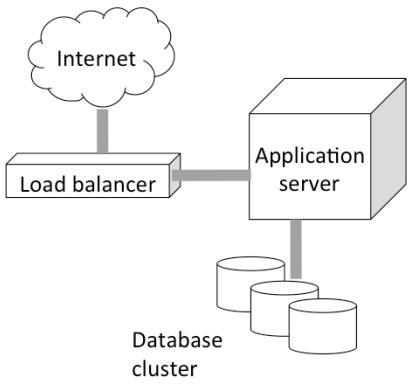
Cloud best practices

TIE-23600 Palvelupohjaiset järjestelmät

Cloud architectures: Transactional web application architecture

- Separation into
  - presentation,
  - business logic, and
  - data storage



[CloudArchitectures]

Siirtäminen pilveen voi tarkoittaa käytettävän ohjelmiston, tallennustilan tai infrastruktuurin siirtämistä verkkopohjaisiin palveluihin tai korvaamista niillä.

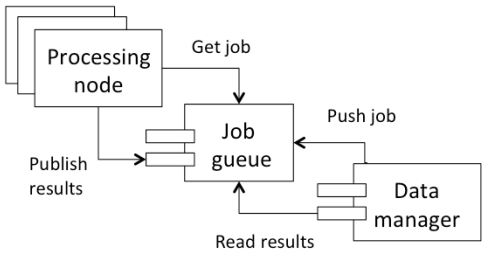
Arkkitehtuuri on olennainen asia, joka vaikuttaa siihen, miten sovelluksen siirtäminen pilveen onnistuu. Ensinnäkin sovelluksella tulee olla selkeä ja dokumentoitu arkkitehtuuri. Tällä ja seuraavalla Kalvolla on esitetty kaksi Web sovelluksille tyypillistä arkkitehtuurityyliä, jotka soveltuvat hyvin siirrettäväksi pilveen.

Web-sovellukset rakentuvat tyypillisesti tietovaraston varaan. Tällöin hyvä käytäntö on erottaa sovelluksen käyttöliittymä, business logiikka ja tietokanta toisistaan.

Grid-arkkitehtuurissa datan prosessointi on erotettu muusta sovelluslogiikasta. Tämä voidaan toteuttaa ns. jonoarkkitehtuurin avulla. Rinnakkain toimivat prosessointiyksiköt hakevat työjonosta töitä sitä mukaa kun ne ovat valmiita käsittelemään uuden työ.

Cloud architectures: Grid application architecture

- Separation of the core application from its data processing nodes



[CloudArchitectures]

Cloud best practices

1. Design for failure
2. Loose coupling
3. Implement “Elasticity”
4. Think Parallel
5. Build security in every layer - Design with security in mind
6. Don't fear constraints - Re-think architectural constraints
7. Leverage different storage options - One DOES NOT fit all

[CloudBestPractices]

Seuraavaksi tarkastellaan muutamia hyviä pilvijärjestelmien arkkitehtuuriratkaisuja (kohdat 1-4).

Kohdat 5-7- kuvaavat muita hyviä toimintatapoja:

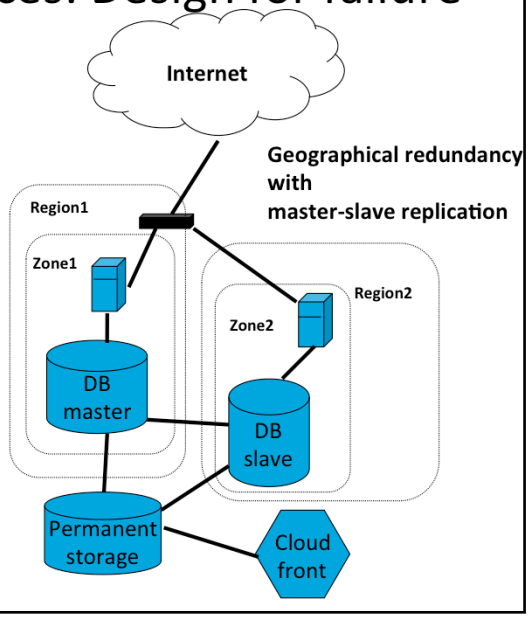
5. Tietoturvan suhteen tulisi noudattaa ns. onion model -mallia, eli tietoturvaan tulee kiinnittää huomiota kaikilla sovelluksen tasoilla (tiedon tallennus, tiedon siirto, käyttöoikeudet, jne.)

6. Vanhat tutut ratkaisut eivät välttämättä päde pilviympäristössä sellaisenaan, vaan jotkin asiat on tehtävä toisin.

7. Esimerkiksi Amazon WS tarjoaa monenlaisia tiedon tallennus paikkoja. Ne on optimoitu eri käyttötarkoituksiin (esim. haut, turvallinen säilytys, jne.). Joista kaikki ei ole suinkaan tarkoitettu pysyväksi säilytys paikaksi. Tämä tulee ottaa huomioon sovelluksen suunnittelussa. Myös sijainnilla on Väliä (saantiviive).

Cloud best practices: Design for failure

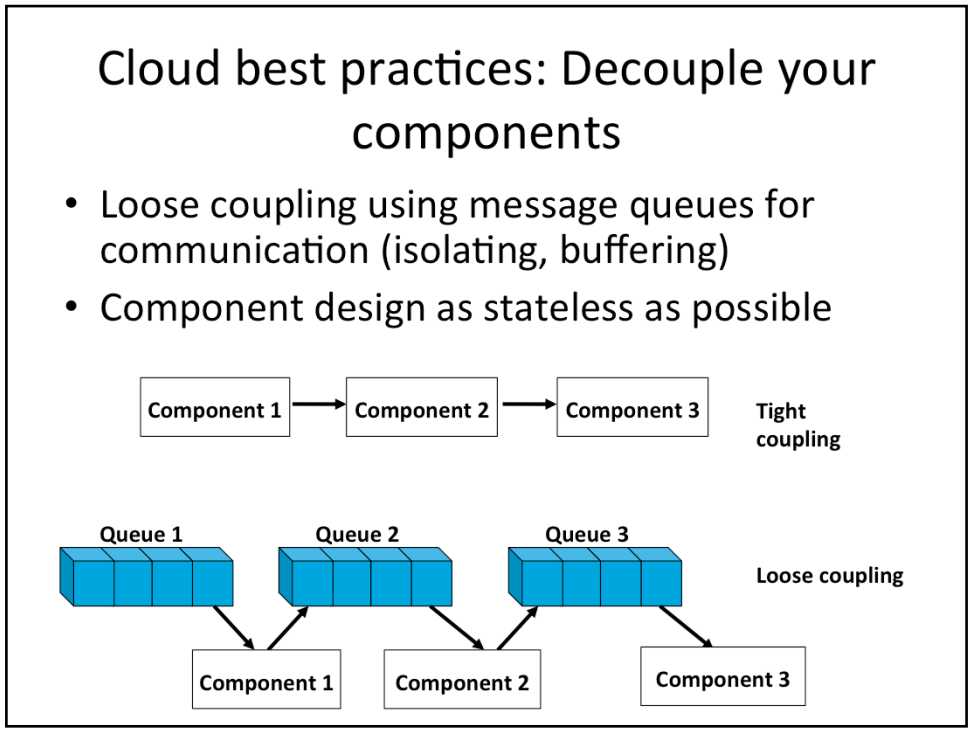
- No single points of failure
  - Replication
  - Monitoring
  - Load balancing
  - Backups
  - Snapshots,
  - …



”Designed to fail” vai ”Design for failure”? Ei tulisi olla yhtä yksittäistä kohtaa, joka virhetilanteessa vaarantaa koko järjestelmän toiminnan. Replikointi on pilviympäristössä yksi tapa välttää edellä mainittu tilanne. Esimerkiksi master ja slave tietokannat voidaan sijoittaa fyysisesti eri puolille maailmaa (maantieteellinen replikointi). Amazon WS ympäristössä virtuaalikoille määritetään aina region, joka vastaa palvelun maantieteellistä sijaintia (Eurooppa, Yhdysvallat jne.)

Ns. snapshot on tietyllä hetkellä tallennettu toimiva konfiguraatio, joka voidaan palauttaa virhetilanteessa.

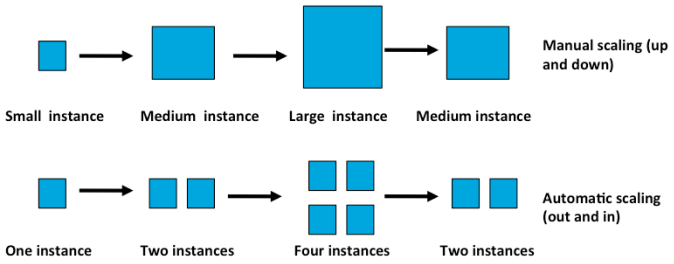
Fyysinen sijainti vaikuttaa myös saantiviiveeseen (latency): prosessoitava tieto säilytetään sielä, missä laskenta tapahtuu ja pysyvä tallennuspaikka puolestaan lähellä asiakasta.



Komponenttien välisiä riippuvuuksia voidaan vähentää käyttämällä viestijonoa. Sen sijaan, että komponentit kommunikoisivat suoraan keskenään, ne kommunikoivat jonon välityksellä. Viestijonon käyttö tukee myös mm. rinnakkaista töiden prosessointia kuten aiemmin esitetyssä Grid-sovellusarkkitehtuurissa.

Cloud best practices: Elasticity

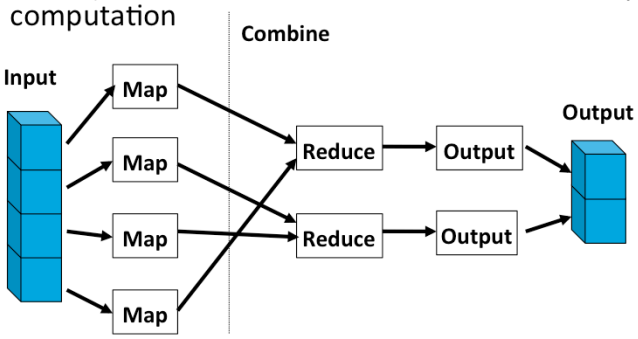
- Scaling (e.g. machine configurations, storage, computing capacity)
- Monitor system metrics
- Use load balancing tools
- Automatize
  - Scale based on variability in usage



Elastisuudella viitataan pilvijärjestelmien yhteydessä tallennus- ja laskentakapasiteetin skaalautuvuuteen kuorman kasvaessa ja pienentyessä. Käytössä olevien resurssien lisäksi tähän liittyy oleellisesti järjestelmän konfiguraation hallinta, monitorointi ja kuormantasaus. Pilvilaskennassa pyrkimys on mahdollistaa käytettävien resurssien skaalautuvuus automaattisesti. Kalvolla on esitetty ratkaisumalli, joka perustuu siihen, että eri kokoisten instanssien sijaan resurssien lukumäärää muutetaan kuorman mukaan. Esimerkiksi laskenta hajautetaan siten, että virtuaalikoneiden määrää voidaan muuttaa tarpeen mukaan. Seuraavalla kalvolla esitetään yksi hajautetun laskennan malli, jossa kaksi yksinkertaista ohjelmaa (map ja reduce) hajautetaan usealle eri (virtuaali)koneelle.

Cloud best practices: Parallel and distributed computing

- E.g., Create job flows using MapReduce
  - Designed for scalable processing of large amount of data
  - Automatic distribution of work load
  - Two simple programs, map(key, value) and reduce(key, values), are distributed in several machines for parallel computation



MapReduce on Googlen kehittämä arkkitehtuuri, jossa kaksi yksinkertaista ohjelmaa (map ja reduce) hajautetaan laskettavaksi rinnakkain useille koneille. Tarkoituksena on skaalautua käsittelemään valtavia datamääriä niin, että kehittäjän ei tarvitse huolehtia hajautukseen liittyvistä yksityiskohdista.

Sisääntulona voi olla mitä tahansa, mistä voidaan tehdä järkeviä avain-arvo pareja. Tieto virtaa sisääntulosta ensin mapin ja sitten reducen läpi ulostuloksi avain-arvo pareina. Avain-arvo-parien käyttö tekee map- ja reduce-ohjelmien rajapinnan yksinkertaiseksi. Myös tiedon hajautus ja yhdistys yksinkertaistuu. Map- ja reduce-funktioita ajetaan rinnakkain useita instansseja useilla koneilla (ensin map ja sitten reduce).

MapReduce

1. **Map**: is run to each key-value pair of the input and it produces a list of preliminary values

$\langle key_{input}, value_{input} \rangle \rightarrow \textbf{map} \rightarrow \langle key_{output}, value_{intermediate} \rangle$

2. **Sort/combine**: values are grouped according to keys

3. **Reduce** is run to a list of values for each key and it produces a list of final values

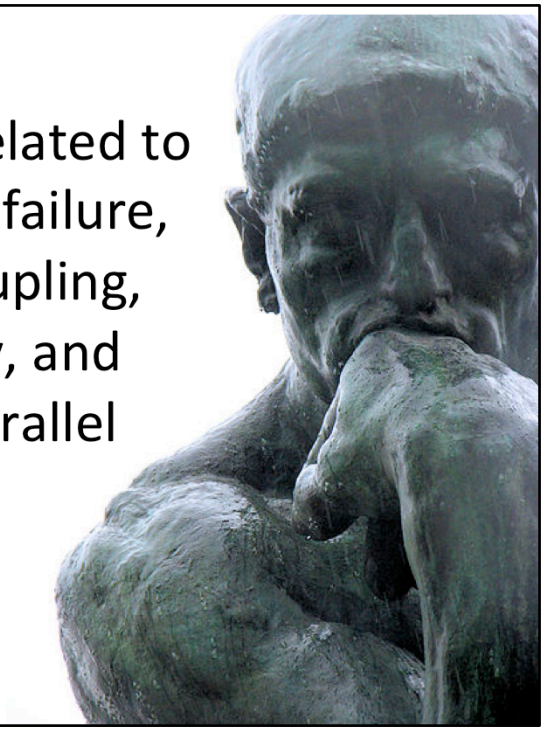
$\langle key_{output}, list(value_{intermediate}) \rangle \rightarrow \textbf{reduce} \rightarrow \langle key_{output}, list(value_{output}) \rangle$

[MapReduce]

MapReduce vaiheet yksinkertaistettuna:

1. map: suoritetaan jokaiselle sisääntulon avain-arvo parille ja se tuottaa väliaikaisia arvoja, joilla on jokin lopullinen avain
2. sort: ohjelmistokehys ryhmittelee arvot avaimien perusteella
3. reduce: reduce funktio suoritetaan ryhmälle arvoja joilla on sama avain ja se tuottaa näistä listan lopullisia arvoja

How SOA is related to
Design for failure,
Loose coupling,
Elasticity, and
Think Parallel



References

- [CloudArchitectures]
  - Cloud Application Architectures: Building Applications and Infrastructure in the Cloud, George Reese, O'Reilly Media, 2009.
- [CloudBestPractices]
  - Architecting for the Cloud: Best Practices: http://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf
- [Hadoop]
  - Open Source MapReduce, Apache Hadoop: http://hadoop.apache.org/
- [MapReduce]
  - MapReduce: Simplified Data Processing on Large Clusters: http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en/us/archive/mapreduce-osdi04.pdf