Vhodnost nasazení jednotlivých webových architektur, sdílení dat, perzistence, webové služby a REST, asynchronnost, messaging

1. Vhodnost nasazení jednotlivých webových architektur

- toto je podle Klímy spíš diskusní téma, to znamená použít zdravý rozum a být schopen přemýšlet, na co se hodí cloud
- třídy aplikací a způsoby jejich nasazení:
  - desktop
  - klasické webové aplikace na aplikačním serveru
  - cloudová řešení

- která řešení by měla běžet ve webovém prostředí?
- rozdíly instalovaného SW a webové aplikace / služby popsáno v otázce 21

- **příklad**:
  - elektronické bankovnictví, které má 20 let starý back-end legacy system naprogramovaný v COBOLu
  - klienti přistupují přes webové JEE (JSP) rozhraní
  - zvážit architekturu systému vzhledem k nárůstu počtu klientů nebo růstu banky
  - zvážit privátní cloud vs. existující provideři AWS, Google, MS Azure
  - další příklady: eshop, náročné vědecké výpočty, něco jako youtube

2. Sdílení dat, perzistence

- hlavně relační DB, protože objektové nejsou moc cool
- nevýhody přímého použití JDBC

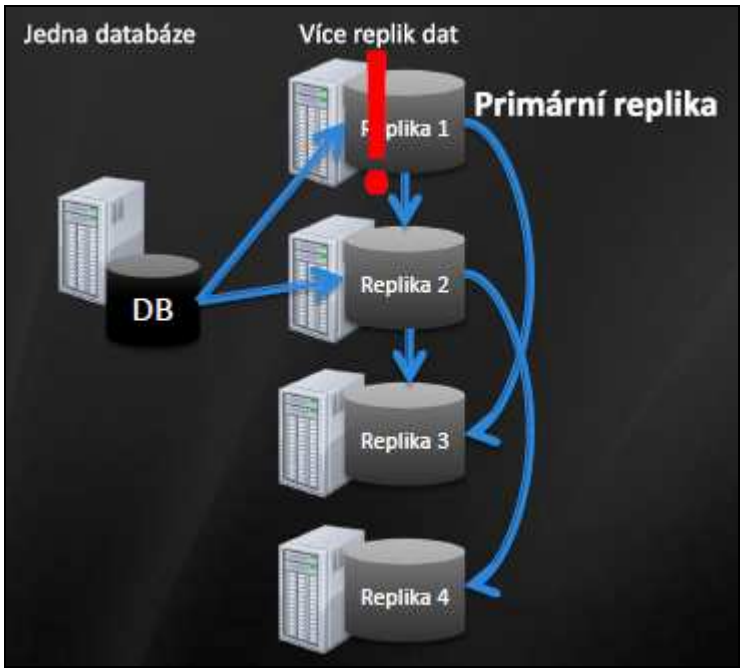
ORM

- blíže **OO** paradigmatu
- **JPA** (konfigurace anotacemi), **Hibernate** (konfigurace v XML), iBatis (konfigurace v XML)
- ORM by měl být neintruzivní - pracuje přímo s **POJO** (implicitní konstruktor, settery a gettery, ne final atributy)
- o perzistenci se starají třídy EntityManager nebo Hibernate Session
- **podpora ISA hierarchie**:
  - SINGLE_TABLE - vše v jedné tabulce + rozlišující sloupec
  - TABLE_PER_CLASS - každá entita má vlastní tabulku s celou sadou atributů
  - JOINED - hlavní tab. má základní attrib., ostatní se s ní spojují (slabé entity)
- podpora **vztahů** (1:1, 1:N, M:N)
- dialekty SQL
- **operace** persist, find, merge, remove, query

3. Cloudová řešení perzistence

- poskytovatelé: Google App Engine, Amazon Web Services, Microsoft Azure

- těží z výhod cloudu: **škálování, replikace, dump**, reporty
- **transakce**, **dotazy**, téměř 100% uptime, konzistence
- administrace přes webové rozhraní nebo klienty, command line
- veřejné **API** (např. Google DataStore), které využívá ORM jako JPA nebo JDO
- přístup zabezpečen autentizací
- platí se jen za uskladněná / přenesená data
- data jsou uložena v rozdílných geografických lokalitách (replikace a recovery)



Obrázek 1 - Primární a sekundární replika

Google App Engine DataStore

- ukládání strukturovaných dat (entit)
- není to relační DB, entita se ukládá jako key-pair value
- DB nemá schéma
- nezajišťuje referenční integritu
- rozhraní JDO a JPA

Google App Engine BlobStore

- pro ukládání velkého množství dat (např. souborů)
- soubor se uloží jako blob poté, co byl uploadován přes HTTP
- každý blob až 2GB
- blob je uložen pod unikátním klíčem
- blob nelze měnit, pouze smazat

Amazon Web Services Simple Storage

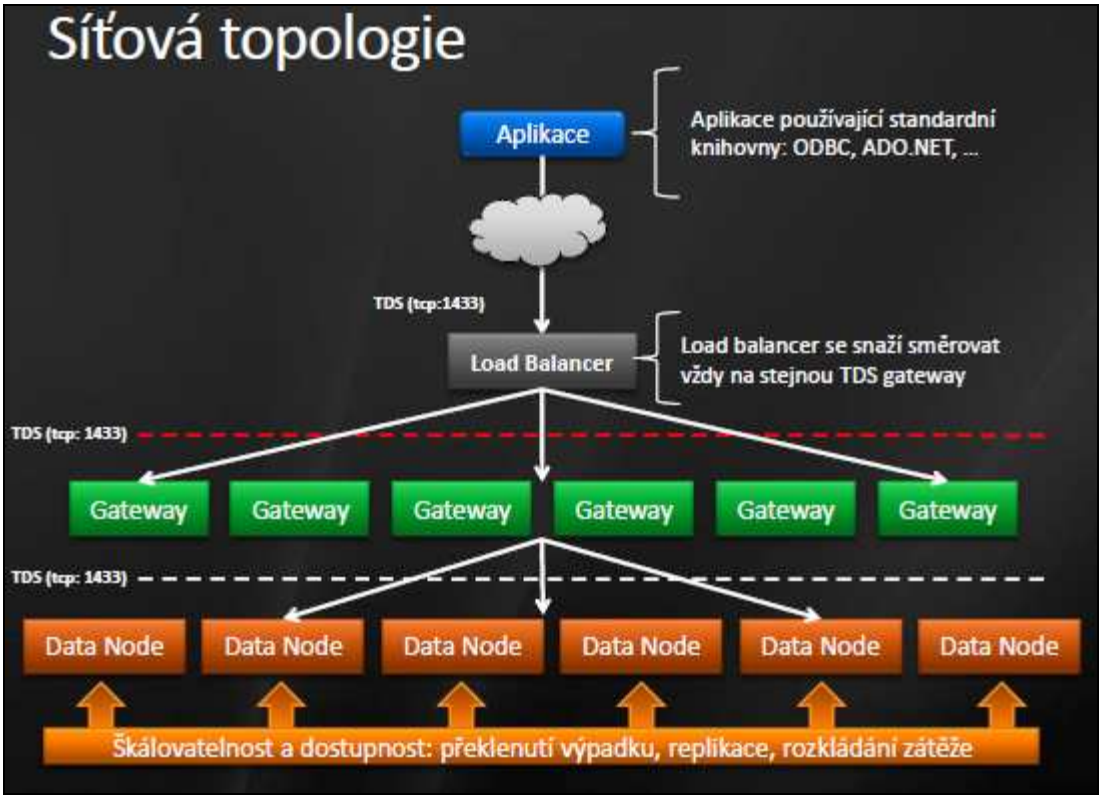
- obdoba GAE BlobStore
- umožňuje sdružovat jednotlivé objekty (1B až 5GB) do skupin (buckets) a těmto skupinám nastavovat práva pro přístup
- každý objekt identifikován pomocí unikátního klíče
- REST a SOAP rozhraní pro přístup

Amazon Relational Database Service
- plnohodnotná relační DB kompatibilní s MySQL

Amazon Simple DB
- strukturované úložiště podobné jako GAE DataStore

Microsoft Azure Storage
- obdoba GAE BlobStore

Microsoft SQL Azure
- relační DB
- obdoba AWS Relational Database Service
- synchronizace pomocí MS Sync framework
- **load balancer** vyvažuje výkon a směruje na gateway
- **gateway** je jakási proxy, která:
    - akceptuje klientská připojení
    - má na starosti bezpečnost a kontrolu paketů
    - dělá účtování za přenesená data (provisioning)
    - rozděluje requesty podle typu dotazu na DDL a DML
- **servery se škálují** podle potřeby



Obrázek 2 - Fyzická architektura SQL Azure

Microsoft Azure Table
- strukturované úložiště podobné jako GAE DataStore

Microsoft Azure Blob
- úložiště velkých objemů dat

- podobné jako AWS Simple Storage a GAE BlobStore

4. REST

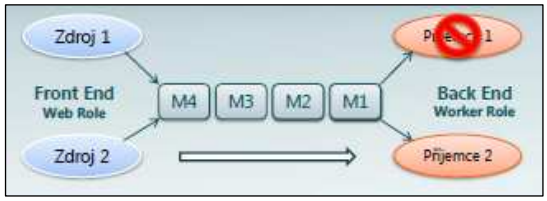
- architektonický styl pro distribuovaná media
- platformně nezávislé
- založeno na technologii **HTTP** a **URI**
- **cíl**: obecné aplikační rozhraní, možnost komunikace přes proxy
- manipulace s **prostředky** (informace, data, soubory)
- prostředky jsou beztypové
- prostředek je identifikován svým URI
- komunikace klient – server, klient i server jsou stateless
- linky obsahují sémantickou informaci, např. mujweb.cz/delete/person/321
- pro popis metadat se používají HTTP hlavičky
- **minimalizace závislostí** klienta a serveru
- server může cacheovat odpovědi
- klient neví, zda komunikuje se serverem nebo prostředníkem (proxy, cache)
- **Amazon** nabízí přístup k prostředkům přes SOAP a REST rozhraní
- typicky se používá pro **CRUD** operace:
  - create (metoda POST)
  - read (metoda GET)
  - update (metoda PUT)
  - delete (metoda DELETE)

5. Messaging

- synchronní (blokující) vs. asynchronní (neblokující)
- granularita
  - **mezi aplikacemi nebo front-endem a back-endem** (message driven beans, middle-ware, webservices, http, REST, sockety, AJAX) slouží zejména ke komunikaci a integraci aplikací
  - **mezi procesy** (cloudová řešení jako AWS Simple Queue Service, GAE Task Queue, MS Azure Queue) slouží zejména k dávkovém zpracování velkého množství dat, ale i na posílání jednoduchých zpráv

MS Azure Queue

- komunikace mezi službami pomocí zpráv
– zajišťuje **škálovatelnost** aplikace
– odděluje Front End (uživatelské rozhraní) od Back End (business logika)
– zajistí korektní funkci v exponovaných momentech (postupné zpracování)
- není vhodné pro dlouhodobé ukládání dat nebo objemná data
- zprávy jsou zpracovávány v libovolném pořadí
– zpráva může být zpracována vícekrát



6. Zdroje

[1] Přednášky z WA2.
http://edux.feld.cvut.cz/courses/A4M39WA2/_media/lectures

[2] Dokumentace Google App Engine DataStore
http://code.google.com/intl/cs/appengine/docs/java/datastore/jdo/relationships.html

[3] Dokumentace Google App Engine BlobStore
http://code.google.com/intl/cs/appengine/docs/java/blobstore/overview.html#Introducing_the_Blobstore