CI / CD PIPELINE CU JENKINS

Eusebiu Rizescu[1]

În prezent, conceptul de CI/CD este intalnit in orice companie care dezvolda software. Atât CI cât și CD formează coloana vertebrală a mediului modern DevOps. Acest concept umple spatiul dintre echipele de dezvoltare și echipele de operations, automatizând build-area, testarea și deployment-ul aplicațiilor. Scopul acestui document este de a identifica, clasifica și compara majoritatea tool-urilor majore care au legatura CI / CD și după aceea sa construiesc un sistem complet pipeline.

Nowadays, the concept of CI/CD is everywhere. Both CI and CD form the backbone of the modern DevOps environment. It bridges the gap between development and operations teams by automating the building, testing, and deployment of applications. I aim to identify, classify and compare many major tools that are related to CI/CD pipeline and after that build an complete pipeline system.

Cuvinte cheie: CI/CD, Jenkins, Pipeline, Build, Deploy

1. Introducere

Viteza cu care companiile pot introduce produse pe piață este esențială pentru susținerea avantajului concurențial, iar reducerea timpului ciclului de dezvoltare a produsului a devenit un obiectiv strategic pentru multe firme care se bazează pe tehnologie. Conceptul de CI/CD explorează modul în care acest lucru poate fi realizat și oferă câteva exemple practice și factori de succes.

Un pipeline CI / CD ajută la automatizarea pașilor în procesul de livrare a software-ului, cum ar fi inițierea automata a buildurilor, rularea testelor automate și deploy-ul într-un mediu de testare, stage sau de producție. Un pipeline elimină erorile manuale, oferă bucle de feedback standardizate pentru dezvoltare ajuta la micsorarea TTL-ului (Time to market).

In figura de mai jos se pot observa principale stage-uri prin care ar trebui sa treaca un produs software pana a fi pus in piata. Acestea sunt: Plan, Code, Build, Test, Release, Deploy si Operate. Cu ajutorul unui pipeline, acest flow este initiat automat la fiecare comit.

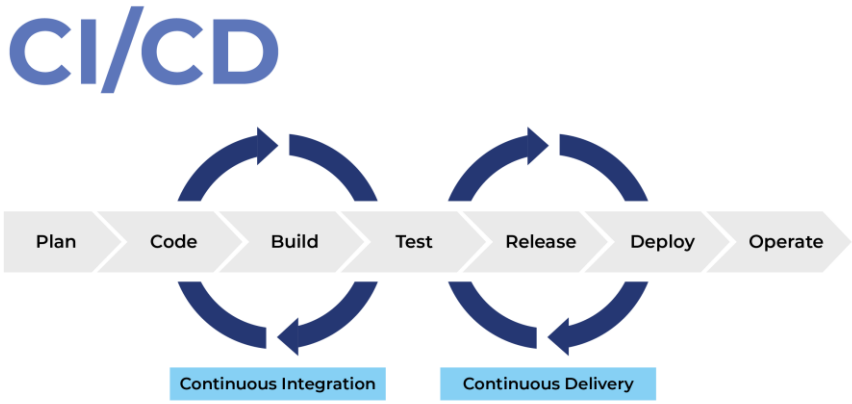

Fig. 1 - Stage-uri pipeline[1]

[1]Facultatea de Automatică şi Calculatoare, Universitatea POLITEHNICA din Bucureşti, România, e-mail: rizescueusebiu@gmail.com

1. Ce inseamna CI/CD?

Continuous integration (CI), continuous delivery (CD) si continuous deployment (CD) includ o cultură, un set de principii de funcționare și o colecție de practici care permit echipelor de dezvoltare a aplicațiilor să furnizeze mai frecvent și în mod sigur modificările de cod. Implementarea este cunoscută și sub denumirea de conductă CI / CD[2].

Continuous Integration este un set de practici care determină echipele de dezvoltare să pună în aplicare mici modificări și să verifice frecvent codul din sistemele de management al codului sursa (Git, Mercurial, SVN) . Deoarece majoritatea aplicațiilor moderne necesită dezvoltarea codului pe diferite platforme și versiuni, echipa are nevoie de un mecanism pentru integrarea și validarea modificărilor sale cat mai rapid si eficient.

Dezvoltatorii care practică continuous intergration își merge-uiesc modificările in branch-ul principal cat mai des posibil. Modificările dezvoltatorului sunt validate prin crearea unui build și executarea de teste automate pentru build-ul nou creat. Făcând acest lucru, se evita eventualele probleme de integrare care se întâmplă de obicei atunci când oamenii așteaptă ziua lansării pentru a-și contopi modificările în branch-ul principal.

Continuous integration pune un accent deosebit pe automatizarea testelor pentru a verifica dacă aplicația este in continuare functionala ori de câte ori noi modificari sunt integrate în branch-ul principal.

Continuous Delivery începe acolo unde se încheie continuous integration. Continuous delivery automatizează livrarea de aplicații în mediile de infrastructură selectate. Majoritatea echipelor lucrează cu mai multe medii, altele decât producția, precum mediile de dezvoltare și testare, iar aceste practici asigură că există o modalitate automatizată de a împinge modificările codului in acele medii.

Continuous delivery este o continuare a integrării continue pentru a asigura că release-ul rapid de noi schimbări clienților într-un mod durabil. Acest lucru înseamnă că, pe lângă faptul că testarea este automatizata (din CI), a fost automat si procesul de release și in consecinta aplicația poate fi deploy-ata in piata în orice moment, făcând clic pe un buton.

Teoretic, cu ajutorul continuous delivery, se poate decide să frecventa release-urilor: zilnice, săptămânale, sau orice se potrivește produsului si circumstantelor. Cu toate acestea, pentru a profita la maxim de beneficiile livrării continue, deploy-ul in productie trebuie facut cat mai devreme cu putinta, pentru ca modificarile sa fie mici, iar in cazul in care apar probleme, acestea sa fie usor de remediat.

Continuous Deployment merge cu un pas mai departe decât continuous delivery. Cu această practică, fiecare modificare care trece prin toate etapele conductei de producție este eliberată clienților. In cazul continuous deployment, spre deosebire de continuous delivery, nu există nicio intervenție umană și doar un test eșuat va împiedica o nouă schimbare să fie implementată în producție.

Continuous Deployment este o modalitate excelentă de a accelera bucla de feedback cu clienții și de a face reduce nivelul de presiune al echipelor de dezvoltare si operatii, deoarece nu mai există o zi fixa pentru un nou release. Dezvoltatorii se pot concentra pe

dezvoltarea de cot și își văd modificarile in productie la câteva minute (ore in functie de caz) după ce au terminat de lucru.

Diferenta dintre dintre **continuous delivery** and **continuous deployment** poate fi confuză din cauza prescurtarii (CD) si din ca cauza ca ambele au responsabilități foarte similare. Delivery este precursorul deployment-ului. La delivery, există o etapă finală de aprobare **manuală** înainte de lansarea in productie.

În faza de **delivery**, dezvoltatorii vor examineaza și îmbina modificările de cod care sunt apoi ambalate într-un artefact. Acest pachet este apoi mutat într-un mediu de producție unde așteaptă aprobarea să fie pus in piata.

În faza de **deployment**, pachetul este deschis și revizuit cu ajutorul unui sistem de verificări automatizate. Dacă verificările nu se termina cu success, pachetul este respins si nu este pus in piata. In cazul in care verificările trec, pachetul este deploy-at automat în producție. Continuous deployment este un pipeline de deploy complet automat de software.

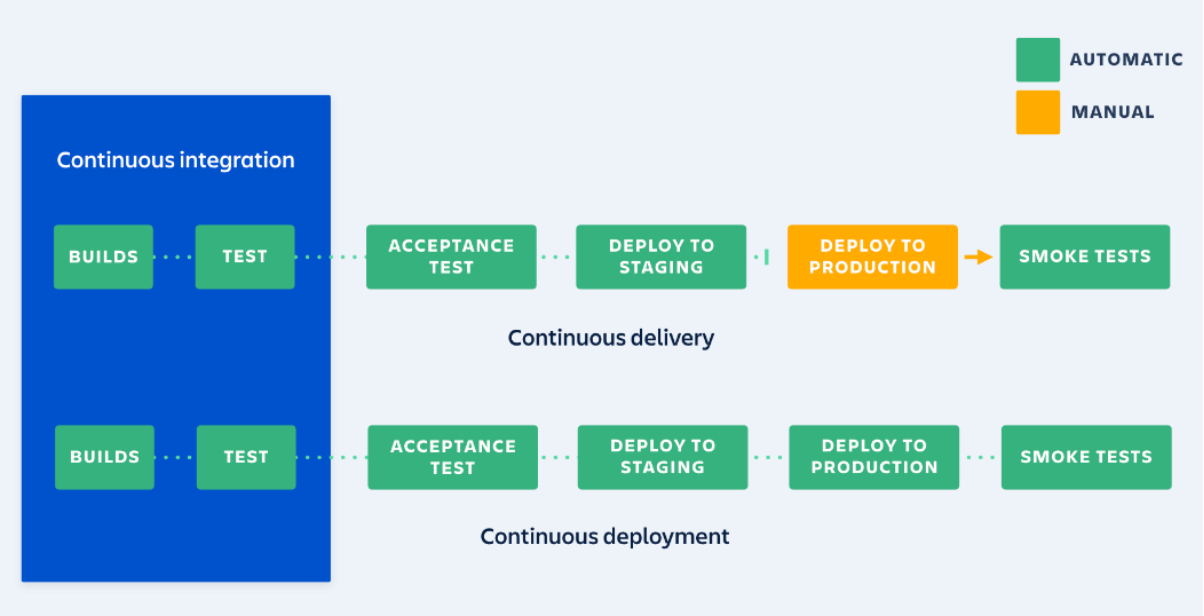

Fig. 2 Reprezentare a CI/CD[3]

2. Descriere in detaliu a unui pipeline

Pe scurt, implementarea unui pipeline CI / CD ajută la automatizarea pașilor în procesul de livrare a software-ului, cum ar fi pornirea automata a buildurilor de aplicatie, rularea testelor automate și in final deploy. Un pipeline bine pus la punct elimină erorile manuale, oferă bucle de feedback standardizate pentru dezvoltare ajuta la micsorarea TTL-ului (Time to market) si la imbunatatirea calitatii codului.[4]

Un pipeline CI / CD poate suna ca un concept in plus si nenecesar, dar chiar nu este. In esență un pipeline poate fi privit ca o specificație corectă a etapelor care trebuie efectuate pentru a livra o nouă versiune a unui produs software. În absența unei proceduri automate, programatorii/operativii ar mai trebui să efectueze aceste etape manual și astfel pot aparea erori manuale, pe langa timpul pierdut facand lucruri repetitive.

Majoritatea release-urilor desoftware trec in principiu prin urmatorii pasi:

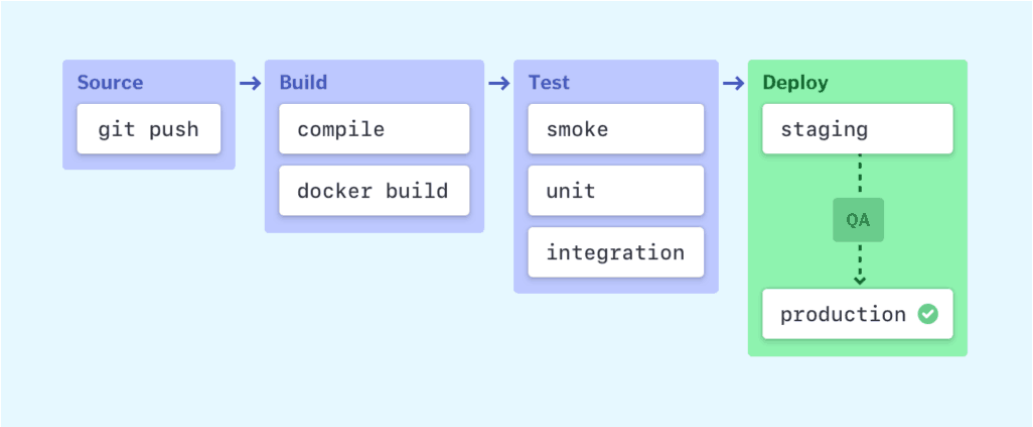

Fig. 3 Pipeline stages[5]

Source

În cele mai multe cazuri, o executie a pipeline-ului este declanșată de o modificare cod sursă din repository. O modificare a codului declanșează o notificare la tool-ul CI / CD, spre exemplu Jenkins, care rulează un pipeline predefinit anterior. Alte declanșatoare pot fi programate sau inițiate manual de catre un utilizator, precum și de catre alte pipeline-uri.

Build

In acest pas, este combinat codul sursă și dependențele sale pentru a construi o instanță executabilă a produsului care poate fi livrata eventual către utilizatorii finali. Spre exemplu, programele scrise în limbaje de programare precum C/C++, Java sau Go trebuie compilate, în timp ce programele Ruby, Python și JavaScript funcționează fără acest pas.

Indiferent de limbaj, software-ul cloud-native este de obicei implementat cu Docker, caz în care această etapă a CI / CD construiește imaginile Docker.

In cazul in care apare o eroare, pipeline-ul se opreste, si este recomandata interventia imediata pentru a rezolva problema, deoarece nu se mai pot release-ui versiuni noi in piata catre utilizatorii finali.

Test

In această fază sunt rulate teste automate pentru a valida corectitudinea codului și comportamentul produsului. Etapa de testare poate fi vazuta ca o plasă de siguranță care împiedică erorile care pot fi reproduse ușor sa ajunga la utilizatorii finali.

Responsabilitatea scrierii acestor teste automate revine dezvoltatorilor și se realizează cel mai bine în timp ce se scrie cod nou.

În funcție de dimensiunea și complexitatea proiectului, această fază poate dura de la câteva secunde până la ore. Multe proiecte pe scară largă efectuează teste în mai multe etape, începând cu smoke-tests care efectuează verificări rapide de corectitudine, până la teste de integrare end-to-end care testează întregul sistem din punctul de vedere al utilizatorului. Un set de teste mari este de obicei paralelizat pentru a reduce timpul de rulare.

Eșecul în timpul etapei de test descopera probleme in codul scris, probleme pe care dezvoltatorii nu l-au prevăzut la scrierea codului. Este esențial ca această etapă să producă

rapid feedback dezvoltatorilor, în timp ce spațiul cu probleme este scris recent, si poate fi rezolvat usor.

Deploy

Daca am ajuns in aceasta etapa a pipeline-ului, inseamna ca am creat o instanță executabilă a codului nostru care a trecut toate testele predefinite, si poate fi pusa in piata. De obicei, există mai multe medii de implementare, de exemplu un mediu „test" sau „stage", care este utilizat intern de echipa de produs și un mediu de „producție" pentru utilizatorii finali.

Echipele care folosesc modelul de dezvoltare Agile, care este ghidat de teste și monitorizare în timp real, de obicei, deploy-aza aplicatia intr-un mediu de testare (stage), pentru a face review si teste in plus, iar apoi se implementează modificările in producție.[6]

3. Comparatie tool-uri Source Code Management

Cele mai cunoscute sisteme de management a codului sursa sunt Git, Mercurial, SVN, CVS. In tabelul de mai jos este o comparatie intre acestea

Nume	Tip	Open Source?	Pret	Popularitate 2019
Git	Distribuit	Da	Gratis	71%
Mercurial	Distribuit	Da	Gratis	1%
SVN	Centralizat	Da	Gratis	24%
CVS	Centralizat	Da	Gratis	1%

Tab. 1 Comparatie Source Code Management

Git este un sistem de control de versiuni versiuni, distribuit, care este gratuit și open source conceput pentru a gestiona totul, de la proiecte mici până la proiecte foarte mari, cu viteză și eficiență.dsfsf

Mercurial este un instrument gratuit pentru controlului surselor, distribuit. Gestionează eficient proiectele de orice dimensiune și oferă o interfață ușoară și intuitivă.

SVN este un sistem centralizat de control al versiunilor. SVN era unul dintre cele mai populare sisteme. În timp ce popularitatea SVN scade, încă milioane de linii de cod stocate. SVN continuă să fie menținut în mod activ, deși de o mică comunitate open source.

CVS este un sistem de control al versiunilor, o componentă importantă a managementului versiunilor codului sursa Folosind SVN, se poate înregistra istoricul fișierelor surse și documentelor. CVS este un sistem de calitate a producției, avand o largă utilizare în întreaga lume, incluzând multe proiecte de software gratuit.

Ce inseamna centralizat si distribuit?

Model centralizat

CVS și SVN sunt cele mai cunoscute exemple în acest sens. În acest model exista un server central pe care este stocat codul sursa. Se poate descarca codul, se modifica și se împingeți înapoi pe server. Motivul pentru care acest model a fost atât de popular este că se potrivește bine modelul pe care majoritatea corporațiilor il folosesc.. Totul este controlat de la un server central.

Model distribuit

Git și Mercurial sunt exemplele principale pentru acest model. Principala caracteristica este disparitia serverului si a controlul central. In acest model orice bază de cod poate deveni serverul „central”. Acest model distribuit permite accesarea repository-ului local (offline sau online), ceea ce înseamnă că se poate verifica codul in orice moment, fără a a exista grija ca cineva pote strica codul altei persoane. Se poate utiliza, de asemenea, dacă nu aveți acces la internet.

Acest model se potriveste bine cu lumea open source. În proiectele open source, există sute de contribuabili, iar în timp ce există mai mulți dezvoltatori seniori, nu există „manageri” sau „proprietari”.

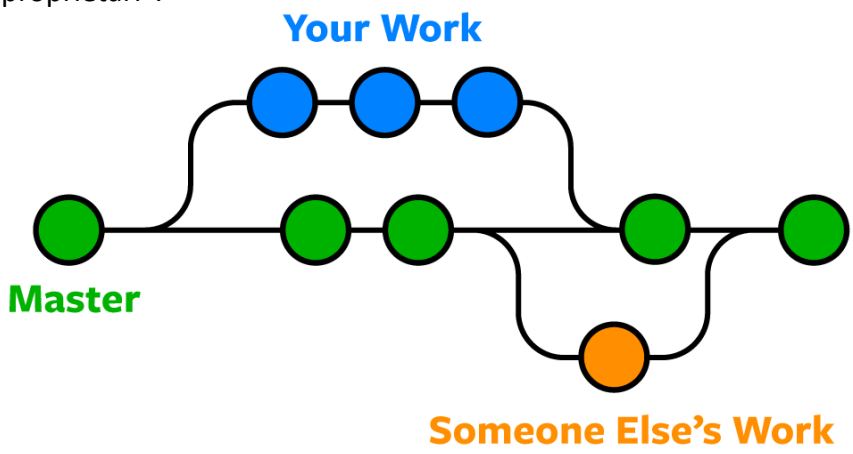

Fig. 4 Branch Git[7]

Deoarece Git este, in prezent, cel mai folosit sistem de management al versiunilor de control, in acest proiect voi folosi Git ca sistem de management al versiunilor. Aceasta alegere a fost usoara, deoarece Git este folosit majoritar in prezent, si un proiect nou care apare este de asteptat sa fie construit tot cu ajutorul Git.

Exista mai multe instrumente care ofera management de repository-uri. Este necesara folosirea unui astfel de tool deoarece ajuta la distribuirea codului intre echipe.

In tabelul de mai jos voi face o comparatie intre cele mai cunoscute tool-uri de acest tip: GitHub, GitLab, Bitbucket

Nume	OpenSource?	Suport proiecte OpenSource	Cloud	On Prem	Pricing Cloud	Pricing OnPrem
GitHub	Nu	Da	Da	Da	Free	Platit
GitLab	Da	Da	Da	Da	Free (basic)	Free (basic)
Bitbucket	Nu	Da	Da	Da	Free (<5 useri)	Platit

Tab. 2 Comparatie platforme Git

GitHub este cel mai bun loc pentru a împărtăși codul cu prietenii, colegii, colegii de clasă și străinii completi. Github are peste 40 de milioane de utilizatori, inclusiv 10 milioane care s-au alăturat platformei în 2019. Github reprezintă peste 44 de milioane de depozite create anul trecut.

GitLab oferă gestionarea depozitelor Git, recenzii de coduri, urmărirea problemelor, fluxuri de activități și wikis-uri. GitLab se poate instala on-premise și se poate conecta cu serverele LDAP și Active Directory pentru autentificare și autorizare sigură. Un singur server GitLab poate gestiona mai mult de 25.000 de utilizatori, dar este de asemenea posibil să se creeze o configurare de disponibilitate ridicată cu mai multe servere active.

Bitbucket oferă echipelor un loc unde să planifice proiecte, să colaboreze la coduri, să testeze și să implementeze, toate cu depozite private Git gratuite. Echipele aleg Bitbucket, deoarece are o integrare Jira superioară, CI / CD integrat și este gratuit pentru echipele de până la 5 utilizatori.

3. Comparatie tool-uri CI/CD

Piața de astăzi este inundată cu multe instrumente de CI/CD, astfel ca alegerea celui potrivit poate fi dificilă. Chiar dacă unele instrumente par a fi mai populare decât altele, exista posibilitatea sa nu functioneze perfect pentru fiecare companie. In figura de mai jos, se poate vedea o analiza in detaliu a celor mai cunoscute tool-uri pentru CI/CD de pe piata.

CONTINUOUS INTEGRATION TOOLS COMPARISON

	Jenkins	TeamCity	Bamboo	Travis	Circle	Codeship
Pricing	Free	$299-$1999	$10-$800	$69-$489	$50-$3150	$75-$1500
Operating system	Windows, Linux, macOS, any Unix-like OS	Windows, Linux, macOS, Solaris, FreeBSD, and more	Windows, Linux, macOS, Solaris	Linux, macOS	Linux, iOS, Android	Windows, macOS
Hosting	On premise/ cloud	On premise	On premise/ Bitbucket as cloud	On premise/ cloud	Cloud	Cloud
Container support	✓	✓	✓	✓	✓	Yes for Pro version
Plugins	*****	****	**	****	***	****
Docs and support	Adequate	Good	Good	Poor	Good	Poor
Learning curve and usability	Easy	Medium	Medium	Easy	Easy	Easy
Use case	For big projects	For enterprise needs	For Atlassian integrations	For small projects and startups	For fast development and high budget	For any project

Fig. 5 - Comparatie tool-uri CI/CD[8]

Criteriile care au fost luate in considerare in alegerea Jenkins, sunt faptul ca se poate utiliza on-premise, si faptul ca este free. Bineinteles, exista si alte tool-uri care sunt free si sunt disponibile si on-premise, dar Jenkins a fost selectat pentru faptul ca este cunoscut, si are o comunitate considerabila in jurul lui. Totodata, Jenkins poate lucra si cu containere, ceea ce este de folos in cazul in care se doreste build-area unei aplicatii intr-un container

4. Descriere tool-uri de testare

Testarea este un instrument esențial în viata de zi cu zi a oricărui dezvoltator de software serios. Cu toate acestea, uneori poate fi destul de dificil a se scrie un test bun pentru o anumită bucată de cod. Având dificultăți în a-și testa propriul cod, dezvoltatorii consideră adesea că problemele lor sunt cauzate de lipsa unor cunoștințe de testare fundamentale.[9]

Testarea continuă automată se potrivește cazurilor de utilizare în care testarea manuală nu oferă beneficii pentru o soluție automatizată, cum ar fi tot ceea ce face parte din UI: teste unitare, teste pe componente, teste API, etc.

SonarQube (numit in trecut Sonar) este o platformă open-source dezvoltată de SonarSource pentru inspecția continuă a calității codului pentru a efectua recenzii automate cu analize statice a codului pentru a detecta erori, bucati de cod care au potential risc și vulnerabilități de securitate pentru mai bine de 20 limbaje de programare. SonarQube oferă rapoarte despre secvente de coduri duplicate, standarde de codare, teste unitare, acoperire cod, complexitate cod, comentarii, bug-uri și vulnerabilități de securitate.

Checkmarx este o soluție de analiză statică, flexibilă și precisă pentru întreprinderi, utilizată pentru identificarea a sute de vulnerabilități de securitate în codul personalizat. Este folosit de echipele de dezvoltare, DevOps și de securitate pentru a scana codul, pentru a identifica vulnerabilitățile și pentru a oferi informații acționabile pentru remedierea acestora. Suport peste 22 de limbaje de codare și scripting.

5. Semestrele urmatoare

Planurile pentru urmatoarele 3 semestre sunt:

Semestrul 2: Inceperea configurarii mediului in care va fi construit acest pipeline. Va trebui decis daca se vor folosi masini virtuale pentru fiecare instanta, sau imagini Docker pentru fiecare instanta. Va trebui vazut si daca acest lucru se poate face pe laptopul personal, sau este nevoie de cloud.

Semestrul 3: Definitivarea sistemului de pipeline, construirea de planuri de build si de deploy pentru o aplicatie dummy, si verificarea de cod pentru aceasta.

Semestrul 4: Unificarea muncii anterioare, alaturi de documentarea, testarea si utilizarea sistemului de pipeline final.

6. Concluzii

Se observa numarul in continua crestere a companiilor care incep sa adopte implementarea si folosirea unui pipeline de CI/CD in viata de zi cu zi. Au trecut zilele cand companiile puneau in piata software anual sau semestrial. Cu ajutorul unui pipeline bine pus la punct, se poate face release chiar si de mai multe ori pe zi, si astfel, time to marked a scazut, greselile se pot repara mult mai repede, si se poate dezvolta cod mult mai usor.

CI/CD cu siguranta nu va disparea asa de curand, din contra, va deveni din ce in ce mai folosit atat de companiile mari cat si de companiile mici care dezvolta software.

B I B L I O G R A F I E

[1] https://www.mginfomatics.com/devops.html
[2] C. Singh, Comparison of Different CI/CD Tools Integrated with Cloud Platform, 2019
[3] https://petercoding.com/devops/2019/07/15/what-does-jenkins-do/
[4] M. Shahin, Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices, 2017
[5] https://www.michielrook.nl/2018/01/typical-ci-cd-pipeline-explained/
[6] https://semaphoreci.com/blog/cicd-pipeline
[7] https://www.nobledesktop.com/learn/git/git-branches
[8]https://www.altexsoft.com/blog/engineering/comparison-of-most-popular-continuous-integration-tools-jenkins-teamcity-bamboo-travis-ci-and-more
[9] https://dzone.com/articles/4-compelling-benefits-of-cicd