Nadgradnja obstoječega sistema za neprekinjeno integracijo in dostavo

Martin Potrč, Nejc Maleš, Matjaž Matjašec, Dušan Bratuša

Novum-RGI Germany GmbH Podružnica Maribor, Maribor, Slovenija
martin.potrc@novum-rgi.si, nejc.males@novum-rgi.si,
matjaz.matjasec@novum-rgi.si, dusan.bratusa@novum-rgi.si

V podjetju smo se odločili za nadgradnja orodja Jenkins iz različice 1 v različico 2 z namenom uvajanja metodologije zaporednega izvajanja podprocesov grajenja aplikacij, ki omogoča vzporedno delovanje in s tem povečuje učinkovitost. Ta nadgradnja je prinesla številne prednosti, med njimi boljši pregled in organizacijo opravil, ter olajšano odkrivanje in reševanje napak v programski kodi. Vizualizacija projektnih odvisnosti omogoča globlje razumevanje povezav med projekti. Kljub prednostim so se pojavili tudi izzivi. Med njimi so se izpostavljali potreba po poenotenju procesov med različnimi projekti in prehod s tradicionalnega načina konfiguracije na uporabo programske kode za določanje procesov. Ta prehod je zahteval prilagoditev miselnosti in učenje novih pristopov v upravljanju orodja.

Ključne besede:
Jenkins
neprekinjena dostava
neprekinjena integracija
vzporedno izvajanje
DevOps
programska konfiguracija
cevovod
vozlišče

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-772-0
DOI https://doi.org/10.18690/um.feri.8.2023.4

1 Uvod

Sistem za neprekinjeno integracijo in dostavo je eden izmed ključnih delov razvojnega procesa. Zagotavlja nam dodatno mero varnosti pri integraciji programske kode in obenem združuje informacije potrebne za analizo težav, ki lahko občasno nastanejo ob le tej.

Skozi zadnja leta je prišlo do sprememb pri procesu neprekinjene integracije. Uvedli so se novi postopki, ki so se bolje obnesli pri grajenju modernih aplikacij, pri katerih je za grajenje bilo potrebno izvesti več različnih faz.

V našem podjetju smo se tega lotevali z nadgradnjo procesa, ki smo ga že imeli in skušali proces grajenja aplikacij v več fazah nekako vzpostaviti z gradniki, ki so na voljo. To nam je skozi leta sicer dobro služilo ampak je postalo z uvedbo novih projektov domala nevzdržno. Za vsako novo stranko, za katero smo morali zgraditi aplikacijo, je bilo potrebno vzpostaviti proces grajenja:

- testiranje enot (angl. unit testing),
- dnevna namestitev (angl. daily build),
- test grafičnega vmesnika (z orodjem Selenium),
- izdaja posodobitev (angl. release).

To je posamezniku vzelo kar nekaj časa, saj je vzpostavitev tega procesa potekala bolj ali manj ročno.

Rešitev se je ponudila v obliki neprekinjene dostave in podpore v novejši različici Jenkins CI. Ovrednotili smo več alternativnih rešitev, ampak na koncu smo se odločili za Jenkins, saj nam je ponujal celostno rešitev našega problema. Prav tako smo že vrsto let uporabljali Hudson CI nato njegov derivat Jenkins CI in to nam je dalo dodatno zaupanje v to odločitev. Taka odločitev je seveda zelo pomembna, saj sistema za proces integracije izvorne kode v podjetjih načeloma na menjujemo prav pogosto.

Odločitev za uporabo Jenkins CI se je na koncu izkazala kot primerna. Prehod na novo različico ter na nov način grajenja je trajal kar nekaj časa in še ni popolnoma končan. Prestaviti je bilo potrebno približno 80 Jenkins opravil, med katerimi so bila opravila za izvajanje testov, dnevnih namestitev ter opravila za grajenje paketov za izdajo. Slednja so povzročala največ težav in so še zmeraj v fazi postopnega prestavljanja na nov proces.

2 Neprekinjena integracija, dostava in namestitev

Neprekinjena integracija (angl. continuous integration) je proces sprotne integracije (vsaj enkrat na dan) programske kode v sistem za upravljanje izvorne kode (SCM) in poganjanje avtomatiziranih opravil za izvajanje avtomatiziranih testov (Unit Test, Acceptance Test).

Neprekinjena dostava (angl. continuous deployment) je razširitev procesa neprekinjene integracije z cevovodom opravil, v katerem lahko zaporedno ali vzporedno izvajamo opravila potrebna za izgradnjo aplikacij oz. rešitev do te mere, da so pripravljene za namestitev bodisi v testna ali produkcijska okolja.

Neprekinjena namestitev (angl. continuous delivery) je razširitev procesa neprekinjene dostave, kjer obstoječ cevovod razširimo z avtomatizirano namestitvijo v bodisi testna ali produkcijska okolja. Uporaba tega procesa ni nujno zmeraj možna, saj lahko avtomatizirano nameščanje zahteva tudi večstopenjsko preverjanje kvalitete.

3 Glavne pomanjkljivosti in izzivi obstoječe rešitve

Čeprav smo obstoječo rešitev neprekinjene integracije skušali pragmatično uporabljati in z njo uvesti neko vrsto cevovodov, preko katerih bi lahko izvajali preproste cevovode (z uporabo projektnih odvisnosti), smo vendarle bili primorani preiti na modernejšo rešitev. Jenkins različica 1.0 je dosegla konec svoje življenjske dobe (angl. End of Life - EOL) že leta 2016 [1]. Takrat še tega nismo jemali tako resno, saj je sistem služil svojemu namenu v obstoječi konfiguraciji še kar nekaj let, preden je prišlo do resne diskusije o tem, da je potrebno tudi v tej smeri kaj postoriti.

EOL je sicer eden izmed glavnih razlogov za nadgradnjo oz. prehod, dejansko je pa razlogov seveda več. Četudi bi Jenkins 1.0 bil še zmeraj vzdrževan, bi bilo za dosego naših ciljev potrebno preiti na drugo rešitev.

Ključnega pomena pri razvoju modernih informacijskih rešitev je uporaba modernega sistema za neprekinjeno integracijo. Ena izmed ključnih praks v razvoju procesa neprekinjene integracije je uvedba cevovodnega povezovanja opravil, kar lahko krajše strnemo kot proces neprekinjene dostave. Jenkins 1.0 ne ponuja podpore za cevovode opravil (angl. job pipelining), kar otežuje implementacijo takega procesa. Rešitve za uvedbo neprekinjene dostave v Jenkins 1.0 so obstajale ampak niso dosegale željene kvalitete.

V obstoječi rešitvi z Jenkins 1.0 smo poskušali v opravila strniti tudi dodatne informacije o projektih, ki bi bile dostopne razvijalcem na enem mestu. Pojavila se je ideja, da bi lahko razvijalec na spletnem mestu opravila našel pomembne informacije o rezultatih Jenkins opravil, kot tudi dostopal do informacij o izvajalnih okoljih ter protokolih izvajanj. To je bilo možno izvesti v dokaj okrnjeni obliki v Jenkins 1.0 z veliko podvajanja informacij in naknadnega vzdrževanja le teh.

Vzdrževanje in uvajanje novih opravil v obstoječi rešitvi z Jenkins 1.0 smo izvajali ročno. Obstoječa rešitev ni ponujala avtomatizacije za opravila brez dodatnih razširitev. Sicer bi lahko to dosegli z alternativnimi rešitvami kot so Jenkins DSL vtičnik ali z Jenkins REST vmesnikom vendar teh rešitev nikoli nismo resnično ovrednotili. Ob poskusu uporabe te prve verzije DSL vtičnika je sama uporaba tega v našem CI okolju kazala na časovno zahtevno uvedbo. Prav tako smo se na tej točki že zavedali modernejših rešitev, kjer je integracija avtomatizacije privzeto dostopna v izvorni rešitvi (angl. out-of-the-box), zato smo uvajanje in ovrednotenje takih razširitev obstoječe Jenkins 1.0 enostavno opustili.

4 Evaluacija CI ponudnikov

Trenutno je na trgu ogromno rešitev za CI/CD, vendar smo pregled orodij zmanjšali na rešitve, ki so ustaljene, imajo dovolj veliko bazo uporabnikov in sprejemljivo ceno. S tem smo želeli doseči stabilnost in razširljivost orodja v prihodnosti.

Trenutno na trgu obstaja veliko rešitev, ki bi jih lahko razdelili v dve skupini. Inovativne (angl. cutting edge) in ustaljene ter razširljive. Največje razlike med tema kategorijama so novejši in lepši vmesniki z modernejšimi pristopi ali močno integracijo z drugimi orodji v nasprotju s funkcionalnostjo in razširitvami. Da bi zmanjšali nabor možnih opcij, je bila naša zahteva podpora naslednjih orodij: Git, SVN, Maven, Java, Docker. Zaradi velikosti projektov in trajanja izvajanja testov je bila dodatna zahteva gostovanje na lastni infrastrukturi. Nekatera orodja pa smo izločili, ker zahtevajo gostovanje izvorne kode v oblaku ali pa celo v odlagališču oziroma repozitoriju (angl. repository) specifičnega ponudnika, npr. BitBucket.

V enakem obdobju smo posodabljali več področij infrastrukture. Med njih spadajo tudi Git repozitorij, zato je potrebno omeniti GitLab in njihov CI. Čeprav nudi zanimiv pristop za ustvarjanje in zaganjanje cevovodov in zelo dobro integracijo s samim Git repozitorijem, so za nas nastale težave z vključitvijo SVN repozitorijev in pregledom nad testi ter okolji, ki se nameščajo.

Med trenutnimi rešitvami in na podlagi naših zahtev je bila izbira Jenkins 2.0 [2], dokaj enostavna in naravna. V primerjavi z ostalimi orodji je najbolj univerzalna rešitev in je ni potrebno združevati z dodatnimi orodji.

5 Cilji uvedbe nove CI rešitve

Namen nove rešitve za neprekinjeno integracijo in neprekinjeno dostavo (CI/CD) je bil doseči jasne cilje: poenostavitev postopkov z izgradnjo, testiranjem in dostavo programske opreme; učinkovita uporaba že vzpostavljenih postopkov; natančna sledljivost vseh sprememb; zmanjšanje operativnih nalog ekipe, ki skrbi za razvojne sisteme (angl. DevOps); samodejno vključevanje CI/CD procesov v skladu s spremembami v repozitorijih ter olajšanje nadaljnjega razvoja s pomočjo gradnje razširljive knjižnice.

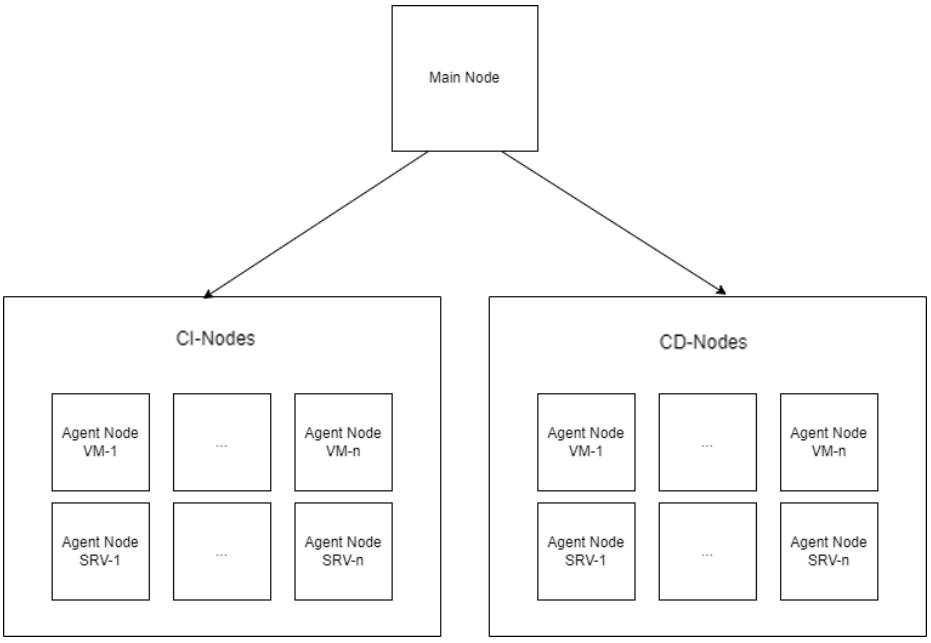
6 Postopek prehoda

Pri menjavi orodja smo imeli dovolj časa, virov in sodelavcev, da smo prehod dobro načrtovali in ga naredili postopoma. Postopek je bil razdeljen na različne faze: načrtovanje, vzpostavitev Jenkins 2.0 strežnika, vzpostavitev delovnih vozlišč (angl. node), implementacija cevovoda za CI za posamezen projekt, preoblikovanje in poenotenje CI cevovodov, implementacija CD cevovodov za dnevno namestitev, implementacija CD cevovodov za testiranje s Selenium orodjem, preoblikovanje in poenotenje CD cevovodov ter vpeljava skupne knjižnice, implementacija CD cevovodov za izdajo posodobitev ter nazadnje še implementacija cevovodov za administrativne naloge.

6.1 Infrastruktura

Po načrtovanju smo vzpostavili vzporeden strežnik za Jenkins 2.0. Da bi lahko zagotovili neodvisnost infrastrukture in dinamičnost vozlišč, smo se odločili, da bo eden izmed strežnikov deloval le kot osnovno vozlišče (angl. main node) brez izvajalcev in bo služil le kot nadzornik opravil. Vzpostavili smo tudi delovna vozlišča na virtualnih izvajalnih okoljih. Čeprav je vzpostavitev dodatnega izvajalca dokaj enostavna, smo odvisni od veliko orodij, ki jih ni mogoče vključiti preko Jenkinsa. Za te namene smo pripravili šablone za virtualna izvajalna okolja Unix in Windows, pri slednjih pa definirali še pravila preko skupinske politike (angl. group policy) objektov. Nastala je tudi skripta, ki z nekaj vhodnimi podatki avtomatizirano namesti izvajalca (agenta) in ga doda v delovno skupino.

Dodatna vozlišča smo vključevali vzporedno z migracijo. V delovne skupine vozlišč smo poleg virtualnih vključili tudi fizične računalnike.



Slika 5: Arhitektura vozlišč

6.2 Konfiguracija cevovodov

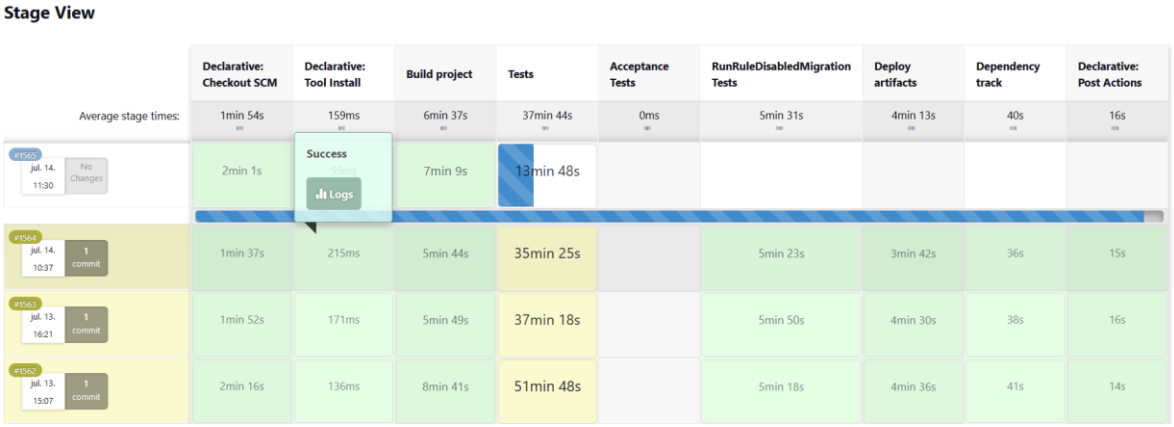
V želji, da bi poenotili in avtomatizirali čim več procesov, smo uvedli tok delovnih procesov z uporabo sledečih funkcij, ki jih Jenkins v verziji 2.0 nudi:

- Jenkins Multibranch Pipeline omogoča ustvarjanje osnovnih opravil (angl. Job), kateri prebirajo vse veje repozitorija po definiciji cevovoda.
- Definicija za izvajanje cevovoda je shranjena v samem repozitoriju kot datoteka s konfiguracijo in vsebuje izvorno kodo za cevovod spisano v jeziku Groovy. Groovy je skriptni jezik, ki se izvaja v JVM in lahko dostopa tudi do razredov in metod v Javi.
- Vpeljali smo tudi globalno knjižnico in preselili vso logiko iz cevovodov na projektih. Tako je ostala na projektih samo konfiguracija za same projekte, ki uporablja definirane cevovode v knjižnici. Na ta način lahko spremenimo cevovod za vse projekte hitro in enostavno.

6.3 Neprekinjena integracija – CI

Z vzpostavljeno infrastrukturo, smo lahko začeli z implementacijo cevovodov za CI, ki je razdeljen na devet faz, kar lahko vidimo na sliki Slika 6

1. Pridobitev izvorne kode (angl. checkout)
2. Deklarativna namestitev orodij (Maven, Java ipd.)
3. Prevajanje kode (angl. compile)
4. Izvajanje testov enot (angl. unit tests)
5. Izvajanje sprejemnih testov (angl. acceptance tests)
6. Opcijska faza za potrebe izvajanja testov pod določenimi specifičnimi nastavitvami (kot na primer za potrebe migracijskih projektov).
7. Nalaganje artefaktov v skupen repozitorij
8. Preverjanje paketnih odvisnosti (angl. dependency track) in varnostnih lukenj
9. Čiščenje delovnega prostora



Slika 6: Prikaz faz za CI na posameznem projektu

Cevovod je bil razvit in pripravljen na projekte. Preden smo migrirali vse projekte, smo naredili preoblikovanje in prestavili definicijo cevovoda v globalno knjižnico. S tem smo na projektu obdržali le minimalno konfiguracijo in možnost spremembe le te, brez da vpliva na ostale projekte. Vsi projekti so konfigurirani po enakem postopku kot kaže spodnji primer Groovy kode na sliki Slika 7.

Na ta način lahko definiramo privzeto obnašanje cevovoda na projektu, obstoječi parametri pa omogočajo tudi ročne spremembe.

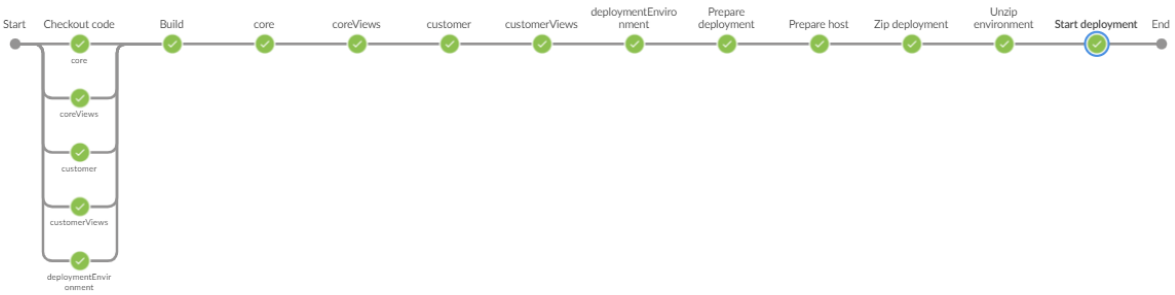
```
def timeout = 2

def mavenSettingsFile = 'build/JenkinsCI/maven-jenkins-settings.xml'
def upstreamProject = 'novum_demo_core/master'
def mavenVersion = 'Maven 3.8.7'

properties([
        parameters([
                booleanParam(name: 'skipTests', defaultValue: false, description: 'Whether to skip the unit tests'),
                booleanParam(name: 'skipAcceptanceTests', defaultValue: false, description: 'Whether to skip the acceptance tests'),
                booleanParam(name: 'skipRuleDissabledTests', defaultValue: true, description: 'Whether to skip the rule dissabled tests')
        ]),
        allowBrokenBuildClaiming()
])

defaultCIPipeline(
  [
    skipTests:skipTests,
    skipAcceptanceTests:skipAcceptanceTests,
    skipRuleDissabledTests:skipRuleDissabledTests,
    mavenSettingsFile:mavenSettingsFile,
    timeoutInHours:timeout,
    upstreamProject:upstreamProject,
    mavenVersion:mavenVersion
  ]
)
```

Slika 7: Primer konfiguracije za Jenkins cevovod

6.4 Neprekinjena dostava – CD

Nadaljevali smo s cevovodi za dnevne namestitve (angl. daily build). Poslužili smo se enakega pristopa za konfiguracijo in definicijo cevovoda. Ker so projekti med seboj sklopljeni, cevovodov nismo vključili v repozitorije projektov, ampak naredili namenskega, v katerem veje predstavljajo okolje, ki ga želimo namestiti. Ker so projekti v produkcijski rabi in imajo lastne razvojne poti, imamo namestitve za vse aktivne različice na projektih. Blue Ocean vmesnik na sliki Slika 8 predstavlja faze, ki so potrebne za namestitev okolja.

Pridobivanje izvorne kode iz repozitorijev se izvaja paralelno, grajenje pa zaporedno. Faze za ta del cevovoda se generirajo dinamično, glede na konfiguracijo okolja. V sklopu cevovoda še pripravimo gostiteljski sistem in preverimo ali obstaja kakšna namestitev tega projekta nato jo ugasnemo in odstranimo. Rezultat se arhivira in prenese na gostitelja, kjer se razpakira in namesti. Namestitev zajema namestitev servisov, pripravo baze, zagon okolja in zagon parametrizacije.



Slika 8: Blue Ocean pogled na Demo dnevne namestitve

6.4.1 CD – Selenium

Selenium okolja za nas predstavljajo okolja podobna okoljem dnevne namestitve, vendar so namenjena zagonu testov s pomočjo Selenium orodja, ki je namenjeno testiranju spletnih strani. V primerjavi s prejšnjo točko, ima cevovod le dodatno fazo zaganjanja testov in dodatno parametrizacijo za teste. Tako kot pri dnevnih namestitvah, se aplikacija prenese na gostitelja, kjer se sistem zažene, testi pa se izvajajo na Jenkins vozlišču. Ker se testira spletni vmesnik, se testi izvajajo počasneje. Uporabljamo malo večje časovne omejitve (na primer čakanje na obnovitev strani pri spremembi vnosnih podatkov), da smo prepričani, da vse funkcionalnosti delujejo, tudi če je v tistem trenutku na voljo manj virov. V primeru da zaznamo večje nihanje v učinkovitosti, izvajamo teste učinkovitosti ločeno.

V sklopu Selenium testov, zajemamo tudi zaslonske slike v primeru napake ali padlega testa, ki so dostopne direktno preko Jenkinsa, kar razvijalcem malo olajša iskanje napak.

V primerjavi z Jenkins 1.0, kjer so delovni tokovi delali avtomatizirano, dostop do informacij pa je bil otežen, potrebno je bilo poiskati in se povezati na gostitelja, na katerem smo ročno iskali te datoteke.

6.4.2 CD – Posodobitev

Tudi posodabljanje ima veliko korakov skupnih s CD cevovodi, saj je tudi tam potrebno pripraviti paket aplikacije za gostiteljski sistem. Razlikuje pa se v tem, da se ne proizvede arhiv aplikacije na podlagi zadnjega stanja kode, ampak na podlagi oznak verzije (angl. tag). Na tej točki smo se namenoma odločili, da se oznake verzij pripravijo vnaprej in ne v sklopu cevovoda za posodobitev. Obnašanje cevovoda se razlikuje glede na projekt, odvisno od tega, ali našo aplikacijo izdajamo le kot namestitveni paket ali pa kot paket v kombinaciji z aplikacijskim strežnikom, na primer v obliki celotnega vsebnika ali "Docker Image". Pripravimo lahko tudi različne arhive. Vse to je nastavljivo preko parametrov na grafičnem vmesniku, privzete vrednosti pa so definirane v konfiguraciji za posamezno stranko.

6.5 Administracija

Na Jenkinsu 1.0 smo imeli veliko več administrativnih oz. nadzornih opravil, ki pa smo jih prestavili na druga namenska orodja, kot so Zabbix, Grafana ipd. Največ opravil je bilo za preverjanje stanja namestitev in njihove dosegljivosti, tudi za produkcijska okolja, ter varnostno kopiranje internih sistemov, nastavitev in podatkov.

Trenutno so administrativni cevovodi namenjeni lažjemu vzdrževanju infrastrukture. Za potrebe Selenium testov, smo vezani na uporabo Firefox brskalnika in Gecko gonilnika. Zaradi omejitev iz strani teh dveh aplikacij, smo se odločili, da te sisteme posodabljamo kar preko Jenkinsa. S tem zagotovimo, da se vsa testna okolja in izvajanje testov obnaša enako. Ne pride do težav, da se na kakšnem strežnik/vozlišču pozabi ali spregleda. Na ta način lahko verzijo tudi fiksiramo, v primeru, da so najnovejše različice nekompatibilne.

6.6 Izzivi in težave pri razvoju

Po naši evaluaciji se je orodje Jenkins izkazalo kot najustreznejše. Prepričala nas je univerzalnost orodja za namene CI in CD, predvsem zaradi odprte kode in razširljivosti s pomočjo vtičnikov. Deloma so ti vtičniki razviti iz strani skupnosti (angl. community), za katere pa ni nujno, da se držijo standardov, da so dobro testirani ali dokumentirani. Ena izmed večjih težav je bila pri uporabi različnih vtičnikov in njihovim obnašanjem v sklopu naših cevovodov. Neobstoječa ali zelo površna dokumentacija nas je privedla do veliko tako imenovanih "poskus in napaka" pristopov (angl. trial and error) pri razvoju cevovodov.

Ko smo začeli ustvarjati cevovode, smo se odločili za definicijo cevovodov kot "konfiguracija v kodi" (angl. configuration by code). Za te namene sta na voljo skriptni in deklarativni cevovod. Oba uporabljata datoteko konfiguracije "jenkinsfile", ki je spisana v Groovy jeziku.

Skriptni cevovodi so najbolj zmogljivi, saj omogočajo direkten dostop in konfiguracijo do vseh modulov, v nasprotju z deklarativnimi cevovodi, ki so enostavnejši in bolj pregledni, kar je bil razlog, da smo se odločili za deklarativni pristop. Hitro smo ugotovili, da samo z deklarativnim pristopom ne bomo uspeli rešiti vseh zastavljenih ciljev in delovnih tokov. Odločili smo se, da vse gradnike poskušamo spisati v deklarativnem načinu in uporabimo skriptni pristop, kjer je potrebno. Kot smo ugotovili kasneje, je to za nas obvezno. Določeni vtičniki ponujajo uporabo samo preko Jenkins vmesnika ali pa preko skriptnega pristopa. Izogniti smo se morali tudi različnim vtičnikom, ki so namenjeni predvsem uporabi preko spletnega vmesnika in jih je skoraj nemogoče konfigurirati preko kode.

7 Zaključek

Splošno zadovoljstvo s prehodom na Jenkins 2.0 je bilo s strani naših razvijalcev izboljšano v primerjavi z različico 1.0. Sam razvoj in vzdrževanje Jenkins cevovodov je sicer še nekoliko težaven, vendar počasi pridobivamo izkušnje in sproti rešujemo različne izzive ter težave. V načrtu je tudi posodobitev samih Jenkins vozlišč in Jave, s katero se poganjajo. Zanimivo pri tem je, da samih vozlišč ni mogoče posodobiti preko Jenkins orodja. Edini način je, da na glavnem vozlišču sprožamo skripte, ki ta vozlišča obvestijo, da naj posodobijo določena orodja, kot so Gecko gonilnik, Firefox brskalnik za potrebe testiranja v Selenium ogrodju ipd.

V ekipi DevOps se že opaža manj ročnega posredovanja pri uvajanju novih cevovodov ali spreminjanju obstoječih. Razvijalci so namreč že navajeni, da smejo take spremembe urediti sami glede na zahteve, ki bi sicer prešle od njih samih na DevOps ekipo.

Literatura

[1] https://www.versio.io/product-release-end-of-life-eol-Jenkins-Jenkins.html, obiskano 17. 6. 2023

[2] https://www.jenkins.io/doc, obiskano 2. 6. 2023