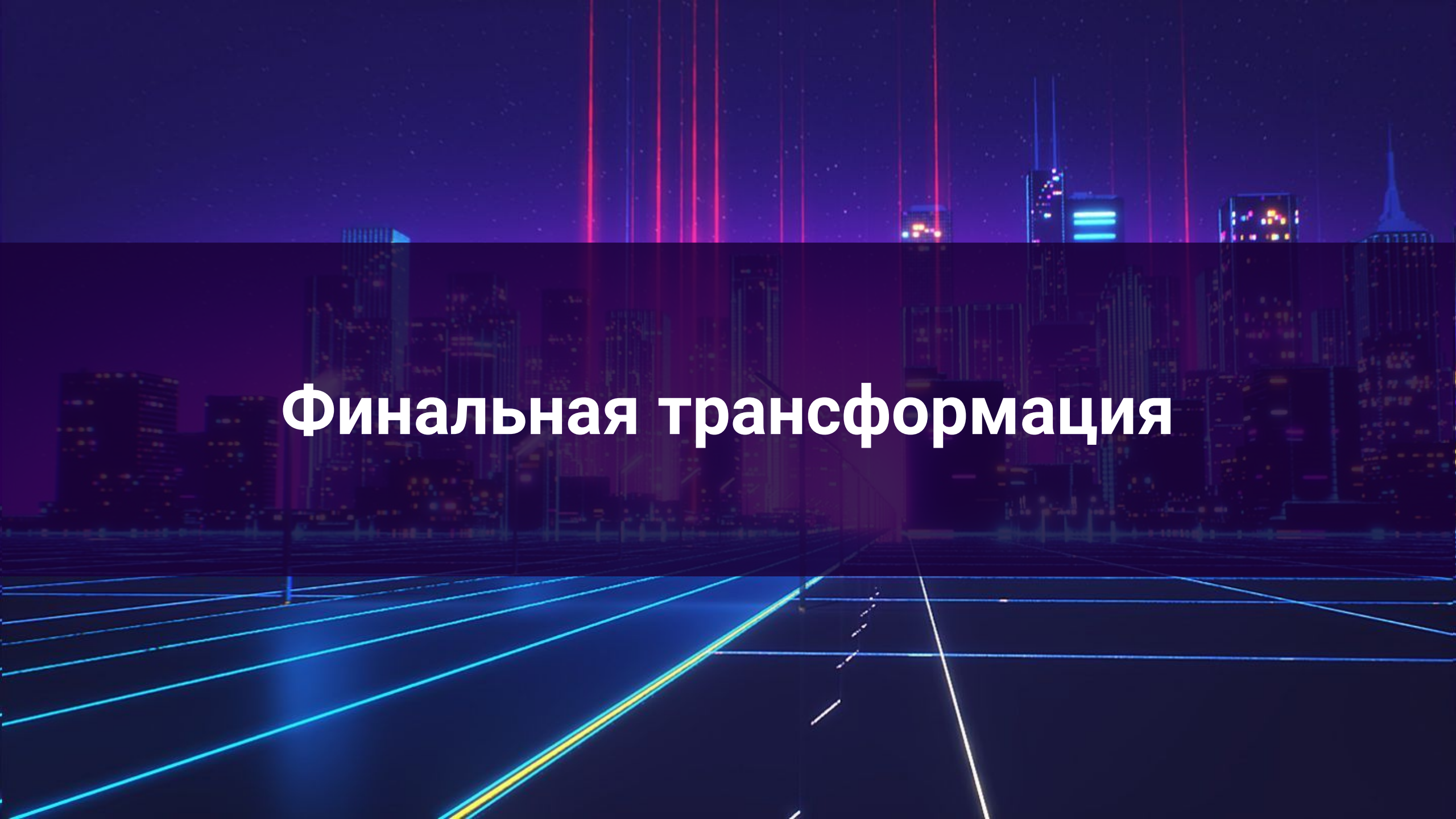
- Инклуды версионированы
- Их можно тюнить через **variables**
- У переменных есть дефолтные значения внутри самих инклудов, если их не обозначать (да, как у ansible-ролей)

.gitlab-ci.yml

```
include:
  - project: 'devops/gitlab-includes/docker-build'
    ref:  "main"
    file: 'docker-build.yml'
  - project: 'devops/gitlab-includes/kubernetes-deploy'
    ref:  "main"
    file: 'deploy-kubernetes.yml'

build php-fpm:
  extends:
    - .docker_build_dind

to prod:
  variables:
    KUBE_PORT: 9000
    KUBE_ENVFILE: ${KUBE_ENVFILE_PROD}
    KUBE_CONFIG: ${KUBE_CONFIG_PROD}
    VALUES: ${VALUES_PROD}
  needs: ["build php-fpm"]
  extends:
    - .kube_deploy_back
```


Финальная трансформация

Финальная трансформация

Что же мы упускаем?

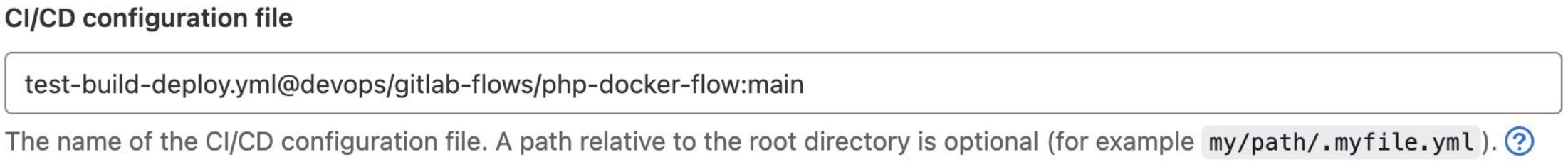
Наши проекты стали выглядеть чистенько, но всё еще что-то не так

- В проектах по прежнему валяется .gitlab-ci.yml
- Более того – он еще и почти всегда одинаковый
- Словно бы мы упускаем самую суть...

Финальная трансформация

Подключаем пайплайн извне

Если наш пайплайн является типовым для проектов и конфигурируется через переменные – он сам может являться неким переиспользуемым кодом!

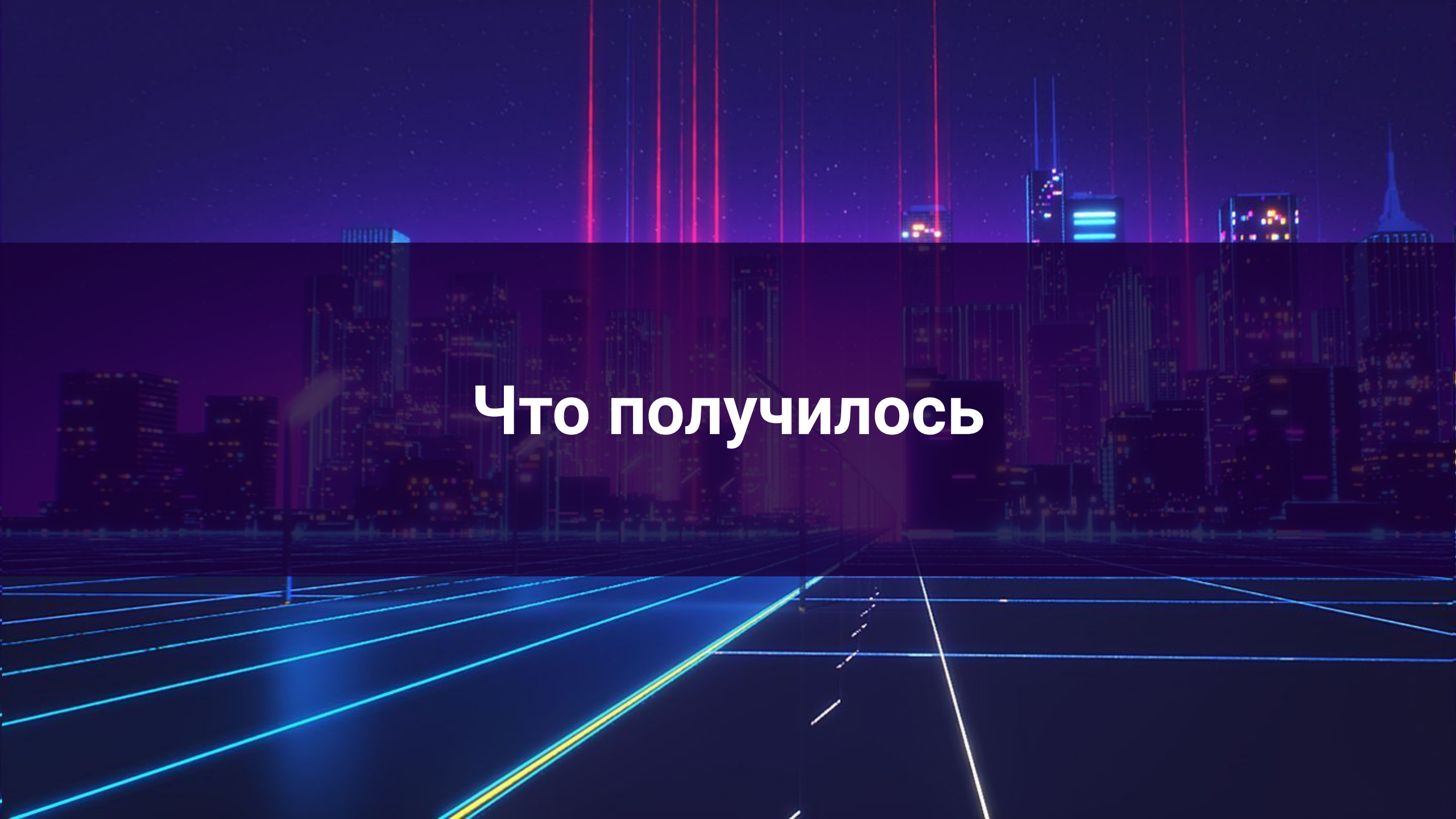


Мы можем указать из какого проекта, какой ветки или тэга необходимо взять ci-файл и использовать его как описание пайплайнов нашего проекта!

Финальная трансформация

Это же всё меняет!

- В проектах больше нет ничего лишнего – только файлы разработчиков
- Получили точку истины для всего “инфраструктурного кода”
- Можем править один пайплайн вместо тысячи
- Никого не аффектим напрасно благодаря версионированию!


Что получилось

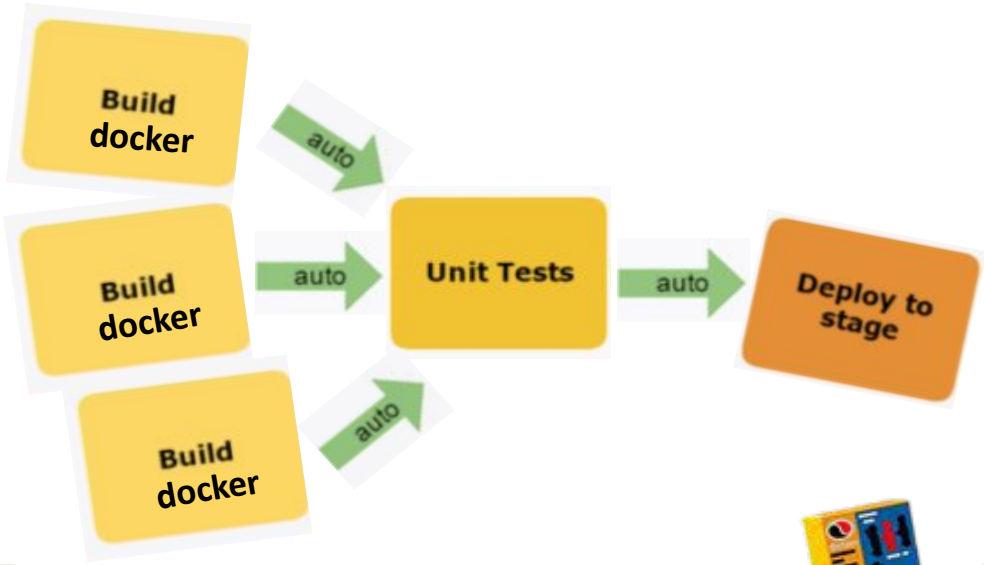
Что получилось

Работа девопсов

- DevOps-специалисты разрабатывают собственные продукты
- Проектные команды становятся “заказчиками” и потребителями продукта
- На весь наш код применяются правила работы с кодом – код ревью, версионирование, тестирование
- DevOps-специалисты подключают проекты к системе
- DevOps-специалисты выступают как разработчики решения, собирая требуемое решение под потребности проекта – составляя cicd-флоу из атомарных готовых сущностей

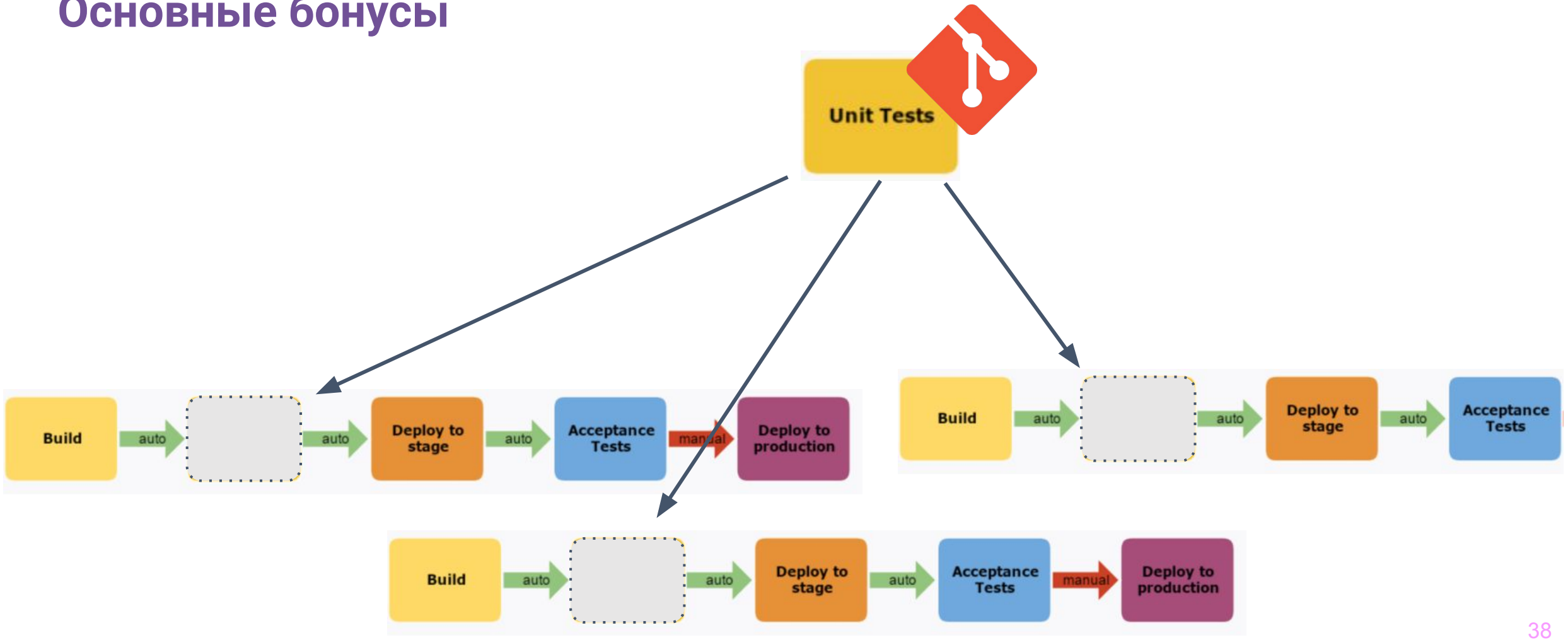
Что получилось

Как это выглядит



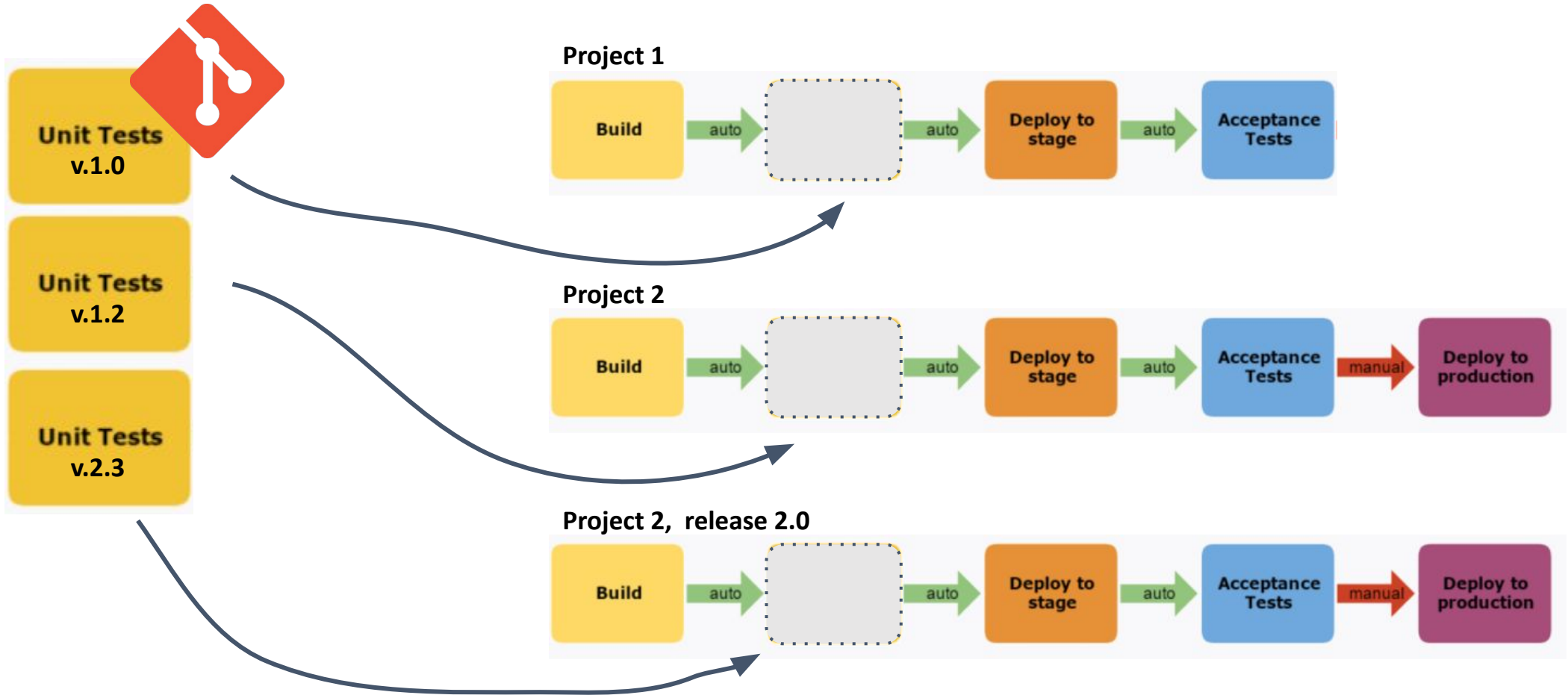
Что получилось

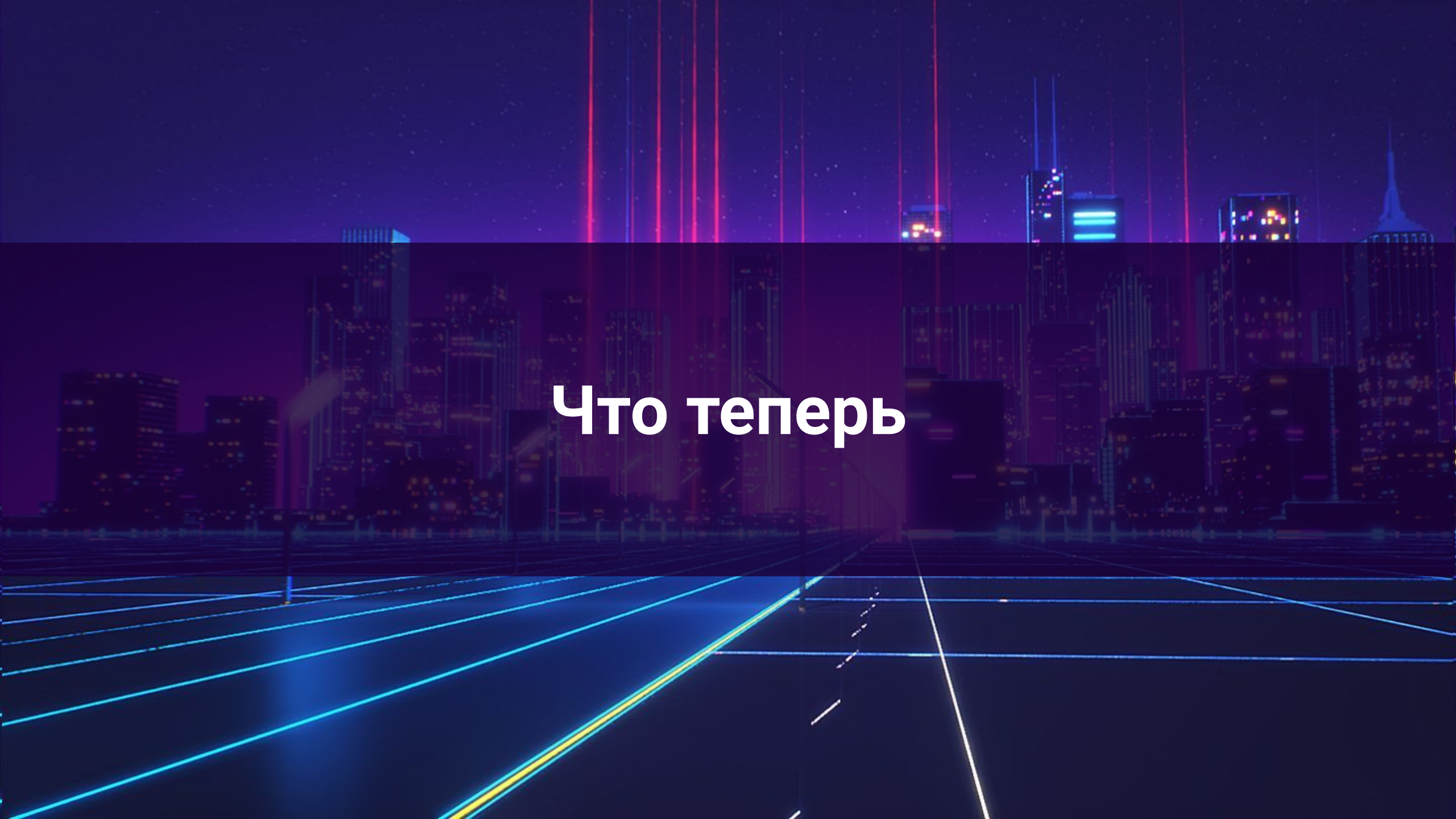
Основные бонусы



Что получилось

Гарантия результата




Что теперь

Что теперь

Новая работа девопсов

- Команда Solution Producers/Providers
- Разработка концепции модульных пайплайнов
- Применение подхода LegoOps (это пока рабочее название)

Что теперь

Погоди, ты же гендир, а не девопс

- Разработка реестра готовых блоков задач
- Быстрое проектирование решения для заказчика
- Единая экосистема решений
- Девопсов можно быстро менять между компаниями
- Компанию можно быстро подключить к налаженной системе и специалисту

Выводы

1. Экономия времени благодаря переиспользованию кода
2. Получение ожидаемого результата от деплоя
3. Легкость чтения кода пайплайна и быстрота внедрения
4. Готовые модули пайплайна as Service

Выводы

> Дайте мне шесть часов, чтобы срубить дерево, и я потрачу первые четыре на заточку топора

-- Авраам Линкольн

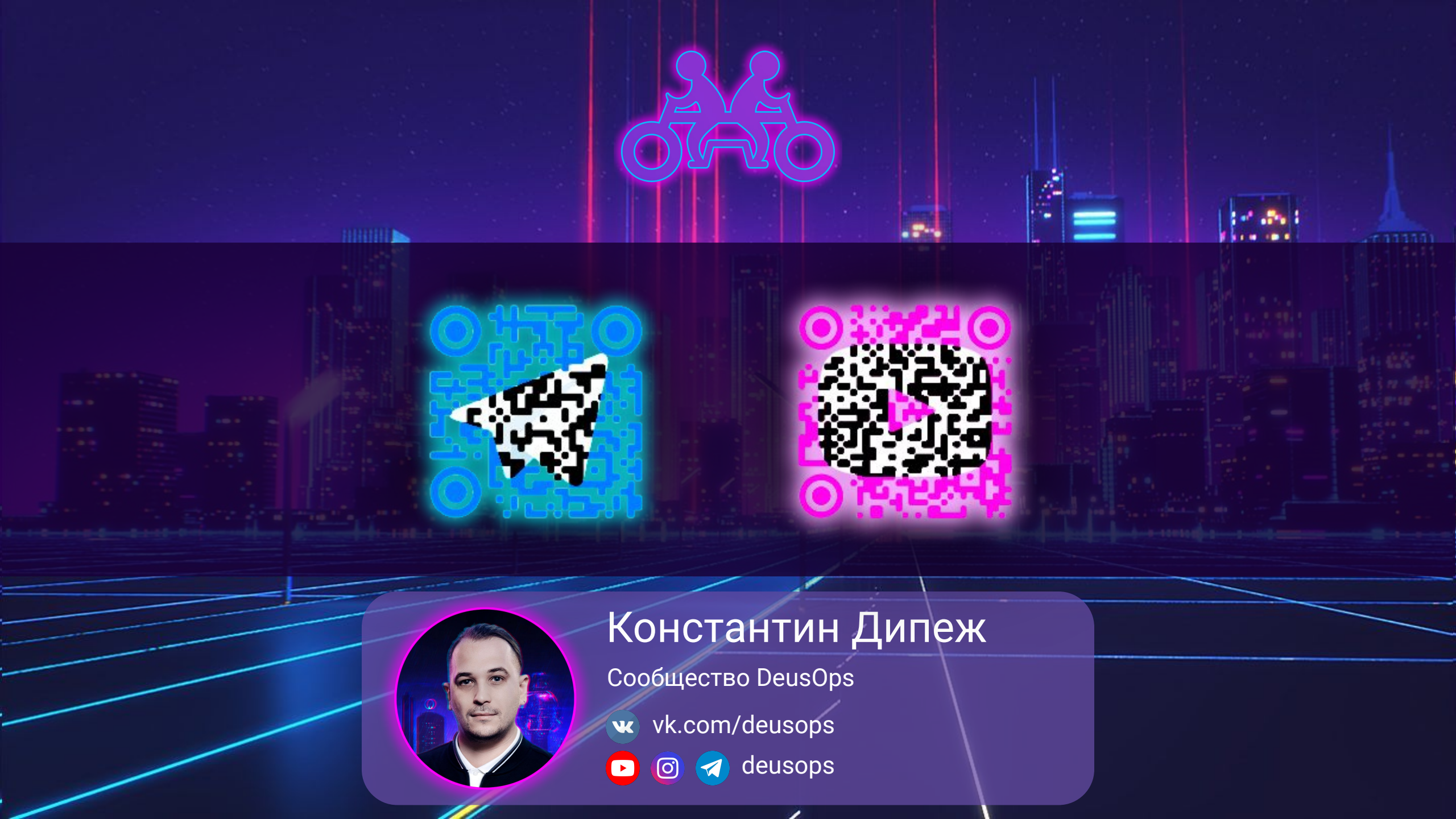
Спасибо за внимание!

Константин Дипеж

Сообщество DeusOps

vk.com/deusops

deusops



Константин Дипеж

Сообщество DeusOps

vk.com/deusops

deusops