DevOps

Koliko god da ste dobri i iskusni softverski inženjeri ne moguće je u prvom pokušaju napraviti idealan softver. Upravo iz ovog razloga, savremeni razvoj softvera zahteva česte izmene već postojećeg softvera (dodavanje novih funkcionalnosti, popravljanje grešaka, optimizacije, povećanje bezbednosti itd.) koji se nalazi u produkciji i redovno koristi od strane korisnika. Izmene treba da budu što pre integrisane u softver, a da se pritom ne naruši stabilnost softvera ili kompletnog sistema. Međutim, uspostavljanje ovakvog procesa razvoja softvera (pre pojave *DevOps*-a) je jako teško, s obzirom na to da su se interesi programera i admnistratora razlikovali. Programeri imaju tendenciju da što brže uvode nove funkcionalnosti i izmene u produkciju, dok nasuprot tome, admnistrator imaju tendenciju da održavaju sistem stabilnim i zbog toga ne preferiraju česte izmene u sistemu. Upravo ovaj problem rešava *DevOps* praksa.

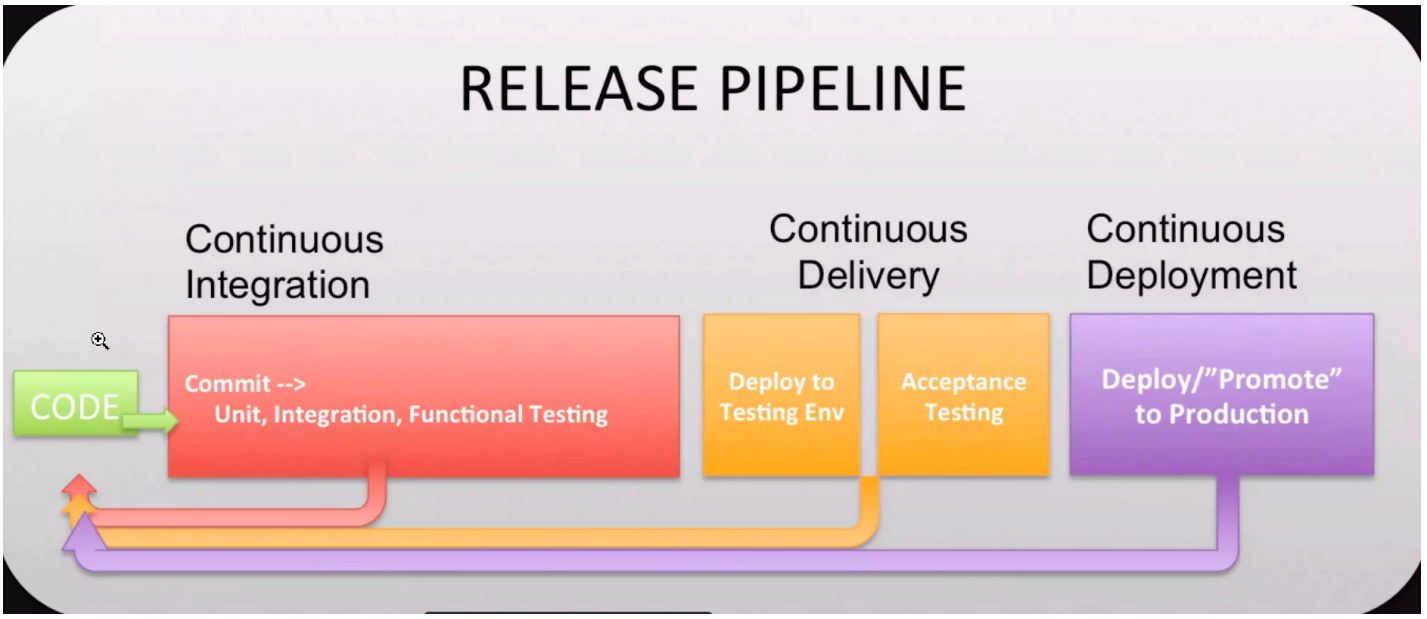
DevOps (**Development and Operations**) je kultura ili filozofija koja ima za cilj da smanji razliku između razvoja softvera (**Development**) u kojem učestvuju programeri i održavanje sistema (**Operations**) koji rade sistemski administratori kako bi omogućila brži i kvalitetniji proces razvoja softvera. DevOps nadmeće potrebu za većom komunikacijom između programera i administratora, kao i uspostavljanjem što veće automatizacije operacija koje se svakodnevno vrše prilikom razvoja softvera. Neke od tih operacija su: *build*-ovanje aplikacije, testiranje aplikacije, itd (ovo se odnosi *Dev* “deo” u *DevOps* definiciji). Takođe, nakon uspešnog *buid*-ovanja, testiranja (i ostalih faza) aplikacije, neophodno je tu aplikaciju plasirati (*deployment*) u različita okruženja i vršiti njeno nadgledanje (*monitoring*) i održavanje (ovo se odnosi na *Ops* “deo” u *DevOps* definiciji) (Slika 1). Zbog toga *DevOps* inženjer mora da poseduje u određenoj meri veštine i znanje programera i administratora (jasna granica ne postoji jer neophodno znanje obuhvata više IT oblasti).

Slika 1: DevOps domen

Automatizacija različitih faza razvoja softvera se uspostavlja pomoću *Continuous Integration / Continuous Deployment* (*CI/CD*) mehanizma (Slika 2). CI/CD mehanizam se implementira kreiranjem *pipeline*-ova sa definisanim zadacima koji se najčećse grupišu u 3 faze:

- **Continuous Integration** - build aplikacije, pokretanje *Unit* i *Integration* testova, *packaging* aplikacije i kreiranje *artifac*t-a (neki vid izvršive datoteke - binary).
- **Continuous Delivery** - plasiranje (*deloyment*) artifact-a aplikacije na test ogruženje (staging environment) na kojem će se testirati aplikacija od strane QA tima izvršavanje različitih testova (E2E, penetration test, performance test).
- **Continuous Deployment** - plasiranje (deployment) istestiranog artifact-a aplikacije u produkciju u kojoj će biti korišćenja od strane krajnjih korisnika.

Sve 3 faze se ponavljaju ukoliko je neophodno napraviti neke promene u produkciji. Postoji bezbroj alata koji omogućavaju kreiranje pipeline-ova i CI/CD mehanizma, a među najpopularnijim su: *GitHub Actions*, *GitLab CI*, *Jenkins*, *AWS CodePipeline*, *Azure DevOps*, *Concourse*, *Bamboo*.



Slika 2: CI/CD

Idealan slučaj razvoja softvera bi bio kada bi programeri razvijali fleksibilan softver u kojem je omogućeno **lako** dodavanje novih funkcionalnosti i izmena, dok bi *DevOps* inženjeri kreirali neophodnu infrastukturu koja će u **najkraćem mogućem roku** plasirati te izmene u produkciju.

Dodatni materijali:

- What is DevOps
- What is DevSecOps
- What is SRE
- Git workflows
- Continuous Integration vs Feature Branch Workflow
- Git commit messages
- Coventional Commits