Szoftvertechnológia és -technikák

13. Előadás - DevOps, Continuous integration, Continuous delivery

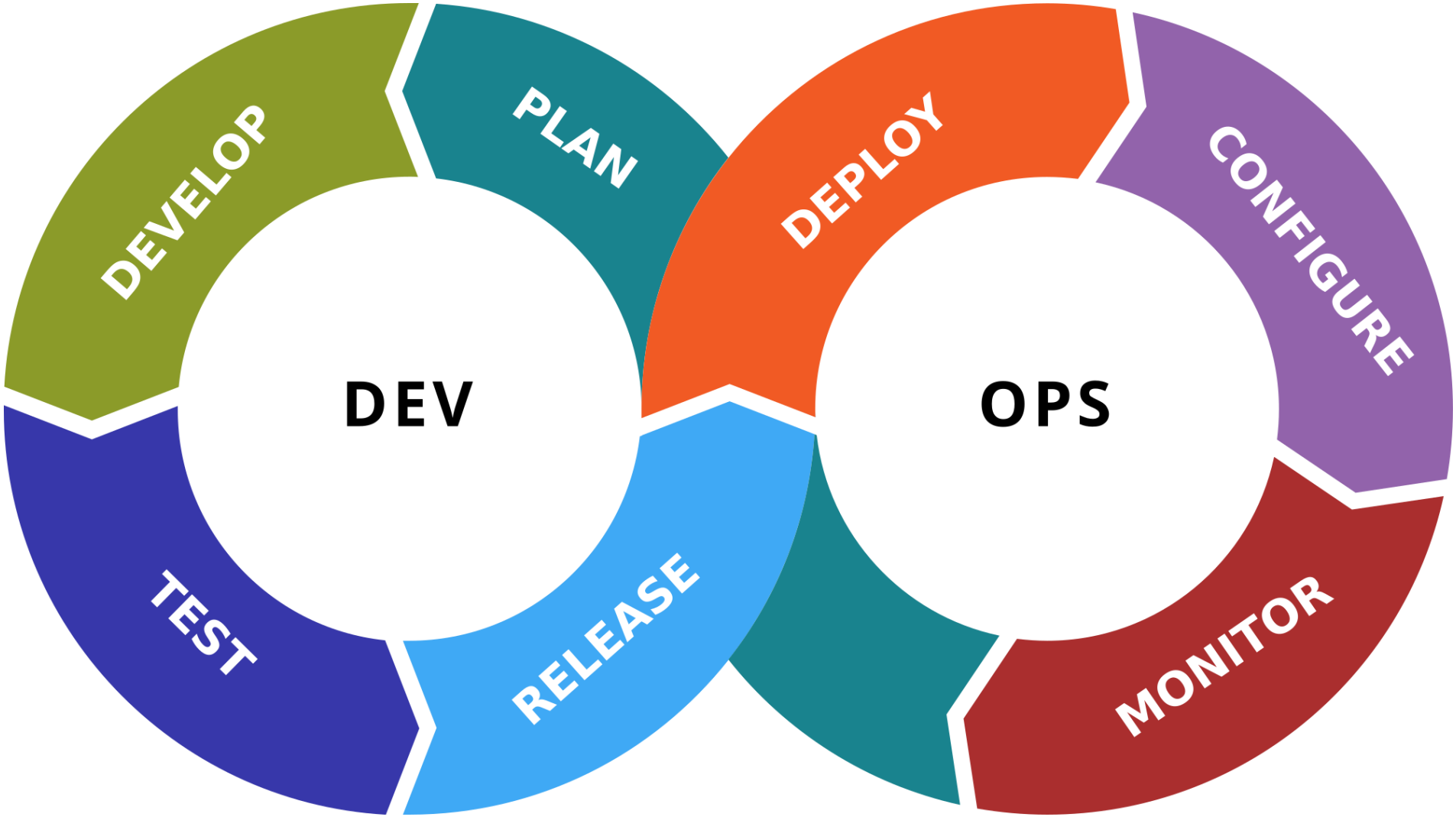
BME AUT
Automatizálási és
Alkalmazott
Informatikai Tanszék

DevOps

Bevezetés

- A szoftver éllettartama nem ér véget a fejlesztésnél
- A szoftverfejlesztés csapatjáték
- Hagyományosan a fejlesztők és az üzemeltetők külön csapatokat alkottak
  - ez a szeparáció csökkenti a kommunikáció hatékonyságát, lassítja az új verziók megjelenését
- Az agilis világban már sokszor nem elegendő

Megoldás


DEVELOP
PLAN
DEPLOY
CONFIGURE
DEV
OPS
TEST
RELEASE
MONITOR

DevOps: Áttekintés

- Software development-ből (Dev) és information-technology operations-ből (Ops) áll
- DevOps egy fejlesztői módszertanok gyűjteménye

- Célja a szoftver fejlesztésének és üzemeltetésének összehozása

DevOps: módszertanok

- **Continuous Integration**
- **Continuous Delivery**
- Infrastructure as Code
  - Pl: Kubernetes
- Monitoring and Logging
- Microservices
- Communication and Collaboration

Continuous Integration

Bevezetés

- Napjaink fontos kérdése az automatizálhatóság
- A szoftverfejlesztés célja sokszor a költséges emberi erőforrás kiváltása
  - A szoftverfejlesztés emberi erőforrásának kiváltása is
- Nem csupán költségcsökkentés a cél, hanem az esetleges hibalehetőségek csökkentése is

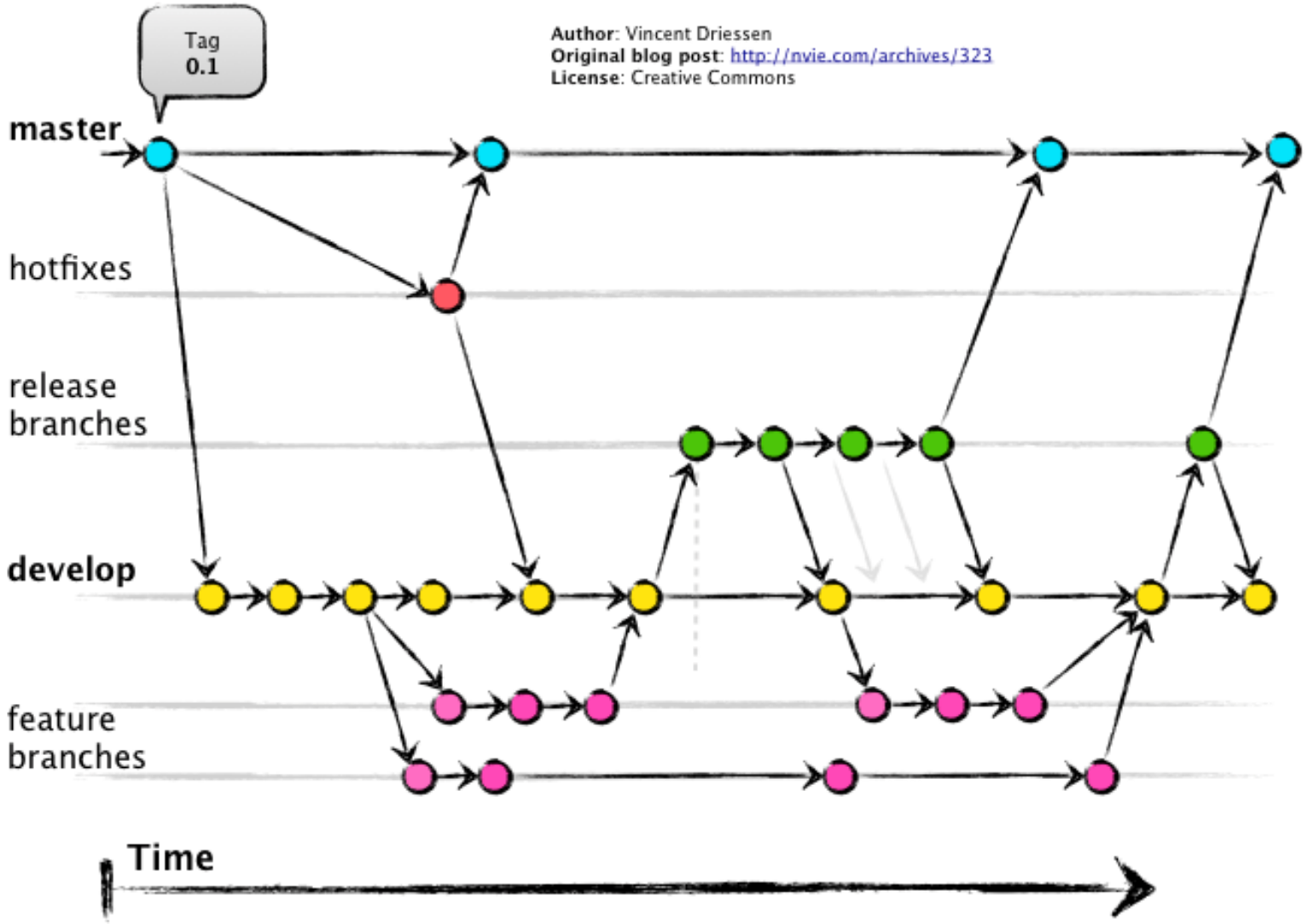
CI elemek

- *Continuous Integration* szerepe és lehetőségei
- *Continuous Integration* eszközök (*Jenkins*)
- *Jenkins* képességek, *script*-elhetőség, *artifact*-ok
- *Lint v. Sonar* kódelemző
- Saját monitoring „*Dashboard*”
- *Continuous Delivery*

Fejlesztő feladatai

- (*meeting*-eken való részvétel)
- Tesztek írása (specifikáció megfogalmazása programkóddal)
- Funkciók implementálása (specifikáció megvalósítása programkóddal)
- Tesztek futtatása
- Program fordítása
- Telepítések
- Értesítés telepítésekről

Egy alkalmazás időbeni alakulása



Egy alkalmazás verziói időben

- Mergelés előtt célszerű megbizonyosodni, hogy mergelés után az alkalmazás helyesen, specifikáció szerint fog működni
  - Ezeket az ellenőrzéseket lehet kézzel minden merge esetén elvégezni, de célszerű lenne automatizálni
- Ennél jobb ha nem csak merge-enként, hanem push-onként is megbizonyosodunk az alkalmazás helyességéről
  - Kézzel ez sok, ráadásul repetitív munka, ami előbb-utóbb el fog maradni

További problémák

- A fejlesztő ismeri egyedül a folyamatot
  - Fluktuáció esetén a tudás elveszik, átadása másik ember felé költséges és veszteséges lehet
- A folyamat minden egyes eleméhez szaktudás szükséges
  - Fejlesztői (rendszergazdai) ismeretek megléte
- Ha éppen nem elérhető a fejlesztő, nem lesz új *release*
- A fejlesztői (rendszergazdai) erőforrás költséges

Megoldás

Continuous Integration Bevezetése

Mit jelent valójában?

- A rendszerünk, jellemzően naponta 1-2-szer hajtja végre a teljes folyamatot
  - Integrált
    - Minden változtatás, művelet egy láncként hajtódik végre
  - Lefordított
    - A kód lefordul és termék keletkezik (például futtatható állomány)
  - Tesztelt
    - Automatizált tesztek lefutnak
  - Archivált
    - Verziózott és tárolt, szükség esetén felhasználható
  - Élesített
    - Feltöltődik abba a célkörnyezetbe, ahol a fejlesztők használhatják

Mikor működik jól?

- Automatizált *build* folyamat
  - Külön gépen, tehát nem veszi el a fejlesztői gép erőforrását
  - A fejlesztő tud fejleszteni, míg a folyamat zajlik
- Tegyük a *build*-et öntesztelővé
  - A sikeres fordítás után fussanak le a tesztek
    - Biztosítja, hogy a szoftver úgy viselkedik, ahogy a fejlesztő elvárja
- Mindenki commitoljon sűrűn
  - Naponta párszor (1-2)
    - Így hamar kiderül, ha valami nem működik

Mikor működik jól?

- A *build* művelet legyen gyors
  - Ha nem gyors, nem segíti a munkánk
    - Ha hiba van az integrációban, könnyen azonosíthatjuk
- Független gépen tesztelés
  - „Nálam megy” hiba elkerülése
  - Hamar detektálhatjuk, ha a fejlesztői környezetem eltér a tesztkörnyezettől
- Tegyük könnyen elérhetővé a legutóbbi stabil *build*-ek eredményeit
  - Ezáltal a mindenkori legfrissebb stabil verzió felhasználhatóvá válik
  - Akár a menedzser is felhasználhatja

Mikor működik jól?

- Mindenki láthatja a *build*-ek eredményeit
  - Könnyen felderíthető, ha egy *build* sikertelen volt és az is, hogy mi okozta a hibát
- Automatikus telepítés
  - A legtöbb CI rendszer a engedélyezi szkriptek futtatását
    - A sikeres *build* folyamat után
  - Éles rendszerekbe célszerűen nem automatizáltan telepítünk
    - Rengeteg commit, nem mindig kész release
    - Vannak ellenpéldák is

Használatának előnyei

- Integrációs *bug*-ok hamar kiderülnek
  - Gyors visszajelzés, a projekt nem kerül rossz útra (sokáig)
    - Ezzel pénzt és időt takaríthatunk meg
- A *release*-k előtti utolsó utáni pillanatbeli brutális *merge*-eket elkerüljük
  - Mert rendszeresen használjuk a CI-t lépésről lépésre
- Jelzés ha elbukik egy teszt / bug keletkezik
  - A rendszeres CI használat miatt könnyen visszaállhatunk egy kevés változtatással ezelőtti változatra
- A mindenkori legfrissebb stabil verzió állandó rendelkezésre állása
- A gyakori *push*-ok akaratlanul is kényszerítik a fejlesztőt az aprólékos fejlesztésre
  - Nem hoznak létre túl nagy modulokat (nem is fér bele az időbe)
- A projekt mérőszámai generálhatóak a *CI*-ből
  - Nem szükséges a menedzsernek grafikonokat rajzolni az előrehaladásról

Használatának költségei

- Ki kell mozdulnunk a komfortzónából
  - Megszokott módszereinket háttérbe szorítani
    - Fokozatosan, hirtelen mindent lehetetlen
- Fegyelmezettség
  - A csapat minden tagjának be kell tartania
- Üzemeltetés és konfigurálás
  - Széles spektrumon mozog, a minimum költség ~0

Szerepe

- Felelősség levétele a fejlesztő válláról
  - Több idő marad a fejlesztésre
- Monitorozhatóság biztosítása
  - Több projekt állapotának párhuzamos követése
  - A fejlesztés mélyebb (részletesebb) követhetősége
- Azonnali visszajelzés
  - A fejlesztőnek
  - A megadott csapattagoknak
  - A főnökségnek
  - A megrendelőnek

CI további szerepei

- Egy élesíthető megoldás bármely pillanatban
  - A mindenkori legújabb *stabil* verzió
- A projekt fejlődésének követése, rögzítése
- Bug felderítés egy független gépen azonnal
- Mindezt úgy valósítja meg, hogy közben nem szól bele az alkalmazás életciklusába

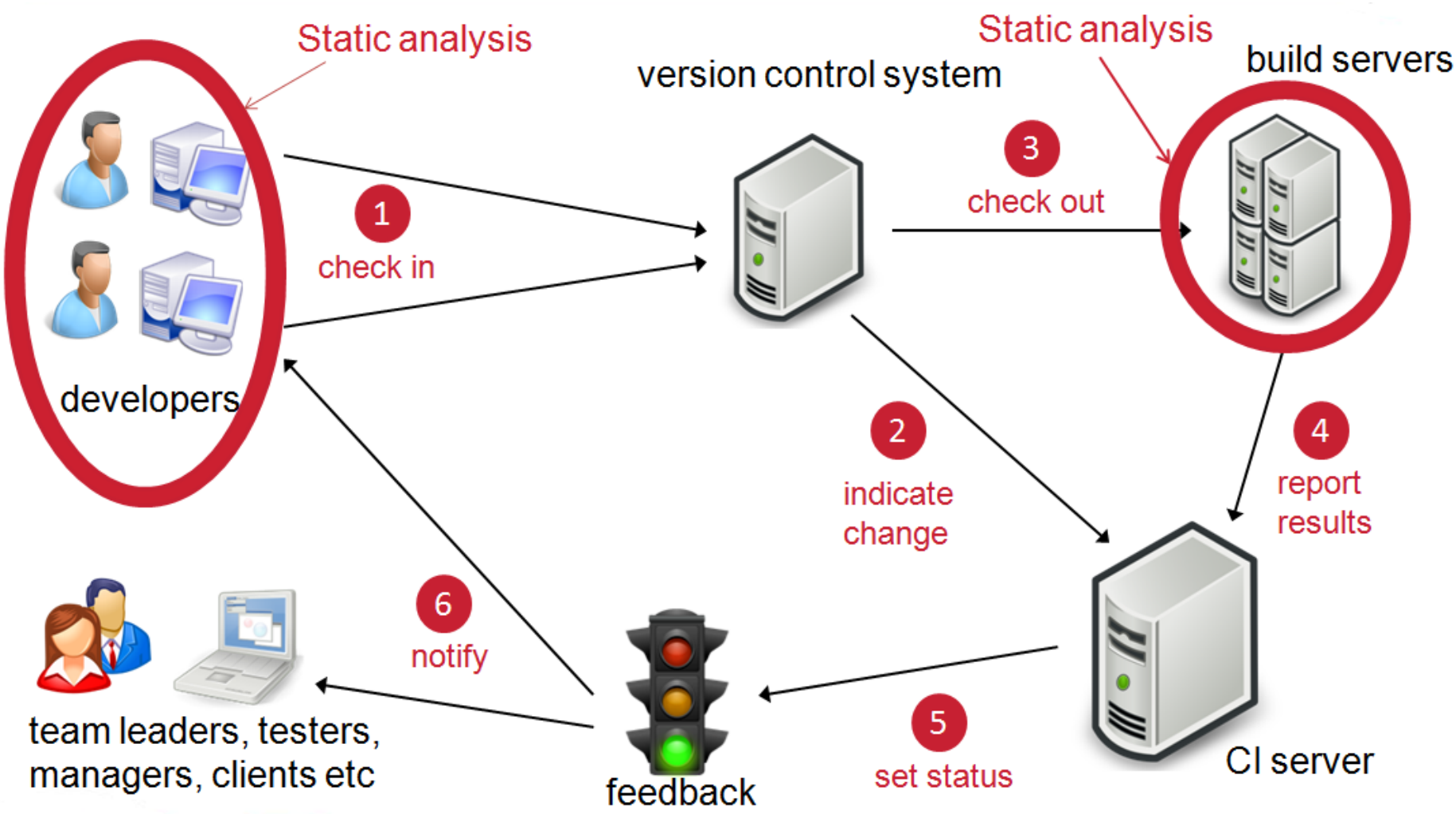
CI gépi feladatai

- (összetett) *Build* folyamatok automatizálása
  - Sorrendiség megadásával
- Egy „tiszta” készüléken futtatás
  - A fejlesztő hibázhat (pl. nem minden kerül a verziókezelőbe)
  - Lokális beállítások / erőforrások a fejlesztői gépen megfelelőek, azonban lehet, hogy a céleszközön kimaradt
  - Biztosíték a működő szoftverre
- További funkciók csatolása
  - Automatizált tesztek futtatása
  - Kódelemzés

Continuous Integration lehetőségei

- Integráció verziókezelőkkel
  - Értesülés *push*-ról
- Teszt riportok generálása
- *Push*-olás különféle *artifact repository*-kba
  - Minden build kimentének automatikus mentése
- Élesítés (deployment) éles- és tesztkörnyezetekbe
- A projekten dolgozók értesítése
- Egyéb tool-okkal való integrálás
  - Pl build rendszerek, mint ant, maven, gradle
- És sok egyéb

Egy tipikus CI rendszer



CI célja

Általánosan tehát a cél az, hogy a céges folyamatok, melyek emberi interakciót is igényelnek, sorrendiséget követelnek meg, egyetlen nagy, összefüggő, teljesen automatizált flow-vá váljanak.

A Continuous Integration eszközök

Szűkebb értelemben…

- *Continuous Integration* eszköz esetén jellemzően egy-egy *CI*-t megvalósító megoldásra asszociálhatunk
  - Ez egy szoftver, ami integrál
  - Ez egy szerver, ami elérhető
- Ez nem egy rossz megközelítés, de valóban csak ennyiből áll?
- Önmagában is megold mindent?

Tágabb értelemben…

- Természetesen nem
- Nem is lenne értelme
  - Fontos, hogy könnyen illeszthető legyen
    - Jelenlegi folyamatainkhoz
    - Jelenlegi eszköztárunkhoz
- Ezért amikor egy CI rendszerről beszélünk, szokás minden komponensét beleértenünk
  - Mivel egy futási lánc hajtódik végre, ez jogos is, hiszen bármelyik elem kimaradása a teljes futás hibáját okozhatja
    - Kis hibatűrés beállítható

Continuous Integration eszközei

- Verziókezelő
  - Innen szedi ki a forráskódot a speciális CI eszköz
- Maga a *CI*-t megvalósító komponens (aki futtatja a build scripteket)
- Adatbázis
- Kódelemző
- Az *artifactory*
- Az alkalmazásszerver
- A ticket kezelő rendszer
- És az eszköz használói
  - A fejlesztő, a menedzser
  - A CI szakember (rendszergazda)
  - *Stakeholder*-ek: vezetőség, megrendelők

CI választáskor összehasonlítási szempontok

Saját host-olás	SaaS megoldás
Testreszabhatóság, bővíthetőség	Egyszerűség, átláthatóság
Védett szoftver	Open Source szoftver
Continuous Integration a fókuszban	Continuous Delivery is fontos szempont

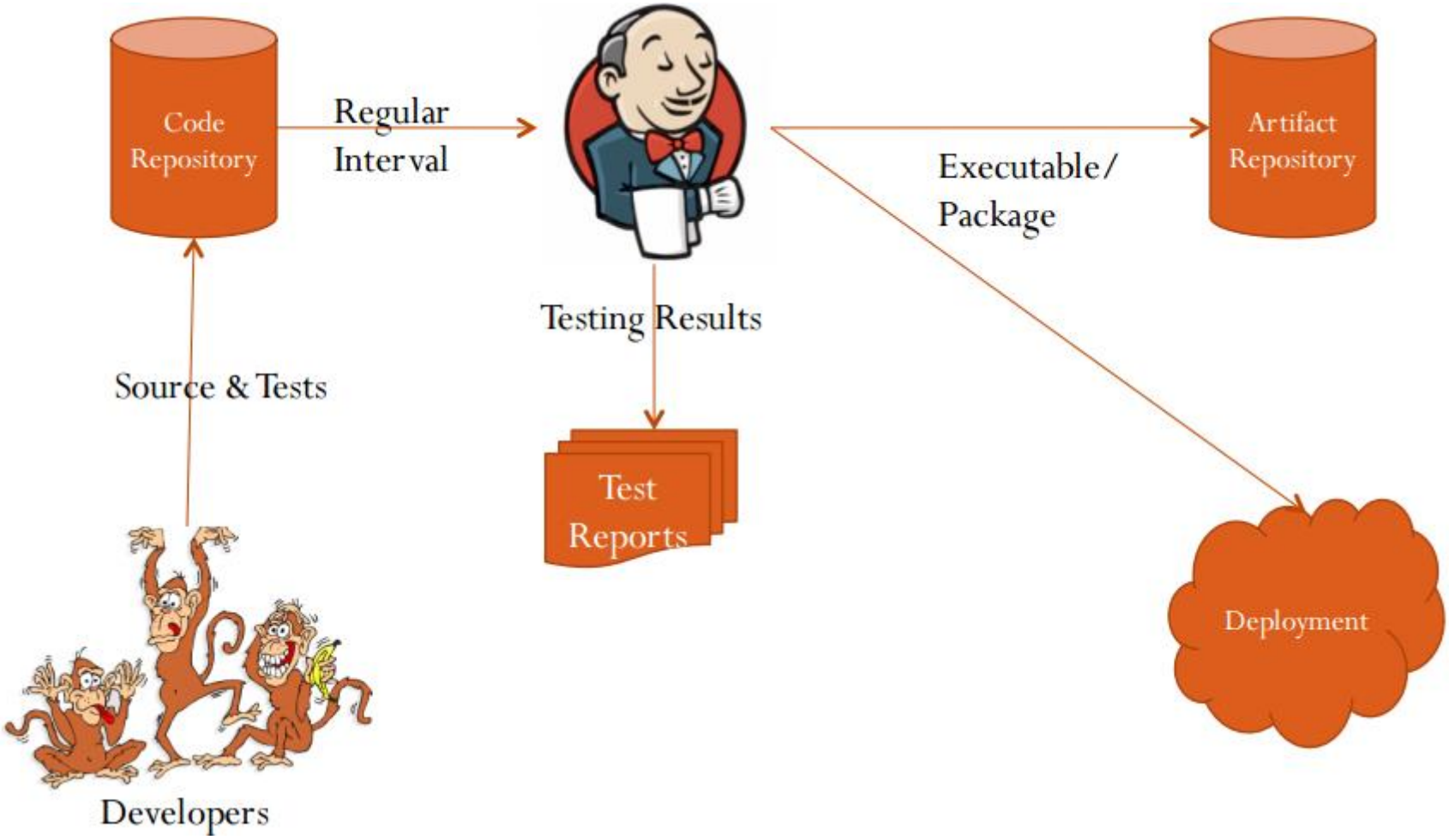
CI választás szempontjai

- Elterjedtség
  - Közösség és fejlesztői támogatás
- Kompatibilitás
  - Támogatott nyelvek
  - Támogatott platformok
- Árazás, licenszelés
  - A hostolás is költséges
- *Repository*-k támogatása
- Értesítési módok

Népszerűbb CI eszközök

- ***Jenkins***
- *(Hudson)*
- *TeamCity*
- ***Codeship** (Cloud alapú)*
- ***Travis CI** (Cloud alapú)*
- ***Circle CI***

CI eszköz működése



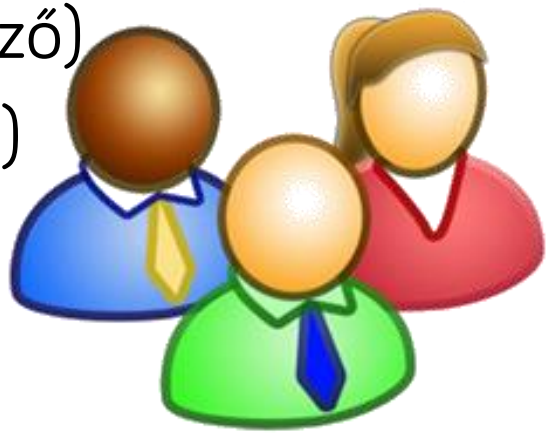
Jenkins

Jellemzői

- Egyik legelterjedtebb megoldás
  - Ingyenes
  - Elérhető *számos platformra*
  - *Minden ismertebb repository támogatása*
- Egyszerű telepítés
  - *Plug 'n play* jellegű
    - Elérhető platformspecifikus telepítők és WAR formájában is
  - *Standalone Tomcat* szervert magától telepít (*default port*: 8080)
- Gazdag funkcionalitás, testreszabhatóság
- REST interface a vezérléshez
- Bővíthetőség
  - 1000+ *plugin*
  - Ebből sok integrációt biztosít más (pl. *ticket* kezelő *JIRA*) rendszerekhez

Hozzáférés szabályozás

- A hozzáférés felhasználói fiókhoz kötött
  - Saját beépített megoldás
  - Delegálható *servlet* konténerhez is
  - *LDAP*
  - Számos egyéb megoldás plugin-okkal
- Felhasználónként állítható jogosultságok
  - Projektenként (adott projektre jellemző)
  - Műveletenként (általánosan jellemző)



Projektek kezelése

- Eltérő tulajdonságokkal rendelkeznek
- Eltérő konfiguráció
- Eltérő technológia
- Más célszerver
- …
- Ezek miatt fontos, hogy elszigetelve legyenek a projektek
  - Egy projektet érdemes egy CI eszközön használni csak
- Azonban nem megkötés, hogy egy cég csak egy(féle) CI eszközt használjon
  - Nagy cégeknél lehetetlen is, túlterhelt lenne

Item

- Önálló és szeparált egység
- A CI szerver kifejezetten alkalmas sok projekt együttes kezelésére
  - A projektek egyesével konfigurálhatóak
- A *Jenkins*-ben a projektek kezelőit ***Item***-eknek nevezzük
  - Projektenként egy *Item*-et hozunk létre jellemzően
    - Vannak kivételek
- Az egyes *Item*-ek tartalmazzák a projektre jellemző beállításokat

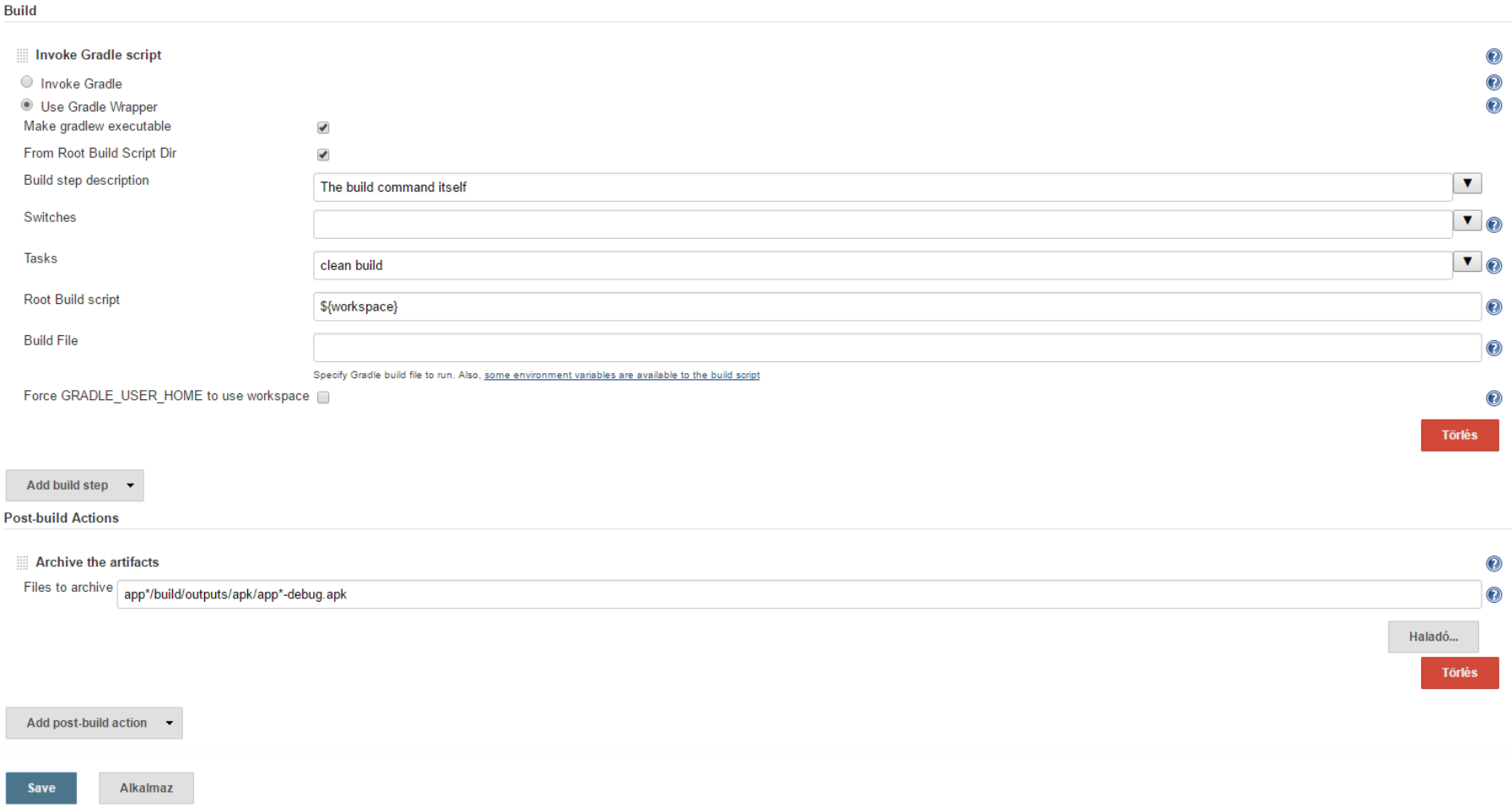
Item jellemzői

- Új *item* létrehozásánál választhatunk, hogy milyen típusú projektet szeretnénk létrehozni, ezzel gyorsítva a konfigurációt
  - *Maven project*
    - *Maven 2, 3* projektek esetén
  - *Freestyle project*
    - Bármilyen verziókezelő bármilyen *build* rendszerrel konfigurálható, akkor ajánlott használni, ha a többi opció nem megfelelő (több konfiguráció)
  - *External Job*
    - Érdemes használni, ha külső automatizálási rendszerrel rendelkezünk és ez a rendszer végzi az automatizálás feladatát. A *Jenkins* ekkor *dashboard* jellegű funkciókat lát el.
  - *Multiconfiguration Job*
    - Összetett, több komponensből (például különböző konfigurációval rendelkező tesztszerverek) álló, esetleg több technológiát támogató, több *build* rendszerre épülő megoldásoknál ajánlott.
  - Meglévő projekt klónozása
    - Létező és vagy megegyező, vagy nagyon hasonló beállítások esetén

Item funkciói

- Figyeli a projekthez tartozó *repository*-t (adott *branch* – *master* – javasolt)
  - *Git*
  - *Svn*
  - *Mercurial*
  - *…*
- *Build*
  - Kódelemzés (opcionális)
  - Tesztek futtatása és kiértékelése (opcionális)
  - Eredménytermék archiválása (opcionális)
- *Post-build akciók*
  - *Pl. APK eltárolása, e-mail küldés, stb.*
- *Deploy (opcionális)*
- *Feedback (opcionális)*

Új Item beállításai (részlet)



Scriptelhetőség

- A folyamat testreszabásához széleskörű eszköztárat kínál a *Jenkins*
- A folyamat részeként futtathatóak *script*ek is
  - *Build* előtt és/vagy után
  - *Batch, Shell*
  - A rendszer számára elérhető (pl. parancssorból futtatható) erőforrások hozzáférhetőek
  - Példák
    - fájlok másolására
    - fájlok módosítására
- *Plugin*ek segítségével szinte a végtelenség tárháza válik elérhetővé

Artifactok kezelése

- A *build* folyamat eredménytermékei
- Felhasználási lehetőségek
  - Feltölthetjük (saját) *artifactory*nkba (pl. *Maven* projektnél)
  - Felhasználhatjuk más projektünkben függőségként
  - Feltölthetjük alkalmazásszerverre
    - *WAR*-t telepítünk *Tomcat* szerverre
    - APK-t menthetünk el vagy telepíthetünk
    - Stb.

Más item buildjének indítása

- Tegyük fel, hogy másik projektünk használja függőségként egy *artifact*unk
  - Egyszerűség kedvéért lokális *JAR*-ként tartalmazva azt
- Az *artifact build*elésénél szeretnénk újrafordítani automatikusan a másik *item*ünk, de már az új *artifact*tal
- FONTOS!!!
  - Ha valamilyen meglévő *interface*-t változtatunk / megszüntetünk, az hibához vezet!
  - Akkor használjuk, ha *bug*ot javítottunk például

Saját monitoring dashboard

Mire jutottunk eddig?

- Tudjuk fordítani projektjeink egy független gépen
  - Ezzel elértük, hogy minden beállítás és erőforrás rendelkezésre áll (pl. *properties* fájlok, adatbázis, *maven* függőségek)
  - Kivédjük vele a *cache*-elés okozta inkonzisztenciákat is
- Automatikus kódelemzés történik a forráskódon
  - Ezzel is javítva a kódminőséget, illetve betartatva a kódolási konvenciókat
- Ezeket a folyamatokat sorosítva, egy flow-ban végezzük

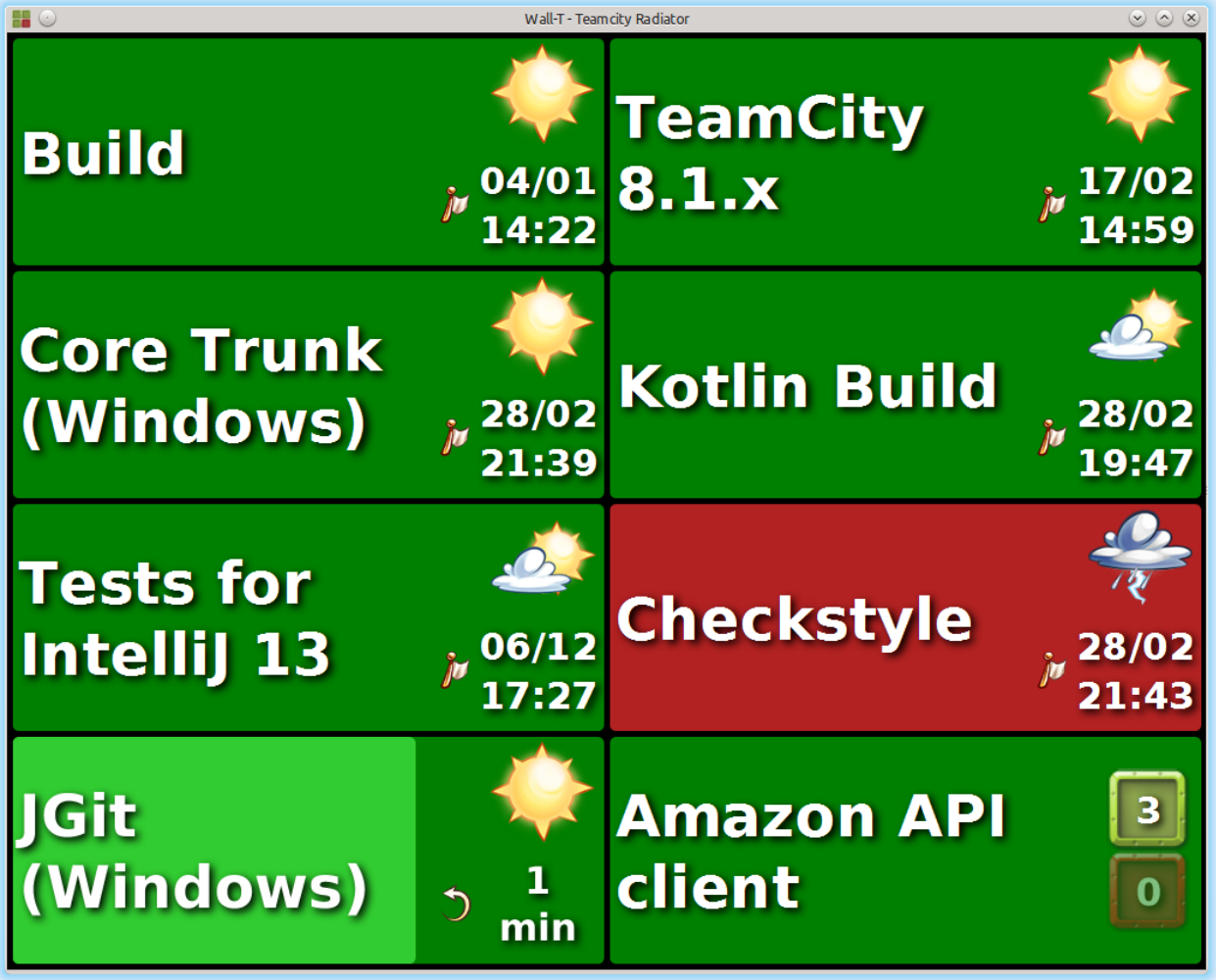
Probléma

- Nem szeretnénk a projektek eredményeit külön nézeteken követni, ha egyen is követhető
  - Például a kódminőséget
  - A fordítás sikerességét
  - Teljesítménytesztek eredményét
- Jellemzően egy cég nem csak egy projektet visz
  - Több projekt párhuzamos követése a cél
    - Szükséges a vezetőknek, akik több projektért is felelnek
    - Motiválhatja a projektcsapatokat, ha látják a többiek előrehaladását is

Megoldás

- Saját monitoring *dashboard* készítése
  - Használható a *Jenkins* beépített megoldása
    - Saját nézetek projektre szűréssel, megjelenítendő adatok választásával is létrehozhatóak
  - Az imént említett problémákra megoldás
  - *Jenkins*ből *plugin*ek segítségével támogatott
  - Külön rendszerek is alkalmasak rá
    - A háttérben *REST* hívásokkal, melyeket a *plugin* elfed
  - Ma már számos kész megoldás közül választhatunk

Konkrét példa



Travis CI

Travis CI

- Online megoldás
  - Telepítés nem szükséges saját szerverre
- Ingyenes verzió
  - Megkötések betartásával
- *Repository*k támogatása
  - Csak GitHub
- Integrálható a népszerű szerverekre
  - Pl. *Heroku*
- Nyelvek támogatása
  - Az elterjedt nyelvek támogatása (25 féle)
- Testreszabhatóság
  - Egy speciális ***.travis.yml*** fájlba leírható
  - Erősen korlátos lehetőségek

Travis CI helyi konfigurálása

- A projekt jellemzőit tartalmazó fájlt hozzuk létre:
  - **.travis.yml* , ahol * a projekt neve *sneak-case* formában
  - Ez a fájl leírja
    - A projekt típusát
    - Verzióját
    - Itt specifikálhatjuk a build előtti és utáni feladatokat
    - Esetleges CD beállításokat

Travis CI konkrét YML példa (Android)

```
language: android
android:
  components:
    - build-tools-23.0.1
    - android-23
    - android-21
    - add-on
    - extra
    - platform-tools
    - tools
    - extra-google-googleplayservices
    - extra-google-m2repository
    - extra-android-m2repository
    - addon-google_apis-google-19
    - sys-img-armeabi-v7a-android-21

deploy:
  provider: releases
  apikey: $GITHUB_KEY
  file: app/build/outputs/apk/app-release.apk
  skip_cleanup: true
  on:
    tags: true

notifications:
  email:
    recipients:
    - myemail@address.com
    onsuccess: always
    onfailure: always
```


Continuous Delivery

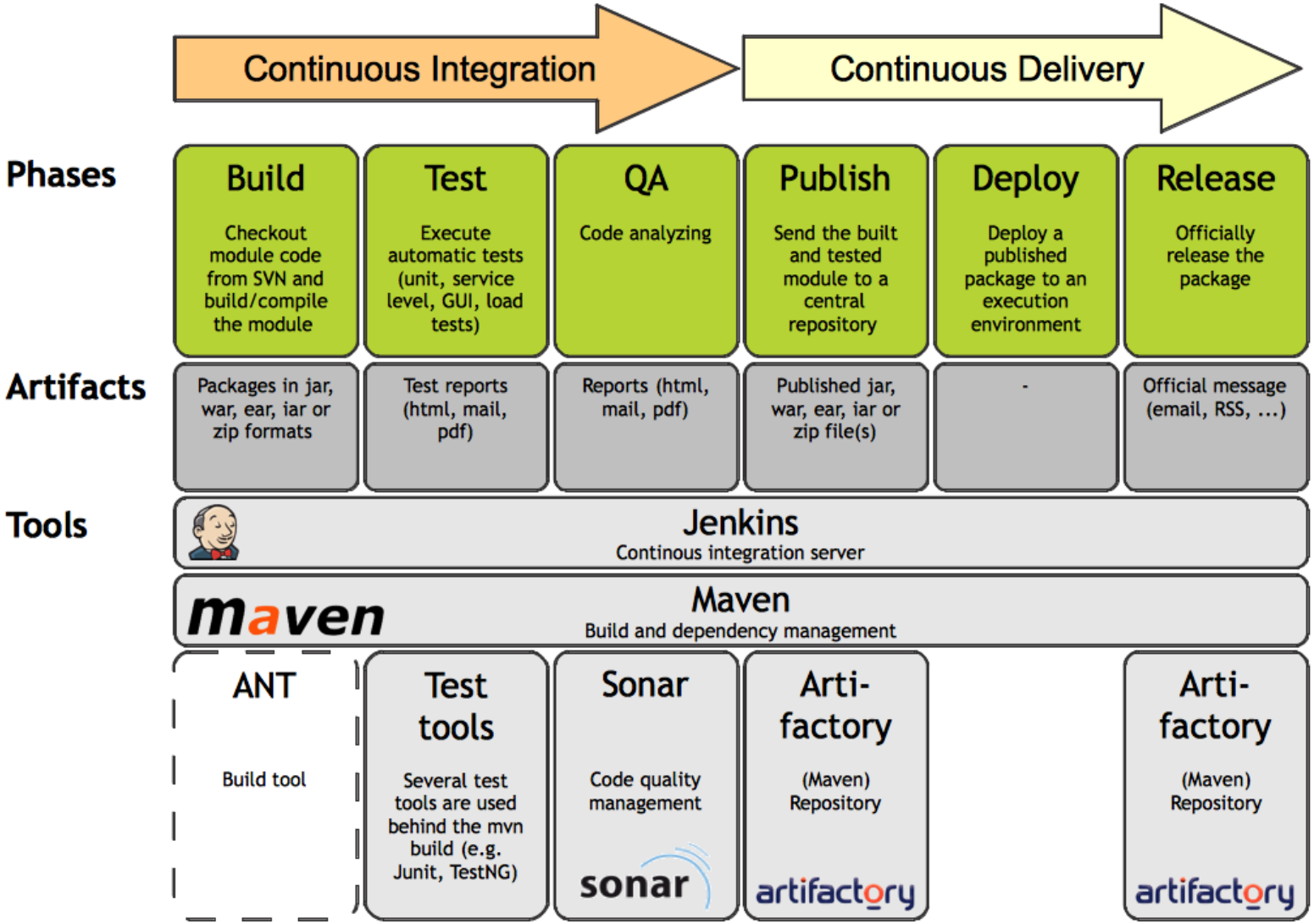
Hol tartunk?

- Tudjuk fordítani projektjeink egy független gépen
  - Ezzel elértük, hogy minden beállítás és erőforrás működik (kivédjük vele a cache-elés okozta hibalehetőségeket)
- Automatikus kódelemzés történik a forráskódon
  - Ezzel is javítva a kódminőséget, illetve betartatva a kódolási konvenciókat
- Tudjuk monitorozni egy elegáns, letisztult nézeten a projektjeink előrehaladást
- Ezeket a folyamatokat sorosítva, egy flow-ban végezzük

Mit érhetünk még el?

- A *flow* (*chain*) korábbi pontjainak stabil kimenetét szeretnénk felhasználni
  - Korábban: a fejlesztő lefordította a saját gépén a kódot, felmásolta egy szerverre, esetleg újraindította, manuálisan tesztelte
  - Ha nem volt tesztrendszer, az itt felmerült hibák az éles rendszer ideiglenes leállását okozták
- Ezen is szeretnénk javítani!

CI+CD, a teljes folyamat



Mikor használjuk a teljes folyamatot?

- Teszt szervereken nyugodtan
  - Hiszen ez egy speciális tesztelő kör számára hozzáférhető
  - Ha bugos, akkor se veszít üzleti presztízséből a cég
- Az éles szerveren / kliensen ne
  - Új verzióról értesíteni kell a felhasználókat, amint összeszedetten célszerű
    - Napi több frissítés már zavaró
    - Túl sok változást nehéz feldolgozni
  - Egy kiadás jellemzően néhány hetes fejlesztési funkciókat tartalmaz
    - Amit már teszteltek a tesztelők és a megrendelők
  - Ezt mindig kézzel csináljuk és alaposan ellenőrizzük, amint lehetséges

Több szerver kezelése

- Nap közben a *frontend*esek kértek az új részhez egy változtatást
  - A backendes megcsinálja, majd *push*ol egy local nevű *branch*re
  - A *Jenkins* érzékeli, majd futtatja a teszteket, stb.
  - Amennyiben minden sikeres, telepíti is a saját teszt szerverünkre
- Miért nem probléma ez?
  - Ha bármi tönkremegy, ez a sajátunk, nem egy éles rendszer, javítjuk
  - Ráadásul ez tényleg gyorsan kell, akár óránként többször is, ügyfél úgysem látja

Több szerver kezelése

- Érdemes még egy *remote test* nevű *branch*-et is készítenünk
  - Ennek a célja, hogy itt kezeljük az ügyfél teszt szerverét
  - A *Jenkins*-ünk (vagy az ügyfélé) figyeli ezt a *branch*-et
  - *Push* után, ha minden sikeres, kiteszi oda is
- Miért nem probléma ez?
  - Továbbra sem éles rendszer (nem számít a felhasználóknak, ha történik valami rossz)
- Miért hasznos?
  - Az ügyfél kér egy gyors változtatást, mert eszébe jutott, hogy valami jobban működhet és ki szeretné próbálni
  - Amennyiben nem biztosít ilyen teszt környezetet, jobb hiányában tesztelheti a mi teszt szerverünkön is
    - De tudjuk, hogy ez nem erre a célra van, gyakori változtatásokkal, nem vállaljuk ezért a felelősséget!

Források

- https://www.jfrog.com/confluence/display/RTF/Build+Integration
- http://www.quora.com/What-is-the-best-continuous-integration-deployment-server
- https://jenkins-ci.org/



Köszönöm a figyelmet!