SOVELLUKSEN KEHITYSPUTKI AVOIMEN LÄHDEKOODIN OHJELMISTOILLA

DevOpsin hyödyt sovelluskehityksessä

Tiia Lindell
Opinnäytetyö (AMK)
Kevät 2025
Tietojenkäsittelyn tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittelyn tutkinto-ohjelma

Tekijä(t): Tiia Lindell
Opinnäytetyön otsikko: Sovelluksen kehitysputki avoimen lähdekoodin ohjelmistoilla
Työn ohjaaja(t): Raili Simanainen
Työn valmistumislukukausi ja -vuosi: kevät 2025
Sivumäärä: 42 + 8 liitettä

Sovelluksia kehitetään tänä päivänä ketterästi ja nykyaikaisia teknologioita käyttäen. DevOps on kulttuurillinen ja toimintatapoihin liittyvä muutos, jonka perusperiaatteita ovat yhteistyön ja automaation lisääminen. DevOpsin menetelmiin kuuluvat muun muassa CI/CD eli jatkuva integraatio ja jatkuva julkaisu. CI/CD parantaa sovelluskoodin laatua, sillä se mahdollistaa koodin katselmoinnin ja testauksen nopeammassa syklissä ja pienemmissä osissa.

Tämän opinnäytetyön tavoitteena oli todentaa, että paikalliseen kehitykseen soveltuvilla teknologioilla on mahdollista rakentaa automaattinen kehitysputki, jossa on otettu huomioon sekä koodin laatuun että turvallisuuteen vaikuttavia asioita. Työssä käytettiin paikalliseen kehitykseen soveltuvia avoimen lähdekoodin ohjelmistoja kuten Tekton, Helm ja Kubernetes.

Työn tuloksena saatiin automatisoitu kehitysputki, jossa GitHubissa sijaitsevaan koodirepositorioon tehdyt muutokset käynnistivät Tekton-pohjaisen kehitysputken. Kehitysputki suoritti sovelluksen koonnin sekä ajoi automaattisesti testit, testikattavuuden laskennan ja haavoittuvuuksien tarkastuksen. Viimeisenä vaiheena putki rakensi konttikuvan ja tallensi sen ulkoiseen repositorioon.

Kehitysputken luominen onnistui paikallisesti ja avoimen lähdekoodin ratkaisuilla. Lisäksi konttikuva latautui onnistuneesti Quay.io-repositorioon. On siis mahdollista panostaa automaatioon ja laatuun myös paikallisessa kehityksessä. Testatut teknologiat ovat pilviyhteensopivia, joten niitä voidaan seuraavaksi kokeilla myös valmiissa pilviympäristöissä.

ABSTRACT

Oulu University of Applied Sciences
Degree Program in Business Information Systems

Author(s): Tiia Lindell
Title of thesis: Development pipeline using open-source technologies
Supervisor(s): Raili Simanainen
Term and year when the thesis was submitted: spring 2025
Number of pages: 42 + 8 appendices

DevOps and CI/CD are modern Agile methodologies in current software development. They emphasize that development should be done in small iterations and that testing, and quality control should also be included in the process.

In this thesis, a simple CI/CD pipeline was created, using open-source technologies. Emphasis was placed in automation. Technologies used included Tekton, Helm and Kubernetes.

Creating the pipeline was successful. Automation and tasks for testing and vulnerability scanning were included in the pipeline. In conclusion, using open-source technologies in a CI/CD pipeline was feasible in local development. Future effort should focus on testing these technologies in cloud environments.

SISÄLLYS

TIIVISTELMÄ 2
ABSTRACT 3
SISÄLLYS 4
SANASTO 5
1 JOHDANTO 8
2 MODERNIT SOVELLUSKEHITYKSEN TEORIAT 9
2.1 DevOps 9
2.2 CI/CD-menetelmät 11
2.3 GitOps 11
3 TEKNOLOGIAT 13
3.1 Konttiteknologiat 13
3.2 Kubernetes 14
3.3 Minikube 16
3.4 Tekton 17
3.5 Helm 18
4 KEHITYSPUTKEN RAKENTAMINEN 20
4.1 Ympäristön perustaminen 20
4.2 Sovelluksen ja julkisen repositorion valmistelu 22
4.3 Kehitysputken resurssit 25
4.4 Testaus ja laadunvalvonta osana kehitysputkea 31
4.5 Paketointi Helmillä 33
4.6 Käynnistyskomennot ja testaus 34
5 AUTOMAATION ONNISTUMINEN 36
6 POHDINTA 37
LÄHTEET 38
LIITTEET 43

SANASTO

ArgoCD	Kubernetes-natiivi deklaratiivinen GitOps-työkalu
Chart	Helm chart, pakattavissa oleva kokoelma tiedostoja, jotka kuvaavat Kubernetes-resursseja
CI/CD	*Continuous integration/continuous delivery (deployment),* jatkuva integraatio ja jatkuva julkaisu tai asennus
ConfigMap	Kubernetes-objekti, jossa säilytetään sovelluksen tarvitsemia ympäristömuuttujia ja muita ympäristökohtaisia konfiguraatioita
DevOps	Ketterä periaate, joka painottaa yhteisöllisyyttä ja automaatiota
EventListener	Tekton Triggerin osa, joka kuuntelee saapuvia tapahtumia ja reagoi niihin
GitOps	DevOpsin oppeja hyödyntävä viitekehys, johon kuuluu muun muassa IaC
Helm	Kubernetes-natiivi pakkausohjelmisto
IaC	*Infrastructure as a Code*, periaate, jossa infrastruktuurin konfiguraatiot säilytetään versionhallinnassa
Jatkuva integraatio	*Continuous integration* (CI), CI/CD:n osa, jossa koodia ajetaan toistuvasti yhteiseen koodirepositorioon
Jatkuva julkaisu	*Continuous delivery* (CD), CI/CD:n osa, jossa koodi kootaan, testataan ja julkaistaan koodirepositorioon, mutta ei asenneta suoraan tuotantoympäristöön
Jatkuva asennus	*Continuous deployment* (CD), CI/CD:n osa, jossa koodi kootaan, testataan ja asennetaan automaattisesti myös tuotantoon

Kehitysputki	*Development pipeline*, CI/CD-periaatteen mukainen kokoelma toimintoja, joiden avulla sovelluskoodi kootaan ja testataan ja lopulta asennetaan asennusympäristöön
Klusteri	*Cluster*, joukko isäntäkoneita Kubernetes-ympäristössä
Kontti	*Container*, yksikkö, johon on sovelluskoodin lisäksi pakattu kaikki tarvittavat riippuvuudet ja kirjastot
Konttikuva	*Container image*, kontin rakennusohje
Kubectl	Kuberneteksen hallintaan käytettävä komentorivityökalu
Kubernetes	Avoimen lähdekoodin konttiorkestrointiteknologia
Minikube	Paikalliseen kehittämiseen tarkoitettu kevyt Kubernetes-klusteri
Nimiavaruus	*Namespace*, looginen kokonaisuus, jollaisiin Kubernetes-klusteri voidaan jakaa
Ohjaustaso	*Control plane*, Kuberneteksen klusterin osa, joka hoitaa klusterin ohjauksen
PersistantVolumeClaim	PVC, Kubernetes-objekti, joka varaa tietyn määrän klusterin muistista käyttöönsä
Pilvinatiivi	Pilvipalveluille optimoitu, esimerkiksi jokin ohjelmisto, joka hyödyntää pilviominaisuuksia tehokkaasti
Pipeline	Tektonin resurssi, jolla voi tehdä kehitysputken
Pipelinerun	Tektonin resurssi, joka antaa Pipelinelle tarvittavat tiedot ja käynnistää ajon
Platform engineering	Alustalähtöinen kehitys, jossa kehittäjä saa käyttöönsä alustan, jolla voi toteuttaa koko sovelluksen elinkaaren
Pod	Pienin ajettava Kubernetes-yksikkö, pod voi sisältää yhden tai useamman kontin

RBAC	*Role based access control*, roolipohjainen käyttäjähallinta Kuberneteksessa
RoleBinding	Kubernetes-objekti, joka antaa tietyn roolin oikeudet käyttäjälle
Secret	Kubernetes-objekti, jossa säilytetään sovelluksen tarvitsemia salattavia tietoja
ServiceAccount	Kuberneteksen tekninen käyttäjä, jolle voi antaa käyttäjäoikeuksia
Siiloutuminen	Yksiköiden erottuminen ja eristäytyminen tekemään tiukasti omaa tehtäväänsä
Solmu	*Node*, virtuaalinen tai fyysinen kone, joka on osa Kubernetes-klusteria
Step	Tektonin kehitysputken pienin rakenneosa, joka suorittaa jonkun toiminnon
Task	Tektonin kehitysputken rakenneosa, joka suorittaa jonkun toimintokokonaisuuden
Tekton	Kubernetes-natiivi CI/CD-putkiin tarkoitettu teknologia
TriggerBinding	Tekton Triggerin osa, joka, yhdistää tapahtumasta saadut tiedot Pipeline-parametreihin
TriggerTemplate	Tekton Triggerin osa, joka luo pohjan PipelineRunille
Vesiputousmalli	Kehitysmalli, jossa jokainen kehitysvaihe suoritetaan tiukasti järjestyksessä
YAML	Ihmisluettava serialisointikieli, jota käytetään konfiguraatioiden kirjoittamiseen

1 JOHDANTO

Sovelluksen matka lähdekoodista tuotantoon koostuu useista vaiheista. Pelkkä toimiva koodi ei riitä, vaan sen toimivuus pitää todentaa esimerkiksi yksikkötesteillä, staattisella koodianalyysillä tai haavoittuvuustarkastuksilla (Foster 7.6.2023; Watts 29.2.2024; OWASP Foundation 2025a.). Näitä vaiheita toistetaan, kunnes ongelmia ei enää ole ja silloin voidaan päättää sovelluksen täyttävän tuotantoon siirtymisen vaatimukset.

Sovelluskehitys voi olla hidasta ja vaatia paljon manuaalista työtä. Nykyaikana työn tehostaminen ja automaation hyödyntäminen ovat nousseet tärkeiksi tavoitteiksi (Bhatt 21.2.2024). Samaan aikaan kyberuhat ovat lisääntyneet ja muuttuneet entistä vakavammiksi, joten sovelluksen laadusta ei pitäisi tinkiä nopeamman tuloksen vuoksi. Haavoittuvuudet voivat johtaa merkittäviin tietoturvaloukkauksiin, joilla on pitkäaikaisia vaikutuksia (Singh 3.2.2023).

DevOps on ketteryyteen perustuva metodologia, jonka tärkeimpiä periaatteita ovat automaatio ja yhteistyön tekeminen. Sen avulla sovelluskehitystä pystytäänkin tehostamaan merkittävästi (Leite, Rocha, Kon, Milojicic & Meirelles 2020; Amazon 2025). DevOpsia voi ottaa käyttöön monella eri tavalla, yhtä oikeaa ratkaisua ei ole.

Tässä opinnäytetyössä esitellään DevOpsin periaatteita ja automatisoidun kehitysputken mahdollisuuksia. Kehitysputkella voidaan mahdollistaa laadukkaan, turvallisen koodin tuottaminen sovelluksiin. Olisi mahdollista tehdä tällainen kehitystyö myös kaupallisen palveluntuottajan pilvipalveluun perustuvalla alustalla. Tällöin kehitysympäristö tarjotaan valmiina, mutta kehitysputkeen olisi mahdollista käyttää samoja tässäkin työssä esiteltyjä teknologioita (kts. Red Hat 2025).

Työn tavoitteena on tuottaa esimerkkitoteutus automaattisesta kehitysputkesta. Samalla osoitetaan, että DevOpsin periaatteita voidaan ottaa käyttöön myös avoimen lähdekoodin teknologioilla.

2 MODERNIT SOVELLUSKEHITYKSEN TEORIAT

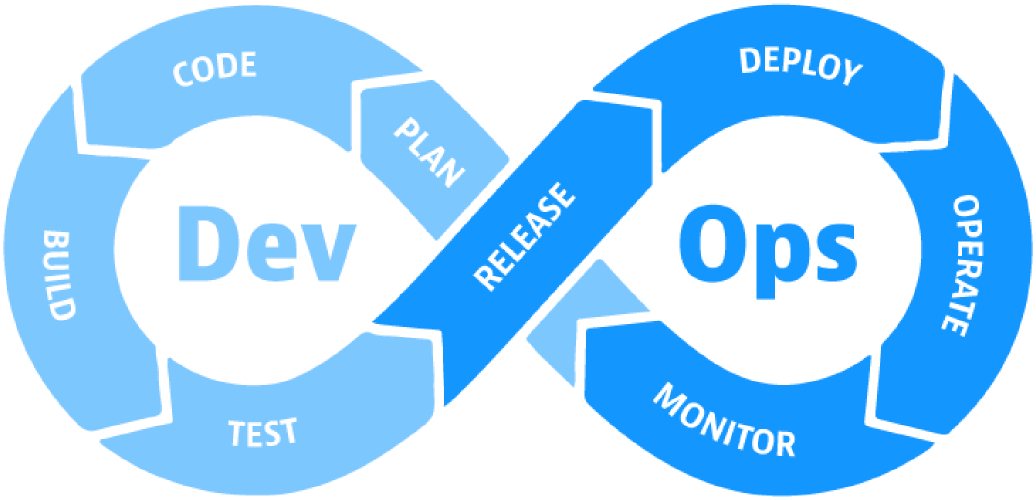
Muutama vuosikymmen sitten sovelluskehitysprojektit toteutettiin yleensä vesiputousmallilla, jossa jokainen vaihe seurasi toisiaan tiukassa järjestyksessä. Vaiheet suunniteltiin etukäteen, eikä projektin aikana voinut palata aiempiin vaiheisiin tekemään muutoksia. Näin jälkikäteen tarkasteltuna malli oli varsin jäykkä ja joustamaton. (Atlassian 2025.)

Modernit sovelluskehityksen teoriat ovat syntyneet tarpeesta tehdä sovelluskehitystä tehokkaammin. Ensimmäinen ketterän kehityksen malli oli Agile, joka sai alkunsa vuonna 2001 julkaisusta nimeltä Agile Manifesto. Manifestissa määritettiin arvoja, joissa korostettiin vuorovaikutusta, yhteistyötä ja muutoskyvykkyyttä (Agile Manifesto 2001). Tämä julistus aloitti ketterien menetelmien aikakauden ja vesiputousmalli alkoi vähitellen väistyä (Robinson, Brush & Silverthorne 2024).

2.1 DevOps

DevOps on kulttuurillinen ja toimintatapoihin liittyvä muutos, jonka tavoitteena on poistaa sovelluskehityksen turhia rajoitteita ja lisätä automaatiota sekä tiimien yhteistyötä (Amazon 2025). DevOps on saanut vaikutteita Agilesta ja siksi se pyrkiikin poistamaan kehittämisen jäykkyyttä, esteitä ja hidasteita (Leite ym. 2020). Parhaassa tapauksessa DevOpsin käyttöönotto johtaa parempaan tehokkuuteen sovelluskehityksessä.

DevOps-termi on muodostettu sanoista kehitys (*development*) ja ylläpito (*operations*). DevOpsin yksi tarkoituksista onkin vähentää yksiköiden välistä siiloutumista eli eriytymistä tiukasti omiin yksiköihin. Yhdistämällä kehittäjiä ja ylläpitäjiä samoihin tiimeihin saadaan tehostettua kehitystä. (Gunja 16.2.2023; Amazon 2025.)



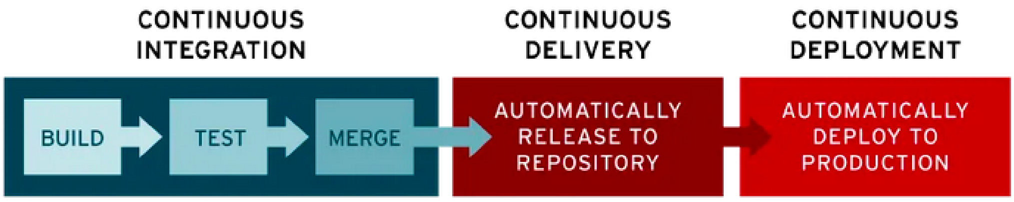
KUVA 1. DevOpsin vaiheet (Gunja 16.2.2023)

DevOpsin keskeiset periaatteet liittyvä ketteryyteen ja automaation lisäämiseen. CI/CD tarkoittaa jatkuvaa integraatiota ja jatkuvaa julkaisua. CI/CD automatisoi kehityksen putkeksi, joka mahdollistaa pienet, useasti toistuvat päivitykset suoraan testi- tai tuotantoympäristöön. Infrastruktuuri koodina eli IaC on periaate, jonka mukaan järjestelmäkonfiguraatiot tulisi säilyttää versionhallinnassa helppoa seurantaa ja palautusta varten. Teknisten käytäntöjen näkökulmasta suositellaan mikropalveluarkkitehtuuria vaihtoehdoksi suurille monoliittisille järjestelmille. Kun sovellus jaetaan pienempiin, toisistaan riippumattomiin osiin, niiden ylläpito ja kehittäminen on huomattavasti joustavampaa. Myös sovellusten jatkuva monitorointi ja lokitietojen kerääminen ovat tärkeässä roolissa, sillä ne mahdollistavat järjestelmän tilan reaaliaikaisen seurannan ja auttavat havaitsemaan mahdolliset ongelmat ennen kuin ne näkyvät käyttäjille. (Amazon 2025.)

DevOpsin toteuttamista tukee myös joukko teknologioita. Tällaisia ovat esimerkiksi avoimen lähdekoodin CI/CD-teknologiat Tekton ja Jenkins tai sovelluskoodin laadun varmistukseen käytettävä SonarQube, josta on olemassa myös ilmainen yhteisökehitteinen versio (Raja Ravi Varman 5.6.2024; SonarSource SA 2025).

2.2 CI/CD-menetelmät

Jatkuva integraatio ja jatkuva julkaisu tai myös asennus eli CI/CD on periaate, joka mahdollistaa sovelluksen julkaisun automaattisesti kehitysputken avulla. Jatkuva integraatio tarkoittaa sitä, että koodimuutoksia ajetaan toistuvasti jaettuun yhteiseen repositorioon. Jatkuva julkaisu tarkoittaa koodin kokoamista, testaamista ja julkaisua johonkin koodirepositorioon. Julkaisun lisäksi käytössä on termi jatkuva asennus, jossa julkaisu tehdään valmiiksi ja lisäksi asennetaan sovellus automaattisesti tuotantoympäristöön. (Red Hat 12.12.2023.)



KUVA 2. CI/CD-periaate (Red Hat 12.12.2023)

Kehitysputki on CI/CD-periaatteeseen perustuva toistettava automatisoitu prosessi, jonka avulla sovellusversio voidaan koota, testata ja ottaa käyttöön hallitusti. Putki koostuu useista peräkkäisistä vaiheista, kuten lähdekoodin hakemisesta, sovelluksen kokoamisesta, automaattisesta testauksesta ja lopulta julkaisusta ja asennuksesta testi- tai tuotantoympäristöön. (Red Hat 28.2.2025.)

Kehitysputken toteuttamiseen on saatavilla useita teknologioita, esimerkiksi Jenkins ja Tekton. Näistä Tekton on Kubernetes-natiivi moderni työkalu ja Jenkins palvelinpohjainen vanhempi teknologia (Raja Ravi Varman 5.6.2024).

2.3 GitOps

GitOps on viitekehys, joka hyödyntää DevOpsin oppeja infrastruktuurin hallintaan. GitOpsiin kuuluu periaate, jonka tavoitteena on säilyttää infrastruktuurin konfiguraatiot versionhallinnassa. Tätä kutsutaan nimellä *Infrastructure as Code* tai lyhennettynä IaC. Käytännössä esimerkiksi Git-repositorion sisältämiä koodeja voidaan käyttää luomaan jonkin ajoalustan kaikki konfiguraatiot ilman tarvetta käyttää erillisiä käyttöliittymiä. Tämä mahdollistaa versionhallinnan ja ympäristön palautuksen häiriötilanteessa. (Red Hat 27.3.2025.)

ArgoCD on avoimen lähdekoodin työkalu, joka toteuttaa deklaratiivista GitOpsia ja on yhteensopiva Kuberneteksen kanssa. Se seuraa Git-repositorion tilaa synkronoiden automaattisesti konfiguraatiot ja resurssit haluttuun tilaan. Konfiguraatiot voivat olla esimerkiksi YAML- tai JSON-muodossa tai pakattuina Helm chartteina. (Argo CD s.a..)

Käytännössä Argo CD mahdollistaa myös esimerkiksi CI/CD:n mukaisten kehitysputkien luonnin suoraan versionhallinnasta. Tällä tavoin kehitysputkien ja muiden resurssien asennus helpottuu, kun Argo CD luo tarvittavat resurssit automaattisesti.

3 TEKNOLOGIAT

DevOpsin mukaiseen kehitykseen liittyviä avoimen lähdekoodin teknologioita on paljon. Kaiken perustana on konttiteknologia. Kontit mahdollistavat luonnollisesti syntyvän ketterän mikropalveluarkkitehtuurin. Kubernetes taas toimii konttien käskyttäjänä ja tuo niiden hallintaan automaatiota. Tektonin muunneltavuus tekee siitä ketterän CI/CD-työkalun ja Helm pakkaa resurssit helppoon muotoon. Avoimen lähdekoodin ohjelmistoilla on mahdollista toteuttaa CI/CD-periaatteen mukainen kehitysputki alusta loppuun saakka.

3.1 Konttiteknologiat

Kontti on yksikkö, johon on pakattu sovellus sekä kaikki sen toimintaan tarvittavat riippuvuudet. Koska kontti sisältää kaiken tarvittavan, se voidaan suorittaa eri alustoilla ilman lisäkonfigurointia. Tämän ansiosta kontit ovat helposti siirrettävissä ympäristöstä toiseen. Kontit jakavat käyttöjärjestelmän ytimen ja muita yhteisiä kirjastoja, mikä tekee niistä kevyempiä kuin perinteiset virtuaalikoneet. Tämän vuoksi kontit käynnistyvät nopeasti ja kuluttavat vähemmän resursseja. Kontit ovat myös toisistaan eristettyjä, joka parantaa niiden vikasietoisuutta ja tietoturvaa. (Sunjara & Smalley 20.5.2024.)

Konttikuva on malli, jolla varsinainen kontti voidaan muodostaa. Näitä konttikuvia voidaan tallentaa erillisiin repositorioihin, joista niitä voidaan jakaa ja ottaa käyttöön. Julkiset konttikuvat tarjoavat myös valmiita pohjia, joiden päälle voi rakentaa omia sovelluksia. (Sunjara & Smalley 20.5.2024.)

Konttikuvalle voidaan antaa tunniste eli tag. Tagin avulla kuvaan voidaan yhdistää tieto esimerkiksi versionumerosta tai Gitin haarasta. Suositeltu tapa on käyttää uusimmassa kehitysversiossa tagia latest, johon pystyy aina viitata riippumatta version todellisesta numerosta. (Nirmalkushwah 30.6.2023.)

Konttiteknologia yleistyi vuonna 2013, jolloin avoimen lähdekoodin Docker kehitettiin (Sunjara & Smalley 20.5.2024). Docker on tälläkin hetkellä yleisesti käytetty konttiteknologia. Konttikuvia voidaan rakentaa myös muilla työkaluilla kuten Podman-komentorivityökalulla tai CI/CD-putkissa käytetyllä buildah-nimisellä Tekton Taskilla (Suttichujit 29.6.2021; Wilson 8.11.2022).

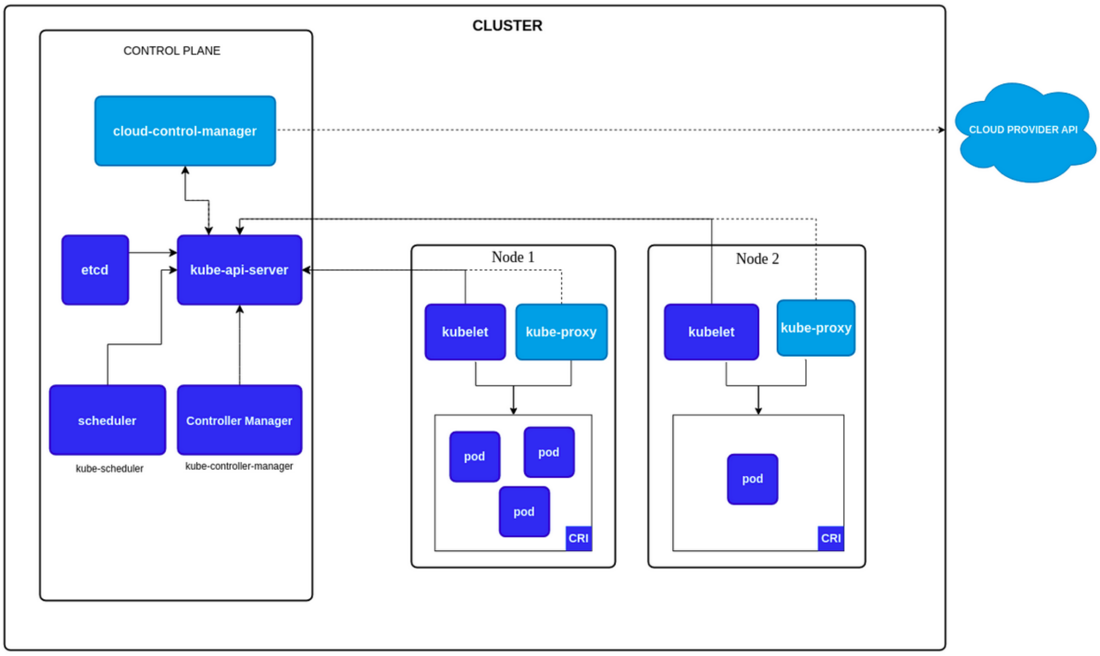
3.2 Kubernetes

Kubernetes on avoimen lähdekoodin alusta, joka on kehitetty konttien hallintaan ja orkestrointiin. Kubernetes automatisoi useita konttien hallintaan liittyviä asioita, kuten asennuksen, skaalauksen ja resurssienhallinnan. Kubernetes on mahdollista ottaa käyttöön monilla alustoilla kuten virtuaalikoneilla, fyysisillä palvelimilla, pilvipalveluissa tai näiden yhdistelmissä. DevOps-käytäntöjen mukaisesti Kubernetes tukee myös CI/CD-putkia. (Red Hat 18.12.2024.)

Kubernetes mahdollistaa konfiguraatioiden ja resurssien tekemisen IaC-periaatteen mukaisesti (Palo Alto Networks 2025). Konfiguraatiot voidaan esimerkiksi tehdä YAML-muotoisina, jolloin ne voidaan säilyttää versionhallinnassa ja palauttaa tarvittaessa.

Kubernetes-ympäristö muodostuu klusterista, joka koostuu kontteja suorittavista isäntäkoneista. Klusterissa on ohjaustaso, joka vastaa klusterin hallinnasta, sekä joukko solmuja, joissa varsinaiset sovelluskuormat sijaitsevat. Solmut ovat yleensä erillisiä virtuaalikoneita. Kukin solmu sisältää podeja, jotka voivat sisältää yhden tai useampia kontteja. Pod on pienin Kuberneteksen hallittava yksikkö, joita on paljon käynnissä jatkuvasti erilaisten resurssien käynnistämänä. (Red Hat 18.12.2024.)

Kubernetes-klusteri voidaan jakaa nimiavaruuksiin, jotta sovelluskokonaisuuksia olisi helpompi hallita (Cicero 16.3.2021). Nimiavaruudet ovat käytössä yleensä suuremmissa organisaatioissa, joissa tehdään useita eri projekteja.



KUVA 3. Esimerkki Kubernetes-klusterista (Garcia 8.8.2024)

RBAC eli roolipohjainen käyttäjähallinta mahdollistaa erilaisten roolien luomisen halutun tason mukaisesti. Rooli voi olla nimiavaruuskohtainen Role tai koko klusterin laajuinen ClusterRole. Roolit sidotaan käyttäjiin RoleBinding-objekteilla. Käyttäjät voivat olla yksittäisiä, ryhmiä tai ServiceAccount-tyyppisiä. ServiceAccount on tekninen tunnus, jolle voidaan antaa samanlaisia rooleja kuin muillekin käyttäjille. ServiceAccount on käytössä esimerkiksi silloin kun kirjaudutaan konttikuvarepositorioon. (Kubernetes Authors 19.11.2024a.; Kubernetes Authors 9.4.2025.)

Sovelluksen tarvitsemia parametreja voidaan konfiguroida klusterissa ConfigMap- ja Secret-objekteilla. ConfigMapiin tallennetaan esimerkiksi ympäristömuuttujia ja Secretiin nimensä mukaisesti salattavia asioita eli salasanoja ja avaimia. Kuberneteksen PersistentVolume-tallennustila mahdollistaa datan jakamisen podien kesken tai säilyttää datan podin uudelleenkäynnistyksen hetkellä. PVC eli PersistentVolumeClaim on käyttäjän tekemä varaus käyttää tuota muistia. (Kubernetes Authors 19.11.2024b.; Kubernetes Authors 18.3.2025.)

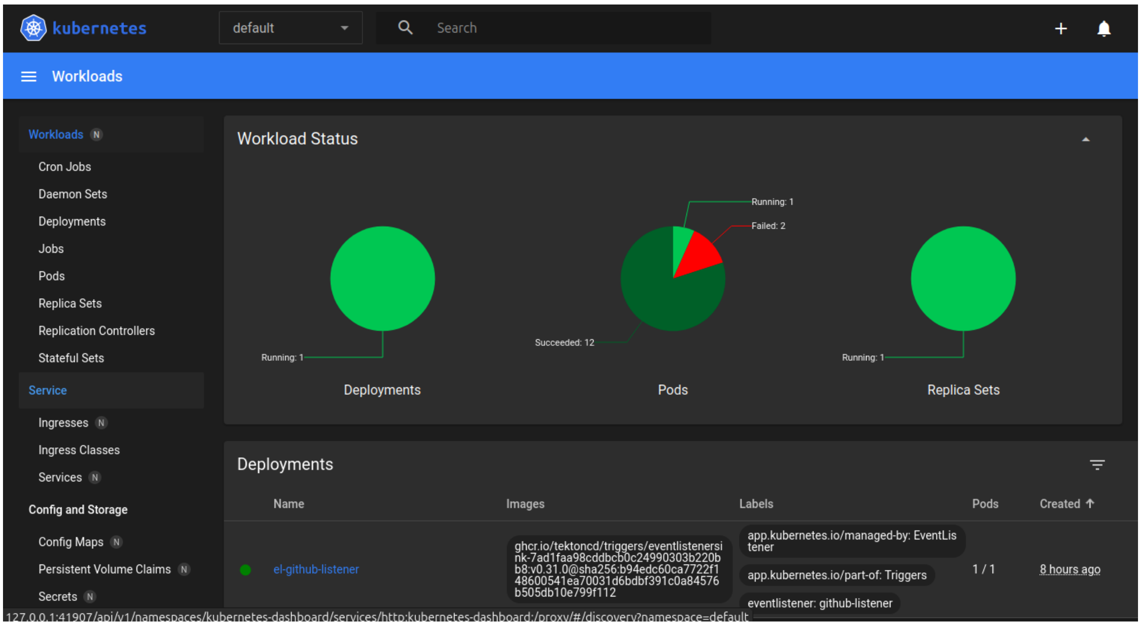
Klusteria hallitaan tehokkaimmin kubectl-komentorivityökalulla. Joissakin tuotteistetuissa Kubernetes-versioissa on graafinen käyttöliittymä, jota kautta klusteria voidaan hallinnoida (Red Hat 2025).

3.3 Minikube

Kuberneteksen asennus voi olla monimutkaista ja siihen kuluu aikaa. Kuberneteksen käyttöön paikallisessa kehityksessä on kuitenkin olemassa keveämpiä työkaluja. Näistä yksi on Minikube, joka on avoimen lähdekoodin toteutus. Minikube luo virtuaalikoneen, johon se asentaa tuotannonkaltaisen Kubernetes-klusterin. Minikubella Kuberneteksen perusteet on helppo opetella. (Isaiah 24.3.2025.)

Minikube on helppo ottaa käyttöön ja eikä se vaadi kohtuuttoman paljon resursseja tietokoneelta. Lisäksi se on mahdollista asentaa monille eri käyttöjärjestelmille. Käyttöä varten tarvitaan kubectl-komentorivityökalu. (Minikube 15.1.2025.)

Minikubessa on myös graafinen käyttöliittymä, joka helpottaa opettelua ja Kuberneteksen osien hahmottamista (Isaiah 24.3.2025). Kuitenkin kubectl on joissakin tapauksissa kätevämpi ja nopeampi.



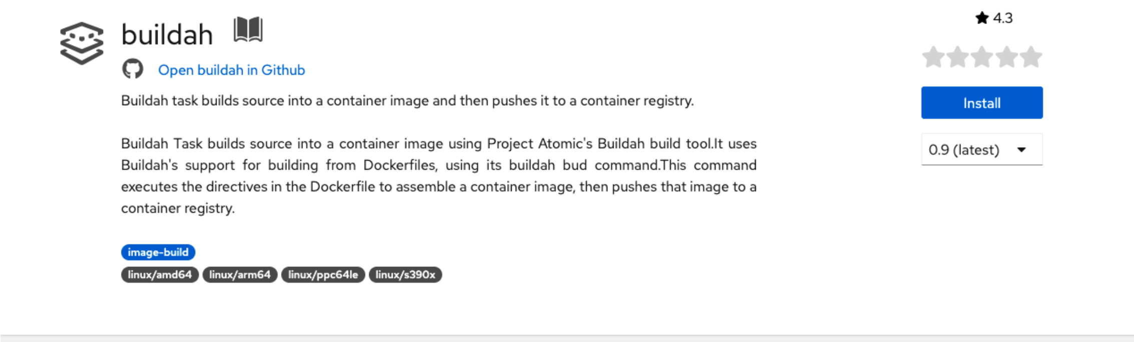
KUVA 4. Näkymä Minikuben dashboard-ikkunaan

3.4 Tekton

Tekton on avoimen lähdekoodin pilvinatiivi CI/CD-työkalu, joka voidaan asentaa Kubernetes-klusteriin. Keskeisin osa on Tekton Pipelines, jolla kehitysputkia rakennetaan. Tektonin resurssien hallintaan käytetään Tekton CLI -komentorivityökalua. (The Linux Foundation 9.3.2023.)

Tektonin resurssit luodaan YAML-tiedostoilla käyttäen ennalta määritettyä rakennetta. Kehitysputken perusta on Pipeline-objekti, joka suorittaa tarvittavat toiminnot järjestyksessä. Se koostuu tehtävistä eli Taskeista, jotka taas koostuvat pienemmistä Stepeistä. Taskien avulla voidaan tehdä hyvin erilaisia kehitysputkia ja tämä modulaarisuus onkin Tektonin tärkeä osa. (8grams 29.5.2023.)

Taskeja voi ladata valmiina Tekton Hubista, joka on Tektonin oma jakelukanava käyttäjien luomille yleisille Taskeille (The Linux Foundation 2025). Taskeja voi myös tehdä itse ja asentaa ne suoraan klusteriin. Yleisiä kehitysputkessa tarvittavia Taskeja ovat esimerkiksi git-clone, joka nimensä mukaisesti lataa lähdekoodin versionhallinnasta, ja buildah, joka rakentaa konttikuvan ja tallentaa sen ulkoiseen repositorioon.



KUVA 5. Buildah Task Tekton Hubissa (Tekton Hub 2025)

Pipelinen lisäksi Tekton-pohjainen kehitysputki tarvitsee myös PipelineRun-objektin. Kun Pipeline määrittelee, mitä tapahtumia putki suorittaa, PipelineRun antaa putkelle tarvittavat parametrit ja käynnistää ajon klusterissa. Käytännössä kehitysputki lähtee ajoon samalla hetkellä, kun PipelineRun asennetaan klusteriin.

Tekton-kokonaisuuteen kuuluu myös Tekton Triggers, joka mahdollistaa webhookien käytön Pipelinen ajon käynnistämiseksi. Tällöin esimerkiksi GitHubiin voidaan määritellä tapahtuma, esimerkiksi git push, joka lähettää Tektonille hyötykuorman, jossa on tarvittavat parametrit PipelineRunin generoimiseksi. Triggerin osia ovat EventListener, TriggerBinding ja TriggerTemplate. EventListener on Kubernetes-objekti, joka kuuntelee saapuvia tapahtumia ja reagoi niihin. TriggerBinding koostuu avain-arvopareista, jotka yhdistävät tapahtumasta saadut tiedot Pipeline-parametreihin. TriggerTemplate on pohja, josta PipelineRun muodostetaan. (8grams 29.5.2023.)

3.5 Helm

Helm on Kubernetes-natiivi paketointityökalu, jonka avulla useita resursseja voidaan asentaa yhdellä kertaa klusteriin (Helm Authors 2025a.). Ilman Helmiä jokainen resurssi tulisi erikseen asentaa klusteriin. Valmiita Helm chartteja on mahdollista ladata ulkoisista repositorioista, ja niitä voi myös käyttää toisten chartien riippuvuutena. Chartteja voi myös tehdä kokonaan itse. Helm chartin rakenne koostuu chartista, template-kansiosta ja values-tiedostosta. Chart kertoo versiotiedot ja siinä määritetään mahdolliset riippuvuudet. Template-kansion sisälle kootaan kaikki Kubernetes-resurssit, kuten Pipelinen ja Triggerin määrittävät tiedostot. Values-tiedostossa on parametreja, joita chart käyttää template-kansion tiedostojen luonnissa. (Cicero 16.3.2021.)

Helm chartit tuovat monia etuja kehitysputkien konfigurointiin. Chartien käyttöä voi automatisoida antamalla ne GitOps-periaatteen mukaisesti versionhallinnasta Argo CD:n hallintaan. Values-tiedostoja voi tehdä useita, jotta kehitysputki saadaan samalla chartilla useaan eri ympäristöön. Helm charteja voi käyttää riippuvuutena, joten on mahdollista määritellä esimerkiksi organisaation oma mallipohja, joka on helppo ottaa käyttöön vaikkei Helm chartit olisikaan ennestään tuttuja. Lisäksi Helm charteja voi hienosäätää Go-ohjelmointikielellä, jolloin charteista voi tehdä todella monikäyttöisiä (Cicero 16.3.2021).

```yaml
apiVersion: v2
name: oppari-helm
description: A Helm chart for general build

# A chart can be either an 'application' or a 'library' chart.
#
# Application charts are a collection of templates that can be packaged into versioned archives
# to be deployed.
#
# Library charts provide useful utilities or functions for the chart developer. They're included as
# a dependency of application charts to inject those utilities and functions into the rendering
# pipeline. Library charts do not define any templates and therefore cannot be deployed.
type: application

# This is the chart version. This version number should be incremented each time you make changes
# to the chart and its templates, including the app version.
# Versions are expected to follow Semantic Versioning (https://semver.org/)
version: 0.1.0

# This is the version number of the application being deployed. This version number should be
# incremented each time you make changes to the application. Versions are not expected to
# follow Semantic Versioning. They should reflect the version the application is using.
# It is recommended to use it with quotes.
appVersion: "1.16.0"
```

KUVA 6. Esimerkki Helm chartin Chart.yaml-tiedostosta

4 KEHITYSPUTKEN RAKENTAMINEN

Kehitystyössä suunniteltiin ja toteutettiin CI/CD-periaatteen mukainen paikalliseen kehitykseen soveltuva kehitysputki. Putkeen kuului lähdekoodin haku versionhallinnasta, sovelluksen kokoaminen, lähdekoodin testaus, haavoittuvuustarkastus ja tallennus ulkoiseen repositorioon. Kehitysputken tuotos oli konttikuva, joka olisi mahdollista julkaista johonkin ajoympäristöön. Aiheen rajaamisen vuoksi sovelluksen julkaisu jätettiin kuitenkin tästä työstä pois.

Työssä käytettävä sovellus sijaitsi GitHub-repositoriossa ja konttikuvan tallentamista varten otettiin käyttöön ilmaisversio Quay.io-repositoriosta. Alustaksi valittiin Minikube sen helppouden ja monipuolisuuden vuoksi. Koska käytössä oli Kubernetes, CI/CD-teknologiaksi oli luontevaa valita Tekton ja paketointiratkaisuksi Helm.

Kehitystyön vaiheet:

- Kehitysympäristön asennus
- Tekton-resurssien koostaminen
- Testauksen ja laadunhallinnan integroiminen
- Helm chartin rakentaminen ja asennus
- Kehitysputken ajaminen

4.1 Ympäristön perustaminen

Ympäristö asennettiin käyttöön kannettavalle työasemalle, jossa käyttöjärjestelmänä oli Ubuntu 24.04.2 LTS. Koneella oli jo valmiiksi QEMU-virtualisointityökalu, joka on yksi vaatimus Minikuben asennukselle. Muita virtualisointiratkaisuja ja käyttöjärjestelmäkohtaisia ohjeita löytyy Minikuben omasta dokumentaatiosta (Minikube 15.1.2025). Helm chartteja varten asennettiin helm-komentorivityökalu ohjeen mukaan (Helm Authors 2025b.).

Asennus suoritettiin ohjeen mukaan lataamalla ja asentamalla Minikuben binääritiedosto. Samalla asennettiin myös kubectl-komentorivityökalu klusterin hallintaa varten (Kubernetes Authors 8.8.2024). Kun tarvittavat ohjelmistot oli asennettu, käynnistettiin Minikube komentoriviltä. Ensimmäisellä käynnistyskerralla Minikube varasi tarvittavat resurssit ja loi klusteria varten virtuaalikoneen. Kun virtuaalikone ja klusteri olivat valmiita, käynnistettiin erillisellä komennolla myös Minikuben konsolinäkymä eli dashboard.

Tarvittavat komennot Minikuben käynnistämiseen komentoriviltä olivat:

minikube start
minikube dashboard

KUVA 7. Minikube-klusteri käynnistyy ensimmäistä kertaa

Seuraavaksi asennettiin Tekton Pipelines, Tekton dashboard ja Tekton Triggers. Tektonin Taskeja ei tehty itse, vaan ladattiin valmiit Taskit Tekton Hubista. Tarvittavat Taskit olivat: git-clone, maven ja buildah.

Tekton ja Tekton dashboard -asennukset komentoriviltä:

kubectl apply --filename https://storage.googleapis.com/tekton-releases/pipeline/latest/release.yaml

kubectl apply --filename https://storage.googleapis.com/tekton-releases/dashboard/latest/release.yaml

Tekton Trigger -asennukset komentoriviltä:

kubectl apply --filename \ https://storage.googleapis.com/tekton-releases/triggers/latest/release.yaml

kubectl apply --filename \ https://storage.googleapis.com/tekton-releases/triggers/latest/interceptors.yaml

Tekton Taskien asennukset komentoriviltä:

kubectl apply -f https://api.hub.tekton.dev/v1/resource/tekton/task/buildah/0.9/raw

kubectl apply -f https://api.hub.tekton.dev/v1/resource/tekton/task/maven/0.4/raw

kubectl apply -f https://api.hub.tekton.dev/v1/resource/tekton/task/git-clone/0.9/raw

Kaikki asennetut resurssit säilyvät Minikubessa niin kauan kuin klusteri on olemassa, joten väliaikainen sammuttaminen ei hävitä resursseja. Näin on kätevää, kun ympäristöä ei paikallisessa kehityksessä tarvitse luoda jatkuvasti uudelleen. Näiden asennusten jälkeen Minikube-ympäristö oli valmiina kehitysputken rakentamista varten.

4.2 Sovelluksen ja julkisen repositorion valmistelu

Sovellus sijaitsi julkisessa Github-repositoriossa, josta sen sai kloonattua ilman tunnistautumista. Kehitysputken testaamista varten tarvittava sovellus oli yksinkertainen Javalla ohjelmoitu SpringBoot-projekti, jonka koontiin on käytetty Apache Mavenia. Sovellus ei sisältänyt mitään liiketoimintalogiikkaa, sillä tärkeintä oli vain sovelluksen käännöksen onnistuminen. Sovelluksen lähdekoodi löytyy versionhallinnasta täältä:

https://github.com/tikkuinen/oppari-rest-app

Mavenilla kootuilla sovelluksilla on pom.xml-niminen tiedosto, joka sisältää konfiguraatioita ja tarvittavat riippuvuudet ja niiden versiot. Kehitysputken testit, testikattavuuden laskenta ja haavoittuvuustarkastus tehtiin Mavenin koonnin yhteydessä ja siksi niihin liittyvät lisäosat määriteltiin valmiiksi pom.xml-tiedostossa.

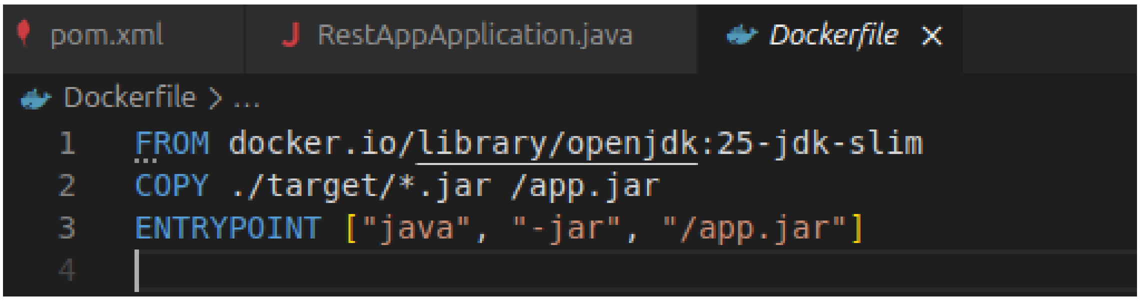
```xml
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.11.0</version>
            <configuration>
                <release>17</release>
            </configuration>
        </plugin>

        <plugin>
            <groupId>org.owasp</groupId>
            <artifactId>dependency-check-maven</artifactId>
            <version>6.1.1</version>
            <executions>
            <execution>
                <goals>
                <goal>check</goal>
                </goals>
            </execution>
            </executions>
        </plugin>

        <plugin>
            <groupId>org.jacoco</groupId>
            <artifactId>jacoco-maven-plugin</artifactId>
            <version>0.8.7</version>
            <executions>
            <execution>
                <goals>
                <goal>prepare-agent</goal>
                <goal>report</goal>
                </goals>
            </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

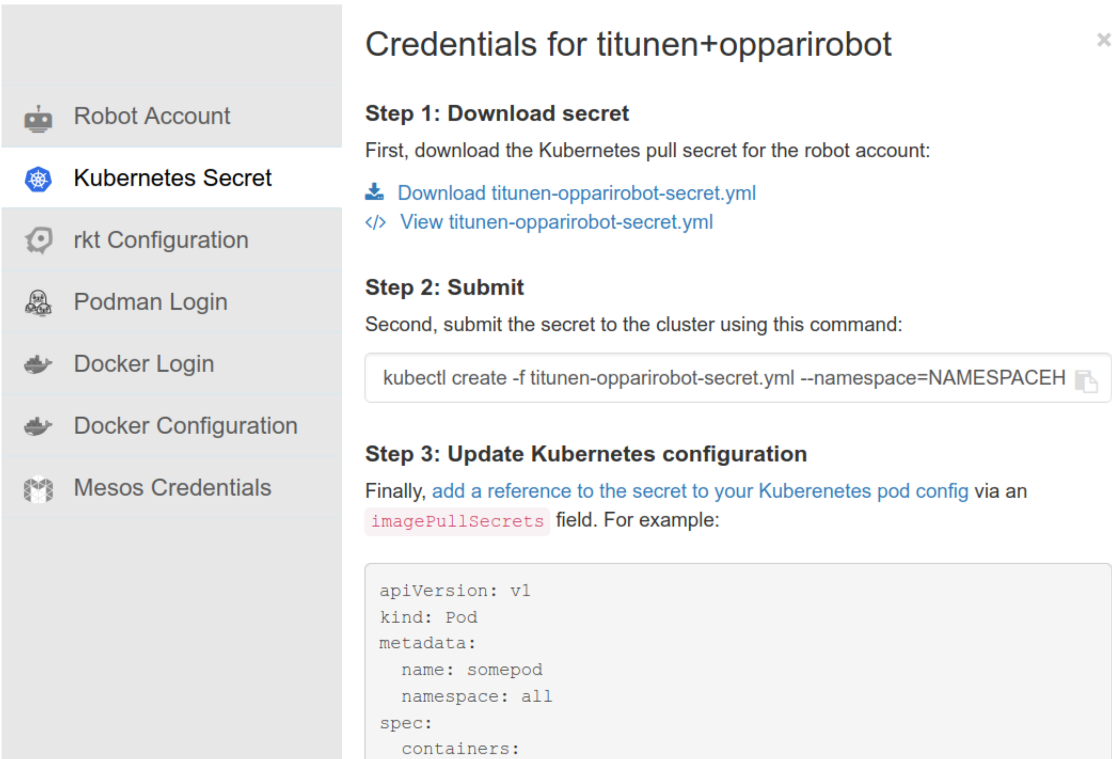
KUVA 9. Sovelluksessa määritellyt riippuvuudet pom.xml-tiedostossa

Konttikuvaa varten sovelluksen lähdekoodiin tuli sisällyttää Dockerfile, joka antaa ohjeet konttikuvan rakentamiselle. Dockerfile voisi olla laajempikin, mutta tässä työssä sen tehtävä on yksinkertainen. Tiedostossa määritetään, mitä konttikuvaa käytetään pohjana ja mistä ja miten sovelluksen koottu versio löytyy.



KUVA 8. Sovelluksen Dockerfile-tiedosto

Konttikuvan tallentamista varten tehtiin ilmainen tili Quay.io-repositorioon. Sinne luotiin robottikäyttäjä, jonka avulla Tekton saa kehitysputken ajon aikana yhteyden repositorioon. Tärkeää oli antaa tälle käyttäjälle oikeus tallentaa konttikuvia. Robottikäyttäjän asetuksista sai tunnistautumistiedot monessa eri muodossa, kuten suoraan Kuberneteksen Secret-objektina.



KUVA 10. Quay.io-repositorioon luodun robottikäyttäjän tunnukset

4.3 Kehitysputken resurssit

Tekton Pipeline

Kaikki kehitysputken objektit tehtiin YAML-muodossa, joka on yleinen konfigurointiin käytettävä kieli ja yhteensopiva Kuberneteksen kanssa. Kehitysputken kaikkien resurssien YAML-tiedostot löytyvät tämän opinnäytteen liitteistä.

Pipeline-objekti luotiin lisäämällä perustiedot ja Taskit. Muuttuvat tiedot tulivat Pipeline-objektille parametreina values-tiedostosta, loput voitiin kovakoodata. Tässä kehitysputkessa valittiin parametrisoiduiksi arvoiksi nimi, nimiavaruus ja Maven-konttikuvan versiotieto. Taskit kirjoitettiin suoritusjärjestyksessä ja ensimmäistä lukuun ottamatta jokaiseen määritettiin kenttä runAfter, joka määrää Taskien suoritusjärjestyksen. Ilman tuota kenttää kaikki Taskit olisivat lähteneet ajoon samaan aikaan.

Ensimmäinen Task on clone-repo, joka kloonaa lähdekoodin repositoriosta. Jokaisella Taskilla on oma nimi ja viittaus Taskiin, jota se käyttää. Esimerkiksi clone-repo käyttää Tekton Hubin Taskia git-clone. Tämän jälkeen maven-nimistä Taskia käyttävä build-jar kokoaa nimensä mukaisesti lähdekoodista Mavenilla jar-paketin. Tämä jar-paketti välitetään edelleen Taskille nimeltä build-and-push, jossa buildah rakentaa Dockerfilen perusteella konttikuvan ja lataa sen Quay.io-repositorioon. Näillä muutamalla Taskilla saatiin aikaan yksinkertainen Pipeline.

```yaml
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: {{ .Values.pipeline.name }}
  namespace: {{ .Values.pipeline.namespace }}
spec:
  params:
    - name: repo-url
      type: string
      description: Git repository
    - name: revision
      type: string
      default: "main"
      description: Branch to clone
    - name: image-url
      type: string
      description: Destination image
  workspaces:
    - name: shared-workspace
    - name: maven-settings

  tasks:
    - name: clone-repo
      taskRef:
        name: git-clone
        kind: Task
      params:
        - name: url
          value: "$(params.repo-url)"
        - name: revision
          value: "$(params.revision)"
      workspaces:
        - name: output
          workspace: shared-workspace
```

KUVA 11. Pipeline.yaml -tiedoston rakennetta

Koska resurssit pakattiin Helmillä, tehtiin Pipelinelle myös values-tiedosto. Näin kehitysputkea saatiin parametrisoitua. Values-tiedostossa määriteltiin aiemmin parametreiksi Pipelineen jääneet kehitysputken nimi, nimiavaruus ja Maven-versio.

! values.yaml

```yaml
pipeline:
  name: oppari-build-pipeline
  namespace: default
  mavenImage: maven:3.8.6-eclipse-temurin-17
```

KUVA 12. Values-tiedoston parametrit

Jotta kehitysputki pystyisi lataamaan rakentamansa konttikuvan Quay.io-repositorioon, sille oli annettava roolipohjaisia oikeuksia. Serviceaccount.yaml -nimiseen tiedostoon tallennettiin hieman aiemmin Quay.io-repositorioon luodun robottikäyttäjän ServiceAccount, jossa on viittaus tarvittavaan Secretiin.

```yaml
kind: ServiceAccount
apiVersion: v1
metadata:
  name: oppari-service-account
  namespace: default
secrets:
  - name: titunen-opparirobot-pull-secret
imagePullSecrets:
  - name: titunen-opparirobot-pull-secret
```

KUVA 13. Kuva serviceaccount.yaml -tiedostosta, jossa määritelty ServiceAccount robottikäyttäjälle

Viimeisenä kehitysputken osana luotiin PersistentVolumeClaim-objekti, jolla varataan tallennustilaa kehitysputken Taskien yhteiseen käyttöön. PVC mahdollistaa tiedon jakamisen kehitysputken aikana Taskien välillä. Viittaus PVC-objektin käyttöön tehdään yleensä PipelineRunissa, tässä kehitystyössä se tehtiin Triggerissä koska PipelineRun generoidaan automaattisesti.

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: oppari-pipeline-pvc
  namespace: default
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

KUVA 14. Määritelty PVC

Tekton Trigger

Tekton Trigger vaatii aina kolme resurssia, jotka ovat EventListener, TriggerBinding ja TriggerTemplate. EventListeneriin tehtiin viittaukset kahteen muuhun objektiin. TriggerTemplatessa on ohjeet luoda PipelineRun, joten siinä annetaan Pipelinessa tarvittavat parametrit. GenerateName-kenttä mahdollistaa uniikin nimen jokaiselle PipelineRunille. Kubernetes ei hyväksy samaan nimiavaruuteen kahta samanimistä resurssia, joten Trigger ei onnistuisi luomaan PipelineRunia ilman uniikkia nimeä.

```yaml
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: github-listener
spec:
  serviceAccountName: tekton-robot
  triggers:
    - name: github-push-trigger
      bindings:
        - ref: github-push-binding
      template:
        ref: oppari-build-template
```

Kuva 15. Kehitysputkeen määritetty EventListener

Triggerin omien resurssien lisäksi sille tarvittiin myös klusterin laajuiset käyttöoikeudet, muuten se ei pystyisi käynnistämään Pipelinea. Tehtiin rbac.yaml-tiedosto jonne lisättiin Triggerin tarvitsemat RoleBinding ja ClusterRoleBinding. Tarvittava ClusterRole oli valmiiksi asentunut Tektonin mukana klusteriin.

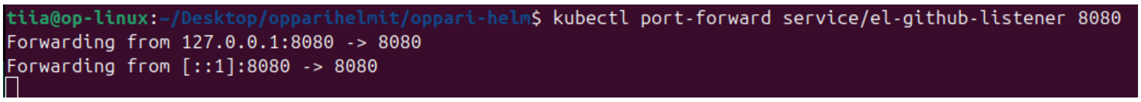
```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: triggers-eventlistener-binding
subjects:
  - kind: ServiceAccount
    name: tekton-robot
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tekton-triggers-eventlistener-roles
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: triggers-eventlistener-clusterbinding
subjects:
  - kind: ServiceAccount
    name: tekton-robot
    namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tekton-triggers-eventlistener-clusterroles
```

KUVA 16. Tiedosto rbac.yaml sisältää myös Triggerin vaatimat oikeudet

Webhook

EventListener reagoi ulkoisiin tapahtumiin, mutta jotta se saa tiedon, täytyy lähdepäässä olla jokin tiedon lähettäjä. Webhook on viesti joka tapahtuman sattuessa lähettää hyötykuorman vastaanottavaan verkko-osoitteeseen (Guay 26.6.2024). Tässä tapauksessa lähettäjä oli GitHub, hyötykuormassa oli tietoja lähettävästä repositoriosta ja vastaanottavassa osoitteessa odotti Minikubessa kuunteleva EventListener.

Koska EventListener oli Kubernetes-klusterissa, ei sillä oletuksena ollut reittiä ulkoverkkoon. Siksi tässä työssä käytettiin teknologiaa nimeltä ngrok, jossa on reititystoiminto (ForgoneReality 6.5.2025a.). Ensin käytettiin kubectl-komentorivityökalun omaa porttiohjauskäskyä ohjaamaan liikenne klusterin portista 80 tietokoneen paikalliseen porttiin 8080.

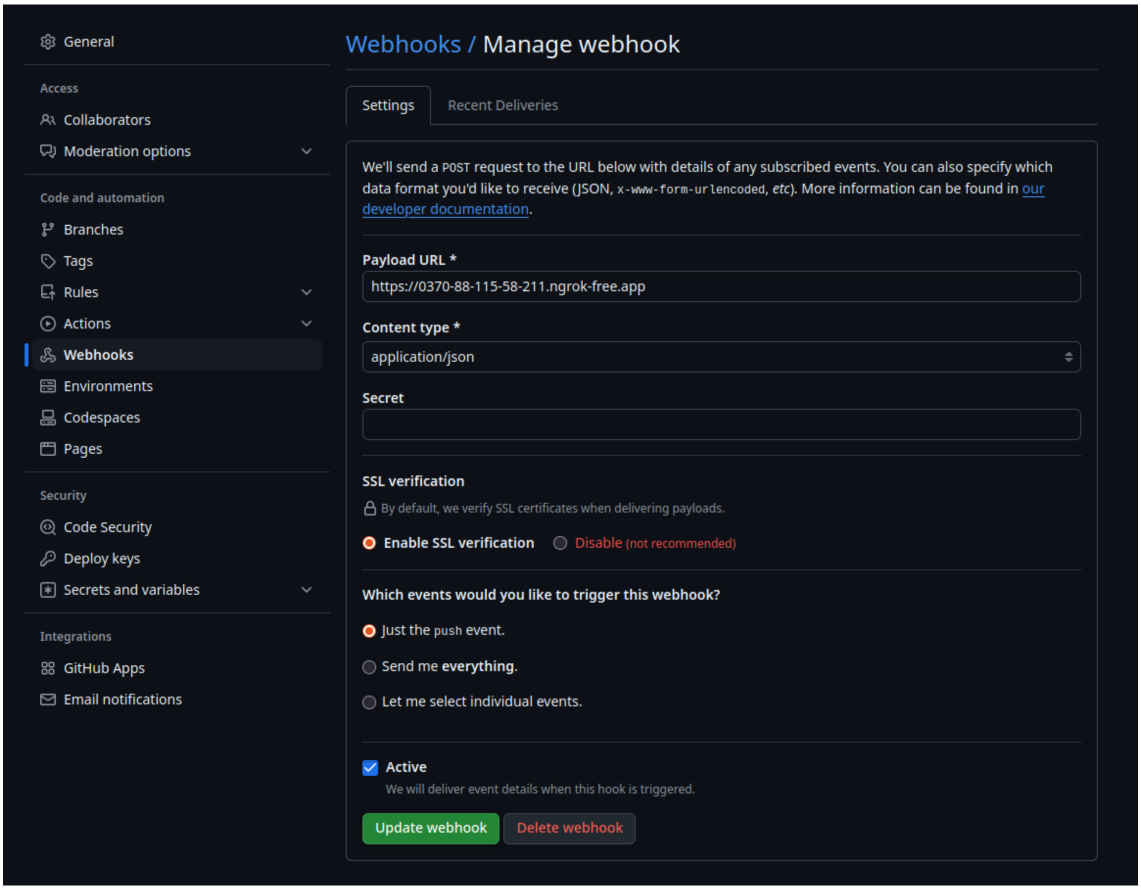


KUVA 17. Porttiohjaus ohjaa EventListenerin liikennettä ulos klusterista

Tämän jälkeen konfiguroitiin ngrok avaamaan tunneliyhteys. Se ohjaa liikenteen ulkoisesta verkko-osoitteesta paikalliseen porttiin 8080. Ilmaisessa versiossa ei ollut mahdollisuutta määrittää omaa osoitetta, joten ohjaava verkko-osoite vaihtui joka kerta kun tunneli avattiin uudelleen (ForgoneReality 6.5.2025b.).

Kuva 18. Näkymä ngrokin tulosteesta, jossa näkyy julkinen osoite

Webhook luotiin sovelluksen repositoriossa GitHubissa. Pakollisia tietoja olivat osoite, johon webhook lähettää viestin ja sisällön tyyppi. Webhookiin olisi ollut mahdollisuus määrittää myös salaus, jolla varmistetaan, että yhteys todella tulee oikeasta paikasta (GitHub 2025). Webhookin toimivuuden pystyi varmistamaan recent deliveries -välilehdeltä, jossa näkyi webhookin lähettämät testit. Onnistunut yhteys näkyi myös ngrokin näkymässä http-statuskoodilla 202 eli hyväksytty.



KUVA 19. GitHubissa luotu webhook ja sen konfiguraatiot

4.4 Testaus ja laadunvalvonta osana kehitysputkea

Kun perusmuotoinen kehitysputki oli saatu valmiiksi, siihen oli helppo lisätä uusia Taskeja muita toimintoja varten. Testausta varten voidaan julkaisuputkeen kytkeä erilaisia automaatio- tai integraatiotestejä. Tässä kehitystyössä luotiin Task, joka ajaa sovelluksen lähdekoodissa olevat yksikkötestit Maven-koonnin yhteydessä (MvnRepository 2025). Toinen Task laskee testikattavuuden samalla tavalla.

```yaml
- name: run-unit-tests
  taskRef:
    name: maven
    kind: Task
  runAfter:
    - clone-repo
  params:
    - name: MAVEN_IMAGE
      value: {{ .Values.pipeline.mavenImage | quote }}
    - name: GOALS
      value:
        - "clean"
        - "test"
  workspaces:
    - name: source
      workspace: shared-workspace
    - name: maven-settings
      workspace: maven-settings
```

KUVA 20. Kehitysputken Task, joka suorittaa yksikkötestit

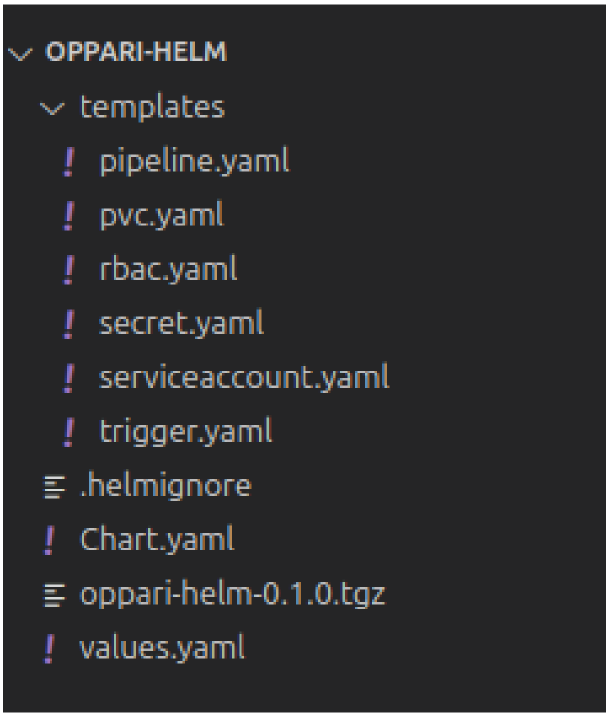
Haavoittuvuustarkastus kytkettiin kehitysputkeen myös Maven-riippuvuudella. OWASP:in komentorivipohjainen skannaus tuottaa podin lokille tulosteen lähdekoodista löydetyistä haavoittuvuuksista (OWASP Foundation 2025b.).

```
2025-04-27T19:14:58.483126723Z [INFO] Analysis Started
2025-04-27T19:14:58.960038849Z [INFO] Finished Archive Analyzer (0 seconds)
2025-04-27T19:14:58.966031242Z [INFO] Finished File Name Analyzer (0 seconds)
2025-04-27T19:14:59.548341618Z [INFO] Finished Jar Analyzer (0 seconds)
2025-04-27T19:14:59.554952844Z [INFO] Finished Dependency Merging Analyzer (0 seconds)
2025-04-27T19:14:59.557359057Z [INFO] Finished Version Filter Analyzer (0 seconds)
2025-04-27T19:14:59.665233706Z [INFO] Finished Hint Analyzer (0 seconds)
2025-04-27T19:15:01.690696714Z [INFO] Created CPE Index (2 seconds)
2025-04-27T19:15:02.735813486Z [INFO] Finished CPE Analyzer (3 seconds)
2025-04-27T19:15:02.747517434Z [INFO] Finished False Positive Analyzer (0 seconds)
2025-04-27T19:15:02.867499422Z [INFO] Finished NVD CVE Analyzer (0 seconds)
2025-04-27T19:15:04.030301359Z [INFO] Finished Sonatype OSS Index Analyzer (1 seconds)
2025-04-27T19:15:04.149217360Z [INFO] Finished Vulnerability Suppression Analyzer (0 seconds)
2025-04-27T19:15:04.157674714Z [INFO] Finished Dependency Bundling Analyzer (0 seconds)
2025-04-27T19:15:04.264587744Z [INFO] Analysis Complete (5 seconds)
2025-04-27T19:15:04.360298419Z [INFO] Writing report to: /workspace/source/target/dependency-check-report.html
2025-04-27T19:15:05.021191617Z [WARNING]
2025-04-27T19:15:05.021226195Z
2025-04-27T19:15:05.021228978Z One or more dependencies were identified with known vulnerabilities in rest-app:
2025-04-27T19:15:05.021231434Z
```

KUVA 21. Ote haavoittuvuustarkastukseen lokitiedostosta

4.5 Paketointi Helmillä

Tekton-resurssit paketoitiin Helm-työkalulla yhdeksi Helm chartiksi. Kaikki Tekton-resurssit laitettiin templates-kansioon ja chart ja values-tiedosto olivat kansion juuressa. Chart.yaml-tiedostoon laitettiin mallin mukaisesti vain chartin nimi ja versiotiedot, koska ulkoisia riippuvuuksia ei tässä ollut. Asennukseen valmis chart muodostui pakatussa muodossa version mukaan nimettynä.

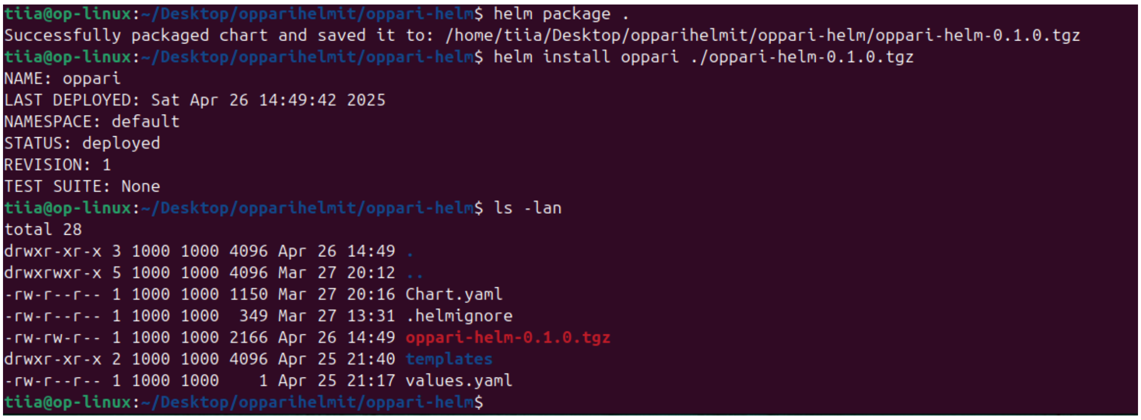


KUVA 22. Helm chartin rakenne

Helm chart paketoitiin yhdellä komennolla, jossa tärkeintä oli tehdä viittaus oikeaan kansioon. Yksinkertaisinta oli tehdä komento chart-kansion sisällä, jolloin viittaukseksi riitti pelkkä piste. Chartin asennus täytyy aina suorittaa kirjautuneena klusteriin. Minikube-klusterissa erillistä kirjautumista ei kuitenkaan tarvittu, vaan asennuskomento toimi suoraan pakkauksen jälkeen. Asennuskomennossa helmin asennukselle annettiin nimi, joka käytännössä yksilöi asennettavan chartin. Samalla nimellä klusteriin ei voi tehdä toista charttia, mutta alkuperäistä charttia voi päivittää. Silloin helm asettaa uudelle versiolle uuden versionumeron. Kun helm-komento oli mennyt onnistuneesti läpi, näkyivät luodut resurssit välittömästi Minikube-klusterissa. Niitä pystyi katsomaan sekä Minikuben että Tektonin dashboardilta tai käyttämällä kubectl-komentorivityökalua.

Helm chartin paketointiin ja asentamiseen tarvittavat komennot:

helm package .
helm install oppari ./oppari-helm.tgz



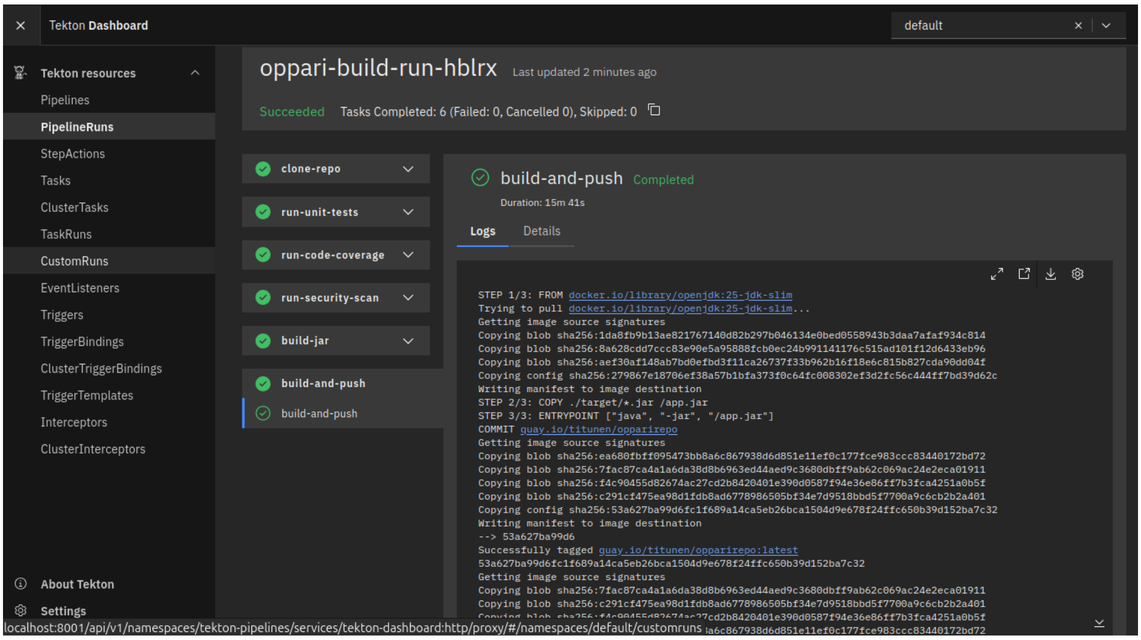
KUVA 23. Helm chartin paketointi ja asennus klusteriin

4.6 Käynnistyskomennot ja testaus

Kun kehitysputken elementit olivat onnistuneesti klusterissa, oli kaikki valmista putken testaamista varten. Koska Trigger ja webhook olivat saatu käyttöön, tehtiin testaus tekemällä muutos testisovelluksen lähdekoodiin. Kun Git vastaanotti push-käskyn, toimi GitHubin webhook välittömästi. Paras paikka seurata kehitysputken ajoa oli Tekton dashboard, jonka sai käyttöön antamalla komentorivillä seuraava käsky:

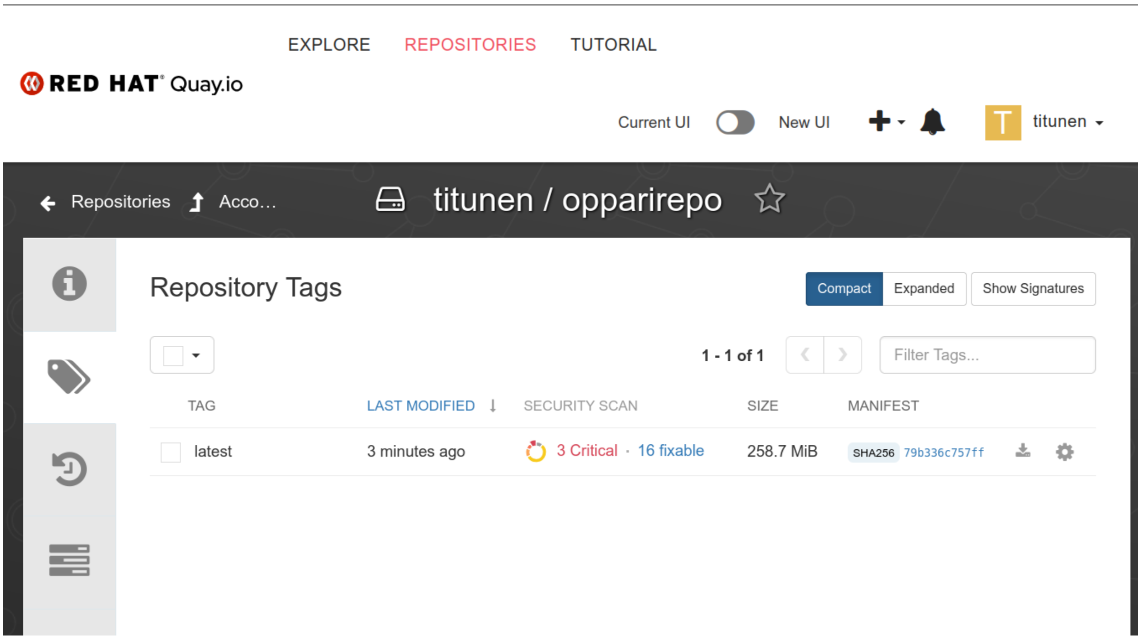
kubectl proxy

Dashboardilla pystyi seuraamaan kehitysputken uniikisti nimettyä PipelineRun-ajoa askel askeleelta. Jokaisen Taskin kohdalta pääsi porautumaan lokille, josta sai seurattua Taskin toimintaa. Jokainen askel merkittiin vihreällä värillä, jos kyseinen Task tai Step suoriutui onnistuneesti. Vastaavasti punainen väri osoitti, että Task on epäonnistunut. Yleensä koko kehitysputki epäonnistuu, jos kaikki Taskit eivät mene onnistuneesti läpi. Taskit voivat epäonnistua joko teknisistä syistä, tai koska niiden muut kriteerit, kuten tarvittava testikattavuus ei täyty.



KUVA 24. Kehitysputken ajo on suoritettu onnistuneesti

Viimeinen Task rakensi konttikuvan ja latasi sen ulkoiseen repositorioon. Tämä vaihe oli kehitysputken hitain ja kesti useita kymmeniä minuutteja. Kun kehitysputken ajo oli päässyt loppuun asti, uusi konttikuva oli myös löydettävissä Quay.io-repositoriosta. Konttikuva oli määritelty TriggerTemplate-objektissa käyttämään tagia latest, joka korvaa vanhan konttikuvan repositoriossa.



KUVA 25. Uusin konttikuva Quay.io-repositoriossa ja päivittynyt latest-tag

5 AUTOMAATION ONNISTUMINEN

Kehittämistyössä saatiin tehtyä avoimen lähdekoodin ohjelmistoilla automaattinen kehitysputki, johon on sisällytetty myös testausta ja haavoittuvuuksien skannausta. Kehitysputken tuotos on konttikuva esimerkkisovelluksesta. Tuon konttikuvan avulla sovellus voidaan asentaa ajoon erilaisiin ympäristöihin.

Minikuben käyttö paikallisena Kubernetes-klusterina onnistui helposti. Tektonin resurssien konfiguraatioita voi tehdä monella tavalla, joten ei ole vain yhtä tapaa tehdä kehitysputki. Tässä työssä esiteltiin yksinkertainen malli, joka käyttää valmiita Taskeja ja suorittaa yksinkertaisia toimintoja. Helm charttien mukaan ottaminen lisäsi mahdollisuutta parametrisoida kehitysputkea ja Trigger toi automaatiota, jolloin kehitysputkea ei tarvitse ajaa manuaalisesti. Tässä työssä kuitenkin jouduttiin käyttämään ngrok-työkalua, jonka ilmaiset toiminnot ovat rajalliset. Sen vuoksi webhookille piti jokaisella uudella klusterin käynnistyskerralla antaa uusi verkko-osoite yhdistämistä varten.

Kehitysputkesta jäi puuttumaan sovelluksen julkaisu, koska se jouduttiin aiheen rajaamiseksi jättämään pois. Jatkokehitystä pitääkin tehdä juuri tuon julkaisun osalta. Lisäksi Helm chartit voisi antaa Argo CD:n hallintaan, jolloin saataisiin automaatiota lisättyä myös GitOps-periaatteen mukaisesti.

Secret-objektit olivat tässä työssä osana Helm charttia, mutta niitä ei ole turvallista säilyttää versionhallinnassa. Salattava tiedot voisi asentaa suoraan klusteriin tai sitten käyttää jotain salaustyökalua kuten Kubeseal (Kumar 6.10.2022). Sen lisäksi kehitysputkea pitäisi vielä optimoida, sillä sen suorittaminen kesti usein melkein tunnin. Pisin vaihe oli viimeinen vaihe eli konttikuvan rakentaminen ja lataaminen Quay.io-repositorioon. Siihen osaltaan vaikutti todennäköisesti ilmaisen version hitaat tietoliikenneyhteydet.

Nyt kun kehitysputki on saatu toimimaan paikallisessa ympäristössä, olisi seuraava askel kokeilla samoja Tekton-resursseja suoraan jossain valmiissa kehitysympäristössä, kenties jopa pilvipalvelussa. Valmiissa ympäristössä monet asiat voivat olla helpompia, mutta toisaalta myös rajoitetumpia.

6 POHDINTA

DevOpsiin ja CI/CD-menetelmiin kuuluu paljon eri teknologioita, ja niiden kaikkien täydellinen hallinta on haastavaa. Tässä kehitystyössä on vain pintaraapaisu kaikesta mitä voi kehitysputkella tehdä. Yksinäisen kehittäjän kynnys lähteä opettelemaan näitä uutena asiana voi olla korkea, mutta suuremmissa organisaatioissa DevOpsista ja etenkin automaatiosta on varmasti etua.

Kehitysputken avulla voidaan DevOps-periaatteen mukaisesti tarkkailla koodin laatua jo varhaisesta kehityksestä alkaen. Sovelluksen laatu ja turvallisuus on olennaista, jotta niissä ei ole vaarallisia haavoittuvuuksia, joiden avulla pahantahtoiset hyökkääjät voivat päästä käsiksi arkaluonteisiin tietoihin. Automaation käyttöönotto taas täyttää jatkuvan integraation periaatteen. Automaatio myös vähentää virheitä ja lisää tuottavuutta. Pitää kuitenkin huomioida, että kehitysputken monipuoliset ominaisuudet eivät saa hidastaa putken suoritusta liikaa. Tällöin uusia ominaisuuksia saadaan hitaammin käyttöön, joten jatkuvan integraation periaate ei täysin täyty.

DevOpsin suosio on ollut suuri jo pidemmän aikaa ja sen käyttöönottoakin on tutkittu useamman kerran (kts. Pajari 2022, Halttunen 2022, Heikkinen 2021). Tästä voisi olettaa, että ketterät mallit ovat todennäköisesti tulleet jäädäkseen. Tulevaisuudessa DevOpsin seuraajaksi voisi kuitenkin tulla esimerkiksi alustalähtöinen kehitys eli *platform engineering*, jossa kehittäjä saa käyttöönsä alustan, jolla voi rakentaa koko sovelluksen elinkaaren alusta loppuun (Galante 2025). Tämä toteuttaisi DevOpsia pidemmälle ja paremmin kuin itse DevOps.

Tämä opinnäyte vahvisti osaamistani monella eri osa-alueella ja pääsin syventymään etenkin Tektonin käyttöön. Osasin yhdistää aiempaa työstä hankittua osaamistani uuden teorian kanssa ja nämä uudet taidot varmasti tukevat myös nykyisiä työtehtäviäni.

LÄHTEET

8grams 29.5.2023. Tekton: The Open Source, Kubernetes-native CI/CD Tools. Medium. Luettavissa: https://8grams.medium.com/tekton-the-open-source-kubernetes-native-ci-cd-tools-55c49db53234. Luettu: 6.4.2025.

Agile Manifesto 2001. Manifesto for Agile Software Development. Luettavissa: https://agilemanifesto.org/. Luettu: 19.4.2025.

Amazon 2025. What is DevOps? Luettavissa: https://aws.amazon.com/devops/what-is-devops/. Luettu: 31.1.2025.

Argo CD s.a. What is Argo CD? Luettavissa: https://argo-cd.readthedocs.io/en/stable/. Luettu: 19.4.2025.

Atlassian 2025. Waterfall Methodology: A Comprehensive Guide. Luettavissa: https://www.atlassian.com/agile/project-management/waterfall-methodology. Luettu: 11.4.2025.

Bhatt, T. 21.2.2024. Automation In Software Development: Benefits, Challenges, Tips. Intelivita. Luettavissa: https://www.intelivita.com/blog/automation-in-software-development/. Luettu: 11.4.2025.

Cicero, I. 16.3.2021. Docker + Kubernetes + Helm: A comprehensive step-by-step using Java. Medium. Luettavissa: https://ignaciocicero.medium.com/docker-kubernetes-helm-a-comprehensive-step-by-step-using-java-df83f6780d80. Luettu: 27.12.2023.

ForgoneReality 6.5.2025a. Universal gateway. ngrok. Luettavissa: https://ngrok.com/docs/universal-gateway/overview/. Luettu: 6.5.2025.

ForgoneReality 6.5.2025ab. HTTP/S Endpoints. ngrok. Luettavissa: https://ngrok.com/docs/universal-gateway/http/. Luettu: 6.5.2025.

Foster, S. 7.6.2023. What Is Static Analysis? Static Analysis Tools + Static Code Analyzers Overview. Perforce. Luettavissa: https://www.perforce.com/blog/sca/what-static-analysis. Luettu: 11.4.2025.

Galante, L. 2025. What is platform engineering? Platform Engineering. Luettavissa: https://platformengineering.org/blog/what-is-platform-engineering. Luettu: 27.4.2025.

Garcia, O. 8.8.2024. What is Minikube and how does it work? Damavis. Luettavissa: https://blog.damavis.com/en/what-is-minikube-and-how-does-it-work/. Luettu: 27.4.2025.

GitHub 2025. Validating webhook deliveries. Luettavissa: https://docs.github.com/en/webhooks/using-webhooks/validating-webhook-deliveries. Luettu: 30.4.2025.

Guay, M. 26.6.2024. What are webhooks? Zapier. Luettavissa: https://zapier.com/blog/what-are-webhooks/. Luettu: 30.4.2025.

Gunja, S. 16.2.2023. What is DevOps? Unpacking the purpose and importance of an IT cultural revolution. Dynatrace. Luettavissa: https://www.dynatrace.com/news/blog/what-is-devops/. Luettu 4.4.2025.

Halttunen, J. 2021. DevOps ja tuotannon monitorointiratkaisu. Opinnäytetyö. Jyväskylän ammattikorkeakoulu, tieto- ja viestintätekniikan tutkinto-ohjelma. Luettavissa: https://urn.fi/URN:NBN:fi:amk-2022052311262. Luettu: 30.4.2025.

Heikkinen, J. 2021. Jatkuva toimittaminen Azure DevOps-ympäristössä. Opinnäytetyö. Jyväskylän ammattikorkeakoulu, tieto- ja viestintätekniikan tutkinto-ohjelma. Luettavissa: https://urn.fi/URN:NBN:fi:amk-2021060113127. Luettu: 30.4.2025.

Helm Authors 2025a. What is Helm? Luettavissa: https://helm.sh/. Luettu: 24.4.2025.

Helm Authors 2025b. Installing Helm. Luettavissa: https://helm.sh/docs/intro/install/. Luettu 24.4.2025.

Isaiah, A. 24.3.2025. Minikube vs Kind: A Comprehensive Comparison. BetterStack. Luettavissa: https://betterstack.com/community/guides/scaling-docker/minikube-vs-kubernetes/. Luettu: 19.4.2025.

Kubernetes Authors 8.8.2024. Install and Set Up kubectl on Linux. Luettavissa: https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/. Luettu: 1.5.2025.

Kubernetes Authors 19.11.2024a. Service Accounts. Luettavissa: https://kubernetes.io/docs/concepts/security/service-accounts/. Luettu: 1.5.2025.

Kubernetes Authors 19.11.2024b. Secrets. Luettavissa: https://kubernetes.io/docs/concepts/configuration/secret/. Luettu: 1.5.2025.

Kubernetes Authors 18.3.2025. Volumes. Luettavissa: https://kubernetes.io/docs/concepts/storage/volumes/. Luettu: 1.5.2025.

Kubernetes Authors 9.4.2025. Using RBAC Authorization. Luettavissa: https://kubernetes.io/docs/reference/access-authn-authz/rbac/. Luettu: 1.5.2025.

Kumar, S. 6.10.2022. Kubeseal & SealedSecret: Make your 'secrets' secure in SCM by using 'sealed secret'. Medium. Luettavissa: https://siddhivinayak-sk.medium.com/kubeseal-sealedsecret-make-your-secrets-secure-in-scm-by-using-sealed-secret-4631bcb39bf8. Luettu 1.5.2025.

Leite, L., Rocha, C., Kon, F., Milojicic, D. & Meirelles, P. 2020. A Survey of DevOps Concepts and Challenges. ACM computing surveys, 52, 6, s. 1-35. Luettavissa: ResearchGate. Luettu: 30.4.2025.

Minikube 15.1.2025. Minikube start. Luettavissa: https://minikube.sigs.k8s.io/docs/start/?arch=%2Flinux%2Fx86-64%2Fstable%2Fbinary+download. Luettu: 30.4.2025.

MvnRepository 2025. JaCoCo Maven Plugin. Luettavissa: https://mvnrepository.com/artifact/org.jacoco/jacoco-maven-plugin. Luettu. 11.4.2025.

Nirmalkushwah 30.6.2023. Docker Image Tagging Strategy. Medium. Luettavissa: https://medium.com/@nirmalkushwah08/docker-image-tagging-strategy-4aa886fb4fcc. Luettu: 1.5.2025.

OWASP Foundation 2025a. Vulnerabilities. Luettavissa: https://owasp.org/www-community/vulnerabilities/. Luettu: 11.4.2025.

OWASP Foundation 2025b. OWASP Dependency-Check. Luettavissa: https://owasp.org/www-project-dependency-check/. Luettu: 22.4.2025.

Pajari, T. 2024. DevOpsin CI/CD-menetelmien käyttöönotto web-sovelluksessa. Opinnäytetyö. Jyväskylän ammattikorkeakoulu, tieto- ja viestintätekniikan tutkinto-ohjelma. Luettavissa: https://urn.fi/URN:NBN:fi:amk-2022102821696. Luettu: 30.4.2025.

Palo Alto Networks 2025. Kubernetes and Infrastructure as Code. Luettavissa: https://www.paloaltonetworks.com/cyberpedia/kubernetes-infrastructure-as-code. Luettu: 20.4.2025.

Singh, J. 3.2.2023. Top Cyber Attacks Due to Vulnerabilities in 2022! Luettavissa: https://www.secpod.com/blog/top-cyber-attacks-due-to-vulnerabilities-in-2022/. Luettu: 11.4.2025.

Raja Ravi Varman, A.J.S. 5.6.2024. Jenkins vs Tekton: Choosing the Right CI/CD Tool for Your DevOps Pipeline. Medium. Luettavissa: https://medium.com/@rajaravivarman/jenkins-vs-tekton-choosing-the-right-ci-cd-tool-for-your-devops-pipeline-727aa2329ca5. Luettu: 6.5.2025.

Red Hat 12.12.2023. 2025c. What is CI/CD? Luettavissa: https://www.redhat.com/en/topics/devops/what-is-ci-cd. Luettu: 7.2.2025.

Red Hat 18.12.2024. What is Kubernetes? Luettavissa: https://www.redhat.com/en/topics/containers/what-is-kubernetes#overview. Luettu 8.2.2025.

Red Hat 2025. What is Red Hat Openshift? Luettavissa: https://www.redhat.com/en/technologies/cloud-computing/openshift. Luettu: 11.4.2025.

Red Hat 28.2.2025. What is a CI/CD pipeline? Luettavissa: https://www.redhat.com/en/topics/devops/what-cicd-pipeline. Luettu 19.4.2025.

Red Hat 27.3.2025. What is GitOps? Luettavissa: https://www.redhat.com/en/topics/devops/what-is-gitops. Luettu 6.4.2025.

Robinson, S., Brush, K. & Silverthorne, V. 2024. What is Agile software development? TechTarget. Luettavissa: https://www.techtarget.com/searchsoftwarequality/definition/agile-software-development. Luettu 19.4.2025.

SonarSource SA 2025. SonarQube Community Build. Luettavissa: https://www.sonarsource.com/open-source-editions/sonarqube-community-edition/. Luettu: 6.5.2025.

Susnjara, S. & Smalley, I. 20.5.2024. What is containerization? IBM. Luettavissa: https://www.ibm.com/think/topics/containerization. Luettu: 8.4.2025.

Suttichujit, C. 29.6.2021. Build Docker Image Using Tekton Pipeline + Buildah. Medium. Luettavissa: https://medium.com/nontechcompany/build-docker-image-using-tekton-pipeline-buildah-fe62a8f70a75. Luettu: 20.4.2025.

The Linux Foundation 9.3.2023. Overview. Tekton. Luettavissa: https://tekton.dev/docs/concepts/overview/. Luettu: 4.4.2025.

The Linux Foundation 2025. Tekton Hub. Luettavissa: https://tekton.dev/docs/operator/tektonhub/. Luettu 1.5.2025.

Watts, B. 29.2.2024. Unit Testing — Pros and Cons — Understanding why testing code is a good practice. Medium. Luettavissa: https://canvascodebw.medium.com/unit-testing-pros-and-cons-understanding-why-testing-code-is-a-good-practice-47a81d67ccde. Luettu: 11.4.2025.

Wilson, B. 8.11.2022. Podman Tutorial For Beginners: Step by Step Guides. DevOpsCube. Luettavissa: https://devopscube.com/podman-tutorial-beginners/. Luettu 20.4.2025.

LIITTEET

Liite 1 Kehitystyössä tehty pipeline.yaml

Liite 2 Kehitystyössä tehty trigger.yaml

Liite 3 Kehitystyössä tehty rbac.yaml

Liite 4 Kehitystyössä tehty secret.yaml

Liite 5. Kehitystyössä tehty serviceaccount.yaml

Liite 6 Kehitystyössä tehty pvc.yaml

Liite 7 Kehitystyössä tehty Chart.yaml

Liite 8 Kehitystyössä tehty values.yaml

Koodi on rakennettu Tektonin dokumentaation pohjalta lisäämällä esimerkkitoteutukseen tarvittavat muutokset (The Linux Foundation 9.3.2023). Tekton vaatii tarkan syntaksin, jota ei voi muokata.

Koodi alkaa:

```
apiVersion: tekton.dev/v1beta1
kind: Pipeline
metadata:
  name: {{ .Values.pipeline.name }}
  namespace: {{ .Values.pipeline.namespace }}
spec:
  params:
    - name: repo-url
      type: string
      description: Git repository
    - name: revision
      type: string
      default: "main"
      description: Branch to clone
    - name: image-url
      type: string
      description: Destination image
  workspaces:
    - name: shared-workspace
    - name: maven-settings

  tasks:
    - name: clone-repo
      taskRef:
        name: git-clone
        kind: Task
      params:
        - name: url
          value: "$(params.repo-url)"
        - name: revision
          value: "$(params.revision)"
      workspaces:
        - name: output
          workspace: shared-workspace

    - name: run-unit-tests
      taskRef:
        name: maven
```

```yaml
    kind: Task
  runAfter:
    - clone-repo
  params:
    - name: MAVEN_IMAGE
      value: {{ .Values.pipeline.mavenImage | quote }}
    - name: GOALS
      value:
        - "clean"
        - "test"
  workspaces:
    - name: source
      workspace: shared-workspace
    - name: maven-settings
      workspace: maven-settings

- name: run-code-coverage
  taskRef:
    name: maven
    kind: Task
  runAfter:
    - run-unit-tests
  params:
    - name: MAVEN_IMAGE
      value: {{ .Values.pipeline.mavenImage | quote }}
    - name: GOALS
      value:
        - "clean"
        - "test"
        - "jacoco:report"
  workspaces:
    - name: source
      workspace: shared-workspace
    - name: maven-settings
      workspace: maven-settings

- name: run-security-scan
  taskRef:
    name: maven
    kind: Task
  runAfter:
    - run-code-coverage
  params:
    - name: MAVEN_IMAGE
      value: {{ .Values.pipeline.mavenImage | quote }}
    - name: GOALS
      value:
```

```yaml
        - "org.owasp:dependency-check-maven:check"
  workspaces:
    - name: source
      workspace: shared-workspace
    - name: maven-settings
      workspace: maven-settings

- name: build-jar
  taskRef:
    name: maven
    kind: Task
  runAfter:
    - run-security-scan
  params:
    - name: MAVEN_IMAGE
      value: {{ .Values.pipeline.mavenImage | quote }}
    - name: GOALS
      value:
        - "clean"
        - "package"
        - "-DskipTests"
  workspaces:
    - name: source
      workspace: shared-workspace
    - name: maven-settings
      workspace: maven-settings
- name: build-and-push
  taskRef:
    name: buildah
    kind: Task
  runAfter:
    - build-jar
  params:
    - name: IMAGE
      value: "$(params.image-url)"
    - name: STORAGE_DRIVER
      value: "overlay"
  workspaces:
    - name: source
      workspace: shared-workspace
```

Koodi on rakennettu Tektonin dokumentaation pohjalta lisäämällä esimerkkitoteutukseen tarvittavat muutokset (The Linux Foundation 9.3.2023). Tekton vaatii tarkan syntaksin, jota ei voi muokata.

Koodi alkaa:

```
apiVersion: triggers.tekton.dev/v1beta1
kind: TriggerBinding
metadata:
  name: github-push-binding
spec:
  params:
    - name: repo-url
      value: $(body.repository.clone_url)
    - name: git-revision
      value: $(body.after)
    - name: git-branch
      value: $(body.ref)
---
apiVersion: triggers.tekton.dev/v1beta1
kind: TriggerTemplate
metadata:
  name: oppari-build-template
spec:
  params:
    - name: repo-url
    - name: git-revision
    - name: git-branch
    - name: image-url
  resourcetemplates:
    - apiVersion: tekton.dev/v1beta1
      kind: PipelineRun
      metadata:
        generateName: oppari-build-run-
      spec:
        pipelineRef:
          name: oppari-build-pipeline
        serviceAccountName: oppari-service-account
        params:
          - name: repo-url
            value: $(tt.params.repo-url)
          - name: git-revision
            value: $(tt.params.git-revision)
          - name: git-branch
```

```yaml
        value: $(tt.params.git-branch)
      - name: image-url
        value: quay.io/titunen/opparirepo:latest
    workspaces:
      - name: shared-workspace
        persistentVolumeClaim:
          claimName: oppari-pipeline-pvc
      - name: maven-settings
        emptyDir: {}
---
apiVersion: triggers.tekton.dev/v1beta1
kind: EventListener
metadata:
  name: github-listener
spec:
  serviceAccountName: tekton-robot
  triggers:
    - name: github-push-trigger
      bindings:
        - ref: github-push-binding
      template:
        ref: oppari-build-template
```

Koodi on rakennettu Kuberneteksen dokumentaation pohjalta lisäämällä esimerkkitoteutukseen tarvittavat muutokset (Kubernetes Authors 9.4.2025). Kubernetes vaatii tarkan syntaksin, jota ei voi muokata.

Koodi alkaa:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: triggers-eventlistener-binding
subjects:
  - kind: ServiceAccount
    name: tekton-robot
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tekton-triggers-eventlistener-roles
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: triggers-eventlistener-clusterbinding
subjects:
  - kind: ServiceAccount
    name: tekton-robot
    namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: tekton-triggers-eventlistener-clusterroles
```

KOODI SECRET.YAML — LIITE 4

Koodi on rakennettu Kuberneteksen dokumentaation pohjalta lisäämällä esimerkkitoteutukseen tarvittavat muutokset (Kubernetes Authors 19.11.2024b.). Secret on siistitty, koska avain ei voi olla kaikkialla sama. Kubernetes vaatii tarkan syntaksin, jota ei voi muokata.

Koodi alkaa:

```
apiVersion: v1
kind: Secret
metadata:
  name: titunen-opparirobot-pull-secret
data:
  .dockerconfigjson: xxxxxxsecretxxxxredactedxxxxx
type: kubernetes.io/dockerconfigjson
```

Koodi on rakennettu Kuberneteksen dokumentaation pohjalta lisäämällä esimerkkitoteutukseen tarvittavat muutokset (Kubernetes Authors 19.11.2024a.). Kubernetes vaatii tarkan syntaksin, jota ei voi muokata.

Koodi alkaa:

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: oppari-service-account
  namespace: default
secrets:
  - name: titunen-opparirobot-pull-secret
imagePullSecrets:
  - name: titunen-opparirobot-pull-secret
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: tekton-robot
```

Koodi on rakennettu Kuberneteksen dokumentaation pohjalta lisäämällä esimerkkitoteutukseen tarvittavat muutokset (Kubernetes Authors 18.3.2025). Kubernetes vaatii tarkan syntaksin, jota ei voi muokata.

Koodi alkaa:

```yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: oppari-pipeline-pvc
  namespace: default
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Koodi on rakennettu Helmin dokumentaation pohjalta lisäämällä esimerkkitoteutukseen tarvittavat muutokset (Helm Authors 2025a). Helm vaatii tarkan syntaksin, jota ei voi muokata.

Koodi alkaa:

```yaml
apiVersion: v2
name: oppari-helm
description: A Helm chart for oppari
type: application
version: 0.1.0
appVersion: "1.17.0"
```

Koodi on rakennettu Helmin dokumentaation pohjalta lisäämällä esimerkkitoteutukseen tarvittavat muutokset (Helm Authors 2025a). Helm vaatii tarkan syntaksin, jota ei voi muokata.

Koodi alkaa:

```
pipeline:
  name: oppari-build-pipeline
  namespace: default
  mavenImage: maven:3.8.6-eclipse-temurin-17
```