DGTL
Raiffeisen

Один пайплайн, чтобы править всеми!

Харченко Евгений
Райффайзен Банк

Кто говорит

О себе

- В университете занимался мелкой автоматизацией, копался с Windows Server
- Работал в организации, представляющей BaaS-решения для UK (United Kingdom) в роли L1-специалиста
- В Райффайзен Банке начинал с инженера техподдержки ServiceDesk
- После, в банке работал как Engineer -> Leading Engineer
- Третий год как Senior Community Lead DevOps
- Platform Owner/TechLead команды DevOps Enabling Team

Про что доклад?

Что ~~нового~~ узнаете

- Про пайплайны и YAML
- Проблемы унификации и локализации
- Создание внутреннего продукта для крауд сорсинга -> inner source
- Реализация DevOps практик и шаринг опыта через продукт

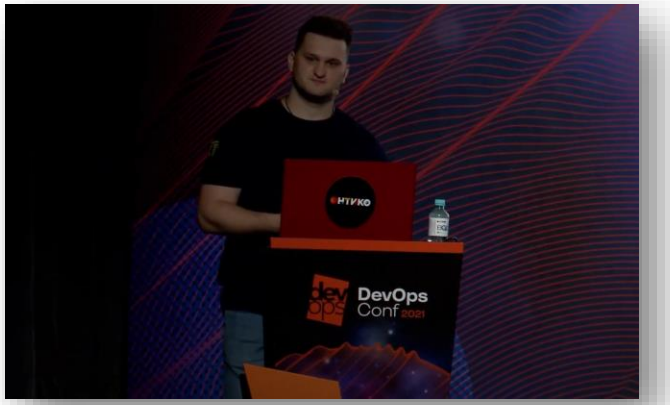
Немного контекста о проблеме

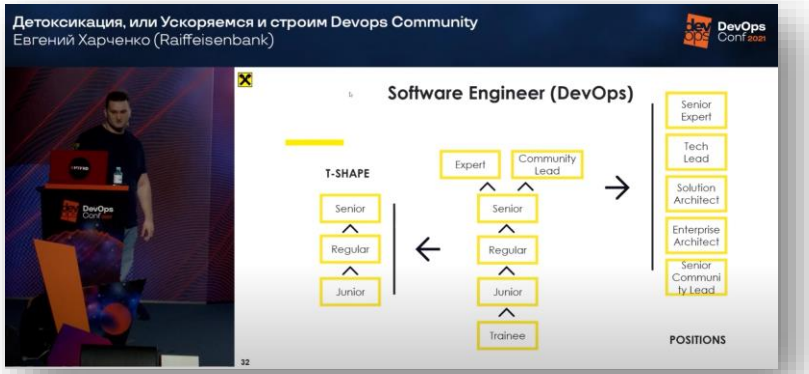
Про сообщество

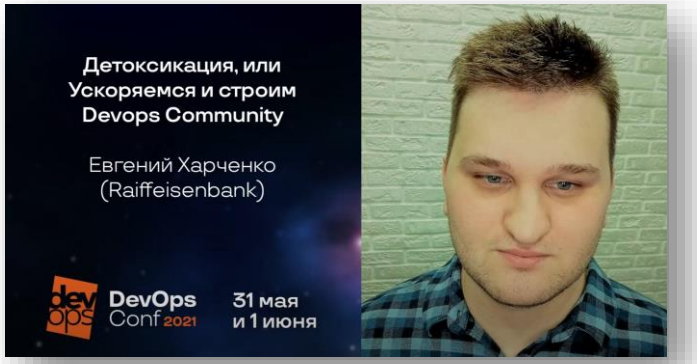
Контекст

	2021	2022	2023
Всего участников	300+	580+	1050+
Из них инженеров	150+	~280	~350

Об этом подробнее можно посмотреть здесь







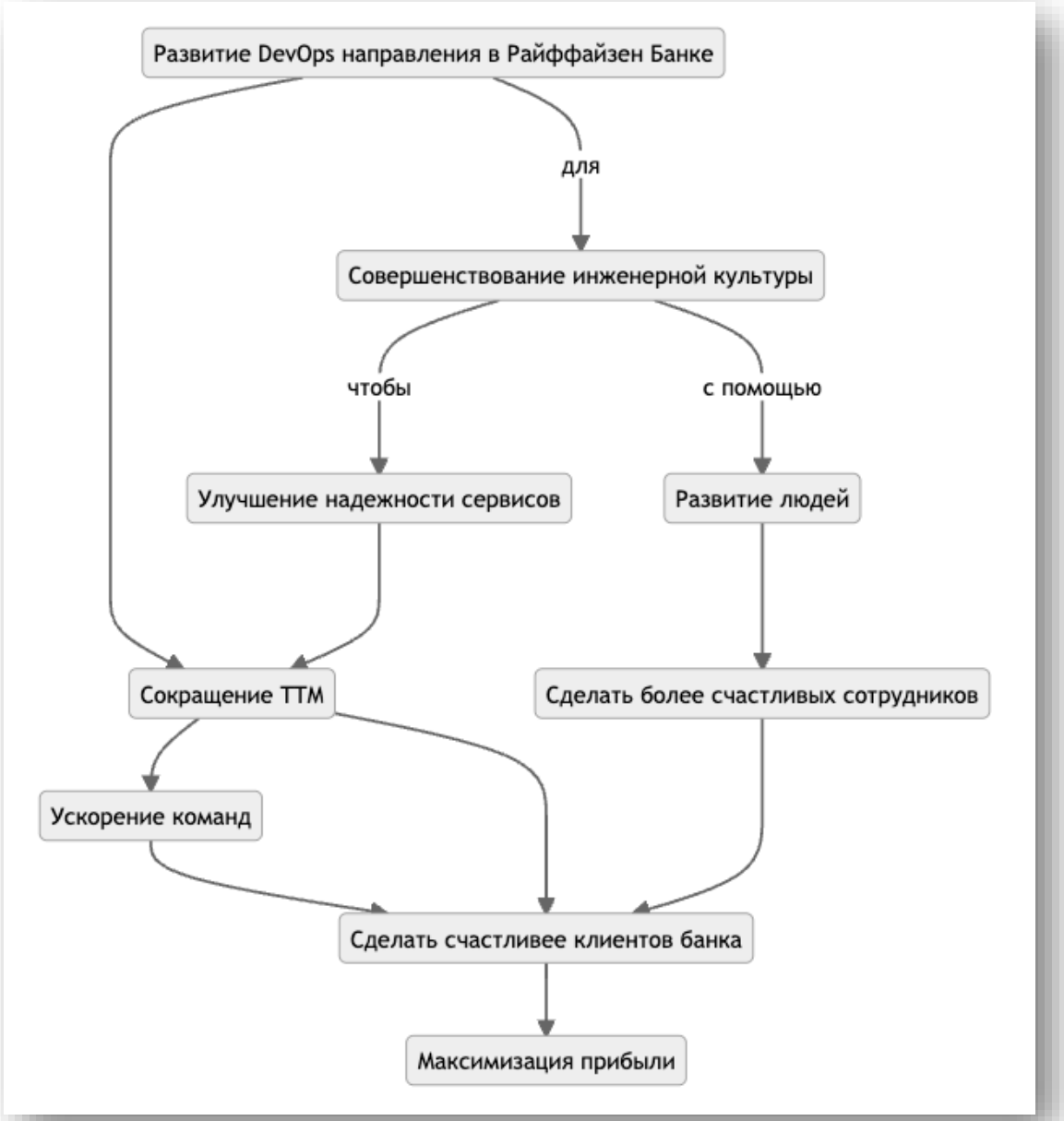
Цель сообщества

К чему мы стремимся

Развитие DevOps направления в Райффайзен Банке

Ускорение команд и их продуктов

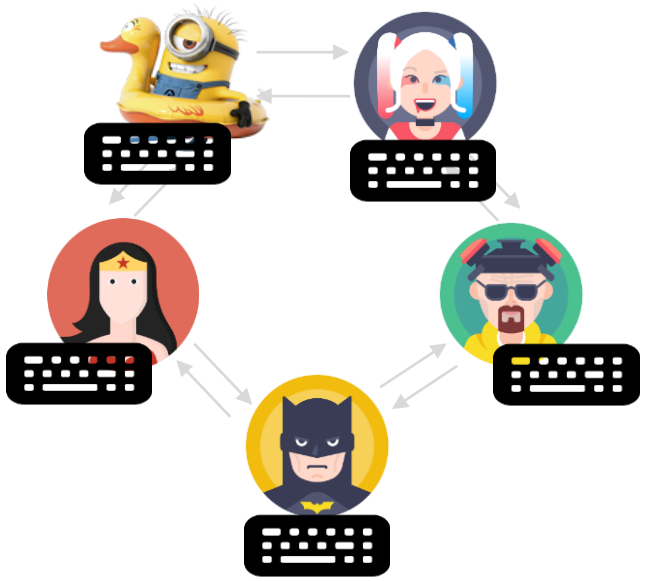
Совершенствование инженерной культуры



Про работу в сообществе

Немного контекста

Инженеры из сообщества



виртуальная команда

delivery

DevOps Community

О задачах

Как появляются артефакты работы из задач

delivery

- Лучшие практики
- Автоматизация рутинных задач
- Исследования и внедрения инноваций
- Создание продуктов
- Шэринг опыта
- Развитие людей

Продукт сообщества

Становление продукта

- Несколько экспертов разных доменов решали похожую задачу
- Эксперты были приглашены в рабочую группу для решения одной задачи
- Разработка совместных правил о ведении продукта и принципах вклада
- Начало работы над продуктом и общего дела для всех

DevOps is about communication

Резюмируем

Подытожим вступление

- Есть сообщество с целями и задачами для реализации
- Внедрение и масштабирование DevOps практик в организации
- Создание продукта для закрытия целей и решения общих задач с помощью inner-source

А теперь о проекте

The Way of CI/CD

Интро

The way of CI/CD - это проект, из частей которого вы можете собрать свой собственный пайплайн, используя различные участки кода или же подключая этапы целиком

github.com/Raiffeisen-DGTL/The-Way-of-CICD-Open-Source-Edition

Область применения

Какую проблему решает продукт

- Сократить время всей организации на создании конвейеров (пайплайнов)
- Передавать знания о лучших практиках и подходах
- Улучшает технические возможности в конченом счете влияющие на коммерческий и не коммерческий уровень производительности организации

TL;DR

QuickStart - Как начать использовать проект

include

```yaml
include:
  - project: 'devops_community/the-way-of-cicd'
    ref: 'master'
    file:
      - '/build/v1/build.yml'
```

Используется для включения внешних YAML-файлов в основной .gitlab-ci.yml файл

управление обновлениями

extends

```yaml
build-via-kaniko:
  stage: build
  extends:
    - .build_kaniko
```

Используется для наследования и расширения задач и шагов из других задач

управление настройками

модульность

!reference [tags]

```yaml
build-via-kaniko:
  stage: build
  image:
    name: !reference [.build_kaniko, image, name]
    entrypoint: !reference [.build_kaniko, image, entrypoint]
  before_script:
    - !reference [.build_kaniko, before_script]
  script:
    - !reference [.build_kaniko, script]
```

Используется для указания тэга YAML !reference, чтобы выбрать определенные секции ключевых слов из других разделов задач и повторно использовать их.

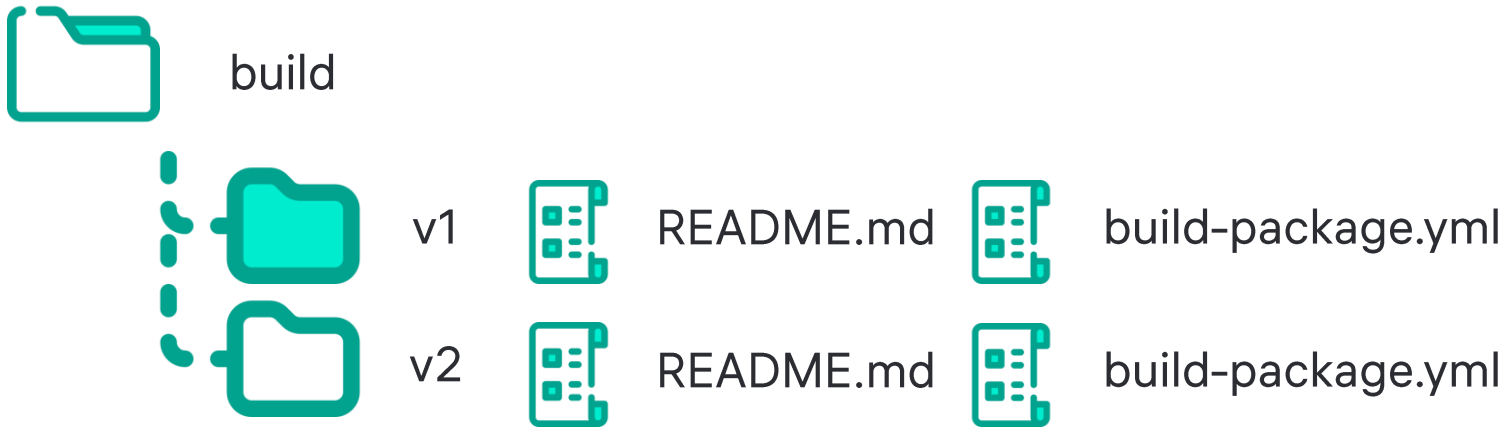
единообразие

переиспользование

Архитектура

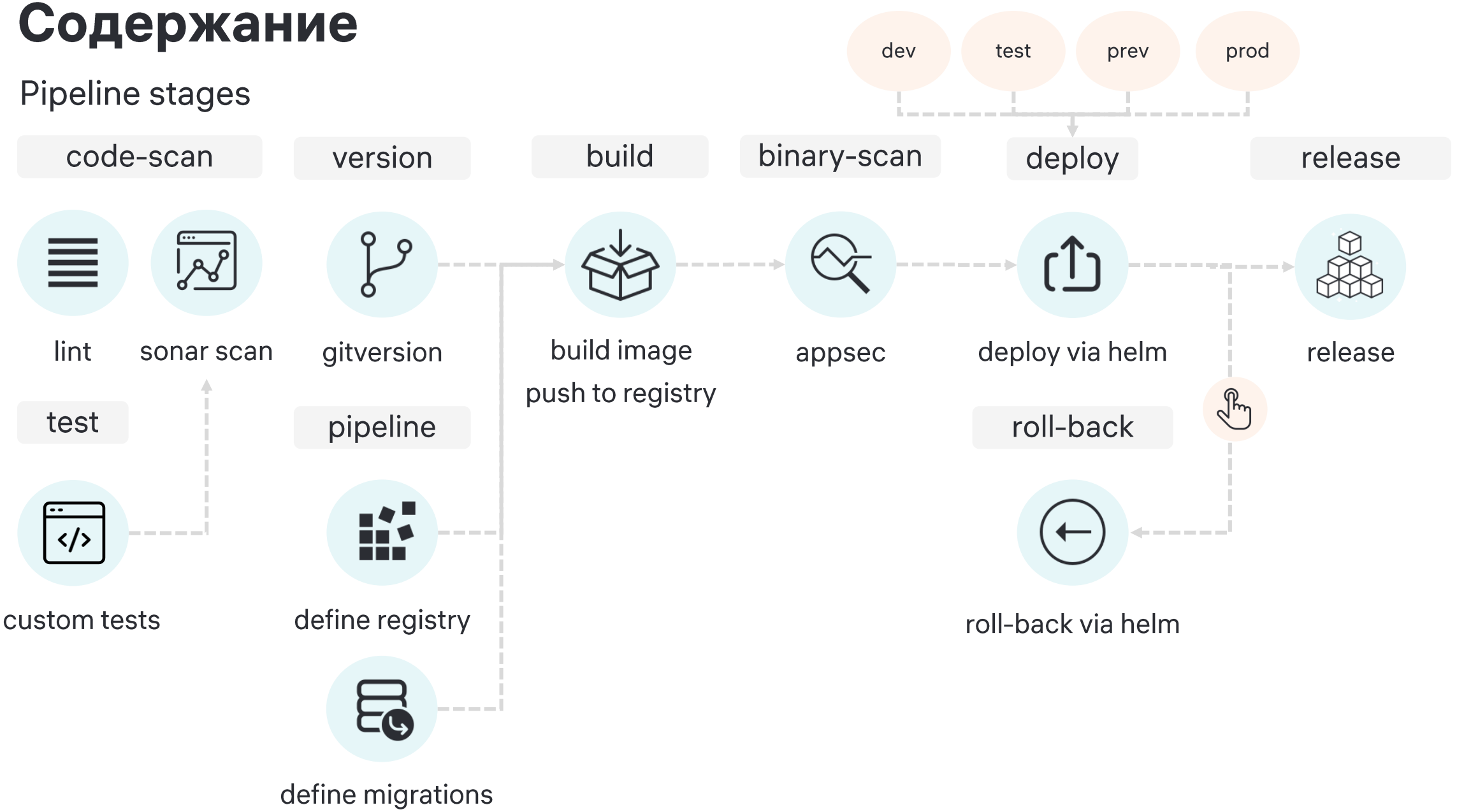
Структура проекта

- Мы используем версионирование с помощью создания новых версий директорий
- Стремимся обеспечивать предсказуемые обновления
- Редактируем существующий функционал исключительно в целях баг-фиксов



Содержание

Pipeline stages



Code Scan

Сканирование кода - SAST

code-scan

- У нас есть различные варианты сканирования для JVM
- Мы можем сканировать любой язык
- Для универсального сканирования нужно создать sonar-project.properties
- Если файла со сканом нет, то в лог будет выведен пример с содержанием и exit 1

```
.sonarqube-scan:
  stage: scan
  image: sonarsource/sonar-scanner-cli:5.0
  script:
    - |
      if [[ -e sonar-project.properties ]]
      then
        sonar-scanner
      else
        echo -n "Please create the sonar-project.properties file to the root of your project.
                 Here is an example of confiugration file for Python:
                 ---
                 sonar.projectKey=${env.SONAR_PROJECT_KEY}.${env.CI_PROJECT_NAME}
                 sonar.projectName=${env.SONAR_PROJECT_KEY}.${env.CI_PROJECT_NAME}
                 sonar.python.coverage.reportPaths=coverage.xml
                 sonar.python.xunit.reportPath=junit.xml
                 sonar.sources=.
                 sonar.inclusions=**/*.py
                 sonar.exclusions=**/migrations/*.py,scripts/**,bin/**,alembic/**,tests/**,docs/**,logo/**
                 sonar.host.url=${env.SONAR_HOST_URL}
                 sonar.login=${env.SONAR_TOKEN}
                 sonar.gitlab.commit_sha=${env.CI_COMMIT_SHA}
                 sonar.gitlab.project_id=${env.CI_PROJECT_ID}
                 sonar.branch.name=${env.CI_COMMIT_BRANCH}
                 sonar.qualitygate.wait=true
                 ---"
        exit 1
      fi
```

Test

Про тесты

test

- Все тесты реализованы для JVM стэка
- В основе реализации используется maven/gradle
- Интегрирована работа с allure
- Реализована публикация результатов в gitlab pages

```yaml
include:
  - 'test/v1/autotests/autotest.yml'

stages:
  - prepare
  - build
  - test
  - test_report
  - deploy-pages
  - send_notification

test_env_prepare:
  stage: prepare
  extends:
    - .test_env_prepare

build_tests:
  stage: build
  extends:
    - .build_tests

run_autotests_java:
  stage: test
  extends:
    - .run_autotests_java

create_autotests_report:
  stage: test_report
  extends:
    - .create_autotests_report

pages:
  stage: deploy-pages
  extends:
    - .pages

send_report_to_mattermost:
  stage: send_notification
  extends:
    - .send_report_to_mattermost
```

Version

Про версионирование

version

github.com/semver/semver

Fork 746 Star 6.1k

GitVersion

From git log to
SemVer in no time

github.com/GitTools/GitVersion

Fork 590 Star 2.3k

```yaml
.version_git:
  stage: version
  image: gittools/gitversion:6.0.0-alpine.3.17-7.0
  variables:
    GIT_STRATEGY: clone
    GIT_DEPTH: 0
  before_script:
    - |
      if [[ ! -f "GitVersion.yml" ]]
      then
        cat <<EOF > GitVersion.yml
        mode: Mainline
        increment: Inherit
        tag-prefix: '[vV]?'
        minor-version-bump-message: 'branch\s?''(feature|minor).*into\s?''(master|main)'''
        patch-version-bump-message: 'branch\s?''(fix|bugfix|hotfix|patch).*into\s?''(master|main)'''
        no-bump-message: 'from\s(none|skip|doc)'
        legacy-semver-padding: 0
        build-metadata-padding: 0
        commits-since-version-source-padding: 0
        commit-message-incrementing: Disabled
        branches: {}
        ignore:
          sha: []
      EOF
      fi
  script:
    - |
      if [ -z "$CI_COMMIT_TAG" ]
      then
          if [[ -f "majorVersion" ]]
          then
            MAJOR_VERSION=$(cat majorVersion | grep MAJOR_VERSION | cut -d '=' -f 2)
          else
            MAJOR_VERSION=0
          fi
      fi
    - |
      if [ -z "$CI_COMMIT_TAG" ]
      then
          git fetch --all
          if (( ${MAJOR_VERSION} > $(gitversion /showvariable Major) ))
          then
              VERSION=${MAJOR_VERSION}.0.0
          else
              VERSION=$(gitversion /showvariable MajorMinorPatch)
          fi
      else
          VERSION=${CI_COMMIT_TAG}
      fi
    - |
      if [ -z "$CI_COMMIT_TAG" ]
      then
          if [ "$CI_COMMIT_BRANCH" == "$CI_DEFAULT_BRANCH" ]
          then
            echo "VERSION=${VERSION}" > version.env
          else
            echo "VERSION=$VERSION-$CI_COMMIT_REF_SLUG-$CI_COMMIT_SHORT_SHA" > version.env
          fi
      else
          echo "VERSION=${VERSION}" > version.env
      fi
      sed -e 's/=/: /' version.env
  artifacts:
    reports:
      dotenv:
        - ./version.env
```

Pipeline – .time_tracker

Про логирование времени

pipeline

- В GitLab нет логирования времени в пайплайне
- Переиспользовали скрипт от сообщества внутри банка
- Внедрили в централизованный пайплайн

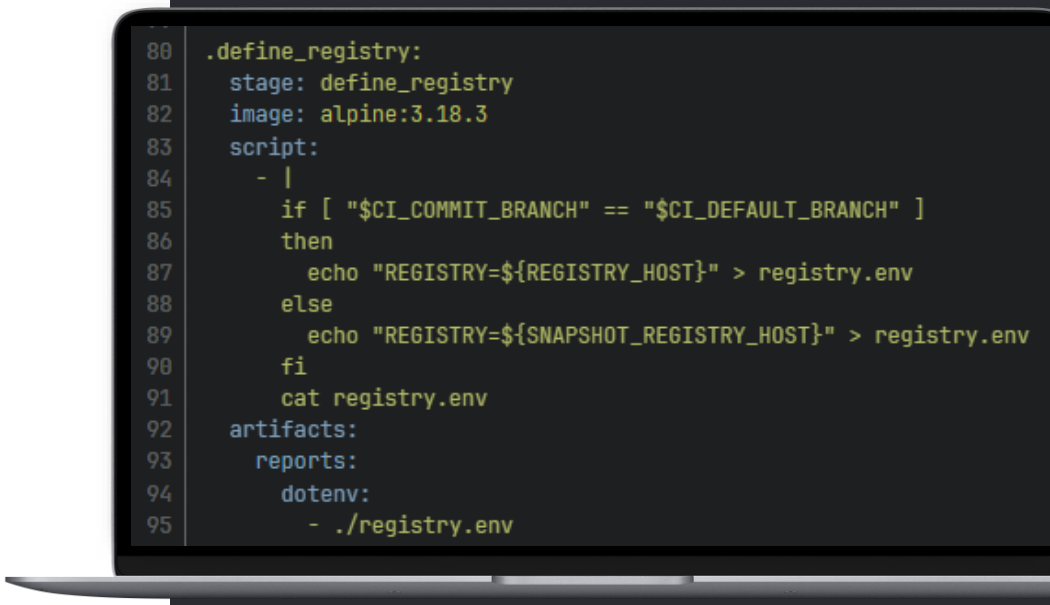
```
.time_tracker:
  image: alpine:3.18.3
  before_script:
    - |
      #!/bin/bash
      set -e
      run () {
        PREFIX=$(if [[ "$DRY_RUN" == "1" ]]; then echo "dry run"; else echo "run"; fi)
        echo "$PREFIX: $*"
        if [[ "$DRY_RUN" == "1" ]]; then
          return 0
        fi
        eval "$@"
      }
      use_printf=`printf "%(true)T\n" -1 2>/dev/null` || use_printf=false
      use_awk=`awk 'BEGIN{print(strftime("true"));}'` || use_awk=false
      echo "logging: use_printf=$use_printf; use_awk=$use_awk"
      if $use_printf ; then
        logging_pipe=$$.logging
        mkfifo $logging_pipe
        (set +x;while read -r line;do printf "%(%H:%M:%S)T  %s\n" -1 "$line";done) <$logging_pipe&
        exec >$logging_pipe 2>&1
        rm $logging_pipe
      elif $use_awk ; then
        logging_pipe=/tmp/$$.logging
        mkfifo $logging_pipe
        awk '{printf("%s  %s\n",strftime("%H:%M:%S"),$0);}' <$logging_pipe&
        exec >$logging_pipe 2>&1
        rm $logging_pipe
      else
        echo "Can't setup timestamped logging."
      fi
      echo "logging: Started."
```

Pipeline - .define_registry

Про работу с registry

pipeline

- Мы определяем registry для различных веток
- SNAPSHOT репозиторий - для всех кроме $CI_DEFAULT_BRANCH
- RELEASE репозиторий - для CI_DEFAULT_BRANCH



```
.define_registry:
  stage: define_registry
  image: alpine:3.18.3
  script:
    - |
      if [ "$CI_COMMIT_BRANCH" == "$CI_DEFAULT_BRANCH" ]
      then
        echo "REGISTRY=${REGISTRY_HOST}" > registry.env
      else
        echo "REGISTRY=${SNAPSHOT_REGISTRY_HOST}" > registry.env
      fi
      cat registry.env
  artifacts:
    reports:
      dotenv:
        - ./registry.env
```

Pipeline – .get_certificate & .get_vault_creds

Про логирование времени и получение секретов

pipeline

Возможность автоматизировано создавать сертификаты

Требует наличие Vault PKI

Получение секретов из Vault в пайплайне

Секреты передаются между job с помощью cache

```
.get_vault_creds:
  image: vault:1.13.3
  variables:
    LOGIN_FIELD: "login"
    PASSWORD_FIELD: "password"
  before_script:
    - export VAULT_TOKEN="$(vault write -field=token auth/jwt/login role=gitlab-role jwt=$CI_JOB_JWT)"
    - echo "LOGIN=`vault kv get -field=$LOGIN_FIELD $VAULT_PATH_TO_SECRET`" > vault.env
    - echo "PASSWORD=`vault kv get -field=$PASSWORD_FIELD $VAULT_PATH_TO_SECRET`" >> vault.env
  cache:
    - key: $CI_PIPELINE_ID
      paths:
        - vault.env
```

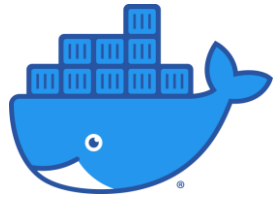
```
.get_certificate:
  variables:
    VAULT_ADDRESS: "https://path.to.your.pki:8200"
    PKI_ROLE_ID: "$PKI_TEST_ROLE_ID"
    PKI_SECRET_ID: "$PKI_TEST_SECRET_ID"
    PKI_COMMON_NAME: "test.example.com"
    PKI_ALT_NAMES: "test2.example.com,test3.example.com"
    PKI_TTL: "8760h"
    PKI_ROLE_NAME: "$PKI_ROLE_NAME"
  image: msvechla/vaultbot:1.13
  script:
    - mkdir ./vaultbot/
    - /root/vaultbot
      --vault_addr=${VAULT_ADDRESS}
      --vault_auth_method=approle
      --vault_app_role_role_id=${PKI_ROLE_ID}
      --vault_app_role_secret_id=${PKI_SECRET_ID}
      --pki_mount=pki_int
      --pki_role_name=$PKI_ROLE_NAME
      --pki_common_name=${PKI_COMMON_NAME}
      --pki_alt_names=${PKI_ALT_NAMES}
      --pki_ttl=${PKI_TTL}
      --pki_renew_time=720h
      --pki_private_key_format=pkcs8
      --pki_cert_path=./vaultbot/certificate.crt
      --pki_privkey_path=./vaultbot/private.key
    - ls -la ./vaultbot/
  artifacts:
    paths:
      - ./vaultbot/
```

Build

Про сборку

build

- Для сборки мы используем kaniko
- Есть сборки с помощью jib & gradle
- Также имеется и билд с помощью DinD
- Сборка и упаковка helm chart



```yaml
.build_kaniko:
  stage: build
  image:
    name: gcr.io/kaniko-project/executor:1.7.0
    entrypoint: [""]
  variables:
    REGISTRY_HOST: ""
    REGISTRY_PATH: ""
    DOCKERFILE: "Dockerfile"
    DESTINATION: "$REGISTRY_HOST/$REGISTRY_PATH/${IMAGE_NAME}:$IMAGE_TAG"
    IMAGE_TAG: "latest"
  before_script:
  script:
    - echo "{\"auths\":{\"$REGISTRY_HOST\":{\"username\":\"$REGISTRY_USER\",\"password\":\"$REGISTRY_PASSWORD\"}}}" > /kaniko/.docker/config.json
    - >
      CMD="/kaniko/executor
      --context $CI_PROJECT_DIR
      --dockerfile $CI_PROJECT_DIR/$DOCKERFILE
      --destination $DESTINATION
      --build-arg REGISTRY_USER=$REGISTRY_USER
      --build-arg REGISTRY_PASSWORD=$REGISTRY_PASSWORD
      $DOCKER_BUILD_ARGS"
    - eval $CMD
```

Deploy

Про развертывание

deploy

- Самый распространённый инструмент для развертывания – helm
- Реализован классический деплой с помощью helm
- Реализован деплой с помощью gitlab-agent

```yaml
.deploy:
  ### Image for deploy
  image: alpine/k8s:1.28.0
  ### Fetch kubeconfig
  fetch_kubeconfig:
    - export KUBECONFIG="$K8S_CONFIG"
  ### Delpoy via kubectl
  kubectl:
    - kubectl apply -f ${YAML_TO_DEPLOY} ${DEPLOY_ARGS}
  ### Deploy via helm
  helm:
    - helm upgrade --install $RELEASE_NAME $CHART_TO_DEPLOY -n $NAMESPACE $HELM_ARGS
```

```yaml
.deploy-with-helm:
  stage: deploy
  image: alpine/k8s:1.28.0
  variables:
    NAMESPACE: "$CI_PROJECT_NAME"
    RELEASE_NAME: "$CI_PROJECT_NAME"
    CHART_TO_DEPLOY: "helm-chart-go-web-server"
    HELM_ARGS: >-
      --wait
  script:
    - CMD="helm upgrade --install $RELEASE_NAME $CHART_TO_DEPLOY -n $NAMESPACE $HELM_ARGS"
    - eval $CMD
```

```yaml
.deploy-with-helm-by-gitlab-agent:
  stage: deploy
  image: alpine/k8s:1.28.0
  variables:
    NAMESPACE: "$CI_PROJECT_NAME"
    RELEASE_NAME: "$CI_PROJECT_NAME"
    GITLAB_AGENT_PROJECT: "YOUR_PROJECT_WITH_GITLAB_AGENT"
    KUBERNETES_CLUSTER_NAME: "NAME_OF_YOUR_KUBERNETES_CLUSTER" # Infrastructure -> Kubernetes clusters
    CHART_TO_DEPLOY: "helm-chart-go-web-server"
    HELM_ARGS: >-
      --wait
      --atomic
  before_script:
    - kubectl config use-context ${GITLAB_AGENT_PROJECT}:${KUBERNETES_CLUSTER_NAME}
  script:
    - !reference [.deploy, helm]
```

Roll-back

Про откат

roll-back

- Реализован roll-back с помощью helm

```
.roll-back:
  ### Image for roll-back
  image: alpine/k8s:1.28.0
  ### Fetch kubeconfig
  fetch_kubeconfig:
    - export KUBECONFIG="$K8S_CONFIG"
  ### Roll-back script
  helm-roll-back:
    - helm rollback $RELEASE_NAME -n $NAMESPACE

### Roll-back for extends
.roll-back-helm:
  stage: roll-back
  image: alpine/k8s:1.28.0
  variables:
    NAMESPACE: "$CI_PROJECT_NAME"
    RELEASE_NAME: "$CI_PROJECT_NAME"
  script:
    - helm rollback $RELEASE_NAME -n $NAMESPACE
```

Release

Про релиз в gitlab

release

- Создание сущности release в gitlab
- Создание тэга в репозитории
- Воспроизводимость версий приложения на уровне кодовой базы

```
.release:
  stage: release
  variables:
    REGISTRY_HOST: ""
    REGISTRY_PATH: ""
    VERSION: "YOUR VERSION"
    ASSET_URL: "https://${REGISTRY_HOST}/${REGISTRY_PATH}/${VERSION}"
    DESCRIPTION: "Artifact can be reached using the following link ${ASSET_URL}"
  image: registry.gitlab.com/gitlab-org/release-cli:v0.15.0 #originally was 0.4.0
  script:
    - release-cli
      --server-url $CI_SERVER_URL
      --job-token $CI_JOB_TOKEN
      --project-id $CI_PROJECT_ID
      create
      --name Release-$CI_PROJECT_NAME-$VERSION
      --description "$DESCRIPTION"
      --tag-name $VERSION
      --ref $CI_COMMIT_SHA
      --released-at $(date -u +"%Y-%m-%dT%H:%M:%SZ")
      --assets-link "{\"name\":\"${CI_PROJECT_NAME}\",\"url\":\"${ASSET_URL}\"}"
  rules:
    - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
      when: manual
    - when: never
```

О проблемах и препятствиях на пути

Проблема индивида и общества

Противостояние локального и глобального

Инженеры любят строить “свои” решения

Разбираться в “чужом” сложнее, чем сделать свое

Айтишники – люди с критическим мышлением и чаще ставят решения под сомнения

Зарабатывайте доверие

Рекламируйте решение

Концентрируйте экспертизу вокруг решения

Проблема унификации

Каждому свое?

Как решить проблему, когда каждому нужно интегрировать что то свое?

Принять факт, что всем не угодишь

Параметризируй

```
build-app-image:
  stage: build
  cache:
    <<: *poetryCache
    policy: pull
  extends:
    - .build_kaniko
  variables:
    OCI_REGISTRY: "$REGISTRY"
    IMAGE_NAME: "$CI_PROJECT_NAME"
    IMAGE_TAG: "$VERSION"
    DESTINATION: "$[redacted]SERVER/$OCI_REGISTRY/${IMAGE_NAME}:$IMAGE_TAG"
    DOCKERFILE: "Dockerfile"
    DOCKER_BUILD_ARGS: >-
      --build-arg BUILD_ENVIRONMENT=gitlab
      --skip-unused-stages
      --target app-image
  needs:
    - job: define-cache
    - job: define_registry
      artifacts: true
    - job: version
      artifacts: true
```

Проблема покрытия экспертизой

Охват

У нас есть много команд, которые нуждаются в развитии в DevOps направлении

Работа с контентом

Проведение внутреннего обучения

Проведение мероприятий

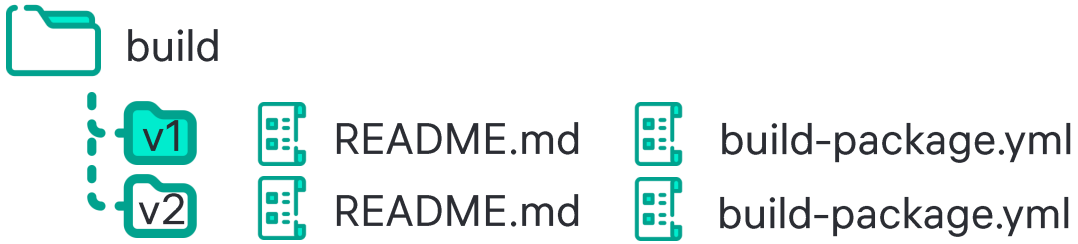
Проблема надежности

Как никому ничего не сломать?

Как гарантировать что при изменениях мы ничего не сломаем?

Мы используем версионирование с помощью создания новых директорий в v1

У нас есть соглашения для вклада в проект



С большой силой приходит и большая ответственность (с)

Заключение

+

Немного выводов и советов

Результаты в цифрах

Цифры не лгут, но лжецы пользуются формулами (с)

41+ Команд использует проект

200+ Проектов делают include

6 Топ контрибьюторов

Экономика

Трудозатраты на создание пайплайна

Затраты на создание среднестатистического пайплайна

2 Спринта **160** Часов

Затраты на создание среднестатистического пайплайна с помощью проекта

< 1 Спринта **2-40** Часов

Экономия

158-120 Часов на каждый пайплайн

Среднее число запускающихся пайплайнов в месяц

1.73 M M - миллион

Экономика

Больше цифры

Немного расчетов о текущих и потенциальных возможностях

1.73 M	M - миллион
280	Команд всего
41	Команда использует проект
5740	Сэкономленных часов
33600	Потенциально можно сэкономить

DevOps практики – по следам манифеста

Истоки

Культурные

- DevOps – это про коммуникации
- Культура ”не ищи виновного”
- Распространяй знания, ломай барьеры

Процессные

- Проверяй гипотезы быстро и отбрасывай их
- Точка истины - VCS
- Сервера – скот, не домашние животные
- Ты сделал это, ты запускаешь это!

Технические

- Инфраструктура как код
- Автоматизируй всё
- Измеряй всё
- Логирование – важно!
- Идемпотентность
- CI/CD

sites.google.com/a/jezhumble.net/devops-manifesto/

Какие практики мы поддерживаем с помощью проекта

Как проект помогает в масштабировании DevOps практик

Культурные	DevOps – это про коммуникации Культура ”не ищи виновного” Распространяй знания, ломай барьеры
Процессные	Проверяй гипотезы быстро и отбрасывай их Точка истины - VCS Сервера – скот, не домашние животные Ты сделал это, ты запускаешь это!
Технические	Инфраструктура как код Автоматизируй всё Измеряй всё Логирование – важно! Идемпотентность CI/CD

- Мы создаем единое информационное пространство
- Мы общаемся для передачи экспертизы
- Мы поддерживаем культуру внедрений и привнесения нового
- Мы даем возможность сделать пайплайн каждому и запустить это
- Постоянно работаем над развитием автоматизации
- Реализуем CI/CD для всех

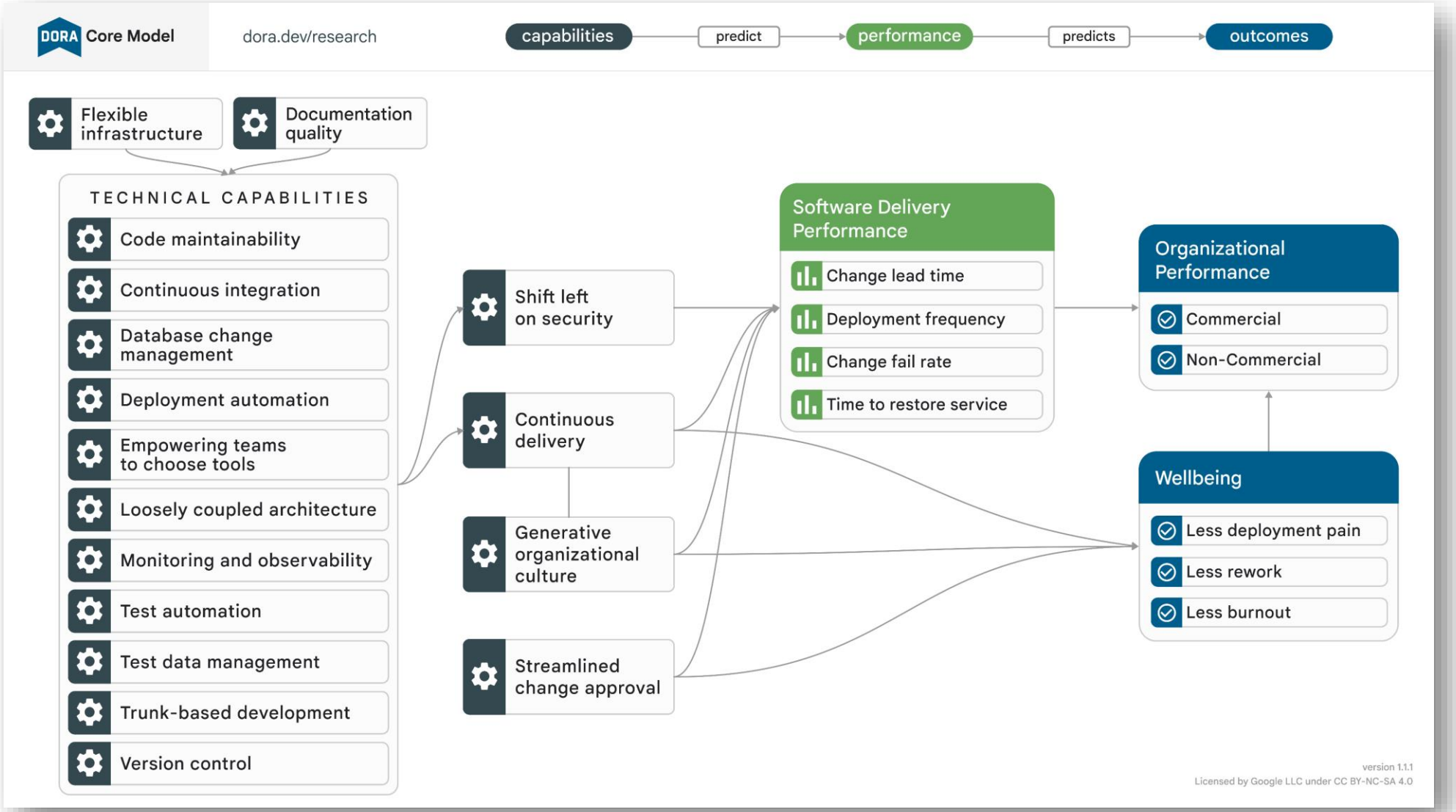
Про крауд сорсинг – “в глубину”

Еще немного мыслей

Создание продукта вокруг которого собираются люди, является ключом к порождению совместного опыта общества, который конвертируется в крауд сорсинг вокруг продукта, тем самым улучшает организационные возможности организаций.

Почему это работает?

Подтверждение мысли с прошлого слайда



Итого

Немного выводов и советов

- Создание общего дела, порождает совместный опыт и развитие переиспользования в организации, как следствие – экономия ресурсов
- Пускай все велосипеды будут в одном месте, зато о них будут знать все и они будут похожи
- Создавая общие решения, всем не угодить и это нормально!

Про людей

В чью карму отправлять энергию :)

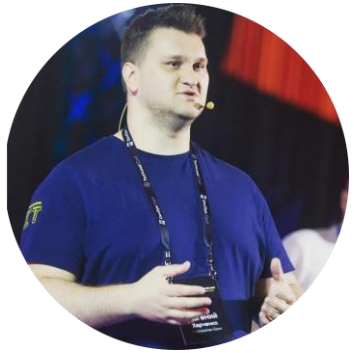
Григорий

Антон

Кирилл

Кирилл

Даниил



Евгений

DGTL Raiffeisen

Найди свой путь CI/CD!

LinkedIn

Telegram

Raiffeisen