
Gruppetime 12

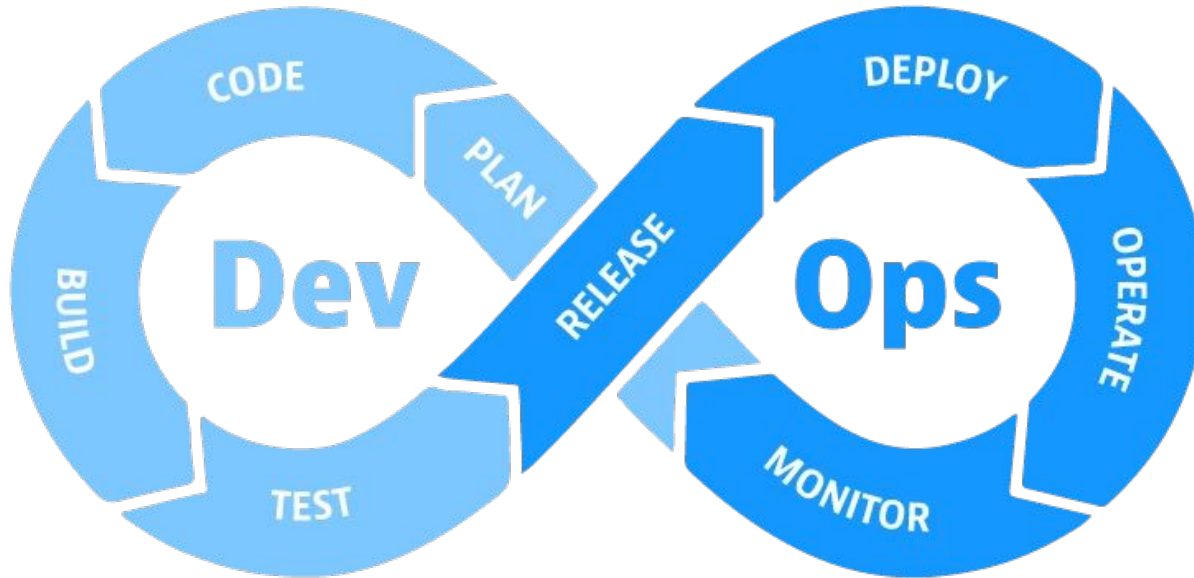
IN1030

Plan for dagen

- DevOps
- Oblig 5

DevOps

→ DevOps: kort for Development & Operations



“Tradisjonelt” - Release and support

- **Tradisjonelt:** Produktutvikling og vedlikehold har vanligvis bestått av separate team:
 - ◆ Utvikling, Utgivelse, Support, Vedlikehold (om den jobben ikke er gjort av utviklingsteamet)
- **Problemer med dette:** Manglende forståelse mellom team, forlenget utviklingssyklus, mangel på samarbeid.
- **DevOps** tar sikte på å løse disse utfordringene ved å fremme samarbeid og integrasjon mellom utviklings-, utgivelses-, support- og vedlikeholdsteam gjennom automatisering, deling av ansvar og en felles forståelse av programvareleveranseprosessen. Dette bidrar til å redusere utviklingssykluser, forbedre programvarekvaliteten og øke kundetilfredsheten.

DevOps prinsipper

DevOps prinsipp	Forklaring
Alle er ansvarlig for alt	Alle i teamet har delt ansvar for utvikling, utgivelse og vedlikehold/support av programvaren.
Alt som kan bli automatisert burde bli det	All testing-, utgivelse- og supportsaktiviteter bør bli automatisert når det er mulig. Legg opp til minst mulig manuelt arbeid med utgivelsen av programvaren.
Mål først, endre etterpå	DevOps burde bli drevet av målingsprogram hvor du samler data om systemet og operasjonene. Avgjørelser som angår endring i DevOps prosessen og verktøy, bør ta med denne dataen som grunnlag.

Fordeler

→ **Raskere utgivelse/leveranse (Deploy)**

- ◆ Fordi: kommunikasjons-delay mellom teamene er blitt drastisk redusert

→ **Redusert risiko**

- ◆ Fordi: inkrementet i hver leveranse er mindre, og dermed enklere å lokalisere kilden til problemet

→ **Raskere reparasjon**

- ◆ DevOps teamene jobber sammen for å få programvaren oppe å går
- ◆ Man trenger heller ikke å finne ansvarlig team og vente på at de fikser det dersom et problem oppstår

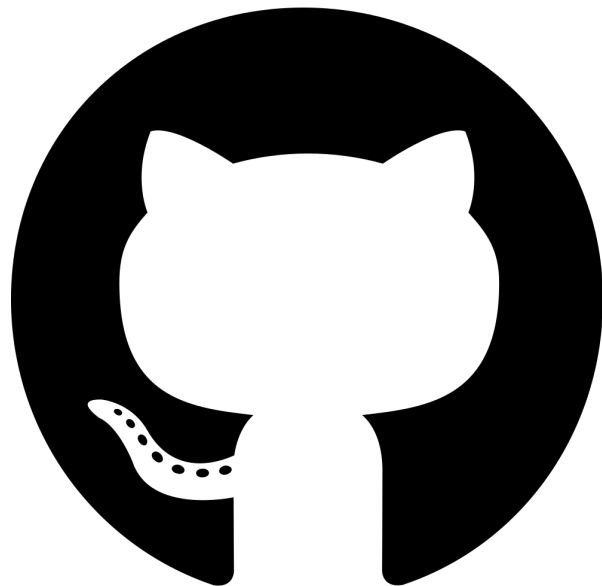
→ **Produktive team**

- ◆ Teamene blir gladere og mer produktive fordi de jobber sammen og ikke på separate områder

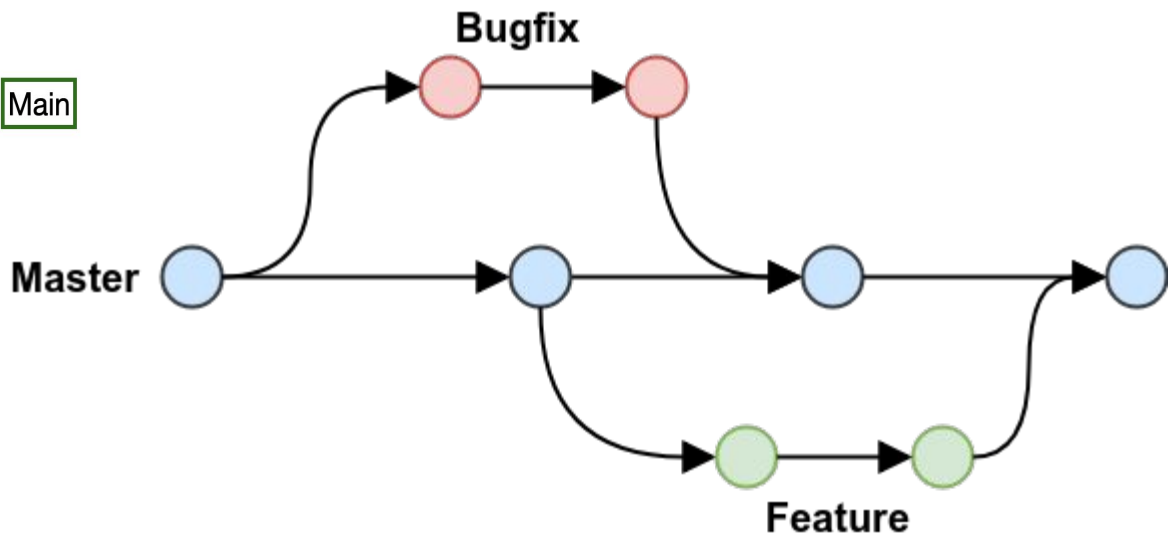
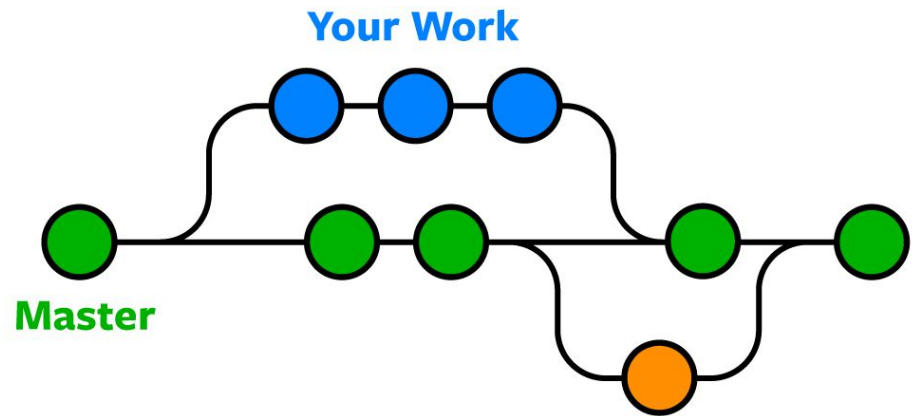
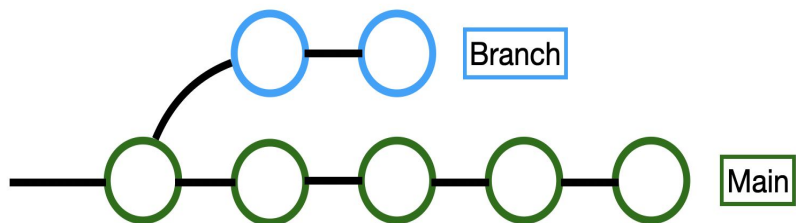
Kodeadministrasjon/Code management

Kodeadministrasjon/Code management i DevOps refererer til bruk av verktøy som Git og plattformer som GitHub for å administrere og samarbeide om koden i en inkrementelt voksende kodebase.

- **Kodeoverføring**
 - ◆ Utviklere arbeider lokalt på sine maskiner og bruker et kodeadministrasjonssystem som Git til å lagre og spore endringene i koden.
- **Versjonskontroll og henting**
 - ◆ Git tilbyr versjonskontroll og mulighet for å hente tidligere versjoner av koden.
- **Merging/branching**
 - ◆ Ved å bruke grener (branches) kan utviklere arbeide parallelt med forskjellige funksjoner eller fikser.
 - ◆ Merging lar utviklere kombinere endringene fra en gren til hovedløsningen når de er ferdige.
- **Versjonsinformasjon**
 - ◆ Git gir detaljert informasjon om endringer som blir gjort, inkludert hvilken utvikler som gjorde endringen, når endringen ble gjort og hva som ble endret.



Git



Den automatiserte prosessen med DevOps

→ Kontinuerlig integrasjon - Continuous integration

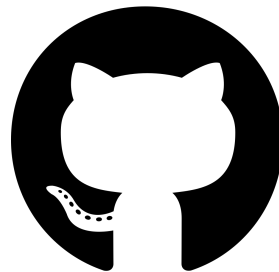
- ◆ Kodeendring blir gjort -> testing av implementert endring skjer
- ◆ Enklere lokalisering av eventuelle feil
- ◆ Systemet blir tilgjengelig for alle i teamet ved jevnlig oppdateringer
- ◆ Kodehåndtering - GitHub

→ Kontinuerlig levering - Continuous delivery

- ◆ Endringer blir sendt til et "production-like environment" som ligner veldig det faktiske systemet
- ◆ Kjøres her automatiske tester

→ Kontinuerlig utgivelse - Continuous deployment

- ◆ Oppdatering når main branch endres (Jenkins)
- ◆ Automatisk test -> automatisk slipp



Jenkins



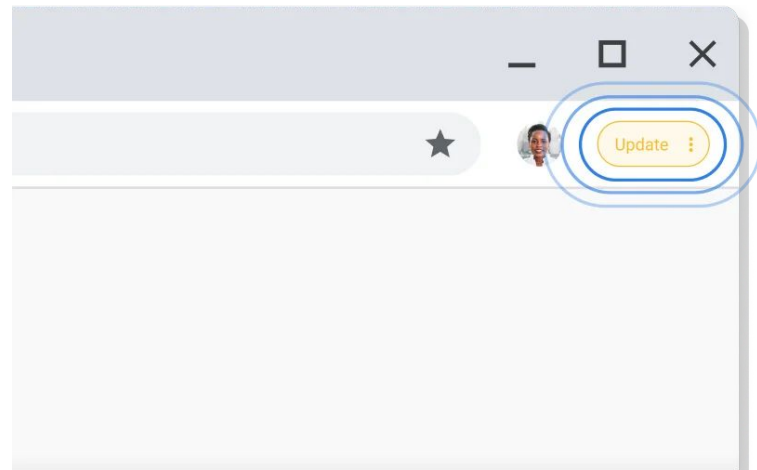
puppet

Fordeler med kontinuerlig utgivelse

- **Redusert kostnad:** kontinuerlig utgivelse er hurtigere og mer effektivt enn manuell - som også ofte feiler
- **Hurtigere problemløsning:** problem påvirker kun en liten del, fordi ting oppdateres kontinuerlig, enklere å finne årsak til feil
- **Hurtigere kundefeedback:** identifisere forbedring gjennom å stadig gi ut nye funksjonaliteter
- **A/B testing:** noen bruker gammel og andre ny versjon - kan sjekke hva som fungerer best

“Obs obs” aspekter med kontinuerlig utgivelse

- **Ukomplett funksjonalitet** - lekkasje av geniale idéer
- Irriterte kunder som ikke oppdaterer fordi de må gjøre det så ofte
 - ◆ Eks: Mac-oppdatering, Chrome-oppdatering
- **Synkronisering av bedriftenes ønsker og dine utgivelser** - husk på det
 - ◆ Eks: lønnssystemet til UiO



Tilretteleggelser for DevOps

→ **“Change of mindset and culture”**

- ◆ Utveksle ansvar, kunnskap og informasjon om softwaren som lages av utviklere og driftsteam
- ◆ Man jobber sammen om oppgavene

→ **“Change of tools”**

- ◆ Ta seg bruke av ulike verktøy som støtter ulike deler i devops. Eksempler:
 - Jenkins (testing)
 - GitHub (kontroll på alle kodeendringer og versjoner → kode integrasjon)
 - SaltStack (kode levering)

→ **“New skills”**

- ◆ For å imøtekomme de overnevnte, kreves nødvendige ferdigheter rundt verktøy og kommunikasjon

Måling

Ønske om å kontinuerlig forbedre seg for å oppnå raskere utgivelse og bedre kvalitet på programvaren → man trenger å måle litt for å se hvordan ståa er

Prosessmetriker:

- Gjennomsnittstid på gjenopprettelse
- Prosentandel feilutgivelser
- Utgivelsesfrekvens
- Endringsvolum
- Forsinkelse fra utvikling til utgivelse

Tjenestemetriker:

- Prosent økning i sluttbrukere
- Antall klager
- Tilgjengelighet
- Ytelse

Prosessendringsforslag

- Disse målingene kan føre til endringer i nåværende og fremtidige prosesser.
- Endringsaspekter kan være:
 - ◆ Revurdere de automatiserte testene
 - ◆ Sjekke kodehåndtering
 - ◆ Ansette flere
 - ◆ Revurdere målsetting

Spørsmål?

Ikke nøl med å sende oss en mail:)

erikaesk@uio.no

sandefo@uio.no
