Optimizacija uporabe CI/CD cevovodov, DevOps pristopa in oblaka

Mihael Škarabot

Cloudvenia, d.o.o., Kranj, Slovenija
mihael.skarabot@cloudvenia.com

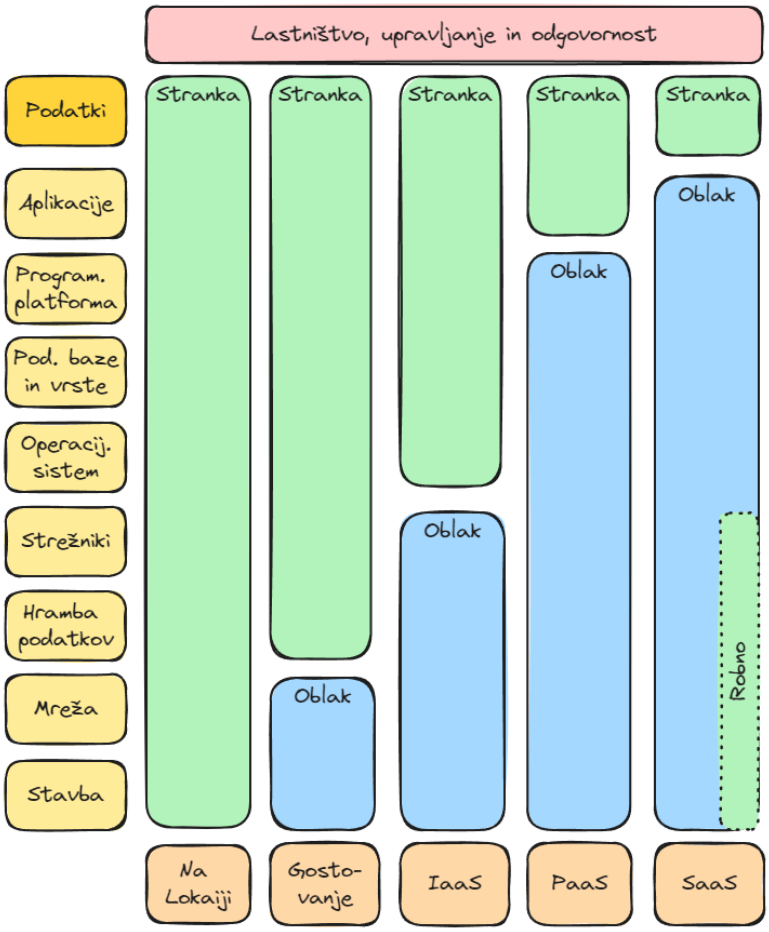
Članek obravnava ključne koncepte neprekinjene integracije, izdaje in nameščanja (CI/CD+CD) kot temeljne elemente za razvoj sodobne programske opreme za nudenje SaaS (ang. Software-as-a-Service). Poudarja pomen DevOps metod in naprednih arhitekturnih pristopov, kot so mikro storitve in modularni monoliti, ki omogočajo večjo skalabilnost, dostopnost, zanesljivost in učinkovitost rešitev. Različni tipi CI/CD cevovodov, prilagojeni specifičnim potrebam, optimizirajo proces razvoja in zagotavljanja programske opreme. Članek izpostavlja prednosti uporabe javnih oblačnih storitev oziroma hiperoblačnih storitev, ki zaradi svoje zanesljivosti, elastične skalabilnosti in transparentnih stroškov predstavljajo idealno infrastrukturo za globalne SaaS rešitve. Prav tako poudarja pomembnost avtomatizacije testiranja in posodobitev skozi CI/CD cevovode, kar prispeva k redkejšim napakam in večji učinkovitosti. Poleg tega članek obravnava strategije za učinkovito upravljanje stroškov v hiperoblaku, vključno z rezervacijami in varčevalnimi načrti, ter različne načine elastičnega skaliranja, kot so vertikalno in horizontalno skaliranje zabojnikov in strežnikov. Poudarja tudi pomen uporabe PaaS storitev za zmanjšanje kompleksnosti in izboljšanje zanesljivosti. Zaključuje z ugotovitvijo, da učinkovita strategija uporabe CI/CD cevovodov in javnih oblakov omogoča podjetjem hitrejše prilagajanje tržnim potrebam ter zagotavljanje visokokakovostnih storitev na globalnem trgu, kar je ključno za uspeh sodobnih SaaS rešitev.

Ključne besede:
oblak
hiperoblačne storitve
DevOps
cevovod
CI/CD
optimizacija

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-894-9
DOI https://doi.org/10.18690/um.feri.4.2024.5

1 Uvod

Neprekinjena integracija, izdaja in nameščanje (CI/CD+CD) so procesi, ki že desetletja predstavljajo ključen del razvoja programske opreme. V zadnjih letih so zaradi naraščajoče priljubljenosti programske opreme v obliki storitev (ang. Software-as-a-Service, SaaS) v oblaku doživeli prelomnico (slika 1). Na podlagi naših izkušenj ugotavljamo, da uporaba sodobnih DevOps metod ter naprednih arhitekturnih pristopov, kot so mikrostoritve, modularni monoliti in celična arhitektura [1], prinaša številne prednosti pri razvoju SaaS rešitev za globalni trg. Tak pristop ne omogoča le izboljšane skalabilnosti in dostopnosti rešitev, temveč tudi povečuje njihovo zanesljivost in učinkovitost.



Slika 1: Poenostavljeni model oblačnega računalništva in opredelitve SaaS.

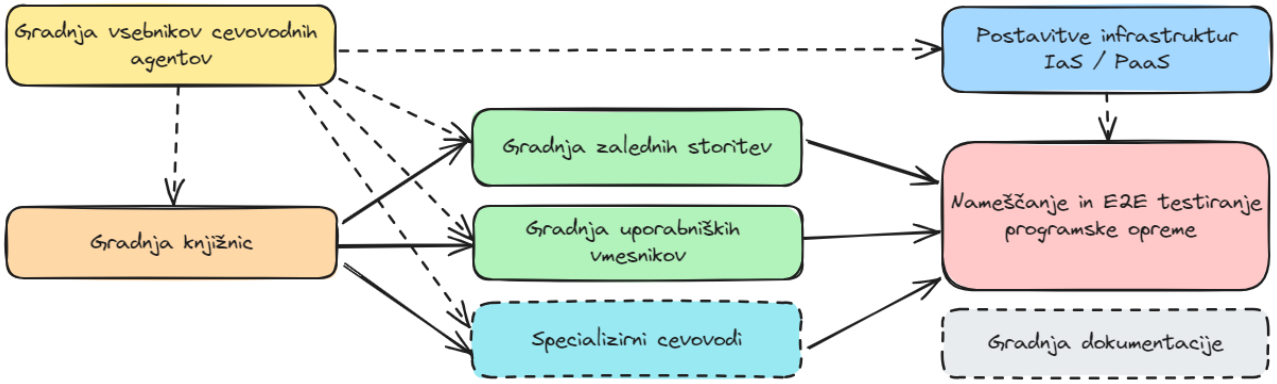
Razvoj programske opreme se lahko izvaja po različnih metodologijah, vključno s klasičnim slapovnim modelom, iterativnim, agilnim ali po pristopu DevOps [5]. DevOps inženirstvo predstavlja sodoben pristop, ki združuje razvojne (Dev) in operativne (Ops) procese v neprekinjen, integriran postopek in preprečuje »metanje čez ograjo«. Ta pristop poudarja sodelovanje, avtomatizacijo ter hitro povratno zanko med ekipami, kar omogoča hitrejše, učinkovitejše in bolj zanesljivo izvajanje programskih rešitev. Avtomatizacija v DevOps omogoča stalno in hitro uvajanje sprememb v produkcijsko okolje brez negativnih učinkov na raven stabilnosti in varnosti.

V slapovnem modelu lahko razvoj nove verzije programske opreme traja od nekaj mesecev do nekaj let. V agilnem razvoju se izdaje programske opreme zgodijo v krajših, rednih in pogostejših ciklih. DevOps pristop gre korak dalje in omogoča uvajanje sprememb v produkcijsko okolje na vsakih nekaj minut, uporabljajoč različne strategije, kot so namestitve kanarčka, modro-zelene namestitve, namestitve na osnovi obročev, zastavice za funkcionalnost, itd. Ta pristop maksimalno skrajša čas od ideje do produkcijske uporabe, poveča agilnost organizacije in izboljša odzivnost na potrebe trga – še posebno to velja za globalni SaaS model.

Globalni trg se zelo učinkovito doseže z uporabo hiperoblačnih storitev. Le-te so se v zadnjih letih izkazale za zelo zanesljive. Nemalokrat bi se jih radi izognili zaradi visokih stroškov, ki lahko nastanejo z nepremišljeno uporabo. Globoko v naši miselnosti je tudi še doktrina lastništva nad strojno opremo. Če želimo to premagati je potrebno oblačno računalništvo pogledati iz vseh vidikov. Primerjave le iz vidika neposrednih stroškov strojne opreme so nepopolne. Ne smemo pozabiti na stroške varnosti, elastične skalabilnosti, dostopnosti, zanesljivosti, tehnične podpore, podpore različnim regulativnim predpisom, hitrosti implementacije, itd.

2 CI/CD cevovodi

2.1. Tipi cevovodov



Slika 2: Tipi cevovodov in njihova odvisnost.

Cevovode za SaaS smo na njihovo namembnost oziroma podobnost korakov razdelili v naslednje tipe:

- *Graditelji vsebnikov cevovodnih agentov:* Uporaba vsebnikov za cevovodne agente omogoča standardizacijo okolja, v katerem se izvajajo vse faze posameznega cevovoda. To je še posebej pomembno, če agente vzdržujemo sami. Takšni vsebniki se lahko uporabijo tudi za agente v oblaku. Alternativa uporabi vsebnikov je nameščanje potrebnih knjižnic in orodij ob vsakem zagonu cevovoda, kar je lahko časovno in iz vidika porabe virov stroškovno neučinkovito.
- *Graditelji knjižnic:* Cevovodi za knjižnice vključujejo specifične korake. Pri gradnji uporabljajo modularne in integracijske teste programov, ki so uporabniki knjižnic. To zagotavlja višji nivo kakovosti knjižnic ob njihovi izdaji.
- *Cevovodi za grajenje storitev:* To so osrednji cevovodi, ki skozi različne faze vodijo do izdaje verzije programske opreme ali namestitvenega artefakta (npr. vsebnika). Vključujejo različne načine testiranja, kot so testi modulov, integracijski testi, zmogljivostni testi in testi ranljivosti. Vse je usmerjeno k cilju zagotavljanja visokokakovostnih izdaj.
- *Cevovodi za grajenje uporabniških vmesnikov:* Cevovodi so podobni grajenju storitev. Ločitev med slednjimi in cevovodi za grajenje uporabniških vmesnikov je smiselna zaradi običajno velikih razlik v tehnologijah, ki so uporabljene za uporabniški vmesnik v primerjavi s tehnologijami za zaledne sisteme.
- *Specializirani cevovodi:* V zalednih sistemih se lahko pojavijo storitve, ki vsebujejo kompleksne algoritme, kot so učenje umetne inteligence, geografsko ali grafično procesiranje, zahtevni matematični procesi, itd. Gradnja takšnih storitev zahteva posebne postopke v cevovodih. To so koraki validacije algoritmov ali rezultatov učenja z uporabo posebnih metrik in širokega nabora raznovrstnih testov.

- *Infrastrukturni cevovodi:* Upravljanje oblaka je učinkovito le s pomočjo koncepta infrastrukture kot kode (ang. Infrastructure-as-Code - IaC). Kodo za IaC je potrebno nadzorovati z vidika statičnih analiz, testiranja in namestitev.
- *Cevovodi za neprekinjeno nameščanje:* GitOps je eden izmed pristopov za izvajanje takšnih cevovodov. Ti cevovodi so zadolženi za implementacijo strategij distribucije in nameščanja programske opreme.
- *Cevovodi za grajenje dokumentacije:* Omogočajo gradnjo artefaktov uporabniških navodil, ki so dostavljena skupaj s programsko opremo in tako neposredno dostopna končnim uporabnikom. Prav tako lahko cevovodi skrbijo za kreiranje tehnične dokumentacije in postavitev portalov za razvijalce.

Tu moramo omeniti, da se instance zgornjih tipov cevovodov lahko sekvenčno ali paralelno povežejo v večje celostne CI/CD cevovode. Njihova soodvisnost je prikazana na sliki 2. Tipizacija cevovodov je uporabna predvsem za postavitev predlog (ang. templates) cevovodov.

2.2. Koraki cevovoda

Skozi praktične izkušnje smo prišli do sledečih korakov v CI/CD cevovodih za SaaS v oblaku (slika 3):

- *Priprava in preverjanje okolja:* Prvi korak vsakega cevovoda je preverjanje, če so na voljo vsi parametri okolja in orodja. S tem deloma preprečimo, da v cevovodu pride do napake šele po dolgem času izvajanja, praviloma v koraku, ker se npr. parametri dejansko uporabijo. Strategija hitrega neuspeha (ang. fail-fast strategy) je zelo učinkovit način izvajanja cevovodov, ki prispeva k manjši uporabi virov in lažjemu iskanju vzrokov. To je še posebno pomembno za cevovode, ki imajo v povprečju relativno dolg čas izvajanja.
- *Statično in oblikovno preverjanje kode (ang. linting and formatting):* Odpravljanje tehničnega dolga v kodi je pomemben del vzdrževanja kode. Statična analiza pomaga ohraniti kodo v stanju, da je njeno vzdrževanje stroškovno učinkovito. Statična analiza in formatiranje kode znatno pripomoreta k zmanjšanju komentarjev in popravkov kode v procesu revidiranja zahtev za združitev vejitev (ang. pull requests, merge requests). V tej stopnji se lahko uporabi kriterij doseganja pragu za določanje uspešnosti tega koraka cevovoda. To omogoči razvijalcem možnost postopne izboljšave kode v primeru, če koda v preteklosti ni bila del takšnih analiz in ima tehnični dolg. V takšnih primerih pomembno, da se prag skozi konsenz ali celo samodejno dviguje.
- *Prevajanje programskih knjižnic, zalednih storitev ali uporabniških vmesnikov:* Pomemben del prevajanja je lokalni predpomnilnik že prevedene kode v agentih cevovodov. S tem znatno pohitrimo prevajanje kode. Korak prevajanja postane zahtevnejši, če je potrebno program prevesti za različne platforme npr. x64 in arm64. Tukaj si lahko pomagamo s paralelnim izvajanjem cevovoda.
- *Testiranje posameznih modulov:* Tem bližje je koda testov poslovni logiki, tem cenejše je izdelava, vzdrževanje in zaganjanje testov [2,3]. Testi modulov (ang. unit-tests) so zato zelo pomemben del cevovoda. Z metrikami testne pokritosti lahko pomagamo razvijalcem, da pokritost kode ohranjajo in/ali povečujejo. Z uvedbo pragov na teh metrikah cevovodi lahko tudi preprečijo padec pokritosti testiranja. Ne smemo pozabiti, da je tudi uporabniški vmesnik možno testirati brez zalednih sistemov s pomočjo imitacij (ang. mocking) in v primeru spletnih aplikacij celo brez spletnega brskalnika. To lahko določenih primerih zelo pospeši postopke testiranja uporabniškega vmesnika.
- *Zagon povezanih storitev:* Za celostno integracijsko testiranje je pomembno, da ima testirana komponenta programske opreme okoli sebe čim več pravi storitev in čim manj imitiranih storitev (ang. mock-ups). Zagon povezanih storitev je korak v cevovodu, ki poskrbi, da se v izvajalno okolje testov zaženejo storitve kot so podatkovne baze, sporočilne vrste ali odvisne programske storitve. V primeru, če je potrebna tudi specialna komponenta iz oblaka (npr. sporočilna vrsta, podatkovna baza, zabojnik za datoteke), je

potrebno s pomočjo IaC poskrbeti, da se postavi v oblaku postavi tudi ta. Če storitve porabijo veliko časa za zagon, imamo lahko tudi bazen takšnih storitev za njihovo ponovno uporabo. Vendar je pomočjo vsebnikov praktično mogoče postaviti najrazličnejše storitve zelo hitro in zanesljivo.

- *Integracijsko testiranje:* S tem korakom preverimo, kako se storitev odziva skozi API-je. Oziroma, kako se obnaša dejanski uporabniški vmesnik v spletnem brskalniku ali mobilni napravi. Stremimo k temu, da se uporabi čim več pravih komponent. To še posebno velja podatkovne baze, sporočilne sisteme in druge storitve izvajalnega okolja. Uporaba imitacij je smiselna v primeru, če nam tretjeosebni dobavitelj ne nudi testnega okolja API-jev, oziroma, če potrebujemo določene nastavitve za vsak potek testiranja, ki jih je nemogoče nastaviti ali ponastaviti samodejno skozi API-je dobavitelja.

- *Testiranje zmogljivosti:* Za takšno testiranje na nivoju posameznih komponent sistema oziroma storitev, je pomembno, da se odločimo ali je testiranje zmogljivosti za to storitev pomembno ali ne. Kandidati so zagotovo osredni deli sistema, ki so kritični iz vidika algoritmov, odločitvenih postopkov, strojnega učenja, itd.

- *Poročanje o kazalnikih testne pokritosti:* Rezultati testiranja modulov, integracijskega testiranja in testiranja zmogljivosti so zelo pomembni za pregled nad tehničnim dolgom, kvaliteto in zadostitvami nefunkcionalnih zahtev. Korak poročanja o kazalnikih je lahko v vlogi preverjanja doseganja določenih pragov in hkrati objave rezultatov testiranja, testne pokritosti, zmogljivosti sistema na posebnih zbirnih mestih. Takšna zbirna mesta služijo razvojnim oddelkom za planiranje vzdrževanja nivoja rezultatov ali planiranje aktivnosti za doseganje določenih pragov kriterijev v prihodnosti.

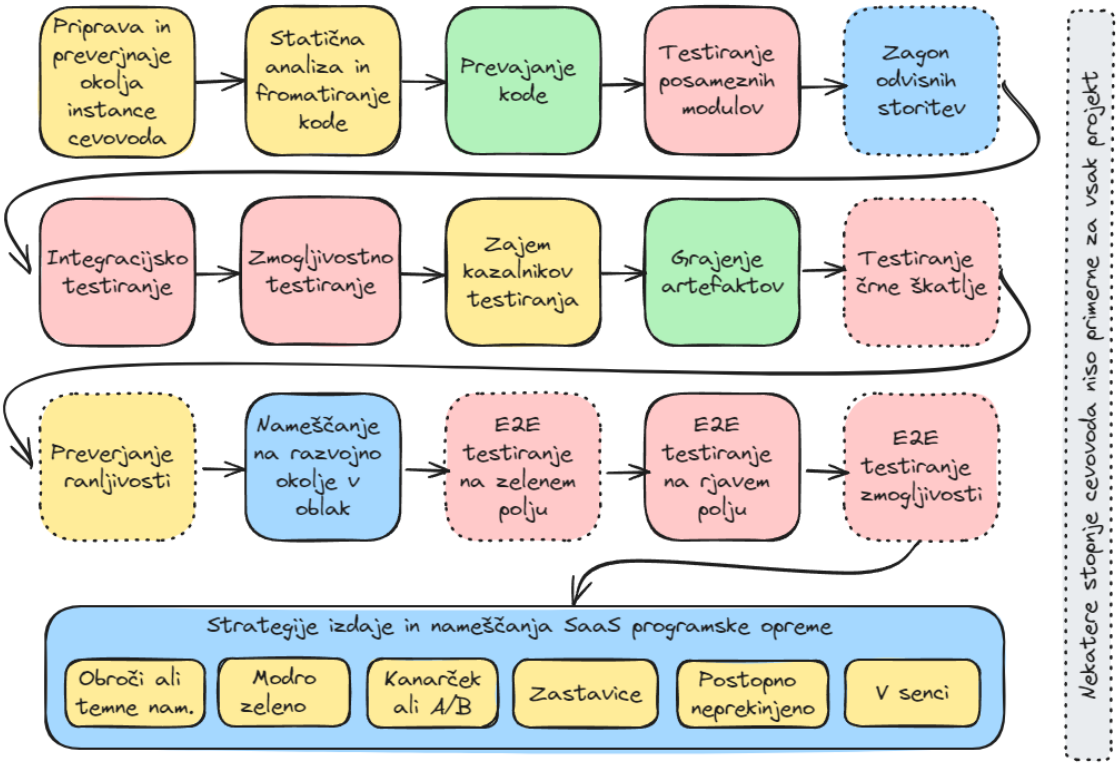
- *Pakiranje in grajenje artefaktov:* Izhodi tega koraka so artefakti programske opreme, kot so najrazličnejše knjižnice programske opreme, vsebniki in druge oblike prevedene in pakirane kode. Ta korak lahko vsebuje tudi določene teste artefaktov. Pogosto se namreč zgodi, da vsi prejšnji nivoji testiranja uspejo, vendar na koncu postopek pakiranja v zabojnik povzroči, da je zabojnik neuporaben oziroma nedelujoč. Če temu koraku neposredno sledijo koraki namestitve zabojnikov v testno okolje, se lahko zanesemo nanje. V primeru, če naš cevovod izda samo verzijo zabojnika in se procesi namestitve v testna in/ali produkcijska okolja izvajajo asinhrono, je smiselno opraviti kratek test zagona zabojnika v obliki črne-škatle. S tem zmanjšamo možnost, da v repozitorij slik vsebnikov objavimo nedelujočo verzijo zabojnika. Predhodna gradnja in uporaba zabojnikov v korakih integracijskega testiranja ni priporočljiva, ker bi težko pridobili metrike pokritosti kode.

- *Preverjanje ranljivosti* (ang. vulnerability check): Namestitve v oblak so lahko postavljene v dveh načinih. V primeru varnostnega modela brez privzetega zaupanja (ang. zero-trust) so storitve dejansko skoraj neposredno izpostavljene internetu. V drugem primeru varnosti na robu (ang. security at the edge), kjer se varnost preverja samo na vhodu v gručo storitev, ima potencialni preboj varnosti še večjo površino. V obeh primerih se priporoča, da se uporabljene knjižnice in druge podsisteme redno pregleduje za ranljivosti.

- *Nameščanje programske opreme na razvojno okolje v oblaku:* V tem koraku poskušamo čim bližje priti postopku, ki je kasneje uporabljen pri strategiji izdaje in nameščanja v produkcijska okolja. Na drugi strani se moramo zavedati, da mora biti postopek hiter in omogočati namestitev oziroma posodobitev le komponent sistema, ki so se dejansko spremenile. Hitrost nameščanja je ključna pri velikih razvojnih ekipah, ki zaradi učinkovitosti stremijo k razvoju na osnovi glavne veje (ang. trunk-based development) [6]. To jim omogoči hitro razrešitev konfliktov v kodi in delovanju sistema.

- *E2E testiranje na zelenem polju:* Celostno testiranje lahko vsebuje testiranje zelenega polja. Tu preverimo, če je naš postopek nameščanja sistema na novo instanco v oblaku še delujoč po spremembah. Oziroma lahko poganjamo tiste teste, ki za svoje delovanje potrebujejo ali prazno podatkovno bazo ali v naprej pripravljen set podatkov, ki je vedno enak

- *E2E testiranje na rjavem polju:* Najbolj običajen način E2E testiranja. Določeno instanco sistema vedno posodabljamo in nad njo izvajamo idempotentne teste. To je tudi korak, ki vsebuje največ E2E testov uporabniških vmesnikov.
- *E2E testiranje zmogljivosti:* Tu gre za korak v cevovodu, ki se ne izvaja vedno. Po potrebi lahko preverimo kritične poti v sistemu, če so po določenih spremembah še zadovoljivo zmogljive. Pomembni del tega koraka je, da imamo popolnoma avtomatizirano postavitev testnega okolja, samo izvajanje testov in izdelavo poročila. Le tako bomo lahko včasih zelo kompleksne zmogljivostne teste po potrebi ponavljaji brez večjih razvojnih in izvajalnih stroškov.



Slika 3: Stopnje celostnega cevovoda za SaaS programsko opremo.

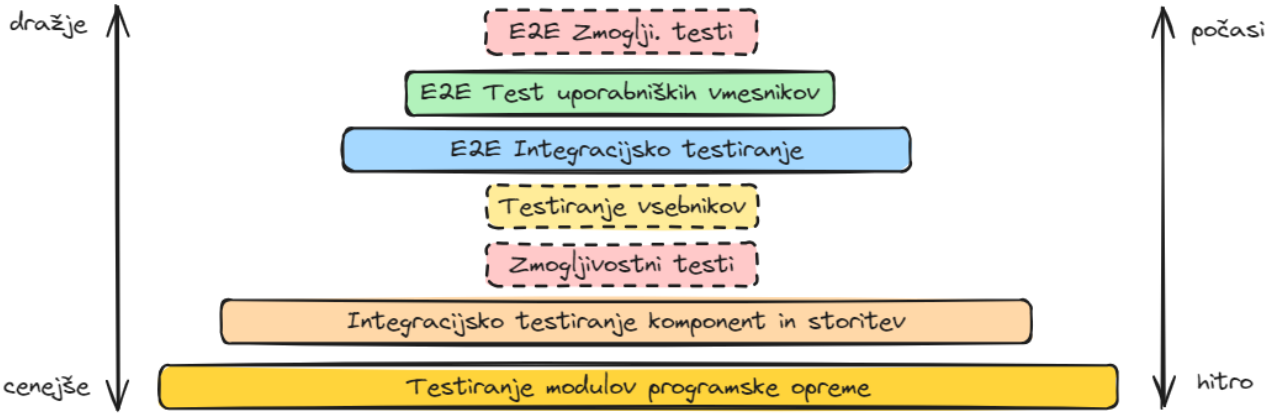
2.3. Izbor korakov cevovoda in optimizacija

Če smo si v prejšnjem poglavju pogledali vse možne stopnje SaaS cevovoda, to še ne pomeni, da mora vsak sistem imeti vse navedene korake v svojem cevovodu. Prav tako ne velja, da mora biti sosledje korakov točno takšen kot je naveden. S tehtnimi analizami moramo opredeliti cevovod, ki bo kar se da najbolje služil kot orodje za doseganje SaaS ciljev oziroma ne-funkcionalnih zahtev. Moramo se zavedati, da vsak korak v cevovodu potrebuje izvajalni čas [3]. Določeni koraki, ko so npr. zmogljivostni testi, lahko v oblaku povzročijo tudi opazne stroške.

Imamo opcijo, da določene korake iz celostnega cevovoda izločimo in jih poganjamo v ločenem cevovodu, ki se izvaja na podlagi urnika. Eden izmed kandidatov je zagotovo korak pregleda ranljivosti kode oziroma vsebnikov. Korak se bo še vedno izvajal, vendar ne bo blokiral izvajanje primarnega cevovoda. Za ločene cevovode ne smemo pozabiti postopke nadzora njihovega izvajanja. Če npr. specializiran cevovod najde ranljivost, se mora sprožiti postopek planiranja odprave ranljivosti v razvojnih ekipah.

K učinkovitemu izvajanju korakov testiranja močno pripomore pravilna struktura testiranja. Čeprav agilni pristopi k razvoju programske opreme brišejo meje med razvojem in testiranju v organizaciji, se pogosto še vedno zgodi, da je razvoj mnenja, da avtomatizirane teste piše specializirano testno osebje za zagotavljanje kakovosti. Slednje je običajno osredotočeno na pisanje E2E testov v zgornjih nivojih testiranja. V tem primeru dobimo sistem, ki za

svoje testiranje v cevovodu porabi veliko časa, saj so E2E testi izmed vseh nivojev testiranja časovno in stroškovno najbolj potratni [1]. Martin Fowler [2] je v letu 2012 opredelil osnovno piramido testiranja, ki jo lahko danes razširimo z novimi nivoji, ki so v povezavi s koraki testiranja v cevovodih (slika 4). Če sledimo strategiji, da je obsežen del vseh testov na nižjih nivojih, smo zagotovo naredili korak v bolj učinkovitemu cevovodu brez negativnih učinkov na doseganje strategije celostnega testiranja, ki je zelo pomembno v cevovodih za SaaS.



Slika 4: Piramida testiranja.

3 DevOps

Osrednji cilji DevOps organizacije so [4]:

- Pogosta posodobitev oziroma namestitev programske opreme,
- hitrejši prihod na trg,
- redkejše napake in odpovedi sistema,
- zmanjšanje časa od iniciacije do prve uporabe nove funkcionalnosti s strani uporabnika,
- zmanjšanje časa od odpovedi do obnovitve.

Cilji DevOps se skladajo z agilnimi pristopi razvoja programske opreme. Ne moremo doseči pogoste posodobitve produkcijskih okolij, če je razvoj programske opreme osnovan na slapovnem pristopu. Agilni pristopi razvoja programske opreme kot so Scrum, Kanban, SAFe, itd. so ključni za doseganje večje frekvence posodobitev SaaS v produkciji [7]. Promovirajo kratke iteracije in integrirano zagotavljanje kvalitete.

Pomemben del pogostih posodobitev in hitrega prihoda na trg je zagotovo strategija izdaje in nameščanja programske opreme. Vzdrževalna okna za posodobitev SaaS v večini primerih ne pridejo v poštev. Za posodobitve SaaS sistema v času uporabe so ključne za strategije nameščanja posodobitev. Lahko izberemo strategijo obroča, temnih namestitev, modro-zelenih namestitev, A/B paralelizma, kanarček namestitev, postopno neprekinjenih posodobitev ali postavitev v senci. Vsaka ima določene prednosti in tudi jasen namen. Izbor prave strategije je odvisen SaaS in pričakovanj trga.

Če želimo pogoste izdaje ne smemo pozabiti tudi na razvojne ekipe. Če je razvojnih ekip več in so odvisnosti velike, lahko pride do zahtevnih usklajevanj. Sledeče strategije nam to poenostavijo:

- Enoten repozitorij (ang. monorepo) [8,9] – skupaj z razvojem v glavni veji (ang. trunk-based development) [6] zmanjša usklajevanje vejitev in verzij različnih modulov. Nevarnost tega pristopa iz vidika DevOps je, da razvijalci pozabijo na združljivost za nazaj med internimi storitvami in sporočilnimi vrstami. Ne glede na strategijo nameščanja na produkcijsko okolje bo v nekem krajšem ali daljšem času tekla stara in nova verzija storitve hkrati.

- Modularna arhitektura, mikro-storitve ali mikro-prednji-deli (ang. micro-frontend) z jasno začrtanimi vsebinskimi mejami (ang. bounded context) omogočijo, da so lahko razvojne ekipe dokaj samostojne in neodvisne. Navkljub temu, da je v literaturi navedenih veliko razlogov zakaj je arhitektura mikro-storitev učinkovita, smo skozi praktične izkušnje našli le štiri prave razloge: Razdelitev odgovornosti za mikro-storitve med različne ekipe, neodvisno skaliranje mikro-storitev, različna tehnologija posameznih mikro-storitev in ponovno uporabljivost mikro-storitev v različnih sistemih. Kljub prednostim, ki jih prinašajo mikro-storitve, je mogoče nekatere koristi, kot je boljša preglednost nad deli sistema, doseči tudi z modularno zasnovo ene mikro-storitve. Pomembno je vedeti, da vsaka mikro-storitev zahteva določene računalniške vire. V povezavi z visoko razpoložljivostjo, ki zahteva vsaj tri instance vsake storitve, lahko to povzroči visoke stroške storitev v oblaku.
- K dvigu kakovosti produkta in izboljšanju sodelovanja med ekipami prispevajo pregledi oziroma revizije kode s strani sodelavcev, princip štirih oči pri testiranju kode ter določitev lastnikov kode. V proces pregledovanja zahtev za združitev vej (ang. pull requests, merge requests) lahko vključimo korake, ki zahtevajo revizijo kode s strani lastnikov kode, ki so odgovorni za določene dele kode oziroma module. S tem se zmanjšuje deljena odgovornost za posamezne dele programske kode, kar povečuje transparentnost repozitorija in motiviranost razvijalcev. Princip testiranja na štiri oči zagotavlja, da teste za določen del kode ne piše avtor sprememb, kar prispeva k večji pokritosti in širini testiranja. Ta pristop ne le izboljšuje kakovost kode, ampak tudi spodbuja sodelovanje in delitev znanja med člani ekipe.
- Testna okolja so lahko lokalna, osebna, ekipna ali globalna. Lokalno testiranje je vedno najbolj učinkovito in cevovodi so v tem primeru le v vlogi validacije. Vendar pa je pogosto sistem prevelik in prezapleten, da bi ga razvijalci poganjali v celoti samo lokalno. Tu se lahko poslužujemo globalnih razvijalnih okolij, ki so postavljeni s strani cevovodov. Razvijalci se iz lokalnih delovnih postaj v tem primeru povežejo v globalno okolje le z eno ali dvema mikro-storitvama. Če takšna strategija ni mogoča, lahko postavimo osebna ali ekipna okolja v oblaku. Tu se lahko poslužimo tudi prekomerne dodelitve virov (ang. over-provisioning). S tem bolje izkoristimo vire v oblaku za namen testiranja.
- Uvedba specializiranih ekip, ki zmanjšajo kognitivno obremenitev razvijalcev za celotno tehnologijo (ang. full stack). Veščine inženiringa platform lahko v veliki meri pripomorejo, da so razvijalci osredotočeni na poslovno vrednost in ne tehnološke specifike SaaS programske opreme in velikih modularnih arhitektur.

Avtomatizacija testiranja in posodobitev skozi cevovode močno pripomore k redkejšim napakam in odpovedim. Ključna je tudi infrastruktura. V oblaku imamo možnost z IaC enostavno postaviti redundantni oziroma celo geografsko porazdeljeni sistem. Čeprav ima oblak že v osnovi zagotovljen visok nivo zanesljivosti (ang. Service Level Agreement – SLA), ga lahko z geografsko porazdelitvijo še povečamo.

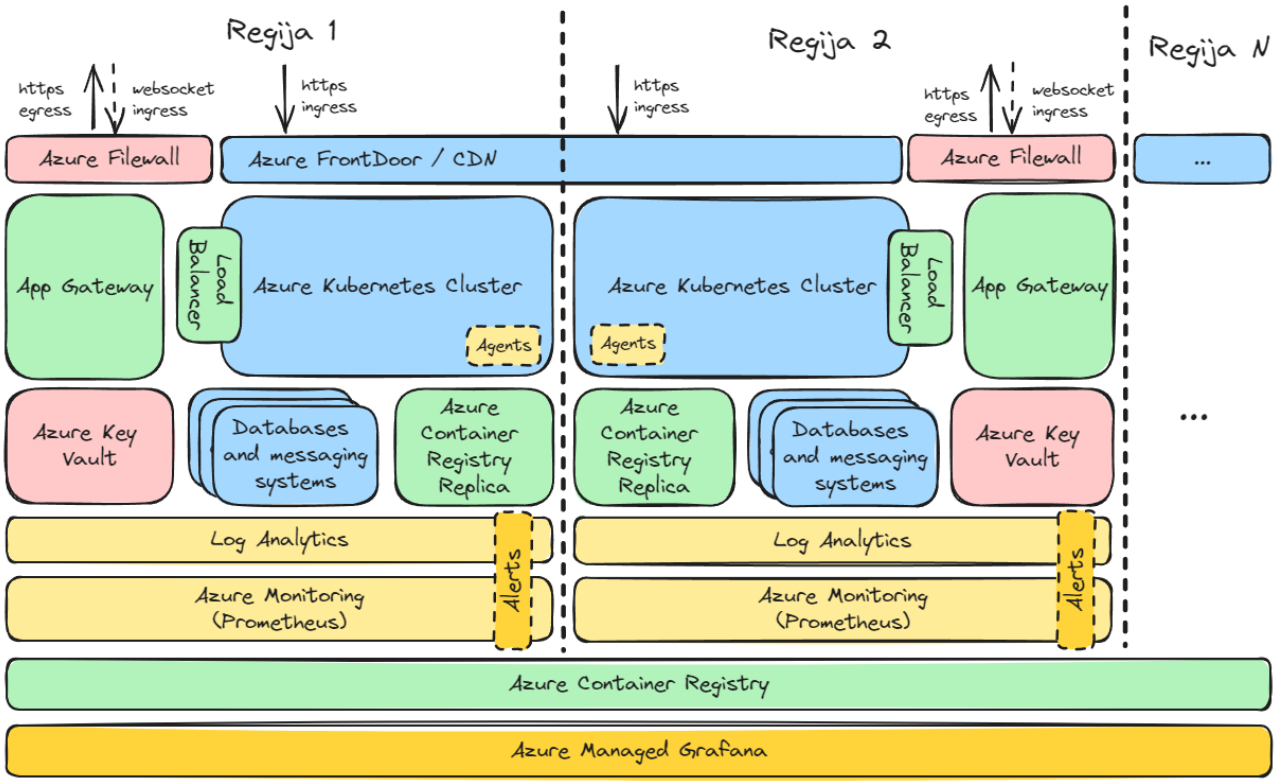
Sestavni del zagotavljanja delovanja SaaS sistema in manjšanja časa med odpovedjo in obnovitvijo je zagotovo tudi spremljava metrik sistema z alarmi in samodejno pregledovanje dnevnikov za napako. Dobra strategija nudenja SaaS je zaznava napak še predno jih stranka prijavi. Da slednje lahko izvajamo, moramo neprestano pregledovati statuse HTTP odgovorov (oziroma statuse drugih protokolov) ali pregledovati dnevnike. Iz vidika pregledovanja dnevnikov je zelo dobra strategija, da se v dnevnik ne zapisuje nič, če sistem deluje normalno. Pogosto se zgodi, da storitve v dnevnike zapisujejo tudi poslovne napake (npr. validacije podatkov), kar ni učinkovito in pregledno. Napačen vnos v masko ali napačen klic API-ja ni napaka v delovanju sistema.

Na koncu še omenimo, da hitrost CI/CD cevovodov močno vpliva hitrost odprave napak v delovanju sistema in čas posodobitev programske opreme. Če moramo za odpravo napake namestiti novo verzijo, moramo za izdajo nove verzije izvesti avtomatizirane teste oziroma celoten cevovod. Slednji je lahko dolgotrajen proces v zelo velikih sistemih. Takšne težave rešujemo s sistemom za samodejno izbiranje potrebnega nabora regresijskih testov glede na dejanske spremembe v kodi in odvisnosti od sprememb. Hkrati s ponovno namestitvijo oziroma posodobitvijo samo odvisnih modulov pohitrimo tudi korak nameščanja v oblak.

Za zagotavljanje hitrih popravkov v kritičnih situacijah, je v nekaterih primerih smiselno imeti bližnjico za postopek hitre gradnje in namestitve. Ena izmed možnih implementacij je ročna selekcija korakov cevovoda, ki se bodo izvedli. Slednje nam ponudi možnost izredno hitre namestitve popravkov, če presodimo, da je tveganje sprejemljivo oziroma je vpliv spremenjene kode omejen.

4 Javni Oblak

4.1. Arhitektura SaaS v javnem oblaku



Slika 5: Primer globalne SaaS arhitekture v hiperoblaku Azure.

Arhitekture v oblaku na sliki 5 prikazuje način postavitve SaaS za globalni trg. Celotna arhitekturo je možno postaviti skozi IaC pristop. Kritičen del IaC je izbor same tehnologije. Na voljo so nam večnamenska orodja kot so Terraform, OpenTofu, Pulumi, Crossplane, itd. Alternativno javne oblačne platforme nudijo specializirana orodja, kot so CLI ukazi in raznovrstni sistemi predlog, kot sta ARM in Bicep v Azure ekosistemu. Uporaba načina z večnamenskimi orodji je smiseln samo v primeru, če moramo zagotoviti IaC namestitve za več različnih ponudnikov oblakov hkrati. Če to ni predpogoj, so boljša izbira predloge in orodja posameznega ponudnika. Predloge zanje lahko kreiramo interaktivno skozi spletni portal in izvozimo v IaC obliki. Hkrati se je skozi uporabo izkazalo, da je iskanje napak pri namestitvah s predlogami veliko bolje podprto in jasno zapisano v dnevnikih, kot pri uporabi alternativ kot je orodje Terraform.

Nadzor nad delovanjem sistema se opravlja s pomočjo metrik, ki jih samodejno analizira sistem za alarme. Pregledne analize in spremljavo se nad metrikami se lahko izvaja preko Grafane. Za upravljanje varnostno občutljivih nastavitev in konfiguracij je primeren specializiran sistem kot je Azure Key Vault.

Povsod, ker je mogoče, je smiselno uporabiti PaaS. Le-ta je celostno upravljan s strani ponudnika oblaka. Sem spadajo PaaS storitve kot so podatkovne baze, sporočilni sistemi, podatkovni direktoriji (ang. storage accounts), trezorji, itd. Čeprav lahko vse te storitve poganjamo v Kubernetes gruči sami, se moramo zavedati, da zagotavljanje storitev zabojnikov s trajnimi plastmi (ang. persistent layers) prinese veliko večjo kompleksnost, kot poganjanje

storitve procesnih zabojnikov. V zadnjem času postajajo aktualni Kubernetes operaterji. Slednji nam prihranijo marsikatero inženirsko delo postavitve podatkovnih storitev v gruči. Težava nastane tam, kjer je potrebno poganjati storitve, ki niso prijazne do oblaka (ang. cloud native). Za primer lahko navedemo storitev sporočanja RabbitMQ. Čeprav za sporočilni sistem operater postavi gručo, se kmalu izkaže, da se vse povezane storitve ustavijo v primeru prenehanja delovanja ene izmed instanc RabbitMQ zabojnika. Te storitve so vezane na specifično instanco RabbitMQ zabojnika v gruči, ki vsebuje podmnožico sporočilnih vrst za te storitve. To povzroči dolgotrajen izpad sistema tudi v HA postavitvi. Zaradi tega postanejo vzdrževalna opravila, kot je opravilo posodobitve Kubernetesa, v načinu 24/7 zelo zahtevna.

Omeniti je potrebno tudi več-regijske oziroma globalne storitve v oblaka. Ena izmed takšnih je Microsoft Azure Front Door. Azure Front Door je storitev na robu oblaka in je glavna vhodna točka za vse SaaS na podlagi HTTPS protokola. Omogoča regijsko neodvisno vstopno točko in CDN (predpomnilnik). Storitev Front Door na podlagi latence določi, katero regijsko postavitev v oblaku se uporabi za posamezno HTTPS zahtevo. Da takšen sistem lahko deluje moramo imeti tudi globalno podatkovno bazo kot je npr. Azure CosmosDB, ki se samodejno regijsko replicira ali pa opcijsko celo zagotavlja več-regijski ACID za transakcije.

Azure Front Door igra tudi vlogo požarnega zidu, ki je specializiran za spletne aplikacije. Varnost v oblaku je izrednega pomena in velja enako kot za druge plasti v SaaS arhitekturi, da je najbolje uporabiti PaaS ponudnika oblaka.

Ne smemo pozabiti tudi na postavitve zabojnikov v oblak v brez-strežniškem (ang. serverless) načinu. V tem primeru se lahko izognemo uporabi Kubernetes gruče. Brez-strežniške namestitve zabojnikov v oblak so uporabne v primerih, ko moramo zaradi stroškov skalirati zabojnik do 0 instanc ob neuporabi. Druga prednost je, da se izognemo procesov posodabljanja Kubernetes gruč in upravljanja skaliranja na dveh nivojih – na nivoju vozlišč in na nivoju zabojnikov.

4.2. Kdaj se odločiti za hiperoblak?

SaaS (ang. Software-as-a-Service) ne more obstajati brez PaaS (ang. Platform-as-a-Service) ter PaaS ne more obstajati brez IaaS (ang. Infrastructure-as-a-Service). Veliko ponudnikov gostovanja se je oklicalo za oblak, nekateri celo enačijo instanco Kubernetes gruče z oblakom. Ločitev med gostovanjem in oblakom je v ekosistemu. Javni oblaki imajo globalno prisotnost, IaC pokritost, transparentno cenovno politiko, finančno pokrite SLA-je, razvojne platforme, 24/7 podporo, elastično skalabilnost, sledenje standardom industrije in podporo regulativi. V poenostavljenem načinu bi lahko zapisali: Če moramo svojemu ponudniku gostovanja za nastavitev omrežja poslati Excel preglednico naslovov omrežja, vrat in požarnih zidov, to ni storitev oblaka.

V primeru, če je za delovanje nekaterih storitev potrebno lokalno procesiranje, to rešujemo s t.i. robnimi namestitvami. Takšne namestitve lahko premostijo pomanjkljivosti oblaka, če so postavljene nefunkcionalne zahteve:

- Zahtevana mikro-lokalnost podatkov,
- zakonska regulativa,
- preprečevanje izpada omrežja oziroma procesiranje blizu realnega časa,
- obsežni vhodni podatki, ki jih je smiselno obdelati na lokaciji, preden se prenesejo v oblak (npr. obdelava zajetega videa)

Ponudniki globalnega SaaS se primarno morajo vprašati, zakaj ne namestiti v javni oblak. Največkrat so v času planiranja najprej vidni operativni stroški. Tu nastane težava v tem, da so stroški v javnem oblaku transparentni, medtem ko druge opcije stroške ocenijo zelo ozko – npr. samo stroške nakupa ali najema strojne opreme.

Eden izmed pomembnih stroškov je zagotovo strošek človeških virov – sistemskih inženirjev. V primeru uporabe PaaS preko IaC moramo upoštevati, da so postopki varnostne posodobitve storitev (npr. podatkovne baze), doseganje visokega nivoja SLA-jev, varnostno kopiranje na geografsko dislocirana mesta in restavriranje na zahtevo že del osnovnega stroška storitve. Strošek človeškega vira kreiranja in vzdrževanja PaaS z IaC je zanemarljiv v primerjavi s stroškom sistemskih inženirjev, ki morajo takšno storitev postaviti, vzdrževati in 24/7 nadzorovati. V velik večni primerov tudi v različnih geografskih regijah oziroma časovnih conah. Hkrati imajo globalne storitve v oblaku neprimerljivo večji nabor uporabnikov, kar jih posredno naredi neprimerljivo bolj zanesljive kot lokalno vzdrževane storitve. Naše dejanske izkušnje z obema pristopoma so to tudi potrdile.

4.3. FinOps

Stroški uporabe javnega oblaka ponujajo visoko stopnjo transparentnosti in so v veliki meri predvidljivi, kar predstavlja pomembno prednost za podjetja. Transparentnost stroškov omogoča organizacijam, da jasno vidijo, za kaj plačujejo, in kako se sredstva porabljajo. To je še posebej koristno, saj lahko podjetja natančno spremljajo svoje operativne izdatke in jih ustrezno prilagajajo.

Ena od ključnih značilnosti javnega oblaka je njegova sposobnost hitrega prilagajanja resursov. To pomeni, da lahko organizacije zelo hitro odreagirajo na spremembe v nefunkcionalnih zahtevah, kot so spremenjene potrebe po obdelovalni moči, prostoru za shranjevanje ali širini pasovne širine. Ta agilnost omogoča, da se storitve v oblaku skalirajo navzgor ali navzdol, odvisno od trenutnih potreb, kar zmanjšuje nepotrebne stroške in povečuje učinkovitost. To ne samo da optimizira porabo virov, ampak tudi zagotavlja, da podjetja maksimalno izkoristijo svoje naložbe v oblak, hkrati pa ohranjajo visoko stopnjo operativne pripravljenosti in zmogljivosti.

Optimizacijo stroškov je možno upravljati tudi s pomočjo zakupa storitev. Ponudniki omogočajo načine kot so rezervacije in varčevalni načrti. Z rezervacijami lahko dosežemo tudi to 60% prihranke medtem kot se varčevalni načrti gibljejo okoli 40%. Rezervacije so za ponudnike oblaka nekaj normalnega, ker jim ni v interesu, da sistemi skalirajo na dnevnem nivoju. Ponudniki morajo namreč imeti zagotovljeno računsko moč tudi ponoči, ne glede na to ali je v uporabi ali ne. Zato ponudniki omogočajo rezervacije, s katerimi lahko zakupite mesečno uporabo procesni kapacitet in ostalih storitev. S to ugodnostjo je porabnikom javnega oblaka smiselno pokriti osnovni odtis delovanje sistema (ang. footprint) in maksimalno dnevno uporabo, če se slednja pokrije s prihrankom rezervacije. Pravo elastično skaliranje je tako stroškovno učinkovito le pri vrhovih uporabe, ki niso osnovani na dnevnem nivoju. Primer takšnih vrhov uporabe bi bili vikendi, prazniki, vreme, letni čas, zagon prodajnih akcij, itd. – odvisno od namembnosti programske opreme. Če smo takšno porabo procesiranja sposobni oceniti mesečno, sezonsko ali letno, se lahko poslužimo »varčevalnega modela«. Z njim zakupimo določeno število ur oziroma sekund procesiranja na mesec ali leto. Pri takšnih izračunih lahko nastopijo kompleksni stroškovni modeli, ki presegajo obseg tega članka.

Iz tehničnega vidika moramo omeniti več načinov skaliranja. V oblaku lahko skaliramo na podlagi uporabe procesorske moči ali pomnilnika. Modernejši pristopi omogočajo skaliranje tudi na podlagi števila zahtev ali dolžine sporočilnih vrst. Skaliranja v Kubernetesu lahko razdelimo med:

- Vertikalne – povečujemo ali zmanjšujemo CPU ali pomnilnik posameznemu zabojniku,
- horizontalne z zabojniki – povečujemo ali zmanjšujemo število instanc zabojnikov,
- horizontalne s strežniki – povečujemo ali zmanjšujemo število instanc vozlišč oziroma strežnikov.

Za učinkovito skaliranje je pomemben kratek čas od zagona zabojnika do začetke strežbe prvih zahtev. V posebnih primerih je potrebno tudi pred-gretje vozlišč, kar omogoči hitrejši odziv na potrebo po povečanju virov pri skaliranju. Uporaba PaaS za podatkovne in podobne storitve v oblaku nam prihrani marsikateri izziv skaliranja. Pri izboru storitev se moramo zavedati, da oblaku prijazne storitve (ang. cloud native) veliko bolje horizontalno skalirajo v gručah, kot to delajo storitve, ki so bile samo prenesene v oblak. Primerjamo lahko npr. PostgreSQL in CockroachDB.

5 Zaključek

Nenehna integracija, izdaja in nameščanje (CI/CD+CD) predstavljajo ključne procese za razvoj sodobne SaaS programske opreme, ki zahteva globalno dostopnost. Uporaba DevOps metod prinese možnost neprestanega posodabljanja SaaS programske opreme brez negativnih učinkov na stabilnost, kakovost in varnost sistema. Zagotavljanje DevOps pa ne more biti učinkovito brez različnih tipov cevovodov. Vsak tip cevovoda moramo prilagoditi specifičnim zahtevam, ki jih pogojuje gradnja različnih artefaktov, kot so knjižnice, zabojniki, dokumentacija in nenazadnje namestitve in E2E testiranje. Vsako stopnjo in korak cevovoda moramo vedno tehtno opredeliti in se vprašati, ali resnično potrebujemo določen korak cevovoda glede na nefunkcionalne zahteve ciljnega SaaS sistema.

Hiperoblaki, s svojo zanesljivostjo, elastično skalabilnostjo in transparentnimi stroški, predstavljajo odlično infrastrukturo za globalne SaaS rešitve. Takšno infrastrukturo moramo vedno korektno stroškovno oceniti. Stroški niso samo stroški najema, ampak moramo pogledati celotne stroške lastništva (ang. TCO). S takšnim pristopom pri odločitvah med oblakom in nakupom hitro vidimo, da hiperoblak ni samo »draga igrača«, ampak nam dejansko omogoča stroškovno učinkovitost in rast podjetja oziroma hitre širitve uporabe SaaS ali njegove elastičnosti. Stroške oblaka nam še dlje optimizira metoda FinOps, ki se nikoli ne konča. Vendar dolgoročno prinese obvladljive, napovedljive in optimizirane stroške oblaka.

Ključne stvari članka, ki jih je potrebno vzeti v vednost in izhajajo iz neposredne prakse: Hiperoblak ni drag, DevOps ni delovno mesto, z WindowsOS raje ne v Kubernetes, avtomatizacija testiranja ni nič manj zahtevna kot razvoj programske opreme, celosten pogled na CI/CD in DevOps ni trivialen.

Literatura

[1] PISANI Erica, GANCARZ Rafal, How Cell-Based Architecture Enhances Modern Distributed Systems, https://www.infoq.com/minibooks/cell-based-architecture-2024/, The InfoQ eMag, Issue 113, 2024.

[2] VOCKE Ham, The Practical Test Pyramid, https://martinfowler.com/articles/practical-test-pyramid.html, Februar 2018.

[3] FOWLER Martin, Test Pyramid, https://martinfowler.com/bliki/TestPyramid.html, Maj 2012.

[4] COWELL Christopher, LOTZ Nicholas, TIMERLAKE Chris »Automating DevOps with GitLab CI/CD Pipelines«, Packt Publishing Ltd., 2023.

[5] COUPLAND Martin, DevOps Adoption Strategies: Principles, Processes, Tools and Trends, Pact Publishing Ltd., 2021

[6] HAMMANT Paul, et.al., Trunk Based Development, https://trunkbaseddevelopment.com/, 2020.

[7] HUMBLE Jez, FARLEY David, Continuous delivery: Reliable software releases through, build, test, and deployment automation, Pearson Education, Inc., 2011.

[8] LOSOVIZ Leonardo, Monorepo vs Multi-Repo: Pros and Cons of Code Repository Strategies, Kinsta inc., 2024.

[9] POWELL Ron, Benefits and challenges of monorepo development practices, CircleCI Blog, 2024.