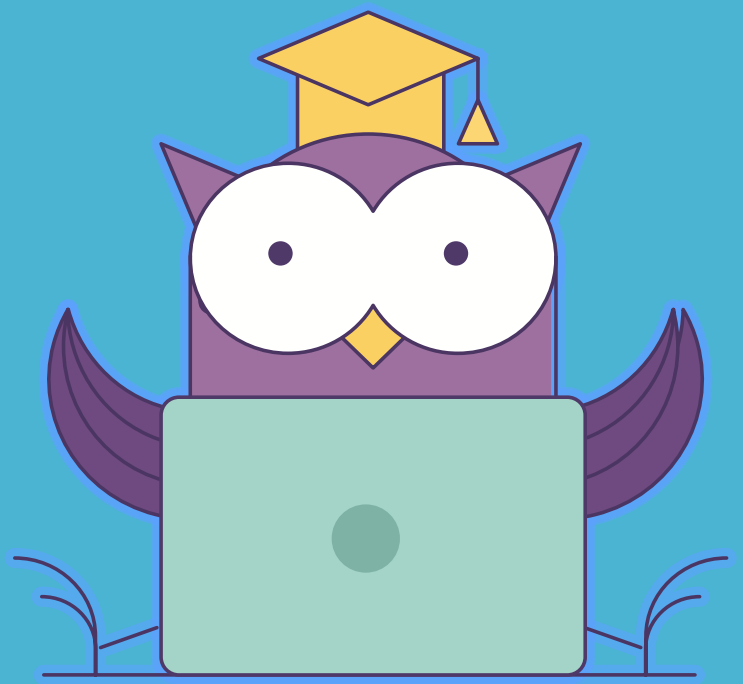
ОНЛАЙН-ОБРАЗОВАНИЕ

OTUS

Как меня слышно и видно?



> Напишите в чат

\+ если все хорошо

\- если есть проблемы со звуком или с видео

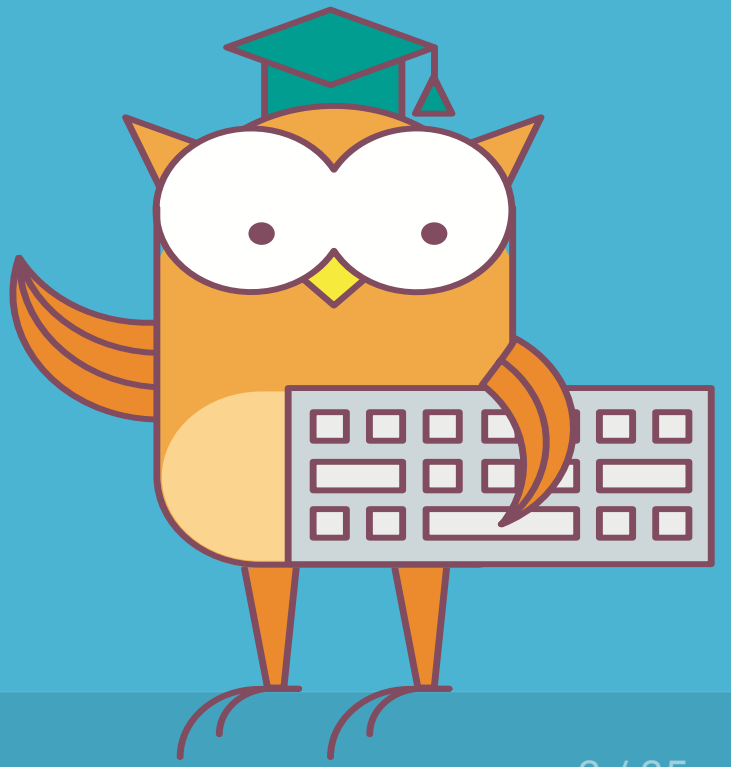
!проверить запись!

OTUS

Gitlab CI

Александр Давыдов

Антон Телышев



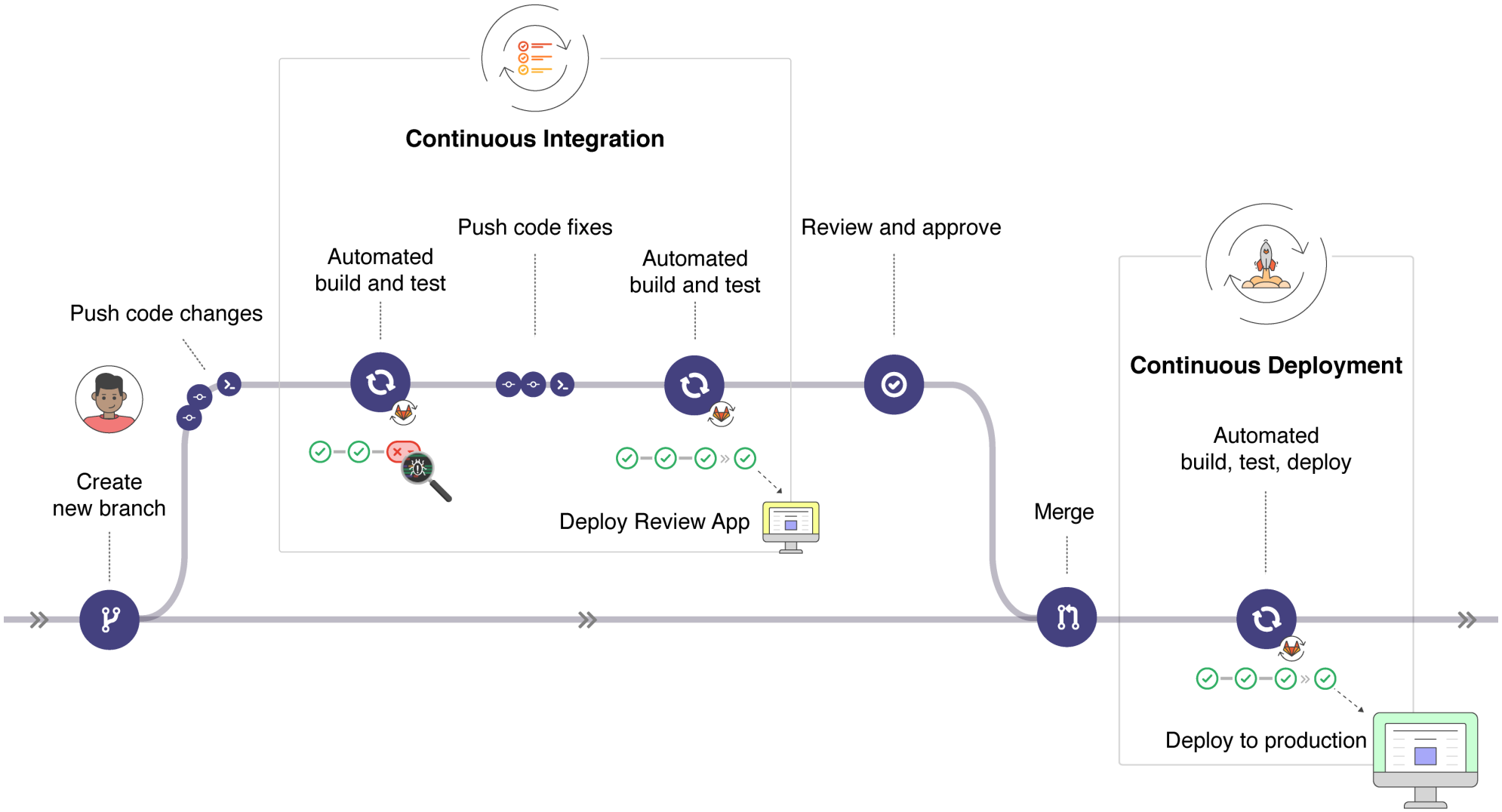
План занятия

- CI/CD
  - Gitlab CI
  - Golang + Gitlab CI

Цель занятия

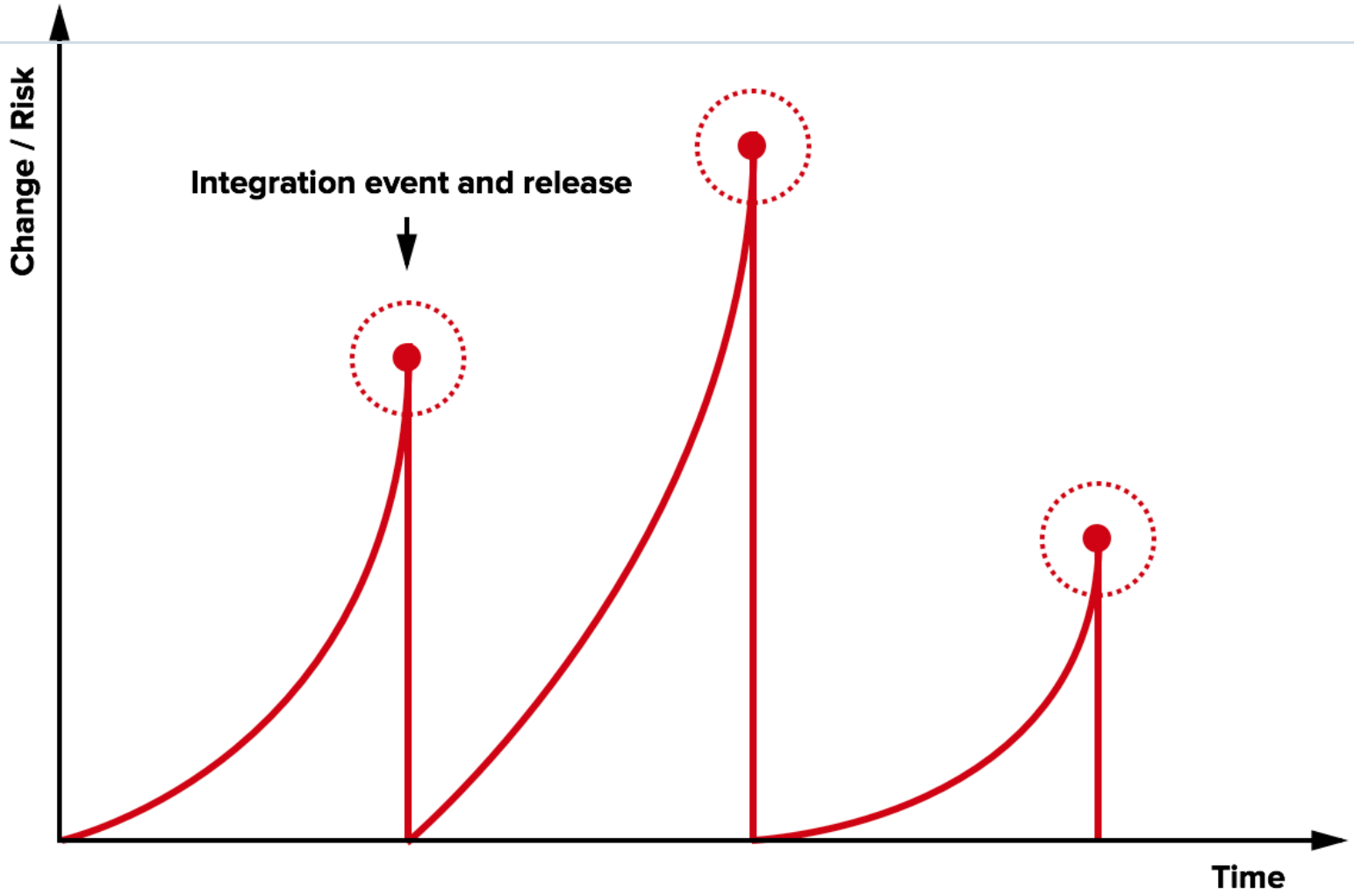
- Научиться "заворачивать" свой проект в CI pipeline

CI / CD workflow



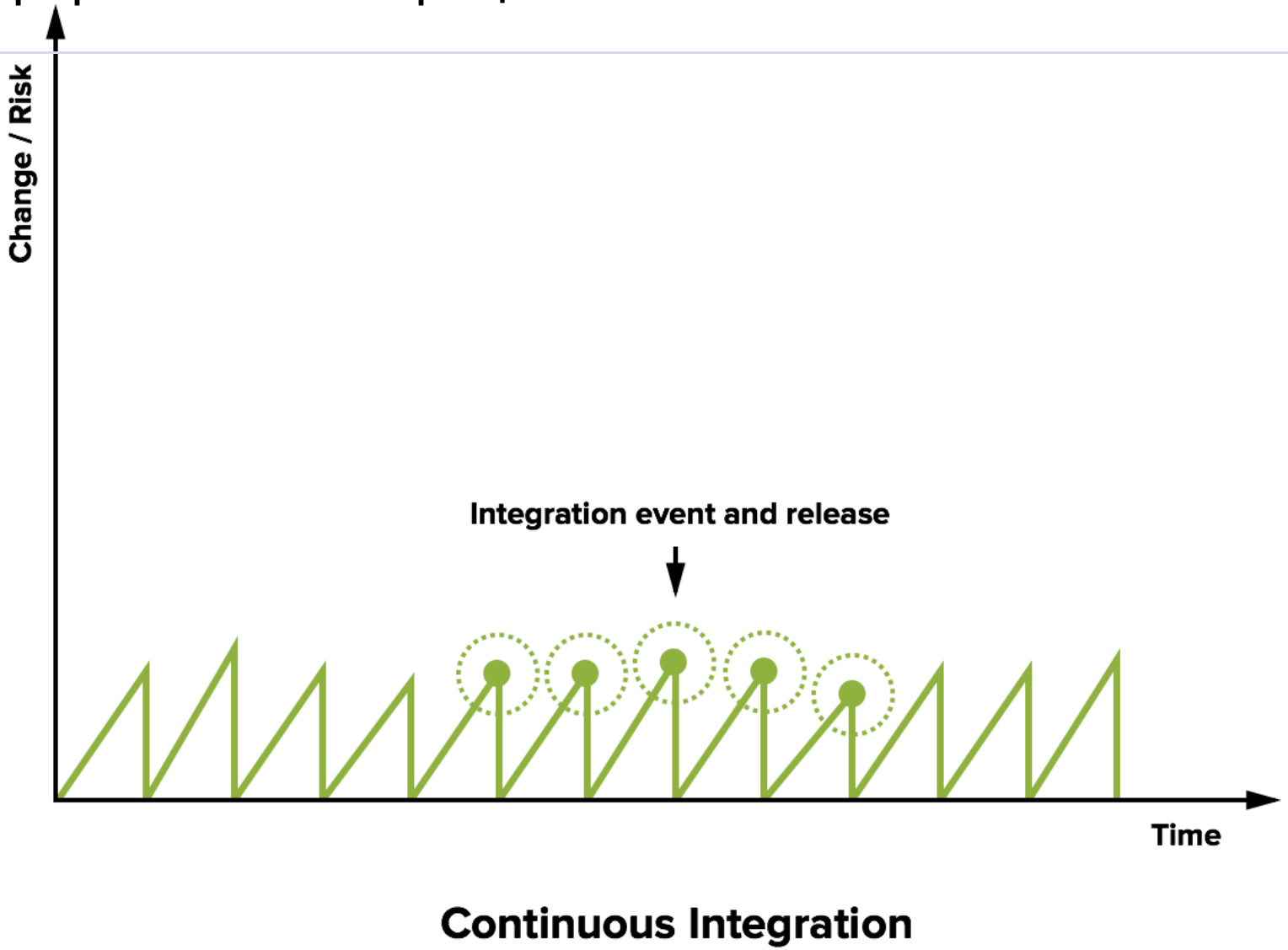
CI, Continuous Integration — непрерывная интеграция - это практика разработки ПО, которая

CI: непрерывная интеграция

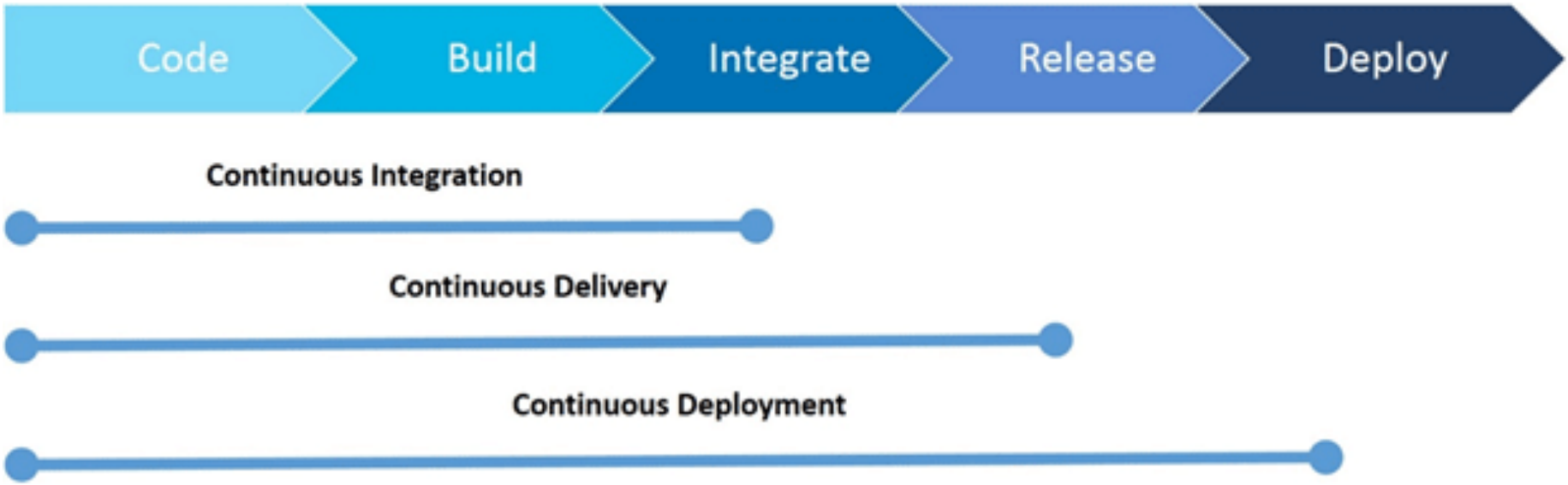


Traditional Integration

CI: непрерывная интеграция



Continuous Integration



- Continuous Integration = build + test
- Continuous Delivery = build + test + manual deploy
- Continuous Deployment = build + test + automatic deploy

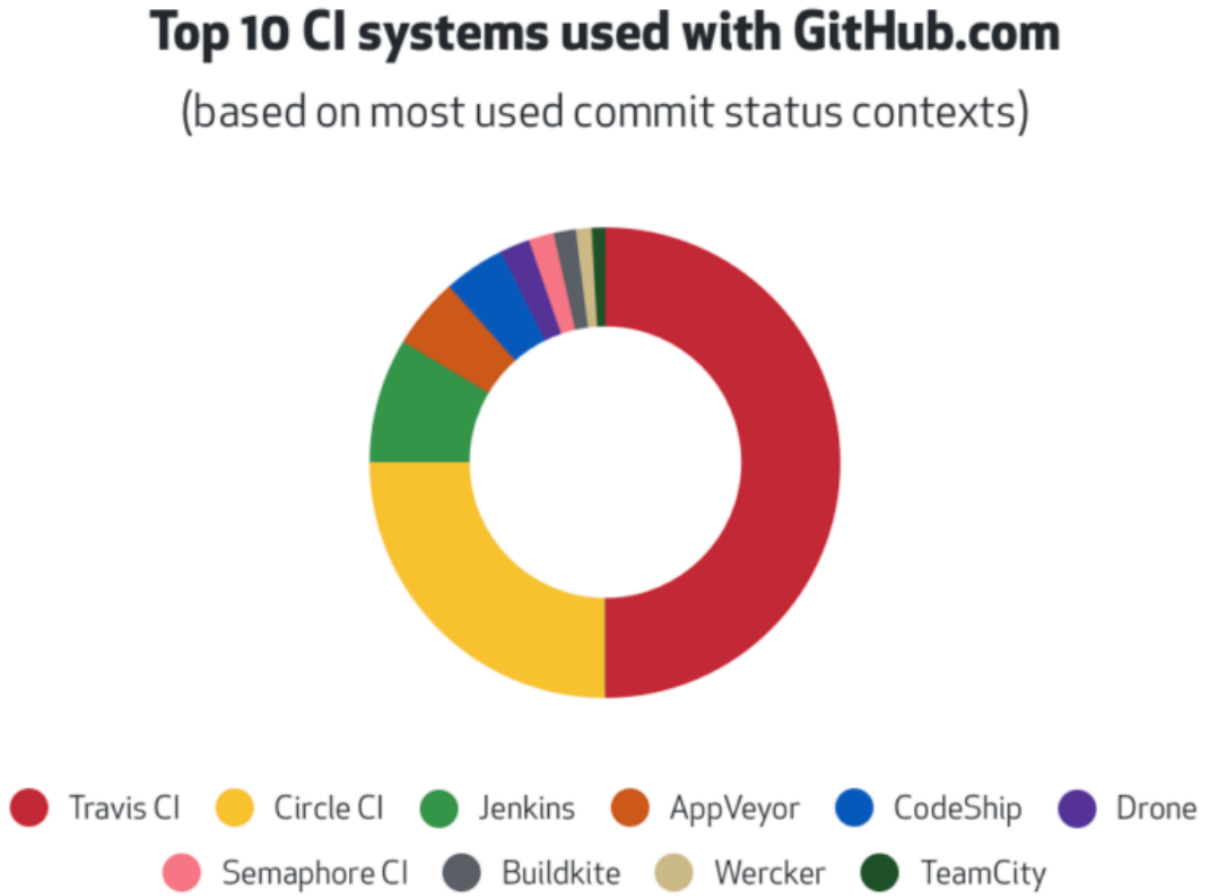
Профит

Continuous Integration

- Быстрое обнаружение ошибок
- Интеграция проще, так как изменения меньше
- Нельзя "отложить фиксы до релиза"

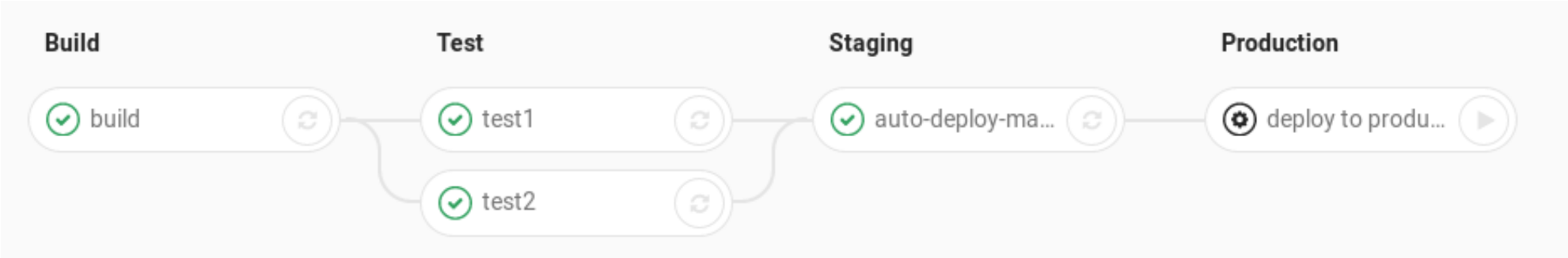
Continuous Delivery

- Код всегда готов к релизу
- Снижение риска при релизах
- Скорость/надежность деплоев
- Быстрая обратная связь от бизнеса



- https://www.browserstack.com/blog/best-ci-cd-tools-comparison/
- https://www.cuelogic.com/blog/best-continuous-integration-ci-tools
- https://techrocks.ru/2019/03/11/10-continuous-integration-systems/

Gitlab CI: действующие лица



- Commit - изменение кода в репозитории.
- Job - набор команд для исполнения Runner'ом.
- Pipeline - набор задач (jobs) распределенных по стадиям (stages).
- Runner - агент / сервер, выполняюший задачи.
- Stages - обозначает стадию, к которой принадлежит задача. Задачи в стадии выполняются параллельно.

Gitlab CI: jobs

Задача определяется именем и имеет как минимум один скрипт.

.gitlab-ci.yml

```
job1:
  script: "execute-script-for-job1"

job2:
  script: "execute-script-for-job2"
```

job:

```
hello:
  script:
  - apt-get install smth
  - echo "hello"
```

Gitlab CI: условия выполнения

```
job:
  only:
    - /^issue-.*$/
  except:
    - branches
```

```
job:
  only: [tags, triggers, schedules]
```

```
stages: [build, cleanup_build]

build_job:
  stage: build
  script: "make build"

cleanup_build_job:
  stage: cleanup_build
  script: "cleanup build when failed"
  when: on_failure
```

Gitlab CI: stages

```yaml
stages:
  - build
  - test
  - deploy

job 1:
  stage: build
  script: make build dependencies

job 2:
  stage: build
  script: make build artifacts

job 3:
  stage: test
  script: make test

job 4:
  stage: deploy
  script: make deploy
```

Gitlab CI: artifacts

```
build:tags:
  image: $CI_IMAGE
  stage: build
  variables:
    CGO_ENABLED: 0
  script:
    - go build -o $CI_PROJECT_DIR/app -ldflags
      "-X main.commit=${CI_COMMIT_SHA}
       -X main.buildNum=${CI_JOB_ID}
       -X main.version=${CI_COMMIT_TAG}" .
  artifacts:
    paths:
    - app
  only:
  - tags
```

Gitlab CI: Docker image

```yaml
image: ruby:2.3

test:2.1:
  image: ruby:2.1
  services:
    - postgres:9.3
  script:
    - bundle exec rake spec

test:2.2:
  image: ruby:2.2
  services:
    - postgres:9.4
  script:
    - bundle exec rake spec
```

Gitlab CI: выполнение для ветки

```
only:
- branch_name
```

```
staging:
  stage: deploy
  script:
    - apt-get update -qy
    - apt-get install -y ruby-dev
    - gem install dpl
    - dpl
      --provider=heroku
      --app=otus-ci-staging
      --api-key=$HEROKU_API_KEY
  environment:
    name: Staging
    url: https://otus-ci-staging.herokuapp.com/
  only:
    - master
```

Gitlab CI: before/after script

```yaml
default:
  before_script: "global before script"

job:
  before_script: "execute this instead of global before script"
  script: "my command"
  after_script: "execute this after my script"
```

```yaml
image: ruby:2.3

before_script:
  - bundle install

pages:
  stage: deploy
  script:
  - bundle exec jekyll build -d public
  artifacts:
    paths:
    - public
  only:
    - master
```

Gitalb CI: services

```yaml
variables:
  POSTGRES_DB: "my_custom_db"
  POSTGRES_USER: "postgres"
  POSTGRES_PASSWORD: "example"
  PGDATA: "/var/lib/postgresql/data"
  POSTGRES_INITDB_ARGS: "--encoding=UTF8 --data-checksums"

services:
  - name: postgres:9.4
    alias: db
    entrypoint: ["docker-entrypoint.sh"]
    command: ["postgres"]

image:
  name: ruby:2.2
  entrypoint: ["/bin/bash"]

before_script:
  - bundle install

test:
  script:
    - bundle exec rake spec
```

Gitlab CI: предопределенные переменные

https://docs.gitlab.com/ee/ci/variables/predefined_variables.html

```
build:tags:
  image: $CI_IMAGE
  stage: build
  variables:
    CGO_ENABLED: 0
  script: go build  -o $CI_PROJECT_DIR/app -ldflags
            "-X main.commit=${CI_COMMIT_SHA}
             -X main.buildNum=${CI_JOB_ID}
             -X main.version=${CI_COMMIT_TAG}" .
  artifacts:
    paths:
    - app
```

- CI_ENVIRONMENT_NAME / CI_ENVIRONMENT_SLUG
- CI_COMMIT_MESSAGE
- CI_COMMIT_REF_NAME
- etc.

Gitlab CI: кастомные переменные

```
variables:
  S3_BUCKET_NAME: "yourbucket"
deploy:
  image: python:latest
  script:
  - pip install awscli
  - aws s3 cp ./ s3://$S3_BUCKET_NAME/
    --recursive --exclude "*" --include "*.html"
```

GitLab -> Settings -> CI/CD -> Variables

Gitlab CI: окружения

```
deploy_review:
  stage: deploy
  script: echo "Deploy a review app"
  environment:
    name: review/$CI_COMMIT_REF_NAME
    url: https://$CI_ENVIRONMENT_SLUG.example.com
  only:
    - branches

deploy_staging:
  stage: deploy
  script: echo "Deploy to staging server"
  environment:
    name: staging
    url: https://staging.example.com

deploy_prod:
  stage: deploy
  script: echo "Deploy to production server"
  environment:
    name: production
    url: https://example.com
  when: manual
```

https://docs.gitlab.com/ee/ci/environments.html

Gitlab CI: кэширование зависимостей

- cache: используем как временное хранилище для зависимостей проекта
- artifacts: результаты выполнения stage, сохраняются между стадиями

```
cache:
  key: one-key-to-rule-them-all
```

```
cache:
  key: ${CI_COMMIT_REF_SLUG} # same branch
```

```
cache:
  key: "$CI_PROJECT_NAME"
  untracked: true
  paths:
    - "$GOPATH/pkg/mod"
```

Gitlab CI tags

```
buildapp:
  image: $CI_IMAGE
  stage: build
  variables:
    CGO_ENABLED: 0
  script: go build -o app.
  artifacts:
    paths: [app]
  only: [tags]

deploys:
  image: docker:stable
  stage: deploy
  script:
    - echo $APP_IMAGE
    - docker build --build-arg bin=build --network host
       -t $APP_IMAGE .
  after_script:
    - docker push $APP_IMAGE
  dependencies:
    - buildapp
  only:
    - tags
```

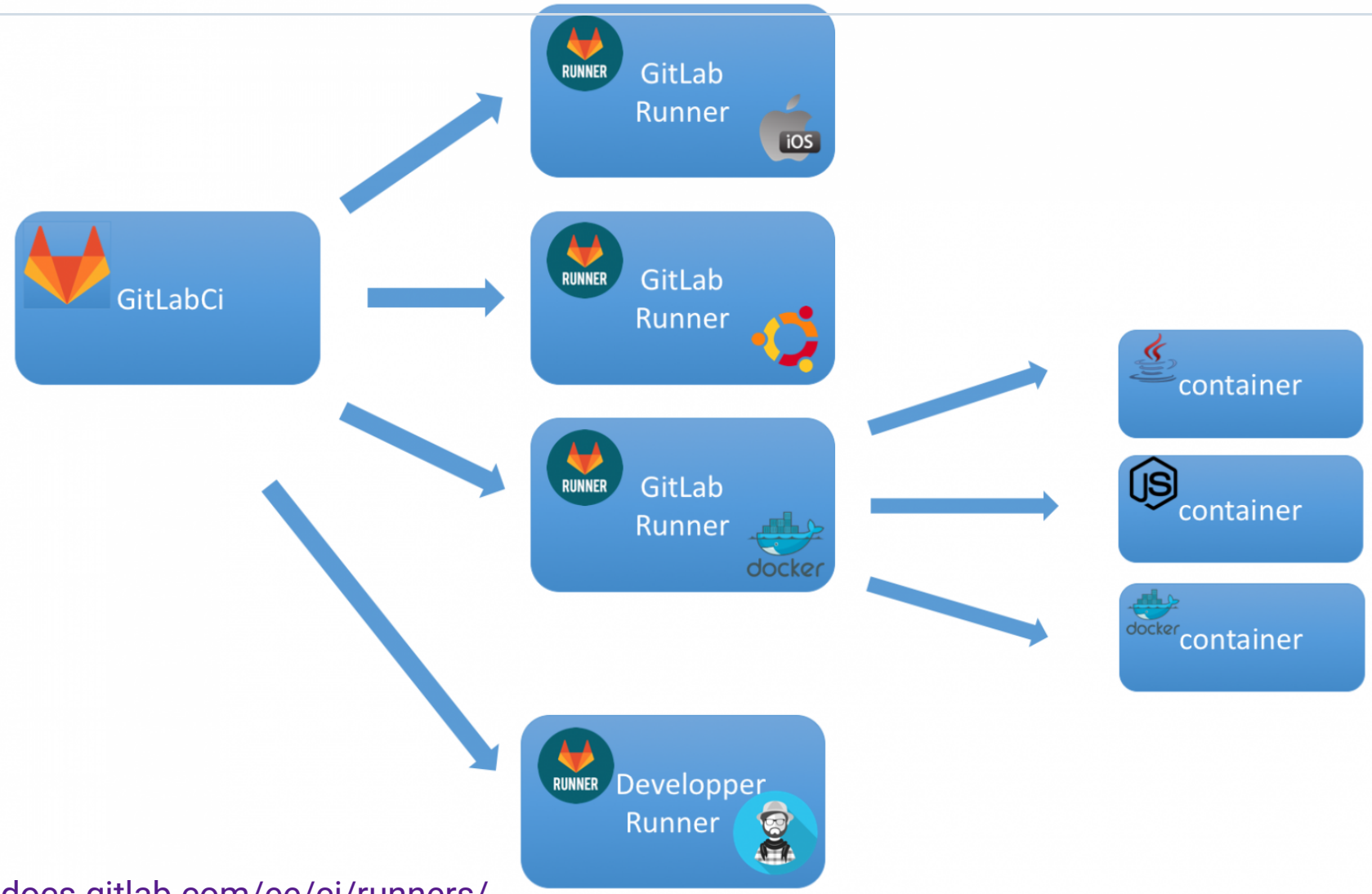
https://docs.gitlab.com/ee/ci/yaml/README.html#dependencies

Gitlab CI: triggers

```
$ git push -o ci.skip
$ git commit -am "[ci skip]"
$ git commit -am "[skip ci]"
```

https://docs.gitlab.com/ee/ci/yaml/#skipping-jobs

Gitlab CI Runner



https://docs.gitlab.com/ee/ci/runners/

Gitlab CI Runner

- Shared Runners - fair usage queue, шарятся между проектами
- Specific Runners - для задач со специфичными требованиями, FIFO
- Group Runners - FIFO, для групп

```
$ brew install gitlab-runner
$ gitlab-runner register
$ gitlab-runner install
$ gitlab-runner start
$ gitlab-runner exec shell test
```

- https://gitlab.com/gitlab-org/gitlab-runner/issues/312
- https://docs.gitlab.com/runner/
- https://medium.com/@umutuluer/how-to-test-gitlab-ci-locally-f9e6cef4f054

Gitlab CI: executors

- SSH
- Shell
- Parallels
- VirtualBox
- Docker
- Docker Machine (auto-scaling)
- Kubernetes
- Custom

GOPROXY

OTUS

- https://github.com/goproxyio/goproxy
- https://arslan.io/2019/08/02/why-you-should-use-a-go-module-proxy/
- GOPRIVATE: https://golang.org/cmd/go/#hdr-Module_configuration_for_non_public_modules

Простой пример

https://gitlab.com/Antonboom/otus-ci-example

- https://about.gitlab.com/blog/2017/11/27/go-tools-and-gitlab-how-to-do-continuous-integration-like-a-boss/

Опрос

Заполните пожалуйста опрос

https://otus.ru/polls/4922/

OTUS

Спасибо за внимание!