Arkkitehtuurityylejä

Luento 5

Oppimistavoitteet

- Arkkitehtuurityylejä
  - Microservices, jaettu tietovarasto, viestinvälitysarkkitehtuurit, vertaisverkkoarkkitehtuuri , map-reduce, (cloud)

LISÄÄ ARKKITEHTUURITYYLEJÄ

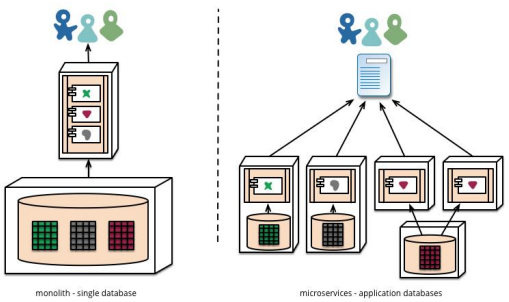
Microservices

- Nousussa oleva, vielä täsmentymätön tyyli yritysjärjestelmien palvelinpuolen arkkitehtuuriksi
  - Vastaveto monoliittisiksi koetuille perinteisille kolmikerrosarkkitehtuureille (N-tier)
  - Tukee Continuous X –käytäntöjä (continuous integration, deployment jne.)
- ~ Unixin *pipes and filters* ajattelutapa sovellettuna palvelujen toteuttamiseen

Microservices

- Perusidea: palvelin*sovellus*
  - Koostuu kokoelmasta pieniä, *erillisiä palveluita*
  - Jokaista palvelua ajetaan omassa prosessissaan
  - Palvelut kommunikoivat kevyitä mekanismeja käyttäen (verrattuna ESB-väyliin), esimerkiksi HTTP resurssi-API:eja
- Palvelu
  - Osituksen perusteena liiketoimintaprosessit ja –toiminnot (yksi per palvelu, *single responsibility principle*)
  - Voidaan ottaa käyttöön (deployment) *itsenäisesti* ja *automaattisesti*
- Minimimäärä pavelujen keskitettyä hallinnointia
  - Palvelujen toteutus mahdollista eri ohjelmointikielillä ja kirjastoilla
  - Palvelut voivat käyttää omia tiedonhallintaratkaisujaan
- Transaktioiden sijaan tarvitaan toisenlaisia menetelemiä tiedon eheyden takaamiseksi

Microservices



http://martinfowler.com/articles/microservices/images/decentralised-data.png

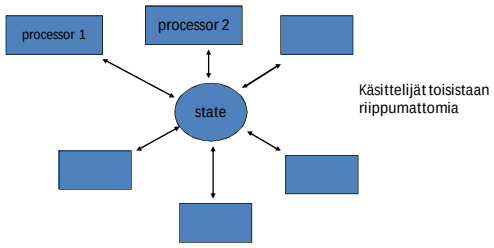
Microservices

- Martin Fowlerin sivut
  - http://martinfowler.com/microservices/
  - http://martinfowler.com/articles/microservices.html
- Katso esimerkiksi Fowlerin key-note esitys aiheesta GOTO Berlin 2014 –konferenssissa youtubesta
  - Linkki videoihin löytyy microservices – resurssisivulta (1. linkki yllä)

Jaettuun tietovarastoon perustuva arkkitehtuuri

- *Data centered - Shared state - Repository*
- Joukko komponentteja ylläpitää yhteistä tilaa tietovarastossa
  - Erilaisia muunnelmia sen mukaan, kuinka aktiivinen rooli tietovarastolla on
- Kahdenlaisia komponentteja
  - Keskitetty tietorakenne: tietovarasto
  - Asiakaskomponentit: hakevat tietoa tietovarastosta ja muokkaavat sitä
- Järjestelmän kontrolli määräytyy tietovaraston tilan mukaan

Jaetun tietovaraston arkkitehtuuri



Jaetun tietovaraston arkkitehtuuri

- Vuorovaikutus käsittelijöiden välillä tapahtuu ainoastaan tietovaraston kautta
- Käsittelijöiden tietovarastoon tekemät muutokset johtavat vaiheittain haluttuun lopputulokseen
- Käsittelijän aktivointi
  - Käsittelijät voivat pollata tietovarastoa tutkiakseen onko tila sellainen, mistä käsittelijä pystyy jatkamaan toimintaa
    - jos on, käsittelijä tuottaa tuloksia, jotka kirjaa tietovarastoon
  - Tietovarasto voi aktivoida käsittelijän jonkin säännön perusteella, esimerkiksi sisällön muuttumisen perusteella
    - vrt tietokantatriggerit

Jaetun tietovaraston arkkitehtuuri

- Käsittelijät voivat toimia rinnakkain
  - Edellyttää samanaikaisuuden hallintaa: tiedon lukitus → miten estetään yhtä käsittelijää varaamasta koko tietovarastoa itselleen (liian pitkäksi aikaa)?
- Esimerkkejä
  - Tekoälysovellukset (*blackboard*-arkkitehtuuri), Wikit, Google docs, MS Sharepoint, muut verkkotyöalustat…

Jaetun tietovaraston arkkitehtuuri

- Etuja
  - Muunneltavuus & laajennettavuus (itsenäiset, toisistaan riippumattomat käsittelijät)
  - Rinnakkaisuuden hyödyntäminen
- Haittoja
  - Rinnakkaisuuden ja päällekkäisten päivitysten hallinta on mietittävä tarkkaan

VIESTINVÄLITYSARKKITEHTUURIT

Viestinvälitysarkkitehtuurit

- Miten toteutetaan asiakkaiden ja palvelujen välinen kommunikointi hajautetussa, heterogeenisessä palveluympäristössä?
- Komponentit kommunikoivat lähettämällä toisilleen *viestejä*
- Viesti voidaan lähettää (1) tietylle kohteelle tai (2) yleislähetyksenä kaikille tunnetuile kohteille tai (3) kaikille kiinnostuneille kohteille
- Viestintä voi olla
  - synkronista (lähettäjä jää odotamaan) tai
  - asynkronista (lähettäjä jatkaa toimintaansa)

Viestinvälitysarkkitehtuurit

- Tyypillinen tilanne (roolit)
  - Julkaisija – Tilaaja (Publish – Subscribe)
  - Tuottaja – Kuluttaja (Supplier- Consumer)
- Yksinkertaisimmillaan *julkaisija* pitää kirjaa vastaanottajista ja tietosisällön muuttuessa välittää tiedon tilaajille proseduurikutsun välityksellä.
- *Tilaaja* rekisteröityy vastaanottajaksi ja toimittaa rekisteröinnin yhteydessä vastaanottorajapinnan (call-back)
- Yllä oleva toimii ohjelman (prosessin) sisäisesti
- Verkkoratkaisussa tarvitaan *kommunikointiprotokolla* ja *välittäjä* (*broker*) huolehtimaan viestinvälityksestä

Viestinvälitysarkkitehtuurit
Välittäjä

- *Välittäjä* (broker) tietää vastaanottajat (rekisteröityminen tai konfigurointi)
- Julkaisija toimittaa viestin välittäjälle
- Välittäjä toimittaa viestin edelleen siitä *kiinnostuneille* tilaajille
- Tilaaja käsittelee viestin
- Välittäjään (broker) ja asynkroniseen viestintään perustuvaa arkkitehtuuria voidaan käyttää yleisesti palveluarkkitehtuurin runkona ”perinteisemmän” etäproseduurikutsuihin perustuvan Asiakas-Palvelin arkkitehtuurin sijaan
  - Palveluekosysteemit

Viestinvälitysarkkitehtuurit
Välittäjä

- Arkkitehtuurissa määriteltävä seuraavat asiat:
  - Keskenään kommunikoivat komponentit
  - Viestit, joiden avulla kommunikointi tapahtuu ilman, että lähettäjä tuntee vastaanottajan (tai vastaanottaja lähettäjän) fyysistä sijaintia (protkolla, syntaksi)
  - Operaatiot, joilla komponentit reagoivat viesteihin (komponenttien ymmärtämä kieli, semantiikka)
  - Säännöt, joiden avulla komponentit ja viestit rekisteröidään järjestelmälle
  - Palvelutasolupaus (viestien elinaika, taattu toimitus jne.)

Viestinvälitysarkkitehtuurit
Välittäjä

- Säännöt, joiden perusteella välittäjä tietää, mille komponentille viesti on lähetettävä
  - Viestin vastaanottajan selvittäminen:
    - yleisviesti (multiple dispatch; viesti kaikille, jotkut käsittelevät, toiset eivät)
      - etu: vastaanottajan ei tarvitse etukäteen kertoa, mitä viestejä se haluaa vastaanottaa
    - komponentit kertovat rekisteröinnin yhteydessä, mitä viestejä (viestin tyyppi tai muu tunniste) ne haluavat vastaanottaa (tai keneltä)

Viestinvälitysarkkitehtuurit

- Rinnakkaisuus: missä määrin välittäjä ja komponentit toimivat rinnakkain
  - Viestinvälittäjillä ja vastaanottajilla viestijonot (välittäjän laatikko, vastaanottajan laatikko)
  - Voiko viestijono olla jaettu eri vastaanottajien kesken
  - Puskurointi, palvelutasot

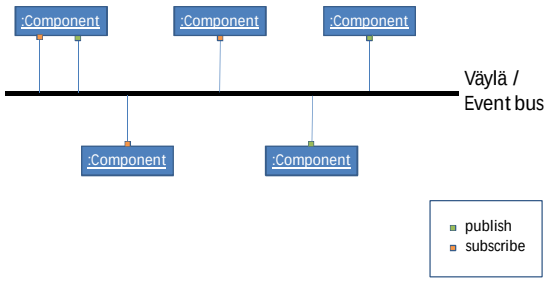
Esimerkki - *RabbitMQ*

- RabbitMQ on suosittu open-source viestinvälityspalvelu, joka tukee monia kieliä/ympäristöjä ja erilaisia viestinvälitys-protokollia http://www.rabbitmq.com
- Ominaisuuksiin palataan tarkemmin harjoituksissa

Tapahtumapohjaiset Viestinvälitysarkkitehtuurit

- *Tapahtumapohjaiset* arkkitehtuurit (*event-based*, *event driven*)
- Komponentit kommunikoivat aiheuttamalla tapahtumia (event)
- Tapahtumat voivat välittyä kaikille muille komponenteille, joista jotkut käsittelevät ja toiset eivät noteeraa
  - Tapahtumien lähetys voidaan myös rajoittaa julkaisija – tilaaja rakenteen tapaan – tapahtuma lähetetään vain tapahtuman tyypistä kiinnostuneille
  - komponentteja ei ole kuitenkaan lokeroitu joko julkaisijoiksi tai tilaajiksi vaan komponentti voi toimia kumpanakin

Tapahtumapohjainen viestintä

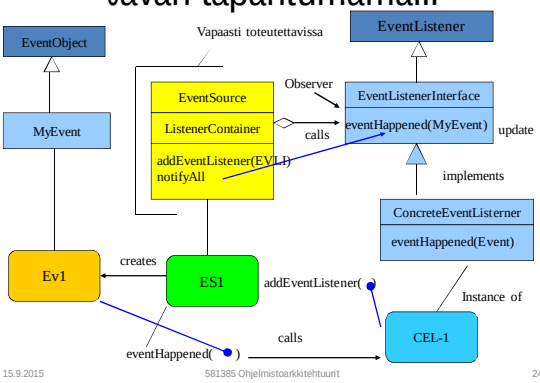


Tapahtumapohjaiset Viestinvälitysarkkitehtuurit

- Tapahtuma
  - Tilanne, joka voi sattua ohjelman suorituksen aikana
  - Edellyttää reagointia joiltain järjestelmän osilta
  - *Lähde* = tapahtuman synnyttävä komponentti
  - *Tarkkailija* = tapahtumaan reagoiva komponentti
- Lähde lähettää tapahtumailmoituksen sitä tarkkailemaan rekisteröityneelle tarkkailijalle
  - Esimerkiksi *Signals & Slots*[1] Qt-sovelluskehyksen olioarkkitehtuurissa
- Lähde tietää dynaamisesti tarkkailijoidensa olemassaolon, mutta ei niiden tarkkaa tyyppiä
  - Tapahtumaväylää käytettäessä lähteet eivät tiedä tarkkailijoistaan (vrt. MVC, Observer -mallit)
- Ilmoituksen lähetys voidaan hoitaa proseduurikutsuna tai sanomanvälityksenä (event bus, message bus)
  - proseduurikutsu synkroninen (esim Qt Signals & Slots)
  - Sanomajono asynkroninen/synkroninen

[1]http://en.wikipedia.org/wiki/Signals_and_slots, http://qt-project.org/doc/qt-5.0/qtcore/signalsandslots.html

Javan tapahtumamalli



Viestinvälitysarkkitehtuurit

- Etuja
  - Joustavuus, muunneltavuus, modulaarisuus
    - Uusien tuottajien/julkaisijoiden ja kuluttajien/tilaajien dynaaminen lisääminen
    - Välittäjän/väylän tekemät automaattiset esitystapamuunnokset, erilaisia palvelutasoja toteutettavissa (varmistettu perillemeno vs. fire and forget)
    - Komponenttien omatahtinen evoluutio
  - Paljon käytettyjen välittäjä-/väyläratkaisujen hyväksi kehittynyt laatu
    - Luotettavuus, skaalautuvuus, suorituskykyoptimoinnit, jne.

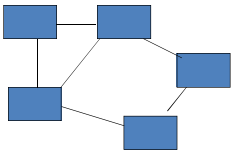
Viestinvälitysarkkitehtuurit

- Haittoja
  - Välittäjän tai tapahtumaväylän tuoma suoritusrasite (overhead) ja suorituskyvyn optimointimahdollisuuksien kaventuminen verrattuna suoriin (point-to-point) yhteyksiin
  - Välittäjä tai väylä on *kriittinen komponentti* (single point of failure), jonka luotettavuus pitää saada paljon korkeammaksi kuin kommunikoivien komponenttien

MUITA HAJAUTETTUJA ARKKITEHTUUREJA

Vertaisverkkoarkkitehtuuri

- *peer-to-peer*
- Verkko, jonka solmut voivat toimia sekä *asiakkaina* että *palvelimina*
- Uusi solmu kytkeytyy verkkoon liittymällä suoraan johonkin tuntemaansa verkon solmuun
  - Solmut voivat dynaamisesti liittyä ja poistua



Puhtaassa vertaisverkossa kaikki solmut ovat tasavertaisia
Verkko on *symmetrinen* ja *ei-hierarkinen*

Vertaisverkkoarkkitehtuuri

- Palvelupyyntö etenee verkossa solmusta solmuun kunnes löytyy solmu, joka kykenee täyttämään pyynnön
  - Pyytäjän ja pyynnön täyttäjän välille voidaan muodostaa suora yhteys - tai sitten ei
- Rakenteettomassa verkossa pyynnön eteneminen sokeaa
  - Solmut tuntevat suorat naapurinsa, mutta eivät näiden tarjoamia palveluita
- Rakenteellisessa solmut jakavat rakennetietoa palveluista ja tätä käytetään ohjaamaan etenemistä
- Edellyttää verkkoprotokollaa

Vertaisverkkoarkkitehtuuri

- Etuja
  - Saatavuus paranee, jos palvelu/resurssi on hajautettu useina (osa-)kopioina verkon solmuihin (esimerkiksi jonkin mediatiedoston fragmentit)
  - Vikasietoisuus kasvaa, koska yksittäisen solmun vikaantuminen ei ole kriittistä (tiedon/palvelun toisteisuus, vaihtoehtoiset saantipolut)
  - Skaalautuvuus ja laajennettavuus ovat hyvät, ei yksittäistä kriittistä komponenttia (no single point of failure)

Vertaisverkkoarkkitehtuuri

- Haittoja
  - Verkkoon voi muodostua saaria tai klikkejä, jonka solmut eivät ole yhteydessä klikin ulkopuolisiin solmuihin
  - Yksittäiset haut saattavat aiheuttaa paljon turhaa liikennettä (resurssia etsitään turhaan, tai se löytyy monista solmuista)
  - Verkon rakenne ei ole vakaa (solmuja tulee ja lähtee)

Vertaisverkkoarkkitehtuuri

- Vertaissolmujen rinnalle onkin usein pakko luoda pysyviä *Super-* tai *Ultra-solmuja,* jotka
  - Liittävät uusia solmuja verkkoon
  - Kytkeytyvät suoraan moniin alempiin solmuihin ja toisiinsa hakujen optimoimiseksi
  - Katso esimerkiksi Gnutella, Skype

Map-Reduce -arkkitehtuuri

- Motivaatio
  - Hyvin suuren datamäärän hajautettu tallennus ja prosessointi
  - Data ja käsittelyoperaatiot suhteellisen yksinkertaisia, mutta niitä on todella paljon
- Perusperiaate
  - Koko datamassa jaetaan pienempiin samankaltaisiin osiin (*splits*)
  - Osat käsitellään rinnakkain suoritettavissa tehtävissä, jotka tuottavat paikallisen välituloksen (*map*-operaatio)
  - Välitulokset yhdistetään globaaliksi tulokseksi (*reduce*-operaatio)
- Katso hyvä esimerkki (YouTube – *Intro To MapReduce* by MapRAcademy) http://www.youtube.com/watch?v=HFpIUBeBhcM

Map-Reduce -arkkitehtuuri

- Etuja
  - Skaalautuvuus, saatavuus, suorituskyky, varmistukset
    - Rinnakkaisen käsittelyn massiivinen hyödyntäminen
  - Katso esim. Rackspace –esimerkki 1. luennolta
- Huomattavaa
  - Suoritusnopeus riippuu siitä, miten split-operaatio onnistuu jakamaan syötedatan ”yhtä vaativiin” ja keskenään samantyyppisiin, rinnakkain käsiteltäviin palasiin
  - Reduce-operaatioita voidaan ketjuttaa, mutta monimutkaisten operaatioiden koordinointi voi osoittautua hankalaksi

Cloud

- Erittäin perusteellinen kuvaus pilvessä tapahtuvasta tietojenkäsittelyn perusteista patternien muodossa
  - http://www.cloudcomputingpatterns.org/Cloud_Computing_Patterns
- Kirjan voi lukea SpringerLink –e-kirjastossa ja ladata sieltä pdf:nä (kuuluu yliopiston tilaukseen)
- Perusteet pilvilaskennasta olisi hyvä jokaisen ohjelmistoarkkitehdin tuntea