Implementacija, obezbeđivanje i upravljanje aplikacijama u javnom oblaku pomoću najsavremenijih alata

Gaurav Agarwal

kompjuter biblioteka
packt

Moderne DevOps prakse

PREVOD DRUGOG IZDANJA

DevOps i oblak su potpuno promenili način razvoja softvera i operacija, što je dovelo do brzog razvoja različitih DevOps alata, tehnika i praksi. Ovo ažurirano izdanje vam olakšava odabir pravih alata pružajući vam sve neophodno da upoznate DevOps kulturu.

Knjiga počinje predstavljanjem savremene arhitekture u oblaku, a zatim i arhitektonskih koncepata implementacije modernog načina razvoja aplikacija. Slede poglavlja posvećena Git sistemu, platformama Docker i Kubernetes, kao i Ansible, Terraform, Packer i sličnim alatima, koji su osnova izgradnje. Zatim su opisani ključni elementi integracije sa oblakom - AWS ECS, GKE i druge CaaS usluge. Takođe, objašnjene su GitOps tehnologija, neprekidna integracija i neprekidna isporuka - GitHub akcije, alati Jenkins i Argo CD - da biste razumeli suštinu savremene isporuke aplikacija. Potom, upravljate svojom kontejnerskom aplikacijom u proizvodnji, pomoću mreže usluga, pa i veštačke inteligencije. U celoj knjizi, predstavljene su najbolje prakse za automatizaciju i upravljanje razvojnim ciklusom, infrastrukturom, kontejnerima, i još mnogo toga.

Kada proučite ovu knjigu o modernoj DevOps kulturi, bićete spremni da razvijate i upravljate aplikacijama pomoću modernih alata i tehnika.

Šta ćete naučiti

- Savremene DevOps prakse sa Git sistemom i GitOps metodologijom
- Osnove kontejnerizacije na platformama Docker i Kubernetes
- Upotrebu AWS ECS, Google Kubernetes Engine i Knative platformi
- Efikasnu izgradnju i upravljanje bezbednim Docker slikama
- Neprekidnu integraciju pomoću alata Jenkins na Kubernetes platformi i pomoću GitHub akcija
- Alat Argo CD za neprekidnu implementaciju i isporuku
- Upravljanje nepromenljivom infrastrukturom u oblaku pomoću alata Packer, Terraform i Ansible
- Upravljanje kontejnerskim aplikacijama u proizvodnji pomoću Istio arhitekture i veštačke inteligencije

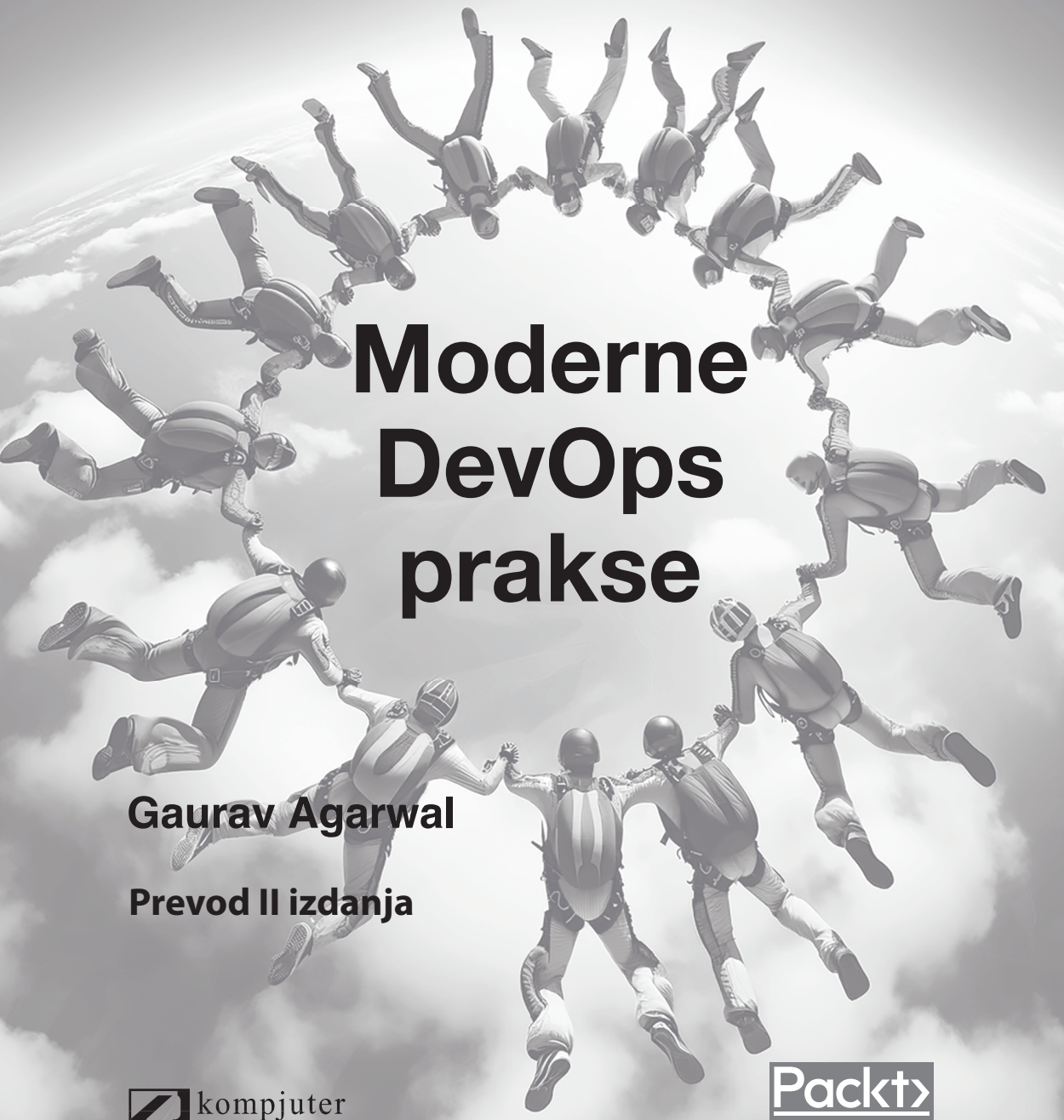
REČ UREDNIKA

*Nakon prevođenja, knjigu je recenzirao **Miroslav Ristić**, redovni profesor na Prirodno-matematičkom fakultetu Univerziteta u Nišu, sa preko 25 godina iskustva u razvoju softvera. Tako da je ova knjiga prošla kroz njegovo stručno vrednovanje, sa ciljem da se osigura da prevod bude ne samo jasan, precizan i prilagođen čitaocima, već i da održi visok kvalitet i stručnu relevantnost knjige.*

Skenirajte QR kod,
registrujte knjigu
i osvojite nagradu

ISBN: 978-86-7310-599-4

Implementacija, obezbeđivanje i upravljanje aplikacijama u javnom oblaku pomoću najsavremenijih alata



Moderne DevOps prakse

Gaurav Agarwal

Prevod II izdanja

kompjuter biblioteka

Packt›

Izdavač:

kompjuter biblioteka

Obalskih radnika 4a
Beograd, Srbija
Tel: 011/2520272
e-pošta: kombib@gmail.com
veb-sajt: www.kombib.rs

Za izdavača:
Mihailo J. Šolajić, direktor

Autor:
Gaurav Agarwal

Prevod: Nemanja Lukić
Recezent: Miroslav Ristić
Slog: Zvonko Aleksić

Znak Kompjuter biblioteke:
Miloš Milosavljević

Štampa: „Pekograf“, Zemun
Tiraž: 500
Godina izdanja: 2024.

Broj knjige: 576
Izdanje: Prvo
ISBN: 978-86-7310-599-4

Naslov originala:

Modern DevOps Practices
2ND Edition
ISBN 78-1-80512-182-4

Moderne DevOps prakse
Autorizovani prevod sa engleskog jezika.

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд

004.722
004.42:004.9

АГАРВАЛ, Гаурав, 1984-

Moderne DevOps prakse: : implementacija, obezbeđivanje i upravljanje aplikacijama u javnom oblaku pomoću najsavremenijih alata / Gaurav Agarwal ; prevod 2. izd. [Nemanja Lukić]. - Izd. 1. - Beograd: Kompjuter Biblioteka, 2024 (Zemun : Pekograf). - XXIII, 539 str.: ilustr.; 24 cm. - (Kompjuter biblioteka ; br. knj. 576)

Prevod dela: Modern DevOps Practices. - Tiraž 500. - O autoru: str. III. - Registar.

ISBN 978-86-7310-599-4

a) Технологија облака
b) Апликативни софтвер -- Пројектовање

COBISS.SR-ID 143839753

SARADNICI

O AUTORU

Gaurav Agarwal je senior inženjer za okruženja u oblaku u kompaniji ThoughtSpot, sa više od decenije iskustva kao inženjer za okruženja u oblaku i DevOps inženjer. Gaurav je pre toga bio arhitekta za rešenja u oblaku u kompaniji Capgemini, a počeo je karijeru kao softverski inženjer u kompaniji TCS. Sa impresivnom listom sertifikata, uključujući HashiCorp Certified Terraform Associate, Google Cloud Certified Professional Cloud Architect, Certified Kubernetes Administrator i Security Specialist, poseduje impresivan tehnički profil. Gauravovo obimno iskustvo obuhvata ključne uloge u postavljanju infrastrukture, upravljanju okruženjima u oblaku i implementaciji CI/CD protočnih struktura. Svedočanstvo njegove tehničke stručnosti proteže se od brojnih tehničkih blog postova do objavljene knjige, što ističe njegovu posvećenost unapređivanju oblasti.

Želim da zahvalim svojoj porodici, posebno mojoj divnoj supruzi, Deepti, što su mi pružili prostor i podršku koja mi je bila potrebna da napišem ovu knjigu. Takođe želim da zahvalim Surbhi, koja mi je pružila priliku da završim ovaj put, i Ashwini, koji me je zadržao na pravom putu. Posebna zahvalnost ide Aniketu, za recenziranje knjige. Ceo Packt uređivački tim mi je neizmerno pomogao, ali posebno želim da zahvalim Isha, Rajatu i Joshui koji su uredili veći deo mog rada.

O RECENZENTIMA

Aniket Mhala je iskusan i progresivan lider, poznat po svojoj nepokolebljivoj posvećenosti strateškoj izvrsnosti i dostignućima. Sa više od 25 godina raznovrsnog i svestranog iskustva, Aniket poseduje impresivnu kombinaciju tehničke stručnosti i poslovnog ugleda. Poznat je po izuzetnim sposobnostima u vođenju tehnoloških ideja, kreiranju i sprovođenju poslovnih i informacionih strategija, vođenju inicijativa digitalne transformacije i modernizaciji DevOps praksi. Njegovo iskustvo obuhvata modernizaciju aplikacija, besprekorno integrisanje više platformi u oblaku (AWS, Azure i OCI), dizajniranje savremenih arhitektura podataka, oblikovanje poslovnih arhitektura i precizno upravljanje složenim programima i isporukama.

Miroslav Ristić je redovni profesor na Prirodno-matematičkom fakultetu Univerziteta u Nišu, sa preko 25 godina iskustva u razvoju statističkog softvera. Posebno se ističe njegov rad na razvoju grafičkog korisničkog interfejsa R Commander za programski jezik R. Dugi niz godina recenzirao je značajan broj knjiga za izdavačku kuću Springer i časopis Journal of Applied Statistics. Od 2023. godine aktivno recenzira najaktuelnija izdanja izdavačke kuće "Kompjuter biblioteka". Nakon prevođenja, svako izdanje prolazi kroz njegovo stručno vrednovanje i recenziju prevoda, sa ciljem da se osigura da prevodi budu ne samo jasni, precizni i prilagođeni čitaocima, već i da održe visok kvalitet i stručnu relevantnost knjiga.

Kratak sadržaj

Sadržaj

POGLAVLJE 4

POGLAVLJE 6

Upravljanje naprednim resursima Kubernetes platforme 161

POGLAVLJE 7

DEO 3

POGLAVLJE 8

Predgovor

Novo i poboljšano drugo izdanje ove knjige prevazilazi osnove DevOps alata i njihove implementacije. Uz praktične primere ćete upoznati kontejnere, automatizaciju infrastrukture, usluge kontejnera bez upravljanja infrastrukturom, neprekidnu integraciju i isporuku, automatizovane implementacije, bezbednost implementacije protočne strukture, kao i upravljanje vašim uslugama u proizvodnom okruženju, sve pomoću kontejnera i sa posebnim fokusom na GitOps.

Za koga je ova knjiga

Ako ste programer, sistemski administrator ili inženjer operacija koji želi da zakorači u DevOps svet u okruženju javnih oblaka, ovo je knjiga za vas. I oni koji već rade u DevOps sferi će pronaći korisne informacije, jer knjiga detaljno objašnjava najbolje prakse, savete i trikove DevOps primene u okruženju javnih oblaka. Nije neophodno prethodno iskustvo sa kontejnerima, ali će vam osnovno poznavanje procesa razvoja i isporuke softvera pomoći da izvučete najviše iz ove knjige.

Šta knjiga obuhvata

Poglavlje 1, Savremen pristup DevOps kulturi, ističe razlike između moderne i tradicionalne DevOps kulture. Opisane su ključne tehnologije koje pokreću savremenu DevOps kulturu, sa posebnim osvrtom na ključnu ulogu kontejnera. Budući da su kontejneri relativno nova tehnologija, detaljno smo opisali najbolje prakse i tehnike razvoja, implementacije i zaštite aplikacija koje se oslanjaju na kontejnere.

Poglavlje 2, Upravljanje izvornim kodom pomoću Git sistema i GitOps metodologije, predstavlja Git, vodeći alat za upravljanje izvornim kodom, kao i njegovu primenu u razvoju i isporuci softvera kroz GitOps.

Poglavlje 3, Kontejnerizacija pomoću Docker platforme, predstavlja Docker, od instalacije i konfiguracije Docker skladištenja, preko pokretanja prvih kontejnera, do praćenja Docker okruženja pomoću sistema Journald i platforme Splunk.

Poglavlje 4, Kreiranje i upravljanje slikama kontejnera, detaljno analizira Docker slike, ključnu komponentu platforme. Proučavamo Docker slike, slojeviti model, Dockerfile direktive, pojednostavljivanje slika, konstrukciju slika i najbolje prakse za izgradnju slika. Takođe, istražićemo distroless slike i njihov značaj za DevSecOps praksu.

Poglavlje 5, Orkestracija kontejnera pomoću Kubernetes platforme, je uvod u Kubernetes. Instaliraćemo Kubernetes pomoću alata minikube i kinD, proučiti arhitektonsku osnovu platforme, kao i njene osnovne gradivne elemente, kao što su ljuske, kontejneri, ConfigMap objekti, tajne, probe i ljuske sa više kontejnera.

Poglavlje 6, Upravljanje naprednim resursima Kubernetes platforme, detaljnije se bavi kompleksnim aspektima platforme, uključujući mrežne funkcionalnosti, DNS, usluge, implementacije, automatsko horizontalno skaliranje ljuski i resurse StatefulSet.

Poglavlje 7, Kontejneri kao usluga (CaaS) i računarstvo bez servera za kontejnere, istražuje hibridnu prirodu Kubernetes platforme, koja povezuje IaaS i PaaS paradigme. Takođe, objašnjene su usluge kontejnera bez servera, kao što je AWS ECS, ali i alternativne opcije - Google Cloud Run i Azure Container Instances. Zaključak poglavlja posvećen je platformi otvorenog koda Knative, koja je prilagođena okruženju oblaka i namenjena implementaciji aplikacija bez upravljanja poslužiteljima.

Poglavlje 8, Infrastruktura kao kod (IaC) sa alatom Terraform, uvodi IaC koncept i objašnjava njegove osnovne principe. Kroz praktične primere pokazaćemo kako kreirati grupu resursa i virtuelnu mašinu na Azure platformi pomoću alata Terraform, pri čemu ćemo detaljno razjasniti osnovne pojmove vezane za Terraform.

Poglavlje 9, Upravljanje konfiguracijom pomoću alata Ansible, predstavlja upravljanje konfiguracijom pomoću alata Ansible, kao i njegove osnovne principe, prilikom konfigurisanja MySQL i Apache aplikacija na Azure virtuelnim mašinama.

Poglavlje 10, Nepromenljiva infrastruktura sa alatom Packer, bavi se nepromenljivom infrastrukturom i alatom Packer. Zajedno sa saznanjima iz poglavlja 8, Infrastruktura kao kod (IaC) sa alatom Terraform, i poglavlja 9, Upravljanje konfiguracijom pomoću alata Ansible, pokrenućemo Linux, Apache, MySQL i PHP (LAMP) konfiguraciju zasnovanu na IaaS okruženju, na platformu Azure.

Poglavlje 11, Neprekidna integracija pomoću GitHub akcija i alata Jenkins, objašnjava neprekidnu integraciju iz perspektive kontejnera, procenjujući različite alate i metodologije za neprekidnu izgradnju aplikacija koje se oslanjaju na kontejnere. Opisani su alati GitHub akcije i Jenkins, kada i kako ih koristiti, prilikom implementiranja primera distribuirane aplikacije zasnovane na mikroservisima, Blog aplikacije.

Poglavlje 12, Neprekidna implementacija/isporuka pomoću alata Argo CD, opisuje neprekidnu implementaciju/isporuku pomoću alata Argo CD. Kao savremen alat za neprekidnu isporuku zasnovan na GitOps metodologiji, Argo CD olakšava implementaciju i upravljanje aplikacijama u kontejnerima. Iskoristićemo njegovu snagu za implementaciju naše Blog aplikacije.

Poglavlje 13, Osiguravanje i testiranje vaše CI/CD protočne strukture, opisuje više strategija za osiguravanje procesa implementacije kontejnera, uključujući analizu slika kontejnera, skeniranje ranjivosti, upravljanje tajnama, skladištenje, integraciono testiranje i binarnu autorizaciju. Integrisaćemo ove tehnike da bismo unapredili bezbednost naših CI/CD protočnih struktura.

Poglavlje 14, Ključni pokazatelji performansi (KPI) vaše proizvodne usluge, predstavlja inženjering pouzdanosti veb sajta i opisuje niz ključnih pokazatelja performansi koji su od vitalnog značaja za efikasno upravljanje distribuiranim aplikacijama u proizvodnji.

Poglavlje 15, Implementacija kontrole saobraćaja, bezbednosti i nadzora pomoću Istio tehnologije, predstavlja popularnu tehnologiju mreža usluga - Istio. Opisaćemo različite tehnike za svakodnevne operacije u proizvodnji, uključujući upravljanje saobraćajem, bezbednosne mere i poboljšanja praćenja naše Blog aplikacije.

Da najbolje iskoristite ovu knjige, potrebno vam je sledeće:

- Pretplata na Azure, da biste mogli da uradite neke od vežbi: trenutno, Azure nudi besplatnu probnu verziju za 30 dana, sa $200 vrednosti besplatnih kredita; možete se prijaviti na https://azure.microsoft.com/en-in/free.
- Pretplata na AWS: trenutno, AWS nudi besplatnu uslugu za neke proizvode. Možete se prijaviti na https://aws.amazon.com/free. Knjiga koristi neke plaćene usluge, ali smo se trudili da smanjimo njihovu upotrebu koliko je to bilo moguće.
- Pretplata na Google Cloud Platform: trenutno, Google Cloud Platform ima besplatnu probnu verziju od $300 za 90 dana; možete da se prijavite na https://console.cloud.google.com/.
- Za neka poglavlja, biće potrebno da napravite kopiju sledećeg GitHub spremišta da biste nastavili sa vežbama: https://github.com/PacktPublishing/Modern-DevOps-Practices-2e

KNJIGA SE OSLANJA NA SLEDEĆI SOFTVER/HARDVER	POTREBAN OPERATIVNI SISTEM
Google Cloud Platform	Windows, macOS ili Linux
AWS	Windows, macOS ili Linux
Azure	Windows, macOS ili Linux
Linux VM	Ubuntu 18.04 LTS ili noviji

Ako koristite digitalnu verziju ove knjige, savetujemo vam da sami kucate kod ili da pristupite kodu iz GitHub spremišta knjige (link je dostupan u sledećem odeljku). Time ćete izbeći potencijalne greške, do kojih može doći ako kopirate kod.

Preuzmite datoteke sa primerima koda

Datoteke sa primerima koda za ovu knjigu možete preuzeti sa GitHub spremišta, na adresi https://github.com/PacktPublishing/Modern-DevOps-Practices-2e. Ukoliko dođe do ažuriranja koda, to će biti objavljeno u GitHub spremištu.

Takođe, drugi paketi koda iz našeg bogatog kataloga knjiga i video materijala dostupni su na https://github.com/PacktPublishing/. Pogledajte ih!

Korišćene konvencije

U ovoj knjizi korišćene su određene tekstualne konvencije.

`Kod u tekstu`: označava delove koda u tekstu, nazive tabela baza podataka, nazive direktorijuma, nazive datoteka, ekstenzije datoteka, putanje, lažne URL adrese, korisnički unos i Twitter korisnička imena. Evo primera: „Hajde da pokušamo da podnesemo zahtev za povlačenje da bismo spojili kod iz grane `funkcija/funkcija1` sa `master` granom".

Blok koda izgleda ovako:

```
import os
import datetime
from flask import Flask
app = Flask(__name__)
@app.route('/')
def current_time():
  ct = datetime.datetime.now()
  return 'Trenutno vreme je : {}!\n'.format(ct)
if __name__ == "__main__":
  app.run(debug=True,host='0.0.0.0')
```

Svaki unos ili izlaz sa komandne linije napisan je na sledeći način:

```
$ cp ~/modern-devops/ch13/install-external-secrets/app.tf \
terraform/app.tf
$ cp ~/modern-devops/ch13/install-external-secrets/\
external-secrets.yaml manifests/argocd/
```

Podebljano: označava novi termin, važnu reč ili reči koje su prikazane na ekranu. Na primer, reči u menijima ili okvirima za dijalog su podebljane. Evo primera: „Kliknite na dugme **Create pull request** da biste kreirali zahtev za povlačenje".

Saveti ili važne napomene prikazane su ovako.

Saveti
ili
Važne napomene
Prikazani su ovako.

Postanite član Kompjuter biblioteke

Kupovinom jedne naše knjige stekli ste pravo da postanete član Kompjuter biblioteke. Kao član možete da kupujete knjige u pretplati sa 40% popusta i učestvujete u akcijama kada ostvarujete popuste na sva naša izdanja. Potrebno je samo da se prijavite preko formulara na našem sajtu.
Link za prijavu: kombib.rs/kblista.php

Skenirajte QR kod
registrujte knjigu
i osvojite nagradu

DEO 1

Osnove moderne DevOps kulture

U ovom delu ćete upoznati modernu DevOps kulturu i kontejnere i izgradićete snažnu osnovu za upoznavanje tehnologije kontejnera. Naučićete kako kontejneri pomažu organizacijama da grade distribuirane, skalabilne i pouzdane sisteme u oblaku.

Ovaj deo se sastoji od sledećih poglavlja:

- *Poglavlje 1, Savremen pristup DevOps kulturi*
- *Poglavlje 2, Upravljanje izvornim kodom pomoću Git sistema i GitOps metodologije*
- *Poglavlje 3, Kontejnerizacija pomoću Docker platforme*
- *Poglavlje 4, Kreiranje i upravljanje slikama kontejnera*

1

Savremen pristup DevOps kulturi

Ovo prvo poglavlje pruža osnovno znanje o DevOps praksama, procesima i alatima. Razumećete savremenu DevOps kulturu i njene razlike u odnosu na tradicionalnu DevOps kulturu. Takođe, upoznaćete kontejnere i do detalja razumeti kako kontejneri unutar oblaka menjaju celokupan IT pejzaž, da bismo mogli da nadogradimo te osnove pomoću ove knjige. Iako fokus ove knjige nisu samo kontejneri i njihova orkestracija, savremene DevOps prakse ih ističu kao ključni element.

U ovom poglavlju su obrađene sledeće glavne teme:

- Šta je DevOps?
- Računarstvo u oblaku
- Koncept modernih aplikacija prilagođenih oblaku
- Razlike između savremene i tradicionalne DevOps kulture
- Potreba za kontejnerima
- Kontejneri i savremene DevOps prakse
- Migracija sa virtuelnih mašina na kontejnere

Do kraja ovog poglavlja, trebalo bi da razumete sledeće ključne aspekte:

- Šta je DevOps i kakvu ulogu ima u savremenom IT pejzažu
- Šta je računarstvo u oblaku i kako je promenilo IT usluge
- Kako izgleda moderna aplikacija prilagođena oblaku i kako je promenila DevOps kulturu
- Zašto su nam potrebni kontejneri i koje probleme rešavaju
- Arhitekturu kontejnera i kako funkcioniše

- Kako kontejneri doprinose savremenim DevOps praksama
- Osnovni koraci za migraciju na kontejnere, sa arhitekture zasnovane na virtuelnim mašinama

Šta je DevOps?

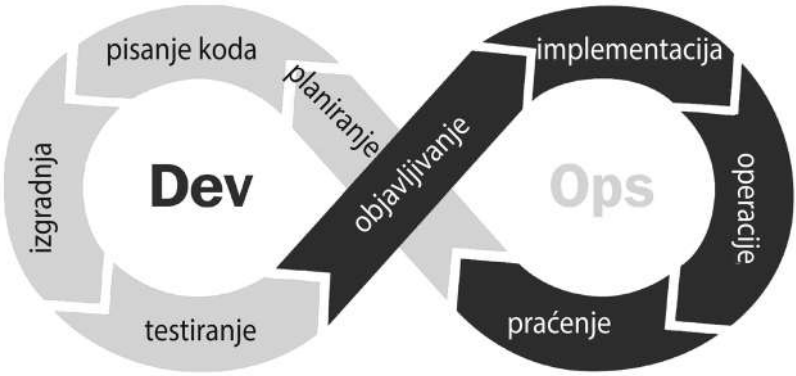
Kao što znate, razvoj softvera i operacije su, tradicionalno, u nadležnosti odvojenih timova, sa različitim ulogama i odgovornostima. Programeri razvojnog tima su fokusirani na pisanje koda i kreiranje novih funkcionalnosti, dok su operativni timovi zaduženi za implementaciju i upravljanje softverom u proizvodnim okruženjima. Ova podela, često, podrazumeva nedostatak komunikacije, spore cikluse objavljivanja i neefikasne procese.

DevOps premošćava jaz između razvoja i operacija promovišući kulturu saradnje, deljenih odgovornosti i neprekidnog povratnog informisanja, pomoću automatizacije celog ciklusa razvoja softvera.

To je skup principa i praksi, ali i filozofija koja podstiče učešće razvojnih i operativnih timova tokom celog ciklusa razvoja softvera, uključujući održavanje softvera i operacije. Zbog toga organizacije koriste različite procese i alate koji pomažu u automatizaciji procesa isporuke softvera. Cilj je poboljšati brzinu i agilnost, smanjiti vreme ciklusa objavljivanja koda protočnim strukturama **neprekidne integracije i neprekidne dostave** (**CI/CD**), i pratiti aplikacije koje se izvršavaju u proizvodnji.

DevOps tim treba da teži tome da, umesto jasno definisanih, izolovanih grupa, koje se bave razvojem, operacijama i podrškom za obezbeđenje kvaliteta, bude jedinstven tim koji brine o celokupnom ciklusu razvoja softvera - to jest, tim će izgraditi, implementirati i pratiti softver. Kombinovani tim prati celu aplikaciju, umesto određenih funkcija. To ne znači da članovi tima nisu specijalizovani za određene oblasti, već je ideja da programeri razvojnog tima poznaju operacije, a inženjeri za operacije poznaju razvoj softvera. Tim za obezbeđenje kvaliteta tesno sarađuje sa programerima razvojnog tima i inženjerima za operacije da bi razumeo poslovne zahteve i različite probleme na koje nailaze u praksi. Na osnovu tih saznanja, trude se da proizvod koji razvijaju ispunjava poslovne zahteve i rešava probleme na koje nailazi u praksi.

U tradicionalnom razvojnom timu, radne zadatke obično zadaje poslovna jedinica i njeni arhitekti. Međutim, za DevOps tim postoje dva izvora svakodnevnih radnih zadataka - poslovna jedinica i njeni arhitekti, ali i korisnici, sa problemima na koje nailaze dok koriste aplikaciju u proizvodnji. Stoga, umesto da slede jednosmeran put isporuke softvera, DevOps prakse prate beskonačnu petlju, kao što je prikazano na sledećoj slici:



Slika 1.1 – Beskonačna DevOps petlja

Da bi se osigurala dobra saradnja između ljudi sa različitim veštinama, DevOps je fokusiran na automatizaciju i alate. Cilj DevOps kulture je pokušaj automatizacije ponavljajućih zadataka, koliko je to moguće, a usmeravanje pažnje na važnije stvari. Ovo osigurava kvalitet proizvoda i brzu isporuku. DevOps je fokusiran na *ljude*, *procese* i *alate*, gde najveći značaj imaju ljudi, a najmanji alati. Uglavnom, koristimo alate da bismo automatizovali procese koji pomažu ljudima da postignu prave ciljeve.

Slede neke od osnovnih ideja i terminologija DevOps kulture. U ovoj knjizi ćemo se fokusirati na sve što sledi:

- **Neprekidna integracija (CI)**

  Neprekidna integracija je praksa razvoja softvera koja uključuje često spajanje promena u kodu, od strane više programera razvojnog tima, u deljeno spremište, obično nekoliko puta dnevno. Ovo osigurava da vaši programeri razvojnog tima redovno spajaju kod u centralnom spremištu, gde se automatski izvršava izgradnja i testiranje, da bi tim, u realnom vremenu, imao povratne informacije. Ovo značajno smanjuje trajanje ciklusa razvoja softvera i poboljšava kvalitet koda. Ovaj proces ima za cilj da greške u kodu smanji na minimum, na početku ciklusa izgradnje, umesto kasnije, tokom faza testiranja. Na taj način su integracioni problemi otkriveni ranije i osigurava se da softver uvek bude u stanju za objavljivanje.

- **Neprekidna dostava (CD)**

  Neprekidna dostava je isporuka spremnog, testiranog softvera u proizvodna okruženja. Dakle, CD protočna struktura će izgraditi vaše promene u pakete i pokrenuti integracione i sistematske testove na njima. Kada ste temeljno testirali svoj kod, možete automatski (ili uz odobrenje) da implementirate promene u testovna i proizvodna okruženja. Dakle, neprekidna dostava ima za cilj da najnoviji skup testiranih proizvoda bude spreman za implementaciju.

- **Infrastruktura kao kod (IaC)**

  Infrastruktura kao kod je praksa u razvoju softvera koja obuhvata upravljanje i isporuku resursa infrastrukture, kao što su serveri, mreže i skladišta, pomoću

koda i konfiguracionih datoteka, umesto ručnih procesa. IaC tretira infrastrukturu kao softver, što omogućava timovima da definišu resurse infrastrukture i da upravljaju njima na programabilan i verzioniran način. Sa pojavom virtuelnih mašina, kontejnera i oblaka, tehnološka infrastruktura je u velikoj meri postala virtuelna. To znači da možemo da izgradimo infrastrukturu pomoću API poziva i šablona. Sa modernim alatima, takođe možemo deklarativno da izgradimo infrastrukturu u oblaku. To znači da sada možemo da izgradimo IaC, sačuvamo potreban kod za izgradnju infrastrukture u spremištu izvornog koda, kao što je Git, i da upotrebimo CI/CD protočnu strukturu za pokretanje i upravljanje infrastrukturom.

- **Konfiguracija kao kod (CaC)**

  Konfiguracija kao kod je praksa u razvoju softvera i administraciji sistema koja obuhvata upravljanje i isporučivanje konfiguracionih podešavanja pomoću koda i sistema za kontrolu verzija. Ona tretira konfiguraciona podešavanja kao proizvode koda, što omogućava timovima da definišu, skladište i upravljaju konfiguracijom na programabilan i reproduktivan način. Ranije, serveri su bili ručno kreirani, od početka, i retko su menjani. Međutim, sa elastičnom infrastrukturom i naglaskom na automatizaciju, konfiguracijom je moguće upravljati pomoću koda. Konfiguracija kao kod tesno sarađuje sa Infrastrukturom kao kodom pri izgradnji skalabilne infrastrukture otporne na greške, da bi aplikacija mogla da se izvršava bez problema.

- **Praćenje i evidencija zapisa**

  Praćenje i evidencija zapisa su ključne prakse u razvoju softvera i operativnim aktivnostima koje obuhvataju prikupljanje i analiziranje podataka o ponašanju i performansama softverskih aplikacija i sistema. Pružaju uvide u stanje, dostupnost i performanse softvera, što omogućava timovima da identifikuju i rešavaju probleme i da donose informisane odluke o poboljšanjima. Praćenje i evidencija zapisa su deo nadzora, ključnog područja za svaki DevOps tim - tj. svest o tome da aplikacija ima probleme i izuzetke, zahvaljujući praćenju i rešavanje problema zahvaljujući prikupljanju zapisa. Ove prakse i alati su oči tima i ključno područje DevOps skupa tehnologija. Pored toga, imaju značajan doprinos za izgradnju radnih zadataka DevOps tima.

- **Komunikacija i saradnja**

  Komunikacija i saradnja su ključni aspekti DevOps praksi. One promovišu efikasan timski rad, deljenje znanja i optimizovane radne tokove u razvoju softvera, operacijama i ostalim segmentima u ciklusu isporuke softvera. Komunikacija i saradnja omogućavaju dobro funkcionisanje DevOps tima. Prošli su dani kada se komuniciralo putem elektronske pošte. Umesto toga, moderni DevOps timovi koriste alate za praćenje radnih zadataka i agilne alate, vode evidenciju svojih saznanja na wiki platformama, a komuniciraju neposredno, pomoću četa i alata za **trenutne poruke** (**IM**).

Ovo je samo nekoliko osnovnih aspekata DevOps praksi i alata, a došlo je do novih promena sa pojavom kontejnera i oblaka - tj. modernog skupa aplikacija prilagođenih oblaku. Sada

kada smo pomenuli neke od ključnih pojmova u ovom delu, hajde da vidimo šta podrazumevamo pod pojmom oblak i računarstvo u oblaku.

Računarstvo u oblaku

Tradicionalno, softverske aplikacije su pokretane na serverima koji su se nalazili na internim računarima (serverima), poznatim kao **centri podataka**. To znači da je organizacija morala da kupi fizičku računarsku i mrežnu infrastrukturu, što je predstavljalo značajan materijalni izdatak, a operativni rashodi su takođe bili visoki. Pored toga, serveri su se često kvarili i bilo je potrebno održavati ih, što je predstavljalo prepreku za manje kompanije koje su želele da isprobaju stvari, s obzirom na ogromne **kapitalne izdatke** (**CapEx**). To je značilo da su projekti morali biti dobro planirani i arhitektonski osmišljeni, a infrastruktura naručena i postavljena u skladu s tim. To znači da nije bilo moguće lako proširiti infrastrukturu u kratkom vremenskom roku, radi prilagođavanja promenljivim potrebama ili povećanju opterećenja. Na primer, ako ste započeli sa malim resursima i niste očekivali veliki saobraćaj, a vaša veb stranica iznenada postane popularna, vaši serveri nisu mogli da podrže taj obim saobraćaja i dolazilo je do kvarova. Brzo skaliranje je podrazumevalo kupovinu nove opreme za centar podataka, što je oduzimalo vreme i dovodilo do propuštanja poslovnih prilika.

U cilju rešavanja ovog problema, internet divovi kao što su Amazon, Microsoft i Google počeli su da razvijaju javnu infrastrukturu za pokretanje svojih internet sistema, da bi, potom, odlučili da tu infrastrukturu stave na raspolaganje javnosti. To je dovelo do novog fenomena poznatog kao **računarstvo u oblaku**.

Računarstvo u oblaku se odnosi na pružanje računarskih resursa po potrebi, kao što su serveri, skladišta, baze podataka, umrežavanje, softver i analitika, preko interneta. Umesto smeštanja ovih resursa lokalno, na fizičku infrastrukturu, računarstvo u oblaku omogućava organizacijama pristup i korišćenje računarskih usluga **pružaoca usluga u oblaku** (**CSP**). Neki od vodećih javnih pružaoca usluga u oblaku su **Amazon Web Services** (**AWS**), **Microsoft Azure** i **Google Cloud Platform**.

U računarstvu u oblaku, pružalac usluga u oblaku poseduje i održava osnovnu infrastrukturu i resurse, a korisnici ili organizacije koriste te resurse za svoje aplikacije i usluge.

Jednostavno rečeno, računarstvo u oblaku podrazumeva korišćenje tuđeg centra podataka za pokretanje vaše aplikacije, što bi trebalo da bude po potrebi. Da bi to bilo moguće, trebalo bi da postoji kontrolna tabla na veb portalu, API interfejsi i slični alati dostupni preko interneta. Za ove usluge, potrebno je platiti zakup za resurse koje isporučujete (ili koristite), po principu naplate po upotrebi.

Na taj način, računarstvo u oblaku pruža brojne prednosti i otvara nove mogućnosti za poslovanje, kojih ranije nije bilo. Neke od tih prednosti su:

- **Skalabilnost**: resursi u oblaku su skalabilni. To znači da možete dodati nove servere ili resurse postojećim serverima, kada je potrebno. Takođe, možete automatizovati skaliranje saobraćaja za vašu aplikaciju. To znači da vam je za pokretanje aplikacije potreban samo jedan server, ali ako se iznenada javi veća potražnja ili u vreme najvećeg opterećenja, vaša aplikacija može automatski da se proširi na pet servera pomoću API interfejsa za računarstvo u oblaku

i ugrađenih upravljačkih resursa. To daje kompanijama veliku moć, jer je moguće početi na malo, bez mnogo brige o budućoj popularnosti i skaliranju.

- **Manji troškovi**: računarstvo u oblaku sledi model **naplate po upotrebi**, gde korisnici plaćaju samo resurse i usluge koje koriste. Ovo eliminiše potrebu za početnim kapitalnim izdacima za hardver i infrastrukturu. Za kompanije je uvek jeftinije da iznajme resurse nego da ulažu u računarsku opremu. Stoga, pošto plaćate samo za resurse koje koristite u određenom periodu, nema potrebe za obezbeđivanjem prekomernih resursa za buduće opterećenje. To dovodi do značajnih ušteda, za većinu malih i srednjih preduzeća.
- **Fleksibilnost**: resursi u oblaku više nisu samo serveri. Možete dobiti mnoge druge stvari, poput jednostavnih rešenja za skladištenje objekata, mrežnog i blok skladišta, upravljanih baza podataka, usluga kontejnera i još mnogo toga. Ovo vam pruža puno fleksibilnosti u vezi sa onim što možete da uradite sa svojom aplikacijom.
- **Pouzdanost**: resursi računarstva u oblaku su vezani **ugovorima o nivou usluge (SLA)**, ponekad na nivou dostupnosti od 99,999%. To znači da većina vaših resursa u oblaku nikada neće biti nedostupna; ako i dođe do toga, nećete to ni primetiti, zahvaljujući ugrađenoj redundantnosti.
- **Bezbednost**: budući da kompanije za računarstvo u oblaku pokreću aplikacije za različite klijente, često imaju strožu bezbednosnu mrežu koju nije moguće izgraditi u lokalnom okruženju. Postoji tim stručnjaka za bezbednost koji 24 sata dnevno, 7 dana u nedelji nadgleda infrastrukturu, kao i usluge koje podrazumevaju šifrovanje, kontrolu pristupa i automatsko otkrivanje pretnji. Kao rezultat toga, kada je dizajnirana na odgovarajući način, aplikacija koja se izvršava u oblaku je mnogo bezbednija.

Raznovrsna ponuda usluga računarstva u oblaku uključuje sledeće:

- **Infrastruktura kao usluga (IaaS)** slična je pokretanju vaše aplikacije na serverima. To je model usluge računarstva u oblaku koji pruža virtuelizovane računarske resurse preko interneta. U IaaS okruženju, organizacije mogu pristupiti i upravljati osnovnim IT infrastrukturnim komponentama, poput virtuelnih mašina, skladišta i umrežavanja, bez kupovine i održavanja fizičkog hardvera. U IaaS modelu, pružalac usluga u oblaku poseduje i upravlja osnovnom fizičkom infrastrukturom, uključujući servere, skladišta, mrežnu opremu i centre podataka. Korisnici ili organizacije, s druge strane, imaju kontrolu nad **operativnim sistemima (OSs)**, aplikacijama i konfiguracijama koje se izvršavaju na virtuelizovanoj infrastrukturi.
- **Platforma kao usluga (PaaS)** pruža vam apstrakciju gde se možete fokusirati na svoj kod i prepustiti upravljanje aplikacijama usluzi u oblaku. To je model usluge računarstva u oblaku koji programerima razvojnih timova pruža platformu i okruženje za razvoj, implementaciju i upravljanje aplikacijama, bez brige o osnovnim infrastrukturnim komponentama. PaaS model apstrahuje složenosti upravljanja infrastrukturom, što omogućava programerima razvojnih timova da se fokusiraju na razvoj i implementaciju aplikacija. U PaaS modelu, pružalac usluga u oblaku nudi platformu koja obuhvata operativne sisteme, radne okvire za razvoj, okruženja za izvršavanje

i razne alate i usluge potrebne za podršku ciklusa razvoja aplikacija. Korisnici ili organizacije mogu da koriste ove resurse platforme za razvoj, testiranje, implementaciju i skaliranje svojih aplikacija.

- **Softver kao usluga (SaaS)** isporučuje gotovu aplikaciju za upotrebu, kao što je usluga praćenja, koja je odmah dostupna za korišćenje i lako se povezuje sa vašom aplikacijom. U SaaS modelu, pružalac usluge u oblaku skladišti i upravlja softverskom aplikacijom, uključujući infrastrukturu, servere, baze podataka i održavanje. Korisnici ili organizacije mogu pristupiti aplikaciji putem veb pregledača ili klijenta zasnovanog na veb tehnologiji. Obično se plaća pretplata na osnovu upotrebe, a softver se isporučuje kao usluga po zahtevu.

Pojava računarstva u oblaku dovela je do nove pojave u industriji, aplikacija prilagođenih oblaku. O njima govorimo u sledećem odeljku.

Koncept modernih aplikacija prilagođenih oblaku

Kada kažemo prilagođene oblaku, govorimo o aplikacijama koje su izgrađene da se pokreću na oblaku. Aplikacija prilagođena oblaku je dizajnirana da se pokreće u oblaku, da koristi sve prednosti i pogodnosti oblaka i da u najvećoj meri iskoristi usluge oblaka.

Ove aplikacije su inherentno **skalabilne**, **fleksibilne** i **otporne** (otporne na greške). One se u velikoj meri oslanjaju na usluge oblaka i automatizaciju.

Neke od karakteristika moderne aplikacije prilagođene oblaku su:

Arhitektura mikroservisa: za moderne aplikacije prilagođene oblaku obično se koristi arhitektura mikroservisa. Mikroservisi su aplikacije koje su razložene na više manjih, labavo povezanih delova, sa nezavisnim poslovnim funkcijama. Nezavisni mikroservisi mogu biti napisani na različitim programskim jezicima, u zavisnosti od potrebe ili specifične funkcionalnosti. Ove manje delove je moguće nezavisno skalirati, fleksibilno pokretati i dizajnirani su da budu otporni.

Kontejnerizacija: aplikacije zasnovane na mikroservisima obično koriste kontejnere za izvršavanje. Kontejneri pružaju **dosledno**, **prenosivo** i **lagano** okruženje za izvršavanje aplikacija, garantujući da su sve neophodne zavisnosti i konfiguracije integrisane unutar kontejnera. Kontejneri, bez razlike, mogu da se pokreću na svim okruženjima i platformama u oblaku.

DevOps i automatizacija: aplikacije prilagođene oblaku intenzivno koriste moderne DevOps prakse i alate, pa se u velikoj meri oslanjaju na automatizaciju. To olakšava razvoj, testiranje i operativnost aplikacije. Takođe, automatizacija donosi **skalabilnost**, **otpornost** i **doslednost**.

Dinamička orkestracija: aplikacije prilagođene oblaku izgrađene su da se skaliraju i inherentno su otporne na greške. Ove aplikacije su obično **privremene** (**prolazne**); stoga se kopije usluga mogu pojaviti i nestati prema potrebi. Dinamičke platforme za orkestraciju, kao što su **Kubernetes** i **Docker Swarm** služe za upravljanje ovim uslugama. Ovi alati pomažu u izvršavanju aplikacije u promenljivim uslovima i obrascima saobraćaja.

Korišćenje usluga za podatke prilagođenih oblaku: aplikacije prilagođene oblaku obično koriste upravljane usluge za podatke u oblaku, kao što su **skladište**, **baze podataka**, **keširanje** i sistemi za **razmenu poruka**, da bi komunikacija između više usluga bila moguća.

Ovi sistemi ističu važnost DevOps kulture, a moderna DevOps kultura je razvijena da bi se upravljalo tim sistemima. Dakle, sada ćemo razmotriti razliku između tradicionalne i moderne DevOps kulture.

Razlike između savremene i tradicionalne DevOps kulture

Tradicionalan DevOps pristup obuhvatao je formiranje tima koji se sastojao od članova za razvoj (**Dev**), obezbeđenje kvaliteta (**QA**) i operativnost (**Ops**), i rad na ubrzanju razvoja kvalitetnijeg softvera. Međutim, iako je fokus bio na automatizaciji isporuke softvera, alati za automatizaciju, kao što su **Jenkins**, **Git** i drugi, instalirani su i održavani ručno. To je dovelo do novog problema, jer je bilo potrebno upravljati još jednim skupom IT infrastrukture. Na kraju se sve svodilo na infrastrukturu i konfiguraciju, a fokus je bio na automatizaciji procesa automatizacije.

Sa pojavom kontejnera i nedavnim procvatom okruženja javnog oblaka, moderan DevOps pristup je stupio na scenu, što podrazumeva opštu automatizaciju. Od isporuke infrastrukture, do konfigurisanja alata i procesa, za sve postoji kod. Dakle, sada imamo **Infrastrukturu kao kod**, **Konfiguraciju kao kod**, **nepromenljivu infrastrukturu** i **kontejnere**. Ovaj pristup DevOps praksama je moderna DevOps kultura, a to je fokus ove knjige.

Sledeća tabela opisuje neke od ključnih sličnosti i razlika između moderne i tradicionalne DevOps kulture:

ASPEKT	MODERNA DEVOPS KULTURA	TRADICIONALNA DEVOPS KULTURA
Isporuka softvera	Fokus na CI/CD protočne strukture, automatizovano testiranje i automatizaciju implementacije.	Fokus na CI/CD protočne strukture, automatizovano testiranje i automatizaciju implementacije.
Upravljanje infrastrukture	IaC model se često koristi za isporučivanje i upravljanje resursima infrastrukture. Česta je upotreba platformi u oblaku i tehnologija kontejnerizacije.	Ručno isporučivanje i konfigurisanje infrastrukture, uz često oslanjanje na tradicionalne centre podataka i ograničenu automatizaciju.
Implementacija aplikacija	Tehnologije kontejnera i orkestracije kontejnera, kao što su Docker i Kubernetes, su usvojene da bi se obezbedila prenosivost i skalabilnost aplikacija.	Koriste se tradicionalne metode implementacije, poput direktnog implementiranja aplikacija na virtuelne mašine ili na fizičke servere bez kontejnerizacije.
Skalabilnost i otpornost	Koristi se automatsko skaliranje mogućnosti platformi u oblaku i orkestracija kontejnera, kao odgovor na promenljiva opterećenja. Fokus je na visokoj dostupnosti i otpornosti na greške.	Skalabilnost se postiže vertikalnim skaliranjem (dodavanjem resursa postojećim serverima) ili ručnim planiranjem kapaciteta. Visoka dostupnost postiže se ručnim dodavanjem redundantnih servera, ručno. Elastičnost ne postoji, a tolerancija na greške nije u fokusu.

ASPEKT	MODERNA DEVOPS KULTURA	TRADICIONALNA DEVOPS KULTURA
Praćenje i evidencija zapisa	Intenzivna upotreba alata za praćenje, beleženje i analitiku u realnom vremenu, da bi se stekli uvidi u performanse aplikacija i infrastrukture.	Ograničena praksa praćenja i beleženja, sa manje alata i analitike na raspolaganju.
Saradnja i kultura	Naglašava saradnju, komunikaciju i deljeno vlasništvo, između timova za razvoj i operativnost (DevOps kultura).	Naglašava saradnju, komunikaciju i deljeno vlasništvo između timova za razvoj i operativnost (DevOps kultura).
Bezbednost	Bezbednost je integrisana u proces razvoja pomoću **DevSecOps** praksi. Testiranje bezbednosti i skeniranje ranjivosti su automatizovani.	Mere bezbednosti često se primenjuju ručno a njima upravlja odvojeni tim za bezbednost. Ograničeno automatizovano testiranje bezbednosti u ciklusu razvoja softvera.
Brzina isporuke	Brzo i često implementiranje softverskih ažuriranja putem automatizovanih protočnih struktura omogućava brži plasman na tržište.	Brza implementacija aplikacija, ali nedostaje automatizacija implementacije infrastrukture.

Tabela 1.1 - Ključne sličnosti i razlike između moderne i tradicionalne DevOps kulture

Važno je napomenuti da razlika između moderne i tradicionalne DevOps kulture nije strogo jasna, jer organizacije mogu da usvoje različite prakse i tehnologije. Fokus modernog DevOps pristupa je na upotrebi tehnologija oblaka, automatizacije, kontejnerizacije i DevSecOps principa, da bi se unapredile saradnja, agilnost i efikasnost razvoja i implementacije softvera.

Kao što je već pomenuto, kontejneri pomažu u implementaciji moderne DevOps kulture i predstavljaju njenu osnovu. U sledećem odeljku smo detaljno opisali kontejnere.

Potreba za kontejnerima

Kontejneri su u poslednje vreme veoma popularni, i to iz odličnog razloga. Oni rešavaju najvažniji problem računarske arhitekture - *pokretanje pouzdanog*, *distribuiranog softvera*, *sa skoro beskonačnom skalabilnošću*, *u bilo kom računarskom okruženju*.

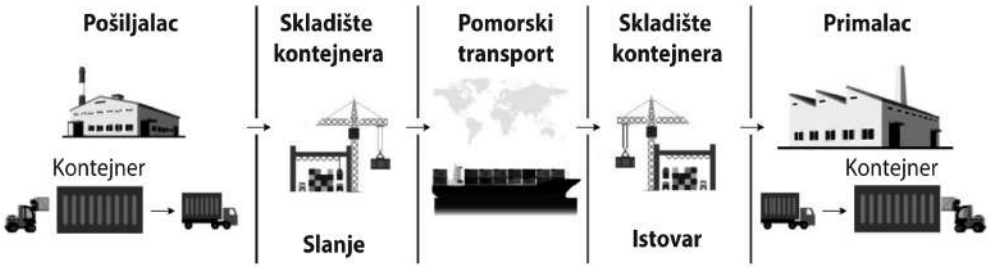
Omogućili su potpuno novu disciplinu u softverskom inženjerstvu - *mikroservise*. Takođe su u tehnologiju uveli koncept *pakuj jednom, implementiraj bilo gde*. U kombinaciji sa oblakom i distribuiranim aplikacijama, kontejneri i tehnologija orkestracije kontejnera doveli su do novog trenda u industriji – *prilagođenost oblaku* – što je dovelo da najvećih promena IT ekosistema.

Pre nego što se upustimo u detalje tehničkih pojedinosti, predstavićemo kontejnere na jednostavan i razumljiv način.

Kontejneri su dobili ime po kontejnerima za transport. Objasniću kontejnere pomoću analogije kontejnera za transport. U prošlosti, zbog poboljšanja transportnih mogućnosti, velike količine robe su transportovane sa različitih geografskih lokacija. Zbog raznolikosti robe, koja je prevožena različitim prevoznim sredstvima, utovar i istovar robe bio je ogroman problem, na svakoj transportnoj tački. Osim toga, zbog sve većih troškova radne snage, za transportne kompanije nije bilo praktično da rade u velikom obimu, uz zadržavanje niskih cena.

Takođe, često je dolazilo do oštećenja robe, gubljenja ili mešanja sa drugim pošiljkama, zbog izostanka izolacije. Postojala je potreba za standardnim načinom transporta robe, koji bi omogućio potrebnu izolaciju pošiljki i olakšao utovar i istovar robe. Transportna industrija je napravila kontejnere za transport, kao elegantno rešenje tog problema.

Tako su kontejneri za transport pojednostavili transportnu industriju. U standardizovanim kontejnerima bilo je moguće transportovati robu sa jednog mesta na drugo, samo transportom kontejnera. Isti kontejner bilo je moguće koristiti u drumskom saobraćaju, na vozovima ili brodovima. Operatori tih vozila nisu morali da brinu o sadržaju kontejnera. Sledeća slika prikazuje celokupan radni tok, radi lakšeg razumevanja:



Slika 1.2 – Radni tok transporta kontejnera

Slično, pojavili su se problemi prenosivosti softvera i upravljanja računarskim resursima u softverskoj industriji. U standardnom ciklusu razvoja softvera, deo softvera prolazi kroz više okruženja, a ponekad brojne aplikacije dele isti operativni sistem. Mogu postojati razlike u konfiguracijama različitih okruženja, pa softver koji je funkcionisao u razvojnom okruženju možda neće raditi u testnom okruženju. Nešto što je radilo na testovima, takođe, možda neće raditi u proizvodnji.

Takođe, kada imate više aplikacija koje se izvršavaju na jednom računaru, nema izolacije između njih. Jedna aplikacija može isprazniti računarske resurse druge aplikacije, što može dovesti do problema u toku izvršavanja.

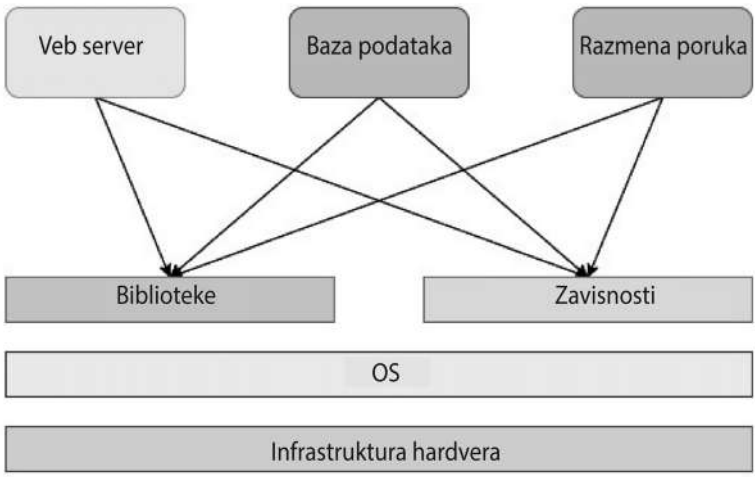
Ponovno pakovanje i konfigurisanje aplikacija, na svakom koraku implementacije, je neophodno, a to zahteva puno vremena i truda i podložno je greškama.

U softverskoj industriji, kontejneri pružaju izolaciju između aplikacija i upravljanja računarskim resursima, a to je optimalno rešenje problema.

Najveći izazov softverske industrije je obezbeđivanje izolacije aplikacija i elegantno upravljanje spoljnim zavisnostima, tako da mogu da rade na bilo kojoj platformi, bez obzira na operativni sistem ili infrastrukturu. Softver se piše na mnogim programskim jezicima i koristi različite zavisnosti i radne okvire. To dovodi do situacije koju nazivamo **matrica haosa**.

Matrica haosa

Pretpostavimo da pripremate server koji će pokretati više aplikacija, za različite timove. Sada, pretpostavite da nemate virtuelizovanu infrastrukturu i da sve morate da pokrenete na jednom fizičkom računaru, kao što je prikazano na sledećem dijagramu:



Slika 1.3 – Aplikacije na fizičkom serveru

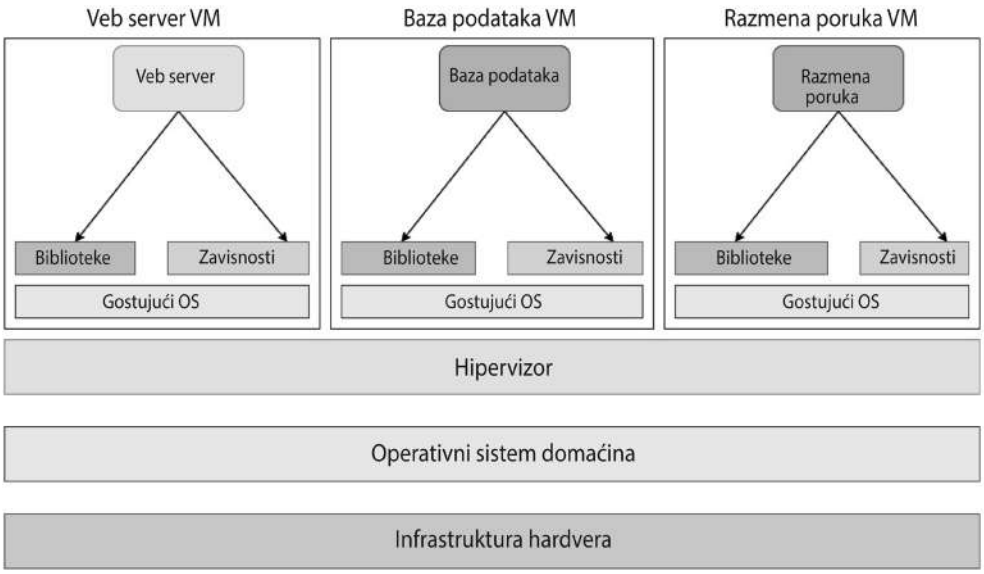
Jedna aplikacija koristi određenu verziju zavisnosti, a druga aplikacija drugu, pa morate da upravljate sa dve verzije istog softvera na istom sistemu. Kada proširite svoj sistem da bi podržao više aplikacija, bićete zaduženi za stotine zavisnosti i za različite verzije koje odgovaraju različitim aplikacijama. To će postepeno postati neodrživo, na jednom fizičkom sistemu. Ovaj scenario je u popularnoj računarskoj terminologiji poznat kao **matrica haosa**.

Iz matrice haosa potekla su brojna rešenja, ali se izdvajaju dva tehnološka doprinosa - *virtuelne mašine* i *kontejneri*.

Virtuelne mašine

Virtuelna mašina emulira operativni sistem pomoću **hipervizora**. Hipervizor može da se pokreće kao softver na fizičkom operativnom sistemu domaćina, ili kao ugrađeni softver na računaru bez operativnog sistema. Virtuelne mašine pokreću se kao virtuelni gostujući operativni sistemi na hipervizoru. Pomoću ove tehnologije moguće je podeliti veliki fizički računar na više manjih virtuelnih mašina, od kojih svaka služi određenoj aplikaciji. Ovo već dve decenije predstavlja revoluciju računarske infrastrukture i još uvek je u upotrebi. Neki od najpopularnijih hipervizora na tržištu su **VMware** i **Oracle VirtualBox**.

Sledeći dijagram prikazuje isti skup tehnologija na virtuelnim mašinama. Možete primetiti da svaka aplikacija sada sadrži poseban gostujući operativni sistem, pri čemu svaki ima svoje biblioteke i zavisnosti:



Slika 1.4 – Aplikacije na virtuelnim mašinama

Iako je ovaj pristup prihvatljiv, to je kao da za svoju robu koristite ceo brod, umesto jednog kontejnera. Virtuelne mašine zahtevaju mnogo resursa, jer za izolaciju aplikacija koriste složen gostujući operativni sistem, umesto lakših alternativa. Svakoj virtuelnoj mašini je potrebno dodeliti odvojene procesorske jedinice i memorije; dodela resursa nije optimalna, jer virtuelne mašine koriste više resursa nego što je potrebno, da bi bile prilagođene vrhunskim opterećenjima. Takođe, sporije se pokreću, a skaliranje virtuelnih mašina je složenije, jer uključuju više pokretnih delova i tehnologija. Zbog toga automatizacija horizontalnog skaliranja (upravljanje većim brojem korisnika dodavanjem više mašina u grupu resursa) pomoću virtuelnih mašina nije sasvim jednostavna. Takođe, administratori sistema moraju da upravljaju većim brojem servera, umesto bibliotekama i zavisnostima na jednom sistemu. Bolje je nego pre, ali što se računarskih resursa tiče, nije optimalno.

Kontejneri

Tu na scenu stupaju kontejneri. Kontejneri rešavaju matricu haosa bez teškog sloja gostujućeg operativnog sistema. Umesto toga, izoluju izvršno okruženje aplikacije i zavisnosti tako što ih enkapsuliraju i stvaraju apstrakciju pod nazivom kontejner. Tako imate više kontejnera koji se pokreću na jednom operativnom sistemu. Brojne aplikacije, koje se izvršavaju na kontejnerima, dele istu infrastrukturu. Kao rezultat toga, ne trošite vaše računarske resurse. Takođe, ne morate da brinete o bibliotekama i zavisnostima aplikacija, jer su izolovane – opšte korisna situacija!

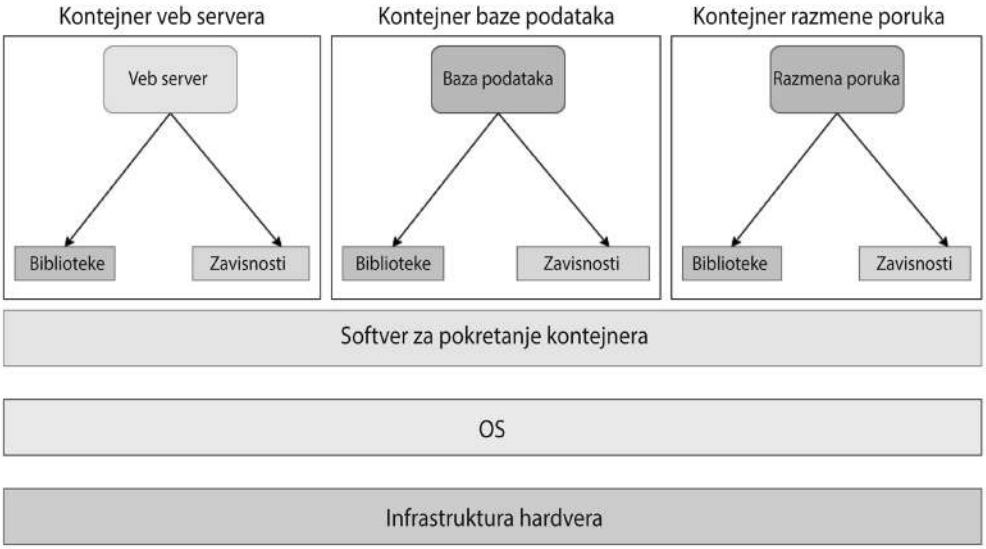
Kontejneri se pokreću pomoću softvera za pokretanje kontejnera. Iako je **Docker** najpopularniji i gotovo standardan softver za pokretanje kontejnera, na tržištu su dostupne i druge opcije, kao što su **Rkt** i **Containerd**. Sva ova softverska rešenja koriste istu funkciju Linux jezgra - *cgroups*, čija osnova potiče iz zajedničkih napora kompanija Google, IBM, OpenVZ i SGI da integrišu **OpenVZ** u glavno Linux jezgro. OpenVZ je bio rani pokušaj implementiranja funkcionalnosti koje omogućavaju virtuelna okruženja unutar Linux jezgra, bez upotrebe sloja gostujućeg operativnog sistema, što danas nazivamo kontejnerima.

Na mojoj mašini radi

Verovatno ste čuli ovu frazu mnogo puta tokom svoje karijere. Tipična je situacija gde imate nepredvidive programere razvojnog tima koji zabrinjavaju vaš tim za testiranje odgovorima poput „*Ali, radi na mom računaru*", a vaš tim za testiranje odgovara sa „*Nećemo isporučiti vaš računar klijentu*." Kontejneri koriste koncepte „*Napravi jednom, pokreni bilo gde*" i „*Pakuj jednom, implementiraj bilo gde*" i rešavaju sindrom „*Na mojoj mašini radi*". Kako kontejneri zahtevaju softver za pokretanje kontejnera, mogu se izvršavati na bilo kojoj mašini na isti način. Standardizovana postavka za aplikacije takođe znači da se posao sistem administratora svodi na brigu o softverima za pokretanje kontejnera i serverima, dok se odgovornosti aplikacije delegiraju razvojnom timu. Ovo smanjuje administrativne troškove prilikom isporuke softvera, a timovi za razvoj softvera sada mogu da deluju bez mnogo spoljnih zavisnosti - zaista velika prednost! Pogledajmo kako su kontejneri dizajnirani da to postignu.

Arhitektura kontejnera

U većini slučajeva, možete da zamislite kontejnere kao male virtuelne mašine - bar tako izgleda. Ali, u stvarnosti, to su samo računarski programi koji se izvršavaju unutar operativnog sistema. Sledeći dijagram prikazuje kako izgleda skup aplikacija unutar kontejnera:



Slika 1.5 – Aplikacije na kontejnerima

Kao što vidite, računarska infrastruktura nalazi se na dnu i formira osnovu, zatim sledi operativni sistem domaćina i softver za pokretanje kontejnera (u ovom slučaju, Docker) koji se izvršava iznad njega. Zatim imamo više kontejnerizovanih aplikacija koje koriste softver za pokretanje kontejnera i izvršavaju se kao odvojeni procesi preko operativnog sistema domaćina pomoću *prostora imena* i *kontrolnih grupa*.

Kao što ste primetili, nemamo sloj gostujućeg operativnog sistema unutar kontejnera, što je nešto što imamo kod virtuelnih mašina. Svaki kontejner je *softverski program* koji se izvršava u korisničkom prostoru jezgra i deli isti operativni sistem, odgovarajuće izvršno okruženje i druge zavisnosti sa samo neophodnim bibliotekama i zavisnostima unutar kontejnera. Kontejneri ne nasleđuju promenljive okruženja operativnog sistema. Morate ih posebno postaviti za svaki kontejner.

Kontejneri kopiraju sistem datoteka, i iako su prisutni na disku, izolovani su od drugih kontejnera. Ovo omogućava kontejnerima da pokreću aplikacije u bezbednom okruženju. Odvojen sistem datoteka kontejnera znači da kontejneri ne moraju da komuniciraju sa sistemom datoteka operativnog sistema u oba smera, što rezultira bržim izvršavanjem u poređenju s virtuelnim mašinama.

Kontejneri su dizajnirani da koriste Linux *prostore imena* za izolaciju i *kontrolne grupe* da bi ograničili potrošnju procesorskih jedinica, memorije i ulaza/izlaza diska.

Ovo znači da ako izlistate procese operativnog sistema, videćete proces kontejnera koji se izvršava zajedno s drugim procesima, kao što je prikazano na sledećem snimku ekrana:

Slika 1.6 – Procesi operativnog sistema

Međutim, kada izlistate procese kontejnera, videćete samo proces kontejnera, kao što sledi:

```
$ docker exec -it mynginx1 bash
root@4ee264d964f8:/# pstree
nginx---nginx
```

Ovo je način na koji prostori imena pružaju određeni stepen izolacije između kontejnera.

Kontrolne grupe imaju ulogu da ograniče količinu računarskih resursa koju grupa procesa može da koristi. Na primer, ako dodate procese u kontrolnu grupu, možete da ograničite procesorske jedinice, memoriju i ulaz/izlaz diska koje procesi mogu da koriste. Osim toga, možete meriti i pratiti upotrebu resursa i zaustaviti grupu procesa kada aplikacija skrene s puta. Sve ove funkcije čine osnovu tehnologije kontejnera, o kojoj će biti više reči kasnije u ovoj knjizi.

Kada imamo nezavisno pokrenute kontejnere, moramo razumeti i njihovu interakciju. Zbog toga je sledeći deo posvećen umrežavanju kontejnera.

Umrežavanje kontejnera

Kontejneri su odvojene mrežne jedinice unutar operativnog sistema. Docker izvršna okruženja koriste mrežne upravljačke programe da bi definisali mrežu između kontejnera, i oni su softverski definisane mreže. **Umrežavanje kontejnera** funkcioniše tako što se koristi softver za manipulaciju *IP tabelama domaćina*, za povezivanje sa spoljnim mrežnim interfejsima, kreiranje tunelskih mreža i obavljanje drugih aktivnosti da bi omogućio povezivanje sa kontejnerima i ka njima.

Iako postoje različite vrste mrežnih konfiguracija koje je moguće implementirati sa kontejnerima, dobro je znati neke koje se najčešće koriste. Ne brinite zbog previše detalja - razumećete ih tokom obavljanja praktičnih vežbi, kasnije u ovoj knjizi, a nije neophodno da znate sve to da biste pratili tekst. Za sada, pogledajmo različite vrste mreža kontejnera koje je moguće definisati:

- **None**: ovo je potpuno izolovana mreža i vaši kontejneri ne mogu komunicirati sa spoljnim svetom. Dodeljena im je interfejs povratna petlja i ne mogu se povezati sa spoljnim mrežnim interfejsom. Možete koristiti ovu mrežu za testiranje vaših kontejnera, pripremu kontejnera za buduću upotrebu ili pokretanje kontejnera koji ne zahteva nikakvu spoljnu vezu, poput grupne obrade.
- **Bridge**: bridge mreža je podrazumevani tip mreže u većini softvera za pokretanje kontejnera, uključujući Docker, i koristi docker0 interfejs za podrazumevane kontejnere. Bridge mreža manipuliše IP tabelama da bi omogućila **prevođenje mrežnih adresa (NAT)** između kontejnera i mreže domaćina, što omogućava spoljnu mrežnu povezanost. Takođe, ne dovodi do konflikta porta i omogućava izolaciju mreže između kontejnera koji se izvršavaju na jednom domaćinu. Zbog toga možete pokretati više aplikacija koje koriste isti kontejnerski port unutar jednog domaćina. Bridge mreža omogućava kontejnerima unutar jednog domaćina da komuniciraju pomoću IP adresa kontejnera. Međutim, ne dozvoljava komunikaciju sa kontejnerima koji se izvršavaju na drugom domaćinu. Zbog toga ne bi trebalo koristiti bridge mrežu za klasterizovanu konfiguraciju (korišćenje više servera istovremeno za pokretanje vaših kontejnera).
- **Host**: host mreža koristi mrežni prostor imena domaćina za sve kontejnere. Slično je pokretanju više aplikacija unutar vašeg domaćina. Iako je host mreža jednostavna za implementaciju, vizuelizaciju i otklanjanje problema, podložna je problemima s konfliktima porta. Dok kontejneri koriste host mrežu za sve komunikacije, oni nemaju mogućnost manipulacije interfejsima host mreže, osim ako se izvršavaju u privilegovanom režimu. Host mreža ne koristi NAT, pa komunicira brzo, direktno na nivou hardvera. Zbog toga možete da koristite host mrežu za optimizovanje performansi. Međutim, budući da nema izolacije mreže između kontejnera, zbog aspekta bezbednosti i upravljanja, u većini slučajeva trebalo bi izbegavati host mrežu.
- **Underlay**: direktno izlaže interfejse host mreže kontejnerima. To znači da možete pokretati svoje kontejnere direktno na mrežnim interfejsima umesto da koristite bridge mrežu. Postoji nekoliko underlay mreža, od kojih su najznačajnija MACvlan i IPvlan. MACvlan omogućava dodeljivanje MAC adrese svakom kontejneru, tako da vaš kontejner izgleda kao fizički uređaj. Ovo je korisno za migraciju postojećeg sistema na kontejnere, posebno kada vaša aplikacija treba da se izvršava na

fizičkom računaru. MACvlan takođe pruža potpunu izolaciju vaše host mreže, pa možete koristiti ovaj režim za stroge bezbednosne zahteve. MACvlan ima ograničenja, jer ne može da radi sa mrežnim prekidačima koji imaju bezbednosnu politiku koja ne dozvoljava lažiranje MAC adresa. Takođe, važno je napomenuti da određene mrežne kartice, poput onih koje proizvodi Broadcom, imaju ograničen broj MAC adresa koje mogu da podrže po interfejsu, a to je 512 MAC adresa.

- **Overlay**: nemojte mešati overlay i underlay - iako izgledaju kao antonimi, zapravo nisu. Overlay mreže omogućavaju komunikaciju između kontejnera na različitim računarima domaćinima putem mrežnog tunela. Dakle, iz perspektive kontejnera, čini se da komuniciraju sa kontejnerima na jednom računaru domaćinu, čak i kada se nalaze na drugom mestu. Ovo prevazilazi ograničenja bridge mreže i posebno je korisno za konfiguraciju klastera, posebno kada se koristi orkestrator kontejnera, kao što su Kubernetes ili Docker Swarm. Neke popularne overlay tehnologije koje koriste softveri za pokretanje kontejnera i orkestratori su **flannel**, **calico** i **VXLAN**.

Pre nego što zaronimo u tehničke detalje različitih vrsta mreža, treba razjasniti nijanse mrežnog povezivanja kontejnera. Na tu temu, posebno ćemo govoriti o orkestratoru Docker.

Svakom Docker kontejneru koji se izvršava na domaćinu dodeljuje se jedinstvena IP adresa. Ako izvršite `exec` (otvorite shell sesiju) u kontejneru i pokrenete `hostname -I`, trebalo bi da vidite nešto slično sledećem:

```
$ docker exec -it mynginx1 bash
root@4ee264d964f8:/# hostname -I
172.17.0.2
```

Ovo omogućava različitim kontejnerima da komuniciraju međusobno putem jednostavne TCP/IP adrese. Docker pozadinski servis služi kao DHCP server za svaki kontejner. Tako možete da definišete virtuelne mreže za grupu kontejnera da biste izolovali mrežu, ako to želite. Takođe, možete da povežete kontejner sa više mreža, da biste ga podelili za dve različite uloge.

Docker svakom kontejneru dodeljuje jedinstveno ime računara, koje podrazumevano odgovara identifikacionom broju kontejnera. Međutim, to se može lako promeniti, pod uslovom da koristite jedinstvena imena računara u određenoj mreži. Dakle, ako izvršite exec u kontejneru i pokrenete `hostname`, trebalo bi da vidite identifikacioni broj kontejnera kao jedinstveno ime računara, kao što sledi:

```
$ docker exec -it mynginx1 bash
root@4ee264d964f8:/# hostname
4ee264d964f8
```

Ovo omogućava kontejnerima da deluju kao odvojene mrežne jedinice, a ne kao jednostavni softverski programi, a lako ih možete zamisliti kao male virtuelne mašine.

Kontejneri takođe nasleđuju DNS podešavanja operativnog sistema domaćina, tako da ne morate previše brinuti ako želite da svi kontejneri dele ista DNS podešavanja. Ako želite da definišete posebnu DNS konfiguraciju za vaše kontejnere, to možete lako učiniti prosleđivanjem nekoliko opcija. Docker kontejneri ne nasleđuju unose u `/etc/hosts` datoteci, pa ih morate definisati navođenjem prilikom kreiranja kontejnera, komandom `docker run`.

Ako vaši kontejneri zahtevaju posrednički server, morate to postaviti ili u promenljivama okruženja Docker kontejnera ili dodavanjem podrazumevanog posrednika u ~/.docker/config.json datoteku.

Do sada smo govorili o kontejnerima i šta su oni. Sada ćemo razmotriti kako kontejneri menjaju DevOps kulturu i zašto je bilo važno naglasiti to na početku.

Kontejneri i savremene DevOps prakse

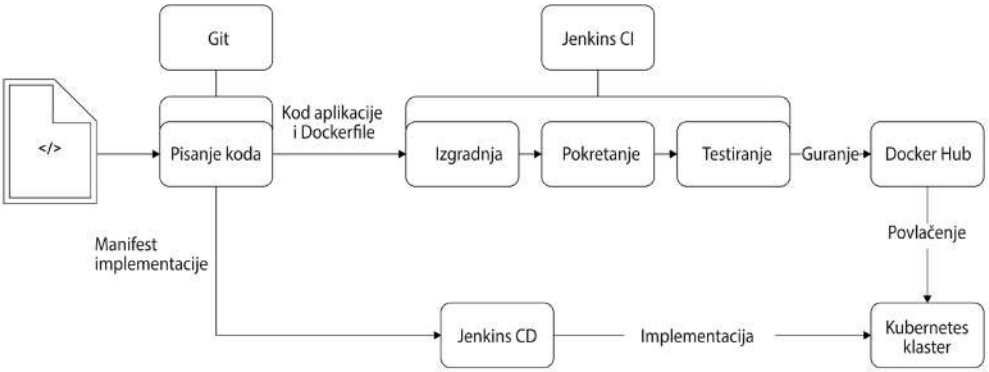
Kontejneri i savremene DevOps prakse su veoma komplementarni i transformisali su način na koji pristupamo razvoju i implementaciji softvera.

Kontejneri imaju sjajnu sintezu sa savremenim DevOps praksama, jer isporučuju potrebnu enkapsulaciju infrastrukture, prenosivost, skalabilnost i agilnost, što omogućava brzu i efikasnu isporuku softvera. Sa savremenim DevOps praksama, kao što su CI/CD, IaC i mikroservisi, kontejneri čine snažnu osnovu organizacijama za brz izlazak na tržište, poboljšan kvalitet softvera i unapređenu operativnu efikasnost.

Kontejneri od samog početka prate DevOps prakse. Ako pogledate tipičan tok izgradnje i implementacije kontejnera, evo šta ćete dobiti:

1. Prvo napišite vašu aplikaciju na bilo kom programskom jeziku.
2. Zatim, kreirajte **Dockerfile** koji sadrži niz koraka za instaliranje zavisnosti aplikacije i konfiguraciju okruženja, da bi se aplikacija pokrenula.
3. Nakon toga, koristite Dockerfile da kreirate slike kontejnera, tako što ćete uraditi sledeće:
   a) Izgraditi sliku kontejnera.
   b) Pokrenuti sliku kontejnera.
   c) Testirati aplikaciju koja se izvršava na kontejneru.
4. Zatim, otpremite sliku u registar kontejnera, kao što je **DockerHub**.
5. Na kraju, kreirajte kontejnere iz slika kontejnera i pokrenite ih u klasteru.

Možete lepo ugraditi ove korake u sledeći primer CI/CD protočne strukture:



Slika 1.7 – Primer CI/CD protočne strukture za kontejnere

To znači da su aplikacija i sve zavisnosti za izvršavanje definisani u kodu. Pratite upravljanje konfiguracijom od samog početka, što omogućava programerima razvojnog tima da tretiraju kontejnere kao prolazna radna opterećenja (prolazna radna opterećenja su privremena opterećenja koja je moguće odbaciti, i ako jedno nestane, moguće je pokrenuti drugo, bez ikakvog uticaja na funkcionalnost). Moguće ih je zameniti ako se ponašaju loše – elegantno rešenje koje nedostaje virtuelnim mašinama.

Kontejneri se veoma dobro uklapaju u moderne CI/CD prakse, jer sada imate standardan način izgradnje i implementacije aplikacija, bez obzira na jezik na kom pišete kod. Ne morate da upravljate skupim softverom za izgradnju i implementaciju, jer sa kontejnerima dobijate sve što vam je potrebno, odmah.

Kontejneri retko kad rade samostalno, i standardna praksa u industriji je da ih uključite u orkestrator kontejnera kao što je **Kubernetes**, ili da koristite platformu **Kontejner kao usluga (CaaS)**, kao što su **AWS ECS** i **EKS**, **Google Cloud Run** i **Kubernetes Engine**, **Azure ACS** i **AKS**, **Oracle OCI** i **OKE**, i drugi. Popularne platforme **Funkcija kao usluga (FaaS)** poput **AWS Lambda**, **Google Functions**, **Azure Functions** i **Oracle Functions** takođe pokreću kontejnere u pozadini. Dakle, iako je osnovni mehanizam apstrahovan za vas, možda već koristite kontejnere, nesvesno.

S obzirom na to da kontejneri nisu glomazni, možete da ugradite manje delove aplikacija u kontejnere, da biste mogli nezavisno da upravljate njima. Kombinujte to sa orkestratorom kontejnera kao što je Kubernetes, i dobijate distribuiranu arhitekturu mikroservisa koja se lako pokreće. Ove manje delove možete skalirati, automatski popravljati i nezavisno objavljivati, što znači da ih možete brže i pouzdanije staviti u proizvodnju.

Takođe, možete da uključite **mrežu usluga** (komponente infrastrukture koje vam omogućavaju da otkrijete, nabrojite, upravljate i omogućite komunikaciju između više komponenata (usluga) vaše aplikacije mikroservisa) kao što je **Istio**, i dobićete napredne operativne funkcije, kao što su upravljanje saobraćajem, bezbednost i posmatranje sa lakoćom. Tada možete da radite zanimljive stvari, ka što su **plavo/zelene implementacije** i **A/B testiranje**, operativni testovi u proizvodnji sa **preslikavanjem saobraćaja**, **usmeravanje na osnovu geolokacije** i još mnogo toga.

Zahvaljujući tome, i velike i male kompanije prihvataju kontejnere brže nego ikada, a oblast se eksponencijalno širi. Prema podacima na veb sajtu businesswire.com, tržište kontejnerskih aplikacija ima godišnji rast od 31% i dostići će 6,9 milijardi dolara do 2025. godine. Eksponencijalni rast od 30,3% godišnje u oblaku, za koji se očekuje da će premašiti 2,4 milijarde dolara do 2025. godine, takođe je doprineo tome.

Zbog toga, moderni DevOps inženjeri moraju da poznaju kontejnere i relevantne tehnologije, da bi efikasno dostavili i isporučili kontejnerizovane aplikacije. To ne znači da su virtuelne mašine nepotrebne, i ne možemo potpuno zanemariti ulogu IaaS rešenja na tržištu, tako da ćemo opisati i upravljanje konfiguracijom pomoću alata **Ansible**, u narednim poglavljima. Zbog pojave oblaka, IaC je nedavno dobio veliki zamah, pa ćemo opisati i **Terraform** kao IaC alat.

Migracija sa virtuelnih mašina na kontejnere

Zbog činjenice da se tehnološko tržište okreće ka kontejnerima, DevOps inženjeri imaju ključni zadatak - *migraciju aplikacija sa virtuelnih mašina na kontejnere*. Dakle, to je jedan od ključnih zadataka DevOps inženjera i predstavlja jednu od najvažnijih aktivnosti koje obavljamo.

Iako je u teoriji kontejnerizacija aplikacije jednostavna, kao pisanje nekoliko koraka, u praksi to može biti komplikovanije, posebno ako ne koristite upravljanje konfiguracijom za postavljanje vaših virtuelnih mašina. Virtuelne mašine koje se danas koriste kreirane su napornim radom sistemskih administratora, koji su poboljšavali servere deo po deo, što otežava praćenje ispravki koje su napravili do sada.

Pošto kontejneri od samog početka slede principe upravljanja konfiguracijom, postupak nije tako jednostavan, poput konvertovanja slike virtuelne mašine u Docker kontejner. Migracija zastarele aplikacije koja se izvršava na virtuelnim mašinama zahteva brojne korake. Hajde da ih detaljnije pogledamo.

Otkrivanje

Prvo, počinjemo od faze otkrivanja:

- Razumevanje različitih delova vaših aplikacija
- Procena koji delovi zastarelih aplikacija se mogu kontejnerizovati i da li je to tehnički moguće
- Definisanje opsega migracije i saglasnost o jasnim ciljevima i prednostima migracije, sa vremenskim okvirom

Procena zahteva aplikacije

Posle faze otkrivanja, potrebno je obaviti procenu zahteva aplikacije:

- Proceniti da li je dobro da se aplikacija razdvoji na manje delove. Ako jeste, koji bi to delovi bili i kakva bi bila njihova interakcija?
- Proceniti koje aspekte arhitekture, performansi i bezbednosti aplikacije treba uzeti u obzir, i razmisliti o ekvivalentima u svetu kontejnera.
- Razumeti relevantne rizike i odlučiti o pristupima za ublažavanje tih rizika.
- Razumeti princip migracije i odabrati pristup, kao što je odluka o tome koji deo aplikacije treba kontejnerizovati na početku. Uvek počnite od aplikacije koja ima najmanje spoljnih zavisnosti.

Dizajn infrastrukture kontejnera

Dizajn infrastrukture kontejnera podrazumeva kreiranje robusnog i skalabilnog okruženja da bi se podržala implementacija i upravljanje kontejnerizovanim aplikacijama.

Dizajniranje infrastrukture kontejnera podrazumeva razmatranje faktora kao što su skalabilnost, umrežavanje, skladištenje, bezbednost, automatizacija i praćenje. Ključno je uskladiti dizajn infrastrukture sa specifičnim zahtevima i ciljevima kontejnerizovanih aplikacija i držati se najbolje prakse za efikasnu i pouzdanu implementaciju i upravljanje kontejnerima.

Kada smo procenili sve zahteve, arhitekturu i druge aspekte, možemo preći na dizajn infrastrukture kontejnera:

- Svest o trenutnom i budućem obimu operacija je ključno za donošenje ove odluke. Postoji mnogo opcija, u zavisnosti od složenosti vaše aplikacije. Prava pitanja su: koliko kontejnera je potrebno da pokrenemo na platformi? Kakve zavisnosti imaju ti kontejneri? Koliko često ćemo implementirati promene na komponentama? Koju količinu saobraćaja aplikacija može da primi? Kakav je obrazac saobraćaja na aplikaciji?
- Na osnovu odgovora na prethodna pitanja, potrebno je znati na kakvoj infrastrukturi ćete pokretati aplikaciju. Da li će to biti lokalno ili u oblaku, i da li ćete koristiti upravljani Kubernetes klaster ili ćete ga sami skladištiti i upravljati njime? Takođe, za jednostavnije aplikacije, treba razmotriti CaaS kao opciju.
- Kako ćete pratiti kontejnere i kako ćete upravljati kontejnerima? Da li će biti potrebno da instalirate specijalizovane alate? Da li će biti potrebna integracija sa postojećim alatima za praćenje? Potrebno je steći svest o izvodljivosti i doneti odgovarajuću dizajnersku odluku.
- Kako ćete obezbediti vaše kontejnere? Da li postoje pravni i usaglašeni zahtevi u vezi sa bezbednošću? Da li odabrano rešenje odgovara tim zahtevima?

Kontejnerizacija aplikacije

Kontejnerizacija aplikacije obuhvata pakovanje aplikacije i njenih zavisnosti u sliku kontejnera, koja se može dosledno implementirati i pokretati u različitim okruženjima.

Kontejnerizacija aplikacije je korisna, zbog poboljšane prenosivosti, skalabilnosti i reproduktivnosti. Pojednostavljuje proces implementacije i omogućava dosledno ponašanje aplikacije u različitim okruženjima.

Kada smo razmotrili sve aspekte dizajna, možemo započeti kontejnerizaciju aplikacije:

- Proučavamo aplikaciju i kreiramo Dockerfile datoteku koja sadrži korake za kreiranje kontejnera u skladu sa potrebama i zahtevima aplikacije. To zahteva konsultacije i procene, naročito ako alati za upravljanje konfiguracijom, kao Ansible, ne grade vašu aplikaciju pokretanjem na virtuelnoj mašini. Određivanje kako je aplikacija bila instalirana može biti dugotrajan proces, i zahteva pažljivo istraživanje, a zatim precizno dokumentovanje koraka.
- Ako planirate da razdvojite vašu aplikaciju na manje delove, možda ćete morati da je izgradite ispočetka.
- Morate da izaberete paket testova koji funkcioniše na vašoj paralelnoj aplikaciji pokrenutoj na virtuelnoj mašini i vremenom ga unapređivati.

Testiranje

Testiranje kontejnerizovanih aplikacija je važno za funkcionalnost, performanse i kompatibilnost.

Implementacijom sveobuhvatne strategije testiranja ćete obezbediti pouzdanost, performanse i bezbednost vaše kontejnerizovane aplikacije. Testiranje na različitim nivoima, integracija automatizacije i pažljivo praćenje ponašanja aplikacije pomoći će vam da identifikujete i rešite probleme u ranoj fazi razvojnog ciklusa, što dovodi do robusnije i pouzdanije kontejnerizovane aplikacije.

Kada smo kontejnerizovali aplikaciju, sledeći korak je testiranje:

- Da biste utvrdili da vaša kontejnerizovana aplikacija funkcioniše upravo kao i ona na virtuelnoj mašini, potrebno je da obavite obimno testiranje, da ne biste propustili nijedan detalj ili deo koji je trebalo uzeti u obzir. Pokrenite postojeći paket testova ili onaj koji ste kreirali za kontejner.
- Pokretanje postojećeg paketa testova može biti pravi pristup, ali takođe treba uzeti u obzir nefunkcionalne aspekte softvera. Testiranje performansi originalne aplikacije je dobar početak, i potrebno je razumeti uticaj koji rešenje kontejnerizovanja aplikacije donosi. Takođe je potrebno podešavati aplikaciju da bi se uklopila u performanse.
- Takođe treba razmotriti značaj bezbednosti i kako je možete realizovati u svet kontejnera. Testiranje prodora će otkriti mnoge bezbednosne propuste kojih možda niste bili svesni.

Raspoređivanje i pokretanje

Implementacija i pokretanje kontejnerizovane aplikacije obuhvata implementaciju kontejnerskih slika u ciljno okruženje i isporuku aplikacije za upotrebu.

Kada smo testirali naše kontejnere i dovoljno smo sigurni, možemo pustiti aplikaciju u proizvodnju:

- Konačno, puštamo aplikaciju u proizvodnju i tamo vidimo da li su potrebne dodatne promene. Zatim se vraćamo procesu istraživanja sve dok ne usavršimo aplikaciju.
- Morate definisati i razviti automatizovanu proceduru za izvršenje i CI/CD protočnu strukturu, da biste smanjili vreme ciklusa i brzo rešavali probleme.
- A/B testiranje sa kontejnerskim aplikacijama koje se pokreću paralelno može vam pomoći da uočite eventualne probleme, pre nego što preusmerite sav saobraćaj na novo rešenje.

Sledeći dijagram rezimira ove korake, i kao što možete videti, proces je cikličan. To znači da ćete možda morati da ponovite ove korake, u zavisnosti od ponašanja kontejnera u proizvodnji:

Slika 1.8 – Migracija sa virtuelnih mašina na kontejnere

Hajde sada da vidimo šta je potrebno da uradimo kako bismo obezbedili da migracija sa virtuelnih mašina na kontejnere bude što bezbolnija i postigli najbolji mogući rezultat.

Koje bi aplikacije trebalo da budu u kontejnerima?

U procesu prelaska sa virtuelnih mašina na kontejnere, prvo treba da procenite šta može, a šta ne može biti u kontejnerima. Uopšteno govoreći, postoje dve vrste opterećenja aplikacija - **bez stanja** i **sa stanjem**. Dok opterećenja bez stanja ne čuvaju stanje i predstavljaju moćne računarske resurse, kao što su API interfejsi i funkcije, aplikacije sa stanjem, poput baza podataka, zahtevaju trajno skladište da bi funkcionisale.

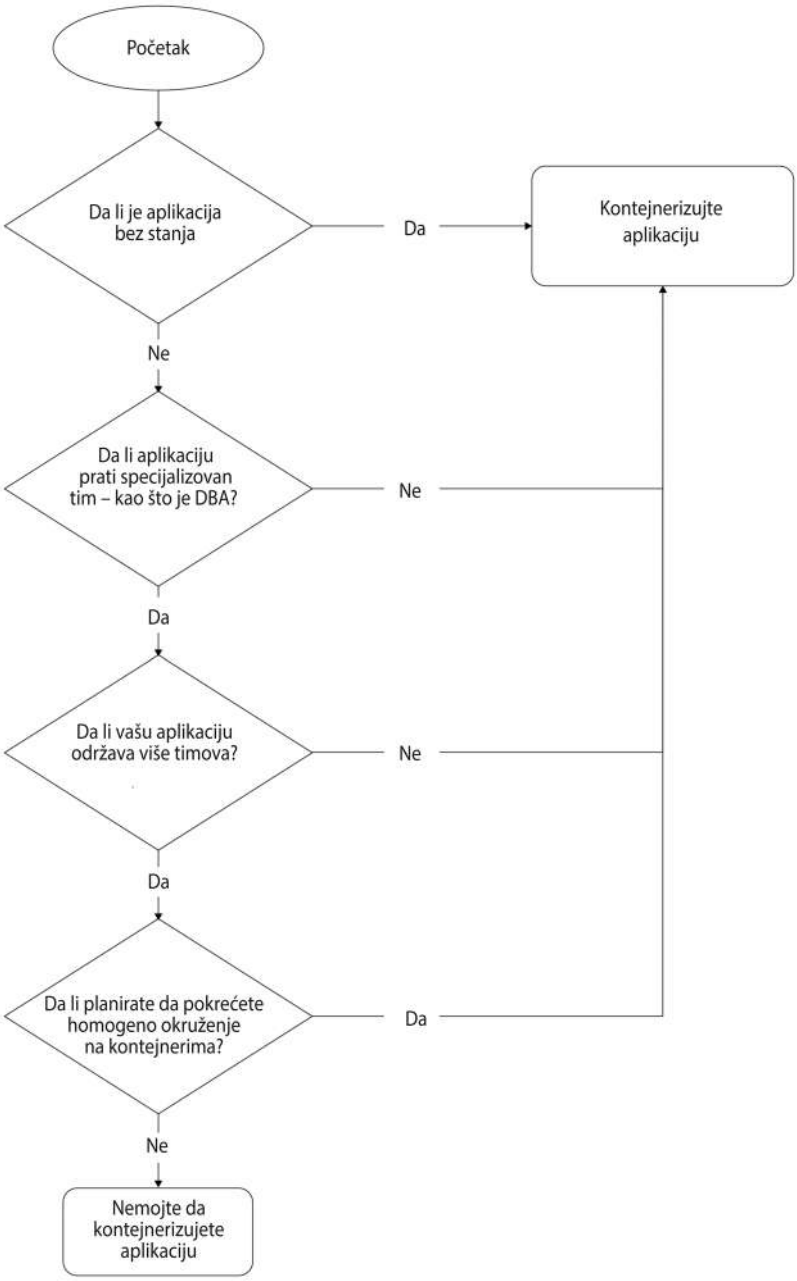
Iako je moguće kontejnerizovati bilo koju aplikaciju koja može da se izvršava na virtuelnoj mašini sa Linux operativnim sistemom, aplikacije bez stanja su prvi i najlakši cilj na koji treba obratiti pažnju. Relativno je lako kontejnerizovati ova opterećenja, jer nemaju zavisnosti od skladišta. Što više zavisnosti od skladišta imate, to je složenija vaša aplikacija u kontejnerima.

Drugo, takođe treba proceniti oblik infrastrukture na kojoj želite da skladištite svoje aplikacije. Na primer, ako planirate da pokrenete čitavu tehničku infrastrukturu na Kubernetes platformi, želećete da izbegnete heterogeno okruženje, kad god je to moguće. U toj situaciji, takođe ćete želeli da kontejnerizujete aplikacije sa stanjem. Sa veb uslugama i slojem posrednika, većina aplikacija se oslanja na neki oblik stanja da bi pravilno funkcionisale. Dakle, u svakom slučaju, morate da upravljate skladištem.

Iako ovo može da otvori Pandorinu kutiju, u industriji ne postoji standardan sporazum u vezi sa kontejnerizacijom baza podataka. Dok neki stručnjaci skeptično gledaju na njegovu upotrebu u proizvodnji, značajan broj njih ne vidi nikakve probleme. Glavni razlog je manjak podataka za ili protiv upotrebe kontejnerizovane baze podataka u proizvodnji.

Moj predlog je da budete oprezni kada je reč o bazama podataka. Iako nemam ništa protiv kontejnerizacije baza podataka, morate uzeti u obzir različite faktore, kao što su adekvatno dodeljivanje memorije, procesorskih jedinica, diska i svake zavisnosti koju imate na virtuelnim mašinama. Takođe, trebalo bi da istražite ponašanje tima. Ako imate tim administratora koji upravlja bazom podataka u proizvodnji (DBA tim), možda će im smetati još jedan sloj kompleksnosti - kontejneri.

Možemo da rezimiramo ove korake procene na visokom nivou pomoću sledećeg dijagrama toka:



Slika 1.9 – Procena migracije virtuelne mašine u kontejner

Ovaj dijagram toka obuhvata najčešće faktore koji se uzimaju u obzir tokom procene. Takođe treba uzeti u obzir i situacije koje su jedinstvene za vašu organizaciju. Dakle, dobra je ideja da i njih uzmete u obzir pre donošenja bilo kakvih odluka.

Hajde da pogledamo neke primene koje su pogodne za kontejnerizaciju kako bismo stekli pravi uvid. Sledeći tipovi aplikacija se često implementiraju pomoću kontejnera:

- **Arhitektura mikroservisa**: aplikacije koje prate arhitekturu mikroservisa, gde je funkcionalnost podeljena na male, nezavisne usluge, pogodne su za kontejnerizaciju. Svaki mikroservis može biti upakovan kao zaseban kontejner, što omogućava lakši razvoj, implementaciju, skaliranje i upravljanje pojedinačnim uslugama.
- **Veb aplikacije**: veb aplikacije, uključujući aplikacije korisničkog interfejsa, serverske API interfejse i veb usluge, mogu biti kontejnerizovane. Kontejneri pružaju dosledno izvršno okruženje, što olakšava pakovanje i implementaciju veb aplikacija u različitim okruženjima, kao što su razvoj, testiranje i proizvodnja.
- **Aplikacije sa stanjem**: kontejneri takođe mogu da služe za pokretanje aplikacija sa stanjem, koje zahtevaju trajno skladištenje podataka. Iskorišćavanjem karakteristika platformi za orkestraciju kontejnera, kao što su trajni kapaciteti ili skupovi sa stanjem, aplikacije sa stanjem, kao što su baze podataka, sistemi za upravljanje sadržajem ili serverski sistemi mogu biti kontejnerizovane i efikasno upravljane.
- **Grupna obrada ili zakazani poslovi**: aplikacije koje obavljaju zadatke grupne obrade ili zakazane poslove, poput obrade podataka, periodičnog pravljenja rezervnih kopija ili generisanja izveštaja, mogu imati koristi od kontejnerizacije. Kontejneri pružaju kontrolisano i izolovano okruženje za pokretanje ovih poslova, i obezbeđuju dosledno izvršavanje i mogućnost umnožavanja.
- **CI/CD alati**: kontejnerizacija CI/CD alata kao što su Jenkins, GitLab CI/CD ili CircleCI omogućava dosledne i reproduktivne tokove izgradnje, testiranja i implementacije. Kontejneri olakšavaju upravljanje zavisnostima, izolovanje okruženja za izgradnju i omogućavaju brzu implementaciju CI/CD infrastrukture.
- **Okruženja za razvoj i testiranje**: kontejneri su korisni za kreiranje izolovanih i reproduktivnih okruženja za razvoj i testiranje. Razvojni timovi mogu da koriste kontejnere da upakuju svoje aplikacije zajedno sa potrebnim zavisnostima, bibliotekama i razvojnim alatima. To omogućava dosledna iskustva razvoja i testiranja na različitim mašinama i od strane raznih članova tima.
- **Aplikacije Interneta stvari (IoT)**: Kontejneri mogu da služe za implementaciju i upravljanje aplikacijama u IoT situacijama. Pružaju lagana i prenosiva izvršna okruženja za aplikacije Interneta stvari i omogućavaju jednostavnu implementaciju na uređajima na ivici mreže, na pristupnim tačkama ili na infrastrukturi u oblaku.
- **Aplikacije za mašinsko učenje i analitiku podataka**: kontejnerizacija se sve više koristi za implementaciju modela mašinskog učenja i aplikacija za analizu podataka. Kontejneri enkapsuliraju potrebne zavisnosti, biblioteke i izvršna okruženja, što omogućava bezbolnu implementaciju i skaliranje aplikacija koje zahtevaju velike količine podataka.

Važno je napomenuti da nisu sve aplikacije idealni kandidati za kontejnerizaciju. Aplikacije sa složenim grafičkim interfejsima, zastarele monolitne arhitekture čvrsto povezane sa osnovnom infrastrukturom ili aplikacije koje zahtevaju direktni pristup hardveru možda nisu pogodne za kontejnerizaciju. U takvim slučajevima su prikladnije virtuelne mašine ili drugi pristupi implementaciji.

Razdvajanje aplikacija na manje komponente

Najviše koristi od kontejnera dobijate ako delove vaše aplikacije pokrećete nezavisno od drugih.

Ovaj pristup ima brojne prednosti, kao što sledi:

- Možete češće da objavljujete vašu aplikaciju, jer sada možete da promenite deo vaše aplikacije bez uticaja na nešto drugo; vaše implementacije će, takođe, biti brže.
- Delovi vaše aplikacije mogu se skalirati nezavisno jedni od drugih. Na primer, ako imate aplikaciju za kupovinu, a vaš modul za porudžbine je preopterećen, može se skalirati više od modula za recenzije, koji možda ima manje posla. Sa monolitnim infrastrukturom, cela vaša aplikacija bi se skalirala sa saobraćajem, što nije najoptimalniji pristup, u smislu potrošnje resursa.
- Ono što utiče na jedan deo aplikacije ne ugrožava ceo sistem. Na primer, korisnici i dalje mogu da dodaju stavke u svoju korpu i da izvršavaju porudžbine čak i kad je modul za recenzije nedostupan.

Međutim, takođe ne bi trebalo razbijati aplikaciju na sitne komponente. To će dovesti do značajnih administrativnih opterećenja, jer nećete moći da razlikujete jedne od drugih. Na primeru veb prodavnice, potreban je kontejner za *porudžbine*, kontejner za *recenzije*, kontejner za *korpu za kupovinu* i kontejner za *katalog*. Međutim, nije potreban kontejner za *kreiranje porudžbine*, *brisanje porudžbine* i *ažuriranje porudžbine*. To bi bilo previše. Razdvajanje aplikacije na logičke komponente koje odgovaraju vašem poslovanju je pravi put.

Da li bi trebalo razdvojiti aplikaciju na manje delove, kao prvi korak? Pa, to zavisi. Većina ljudi će želeti da dobije **povrat ulaganja (ROI)** za svoj rad na kontejnerizaciji. Recimo da izvršite premeštanje sa virtuelnih mašina na kontejnere, iako imate vrlo malo promenljivih i možete brzo da pređete na kontejnere. U tom slučaju, nećete imati nikakvu korist - posebno ako vaša aplikacija ima ogromnu infrastrukturu. Umesto toga, dodali biste teret aplikaciji, zbog sloja kontejnera. Dakle, preuređivanje vaše aplikacije tako da se uklopi u kontejnerski pejzaž ključno je za napredak.

Da li smo stigli?

Dakle, možda se pitate, da li smo već stigli? Ne baš! Virtuelne mašine će još dugo biti prisutne. Postoji dobar razlog za to, jer dok kontejneri rešavaju većinu problema, nije moguće kontejnerizovati baš sve. Mnogi zastareli sistemi funkcionišu na virtuelnim mašinama koje nije moguće migrirati na kontejnere.

Sa pojavom oblaka, *virtuelizovana infrastruktura* formira njegovu osnovu, a virtuelne mašine su u njegovoj srži. Većina kontejnera se pokreće na virtuelnim mašinama unutar oblaka, i iako možda pokrećete kontejnere u klasteru čvorova, ti čvorovi su i dalje virtuelne mašine.

Međutim, najbolja stvar u eri kontejnera je to što se virtuelne mašine smatraju delom standardne postavke. Instalirate softver za pokretanje kontejnera na svoje virtuelne mašine i ne morate da ih razlikujete. Možete da pokrenete vaše aplikacije unutar kontejnera, na bilo kojoj virtuelnoj mašini. Sa orkestratorom kontejnera, kao što je *Kubernetes*, takođe imate korist od toga što orkestrator odlučuje gde će se pokrenuti kontejneri, uzimajući u obzir različite faktore - dostupnost resursa pre svega.

Ova knjiga će istražiti različite aspekte modernih DevOps praksi, uključujući upravljanje infrastrukturom u oblaku, virtuelnim mašinama i kontejnerima. Iako ćemo se uglavnom fokusirati na kontejnere, takođe ćemo istražiti upravljanje konfiguracijama pomoću alata Ansible i naučićemo da postavimo infrastrukturu, pomoću alata Terraform.

Takođe ćemo istražiti moderne CI/CD prakse i naučiti da efikasno i bez grešaka isporučimo aplikaciju u proizvodnju. Za to ćemo koristiti alate **Jenkins** i **Argo CD**. Ova knjiga će vam pružiti sve što vam je potrebno da preuzmete ulogu modernog DevOps inženjera u eri oblaka i kontejnera.

Rezime

U ovom poglavlju smo razumeli moderne DevOps prakse, oblak i moderne aplikacije prilagođene oblaku. Zatim smo sagledali kako se softverska industrija brzo kreće ka kontejnerima i kako, uz oblak, postaje sve važnije da moderni DevOps inženjeri imaju potrebne veštine da se nose sa oba aspekta. Nakon toga, bacili smo pogled na arhitekturu kontejnera i diskutovali o nekim koracima u prelasku sa arhitekture zasnovane na virtuelnim mašinama na kontejnerizovanu arhitekturu.

U sledećem poglavlju, istražićemo upravljanje izvornim kodom pomoću **Git** sistema, što će biti osnova za sve što ćemo raditi u ostatku ove knjige.

Pitanja

Odgovorite na sledeća pitanja da biste proverili svoje znanje nakon ovog poglavlja:

1. Računarstvo u oblaku je skuplje od lokalnog računarstva. (Tačno/Netačno)
2. Računarstvo u oblaku zahteva veće **kapitalne izdatke (CapEx)** nego **operativne izdatke (OpEx)**. (Tačno/Netačno)
3. Koja od sledećih tvrdnji je tačna u vezi sa aplikacijama prilagođenim oblaku? (Izaberite tri)

   A. Obično prate arhitekturu mikroservisa

   B. Obično su monoliti

   C. Koriste kontejnere

D. Koriste dinamičku orkestraciju

E. Koriste lokalne baze podataka

4. Kontejnerima je potreban hipervizor za pokretanje. (Tačno/Netačno)

5. Koja od sledećih izjava u vezi sa kontejnerima nije tačna? (Izaberite jednu)

A. Kontejneri su virtuelne mašine unutar virtuelnih mašina

B. Kontejneri su jednostavni procesi operativnog sistema

C. Kontejneri koriste kontrolne grupe za pružanje izolacije

D. Kontejneri koriste softver za pokretanje kontejnera

E. Kontejner je privremeno radno opterećenje

6. Sve aplikacije mogu biti kontejnerizovane. (Tačno/Netačno)

7. Koje od sledećih je softver za pokretanje kontejnera? (Izaberite dva)

A. Docker

B. Kubernetes

C. Containerd

D. Docker Swarm

8. Koje aplikacije treba prvo kontejnerizovati?

A. API interfejse

B. Baze podataka

C. Velike računare

9. Kontejneri automatski prate CI/CD principe. (Tačno/Netačno)

10. Koja od sledećih je prednost razdvajanja vaših aplikacija na više delova? (Izaberite četiri)

A. Otpornost na greške

B. Kraći period ciklusa izdavanja

C. Nezavisno, fino skaliranje

D. Jednostavnost arhitekture aplikacije

E. Jednostavnija infrastruktura

11. Prilikom razdvajanja aplikacije na mikroservise, na koji aspekt treba da obratite pažnju?

A. Razdvajanje aplikacija na najmanje moguće komponente

B. Razdvajanje aplikacija na logičke komponente

12. Koju vrstu aplikacije treba prvo kontejnerizovati?

A. Bez stanja

B. Sa stanjem

13. Koji od sledećih su CaaS primeri? (Izaberite tri)

- **A.** Azure Functions
- **B.** Google Cloud Run
- **C.** Amazon ECS
- **D.** Azure ACS
- **E.** Oracle Functions

Odgovori

1. Netačno
2. Netačno
3. A, C, D
4. Netačno
5. A
6. Netačno
7. A, C
8. A
9. Tačno
10. A, B, C, E
11. B
12. A
13. B, C, D