

Přednáška 1

- Slovo virtuální
 - Z latiny - virtus (virtue)
 - “Close to being something without being it”
 - Zavedené ve výpočetní technice
 - První použití “virtuální paměť”
- První era virtualizace v 70. letech IBM
 - Z různých důvodů to zaniklo
- Koncept vzkříšen až v 90. letech firmou VMware
- S nástupem Cloudů to získalo na významu (základ Cloudu)
- IBM tenkrát vyvíjelo HW spolu se SW, tak to měli připravené na tu virtualizaci
- Když VMware zkoušel vzkřísit virtualizaci narazilo se na HW problémy, protože HW byl starší a s tímto konceptem se nepočítalo
- Stalo se populární až někde v 2005, výrobci byli donuceni přidat HW podporu
- 1999-2005 obcházení tohoto problému obskurními metodami (později)
- Prakticky všechno, co v počítačích je má nějakou virtuální variantu (NIC, disky, ...)
- Chceme aby rozdíl mezi virtuálním a fyzickým prostorem byl co nejmenší (pokud možno aby to virtuální stroj nevěděl)
- Webhosting
 - Někdo chce oracle v7, někdo v8
 - Programy ani OS nejsou připravené na to, aby bylo možné mít nainstalováno více verzí stejného programu
 - Sdílené knihovny různých verzí
 - Nakonec stejně každý musel mít vlastní počítač → drahé, brzdilo rozvoj webu
- Hypervisor = to co spravuje virtuální stroje
 - Stará se o to, aby vznikla iluze virtuálních počítačů (nižší úroveň)
 - Někáká management console pro adminy (vyšší úroveň)
- Prakticky vždy potřebujeme virtualizovat NIC
 - NAT
 - Bridge
- Tenant
 - Ten kdo si pronajímá službu
 - Jiný člověk než vlastník HW
- Veřejný Cloud
 - Amazon - vlastník a správce HW
 - Nájemce - já
- Motivace pro virtualizaci
 - Víceru zákazníků nechce/nemůže sdílet tutéž mašinu
- Výhody virtualizace
 - Rychlá migrace stroje
 - Jednoduchá změna HW prostředků stroje
- Historicky všechny OS byly navrženy tak, aby umožnili komunikaci procesů
- Všechny mechanismy, které toto znemožňují byly vlastně dolepeny do OS → nestačí

- Kontejnerizace
 - Kernel se snaží lépe oddělit procesy od sebe
 - Zejména jmenné prostory
 - Procesy nevědí, že běží v kontejneru, nechceme aby se tím zabývali
 - Kernel o tom ví a spravuje to
 - Čísla portů, jména souborů, přidělování strojového času
- (Opravdová) virtualizace
 - To co provozovalo IBM
 - Oddělení mezi HW a SW
 - Ani kernel neví, že běží ve virtuálním stroji
 - Netriviální záležitost, potřebujeme podporu HW (jako měla IBM)
 - VT-x, AMD-v
 - Bez tohoto pouze firma VMware binární překlad (masakr)
- Paravirtualizace
 - Většina pokusů o virtualizaci do roku 2005 (2006 u AMD) spadala sem
 - Iluzi vytvoříme tak, že modifikujeme kernel VM
 - Jedna část virtualizace byl tedy hypervisor a druhá pak modifikovaný kernel VM
 - U Windowsu (logicky) musel paravirtualizaci implementovat Microsoft (stalo se)
 - Dnes už není tvrdý důvod pro paravirtualizaci
 - Stále se používá, umožní zvýšit výkon VM
- Z pohledu programátora aplikace je jedno jestli použijeme kontejnery nebo virtualizaci
- Z pohledu správce, ale rozdíl je značný, např. aktualizace kernelu
- Virtuální stroje byly stavěny opačně než OS, VM nemohly vůbec komunikovat, dnes přidáváme možnosti jak můžou nějak spolu komunikovat
 - Je blbost totiž vyrábět pakety a posílat je přes virtuální NIC ...
 - Když se celá komunikace odehrává v jednom paměťovém prostoru
- Virtualizace je daleko bezpečnější způsob oddělení oproti kontejnerizaci
- Dnes se kombinuje virtualizace a kontejnerizace
 - Virtualizace = tvrdé bezpečné oddělení pro zákazníky
 - Tedy v 99% nesdílí více zákazníků jeden VM!
 - Kontejnerizace = sdílení výkonu a různé oddělené konfigurace pro různé aplikace jednoho zákazníka
 - Menší nároky na HW
 - Sdílíme kernely OS
 - Šetříme prostor (duplicitní OS a jeho soubory)
- Kontejnerizace (techničtěji)
 - Oddělení procesů do skupin
 - Různé jmenné prostory (jména souborů, čísla portů, pojmenované roury)
 - Na hranicích mezi kernelem a procesem jsou identifikátory a jména přemapovávána
 - Soupeření o zdroje
 - Místo prioritních front je obvykle použit nějaký sofistikovanější mechanismus
 - Jedná se především o filesystem

- Balík souborů, který se nějak prezentuje procesům, který běží v kontejneru
- Obvykle se se startem kontejneru spustí nějaký příkaz, který spustí jeden proces
- Ten jeden proces si pak může vytvořit děti (UNIX spouštění procesů)
- 1 kontejner má obvykle více procesů
 - Typicky sdílejí pohled na svět, stejná mapa jmen
- Když máme jeden virtuální stroj tak na něm můžu mít 1 nebo i více kontejnerů
 - Je možné nakonfigurovat dva kontejnery uvnitř VM, aby sdíleli jeden pohled na svět
 - Často se to hodí, pokud dostaneme hotové kontejnery, od nějakých producentů SW (nechceme je upravovat), a potřebujeme je nasadit a zároveň chceme aby komunikoval již s naším existujícím
- Někdy můžeme potřebovat rozdělit jeden kontejner jenom kvůli dependency hell, ačkoliv zbylý pohled na svět sdílí
- Výběru OS při vyrábění kontejnerů vlastně znamená jenom jaký rootfs budeme chtít (vyšší vrsta OS)
 - Chceme yum nebo apt
- glibc je vrstvička mezi syscalls a veřejným UNIXovým rozhraním

Přednáška 2

- X86 měla nevhodnou instrukční sadu pro virtualizaci (původně)
 - Paravirtualizace
 - VT-x, AMD-V
- Nešlo zabránit kódu aby nepřišel na to, že kernel VM běží v neprivilegovaném režimu
- Paravirtualizované kernely jsou zejména modifikovány
 - Ve správě paměti (virtuální paměť, ...)
 - Ovladače disků a NIC
 - Šlo by emulovat, bylo by pomalé
- Obvykle všechny tyto věci neřeší pouze hypervisor, který dostává požadavky od VM
- Obvykle s guest OS ještě běží tzv. VM OS kernel, který tyto záležitosti řeší (nějaký modifikovaný Linux obvykle)
- VM OS kernel je často běží skoro podobně jako jiné virtuálky, hlavní rozdíl je v tom, že hypervisor mu dává exkluzivní přímý přístup k HW
 - Nemusíme v hypervisoru implementovat milion driverů a tak, využijeme ty z Linuxu
 - Chceme aby hypervisor byl malý a bezpečný
- Součástí virtualizačního nástroje ještě bývá VM Controller
 - Vysokoúrovňová ovládací vrstva
 - Řeší migrace stroje na jiný HW
 - Část, která komunikuje s nějakým administrátorem
- Paravirtualizace byla kvůli otevřenosti kernelu nejprve záležitostí Linuxu
- Microsoft paravirtualizaci ale implementoval do NT kernel také, jenom 5-10 let později
 - Hyper-V byla původně paravirtualizace

- V paravirtualizaci původně musely být kernely stejného druhu (nemohl koexistovat NT kernel s Linuxem)
 - Části kernelů komunikující s hypervisorem, které byly modifikovány, byly původně využívali rozhraní host OS
 - Xen = jenom Linuxy
 - Hyper-V = jenom Windowsy
 - Dnes rozšíření interfaců hypervisorů → Hyper-V se vyrovná s Linuxem
 - Už víceméně neplatí
- Dnes je práce hypervisoru zjednodušená HW podporou, pokud kernel modifikujeme, obvykle se nejedná o obrovské modifikace (malé modifikace zvyšující výkon)
- Před rokem 2005 nebyla opravdovská virtualizace (prakticky možná)
 - Borci z VMware binární překlad (1999)
 - Krkolomné
 - Pouze x86
 - Pomalé
- Při opravdové virtualizaci guest OS neví o tom, že je virtualizován
 - (hacky jak zjistit, jestli jsem VM) Testování výkonnosti, některé instrukce trvají více času...
- Dnešní situace je, že používáme mix opravdové virtualizace s paravirtualizací
 - Guest OS mají ovladače pro virtuální zařízení (dnes to je už součást OS)
 - Když OS hledá připojený HW (standarty sběrnice PCI třeba), tak hypervisor odemuluje domnělý HW → guest OS si najde příslušný ovladač
 - Potom co se ovladač v OS načtený (díky emulaci nějakého přihlašovacího standardu) tak zbytek práce domnělého zařízení se už neemuluje, ale provádí nějak výhodněji/efektivněji (např. virtuální disk forwarduje příkazy pro čtení hned hypervisoru)
 - Navýšení výkonu
 - Samotná opravdová virtualizace je pomalá, dělá zbytečné věci
- Kombinace kontejnerizace a virtualizace
 - 1 Kernel na jeden VM (zákazník oddělen opravdovou virtualizací nebo paravirtualizací)
 - Zákazníci si sami oddělují svoje věci kontejnerizací, případně řeší nějaké dependency problémy mezi jejich službami
- V paranoidních situacích se stále přiděluje 1 fyzický stroj jednomu zákazníkovi, případně zákazník má vlastní privátní Cloud
- Některé kontejnerizace využívají HW podporu pro virtualizaci
 - Pokusy Windows o kontejnerizaci (asi to umře)
 - Některé docker konfigurace na Linuxu
- Provozování hypervisoru nad hypervisorem?
 - Je možné
 - Kdysi, když to dělala IBM, tak už tenkrát demonstrovali 5 vrstev nad sebou
 - Dnes vícevrstevná až 2008 nebo 2010
 - Relativně běžné
- Historie

- IBM S/370 (1983-1992)
 - Měli jsme na MFF místo UFALu
 - V 90. letech darovali nějaká kanadská univerzita
 - Bootoval se z 8" diskety
 - Z hlediska instrukcí CPU bylo načtení z disku jednodušší než dnes
 - Mělo to HW debugger
 - Tenkrát bylo hodně OS
 - Mzdy se počítali na OS, kde byl Cobol
 - Pokud jsme chtěli třeba Fortran, tak na to jsme měli jinej OS
 - Měla virtuální paměť
 - Předchozí OS nerozuměli virtuální paměti
 - Bylo potřeba rozjet staré OS → virtualizace
 - Dalším důvodem bylo, že tenkrát existovalo mnoho OS, které bylo vždy nutné přebootovat
 - 370 však je dostatečně výkonná aby stačila na koexistenci více OS
 - Virtualizace, kde koexistují dva OS (jeden počítá mzdy - cobol, v druhém běží fortran)
 - Tedy důvod IBM byl víceméně, že byly donuceni, aby dali zákazníkům důvod pro přechod na nový drahý IBM 370
 - Byli vlastně dvě možnosti jak bylo možné tento stroj provozovat
 - IBM přišla s OS 370, který rozuměl virtuální paměti a podobal se dnešním OS
 - Nebo jsme používali hypervisor s virtualizací starších OS
 - Postupný zánik
 - Mainframe vytlačeny minipočítači (to jsou počítače, které nebyly velké jako místnost ale pouze skříň)
 - Stavěno jako levné, tyto aspekty se neřešili
 - Další počítače IBM PC
 - Opět krok zpátky
 - Tlak na cenu
 - Mělo to parodii na OS
 - Nástup Unixu
 - Unix je založen na komunikaci mezi procesy (roua)
 - Virtuální stroje komplikují komunikaci mezi procesy
- Popek-Goldberg požadavky na virtualizaci (všichni virtualizátoři se o toto snaží)
 - Pozn. VMM = hypervisor
 - Program běžící pod VMM se musí chovat v zásadě stejně, jako by běžel na ekvivalentním fyzickém stroji (equivalence)
 - V zásadě to znamená, že to je stejné až na rychlost
 - VMM musí mít úplnou kontrolu nad virtualizovanými zdroji (safety)
 - Pokud nejsem schopni zajistit bezpečnost prováděné instrukce, tak trap and emulate – VMM se vzbudí a odemuluje

- Statisticky převládající část strojových instrukcí musí být prováděna bez zásahu VMM (efficiency)
 - Nemůžeme moc emulovat, co není potřeba tak neemulovat
- Proč byl na x86 problém virtualizovat
 - Rozhraní mezi bezpečnými a nebezpečnými instrukcemi je jiné než rozhraní mezi privilegovanými a neprivilegovanými instrukcemi
 - Nemůžeme zajistit, aby se hypervisor vzbudil u nebezpečné instrukce
- Paravirtualizace (před HW podporou virtualizace)
 - Vyházíme nebezpečné instrukce, zajistíme, že modifikovaná část kernelu budí VMM
- Binární překlad
 - Troufla si na to pouze firma VMware v 1999
 - Kernel OS je strojově přeloží z binární podoby (jakoby JIT překladač) po částech
 - Každý kousek, který se má vykonávat se přeloží
 - Děláme něco jako paravirtualizaci mechanických způsobem
 - Odstraňujeme z kódu instrukce, které by mohly dělat problém
 - Tenkrát bylo potřeba zmodifikovat zhruba každou 3 instrukci
 - Musíme být schopni celý kernel takto přeložit
 - Musí se vyřešit spousta technických problémů
 - Jak zajistit nepřímé skoky, který skáčí na adresu, kde je nepřeložený kód?
- Jaké instrukce jsou teda nebezpečné?
 - Instrukce testující stav procesoru
 - Neprivilegovaná
 - Intel si myslel, že přečíst stav si může každý
 - Modifikace je privilegovaná
 - Problém: takto se OS dozví, že je virtualizován
 - Některé instrukce se dokonce chovají jinak pokud běží neprivilegovaně nebo privilegovaně
 - Stránkovací tabulky jsou realizované HW page walkem, když OS zapisuje, když se rozhodne přemapovat procesům jejich virtuální paměť, tak se to dělá neprivilegovanou instrukcí. Hypervisor toto ale musí odchytit, aby dokázal skládat mapování
 - Toto se neřeší binárním překladem, ale úpravou stránkovací tabulky aby došlo k výpadku stránky (přestože výpadek nastat neměl)

Přednáška 3

- 1999 - VMware Workstation
- VMM jako aplikace Windows NT
- V roce 2002 přišli s VMware ESX Server
 - Zbavili se VMM jako aplikace pro Windows NT
 - VMM nějaký Linux, který je ořezaný a přizpůsobený pro VMware
- 2003 - Xen
 - V prostředí Linuxu

- Paravirtualizace (modifikace kernelu Linuxu)
- Toto dohromady přesvědčilo výrobce HW, že musí napravit co zkazil (Intel)
- V roce 2005 náprava Intelu v podobě VT-x
- V roce 2006 přišlo AMD s AMD-V
- Prvotní rozšíření instrukční sady, které řešili problémy s virtualizací
- Dnes drobné změny na úrovni HW, spíše se zlepšuje výkon na úrovni SW (který je ale už velmi dobrý), přidává se podpora pro hypervisory do kernelů a tak
- Dnes jsou VM nevhodné pro
 - měření výkonu
 - Real-time aplikace (obchodování na burze)
- Implementace VMM
 - Jak se virtualizují jednotlivé komponenty (CPU, ...)
 - Začneme s opakováním
 - 2 použitelné režimy CPU (neprivilegovaný, privilegovaný)
 - Neprivilegovaný
 - Můžeme někdy poznat nastavení nějakých privilegovaných registrů, protože se některé operace chovají jinak
 - RW operace neprivilegovaných registrů
 - Stav CPU (čtení)
 - Privilegovaný
 - I/O operace
 - Manipulace se stránkováním (TLB)
 - Obsluha přerušení
 - Atomicky skočíme na obslužnou adresu, změní režim do privilegovaného módu
 - Nejprve se budeme bavit o tom, že nemáme HW podporu virtualizace
 - Virtualizace CPU
 - Ani Kernel OS nesmí poznat, že na HW není sám
 - Pro aplikační procesy není problém, v podstatě běží úplně stejně. VM dostane jenom některé HW přerušení, ta která mu patří, aplikační proces bývá přerušen častěji.
 - Fyzické CPU tráví čas obsluhou přerušení jiných VM
 - Obsah neprivilegovaných registrů vyměňuje hypervisor při přepínání
 - Privilegované instrukce
 - Nemůžeme jen tak nechat VM provádět je
 - emulace?
 - Skutečný stav privilegovaných registrů nemůžeme přiznat
 - Virtuální privilegované registry
 - Zákaz manipulace s TLB
 - Easy, instrukce jsou privilegované
 - Paměť
 - Hypervisor nějak musí rozdělit paměť mezi VM
 - Každý OS virtuálního počítače má mapování (domněle fyzické)
 - Hypervisor chce taky někde paměť mapovat

- Extrémně těžké je odpozorovat co kernel dělá se svým virtuálním paměťovým prostorem a spojit do s mapováním hypervizoru, případně mu v tom bránit
- I/O
 - Nepustíme VM k I/O operacím
 - Pokud teda není nastaveno v tzv. exkluzivním přístupu
 - Obvykle tyto operace půjdou přes hypervisor to VM OS kernel (nastražená virtuálka VMware, upravený Linux, který vykonává tyto požadavky)
- Mechanismus volání jádra OS
 - Nelze použít původní způsob, kdy jednomu VM dáme privilegovaný režim, kdy má kontrolu nad celým fyzickým PC
- Obvykle se to implementace řeší 2 způsoby (pokud nemáme podporu HW)
 - Trap and Emulate
 - I kernel hodíme do neprivilegovaného režimu
 - Všechny privilegované instrukce budou generovat výjimky
 - Odemulujeme
 - Čím více jich existuje, tím pomalejší to bude
 - Binary translation
 - Kernel je fakticky pouze zdroj pro binární překlad
 - Části, které by se to pokoušelo spouštět jsou zpozorovány VMwarem a přeloženy do jiné binární formy (nebezpečné instrukce, přemapovávání instrukcí)
 - Modifikace kódu kernelu za běhu
 - Rychlejší než trap and emulate
 - Některé privilegované instrukce není třebaodemulovávat, ale jde nějak přeložit
 - Implementačně neskutečně náročné
 - Nepoužitelné, zastaralé, za zenitem
- Hardwarové podpora
 - AMD přišlo 64-bit verzí CPU
 - A celý binární překlad je k ničemu (VMware použil segmentaci, která byla v 64-bit procesorech odstraněna)
 - Další úroveň privilegovanosti (root režim CPU)
 - Kromě neprivilegovaného (aplikačního) a privilegovaného tak máme ještě root režim, ve kterém běží hypervisor
 - Neprivilegovaný režim ~ aplikace VM
 - Privilegovaný režim ~ kernel virtuální mašiny
 - Root režim ~ hypervisor
 - Stínové registry, prostřednímu režimu se prezentují jiné obsahy registru než root režimu
- Paravirtualizace
 - Upravený kernel virtuálky nepoužívá privilegované instrukce
 - Dneska ale i paravirtualizace využívá HW podporu

- Využívají tedy privilegovaný režim, dnešní smysl je spíš v navýšení výkonu
- **Kompresi privilegií**
 - Pokud nemám HW podporu v CPU
 - Hypervisor běží v N+1 úrovni privilegií, kernel běží v N-té úrovni
 - My ale máme k dispozici pouze N úrovní
 - Musíme to celé (hypervisor) namačkat do N úrovní
- Dnes máme 8 úrovní privilegií, který běžící kód může mít
- Přepnutí adresového prostoru je velmi drahé
 - ~~CPU nepřepíná adresový prostor při volání z jádra~~
 - Meltdown, Spectre (dnes už přepínáno)
 - Ochrana řešena privilegovanými stránkami
- Kam dát ale hypervisor
 - VMware to řešil segmentací paměti
 - Opět nám pomůže úprava HW
- To stránkování a virtuální paměť radši slidy, strana 55
 - Něco o dvojitým mapování
- VT-x, VT-d
 - Významné virtualizační softwary umějí použít skoro všechny features
 - Intel spolupracuje s producenty virtualizačních nástrojů
- AMD-V
 - Úpravy o rok později
 - Většina není kompatibilních s Intelem
- Situace je velmi znepřehledněna obchodní politikou
 - Obchodní názvy maskují podstatu věci
 - Některá rozšíření (často velmi propagovaná) jsou triviality
 - Jiná jsou zas velmi netriviální
- Ještě do nedávna byly dražší verze CPU pouze s virtualizací
- Dnes už ale to je opravdu v každém procesoru
- Klíčové vlastnosti VT-x
 - Root/non-root execution (už jsme říkali)
 - Non-root režim má omezen přístup k privilegovanému stavu
 - Nežádoucí akce způsobí "VM exit"
 - Extended Page Table (EPT)
 - Řešení problému virtualizace paměti
 - U AMD se to jmenuje NPT (Nested Page Tables, není to ekvivalentní, ale výsledek je ten samý)
 - VMCS Shadowing
 - Podpora rekurzivní virtualizace
 - U Intelu až 2013! (konečně ve stavu kdy to IBM uměla v 1973)

Přednáška 4

- Budeme si povídat o dalších součástích VT-x
- VT-x
 - Změny v CPU

- VT-d
 - Změny v HW
- VGuest Preemption Timer
 - Pro virtualizaci aplikací s mírnými real-time nároky
- FlexMigration
 - Mechanismus, který umožňuje zamaskovat schopnosti CPU
 - Pokud máme hromady počítačů mezi kterými potřebujeme migrovat virtuálky, tyto počítače však mají různé procesory, některé novější instrukční sady, některé starší
 - Potřebujeme aby virtuálky fungovaly na všech počítačích
- Virtual Processor ID (VPID)
 - Stránkovací tabulky se při přepínání procesů přepínají a zahodí se celé TLB
 - Totéž dělá VMM při přepínání strojů
 - Potřebujeme přepínat stroje rychleji
 - Je vidět cena za zahazování TLB
 - Řešení
 - Procesor dostal nový registr, kde je klíč běžícího procesu, záznamy v TLB ho obsahují také
 - Při rychlém střídání procesů se celá TLB nezahazuje
 - Musí tomu rozumět i OS – dnes už rozumí
 - Opět tento mechanismus mělo i IBM S370
- Real-mode support
 - Když bootově PC, tak procesor se probouzí v archaickém režimu real-mode (stále 16-bit režim).
 - Nefunguje v tomto režimu stránkování
 - Pak se to přepne do 32-bit, pak se zapne stránkování a nakonec se zapne 64-bit režim
 - V Hypervisoru jsme donedávna museli emulovat instrukce v real-modu (při spouštění OS)
 - Dnes HW podporovaná emulace instrukcí real-modu
- **IOMMU** (tohle už patří do VT-d)
 - Důležitý a složitý na implementaci
 - IO zařízení chce přistoupit k paměti
 - Kdysi dávno (IBM S370) se tento přístup odehrával bez CPU, dneska ten přístup zprostředkovává CPU
 - Nějakým povelům PCI sběrnice požádá CPU o provedení operace s pamětí
 - Dělá to nějaký HW stranou v čipu procesoru
 - IO zařízení pracují s fyzickými adresami, jenže fyzické adresy ve skutečnosti uvnitř virtuálky nejsou fyzické a budou se přemapovávat na fyzické VMM
 - Hypervisor musel zasahovat do komunikace mezi kernelem a IO zařízením
 - Akce musela být odemulována a adresa přeložena VMM
 - Pomalé a nešel udělat přímý přístup k IO zařízením
 - Intel spolu s AMD se rozhodly s tím něco udělat

- Změna zahrnovala změnu sběrnice PCI Express
 - Address Translation support (ATS)
- Hypervisor musí překladově služby v CPU nějak komplikovaně nakonfigurovat aby CPU vložil do IO zařízení správnou adresu
- Vzhledem k tomu, že to zlepšilo efektivitu dovedlo se všechny donutit změnit standard a respektovat/implementovat ho
- Single-Root I/O Virtualization (SR-IOV) (také patří do VT-d)
 - IO zařízení deklarují schopnosti virtualizace
 - Rozšíření standardu PCI Express
- Další technologie týkající se virtualizace
- Celý svět se snaží do virtualizace zapojit GPU
- Graphics Virtualization Technology
 - Všichni chtějí dělat umělou inteligenci
 - Cloudový poskytovatelé chtějí efektivně prodávat výpočetní výkon svých GPU přes virtualizaci
 - GPU jsou v tomto ohledu 40 let za CPU
 - Intel má nějaké své standardy (GVT-D, GVT-S, GVT-G)
 - GVT-D
 - Exkluzivní přístup
 - GVT-S
 - Sdílený přístup
 - GVT-G
 - Časový multiplex
 - Lol Intel není moc velký hráč na poli grafických karet, nVidia s AMD ano a ty tenhle “standard” neřeší
 - Každý má svůj názor jak to dělat
 - Dneska pokud máme super drahou kartu (od nVidie) a super drahou licenci VMware tak nějaká slušná virtualizační podpora existuje
 - Je to spíš business ty jejich dohody co budou podporovat
 - Nevíme jak to funguje
- AMD má obdobné features, které sice mají různé jméno, ale cíl je stejný
- VMM musí vědět jaký CPU máme
- Některé věci měli dříve AMD některé Intel
- Dnes oba mají všechno
- I když trochu jinak
- Virtualizace I/O
 - Radši slides někde 77 strana
 - Exkluzivní přístup
 - Sdílený přístup
 - Spousta práce navíc
 - Hypervisor musí odchytnout všechny instrukce, kterými jádro programuje I/O zařízení, tyto instrukce musí odemulovat
 - Nemodifikovaný kernel si myslí, že pracuje s reálným IO

- Hypervisor si přečte z hromady bytu v paměti (kam zapsal kernel, když si myslel, že komunikuje s IO zařízením)
 - Dekóduje jakou akci mělo I/O zařízení provádět
 - Překóduje tuto akci pro I/O zařízení případně
 - Zařadí tuto akci do fronty (I/O zařízení může být využíváno)
 - Potom až bude IO zařízení k dispozici se tedy naprogramuje, až skončí tak dojde k fyzickému přerušení to odchytl hypervisor a odemuluje přerušení kernelu
 - Paravirtualizace implementací ovladače zjednodušuje práci
- Komunikace VM a VMM
 - Příklad na Hyper-V
 - Hypervisor může sloužit pouze k zprostředkování komunikace mezi virtuálními stroji
 - S tímto přišel M\$
 - Realizoval hypervisory tak, že je vložil pod OS
 - VMware to měl tak, že hypervisor běží nad OS (vzali ořezanej Linux a přidali si tam nějakou svoji podporu)
 - Když stroj něco potřeboval tak to odchytil hypervisor, který požádal OS aby to provedl a ten to provedl
 - V M\$ na to šli jinak
 - Hypervisor odráží požadavky zpátky jedné vybrané virtuální mašině, kde běží jedny Windows (tzv. root partition)
 - Všechno hypervisor forwarduje root partitionu, která to nějak udělá
 - Child partition = virtuálka
 - Nejlepší je, když child partition je spolupracující Windows
 - Pomocí VMBus se to přepoše přímo Root partitionu (nějak se na tom podílí i hypervisor)
 - Tam se to fyzicky vykoná
 - Obdobné to je, když máme spolupracující Linux s ovladači Hyper-V
 - Taky tam bude ten VMBus
 - Hyper-V podporuje i nespolečující kernel
 - Trvá na tom, že pracuje s nějakými I/O zařízeními, hypervisor bude emulovat
 - MSR
 - Machine status register
 - Jenom některé jsou obsazeny skutečným registrem
 - Některé jsou rezervované
 - Když kernel použije instrukci, kde je korektní číslo registru, tak to provede přímo HW
 - Když není, tak to vyvolá trap → zbudí se hypervisor, odemuluje hypervisor
 - Kernel se snaží zapsat do virtuálního MSR, který patří hypervisoru
 - Kanál, kterým může hypervisor číst data zapsaná do MSR (a naopak)

- Takto kernel může zjistit jestli běží pod hypervisorem nebo ne
- Pokusí se přečíst neexistující MSR
- Sám je kernelem tak si nakonfiguruje odchycení sám době
 - Výjimka → nikdo tam není, běžím na fyzickém stroji
 - Žádná výjimka, objeví se tam data → nacpal je tam hypervisor
- Nicméně MSR jsou pomalé pro komunikaci, takže se to používá pouze pro domluvu komunikačního kanálu (rychlejšího)
 - Rychlejší komunikace se odehrává mechanismem, kterému MS říká hypercall
- Hypercall
 - Hypervisor naaranžuje na požadavek od kernelu nějakou virtuální stránku, která se kernelu jeví jako fyzická
 - Pomocí MSR se předtím dohodli která adresa
 - Guest provede instrukci volání procedury, která leží na této adrese
 - Tam bude ležet nějaký kód procedury
 - Nebo něco jiného
 - Nebo Hypervisor tam nenamapuje nic
 - Výpadek stránky, odchytí hypervisor, který dostane informaci, že kernel něco chtěl → hypervisor může emulovat
 - Data od hypervisoru k kernelu mohou běžet dvěma způsoby
 - Pokud jsou malá tak můžou být v registrech
 - Nebo si předtím pomocí MSR domluvili VPAP (Virtual Processor Assist Page)
 - Další stránka/y, které je namapovaná na fyzickou paměť a slouží jako zápisník
 - Hypercall je možné chainovat
- Víme kulové o VMware (protože proprietární SW)
 - Tušíme, že asi přestěhovali OS z pozice pod Hypervisorem do privilegované partition nad hypervisorem

Přednáška 5

- Kontejnery (zaměříme se na Linux a do větší hloubky)
- Spolehá na prostředky kernelu pro oddělení kontejnerů od sebe
 - Kernely však slouží spíše ke vzájemné komunikaci → lepení děr, aby komunikovat nemohli
 - Opačný proces než co ve virtualizaci (zde naopak postupně jsou přidávány volitelné prvky pro komunikaci)
- Výrazně levnější na implementaci
 - Ve spoustě prostředí dostatečné (virtualizace je overkill)
 - Nepotřebujeme mít x kernelů
- Motivace
 - Každý proces má vlastní environment
 - Dependency hell

- Způsob jak v Linuxu smazat rozdíl mezi distribucemi
 - Mám nějaký program, který je odladěn a dobře funguje a Ubuntu, abych se s tím na jiné distribuci nemusel srát, tak to hodím do Ubuntu kontejneru
 - POZOR: je nutný tentýž, nebo kompatibilní kernel, který rozumí stejným syscallům
 - Zlepšení izolace mezi procesy
- Výrazně vítězí Linuxové kontejnery
- Pravděpodobně prvním však nejsou Linuxové kontejnery
- První byl M\$
 - V roce 1993 přišli s NT 3.1
 - Omluvenka M\$ za to co předváděl předtím (MS-DOS, Windows 16, Windows 3.1, ...)
 - Tyto si nezaslouží nálepku operační systém
 - Úplně nový systém, nebyly pro něj aplikace
 - Jak staré DOS aplikace tady rozjet?
 - Jak rozjet POSIX aplikace nebo OS/2 aplikace?
 - Použili něco podobného...
- NT Kernel vždy měl podporu pro izolaci namepaců (viz předchozí bod)
 - Nikdo to do 2016 nepoužíval
- Windows Subsystem for Linux (2016)
 - Emulace POSIX syscallů na Windows kernel
 - (s nějakou úpravou glibc)
 - Linux aplikace šlo provozovat nad Windows
 - Nefungují cgroups - takže spíše parodie
- Windows Containers - 2016
 - Koupil část týmu z Dockeru
 - Reimplementovat Docker pro Windows (Server)
 - 2 varianty
 - Process Isolation
 - Kernel Windows zajišťuje izolaci
 - Hyper-V Isolation
 - Každý z kontejnerů dostane vlastní virtuální stroj, kde běží vlastní kernel
 - Uvnitř kontejneru pouze Windows Server environment podporován
 - Snad to jde nějak spustit na Windows 10, ale to prostředí se pak tváří jako Windows Server
- Windows Subsystem for Linux
 - Verze 1 byla parodie viz výše
 - Verze 2
 - (LOL) už to není emulace, ale obyčejná virtuálka v Hyper-V
 - Zde běží skutečný Linux (kernel)
 - Podporuje i Linux kontejnery
- Linuxové kontejnery

- Založené na schopnostech kernelu prezentovat různý pohled na svět různým procesům (Linux namespaces)
- Potřebujeme mechanismus na oddělení procesů od sebe z hlediska konzumace zdrojů (čas CPU, ...) – v Linuxu na to je mechanismus cgroups
- Potřebujeme nějaký složitější mechanismus na oprávnění
 - Nestačí, že admin spravuje všechno a ostatní nic
- procFS
 - Stav kernelu a jeho konfigurace se prezentuje jako filesystem
 - S časem a konfigurací kernelu se mění hodnoty v souborech
 - Některé soubory jsou zapisovatelné (změna konfigurace)
 - Technicky to stále je syscall
 - Touto prezentací jako filesystem se vyřešila spousta věcí
 - přístupová práva
- Linux namespaces
 - ½ toho co slouží k oddělení kontejnerů
 - Přístupová práva a přemapování identifikátorů (aby každý kontejner měl svůj)
 - Aplikované na každý syscall
 - CGROUP (to není cgroups)
 - Podílí se na konfigurace kernelu
 - Definuje které cgroups patří k danému procesu
 - NET
 - Přemapování síťových zařízení
 - Vlastní konfigurace routovacích tabulek
 - Přemapování kompletní konfigurace síťování
 - FS (tak starý, že nikdo ani neví kdy vzniknul)
 - Mechanismus, který si pro každý proces pamatuje jeho aktuální adresář a jeho kořenový adresář
 - NS (mount namespace)
 - Ve kterém místě stromu máme něco namountované
 - Přináší, že každý proces může mounty vidět jinak
 - UTS
 - Jméno stroje
 - (každý kontejner může vidět jinak)
 - IPC
 - Meziprocesová komunikace, semaforey
 - USER
 - Přemapování na identifikace uživatelů a skupin
 - Na to jsou navázána uživatelská práva
 - PID
 - Přemapování identifikátorů procesů
- cgroups
 - Původně nesouviseli s namespaces
 - Mechanismus pro přidělování zdrojů

- Jedna groupa má společný balík zdrojů
- Tvoří hierarchii
 - Jako unixové procesy
- Kořenová cgroup má všechny zdroje, které jsou děleny podél hierarchie cgroup
- Verze 1 je dnes opuštěná
- Dnes verze 2
- cgroup je množina controllerů, což je modul kernelu
 - IO
 - Jednotlivé procesy aby dostali přiměřenou část propustnosti blokových zařízení
 - Memory
 - Ovládá přidělování paměti, včetně prostoru ve swapu
 - Pids
 - Maximální počet procesů
 - Perf_event
 - Monitorování výkonnosti
 - Cpu
 - Redefinuje způsob, jakým kernel zachází s časem procesoru (nahrazuje klasický Linuxový mechanismus priorit)
- Od Linuxu 4.6 jsou cgroups prezentovány jako typ namespace
- Nový namespace je možné vyrobit
 - Voláním unshare
 - Volající proces sám sebe přepíchne do nového namespace
 - Dá se do toho ještě nějak šťourat
 - Voláním clone
- Tímto způsobem vytvoření vzniká hierarchie (každý má otce)
- Každý namespace má ještě vlastníka
 - Vždycky namespace typu USER
 - Ten, do kterého by patřil proces, který vyvolal akci vytvoření nového namespace
 - **Vlastníkem není proces, ale namespace**
 - Uživatelé mají kapability
 - Ti co mají dostatečné pak mohou kontrolovat vlastněné namespace
- Každý namespace je namapován do procFS
 - /proc/<pid>/uid_map
- Namespacy a cgroups jsou staré záležitosti, některé jejich části byly ale modifikovány nedávno
 - Některé kontajnerové mechanismy neumí použít aktuální features
 - Docket na to trpí také
 - Dnes řeší věci, které nemusí, protože to nativně podporují namespaces a cgroups

- Docket je executable, který pojmenovanou rourou posílá data do dockerd daemon, který běží s root právy → před 15 lety to byla jediná možnost jak modifikovat namespaces a cgroups v kernelu
 - Dnes je to nevhodné, dockerd má moc oprávnění, které nepotřebuje
 - slabina systému, zranitelnost
- Proto Red Hat to reimplementoval → podman
 - Nechtějí nést zodpovědnost za zranitelnost v podobě dockerd
 - Dokonce je to nadmnožina, všechny staré docket příkazy fungují s podmanem pokud uděláme alias
 - Využívá aktuální features namespaces a cgroups
 - Pokyny do kernelu jsou přímo z podman → žádné dockerd
- Stále existují nějaké problémy s kontejnery i s nejsystematičtějším podmanem
 - Obvykle pocházejí z historie Unixu
 - Zadrátované konvence
 - systemctl status (v kontejneru)
 - Failed to connect to bus: Operation not permitted
 - Problémy s rourou
 - Nefunguje se sudo
 - /etc/sudo.conf ve vlastněn uid 65534, should be 0
 - Zjistíme, že uid 65534 má user nobody (Kernel Overflow User)
 - V našem user namespace, ve kterém ten kontejner běží, není náš globální root
 - Tam je nějaký další root, který ale nemá žádná práva
 - Id se musí přemapovat z globálních na lokálních
 - Zpětné mapování ale nefunguje, a tak jsou přemapovány na user nobody
- PID=1 má v Unixu speciální role
 - První proces
 - Startuje deamony
 - Jsou mu přidělovány zombie procesy (procesy, které přežily svého otce)
- V kontejneru může být ale PID=1 bash
 - První proces, který se zapne
 - Procesy nebudou reagovat na signály od kernelu
 - Bash tomu nerozumí
- Modernější varianty kontejnerizace toto umí nějak vyřešit
 - Vrazí tam nějaký speciální proces 1
 - Není to full systemd, je to nějaká parodie, která se tváří, že dělá něco jako systemd a plní některé jeho služby
- Ideový problém kontejnerů, filozofie
 - Docker má názor, že kontejner má být lehoučká entita, má mít jeden proces. Pokud potřebujeme složitější aplikace, kde propojíme několik procesů dohromady, tak máme mít více kontejnerů, které zorchestrujeme např. pomocí kubernetes.

- Aktualizace se mají dělat tak, že kontejner vyhodíme a nainstalujeme nový s novějším SW
 - Není to taková blbost, kontejnery se stejně většinou používají v prostředích, kde je load-balancing, takže nedojde k výpadku
 - Správný kontejner přežije odstřelení v nečekaném okamžiku, i restart na jinou verzi SW
- Ne všechny kontejnery jsou určeny do takového prostředí
- Některé aplikace jsou tvořeny hromadou procesů (apache, mysql, java, cron)
- Často k nim máme nějaký instalační skript, nainstaluje to všechny ty věci co potřebujeme. Nikdo se nechystá tyto skripty předělávat pro kubernetes.
 - Potřebujeme aby fungovaly věci jako yum (vyžaduje funkční PID 1)
 - Potřebujeme tedy systemd (nějak funkční)
- Redhat má názor, že kontejnery jsou jako počítače. Mají svůj vlastní vnitřní ekosystém. Instalační skripty fungují uvnitř kontejnerů.
- Přesuneme se k vrchním vrstvám kontejnerů
- Programy docker a podman
- Terminologie
 - Image
 - Read-only filesystem (filesystem nějak naplněn soubory)
 - Někaký defaultní nastavení
 - Je vytvářen z vrstev → přilepujeme další
 - Pomocí příkazů
 - Které okopírují soubory odněkud
 - Nebo částečně vytvořený image necháme běžet nějaké příkazy
 - Vytvoří se jakoby write vrstvička, která je jakoby vypálí a stane imagem
 - Image je posloupnost vrstev, která se nakonec slepí dohromady
 - Tzv. union filesystem
 - Upper filesystem
 - Lower filesystem
 - Problém whiteout: když v horním nebo smažeme, tak se to musí projevit i v tom spodním
 - Není to jednoduché vyrobit, ale je to součástí Linuxu už od dob před kontejnery (overlayfs)
 - Mohou se až po nějaký layer dvě image shodovat, kontejner manager to může poznat → šetří prostor, dva kontejnery sdílí nějakou část spodní vrstvy (proto se tam nechávají informace o layerech)
 - Container
 - Běžící je nějaký image doplněný o vrstvu, která umožňuje zápis

- Když ho zničíme, tak tahle vrstva zmizí
- Ale když ho zastavíme a pustíme znova tak o data nepřijdete
- Když ho chceme aktualizovat, tak vlastně použijeme nový image pro kontejner (s tím zahodíme všechno co jsme zapsali do kontejneru tou zapisovatelnou vrstvou)
- Pokud bychom potřebovali zachovat nějakou konfiguraci, tak to můžeme vyřešit tak, že kontejner bude mountovat nějaké místo vně kontejner
- Běžící kontejner vždycky začíná tím, že se spustí jeden proces, ten je definovaný už v image - pomocí entrypoint
- Nebo pomocí docker run můžeme předefinovat natvrdo, co má startovat
- Vždy je ale jeden proces, který se stane PID 1
- Pokud je PID 1 aktivní, tak může vytvářet další procesy
- Příkazy, kterými se kontejner startuje umožňují připojit stdin stdout

Přednáška 6

- Datové centra
- Vlastní si asi nebudeme stavět (pokud nejsem Amazon)
- Většinou pronájem
 - I prostor lze (kde máme k dispozici elektriku a chlazení, zbytek musíme dodat sami)
- Důvody pro DC
 - Konsolidace a standardizace (historie spíš)
 - Řadu věcí stáhneme na jedno místo
 - Snažíme se mít konzistentní HW (ne 60 výrobců)
 - Konzistentní SW
 - Virtualizace
 - Můžeme uživatelům dát počítače, které budou pouze konzole
 - Uživatelé mohou mít levné PC → zlevnění
 - Virtuální PC v našem DC se lépe spravuje (snapshots, aktualizace)
 - Automatizace
 - Jednak virtualizační nástroje mají dost vestavěných automatizačních nástrojů
 - Snazší správa a podpora pro skriptovací jazyky (VMware a Powershell)
 - Bezpečnost
 - Fyzická
 - Nikoho tam nepustíme
 - Data máme na pořádných diskových polích s RAID
 - Budeme podrobně mluvit (více elektrik a Internetů)
 - **Dostupnost**
 - Hlavní důvod pro DC
 - Existuje na to standardizace – maximální počet minut výpadků za rok
 - Tier 1 až 4 (Tier 4 je nejvíce)

- Tier 1
 - Jeden přívod elektřiny
 - Tier 2
 - Některé věci jsou zdvojené, elektřina stále jedna
 - Tier 3
 - Máme více **dodavatelů** elektřiny
 - Tier 4
 - Navíc ještě všechno všechno chlazení dvojitě napájeno
 - (Z jednoho jsou schopny běžet)
- “Zdaleka nejcennější pro firmy jsou data – ty musíte chránit za cenu vlastního života”
 - Řada firem do podceňuje do okamžiku než přijdou o data
- Určit co jsou cenná a co nejsou cenná data
- Jak budou odděleny místnosti?
- Jak dostaneme větší zařízení dovnitř?
 - Zákaz schodiště
 - Výtahy taky spíše ne
 - Velké dveře
- Kudy distribuce elektřiny?
 - Elektrické sloupy, podzemí
- Jak bude provedeno chlazení?
- HVAC (heating, ventilation, air conditioning)
 - Základní problémy, které se řeší při designu DC
 - Heating - odvod tepla
 - Ventilace - udržení správné vlhkosti
 - Vlhko – kondenzuje se voda (jasné, že to nemají rádi)
 - Sucho – také není dobré (zkoušelo se vyrábět datové centrum na poušti – odchází mechanické části a ložiska výrazně rychleji)
 - 45-55% zhruba ideál
 - Čistota vzduchu
- Natlakování
 - Požární ochrana (budeme se bavit později)
- Dnes se začíná uvažovat i o vodním chlazení v DC, pokud máme GPU servery a potřebujeme lepší hustotu na m³
 - Voda = nějaká tekutina s dobrou tepelnou vodivostí
- Elektřina (silnoproud)
 - Od někud přijde přívod
 - Stovky kW nebo mega wattů
 - Často DC mají vlastní měnící elektřiny a transformátory
 - K nim přijdou po stožárech vysoké napětí
 - Oni si to mění na 380V (3 fáze)
 - Platí se za velikost kabelu (ne využitou elektřinu jako v domácnosti)
 - Příklad na MFF
 - V troji máme 1.6MW, budovy s jednou serverovnou sebrali 1.2MW, zbývá 400kW na novou serverovnu

- Když uvážíme, že GPU server bere 2kW → 200 tlustých GPU serverů
- UPS
 - Obrovitá baterka
 - Proud za normálních okolností jde kolem ní
 - Když vypadne proud, tak UPS musí **extrémně rychle** zareagovat a začít dodávat proud ona
 - Nesmí se vyrobít žádné peaky
 - Existují mechanické věci, které jsou dostatečně rychlé jako např. relé, ale ty mají tendenci při těchto vysokých proudech vytvořit nějaký peak
 - Složitý problém
 - Když elektrika začne fungovat, tak se UPS musí šikovně odpojit
 - Musí se sladit fáze a ve správný okamžik vypnout baterka
 - **Je s tím harakiri**
- Generátor
 - UPS pouze kvůli tomu než se stihne nastartovat generátor → jeho startování trvá pár minut
 - Dnes baterky jsou strašně drahé a obvykle v DC jsou tak na 5-10 minut výpadku
 - Spočítám si kolik kWh baterku musím mít kapacitu na 10 minut běhu
 - Bývá někde kousek mimo
 - Hrozně to čoudí → potřebujeme komín
 - Randál
 - Opět se musí sesynchronizovat fáze
 - Jinak to shoří
 - Musí se dodávat nafta
 - Nesmí se nechat dlouho odpočívat → nenastartuje
 - Zrosolovatí nafta
 - Zamrzne nafta
 - Musí se čas od času cvičně pouštět
- UPS a generátor výrazně zvýší dostupnost DC (pokud fungují)
- Rozvod napájení
 - Dnes standardizovaný konektor pro zdroje (šestiúhelník co známe ze zdrojů k PC)
 - Nicméně konektorů se stále používá více, někteří výrobci mají vlastní
 - Pozor na to, abychom se nedostali do závislosti na výrobcích
 - Obvykle podlahou nebo stropem
 - Spíše podlahou
 - Zvedací podlaha
- Každý rack má vlastní pojistky abychom nevyhodili celé centrum
- Slaboproudové kabely
 - Ethernet, Infiniband, Fibrechannel
 - K počítačům v DC se nechodí

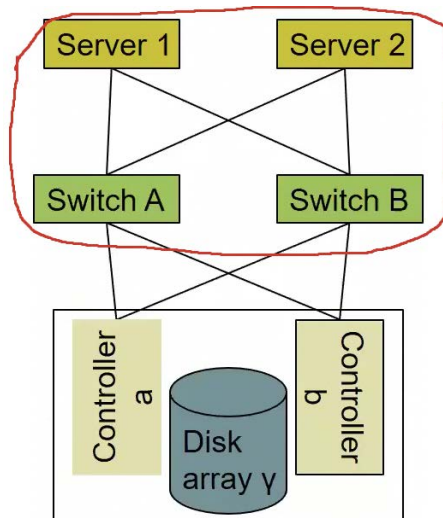
- Všechno musí mít vzdálenou správu
 - SSH, RDP
 - Navíc i management pro spuštění, restart, přístup do BIOSu, ...
- Nechceme aby to bylo u silnoproudových kabelů
- Obvykle to vede stropem
 - Jsou tam takové vany (kanály)
- Je nutné počítat nejen s nákupní cenou ale i **udržovací cenou**
 - Chladicí kapaliny do klimatizací
 - Měnit baterky
 - Nafta (každý měsíc musíme trochu propálit, když ji nepropálíme tak zrosolovají)
- Kde vůbec postavit serverovnu?
 - (Pokud si nestavíme mini serverovnu jako na malé straně, kde je místo dané)
 - Potřebuju chladnější místo
 - Seznam stavěl DC na Šumavě (celoroční průměr o pár stupňů méně → ušetřím)
 - Zas nemůžu na polárním kruhu
 - Nemám dráty a elektriku
 - Nemáme Internety
 - Potřebujeme dostupnost bezpečnostních složek → hasiči
- Teplota
 - 16-24°C
 - Při méně než 16 snadnější kondenzace
 - Při více než 24 se to snadněji uvaří
 - Problém je snížit teplotu
 - Zvýšit je triviální
- Požární ochrana
 - Správné tlakování serverovny → aby tam nebyl komín
 - Detektory kouře
 - **Nehasit vodou**
 - Odpálíme nebo vyvoláme ještě větší požár
 - Hasí se plynovým dusícím zařízením
 - Po obvodu serverovny je dusící plyn v sudech
 - Při detekci mají pracovníci 30 sekund na opuštění serverovny
 - Okamžitě všechno zahodit a sprintovat
 - Pak se tam automaticky zamkne
- Fyzická bezpečnost
 - Přímo k serverům nikdo nesmí líst
 - (Disky se šifrují, aby někdo nesebral disk)
 - Na vstupu DC je vrátnice
 - Seznam povolených osob, který můžou vstoupit
 - Každý má visačku
 - Racky jsou zavřené do klecových skříní (jako na gymplu)
 - Pověřené osoby mají klíč

- Aby jiný administrátor nemohl ukrást z jiného serveru (pokud ho pronajímáme)
- Efektivita využití elektrické energie
 - $PUE = \frac{Total\ power}{IT\ equipment\ power}$
 - State-of-the-art DC mají zhruba 1.2
 - Všecko něco žere – světla, klimačka, ...
- Napájení je největší cena, kterou platíme průběžně
- Pořádné připojení optickými kabelama na Internet
 - Netriviální investice na začátek → musí se dokopat a přivést kabely
 - Není to ani levné potom (roční poplatky)
- V DC je tendence, že na některých místech je větší teplo a na některých menší
 - Nikdy se nám nepovede to udělat uniformě, i když to dobře navrhne
 - Chceme všude stejně, aby se někde nekondenzovala voda
 - Obvykle se měří s IR kamerou a hledají tyto spoty
 - Nějakým způsobem přemístíme zařízení (mechanický pohyb racků)
 - O půl metru je posuneme
 - Musíme být schopni hýbat s napájením a racky → dvojité podlahy
 - Problém, když jsou racky zavřené
 - Maximálně můžeme posunout celou klec
- Síťová infrastruktura
 - Často racky mají TOR switch (top of rack switch)
 - Servery v racku jsou do něho připojeny (1G nebo 10G Ethernet třeba)
 - TOR switche pak jsou třeba 40G Ethernet (nebo i více) připojeny dále
- Odvod tepla
 - Teplá a studená ulička
 - Teplá = uzavřený prostor vzadu za serverama
 - Shromažďuje se v ní teplý vzduch a vydává se ven (nebo se používá podlahové topení v nějakém jiném baráku)

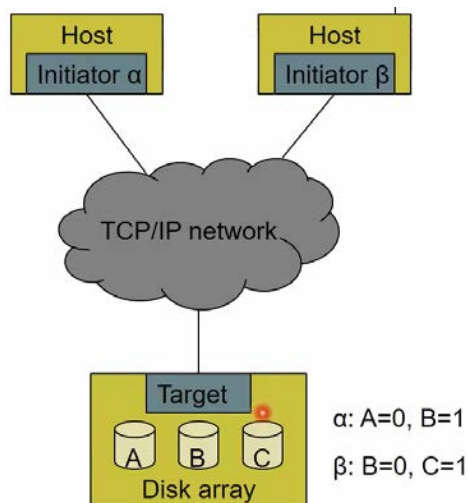
Přednáška 7

- Racky jsou normalizované
 - Hloubky několik variant
 - Šířka daná
 - Výška je na počet U pozic (U = unit)
 - Klasika je 42U, o něco málo vyšší než člověk
 - Moc větší se nedělají, pak to nevydrží podlaha
- Servery se vkládají do racku na kolejnicích, které mohou být i teleskopické (jako v troubě)
- Přenositelné DC
 - Kontejner, který je na kamionu, používá např. F1 a jiné sportovní akce
- Blade servery a blade šasí

- Jedna skříň, která se stará o infrastrukturu (napájení, síťování, připojení k diskům, větrání, management – vzdálený přístup)
- Vkládáme tam jakoby “desky” → blade servery
 - Spoléhají na vnější prvky toho blade šasi
 - Obsahují pouze procesor, paměť, síťovou kartu, ...
- Součástí bladu může být místo na blade diskové pole
 - Tyto disky pak můžeme rozdělit mezi blade servery, které je vidí jako lokální disky
- Dnes populární a hojně se používají → zvyšuje to hustotu výpočetních prostředků na m³
- Obvykle všechno vyndavatelné za chodu
- Některé věci mohou být (volitelně) redundantní
- Obvykle se do nich nevlezou GPU
 - Ani chlazení nebývá dostatečné
- Konvergovaná infrastruktura
 - Jakési vylepšené blade šasi
 - Mohou tam být disková pole
 - Blade servery bez disků
 - Blade servery s disky
 - Softwarově jde dost dorozdělit co komu přiřadím
 - Dost nás to lockne na výrobce (tzv. vendor lock)
- Tadyty bladové věci vyrábí zhruba tak 10 firem na světě
 - Každý mají své proprietární konektory a další
 - Vše je nekompatibilní (jediné co je unifikované jsou rozměry racků)
 - Do bladu Dellu nedám HP (technicky úplně jiné)
- Běžná záruka na tyto produkty bývá 5-7 let při zátěži 24/7
 - Docela důležitý parametr
- U blade serverů obvykle nemám skoro žádný rozumný diskový prostor
 - Obvykle tam jsou disky na OS
 - Ale není seriózní místo na data
 - Často mají místo pouze na 2.5" → nevyplatí se investice ani na nějaké velké disky a právě se tam dávají co nejmenší
 - Nebo rovnou máme diskless blady (třeba VMware hypervisor se vejde na SD)
 - Data musí být někde bokem → viz dále
- Network Area Storage (NAS)
 - Pomocí nějakého **síťového protokolu** (Samba, NFS) se připojujeme k nějakému diskovému prostoru (obvykle na LAN)
- Storage Area Network (SAN)
 - Potřebuju k serverům dostat diskový prostor
 - Přistupujeme nějakou technologií k diskovým polím
 - Nekomunikují pomocí síťového protokolu, ale pomocí blokového protokolu
 - Z pohledu server vznikne lokální disk
 - Není problém z těchto disků bootovat

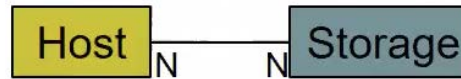


-
- Zakroužkovaná část může být součástí bladu
- Když si server dvěma portama připojí ten samý disk, tak bude mít více cest k němu po síti, pozná ale že se jedná o ten samý disk, díky UUID
 - Pokud jsou ty řadiče dostatečně dobré tak mohou dostávat pokyny oba dva naráz na jeden disk → zvýšíme propustnost
- Protocols, které se používají s SAN (na backendu, frontend je to síťové připojení)
 - iSCSI
 - Namapování SCSI na TCP/IP
 - Původní SCSI je lepší varianta ATA (do serverů)
 - Stejně jako PATA → SATA, tak SCSI → SAS (z paralelní verze se přešlo na sériovou)
 - SAS disky jsou rychlejší, vždy o jednu generaci dál 10k - 15k RPM, ...
 - Rychlost Ethernetu (1, 10 Gbps)
 - Nic moc (spíš taková dětská rychlost)
 - Pouze toto dnes to umí ty nejobyčejnější řadiče
 - Lowend

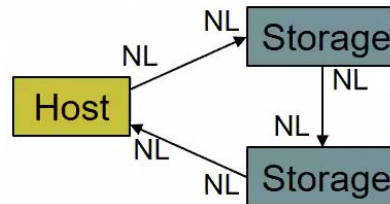


- Disky v polích můžeme číslovat (LUN) pro každého iniciátora to číslování může být jiné
 - LUN = logical unit number
- CHAP
 - Srandovní technologie umožňující “zabezpečení”
 - Heslo v plain textu
- VLAN
 - Nebo je můžeme oddělit do VLANy, aby na sebe viděli jen ty co mají
- iSER
 - Rozšíření iSCSI přes RDMA
 - RDMA = remote direct memory access
 - Procesor se neúčastní komunikace při zápisu do paměti
 - Speciálně šité pro InfiniBand
 - Highend
 - Technologie pro HPC
 - Všechno se řeší na úrovni HW
 - Extrémně výkonné a drahé
 - 56G lowend, 100G, 200G
 - Velmi nízká latency
 - 10 - 40x nižší latence než na Ethernetu
 - Dnes stále málo diskových polí, které umí Infiniband, extra drahé
- Fibrechannel
 - Nejčastěji používané
 - Protokol navržený přímo pro tohle (pro SAN)
 - Dnes jsou tyto technologie pro síťové disky výkonnější než lokální disky (SAS)
 - Rychlost 32G (nejčastěji)
 - 4, 8, 16, 32, (64)G
 - In-order
 - Drží pořadí paketů
 - Lossless
 - Nic se neztratí (HW zaručeno)
 - Neřešíme potvrzování
 - Velmi vysoká průchodnost (opravdu bychom měli naměřit skoro teoretické maximum)
 - Běhá tam o hodně méně paketů
 - Bezpečnost řešena pomocí zoning
 - Vytvoříme bezpečnostní zónu
 - Každý prvek má nějaké unikátní FCID
 - Něco jako MAC u Ethernetu
 - Na switchích se nastaví zóny na porty

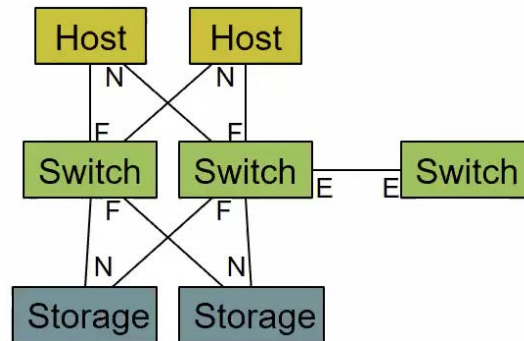
- Nechceme aby servery viděli na sebe a na diskové pole, které nemají vidět
- Zoning je povinný, každý musí být přidělen zónu
- Umí několik topologií připojení
 - Point-to-point
 - K serveru přímo připojíme diskové pole



- Loop
 - Řešení pro zoufalce



- Je to jednosměrné a pomalejší
- Switchovaná



- Písmenka na spojích jsou proto, že Fibrechannel rozeznává různé druhy portů
- Každé zařízení má jako svůj parametr napsáno jaké druhy portů umí
 - Tím je určené jaké topologie s tím můžeme udělat
 - N - node
 - NL - node loop
 - F, FL, E, G, U
 - Fyzicky jsou ty porty stejné, jde o jejich logickou funkci, kterou umí
- FCoE (Fibrechannel over Ethernet)
 - Řešení pro zoufalce
 - Použijeme FC technologii ale chceme ušetřit na drátech a switchích
 - Pokud máme moc pomalý Ethernet, tak si to budeme brzdit
 - Po Ethernetových kabelech budu posílat Fibrechannelový protokol

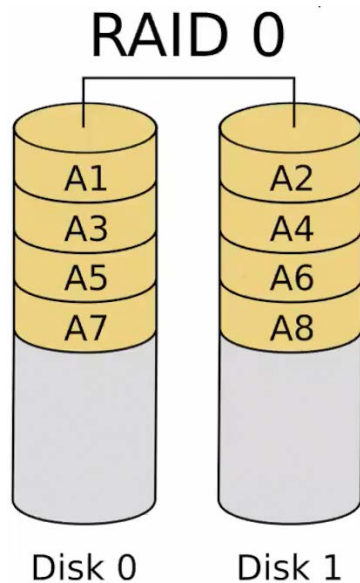
- Potíže je, že Fibrechannel je in-order a bezztrátový
- Potřebujeme na to extra switche → musíme umět lossless Ethernet
- Nějak namapovat FCID na MAC (FCID je delší)
- Jestli chceme aby to nějak rozumně fungovalo, tak musíme jít do 40G Ethernetu, což tak levné už není
- **Je s tím harakiri**

Přednáška 8

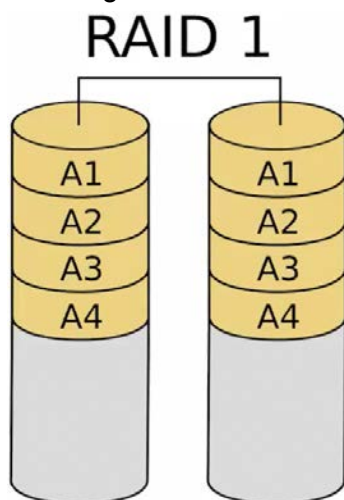
- Clusterování ve virtualizaci
- Disková pole
 - Samostatná malé počítače řídící disky
 - Mechanicky uzpůsobené
 - Enterprise diskové pole (ty opravdu velké) používají WSS (Windows Storage Server)
 - Speciálně verze Windows Server pro disková pole
 - Často mívají x86 procesory + FPGA na akceleraci
 - V levnějších polích bývá nějaký jednoduchý Linux s ARMem
 - Co obsahuje diskové pole
 - Alespoň jeden řadič (1 řadičový je fakt lowend)
 - Nabízí připojení pole (dle některých technologií, které umí)
 - Vyměnitelný za tepla (hot-swap)
 - Cache
 - Sběrnice jsou mnohem rychlejší než SSD disky
 - Kdyby spadla elektrika, když data jsou ještě v cachi, tak disková pole často mají baterku
 - Disky (LOL)
 - Chceme aby byly vyměnitelné za tepla (hot-swap)
 - **Celý to je harakiri**
 - Cílem je vysoká dostupnost → RAID
 - Co musí nabízet
 - Failover
 - Když se něco pokazí, tak ho redundantní součástka nahradí (pokud je přítomna)
 - Umře řadič, máme druhý
 - Umřel disk, máme RAID
 - Snapshots
 - Zamrazení dat k okamžiku s možností obnovení zálohy
 - Deduplikace
 - Když bych měl mít dva stejné bloky na disku, tak místo toho ho budu mít pouze 1x
 - Kontrolují se hashe bloků (s kontrolou kolíží), když jo, tak je zbytečné ho ukládat vícekrát
 - Replikace

- Synchronní replikace (jakoby replikace v reálném čase)
 - Máme dvě diskové pole, tu druhé je v reálném čase synchronizováno s tím prvním a je připraveno, kdyby se to první rozbilo okamžitě převzít veškerou práci prvního diskového pole
 - Synchronní replikace nemá význam v rámci jednoho DC, spíše máme dvě geograficky oddělená a mezi nimi se synchronní replikace provádí
- Kompresi dat
 - To dělají jenom ty fakt lepší
 - Ušetříme třeba okolo 30%
- Virtuální volume
 - Vytvoříme větší volume než máme fyzicky k dispozici
 - Spoléháme na deduplikaci
- Tiering
 - Disky máme různě rychlé
 - Můžeme přiřazovat disky do různých tierů a řadiče pak dělají migraci dat mezi disky
 - Teplejší data jdou do nejrychlejších disků, ve volných chvílích se pak data vhodně přeskupují
 - Nejméně používaná data jdou do nejpomalejších tierů
 - Nejlepší diskové pole nabízí 4 tiery, běžně dnes 2 tiery
 - Hodně to zrychlí diskové pole, přitom hodně ušetříme, protože all flash SSD diskové pole jsou extrémně drahá
 - Kapacitu naženeme rotačními disky
 - Rychlost naženeme pár SSD disky
- Spare disk
 - Disky prostě umírají
 - RAID je hezká záležitost, my ale potřebujeme disk nahradit
 - Jeden (nebo i víc) disků jenom čekají na to, až jeden z disků umře
 - Okamžitě tento disk nahradí umřelý disk
 - Pokud jde RAID zrekonstruovat
- RAID (obecně)
 - Původně redundant array of inexpensive disks (LOL)
 - Redundant array of **independent** disks
 - MTBF (mean time between failure)
 - 400 000 hodin pro běžné disky pro obvyklé uživatele
 - 1 400 000 hodin pro enterprise disky
 - Problém je, že to je **střední čas**
 - Většinou není dobrý nápad naplnit jedno diskové pole jednou stejnou šarží disků
 - Často mají podobné charakteristiky a odejdou všechny naráz
 - **MTTR**
 - Doba jak dlouho trvá než se disk nahradí spare diskem
 - V této době je pole zranitelné, kdyby odešel další disk, tak jsme v prdeli

- Potřebujeme disky stejné velikosti (pozor na 8G od jednoho výrobce je jiná hodnota než 8G od druhého)
- JBOD
 - Just Bunch of Disks
 - Slejváme disky jakoby do jednoho většího
 - Vůbec neřeší redundanci ani zrychlení
 - Nepoužívá se
- RAID 0
 - Striping
 - Potřebujeme alespoň 2 disky
 - Zvyšuje přenosovou kapacitu

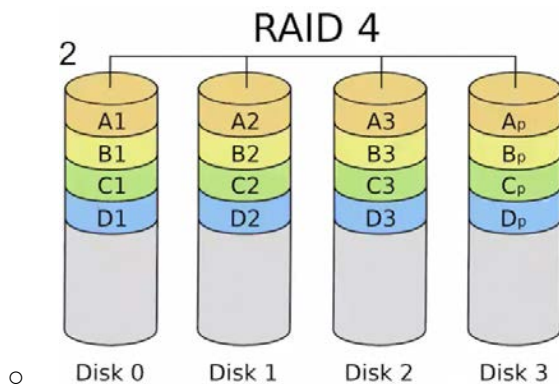


- Disk 0 Disk 1
- Nejrychlejší raid, nevadí nám, že můžeme data ztratit
- RAID 1
 - Mirroring



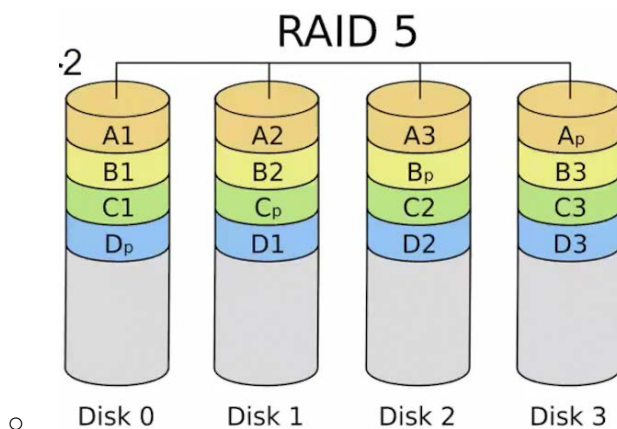
- Disk 0 Disk 1

- Nejodolnější, ale také máme furt kapacitu jednoho disku
- RAID 2, 3, 4
 - Nepoužívají se
- RAID 4



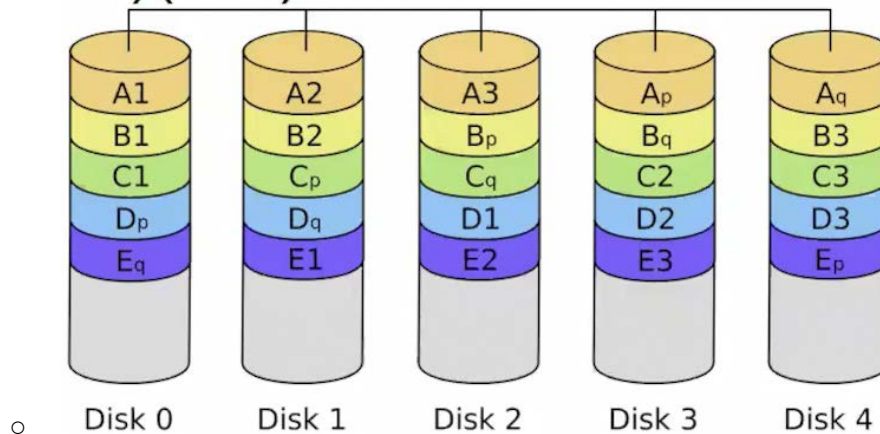
- Disk 0 Disk 1 Disk 2 Disk 3
- Paritní data jsou všechna na jednom disku → je nejvíc vytížen, vždy odejde jako první
- $A_p = A_1 \text{ xor } A_2 \text{ xor } A_3$
 - Upadne disk, kde je A1
 - $A_1 = A_2 \text{ xor } A_3 \text{ xor } A_p$
- Všechno tohle je akcelerováno FPGAčkem

- RAID 5



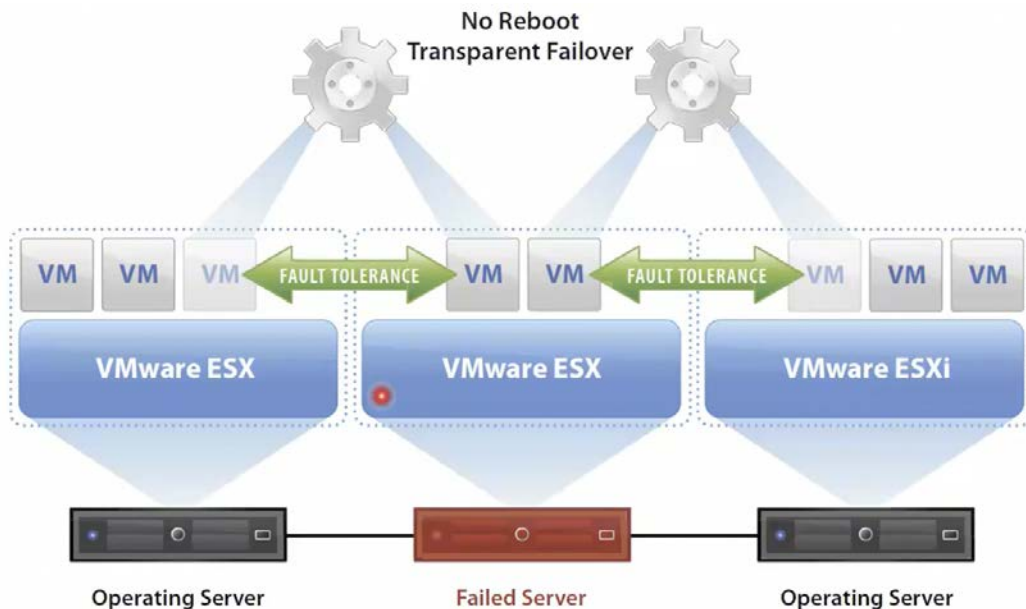
- Disk 0 Disk 1 Disk 2 Disk 3
- Jako RAID 4, ale paritní bloky jsou rozmístěny po všech discích
- Potřebujeme alespoň 3 disky
- Může umřít jenom jeden disk
- Kapacitu mám n-1 disků (jeden disk je na partitu jako u RAID 4, akorát to je rozmístěno)
- Max je doporučeno 8 disků
 - Pak už roste výrazně pravděpodobnost toho, že to celé umře
- RAID 6
 - Potřebuju alespoň 4 disky
 - 2 paritní bloky
 - Max 16 disků (víc nedoporučováno)
 - Ztratíme 2 disky z kapacity

$\frac{1}{n-2}r^n$ RAID 6



- Disk 0
- Disk 1
- Disk 2
- Disk 3
- Disk 4
- Ještě se používají kombinace RAIDů
 - RAID 1+0
 - RAID 0+1
 - RAID 5+0
 - Nejdřív se udělá ten RAID, který je pravý operand +
- Teď bude pokračovat něco o hypervisoru a virtuálkách od Yaghoba
- Hypervisor zajišťuje izolaci
 - Jedna virtuálka nemůže ovlivnit ostatní
 - Nejlepší, pokud máme pouze jednu službu na virtuálce
 - Pak se potenciální útočník ani k ničemu dalšímu nedostane
- Každá virtuálka je soubor (nebo množina souborů)
 - Migrace
 - Snapshoty
 - Vyrobit se nový overlay, do kterého se pak zapisují nové změny
 - Když se chci vrátit, tak se overlay zahodí
 - Nebo můžu snapshot natvrdo spojit s diskem
 - Klonování
- Virtuálka neví nic o fyzickém HW pod
- Distribučované systémy
 - Na kterém z hostů spustit virtuálku
 - Hodí se na nejméně vytíženého hosta
 - Dynamický load balancing
 - Migrace za běhu z jednoho hosta na druhého
 - Power management
 - Když se nic moc nepočítá na celém clusteru, tak některé fyzické servery vypnu (ušetřím elektriku)
 - Nejdřív teda odmigruju všechny virtuálky, pokud tam nějaké jsou
 - Pokud začne narůstat zátěž, tak zas musím zapnout
 - Je nutné vzít v potaz, že zapnutí serveru chvíli trvá, je potřeba dobře nastavit hysterezi
- Live VM Migration (živá migrace)

- Jak přesunout za běhu virtuálku
- Soubory se kopírovat nemusí, virtuálka je na distribuovaném FS (někde na SANu)
- Musím vyřešit přesunutí paměti a kontextu procesoru
- Obarvím na původním stroji stránky na read only a začnu kopírovat paměť na druhého hosta
- Když se někdo pokusí zapsat, tak mu ten zápis dovolím
 - Musím si poznačit, že stránka byla změněna
- Iterativně běžím znova se změněnými stránkami
- Toto párkrát udělám než se dostanu do stavu co je těchto stránek málo
- Zastavím virtuálku, dokopíruju zbytek virtuální paměti, vezmu kontext procesoru a přesunu ho a pustím to na druhé straně
- I tak virtuálka většinou vypadne na něco kolem vteřiny
- Všimnou si toho musí i switche, najednou virtuálka komunikuje z jiného portu
- Distributed High availability (distribuovaná vysoká dostupnost)
 - Co když náhodou umře jeden server?
 - V okamžiku, kdy umře jeden server, tak ostatní servery si toho všimnou rozeberou si jeho virtuálky a restartují si je u sebe
 - Amatérské řešení
 - Můžeme přijít o práci, kterou virtuálky prováděly
 - Lepší řešení je fault tolerance
 - Budeme mít virtuálky naklonované na hostech
 - Jedna je primární, druhá sekundární, jenom to sleduje a změny se propagují do té sekundární přes síť



- Když umře host se sekundární, tak se nic neděje stále máme primární
 - Někde jinde nashutuju novou sekundární
- Pokud umře primární, tak hned je připravená sekundární

- **Lag je asi jenom 5ms**

Přednáška 9

- Budeme se dívat na cloud computing z pohledu vývojáře
- Exekuční modely = Computing services
 - Různé způsoby, kterými je možné provozovat kód v cloudovém prostředí
- Nejdříve se budeme bavit o těchto exekučních modelech
- Exekuční model určuje, co všechno budeme moci dělat s naším prostředím
 - O co se budu muset starat
- Pro různé druhy aplikací jsou různé modely vhodné
- Private
 - Všechno řeším sám
 - Síťování
 - HW
 - Virtualizace
 - SW vrstvy
 - Vlastní aplikace
- V cloudovém prostředí vždy jedna z následujících
- IaaS
 - Cloud nám dává k dispozici všechno co se týká HW
 - Servery, virtualizace, síťování, ...
 - SW vrstvy si spravujeme sami
- PaaS
 - Součástí je i základní systémový a aplikační SW
 - DB, runtimey
 - Chci MSSQL a .NET Core x.x
- SaaS
 - V tomto případě nám cloud poskytuje i příslušný SW
 - Tohle je spíš z pohledu zákazníka
- Všechny cloudové platformy umožňují spouštění virtuálních strojů
 - Naši vlastních
 - Nebo z jejich galerie, kde je na výběr z distribucí Linuxů a různých Windowsů
 - Často si můžeme naklikat i s jakým SW přímo chceme ten image
 - Galerie jsou pravidelně udržované a aktualizované
- Finanční úhrada se skládá ze dvou částí
 - Konfigurace
 - To co si nakonfigurujeme jako HW
 - Vlastní používání
 - Za každou vteřinu (nějaké desetiny centů)
- Tedy platíme i za stojící mašinu, protože poskytovatel musí alokovat nám dodat HW, když ho budeme potřebovat
- Cloud můžeme použít nejen pro běh aplikací, ale i jako testovací prostředí, protože máme i dispozici tu platformu a OS
- Práce s virtuálními stroji je obvykle na úrovni IaaS

- SLA - service level agreement
 - Kolik nám garantuje poskytovatel uptime
 - 99.9x%
 - Počet devítek za desetinnou tečkou zhruba odpovídá tierům
 - Každá další devítka stojí mnohem víc peněz
- Availability set
 - V konfiguraci deployment naší aplikace, řekneme aby naše repliky byly umístovány na různé uzly v různých datových centrech
 - Poskytovatel obvykle garantuje, že nevypne všechny naše uzly v rámci jednoho availability setu
 - Upgrady dělá postupně, nikdy ne naráz přes náš celý availability set
 - Obvykle i naše updaty provádíme pouze na jednu repliku a když se neobjeví nějaký problém, tak pak updatujeme všechny repliky
 - I při sebedetailnějším testování se může stát, že někde je nějaká chyba
- Cloud Design Patterns
 - Návrhové vzory pro design cloudových aplikací
 - Jeden z nich je právě Multiple Datacenter Deployment, což odpovídá viz výše
- Load balancer
 - Rozhoduje o tom, která replika obslouží klienta naší aplikace
- Network Security Group (NSG)
 - Definuje kdo může (pomocí pravidel) přistupovat k našim replikám
- Obvykle se z Internetu moc nedá dostat dál než k nějakému frontendu naší aplikace
 - Pro management je vytvořen tzv. bastion host
 - Přísně bezpečnostně chráněn
 - Z tohoto management stroje je možné spravovat jednotlivé repliky, DB, ...
- Často jsou ale virtuální stroje otravné, pokud potřebujeme deploynout jenom nějakou "Web App"
 - Vůbec nechceme řešit nějaké operační systémy
 - Obvykle na to Cloudový poskytovatelé mají právě služby nazvané Web Sites / Web Apps, kde se nemusíme o nic jiného starat než o kód naší aplikace
- Web Sites / Web Apps
 - Platform-as-a-Service (PaaS)
 - Dokonce nemusíme ani řešit škálovatelnost
 - Stačí když ji povolíme a nastavíme si vhodný finanční limit
 - WebJobs
 - Často je možné nastavit si spouštění vlastních skriptů při nějakých událostech
- Cloudové platformy umožňují rozdělení na produkční a vývojové prostředí
 - Jedním klikem je možné prohodit vývojové a produkční prostředí
- Hlavním rozdílem kontejnerů oproti VM je absence OS
 - Méně náročnější na systémové prostředky
 - Úspora z pohledu poskytovatele služeb
 - Poskytovatele se je snaží protlačit co nejvíc
- Kontejnery mohou být výhodné i pro zákazníky

- Pokud poskytovatelé umožňují používat stejný kontejnerový systém, tak není problém s přechodem od jednoho poskytovatele k druhému
- Řeší to problém tzv. Vendor lock-inu
- Microservices (souvisí s kontejnerama)
 - Návrhový vzor
 - Rozsáhlejší cloudová aplikace by měla být rozdělena na jednotlivé samostatné relativně malé služby (microservice)
 - Každá služba má přesně zadefinovanou funkčnost a rozhraní
 - (Dá se to přirovnat k modulům v programovacích jazycích)
 - Velice efektivně implementovatelné pomocí kontejnerů
 - Původní představa – jeden microservice je naimplementován pomocí jednoho kontejneru
 - Ukázalo se jako ne úplně efektivní řešení (i tak kontejner má nějakou režii)
- Resources Consolidation
 - Další návrhový vzor
 - Pečlivě si rozmyslete jakým způsobem budeme mapovat microservisy na jednotlivé kontejnery
 - Příliš drobná fragmentace způsobuje výraznější zvýšení režie
 - Nároky na HW
 - Finanční nároky
 - Jednotlivé microservisy by měly být rozdělené do kontejnerů tak, aby měly stejné chování
 - Microservisy umístěné uvnitř jednoho kontejneru by měly zhruba stejně škálovat
 - Zhruba stejně aktualizovat
 - Měly by mít podobné HW nároky
 - Zůstane nám velice jemná dekompozice v rámci mezích
 - Neplatíme šílenou režii
- Pipes and Filters
 - Další návrhový vzor
 - Některé aplikace, zejména ty co zpracovávají data, je vhodné rozdělit na malé filtry
 - kusy aplikace, které akceptují data z jednoho nebo více zdrojů
 - Potom na nich něco počítají
 - Potom data dále posílají pomocí pipes do filterů
 - Celá cloudová aplikace je pak posloupnost filterů, které jsou implementované pomocí microservices, které jsou zase implementované pomocí kontejnerů
- Stateless service
 - Jednodušší na provozování
 - Dostane vstup a na základě tohoto vstupu spočítá výstup
 - Aplikace může mít svůj stav → ten ale není udržovaný ve vnitřních proměnných aplikace, ale někde externě v DB

- Musíme mít vlastní mechanismus na udržování stavu, což může být nepříjemné
- Stateful services
 - Vyvíjíme modul, který je automaticky replikovaný
 - Synchronizace mezi replikami je automaticky prováděna platformou
- Kontejnery bez orchestrace jsou jen hračka
- Jak je spravovat hromadně? (tisíce kontejnerů)
- Orchestrace
 - Souhrnný pojem pro aktivity ohledně deploymentu a běhu kontejnerů
 - Jak budou kontejnery využívat dostupný HW?
 - Celá řada systémů
 - Kubernetes
 - 2 metody jak orchestrátoru říct co chci:
 - Procedurální popis pomocí pravidel (starší způsob, ne moc používaný)
 - Sepisování a udržování těchto pravidel je ke zbláznění
 - Nedá se debuggovat
 - Deklarativní popis
 - Administrátor popíše stav jaký by chtěl
 - Platforma se stará o to, aby výsledný stav odpovídal požadovanému
- Orchestrátory
 - Borg
 - Vyvíjené interně Googlem
 - Pracuje s joby
 - Produkční
 - Kritické
 - Musí fungovat
 - Neprodukční
 - Můžou i chvíli neběžet
 - Vyvažování rozmístění produkční a neprodukční jobů mezi uzly, aby na všech uzlech bylo zhruba stejný poměr produkčních a neprodukčních jobů
 - Při dosažení hranice se pozastaví neprodukční joby
 - Borgmaster
 - Jádro ovládající uzly
 - 5-násobně replikovaná služba synchronizovaná PAXOSEm
 - PAXOS je alg na dosažení dist konsenzu
 - Detekce havárií, ...
 - Vytvořena okolo Borgu hromada nástrojů
 - Po zhruba 15 letech přestav vyhovovat → redesign a reimplementace Omega
 - Omega
 - Hlavním důvodem bylo dosažení vyššího stupně škálování
 - Je to proprietární a moc se o tom neví

- Kubernetes
 - Oba předchozí systémy dali vzniku tomuto systému
 - Inženýři z Googlu
 - Od počátku open-source
 - **Velmi populární**
 - 1 pod je množina kontejnerů na jednom fyzickém uzlu
- Azure Service Fabric
 - Microsofti v Azure od roku 2008
 - Open source od 2018
- EC2 Container Service
 - Privátní proprietární řešení Amazonu
 - Tady dneska podporují Kubernetes v Amazonu
- Terraform
 - Nezávisle vyvíjený
- Vývoj deployment modů
 - Jakým způsobem jsou nasazované a konfigurované aplikace na virtuální infrastruktuře
 - Původně pouze virtualizace
 - Dnes často kontejnerizace
 - Aplikace jsou stále vázané na kontejnery → řešení serverless computing
 - Serverless computing
 - Vývojáři nasazují jednotlivé samostatné funkce
 - Nestaráme se jakým způsobem jsou funkce zabalené (kontejnery nebo VM)
 - Nestaráme se o to kolik replik je spuštěno
 - Stará se o to příslušná platforma (Cloud)
 - Dle toho co jsme si za to zaplatili (nebo kolik jsme ochotni zaplatit)
 - Function-as-a-Service
 - Just write a function
 - Velice silně protlačované
 - Umožňuje poskytovatelům snížit náklady
 - Na jednotku HW vměstnaní daleko více než kontejnerů (o virtuálkách se ani nebudeme bavit)
 - Dnes to ale vychází i pro nás levněji (musí nás nějak donutit to používat)
 - Peníze se počítají pouze za využití!
 - Komerční názvy
 - AWS Lambda
 - Azure Functions
 - Google Cloud Functions
 - Původně asynchronní vyvolávání funkcí
 - Event-based
 - Synchronní volání funkce

- Z různých programovacích jazyků ji můžeme volat když ji nahrajeme do Cloudu
- Poll-based
 - Funkce může být volána z pohledu konzumenta
- Jde tím dělat všechno, co je založené na událostech
- Triggerů může být celá řada
 - Timer
 - Http request
 - ...
- Snížené nároky na administrátora
 - Je to mnohem triviálnější používat než kontejnery nebo virtuálky
- Z pohledu vývojářů neřešíme backend
 - Neřešíme škálování
- Největším problémem je výkonnost a latence při vyvolávání funkcí
 - Latence je větší než na dedikovaném VM
 - Chvilí ještě trvá wakeup runtime
 - Frameworky a knihovny nemají stále v aktivním stavu
 - Probuzení trvá netriviální dobu
- Disjunktní rozdělení na pouze funkce bez stavu a jejich provázání
 - Na tohle vývojáři nejsou zvyklí
 - Složité navrhnout
- Monitorování a ladění
 - Všechno je proprietární
 - Neexistuje příliš mnoho nástrojů
 - Nejasné jak to funguje v Cloudu
 - Nejde simulovat u sebe doma
- Bezpečnostní problémy
 - Nemáme to vůbec pod kontrolou
 - (není to o moc lepší u kontejnerů a VM)
 - Není to open-source
 - Nevhodné pro citlivé aplikace
- Mobile services
 - Podpora cloudových platforem pro mobilní aplikace je široká
 - Ve zkratce: mobilní backend za minutu
 - Někdy nazýváno jako backend-as-a-service
 - Cílem je výrazně zjednodušit vývoj backendu pro mobilní zařízení
 - Mobily mají omezenou konektivitu
 - Není k dispozici signál
 - Když je k dispozici
 - Máme málo dat
 - Kanál je tenký
 - Když aplikace dělá spoustu věcí s databázema

- Vyrobí se jakási proxy pro specifickou složitou operaci a klient pomocí tenké linky zavolá proxy kam dá základní parametry a proxy provede celou operaci místo mobilu
 - Ušetříme za provoz koncovému mobilnímu zařízení
- Backendové joby
 - Mobilní aplikace často potřebují provádět pravidelné akce (zpracování dat pořízených mobilním zařízením)
 - Cloudový mobilní backend umožňuje vytváření těchto jobů a časový scheduling, jejich spuštění
- Podpora pro synchronizaci offline dat
 - Řešení konfliktů
 - Nemusíme řešit sami
 - Synchronizace až bude konektivita dostupná
- Autentikační služby
 - Přihlášení se do našeho mobilního klienta službou třetí strany
 - Opět to máme prakticky zadarmo
- Push notifikace
 - Na mobilech nemůže běžet milion služeb v backgroundu
 - Baterie
 - Potřebujeme redukovat kód
 - Nahrazení aplikací na pozadí notifikacema
 - Backend bude něco počítat a mobil pak jenom dostane notifikaci
 - PNS = platform notification service
 - Problém je, že tyto služby pro notifikace dělají do samé, ale mají jiné API
 - APNS = Apple Push Notification Service
 - WNS = Windows Notification Service
 - Řeší Notification Hub
 - Cloudová komponenta
 - Mezi backendem a PNS
 - Rozumí všem PNS (řeší multiplatformnost)
- Hi-Perf Cloud Computing (a použití Cloudu na toto)
 - V grantech (a dalších finančních záležitostech) se obvykle dělí peníze na provozní a investiční
 - Problém jak si koupit výpočetní výkon, aniž bych po skončení grantu mě zůstal super výkonný počítač (zakázané grantem)
 - Pronájem ale není problém
- Problémy s Cloudem
 - Nevhodné pro privátní a citlivé data
 - Banky to mají zakázané často
- Druhy HPC
 - Loosely coupled cluster/grid computing
 - Aplikace se rozděluje na součásti, které nemají mezi sebou vazbu (nebo nějakou tenkou)

- Tightly coupled HPC
 - Menší počet uzlů zpracovává dat
 - Vazby jsou daleko intenzivnější
 - Výstup částečného zpracování jednoho modulu je použit jako vstup druhého
 - Je zapotřebí pečlivě rozmístit moduly na fyzické uzly
- Data-intensive computing
 - Není tlak na výkonnost uzlů, ale aby se co nejrychleji zpracovávali objemy dat
 - Nelineární data-flow
 - Těsná HW interakce mezi uzly
- Poskytovatelé cloudů dost podporují HPC
 - Azure - Batch / Big Compute (dvě služby)
 - Od roku 2017 i GPGPU
 - Nízká latence
 - Google - Cloud Dataflow
 - Nejlépe pasuje na zpracování velkého množství dat
 - Také podporují GPGPU
 - Amazon - AWS HPC
 - Nabízí i FPGAčka (asi jenom Xilinx)
- Management cloudové platformy
 - Přes webový portál
 - Základem každého Cloudu
 - Skriptování
 - Azure - Powershell
 - Hromada Cmdlets
 - Google Cloud Shell
 - Shell (opravdověj POSIX, žádná Power bazmek)
 - Amazon
 - Infrastrukturní šablony/soubory
 - XML
 - Infrastructure-as-a-Code
 - Celý popis infrastruktury je někde v kódu
 - API a bindingy pro programovací jazyky
 - Azure má integraci s Visual Studio a .NETem
 - Google Cloud SDK
 - Bindingy pro Javu, Python, Node.js, Ruby, ...
 - AWS SDK
 - Bindingy pro Javu, .NET, Ruby, ...
 - REST API
 - Např. pokud není binding pro náš programovací jazyk
- Monitorování
 - Třeba i v mobilní aplikaci
- Typické využití Cloudu je, když dopředu nevíme jaké budeme mít využití

- Škálování

Přednáška 10

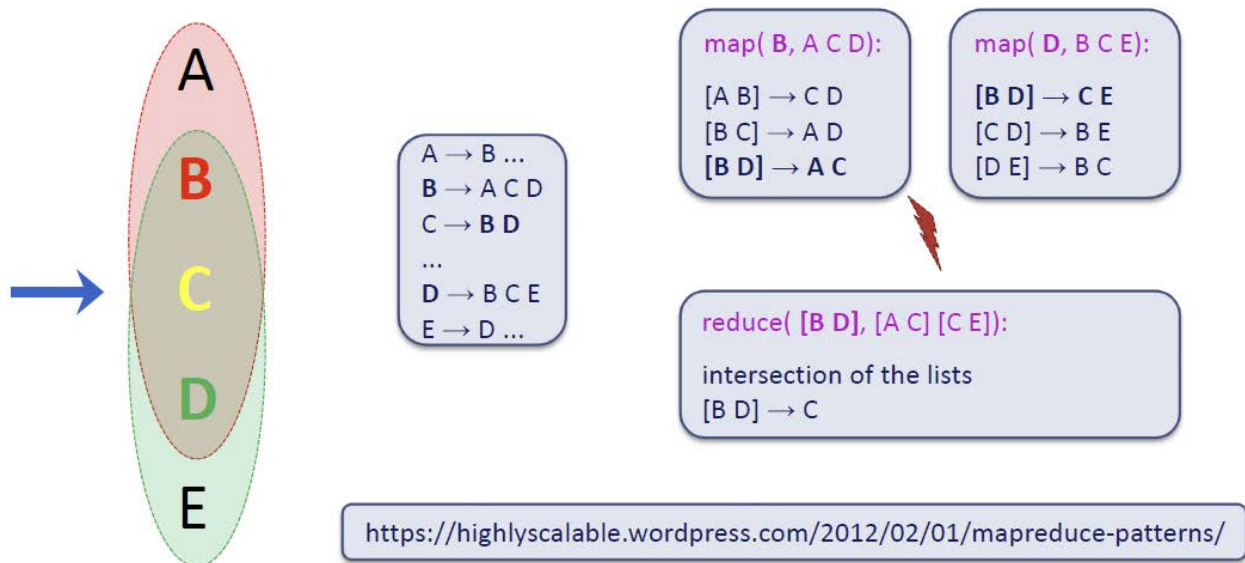
- Zaměříme se na data, jejich ukládání a zpracování
- Možnosti, kterými disponují cloudové platformy pro ukládání dat
- Vertikální škálovatelnost
 - Navýšení vlastností prostředku
 - Navýšení kapacity disku
 - Zvýšení frekvence CPU
 - Jednodušší možnost
 - Problém: technologické limity
 - Cena nad určitou hranici je větší než lineární, takže se to nevyplatí
- Horizontální škálovatelnost
 - Replikace jednotlivých prostředků tak, aby potřebný výkon byl mezi jednotlivé prostředky rozložen
 - Místo jednoho gigantického disku budu mít 20 menších
 - Můžeme takhle přesáhnout libovolné technologické limity
 - Levnější
 - Aby to rozumně fungovalo tak potřebujeme netriviální synchronizační mechanismy
- SQL
 - Všechny cloudové platformy poskytují několik druhů SQL databází
 - Jsou replikované
 - Využití cloudových SQL databází je stejné jako když běží SQL databáze na našem PC
- NoSQL
 - Velice často se používá
 - Not-Only-SQL
 - Zvýšená výkonnost (NoSQL je hlavně optimalizované pro výkonnost)
 - V SQL musí data dodržovat určitá pravidla
 - Také transakční zpracování vyžaduje zvýšenou režii
 - Chybí jim některé vlastnosti, které jsou obvyklé pro SQL
 - Potřebujeme ale obvykle zpracovávat obrovské quanta dat → hodí se nám NoSQL
- Key-Values tables
 - Nejjednodušší způsob jak uchovávat velké množství dat (nejen v Cloudu)
 - Nejde o relační databázi → jde o NoSQL
 - Umožňuje pod klíčem uchovávat hodnoty
 - Navržené pro horizontální škálovatelnost
 - V typické implementaci jsou data rozdělena na jednotlivé partitiony
 - Každé jedna data jsou **právě** na jedné partitioně
 - Nejedná se o replikace
 - Ale o **sharding** (disjunktní rozdělení)
 - Každé jedna data jsou indexovány pomocí dvojce partition key a row key

- Partition key určuje shard (neboli partition)
 - Row key je identifikátor v rámci partitionu
 - Jsou velice výkonné
 - Nezdržují se transakcema, indexováním, ...
 - Výpočetní síla oproti SQL databázím je omezená
 - Podporuje pouze je vyhledávání a uspořádání podle složeného klíče partition key a row key (žádné joiny a tak)
 - Struktura přístupu
 - Několikaúrovňová
 - Pod každým účtem (Account) máme množinu tabulek (Table - KeyValue table)
 - V rámci každé tabulky první část klíče jednoznačně určuje partition
 - Druhá část určuje entity
 - Nemusí být jednoduchý typ
 - Denormalizované ukládání dat
 - Designer by měl odhadnout nejfrekventovanější typ dotazu a podle toho data uspořádat
 - Žádné pevné schéma neexistuje
 - Různé entity mohou mít různé atributy
 - Dvousečná zbraň
 - Partition key a row key můžou být libovolně uživatelsky definované
 - Partitioning
 - Operace na jedné partitioně jsou atomické
 - Příklad (blogovací server)
 - Partition key: téma (identifikace tématu)
 - Row key: datum a čas příspěvku
 - Další atributy: text, hodnocení
- Sharding (další informace, které s ním souvisí)
 - Část klíče, která jednoznačně určuje partition se převede na identifikaci disku
 - Pro intervalové dotazy jsou vhodnější range keys (partition A je od nějaké hodnoty do nějaké)
 - Pokud nepotřebujeme intervalové dotazy, tak vhodnější je hashový klíč (na základě hodnoty se spočítá hash, který určuje partition/disk kde data jsou)
- Column stores
 - Sloupcově orientované databáze
 - Identifikace a adresace je složitější oproti keyvalue tables
 - Klíč je složen ze 3 částí
 - Column family
 - Row
 - Column
 - Jedna řádka představuje instanci materializovaného dotazu
 - Optimalizované pro ně
 - Sloupců může být obrovské množství
 - Nepodporují joiny

- Jde to naprogramovat maličko jednodušeji než u Key-Values tables
 - Konzistence jejich replik
 - Každá DB může být na libovolném počtu replik
 - Jednotlivé dotazy můžou mít specifikovanou jakou konzistenci podporují
 - Sekvenční konzistence
 - Eventuální konzistence
- Dokumentové databáze (Document stores)
 - Zavádějící název → neukládají textové dokumenty ale strukturované data
 - Engine vždy podporuje nějakou množinu dokumentů (XML, JSON, CSV, ...)
 - Rozumí jejich obsahu
 - Uloží si je efektivně, aby se mu s tím co nejefektivněji pracovalo
 - Umožňují sharding
 - Journaling
 - Vnitřní struktura (jedno z následujících)
 - Referenční model
 - Blíží se relačnímu SQL modelu
 - Máme jednotlivé entity rozdělené jako samostatné objekty
 - Když samostatný objekt se objeví uvnitř objektů (více), tak je na něj vedený odkaz
 - Řešíme prostor
 - Referenční integrita
 - Embeddovaný model
 - Data jsou uložena serializovaně
 - Téměř žádná režie navíc
 - Nemá podporu pro vícenásobné klíče a vazby
- Grafové databáze
 - Obzvlášť v poslední době jsou čím dál více populární
 - Umožňují ukládat rozsáhlé grafy a dělat nad nimi dotazy
 - Sociální sítě
 - Hledání přátele přátel uživatelů, kteří mají nějakou vlastnost
 - Modelovat v SQL databázi, tak se zblázníme + drahé joiny
 - Optimalizované právě pro tyto operace
 - Jazyk podobný SQL (nějak upravený)
 - Zatím není žádný standard, každá společnost, která je velkým hráčem na poli DB má vlastní názor na to, jak dotazovací jazyk má vypadat
- BLOB
 - K uchovávání velkých nestrukturovaných dat
 - BLOB = velká posloupnost bytů bez struktury
 - Nepotřebujeme nad nima dělat DB operace
 - Víde
 - Block BLOB
 - Sekvenční přístup
 - Umožňují append (přidání dat na konec blobu)
 - Page BLOB

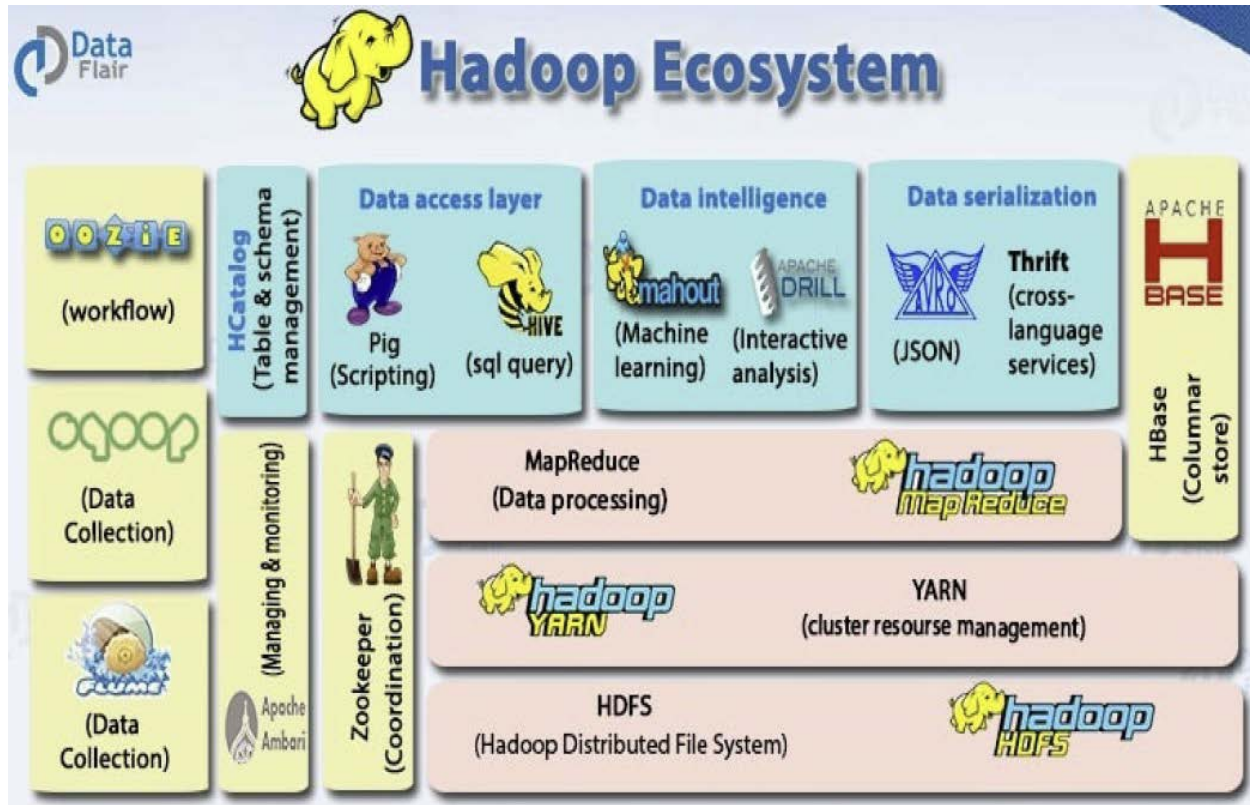
- Stránkově orientované
 - Náhodný přístup read/write
 - Můžeme přistupovat k libovolné stránce
 - Vyšší režie
- Business Analytics
 - Reporty
 - Aby obrovské množství dat bylo užitečné, tak se musí zpracovat, abychom z nich získali "moudro"
 - 2 typy uživatelů
 - Uživatel s kravatou
 - Na základě dat dokáže vydělávat miliony/miliardy
 - Nerozumí low-level technickým detailům
 - Uživatel bez kravaty
 - Neřeší miliardy
 - Rozumí technickým detailům
 - Umí pracovat s datama
 - Uživatel s kravatou řekne uživateli bez kravaty, co by potřeboval z obrovských dat získat
 - Uživatel bez kravaty to udělá
 - Existuje řada nástrojů jak tyto reporty získat a udělat
 - RDL
 - Report definition language
 - Takové pa SQL
- Teď se začneme zabývat zpracováním dat
- Big Data = data, které není možné zpracovat jedním DBMS
- Map / Reduce paradigma
 - Pro masivně paralelní výpočty
 - Pro fyziky a biology prvotně
 - Uživatelé chtějí napsat pouze jádro výpočtu, paralelizaci a distribuci řeší engine
 - Omezené a nedokonalé (z dnešního pohledu)
 - Díky jednoduchosti a omezenosti umožňoval a umožňuje použití masivně paralelního HW nad velkými daty
 - V mnoha různých implementacích
 - Budeme se hlavně bavit o Hadoop
 - Základní schéma
 - Vstupní data je rozdělena v rámci distribuovaného FS na splity
 - Tyto splity jsou zpracovávány první funkcí, kterou implementuje uživatel
→ **map**
 - Výsledkem funkce je množina klíč hodnota
 - Potom co jsou výsledky (klíč hodnota) uloženy zpátky do distribuovaného systému, tak proběhne **shuffle / merge**
 - Všechny hodnoty se stejným klíčem se umístí na jednu repliku
 - Ty jsou zpracovány jedním **reducerem**
 - Výsledek reduceru je klíč/hodnota

- Map
 - Funguje na každém uzlu nezávisle
 - Input $\rightarrow$ (key, value)
- Reduce
 - (key, list of values) $\rightarrow$ output
 - Sjednocení hodnot pro každý klíč
- Příklad
 - Spočítat četnost slov v mnoha dokumentech
 - Soubory jsou v mnoha replikách na distribuovaném FS
 - $\text{map}(\text{file}) \rightarrow (\text{word}, \text{count})$
 - file = soubor
 - word = slovo
 - count = počet jeho výskytů v rámci tohoto souboru
 - (Toto je udělá paralelně na tisíci replikách)
 - Po shufflingu reducer dostane word (slovo) a množiny jeho výskytů v souborech (list of counts)
 - $\text{reduce}(\text{word}, \text{list of counts}) \rightarrow \text{count}$
- Příklad (Common friends)
 - Pro každou dvojici osob, chceme zjistit jaké mají společné přátele
 - $\text{map}(\text{person}, [\text{friends}]) \rightarrow ([\text{person}, \text{one friend}], [\text{other friends of person}])$
 - Všechny [p1, p2] jsou zmergovány a poslány do reduceru
 - $\text{reduce}([\text{person1}, \text{person2}], [[\text{friends}]]) \rightarrow [\text{person1}, \text{person2}], [\text{common friends}]$



- Master/slave architektura
 - Master je jobtracker, zprostředkovává interface pro klienty (programátory)
 - Sám si udržuje frontu jobů a přiřazuje ji TaskTrackerům
 - TaskTracker
 - Je umístěn na každém výpočetním uzlu
 - Nekomunikují přímo s klientem, vše dostávají od job trackeru

- Zpracovávají **map** a **reduce** funkce
- Vhodné pro
 - Velké data (velké objemy)
 - Nestrukturované data
- Nevhodné pro
 - Malé data
 - Nevypatí se celá tato mašinérie
 - Pokud úlohy nejsou snad zparallelizovatelné
- Hadoop



- Celá zoo okolo map / reduce
- HDFS
 - Hadoop Distributed File System
 - Optimalizován pro Hadoop
 - Není problém 10 000 uzlů a 1 000 000 souborů
 - Stupeň replikace je defaultně 3
 - S výpadkami se počítá jako součást distribuovaného systému
 - Silně orientován na velké soubory
 - Původně pouze imutabilní soubory
 - Dnes podporuje append do souborů
 - NameNode
 - Master
 - Stará se o soubory a adresáře
 - Spravuje metadata

- DataNode
 - Slave
 - Úložiště bloků
 - Nic neví o souborech
 - Provádí operace zápisu a čtení na bloky
 - Pig
 - Map / reduce je příliš low level
 - Zároveň nepodporuje složitější data flow
 - Pig je vlastně nadstavba nad map / reduce
 - Implementuje procedurálně orientovaný jazyk PigLatin
 - Je možné pomocí něho vytvářet joby, které se transformují na map / reduce
 - Hive
 - Pokud Pig s PigLatin nestačí
 - Vlastní jazyk Hive QL
 - Deklarativní jazyk
 - Zhruba jako SQL
 - Neříkáme co má engine dělat, ale říkáme jaký chceme výsledek
- Apache Spark
 - Založené na map / reduce
 - Optimalizované pro iterativní operace, kdy není zapotřebí, aby se všechny výsledky ukládaly do databáze
 - 10-100x rychlejší než standardní implementace map / reduce v Hadoop
 - Snadnější programování
 - Podporuje více operací
 - Join, group-by, filter
 - Existuje několik dalších nadstavb nad ním
- Apache Flink
 - Optimalizované pro zpracování dat, které je rozdělené na nelineární data-flow
 - Optimalizace datových toků
- Presto
 - Hive je pěkný ale šíleně pomalý
 - Oproti ruční implementaci je to mnohonásobně pomalejší
 - Původně proprietární technologie Facebooku
 - V roce 2013 open-source
 - Hodně modulární, je možné použít s
 - HDFS, Kafka, různé SQL ...
 - Výkonné
 - 10x rychlejší než Hive
 - Hlavní myšlenkou je neukládání všech dat na disk ale jejich uchovávání v paměti

Přednáška 11

- Síťování a komunikace
- Virtuální síť

- Umožňuje přiřadit vybraným prvkům v cloudové infrastruktuře síťové adresy, které jsou součástí naší podnikové sítě
 - Můžeme přímo komunikovat s VM v Cloudu
 - Point-to-Point
 - Přímé propojení dvou uzlů
 - Point-to-Site
 - Propojení k množině uzlů
- Pokud chce aby provoz vůbec nešel přes Internet, tak velcí poskytovatelé nabízí dedikované linky, připojíme se přímo na jejich Edge Routery
- Každé datacentrum je propojeno do (alespoň dvou) Edge Availability Domain
 - Podobný mechanismus jako u Availability Set
 - Když dochází k upgradu a údržbě v DC, tak vždy se pracuje pouze s jedním Availability Setem
 - Způsob jak zařídit aby údržba DC neohrozila naši aplikaci
 - Jediný rozdíl je ten, že Edge Availability Domain je na síťové úrovni
- Traffic Manager
 - Máme hodně replik geograficky oddělených (např. pro kontinenty)
 - Traffic manager řeší, který požadavek půjde na kterou repliku
 - Jeho funkce záleží na konkrétním nastavení
 - Load balancing (podle vytížení replik)
 - Routování ke geograficky nejbližšímu
 - Round robin plánování
 - 2/3 sem, 1/3 tam
- Cloud Messaging
 - U komunikace mezi Cloudovými aplikacemi je zásadní rozdíl jestli je pomocí Relay nebo Brokeru
 - Relay
 - Zprostředkovává komunikaci mezi dvěma koncovými uzly
 - Synchronní obousměrné spojení
 - Zajišťuje komunikaci dvou klientů za NATem
 - Broker
 - Samostatný uzel
 - Odesílat a příjemce nemusí běžet v tu samou dobu
 - Poskytovatelé obvykle umožňují komunikaci s externími komponentami
- Message Queueing
 - Základní způsob komunikace pomocí brokerů je komunikace přes zprávy
 - Asynchronní jednosměrné spojení
 - Oba dva konce nemusí být naživu
 - Fronta zajistí persistenci a zachování zpráv do té doby než je zpracována příjemcem
 - 1 zpráva je pro 1 příjemce pouze
- Vyzvedávání zprávy
 - Příjemce si požádal o zprávu, fronta mu vydá první zprávu na řadě a zprávu smaže

- Není to odolné proti výpadkům
- Peek/Lock systém vyzvedávání
 - Když si příjemce vyzvedne zprávu, tak ji fronta nesmaže ale uzamkne
 - Příjemce až tu zprávu smaže, tak frontě pošle zprávu, že zpráva je zpracovaná a může si ji smazat
 - Ještě tam existuje Abandon příkaz od klienta
 - Říká frontě, že nechce zprávu
- Competing consumers (návrhový vzor, který s tímto souvisí)
 - Způsob doručování, o kterém jsme si povídali
 - Nějaká množina odesílatelů posílá zprávy do kanálu a množina příjemců je přijímá a reaguje na ně
 - Každý příjemce se hned po tom zpracuje zprávu hlásí o další zprávu
 - Automatický load-balancing
 - Vyrovnávání využití replik
- Throttling (návrhový vzor, který s tímto souvisí)
 - Některé služby jsou kritické a musíme je zachovat za každou cenu
 - Některé jsou nekritický (nevadí když nějakou dobu nebudou fungovat)
 - Pokud máme omezené prostředky a zátěž stoupá
 - Nekritické služby zastavíme na nějakou dobu
 - Tím si uvolníme prostředky
 - Nechám běžet pouze kritické služby
 - Až budu mít dostatek prostředků, tak opět spustím nekritické služby
 - Velice těsně souvisí s návrhovým vzorem Microservices
- Topic (někdy publish / subscribe)
 - Dalším typem komunikace
 - Je brokerovaná
 - Každá zpráva má nějaké vlastnosti, na tyto vlastnosti je možné se dotazovat
 - Z pohledu příjemce: chci jenom zprávy, které ve strukturované části mají nějaké příslušné hodnoty
 - Faktury nad 10 000
 - 1 zpráva může být pro několik různých příjemců
 - Můžu si zvolit jestli chci nespolehlivé doručování, nebo spolehlivé doručování
- Partitioned Broker
 - Jelikož zpráv přes publish / subscribe systémy může jít hromada
 - Aby broken se nestal brzdou tak může být partitionovanej
 - Typické horizontální škálování
 - Máme jakoby více replik brokerů
 - Jaká replika se vybere
 - Třeba pomocí round robin
- Event Hub (někdy IoT Hub)
 - Pro zpracování velkého množství rychle se opakujících zpráv
 - Vysoká orientace na průchodnost, výkon, spolehlivost
 - Předpokládá se, že zdrojem zpráv jsou IoT zařízení
 - Telemetrie

- Informace o tom, že nám v lednici dochází pivo
- Narozdíl od queue je zde preferovaný partition consumer model
 - Jednotlivé zprávy jsou rozděleny do partitionů
 - Partition je pevné množství
 - Jeden consumer odebírá jednu partitionu
- Je možné používat HTTP protocol
 - Nízká výkonnost
- AMQP (Advanced Message Queueing Protocol)
 - Velice často se používají specifické protokoly
 - Binární
 - Vysoce orientovaný na výkonnost
 - Velmi rozšířený
- Je podporováno odesílání batch zpráv (balíků zpráv)
- V partitionu jsou data postupně uchovávaná
 - Na jedné partitioně může být několik příjemců
- Jedna aplikace může mít data rozdělená do několika partitionů
- Pro každou partitionu existuje příjemce
- Sjednocení toho, kde consumeri mají pozici v partitionech se říká view
 - Který zprávy dočetli



- Může to být používáno k ukládání checkpointů
 - Kdyby došlo k haváriím, tak můžeme pokračovat ve zpracovávání dalších zpráv
- Zprávy se nezahazují po zpracování jedním consumerem
- Nechávací se tam nějakou dobu (např. 1 den)
 - Dá se nastavit na jinou dobu
- Push-based messaging (svázané se serverless computingem)
 - V serverless computingu když vývojář napíše funkci, tak ta pasivně čeká dokud není událostí vyvolaná
 - Tomuto se říká push-based messaging

- Všechny předchozí způsoby komunikace bylo iniciované příjemcem
 - Tady to funguje obráceně
 - Příjemce se zaregistruje a v okamžiku, kdy zpráva je k dispozici, tak automaticky je funkce zavolaná na příslušných datech
- Celá řada dalších funkcí, které nám Cloudové služby nabízí
 - Dead Letter
 - Speciální fronta na prošlé nebo nedoručitelné zprávy
- Caching
 - Key-values tables jsou rychlé
 - Ale uchování dat v cachi je mnohem rychlejší
 - Souvisí v návrhovém vzorech cache-aside
 - Všechny data u kterých je nenulová pravděpodobnost, že naše aplikace je bude potřebovat v dohledném čase, tak si je nacachovat
 - Je možné dělat na mnoha různých úrovních
 - Uvnitř jedné aplikace
 - Mezi uzly v Cloudu
 - Dva cachovací frameworky
 - Memcached
 - Starší
 - Dříve to byl pouze lokální cachovací software
 - Vlastní nacachovaná data jsou uloženy ve formě vlastní distribuované hashovací tabulky
 - Redis
 - Persistentní in-memory DB
 - Všechny data jsou vlastně na disku, ale pomocí write-through cache je poslední množina záznamů v paměti
- CDN (Content delivery Network)
 - Při distribuci multimediálního obsahu (zejména)
 - Přímý přenos finále fotbalu (mistrovství světa)
 - To žádný jeden uzel nezvládne
 - Uživatel se připojí k PoP (Point of Presence)
 - Na základě požadavky se podívá jestli má soubor nacachovaný)
 - Když jo, tak vrátí data z cache
 - Jinak požádá vlastníka (nebo další PoP v hierarchii) o data a nechá si je v cachi
 - (Nechá si tam data až do TTL)
- V dnešní době roste nabídka služeb z oblasti AI, Machine Learning, Deep Learning
- Mapy a geografické informační systémy

Něco o bezpečnosti

Jaké základní problémy a rizika jsou spojeny s během firemních aplikací v cloudu. Jak byste je při návrhu aplikace minimalizovali.

- Cizí přístup k datům
 - Do DC centra když si dáme vlastní rack, tak si ho můžeme zamknout v kleci
 - U cloudu pomalu nevíme kde je
- Dostatečné fyzické zabezpečení
 - Souvisí s předchozím
- Nejistota uložených dat
- Likvidace poškozených disků
 - Jsou poškozené disky opravdu zlikvidovány?
 - Jak?
- Dodržování právních předpisů
- Dlouhodobá životaschopnost
 - Bude tady Cloudový poskytovatel příští rok?

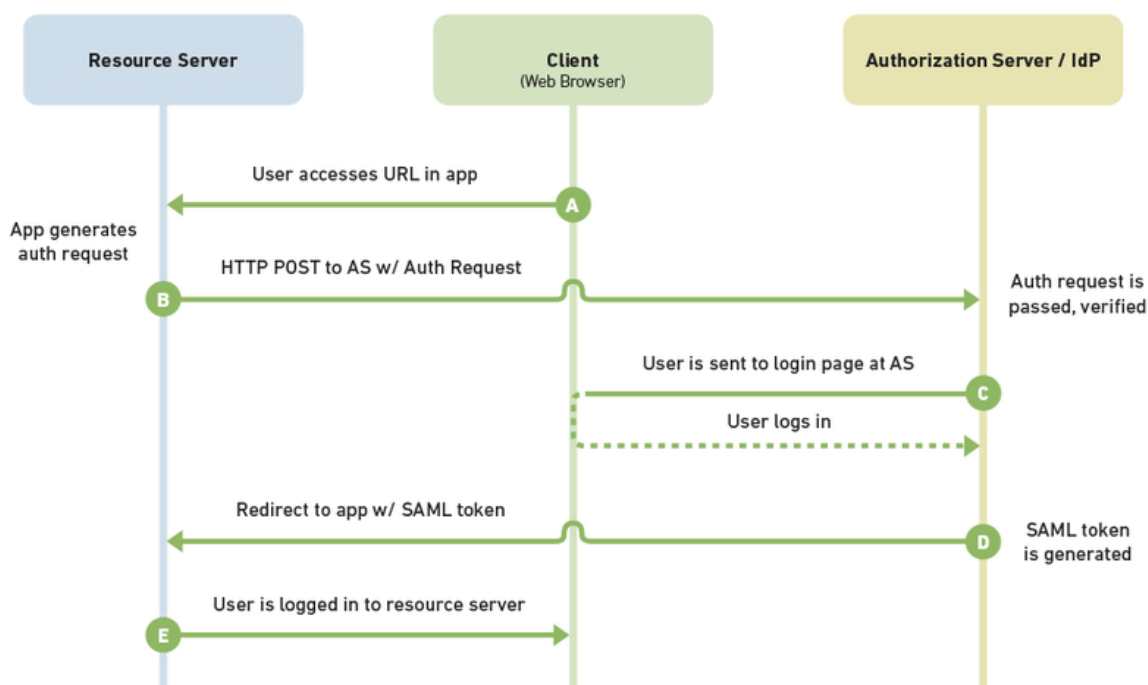
Co je síťová virtualizace a jaké jsou její výhody. Jaké technologie/protokoly se v této oblasti používají - IEEE 802.1Q - VLAN Tagging, NVGRE.

- IEEE 802.1Q
 - Jednu fyzickou Ethernetovou síť rozdělíme do vícero logických sítí (VLAN)
 - Do hlavičky Ethernetového rámce se přidá položka VLAN ID
 - Tagované Ethernetové rámce jsou doručeny pouze těm zařízením, které patří do dané virtuální sítě
 - Je možné routovat mezi VLANy
 - Switch bude mít nastavený trunkový port k routeru, který bude znát VLANy a mít mezi nimi povolené routování
- Tagování můžu provést na úrovni switchu, kde jeden port přidám do VLANy
 - `switchport access vlan x` (na Cisco)
- Nebo i virtuálním switchem (ve VMware ESXi)
- NVGRE (Network Virtualization using Generic Routing Encapsulation)
 - Představen Microsoftem (s jeho partnery)
 - Windowsy dlouho neuměli VXLAN, kvůli lobby Microsoftu
 - Stejný problém řeší protokol VXLAN
 - Maximální počet VLAN je 4094 (12-bit + 2 rezervované hodnoty)
 - NVGRE řeší problém, když nám dojdou VLANy
 - Tuneluje Layer 2 (Ethernetová rámce) přes Layer 3 (Pakety)
 - Definuje pouze jak enkapsulovat a přeposlat "data"
- VXLAN (Virtual Extensible LAN)
 - Přidá dalších 12 bitů pro VLAN ID (dohromady máme 12 + 12 = 24 bitů)
 - 16 mil VLAN
 - Navrženo a protlačováno Ciscem, VMwarem

Vysvětlete koncept claims-based identity. Jaké přináší výhody z hlediska bezpečnosti. Uveďte příklady používaných protokolů a jeden vybraný popište včetně sekvenčního diagramu.

- Claims-based identity
 - Běžné metody používané aplikacemi pro získání identifikačních informací uživatele, který byl autentikován jinou aplikací
 - Claim
 - Informace o uživateli
 - Aplikace věří claimu, pokud věří entitě, která vydala claim
 - Autentikace skrze celý ekosystém něčeho, aniž bychom se museli 60-krát loginovat
- Např. SAML
 - Zde je claim SAML token
 - Další příklady JWT, OpenID Connect

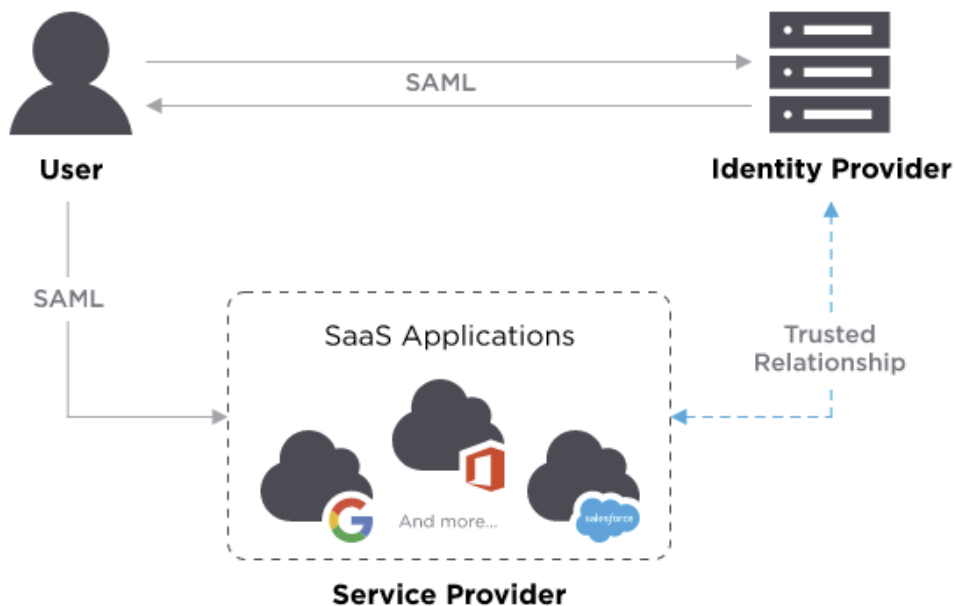
SAML 2.0 Flow



Porovnejte SAML a JWT tokeny. Které vlastnosti mají společné a v čem se naopak liší. Ve kterých protokolech se používají.

- Oboje jsou security tokeny nezávislé na programovacích jazycích, které nám umožní přistoupit na různé webové aplikace pomocí jednoho přihlašovacího účtu
 - Autentikační informace jsou předány mezi identity providerem (IdP) a webovou aplikací (SP - service provider)

- Uživatelská autentikace je centralizovaná v identity provideru
- SAML (Security Assertion Markup Language)
 - Založené na XML



- Zahrnuje i vlastní protokol
- SAML vs OAuth
 - Oboje pro řešení přístupu
 - SAML je pro autentikaci
 - OAuth je pro autorizaci
- JWT (JSON Web token)
 - Novější než SAML
 - Pouze formát na token (nezahrnuje žádný protokol)
- OpenID Connect
 - Rozšíření OAuth spolu s JWT a HTTPS
 - Pro autentikaci

Co je to multifaktorová autentizace. Uveďte příklady mechanismů ověření. Co je TOTP a jak funguje. Jaké jsou vhodné způsoby řešení ztráty kryptografického tokenu.

- Multifaktorová autentizace
 - Alespoň dvě různé ověřovací metody pro zajištění dostatečné bezpečnosti
 - Heslo + potvrzení na mobilu
- Dnes běžné
 - Google, GitHub, ...
- TOTP (Time-based one-time password)
 - Algoritmus pro generování hesla
 - Používá aktuální čas jako zdroj jednoznačnosti
 - Takové to jak steam guard generuje kódy na login

- Ztráta kryptografického tokenu
 - Mít k dispozici nějaký záložní způsob (třetí token)
 - Jako na steamu, když ztratím steam guard (ověření přes SMS)

Bluepill attack

- Útok na hypervisor
- Nainstalovaný rootkit, který zachytí všechna volání a přesměruje je bez vědomí operačního systému
- Spustí se velmi tenký hypervisor, který bude virtualizovat zbytek stroje pod sebou
- Všechny call a interrupty zachyceny hypervisorem a mohou být falšovány

SSO

- Uživatel se autentikuje na serveru A, pak se chce přihlásit na server B
- B zjistí že vstupuje ze serveru A, zeptá se serveru, jestli už se neautentikoval u něj a může rovnou povolit vstup
- Podporuje SAML i JWT

Agentless antivirus

- Klasické antivirové programy (jako např. ESET Nod32) jsou tzv. Agent-based
 - V PC běží nějaký "agent"
- Agentless antivirus je bezpečnostní mechanismus, který se používá ke chránění virtuálů
 - Abychom nemuseli instalovat antivirus na každé virtuálce zvlášť
 - Velcí antivirový hráči nabízí svoji verzi antiviru i v agentless verzi, která umí komunikovat např. s VMware API a dokáže ochránit všechny virtuálky