

От Bash к Terraform — эволюция централизованного управления GitLab-проектами

Артём Семёнов

2ГИС

Спойлер: что вы узнаете

- Pipeline: 60 минут → 4–5 минут
- ~1100 проектов GitLab, 46 групп, одна команда
- Bash + 9 000 строк YAML → Terraform + Terragrunt
- Как не заDDoS'ить собственный GitLab
- Как выбирали, на чём споткнулись и что забрать себе



2ГИС

>700k RPS

81 млн MAU

СУБД > 35 Тб

40000+ POD

Кто я?

Учёный, физик-мечтатель



Кто я?

Учёный, физик-мечтатель

Системный администратор

TeamLead DevOps



Кто я?

Учёный, физик-мечтатель

Системный администратор

TeamLead DevOps

DevOps инженер в 2ГИС



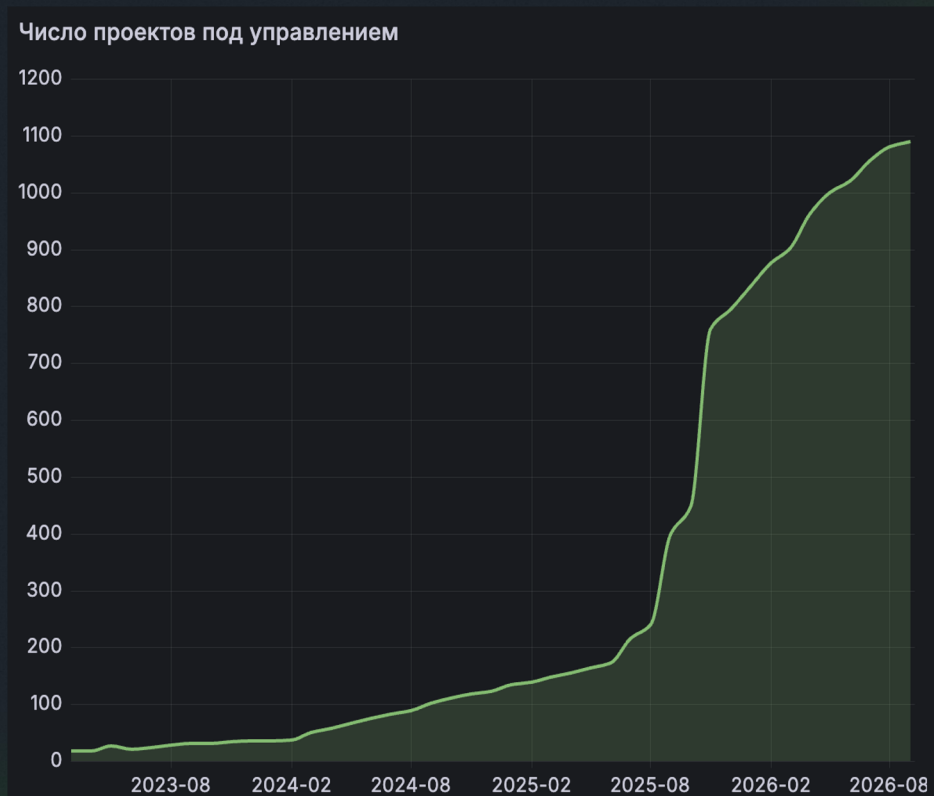
Про что расскажу

- История вопроса и предпосылки изменений
- Муки выбора нового инструмента
- Что получили
- Планы на будущее

С чего началось

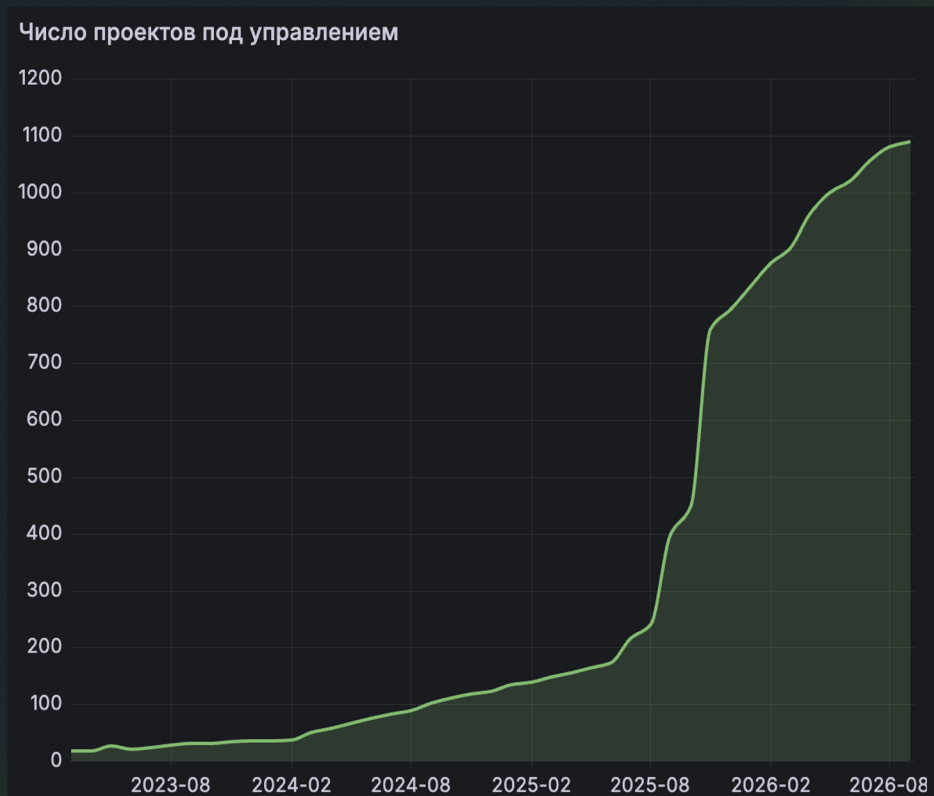
- **Март 2023—17 проектов**
Настройка интеграций с ММ и шаблон для MR с списком ревьюверов.

- **Сентябрь 2026—~1100 проектов**



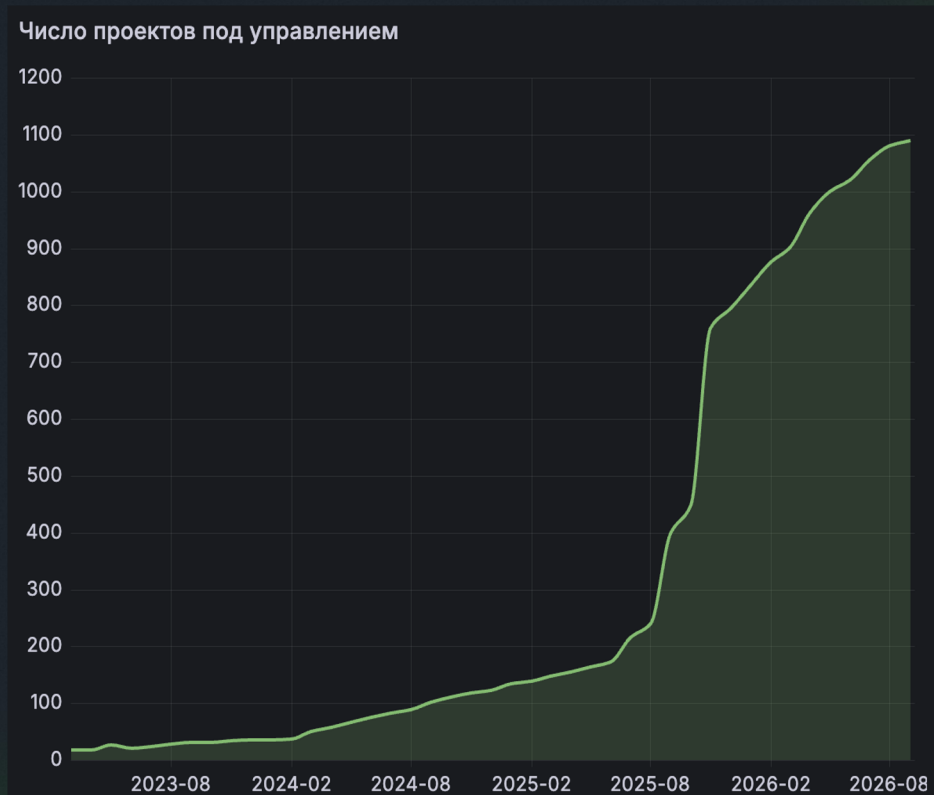
С чего началось

- **Март 2023—17 проектов**
Настройка интеграций с ММ и шаблон для MR с списком ревьюверов.
- **Август 2023—28 проектов**
Создание ролей в vault-gitlab-plugin.
- **Сентябрь 2026—~1100 проектов**



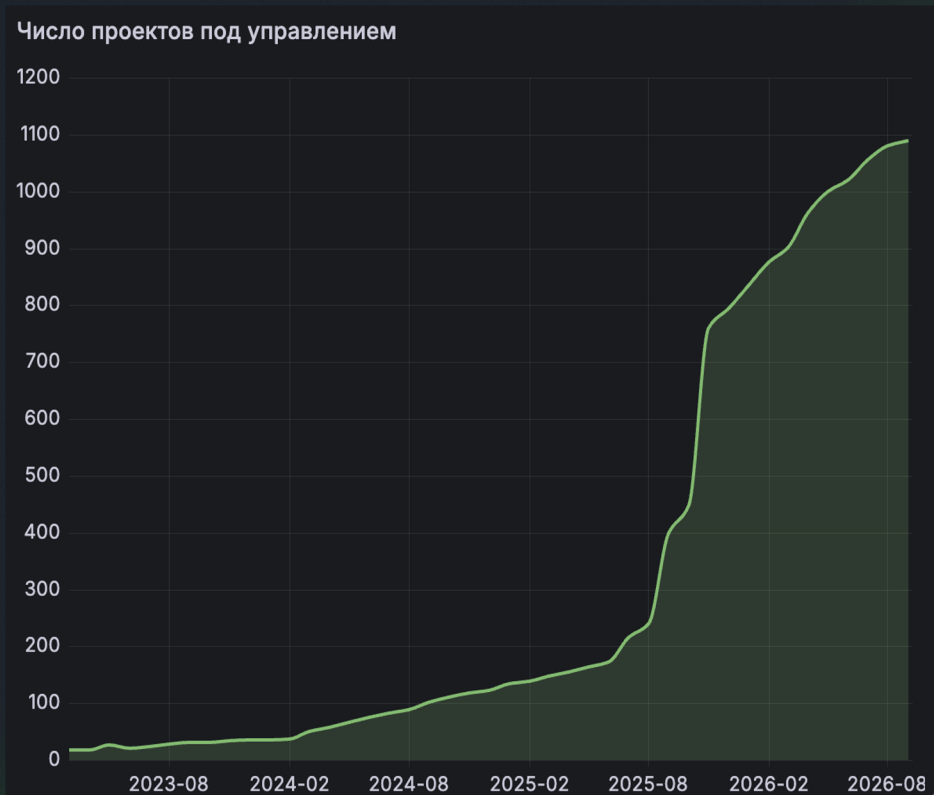
С чего началось

- **Март 2023—17 проектов**
Настройка интеграций с ММ и шаблон для MR с списком ревьюверов.
- **Август 2023—28 проектов**
Создание ролей в vault-gitlab-plugin.
- **Март 2024—40 проектов**
Protected tags/branch, генерация политик и ролей для Vault, миграция списка проектов на yaml
- **Сентябрь 2026—~1100 проектов**



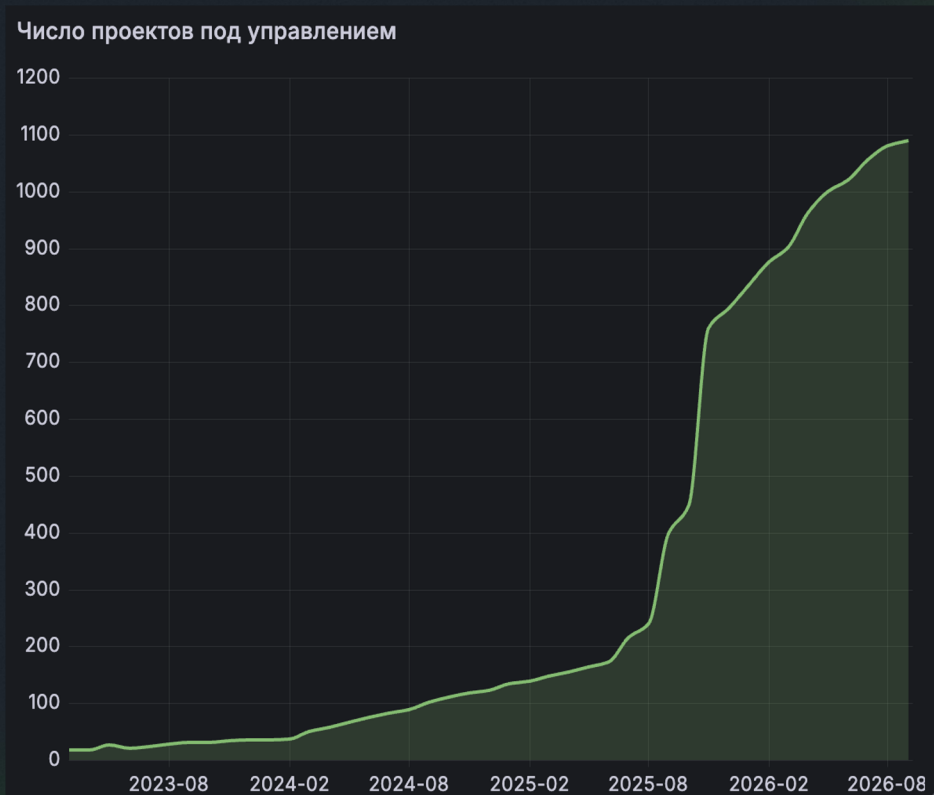
С чего началось

- **Март 2023—17 проектов**
Настройка интеграций с ММ и шаблон для MR с списком ревьюверов.
- **Август 2023—28 проектов**
Создание ролей в vault-gitlab-plugin.
- **Март 2024—40 проектов**
Protected tags/branch, генерация политик и ролей для Vault, миграция списка проектов на yaml
- **Июль 2024—8 групп 92 проекта**
дробление yaml конфига по группам
- **Сентябрь 2026— ~1100 проектов**



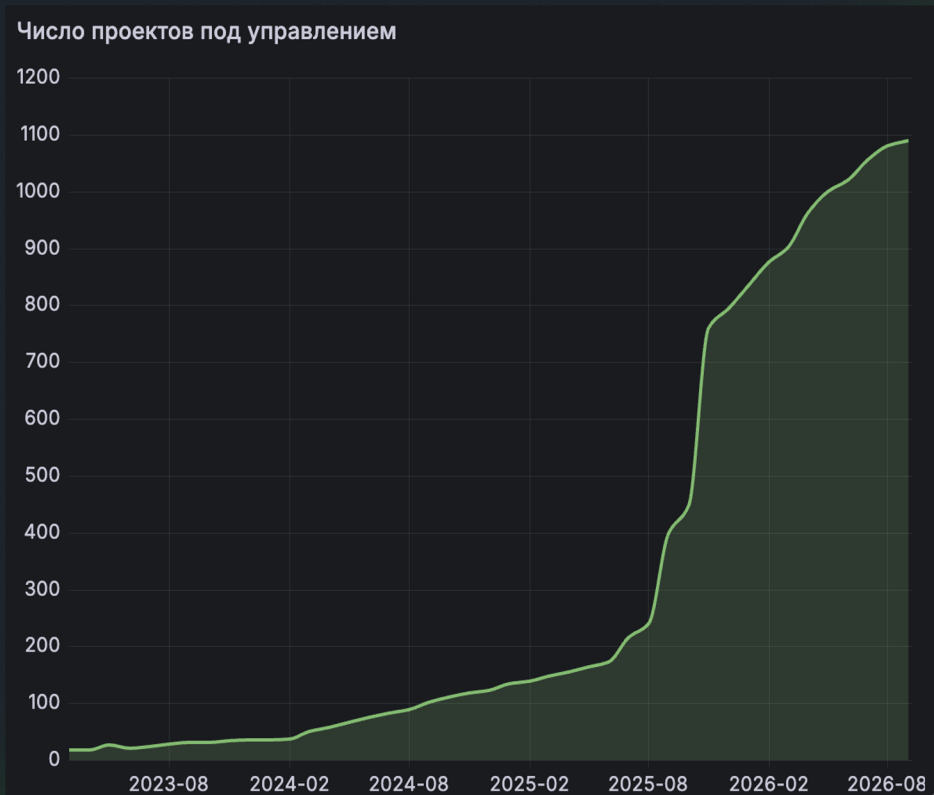
С чего началось

- **Март 2023—17 проектов**
Настройка интеграций с ММ и шаблон для MR с списком ревьюеров.
- **Август 2023—28 проектов**
Создание ролей в vault-gitlab-plugin.
- **Март 2024—40 проектов**
Protected tags/branch, генерация политик и ролей для Vault, миграция списка проектов на yaml
- **Июль 2024—8 групп 92 проекта**
дробление yaml конфига по группам
- **Сентябрь 2025—**
конфигурирование Appscreeener
- **Сентябрь 2026— ~1100 проектов**



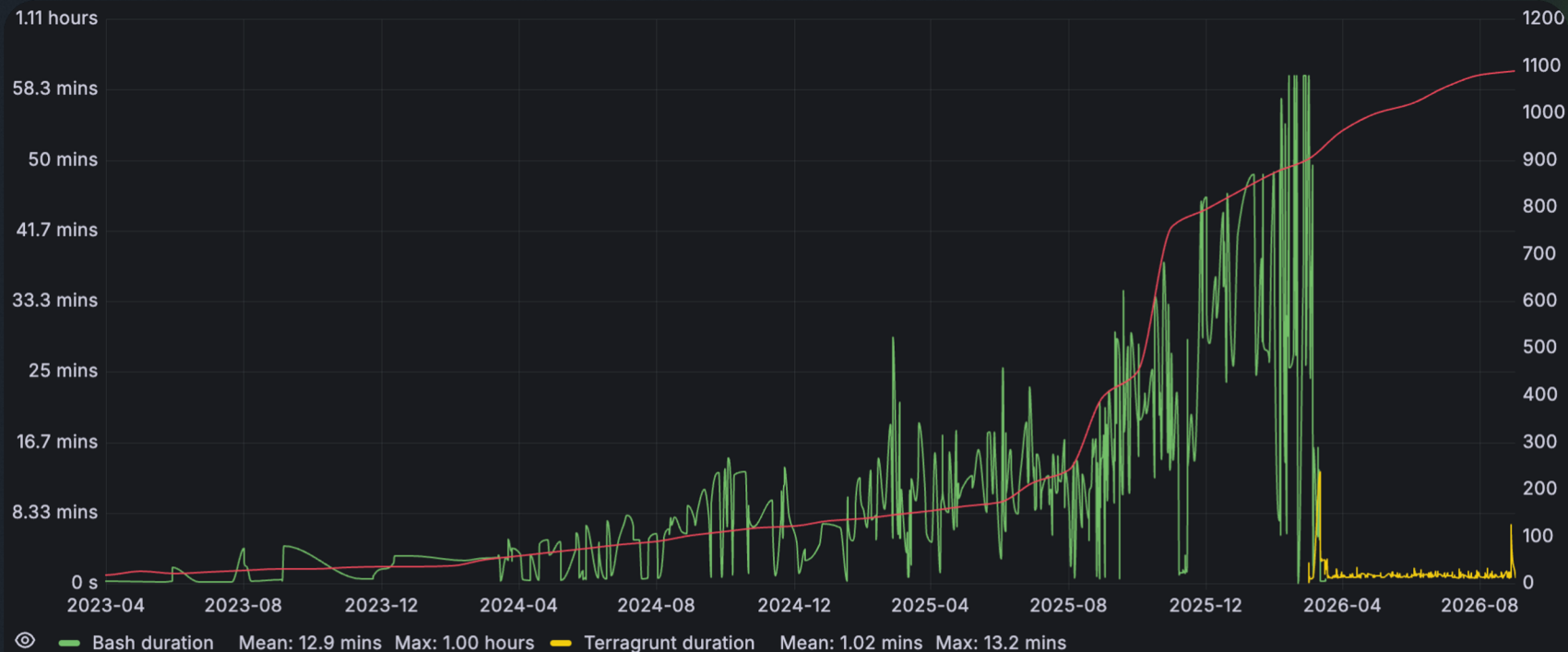
С чего началось

- **Март 2023—17 проектов**
Настройка интеграций с ММ и шаблон для MR с списком ревьюверов.
- **Август 2023—28 проектов**
Создание ролей в vault-gitlab-plugin.
- **Март 2024—40 проектов**
Protected tags/branch, генерация политик и ролей для Vault, миграция списка проектов на yaml
- **Июль 2024—8 групп 92 проекта**
дробление yaml конфига по группам
- **Сентябрь 2025—**
конфигурирование Appscreeener
- **Март 2026— ~850 проектов**
миграция на terraform
- **Сентябрь 2026— ~1100 проектов**



Рост числа проектов и времени

Применения конфигурации



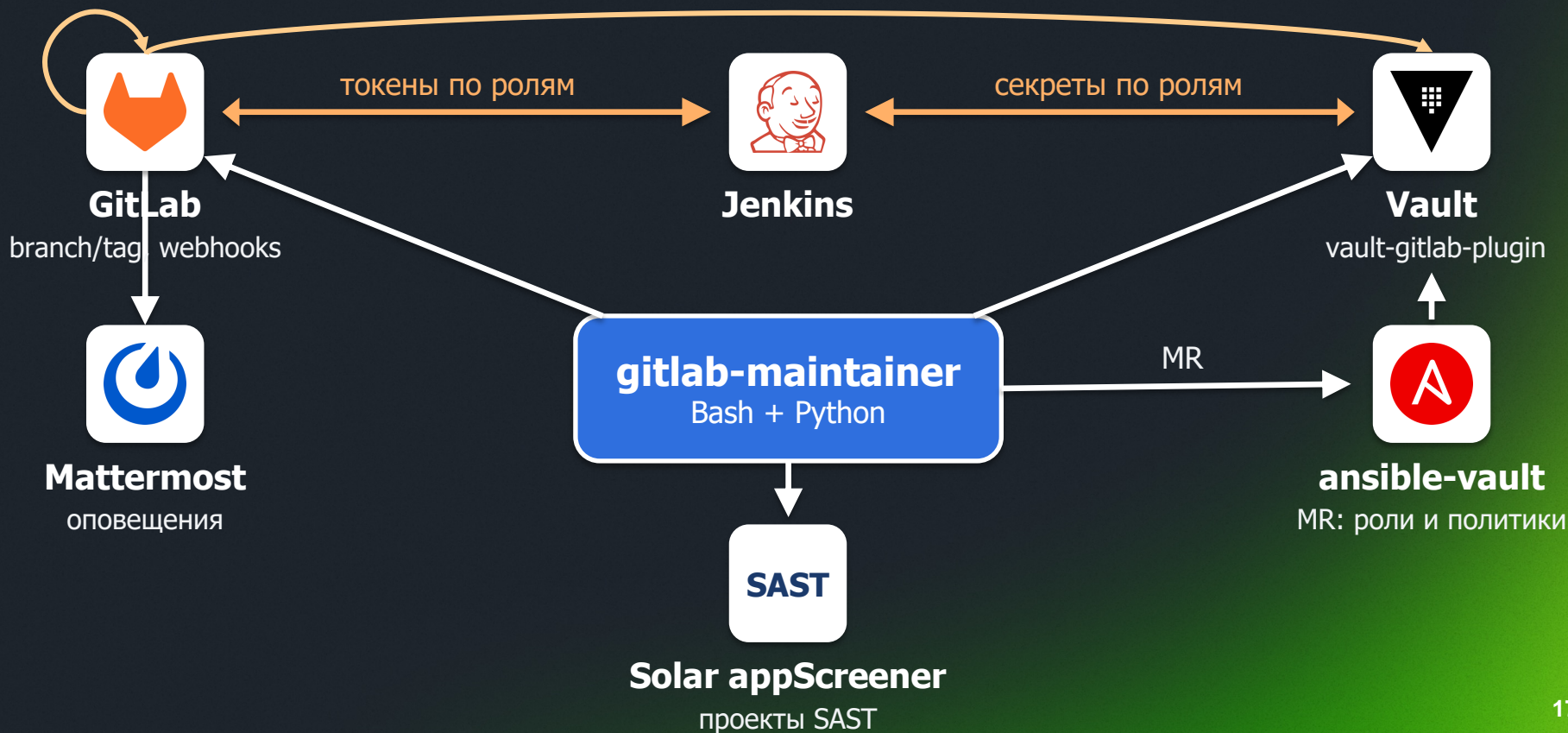
Цифры

- Проекты 17 -> ~1100
- Devops 2 -> 14 человека
- Pipeline time 2 -> больше 60 мин
- Конфиги 30 -> ~27 000 строк

YAML-монстр: 24 500 строк

```
world:ac:build: &world-ac-build
  configure:
    mm: false
    reviewers:
      - world_wac_dev
      - world_wac_qa
    vault: true
  team: world
  kv prefix: world/data/ci
  gitlab: {}
  kv:
    _self: [read]
    appscreeener/sast-token: [read]
world:all:build: &world-all-build
  configure:
    mm: false
... 54 файла, 875 проектов
9 093 строки с якорями = 24 488 раскрытых
```

Что делает gitlab-maintainer



Проблемы

- Долгое время выполнения всего Pipeline > 60 минут
- Зоопарк скриптов bash + Python
- Каждый раз все настройки всех проектов перенакатываются
- Большие и сложные конфиги

Цена одной правки: 44–85 минут

- Команда 7–8 человек, дежурный следит за MR
- Увидеть MR, переключиться, ревью, merge: 2–15 мин
- Pipeline после merge: 40–55 мин
- Merge политик Vault + ревью в соседней команде: 2–15 мин
- Итого 44–85 мин и две команды на одну правку

Честно: пробовали на YAML

- Якоря и шаблоны YAML, чтобы не копировать
- JSON Schema и yamllint в каждом MR
- Дробление конфига на 54 файла по командам
- Итог: поддержка съедает время
- Каждая правка — перераскат всех проектов

Поиск оптимального решения

- Gitlabform
- Tofu
- собственное cli приложение
- полноценное приложение с front и back

GitlabForm

- Проект на Python
- YAML конфиги
- Поддержка шаблонизации на Jinja2
- Практически полное покрытие API Gitlab



GitlabForm

```
config_version: 3
projects_and_groups:
  "devops/*":
    integrations:
      mattermost:
        active: true
        push_events: true
        push_channel: "navi-devops-code-review"
    tags:
      "*":
        protected: true
        create_access_level: maintainer
```

GitlabForm

Плюсы:

- Минимальные временные затраты на реализацию
- Обработку конфига можно делать по частям (запустить N Job)
- unixway

GitlabForm

Минусы:

- Реализация задержки при обращении к API/rateLimits
- Параллельная обработка
- Дополнительный не лаконичный конфиг

Tofu

- Стандарт IaC
- State, plan, apply
- Провайдеры GitLab и Vault
- --parallelism



Tofu

Плюсы:

- Широко известный инструмент
- State
- Parallelism
- Большое кол-во различных Providers
- Открытый SDK

Tofu

Минусы:

- Реализация задержки при обращении к API или rateLimits — костыльно, только `--parallelism`
- Часть функционала не реализована в terraform или вызывается через provisioner "local-exec" в ресурсах
- Всё ещё неудобно управлять структурой проектов

Собственное cli приложение

Приложение со своим state, формат как у terraform или проще; state храним в GitLab.

Плюсы:

- Пишем на чём угодно Go/Python
- Можно организовать задержки при обращении к API или rateLimits
- Полная свобода в организации логики работы

Собственное cli приложение

Минусы:

- Свой велосипед :)
- Стейт, параллельность, архитектуру — всё писать самим

Полноценное приложение с front и back

Видится как сферический конь в вакууме. Требуется хорошего анализа и много чего ещё.

Концепт:

- Реализация UI для пользователей
- RBAC: доступ только к своим группам (Keycloak или GitLab)
- Хранение данных в базе
- Модульность: сегодня Vault/AppScreener, завтра что-то ещё

Полноценное приложение с front и back

Плюсы:

- Для пользователя всё красиво и удобно
- Не надо ни каких MR и апрувов (кроме *InfraWeb/ansible-vault*)
- все изменения хранятся в базе

Полноценное приложение с front и back

Минусы:

- Время
- Нужен front и back (последний ещё можно силами devops, с фронтом всё сложно)
- в общем, сильно сложно

Terragrunt + Tofu

- Окружения = группы
- Параллелизм на уровне окружений и ресурсов
- Описание конфигураций и дефолтов в HCL

Terragrunt + Tofu

- Окружения = группы
- Параллелизм на уровне окружений и ресурсов
- Описание конфигураций и дефолтов в HCL



Итоговая таблица

Инструмент/ функционал	Тип конфигов	дедубликация/ валидация	хранение состояния	RateLimits	Работа с API Gitlab	appscreeener	создание MR приносим изменения в InfraWeb/ansible- vault	Комментарии
gitlabform	yaml	yaml-lint	нет	нет, можно запускать обработку пачками --start-from START_FROM --start-from-group START_FROM_GROUP	более полная чем у terraform	оставляем Python	оставляем bash	Самый просто вариант, одна задача - одни сервис.
terraform + terragrunt	tvars или json	terraform validate	есть	из коробки только через использование parallelism	gitlab provider	оставляем Python	оставляем bash	Простой вариант, можно оставить текущие конфиги в Yaml и добавить скрипт генерации в json для использования в качестве vars для terraform
terraform + custom provider						пишем два своих Provider на Go		Более элегантное решение, но и затратное, получим два провайдера - для управления проектами в appscreeener и генерация/создание MR в репозиторий InfraWeb/ansible- vault
Awx + ansible	yaml (какой захотим)	множество вариантов	индепотентность	реализуемо	role	role	role	на AWX можем нарисовать UI для всего остального ansible/python
cli приложение (Go/Python)	yaml (какой захотим)	своя реализация	делаем аналогично terraform (храним в Gitlab)	Своя реализация				реализовать, наверное чуть сложнее чем написать провайдеров. Зато имеем все ручки.
полноценное приложение с front и back			Хранить можно в любом месте (база/S3/etc)	Своя реализация				Сложность в реализации Front и Back - нет необходимых компетенций

Terragrunt + Terraform

Реализация

Terragrunt + Terraform Реализация

- Понятная структура с проектами

Terragrunt + Terraform Реализация

- Понятная структура с проектами
- Возможность задавать дефолты

Terragrunt + Terraform Реализация

- Понятная структура с проектами
- Возможность задавать дефолты
- Получать секреты из Vault

Terragrunt + Terraform Реализация

- Понятная структура с проектами
- Возможность задавать дефолты
- Получать секреты из Vault
- Валидация набора параметров для Gitlab

Как поделён репозиторий

terragrunt/

main.hcl

state в GitLab, секреты из Vault

modules/gitlab-maintainer

весь Terraform-код

groups/

46 групп GitLab



groups/devops/

одна группа = один state



gitlab_default.hcl

дефолты группы

gitlab_devops.hcl

проекты команды

gitlab_hr.hcl

проекты подгруппы

terragrunt run --all: группы идут параллельно, внутри группы Terraform ограничен --parallelism

Итоговая структура проекта

```
terragrunt/
├── main.hcl                # state, токены из Vault
├── terragrunt.hcl          # один на все группы (симвлинк)
├── modules/gitlab-maintainer/
│   ├── variables.tf        defaults.tf  locals.tf
│   └── gitlab_props.tf      vault_roles.tf
├── groups/                 # 46 групп
│   └── devops/
│       ├── gitlab_default.hcl    # дефолты группы
│       ├── gitlab_devops.hcl     # проекты
│       └── gitlab_hr.hcl         # подгруппа
```

Структура конфигов для проектов

```
locals {  
  default = read_terragrunt_config("gitlab_default.hcl")  
  
  # все gitlab_*.hcl группы подхватываются сами  
  all_projects = merge([  
    for f in fileset(".", "gitlab_*.hcl") :  
      lookup(read_terragrunt_config(f).inputs,  
            "projects", {})  
  ]...)  
}  
  
inputs = merge(local.default.inputs,  
  { projects = local.all_projects })
```

Иерархия дефолтов



приоритет растёт слева направо

- Новое поле: добавили в `variables.tf` и `defaults.tf` — проекты не трогаем
- Старый конфиг без поля работает на дефолтах
- «Не задано» (`null`) не затирает дефолт; пустую строку можно задать сознательно

merge(): три уровня за один проход

```
# упрощено: показана только секция mm (protect и dino_bot — так же)
resolved_mm = {
  for path, config in var.projects : path => merge(
    local.mm_defaults,                                # 1. модуль
    { for k, v in coalesce(try(var.default_gitlab_props.mm, null), {})
      : k => v if v != null },                          # 2. группа
    { for k, v in coalesce(try(config.gitlab_props.mm, null), {})
      : k => v if v != null },                          # 3. проект
  )
}
```

defaults.tf: дефолты модуля

```
# defaults.tf — последний уровень приоритета
locals {
  protect_defaults = {
    branch_name = "master"
    tag_name    = "*"
    # ...
  }
  mm_defaults = {
    enable           = false
    default_channel = ""
    push_events      = false
    # ... ещё ~20 полей
  }
}
```

Новый проект — одна строка

```
# groups/devops/gitlab_default.hcl
default_gitlab_props = {
  protect = { branch = true, tag = true }
  mm      = { enable = true }
}

# groups/devops/gitlab_starter_kit.hcl
projects = {
  "devops/starter-kit/app-build" = {}
  "devops/starter-kit/redis-ha"  = {
    gitlab_props = { mm = { enable = false } }
  }
}
```

Было: bash, 53 строки на проект

```
# protect-tags.sh (сокращено, всего 53 строки)
PROJECT="$1"; GITLAB_VAULT_TOKEN="$2"
PROJECT_ENC="$(printf '%s' "${PROJECT}" | sed -r 's@/@%2F@g')"

if test -n "${DRY_RUN}"; then
    curl -sLg -X GET -H "PRIVATE-TOKEN: ${TOKEN}" \
        "${CI_API_V4_URL}/projects/${PROJECT_ENC}/protected_tags"
    exit 0
fi

# без --fail: если тег уже защищён, вернётся 422
curl -sLg -X POST -d '{"name": "*"}' \
    -H "PRIVATE-TOKEN: ${GITLAB_VAULT_TOKEN}" \
    "${CI_API_V4_URL}/projects/${PROJECT_ENC}/protected_tags"
```

Стало: tofu, один ресурс

```
# 2. ресурс: Tofu сам решает, что создать, изменить или удалить
resource "gitlab_tag_protection" "protect_tags" {
  for_each      = local.projects_map_protect_tags
  project       = each.value.project
  tag           = each.value.tag_name
  create_access_level = "maintainer"
  provisioner "local-exec" {
    command = "sleep ${var.gitlab_api_request_pause}"
  }
}
```

Vault: откуда берутся секреты



- В репозитории секретов нет: всё берётся из Vault при запуске
- Роль определяет права: ci — токен джобы, prod — токен из Vault
- State хранится в GitLab (http backend), доступ по токену джобы
- Тот же Vault — объект управления: роли для токенов проектов

main.hcl: один код — два режима

```
# секреты берём только из Vault
locals {
  vault_env = split("_", local.vault_role)[0] # ci | prod

  gitlab_token = local.vault_env == "ci" ? get_env("CI_JOB_TOKEN")
    : run_cmd("vault", "kv", "get", "-field=key", ".../uk.devops.ci")

  vault_token = local.vault_env == "ci" ? get_env("VAULT_TOKEN")
    : run_cmd("vault", "write", "-field=token", "auth/jwt_v2/login", "...")
}
```

Vault: роли токенов для проектов

```
resource "vault_generic_endpoint" "self_roles_ro" {  
  for_each = local.projects_map_vault  
  path      = "gitlab/roles/ro_${each.value.roles}"  
  data_json = jsonencode({  
    path          = each.value.name          # путь проекта  
    access_level  = "reporter"  
    scopes        = ["read_api", "read_repository"]  
    ttl           = var.gitlab_token_ttl     # 48 часов  
  })  
}  
# self_roles_rw: maintainer, api + write_repository
```

Validate

```
mm_validation = {  
  for key, mm in local.projects_map_mattermost : key => {  
    has_any_event = anytrue([  
      mm.issues_events, mm.push_events,  
      mm.pipeline_events, # ... еще 6  
    ])  
    has_any_channel = anytrue([  
      mm.default_channel != "",  
      mm.push_channel != "", # ... еще 8  
    ])  
  }  
}
```

Validate + delay

```
resource "gitlab_project_integration_mattermost" "mattermost" {
  for_each = local.projects_map_mattermost
  project  = each.value.project
  lifecycle {
    precondition {
      condition      = local.mm_validation[each.key].has_any_event
      error_message = "mm.enable=true, но нет ни одного события"
    }
    precondition {
      condition      = local.mm_validation[each.key].has_any_channel
      error_message = "mm.enable=true, но не указан ни один канал"
    }
  }
  provisioner "local-exec" {
    command = "sleep ${var.gitlab_api_request_pause}"
  }
}
```

Миграция проектов внутри Gitlab

Часть проектов перенесли в другую корневую группу



Things to be aware of before transferring:

- Be careful. Changing the project's namespace can have unintended side effects.
- You can only transfer the project to namespaces you manage.
- You will need to update your local repositories to point to the new location.
- Project visibility level will be changed to match namespace rules when transferring to a group.

Select destination namespace

DevOps / Chaos

Transfer project

Проекта уже нет, а State есть

```
# gitlab_branch_protection.protect_branch["world_audit_benchmark-web"]  
#   will be destroyed  
#   (key is not in for_each map)
```

```
gitlab_branch_protection.protect_branch[...]: Destroying...  
Error: Failed to delete gitlab branch protection  
405 {message: Non GET methods are not allowed for moved projects}
```

Очистка state перед apply

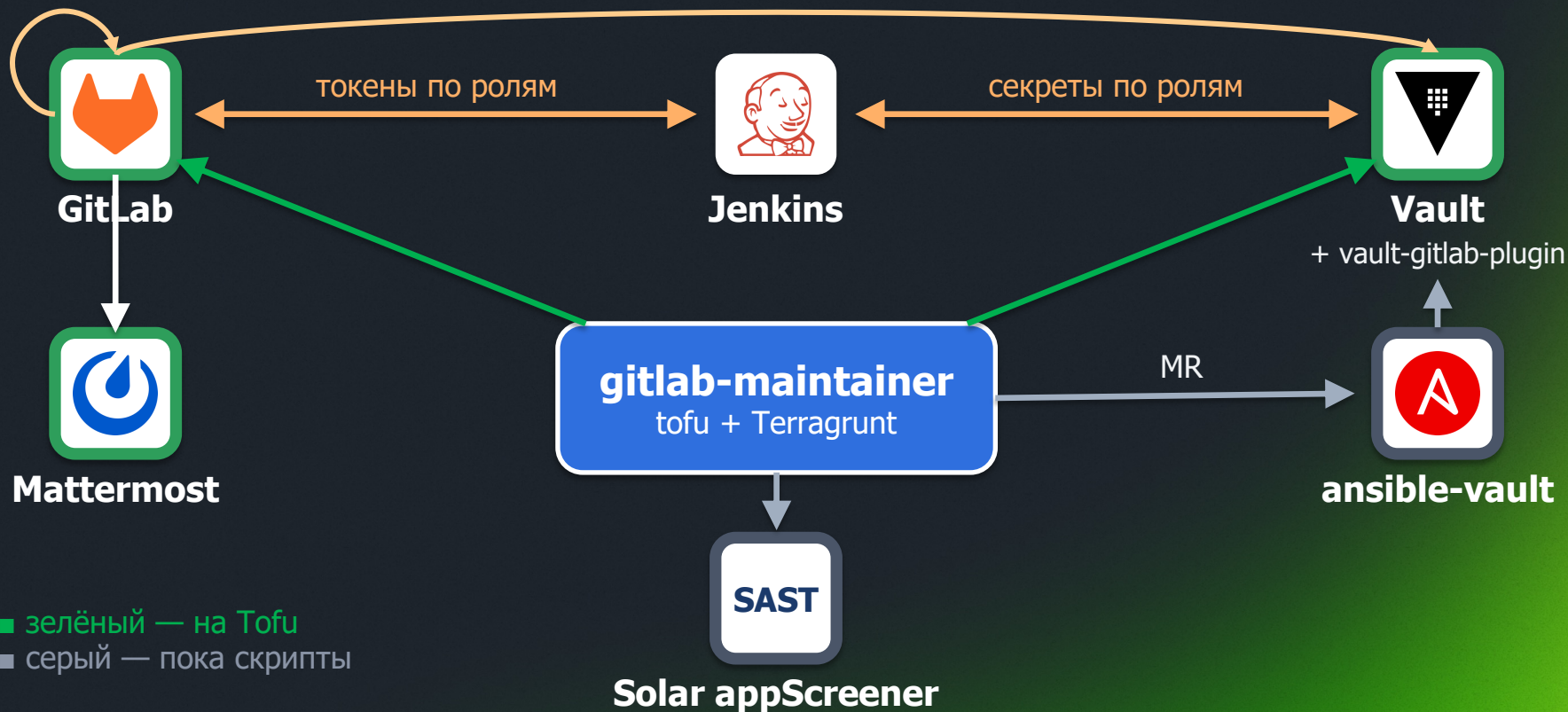
- Plan – получаем Gitlab ресурсы на удаление
- Перед apply проверяем наличие репозитория в Gitlab
- Чистим State

```
==> Processing cleanup manifest
Project 'artem.semenov/test-tf' is not accessible
Removed gitlab_branch_protection.protect_branch["artem.semenov_test-tf"]
Removed gitlab_tag_protection.protect_tags["artem.semenov_test-tf"]
==> Done: removed 2 resource(s) from state before apply
```

Было → Стало

	Было	Стало
Pipeline	> 60 минут	4–5 минут
Проектов	875 (фев 2026)	~1100, 46 групп
Запросы к GitLab	каждый проект, каждый раз	только diff по state
Код	6 bash-скриптов, ~460 строк	модуль HCL + 3 скрипта
Цикл правки	44–85 мин, 2 команды	8–35 мин (с OpenBao: 6–20)
Новый проект	правка YAML и прогон всех	1 строка HCL + plan в MR
Эффект	—	≈55 мин с каждого прогона

Что уже на Terraform



Pipeline

latest branch 16 jobs 4 minutes 47 seconds, queued for 1 seconds

Pipeline Jobs 16 Tests 0

Group jobs by Stage Job dependencies

lint

- ✓ lint:projects ↺
- ✓ lint:terraform ↺
- ✓ vault-policies-jenkins:convert-to-json ↺
- ✓ vault-policies:convert-to-json ↺

dryrun

- ✓ dryrun:appscreeener ↺
- ✓ vault-policies-jenkins:generate ↺
- ✓ vault-policies:generate ↺

self

- ✓ self:appscreeener ↺
- ✓ vault-policies-jenkins:convert-to-yaml ↺
- ✓ vault-policies:convert-to-yaml ↺
- ✓ vault-policies:create-mr:dryrun ↺

all

- ✓ all:appscreeener ↺
- ✓ all:terraform:apply ↺
- ✓ all:terraform:plan ↺
- ✓ prepare:auth-token ↺
- ✓ vault-policies:create-mr ▶

ADR: побочный эффект

- Выбор инструмента оформили как: контекст, 5 вариантов, критерии, решение
- Решение и причины не теряются: к ним возвращаемся, когда меняются условия
- ADR стал стандартом для архитектурных решений в команде

Вывод для слушателей

- State + plan вместо скриптов по API
- Дефолты и merge(): меньше кода, меньше ошибок
- Проверки в коде: ошибка на plan, а не в проде
- Не завязывайтесь в Tofu на пути имена проектов
- ADR: фиксируйте выбор инструмента письменно

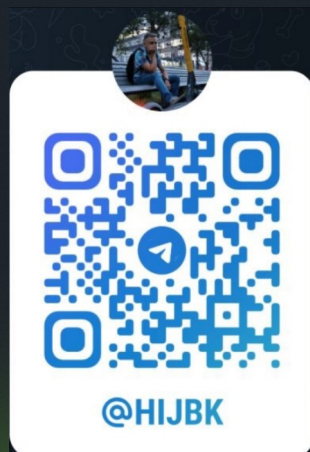
Развитие проекта

- Написать свой terraform provider для Solar AppScreaner
- Мигрировать с Vault на OpenBao
- Переиспользования код или подход в разрабатываемом проекте IDP
- Сделать его атомарным и отдать в команды.
Чтоб у каждой команды был свой мини проект.
- Оставить как есть =)

ОpenБао: что изменится (оценка)

- Роли и политики Vault — в HCL, с дефолтами
- YAML сегодня ~27 тыс. → ~6 тыс. строк (−75...80%)
- Уходят Python (538 строк), MR в ansible-vault, 4 CI-джоба
- Код: −850 +~150 строк, нетто $\approx$ −20%
- Цикл правки: 8–35 → 6–20 мин

Спасибо!



Артём

Как надо было: ключ — id проекта

Сейчас: ключ = путь

world_frankie → world/frankie

перенос = destroy + create

скрипт и чистка state

Надо было: ключ = id

1234 → проект с id 1234

перенос = пустой plan

путь в YAML — просто подпись

Id проекта в GitLab не меняется при переносе и переименовании

В YAML рядом с путём храним id (скриптом заполнили один раз)

data "gitlab_project" ищет проект по id; for_each-ключ ресурсов — id

Порядок не важен: перенос и MR — в любой очерёдности

Как надо было: варинат через data

```
# YAML:  world/frankie: { id: 1234, gitlab_props: { ... } }

data "gitlab_project" "p" {
  for_each = var.projects
  id       = each.value.id           # ищем по id, а не по пути
}

locals {
  projects_map_protect_branch = {
    for path, config in local.projects_with_protect_branch :
    data.gitlab_project.p[path].id => {   # ключ ресурса = id
      project = data.gitlab_project.p[path].id
      branch  = local.resolved_protect[path].branch
    }
  }
}
```