Tehnologija

Veljko Petrović
Oktobar, 2022

Tehnologije računarstva u oblaku

Hardversko/softverska pozadina

Svrha predavanja

- Ovo predavanja pokušava da prođe kroz osnove tehnologija koje čine računarstvo u oblaku mogućim
- Glavne tehnologije koje nas zanimaju su
  - Mrežne tehnologije visokog kapaciteta
  - Tehnologije računarskih postrojenja
  - Virtualizacija
  - Web tehnologija
  - Tehnologija višestruke upotrebe
  - Tehnologija servisnih arhitektura

Šta nije svrha predavanja

- Nije svrha ovoga da se sve ove tehnologije u potpunosti nauče i razumeju ovde
- Prvo, nema vremena.
- Drugo, trebalo bi da dosta ovoga znate, barem na nekakvom nivou.
- Umesto toga, ovo nabrajanje, koje je nužno na visokom nivou i vrlo komprimovano služi da bi razumeli kako se sve ovo povezuje i kako čini računarstvo u oblaku i *mogućim* ali i ograničenim.

Ograničenost i mogućnost

- Generalno bilo koja tehnologija koja omogućava nekakvu pod-oblast računarskih nauka, je takođe i glavna međa koja *ograničava* tu pod-oblast.
- Asinhrona priroda komunikacija u modernoj Web aplikaciji povezana sa DOM stablom kao univerzalnim jezikom prikaza (sa svim svojim ograničenjima) je glavni razlog zašto razvoj interfejsa izgleda kako izgleda i ima prioritete koje ima.
- Da su se svi browser alati dogovorili da imaju brz način piksel-savršene kontrole prikaza na vreme, na primer, situacija bi bila znatno drugačija.

Mrežne tehnologije visokog kapaciteta

Kako se podaci kreću?

“Oblak” zavisi od Interneta

- Računarstvo u oblaku nikada ne bi uspelo da nije internet komunikacija
- Sa jedne strane Internet je vitalan da bi mogli da imamo dovoljno korisnika da nam uopšte *treba* CC
- Sa druge, samo Internet omogućava mrežnu fleksibilnost da koristimo računarske resurse distribuirane preko celog sveta
- Naravno, dovoljno velika firma bi mogla da ima potpuno lokalni oblak čiji su korisnici unutar firme i gde su sve veze preko LAN tehnologije.

CC bez Interneta

- Kasnije kada budemo diskutovali arhitektonske šablone CC instalacija biće više reči o ovom pod-tipu
- Dovoljno je reći da
  - Nije dominantan: većina upotrebe CC je preko Interneta i to sa dobrim razlogom.
  - Organizacija dovoljno velikog LANa će biti veoma slična onoj koja se koristi za Internet, barem danas.

Hijerarhija ISP

- Internet nema gotovo nikakvo centralizovano upravljajuće telo izuzimajući ICANN (mada je čak i on onoliko koliko može biti distribuiran)
- Uprkos tome, Internet nije, baš, decentralizovan koliko je distribuiran
- On postoji kao kontinuirano povezivanje ogromnih mreža koje opslužuju (kao unificirajući entiteti) Internet Service Provider institucije prvog ranga

Hijerarhija ISP

- Te ogromne mreže (često nacionalne ili čak trans-nacionalne) se sastoje uglavnom od velikih spojenih mreža
- Te velike mreže, često regionalnog karaktera, koje se spajaju su mreže koje stvaraju ISP drugog ranga
- A regionalne mreže se, pak, sastoje uglavnom od srednjih spojenih mreža: to su mreže ISP trećeg ranga i, iako su male u odnosu na mreže čiji konstituent čine, i dalje su velike.
- Bilo koji rang ISP može imati i obične, individualne klijente ili spajati mreže.

Hijerarhija ISP

- Dok ste na FTN-u vi ste korisnici Akademske Računarske Mreže Univerziteta u Novom Sadu (ARMUNS) koja je za vas ISP
- Ceo ARMUNS se spaja sa ARMU-ovima drugih univerziteta u Srbiji da tvori akademsku mrežu cele Srbije, koja funkcioniše kao ISP za univerzitete i spaja njihove mreže.
- Dalje, ta akademska mreža Srbije je (verovatno) objedinjena sa drugim mrežama istog tipa u okviru nekakve objedinjene mreže Evrope, koja funkcioniše kao ISP za tu mrežu.
- Ova arhitektura je ugrubo

Zašto nas ovo zanima?

- Ovakva arhitektura Interneta pokazuje da postoji zanimljiva *lokalnost* u mrežama.
- Razdaljine nisu nužno geografski određene (mada tiranija brzine svetlosti i dalje opstaje) nego zavise u velikoj meri od broja skokova između uređaja neophodnih da se ode od jednog do drugog dela mreže.
- To znači da se određenim entitetima isplati da imaju direktno naloge kod provajdera prvog ranga
- Primer: Netflix

Priroda Internet komunikacije

- Internet komunikacija ne ide uvek istim putanjama
- Ako želimo da komuniciramo, recimo, sa drugim ARMUNS računarom šanse su ogromne da nikad nećemo napustiti ARMUNS mrežu i *najverovatnija* putanja je uz nivoe hijerarhije i onda horizontalno, to nije neophodno ili sigurno
- Sistem je konfigurisan tako da podaci mogu da idu bilo kojom putanjom od čvora do čvora dok ne stignu do svog odredišta, potencijalno istovremeno iz mnogo strana
- To znači da ako se neki link optereti ili prekine, mreža se ‘samoizleči’ i nađe se alternativna putanja

Kako je ovo moguće?

- Postoje dve fundamentalne tehnologije koje omogućavaju ovo ponašanje:
  - Datagramski paketni mrežni protokoli
  - Dinamičko rutiranje

Datagramski paketni mrežni protokoli

- Svi podaci u ovakvoj mreži se pakuju u pakete sa maksimalnom dužinom i svim metapodacima neophodnim da podaci unutar paketa stignu na pravo odredište
- Paketi su numerisani i može se identifikovati pravi redosled paketa ako neki izostane ili stigne na odredište u pogrešnom redosledu
- Paketi su adresirani sa jedinstvenim identifikatorom odredišta
  - Na IP mrežama to je IP adresa koja je ili 32 ili 128 bita u zavisnosti da li govorimo o IPv4 ili $IPv6% adresama
  - Na nižem nivou, unutar mreža Ethernet tipa to je MAC odnosno Media Access Control adresa koja je 48-bita i određena na nivou fizičkog uređaja.
  - Kada podatak stigne na računar, softver odgovoran da obrađuje podatke se bira na osnovu *porta.* proste 16-bitne numeričke vrednosti koja se tipično označava kao broj od 1 do 65535

Slojevi?

- OSI mrežni model priznaje sedam slojeva
  - Fizički - Vodi računa o tome kako variranjem fizičkih veličina da se prenose binarni podaci (osmožilni kabl)
  - Logičko-vezni - Vodi računa o tome kako da se tok bita prenese od jednog do drugog uređaja koji dele fizički medijum, jedinica transporta je *frejm* (Ethernet)
  - Mrežni - Vodi računa o komunikaciji između različitih fizičkih medija, jedinica transporta je *paket* (IP)
  - Transportni - Vodi računa o tome da svi podaci iz jednog pokušaja slanja odista stignu na drugu stranu i podpadnu pod kontrolu aplikacije koja je odgovorna za njih, jedinica transporta je *segment* (TCP)
  - Sesijski - Vodi računa o tome da se zna koje komunikacije pripadaju kojoj sesiji i da se prati stanje konekcije, autorizacije i identiteta (HTTP ali vidi kasnije)
  - Prezentacioni - Vodi računa o formatiranju podataka koji se šalju (JSON/utf8)
  - Aplikativni - Radi nešto korisno sa podacima (REST applikacija)

Slojevi?

- U praksi se koristi model koji proističe iz TCP/IP protokol seta i koji ima *pet* slojeva zato što su sesijski, prezentacioni, i aplikativni sloj kombinovani u samo jedan, aplikativni sloj.
- Zbog toga je ono sortiranje na prošlom slajdu imalo ‘vidi kasnije’ u sebi.
- U praksi odgovornosti zadnjih tri sloja su kombinovane

Kombinacija slojeva

- Ono što čini ovaj model moćnim jeste sposobnost slaganja mrežnih slojeva tako da Ethernet frejm sadrži IP paket koji sadrži TCP segment koji sadrži...
- Budući da svaki nivo ne zna šta znače bitovi koje prenosi (tj. IP ne zna da li su bitovi koje prenosi TCP ili UDP ili nešto sasvim novo), ovaj sistem može lako da se menja.
- Najintuitivnije je kako se isti TCP/IP stek prenosi i kroz bežične i žičane mreže

Tuneliranje

- To štu su sadržaji jedinica transporta proizvoljni setovi bajtova omogućava trik poznat kao *tuneliranje*
- Ideja je da se u protokol višeg nivoa kao sadržaj umetnu podaci koji pripadaju nižem protokolu
- Onda se na stani primaoca softver konfiguriše tako da ekstrahuje te podatke i tretira ih kao da opisuju taj niži protokol
- Ovo je neverovano moćna tehnologija koja čini koncept VPN tehnologije mogućim

koja omogućuje da se dve lokalne mreže objedine u jednu.

Tuneliranje

- Klasičan primer tuneliranja jeste upotreba SSH protokola (koji pripada aplikativnom sloju) da u svom sadržaju (gde normalno idu komande za udaljenu kontrolu računara) šalje pakete i segmente klasične TCP/IP veze
- Time je moguće da pristupate lokalnim mrežnim resursima udaljeng računara kroz mrežu na bezbedan način, budući da morate uspostaviti autorizovan enkriptovan kanal pre bilo kakve komunikacije.
- Ovakvi tuneli se mogu uspostaviti i protokolima namenjenim za baš ovo, ako je takva veza dugotrajna onda se ovo još i zovu virtualizovane privatne mreže, budući da se preko tog šifrovanog kanala formira veza

Dinamičko rutiranje

- Ruter je mrežni uređaj (interno običan računar) koji ima više mrežnih interfejsa koji su deo različitih mreža
- Funkcija rutera je da više mreža objedini u jednu tako što usmerava (rutira) komunikaciju između tih mreža
- Ruter ne može da zna topologiju kompleksnog sistema u koji je uključen, nužno, ali mora da zna koje linkove ima i da ima nekakvo pravilo koje šalje pakete u jednom smeru ili drugom
- To ‘pravilo’ se tipično zove *tabela rutiranja*

Dinamičko rutiranje

- Neki uređaji imaju lak posao
- Vaš kućni ruter mora da pamti samo dva pravila
  - Ako adresa počinje sa, na primer, 192 . 168 . 1 i ima kao zadnji bajt bilo šta od 1 do 255, onda se to šalje na lokalni mrežni interfejs (za ovaj interfejs je često fabrički i neraskidivo vezan svič)
  - Sve ostalo ide na WAN interfejs ka vašem provajderu.
- Kompleksniji ruteri imaju kompleksnija pravila ali statičke tabele rutiranja imaju ograničenu upotrebnu vrednost

Dinamičko rutiranje

- Zamislite da vam je jako bitan internet protok i zato plaćate račune kod dva provajdera
- Onda vam treba ruter sa *tri* mrežna interfejsa
- Takođe vam treba pravilo kada koji paketi idu kroz koji link
- Oba WAN linka vode, potencijalno, ka celom ostatku interneta
- Pravilo zavisi od potreba sistema, uspostavljenih veza i brda drugih faktora
- Treba vam dinamičko rutiranje

Dinamičko rutiranje

- Za provajdere situacija je još komplikovanija
- Imaju razne linkove ka raznim drugim mrežama
- Potencijalno imaju više linkova ka istoj drugoj mreži
- Svaki link je jednako dobar u smislu toga što će, eventualno, sve što pošalju da stigne tamo gde treba
- Ali koja putanja je adekvatno brza?

Dinamičko rutiranje

- Ruteri u ovakvim situacijama koriste kompleksne algoritme koji na osnovu topologije veza i trenutnog stanja linkova dinamički biraju odredište za bilo koji specifičan paket
- Faktori koji utiču na ‘sudbinu’ paketa su, naravno, adresa, veličina, i tajming, baš kao i trenutno stanje mreže i njenog opterećenja
- Jednako bitno kao i sve ovo je i faktor poznat kao QoS

Quality of Service

- Ovaj termin se koristi u dva konteksta
  - Kvalitet komunikacije između neke dve tačke u mrežnom sistemu
  - *Traženi* kvalitet komunikacije
- Ako uređaj kaže da podržava QoS onda se misli na značenje 2
- Kada ima više paketa, sistem može birati da odabere neke kao hitnije u odnosu na druge

Uticaj ovih faktora na CC

- Dva mrežna faktora su glavna u slučaju komunikacije za računarstvo u oblaku:
  - Protok
  - Kašenjenje
- Protok je količina informacija koja teče između neke dve tačke (vašeg sistema u oblaku i krajnjeg korisnika)
- Kašenjenje je koliko vremena prođe od zahteva za podacima i prvih podataka koji stignu, tipično se meri u milisekundama

Uticaj ovih faktora na CC

- Protok je problem ako komunikacija nadmaši kapacitet linka
- Ovo se može rešiti novčano, tako što se zakupi još kapaciteta
- Jedini rizik je u slučaju prenosa velikih količina podataka gde postoje i ekonomski faktori i faktori kapaciteta infrastrukture
- Kašnjenje je mnogo veći problem

Kašnjenje i otvoreni internet

- Nemoguće je a priori ustanoviti kojom će putanjom da ide nekakav podatak kroz mrežu sa dinamičkim rutiranjem koja je dovoljno kompleksna.
- To znači da je kašnjenje bilo kog paketa nepredvidivo u zavisnosti od smera u kome dinamičko rutiranje pošalje neki ključan paket
- Čak i ako postoje QoS garancije, vrlo je teško održati te tokom prolaska kroz domene nekoliko različitih provajdera
- Ovo znači da je CC rešenje za probleme gde je potrebno vrlo malo kašnjenje neprikladno.

Moguća rešenja

- Jedini način da se utiče na kašnjenje jeste ako je moguće nešto uraditi povodom mrežne topologije obe strane komunikacije
- Dakle, ne u slučaju krajnjeg korisinika, ali možda u slučaju komuinikacije dva sistema
- Tajna može biti da se sistemi smeste bliže u kontekstu mrežne topologije
- Alternativa je upotreba više provajdera usluga računarstva u oblaku

Tehnologije računarskih postrojenja

Industrijalizacija računarskih postrojenja

Šta je računarsko postrojenje

- Računarsko postrojenje jeste prostor, infrastruktura, i oprema koji su namenjeni za pružanje računarskih usluga na skali industrije
- To znači da je organizacija prostorija, prisutna infrastruktura, i korišćene tehnologije sva isključivo namenjena navedenoj svrsi, bez kompromisa
- Ovako nešto je neophodno da bi bilo isplativo da se održavaju dovoljno velike koncentracije računarskih kapaciteta da bi njihovo prodavanje kao usluge računarstva u oblaku imalo smisla.

Modifikacija prostora

- Tipično za ovakav prostor je da očekujete posebno spuštene tavanice i podignute podove radi lakšeg provlačenja kablova i kontrolisanja sistema za hlađenje.
- Koriste se napredni protivpožarni sistemi koji ponekad koriste tehnike izmeštanja kiseonika
- Posebni sistemi napajanja

Modularizacija

- Generalno svi računari u ovoj primeni su apsolutno standardizovanog oblika i napravljeni da idu na standardni rek za mrežnu opremu
- Standardni rek je 19 inča (482.6 mm) u unutarnjoj širini i definiše ga međunarodni standard IEC 60297
- Isti standard specificira ‘rek jedinicu’ koja se tipično označava sa U
- Jedan U je 44.45 mm odnosno jedan i tri četvrt inča i jedinice visine na reku su umnožak ove vrednosti
- Standardni rek je 42U visok

Modularizacija

- Ideja je da se hardver može smestiti optimizovanim radosledom i rasporedom bez brige da nešto neće stati
- Ovo je naročitno bitno zato što se očekuje da će delovi sistema morati brzo i lako da se menjaju, a standardizacija znači da je pristup individualnim koimponentama brz i relativno bezbolan.

Automatizacija i udaljen rad

- Računarska postrojenja su maksimalno automatizovana i često mogu da operišu polu-autonomno, pokrećući još računara kada je to potrebno i gaseći one koji nisu
- Jednako automatizovano je i kreiranje novih sistema sa mehanizmima automatskog boot mehanizma preko mreže i headless instalacijama, što omogućava ne samo podizanje virtuelne mašine automatski nego čak i instalaciju na ‘metalu’
- Kada je i potrebna ljudska interakcija, ona je kade god je to moguće, odrađena udaljeno koristeći obezbeđen link za transfer komandi.

- Jedino mesto gde je ljudska interakcija neminovna jeste rešavanje hardverskih problema.

Pouzdanost

- Kada je u pitanju pravljenje računarskog postrojenja koje je jako otporno na otkaz, vodeće pravilo je one is none
- Drugim rečima, ne dozvoljavaju se jedinstvene tačke otkaza (SPOF) i svi ključni sistemi su duplirani
- U slučaju, na primer, servera, tipično je imati dva potpuno odvojena, posebna napajanja gde jedno od njih služi manje-više isključivo da ima izvor struje spreman na licu mesta istog trenutka kada primarni izvor otkaže

Bezbednost

- Budući da se računarski resursi (pa i podaci koje smeštaju i obrađuju) nalaze na jednom centralnom mestu lakše je fizički obezbediti uređaje
- Fizička zaštita servera je izuzeztno bitna
- Većina računarskih sistema nema nikakvu zaštitu od direktnog fizičkog pristupa
- Rešenja su rigorozna fizička bezednost, bezbednosni protokoli
- Podaci u mirovanju valja da budu enkriptovani da bi zaštitili sistem od krađe podataka direktnim putem

Prilagođen primarni hardver

- Primarni hardver sistema su njegovi element sposobni za proračune
- Ovi su prilagođeni industrijskoj eksploataciji
- Veće performanse
- Hot-swap mehanizmi
- Povećana pouzdanost

Prilagođen hardver za čuvanje podataka

- Jedan od velikih poslova bilo kog računarskog centra je čuvanje ogromnih količina podataka
- Ovo zahteva posebnu tehnologije
- Prva stavka jeste kako će se uređaji za čuvanje podataka spojiti sa sistemom, glaven dve opcije su:
  - DAS (device attached storage) što znači da je uređaj zakačen direktno za primarni hardver kroz host bus adapter (HBA) što odgovara klasičnom slučaju diska zakačenog za računar.
  - NAS (network attached storage - u širem smislu) što znači da je uređaj zakačen za nekakvu mrežu na koju se kače uređaji primarnog hardvera

NAS

- Ako pravimo mrežu diskova imamo izbor da li povezujemo *direktno* diskove i omogućavamo da se tom umreženom sistemu pristupa u blok-po-blok režimu ili da postoji poseban uređaj koji upravlja diskovima, a preko mreže se pristupa na nivou individualnog fajla.
- Prvi pristup se zove SAN (storage area network) i implementira se kroz tehnologije kao što je SCSI ili Fibre Channel ili Fibre Channel over Ethernet ili InfiniBand
- Drugi pristup je NAS u užem smislu i koristi se ista mreža kao i za sve ostalo sa protokolima koji operišu na nivou fajla kao što su SMB ili NFS

Kombinacija diskova

- Jedan disk nema zadovoljavajuće karakteristike kada je CC u pitanju
- Nije dovoljno velik, brz, ili pouzdan
- Srećom moguće je kombinovati diskove na načine koji povećavaju sve ove osobine
- %RAID%, JBOD ili napredni fajl sistemi kao što je btrfs

Prilagođen mrežni hardver

- Računarskim centrima je potreban jako brz mrežni pristup, kao što smo pričali
- To se rešava redudantnim vezama kao i jako brzim mrežnim tehnologijama
- Optika se jako često koristi gde se jedno optičko vlakno može koristiti da, multipleksirano, prenosi sav saobraćaj mnogo gigabitnih mrežnih linkova.

Mrežni hardver svestan sadržaja

- U potrebi za većom brzinom koriste se često posebni mrežni elementi koji služe da ubrzavaju web
- To se radi tako što se mrežni transport osposobi da razume šta paketi rade na aplikativnom nivou
- Ovo znači da se podaci mogu preprocesirati ili enkriptovati/dekriptovati još u mreži
- Naročito bitno je znati čemu su paketi namenjeni što omogućava da se radi efektno balansiranje opterećenja

Virtualizacija

Zamišljeni a stvarni računari.

Šta je virtualizacija

- Virtualizaicja je mehanizam kojim se jedan fizički računar ponaša kao više nezavisnih računarskih sistema koji su potpuno odvojeni.
- Virzualizacija nema jako dobro rešenje zato što računari već decenijama rade ovaj trik na ovaj način ili onaj
- Procesi, na kraju krajeva, su barem delimično odeljeni i ponašaju se (donekle) kao ‘sami na svetu’ što se tiče memorije
- Još teže je konceptualno odvojiti virtuelizaciju od *emulacije*

Emulacija i virtuelizacija

- Mogućnost jednog računarskog sistema da se pretvara da je drugi računarski sistem je nešto što je ugrađeno u same osnove ove oblasti
- Čerč-Tjuringova teza znači da, performanse i prostor na stranu, bilo koji računarski sistem se može u principu ponašati kao bilo koji drugi računarski sistem
- U najgorem slučaju, vi možete napisati emulator koji replicira ponašanje bilo kakvog digitalnog čipa u sistemu apsolutno tačno
- Čak i postoje emulatori za, npr. neke stare igračke mašine koje imaju ovaj nivo granualrnosti

Razlika između emulacije i virtualizacije

- Razlika nije u nameni ili čak i efektu nego u tehnologiji i očekivanim performansama
- Ako nešto opisujemo kao virtuelizovano to znači da očekujemo performanse koje su uporedive sa performansama pravog sistema
- Sa druge strane, voljni smo da prihvatimo ograničenja
- PC može da *emulira* 6502 čip, ali ga nikad neće virtuelizovati

Najopštija definicija virtuelizacije

- Ova sekcija je uglavno o virtuelizaciji računarskih sistema
- Strikno govoreći skoro bilo koji servis se može ‘virtualizovati’
- Ova najšira forma virtuelizacije je mehanizam kojim se heterogeno polje resursa predstavlja krajnjem koisniku kao nešto homogeno
- Divan primer je elektrosnadbevanje: kranji potrošač ne zna da li je struja poreklom iz termoelekgrane, hidroelektrane, nuklearne elektrane...
- Ceo taj kapacitet generacije, skladištenja, transporta, itd. je apstrahovan kroz virtuelizaciju.

Bazična arhitektura virtualizacije

- Virtualna mašina (Od sada VM) se koncipira kao da je stvaran računar koji se u stvarnosti nalazi na nekvom fizičkom računaru koji se obično zove ‘domaćniski’ odnosno host računar.
- Za taj host računar kažemo i da ima host operativni sistem.
- Konsekventno, VM onda postaje računar-gost, odnosno guest računar i ima svoj, guest operativni sistem.
- Ključna osobina tehnike virtuelizacije jeste da guest operativni sistem i sav softver koji se izvršava na njemu *ne zna* da je na VM-u, a ne na fizičkom računaru.

Da li VM zna da je VM?

- Ovo nije 100% istina
- Tipično, sofver apsolutno ne zna da li je u virtuelnoj mašini ili ne
- Operativni sistemi, donekle, jesu svesni da su u promenjenom okruženju
- Ovo nije neophodno, ali se često operativni sistemi podifikuju ne bi li se dobile bolje performanse
- Tipićan primer: Guest Additions

Šta omogućava virtuelizaciju?

- Virtuelizacija je omogućena kroz mehanizam koji upravlja servisima virtuelizacije
- Ovaj mahanizam ima različita imena: Meandžer virtuelnih mašina, Monitor virtuelnih mašina, ali najčešće je poznat pod standardni industrijskim nazivom ‘Hipervizor’ (eng. Hypervisor)
- Da bi razumeli ovaj malo čudni naziv valja napomenuti da se kernel operativnog sistema vrlo često zove supervisor
- Stoga nešto iznad super mora biti hyper

Tipovi hipervizora

- Hipervizori dolaze u dve glavne forme, hipervizori tipa 1, i hipervizori tipa 2
- Ovo se ponekad još zovu i hipervizori ‘golog metala’ i ‘ugošćeni’ hipervizori
- Engleski termini su bare metal i hosted
- Ova podela nije nužno iscrpna

Hipervizori tipa 1

- Hipervizor tipa 1 se izvršava direktno na hardveru, bez bilo čega ispod sebe
- To znači da takav hipervizor ima određene osobine operativnog sistema, ali nije klasičan operativni sistem
- Ovo je najperformantnija forma zato što ima što manje između vritualne mašine i stvarnog hardvera
- Primeri hipervizora tipa 1 su Xen i VMware ESXi

Hipervizori tipa 2

- Hipervizor tipa 2 se izvršava kao proces operativnog sistema
- Ovo ima bezbednosne prednosti (zato što je proces dodatno ograničen na način na koji hipervizor tipa 1 ne može biti), i olakšava primenu takvog sistema na računarima za ličnu upotrebu ili na računarima mešane upotrebe
- Glavna mana hipervizora tipa 2 jesu performanse: pošto svaki sistemski poziv esencijalo poziva emulirani sloj hardvera koji poziva sistemske pozive operativnog sisteam domaćina koji poziva stvarni hardver, generalno performanse hipervizora tipa 2 su *znatno* lošije

- Primer koji je vama *jako* poznat bi trebao da je VirtualBox

Hipervizori tipa... 1.5?

- Ova podela izgleda iscrpno ali nije
- Šta recimo uraditi sa KVM mehanizmom
- KVM mehanizam je modul koji se dodaje u kernel Linux operativnog sistema koji efektivno pretvara Linux u hipervizor prvog tipa
- Drugim rečima dozvoljava aplikacijama koje se izvršavaju na nivou korisnika da zatraže da se nešto virtuelizuje na ‘metalu’ umesto u korisničkom prostoru (Tipična kombinacija i ona koja pokreće VM u kojem se pišu baš ove reči je QEMU/KVM)

Pitanje hardvera

- Ovaj mehanizam funkcioniše utoliko ukoliko može da izvršava kod odvojeno i može da drži memoriju odvojeno ali udara u problem čim treba da se uradi bilo šta vezano za ulaz/izlaz
- Nemoguće je jednostavno nekomplikovano premeštati ove stvari direktno u pravi hardver: ako imamo deset VM-ova i samo jedan disk, ko dobije da piše u superblok? Čiji je koji mrežni paket?
- Jedino rešenje jeste da se hardver emulira

Emulirani hardver

- Svi hardverski uređaji zakačeni za VM postoje kao emulirane softverske apstrakcije
- Tipičan primer je QEMU
- QEMU je emulator sistema (ima i druge funkcije, ali je ova najzanimljivija)
- Ono što ga čini posebno zanimljivim jeste da je to potpun emulator sa opcionom primenom hipervizora

Potpun emulator sa opcionom primenom hipervizora

- Šta ovo znači?
- Znači da QEMU može *ili* da emulira sav hardver na računaru što uključuje i procesor *ili* da emulira sve osim procesora koji se onda virtuelizuje kroz neki hipervizor, recimo KVM.
- Ovo znači da možemo da sastavimo virtuelnu konfiguraciju kada pokrećemo virtuelnu mašinu

O ilustraciji

- Svaka virtuelna mašina emulira *nešto*
- Mi ovde koristimo kao primer nešto što je lako testirati u kućnoj radinosti
- Sve što vam treba je noviji Linux ali instaliran kao host operativni sistem
- Primer koji mi koristimo je KVM (kao hipervizor), QEMU kao emulator, i libvirt kao API sloj za upravljanje virtuelnim mašinama.

Libvirt

- libvirt nismo do ovog trenuka pominjali
- On sam po sebi ne daje neke nove sposobnosti, koliko pruža uniformni način da se specificira ponašanje virtuelnih mašina
- Obuhvata brdo elemenata i faktora, ali nas će najviše zanimati ugrađeni XML jezik za opis konfiguracije virtuelnih mašina

(Primeri iz **https://libvirt.org/formatdomain.html**)

Domeni

- Bilo šta što emulira, libvirt zove 'Domen'

```
<domain type='kvm' id='1'>
  <name>MyGuest</name>
  <uuid>4dea22b3-1d52-d8f3-2516-
        782e98ab3fa0</uuid>
  <genid>43dc0cf8-809b-4adb-9bea-
        a9abb5f3d90e</genid>
  <title>A short description - title - of
        the domain</title>
  <description>Some human readable
        description</description>
  <metadata>
    <app1:foo
        xmlns:app1="http://app1.org/app1/">.
        </app1:foo>
    <app2:bar
```

Emulacija procesora

- Za %Cloud Computing% nema velike koristi od emulacije procesora
- To je od koristi prilikom razvoja za egzotične platforme, eksperimenata, softverske arheologije...
- Prvi korak je promeniti tip domena tako da više ne koristi hipervizor, vrednost qemu recimo.
- Zatim se mora podesiti željeni oblik procesora

Željeni oblik procesora

```
<cpu match='exact'>
  <model fallback='allow'>core2duo</model>
  <vendor>Intel</vendor>
  <topology sockets='1' dies='1' cores='2'
        threads='1'/>
  <cache level='3' mode='emulate'/>
  <maxphysaddr mode='emulate' bits='42'/>
  <feature policy='disable' name='lahf_lm'/>
</cpu>
```

Virtualizacija kroz hipervizor i procesor

- Kada želimo da ne emuliramo procesor no da koristimo onaj koji je već u mašini kao da je i u virtuelnoj mašini, barem što se tiče sposobnosti što se tiče, npr. pod-funkcionalnosti, onda moramo da kažemo da korsitimo isti procesor kao onaj koji je već na mašini
- Srećom postoji prečica za ovo

Prečica

```
<cpu mode="host-model" check="partial"/>
```

Ovo samo kopira ponašanje stvarnog fizičkog modela procesora na virtualizovani (ali ne emulirani) procesor

Alokacija procesora

- Ne valja da svaka virtuelna mašina nužno ima apsolutno iste resurse na raspolaganju
- Ponekad želimo da uradimo baš ono što nam CC obećava: agregiramo resurse na jednom mestu i particionišemo ih na drugom
- Srećom to je moguće: kod virtualizacije mogu da se sistemu dodele virtuelna procesorska jezgra, a da se ta virtuelna jezgra izvršavaju na onim jezgrima fizičkog procesora koja su odabrana

Primer alokacije virtuelnih procesora

```
<vcpu placement='static' cpuset="1-4,^3,6"
      current="1">2</vcpu>
<vcpus>
  <vcpu id='0' enabled='yes'
      hotpluggable='no' order='1'/>
  <vcpu id='1' enabled='no'
      hotpluggable='yes'/>
</vcpus>
```

Primer alokacije virtuelnih procesora

- Ovo statički alocira 2 virtuelna procesora za koja su odgovorna fizička jezgra iz skupa 1-4, ne računajući jezgro 3, i dodatno jezgro 6.
- Zatim specificira detalje ponašanja oba virtuelna CPU-a.

Emulacija blokovskih uređaja

- Procesori nisu problem, mnogo ozbiljnije pitanje jeste šta uraditi kada je u pitanju, recimo, disk
- Da bi instalirali vaš guest operativni sistem, morate imati nekakav disk
- libvirt će specificirati a QEMU emulirati onakav disk kakav vi odlučite da imate

Emulacija blokovskih uređaja

```
<disk type="file" device="disk">
  <driver name="qemu" type="qcow2"/>
  <source
    file="/putanja/do/fajla/gde/je/disk"/>
  <target dev="vda" bus="virtio"/>
  <address type="pci" domain="0x0000"
    bus="0x04" slot="0x00"
    function="0x0"/>
</disk>
```

Emulacija blokovskih uređaja

- Ovo je emuliran disk kome je na host mašini u pozadini običan fajl
- Specifično to je fajl tipa qcow2
- To je poseban format diska optimiziovan za ovaj slučaj
- Karakteriše ga odlična implementacija copy on write semantike
- Fajl se povećava tek kada postoji prava potreba za prostorom na disku što smanjuje bačen prostor

Problem sa emulacijom hardvera

- Emulacija hardvera je *spora*
- Ovo je naročito nezgodno kod bilo koje I/O operacije
- Još je gore ako je VM namenjen za interaktivno korišćenje
- Postoje dva moguća rešenja:
  - Paravirtualizacija
  - Prespoj na nivou uređaja

Paravirtualizacija

- Paravirtualizacija (PV) je tehnika gde, umesto da je proces virtuelizacije potpuno transparentan za operativni sistem - gost, taj operativni sistem je modifikovan da bi mogao lakše da se virtuelizuje
- Ovo ima dve forme
  - Potpuna paravirtuelizacija
  - Parcijalna paravirtuelizacija

Potpuna paravirtuelizacija

- Ovde je ceo operativni sistem modifikovan na najdubljem nivou da funkcioniše u virtuelizovanom okruženju
- Takav operativni sistem operiše u potpuno ne-transparentnom okruženju i na mestima gde bi, npr, pričao sa hardverom on jednostavno poziva sistemske pozive hipervizora (ponekad se koristi termin hypercalls mada je on relativno redak)
- Takvi sistemi imaju i sekvencu pokretanja koja ‘zna’ da se ne pokreće prava mašina pa se ne bave nikavom inicijalizacijom hardvera (koji ne postoji) no se kod kernela pokreće istog trenutka

Potpuna paravirtuelizacija

- Potpuna paravirtualizacija je bila fantastično brza (komparativno) zato što, umesto da koristi osobine procesora i mikroarhitekture koje je relativno teško virtualizovati i emulirati, sistem jednostavno kaže hipervizoru šta želi i to se desi, veoma brzo i efektno
- Sistem je i dalje transparentan prema softveru u korisničkom prostoru zato što se onaj deo operativnog sistema koji pruža interfejs prema korisničkim aplikacijama ne menja

Smanjenje važnosti paravirtuelizacije

- Jedno vreme, jedini praktičan način da se dobiju prihvatljive performanse prilikom virtuelizacije je bilo da se ‘vara’ kroz PV
- Danas ovo više nije slučaj: potpuna PV je i dalje brza, kao i uvek, ali ju je alternativa (koja se često zove HVM) dostigla pa čak i prestigla u performansama

Smanjenje važnosti paravirtuelizacije

- Kako? Kroz tehnologiju procesora koje imaju posebne tehnike koje eliminišu potrebu da se neke stvari virtuelizuju u softveru
- Ovo znači da se može očuvati nepromenjen procesor i ipak ostvariti performanse koje su ekvivalentne perofrmansama procesora na ‘golom metalu.’
- PV je i dalje jedina praktična opcija ako nema hardverske podrške

Parcijalana paravirtualizacija

- Uprkos tome, paravirtualizacija se i dalje koristi samo u parcijalnoj formi
- Uprkos dobrim performansama koje postiže HVM i dalje postoji usko grlo koje nas je dovelo ovde: simulirani hardver
- Veliko je traćenje vremena što sistem mora da se pretvara da, na primer, priča sa potpuno fiktivnim hard diskom umesto da jednostavno pošalje zahtev za podacima koji su potrebni direktno hipervizoru koji to onda može proslediti gde treba

Parcijalana paravirtualizacija

- Uvid parcijalne paravirtualizacije (koja je još poznata pod terminom PV on HVM što je, recimo, kako to zove kompanija Amazon) jeste da je moguće promeniti samo *deo* operativnog sistema da bi dobili dobre performanse
- Koji to deo operativnog sistema može da se promeni relativno lako? **Drajveri**
- Da stvar bude slađa, nama je potrebna paravirtuelizacija baš da bi radili I/O što, pak, se baš postiže kroz drajvere

Parcijalana paravirtualizacija

- Stoga, umesto da modifikujemo ceo operativni sistem, samo napišemo drajvere za naš virtuelni hardver koji, umesto da rade stvari sporo, rade stvari kroz paravirtuelizaciju i pozivaju hipervizorske sistemske pozive direktno
- Ovim se može postići najbolje od svih svetova: sve performanse sa maksimalnom fleksibilnošću

Primer diska (opet)

```
<disk type="file" device="disk">
  <driver name="qemu" type="qcow2"/>
  <source
    file="/putanja/do/fajla/gde/je/disk"/>
  <target dev="vda" bus="virtio"/>
  <address type="pci" domain="0x0000"
    bus="0x04" slot="0x00"
    function="0x0"/>
</disk>
```

Primer diska (opet)

- Ovde je već angažovana tehnika parcijalne paravirtuelizacije
- QEMU/KVM radi HVM virtuelizaciju, ali ima podršku za podsistem za parcijalni PV
- Zaj podsistem ze zove virtio i on je ovde uključen

Prespoj na nivou uređaja

- HVM + PPV je moćna kombinacija ali ona radi samo gde se hardver i dalje virtuelizuje
- Nekakav hardver nije baš tako lako virtuelizovati
- Klasičan primer je GPU
- Rešenje koje je formirano je korišćenje hardverskih funkcionalnosti (Ključne reči su Vt-d i IOMMU u zavisnosti od tehnologije) da se sva komunikacija između računara i fizičkog komada hardvera usmeri na jednu virtuelnu mašinu

Prespoj na nivou uređaja

- Možete misliti o ovome kao o tehnici koja vam omogućava da ‘utaknete’ pravi hardver u virtuelnu mašinu
- Može se raditi na nivou individualne PCI kartice, tipično to je GPU uređaj, ali u praksi može i nešto drugo
- Može se raditi na nivou i individualnog USB uređaja kada se obično zove USB redirekcija
- Ovo je tehnologija koja omogućava, npr. da se dobiju CC instance sa grafikom

Moderni standard

- Današnji standard za virtualizaciju koji postiže performanse koje se vrlo teško razlikuju od performansi bez virtualizacije jeste
  - %HVM& za glavninu posla
  - PV parcijalno za emulirani hardver (Gde se emulira većina I/O hardvera)
  - Prespoj na nivou uređaja za GPU uređaje ako su potrebni
- Ovaj standard je sada već dostupan na praktično svakom operativnom sistemu i u serverskom i u lično-računarskom okruženju

Kontejneri i virtuelne mašine

- Tehnika virtuelizacije je neizbežna i korisna
- Ali za veliki broj slučajeva primećeno je da se VM tehnologija koristi da bi se u okviru operativnog sistema X pokreću druge instance istog operativnog sistema, samo da bi se individualne aplikacije držale ‘zatvorene’ u svojim okruženjima.
- Ovo znači da su sposobnosti virtuelne mašine, sva ta mogućnost simuliranog hardvera, drugih operativnih sistema, transparentnosti i svega toga, fundamentalno bačena.

Zašto emulirati Linux u Linux-u?

- Glavni razlog zašto se u produkciji pokreće virtuelna mašina koja je identična svom ‘domaćinskom’ sistemu je
  - Portabilnost
  - Ponovljivost
  - Izolacija

Portabilnost

- Da bi nekakva aplikacija radila potrebno je imati veliki broj potporno softvera: biblioteka, alata, servera, itd.
- To znači da je vrlo teško poslati nekome aplikaciju
- Kada je u pitanju distribucija aplikacija za korisničke sisteme često je potrebna ili kompleksna instalaciona procedura ili impersivno kompleksan sitem za menadžment paketima
- Situacija za servere je još gora jer, osim što je potrebno sve to, mimo toga je još problem konflikta zahteva serverskih aplikacija

Konflikt zahteva

- Serverske aplikacije se često izvršavaju u paraleli na istim mašinama sa mnogo drugih serverskih aplikacija
- Problem sa ovim jeste što jedna aplikacija može da zahteva jednu verziju ključne biblioteke, a druga drugu
- Čija je onda ‘starija?’
- Čak i ako se taj problem reši kroz, recimo, virtualizaciju okruženja ili pakovanje zahteva, šta da se radi sa zahtevima sa različitim verzijama potpornog servera?
- Recimo: šta ako imamo dve Java aplikacije gde svaka zahteva drugačiju verziju `Tomcat` servera?

Konflikt zahteva

- A čak i ako se reši taj problem, ostaje problem interfejsa koji se ne mogu nikako replicirati
- Šta raditi ako imamo dva web servera? Ko dobije portove 80 i 443?
- Ovo sve dovodi do toga da je nemoguće samo ‘pokrenuti’ neku aplikaciju na datom serveru, no je to produžen i mučan proces koji zahteva jako puno konfiguracije i kompromisa i ne može da se lako automatizuje.
- Budući da je ‘samo pokreni’ neophodan korak ako želimo CC u bilo kojoj formi, portabilnost kroz virtuelizaciju je neophodna

Ponovljivost

- Vrlo bitna stavka kada se softver nađe u produkciji jeste da ne može da se desi da se neki sitan element hardversko-softverske konfiguracije promeni i softver naprasno prestane da funkcioniše ili, možda još gore, počne da radi *drugačije*
- Ovo je boljka dobro poznata svakome u ovoj oblasti i zove se “Ali radi na mojoj mašini”
- Stoga, pojavila se potreba da postoji nekakav recept (idealno takav da se može automatizovati) koji, ako se prati, proizvede *tačno* okruženje u kome aplikacija treba da radi

Izolacija

- Ne samo što želimo da sprečimo slučajne konflikte između aplikacije, takođe želimo da sprečimo namerne konflikte
- Ako neko uspešno izvrši napad na bilo koji serverski proces na nekom sistemu i dobije šansu da šalje udaljene komande, apsolutno svi procesi na tom sistemu su sada ugroženi
- Postoji separacija između procesa, ali ona (u podrazumevanom okruženju) je nesavršena
  - Što?

Izolacija

- Izolacija između virtuelnih mašina je znatno bolja
- I dalje je moguće, kako se to kaže, da se ‘pobegne’ iz virtuelne mašine, ali je ovo barem za red veličine teže i lakše za odbranu zato što je površina napada mnogo bolje definisana
- Tako se potencijalno maliciozan sadržaj može držati ‘zarobljen’ što povećava bezbednost celog sistema

Motivacija za konejnere

- Budući da nam ne treba poseban hardver, ne treba egzotična CPU arhitektura, i ne trebe nikakav novi operativni sistem, virtuelizacija ima dosta bačenih resursa
- Da li bi mogla virtualizacija ali lakša po sistem, ograničena samo na separaciju userspace okruženja koja dele, nekako, kernel?
- Ako da, kako bi to radilo?

Opšta ideja

- Opšta ideja ovog problema se zove virtualizacija na nivou operativnog sistema
- Umesto da se pretvaramo da imamo ceo nov računar, mi umesto toga ubedimo operativni sistem da se pretvara da je više različitih instalacija
- Onda još samo ubedimo operativni sistem da se pretvara da je druga instalacija prema drugim procesima i time obezbeđujemo potpunu izolaciju

Opšta ideja

- Možemo ovo da uradimo zato što procesi u korisničkom prostoru u potpunosti zavisni od operativnog sistema da otkriju šta se dešava
- Ako operativni sistem ‘laže’ na sistemske pozive, program u korisničkom prostoru to ne može da otkrije
- Jedini izazov jeste kako ubediti operativini sistem da drži konzistentne skupove laži i da govori prave laži pravom softveru
- Zato se ovo uvek radi preko alata koji delimično imaju podršku na nivou kernela, a delimično kao alati u korisničkom prostoru.

Preteča – chroot

- Najranija forma koja podseća na kontejner jeste koncept `chroot` u POSIX sistemima
- Ova komanda (i sistemski poziv) služi da, kada se pokrene neki proces, može da mu se promeni šta je to tačno `root` direktorijum za njega
- Ovo je način da se proces zarobi u hijerarhiju direktorijuma koja se odabere i taj proces ne može ni da imenuje (pa stoga ne može da otvori) fajlove fan te hijerarhije
- Ovo znači da možemo da napravimo mini Linux instalacije za svaku aplikaciju gde svaka ima svoje zahteve prema fajlovima ispunjene.

Ograničenja

- Naravno `chroot` je stara ideja i esencijalno ima vrlo slabu bezbednost i nikakav način da se zaštiti bilo šta *osim* fajlova
- Uprkos tome, ovo je početak ove ideje i doveo je direktno to izuma, 1999, BSD `jail` komande koja omogućava da se sistem particioniše u 'zatvore' koji omogućavaju da se servisi sistema rigorozno odvoje jedni od drugih.
- Za razliku od `chroot`, `jail` komanda ima i bezbednosne komponente i predstavlja nešto veoma blizu modernim kontejnerima.

Tehnologija iza modernih kontejnera

- Najčešće korišćeni modenri kontejneri proističu iz mehanizma koji se nalazi u modernim Linux kernel-ima i koji se zove *sistemski prostori imena* (eng. namespaces)
- Ideja između sistemski prostora imena (SPI) jeste da je moguće definisati, unutar kernela, particije resursa sistema i prikazivati samo jednu particiju određenim procesima.
- To znači da možemo imati dva procesa koji vide dva potpuno disjunktna skupa resursa

Šta se može sortirati u SPI

Linux poseduje osam tipova SPI: 1. `Cgroup` - Omogućava da svaki SPI vidi poseban set kontrolnih grupa koje Linux koristi da meri i ograničava resurse sistema 2. `IPC` - Omogućava da svaki SPI vidi poseban set resursa za komunikaciju između procesa 3. `Network` - Omogućava da svaki SPI vidi svoj sopstveni mrežni interfejs, pravila rutiranja, firewall i, generalno, ceo mrežni stek 4. `Mount` - Omogućava da svaki SPI vidi samo određeni podskup integrisanih sistema fajlova

Šta se može sortirati u SPI

5. `PID` - Omogućava da svaki SPI ima naizgled svoju tabelu procesa u kojoj se procesi drugih SPI ne nalaze
6. `Time` - Omogućava da svaki SPI ima svoja podešavanja vezana za sistemsko vreme
7. `User` - Omogućava da svaki SPI ima potpuno svoj set korisnika uključujući i 'svog' `root` korisnika i svoju interpretaciju dozvola za svoj pod-domen fajlova
8. `UTS` - Omogućava da svaki SPI vidi potencijalno različit `hostname`

Kako odavde do kontejnera

- Ako se resursi sistema rigorozno iseku na SPI i koristi se `chroot` da se neki proces zarobi u svoj pod-direktorijumu nešto što neodoljivo podseća na kontejner postaje moguće
- Ovo znači da je moguće maltene generisati kontejner koristeći prostu komandu kojai je deo `utils-linux` paketa

Unshare

```
unshare -r -i -p -f --kill-child -R
        /home/veljko/tmp2 ./tst
```

Objašnjenje

- `-r` podešava da će se korisnik unutar ovog našeg kontejnera mapirati na ‘root’ korisnika
- `-i` podešava da pravimo novi, odvojen IPC SPI
- `-p` podešava da pravimo novi, odvojen PID SPI
- `-f` podešava da pokrećemo potpuno nov proces sa ovom komandom za koji će biti podešeno sve što smo specificirali
- `--kill-child` grozomorno nazvana opcija koja kaže da se svi procesi-potomci terminiraju kada se terminira `unshare`
- `-R` podešava nov `root` direktorijum za nov proces

Objašnjenje

- Sama ./tst komanda je trivijalni C program koji simulira ls komandu nad root direktorijumom
- Služi da se demonstrira da smo za ovaj naš nov proces `root` nešto sasvim drugo

tst.c

```c
#include <dirent.h>
#include <stdio.h>

int main() {
  DIR *d;
  struct dirent *dd;
  d = opendir("/");
  if (d) {
    while ((dd = readdir(d)) != NULL) {
      printf("%s\n", dd->d_name);
    }
    closedir(d);
  }else{
    fprintf(stderr, "Ne mogu da otvorim
        korenski direktorijum.\n");
```

Rezultat

```
.
..
chroot_works
tst.c
tst
usr
lib
lib64
```

Rezultat

- Izlistan, kao da je `root` je u stvari `tmp2` direktorijum
- Jedino ograničenje je to što, da bi sistem radio, mora imati pristup `glibc`
- Stoga, radi jednostavnosti, direktorijumi sa bibliotekama su ulinkovani
- U praksi, mi bi koristili posebne alate koji zahtevaju malo biblioteka ili su kompajlirani u statičkoj formi

Šta nam ovo govori?

- Verovali ili ne, ovo je dovoljno da se počne razvoj vaše verzije alata Docker
- Naravno, Docker radi mnogo više stvari na mnogo kompleksniji način
- Centralna ideja, sa druge strane, stvarno jeste ista
- Umesto da se poziva komanda, poziva se sistemski poziv (isto ime), ali centralna ideja je identična

Šta je još potrebno?

- Ako imamo sve ovo u običnoj korisnoj komandi, koji posao nam radi nešto kao što je Docker
- Jako puno stvari, naravno, ali jedna od glavnih je nešto preko čega sam je preskočio u primeru
- Linkovanje (Pomoću, ako vas zanima `mount -o bind` komande; bonus pitanje: zašto sam koristio `mount -o bind` umesto tvrdih linkova?) svih onih biblioteka nije trivijalan posao
- Veliki izazov je pripremiti okruženje za kontejner

Šta je još potrebno?

- Okruženje za VM je komplikovano, ali je to barem nešto vrlo slično instalaciji operativnog sistema
- Znamo kako da radimo *to*
- Kako napraviti instalaciju nečega što malo jeste poseban OS, a malo nije?
- Ovo se u svetu kontejnera tipično zove image odnosno slika i predstavlja neku vrstu ‘dehidriranog’ kontejnera
- Više o ovome kasnije