POSLOVNA SOFTVERSKA OS REŠENJA

školska 2024/2025 godina

Vežba 8: Automatizacija i DevOps u poslovnim okruženjima

DevOps je **skraćenica od "Development" i "Operations"** i predstavlja pristup razvoju softverskih rešenja koji integriše:

- **Ljude** – kroz timsku saradnju i komunikaciju,
- **Procese** – kroz standardizaciju i automatizaciju,
- **Alate** – kroz upotrebu savremenih softverskih rešenja za razvoj, testiranje, postavljanje i nadgledanje aplikacija.

Cilj DevOps-a je da se omogući **brza, pouzdana i česta isporuka softverskih rešenja**, uz očuvanje **kvaliteta, stabilnosti i bezbednosti sistema**.

DevOps nije samo skup alata – to je **filozofija rada** koja podrazumeva promenu organizacione kulture, razbijanje silosa između timova i usvajanje fleksibilnih metoda rada.

Zašto je DevOps nastao?

U tradicionalnim IT organizacijama, dominirala je **rigidna podela odgovornosti**, što je često usporavalo razvoj i uvođenje inovacija:

- **Razvojni timovi** su pisali kod, fokusirajući se na funkcionalnost i nove mogućnosti.
- **Operativni timovi** (operations) su bili odgovorni za infrastrukturu, postavljanje aplikacija, sigurnost i stabilnost sistema.

Ovakav pristup je imao nekoliko ključnih problema:

- **Loša komunikacija** između timova dovodila je do neusklađenih ciljeva.
- Novi kod je često pravio probleme u produkcionom okruženju.
- Rešavanje problema je bilo sporo jer timovi nisu imali zajednički uvid u ceo sistem.

DevOps je nastao **kao odgovor na potrebu za boljom koordinacijom, bržim isporukama i većom pouzdanošću softverskih sistema**, naročito u okruženju gde se promene dešavaju često i brzo (npr. web aplikacije, mobilne platforme, SaaS rešenja).

Kako DevOps rešava ove probleme?

DevOps donosi **automatizaciju i saradnju** u sve faze softverskog razvoja:

Tradicionalni pristup	**DevOps pristup**
Razvoj i operacije su odvojeni	Razvoj i operacije su integrisani
Ručno testiranje i postavljanje	Automatizovano testiranje i isporuka
Nepredvidivo ponašanje softvera	Standardizovana okruženja (npr. Docker)
Spora reakcija na greške	Brza identifikacija i oporavak

DevOps kao kultura i skup vrednosti

Jedan od osnovnih principa DevOps-a jeste **kultura saradnje i poverenja**. DevOps promoviše:

1. Timska saradnja (Collaboration)

- Razvojni i operativni timovi ne rade „jedni protiv drugih“, već **zajedno**.
- Zajednički su odgovorni za uspeh aplikacije – od pisanja koda do monitoringa u produkciji.

🔷 *Primer: U DevOps okruženju, developer ne „završava posao“ kada napiše kod – već učestvuje u testiranju, praćenju performansi i rešavanju problema u radu aplikacije.*

2. Poverenje i odgovornost

- Timovi veruju jedni drugima.
- Svaki član tima ima uvid u celokupan proces i oseća se odgovornim za kvalitet softvera.

3. Kontinuirano učenje

- Uči se iz grešaka (tzv. „blameless postmortems“).
- Greške nisu povod za krivicu, već za unapređenje procesa.

Merenje i povratne informacije

DevOps nije samo skup alata i automatizacija, već stalno praćenje i unapređivanje procesa razvoja i isporuke softvera. Bez preciznih podataka o radu sistema i povratnim informacijama, teško je poboljšati kvalitet. Zato su merenje i povratne informacije ključni.

🔍 **Ključna pitanja koja timovi postavljaju:**

- Da li su isporuke softvera brze, pouzdane i bez grešaka?
- Koliko brzo možemo da otkrijemo i rešimo probleme u radu aplikacije?
- Da li korisnici dobijaju funkcionalnosti koje zaista koriste i vrednuju?

📈 **Metrike koje se redovno prate uključuju:**

- **Frekvenciju isporuka (Deployment Frequency):** Koliko često se pušta nova verzija softvera u produkciju – visoka učestalost ukazuje na agilnost tima.
- **Vreme oporavka nakon incidenta (Mean Time to Recovery – MTTR):** Koliko je potrebno da se sistem oporavi nakon pada – niže vreme znači bolju otpornost.
- **Stopu neuspešnih implementacija:** Koliko često puštanje nove verzije dovodi do greške i potrebe za povlačenjem – niža stopa ukazuje na bolji kvalitet.
- **Broj grešaka u produkciji:** Prati se kroz alate za logovanje, nadzor i prijave korisnika.

📬 **Povratne informacije korisnika**

Savremene DevOps platforme koriste **automatsko prikupljanje korisničkih podataka** (telemetriju) kao što su:

- Klikovi i ponašanje korisnika na sajtu,
- Učestalost korišćenja određene funkcije,
- Vreme provedeno na određenim delovima aplikacije,
- Napuštanje određenih stranica ili procesa (npr. checkout).

Ovi podaci se automatski analiziraju i daju timovima uvide u to **šta treba poboljšati**.

🔷 **Primer iz prakse:**
Zamislimo da korisnici često odustaju od rezervacije na poslednjem koraku – stranici za plaćanje. Zahvaljujući telemetrijskim podacima i analitičkim alatima (npr. Google Analytics, Mixpanel, Datadog), DevOps tim brzo dobija informaciju o problemu. Već u roku od nekoliko sati može se razviti, testirati i isporučiti unapređena verzija te stranice.

Ključne DevOps prakse

Kontinuirana integracija (CI)

U okviru CI, svaki put kada developer napravi izmene u kodu, te izmene se **automatski integrišu** u zajednički kodni repozitorijum. Nakon toga, CI sistem:

- Pokreće automatske testove
- Obaveštava tim ako postoji greška

Ovo sprečava da greške „prođu dalje“ u razvojni ciklus i skupi problemi se otkrivaju na vreme.

Kontinuirana isporuka (CD)

Kontinuirana isporuka omogućava **brzu i sigurnu isporuku softvera korisnicima**.

CD uvodi **automatizovane korake za postavljanje softvera**, uključujući:

- Pripremu paketa
- Postavljanje u testno okruženje
- Verifikaciju rada aplikacije

Ako se koristi **Continuous Deployment**, svaki prolaz testova automatski pokreće implementaciju u produkciono okruženje.

Alati koji omogućuju DevOps

Alat	Svrha
Jenkins	Automatizacija CI/CD procesa — pokreće buildove, testove i deployment automatizovano.
Docker	Pakovanje aplikacija u kontejnere sa svim zavisnostima.
Kubernetes	Orkestracija i automatsko skaliranje kontejnera u produkciji.
Git	Upravljanje verzijama koda i olakšava saradnju među programerima.
Terraform	Infrastruktura kao kod (IaC) — automatizuje kreiranje i održavanje serverske infrastrukture.
Prometheus + Grafana	Monitoring i vizualizacija performansi sistema i aplikacija u realnom vremenu.

Primer iz prakse: *Web aplikacija u DevOps okruženju*

🎯 **Scenario:**

Zamislimo firmu koja razvija **online platformu za rezervaciju smeštaja**, sličnu Booking.com ili Airbnb. Tim stalno radi na unapređenju – dodaju se nove funkcionalnosti kao što su:

- pametna pretraga po datumu i lokaciji,
- plaćanje karticama,
- ocene i komentari korisnika,
- preporuke na osnovu prethodnih rezervacija.

U takvom okruženju, gde se softver neprestano razvija i korisnici očekuju stabilnu i dostupnu uslugu 24/7, **DevOps pristup omogućava brzinu, kvalitet i pouzdanost isporuke**.

🔄 **1. Kontinuirana integracija i isporuka (CI/CD) sa Git i Jenkins**

🔧 **Tehnologije: GitHub (ili GitLab, Bitbucket) + Jenkins**

Šta se dešava?

- Svaki put kada developer završi deo funkcionalnosti (npr. "dodaj komentar na rezervaciju"), on šalje (commit/push) svoj kod na **GitHub repozitorijum**.
- Jenkins, kao alat za **automatsku integraciju (CI)**, detektuje promenu i automatski:
    - preuzima novi kod,
    - kompajlira aplikaciju (pretvara kod u izvršni oblik),
    - pokreće automatske testove (npr. da li nova funkcija radi i da li nije pokvarila postojeće).

✅ Ako svi testovi prođu:

- Jenkins automatski napravi novu verziju aplikacije,
- i (ako je deo CD procesa) postavi je na testno ili produkciono okruženje.

📌 **Cilj:** Brzo otkrivanje grešaka i **ubrzana isporuka funkcionalnosti**, bez čekanja ili manuelnog postavljanja koda.

📦 2. Kontejnerizacija sa Docker-om

🔧 Tehnologije: Docker

Zašto Docker?
U tradicionalnom razvoju, aplikacija koja radi na jednom računaru često ne radi na drugom zbog razlika u:

- operativnom sistemu,
- verzijama biblioteka,
- podešavanjima okruženja.

Docker rešava ovaj problem tako što:

- celokupnu aplikaciju (uključujući konfiguraciju, baze, sve zavisnosti) pakuje u **Docker kontejner**,
- kontejner funkcioniše **isto na bilo kom sistemu** – laptopu developera, test serveru ili produkcionom serveru.

📌 **Cilj:** Stabilnost i **jednako ponašanje aplikacije u svim okruženjima** (dev, test, prod).

⚙️ 3. Orkestracija sa Kubernetes-om

🔧 Tehnologije: Kubernetes (K8s)

Zamislimo sledeću situaciju:

- Platforma postaje popularna i ima 10,000 aktivnih korisnika dnevno.
- Jedan kontejner više ne može da opsluži sve zahteve – aplikacija usporava.

Kako Kubernetes pomaže?

- Automatski prepoznaje da aplikaciji treba više resursa.
- Pokreće dodatne kopije aplikacije u novim kontejnerima (horizontal scaling).
- Ako neki kontejner prestane da radi – Kubernetes ga automatski restartuje.
- Ako više korisnika prestane da koristi aplikaciju – smanjuje broj kontejnera.

📌 **Cilj:** Visoka dostupnost, **automatsko skaliranje i otporan sistem** bez ljudske intervencije.

📊 4. Monitoring i upozorenja sa Prometheus i Grafana

🔧 Tehnologije: Prometheus + Grafana

Zašto je važno nadgledanje?

- I najbolji sistemi mogu naići na probleme: spora baza, greške u kodu, problemi sa mrežom.
- DevOps tim mora znati **u realnom vremenu** šta se dešava.

Kako to funkcioniše?

- Prometheus beleži podatke kao što su:
  - broj aktivnih korisnika,
  - broj grešaka (npr. 500 – greška na serveru),
  - iskorišćenost CPU-a i memorije,
  - brzina odgovora aplikacije.
- Grafana prikazuje sve te podatke na **interaktivnim vizuelnim tablama**.
- Ako neki parametar pređe granicu (npr. memorija preko 90%), sistem **automatski šalje upozorenje** (Slack, e-mail, SMS...).

📌 **Cilj:** Proaktivno otkrivanje problema i **brzo reagovanje pre nego što korisnici primete**.

<u>Prednosti DevOps pristupa u realnim firmama</u>

- **Amazon:** Isporuka nove verzije softvera svake 11 sekunde – zahvaljujući automatizaciji i mikroservisima, promene se bezbedno puštaju u rad gotovo u realnom vremenu. To omogućava brzu reakciju na potrebe korisnika i konkurenciju bez zastoja u radu sistema.
- **Netflix:** Neprekidna isporuka i automatski oporavak sistema – uz alate za testiranje otpornosti, sistem sam detektuje i rešava probleme bez prekida za korisnike. Njihov DevOps tim koristi simulacije kvarova da bi unapred identifikovao slabe tačke u infrastrukturi.
- **Spotify:** Brzo testiranje novih funkcionalnosti uz minimizaciju rizika – nove opcije se prvo isprobavaju na ograničenom broju korisnika kako bi se izbegli problemi u širokoj upotrebi. Ovakav pristup omogućava da se inovacije brzo razvijaju, a istovremeno očuva stabilnost platforme.