

Ohjelmistotuotanto

Syksy 2025

Luento 7

17.11.2025

Kurssipalaute

- ▶ Kurssipalaute
 - ▶ Kurssilla lopussa kerättävän palautteen lisäksi ns. jatkuva palaute <https://norppa.helsinki.fi>

Paja

- ▶ BK107
 - ▶ ma 14.30-16.30
 - ▶ ti 12-16
 - ▶ to 12-16
 - ▶ pe 12-14

Miniprojektit

- ▶ Käynnissä!
- ▶ Tämä ja pari seuraavaa viikkoa: sprinttien katselmus ja suunnittelu samassa aikaikkunassa
- ▶ Loppudemot
 - ▶ ke 10.12. klo 10-12 B123
 - ▶ to 11.12. klo 10-12 A111

CurreChat

- ▶ Käytettävissä HY:n GPT-chat
 - ▶ <https://curre.helsinki.fi/chat>

CurreChat

- ▶ Käytettävissä HY:n GPT-chat
 - ▶ <https://curre.helsinki.fi/chat>
- ▶ Keskusteluja ei käytetä koulutusdatana

- ▶ Käytettävissä HY:n GPT-chat
 - ▶ <https://curre.helsinki.fi/chat>
- ▶ Keskusteluja ei käytetä koulutusdatana
- ▶ Kurssimateriaalichat
 - ▶ Ei (ehkä) hallusinoi
 - ▶ Osin herkkä kysymysten sanamuodon suhteen

Ketterien menetelmien testauskäytänteitä

Ketterien menetelmien testauskäytänteitä

- ▶ Sprintissä toteutettavat storyt integroidaan ja testataan sprintin aikana

Ketterien menetelmien testauskäytänteitä

- ▶ Sprintissä toteutettavat storyt integroidaan ja testataan sprintin aikana
- ▶ Automatisointi tärkeässä roolissa

Ketterien menetelmien testauskäytänteitä

- ▶ Sprintissä toteutettavat storyt integroidaan ja testataan sprintin aikana
- ▶ Automatisointi tärkeäässä roolissa
- ▶ Ideaalitilanteessa testaajia sijoitettu kehittäjätiimiin, myös ohjelmoijat kirjoittavat testejä
 - ▶ tiimit *cross functional*

Ketterien menetelmien testauskäytänteitä

Ketterien menetelmien testauskäytänteitä

- ▶ Test driven development (TDD)
 - ▶ Sivutuotteena paljon automaattisesti suoritettavia testejä

Ketterien menetelmien testauskäytänteitä

- ▶ Test driven development (TDD)
 - ▶ Sivutuotteena paljon automaattisesti suoritettavia testejä
- ▶ User storyjen tasolla tapahtuva automatisoitu testaus
 - ▶ Robot

Ketterien menetelmien testauskäytänteitä

- ▶ Test driven development (TDD)
 - ▶ Sivutuotteena paljon automaattisesti suoritettavia testejä
- ▶ User storyjen tasolla tapahtuva automatisoitu testaus
 - ▶ Robot
- ▶ Exploratory testing, suomeksi tutkiva testaus
 - ▶ Järjestelmätestauksen tekniikka, jossa testaaminen tapahtuu ilman formaalia testaussuunnitelmaa

Ketterien menetelmien testauskäytänteitä

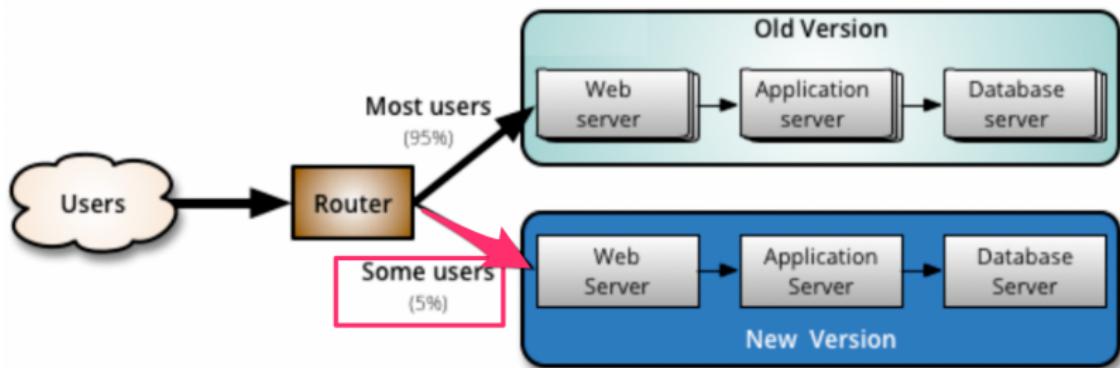
- ▶ Test driven development (TDD)
 - ▶ Sivutuotteena paljon automaattisesti suoritettavia testejä
- ▶ User storyjen tasolla tapahtuva automatisoitu testaus
 - ▶ Robot
- ▶ Exploratory testing, suomeksi tutkiva testaus
 - ▶ Järjestelmätestauksen tekniikka, jossa testaaminen tapahtuu ilman formaalia testaussuunnitelmaa
- ▶ Jatkuva integraatio (CI) ja jatkuva toimittaminen (CD)
 - ▶ Ulottuu jopa sovelluksen tuotantoonviemiseen

Ketterien menetelmien testauskäytänteitä

- ▶ Test driven development (TDD)
 - ▶ Sivutuotteena paljon automaattisesti suoritettavia testejä
- ▶ User storyjen tasolla tapahtuva automatisoitu testaus
 - ▶ Robot
- ▶ Exploratory testing, suomeksi tutkiva testaus
 - ▶ Järjestelmätestauksen tekniikka, jossa testaaminen tapahtuu ilman formaalia testaussuunnitelmaa
- ▶ Jatkuva integraatio (CI) ja jatkuva toimittaminen (CD)
 - ▶ Ulottuu jopa sovelluksen tuotantoonviemiseen
- ▶ Tuotannossa tapahtuva testaus
 - ▶ Tehdään osa laadunhallinnasta *monitoroimalla* tuotannossa olevaa ohjelmistoa

Canary release

- ▶ Kaksi rinnakkaista tuotantoymäristöä, joista uudet ominaisuudet viedään toiseen



- ▶ Uuden ominaisuuden sisältävään ympäristöön ohjataan osa järjestelmän käyttäjistä
- ▶ Uuden ominaisuuden sisältämää versiota *monitoroidaan*
 - ▶ jos ei ongelmia ohjataan kaikki liikenne uuteen versioon

Feature toggle

- ▶ Koodiin *ehtolauseita*: osa liikenteestä ohjataan vanhan toteutuksen sijaan testauksen alla olevaan toteutukseen

Feature toggle

- ▶ Koodin ehtolauseita: osa liikenteestä ohjataan vanhan toteutuksen sijaan testauksen alla olevaan toteutukseen
- ▶ Esim. some-palvelussa feature toggle: *osalle käytetään näytetään uuden algoritmin perusteella generoitu lista uutisia*

```
def recommended_news_generator(user):  
    if is_in_canary_release(user):  
        return experimental_recommendation_algorithm(user)  
    else:  
        return recommendation_algorithm(user)
```

Feature togglejen soveltaminen

- ▶ Aluksi piilotetaan uusi ominaisuus käyttäjiltä feature toggleilla
 - ▶ eli toggle palauttaa vanhan version normaaleille käyttäjille

Feature togglejen soveltaminen

- ▶ Aluksi piilotetaan uusi ominaisuus käyttäjiltä feature toggleilla
 - ▶ eli toggle palauttaa vanhan version normaaleille käyttäjille
- ▶ Kun valmiina laajempaan testiin, julkistaan esim.
 - ▶ ensin kehittäjärytyksen omaan käyttöön
 - ▶ sitten osalle käyttäjistä canary releasena

Feature togglejen soveltaminen

- ▶ Aluksi piilotetaan uusi ominaisuus käyttäjiltä feature toggleilla
 - ▶ eli toggle palauttaa vanhan version normaaleille käyttäjille
- ▶ Kun valmiina laajempaan testiin, julkistaan esim.
 - ▶ ensin kehittäjärytyksen omaan käyttöön
 - ▶ sitten osalle käyttäjistä canary releasena
- ▶ Käytetään paljon A/B-testaamiseen

Feature togglejen soveltaminen

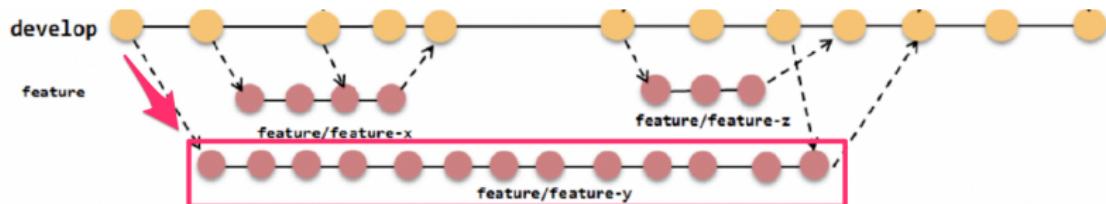
- ▶ Aluksi piilotetaan uusi ominaisuus käyttäjiltä feature toggleilla
 - ▶ eli toggle palauttaa vanhan version normaaleille käyttäjille
- ▶ Kun valmiina laajempaan testiin, julkaistaan esim.
 - ▶ ensin kehittäjäyrityksen omaan käyttöön
 - ▶ sitten osalle käyttäjistä canary releasena
- ▶ Käytetään paljon A/B-testaamiseen
- ▶ Lopulta feature toggle ja vanha toteutus voidaan poistaa

Versionhallinnan käytötavoista

Feature branchit

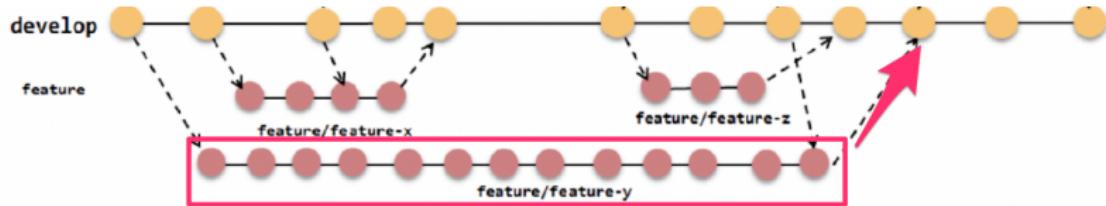
Feature branchit

- ▶ Uusi ominaisuus, esim. user story toteutetaan ensin omaan versionhallinnan haaraansa



Feature branchit

- ▶ Uusi ominaisuus, esim. user story toteutetaan ensin omaan versionhallinnan haaraansa



- ▶ ja ominaisuuden valmistuttua haara mergetään pääkehityshaaraan

Feature branchit ja merge hell

- ▶ Monet pitävät feature brancheja versionhallinnan *parhaana käytänteenä*

Feature branchit ja merge hell

- ▶ Monet pitävät feature brancheja versionhallinnan *parhaana käytänteenä*
- ▶ Feature branchit aiheuttavat helposti pahoja *merge-konflikteja* sprintin lopussa

Feature branchit ja merge hell

- ▶ Monet pitävät feature brancheja versionhallinnan *parhaana käytänteenä*
- ▶ Feature branchit aiheuttavat helposti pahoja *merge-konflikteja* sprintin lopussa
- ▶ Seurausena pienimuotoinen integraatiohelvetti: *merge hell*

Feature branchit ja merge hell

- ▶ Monet pitävät feature brancheja versionhallinnan *parhaana käytänteenä*
- ▶ Feature branchit aiheuttavat helposti pahoja *merge-konflikteja* sprintin lopussa
- ▶ Seurausena pienimuotoinen integraatiohelvetti: *merge hell*
- ▶ Martin Fowler kuuluisassa artikkelissa Continuous integration: *Everyone Commits To the Mainline Every Day*

Feature branchit ja merge hell

- ▶ Monet pitävät feature brancheja versionhallinnan *parhaana käytänteenä*
- ▶ Feature branchit aiheuttavat helposti pahoja *merge-konflikteja* sprintin lopussa
- ▶ Seurausena pienimuotoinen integraatiohelvetti: *merge hell*
- ▶ Martin Fowler kuuluisassa artikkelissa Continuous integration: *Everyone Commits To the Mainline Every Day*
- ▶ Voidaanko edes puhua jatkuvasta integraatiosta jos feature branchit ovat käytössä?

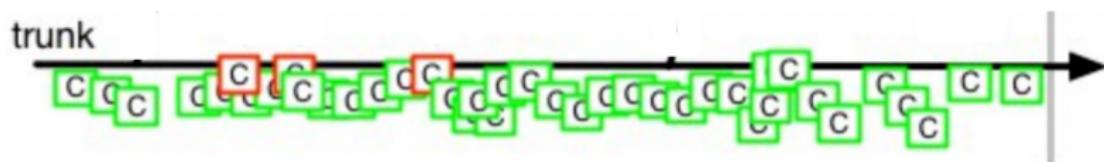
Trunk based development

Trunk based development

- ▶ Uusi trendi *trunk based development*: pitkäikäisiä feature brancheja ei käytetä ollenkaan
 - ▶ Kaikki koodi suoraan pääkehityshaaraan
 - ▶ ... josta käytetään nimitystä *trunk*

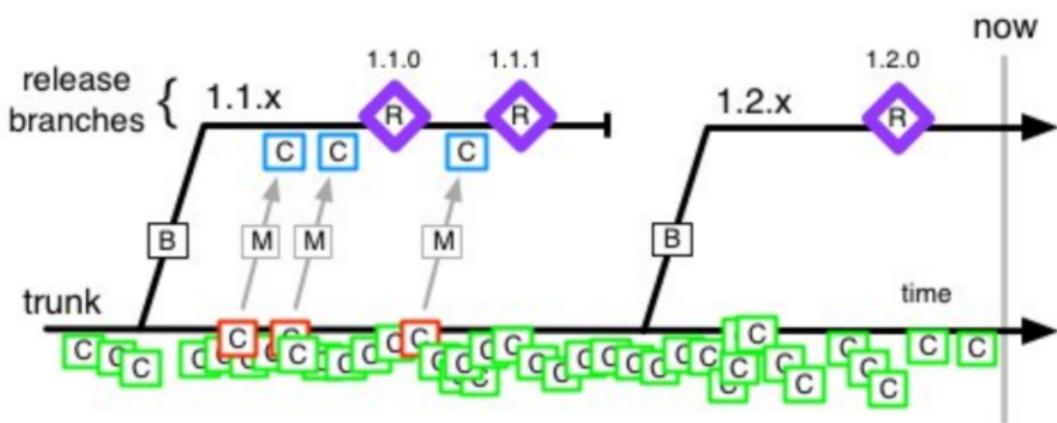
Trunk based development

- ▶ Uusi trendi *trunk based development*: pitkäikäisiä feature brancheja ei käytetä ollenkaan
 - ▶ Kaikki koodi suoraan pääkehityshaaraan
 - ▶ ... josta käytetään nimitystä *trunk*



Trunk based development

- ▶ Uusi trendi *trunk based development*: pitkäikäisiä feature brancheja ei käytetä ollenkaan
 - ▶ Kaikki koodi suoraan pääkehityshaaraan
 - ▶ ... josta käytetään nimitystä *trunk*



- ▶ Ohjelmiston kustakin julkaistusta versiosta saatetaan tehdä oma *release branch*

Trunk based development

- ▶ Pakottaa sovelluskehittäjät tekemään pieniä, nopeasti päähaaraan mergettäviä muutoksia

Trunk based development

- ▶ Pakottaa sovelluskehittäjät tekemään pieniä, nopeasti päähaaraan mergettäviä muutoksia
- ▶ Käytetään feature toggleja
 - ▶ puolivalmiitakin ominaisuuksia voidaan helposti ohjelmoida päähaaraan ilman toiminnallisuuden rikkomista

Trunk based development

- ▶ Pakottaa sovelluskehittäjät tekemään pieniä, nopeasti päähaaraan mergettäviä muutoksia
- ▶ Käytetään feature toggleja
 - ▶ puolivalmiitakin ominaisuuksia voidaan helposti ohjelmoida päähaaraan ilman toiminnallisuuden rikkomista
- ▶ Edellyttää sovelluskehittäjiltä *todella hyvää* kuria ja systemaattisuutta

Trunk based development

- ▶ Pakottaa sovelluskehittäjät tekemään pieniä, nopeasti päähaaraan mergettäviä muutoksia
- ▶ Käytetään feature toggleja
 - ▶ puolivalmiitakin ominaisuuksia voidaan helposti ohjelmoida päähaaraan ilman toiminnallisuuden rikkomista
- ▶ Edellyttää sovelluskehittäjiltä *todella hyvää* kuria ja systemaattisuutta
- ▶ Kehitysmallia noudattavat esim. Google, Facebook, Netflix ja *GitHub*

TAUKO 10 min, ehkä nyt

Dev vs Ops

- ▶ Jatkuva toimittaminen ja toimitusvalmius (CD) sekä tuotannossa testaaminen on haastavaa

Dev vs Ops

- ▶ Jatkuva toimittaminen ja toimitusvalmius (CD) sekä tuotannossa testaaminen on haastavaa
- ▶ Perinteisesti tarkka erottelu *sovelluskehittäjien* (developers, dev) ja *ylläpitäjien* (operations, ops) välillä
 - ▶ sovelluskehittäjät eivät pääse kirjautumaan tuotantopalvelimille
 - ▶ tuotantoon vieminen ja tietokantaan skeeman päivitykset tapahtuvat ylläpitäjien toimesta

Dev vs Ops

- ▶ Jatkuva toimittaminen ja toimitusvalmius (CD) sekä tuotannossa testaaminen on haastavaa
- ▶ Perinteisesti tarkka erottelu *sovelluskehittäjien* (developers, dev) ja *ylläpitäjien* (operations, ops) välillä
 - ▶ sovelluskehittäjät eivät pääse kirjautumaan tuotantopalvelimille
 - ▶ tuotantoon vieminen ja tietokantaan skeeman päivitykset tapahtuvat ylläpitäjien toimesta
- ▶ Jos näin on, tuotantopalvelimelle pystytään viemään uusia versioita vain harvoin, esim 4 kertaa vuodessa

Dev vs Ops

- ▶ Jatkuva toimittaminen ja toimitusvalmius (CD) sekä tuotannossa testaaminen on haastavaa
- ▶ Perinteisesti tarkka erottelu *sovelluskehittäjien* (developers, dev) ja *ylläpitäjien* (operations, ops) välillä
 - ▶ sovelluskehittäjät eivät pääse kirjautumaan tuotantopalvelimille
 - ▶ tuotantoon vieminen ja tietokantaan skeeman päivitykset tapahtuvat ylläpitäjien toimesta
- ▶ Jos näin on, tuotantopalvelimelle pystytään viemään uusia versioita vain harvoin, esim 4 kertaa vuodessa
- ▶ Joustavammat toimintamallit vaativat kulttuurinmuutoksen

DevOps

- ▶ *DevOps*: toimintamalli missä kehittäjät (dev) ja ylläpito (ops) työskentelevät tiiviisti yhdessä
 - ▶ Sovelluskehittäjille pääsy tuotantopalvelimelle
 - ▶ Scrum-tiimiin sijoitetaan ylläpitovastuilla olevia ihmisiä

DevOps

- ▶ *DevOps*: toimintamalli missä kehittäjät (dev) ja ylläpito (ops) työskentelevät tiiviisti yhdessä
 - ▶ Sovelluskehittäjille pääsy tuotantopalvelimelle
 - ▶ Scrum-tiimiin sijoitetaan ylläpitovastuilla olevia ihmisiä
- ▶ DevOps on hypetermi, jonka merkitys osin epäselvä
 - ▶ työpaikkailmoituksissa voidaan arvostaa DevOps-taitoja
 - ▶ tai etsiä ihmistä DevOps-tiimiin
 - ▶ myynnissä mitä erilaisempia DevOps-työkaluja

DevOps

- ▶ *DevOps*: toimintamalli missä kehittäjät (dev) ja ylläpito (ops) työskentelevät tiiviisti yhdessä
 - ▶ Sovelluskehittäjille pääsy tuotantopalvelimelle
 - ▶ Scrum-tiimiin sijoitetaan ylläpitovastuilla olevia ihmisiä
- ▶ DevOps on hypetermi, jonka merkitys osin epäselvä
 - ▶ työpaikkailmoituksissa voidaan arvostaa DevOps-taitoja
 - ▶ tai etsiä ihmistä DevOps-tiimiin
 - ▶ myynnissä mitä erilaisempia DevOps-työkaluja
- ▶ Järkevä määritelmä: *DevOps kehittäjien ja järjestelmälläpidon yhteinen työnteron tapa*, DevOps-kulttuuri

DevOps

- ▶ *DevOps*: toimintamalli missä kehittäjät (dev) ja ylläpito (ops) työskentelevät tiiviisti yhdessä
 - ▶ Sovelluskehittäjille pääsy tuotantopalvelimelle
 - ▶ Scrum-tiimiin sijoitetaan ylläpitovastuilla olevia ihmisiä
- ▶ DevOps on hypetermi, jonka merkitys osin epäselvä
 - ▶ työpaikkailmoituksissa voidaan arvostaa DevOps-taitoja
 - ▶ tai etsiä ihmistä DevOps-tiimiin
 - ▶ myynnissä mitä erilaisempia DevOps-työkaluja
- ▶ Järkevä määritelmä: *DevOps kehittäjien ja järjestelmälläpidon yhteinen työnteron tapa*, DevOps-kulttuuri
- ▶ Työkaluja/asioita jotka liittyvät DevOpsiin:
 - ▶ automatisoitu testaus
 - ▶ jatkuva integraatio ja toimittaminen (CI/CD)
 - ▶ virtualisointi ja kontainerisointi (Docker)
 - ▶ infrastructure as code
 - ▶ pilvipalveluna toimivat palvelimet ja sovellusympäristöt (PaaS, IaaS, SaaS)

- ▶ Monet listatuista kehittyneet viimeisen 5-10 vuoden aikana ja mahdollistaneet DevOpsin helpomman soveltamisen

- ▶ Monet listatuista kehittyneet viimeisen 5-10 vuoden aikana ja mahdollistaneet DevOpsin helpomman soveltamisen
- ▶ Eräs tärkeimmistä DevOpsia mahdollistavista asioista *infrastructure as code*
 - ▶ fyysisen palvelinten sijaan virtuaalisia ja pilvessä toimivia palvelimia, joita voi konfiguroida "ohjelmoimalla"

- ▶ Monet listatuista kehittyneet viimeisen 5-10 vuoden aikana ja mahdollistaneet DevOpsin helpomman soveltamisen
- ▶ Eräs tärkeimmistä DevOpsia mahdollistavista asioista *infrastructure as code*
 - ▶ fyysisen palvelinten sijaan virtuaalisia ja pilvessä toimivia palvelimia, joita voi konfiguroida "ohjelmoimalla"
- ▶ Raudastakin on tullut "koodia"
 - ▶ palvelinten konfiguraatioita voidaan tallettaa versionhallintaan ja jopa testata
 - ▶ sovelluskehitys ja ylläpito ovat alkaneet muistuttaa toisiaan

- ▶ Monet listatuista kehittyneet viimeisen 5-10 vuoden aikana ja mahdollistaneet DevOpsin helpomman soveltamisen
- ▶ Eräs tärkeimmistä DevOpsia mahdollistavista asioista *infrastructure as code*
 - ▶ fyysisen palvelinten sijaan virtuaalisia ja pilvessä toimivia palvelimia, joita voi konfiguroida "ohjelmoimalla"
- ▶ Raudastakin on tullut "koodia"
 - ▶ palvelinten konfiguraatioita voidaan tallettaa versionhallintaan ja jopa testata
 - ▶ sovelluskehitys ja ylläpito ovat alkaneet muistuttaa toisiaan
- ▶ Työkalujen käyttöönotto ei riitä, DevOpsin "tekeminen" lähtee kulttuurisista tekijöistä, tiimirakenteista, sekä asioiden sallimisesta

The Big DevOps Misunderstanding (Oliver Wolf)

The Big DevOps Misunderstanding (Oliver Wolf)

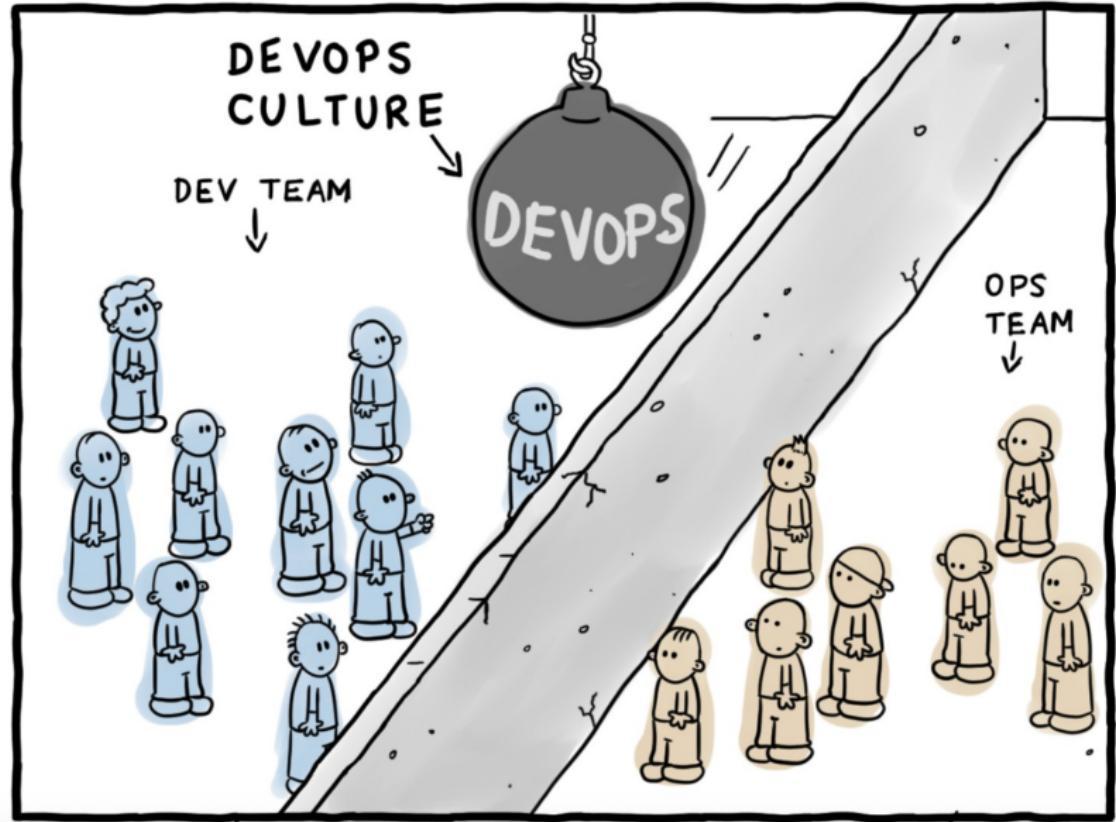
- ▶ When the term DevOps came up, it was a very simple idea:
 - ▶ You build it, you run it — Werner Vogels

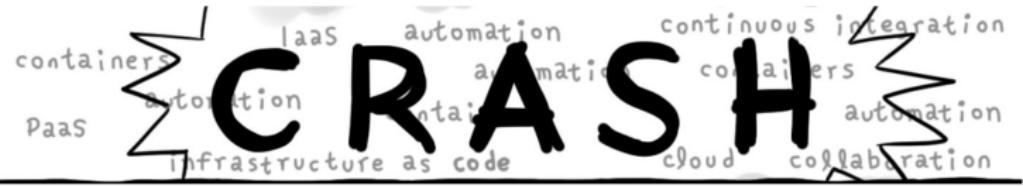
The Big DevOps Misunderstanding (Oliver Wolf)

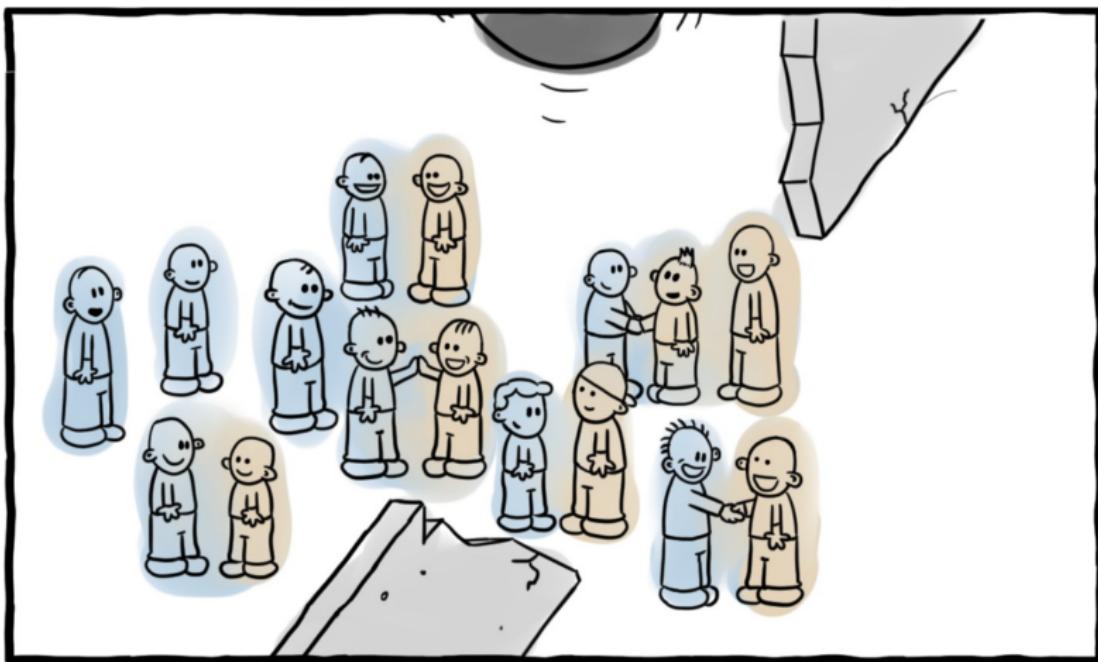
- ▶ When the term DevOps came up, it was a very simple idea:
 - ▶ You build it, you run it — Werner Vogels
- ▶ Somewhere along the way, the idea was misunderstood and the wrong definition of DevOps became the right one
 - ▶ We now have specific roles or departments that write “infrastructure as code”

The Big DevOps Misunderstanding (Oliver Wolf)

- ▶ When the term DevOps came up, it was a very simple idea:
 - ▶ You build it, you run it — Werner Vogels
- ▶ Somewhere along the way, the idea was misunderstood and the wrong definition of DevOps became the right one
 - ▶ We now have specific roles or departments that write “infrastructure as code”
- ▶ This is not DevOps, but an evolution of Systems Operations (SysOps)







DANIEL STORI {TURNOFF.US}

DevOps: ketteryyys laajennettuna

DevOps: ketteryyys laajennettuna

- ▶ Scrumin ja ketterän eräs tärkeimmistä periaatteista on tehdä kehitystiimeistä itseorganisoituvia ja “cross functional”

DevOps: ketteryyys laajennettuna

- ▶ Scrumin ja ketterän eräs tärkeimmistä periaatteista on tehdä kehitystiimeistä itseorganisoituvia ja “cross functional”
- ▶ DevOps on keino viedä ketteryyttä askeleen pitemmälle
 - ▶ kehitystiimi pystyy viemään uudet toiminnallisuudet tuotantojärjestöön
 - ▶ ja jopa testaamaan sekä operoimaan niitä tuotannossa

DevOps: ketteryys laajennettuna

- ▶ Scrumin ja ketterän eräs tärkeimmistä periaatteista on tehdä kehitystiimeistä itseorganisoituvia ja “cross functional”
- ▶ DevOps on keino viedä ketteryyttä askeleen pitemmälle
 - ▶ kehitystiimi pystyy viemään uudet toiminnallisuudet tuotantojärjestöön
 - ▶ ja jopa testaamaan sekä operoimaan niitä tuotannossa
- ▶ DevOps siis laajentaa ketteryyden koskemaan myös järjestelmälläpitoa

DevOps: ketteryyys laajennettuna

- ▶ Scrumin ja ketterän eräs tärkeimmistä periaatteista on tehdä kehitystiimeistä itseorganisoituvia ja “cross functional”
- ▶ DevOps on keino viedä ketteryyttä askeleen pitemmälle
 - ▶ kehitystiimi pystyy viemään uudet toiminnallisuudet tuotantojärjestelmään
 - ▶ ja jopa testaamaan sekä operoimaan niitä tuotannossa
- ▶ DevOps siis laajentaa ketteryyden koskemaan myös järjestelmälläpitoa
- ▶ Asettaa sovelluskehittäjille lisää osaamisvaatimuksia
 - ▶ kehittäjien pitää hallita enenevissä määrin ylläpitoasioita

-11.1.2025 ja 1.3.2026-

<https://courses.mooc.fi/org/uh-cs/courses/devops-with-docker>



In this course you'll...

Learn the basics of containerization with Docker and relevant concepts such as image and volume.

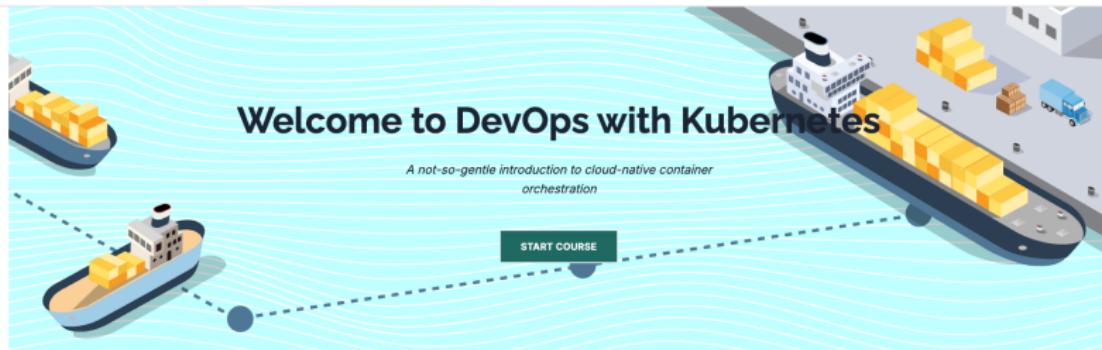
Get familiar with container orchestration with docker-compose and its usage in single host deployments.

Learn production-ready practices such as container optimization and deployment pipelines.

► <https://devopswithdocker.com/>

1.6.2026-

<https://courses.mooc.fi/org/uh-cs/courses/devops-with-kubernetes>



In this course you'll...

Learn how to create and run a Kubernetes cluster locally using k3d, and deploy applications to Kubernetes using deployments, services, ingress, and gateway resources.

Gain knowledge on advanced Kubernetes concepts such as Custom Resource Definitions (CRDs), service meshes, and GitOps for automated deployment pipelines.

Get familiar with implementing auto-scaling for applications and clusters in Kubernetes, and how to set up monitoring and logging using tools like Prometheus and Grafana.

► <https://devopswithkubernetes.com/>

TAUKO 10 min, viimeistäään nyt

Ohjelmiston elinkaaren vaiheet

- ▶ Riippumatta tyylistä ja tavasta jolla ohjelmisto tehdään, ohjelmistojen tekemiseen kuuluu
 - ▶ vaatimusten analysointi ja määrittely
 - ▶ **suunnittelu**
 - ▶ toteutus
 - ▶ testaus/laadunhallinta
 - ▶ ohjelmiston ylläpito

Ohjelmiston elinkaaren vaiheet

- ▶ Riippumatta tyylistä ja tavasta jolla ohjelmisto tehdään, ohjelmistojen tekemiseen kuuluu
 - ▶ vaatimusten analysointi ja määrittely
 - ▶ **suunnittelu**
 - ▶ toteutus
 - ▶ testaus/laadunhallinta
 - ▶ ohjelmiston ylläpito
- ▶ Suunnittelun tavoite *miten saadaan toteutettua vaatimusmäärittelyn mukaisella tavalla toimiva ohjelma*

Ohjelmiston elinkaaren vaiheet

- ▶ Riippumatta tyylistä ja tavasta jolla ohjelmisto tehdään, ohjelmistojen tekemiseen kuuluu
 - ▶ vaatimusten analysointi ja määrittely
 - ▶ **suunnittelu**
 - ▶ **toteutus**
 - ▶ testaus/laadunhallinta
 - ▶ ohjelmiston ylläpito
- ▶ Suunnittelun tavoite *miten saadaan toteutettua vaatimusmäärittelyn mukaisella tavalla toimiva ohjelma*
- ▶ Osa suunnittelusta tapahtuu vasta toteutusvaiheessa, joten suunnittelun ja toteuttamisen käsitteilyä ei voi eriyttää

Ohjelmiston suunnittelu

- ▶ Jakautuu kahteen vaiheeseen:
 - ▶ arkkitehtuurisuunnittelu
 - ▶ olio/komponenttisuunnittelu

Ohjelmiston suunnittelu

- ▶ Jakautuu kahteen vaiheeseen:
 - ▶ arkkitehtuurisuunnittelu
 - ▶ olio/komponenttisuunnittelu
- ▶ Ajoittuminen riippuu käytettävästä tuotantoprosessista:

Ohjelmiston suunnittelu

- ▶ Jakautuu kahteen vaiheeseen:
 - ▶ arkkitehtuurisuunnittelu
 - ▶ olio/komponenttisuunnittelu
- ▶ Ajoittuminen riippuu käytettävästä tuotantoprosessista:
- ▶ Vesiputoosmallissa vaatimusmäärittelyn jälkeen, ennen toteutuksen aloittamista
 - ▶ tarkasti dokumentoitu

Ohjelmiston suunnittelu

- ▶ Jakautuu kahteen vaiheeseen:
 - ▶ arkkitehtuurisuunnittelu
 - ▶ olio/komponenttisuunnittelu
- ▶ Ajoittuminen riippuu käytettävästä tuotantoprosessista:
- ▶ Vesiputoosmallissa vaatimusmäärittelyn jälkeen, ennen toteutuksen aloittamista
 - ▶ tarkasti dokumentoitu
- ▶ Ketterissä menetelmissä suunnittelua tehdään tarvittava määrä jokaisessa iteraatiossa
 - ▶ ei yleensä tarkkaa suunnitteludokumenttia

Ohjelmiston suunnittelu

- ▶ Jakautuu kahteen vaiheeseen:
 - ▶ arkkitehtuurisuunnittelu
 - ▶ olio/komponenttisuunnittelu
- ▶ Ajoittuminen riippuu käytettävästä tuotantoprosessista:
- ▶ Vesiputoosmallissa vaatimusmäärittelyn jälkeen, ennen toteutuksen aloittamista
 - ▶ tarkasti dokumentoitu
- ▶ Ketterissä menetelmissä suunnittelua tehdään tarvittava määrä jokaisessa iteraatiossa
 - ▶ ei yleensä tarkkaa suunnitteludokumenttia
- ▶ Vesiputoosmallin suunnitteluprosessi tuskin on enää käytössä
 - ▶ ”jäykimmässäkin” prosesseissa ainakin vaatimusmäärittely ja arkkitehtuurisuunnittelu limittyvät

Ohjelmiston suunnittelu

- ▶ Jakautuu kahteen vaiheeseen:
 - ▶ arkkitehtuurisuunnittelu
 - ▶ olio/komponenttisuunnittelu
- ▶ Ajoittuminen riippuu käytettävästä tuotantoprosessista:
- ▶ Vesiputoosmallissa vaatimusmäärittelyn jälkeen, ennen toteutuksen aloittamista
 - ▶ tarkasti dokumentoitu
- ▶ Ketterissä menetelmissä suunnittelua tehdään tarvittava määrä jokaisessa iteraatiossa
 - ▶ ei yleensä tarkkaa suunnitteludokumenttia
- ▶ Vesiputoosmallin suunnitteluprosessi tuskin on enää käytössä
 - ▶ "jäykimmässäkin" prosesseissa ainakin vaatimusmäärittely ja arkkitehtuurisuunnittelu limittyvät
- ▶ Näiden lisäksi UI/UX-suunnittelu

Ohjelmiston arkkitehtuuri

Ohjelmiston arkkitehtuuri

- ▶ Abstraktimpi kuvaus joka määrittelee ohjelmiston suuret linjat

Ohjelmiston arkkitehtuuri

- ▶ Abstraktimpi kuvaus joka määrittelee ohjelmiston suuret linjat
- ▶ IEEE: Ohjelmiston arkkitehtuuri on järjestelmän perusorganisaatio, joka sisältää
 - ▶ järjestelmän osat,
 - ▶ osien keskinäiset suhteet,
 - ▶ osien suhteet ympäristöön
 - ▶ sekä periaatteet, jotka ohjaavat järjestelmän suunnittelua ja evoluutiota

Ohjelmiston arkkitehtuuri

- ▶ **Ei-toiminnallisilla vaatimuksilla** suuri vaikutus arkkitehtuuriin
 - ▶ käytettävyyss, saavutettavuus
 - ▶ suorituskyky, skaalautuvuus
 - ▶ vikasietoisuus, tiedon ajantasaisuus
 - ▶ tietoturva
 - ▶ ylläpidettävyyss, laajennettavuus
 - ▶ hinta, time-to-market, ...

Ohjelmiston arkkitehtuuri

- ▶ **Ei-toiminnallisilla vaatimuksilla** suuri vaikutus arkkitehtuuriin
 - ▶ käytettävyys, saavutettavuus
 - ▶ suorituskyky, skaalautuvuus
 - ▶ vikasietoisuus, tiedon ajantasaisuus
 - ▶ tietoturva
 - ▶ ylläpidettävyys, laajennettavuus
 - ▶ hinta, time-to-market, ...
- ▶ Myös **toimintaympäristö** vaikuttavaa arkkitehtuuriin
 - ▶ integraatiot muihin järjestelmiin
 - ▶ käytettävät sovelluskehykset ja tietokannat
 - ▶ lainsääädäntö

Ohjelmiston arkkitehtuuri

- ▶ **Ei-toiminnallisilla vaatimuksilla** suuri vaikutus arkkitehtuuriin
 - ▶ käytettävyys, saavutettavuus
 - ▶ suorituskyky, skaalautuvuus
 - ▶ vikasietoisuus, tiedon ajantasaisuus
 - ▶ tietoturva
 - ▶ ylläpidettävyys, laajennettavuus
 - ▶ hinta, time-to-market, ...
- ▶ Myös **toimintaympäristö** vaikuttavaa arkkitehtuuriin
 - ▶ integraatiot muihin järjestelmiin
 - ▶ käytettävät sovelluskehykset ja tietokannat
 - ▶ lainsäädäntö
- ▶ Arkkitehtuuri syntyy joukosta *arkkitehtuurisia valintoja*
 - ▶ tradeoff

Arkkitehtuurityyli

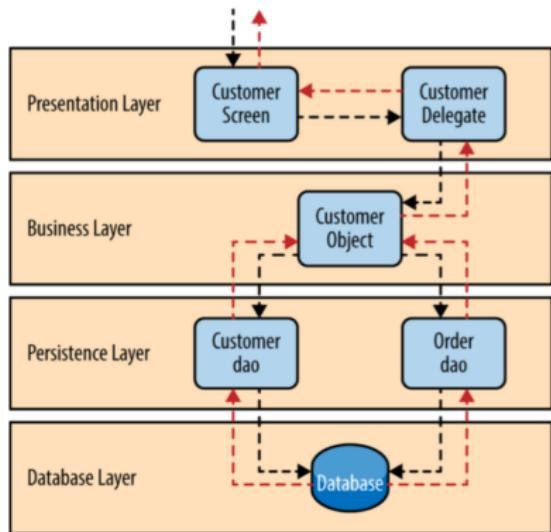
- ▶ Ohjelmiston arkkitehtuuri perustuu yleensä yhteen tai useampaan **arkkitehtuurityyliin**
 - ▶ hyväksi havaittua tapaa strukturoida tietyytyyppisiä sovelluksia

Arkkitehtuurityyli

- ▶ Ohjelmiston arkkitehtuuri perustuu yleensä yhteen tai useampaan **arkkitehtuurityyliin**
 - ▶ hyväksi havaittua tapaa strukturoida tietyytyyppisiä sovelluksia
- ▶ Tyylejä suuri määrä
 - ▶ Kerrosarkkitehtuuri
 - ▶ Mikropalveluarkkitehtuuri
 - ▶ MVC
 - ▶ Pipes-and-filters
 - ▶ Repository
 - ▶ Client-server
 - ▶ Publish-subscribe
 - ▶ Event driven
 - ▶ REST
 - ▶ ...

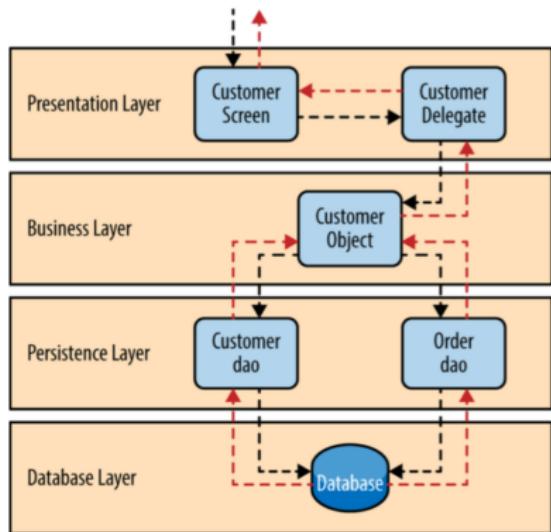
Kerrosarkkitehtuuri

- ▶ Kerros on kokoelma toisiinsa liittyviä olioita, jotka muodostavat toiminnallisuuden suhteiden loogisen kokonaisuuden



Kerrosarkkitehtuuri

- ▶ Kerros on kokoelma toisiinsa liittyviä olioita, jotka muodostavat toiminnallisuuden suhteiden loogisen kokonaisuuden



- ▶ Kerros käyttää ainoastaan alempaan olevan kerroksen palveluita

- ▶ Kerrokset omalla abstraktiotasollaan
 - ▶ UI vs tallennus

- ▶ Kerrokset omalla abstraktiotasollaan
 - ▶ UI vs tallennus
- ▶ Kerroksittaisuus helpottaa ylläpitoa

- ▶ Kerrokset omalla abstraktiotasollaan
 - ▶ UI vs tallennus
- ▶ Kerroksittaisuus helpottaa ylläpitoa
- ▶ Sovelluslogiikan riippumattomuus käyttöliittymästä helpottaa ohjelman siirtämistä uusille alustoille

- ▶ Kerrokset omalla abstraktiotasollaan
 - ▶ UI vs tallennus
- ▶ Kerroksittaisuus helpottaa ylläpitoa
- ▶ Sovelluslogiikan riippumattomuus käyttöliittymästä helpottaa ohjelman siirtämistä uusille alustoille
- ▶ Alimpien kerroksien palveluja, voidaan osin uusiokäyttää myös muissa sovelluksissa

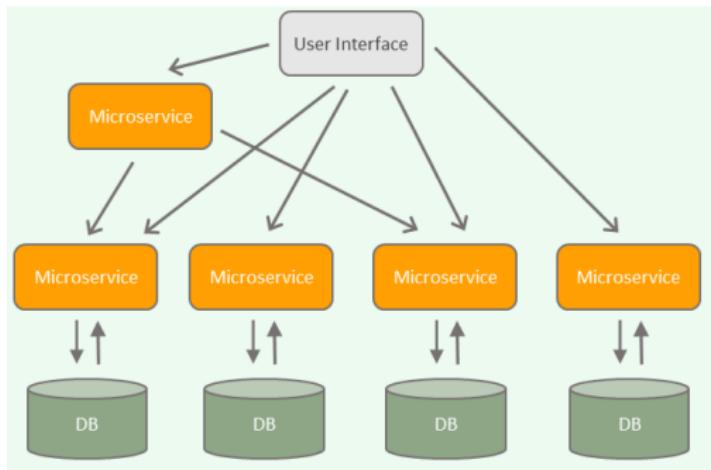
- ▶ Kerrokset omalla abstraktiotasollaan
 - ▶ UI vs tallennus
- ▶ Kerroksittaisuus helpottaa ylläpitoa
- ▶ Sovelluslogiikan riippumattomuus käyttöliittymästä helpottaa ohjelman siirtämistä uusille alustoille
- ▶ Alimpien kerroksien palveluja, voidaan osin uusiokäyttää myös muissa sovelluksissa
- ▶ Saattaa johtaa massiivisiin monoliittisiin sovelluksiin
 - ▶ vaikea laajentaa ja skaalaata suurille käyttäjämääritteille
 - ▶ haastavaa kehittää jos sovelluskehittäjiä suuri määrä

Mikropalveluarkkitehtuuri

- ▶ Mikropalveluarkkitehtuuri (microservice) pyrkii vastaamaan näihin haasteisiin

Mikropalveluarkkitehtuuri

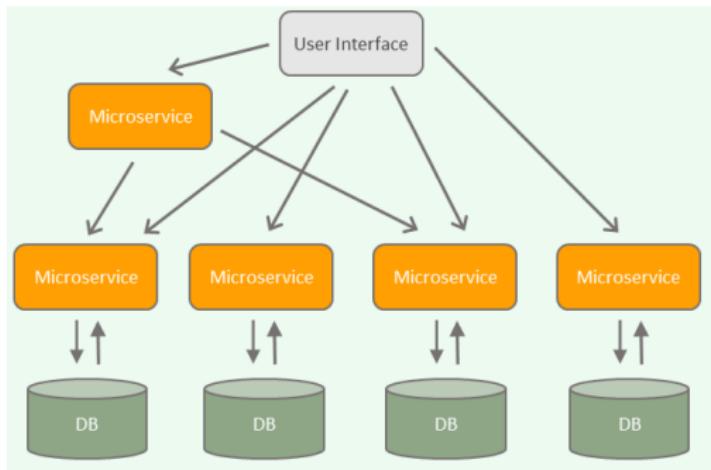
- Mikropalveluarkkitehtuuri (microservice) pyrkii vastaamaan näihin haasteisiin



- sovellus koostataan useista (jopa sadoista) pienistä verkossa toimivista autonomisista palveluista

Mikropalveluarkkitehtuuri

- ▶ Mikropalveluarkkitehtuuri (microservice) pyrkii vastaamaan näihin haasteisiin



- ▶ sovellus koostataan useista (jopa sadoista) pienistä verkossa toimivista autonomisista palveluista
- ▶ jotka keskenään verkon yli kommunikoiden toteuttavat järjestelmän toiminnallisuuden

Mikropalveluarkkitehtuuri

- ▶ Yksittäisistä palveluista pyritään tekemään mahdollisimman riippumattomia
 - ▶ palvelut eivät kutsu toistensa metodeja, kommunikointi aina verkon välityksellä
 - ▶ eivät käytä yhteistä tietokantaa
 - ▶ eivät jaa koodia

Mikropalveluarkkitehtuuri

- ▶ Yksittäisistä palveluista pyritään tekemään mahdollisimman riippumattomia
 - ▶ palvelut eivät kutsu toistensa metodeja, kommunikointi aina verkon välityksellä
 - ▶ eivät käytä yhteistä tietokantaa
 - ▶ eivät jaa koodia
- ▶ Mikropalvelut ovat pieniä ja huolehtivat vain ”yhdestä asiasta”

Mikropalveluarkkitehtuuri

- ▶ Yksittäisistä palveluista pyritään tekemään mahdollisimman riippumattomia
 - ▶ palvelut eivät kutsu toistensa metodeja, kommunikointi aina verkon välityksellä
 - ▶ eivät käytä yhteistä tietokantaa
 - ▶ eivät jaa koodia
- ▶ Mikropalvelut ovat pieniä ja huolehtivat vain ”yhdestä asiasta”
- ▶ Verkkokaupan mikropalveluita voisivat olla
 - ▶ käyttäjien hallinta
 - ▶ tuotteiden hakutoiminnot
 - ▶ tuotteiden suosittelu
 - ▶ ostoskorin toiminnallisuus
 - ▶ ostosten maksusta huolehtiva toiminnallisuus

Mikropalveluiden etuja

- ▶ Kun lisätään toiminnallisuutta: toteutetaan uusi palvelu tai laajennetaan ainoastaan *jotain* palvelua
 - ▶ Sovelluksen laajentaminen voi olla helpompaa kuin kerrosarkkitehtuurissa

Mikropalveluiden etuja

- ▶ Kun lisätään toiminnallisuutta: toteutetaan uusi palvelu tai laajennetaan ainoastaan *jotain* palvelua
 - ▶ Sovelluksen laajentaminen voi olla helpompaa kuin kerrosarkkitehtuurissa
- ▶ Skaalaaminen helpompaa kuin monoliittisten sovellusten
 - ▶ suorituskyvyn pullonkaulan aiheuttavia mikropalveluja voidaan suorittaa useita rinnakkain

Mikropalveluiden etuja

- ▶ Kun lisätään toiminnallisuutta: toteutetaan uusi palvelu tai laajennetaan ainoastaan *jotain* palvelua
 - ▶ Sovelluksen laajentaminen voi olla helpompaa kuin kerrosarkkitehtuurissa
- ▶ Skaalaaminen helpompaa kuin monoliittisten sovellusten
 - ▶ suorituskyvyn pullonkaulan aiheuttavia mikropalveluja voidaan suorittaa useita rinnakkain
- ▶ Sovellus voidaan helposti koodata monella ohjelmointikielellä ja sovelluskehysillä, toisin kuin monoliittisissä projekteissa

Mikropalveluiden etuja

- ▶ Kun lisätään toiminnallisuutta: toteutetaan uusi palvelu tai laajennetaan ainoastaan *jotain* palvelua
 - ▶ Sovelluksen laajentaminen voi olla helpompaa kuin kerrosarkkitehtuurissa
- ▶ Skaalaaminen helpompaa kuin monoliittisten sovellusten
 - ▶ suorituskyvyn pullonkaulan aiheuttavia mikropalveluja voidaan suorittaa useita rinnakkain
- ▶ Sovellus voidaan helposti koodata monella ohjelmointikielellä ja sovelluskehysillä, toisin kuin monoliittisissä projekteissa
- ▶ Työn jakaminen isolle kehittäjämääärälle helpompaa

Lähtökohta: The Bezos mandate

FROM: Jeff Bezos
TO: All Development
SUBJECT: Bezos Mandate



All teams will henceforth expose their data and functionality through service interfaces. Teams must communicate with each other through these interfaces.

There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.

It doesn't matter what technology they use.

All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.

Anyone who doesn't do this will be fired. Thank you; have a nice day!

Lähtökohta: The Bezos mandate

FROM: Jeff Bezos
TO: All Development
SUBJECT: Bezos Mandate



All teams will henceforth expose their data and functionality through service interfaces. Teams must communicate with each other through these interfaces.

There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.

It doesn't matter what technology they use.

All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.

Anyone who doesn't do this will be fired. Thank you; have a nice day!

Lähtökohta: The Bezos mandate

FROM: Jeff Bezos
TO: All Development
SUBJECT: Bezos Mandate



All teams will henceforth expose their data and functionality through service interfaces. Teams must communicate with each other through these interfaces.

There will be no other form of inter-process communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.

It doesn't matter what technology they use.

All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.

Said with a smile. Thank you; have a

Mikropalveluiden haasteita

- ▶ Sovelluksen jakaminen järkeviin mikropalveluihin on vaikeaa

Mikropalveluiden haasteita

- ▶ Sovelluksen jakaminen järkeviin mikropalveluihin on vaikeaa
- ▶ Testaaminen ja debuggaus voi olla vaikeaa koska asioita tapahtuu niin monessa paikassa

Mikropalveluiden haasteita

- ▶ Sovelluksen jakaminen järkeviin mikropalveluihin on vaikeaa
- ▶ Testaaminen ja debuggaus voi olla vaikeaa koska asioita tapahtuu niin monessa paikassa
- ▶ Kymmenistä tai jopa sadoista mikropalveluista koostuvan ohjelmiston operoiminen on haastavaa
 - ▶ vaatii pitkälle menevää automatisointia

Mikropalveluiden haasteita

- ▶ Sovelluksen jakaminen järkeviin mikropalveluihin on vaikeaa
- ▶ Testaaminen ja debuggaus voi olla vaikeaa koska asioita tapahtuu niin monessa paikassa
- ▶ Kymmenistä tai jopa sadoista mikropalveluista koostuvan ohjelmiston operoiminen on haastavaa
 - ▶ vaatii pitkälle menevää automatisointia
- ▶ Mikropalveluiden menestyksekäs soveltaminen edellyttää vahvaa DevOps-kulttuuria

Mikropalveluiden haasteita

- ▶ Sovelluksen jakaminen järkeviin mikropalveluihin on vaikeaa
- ▶ Testaaminen ja debuggaus voi olla vaikeaa koska asioita tapahtuu niin monessa paikassa
- ▶ Kymmenistä tai jopa sadoista mikropalveluista koostuvan ohjelmiston operoiminen on haastavaa
 - ▶ vaatii pitkälle menevää automatisointia
- ▶ Mikropalveluiden menestyksekäs soveltaminen edellyttää vahvaa DevOps-kulttuuria
- ▶ Kehitetty massiivisiin järjestelmiin (mm Amazon, Netflix)
 - ▶ onko järkevä kaikkialla?