Rollback-strategioiden merkitys ja automatisointi web-kehityksessä

Turun yliopisto
Tietotekniikan laitos
TkK-tutkielma
Tietotekniikka
Maaliskuu 2025
Aaro Eronen

TURUN YLIOPISTO
Tietotekniikan laitos

Aaro Eronen: Rollback-strategioiden merkitys ja automatisointi web-kehityksessä

TkK-tutkielma, 18 s.
Tietotekniikka
Maaliskuu 2025

Web-kehityksessä jatkuva integraatio ja toimitus mahdollistavat entistä nopeamman ohjelmistokehityksen. Nopea kehityksen tahti ja suuret päivitysmäärät verkkosovelluksiin altistavat ne mahdollisille virheille ja käyttökatkoille. Rollback-strategiat ja niiden automatisointi ovat keskeinen osa luotettavaa julkaisuprosessia, sillä ne mahdollistavat nopean reagoinnin ja palautumisen edelliseen toimivaan versioon virhetilanteissa.
Tässä tutkielmassa tarkastellaan rollback-strategioiden roolia web-kehityksessä ja vertaillaan eri strategioita. Manuaalinen rollback, blue-green deployment ja canary deployment tarjoavat erilaisia lähestymistapoja julkaisuprosessin hallintaan. Lisäksi tutkielmassa tarkastellaan rollback-strategian automatisointia GitLabin, Kuberneteksen ja Prometheuksen avulla, mikä voi parantaa reagointinopeutta ja vähentää inhimillisiä virheitä.
Tutkielman tulokset osoittavat, että hyvin suunniteltu rollback-strategia ja sen mahdollinen automatisointi ovat tärkeä osa luotettavaa ja tehokasta jatkuvan integraation ja toimituksen prosessia.

Asiasanat: rollback-strategia, automaatio, web-kehitys, DevOps, CI/CD

Sisällys

1 Johdanto

Tänä päivänä web-kehityksessä sovelluksia päivitetään ja julkaistaan yhä nopeammin, mikä usein saattaa aiheuttaa mahdollisia virheitä internetissä oleviin palveluihin ja verkkosovelluksiin. Nykyajan kehittyneen teknologian myötä verkkosovellukset muistuttavat jo työpöytäsovelluksia ja sisältävät paljon dynaamisia ja interaktiivisia ominaisuuksia [1]. Yhä useammat palvelut sijaitsevat verkossa ja niitä käyttävät niin tavalliset ihmiset kuin monet viranomaisetkin. On siis entistä tärkeämpää, että verkkosovellukset ovat luotettavia ja niiden käyttökatkot mahdollisimman lyhyitä. Esimerkiksi pankkipalvelut ovat yksi kriittisistä palveluista, joita on mahdollista käyttää verkossa. Tämän takia myös datan, kuten esimerkiksi pankkitilien reaaliaikaisten saldojen säilyminen virhetilanteissa on entistä tärkeämpää.

Jatkuva integraatio ja toimitus (CI/CD) ovat yleistyneet ja kehittyneet paljon web-kehityksessä. Continuous integration (CI) eli jatkuva integraatio tarkoittaa verkkosovelluksen lähdekoodin muutosten jatkuvaa integrointia jo olemassa olevaan lähdekoodiin [2]. Tämä helpottaa virheiden paikantamista, kun muutoksia tehdään pienin askelin kerrallaan. Continuous delivery (CD) eli jatkuva toimitus taas tarkoittaa verkkosovellukseen tehtyjen päivitysten jatkuvaa julkaisua käyttäjille [2]. Lähdekoodiin tehdyt muutokset kulkevat automaattisten testien läpi ja ne läpäistyään päätyvät käyttäjien käyttöön.

Suuret yritykset kuten Amazon ja Netflix päivittävät tai julkaisevat uusia ominaisuuksia palveluihinsa jopa tuhansia kertoja päivässä [2]. Yritysten on tärkeää

käyttää aikaa ja resursseja hallitakseen suureen julkaisu- ja päivitysmäärään liittyviä mahdollisia virhetilanteita, jotta voidaan varmistaa luotettava ja vakuuttava käyttökokemus. Rollback-strategiat ovat yksi keino hallita näitä ongelmia ja niiden avulla sovellukset voidaan nopeasti palauttaa edelliseen toimivaan tilaan virheen tai ongelman ilmetessä.

Tämän tutkielman tavoitteena on käsitellä web-sovellusten rollback-strategioita, sekä tarkastella niiden mahdolliseen automatisointiin käytettäviä työkaluja jatkuvan integraation ja toimituksen (CI/CD) ympäristössä. Tutkielma toteutetaan kirjallisuuskatsauksena, ja siinä hyödynnetään alan tutkimusartikkeleita, tieteellisiä julkaisuja, kirjoja sekä työkalujen dokumentaatioita. Tavoitteena on vastata seuraaviin tutkimuskysymyksiin:

TK1 Mikä on rollback-strategioiden merkitys web-kehityksessä?

TK2 Mitä erilaisia rollback-strategioita on?

TK3 Mitä työkaluja rollback-strategian automatisointiin tarvitaan?

Tutkielmassa tarkastellaan web-kehityksen nykytilaa ja yleisiä käytäntöjä sekä rollback-strategioiden tärkeyttä nopeassa kehityksen tahdissa (TK1). Tarkastelun kohteena ovat myös erilaiset rollback-strategita ja miten ne voivat tukea julkaisuprosessin luotettavuutta ja tehokkuutta (TK2). Lisäksi tutkielmassa kuvataan, mitä työkaluja rollback-strategioiden automatisointi vaatii ja miten se voi tehostaa CI/CD-prosessia ja parantaa sovellusten toiminnan jatkuvuutta (TK3).

Tutkielmassa käytettyjä aineistoja on koottu IEEE Xplore-, arXiv-, Springer- ja ACM-tietokannoista. Hakusanoina on käytetty ”rollback”, ”rollback-strategy”, ”CI/CD-pipeline”, ”devops”, ”deployment models” ja ”automation”. DevOpsista ja

CI/CD-putkista löytyi runsaasti tuloksia ja niistä on valittu mukaan relevanteimmat tiivistelmän perusteella. Näistä tuloksista pyrittiin valitsemaan myös kohtuullisen ajankohtaisia julkaisuja ja tuloksia rajattiin vuoden 2015 jälkeen julkaistuihin. Rollback-strategioihin ja ohjelmistokehitykseen liittyviä hakutuloksia oli taas melko vaikea löytää. Rollback-strategiat ja niiden automatisointi on melko uutta, joten tulokset tähän liittyen pyrittiin rajaamaan vuonna 2020 ja sen jälkeen julkaistuihin aineistoihin. Tämän rajauksen avulla pyrittiin löytämään relevantteja ohjelmistokehitykseen ja tietotekniikkaan liittyviä julkaisuja. Tieteellisten aineistojen lisäksi tutkielmassa on hyödynnetty kolmea alan oppikirjaa sekä GitLabiin, Kubernetekseen ja Prometheukseen liittyviä dokumentaatioita ja tutoriaaleja.

Tutkielman toisessa luvussa käsitellään jatkuvaa integraatiota (CI) ja jatkuvaa julkaisuprosessia (CD). Luvussa esitetään myös, millaista nykypäivän web-kehitys on ja millainen merkitys rollback-strategioille on sen myötä kehittynyt. Toisessa luvussa vastataan ensimmäiseen tutkimuskysymykseen (TK1 Mikä on rollback-strategioiden merkitys web-kehityksessä?). Kolmas luku taas keskittyy erilaisiin rollback-strategioihin ja riskien hallintaan virhetilanteissa. Luvussa vastataan toiseen tutkimuskysymykseen (TK2 Mitä erilaisia rollback-strategioita on?). Neljännessä luvussa tutustutaan rollback-strategian toteuttamiseen ja siihen tarvittaviin työkaluihin ja niiden rooliin. Luvussa vastataan tutkielman kolmanteen tutkimuskysymykseen (TK3 Mitä työkaluja rollback-strategian automatisointiin tarvitaan?).

2 DevOps, CI/CD ja rollback

DevOps on lähestymistapa joka, yhdistää ohjelmistokehityksen (Dev) ja IT-toiminnot (Ops/operations). Sen tavoitteena on tiivistää ohjelmistokehityksen ja järjestelmän ylläpidon välistä yhteistyötä. Se takaa toimitusprosessin nopeuden ja ohjelmiston laadun. Jatkuva integraatio (CI) ja jatkuva toimitus (CD) ovat DevOpsin tärkeimmät tekniset komponentit. Jatkuvan integraation perusajatuksena on kehittäjän jatkuva työnsä yhdistäminen eli integraatio yhteiseen ohjelmistoon. Jatkuvan integraation jälkeen jatkuva toimitus mahdollistaa uuden koodin jatkuvan ja automaattisen siirtymisen tuotantoon. DevOps ja CI/CD ovat yleistyneet paljon ja se on muuttanut ohjelmistotuotantoa ja -kehitystä. [3]

Rollback-strategia web-kehityksen kontekstissa voidaan määritellä toimintana, jossa verkkosovellus palautetaan aiempaan toimivaan tilaan julkaisussa ilmenneiden virheiden tai häiriöiden korjaamiseksi. Sen tarkoituksena on varmistaa palvelun jatkuvuus ja minimoida vaikutukset verkkosovelluksen toimintaan. Tämä sisältää aiemman toimivan version palauttamisen ja siihen liittyvien tietokantamuutosten hallinnan. [4]

2.1 Web-kehityksen nykytila

Nykyisin web-kehityksessä markkinoilla menestyminen vaatii kykyä reagoida nopeasti käyttäjäpalautteeseen. Web-teknologioiden nopea kehitys ja jatkuva innovaatio edellyttävät kehittäjiltä ja yrityksiltä jatkuvia päivityksiä sekä uusien teknologioiden omaksumista pystyäkseen vastaamaan käyttäjien vaatimuksiin [1]. DevOps ja siihen liittyvät CI/CD-periaatteet ovat ratkaisevia tekijöitä web-kehityksen nopeuden kasvattamisessa. DevOpsin avulla voidaan kehittää, testata ja julkaista päivityksiä nopeammin, mikä auttaa pysymään kilpailijoiden tahdissa. CI/CD-periaatteilla on merkittävä vaikutus käyttäjäkokemukseen, sillä ne vähentävät julkaisuprosessiin liittyviä käyttökatkoja ja häiriöitä. Nämä periaatteet mahdollistavat hallitut ja usein myös täysin automaattiset päivitykset, jotka eivät häiritse käyttäjiä merkittävästi. [3]

Automaattiset testit ovat osa jatkuvaa integraatiota. Jokaisen uuden muutoksen jälkeen lähdekoodi testataan automaattisesti ennalta määriteltyjen testien avulla. Ne auttavat havaitsemaan inhimillisiä virheitä ja pyrkivät varmistamaan, että koodi on yhteensopivaa edellisen version kanssa ja toimisi ilman ongelmia. Tämä tehostaa kehittäjän toimintaa ja vapauttaa aikaa tehdä muita tehtäviä. [5]

Virheiden tunnistaminen tapahtuu automaation myötä aikaisessa vaiheessa ja niiden paikantaminen on helppoa. Tämän seurauksena virheiden korjaamisen kustannukset sekä vaikutukset tuotantoympäristöön voidaan minimoida. Automatisoinnin avulla on siis mahdollista siirtää uusi koodi tuotantoon nopeammin ja turvallisemmin. [5] CI/CD ja DevOps eivät siis ole ainoastaan teknisiä ratkaisuja, vaan ne ovat välttämättömiä vastauksia liiketoiminnan ja käyttäjäkokemuksen nykyisiin vaatimuksiin [3].

2.2 Julkaisun haasteet ja rollbackin merkitys

Nykyiseen nopeaan kehitys- ja julkaisutahtiin liittyy paljon haasteita ja riskejä. Nopeassa julkaisutahdissa jatkuva integrointi ja yhteensopivuuden varmistaminen voi olla haastavaa, erityisesti suurissa tiimeissä, joissa useat kehittäjät tekevät jatkuvasti muutoksia lähdekoodiin. Tämä johtaa usein konflikteihin, joissa yksittäisellä kehittäjällä voi olla erilainen tapa ratkaista ongelmia muihin verrattuna. Koodin yhdistäminen voi aiheuttaa haasteita, kun kehittäjät työskentelevät eri komponenttien parissa rinnakkain. Läpinäkyvyyden ja tiedonkulun puutteen vuoksi kehittäjät eivät aina ole välttämättä täysin tietoisia projektin eri osa-alueiden muutoksista. Tällöin virheiden määrä suurissa tiimeissä ja nopeassa kehitystahdissa voi kertaantua. [5]

CI/CD-putki tarkoittaa automaattista prosessia, jossa lähdekoodi kootaan, integroidaan, testataan ja otetaan käyttöön. CI/CD-putkesta voi laajoissa projekteissa tulla erittäin monimutkaisia, mikä heikentää hallittavuutta. Suurissa ja monimutkaisissa projekteissa yhtenä haasteena on myös testauksen kattavuuden puute. Jokaisen mahdollisen virheen huomioiminen testauksessa on vaikeaa, minkä seurauksena virheitä pääsee joskus tuotantoon. Nopeaan julkaisutahtiin liittyy usein myös tietoturvahaasteita, koska kaikki lähdekoodiin tehdyt muutokset voivat sisältää uusia haavoittuvuuksia [6]. Nämä tietoturvahaasteet voivat olla erittäin kriittisiä ja vaativat nopeaa reagointia luotettavan käyttäjäkokemuksen takaamiseksi.

Heinäkuussa 1999 ebay.com-verkkosivusto oli pois toiminnasta neljä kertaa toistuvien tietokantapalvelimen ongelmien takia. Pisin katkoista kesti jopa 21 tuntia ja tuotti yritykselle noin viiden miljoonan dollarin tappiot. Syyskuussa 2011 Applen MobileMe-palvelut kokivat myös samanlaisia käyttökatkoja, jolloin palvelut olivat 75 prosentilla asiakkaista käyttökelvottomia useasti tuntien ajan. Näistä katkoista Apple sai paljon kritiikkiä ja brändi koki kolauksen. Molemmat tapauksista ovat vanhoja, mutta suurten verkkosovellusten julkaisuun ja päivittämiseen liittyy edelleen samanlaisia riskejä. Tällaisissa tilanteissa tulee olla suunnitelma ja tulee pystyä

reagoimaan nopeasti yrityksen maineen ja taloudellisen edun suojaamiseksi. Nopea vaihtoehto on palata edelliseen toimivaan versioon (”rollback”). [4]

Nykypäiväisessä nopeassa kehityksen tahdissa laadun on siis oltava hyvä kilpailukyvyn takaamiseksi. Käyttökatot on pyrittävä minimoimaan. Julkaisuprosessin haasteiden myötä virheitä sattuu ja ne voivat päätyä tuotantoon. Tällöin rollback-toimintojen on pystyttävä nopeasti palauttamaan verkkosovelluksen edellinen toimiva versio, sillä virheellisten päivitysten vaikutukset on pyrittävä minimoimaan [5]. Rollback-strategiat ovat suunniteltuja toimintastrategioita ja menetelmiä uuden julkaisun tai päivityksen käyttöönottoon virhetilanteiden varalta. Virheen päädyttyä tuotantoon rollback-strategiat mahdollistavat turvallisen palautumisen aiempaan vakaaseen sovellusversioon. Rollback-strategian puute jatkuvan integraation ja toimituksen prosessissa on yksi merkittävimmistä huonoista ohjelmistotuotannon käytännöistä [7]. Rollback-strategia on myös mahdollista automatisoida, joka lisää tuotantoympäristön luotettavuutta [6]. Kun julkaisutahti on nopea, virheisiin reagointi nopeasti on tärkeää. Rollback-strategian automatisointi on yksi ratkaisuista tähän.

3 Rollback-strategiat

Web-kehityksessä rollback-strategiat ovat siis keskeisiä julkaisuprosessien hallinnassa. Niiden avulla voidaan minimoida nopeaan julkaisu- ja päivitystahtiin liittyvät riskit ja varmistaa verkkosovelluksen vakaus ja jatkuvuus myös virhetilanteissa. Tässä luvussa tehdään katsaus erilaisiin lähestymistapoihin, rollback-strategioihin, niiden toimintaperiaatteisiin ja käyttötilanteisiin. Oikean strategian valitsemisen tulisi perustua verkkosovelluksen monimutkaisuuteen, käytettävissä olevaan infrastruktuuriin ja sovelluksen tavoitteisiin [8].

3.1 Manuaalinen rollback

Manuaalinen rollback-strategia on vaihtoehdoista suoraviivaisin ja yksinkertaisin toteuttaa. Se vaatii manuaalisesti komentojen suorittamisen, jotta saavutetaan edellinen verkkosovelluksen toimiva tila versionhallintatyökalun avulla. Tätä vaihtoehtoa käytetään usein silloin, kun muita vaihtoehtoja ei ole mahdollista käyttää tai rollback-prosessin monimutkaisuus vaatii ihmisen näkemystä. [8]

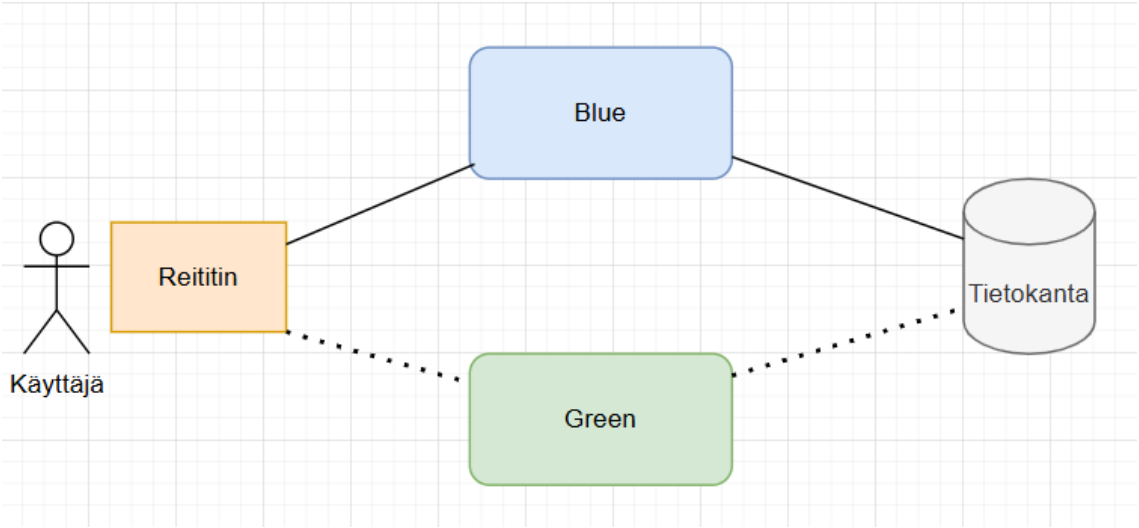
Manuaalista rollback-strategiaa käytettäessä on ensin vian ilmetessä määriteltävä missä kohtaa vika on ilmennyt ja valita mihin toimivaan versioon palataan. Sen jälkeen manuaalisesti otetaan toimiva versio käyttöön. Usein tähän käytetään komentorivityökaluja tai pilvipalvelun käyttöliittymää. Suurin etu manuaalisessa strategiassa on sen yksinkertaisuus ja mahdollisuus kontrolloida itse koko prosessia.

Tämä on kuitenkin muihin strategioihin verrattuna aikaa vievää ja mahdollisuus inhimillisille virheille on suurempi. Ilman automaattista hälytysjärjestelmää tai verkkosovelluksen valvontaa virheen huomaaminen saattaa myös viivästyä pitkään. [8]

3.2 Blue-green deployment

Blue-green deployment on rollback-strategia, joka pyrkii minimoimaan käyttökatkot olemattomiksi. Kahta identtistä ympäristöä ylläpidetään samanaikaisesti. Toista ympäristöä kutsutaan siniseksi (blue) ja toista vihreäksi (green). Tässä strategiassa toinen ympäristöistä on käyttäjien käytössä ja toista ympäristöä voidaan kehittää tai testata. Kun uusi versio tai päivitys verkkosovelluksesta on valmis ja todettu toimivaksi se julkaistaan. Silloin toinen aikaisemmin käytössä ollut ympäristö pidetään varalla valmiudessa virheiden varalta. Virheen ilmetessä voidaan välittömästi suorittaa rollback eli palata edelliseen toimivaan ympäristöön. Uuden ympäristön toiminnan varmistuttua käytössä voidaan toista ympäristöä alkaa kehittämään. Kuvassa 3.1 nähdään tilanne, jossa sininen ympäristö on käytössä. Tässä tilanteessa vihreä ympäristö on, joko varalla virheen varalta tai kehitystyön alla.

Blue-green -deployment-strategia mahdollistaa lähes olemattomat käyttökatkot ja on erittäin luotettava vaihtoehto. Se on hyvä valinta verkkosovellukselle, jonka on pysyttävä toiminnassa ilman katkoja. Tätä strategiaa käytettäessä tarvitaan kaksi samanaikaisesti toimivaa ympäristöä, mikä kuitenkin vaatii paljon resursseja ja lisää monimutkaisuutta. Myös datan synkronointi kahden ympäristön välillä voi tuottaa haasteita. [8] [9]



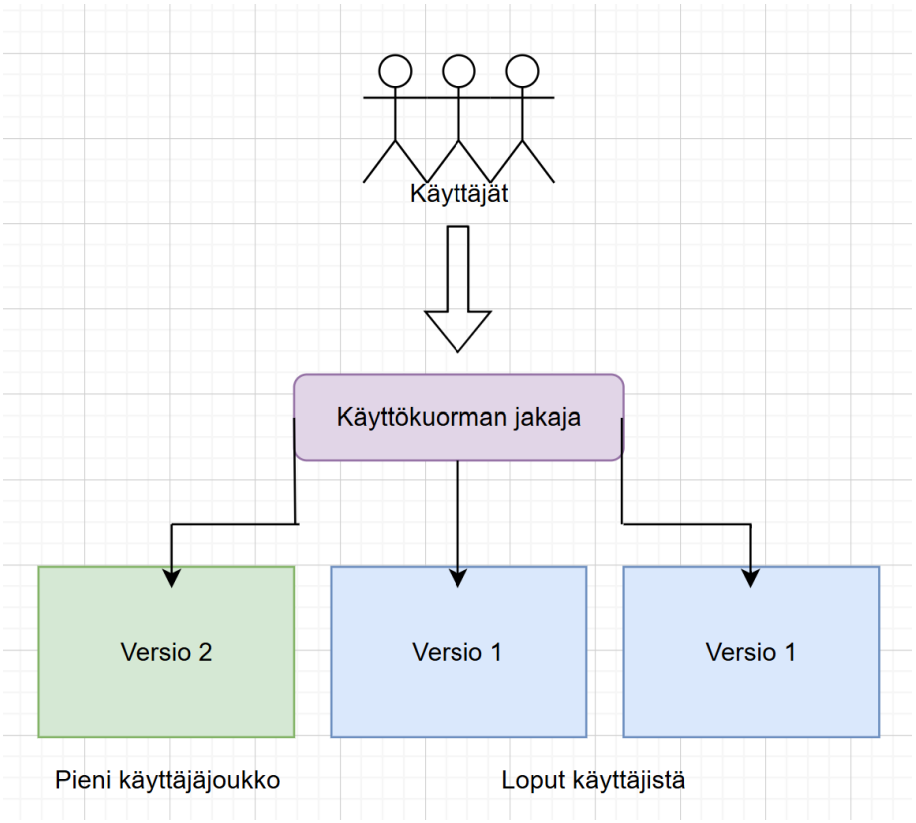
Kuva 3.1: Blue-green deployment. Sininen ympäristö on käytössä, jolloin vihreä voi olla työn alla. Mukailtu lähteestä [9].

3.3 Canary deployment

Blue-green deploymentissa käyttäjät siirretään uuteen tuotantoympäristöön ja vanha ympäristö jätetään varalle. Canary deploymentissa taas verkkosovelluksen uusi versio otetaan käyttöön vain pienellä osalla käyttäjistä kerralla. Käyttäjien määrää lisätään asteittain, jotta kehittäjät ymmärtävät paremmin uuden version toiminnan tuotantoympäristössä. Mahdollisen virheen ilmaantuessa vain pieni osa käyttäjistä kokee sen vaikutuksen ja palaaminen vanhaan vakaaseen versioon on mahdollista. Esimerkiksi Facebook on julkaissut uusia ominaisuuksia palveluissaan käyttöön ensin työntekijöilleen ja vasta sen jälkeen lopuille asiakkailleen. Tällä tavalla kehitystiimi näkee kuinka uudet ominaisuudet toimivat pienemmällä määrällä käyttäjiä. [5]

Käyttäjäjoukkoa voidaan jakaa erilaisilla tavoilla. Käyttäjät voidaan jakaa esimerkiksi maantieteellisen sijainnin mukaan. Uusi versio voidaan ottaa käyttöön ensin Suomessa oleville käyttäjille ja jos se toimii virheittä, voidaan uusi versio ottaa käyttöön asteittain muuallakin. Canary deployment vaatii tavan jakaa käyttökuormaa suunnitellusti uuden ja vanhan version välillä [9].

Kuvassa 3.2 käyttökuorman jakaja jakaa käyttäjät kolmeen osaan ja verkkosovelluksen uusi versio on otettu käyttöön vain kolmasosalla käyttäjistä. Jos uusi versio toimii odotetusti, voidaan se ottaa seuraavaksi käyttöön kahdelle kolmasosaa käyttäjistä ja lopulta kaikille.



Kuva 3.2: Canary deployment. Uusi versio on otettu käyttöön yhdellä kolmasosalla käyttäjistä. Mukailtu lähteestä [9].

Canary deployment -strategian etuna on se, että mahdolliset ongelmat havaitaan jo varhaisessa vaiheessa. Tällöin verkkosovelluksen kaikki käyttäjät eivät koe virheiden vaikutuksia. Tämän strategian avulla saadaan myös testattua sovellusta oikeassa käytössä mahdollisimman pienellä riskillä. Haasteena kuitenkin on se, että tarvitaan monimutkaista liikenteen hallintaa ja tarkka seurantajärjestelmä, jotta virhetilanteisiin pystytään reagoimaan nopeasti ja tehokkaasti. [8]

4 Rollback-strategian automatisointi

Nykyisen nopean web-kehityksen myötä nopea reagointi virheisiin on tärkeää kilpailukykyisen tuotteen tai palvelun saavuttamiseksi. Ihminen on altis inhimillisille virheille, mutta kone ei. Tehokas tapa toteuttaa nopeasti reagoiva rollback-strategia on siis implementoida mukaan automaatio. Tässä luvussa paneudutaan erityisesti automaattiseen rollback-strategiaan. Luvussa analysoidaan pintapuolisesti yhtä mahdollista tapaa toteuttaa automaattinen strategia ja tarkastellaan siihen käytettyjä työkaluja sekä niiden roolia kyseisessä strategiassa.

Automaattinen rollback-strategia toteuttaa nopean ja automaattisen siirtymisen takaisin verkkosovelluksen edelliseen toimivaan versioon ilman ihmisen toimia. Tämä tapahtuu etukäteen CI/CD-putkeen konfiguroitujen komentosarjojen ja mekanismien avulla, jotka laukeavat tiettyjen ehtojen täyttyessä. Automaattisen rollback-strategian toteuttamiseen tarvitaan versionhallintatyökalu, monitorointityökalu ja jokin tapa toteuttaa automaatio. Automaation suurimpina etuina on sen nopeus ja kyky seurata sovelluksen tilaa jatkuvasti. Automaattinen rollback-strategia vaatii kuitenkin erittäin tarkkaa suunnittelua ja läpikotaista testaamista, jotta vältytään ei-halutuilta seuraamuksilta, kuten datan katoamiselta tai ongelman laajenemiselta. [8]

4.1 Komponentit ja tekniset työkalut

Seuraavaksi tarkastellaan kolmea vaihtoehtoista työkalua ja niiden roolit automaattisen rollback-strategian toteuttamisessa. Tässä rollback-strategiassa GitLab toimii versionhallintatyökaluna ja isännöi CI/CD-putkea. Kubernetes orkestroi verkkosovellusta tuotannossa ja tarjoaa rollback-mekanismeja. Prometheusta taas käytetään monitorointijärjestelmänä ja se laukaisee hälytyksiä, jos sovellus ei toimi odotetusti. Automaattisen rollback-strategian ja sen arkkitehtuurin toteuttamiseen on useita eri tapoja ja seuraavaksi analysoidaan yhtä mahdollista toteutusta.

GitLab

Git on hajautettu versionhallintatyökalu, jossa jokaisella kehittäjällä on täydellinen kopio projektin historiasta, mikä mahdollistaa paikallisen työskentelyn ja tehokkaan yhteistyön ilman jatkuvaa yhteyttä palvelimeen. Sen avulla voidaan seurata koodimuutoksia, mahdollistaa yhteistyö kehittäjien välillä ja hallita projektin versioita. Gitin käytössä yksi keskeisimmistä osista ovat haarat (engl. branches). Oletuksena projektissa on yksi päähaara nimeltä ”main”. Kehittäjät voivat luoda uusia haaroja eli kopioita päähaarasta kokeillakseen uusia ominaisuuksia tai korjatakseen virheitä ilman, että muutoksia tehdään suoraan päähaaraan. Tällöin ei tarvitse pelätä, että koko jo toimiva koodi hajoaisi esimerkiksi uusia ominaisuuksia kehitettäessä. Kun haaran kehitys on valmis, sen muutokset voidaan yhdistää päähaaraan. [10]

GitLab on avoimen lähdekoodin alusta, joka tarjoaa yhdistetyn ympäristön koodin versionhallintaan ja jatkuvaan integraatioon (CI) sekä jatkuvaan toimitukseen (CD). GitLabin CI/CD-ominaisuudet mahdollistavat tehokkaan koodin rakennuksen, testaamisen ja käyttöönoton. Sen avulla tiimin on helppo kommunikoida ja yhdessä työstää yhteistä lähdekoodia, mikä mahdollistaa ketterän kehityksen. [11] GitLabin avulla jokainen muutos lähdekoodiin voidaan laittaa kulkemaan ennalta konfiguroitujen testien läpi automaattisesti ennen kuin se integroidaan päähaaraan.

GitLab voi myös toimia alustana, johon on mahdollista integroida muita tarvittavia työkaluja [11].

Tässä luvussa tarkasteltavassa automaattisessa rollback-strategiassa GitLabin roolina on mahdollistaa kehittäjien yhteistyö, toimia hajautettuna versionhallintatyökaluna ja olla myös alustana projektin CI/CD-toiminnoille [12]. Kehittäjät voivat jatkuvasti integroida pieniä muutoksia kerrallaan lähdekoodiin. Automaattiset yksikkötestit varmistavat koodin pienimpien osien toimivuuden, kun taas end-to-end-testit takaavat kokonaisuuden yhteensopivuuden ja käyttäjäkokemuksen laadun [13]. Lähtökohtaisesti virheitä ei pitäisi syntyä, mutta joskus niin kuitenkin käy. GitLab-projektin yhdistäminen Kuberneteksen kanssa onnistuu käyttämällä GitLabin Cluster Management Project -mallia [12].

Kubernetes

Kubernetes on avoimen lähdekoodin ohjelmistojärjestelmä, joka automatisoi konttipohjaisten sovellusten käyttöönoton, skaalauksen ja hallinnan [14]. Konttipohjaisuus tarkoittaa erityistä ympäristöä, joka sisältää kaiken tarvittavan sovelluksen suorittamiseen, kuten lähdekoodin, kirjastot ja riippuvuudet. Kontteja on helppo siirtää paikasta toiseen ilman että sovellusta tai sen asetuksia, kirjastoja ja riippuvuuksia tarvitsee muokata. Sovellus toimii siis samalla tavalla ympäristöstä riippumatta. Kontit ovat kuin kevyempiä versioita virtuaalikoneista. Ne eivät tarvitse omaa käyttöjärjestelmää ja ne sisältävät sovelluksen lisäksi vain välttämättömät riippuvuudet. [15]

Kubernetes koostuu klusterista, joka tarkoittaa solmujen muodostamaa yhdessä toimivaa ryhmää. Solmu taas on yksittäinen palvelin tai virtuaalikone. Kubernetes klusterissa on hallintasolmu ja työsolmu. Hallintasolmujen tehtävänä on ylläpitää klusterin tavoitetilaa ja hallita työsolmua, jossa itse sovellus pyörii konttina. Kubernetes vähentää manuaalisia prosesseja sovellusten julkaisussa ja skaalauksessa. Se

tukee myös tarvittaessa erilaisia rollback-strategioita, kuten canary deployment ja blue-green deployment. [14]

Tässä luvussa tarkasteltavassa automaattisessa rollback-strategiassa Kuberneteksen rooli on hallita verkkosovellusta tuotannossa ja tarjota rollback-mekanismi virheen havaitsemisen jälkeen. Kubernetes saa verkkosovelluksen lähdekoodin GitLab:sta. Kuberneteksen deployment-objekti on resurssi, jolla hallitaan sovellusten elinkaarta Kubernetes-klusterissa. Se hallitsee sovelluksen replikoita eli versiota. Kubernetes voidaan konfiguroida tallentamaan tietty määrä aiempia deployment-versioita. Deploymentin epäonnistuessa tai myöhemmin virheen ilmetessä rollback toteutetaan komennolla:

```
kubectl rollout undo deployment <deployment-tag>
```

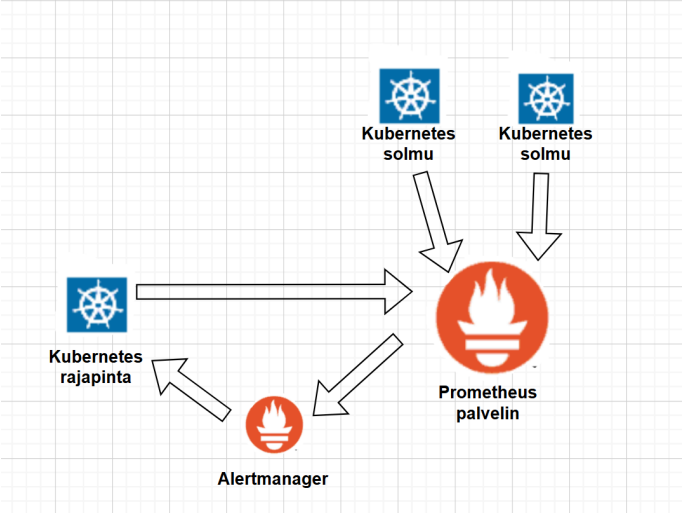
[16]

Prometheus

Prometheus on avoimen lähdekoodin monitorointi- ja hälytysjärjestelmä, joka on suunniteltu erityisesti pilviympäristössä toimivien sovellusten ja palveluiden tarpeisiin. Sillä voidaan kerätä tietoa erilaisista metriikoista, kuten resurssien käytöstä ja tallentaa ne tietokantaan. Prometheus query language (PromQL) on kyselykieli, joka mahdollistaa tehokkaan kyselyiden muodostamisen tallennettuun dataan. Automaattisen rollback-strategian kontekstissa tärkeimpänä ominaisuutena Prometheus tukee hälytyksiä sen Alertmanagerin avulla. Alertmanager on Prometheuksen yksi työkaluista, joka hallitsee ja reitittää hälytyksiä, joita Prometheus tuottaa valvonnan ja seurantadatan perusteella. Sille voidaan määritellä sääntöjä, joiden pohjalta se lähettää ilmoituksia eri kanaviin. [17]

Tässä tutkielmassa tarkasteltavassa automaattisessa rollback-strategiassa Prometheuksen rooli on havaita ongelmia, seurata palveluiden suorituskykyä ja varmistaa niiden luotettavuus [17]. Prometheus kerää siis metriikoita ja kun jokin ennalta

määritelty hälytysehto täyttyy sen Alertmanager reitittää hälytyksen ennalta konfiguroitujen kanavien avulla eteenpäin [18]. Kuvassa 4.1 Prometheus-palvelin on yhteydessä Kubernetes-klusteriin ja sen solmuihin rajapinnan avulla [19]. Alertmanager välittää hälytyksen ja laukaisee rollbackin toteuttavan komennon Kuberneteksessä.



Kuva 4.1: Yhteys Kuberneteksen ja Prometheuksen välillä. Muokattu lähteestä [19].

Tässä luvussa käsiteltiin yhtä mahdollista tapaa toteuttaa sellainen automaattinen rollback-strategia, joka mahdollistaa verkkosovelluksen nopean palauttamisen edelliseen toimivaan versioon ilman ihmisen manuaalisia toimia. Tärkeimpinä työkaluina on käytetty GitLabia, Kubernetesta ja Prometheusta. GitLab vastaa versiohallinnasta ja CI/CD-toiminnoista, Kubernetes huolehtii sovelluksen hallinnasta ja rollback-mekanismin toteutuksesta tuotannossa, kun taas Prometheus tarkkailee sovelluksen tilaa ja laukaisee rollbackin mahdollisissa ongelmatilanteissa. Automatisoinnin etuja ovat nopea reagointi ja jatkuva valvonta, mutta se vaatii tarkkaa suunnittelua ja huolellista testausta ei-toivottujen seurausten välttämiseksi. Strategian arkkitehtuuri saattaa tulla monimutkaiseksi ja se vaatii merkittävän määrän teknistä osaamista.

5 Yhteenveto

Tutkielmassa tarkasteltiin rollback-strategioita ja niiden roolia nykypäiväisessä web-kehityksessä erityisesti DevOps- ja CI/CD-periaatteiden kontekstissa. Tutkielmassa keskityttiin vastaamaan kolmeen päätutkimuskysymykseen. TK1 Mikä on rollback-strategioiden merkitys web-kehityksessä? TK2 Mitä erilaisia rollback-strategioita on? TK3 Mitä työkaluja rollback-strategian automatisointiin tarvitaan?

Toisessa luvussa vastattiin tutkimuskysymykseen TK1. Luku osoittaa, että rollback-strategiat ovat keskeinen osa nykypäiväistä nopean tahdin web-kehitystä. Ne mahdollistavat virheisiin nopean reagoinnin ja verkkosovellusten palauttamisen toimivaan tilaan käyttökatkot minimoiden. Tämä parantaa verkkosovellusten käyttökokemusta, joka on tärkeää kilpailukyvyn takaamiseksi. Nopea reagointi virheisiin on myös erityisen tärkeää tilanteissa, joissa verkkosovellusten käyttökatkot voivat aiheuttaa merkittäviä taloudellisia tappioita tai mainehaittoja yritykselle.

Kolmannessa luvussa vastattiin tutkimuskysymykseen TK2. Luvussa esiteltiin kolme keskeistä rollback-strategiaa, jotka ovat manuaalinen rollback, blue-green ja canary deployment. Manuaalinen rollback on yksinkertaisin vaihtoehto, mutta se vaatii ihmisen toimia. Blue-green deployment minimoi käyttökatkot ylläpitämällä kahta identtistä sovellusympäristöä, joista toisessa pidetään varalla edellinen vakaa versio verkkosovelluksesta. Toisessa ympäristössä on käytössä uusi versio, josta mahdollisen virheen sattuessa voidaan siirtyä takaisin vanhaan vakaaseen ympäristöön. Canary deploymentissa sovelluksen uusi versio otetaan käyttöön kuormanjakajan

avulla vain pienelle osalle käyttäjistä asteittain. Jos virhe ilmenee, se vaikuttaa silloin vain pieneen osaan käyttäjistä. Tällöin vanha toimiva versio palautetaan kaikille käyttöön virheen korjaamisen ajaksi.

TK3 vastattiin neljännessä luvussa. Rollback-strategian automatisoitiin paneuduttiin tarkastelemalla siihen tarvittavia työkaluja. Tutkielmassa kuvattiin automaattinen rollback-strategia, jossa oli käytössä GitLab, Kubernetes ja Prometheus. GitLab toimii versiohallinnan ja CI/CD-putken alustana. Kubernetes vastaa verkkosovelluksen toiminnasta tuotantoympäristössä ja tarjoaa mahdollisuuden erilaisten rollback-mekanismien implementoimiseen. Prometheus seuraa ja valvoo sovelluksen toimintaa sekä laukaisee tarvittaessa sen Alertmanagerilla hälytyksiä, jotka käynnistävät automaattisesti rollback-prosessin. Automaattinen rollback-strategia tarjoaa nopean ja tehokkaan tavan reagoida virheisiin, mutta sen toteutus vaatii tarkkaa suunnittelua ja kattavaa testausta.

Tutkielma tarjoaa tarkastelun rollback-strategioiden merkityksestä ja automatisoinnista web-kehityksessä. Tutkielman tulokset ovat hyödyllisiä erityisesti verkkosovellusten nopean kehityksen ja julkaisuprosessin riskien hallinnassa, mutta ne ovat hyviä käytäntöjä myös yleisesti ohjelmistokehityksessä.

Lähdeluettelo

[1] V. Panwar, "Web evolution to revolution: Navigating the future of web application development", *International Journal of Computer Trends and Technology*, vol. 72, nro 2, s. 34–40, 15. helmikuuta 2024, ISSN: 22312803. DOI: 10.14445/22312803/IJCTT-V72I2P107. url: https://www.ijcttjournal.org/archives/ijctt-v72i2p107.

[2] A. Agarwal, S. Gupta ja T. Choudhury, "Continuous and integrated software development using DevOps", teoksessa *2018 International Conference on Advances in Computing and Communication Engineering (ICACCE)*, Paris: IEEE, kesäkuu 2018, s. 290–293, ISBN: 978-1-5386-4485-0. DOI: 10.1109/ICACCE.2018.8458052. url: https://ieeexplore.ieee.org/document/8458052/.

[3] L. Leite, C. Rocha, F. Kon, D. Milojicic ja P. Meirelles, "A survey of DevOps concepts and challenges", *ACM Computing Surveys*, vol. 52, nro 6, s. 1–35, 30. marraskuuta 2020, ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3359981. url: https://dl.acm.org/doi/10.1145/3359981.

[4] N. Kerzazi ja B. Adams, "Botched releases: Do we need to roll back? empirical study on a commercial web app", teoksessa *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Suita: IEEE, maaliskuu 2016, s. 574–583, ISBN: 978-1-5090-1855-0. DOI: 10.

1109/SANER.2016.114. url: http://ieeexplore.ieee.org/document/7476676/.

[5] M. Shahin, M. Ali Babar ja L. Zhu, "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices", *IEEE Access*, vol. 5, 2017, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2685629. url: http://ieeexplore.ieee.org/document/7884954/.

[6] H. van Merode, *Continuous Integration (CI) and Continuous Delivery (CD): a practical guide to designing and developing pipelines*. New York, NY: Apress, 2023, 422 s., ISBN: 978-1-4842-9227-3.

[7] F. Zampetti, C. Vassallo, S. Panichella, G. Canfora, H. Gall ja M. Di Penta, "An empirical characterization of bad practices in continuous integration", *Empirical Software Engineering*, vol. 25, nro 2, s. 1095–1135, maaliskuu 2020, ISSN: 1382-3256, 1573-7616. DOI: 10.1007/s10664-019-09785-8. url: http://link.springer.com/10.1007/s10664-019-09785-8.

[8] Yogeswara Reddy Avuthu, "Change management and rollback strategies using IaC in CI/CD pipelines", *International Journal of Science and Research Archive*, vol. 2, nro 1, s. 160–168, 30. huhtikuuta 2021, ISSN: 25828185. DOI: 10.30574/ijsra.2021.2.1.0037. url: https://ijsra.net/node/6605.

[9] A. Malhotra, A. Elsayed, R. Torres ja S. Venkatraman, "Evaluate canary deployment techniques using kubernetes, istio, and liquibase for cloud native enterprise applications to achieve zero downtime for continuous deployments", *IEEE Access*, vol. 12, s. 87 883–87 899, 2024, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2024.3416087. url: https://ieeexplore.ieee.org/document/10560002/.

[10] S. Chacon ja B. Straub, *Pro Git*, 2. painos. Apress, 2014, ISBN: 978-1-4842-0077-3. url: https://git-scm.com/book/en/v2.

[11] M. S. Arefeen ja M. Schiller, ”Continuous integration using gitlab”, *Undergraduate Research in Natural and Clinical Science and Technology (URNCST) Journal*, vol. 3, nro 8, s. 1–6, 11. syyskuuta 2019. DOI: 10.26685/urncst.152. url: https://urncst.com/index.php/urncst/article/view/152.

[12] ”GitLab Documentation”, viitattu 9. lokakuuta 2024. url: https://docs.gitlab.com/.

[13] ”DevOps test automation | GitLab”, viitattu 6. helmikuuta 2025. url: https://about.gitlab.com/topics/devops/devops-test-automation/.

[14] S. I. Shamim, J. A. Gibson, P. Morrison ja A. Rahman, ”Benefits, challenges, and research topics: A multi-vocal literature review of kubernetes”, 13. marraskuuta 2022. url: https://arxiv.org/abs/2211.07032.

[15] ”Docker Docs”, viitattu 7. maaliskuuta 2025. url: https://docs.docker.com/.

[16] ”Kubernetes Documentation | Kubernetes”, viitattu 28. tammikuuta 2025. url: https://kubernetes.io/docs/home/.

[17] M. Chakraborty ja A. P. Kundan, ”Prometheus”, teoksessa *Monitoring Cloud-Native Applications*. Berkeley, CA: Apress, 2021, s. 99–131, ISBN: 978-1-4842-6887-2 978-1-4842-6888-9. DOI: 10.1007/978-1-4842-6888-9_4. url: https://link.springer.com/10.1007/978-1-4842-6888-9_4.

[18] ”Overview | Prometheus”, viitattu 15. tammikuuta 2025. url: https://prometheus.io/docs/introduction/overview/.

[19] ”How to Install Prometheus on Kubernetes & Use It for Monitoring”, viitattu 6. helmikuuta 2025. url: https://phoenixnap.com/kb/prometheus-kubernetes.