# Programozási technológia 2.

## Continuous integration & delivery

Cserép Máté
ELTE Informatikai Kar
2019.

# Continuous integration & delivery

## Folyamatos integráció

- A *folytonos integráció* (*continuous integration*, *CI*) egy olyan gyakorlati módszer, amely lehetővé teszi a programkódok ellenőrzésének és tesztelésének felgyorsítását
  - célja a lehetséges hibák, integrációs problémák azonnali, automatizált kiszűrése, visszajelzés a fejlesztőnek
  - a programkódok verziókezelő rendszer segítségével egy központi tárhelyre kerülnek, naponta többször
  - a tárhely tartalma minden módosítást követően automatikusan fordításra kerül (*build automation*), a fordítással pedig a lekódolt tesztek is végrehajtódnak
  - az így ellenőrzött kódot további tesztelés követheti

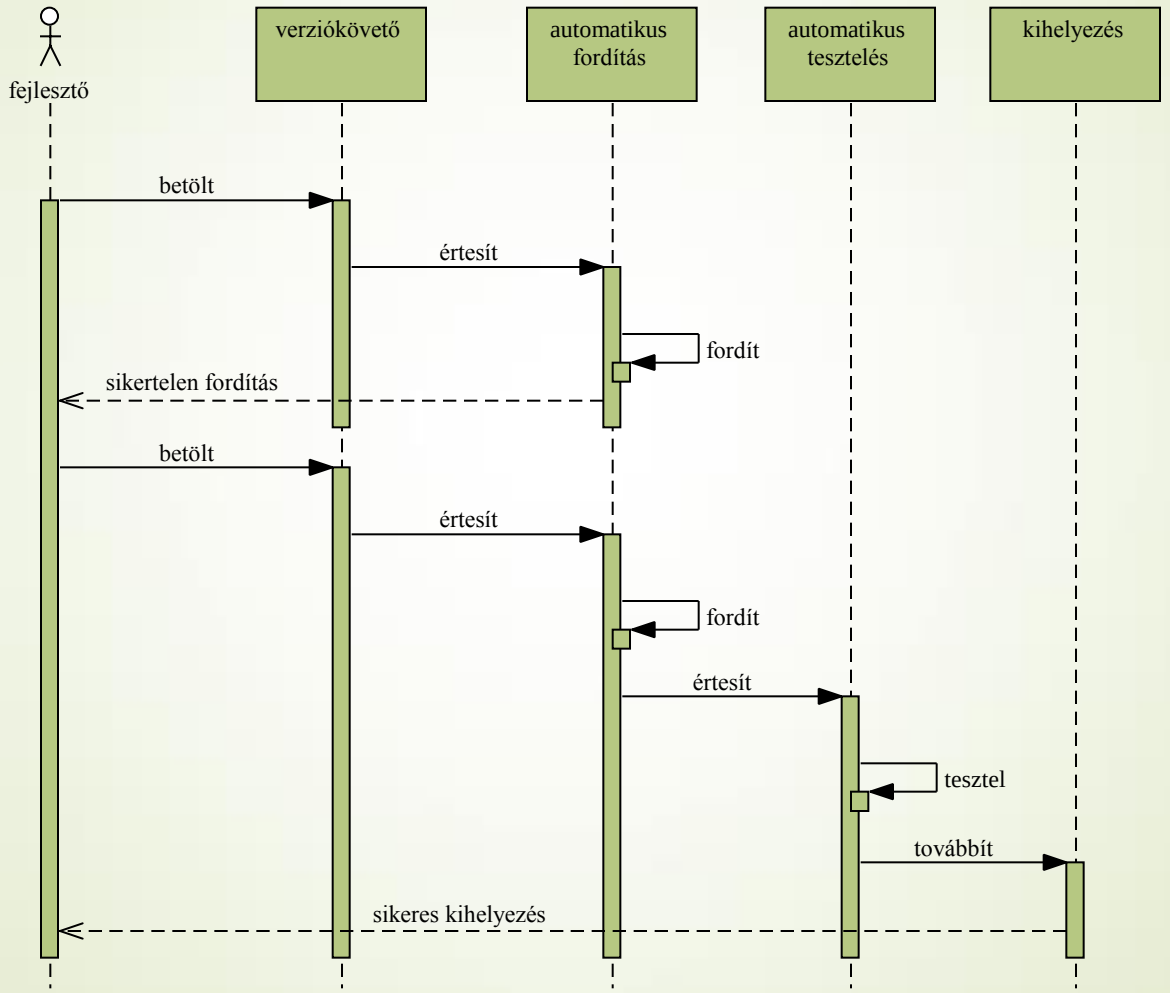
# Continuous integration & delivery

## Folyamatos teljesítés

- Az agilis szoftverfejlesztés (*agile software development*) célja a gyors alkalmazásfejlesztés megvalósítása, inkrementális alapon
  - a szoftver folyamatos fejlesztés és kiadás alatt áll (*continuous delivery*), a sebesség állandó, a változtatások minden lépésben beépíthetőek (*welcome changes*)
  - a működő szoftver az előrehaladás mérőeszköze, előtérben az egyszerűség, ugyanakkor folyamatos odafigyelés a megfelelő tervezésre, optimalizációra
  - a fejlesztést általában önszervező, kis csapatok végzik, megosztott felelősséggel, folytonos interakcióval, gyors visszajelzésekkel
  - a folyamatos kiadások automatizálhatók, ekkor *continuous deployment*-ről beszélünk

# Continuous integration & delivery

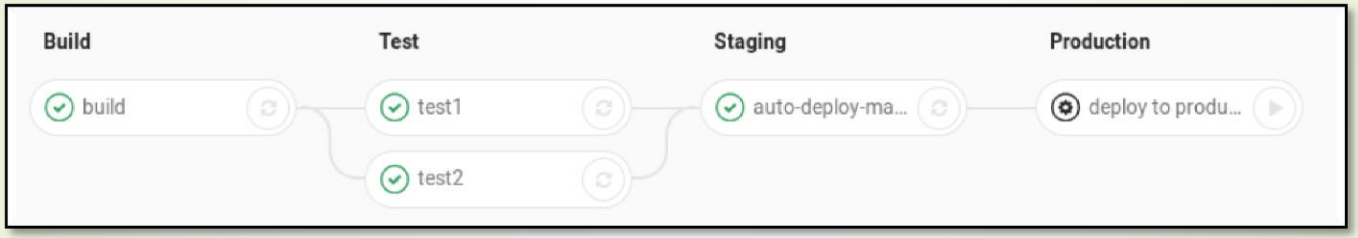
## Folyamatos integráció és teljesítés

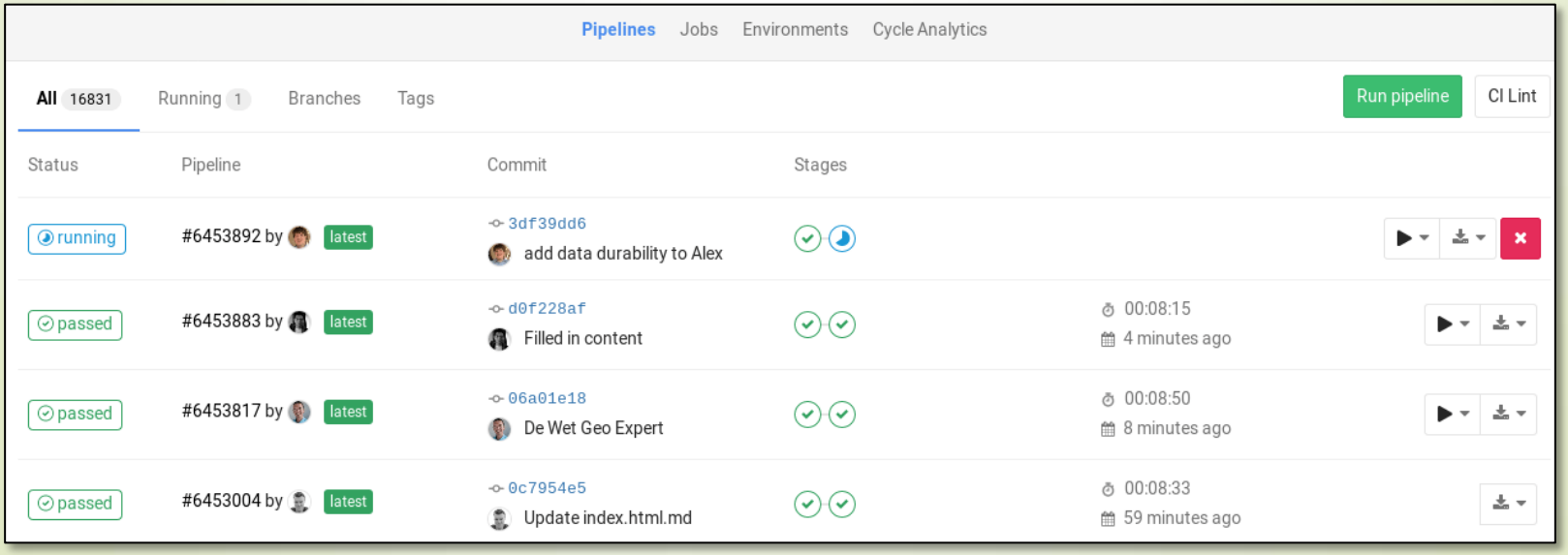


# Continuous integration & delivery

## Feladatok

- A folyamatos integráció és teljesítés lépéseit egymásra épülő feladatok (*jobs*) láncolataként (*pipelines*) definiálhatjuk

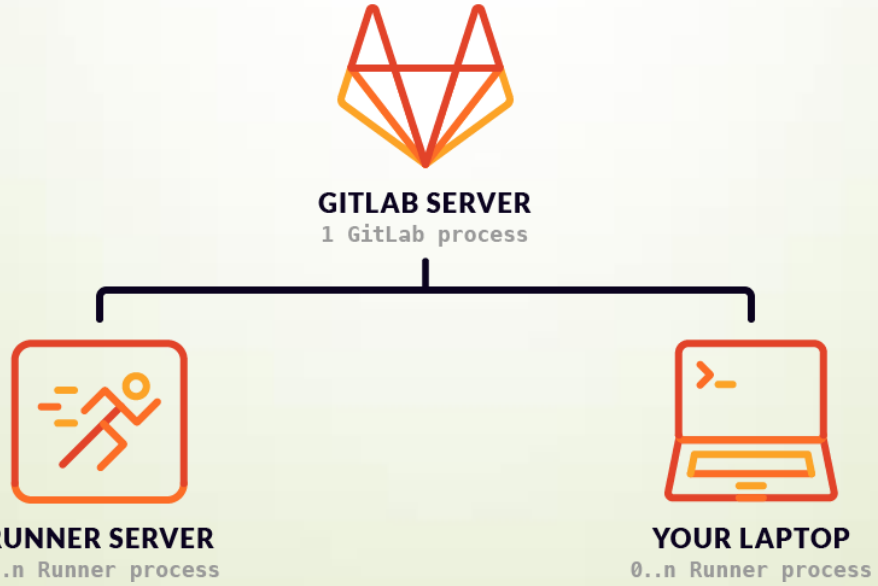


# Continuous integration & delivery

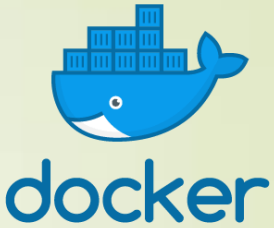
## GitLab Runners

- A GitLab rendelkezik integrált, saját megoldással a folyamatos integráció és teljesítés támogatására
  - a feladatokat (jobs) a GitLab szervertől független ún. *GitLab Runner* példányok hajtják végre
    - Shell, SSH, VirtualBox, Docker, Kubernetes, stb.
  - a *runnerek* egyedileg konfigurálhatóak, lehetnek megosztottak vagy projekthez rendeltek



# Continuous integration & delivery

## Docker




- A folyamatos integrációt egy izolált, reprodukálható környezetben érdemes végezni
- A Docker napjainkban a legelterjedtebb *container framework*
  - a *container* hasonlít a virtuális gépekhez (*VM)* olyan tekintetben, hogy egy teljesen elkülönített, virtualizált környezett biztosít, amelynek a gazdaszámítógép szolgáltat erőforrásokat
  - a fő különbség a *containerek* és a virtuális gépek között, hogy minden *container* osztozik a gazda kerneljén a többi *containerrel*, a virtualizált hardver és az OS nem része, csupán az alkalmazásunkhoz kötődő könyvtárak, binárisok és a felhasználói terület
  - a *containerek* ezáltal nagyságrendekkel kisebb *overheaddel* bírnak a VM-ekhez képest, így könnyebb súlyú megoldást nyújtanak a virtualizációra

# Continuous integration & delivery

## GitLab CI/CD

- A folyamatos integráció konfigurációját a `.gitlab-ci.yml` fájl tartalmazza, YAML formátumban
  - a YAML (*YAML Ain't Markup Language*) egy emberi szemmel könnye(bbe)n olvasható strukturált leíró nyelv
  - https://yaml.org/spec/1.2/spec.html
- A GitLab webes felületén elérhető egy CI Lint funkció a formátum validálására beküldés előtt

```
before_script:
  - apt-get update -qq
  - apt-get install -yqq openjdk-11-jdk ant

build_program:
  script:
    - ant compile
    - ant jar
```

# Continuous integration & delivery

## GitLab CI/CD: jobs

- Több feladat (*job*) definiálása:

```
before_script:
  - apt-get update -qq
  - apt-get install -yqq openjdk-11-jdk ant junit

build_program:
  script:
    - ant compile
    - ant jar

test_program:
  script:
    - ant compile-test
    - ant test
```

# Continuous integration & delivery

## GitLab CI/CD: stages

- A folyamatos integráció feladatait egymást követő szakaszokra (*stages*) oszthatjuk
  - alapértelmezetten 3 *stage* van: *build*, *test*, *deploy*
  - ez tetszőlegesen felüldefiniálhatjuk

```
stages:
  - lint
  - build
```

- Egy *stage* feladatai egymástól függetlenül párhuzamosítva végrehajthatóak (több *runner* bevonásával)
  - a *stagek* egymásra épülnek, amennyiben egy *stage* valamely feladata hibával zárul, a rá épülő *stagek* nem kerülnek végrehajtásra

# Continuous integration & delivery

## GitLab CI/CD: stages

- A folyamat *stagekre* osztása:

```
before_script:
  - apt-get update -qq
  - apt-get install -yqq openjdk-11-jdk ant junit

build_program:
  stage: build
  script:
    - ant compile
    - ant jar

test_program:
  stage: test
  script:
    - ant compile-test
    - ant test
```

# Continuous integration & delivery

## GitLab CI/CD: artifacts

- A CI feladatok részeként előállított bináris vagy egyéb állományokat megőrizhetjük (*artifact*)

```
pdf:
  script: pdftex paper.tex
  artifacts:
    paths:
      - paper.pdf
    expire_in: 1 month
```

- Az *artifactok* a GitLab webes felületéről könnyen letölthetőek

# Continuous integration & delivery

## GitLab CI/CD: artifacts

- *Artifactok* definiálása:

```
before_script:
  - apt-get update -qq
  - apt-get install -yqq openjdk-11-jdk ant

build_program:
  stage: build
  script:
    - ant compile
    - ant jar
  artifacts:
    paths:
      - dist/program.jar
    expire_in: 1 week
```

# Continuous integration & delivery

## GitLab CI/CD: dependencies

- *Artifactok* átadása *jobok* között:

```
before_script: ...

build_program_A:
  stage: build
  script:
    - cd module_A
    - ant jar
  artifacts:
    paths:
      - module_A/dist/program_A.jar

build_program_B: # depends on program_A.jar
  stage: build
  script:
    - cd modue_B
    - ant jar
  dependencies:
    - build_program_A
```

# Continuous integration & delivery

## Docker images

- Docker container alapú GitLab Runner esetén megadhatjuk melyik *docker image*-ből kívánunk kiindulni:

```
image: ubuntu:18.04

before_script: ...
```

- Docker image-t egy *docker registryből* kérhetünk:
  - Használható a publikus Docker Hub, ahová saját image is feltölthető: https://hub.docker.com/
  - Használható privát docker registry is (pl. vállalati környezet)
- Ha nem adjuk meg, akkor a runner konfigurációja adja meg a használandó image-t
  - a szofttech.inf.elte.hu runnerjei az `ubuntu:18.04` imagere vannak konfigurálva

# Continuous integration & delivery

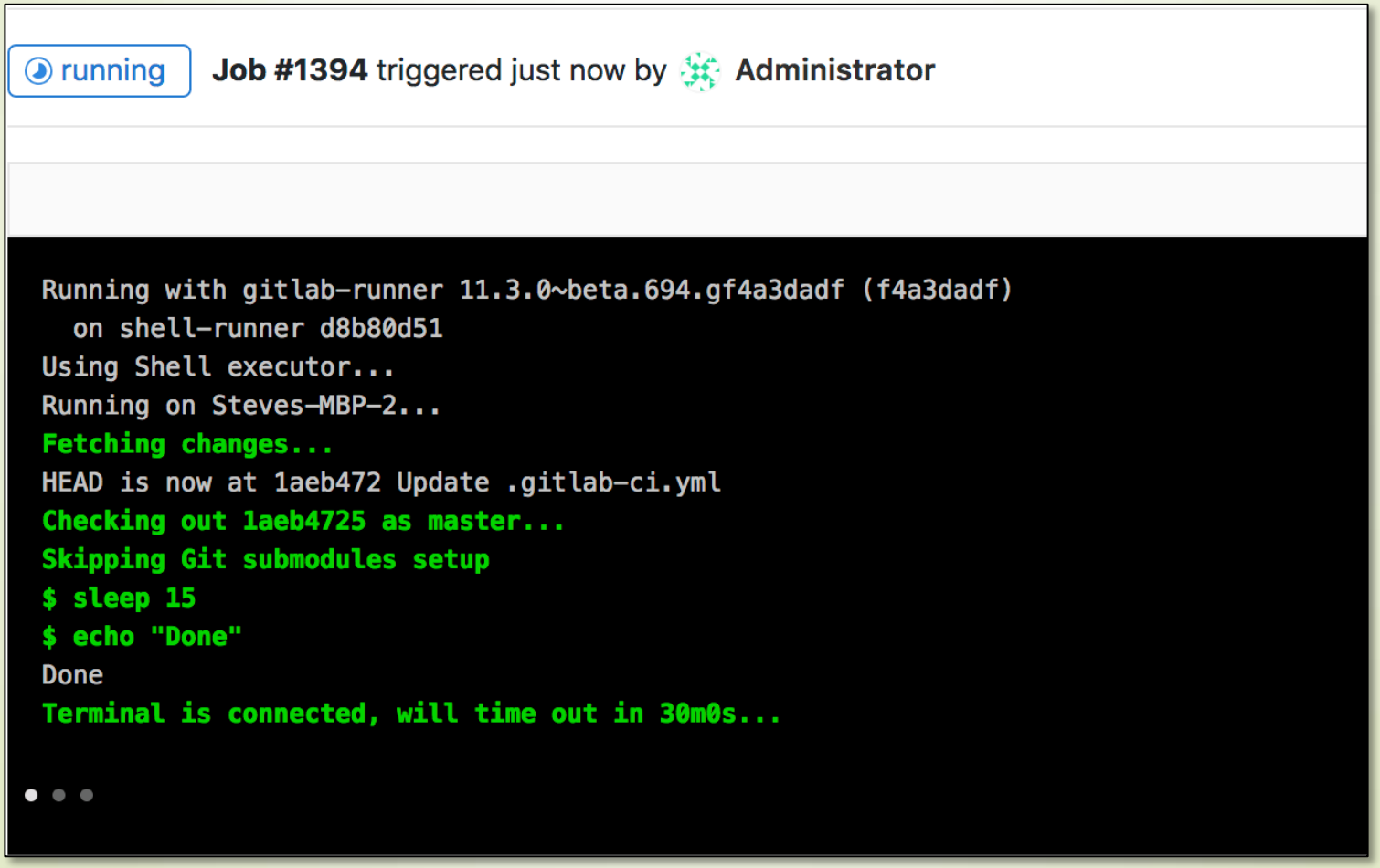
## GitLab CI/CD

- További lehetőségek (teljesség igénye nélkül):
  - `only, except:` CI jobok végrehajtásának feltételhez kötése (például csak a *master* branch-en futtatni)
  - `when:` CI jobok végrehajtásának feltételhez kötése (manuális vs. automatikus végrehajtás)
  - `cache:` fájlok, könyvtárak (tipikusan függőségek) megőrzése CI jobok között
  - `variables:` változók definiálása.
    A futtató környezetre számos változó már előre definiált: https://docs.gitlab.com/ce/ci/variables/
  - `services:` szolgáltatások (pl. adatbázis motor) külön Docker containerben futtatása (*docker-compose*)

# Continuous integration & delivery

## GitLab CI/CD: terminals

- A folyamatos integráció feladatainak végrehajtását egy online terminál ablakon keresztül követhetjük a GitLab webes felületén



# Continuous integration & delivery

## GitLab CI/CD: példa projekt

- Hálózati Pacman játék Java implementációval: https://szofttech.inf.elte.hu/mate/pacman-java/

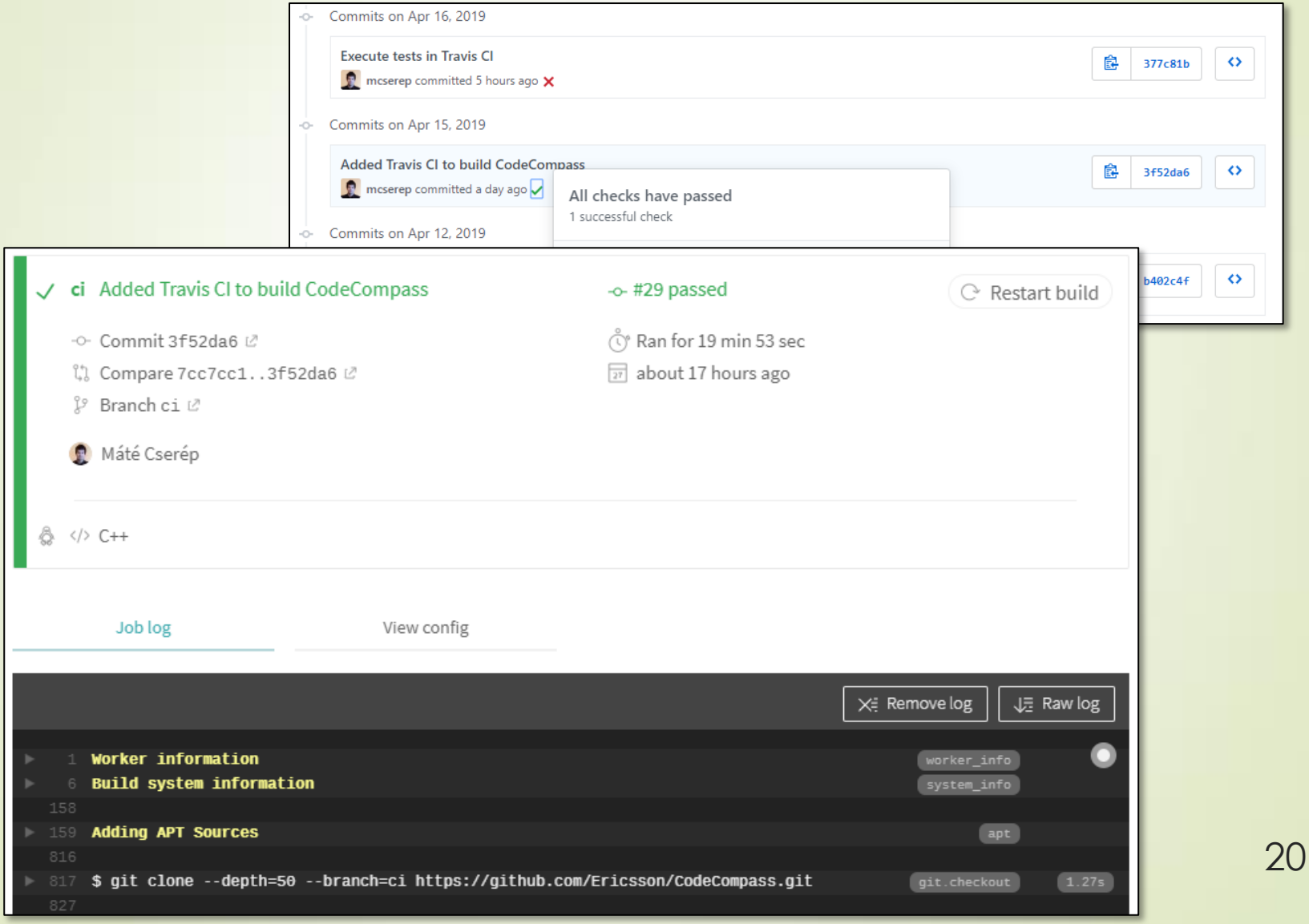
# Continuous integration & delivery

## Travis CI


- Folyamatos integrációs szolgáltatás GitHub projektekhez
  - Nyílt forráskódú projektekhez ingyenesen használható
  - https://travis-ci.com/
- Támogatja a Linux, a Windows és a macOS operációs rendszereket, több előkészített környezettel (*image*)
  - A környezetet a használt programozási nyelvhez állíthatjuk be, az elterjedtebb fordító eszközökkel és könyvtárakkal
- A CI konfigurációt a `.travis.yml` fájlban adhatjuk meg

# Continuous integration & delivery

## Travis CI – GitHub integration



# Continuous integration & delivery

## Travis CI

- Például:

```
os: linux
dist: xenial
language: java

stages:
  - compile
  - test
  - deploy

# ...

jobs:
  include:
    - stage: compile
        - mvn compile
    - stage: test
        - mvn test

# ...
```

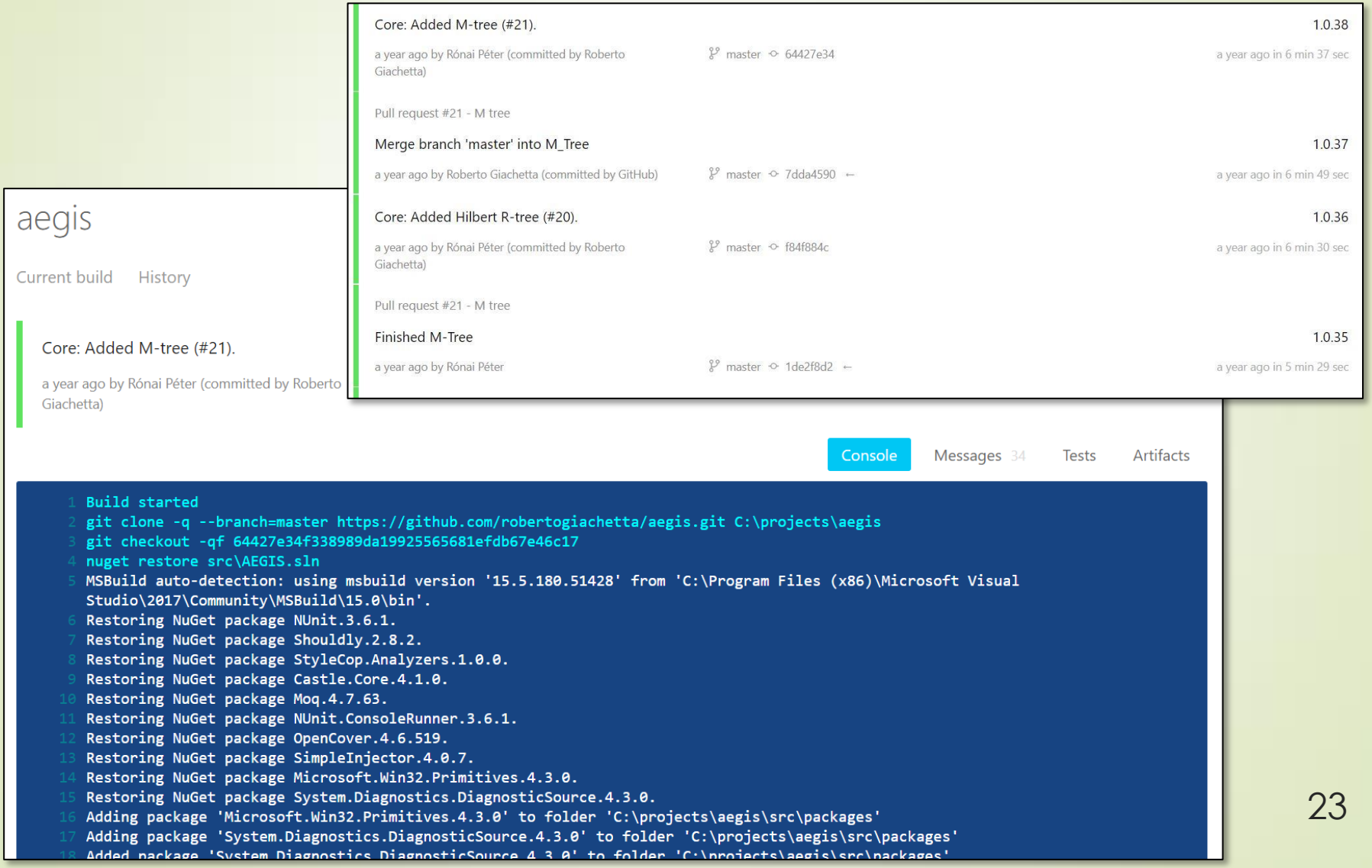
# Continuous integration & delivery

## AppVeyor CI

- Folyamatos integrációs szolgáltatás
  - Integrálható a GitHub, GitLab, BitBucket, Visual Studio Team Services platformokkal
  - Nyílt forráskódú projektekhez ingyenesen használható
  - https://www.appveyor.com/
- Támogatja a Linux, a Windows operációs rendszereket, több előkészített környezettel (*image*)
  - Kiemelt .NET és Visual Studio támogatás
- A CI konfigurációt a `appveyor.yml` fájlban adhatjuk meg

# Continuous integration & delivery

## AppVeyor CI – webes interfész



# Continuous integration & delivery

## AppVeyor CI

- Például:

```
version: 1.0.{build} # Version format
image: Visual Studio 2017 # Build worker image
platform: Any CPU # Build platform
configuration: Debug # Build Configuration

# Execute script before build
before_build:
  - dotnet restore src\MyProject.sln

# Execute build script
build_script:
  - dotnet build src\MyProject.sln

# Execute test script
test_script:
  - dotnet test src\MyProject.sln
```

# Continuous integration & delivery

## Jenkins

- Nyílt forráskódú folyamatos integrációs szolgáltatás
  - Integrálható a GitHub, GitLab, és egyéb projektvezető szolgáltatásokkal
  - Nem nyújt hoszting szolgáltatást, de más vállalkozások kínálnak (pl. CloudBees)
  - https://jenkins.io/
- A CI konfigurációt a `Jenkinsfile` adja meg, például:

```
pipeline {
    agent { docker { image 'maven:3.3.3' } }
    stages {
        stage('build') {
            steps {
                sh 'mvn compile'
            }
        }
    }
}
```