



# Со своим самоваром

как переносили распределенный монолит  
из одной СІ-системы в другую

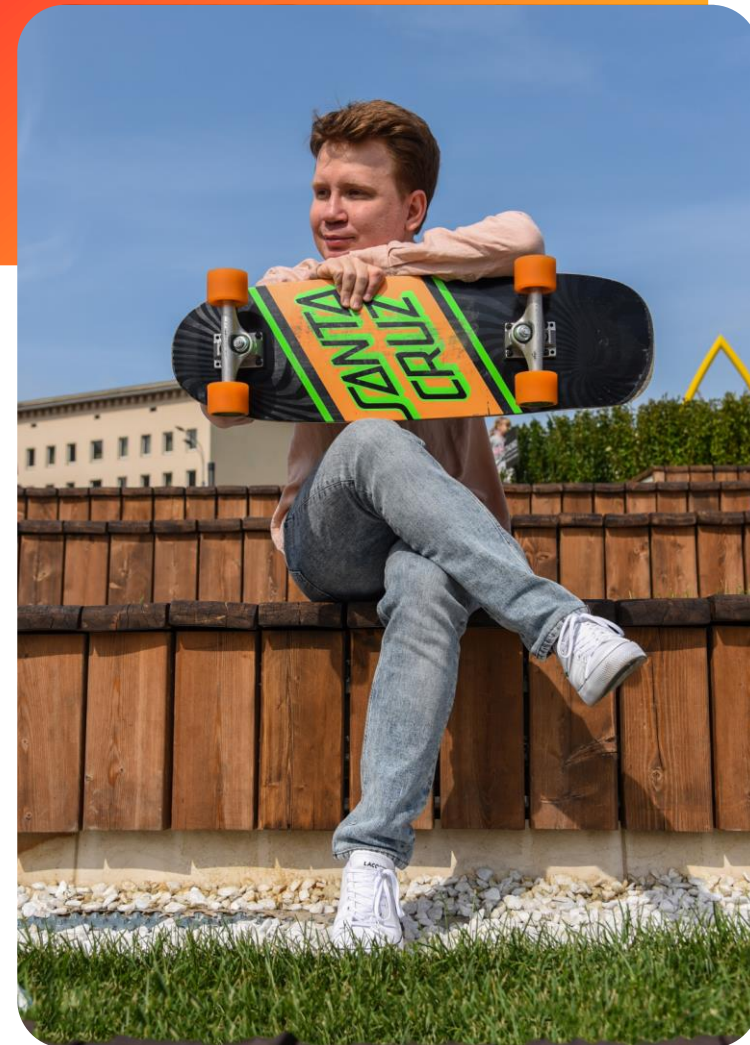
# А кто я?

**В ПСБ работаю 4 года и за это время успел:**

- Поработать с 7-ю проектами
- Принять непосредственное участие в запуске 2 проектов с 0 до ОП
- Перевести проект из одной системы непрерывной интеграции в другую
- Мигрировать проект из одной системы контроля версий в другую
- Поучаствовать в выстраивании автоматизации проочки пакетов в систему хранения артефактов и перевода проектов на внутренние репозитории
- Принять участие в запуске школы DevOps
- Застать проекты на Java 1.5
- Получить ментора и обучать стажеров

## Пётр Галонза

управляющий эксперт отдела внедрения системных сервисов





**Томар** - для разработки, развертывания и запуска корпоративных приложений

**Велби** - для разработки, развертывания и запуска корпоративных приложений – контейнер сервлетов, используется как веб-сервер

**Док** (платформа) - Enterprise Content Management, управление документами и процессами их обработки.

**Джаванна** – Основной язык программирования в проекте, на данный момент используется версии 1.7, 1.8 и 11

**Сприн 1,2** – Фреймворк упрощающий разработку веб-приложений

**Спринго** – проект упрощающий настройку проектов и разработку на Spring

**Джимикс** – Высокоуровневая платформа для разработки приложений и сервисов. В прошлом Cuba Platform

**Орктур** – база данных версии 12

**Мавена** – сборщик проектов, используется для основной части

**Грэд** – сборщик проектов, используется для микросервисов



**Maven™**

**ORACLE®**  
DATABASE



 **Jmix**  
by Haulmont



**CUBA.platform**

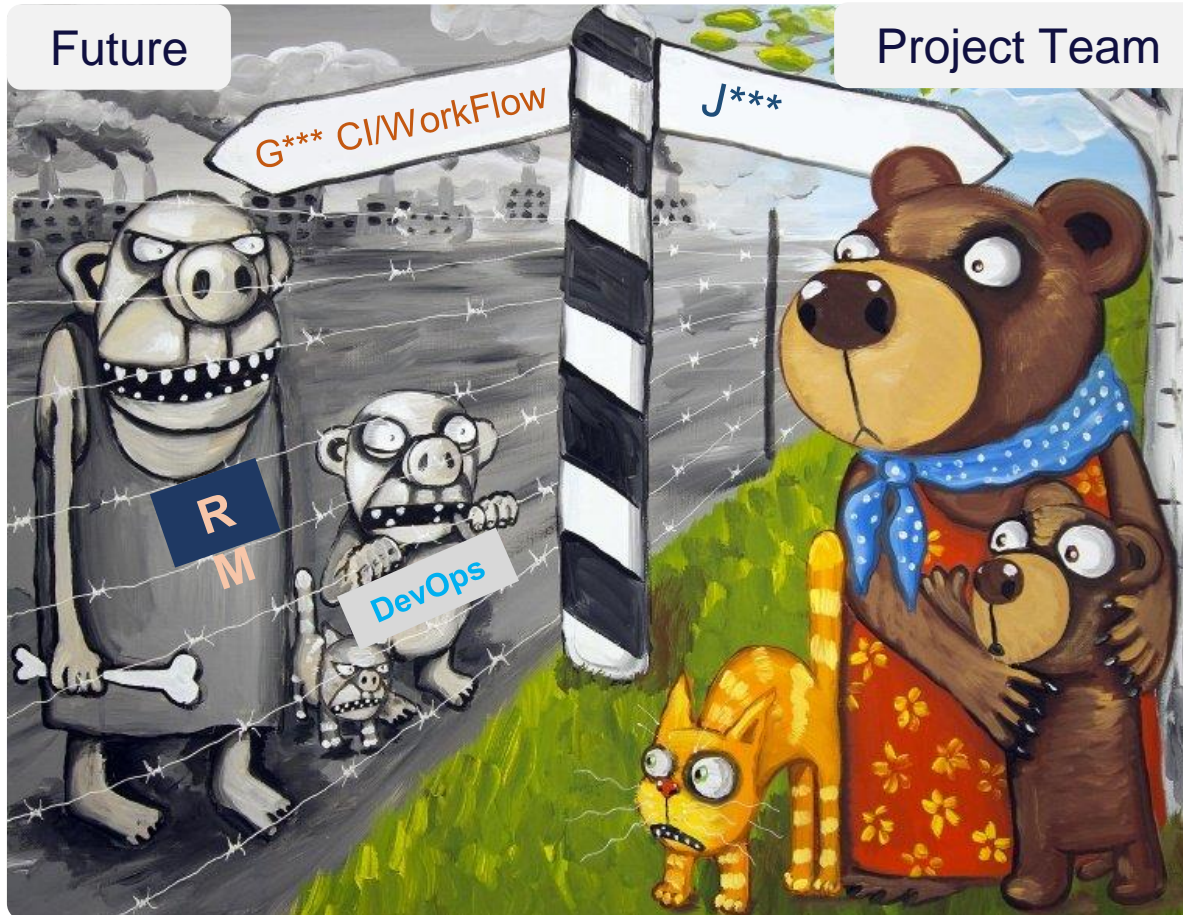
 **spring®**

  
Apache Tomcat

**BELLSOFT**

# Первое впечатление о планах RM и DevOps

- ◇ Изменения Workflow
  - ◇ Культ Гилеандр CI
- ◇ Уберём Мавену и Антар
  - ◇ Гилеандр Flow
- ◇ Контроль и проверка релизов
  - ◇ Отберем права у разработчиков



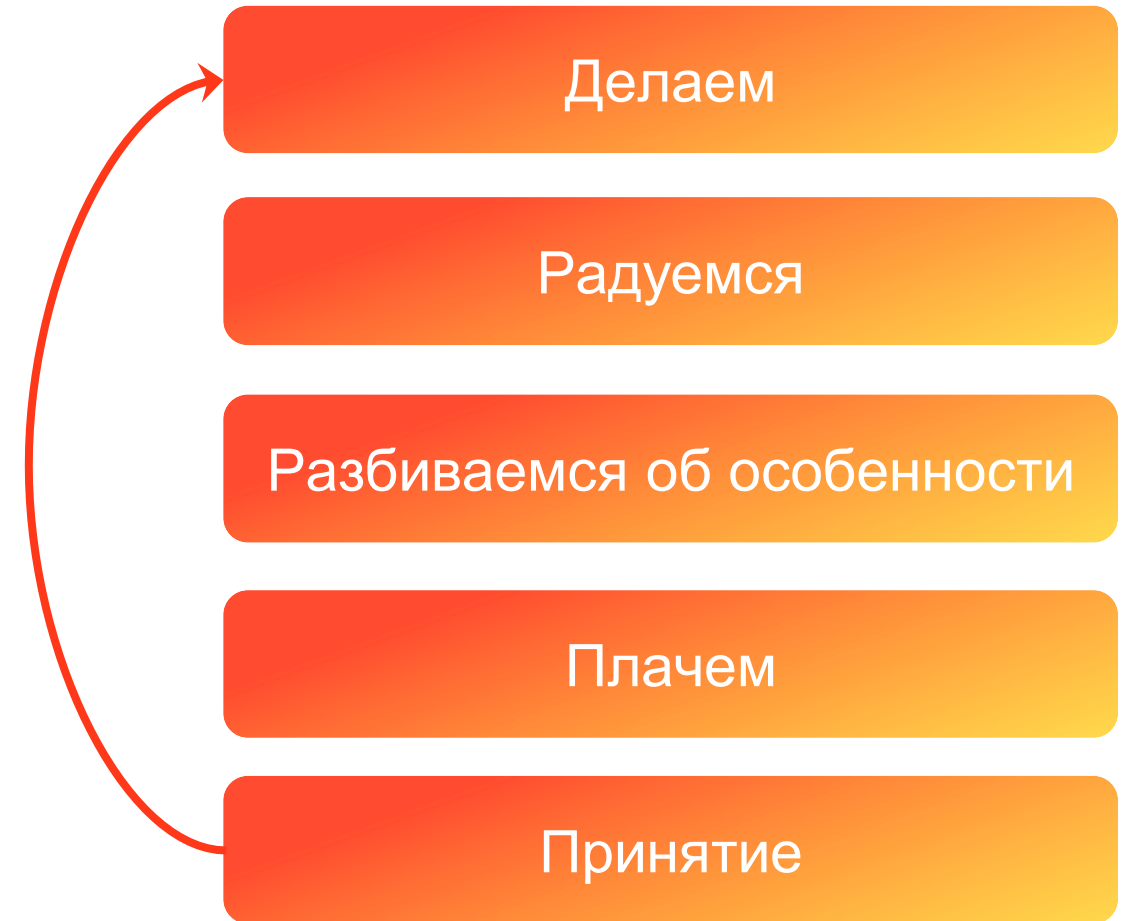
- ◇ Устоявшиеся процессы
- ◇ Отточенные годами практики, подходы и инструменты
- ◇ Процессы проходят внутри команды
- ◇ Новое требует времени, усилий на внедрение и не всегда это оправдано
  - ◇ Выбор инструмента исходя из потребностей

- Уход от Дженкс, Гилеандр CI Pipeline
- Что под капотом? Скрипты и не только

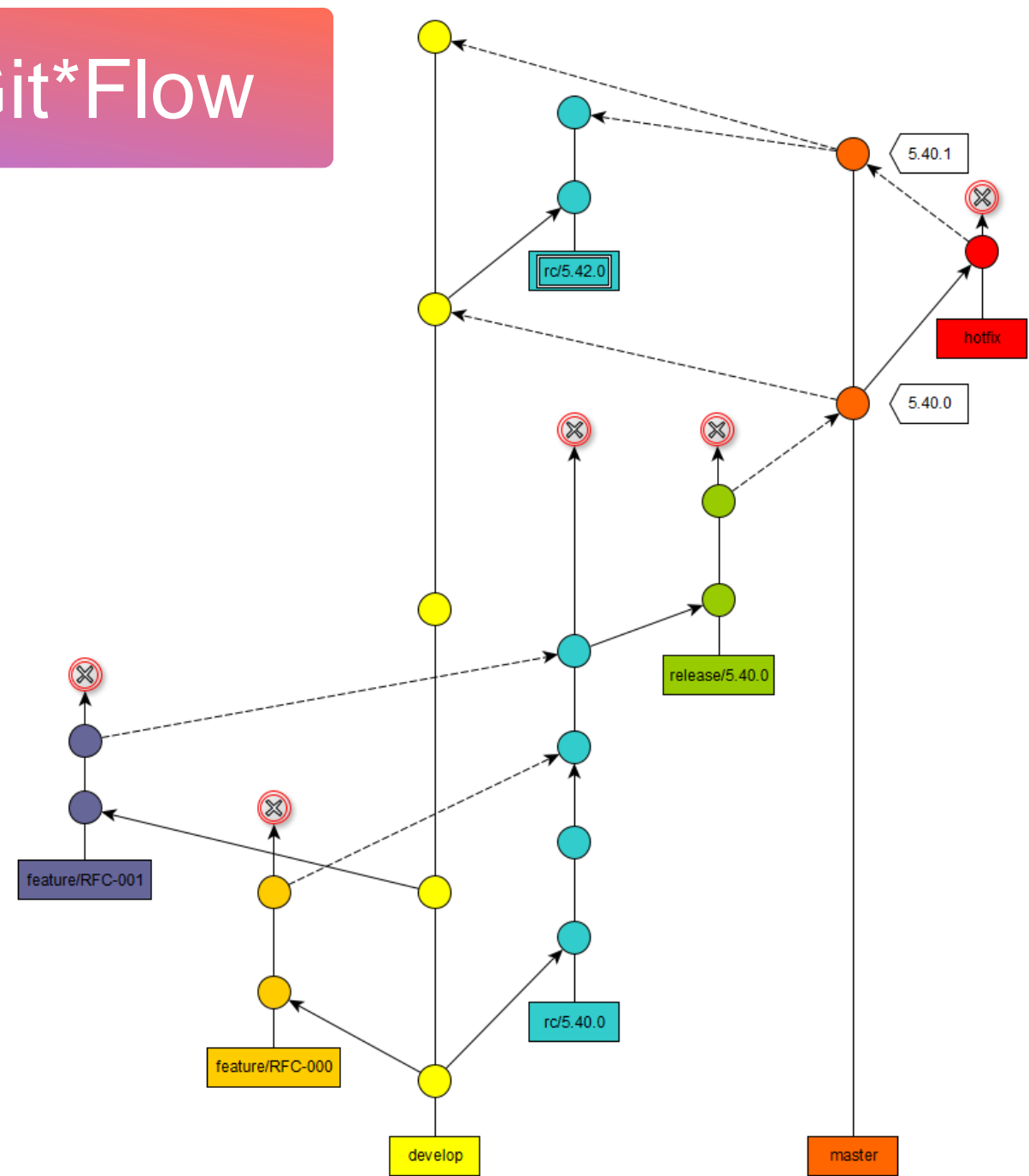
# Собираем, анализируем и учитываем особенности, узкие места проекта

## Особенности установки релиза:

- Порядок установки и перезапуска
- Время установки
- Зависимости сборки
- Особенности в процессах разработки
- Что и как исторически сложилось
- Артефакты привязаны к контуру
- Что бы ни случилось пересобираем проект
- Имеется возможность запустить установку на продуктивный контур пользователям имеющим доступ к проекту



# Немного про Git\*Flow





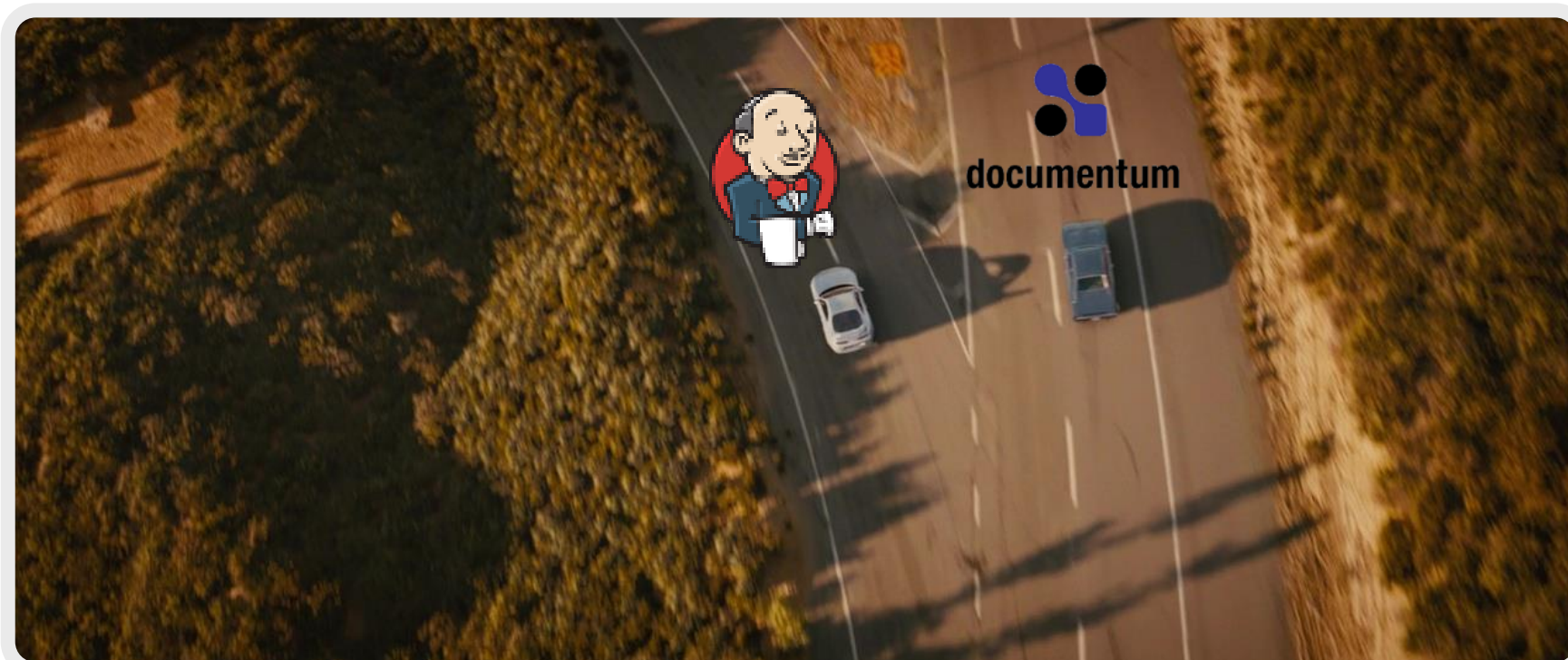
# А зачем уходить от Дженкс?

Отсутствует поддержка

Есть перечень  
вопросов со стороны  
СИБ

Целевым CI инструментом  
выбран Гилеандр CI

Развернут силами  
разработчиков и пущен в  
свободное плавание





# Приступаем к переносу логики деплоя

## Трудности:

- Pipeline на Groovy
- Реалии требуют возможность деплоя разное количество модулей на разные контура
- Деплоить все слишком долго
- Жесткая последовательность деплоя
- Отсутствует сопоставление контура и веток

## Спрашиваем у комьюнити:

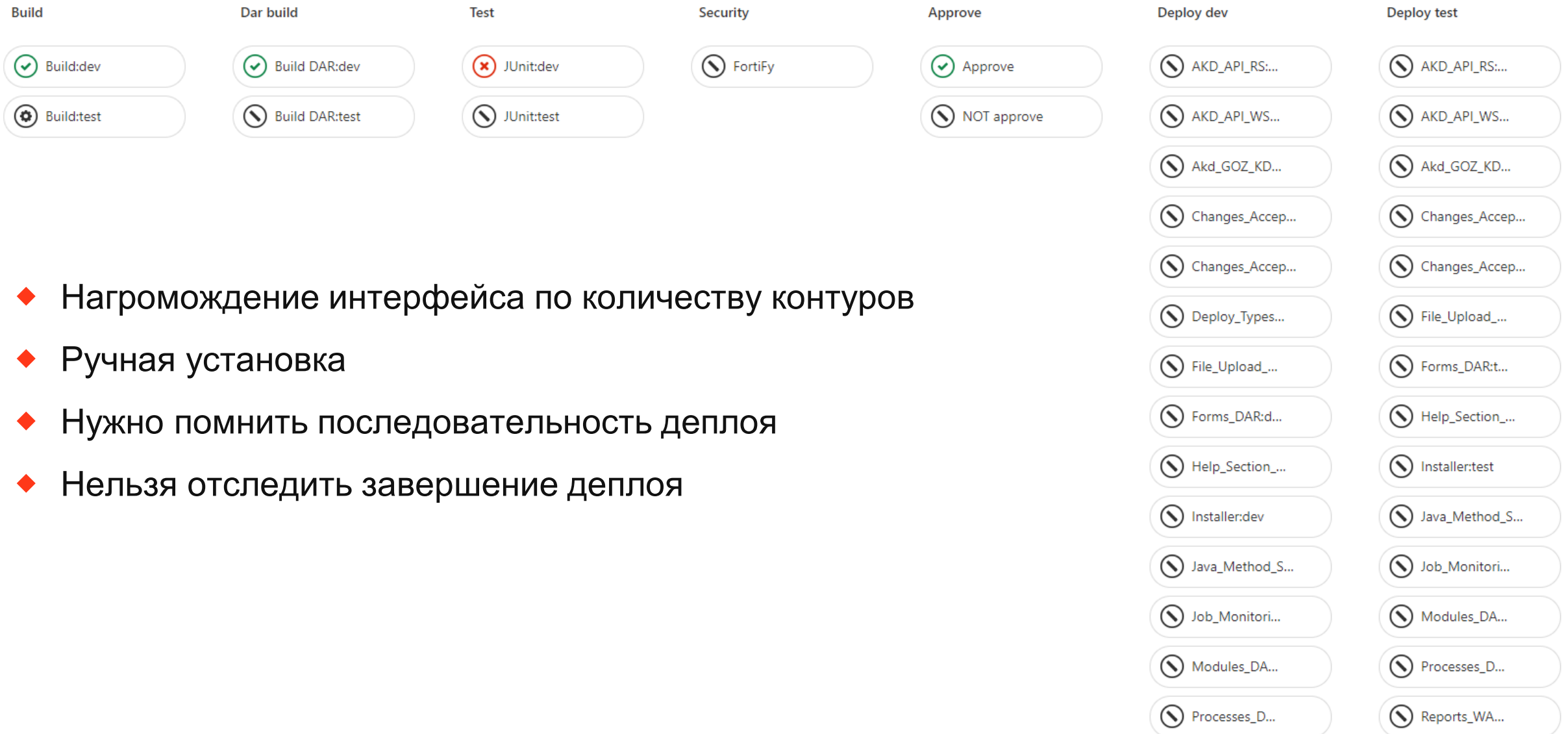
- А точно ли нужен Гелиандр?
- Нужно перепилить проект и поменять процессы
- Нужно поменять инженера за такие вопросы

## Pipeline AKD

Для этой сборки необходимы следующие параметры:

PROFILE	<input type="text" value="prod"/>
	<small>Профиль окружения</small>
ENVIRONMENT	<input type="text" value="Промышленная среда"/>
	<small>Среда для установки</small>
BRANCH	<div><div><div>3.30.0</div><div></div></div><div>3.3</div></div>
	<small>Тег/ветка в git</small>
Installer	<input checked="" type="checkbox"/>
Types_DAR	<input checked="" type="checkbox"/>
Modules_DAR	<input checked="" type="checkbox"/>
Forms_DAR	<input type="checkbox"/>
Processes_DAR	<input type="checkbox"/>
Taskspace_WAR	<input type="checkbox"/>
Reports_WAR	<input type="checkbox"/>
Uploader_BIN	<input type="checkbox"/>
Uploader_WAR	<input type="checkbox"/>
Changes_Acceptor_EAR	<input type="checkbox"/>
Changes_Acceptor_BIN	<input type="checkbox"/>
Java_Method_Server	<input type="checkbox"/>
AKD_API_WS	<input type="checkbox"/>
AKD_API_RS	<input type="checkbox"/>
<input type="button" value="Собрать"/>	

# Первая реализация Pipeline



- ◆ Нагромождение интерфейса по количеству контуров
- ◆ Ручная установка
- ◆ Нужно помнить последовательность деплоя
- ◆ Нельзя отследить завершение деплоя

# Попытка №2

- ◆ Требуется в задачах менять ветку или указывать контур через переменные
- ◆ Ручная установка
- ◆ Нужно помнить последовательность деплоя
- ◆ Нельзя отследить завершение деплоя

Build

✓ Build

Dar build

✓ Build DAR

Test

✓ JUnit

Deploy

✓ AKD\_API\_RS

✓ AKD\_API\_WS

✓ Akd\_GOZ\_KD...

! Changes\_Acce...

✓ Changes\_Acce...

✓ File\_Upload\_...

✓ Forms\_DAR

✓ Help\_Section...

✓ Installer

✓ Java\_Method...

! Job\_Monitori...

✓ Modules\_DAR

✓ PRINT\_FORM...

✓ Processes\_D...

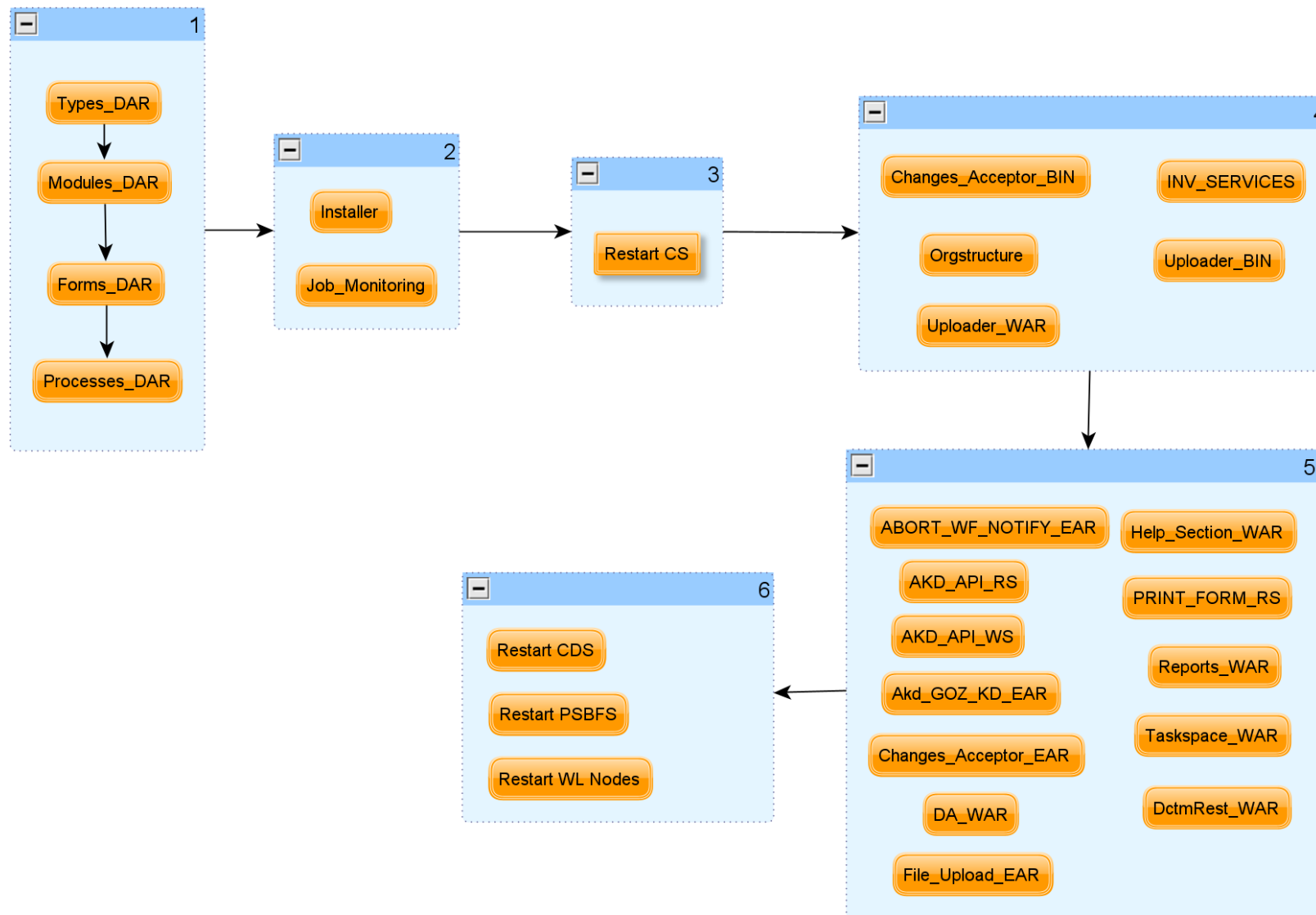
All 3 Active 0 Inactive 3

New schedule

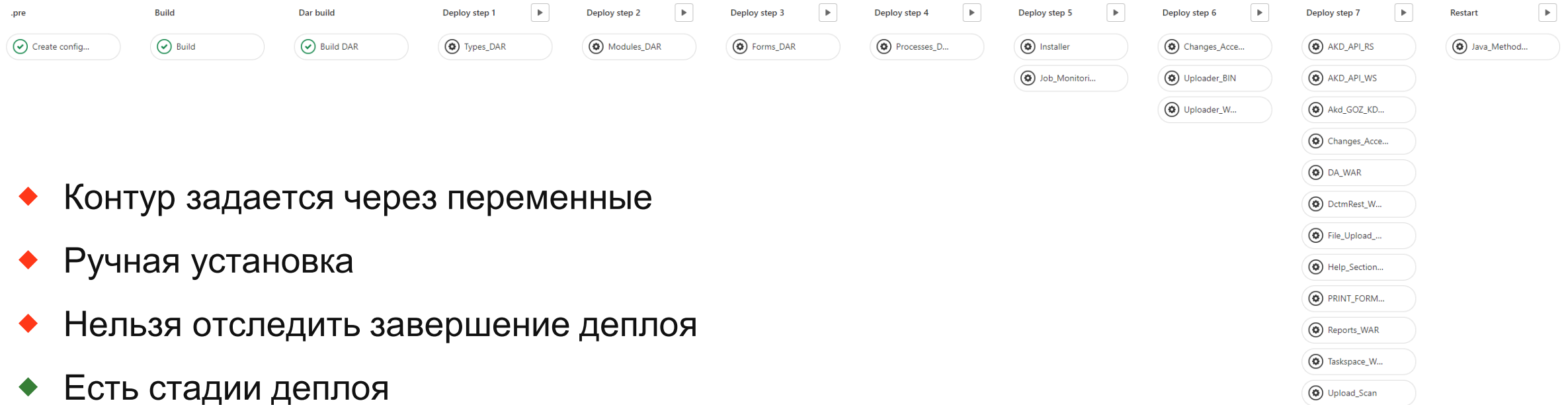
Description	Target	Last Pipeline	Next Run	Owner	
PROD:Deploy	Y pipeline-dev	#180019	Inactive	Galonza Petr Valerevich	▶ ✎ 🗑
TEST:Deploy	Y pipeline-dev	#177646	Inactive	Galonza Petr Valerevich	▶ ✎ 🗑
DEV:Deploy	Y develop	#243032	Inactive	Galonza Petr Valerevich	▶ ✎ 🗑



# Выделение этапов деплоя

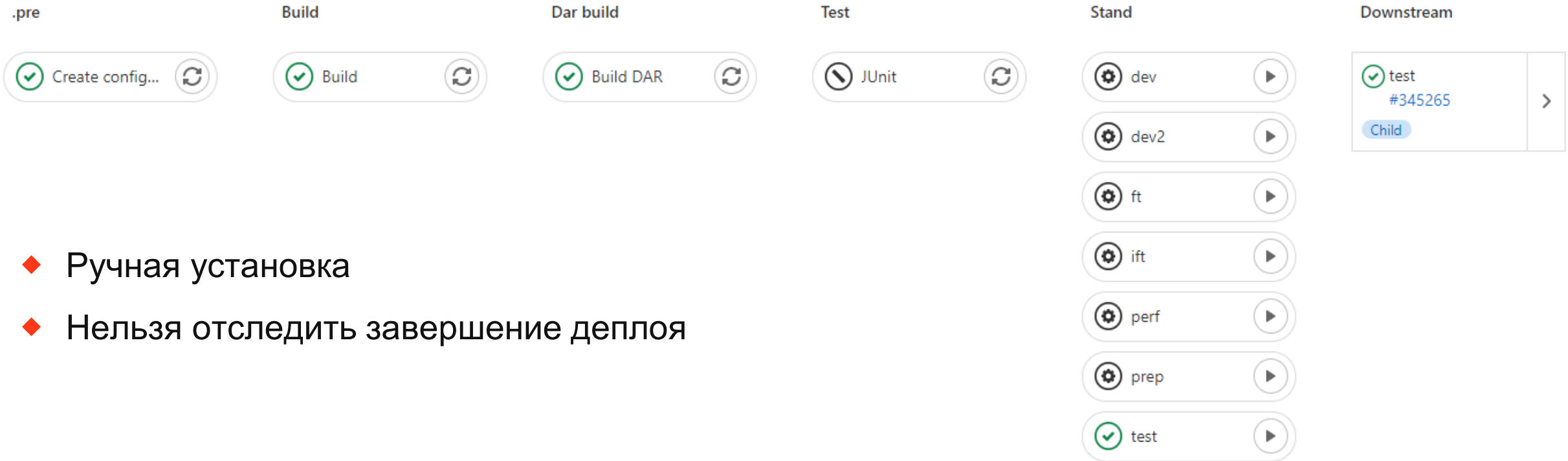


# Разбиение деплоя на стадии



- ◆ Контур задается через переменные
- ◆ Ручная установка
- ◆ Нельзя отследить завершение деплоя
- ◆ Есть стадии деплоя

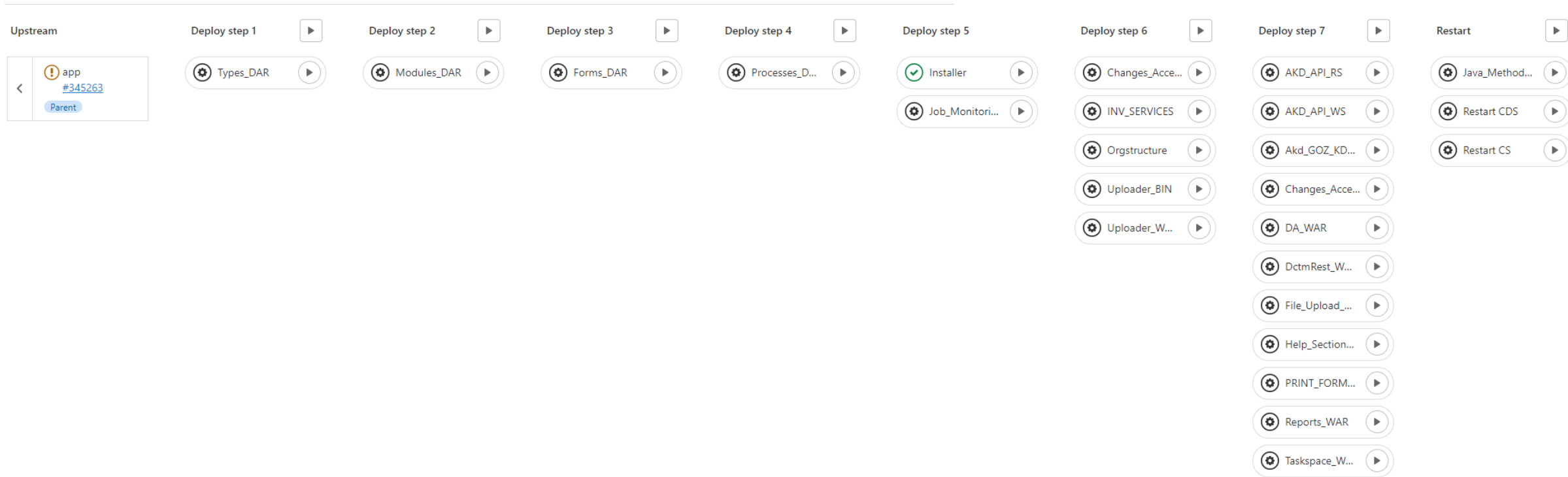
# Конечная реализация



- ◆ Ручная установка
- ◆ Нельзя отследить завершение деплоя



# Pipeline для деплоя



# Итог

1

Сборка и деплой в Гелиандр CI

3

Артефакты теперь в Нейрис

2

Совместно проведены работы по обезличиванию артефактов

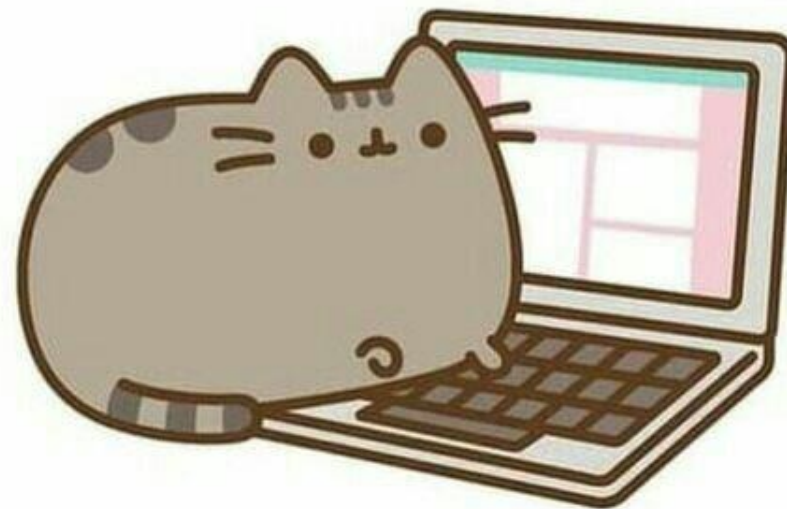
4

Параллельное исполнение задач

# А зачем автоматизировать?

- Мы же за автоматизацию
- Экономия времени
- Удобство в установке на несколько контуров + разный состав релиза
- Знаем начало и окончание установки

Я ПРОГРАММИСТ

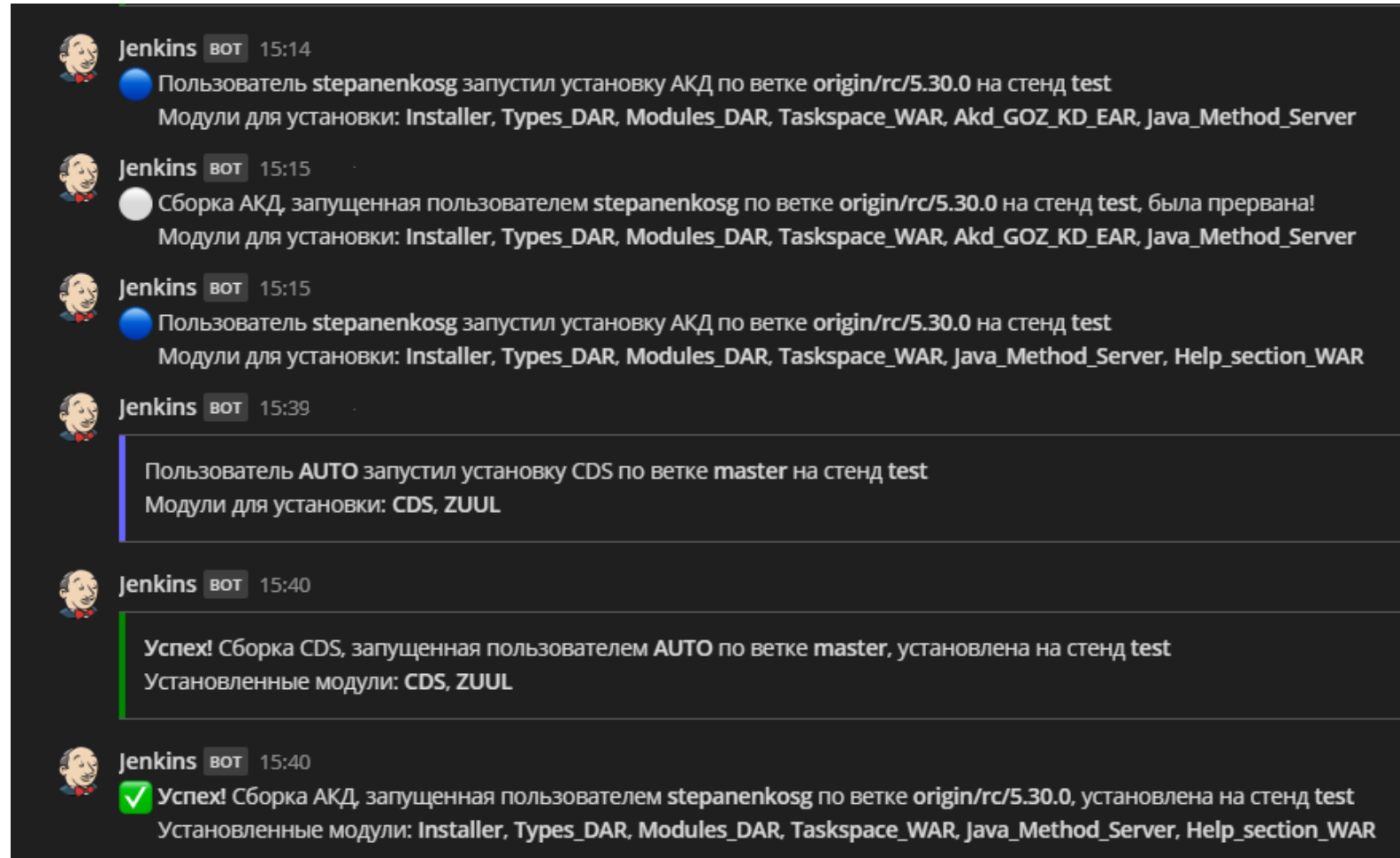


я делаю на компьютере  
клац клац клац



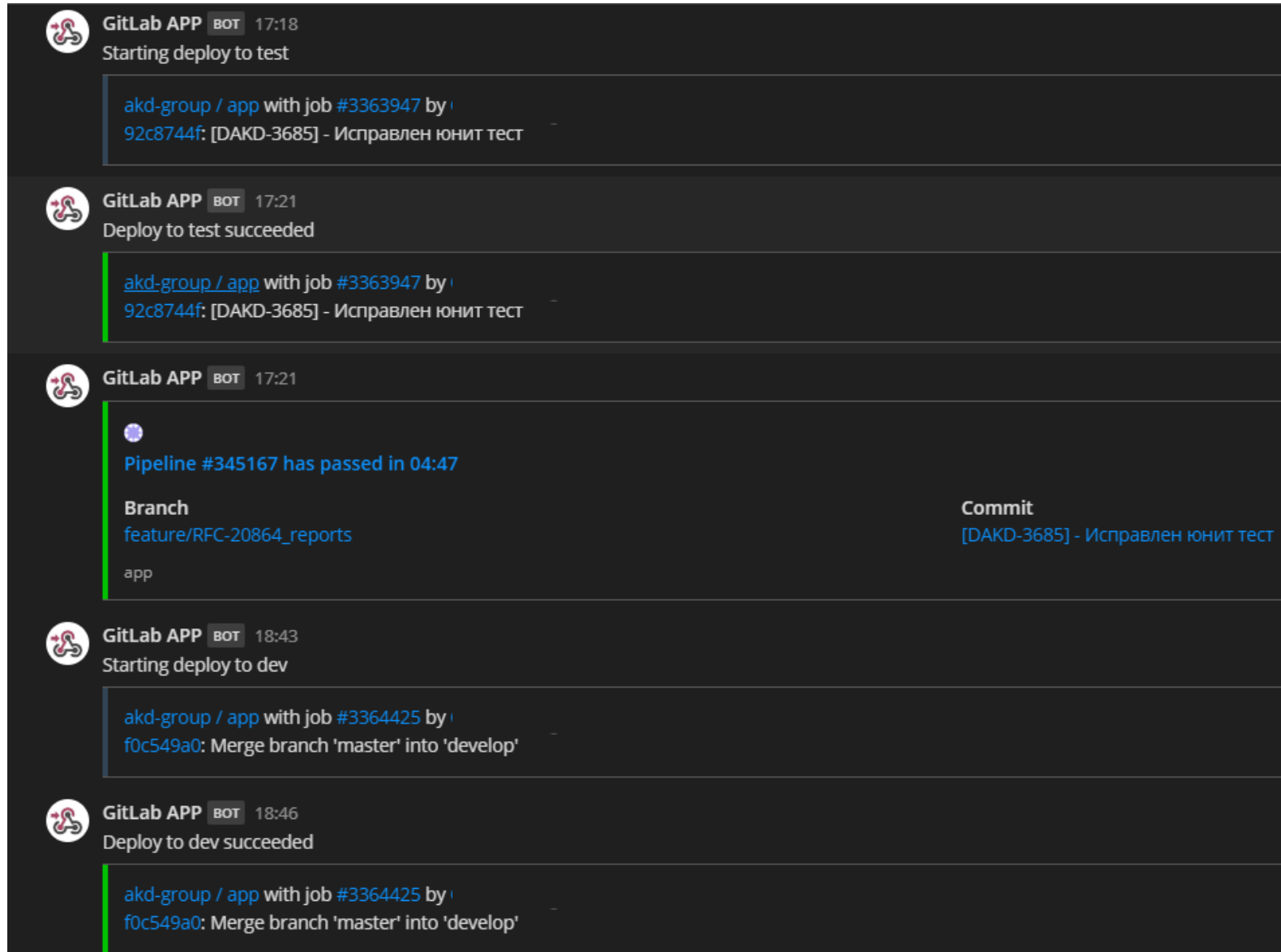
# Зачем знать о начале и завершении установки?

- Тестирование получает информацию о готовности стенда
- Если идет деплой все знают, по составу установки, какой контур и какие модули не трогать
- Дополнительные действия



# Зачем знать о начале и завершении установки?

- Не позволяет понять состав релиза
- Не позволяет понять о конце установки
- Контур установки
- С manual много спама



The screenshot displays a series of chat messages from the GitLab APP BOT, providing real-time updates on deployment processes. The messages are as follows:

- 17:18:** "Starting deploy to test". The message body contains a link to the deployment job: [akd-group / app with job #3363947 by 92c8744f: \[DAKD-3685\] - Исправлен юнит тест](#).
- 17:21:** "Deploy to test succeeded". The message body contains the same link as above.
- 17:21:** "Pipeline #345167 has passed in 04:47". Below this, it lists the branch as `feature/RFC-20864_reports` and the commit as `[DAKD-3685] - Исправлен юнит тест`.
- 18:43:** "Starting deploy to dev". The message body contains a link to the deployment job: [akd-group / app with job #3364425 by f0c549a0: Merge branch 'master' into 'develop'](#).
- 18:46:** "Deploy to dev succeeded". The message body contains the same link as above.

GitLab APPBOT13:07

Deploy to prep succeeded

akd-group / app with job #3356426 by 9c7e7b71: Merge branch 'help\_5.40' into 'rc/5.40.0'

GitLab APPBOT13:07

Starting deploy to prep

akd-group / app with job #3356447 by 9c7e7b71: Merge branch 'help\_5.40' into 'rc/5.40.0'

GitLab APPBOT15:59

Deploy to dev succeeded

akd-group / app with job #3343685 by 603fe187: Merge branch 'feature/RFC-21526' into 'rc/5.40.0'

GitLab APPBOT11:45

Pipeline #344316 has passed in 10:24

Branch

help\_5.40

Commit

DAKD-3778

app

GitLab APPBOT15:59

Starting deploy to dev

akd-group / app with job #3343685 by 603fe187: Merge branch 'feature/RFC-21526\_manual\_load\_mortgage\_stage\_1' into 'rc/5.40.0'

GitLab APPBOT12:13

Starting deploy to dev2

akd-group / app with job #33410 f12f0a2f: [RFC-21335 DAKD-3774] Добавлена зависимость между модулями

GitLab APPBOT14:16

Deploy to prep succeeded

akd-group / app with job #3329013 by 09cf0835: Merge branch 'feature/RFC-21526\_manual\_load\_mortgage\_stage\_1' into 'rc/5.40.0'

GitLab APPBOT17:29

Pipeline #342886 has failed in 00:10

Branch

rc/5.40.0

Commit

Merge branch 'feature/RFC-21526\_manual\_load\_mortgage\_stage\_1' into 'rc/5.40.0'

app

GitLab APPBOT16:23

Pipeline #343415 has passed with warnings in 00:14

Branch

rc/5.40.0

Commit

Merge branch 'feature/RFC-21526\_manual\_load\_mortgage\_stage\_1' into 'rc/5.40.0'

Failed stage

Deploy step 5

Failed job

Organization

app

GitLab APPBOT14:06

Deploy to prep failed

akd-group / app with job #3329533 by 09cf0835: Merge branch 'feature/RFC-21526\_manual\_load\_mortgage\_stage\_1' into 'rc/5.40.0'



# Автоматизируем установку



# Автоматизируем установку

## Что нам нужно знать для автоматизации



Состав релиза

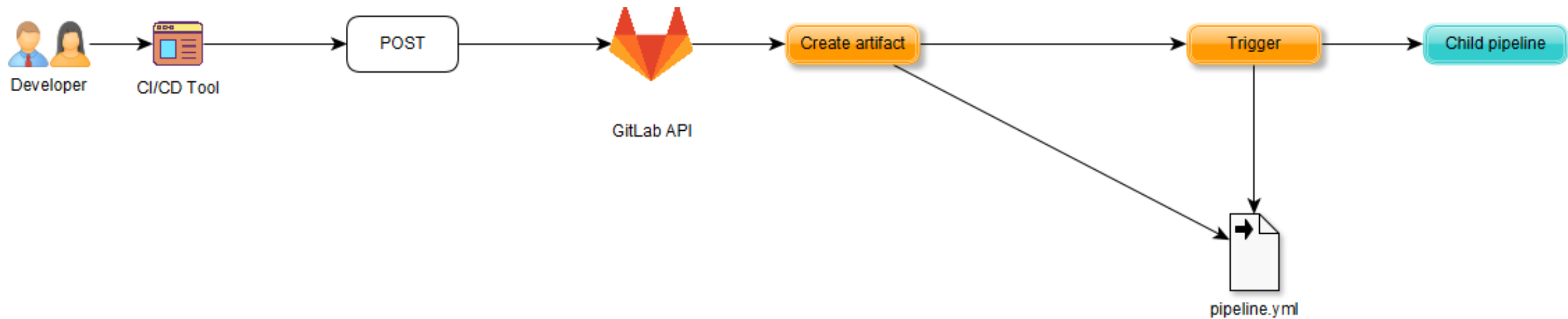


Ветка или  
сборка к  
деплою



Контур для деплоя

# Автоматизируем установку



.pre



Create config...



Deploy step 1



Generate pip...



Deploy step 2



Child-pipeline

Downstream

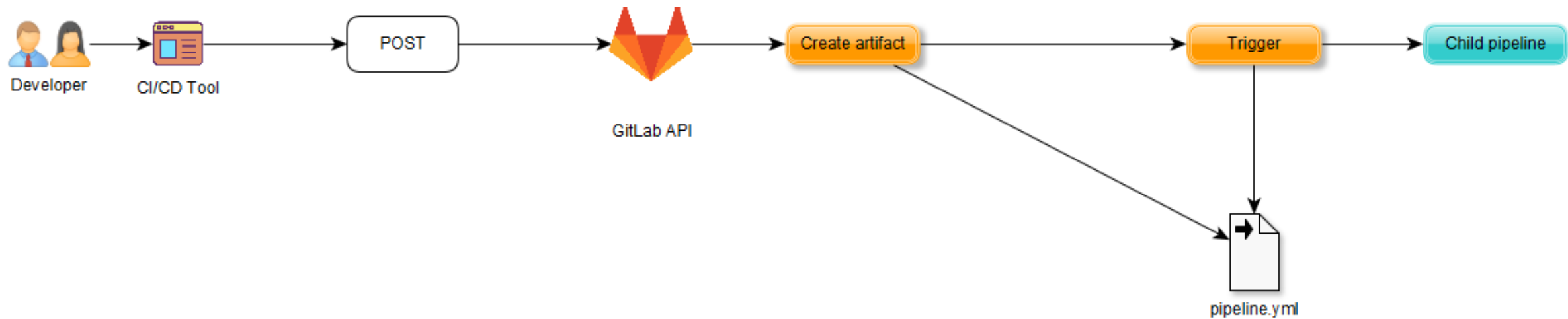


Child-pipeline  
#343277

Child



# Автоматизируем установку



Upstream

<

✓ app  
#341776

Parent

.pre

✓ Create config...

✓ MM start

Deploy step 1

✓ Modules\_DAR

Deploy step 2

✓ Installer

Deploy step 3

✓ INV\_SERVICES

Deploy step 4

✓ Taskspace\_W...

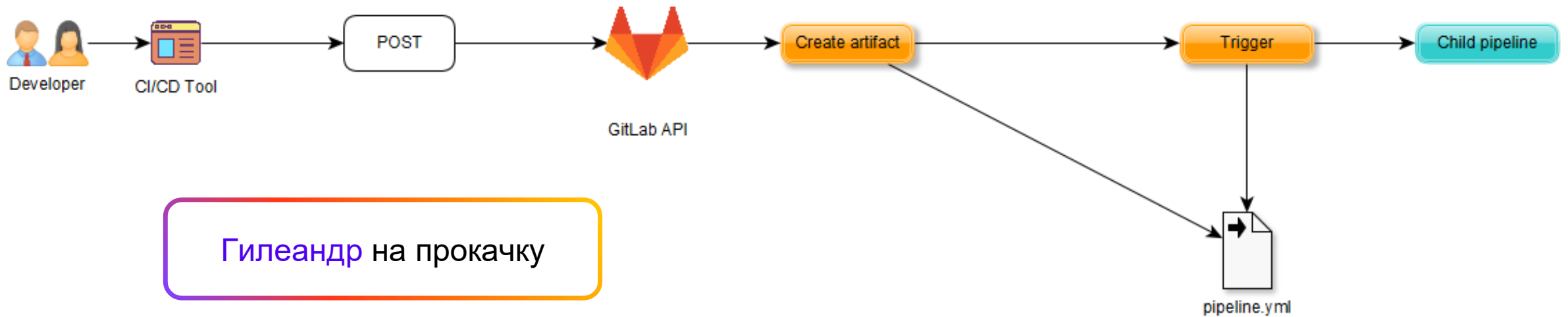
Restart

✓ Java\_Method...

.post

✓ MM finish

# Автоматизируем установку



Мы встроили в pipeline еще один pipeline, чтобы вы могли запускать pipeline пока работает pipeline!

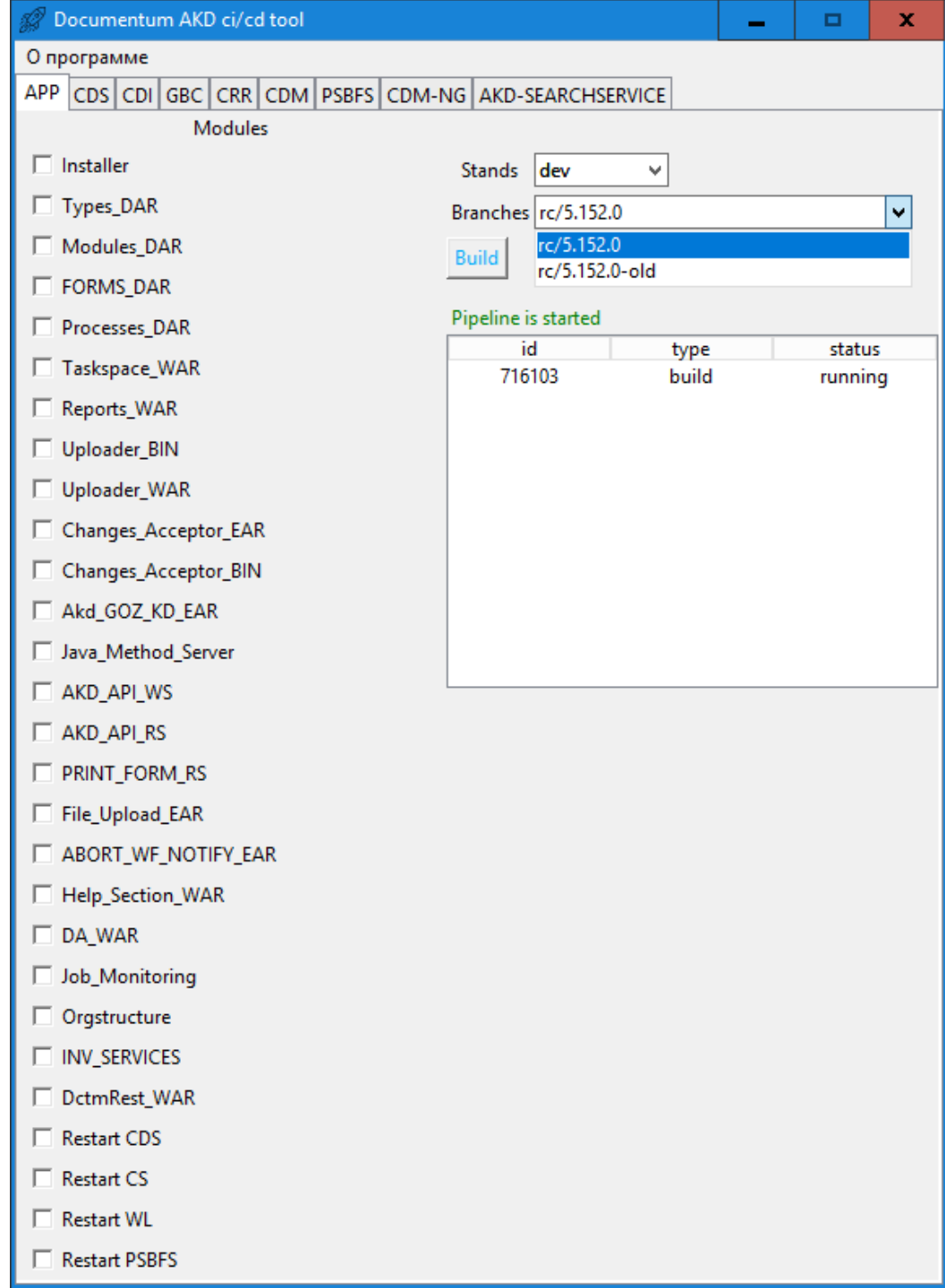


# CI/CD tool

## Реализация:

- tkinter
- threading
- requests
- PyYaml
- GitLab API
- PyInstaller

Сборка Гелиандр CI + Нейрис



# CI/CD tool

## Create a new pipeline

POST /projects/:id/pipeline



Attribute	Type	Required	Description
id	integer/string	yes	The ID or URL-encoded path of the project owned by the authenticated user
ref	string	yes	Reference to commit
variables	array	no	An array containing the variables available in the pipeline, matching the structure [{ 'key': 'UPLOAD_TO_S3', 'variable_type': 'file', 'value': 'true' }]

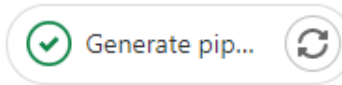
- DEPLOY\_PARAMETERS: {"branch": "rc/5.42.0", "stand": "dev", "modules": ["installer", "taskspace", "help\_section"]}
- DYNAMIC\_PIPELINE: True

# CI/CD tool

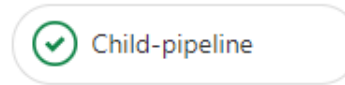
.pre



Deploy step 1



Deploy step 2



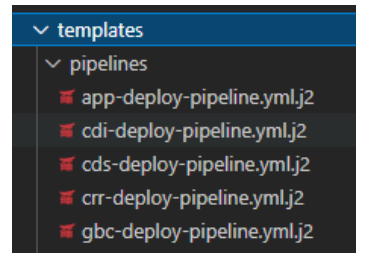
Downstream



```
Generate pipeline:
  stage: deploy step 1
  tags:
    - build01
  extends:
    - .devops_scripts
    - .generate_deploy_pipeline
  script: python devops-scripts/generate-pipeline.py
  artifacts:
    paths:
      - ci-temp/deploy-pipeline.yml
    expire_in: 1 week
```

DEPLOY\_PARAMETERS  
: {"branch":  
"rc/5.42.0", "stand":  
"dev", "modules":  
["installer",  
"taskspace",  
"help\_section"]}

```
--- You, 4 months ago *
deploy:
  app:
    step1:
      - types_dar
    step2:
      - modules_dar
    step3:
      - forms_dar
    step4:
      - processes_dar
    step5:
      - installer
      - job_monitoring
    step6:
      - akd_changes_acceptor
      - orgstructure
      - inv_services
      - uploader_bin
      - uploader_web
    step7:
      - akd_api_ear
      - akd_api_rs_ear
      - goz_kd_ear
      - akd_changes_ear
      - da
      - dctm_rest
      - akd_file_upload_ear
      - help_section
      - print_form_rs
      - reports_war
      - taskspace
```

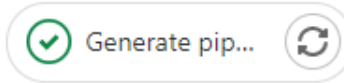


# CI/CD tool

.pre



Deploy step 1



Deploy step 2



Downstream



```
---
include:
  - project: 'AKD_DEVOPS/gitlab-ci'
    ref: develop
    file: '/general/.general-ci.yml'
  - project: 'AKD_DEVOPS/gitlab-ci'
    ref: develop
    file: '/app/.extensions-ci.yml'

stages:
  {% for stage_number in range(1, stages|length + 1) %}
  - deploy step {{ stage_number }}
  {% endfor %}
  - restart

variables:
  PROFILE_ENV: {{ stand }}

{% if 'types_dar' in modules -%}
Types_DAR:
  stage: deploy step {{ stages['step1'] }}
  extends:
    - .general_parameters
    - .deploy_types_dar
    - .devops_scripts
    - .mm_deploy_notification_onfailure
  environment: $PROFILE_ENV

{% endif -%}
{% if 'modules_dar' in modules -%}
Modules_DAR:
  stage: deploy step {{ stages['step2'] }}
  extends:
    - .general_parameters
    - .deploy_modules_dar
    - .devops_scripts
    - .mm_deploy_notification_onfailure
  environment: $PROFILE_ENV
```

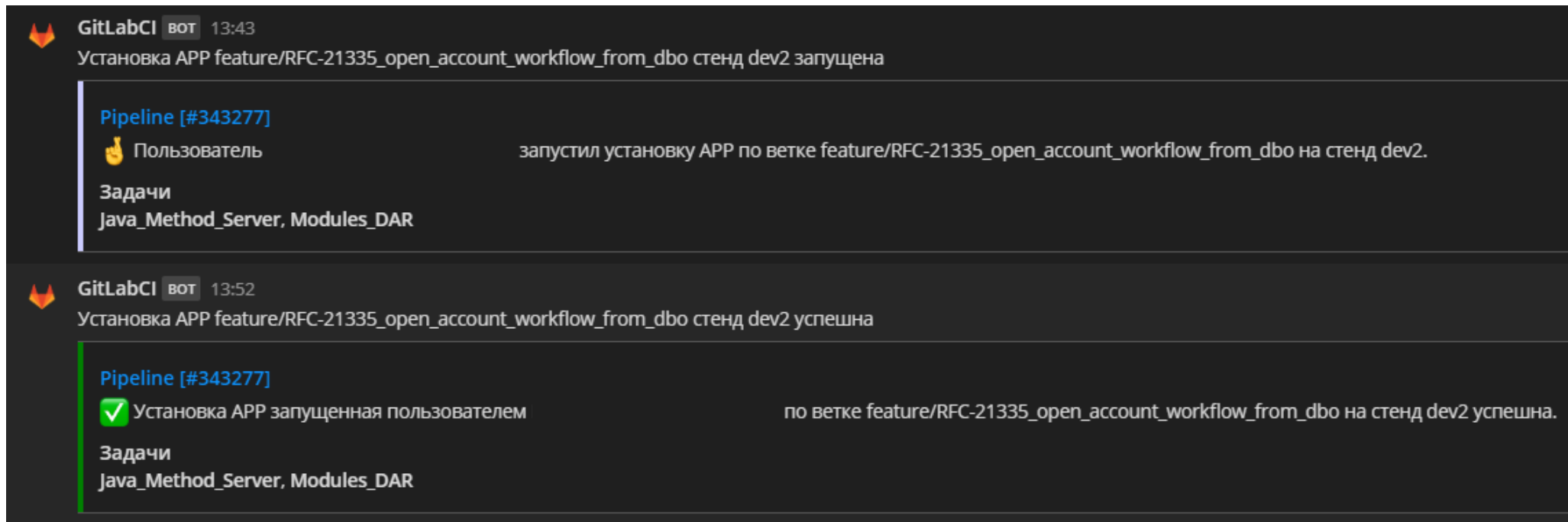


```
Child-pipeline:
  stage: deploy step 2
  extends:
    - .generate_deploy_pipeline
  trigger:
    include:
      - artifact: ci-temp/deploy-pipeline.yml
        job: Generate pipeline
  strategy: depend
```

You, 2 months ago •

# CI/CD tool итоги


- ◆ Вернули как было, но с новыми плюшками
- ◆ Отслеживание статуса pipeline
- ◆ Уведомления в MatterMost
- ◆ Переключение между проектами
- ◆ Достаточно обновлять только конфигурационный файл
- ◆ Дублирование pipeline
- ◆ Еще одна сущность
- ◆ Это desktop приложение



The screenshot displays two chat messages from GitLabCI. The first message, timestamped 13:43, reports the start of an APP installation for pipeline #343277 on the dev2 environment. It includes a 'Pipeline' section with a link to #343277, a 'Пользователь' (User) section with a hand icon, and a 'Задачи' (Jobs) section listing 'Java\_Method\_Server' and 'Modules\_DAR'. The second message, timestamped 13:52, reports the successful completion of the same installation. It includes a 'Pipeline' section with a link to #343277, a 'Установка APP' (APP Installation) section with a green checkmark icon, and a 'Задачи' section listing 'Java\_Method\_Server' and 'Modules\_DAR'.

**GitLabCI** бот 13:43  
Установка APP feature/RFC-21335\_open\_account\_workflow\_from\_dbo стенд dev2 запущена


[Pipeline \[#343277\]](#)

 Пользователь запустил установку APP по ветке feature/RFC-21335\_open\_account\_workflow\_from\_dbo на стенд dev2.

**Задачи**  
Java\_Method\_Server, Modules\_DAR

**GitLabCI** бот 13:52  
Установка APP feature/RFC-21335\_open\_account\_workflow\_from\_dbo стенд dev2 успешна

[Pipeline \[#343277\]](#)

 Установка APP запущенная пользователем по ветке feature/RFC-21335\_open\_account\_workflow\_from\_dbo на стенд dev2 успешна.

**Задачи**  
Java\_Method\_Server, Modules\_DAR



- Уход от Дженкс, Гилеандр CI Pipeline
- Что под капотом? Анси, различные сценарии и не только.

# Собираем, анализируем и учитываем особенности проекта

## Особенности установки релиза:

Велби deployment

Томар

DAR-modules

Microservices

Installer

- Зависимости сборки
- Что и как исторически сложилось
- Весь деплой в Мавена, ant + bash



# Изучаем проект и отмечаем узкие места

1

Версии Java различаются на сборочном и целевых серверах

5

Часть артефактов при сборке привязывалась к контуру

2

Для каждого контура был отдельный Composer Headless (это компонент разработки)

6

Артефакты не самодостаточны

3

На сборочном сервере располагались конфигурационные файлы для каждого контура

7

Установка 1го DAR-модуля могла занимать 40 минут

4

Конфигурационные данные для сервисов хранились в персональном рот-файле, что приводило к дублированию

8

Секреты!!!

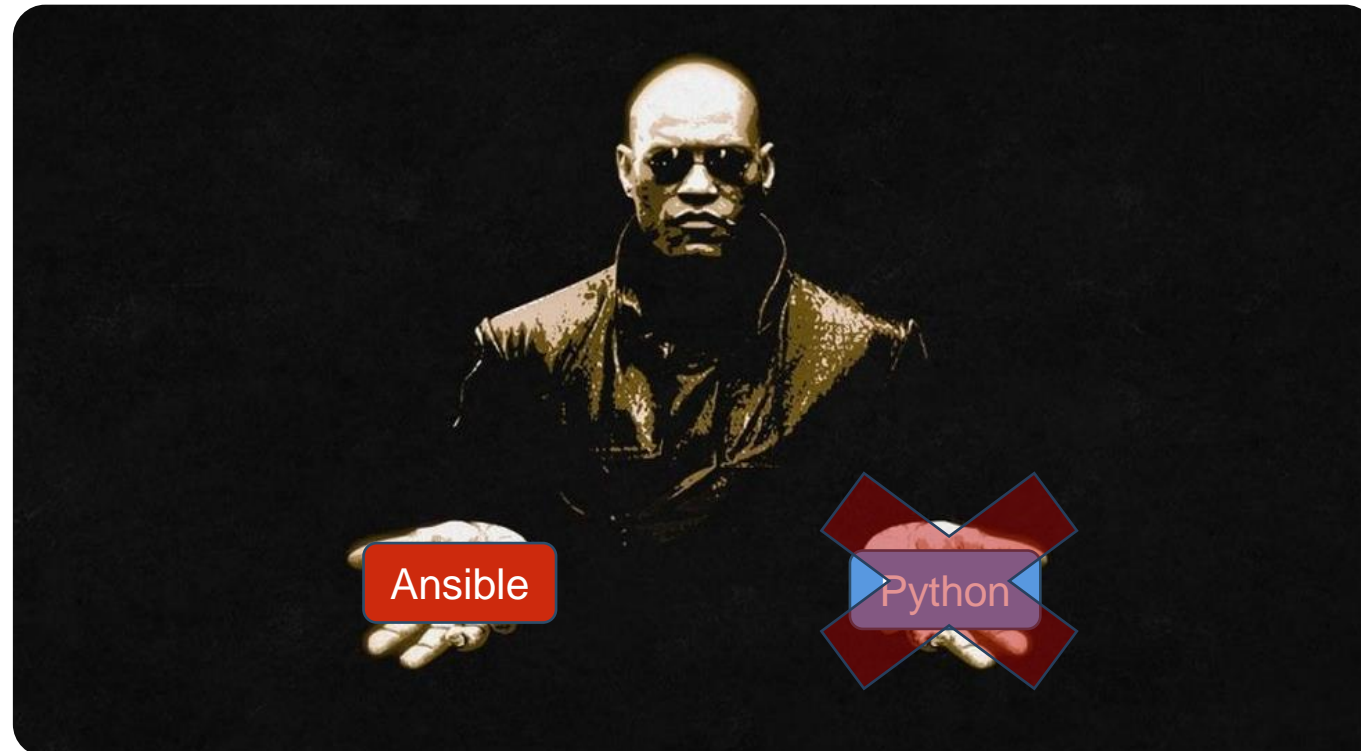
# План

Вынести весь деплой  
из Мавена, ant

Убрать устаревшие  
или ненужное

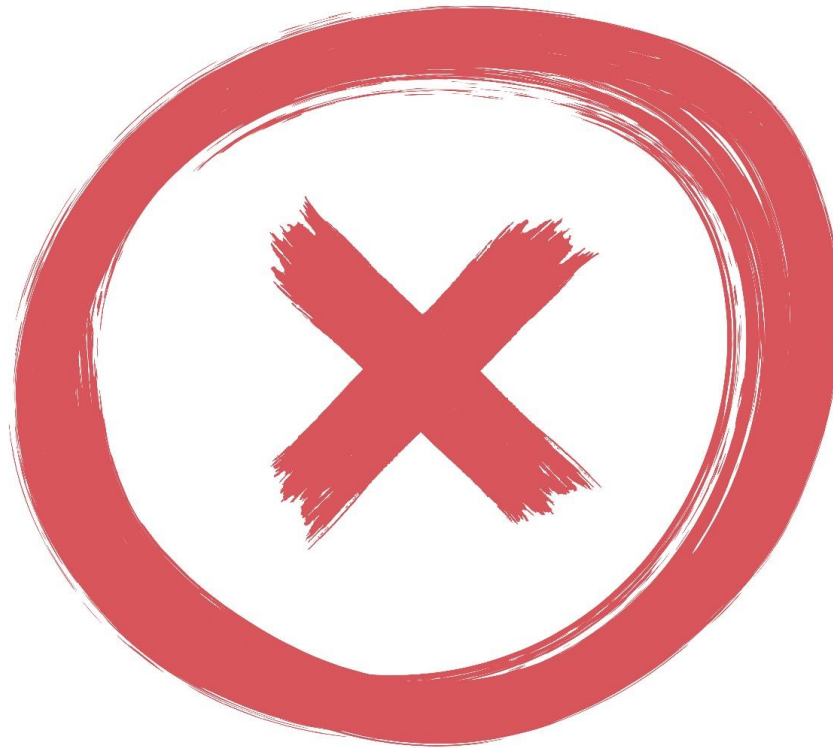
Систематизировать

Собрать конфигурации  
в едином месте



# Почему Python скрипты?

- ◆ Гибкость, не связаны руки (капитан очевидность)
- ◆ Много логики деплоя
- ◆ После улучшений легче перенести на Анси



- ◆ Много строк кода
- ◆ Труднее поддерживать
- ◆ Труднее передать коллегам



# Зачем выносить деплой, и переписывать скрипты?

- Отсутствует унификация
- Maven, ant вызывают скрипты
- Вся конфигурация размазана по pom-файлам(особенность maven)
- Используются maven и плагины старых версий
- Скрипты могут вызывать друг друга
- Конфигурации в xml, ini форматах

- Время на изучение и изменение
- Разработчики уже не в курсе деплоя



## 5 способов деплоя и результат разный

API

Велби.Deployer

WLST

Мавена  
plugin

Велби Admin  
console

- Не можем воспользоваться горячим деплоем, требуется перезапуск нод и очистка кэша (это для каждого модуля)
- Не очень понятная документация
- Странности с отображением
- Нода могла не перезапуститься
- Старый Велби-Мавена-plugin
- Перезапуск через скрипты
- WebLogic Deployment Plan

# Ansible-модуль

- ◆ Проверка существования модуля
- ◆ Деплой
- ◆ Редеплой
- ◆ Deployment.plan
- ◆ Остановка, запуск и перезапуск нод
- ◆ Ожидание корректных статусов
- ◆ Синхронные вызовы
- ◆ Однозадачный

```
- name: Deploy
  weblogic_deploy:
    api_url: "{{ weblogic_web.api_url }}"
    service_name: "{{ lookup('vars', module_name).deploy_name }}"
    servers: "{{ lookup('vars', module_name).weblogic_servers }}"
    file: "{{ artifact_file }}"
    plan: "{{ plan_status.dest | default(None) }}"
    username: "{{ service_properties.username }}"
    password: "{{ service_properties.password }}"
```

# Что ещё

1

Логика bash  
сценариев перенесена  
в Анси

2

Автоматизированы  
повторяющиеся  
ручные действия

3

Добавлена проверка  
доступности по порту

4

Реализован перезапуск  
компонентов content-  
сервера

**Docbroker**

**Content server**

**Java Method Server**

# Пару нюансов

Анси-модули не разрабатываются с учетом ограниченных прав

ACL (Access Control List) это удобно

ACL default не прописываются если файл перемещается

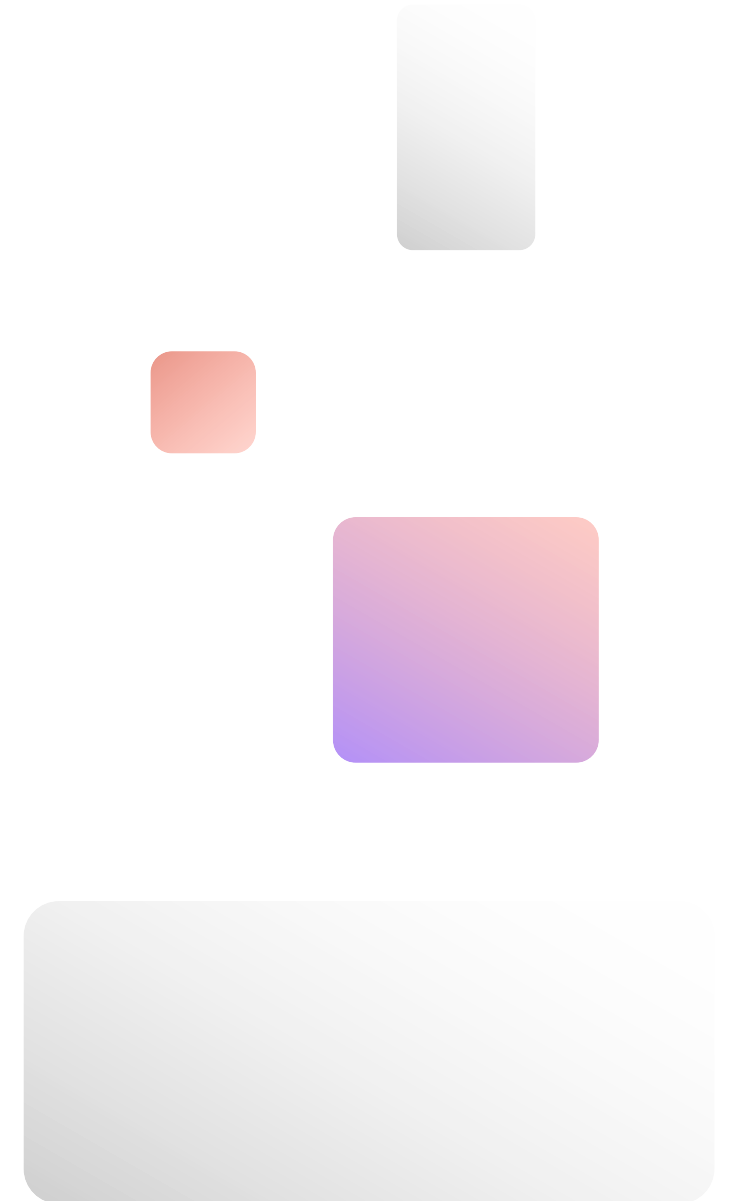
ACL зависят от атрибутов группы. Нужно заботиться об восстановлении прав

Пришлось дополнить модуль systemd для работы с описанными разрешениями на вызов systemctl в sudoers



# Итоги

- Собрали в едином месте конфигурации
- Нашли узкие места и ненужные части
- Легче править и вносить изменения без влияния на репозиторий проекта
- Код и логика деплоя разделены
- Обработка ошибок
- Детализация этапов
- Документировано в confluence + ручной деплой
- Деплой в едином месте целевым инструментом
- Легче передать коллегам.
- Стабильность
- Много опыта



# Горит сарай, гори и хата. Артефакты и Nexus

- ◆ Целевой инструмент для размещения артефактов Nexus
- ◆ Артефакты в jar, war, ear, zip, text форматах
- ◆ Раскиданы по всей структуре
- ◆ Имена артефактов разнятся и могут содержать версию
- ◆ Гилеандр после выполнения задачи уничтожает результат работы, а мы хотим работать в несколько worker`ов и даже на нескольких серверах внедрение, передеплоивать без пересборки
- ◆ Сборка должна быть самодостаточной
- ◆ Вынести конфигурирование из сборки
- ◆ Есть зависимости из предыдущей стадии
- ◆ Сборка app 4GB+ и нужно собирать все

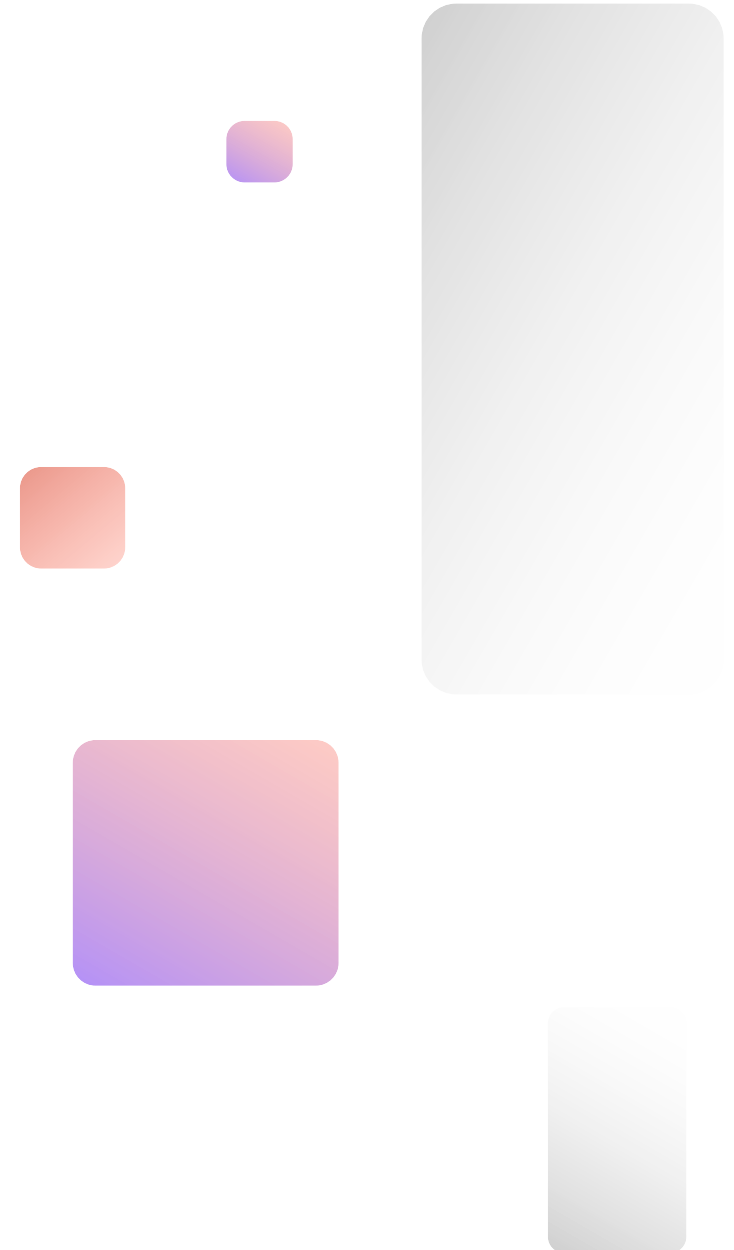
# Горит сарай, гори и хата. Артефакты и Нейрис

```
installer:
  name: installer
  work_dir: installers/installer/target
  regexp_artifacts:
    - regexp: '^installer-[0-9]\.[0-9]\.jar$'
      path: installers/installer/target
taskspace:
  name: taskspace
  work_dir: taskspace/target
  artifacts:
    - taskspace/target/taskspace.war
reports_akd:
  name: reports_akd
  work_dir: reports_akd/target
  artifacts:
    - reports_akd/target/reports_akd.war
reports_drs:
  name: reports_drs
  work_dir: reports_drs/target
  artifacts:
    - reports_drs/target/drs_cr.war
uploader_bin:
  name: ptibr_uploader_bin
  work_dir: uploaders/ptibr_uploader/uploader_bin/target
  regexp_artifacts:
    - regexp: '^ptibr_uploader_bin-[0-9]\.[0-9]\.zip$'
      path: uploaders/ptibr_uploader/uploader_bin/target
```

- Артефакты поставляются в tar
- Сборки для тестовых контуров хранятся с хэшем коммита
- Сборки для продуктивного контура хранятся под номером релиза

# Итог

- Работа с любым набором артефактов
  - Универсальность в настройке загрузки и выгрузки (применима и на других проектах)
  - Изменена сборка и скрипты для упаковки всего необходимого
  - Можем в любой момент получить сборку из Нейрис
  - Можем раскатываться разом на несколько конутов
  - Конфигурирование вынесено в шаблонизацию
- 
- Громоздкое решение
  - Время на реализацию
  - Только для монолита
  - Собираем полностью все



# ИТОГ ИТОГОВ



Перешли с Дженкс на Гилеандр CI



Убраны устаревшие части логики и конфигураций



Совместно с командой разработки было проведено обезличивание и самодостаточности сборки проекта



Деплой полностью вынесен в целевой инструмент



Шаблонизация конфигураций



Использование Нейрис для хранения артефактов



Секреты вынесены из неподобающих мест



Различные оптимизации



Классные оповещения



Предрелизный и импакт отчеты



The background of the image is a collection of numerous ornate silver trophies and awards, including large cups, vases, and smaller figurines, arranged on shelves. A large, semi-transparent rectangular box with a horizontal gradient from orange to purple is centered over the image. Inside this box, the Russian text "Благодарю за внимание!" is written in white. There are also five smaller, semi-transparent square boxes with similar gradients: one in the top-left, one in the top-center, one in the top-right, one in the bottom-left, and one in the bottom-right.

Благодарю за внимание!





Мои заметки  
и наработки

GitHub Profile



Telegram