Со своим самоваром

как переносили распределенный монолит
из одной CI-системы в другую

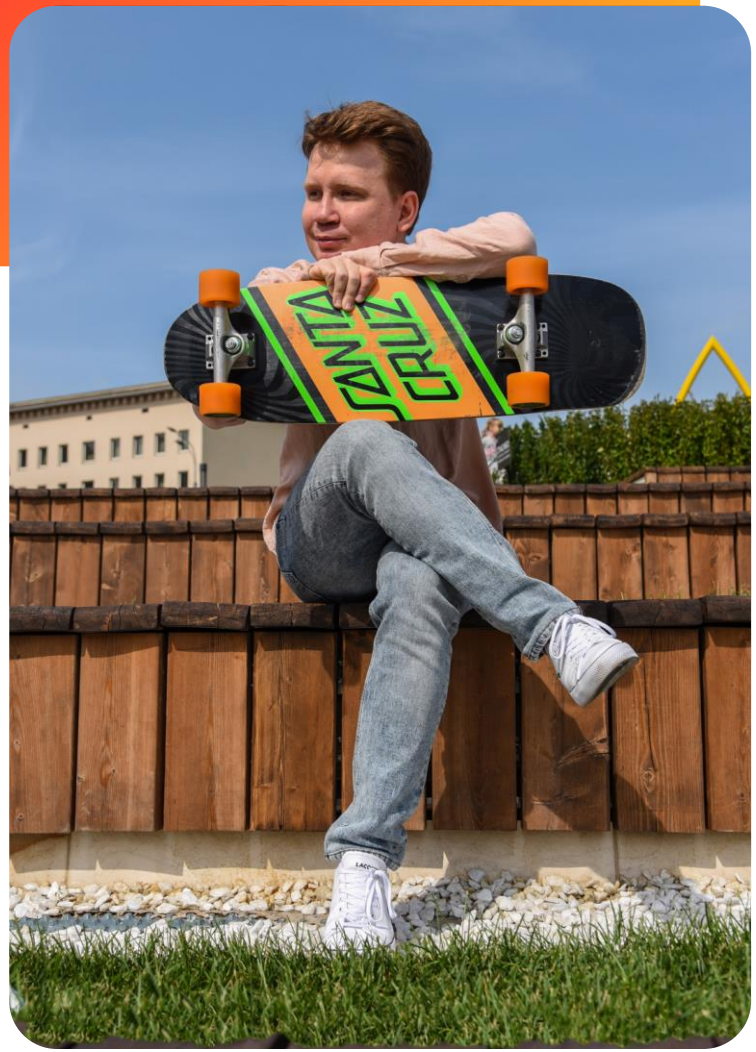
А кто я?

Пётр Галонза

управляющий эксперт отдела внедрения системных сервисов

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 прокачки 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



Стек технологий

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

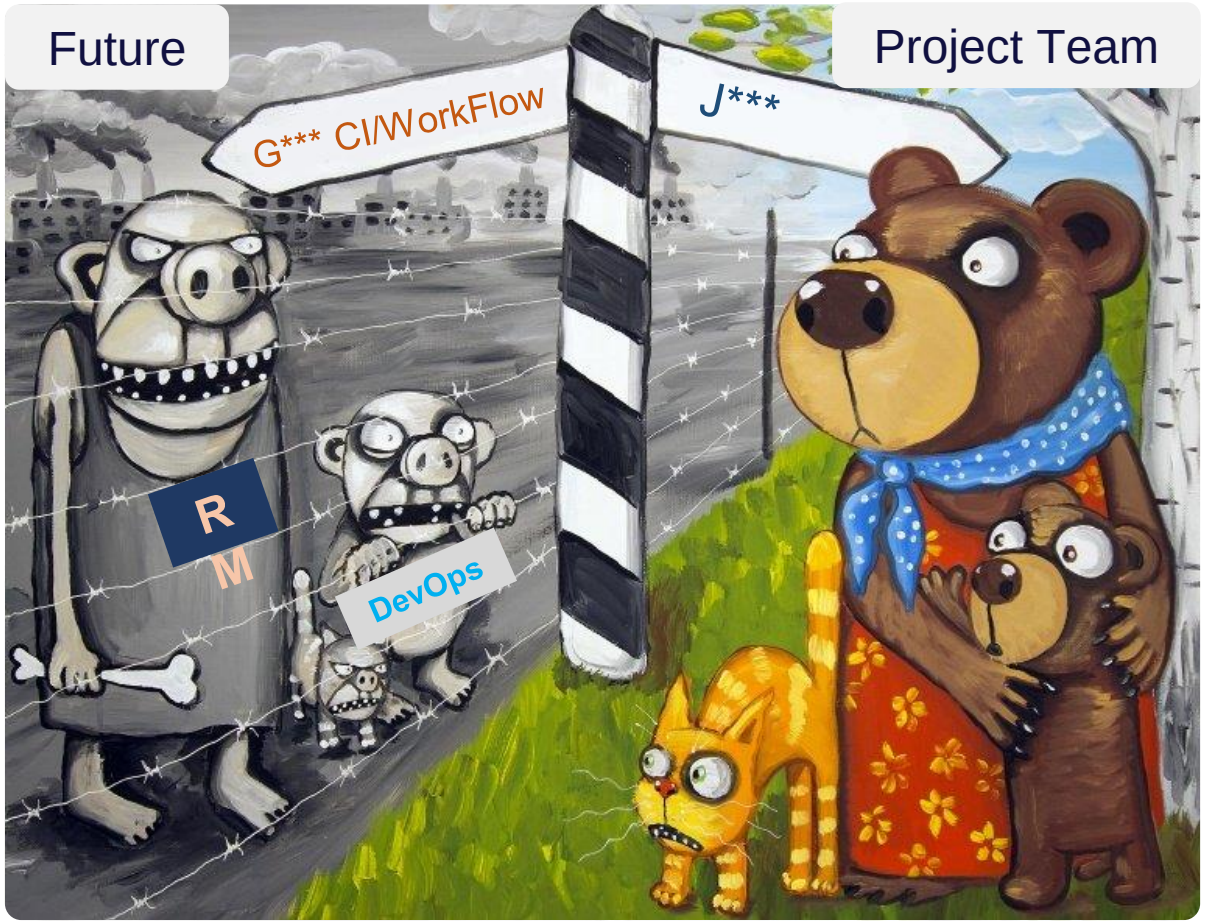
Орктур – база данных версии 12

Мавена – сборщик проектов, используется для основной части

Грэд – сборщик проектов, используется для микросервисов


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



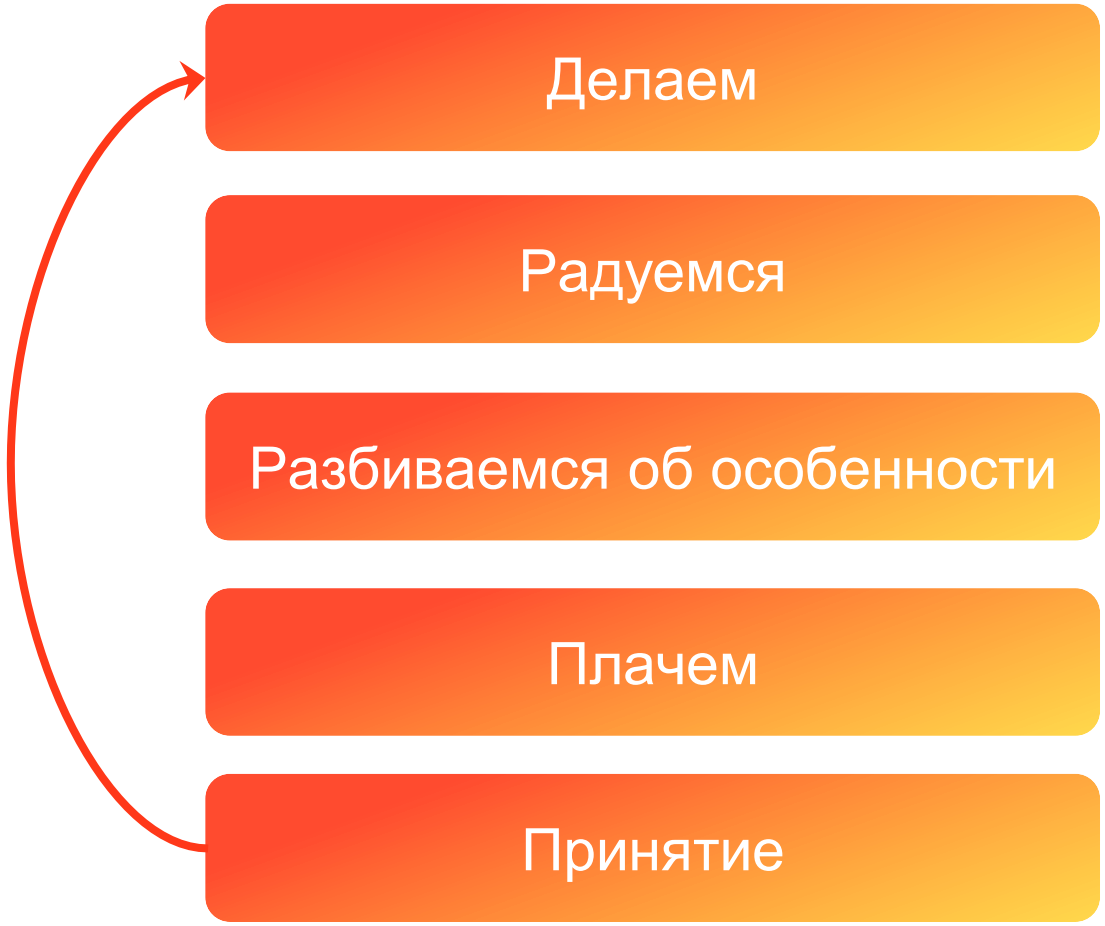
- Устоявшиеся процессы
- Отточенные годами практики, подходы и инструменты
  - Процессы проходят внутри команды
- Новое требует времени, усилий на внедрение и не всегда это оправдано
  - Выбор инструмента исходя из потребностей

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

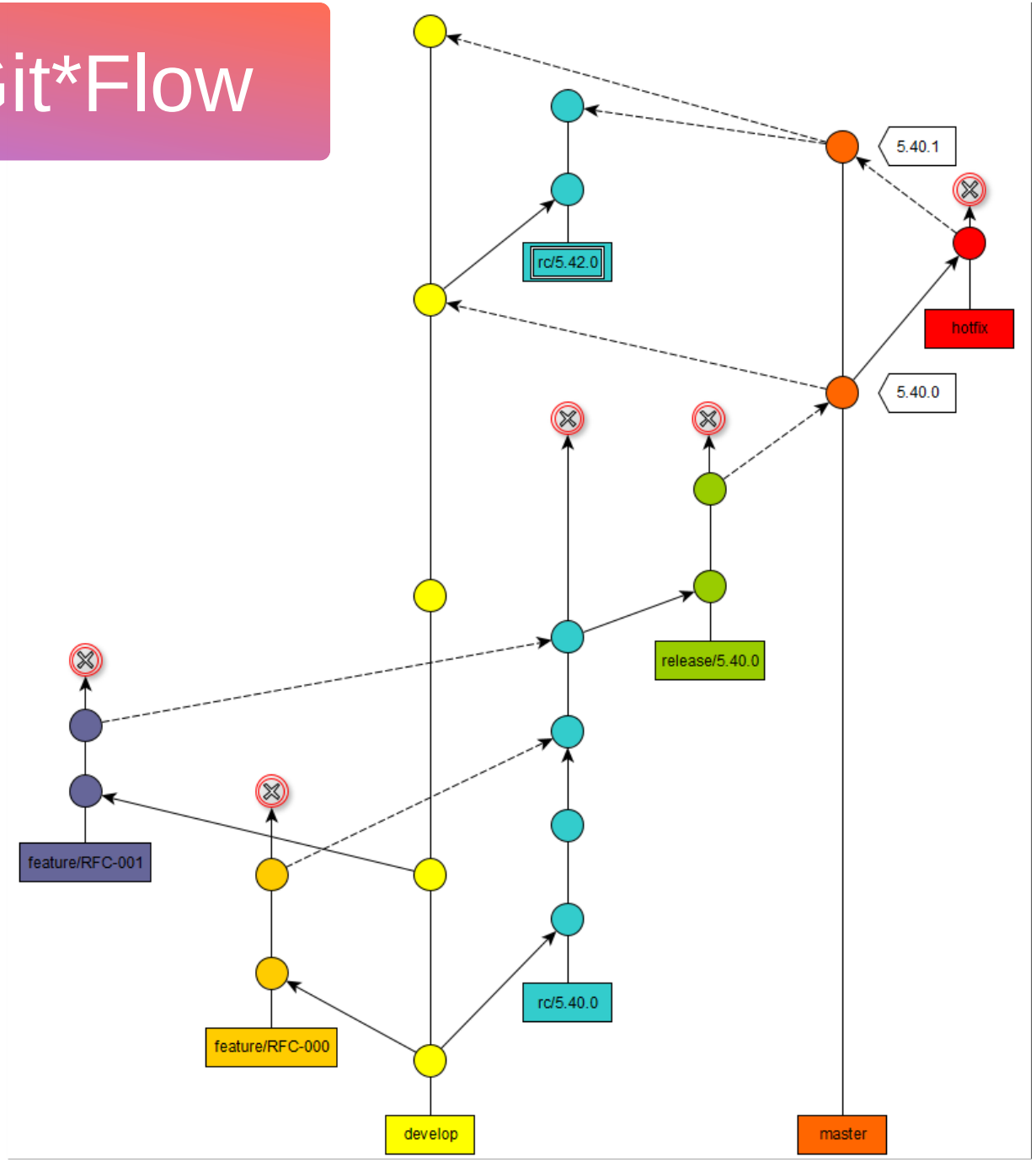
Собираем, анализируем и учитываем особенности, узкие места проекта

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

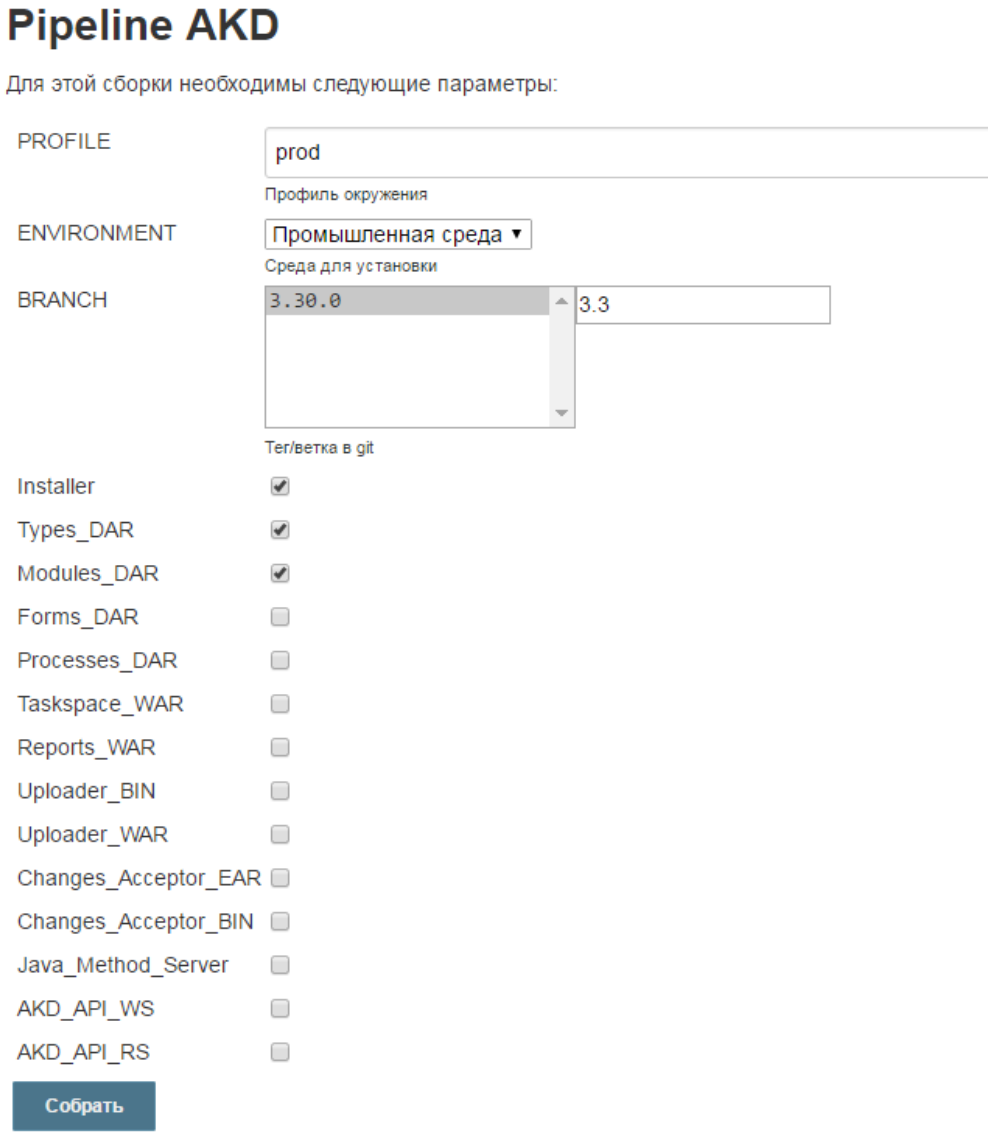
Приступаем к переносу логики деплоя

Трудности:

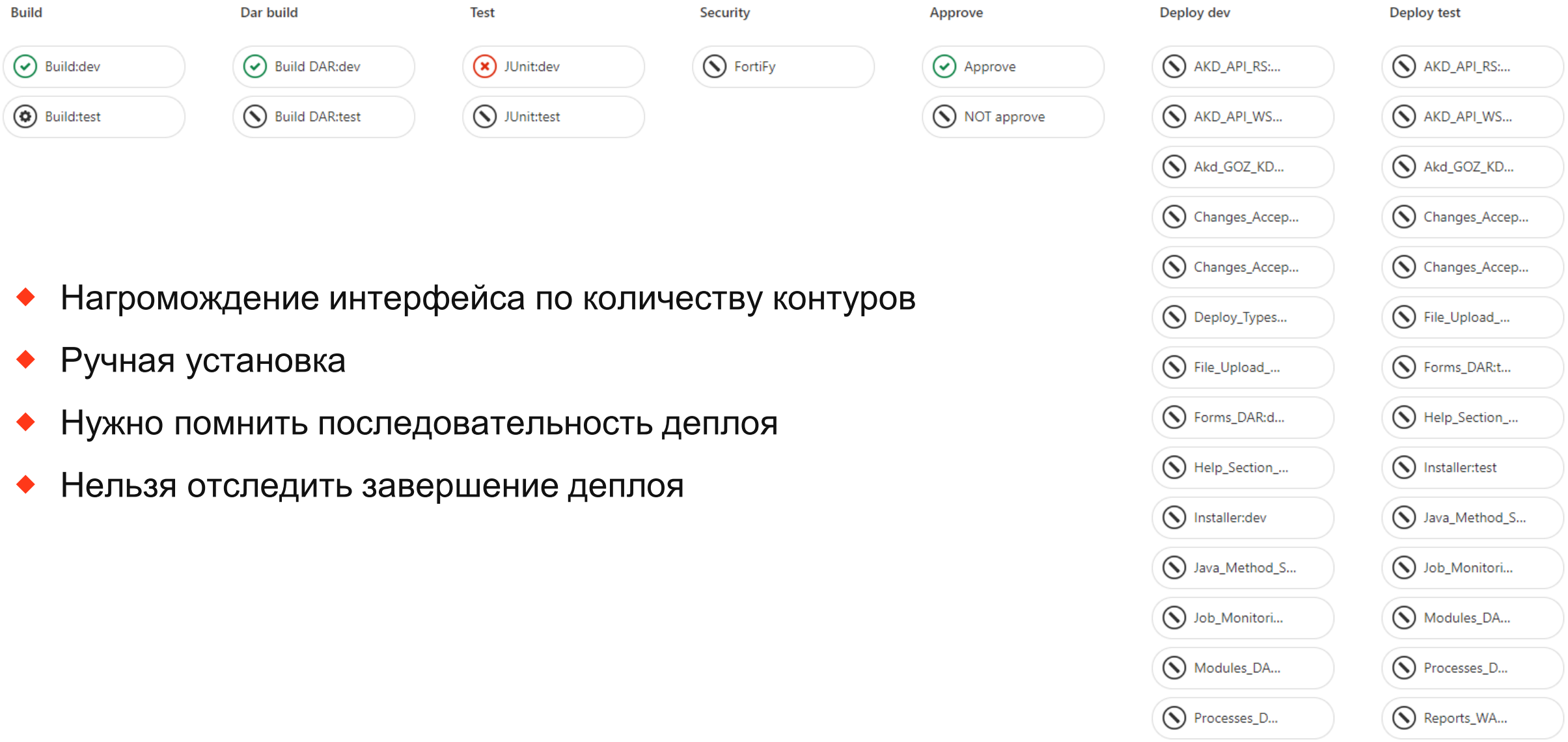
- Pipeline на Groovy
- Реалии требуют возможность деплоя разное количество модулей на разные контура
- Деплоить все слишком долго
- Жесткая последовательность деплоя
- Отсутствует сопоставление контура и веток

Спрашиваем у комьюнити:

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



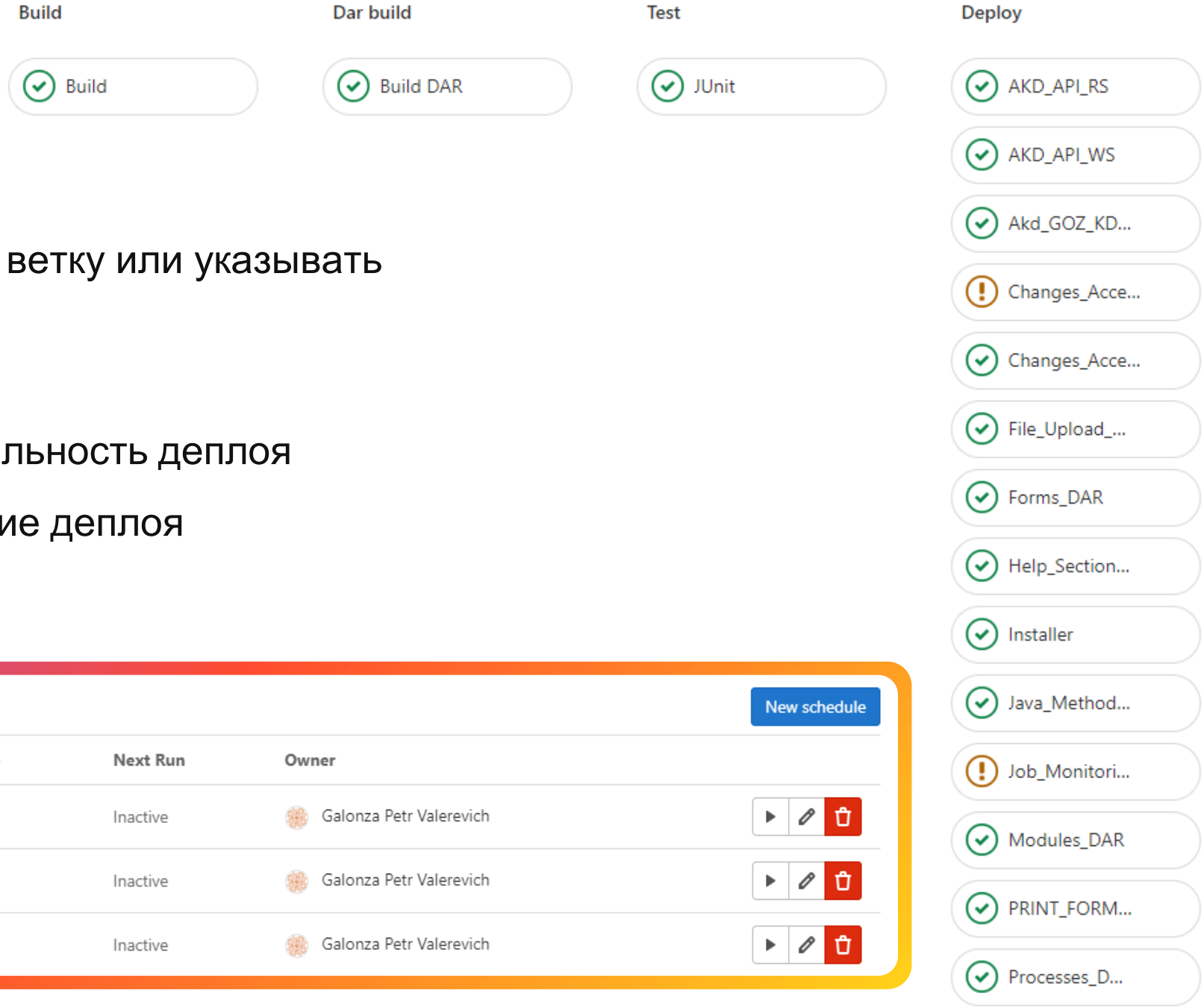
Первая реализация Pipeline



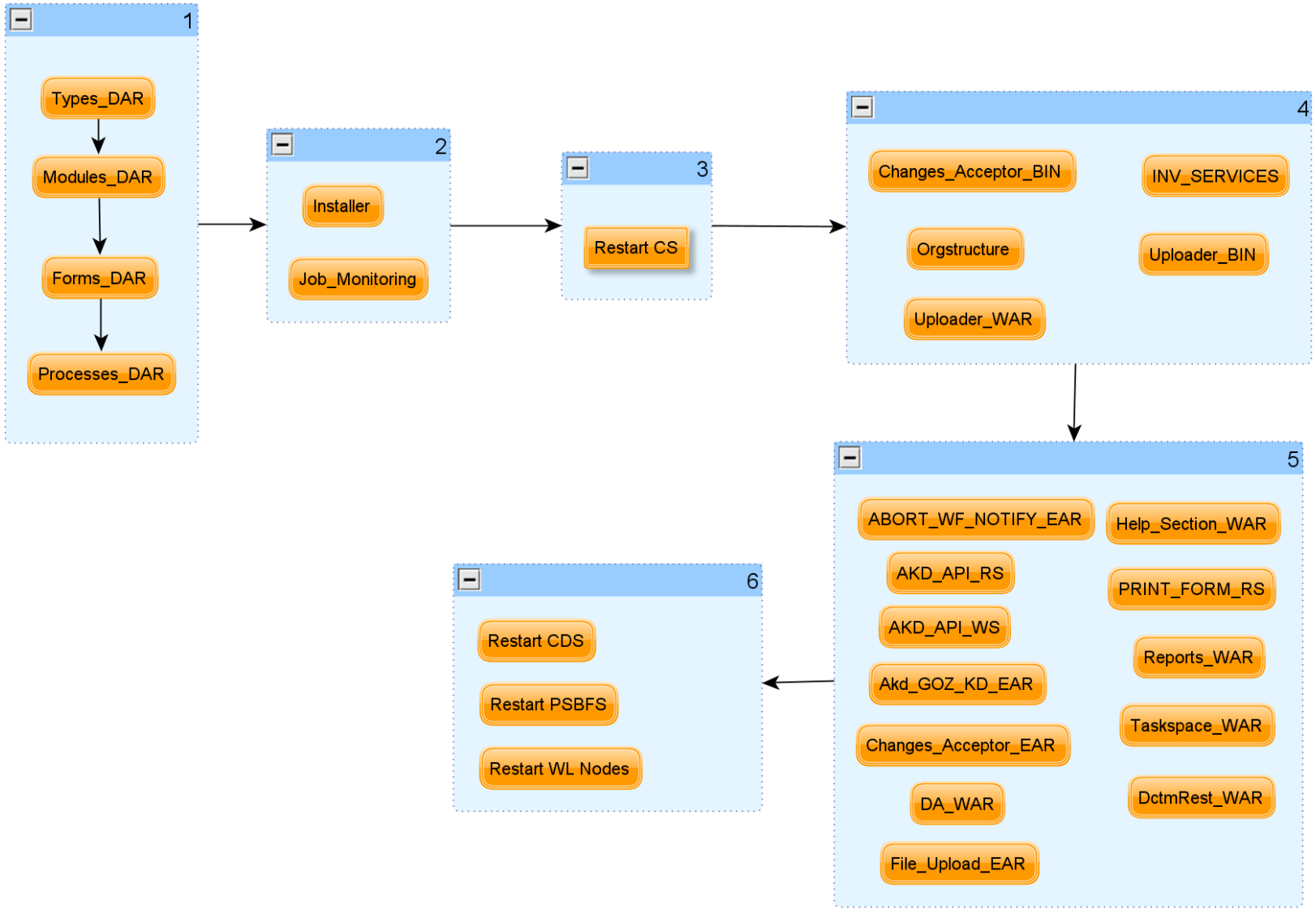
- Нагромождение интерфейса по количеству контуров
- Ручная установка
- Нужно помнить последовательность деплоя
- Нельзя отследить завершение деплоя

Попытка №2

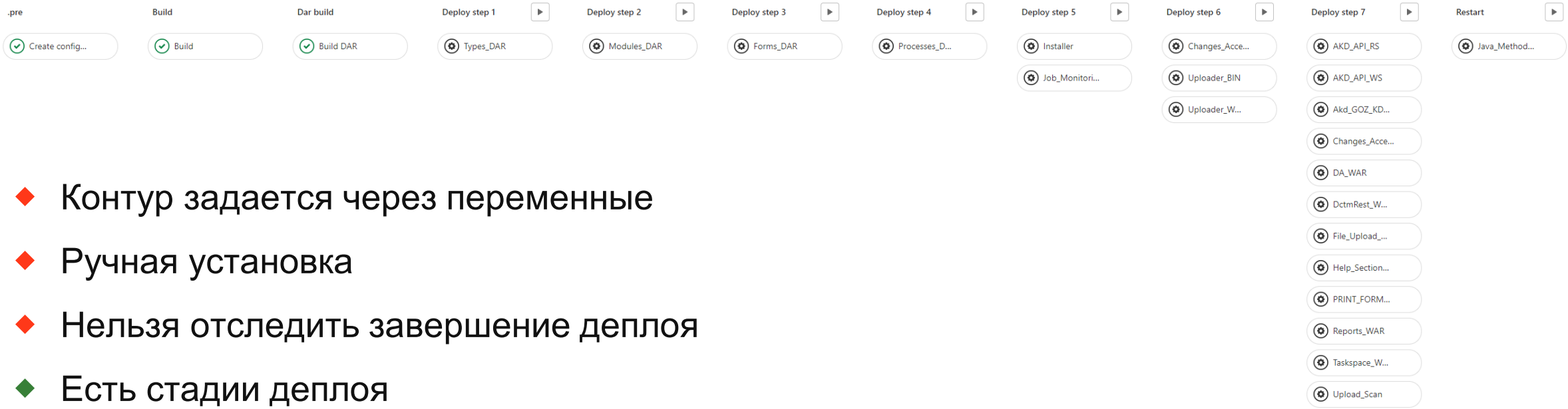
- Требуется в задачах менять ветку или указывать контур через переменные
- Ручная установка
- Нужно помнить последовательность деплоя
- Нельзя отследить завершение деплоя



Выделение этапов деплоя

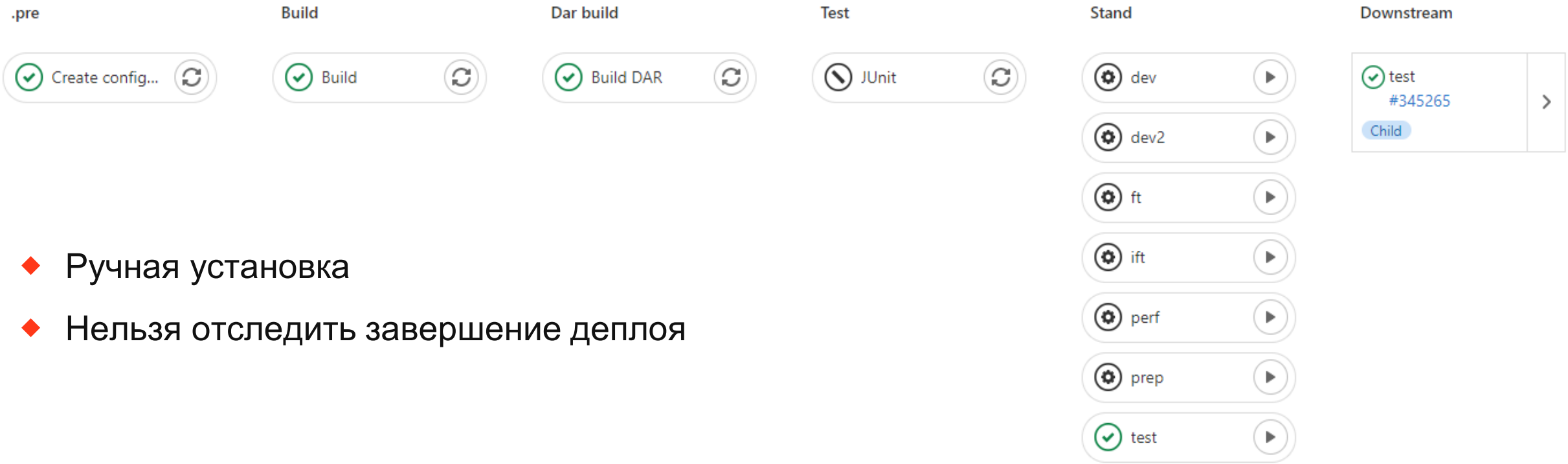


Разбиение деплоя на стадии



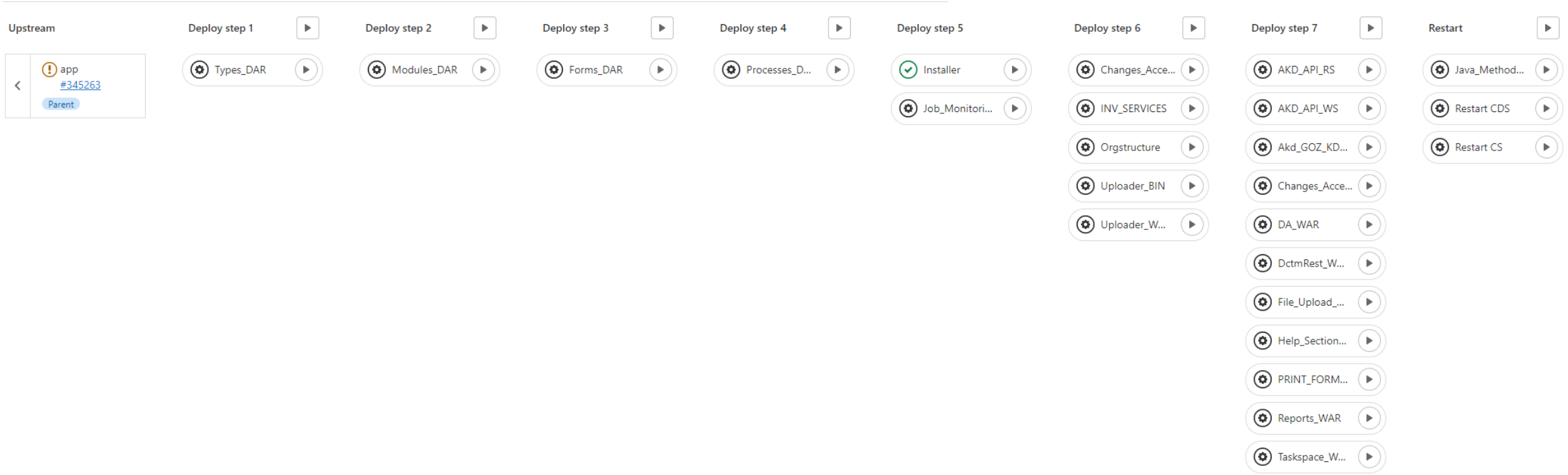
- Контур задается через переменные
- Ручная установка
- Нельзя отследить завершение деплоя
- Есть стадии деплоя

Конечная реализация



- Ручная установка
- Нельзя отследить завершение деплоя

Pipeline для деплоя

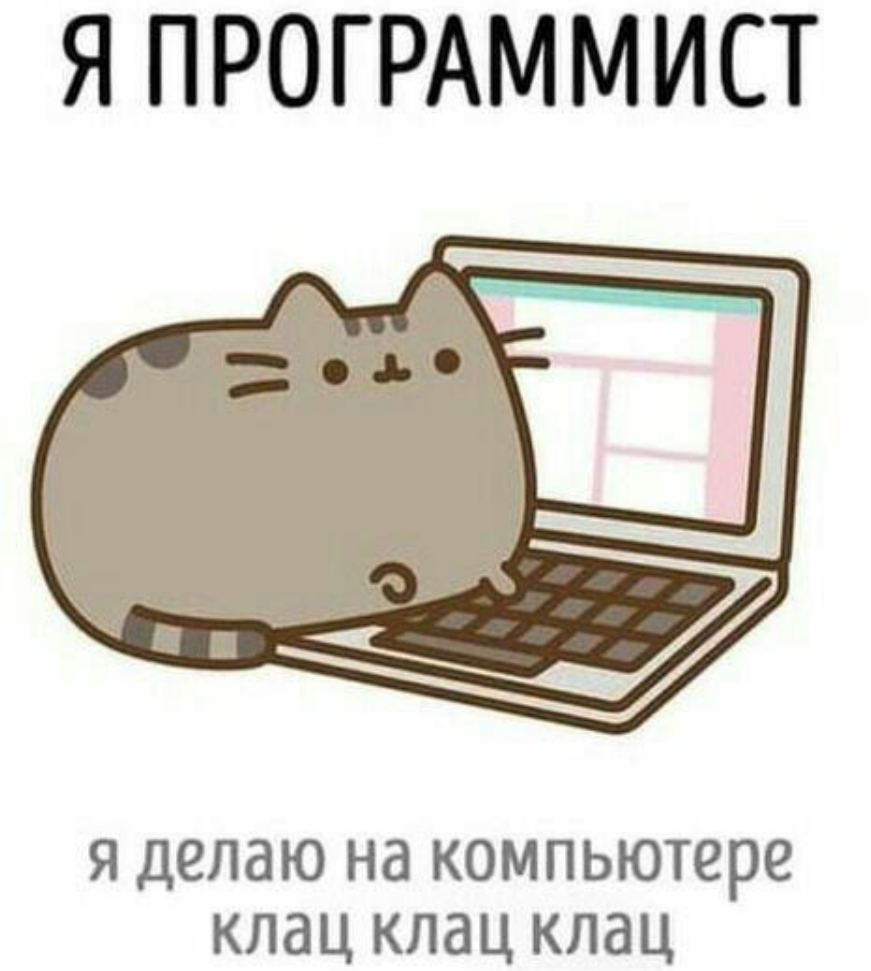


Итог

1. Сборка и деплой в Гелиандр CI
2. Совместно проведены работы по обезличиванию артефактов
3. Артефакты теперь в Нейрис
4. Параллельное исполнение задач

А зачем автоматизировать?

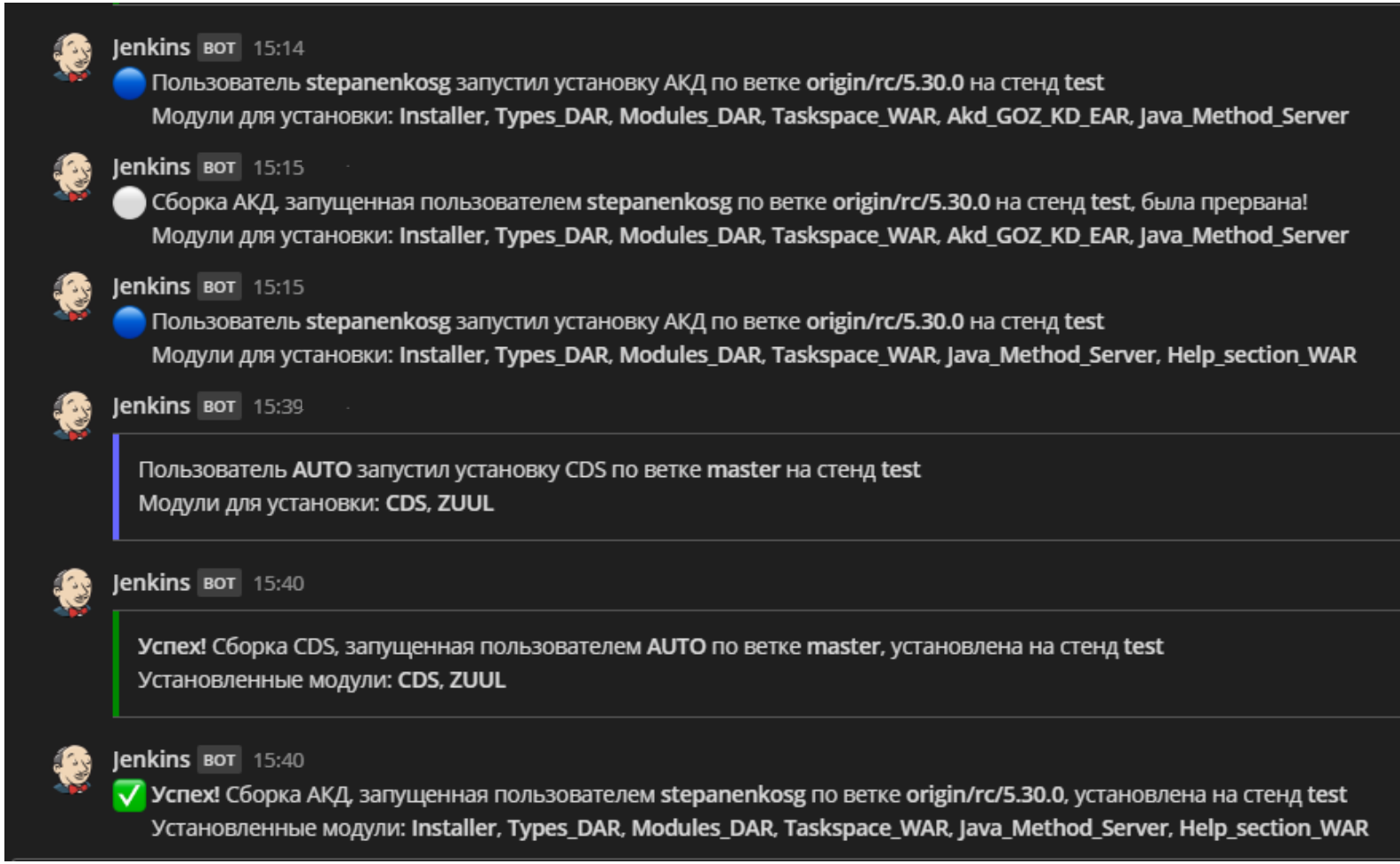
- Мы же за автоматизацию
- Экономия времени
- Удобство в установке на несколько контуров + разный состав релиза
- Знаем начало и окончание установки



Зачем знать о начале и завершении установки?

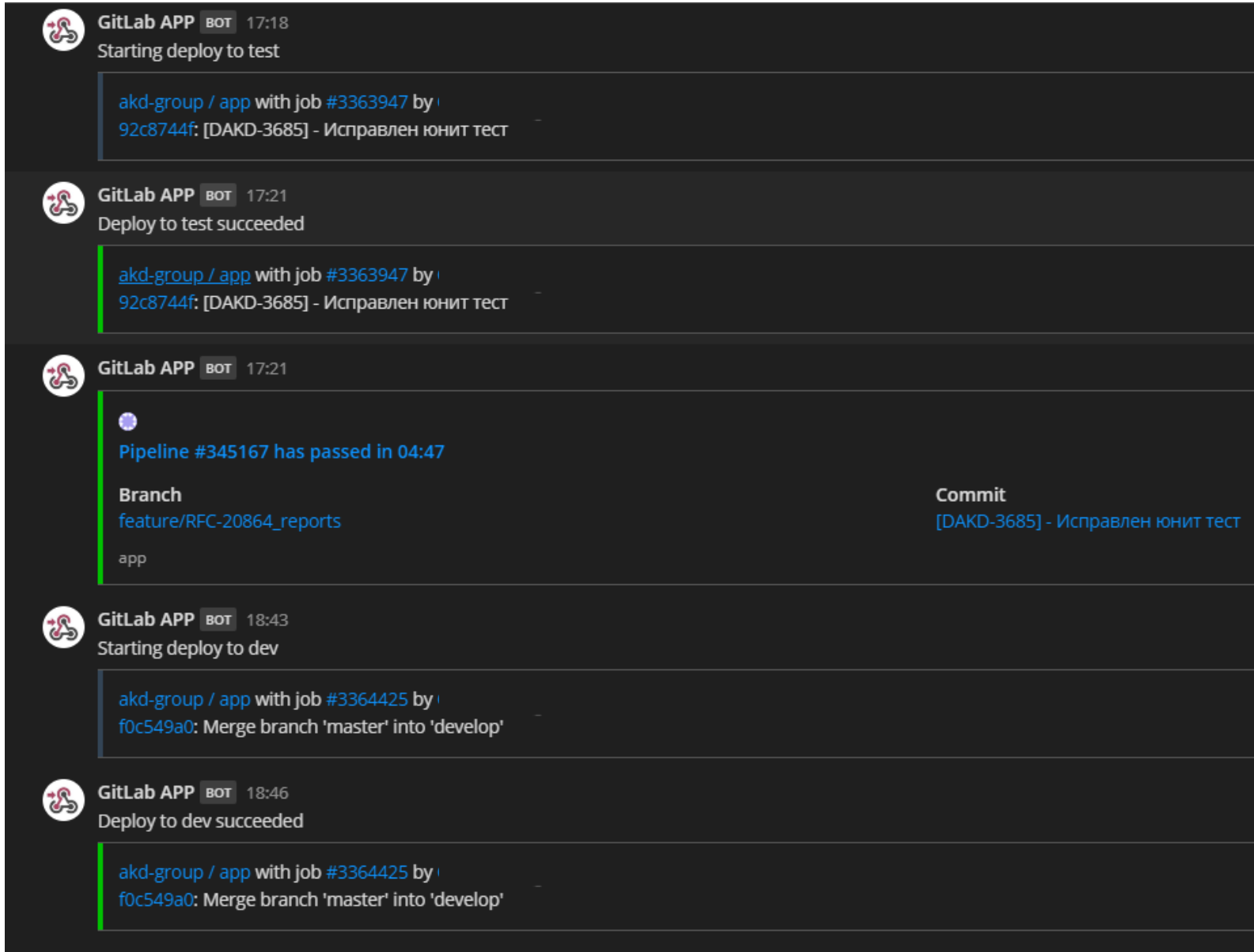
- Тестирование получает информацию о готовности стенда

- 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



GitLab APP BOT 13:07
Deploy to prep succeeded

akd-group / app with job #3356426 by
9c7e7b71: Merge branch 'help_5.40' into 'rc/5.40.0'

GitLab APP BOT 13:07
Starting deploy to prep

akd-group / app with job #3356447 by
9c7e7b71: Merge branch 'help_5.40' into 'rc/5.40.0'

GitLab APP BOT 15:59
Deploy to dev succeeded

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

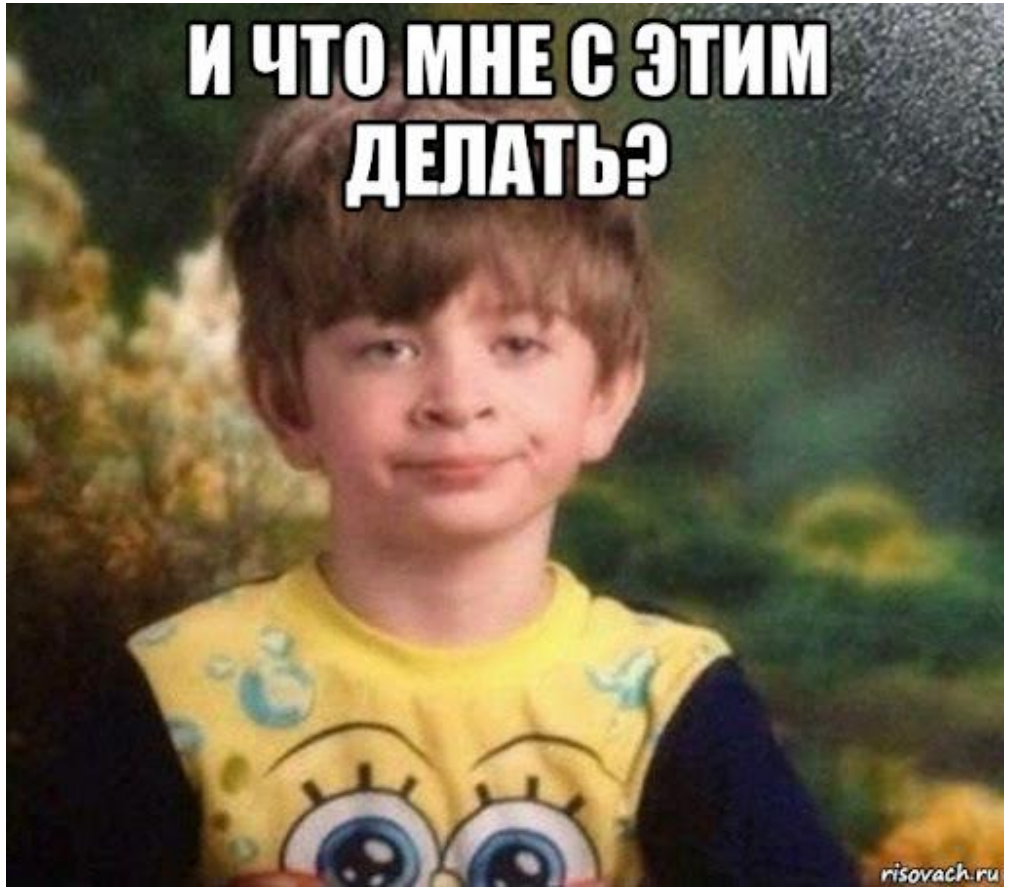
GitLab APP BOT 11:45

Pipeline #344316 has passed in 10:24

Branch
help_5.40

Commit
DAKD-3778

app



GitLab APP BOT 15:59
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 APP BOT 16:23
Deploy to dev failed

akd-group / app with job #3343897 by
603fe187: Merge branch 'feature/RFC-21526_manual_load_mortgage_stage_1' into 'rc/5.40.0'

GitLab APP BOT 12:13
Starting deploy to dev2

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

GitLab APP BOT 14:16
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'

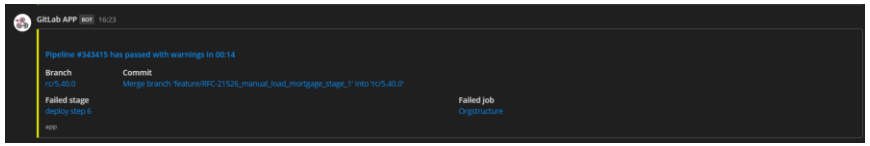
GitLab APP BOT 17:29

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 APP BOT 14:06
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

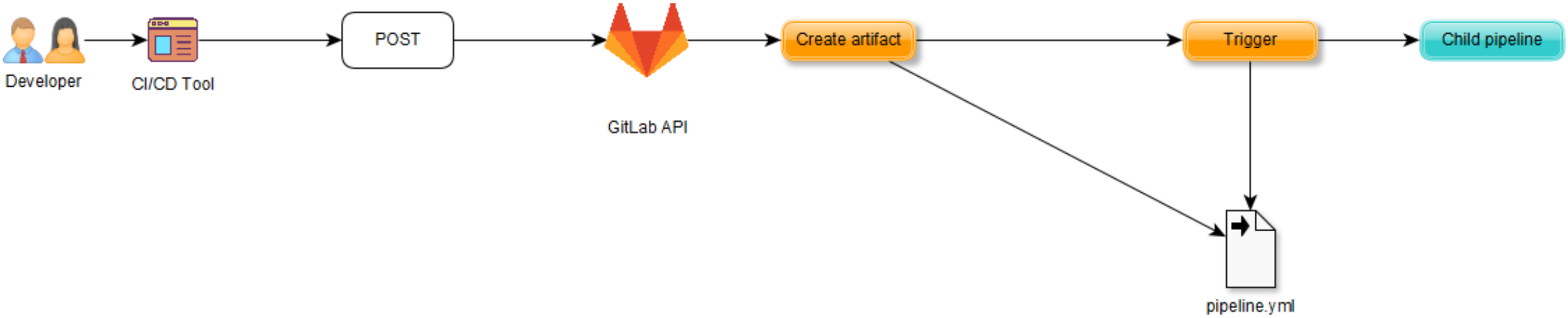
Что нам нужно знать для автоматизации

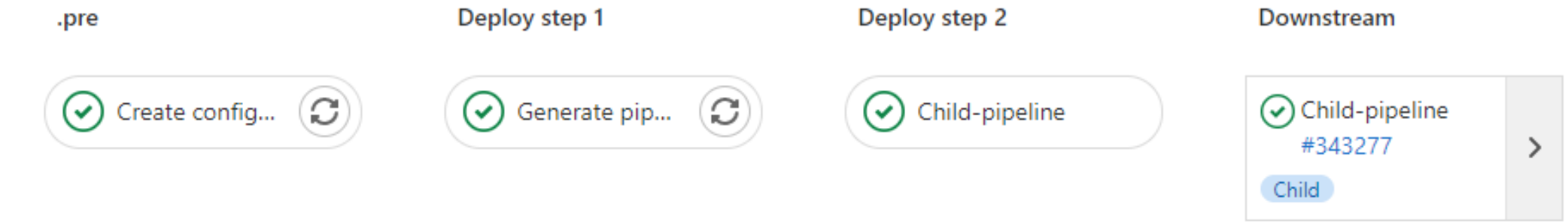
Состав релиза

Ветка или сборка к деплою

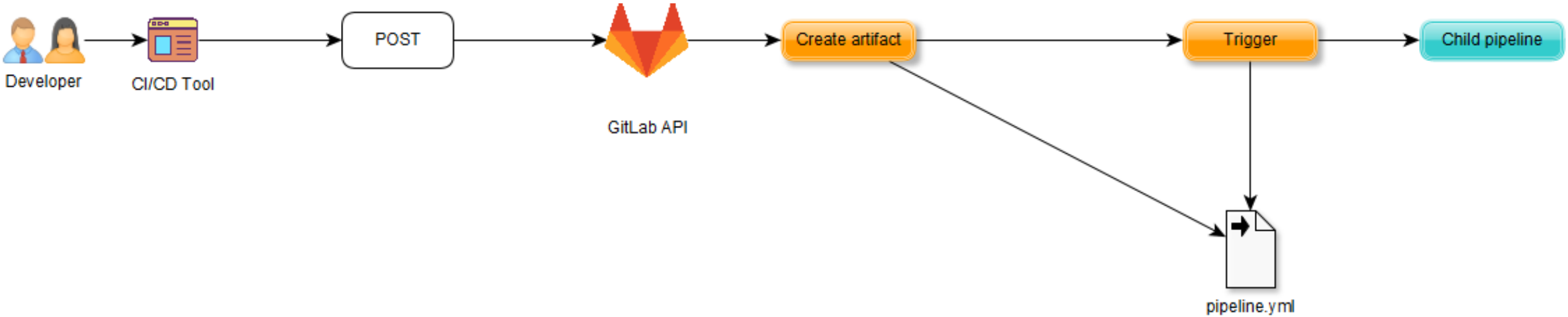
Контур для деплоя

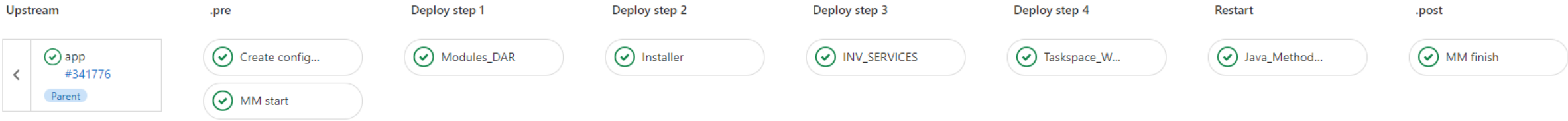
Автоматизируем установку



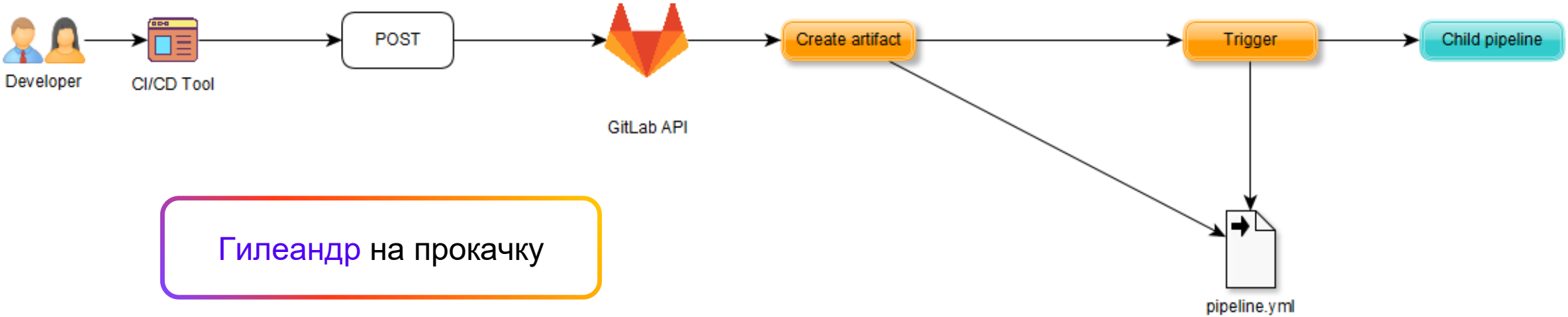


Автоматизируем установку





Автоматизируем установку



Гилеандр на прокачку

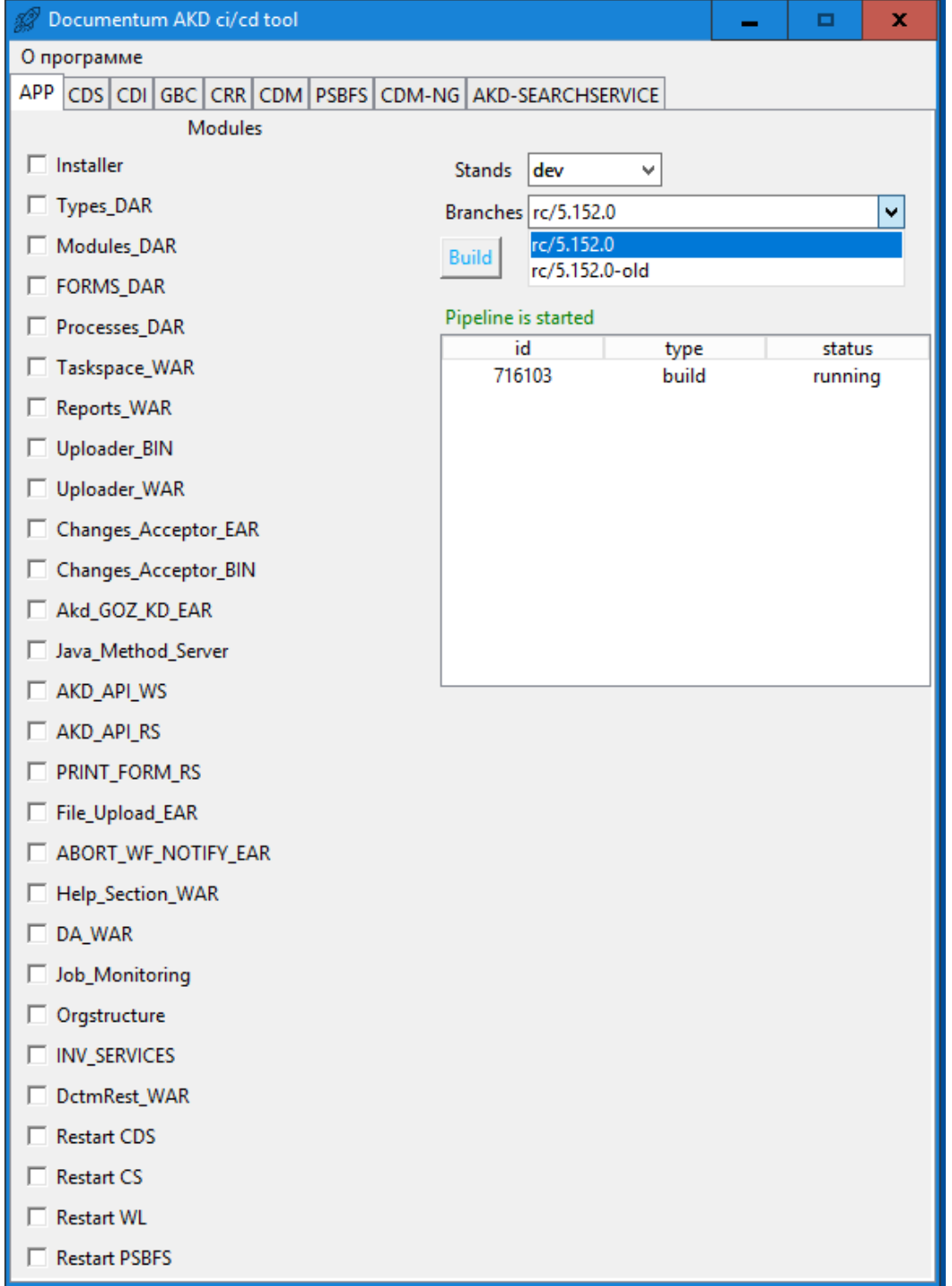
Мы встроили в pipeline еще один pipeline, чтобы вы могли запускать pipeline пока работает pipeline!

CI/CD tool

Реализация:

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

Сборка Гелиандр CI + Нейрис



CI/CD tool

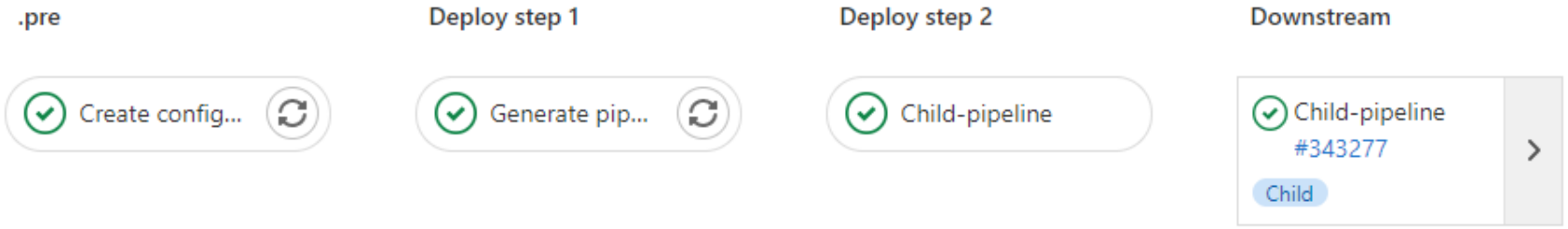
Create a new pipeline

```
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

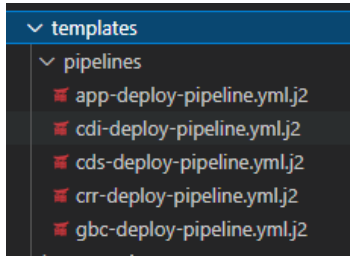
CI/CD tool



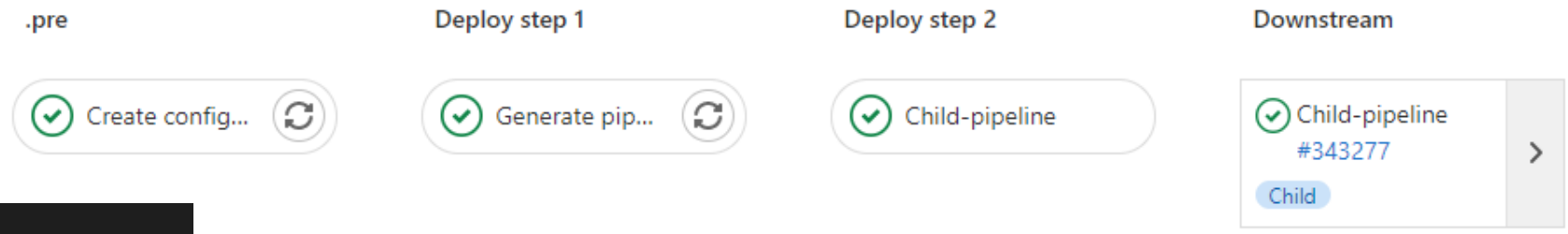

```yaml
Generate pipeline:
  stage: deploy step 1
  tags:
    - build01
  extends:
    - .devops_scripts
    - .generate_deploy_pipeline
  script: python devops-scripts/generate-pipeline.py
  artifacts:
    paths:
      - ci-temp/deploy-pipeline.yml
    expire_in: 1 week
```

DEPLOY_PARAMETERS : {"branch": "rc/5.42.0", "stand": "dev", "modules": ["installer", "taskspace", "help_section"]}

```yaml
---      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



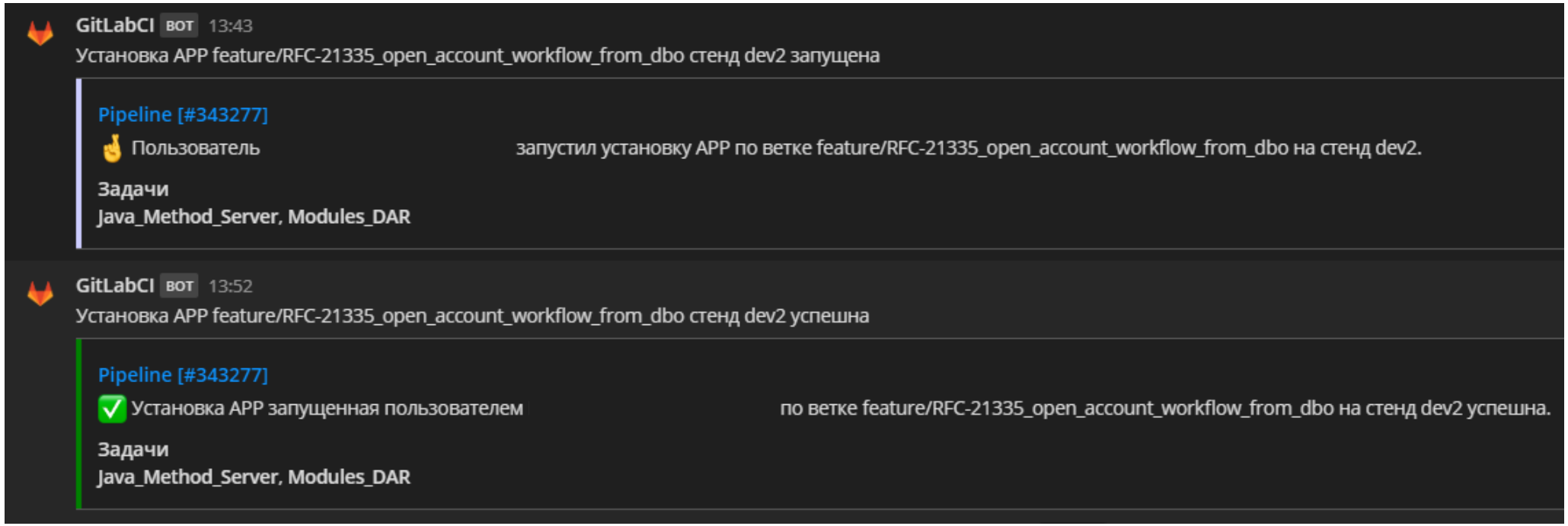

```yaml
---
include:
  - project: 'AKD_DEVOPS/gitlab-ci'
    ref: develop
    file: '/general/.general-ci.yml'
  - project: 'AKD_DEVOPS/gitlab-ci'
    ref: develop
    file: '/app/.extensions-ci.yml'

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
```

```yaml
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



- 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

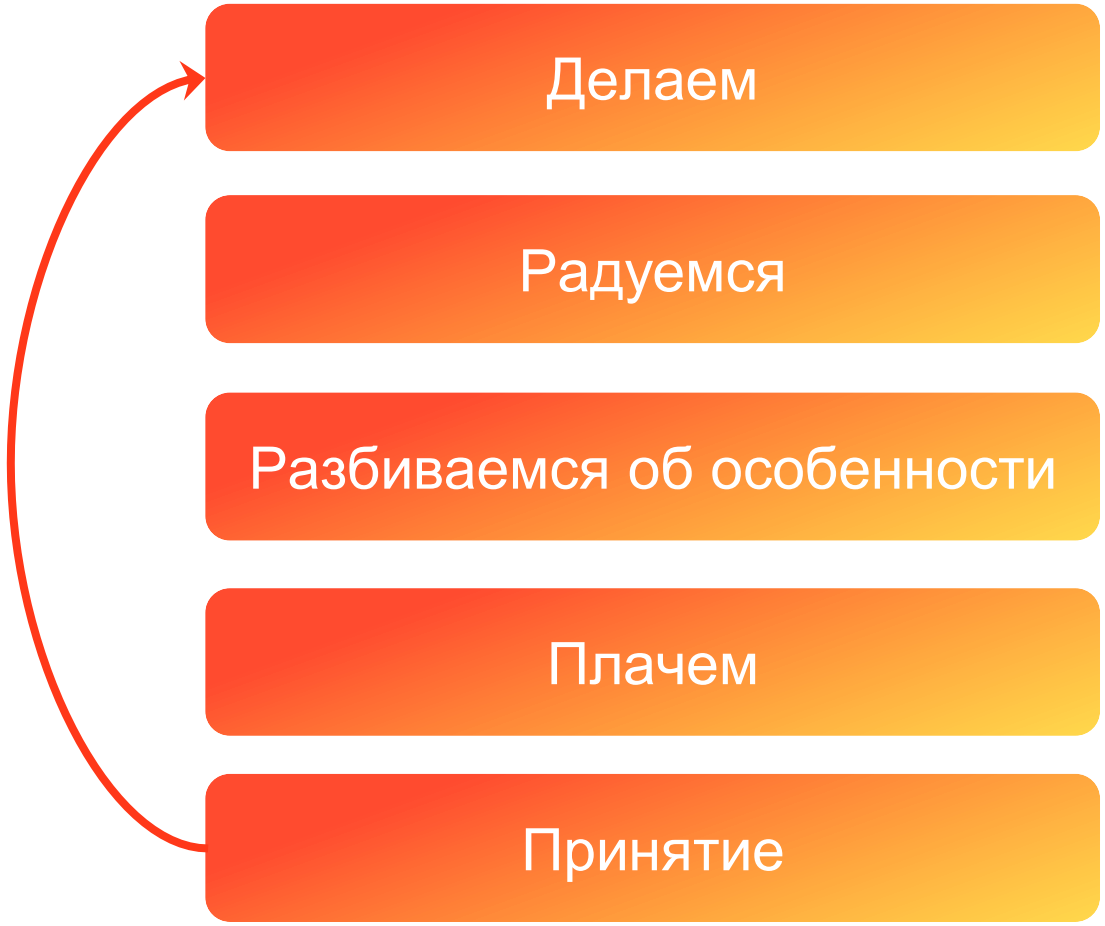
Томар

DAR-modules

Microservices

Installer

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

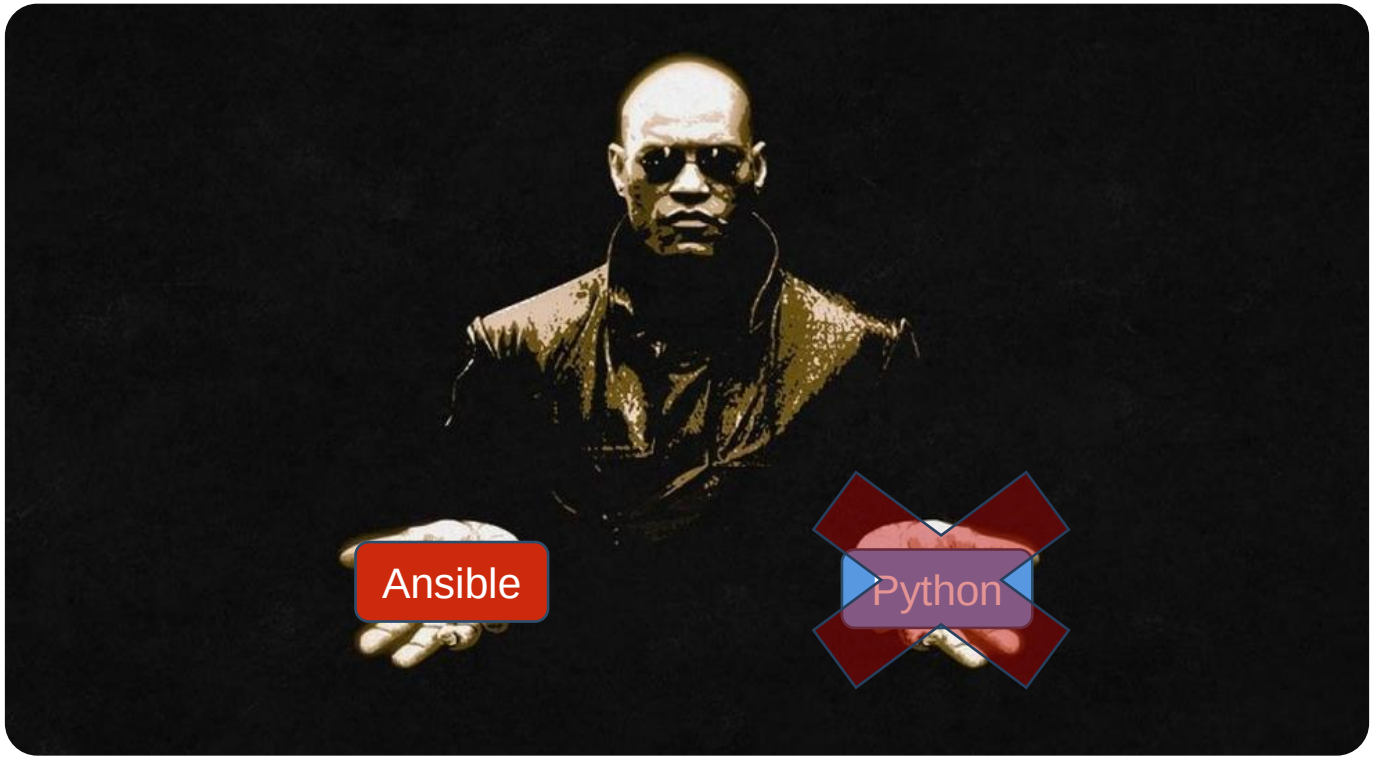


Изучаем проект и отмечаем узкие места

1. Версии Java различаются на сборочном и целевых серверах
2. Для каждого контура был отдельный Composer Headless (это компонент разработки)
3. На сборочном сервере располагались конфигурационные файлы для каждого контура
4. Конфигурационные данные для сервисов хранились в персональном pom-файле, что приводило к дублированию
5. Часть артефактов при сборке привязывалась к контуру
6. Артефакты не самодостаточны
7. Установка 1го DAR-модуля могла занимать 40 минут
8. Секреты!!!

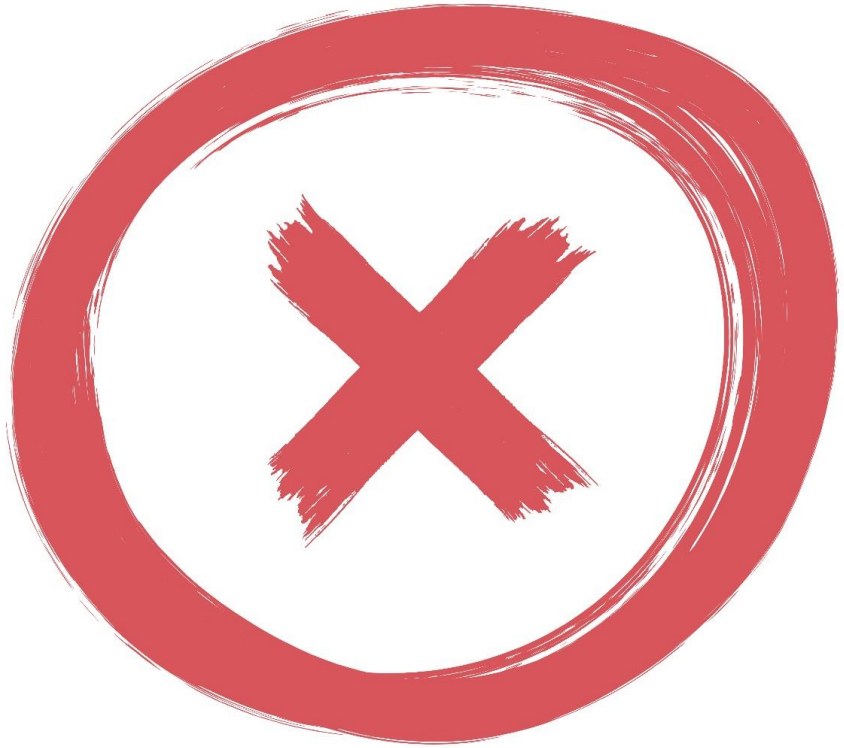
План

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

WebLogic

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

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

```yaml
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 конутров
- 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


Благодарю за внимание!

Мои заметки
и наработки

GitHub Profile

Telegram