Univerzitet u Kragujevcu

Fakultet tehničkih nauka u Čačku

Marjan Milošević

Žarko Bogićević

Računarstvo u oblaku

Čačak, 2026.

Autori:

Dr Marjan Milošević, vanredni profesor, Fakultet Tehničkih nauka u Čačku, Univerzitet u Kragujevcu

Žarko Bogićević, MSc, AWS Solution Architect, X1F

Recenzenti:

Prof. Dr Miloš Ivanović, redovni profesor, Prirodno-matematički fakultet, Univerzitet u Kragujevcu

Prof. Dr Danijela Milošević, redovni profesor, Fakultet tehničkih nauka u Čačku, Univerzitet u Kragujevcu

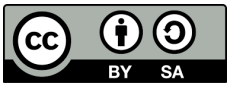
Izdavač: Fakultet tehničkih nauka u Čačku, Univerzitet u Kragujevcu

Za izdavača: Prof. Dr Ivan Milićević, v.d. dekana Fakulteta tehničkih nauka u Čačku

Lektor: Dragana Mijušković, profesor srpskog jezika i književnosti

Tehnički urednik: Marjan Milošević

Nastavno-naučno veće Fakulteta tehničkih nauka u Čačku, Univerziteta u Kragujevcu je svojom odlukom broj 012-9-3122/14 od 24.12.2024. godine odobrilo izdavanje ovog udžbenika.



ISBN 978 86 77762 90 2

SADRŽAJ

Predgovor

Oblak je danas „građanin prvog reda“ u svetu računarstva. Ogroman broj programa (ili, kako se sada uobičajeno kaže – aplikacija) od samog početka se razvija za korišćenje u oblaku, a mnogi od onih koji to već nisu, migriraju se u oblak. Biznis-model mnogih firmi, ne samo onih iz IT sektora, transformisao se i unapredio zahvaljujući korišćenju oblaka, te je u prvi plan izbila i paradigma DevOpsa (Development/Operations), kulture koja se prirodno integriše u ono što oblak pruža – automatizaciju procesa, kontinuiranu isporuku, pouzdanost itd. Time je postala istaknuta i funkcija DevOps inženjera, lica koje predstavlja link između razvoja i infrastrukture. Verujemo da su ova knjiga i predmet sa kojim je povezana, veoma koristan resurs za sve buduće DevOps inženjere, ali i za druge pozicije vezane za razvoj i isporuku softvera.

Prirodan tok kojim nastavni materijali posle nekog vremena sazre i postaju osnova za udžbenik, pratio je i pisanje ovog štiva. Autori su uložili naročit napor da integrišu postojeće resurse, kreiraju niz novih, te posebno porade na jačanju metodičke strane udžbenika. Danas se ponekad mogu čuti teze da su udžbenici zastareli, da je sve već dostupno na Internetu i da pored modernih AI alata, nema svrhe baviti se pisanjem knjiga, posebno u efemernim oblastima kao što su informacione tehnologije. Mi opet verujemo da je potreban „pivot“, osnovni izvor znanja, koji je kvalitetno napisan i koji je pre svega čitljiv i prožet primerima, a da Internet, koji ume da bude „močvara“ u kojoj se lako gubi fokus, treba da daje dodatne smernice i detaljnija objašnjenja. Svakako, pošto IT jeste oblast koja se brzo menja, izabrali smo da izdanje osvane u elektronskom obliku, čime se čuva priroda i olakšava ažuriranje znanja i pisanje novih izdanja.

Udžbenik je koncipiran tako da kombinuje teorijska i praktična znanja i veštine kroz 11 poglavlja i 2 dodatka. Namera nam je bila da prikažemo oblak „i spolja i iznutra“, te je u tom smislu značajna pažnja posvećena i konkretnoj infrastrukturi koja stoji ispod oblaka: mreži, hardveru i protokolima koji se koriste u data-centrima. Praktični primeri su prevashodno realizovani na AWS (Amazon Web Services) platformi, prvenstveno iz razloga što Fakultet tehničkih nauka ima status AWS akademije, te sami studenti imaju mogućnost korišćenja AWS platforme za izvođenje zadataka. Svakako, opšti delovi su zajednički i važe za bilo koju varijantu oblaka i pozivamo čitaoce da istraže i druge platforme.

Udžbenik Računarstvo u oblaku namenjen je prvenstveno studentima Fakulteta tehničkih nauka u Čačku - istoimenom predmetu na četvrtoj godini osnovnih akademskih studija IT, ali ga svakako preporučujemo i studentima drugih fakulteta, kao i svim zainteresovanim čitaocima, koje interesuje ova materija.

Udžbenik smo napisali ravnopravno, s tim što je za teorijski deo uglavnom bio zadužen Marjan Milošević, dok je praktične primere pisao Žarko Bogićević.

Zahvaljujemo se recenzentima, profesoru Milošu Ivanoviću i profesorki Danijeli Milošević, na vremenu i ekspertizi uloženim u čitanje i analizu rukopisa i dragocenim sugestijama kojima su doprineli da on bude još kvalitetniji.

Zahvalnost dugujemo i svojim porodicama, za strpljenje i istinsku snažnu podršku tokom pisanja.

Čitaoce pozivamo da sa nama podele svoje utiske o udžbeniku, a zahvalni smo na ukazivanju na eventualne greške.

Autori

U Čačku i Užicu, 2025.

1. Uvod

Cilj uvodnog poglavlja je diskutovanje o osnovama računarstva u oblaku, motivima nastanka i istoriji razvoja. Student će nakon savladanog poglavlja moći da:

- Navede tipove infrastrukture
- Dâ definiciju računarstva u oblaku
- Objasni sva osnovna svojstva računarstva u oblaku
- Navede ključne događaje u istoriji računarstva u oblaku

Paradigma „klaud kompjuting“, prevedeno sa engleskog *računarstvo u oblaku*, intenzivno je prisutna u poslednjih desetak godina. Današnji mladi inženjeri su „cloud-natives“; to su tzv. digitalni urođenici, koji su u svet razvoja i održavanja softvera zakoračili sa oblakom kao tehnologijom koja je podrazumevana, za koju nije potrebno prilagođavanje. Sa druge strane, seniori su morali da prođu (i još uvek prolaze) reklo bi se bolnu tranziciju od konvencionalnog računarstva ka računarstvu u oblaku. Usvajanje radikalno drugačijih paradigmi u računarstvu obuhvata i druge, manje ili više srodne tehnologije, te se sličan scenario javlja i sa blokčejnom *(blockchain)*, mobilnim računarstvom itd. Ono što je veoma interesantno jeste da postoji određena ciklična evolucija u razvoju računarstva, koja je počela od glomaznih računara (mejnfrejmova) na koje su povezivani klijenti 60-ih godina 20. veka, razvijala se dalje uz „ojačavanje“ klijenata, koji su postali jake radne stanice krajem 20. veka, da bi se, povećanjem kapaciteta mreža, opet došlo do koncepta „glomaznih računarskih resursa“ na koji su povezani klijenti u prvoj deceniji 21. veka. Danas, računarstvo je i dalje „negde između“: zbog optimizacije performansi i distribucije opterećenja, obrada i čuvanje podataka ponovo se decentralizuju u okviru tzv. računarstva na ivici *(edge computing)*. Svakako, ne može se izvršiti stroga generalizacija epoha i računarstvo u svim delatnostima pripisati jednom ili drugom pristupu: u zavisnosti od konkretne delatnosti postavlja se i nivo centralizovanosti rešenja.

Na pitanja „šta je računarstvo u oblaku“, „šta je klaud“, „da li koristite klaud“, informatički laici će verovatno dati odgovor da je u pitanju „sistem“, da se koriste računarski resursi koji su „tamo negde“ itd. Ali pitanje definicije oblaka i računarstva u oblaku može biti izazov i za same informatičare. Da li je oblak isto što i server? Ako bi se krenulo korak dalje, da li je oblak isto što i klaster povezanih servera?

Pre nego što se dođe do formalne definicije oblaka, u nastavku će biti prikazana evolucija infrastrukturnih rešenja i agregacija motiva za uvođenje nove filozofije računarstva: računarstva u oblaku.

Tradicionalna infrastruktura

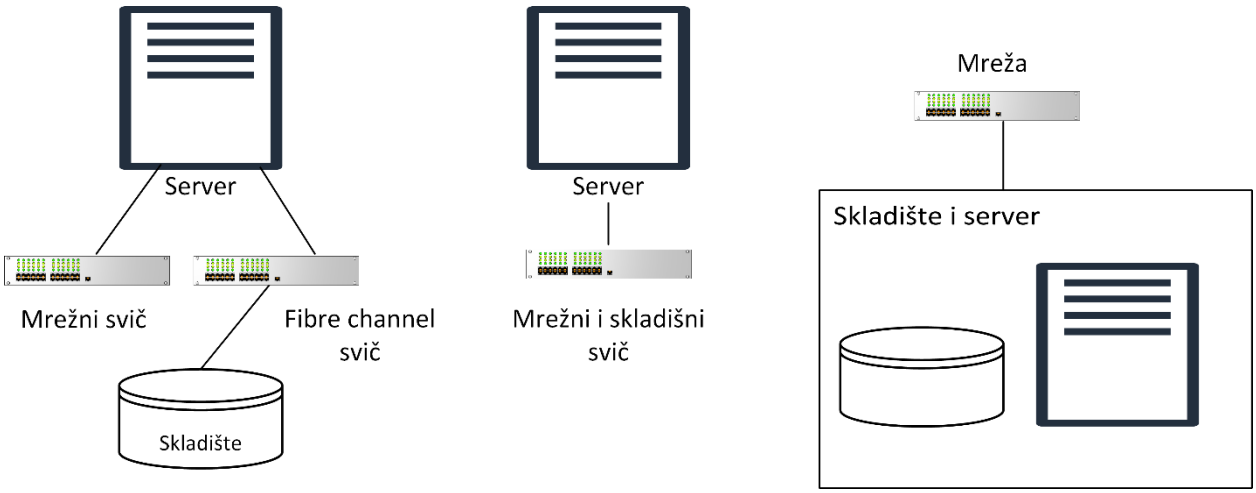
Šta je potrebno da bi se sklopio jedan server? Dakle, jedan računar koji bi bio umrežen i u stanju da neprekidno (24/7) pruža određene usluge. Jedan pristup je tradicionalna infrastruktura. Tri komponente ovakve infrastrukture jesu jedinica za obradu (compute unit), jedinica za skladištenje podataka (storage unit) i mrežna jedinica. Ove tri komponente često se nabavljaju od potpuno različitih dobavljača. Ovakav vid infrastrukture omogućava visok nivo prilagođavanja. Međutim, pošto su komponente potpuno odvojene, za upravljanje obradom, skladištem i mrežom neophodno je koristiti tri različita alata i poznavati pojedinačne sisteme. Takođe, dodavanje novih komponenti može biti problematično.

Konvergirana infrastruktura

Konvergirana infrastruktura pre svega podrazumeva olakšano uspostavljanje i održavanje servera. Tri ključne komponente (obrada, skladište, mreža) deo su jedne predefinisane celine, koja se kupuje od jednog dobavljača i za upravljanje se koristi jedinstven sistem (npr. upravljačka konzola). Konvergirana

infrastruktura se lako dograđuje, ali sa druge strane podrazumeva i vezanost za jednog konkretnog dobavljača.

Obe infrastrukture: i tradicionalna i konvergirana fokusirane su na hardver. Da bi se dalje dobilo na fleksibilnosti, razvijena je **hiperkonvergirana infrastruktura** (HCI – hyper-converging infrastructure). Ona podrazumeva okretanje fokusa ka softveru. U tom smislu, pored osnovne tri komponente koje su prethodno razmotrene, uključene su virtualizacija i softverski definisane mreže.



Slika 1 Tradicionalna, konvergirana i hiperkonvergirana infrastruktura

Za izgradnju infrastukture koja će činiti osnovu računarstva u oblaku mogu se koristiti sva tri pristupa (Slika 1). U modernoj praksi razvoja računarske infrastrukture za oblak, najviše je u upotrebi hiperkonvergirana infrastruktura.

Data-centar

Data-centar predstavlja skup umreženih računarskih resursa visokog kapaciteta, lociranih u specijalnom prostoru, najčešće u posebnim objektima, izgrađenih prema specijalnim standardima. Manje institucije nemaju sopstveni data-centar, već raspolažu uglavnom server-salom, prostorijom, gde su postavljeni serveri u odgovarajućim ormanima zajedno sa opremom za umrežavanje. Veće institucije mogu imati svoj data-centar ili iznajmljivati resurse u komercijalnim data-centrima. Veliki dobavljači usluga u oblaku, kao što su Amazon, Microsoft ili Google, imaju svoje data-centre, dok manji dobavljači često iznajmljuju prostor u komercijalnim data-centrima, gde postavljaju svoju opremu.

Data-centri su izgrađeni po standardima koji omogućavaju pre svega visok stepen pouzdanosti: postoje višestruki izvori električne energije, kao i autonomni sistem za napajanje (npr. dizel agregat), zatim redundantne internet veze, specijalni sistemi hlađenja, protivpožarne zaštite i fizičkog obezbeđenja.

Računarski klasteri

Pojam klastera je tesno povezan sa pojmom računarstva u oblaku. Računarski klaster jeste skup međusobno umreženih računara, sa instaliranim posebnim operativnim sistemima, koji zajednički rade na određenom zadatku. Računari koji čine klaster nazivaju se nodovima. Raspoređivanje izvršavanja na više računara doprinosi efikasnosti i veoma često su računari (virtuelni ili fizički) u oblaku grupisani u klastere. To je često nevidljivo za korisnika, ali mogu postojati i slučajevi u kojima korisnik definiše klastere računara i zadatke koje klaster izvršava.

Sumarno, klastere odlikuju sledeća svojstva:

- Obično se nalaze na jednoj lokaciji, izvršavaju zadatke jedne organizacije;
- Koriste homogen hardver, upravlja se centralizovano i klaster se ponaša kao jedan sistem;
- Sistem nije dinamičan, resursi su rezervisani za tačno određene zadatke i aplikacije nisu prenosive na druge sisteme.

1.1. Svojstva oblaka

„Oblak“ nas asocira na to da je odgovarajuća računarska infrastruktura „tamo negde“, odnosno da nije u prostorijama firme koja je koristi. Suprotnost ovakvoj dislokaciji jeste infrastruktura koja je u prostorijama. Ustaljen je anglosaksonski termin *on premise*.

Da li je to što je infrastruktura udaljena, dovoljno da se klasifikuje kao oblak? Dolazi se ponovo do pitanja da li je udaljeni server isto što i oblak. Mnogi dobavljači usluga imaju fizičke ili virtuelne servere koje iznajmljuju na korišćenje firmama ili pojedincima za potrebe npr. hostovanja web-sajta. Da li je takav server u stvari oblak? Dislociran može biti i server koji je u vlasništvu firme, ali koja ga je premestila u data-centar da bi bio bezbedniji, da bi imao kontinualno napajanje itd. Dolazi se do zaključka da bi oblak uglavnom trebalo da bude udaljen, ali da ta činjenica nije dovoljna da bi se na primer jedan server nazvao oblakom. (Kasnije će se razmotriti i poseban slučaj kada oblak čak nije udaljen!)

Firma ili pojedinac koji koriste resurse u oblaku pristupaju im preko interneta i sve usluge koje se pri tom koriste trebalo bi da su stalno dostupne, što iziskuje visoku pouzdanost infrastrukture. Dalje, korisnici bi trebalo da mogu da na jednostavan način izaberu tip i kapacitet resursa koji im je potreban, preko odgovarajuće apstrakcije (na primer, označi se disk određene veličine i doda postojećim diskovima putem intuitivne web-platforme). Za razliku od fizičke infrastrukture, koju nije jednostavno nadograditi, oblak bi trebalo da omogući lako proširenje kapaciteta ukoliko dođe do porasta opterećenja. Na primer, za vreme praznika broj kupovina preko web-prodavnice poklona se udesetostručuje. Povećanje resursa u skladu sa porastom opterećenja je svojstvo poznato kao skalabilnost. Uopšteno, kada se govori o usklađivanju kapaciteta sa promenljivim opterećenjem, reč je o elastičnosti. Elastičnost podrazumeva da resursi mogu da se smanje ili povećaju, zavisno od opterećenja. Oblak bi trebalo da pruža mogućnost automatskog skaliranja, tako da se na primer broj virtuelnih mašina koje primaju zahteve poveća sa 3 na 10, bez intervencije korisnika. Svi navedeni primeri su dati iz ugla korisnika. U ovom kontekstu, korisnik je korisnik usluga računarstva u oblaku – to može biti firma koja održava dati web-sajt. Prema tome, postoji fizička računarska infrastruktura koju zajedno sa pratećim sistemskim softverom uspostavlja i održava dobavljač usluga u oblaku *(cloud provider)* i postoji korisnik – klijent ovog dobavljača, koji koristeći resurse u oblaku realizuje svoj softver ili koristi gotov softver dobavljača.

Sopstvena infrastruktura iziskuje značajna početna ulaganja: potrebno je kupiti i instalirati opremu i kasnije, tokom eksploatacije, troškovi su manji i svode se na održavanje infrastrukture, iznajmljivanje prostora, plaćanje električne energije. Pri kupovini opreme potrebno je predvideti koji je kapacitet potreban. Ako se potcene potrebe, ubrzo može doći do pada performansi i neophodnosti dokupljivanja komponenti. Ako se precene potrebe u budućnosti, ili čak i ako se dobro procene, sigurno će jedno vreme iskorišćenost opreme biti veoma niska.

Dobavljači usluga u oblaku u mogućnosti su da pruže povoljne usluge zahvaljujući pojavi poznatoj kao *Economy of scale*. Ukratko: kada postoji veliki broj korisnika i veliki broj resursa i usluga, jedinična cena usluge je jeftinija. Na primer, ako fabrika serijski proizvodi stotine komada nameštaja dnevno, moći će da optimizuje troškove proizvodnje, transporta, određeni troškovi će biti fiksni i na kraju će cena jednog komada nameštaja biti povoljnija nego kod konkurenta koji proizvodi desetine komada nameštaja. Slično je i kod računarstva u oblaku: dobavljač koji ima velike homogenizovane resurse, veliki broj

korisnika i specijalizovanu podršku može da ponudi usluge po nižoj ceni nego što bi pojedinca koštalo da sam izgradi sopstvenu malu infrastrukturu.

Kod oblaka, sistem plaćanja trebalo bi da je definisan zavisno od toga šta se i koliko troši, tzv. *pay-as-go* model. Bez obzira na to da li se cena definiše prema broju nekih resursa, njihovom tipu, kapacitetu i slično, važno je da korisnik ima jasan pregled troškova. Da bi to bilo moguće, dobavljač usluga mora uspostaviti pouzdan sistem obračunavanja, koji u realnom vremenu pruža uvid u to šta je koliko naplaćeno. U praksi, svi dobavljači usluga u oblaku obezbeđuju alate koji klijentima pomažu da predvide svoje troškove, odnosno da ih optimizuju.

Koncept računarstva u oblaku često se ilustruje primerom resursa koje trošimo u domaćinstvima: električnom energijom i vodom. Prosečan građanin ne mora da razmišlja o tome odakle tačno dolazi električna energija u stan – na njemu je da koristi svoje uređaje, potrošnja se meri i račun dostavlja na mesečnom nivou. Elektrane, razvodna i distributivna postrojenja, trafo-stanice, dalekovodi i tako dalje elementi su koji su vitalni za funkcionisanje električne mreže, ali krajnji korisnik ne mora da zna kako oni funkcionišu niti gde se tačno nalaze.

Sada je moguće sumirati različita svojstva računarstva u oblaku:

- U pitanju je veliki broj računarskih resursa koje koristi više korisnika.
- Pristupa se putem interneta i korisnik (klijent) ima jednostavan način da zahteva resurse i upravlja njima.
- Resursi su skalabilni: na jednostavan način je moguće povećati kapacitet resursa koji su u upotrebi.
- Potrošnja resursa se meri i naplaćuje prema potrošnji.

Formalnu definiciju računarstva u oblaku dao je institut NIST:

Računarstvo u oblaku je model koji omogućava sveprisutni, praktični i na zahtev dostupni mrežni pristup do zajedničkog skupa konfigurabilnih računarskih resursa (npr. mreže, serveri, skladišta, aplikacije i usluge) koji se mogu brzo obezbediti i pustiti u rad uz minimalan napor u upravljanju ili interakciju sa dobavljačem usluga (Mell & Grance, 2011).

Institut NIST je dao detalje o pet osnovnih svojstava računarstva u oblaku:

- Samostalno usluživanje na zahtev. Korisnik može jednostrano da obezbedi računarske resurse (vreme izvršavanja servera, mrežno skladište i dr.) prema potrebi, bez posebne interakcije sa dobavljačem. To znači da nije neophodno da korisnik šalje posebne zahteve, zove telefonom i slično da bi dobio, na primer, server ili dodatni disk. (Naravno, svaki dobavljač ima i korisničku podršku u slučaju pomoći, rešavanja posebnih zahteva i sl.)
- Širokopojasni mrežni pristup. Resursi su dostupni putem interneta i pristupa im se kroz standardne mehanizme, dostupne na raznovrsnim platformama (pametni telefoni, računari, tableti).
- Udruživanje resursa *(resource pooling).* Računarski resursi dobavljača resursa su udruženi kako bi služili većem broju korisnika u modelu više zakupaca *(multi-tenant)* sa različitim fizičkim i virtuelnim resursima koji se dinamički dodeljuju i oduzimaju prema zahtevu korisnika. Postoji osećaj nezavisnosti od lokacije, jer korisnik generalno nema kontrolu niti saznanje o tačnoj lokaciji pruženih resursa, ali može biti u mogućnosti da tačno izabere lokaciju na višem nivou apstrakcije (npr. zemlja, država ili data-centar).

- Rapidna elastičnost. Resursi se mogu elastično obezbediti i osloboditi, u nekim slučajevima automatski, kako bi se brzo skalirali prema spolja ili unutra, u skladu sa potražnjom. Korisniku deluje da su resursi neograničeni i da se mogu dobaviti u bilo kojoj količini i u bilo koje vreme.
- Merena usluga. Sistemi oblaka automatski kontrolišu i optimizuju upotrebu resursa koristeći mogućnost merenja na određenom nivou apstrakcije, prikladnom za tip usluge (npr. skladištenje, procesiranje, propusni opseg i aktivni korisnički nalozi). Upotreba resursa može se nadgledati, kontrolisati i izveštavati, pružajući transparentnost za dobavljača i korisnika usluge.

1.2. Kratka istorija računarstva u oblaku

Osvit računarske ere obeležio je koncept mejnfrejm računara, koji su bili veoma skupi, smešteni u posebne prostorije i kojima se pristupalo sa tzv. tankih klijenata, računara sa veoma skromnim resursima, sa kojih je korišćen glavni računar, tj. pomenuti mejnfrejm.

Početak 1980-ih obeležen je eksplozijom ličnih računara. Računari se pojavljuju u domaćinstvima, ali umrežavanje je problematično, veze su veoma spore i sve se faktički obavlja na samom pojedinačnom računaru.

U ranim 1990-im pojavljuje se koncept Grid računarstva. Iniciran je od strane National Labs, u SAD, prevashodno za potrebe nauke. Može se definisati kao sakupljanje računarskih resursa sa više lokacija kako bi se postigao zajednički cilj. Mreža se može smatrati distribuiranim sistemom sa neinteraktivnim radnim opterećenjima koja uključuju veliki broj fajlova. Računari se umrežavaju u lokalnoj mreži i mogu biti heterogeni. Za proširenje ovog koncepta tako da računari mogu bit i izvan institucije, u drugoj mreži, bilo je neophodno rešiti bezbednosne izazove (udruživanje računara iz različitih organizacija). Moderni Grid sistemi su distribuirani i decentralizovani. Druga važna razlika u odnosu na klaster je što je svaki nod (čvor) nezavisan.

Drugi koncept, poznat kao Utility computing, iniciran je od strane IT kompanija 2000-ih godina. Ovaj model je zapravo ono što se dalje razvilo kao računarstvo u oblaku: specijalizovane firme pružaju klijentima računarske resurse na zahtev i naplaćuju prema konkretnoj potrošnji.

Sa povećanjem kompleksnosti sistema i razvojem veštačke inteligencije (AI – *artificial intelligence*), počeo je razvoj autonomnih sistema. U pitanju su sistemi koji mogu samostalno da donose odluke vezane za konfiguraciju, popravku i zaštitu. Upravljanje resursima u oblaku je danas velikim delom automatizovano i samoregulišuće.

Jedan od važnih koraka u razvoju računarstva u oblaku jeste implementacija servisa za upravljanje klijentima (CRM – *client management system*), softvera koji pomaže u evidenciji komunikacije sa klijentima, prodaje itd. Kompanija salesforce.com je 1999. implementirala ovaj servis, koji se koristi putem *web-browsera*.

Sledeću prekretnicu u razvoju oblaka postavio je Amazon 2002. godine, kada je ova kompanija objavila svoj novi proizvod Amazon Web servis (eng. Amazon Web Services – AWS).

Godine 2006. Amazon objavljuje svoj novi komercijalni Web servis Elastic Compute Cloud (EC2), putem kojeg klijenti, kao što su manje organizacije i pojedinci, mogu da iznajmljuju virtuelne računare za potrebe instalacije i izvršavanja sopstvenih aplikacija.

U to vreme su, sa pojavom Web 2.0 tehnologije, i kompanije poput Google-a počele sa razvojem korporativnih aplikacija koje se izvršavaju i isporučuju korisnicima pomoću običnog pregledača - web browsera. Kompanija Google je 2008. godine lansirala proizvod Google App Engine beta.

Početkom 2008. godine agencija NASA (National Aeronautics and Space Administration) lansirala je platformu OpenNebula koja je postala prvi softver otvorenog koda za razvoj privatnih i hibridnih klaud okruženja i federacije oblaka.

Sredinom 2008. godine Gartner[1] je prepoznao da bi primena koncepta klaud računarstva mogla da transformiše odnos između korisnika računarskih resursa i usluga i onih koji te resurse prodaju.

U februaru 2010. godine kompanija Microsoft je lansirala platformu Microsoft Azure, dok su u julu iste godine kompanija Rackspace Hosting i NASA objavili klaud platformu otvorenog koda pod nazivom OpenStack projekat. Cilj projekta OpenStack je bio da se omogući kompanijama da isporučuju servise računarstva u oblaku korišćenjem standardne hardverske infrastrukture.

Kompanija IBM je 2011. godine objavila IBM SmartCloud okvir, da bi 2012. godine kompanija Oracle predstavila svoj Oracle Cloud.

Maja 2012. godine kompanija Google je premijerno prikazala Google Compute Engine, koji je postao dostupan klijentima širom sveta decembra 2013.

Evolucija računarstva u oblaku ide u korak sa metodologijama razvoja softvera i novim softverskim arhitekturama. Sa jedne strane, teži se razdvajanju hardvera i softvera, gde na scenu stupaju virtualizacija računara i mreže (softverski definisane mreže), a sa druge strane sam softver postaje modularan, čime se dobija na fleksibilnosti razvoja, održavanja i eksploatacije. Modularnost softvera se kod modernih aplikacija ostvaruje prevashodno kroz servisno orijentisanu arhitekturu (SOA). SOA podrazumeva da se funkcije softvera razbijaju na veći broj servisa koji se realizuju nezavisno. Na primer, u zdravstvenom informacionom sistemu mogu se realizovati posebni servisi za rad sa podacima (kartonima) pacijenata, zatim poseban servis za zakazivanje pregleda i poseban servis za naplaćivanje usluga. Ovakva modularizacija omogućava bolje skaliranje, jednostavnije održavanje, olakšano dodavanje novih funkcija, povezivanje sa drugim sistemima (format za razmenu podataka je standardizovan, npr. XML ili JSON).

1.3. Oblak i virtuelizacija

Virtuelizacija je jedan od osnovnih gradivnih elemenata računarstva u oblaku. Pomoću softvera realizuje različite sisteme koji se ponašaju kao da su nezavisni – na jednom istom fizičkom sistemu. Ona omogućava optimizaciju korišćenja fizičkih resursa od strane više logičkih sistema i realizaciju servisa računarstva u oblaku. Može se konstatovati da je oblak dodatni sloj apstrakcije iznad virtualizacije.

Korisnici putem interneta mogu koristiti usluge računarstva u oblaku bez specijalnog odobrenja pristupa virtuelnim mašinama, najčešće samo uz browser. Oblak je proširiv, dok virtuelna mašina ima ograničen kapacitet.

1.4. DevOps

Sa razvojem oblaka, u fokus je dospeo i pojam DevOps (Development – Operations). DevOps je kombinacija kulturnih filozofija, praksi i alata koja povećava sposobnost organizacije da isporučuje aplikacije i usluge velikom brzinom: razvijajući i unapređujući proizvode bržim tempom nego kod tradicionalnog pristupa. DevOps inženjer je osoba koja upravlja procesima automatizacije i pouzdane isporuke softvera uz pomoć odgovarajućih alata. Danas, DevOps alati su najčešće upravo alati računarstva u oblaku.

[1] https://www.gartner.com/en

Više informacija o pojmu DevOps-a može se naći kod (Grande, Vizcaíno, García, 2024) i na https://aws.amazon.com/devops/what-is-devops/

Pitanja za proveru znanja

1. Koje su to osnovne tri komponente servera?
2. U čemu je razlika između konvergirane i hiperkonvergirane infrastrukture?
3. Kako se tačno princip „economy of scale“ ogleda na primeru računarstva u oblaku?
4. Kako biste apsolutnom laiku objasnili pojam računarstva u oblaku?
5. Koje kompanije su odigrale značajnu ulogu u popularizaciji računarstva u oblaku?

2. Modeli računarstva u oblaku

Cilj poglavlja je diskusija o različitim taksonomijama modela računarstva u oblaku i upoznavanje sa elementima migracije.
Student će nakon savladanog poglavlja umeti da:

- Navede i objasni modele usluga računarstva u oblaku
- Navede prednosti i nedostatke različitih modela
- Objasni modele implementacije računarstva u oblaku
- Objasni tipove migracije
- Navede stavke AWS-ove strategije migracije 7R
- Proceni koja strategija migracije je odgovarajuća za određeni scenario

U diskusiji o konceptu računarstva u oblaku pomenut je scenario da korisnik razvija svoj softver koristeći resurse u oblaku, kao i opcija u kojoj korisnik od dobavljača dobija softver tako da se ne bavi razvojem već sve ima "na gotovo". Primeri ovakvog pristupa, gde se softver dobija kao usluga, može biti Google WorkSpace ili Microsoft-ov Office 365. Krajnji korisnik se pretplaćuje i dobija određeni paket usluga. Zatim koristi npr. Word onlajn, čuva podatke na OneCloud-u bez ikakve brige o samom softveru ili infrastrukturi na kojoj se softver izvršava. Za veliki broj aplikacija ovakav pristup je sasvim zadovoljavajući. U pitanju su softveri koji su namenski, tipski. Pomenuti primeri su tzv. office paketi, a u pitanju mogu biti i softveri druge namene, kao što je ZOOM, za video konferencije, ili Dropbox, za čuvanje podataka. Korisnik ili klijent usluga u oblaku u ovom slučaju je i krajnji korisnik.

Veoma često je neophodno razviti nov softver za specifičnu namenu i uz posebne zahteve naručioca i krajnjih korisnika. U tom slučaju prethodno pomenuta rešenja nisu odgovarajuća, te je neophodno krenuti u razvoj softvera. Dobavljač usluga računarstva u oblaku tada omogućava resurse na kojima će se razvijati i pokretati softver. Najjednostavniji primer je da se u oblaku kreira virtuelna mašina i da se na nju instalira određeni operativni sistem, te da takva mašina služi kao server na kojem se pokreće dati softver. U takvom scenariju, dobavljač usluga u oblaku bi obezbedio infrastrukturu (virtuelnu mašinu sa određenim svojstvima – brojem procesora, memorijom, skladištem), a razvojem bi se dalje bavio korisnik oblaka. Kada se softver razvije, instalira i pokrene, krajnji korisnici će dobiti mogućnost da mu pristupaju putem web-aplikacije, mobilne aplikacije i slično. Razvijena aplikacija može biti web-prodavnica, video-igra, program za knjigovodstvo, aplikacija za učenje stranih jezika ili bilo šta drugo.

2.1. Modeli usluga računarstva u oblaku

Između ove dve krajnosti, korišćenja gotovog softvera dobavljača usluga i razvoja svog softvera na infrastrukturi dobavljača. Postoji i treća mogućnost, a to je da se timu za razvoj softvera olakša posao tako što će se obezbediti pojedini elementi neophodni za razvoj i pokretanje softvera. U takvom scenariju dobavljač ne stavlja na raspolaganje "sirovu" infrastrukturu, već postavlja razvijenu aplikaciju u okruženje dobijeno od dobavljača, gde postoji podrška za dati programski jezik, bazu podataka i slično. Klijent ne instalira operativni sistem, niti bilo kakve biblioteke; na njemu je da postavi softver i eventualno konfiguriše okruženje u smislu biblioteka koje softver koristi.

Opisani načini korišćenja usluga dobavljača jesu tri modela usluga.

Softver kao usluga (Software as a Service – SaaS). Korisnik dobija pristup gotovoj aplikaciji i pristupa joj putem određenog interfejsa, najčešće preko a. Za korisnika su nevidljivi fizički ili virtuelni serveri na kojima se izvršava softver, kao i prateći sistemski softver, baza podataka, biblioteke za pokretanje i sl. Korisnici na zahtev dobijaju licence za aplikacije koje su im potrebne i koriste ih do onog vremenskog trenutka kada su im potrebne.

Prednosti SaaS modela jesu:

- Optimalna upotreba raspoloživih računarskih resursa;
- Izbegavanje troškova koji bi nastali kupovinom licenci za korišćenje aplikacija;
- Eliminacija vremena, utrošene energije i troškova koji bi nastali na osnovu implementacije i održavanja svih ovih resursa;
- Brz razvoj;
- Klijent ne treba da brine o upravljanju aplikacijama;
- Aplikacije su obično stabilne i pouzdane zato što podršku realizuje čitava infrastruktura i tim dobavljača usluge.

Nedostaci SaaS modela:

- Klijent nema kontrolu nad sistemom;
- Slaba kontrola nad razvojem, nadogradnjom i testiranjem aplikacija;
- Integracija sa drugim softverima na drugim sistemima je otežana;
- Dobavljač ima punu kontrolu nad podacima klijenta, osim u slučajevima kada se koriste tehnike kriptografije;
- Vezanost za dobavljača *(vendor lock-in).*

Platforma kao usluga (PaaS). Korisnik razvija, testira i pokreće softver koristeći okruženje dato od strane dobavljača usluga. Ne bavi se infrastrukturom. Za razliku od klasičnih računarskih platformi (hardver + operativni sistem), PaaS predstavlja povoljniji model za razvoj skalabilnih aplikacija.

Prednosti PaaS-a:

- Model PaaS je isplativiji od modela u kojem se iznajmljuje "sirova" infrastruktura zato što klijent iznajmljuje softversku platformu, ne hardverski resurs;
- Potpuna kontrola nad korisnicima koji pristupaju softveru i obrađuju podatke;
- Poboljšana i olakšana integracija sa drugim sistemima;
- Minimalni napor koji se ulaže u upravljanje virtuelnim mašinama zato što se tim pitanjima bavi isporučilac usluge;

Nedostaci PaaS:

- Nema kontrole nad platformom;
- Platforma se najverovatnije deli sa drugim klijentima, što može izazvati pad performansi;
- Klijent dosta vremena i energije ulaže u ažuriranje i nadogradnju aplikacija;
- Kriva učenja: programeri moraju da se prilagode alatima i servisima koje nudi PaaS platforma.

Infrastruktura kao usluga (IaaS). Korisnik dobija najčešće virtuelne resurse (za obradu, čuvanje podataka, za umrežavanje). On instalira sistem, konfiguriše mrežu, zaštitu, potrebne biblioteke. Ovaj model nudi najveću fleksibilnost. Provajderi IaaS usluga pored standardnih hardverskih resursa nude i

firewall opremu, komponente za balansiranje opterećenja mreže i obrade podataka (eng. load balancers), mrežne uređaje (eng. network appliances), kao i ostale računarske komponente.

Prednosti IaaS:

- Interoperabilnost i portabilnost
- Skalabilnost i fleksibilnost
- Potpuna kontrola nad virtuelnim mašinama
- Laka administracija virtuelnih mašina
- Efikasno i fleksibilno iznajmljivanje računarskih resursa

Potencijalni nedostaci IaaS:

- Najskuplji model iznajmljivanja resursa
- Klijent je odgovoran za čuvanje i pravljenje rezervnih kopija podataka
- Klijent ne zna gde je fizički lociran server
- Zavisnost od mreže dobavljača

Pored navedene kategorizacije, pojavio se i niz drugih, koji se u suštini svode na neku od varijanti SaaS. Na primer: DaaS *(Data as a Service)* – čuvanje podataka u oblaku, *Desktop as a Service* – aplikacije su u oblaku, pristupa se sa tankih klijenata. Npr. već pomenuti Zoom i slični alati za videokonferencije spadaju u *Communications as a Service*. Amazonova usluga RDS *(Relational Database Service)* podrazumeva da se baza podataka koristi kao usluga, bez zadiranja u instalaciju bilo kakve pojedinačne baze podataka, uz podršku za MySQL i PostgreSQL.

SECaaS (*Security as a Service* – bezbednost kao usluga) predstavlja koncept autsoursovanja sajber-bezbednosti kroz servise u oblaku. I u ovom slučaju, pogodnosti u odnosu na konvencionalne pristupe jesu u skalabilnosti, dostupnosti poslednjih zakrpa, uštedi itd. Neki od primera bezbednosnih delatnosti koje se mogu realizovati kroz SECaaS su:

- Prevencija gubitka podataka, kroz upotrebu alata za praćenje i zaštitu podataka – onih u upotrebi i onih koji su arhivirani.
- Oporavak u slučaju katastrofa.
- Upravljanje autentifikacijom i autorizacijom: ko ima pravo pristupa mreži i sa kakvim tačno privilegijama.
- Zaštitna barijera (fajervol) kao usluga: štiti saobraćaj barijerom naredne generacije, koja radi na aplikativnom sloju.
- Bezbednost e-pošte: Filtrira se spam i pokušaji fišinga, kao i drugi malicioznoi sadržaji u e-pošti.
- Detekcija i prevencija upada: prati se aktivnost u mreži sa ciljem ustanovljavanja neregularnog saobraćaja.

2.2. Modeli implementacije

U zavisnosti od toga kako je oblak koncipiran po pitanju vlasništva, lokacije i pristupa razlikuju se modeli implementacije *(deployment models):* javni, privatni oblak, oblak zajednice i hibridni oblak.

Javni oblak. Resursi u oblaku mogu se koristiti od strane javnosti. U vlasništvu je akademske, vladine ili komercijalne organizacije, a sama oprema se nalazi kod dobavljača usluga. U pojedinim slučajevima,

usluge javnog oblaka mogu da se, do određenog nivoa korišćenja, ponude besplatno, dok se u većini slučajeva korišćenje naplaćuje na osnovu nivoa ostvarene potrošnje. Infrastruktura javnog oblaka se sastoji iz obimnog hardvera koji je distribuiran na nekoliko različitih lokacija širom jedne zemlje ili širom sveta. Javni oblak je visoko efikasan u smislu troškova, zato što kompanije plaćaju samo za one računarske resurse koje su potrošile, dok sa druge strane kompanije dobijaju pristup sofisticiranoj i naprednoj računarskoj opremi koju održava i kontroliše dobavljač klaud usluga. Prednosti javnog oblaka:

- Veća skalabilnost računarskih resursa
- Troškovna efikasnost
- Pouzdanost
- Usluge poput SaaS, PaaS i SaaS su široko dostupne na platformama javnog oblaka i može da im se pristupi sa bilo koje lokacije u svetu

Nedostaci javnog oblaka:

- Slaba kontrola bezbednosti/privatnosti
- Nemogućnost korišćenja za potrebe osetljivih aplikacija i podataka (npr. baza podataka banke sa računima klijenata)
- Nedostatak u vidu potpune fleksibilnosti zato što platforma zavisi od CP
- Nepostojanje striktnih protokola za upravljanje podacima.

Privatni oblak. Infrastruktura oblaka se koristi isključivo od strane jedne organizacije. Svaki sektor organizacije može imati definisan pristup za svoje potrebe (na primer, računovodstvo, tehnička podrška, prodaja). Najčešće je ovakav oblak smešten u okviru prostorija same organizacije i upravljanje vrši sama organizacija. Šta to znači u praksi? Na primer, banka ima potrebu da zbog bezbednosti podataka i usklađenosti sa određenim zakonima i standardima čuva podatke na svojoj infrastrukturi. Formira se data-centar sa određenim brojem fizičkih servera, instalira odgovarajući sistemski softver za virtualizaciju i upravljanje, postavlja korisnički softver i definišu prava pristupa. Ovakav data-centar jeste podloga za usluge u oblaku na jedan prilagođen način. I dalje se pristupa putem mreže, jedan deo oblaka može biti otvoren za pristup (npr. klijentima za e-banking), dok su drugi delovi strogo interni. Kod privatnog oblaka, organizacija je faktički i dobavljač usluga i korisnik, tako da se plaćanje usluga ne sprovodi. Ipak, merenje servisa je i dalje na snazi: za svaki sektor organizacije se generiše izveštaj o potrošnji resursa, koji pomaže u planiranju resursa i optimizaciji.

Prednosti privatnog oblaka:

- Unapređenje bezbednosti
- Povećan nivo pouzdanosti
- Poboljšanje performansi
- Povećana fleksibilnost
- Potpuna kontrola

Nedostaci privatnog oblaka:

- Visoki početni troškovi i visoki ukupni troškovi održavanja
- Neiskorišćenost resursa
- Otežano skaliranje računarske platforme

Zajednički oblak *(community cloud)*. Infrastruktura oblaka se koristi od strane zajednice (zajednica obuhvata više organizacija). Oblakom upravlja jedna od organizacija iz zajednice ili posebna

organizacija. Na primer, ako bi nekoliko zdravstvenih organizacija (bolnica, instituta) trebalo da bezbedno dele informacije o pacijentima, mogao bi se formirati jedan zajednički oblak.

Hibridni oblak. Kombinuje svojstva pomenutih oblaka. Primer bi bio kombinacija privatnog i javnog oblaka, gde se osnovne operacije izvršavaju na resursima privatnog oblaka, a javni oblak služi za bekap ili za situacije preopterećenja.

Kada se bira javni oblak?

- Kada se razvija i testira jedna izolovana aplikacija
- Kada je brzo potrebno izvršiti nadogradnju i proširivanje resursa privatnog oblaka
- Kada je potrebno brzo i na zahtev pribaviti računarske resurse za jednokratnu upotrebu
- Kada organizacija hoće da uspostavi kontrolisanu potrošnju računarskih resursa na osnovu nivoa utrošenih jedinica resursa.

Kada se bira privatni oblak?

- Kada je potrebno da se upravlja životnim ciklusom više aplikacija
- U slučaju implementacije platforme za e-trgovinu, kada organizacija čuva osetljive podatke o svojim klijentima, kao što su brojevi kreditnih kartica
- Kada postoje permanentni zahtevi za „stabilnim“ računarskim resursima
- Kada postoje zahtevi za „trajnim“ računarskim resursima koje aplikacije u klaudu koriste i
- Kada postoje zahtevi u pogledu raspoloživosti računarskih resursa u različitim poslovnim jedinicama koje su međusobno izolovane.

Ovde vredi pomenuti da kompromis između privatnog i javnog oblaka leži u tzv. virtuelnom privatnom oblaku (VPC). Kao što se virtuelna privatna mreža kroz javnu mrežu (internet) formira uz pomoć VPN usluge, tako je moguće i privatni oblak „simulirati“ kroz javni oblak. Više reči o VPC-u biće u kasnijim poglavljima.

2.3. Migracija

Mnoge aplikacije nisu kreirane za izvršavanje u oblaku, već u konvencionalnom računarskom okruženju, na serverima organizacije (fizičkim ili virtuelnim) ili dobavljača usluga hostinga. Ukoliko je potrebno preneti takve aplikacije u okruženje oblaka (zbog boljeg skaliranja, smanjenja troškova i dr.), neophodno je izvršiti migraciju. Migraciju je neophodno izvršiti i u suprotnom smeru, npr. ukoliko izvršavanje na virtuelnoj mašini u oblaku nije dovoljno efikasno, može se pribeći migraciji na fizičku mašinu, sa hardverom koji je ekskluzivno dodeljen datoj aplikaciji.

U nastavku će se prvo diskutovati o četiri tipa migracije koje uključuju virtuelne i fizičke mašine. Zatim će biti više reči o migraciji usluga.

Kombinovanjem fizičke i virtuelne mašine nastaju 4 opcije za migraciju sa mašine na mašinu: fizička na virtuelnu, virtuelna na virtuelnu, fizička na fizičku i virtuelna na fizičku mašinu.

Migracija fizička – virtuelna mašina

Scenario za ovu migraciju je sledeći: organizacija želi da optimizuje troškove i eliminiše fizičke mašine ili smanji njihov broj. Fizičke mašine će migrirati u virtuelne i takve mašine pokrenuti u oblaku. (Naravno, nije neophodno da ovaj scenario uključuje oblak; organizacija može i dalje imati svoj data-centar u kojem će čuvati i pokretati ove virtuelne mašine na konvencionalnom serveru sa instaliranim virtualizatorom.) Prvi, neautomatizovani oblik migracije je kada se na virtuelnu mašinu u oblaku

instalira sveža instanca operativnog sistema, na koju se zatim pomoću specijalnih alata migriraju aplikacije i podaci iz lokalnog na okruženje oblaka.

Drugi, automatizovani način, odnosi se na korišćenje specijalnih softvera i alata za konverziju cele fizičke serverske platforme (operativni sistem plus aplikacije) u virtuelnu mašinu. Mnogi dobavljači usluga oblaka nude ovakve alate i kao primeri navode se *VMware vCenter Converter* i *Microsoft Virtual Machine Manager*. Na internetu se može pronaći još alata, npr. Disk2VHD. Koji alat će se koristiti zavisi prevashodno od toga koji je ciljni virtuelizator (VmWare, HyperV, ProxMox) i koji format podržava.

Migracija virtuelna – virtuelna mašina

Postupak migracije sa jedne na drugu virtuelnu mašinu mnogo je jednostavniji od prethodnog tipa migracije jer se u većini slučajeva svodi na kloniranje postojeće virtuelne mašine u lokalnom okruženju i uvozu slike klonirane virtuelne mašine u virtuelizovano okruženje dobavljača usluga oblaka. Potrebno je uzeti u obzir vrstu hipervizora pod kojim se izvršava virtuelna mašina u lokalnom okruženju organizacije. U retkim slučajevima može se desiti da format lokalne virtuelne mašine nije kompatibilan sa formatom koji je podržan od strane virtuelizatora dobavljača.

Migracija fizička – fizička mašina

Koristi se kada organizacija hoće da izvršava svoje servise na namenskom fizičkom server oblaka, koji se ne deli sa drugim klijentima, ili kada određene poslovne aplikacije ne mogu da se izvršavaju na virtuelnom hardveru. Treba obezbediti da fizički server u oblaku bude istih performansi kao i lokalni fizički server.

Migracija virtuelna – fizička mašina

Ova migracija je najzahtevniji tip migracije i pre njenog sprovođenja potrebno je detaljno proučiti hardverske i softverske zahteve operativnog sistema, aplikacija i podataka koji se migriraju.

O virtualizaciji i virtualnim mašinama, koje predstavljaju osnovni gradivni element računarstva u oblaku, biće više reči u narednom poglavlju. Ovde je ipak važno napomenuti da računarstvo u oblaku nije isto što i virtualizacija, odnosno da se izgradnja infrastrukture u oblaku ne svodi (samo) na korišćenje virtuelnih mašina i da su do sada prikazani režimi migracije samo jedan element asortimana scenarija za migraciju, na nivou infrastrukture kao usluge (time i čitavih mašina, bilo fizičkih, bilo virtuelnih). Na drugoj strani, paradigma serverless-a upravo usmerava ka tome da se mašine uopšte ne koriste neposredno, tj. da su nevidljive za korisnika, već da se usluge virtuelizuju. To bi na primer značilo da korisnik ne vidi virtuelnu mašinu na kojoj se izvršava neka funkcija, već da samo definiše funkciju i pozove je. Naravno, „ispod haube“ će se inicirati virtuelna mašina na kojoj se izvršava data funkcija, ali korisnik neće imati nikakvog dodira sa njom.

Stepen migracije varira od najblažeg oblilka, koji je poznat kao *lift&shift*, gde se tehnologije zadržavaju i postupak se svodi na migraciju na virtuelnu mašinu dobavljača; do potpunog refaktorisanja i prelaska na serverless. Lift&Shift zahteva najmanje promena i prilagođavanja, dok prelazak na izvorne tehologije u oblaku zahteva puno posla. Taksonomija migracije varira od dobavljača do dobavljača. U nastavku će biti prikazan Amazonov (AWS) pristup, tzv. AWS 7R.

1. *Refactor/re-architect* – Aplikacija se premešta i menja joj se arhitektura, tako da u potpunosti koristi prednosti izvornih opcija oblaka, radi unapređenja agilnosti, performansi i skalabilnosti. Obično uključuje portovanje operativnog sistema i baze. Primer: migracija on-premises Oracle baze na Amazon Aurora PostgreSQL - compatible edition.

2. *Replatform (lift and reshape)* – Aplikacija se premešta na oblak i uvode novi nivoi optimizacije radi iskorišćenja mogućnosti oblaka. Primer: On-premises Oracle baza se migrira na Amazon Relational Database Service (Amazon RDS) for Oracle.

3. *Repurchase (drop and shop)* – Prelazak na drugi proizvod, obično prelaskom sa tradicionalnog modela licenci na SaaS model. Primer: migracija CRM (customer relationship management-a) na salesforce.com.

4. *Rehost (lift and shift)* – Aplikacija se premešta u oblak bez ikakvih izmena u smeru iskorišćenja mogućnosti oblaka. Npr. migrira se on-premises Oracle baza na instancu EC2 AWS oblaka.

5. *Relocate (**hypervisor-level** lift and shift)* – Infrastruktura se premešta u oblak, bez kupovine novog hardvera, ponovnog pisanja aplikacije ili izmena postojećih operacija. Konkretan scenario vezan je za VMware hipervizora na on-premises mašinama, koji se premešta na AWS. (I dalje su u pitanju vmware mašine, koje sada rade na VMware virtualizatoru AWS-a).

6. *Retain (revisit)* – Aplikacije se zadržavaju u izvornom okruženju jer zahtevaju obimno refaktorisanje i posao se odlaže do daljeg. Ove aplikacije „završavaju posao“, a ne isplati se migrirati ih.

7. *Retire* – Napuštaju se i uklanjaju aplikacije iz izvornog okruženja.

Pitanja za proveru i obnavljanje znanja

1. Pod kojim okolnostima biste se opredelili za model usluge IaaS?
2. Koje su prednosti modela usluga SaaS?
3. U čemu su nedostaci javnog oblaka?
4. Kako se obračunava plaćanje usluga kod privatnog oblaka?
5. Koji vid migracije je najzahtevniji? Zašto?
6. Potrebno je ubrzati aplikaciju i pojednostaviti upravljanje bazom podataka. Koji pristup (ili pristupe) prema AWS 7R taksonomiji izabrati? Obrazložite.

3. Infrastruktura oblaka

Ciljevi poglavlja su upoznavanje sa osnovama infrastrukture računarstva u oblaku, konceptom i realizacijom data centara, uz poseban osvrt na AWS-ovu infrastrukturu.
Student će po savladanom poglavlju moći da:

- opiše osnovne elemente data-centra
- objasni kako se realizuje pouzdanost data-centra
- diskutuje o elementima realizacije računarske infrastrukture
- diskutuje o elementima fizičke infrastrukture vezane za napajanje i hlađenje
- objasni AWS koncepte infrastrukture
- izvrši izbor konfiguracije instance kod AWS-a

Za klijente i krajnje korisnike, oblak je „tamo negde“ i suština usluga u oblaku upravo jeste da se klijenti što manje bave infrastrukturom, da ne brinu o lokaciji, količini resursa i povećanju zahteva, te da se fokusiraju na aplikacije i servise koje razvijaju. Sa druge strane, tehnologije koje omogućavaju ovakav način rada su konkretne, materijalizovane u posebnim data-centrima i iziskuju razvoj i inovacije kako bi se odgovorilo povećanim zahtevima za performansama, funkcijama i pouzdanošću. Oblak je itekako opipljiv i za razumevanje celokupne slike vezane za računarstvo u oblaku, neophodno je zaviriti iza korisničkog interfejsa i upoznati tehnologije koje se kriju „ispod haube“. U prethodnim poglavljima opisane su varijante virtualizacije, koncepta koji leži u srži računarstva u oblaku. U predstojećim poglavljima elementi oblaka biće predstavljeni sa obe strane: sa strane data-centra i nosećih tehnologija i sa strane korisnika usluga u oblaku. Na taj način će se zaokružiti priča o složenom fenomenu računarstva u oblaku.

U uvodnom poglavlju udžbenika dat je razvoj računarskih tehnologija, koji je doveo do konvergirane i hiperkonvergirane infrastrukture. U nastavku se dalje elaborira o organizaciji samog hardvera i pratećim tehnologijama.

3.1. Data-centar oblaka

Data centar oblaka predstavlja infrastrukturnu osnovu oblaka. U pitanju je skup velikog broja povezanih servera koji su organizovani unutar posebno dizajniranog objekta, sa obezbeđenim redundantnim napajanjem i vezom na internet, naprednim sistemima za dojavu i gašenje požara, fizičko-tehničkim obezbeđenjem itd. Pojam koji se može sresti u literaturi je WSC – *warehouse-scale computer.* Veliki dobavljači usluga u oblaku (npr. Google, Amazon, Microsoft) imaju čitave data-centre posvećene WSC-u. Dakle, sva oprema je angažovana upravo za usluge u oblaku. Svakako jedan od vitalnih elemenata data-centra jeste osoblje, koje je kompetentno za različite zadatke, od nivoa fizičkog obezbeđenja, preko tehničara do inženjera.

Slika 2 Državni data-centar (Kragujevac, Srbija)

Postoje i slučajevi kada je deo opreme u data-centru određen za WSC, a drugi deo opreme služi drugoj svrsi. Na primer, to može biti telehausing, čuvanje konvencionalnih servera itd. Radi poboljšanja performansi i smanjenja kašnjenja, dobavljači se mogu odlučiti da iznajme krilo postojećeg data-centra i u njemu instaliraju opremu, a bez izgradnje čitavog posvećenog data-centra. Kompanija Oracle je 2023. godine pokrenula svoj region u okviru državnog data-centra u Kragujevcu (Slika 2). Kada koriste usluge oblaka Oracle *(OCI – Oracle Cloud Infrastructure),* klijenti mogu izabrati region Oracle Cloud Jovanovac Region (oznaka: eu-jovanovac-1) i to znači da su date usluge pokrenute na Oracle serverima lociranim u ovom regionu.

Planiranje izgradnje data-centra je ozbiljan poduhvat jer je neophodno ne samo izgraditi objekat, već i obezbediti redundantno napajanje električnom energijom, redundantnu vezu na internet, zatim sama lokacija za gradnju mora biti udaljena od vodenih tokova, na području koje nije trusno ili sklono klizištima, a sa druge strane mora postojati i putna infrastruktura kako bi se dopremio materijal za gradnju, a kasnije, kako bi i sam centar bio dostupan za potrebe održavanja i posete.

U data-centru serveri su organizovani u rack-ormanima. U jednom ormanu se obično nalazi više desetina servera: ormani su standardno visine 40 U. („U“ je jedinica – *unit* za smeštanje u orman, servere, ali i skladišta. U nekim slučajevima serveri mogu zauzimati i više jedinica, npr. 3U. Visina jedinice je 1.75 inča (44,45 mm)).

Više ormana može biti organizovano u klastere, što znači da dele mrežu, hlađenje, napajanje.

Pouzdanost data centra oblaka

Prenošenje težišta računarstva u oblak pojačava akcenat na pouzdanosti same opreme i kompleksa data-centra. Dostupnost se uopšteno izražava kao odnos između vremena koje je sistem proveo u funkcionalnom stanju – vremenu rada, u kojem nesmetano izvršava usluge i ukupnog vremena (dostupnosti plus prekida rada, tzv. downtime-a).

$$dostupnost = \frac{vreme\ rada}{vreme\ rada + vreme\ prekida}$$

Treba napomenuti da prekidi u radu nisu uvek prouzrokovani iznenadnim i nepredviđenim događajima, već se može raditi i o vremenu održavanja (npr. zamene hardvera ili premeštanja servera).

Redundantnost je način da se poboljša dostupnost sistema. Ona podrazumeva postojanje dodatnih angažovanih ili rezervnih kopija sistema, koje pomažu u tome da celokupan sistem radi. U tom smislu mogu se definisati različiti vidovi redundantnosti: N+1, 2N, 2N+1.

Oznaka N predstavlja kapacitet koji je aktivan kada sistem funkcioniše. Redundantnost N+1 podrazumeva postojanje jedne rezervne komponente koja se aktivira u slučaju otkaza. Npr. ako se koristi 10 UPS (Uninterruptible Power Supply) uređaja, koji su aktivni i spremni da premoste opterećenje usled nestanka električne energije, na raspolaganju je 11 UPS uređaja; jedan UPS može da se iskoristi kao rezervni, da zameni eventualno pokvaren uređaj.

Redundantnost 2N podrazumeva da svaki element poseduje svoju rezervu. Dakle, 10 UPS uređaja radi, a još 10 je na raspolaganju. Na kraju, redundantnost 2N+1 podrazumeva kombinaciju prethodna dva pristupa. U primeru sa UPS uređajima: aktivno je 10 uređaja, a instalirano ukupno 21.

Oblici održavanja u data-centru

Održavanje je jedna od ključnih aktivnost koja obezbeđuje nesmetan i neprekidan rad čitavog data-centra. Izvodi se kombinacijom automatizovanih radnji i aktivnosti osoblja.

Preventivno održavanje podrazumeva planirano održavanje usmereno ka predupređivanju kvarova i otkaza. U tom smislu ono uključuje periodične provere opreme, prema preporukama proizvođača, i zamenu dotrajalih delova. Svako planirano održavanje – kreiranje mesečnog bekapa, zamena diska pri kraju garancije i sl, vodi se kao zakazano održavanje.

Prediktivno održavanje podrazumeva kontinualno praćenje stanje opreme, kao i periodične provere, sa idejom obavljanja održavanja u zakazanom periodu vremena kada je to najpovoljnije.

NCPI

NCPI (Network-Critical Physical Infrastructure) predstavlja sloj ispod fizičkog. Podrazumeva sve ono što je potrebno računarskim sistemima da rade uz visoku pouzdanost i bezbednost u data centru. Može se izdvojiti sedam ključnih elemenata NCPI-a:

1. Napajanje. Električna energija se dovodi sa najmanje dve strane (ukoliko u zemlji postoje dva dobavljača, onda bi trebalo da dva linka budu od posebnih dobavljača). Postoji rezervno napajanje (agregat) na koje se prelazi ukoliko oba dovoda električne energije otkažu. Pomoću UPS sistema sistemi se održavaju u funkciji, dok se ne izvrši prelaz na rezervno napajanje.
2. Hlađenje i kontrola okruženja. Sistemi za hlađenje i održavanje nivoa vlažnosti vazduha se koriste u cilju održavanja dugotrajnosti opreme[2].
3. Fizička struktura. Podrazumeva rack-ormane za opremu, spuštene plafone i izdignute podove koji omogućavaju ventilaciju i prohodnost za instalacije.
4. Bezbednost. Podrazumeva sisteme video-nadzora, senzore pokreta, alarme, kontrolu fizičkog pristupa, kao i zaštitu od požara.
5. Praćenje. Monitoring svih angažovanih sistema omogućava pravovremenu reakciju u slučaju pojave anomalija.

[2] HVAC (Heating, Ventilation, Air Conditioning) podrazumeva kompletno rešenje koje se angažuje sa datim ciljevima održavanja parametara okruženja.

Izgradnja data-centra i obezbeđivanje elemenata NCPI definisani su međunarodnim standardima. Konkretno, standard ISO/IEC 22237 definiše zahteve i preporuke vezane za aspekte gradnje, distribucije električne energije, upravljanje okruženjem i bezbednosnim sistemima.

3.2. Računarska infrastruktura

Osnova računarstva u oblaku je virtualizacija. Virtuelne mašine su kod različitih dobavljača poznate pod različitim imenima: virtuelni serveri, virtuelne instance ili samo – instance. Virtuelne mašine se nude pod različitim predefinisanim svojstvima, koja uključuju broj jezgara i količinu memorije.

Za specifične zahteve u ponudi su i *„custom“* konfiguracije, gde korisnik može odabrati svojstva svoje instance. *Multi-tenant* instance se izvode na fizičkom hardveru zajedno sa instancama drugih korisnika. *Single-tenant* instance – virtuelne mašine jednog korisnika izvode se na posebnom, posvećenom hardveru.

Iako je virtualizacija tehnologija prvog reda i daleko najveći broj softvera se izvršava u nekoj varijanti virtualizacije, bilo na posvećenim instancama, bilo u sklopu kontejnera, postoje situacije kada performanse virtuelnih mašina ne mogu da ispune očekivanja aplikacija. Ovakve situacije javljaju se:

- Ako postoji potreba za HPC (high performance computing-om) koji povlači veoma intenzivne I/O (Input-output) operacije, koje virtuelni uređaji ne mogu da isprate;
- Kada je u pitanju korišćenje obrade na grafičkim karticama, npr. za mašinsko učenje ili blokčejn;
- Ukoliko mora da se omogući visok nivo upravljanja bezbednošću.

Tada je neophodno obezbediti fizički, posvećen računar. (Ne nude svi dobavljači ovu opciju.)

Skladište *(storage)*

Skladišta se kod PC računara organizuju generalno na jednostavan način, neposrednim povezivanjem skladišnih jedinica (diskova) na matičnu ploču računara. Ovakav pristup je poznat kao direktno povezano skladište i može se koristiti i u oblaku. Međutim, tesna povezanost sa samim serverom znači da isključenjem servera skladište ne može dalje da čuva podatke.

Drugi oblici skladišta, koji su robusniji i fleksibilniji za korišćenje su:

- Fajl-skladište. Dostupno je kroz mrežu – Ethernet (koristi NFS protokol). Više servera može deliti jedno ovakvo skladište. Nedostatak je sporiji pristup, tako da je pogodan kao repozitorijum fajlova i za deljenje dokumenata.
- Blok-skladište. Podaci se prenose kroz brzu optičku mrežu, čime se omogućavaju brze performanse.
- Objektno skladište je apstraktan vid, koji nije definisan fizičkim povezivanjem. Pristupa mu se putem API-ja. Spor je, povoljan, neograničenog kapaciteta i može se deliti. Pogodan je za čuvanje raznovrsnih podataka (programa, bekapa, IoT podataka, logova, slika virtualnih mašina). Ova skladišta su poznata i kao „kofe“ (bucket, kod AWS-a je S2 bucket). Pogodna su za nestrukturirane, statične podatke, koje prate metapodaci.

U narednom poglavlju biće više reči o skladištima.

3.3. Infrastruktura AWS-a

Kao što je već rečeno, Amazon Web Services (AWS) je jedan od vodećih globalnih provajdera *cloud* usluga, koji organizacijama i pojedincima omogućava da hostuju aplikacije, čuvaju podatke i koriste napredne servise bez potrebe za sopstvenom infrastrukturom. AWS nudi širok spektar servisa, uključujući računarske resurse, baze podataka, analitiku, mrežne servise i veštačku inteligenciju.

Jedan od ključnih faktora koji AWS čini pouzdanim i fleksibilnim jeste njegova infrastruktura, koja je izgrađena na globalno distribuiranoj mreži data centara. Ova infrastruktura omogućava korisnicima da biraju gde će hostovati svoje resurse, uz visoku dostupnost, skalabilnost i sigurnost.

U ovom poglavlju upoznaćemo se sa osnovnim pojmovima AWS infrastrukture, kao što su regioni, zone dostupnosti *(Availability Zones)* i osnove konfigurisanja virtuelnih mašina (EC2 instanci).

Region (Regioni)

Region u AWS-u predstavlja fizičku lokaciju širom sveta gde se nalaze data centri. Svaki region je nezavisan entitet i sadrži više zona dostupnosti (*Availability Zones* – AZ-s) koje omogućavaju redundanciju i visoku dostupnost servisa. AWS trenutno nudi 36 regiona globalno, a svaki region je projektovan da zadovolji specifične zahteve korisnika u pogledu performansi, latencije i usklađenosti sa regulativama.

Faktori koji utiču na izbor regiona:

- **Blizina korisnicima:** Region koji je geografski bliži korisnicima pruža nižu latenciju i brži odziv aplikacija.
- **Regulative i pravni zahtevi:** Neki regioni su dizajnirani da zadovolje specifične zakonske propise, kao što su GDPR u Evropi ili CCPA u SAD-u.
- **Cena servisa:** AWS resursi mogu imati različite cene u zavisnosti od regiona, zbog čega je važno analizirati troškove pre izbora.

AWS ima više regiona širom sveta (Slika 3), a neki od najpopularnijih su:

- **US East (N. Virginia)** – **us-east-1:** Najkorišćeniji region zbog široke podrške servisa i nižih cena.
- **US West (Oregon)** – **us-west-2:** Pogodan za korisnike sa zapadne obale SAD-a.
- **Europe (Frankfurt)** – **eu-central-1:** Ključan za korisnike u Evropi zbog usklađenosti sa GDPR regulativama.
- **Asia Pacific (Tokyo)** – **ap-northeast-1:** Pogodan za kompanije sa tržištem u Aziji.
- **South America (São Paulo)** – **sa-east-1:** Jedini AWS region u Južnoj Americi, omogućava nisku latenciju za korisnike u tom regionu.

Slika 3 Mapa AWS regiona

Zone dostupnosti

Zone dostupnosti predstavljaju zasebne data-centre unutar jednog AWS regiona (Slika 4). Svaki region ima najmanje dve zone dostupnosti, a neki regioni imaju i do šest AZ. Ključna prednost zona dostupnosti je njihova fizička razdvojenost, što omogućava visok nivo redundancije i otpornosti na kvarove.

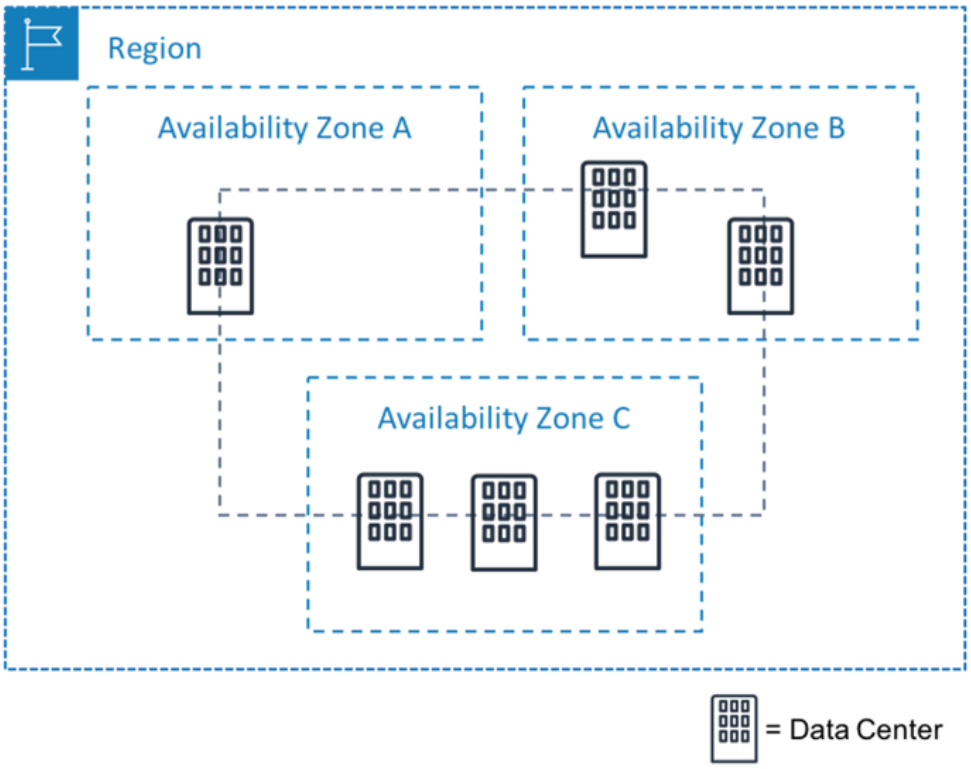
AWS preporučuje korisnicima da distribuiraju svoje aplikacije na više dostupnih zona kako bi se izbegli problemi izazvani kvarovima infrastrukture.

Razlike između regiona i zona dostupnosti

- **Region:** Fizička lokacija koja obuhvata više zona dostupnosti.
- **Zona dostupnosti:** Samostalan data centar unutar regiona, povezan brzim mrežnim linkovima sa ostalim zonama u istom regionu.
- Regioni su izolovani međusobno, dok zone dostupnosti unutar jednog regiona mogu međusobno komunicirati uz pomoć mreže niskog kašnjenja.

Da bi se osigurala otpornost aplikacija, preporučuje se korišćenje više zona dostupnosti u okviru istog regiona. Neki primeri implementacije su:

- **Raspodela instanci u više zona:** EC2 instance mogu biti distribuirane između više AZ kako bi se izbegla nedostupnost usluga u slučaju pada jedne AZ.
- **Baze podataka sa višezonskom replikacijom:** Servisi poput Amazon RDS omogućavaju replikaciju podataka u više dostupnih zona.
- **Load Balancer između zona:** AWS Elastic Load Balancer (ELB) može distribuirati saobraćaj između instanci u više dostupnih zona.



Slika 4 Regioni i zone dostupnosti

AWS Edge lokacije i Point of Presence (PoP)

AWS Edge lokacije i **Point of Presence (PoP)** čine globalnu mrežu krajnjih tačaka koje omogućavaju bržu i sigurniju isporuku podataka krajnjim korisnicima.

Edge lokacije su distribuirani serverski čvorovi koji služe za keširanje podataka i optimizaciju performansi servisa poput Amazon CloudFront-a. Ove lokacije smanjuju kašnjenje time što omogućavaju korisnicima pristup podacima iz najbližeg mogućeg serverskog čvora, umesto da zahtevi putuju do primarnog AWS regiona.

PoP (Point of Presence) su mrežne lokacije koje omogućavaju AWS-u da poveže svoju infrastrukturu sa globalnim internet provajderima i telekomunikacionim mrežama. PoP tačke se koriste za ubrzanje isporuke podataka i poboljšanje mrežne povezanosti za krajnje korisnike. Trenutno AWS raspolaže sa preko 700 ovakvih lokacija.

AWS Edge lokacije i PoP poboljšavaju performanse na sledeće načine:

- **Smanjuju kašnjenje:** Korisnici dobijaju podatke sa najbliže Edge lokacije, čime se minimizuje vreme odziva.
- **Optimizuju isporuku sadržaja:** AWS CloudFront koristi Edge lokacije za bržu distribuciju statičkog i dinamičkog sadržaja.
- **Povećavaju sigurnost:** Distribuirani model omogućava bolju zaštitu od DDoS napada i povećava otpornost mreže.

AWS Edge lokacije i PoP su ključni elementi za optimizaciju performansi aplikacija, posebno za globalne servise i streaming platforme.

3.4. Virtuelne mašine i konfiguracije (EC2 instance)

Kao što je već pomenuto, AWS EC2 *(Elastic Compute Cloud)* je web servis koji omogućava pravljenje i upravljanje virtuelnim mašinama (instancama) u AWS oblaku. Postoji nekoliko osnovnih tipova EC2 instanci:

- ***General Purpose*:** Balansiran odnos između CPU-a, memorije i mrežnih performansi. Pogodne za širok spektar aplikacija, uključujući web servere, aplikacije i male baze podataka.
- ***Compute Optimized:*** Dizajnirane za aplikacije koje zahtevaju visok nivo procesorske snage, kao što su naučne simulacije, video enkodiranje i serveri za igre.
- ***Memory Optimized:*** Namenjene aplikacijama koje zahtevaju veliki kapacitet memorije, kao što su baze podataka u memoriji (Redis, Memcached) i analitičke aplikacije.
- ***Storage Optimized:*** Pružaju visoke performanse skladišta podataka i optimizovane su za aplikacije koje obrađuju velike količine podataka, kao što su analitika i NoSQL baze podataka.
- **GPU:** Koriste grafičke procesore za aplikacije koje zahtevaju paralelno procesiranje, uključujući mašinsko učenje, veštačku inteligenciju, rendering i naučne simulacije.

AWS EC2 instance koriste različite arhitekture procesora, uključujući:

- **Intel Xeon:** Pogodan za aplikacije koje zahtevaju visoke performanse i kompatibilnost sa širokim spektrom softvera.
- **AMD EPYC:** Nudi konkurentne performanse uz često niže troškove u odnosu na Intel alternative.
- **AWS Graviton (ARM arhitektura):** Dizajniran od strane AWS-a za optimizaciju troškova i efikasnosti, posebno za aplikacije koje ne zahtevaju specifične x86 instrukcije.

Na osnovu zadatka koji treba da se obradi može se izvršiti selekcija procesora:

- **Intel Xeon:** Idealan za enterprise aplikacije, baze podataka i virtualizaciju.
- **AMD EPYC:** Pogodan za general-purpose workloads i aplikacije koje traže bolji odnos cena-performanse.
- **AWS Graviton:** Odličan za skalabilne cloud aplikacije, serverless computing i kontejnerizovane aplikacije zbog nižih troškova i energetske efikasnosti.

Pri izboru količine RAM-a i vCPU-a treba obratiti pažnju na samo opterećenje (workload) i spram njega odabrati odgovarajuću konfiguraciju. Na primer:

- **Malo opterećenje (1-2 vCPU, 2-4GB RAM-a):** Pogodni za male web aplikacije i testiranje.
- **Srednje opterećenje (4-8 vCPU, 8-16GB RAM-a):** Pogodni za baze podataka srednje veličine i analitičke aplikacije.
- **Veliko opterećenje (16+ vCPU, 32+GB RAM-a):** Pogodni za mašinsko učenje, video obradu i high-performance computing.

Tabela 1 Primeri instanci visokih performansi prema kategorijama u kojoj se nalaze

Kategorija	Tip instance	vCPUs	Memorija(GiB)	Skladište (TB)	GPUs	GPU Memorija (GiB)	Mreža (Gbps)	EBS propusnost (Gbps)
General Purpose	m8g.metal-48xl	192	768	-	-	-	50	40
Compute Optimized	c7i.metal-48xl	192	384	-	-	-	50	40
Memory Optimized	u7inh-32tb.480xlarge	1920	32768	-	-	-	200	160
Storage Optimized	i7ie.48xlarge	192	1536	16 x 7,500 GB = 120,000 GB	-	-	100	60
Accelerated Compute (GPU)	p5en.48xlarge	192	2048	-	8 x H200	1128 GB HBM3	3200 EFAv3	100

Pitanja za obnavljanje i proveru znanja

1. Da li svaki dobavljač usluga u oblaku mora posedovati data-centre? Obrazloži.
2. Šta znači redundantnost 2N+1?
3. Navedite elemente fizičke infrastrukture – NCPI-a.
4. Šta je to zona dostupnosti kod AWS-a?
5. U čemu je razlika između regiona i zona dostupnosti?
6. Kakav tip EC2 instanci je najefikasniji za korišćenje kod mašinskog učenja?

3.5. AWS IAM

U AWS-u posao kontrole identiteta i pristupa obavlja **Identity and Access Management (IAM).** To je servis koji definiše način i uslove obavljanja autentifikacije i autorizacije. Bez dobro postavljenih IAM polisa, svaki naredni sloj, od S3 bucket-a do EC2, počiva na lošoj i osetljivoj osnovi.

Svaki zahtev u AWS-u sadrži tri pitanja:

- **Ko** (principal) – Principal može biti čovek (federated user), servis (Lambda), mašina (virtuelna mašina sa IAM profilom) ili neki eksterni partner.
- **Šta** (akcija nad resursom) – Akcija može biti čitanje, pisanje, listanje, opisivanje itd.
- **Pod kojim uslovima** (kontekst) – Uslovi mogu biti vreme, IP opseg, prisustvo multifaktorske autentifikacije (MFA), tagovi, izvor itd.

IAM zatim kombinuje **polise** (policies) i donosi odluku. Podrazumevano, sve je zabranjeno (least privilege access). Dopušteno postaje tek kada postoji eksplicitno **Allow** i kada ga ne „pregazi" **Explicit Deny**. Pravilo gde je podrazumevano deny, a explicit deny nadjačava allow, stub je sigurnog dizajna i objašnjava zašto pogrešno postavljena jedna zabrana može „ugasiti" pristup i kad deluje da je sve dozvoljeno.

Korisnik i grupa i prateće operacije (kreiranje korisnika, dodela lozinke/ključeva i polise preko grupa) su istorijski pojmovi. U modernoj praksi, za ljude se preferira **federacija** (npr. univerzitetski IdP, Google/Microsoft) i **role** (korisnik se prijavi kod identity provajdera i privremeno preuzme AWS rolu – *assume role*). Time se izbegavaju statični pristupni ključevi i obezbeđuje značajno bezbedniji pristup.

Rola (Role) je entitet koji nema kredencijale dok ga neko ne preuzme preko **AWS STS** (Security Token Service). Rola ima **trust policy** koji definiše ko sme da je preuzme i **permission policy** koji definiše šta ta rola sme da izvrši. Isti obrazac ponašanja važi za ljude, aplikacije i servise kada su role u pitanju. Na primer, kada Lambda funkcija dobije rolu, ona na osnovu te role ima dozvole da prozove S3, DynamoDB itd., bez čuvanja bilo kakvih kredencijala u samom kodu.

Service role i **service-linked role** su role koje AWS servisi prave i korišćenjem njih upravljaju nekim servisima kao što su EKS (Elastic Kubernetes Service), Karpenter, ECS (Elastic Container Service) itd. **Instance profile** ima ulogu da veže EC2 instancu za određenu rolu. U slučaju Kubernetesa to je **IRSA** (OIDC federacija) koja veže rolu za service account u samom podu. Kod ECS zadataka (ECS Task) postoji task role koji vezuje rolu i sam ECS proces. Bez obzira na način i naziv vezivanja role sa samim resursom, princip rada i dozvola ostaje isti.

Polise (Policy) su JSON[3] dokumenti sa nizom **Statement** blokova kao što su:

- **Effect** – Allow/Deny;
- **Action** – Koje akcije mogu da se izvrše ;
- **Resource** – Nad kojim resursima mogu da se izvršavaju te akcije ;
- **Condition** – Opcioni blok, može da diktira pod kojim uslovima se izvršavaju, kao što su određeni tagovi, sa određenih IP adresa itd.

Postoji nekoliko vrsta polisa:

[3] JSON (JavaScript Object Notation) – popularan format za razmenu podataka.

- **Identity-based** polise – povezane za rolu, korisnika, grupu;
- **Resource-based** polise – vezane za sam resurs: S3 bucket policy, SQS queue policy, Lambda resource policy itd.
- **Session** polise – dodatno sužavaju prava privremenih kredencijala pri akciji AssumeRole
- **SCP** (Service Control Policies) – na nivou **AWS Organizations** služi kao ograda za naloge.

Dve važne tehnike daju finu kontrolu:

- **Kontekst i uslovi** (Condition) – na primer, „dozvoli s3:PutObject samo ako korisnik ima MFA", „samo iz IP-opsega fakulteta", „samo za resurse sa tagom env=dev".
- **ABAC** (Attribute-Based Access Control) – na primer, koristite tagove na principalima i resursima i pravila tipa „dozvoli ako se tagovi poklapaju", što je idealno za velike ekipe i mnoge projekte bez velike količine statičnih politika.

Polise mogu biti **managed** (deljene, verzionisane, AWSove ili vaše) ili **inline** (ugrađene direktno u jedan entitet). Za bezbedno delegiranje koristi se **permission boundary** koja govori koja je gornja granica preko koje se ne može ni ako korisnik sebi doda šire dozvole. U praksi, granice i SCP su komplement – SCP ograničava nalog, boundary ograničava entitet.

AWS promoviše princip kratkotrajnih kredencijala (no long-lived credentials). Pristup ljudima se obično vrši preko **IAM Identity Center** (SSO) ili SAML/OIDC federacije, aplikacije korisit **AssumeRole** (STS) i dobijaju privremene kredencijale sa rokom trajanja i kontekstnim ograničenjima.

MFA (Multi Factor Authentification) je obavezna praksa za role sa visokim privilegijama i root naloge. Mnoge osetljive operacije se mogu usloviti MFA polisama, na primer, brisanje podataka iz S3 itd.

Kada zahtev stigne, IAM prikupi sve relevantne izjave iz identity i resource-based polisa, primeni kontekst (Condition ključeve, tagove, IP, vreme, prisutnost MFA, izvor role itd.) i generiše ishod. Ako se bilo gde pojavi Explicit Deny, zahtev pada. Ako postoji Allow i nema Deny, zahtev prolazi. Ako nema ni Allow ni Deny, onda se podrazumeva Deny. Ove mehanizme je moguće proveriti uz pomoć alata kao što su **IAM Policy Simulator** i **Access Analyzer**. Uz pomoć njih moguće je unapred proveriti ponašanje.

IAM Access Analyzer je alat koji analizira politike i prijavljuje neočekivane spoljne pristupe. **IAM Credential Report / Access Advisor** je alat koji daje inventar pristupnih podataka (ključevi, lozinke, starost) i pokazuje koje servise je entitet stvarno koristio, što može biti korisno za identifikovanje suvišnih dozvola servisima. **CloudTrail** je log svih kontrolnih poziva (ko, šta, kada) i osnova za forenziku i usaglašenost. Ovo je jedan od obaveznih alata u enterprise okruženjima čiji se logovi obično čuvaju u posebnom AWS nalogu i zaštićeni su od izmena (immutable), pa se uz pomoć alata kao Athena vrši analiza i pregled. Tipično se skladište u S3 bucket.

Operativno, za bezbedno pristupanje okruženju oblaka ključni su zaštita root naloga (MFA, bez pristupnih ključeva, „break-glass" procedura), uvezivanje MFA za privilegovane role, uklanjanje nepotrebnih access ključeva, uvođenje rotacije lozinki i ključeva, alarmiranje na „neobične" akcije.

Praktični modeli dizajna

Bezbedne postavke gotovo uvek biraju federaciju + role, ne lokalne IAM korisnike. Projekti i okruženja (dev, staging, prod) dobijaju skupove dozvola, preko Identity Center-a, sa što je moguće ograničenijim dozvolama akcija i ABAC pravilima da se spreči „prelivanje" pristupa. Aplikacije ne poseduju tajne, tj. kredencijale, EC2/ECS/EKS dobijaju role, a spoljne integracije koriste cross-account role sa ExternalId.

Delegiranje ostalim timovima radi se kroz permission boundary tako da oni mogu kreirati svoje role, ali ne i super-korisnike.

Pri prelazu između naloga ili organizacionih jedinica važi jednostavna mapa:

- **SCP** – organizacione „ograde“
- **Permission boundary** – granica entiteta
- **Identity/resource politike** – konkretna dozvola
- **Session policy** – dodatni filter u trenutku preuzimanja role.

Neke od čestih grešaka koje se prave su:

- Stalni pristupni ključevi (access keys) za ljude. Obavezno ih zameniti federacijom i privremenim sesijama.
- Preširoke polise koje dozvolje sve (na primer, Action: “*”, Resource: “*”) koje se podese za svrhe testiranja i tako ostanu kao trajne. Uvek krenuti sa principom najmanjih privilegija i postepeno širiti po potrebi.
- Trust polise u okviru kojih rola ima dozvole, ali ne i ograničenje ko je može preuzeti, što dozvoljava svima da je koriste.
- Mešanje uloga između SCP i IAM. SCP ne daje dozvole, već postavlja ograničenja. Ako je na SCP neka akcija ograničena, IAM polisa je ne može vratiti.

Kratki primeri

Sledi nekoliko primera IAM polisa i njihova interpretacija.

Polisa:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": ["ec2:*"],
    "Resource": "*",
    "Condition": {
      "StringEquals": { "aws:RequestedRegion": "eu-central-1" }
    }
  }]
}
```

Interpretacija: Dozvoli sve EC2 akcije (ec2:*) nad bilo kojim resursom (*), ali samo kada je zahtev usmeren ka regionu eu-central-1 (Frankfurt).

Polisa:

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Action": "ec2:*",
    "Resource": "*",
    "Condition": {
      "IpAddress": {
        "aws:SourceIp": ["252.152.23.11/32"]
      }
    }
  }
}
```

Interpretacija: Dozvoli sve EC2 akcije (ec2:*) nad svim resursima (*), samo ako zahtev dolazi sa određene IP adrese (252.152.23.11). Ideja je da se pristup EC2-u ograniči na pozive koji stižu iz određene mreže/NAT-a.

Polisa:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VisualEditor1",
      "Effect": "Deny",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::conf-*",
        "arn:aws:s3:::conf-*/*"
      ]
    }
  ]
}
```

Interpretacija: Ovo je eksplicitna zabrana (Deny) za sve S3 akcije (s3:*) nad svim bucket-ima čije ime počinje sa conf- i nad svim objektima u tim bucket-ima. Važi za svakog principala kome je ova polisa prikačena (korisnik/rola), u svim regionima (S3 je globalan). Pošto je Deny eksplicitan, on nadjačava bilo koji Allow iz drugih polisa za te bucket-e i objekte.

IAM je kostur AWS bezbednosti i operativne kontrole. Razumevanje pojmova (principal, rola, trust/permission polisa, uslovi), mehanike evaluacije (default deny, explicit deny) i alata (STS, Identity Center, Access Analyzer, CloudTrail) omogućava da se grade sistemi koji su istovremeno upotrebljivi i sigurni. Ako se ovaj pristup primeni od prvog dana, ostatak arhitekture će imati čvrst oslonac na kome može sigurno i bezbedno da raste.

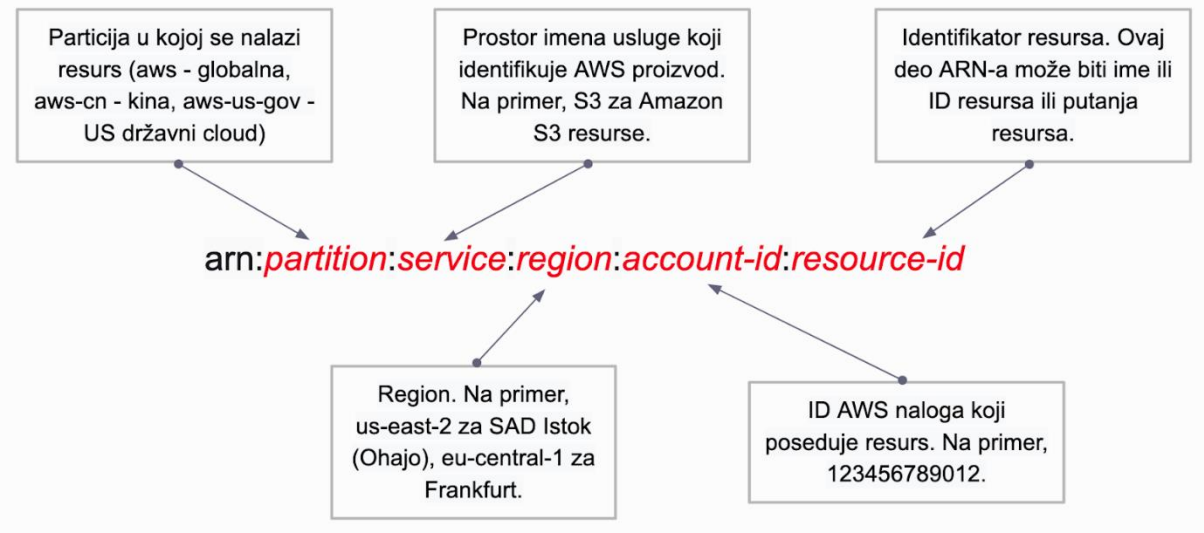
Zadatak

Potrebno je napraviti S3 rolu sa polisom da dozvoli sve akcije nad S3 samo za resurse koji su tagovani sa „laboratorija".

1. Napraviti cross account rolu gde može da se izvršava samo preuzimanje objekata iz S3 pod putanjom /billing/.

3.6. ARN (Amazon Resource Name)

ARN je jedinstveni identifikator svakog AWS resursa i funkcioniše kao forma kojom polise (IAM, S3 bucket policy itd.) precizno identifikuje na koji tačno resurs se dozvola ili zabrana odnosi. Standardni format je prikazan na slici (Slika 5):



Slika 5 Prikaz strukture ARN

Primeri ARN:

- EC2 instanca - arn:aws:ec2:eu-central-1:123456789012:instance/i-0abc123d
- S3 bucket - arn:aws:s3:::my-bucket
- S3 objekat - arn:aws:s3:::my-bucket/path/to/file.txt
- IAM rola - arn:aws:iam::123456789012:role/BackendApp
- Lambda funkcija - arn:aws:lambda:eu-central-1:123456789012:function:process-order

- SQS - arn:aws:sqs:eu-central-1:123456789012:orders-queue
- KMS ključ - arn:aws:kms:eu-central-1:123456789012:key/2b9e…
- Route 53 zona - arn:aws:route53:::hostedzone/Z0123456789ABCDE

4. Virtuelizacija

Ciljevi ovog poglavlja su upoznavanje sa motivima virtuelizacije, taksonomijom virtuelizatora, primenama virtualizatora i virtuelnih mašina kod AWS-a.
Po savladanom poglavlju student će moći da:

- navede tipove virtuelizatora i primere konkretnih softvera
- objasni pojmove pune virtuelizacije i paraviertuelizacije
- uporedi emulaciju i simulaciju
- diskutuje o prednostima i nedostacima virtuelizacije.

Student će umeti da:

- instalira i konfiguriše virtuelnu mašinu u VirtualBox-u
- konfiguriše virtuelnu mašinu u AWS-u.

U prethodnom poglavljima dat je širi prikaz pojma računarstva u oblaku, evolucije tehnologija koje su dovele do oblaka, motiva za korišćenje oblaka i pristupa migraciji aplikacija na oblak, kao i prikaz infrastrukture koja se nalazi iza oblaka. U više navrata istaknut je značaj virtualizacije za razvoj računarstva u oblaku, ali i naglašeno da oblak nije isto što i virtualizacija. Takođe, kao bitan korak u razvoju računarstva uopšte, navedeno je razdvajanje hardvera i softvera. U ovom poglavlju će se detaljnije obraditi koncept virtualizacije, kao i konkretni mehanizmi koji se koriste generalno ili u funkciji računarstva u oblaku.

4.1. Pojam virtuelizacije

U konvencionalnom računarstvu, tesna povezanost hardvera i softvera se podrazumeva. Da bi određeni softver mogao da se pokrene, neophodna je fizička mašina određenih svojstava, na primer dvojezgarni x86-64 procesor sa 8 GB RAM-a i određen operativni sistem, na primer Ubuntu 16. Ukoliko je potrebno pokrenuti softver koji zahteva Windows 7, postojeći fizički računar neće moći da se lako iskoristi, tj. moguće rešenje je uspostaviti DualBoot i naizmenično koristiti ova dva programa. Alternativa je koristiti poseban fizički računar. Gomilanje hardvera, pored neposrednih troškova, donosi i povećanu potrošnju električne energije i dodatni fizički prostor, umnožava se administracija itd.

Virtualizacija omogućava da se u okviru jednog fizičkog računara formira više odvojenih virtuelnih računara. To se postiže virtuelizacijom hardvera (procesora, memorije, diska, mreže). Virtuelizacija apstrahuje fundamentalne resurse; uprošćava njihovo korišćenje, izoluje korisnike jedne od drugih i podržava replikaciju, koja dalje povećava elastičnost sistema.

Virtualizacija simulira interfejs ka fizičkom objektu kroz sledeće funkcije:

- Multipleksiranje: Kreira se više virtuelnih objekata od jedne pojave fizičkog objekta. Primer: procesor se multipleksuje između više procesa ili niti.
- Agregacija: Jedan virtuelni objekat se kreira od više fizičkih objekata. Primer: više fizičkih diskova se agregira u jedan RAID disk.
- Emulacija: Formira se virtuelni objekat određenog tipa od drugog tipa fizičkog uređaja. Primer: fizički disk emulira RAM memoriju.
- Multipleksiranje i emulacija. Primer: virtuelna memorija sa straničenjem multipleksira realnu memoriju i disk; virtuelna adresa emulira stvarnu adresu.

Softverski modul koji koordiniše proces virtualizacije i obezbeđuje međusobnu izolaciju virtuelnih mašina (VM) jeste hipervizor[4]. Hipervizor deli resurse računarskog sistema na **virtuelne mašine (VMs)**.

[4] Kao sinonim se često koristi i termin virtualizator i monitor virtuelnih mašina (VMM – virtual machine monitor).

Omogućava da se više OS izvršava konkurentno na jednoj hardverskoj platformi. Virtuelna mašina jeste izolovano okruženje koje se pojavljuje kao ceo računar, ali u stvari ima pristup samo delu računarskih resursa.

4.2. Simulacija i emulacija

Na ova dva termina se često može naići u kontekstu virtualizacije. Iako veoma slični, termini nisu sinonimi. Simulacija predstavlja izgradnju funkcionalnog modela nekog realnog sistema kojim se mogu testirati njegova svojstva ili performanse. Primer simulatora je avio-simulator, na kojem se budući piloti obučavaju za upravljanje letelicom. Upravljač i komande su kreirane da se ponašaju kao u stvarnom avionu, a sam simulator se nalazi u trenažnom centru. Simuliran je deo aviona koji je značajan za obuku, na primer nije bitno da postoji razglas ili erkondišn. Sa druge strane, emulator prevodi instrukcije namenjene realnom sistemu. U primeru sa avio-obukom, emulator bi omogućio da se pomoću zemaljskih komandi upravlja stvarnom letelicom. Tada bi reč bila o emulatoru leta: kada korisnik emulatora zada određenu visinu letelice, na ekranu se neće samo pokazati simulirana slika uzletanja i broj koji označava visinu, već će se komande direktno preneti na pravu letelicu koja će uzleteti. Emulacija bi trebalo da može da poveže i nesrodne interfejse, na primer ako ne postoji ručica za otvaranje zakrilaca aviona, taster F10 bi mogao da emulira tu ručicu i izvrši datu funkciju.

Kod računarskih sistema, emulacijom se postiže izvršavanje sistema različite arhitekture, npr. na Intel x86 procesoru se izvršava aplikacija pisana za ARM.

4.3. Tipovi hipervizora

Sistem na kojem je instaliran hipervizor i na kojem se izvršavaju virtuelne mašine jeste domaćin, odnosno host. Virtuelne mašine koje se izvršavaju na hostu su gostujući sistemi *(guest)*.

Postoje dva tipa hipervizora. Prvi tip podrazumeva instalaciju neposredno na hardveru. Nakon što se instalira hipervizor, formiraju se virtuelne mašine sa odgovarajućim operativnim sistemima. Ovaj tip hipervizora se naziva i bare-metal upravo zato što se instalira neposredno na hardver. Ovakav pristup je efikasan, pouzdaniji i bezbedniji. Nedostaci su kompleksnija administracija i uska podrška za uređaje. Primeri hipervizora tipa 1 jesu VmWare ESXi, Hyper-V, XEN, IBM z/VM, AWS Nitro.

Hipervizor tipa 2 podrazumeva instalaciju u okviru operativnog sistema, tzv. hosted hipervizor. Dakle, prvo se instalira osnovni operativni sistem (na primer, Windows 11), a zatim se, postupkom standardne instalacije softvera, instalira hipervizor. Na kraju, formiraju se virtuelne mašine. Hipervizor tipa 2 podržava raznovrsniji hardver jer OS domaćina obezbeđuje drajvere za uređaje, a pogodan je za razvoj aplikacija i obuku. Takođe, jednostavniji je za instalaciju i administraciju. Sa druge strane, hipervizor ne pristupa direktno uređajima (već to radi preko OS-a domaćina), te performanse mogu trpeti. Takođe, pošto OS domaćin izvodi raspoređivanje procesa, ne može se očekivati da gostujući sistem radi u realnom vremenu. Primeri hipervizora tipa 2 su: user-mode Linux, VirtualBox, VmWare Workstation, Parellels Desktop for Macs.

Interesantno je napomenuti način funkcionisanja hipervizora Hyper-V. U pitanju je hipervizor koji je tipa 1, ali sa druge strane dobija se u sklopu instalacije Windows-a (potrebno je eventualno uključiti ga i pokrenuti). Uprkos tome što se dobija kao deo Windows-a, Hyper-V ipak radi neposredno na hardveru. Kada se pokrene, Windows postaje gost na Hyper-V hipervizoru, koji radi neposredno na hardveru. Više detalja o Hyper-V arhitekturi dostupno je u zvaničnoj dokumentaciji[5].

[5] https://learn.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/hyper-v-architecture

KVM (Kernel-based Virtual Machine) je takođe specifičan virtualizator. Realizovan je tako da se Linux kernel ponaša kao virtualizator, tako da na prvi pogled može izgledati da je u pitanju tip 2 hipervizora. Međutim, kernel koji je instaliran na hardveru se ne smatra operativnim sistemom-domaćinom, već virtualizatorom, tj. hipervizorom. Najčešće radi u kombinaciji sa QEMU-om, koji vrši emulaciju različitih uređaja i omogućava brzo izvršavanje aplikacija uz minimalne gubitke usled virtualizacije. KVM je čest izbor za data-centre.

U tabeli 2 dati su primeri hipervizora. ISA = *Instruction Set Architecture* predstavlja arhitekturu i prateći skup instrukcija određene platforme

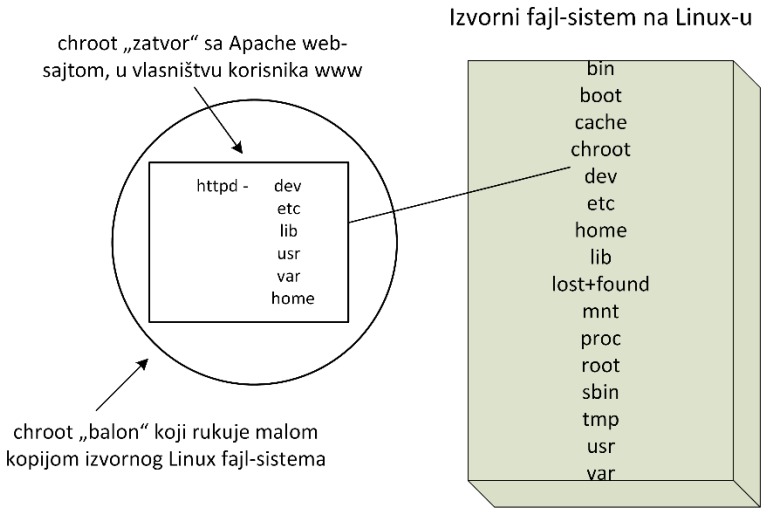
Tabela 2 Primeri hipervizora

Naziv	Host OS	OS gosta	Proizvođač
Power VM	No host os	Linux, AIX	IBM
z/VM	No host os	Linux on z-ISA	IBM
Lynx Secure	No host os	Linux, Windows	LinuxWorks
Hyper-V Server	Windows	Windows	Microsoft
Oracle VM	No host os	Linux, Windows	Oracle
RTS Hypervisor	No host os	Linux, Windows	Real Time Systems
SUN xVM	No host os	Linux, Windows	SUN
VMware EX Server	No host os	Linux, Windows Solaris, FreeBSD	VMware
VMware Fusion	MAC OS x86	Linux, Windows Solaris, FreeBSD	VMware
VMware Server	Linux, Windows	Linux, Windows Solaris, FreeBSD	VMware
Denali	Denali	ILVACO, NetBSD	University of Washington
Xen	Linux Solaris	Linux, Solaris NetBSD	University of Cambridge

4.4. Virtuelizacija na nivou operativnog sistema

Prethodno navedeni tipovi virtualizacije podrazumevaju postojanje hipervizora na kojem se pokreću virtuelne mašine. Ispod virtuelizatora može stajati direktno hardver (kod tipa 1) ili operativni sistem – domaćin (kod tipa 2). Kod virtualizacije, na nivou operativnog sistema izostavljen je hipervizor. Virtuelne mašine nisu nezavisne celine koje sadrže posebne operativne sisteme, već su u pitanju izolovane virtuelne instance koja dele jedinstven kernel domaćina i izvršavaju se u korisničkom režimu. U zavisnosti od implementacije, ovakve instance se nazivaju kontejneri, virtuelni privatni serveri, virtuelni kerneli ili „zatvori“ *(jails)*. U narednom poglavlju će biti više reči o kontejnerima i kontejnerizaciji aplikacija.

Rudimentaran oblik kontejnera koji se može formirati na Linux/UNIX sistemima jeste tzv. chroot jail. Ukratko: formira se posebno, izolovano okruženje u kojem aplikacija funkcioniše kao da joj je na raspolaganju čitav računar. Ovakav pristup se koristi najviše iz razloga bezbednosti. Na primer, ako je potrebno izolovati web-server tako da eventualni upadi ne narušavaju čitav sistem, može se kreirati jedan poseban direktorijum, u njemu kreirati stablo potrebnih direktorijuma i prekopirati sve aplikacije i biblioteke potrebne za rad web-servera. Na primer, ako se pri standardnoj instalaciji web-server Apache nalazi u /usr/bin i koristi biblioteke iz /lib, a konfiguraciju iz /etc, potrebno je ovakvu strukturu napraviti posebno (Slika 6) u izdvojenom direktorijumu, sa svim potrebnim fajlovima. Na kraju, pokreće se komanda chroot i aplikacija Apache uz naznaku u kojem direktorijumu radi. Sada Apache „vidi“ sve ono što mu je potrebno za rad, kao da je instaliran u standardnom direktorijumu, sa pregledom celog fajl-sistema, a u stvari se izvršava u svom „matriksu“.



Slika 6 chroot "zatvor"

4.5. Uslovi efikasne virtuelizacije

Uslovi za efikasnu virtuelizaciju definisani su još pre četiri decenije (Popek & Goldberg, 1974[6]):

1. Program koji se izvršava pod hipervizorom mora da ispolji suštinski isto ponašanje kao da se izvršava neposredno na ekvivalentnoj mašini.
2. Hipervizor u potpunosti kontroliše virtualizovane resurse.
3. Značajan procenat mašinskih instrukcija trebalo bi da se izvrši bez mešanja hipervizora.

[6] *Popek, G. J.; Goldberg, R. P. (July 1974). "Formal requirements for virtualizable third generation architectures". Communications of the ACM. **17** (7): 412–421. doi:10.1145/361011.361073. S2CID 12680060.*

Kod pune virtualizacije, sama arhitektura hardvera podržava gostujući OS i hipervizor faktički prosleđuje instrukcije hardveru. Gostujući sistem apsolutno je „nesvestan" da radi kao virtuelna mašina.

Kod paravirtuelizacije, gostujući OS mora biti prilagođen, da bi mogao da funkcioniše na datom hardveru i u saradnji sa određenim hipervizorom. Hipervizor igra ulogu emulatora za pojedine uređaje. Poboljšane su performanse. Gostujući sistem se za sve instrukcije koje ne mogu da budu virtuelizovane obraća hipervizoru kroz tzv. hiperpozive (hypercalls). To su pozivi nalik sistemskim, putem kojih se zahteva usluga od hipervizora.

Za performanse virtuelizacije je značajno da se maksimalno smanji emulacija i posredovanje na liniji gostujući sistem – hardver. Da bi to bilo moguće, potrebna je podrška hardvera. AMD-V i Intel-VT omogućavaju pojedinim privilegovanim instrukcijama da se izvršavaju direktno na hardveru. Ove postavke se uključuju u BIOS-u računara.

4.6. Ugneždena virtualizacija

Predstavlja dodatni nivo virtuelizacije, gde se u okviru virtuelne mašine instalira poseban hipervizor i onda u njemu nove virtuelne mašine. Neophodna je podrška hardvera (AMD-V/IntelVT), kao i hipervizor koji omogućava ovu opciju. Ugneždena (nested) virtuelizacija je pogodna za testiranje i obuku vezanu upravo za virtuelizaciju. Posebno je korisna u okruženju u oblaku. Dobavljač oblaka već koristi svoj podrazumevani hipervizor: kada korisnik želi novu virtuelnu mašinu, iza nje stoji hipervizor dobavljača. Ukoliko je korisniku potrebno da u okviru takve virtuelne mašine instalira i testira više drugih operativnih sistema, rešenje je ugneždena virtuelizacija.

Moguće je koristiti isti hipervizor (na primer, na virtualnim mašinama pokrenutim u Hyper-V pokrenuti nove mašine preko Hyper-V-a) ili kombinovati različite hipervizore.

4.7. Prednosti virtuelizacije

Kroz uvod u motive virtuelizacije već su se iskristalisale njene određene prednost. Sledi upotpunjen pregled:

- Bolja iskorišćenost resursa i niži troškovi hardvera. Fizički računar veoma često koristi samo deo svojih resursa. Pokretanjem više virtuelnih mašina na istom hardveru, raste iskorišćenost. Takođe, moguće je dinamički balansirati resurse: ako je jednoj mašini u nekom tenutku potrebno više resursa, a druga troši veoma malo, prvoj mašini se privremeno dodeljuje više resursa nauštrb druge mašine. Dobrim planiranjem nabavke fizičkih resursa ostvaruju se uštede.
- Smanjenje troškova infrastrukture. Pored hardvera, postoji niz aspekata na kojima se štedi korišćenjem virtualizacije: prostor, potrošnja električne energije, hlađenje, kadrovi (ljudski resursi).
- Povećana robusnost sistema i smanjeno vreme otkaza. Virtuelizacija povlači portabilnost sistema. U slučaju otkaza hardvera, jednostavno je migrirati virtuelnu mašinu na nov sistem.
    - *Live* migracija podrazumeva migraciju pokrenute virtuelne mašine na drugi hardver, bez diskonektovanja klijenata. U idealnom slučaju, migracija je neprimetna; *downtime* je toliko kratak da korisničko iskustvo ne trpi nikakve posledice.
- Pojednostavljena administracija. Virtuelne mašine se centralizovano prate i kontrolišu.
- Jednostavnija dogradnja kapaciteta. Mnogo je jednostavnije proširiti kapacitet virtuelne mašine, nego fizičke.
- Pojednostavljena instalacija sistema. Kloniranje sistema je veoma jednostavno, a i instalacija je od početka pojednostavljena: za pojedine operativne sisteme (npr. Ubuntu kod VmWare-a)

dostupna je i nenadgledana instalacija, uz unošenje osnovnih informacija na početku instalacije.

- Podrška za starije aplikacije. Virtuelizacija je često jedini način za pokretanje starijih aplikacija, koje su iz nekog razloga i dalje nezamenjive. Hardver je neophodno inovirati, novi hardver zahteva npr. Windows 11 i na njemu se ne može pokrenuti stari program za Windows XP i onda se kao rešenje uvodi virtualna mašina.
- Jednostavniji razvoj softvera, testiranje, edukacija. Moguće je na jednostavan način uspostaviti okruženje za razvoj za različite platforme, testirati performanse programa uz zadat kapacitet servera, pre puštanja u produkciju. Ukoliko se u razvoju softvera zahteva restartovanje, to mnogo kraće traje nego kod fizičkog hardvera. Čuvanje snimaka (snapshots) virtuelne mašine omogućava jednostavnije testiranje softvera i eventualno vraćanje na prethodne postavke.
- Bezbednost. Virtualne mašine su izolovane od osnovnog sistema i dodaju još jedan sloj iznad hardvera. Na primer, uništavanje virtuelnog diska ne reflektuje se ne fizički disk.

Sve navedene prednosti se reflektuju i na računarstvo u oblaku.

4.8. Nedostaci virtualizacije

Svaka tehnologija poseduje i određene nedostatke koje treba imati u vidu pri projektovanju sistema. Kao nedostaci virtuelizacije mogu se navesti:

- Problem jedinstvene tačke otkaza. Veći broj virtuelnih mašina se izvršava na jedinstvenom hardveru. Otkaz datog hardvera povlači otkaz svih virtuelnih mašina. Rešenje se nalazi u održavanju kopija virtuelnih mašina.
- Slabije performanse u odnosu na fizički hardver. U velikom broju slučajeva pad perfomansi je veoma mali u odnosu na izvršavanje na fizičkim procesorima. Postoje specifični slučajevi koji zahtevaju neposredno izvršavanje na hardveru (npr. neposredan pristup grafičkoj kartici za kriptografsku obradu), gde je razlika u odnosu na virtuelnu alternativu drastična.
- Izvorno, hardver koji je korišćen za potrebe virtualizacije namenjen je za neposredno izvršavanje, bez virtuelizacije. Uvođenje većeg broja virtuelnih mašina (koje često pripadaju različitim korisnicima) donosi rizik da će podaci procureti između korisnika. Bezbednosni propusti Spectre i Meltdown[7] su podigli budnost i usmerili pažnju stručne javnosti, proizvođača procesora i dobavljača oblaka na problem bezbednosti u distribuiranom okruženju.

Bezbednost i virtuelizacija

Virtuelizacija je i napadačima donela nove mogućnosti. Jedan primer je zlonamerni hipervizor, tzv. Virtual-Machine Based Rootkit (VMBR). Konvencionalni rutkit (rootkit) je zlonamerni program koji ostvaruje neovlašćen pristup računarskom sistemu, uz skrivanje svog prisustva. Može se integrisati duboko u sistem: kernel-rutkit ima najviši nivo pristupa, može menjati sistemske pozive i kompletno kontrolisati i sam sistem i sve korisničke aplikacije. VMBR se umeće između hardvera domaćina i operativnog sistema ili, ako već postoji hipervizor tipa 1, između hardvera i legitimnog hipervizora. U okviru zlonamernog hipervizora pokreće se poseban zlonamerni operativni sistem koji dalje može da prikuplja podatke potekle od realnog sistema, distribuira spam, inicira DDOS napade i slično.

[7] https://meltdownattack.com/

4.9. Udruživanje resursa, deljenje i obezbeđenje resursa

Udruživanje (Pooling) je jedan od osnovnih koncepata na kojim se zasniva računarstvo u oblaku. Pool predstavlja grupu resursa – procesorske snage, memorije, diskova – iz koje se po potrebi, na zahtev, može rezervisati određen kapacitet, a zatim, kada više nije potreban, vratiti u grupu

Grupišu se fizički ili virtuelni resursi. Virtuelni resursi se mogu grupisati tako što će biti raštrkani po različitim serverima i onda organizovani kroz softverski definisanu mrežu. Ili se, što može dati bolje performanse, prvo kreira grupa fizičkih resursa, na kojem se posle zasniva grupa virtuelnih resursa.

Udruživanjem resursa se vrši fleksibilno dodeljivanje, tako da se svaki resurs koji postane višak lako vrati u grupu slobodnih resursa, a ako poraste opterećenje, iz grupe „povlače“ novi resursi da bi se ispunili zahtevi.

Jedan od osnovnih faktora koji omogućavaju udruživanje jeste tzv. komodizacija hardvera. Umesto skupog, specijalizovanog hardvera, u data centrima se koristi serijski proizveden, relativno povoljan hardver, koji je onda jednostavno zameniti ili dodati.

Deljenje resursa dovodi do veće stope iskorišćenja resursa u oblaku. Kako veliki broj aplikacija radi preko zajedničke grupe resursa, prosečno iskorišćenje svake komponente resursa može se povećati njihovim deljenjem među različitim aplikacijama, budući da sve aplikacije generalno svoje vršne zahteve ne dostižu u isto vreme.

Računarstvo u oblaku omogućava deljenje grupisanih i virtuelizovanih resursa između aplikacija, korisnika i servera. Implementacija ovog koncepta zahteva odgovarajuću arhitektonsku podršku. Dok se serveri dele između korisnika i aplikacija, resursi poput skladišta, I/O operacija i komunikacionog propusnog opsega obično se dele između više virtuelnih mašina.

Deljenje resursa u okruženju uslužnog modela (utility service) ne dolazi bez izazova. Glavni izazov je obezbeđivanje kvaliteta usluge (Quality of Service – QoS), jer je izolacija performansi ključni uslov za QoS. Deljenje resursa može uticati na ponašanje drugih aplikacija u vreme izvršavanja, budući da više aplikacija konkuriše za isti skup resursa. Optimizacijom i odgovarajućom automatizacijom se mogu postići odgovarajuće performanse i uskladiti suprotstavljeni zahtevi.

Karakteristika sistema koja omogućava da jedna komponenta resursa služi različitim korisnicima (korisnicima, stanarima – tenants), pri čemu su oni međusobno izolovani, naziva se multi-tenancy (višekorisničko okruženje). Ovo je funkcionalnost koja je u svom pravom i potpunom obliku dostupna u modelu javnog oblaka. Stoga, bilo kakva rasprava o multi-tenancy konceptu u cloud računarstvu primarno se odnosi na implementaciju u javnom oblaku, mada postoje i druge varijante, o kojima će biti reči nešto kasnije.

U modelu višekorisničkog okruženja, korisnici ne mogu unapred zauzeti neku specifičnu fizičku komponentu resursa. Resursi se koriste od strane različitih korisnika privremeno, kada i koliko je potrebno. Ovaj model dobavljačima usluga omogućava da pružaju računarske usluge višestrukim izolovanim korisnicima (koji nisu međusobno povezani) koristeći jedan skup zajedničkih resursa.

Višekorisnički model u cloud računarstvu realizovan je na osnovu koncepata deljenja resursa bez vlasništva u virtuelizovanom režimu i privremene alokacije resursa iz grupe resursa. Ovaj pristup vodi smanjenju troškova obrade i povećanju iskorišćenja resursa, što su ključni aspekti uslužnog modela računarstva (utility computing).

Komponente resursa koje su grupisane na strani obavljača usluga u oblaku koriste se za pružanje usluga velikom broju korisnika. Računarstvo u oblaku ne dozvoljava trajnu akviziciju bilo koje komponente

resursa od strane korisnika. Kada korisnik zatraži resurse, odgovarajuće komponente resursa se isporučuju u virtuelizovanom režimu. Nakon što korisnikove potrebe budu ispunjene, resursi se vraćaju u grupu dostupnih ili slobodnih resursa, koji se mogu koristiti za ispunjavanje zahteva drugih korisnika u budućnosti.

Stoga, ne postoji jednoznačno mapiranje između korisnika i komponenti resursa u smislu korišćenja resursa. Takođe, nijedna komponenta resursa nije posvećena određenoj aplikaciji. Komponente resursa dodeljuju se korisnicima i aplikacijama isključivo na osnovu dostupnosti. Ovo povećava stopu iskorišćenja resursa, što zauzvrat smanjuje investicione troškove.

Korisnici mogu biti spoljni ili čak unutrašnji korisnici unutar organizacije (kao što su različiti odeljenja unutar iste organizacije), gde svaki korisnik zahteva svoje zaštićene zone. U računarstvu u oblaku, kada se govori o multi-tenancy konceptu, fokus se uglavnom pomera ka javnom oblaku, gde su zajednički stanari međusobno isključivi ili nepoznati jedni drugima, ali dele istu računarsku infrastrukturu idealno bez ugrožavanja individualne sigurnosti i privatnosti. Međutim, ideja multi-tenancy okruženja postoji i u zajedničkom (community) oblaku, pa čak i u privatnom oblaku, ali u alternativnom smislu.

Za razliku od ekskluzivnosti izolacije i individualnosti svakog stanara u javnom oblaku, zajednički stanari u zajedničkom oblaku nisu međusobno isključivi jer pripadaju istoj zajednici i dele slične interese. U privatnom oblaku, koncept multi-tenancy je ograničen na podstanare unutar jedne organizacije (ili pod okriljem jednog glavnog stanara).

Zahtevi korisnika u javnom i privatnom oblaku su gotovo slični, bez obzira na to što u jednom slučaju postoje spoljašnji stanari, dok su u drugom slučaju stanari unutrašnji.

4.10. Obezbeđivanje *(provisioning)* resursa

U tradicionalnom računarstvu, kada je potreban novi server (fizički ili virtuelni) za podršku određenom opterećenju, administratori ulažu mnogo napora i vremena kako bi instalirali i obezbedili server. Moderno računarstvo zahteva opciju brzog obezbeđivanja infrastrukture kako bi se zadovoljile promenljive potrebe korisnika. Sa pojavom tehnologije virtuelizacije i modela IaaS *(Infrastructure as a Service)* u računarstvu u oblaku, sada je potrebno samo nekoliko minuta da se postigne isti rezultat pod uslovom da je potrebna količina resursa dostupna. Na taj način, obezbeđivanje nove virtuelne mašine štedi mnogo vremena i truda u okruženju oblaka. Virtuelni server se može kreirati putem interfejsa za samoposluživanje, što se smatra jednom od najprivlačnijih karakteristika oblaka.

Statičko obezbeđivanje resursa je pogodno za aplikacije koje imaju predvidive i generalno nepromenljive zahteve za opterećenje. U ovom pristupu, nakon što se virtuelna mašina (VM) kreira, očekuje se da radi duže vreme bez dodatnih odluka o alokaciji resursa, čime se smanjuje opterećenje sistema. Odluka o alokaciji resursa donosi se samo jednom, i to na samom početku, kada korisnikova aplikacija počne sa radom.

Ovaj pristup omogućava više vremena za donošenje odluke o alokaciji resursa, jer to ne utiče negativno na performanse sistema.

Kod dinamičkog obezbeđivanja resursa, resursi se dodeljuju i uklanjaju prema potrebama tokom rada, čime se omogućuje elastičnost sistema. Dobavljači više ne moraju da održavaju određeni obim resursa koji je neiskorišćen za svaki sistem posebno, već održavaju zajednički pool resursa i obezbeđuju resurse iz njega kada su potrebni. Resursi se uklanjaju sa VM-ova kada više nisu potrebni i vraćaju se u pool. Ovim dinamičkim pristupom, usluge se naplaćuju prema korišćenju.

Dinamičko obezbeđivanje resursa rešava probleme statičkog pristupa, ali uvodi preopterećenje tokom rada. Da bi se rešio ovaj problem, predlaže se hibridni pristup dodeli resursa koji kombinuje statičko i

dinamičko obezbeđivanje. On počinje sa statičkom tehnikom dodeljivanja resursa u početnoj fazi kreiranja VM-a, a zatim prelazi na dinamičko ponovo dodeljivanje resursa. Ovaj pristup često može efikasno rešiti scenarije u stvarnom vremenu sa promenljivim opterećenjem u oblaku.

Tradicionalni računarski sistemi uglavnom prate pristup statičkog obezbeđivanje resursa. Međutim, veoma je teško tačno predvideti buduće zahteve bilo koje aplikacije (a samim tim i potrebe za resursima aplikacije) uprkos rigoroznom planiranju i naporima. Ovo prirodno dovodi do nedovoljno ili prekomerno dodeljenih resursa u tradicionalnom okruženju.

Kada potražnja za računarskim resursima pređe granicu dostupnih resursa, dolazi do nestašice resursa. Ovaj scenario je poznat kao nedovoljno obezbeđivanje (under-provisioning). Jednostavno rešenje za ovaj problem je da se rezerviše dovoljno resursa za aplikaciju kako bi nestašica resursa bila nemoguća. Međutim, ovim se uvodi novi problem. U tom slučaju, većina resursa će ostati neiskorišćena većinu vremena. Ovaj scenario je poznat kao prekomerno obezbeđeni resursi (over-provisioning).

Pitanja za proveru i obnavljanje znanja

1. Hteli bismo da na svom novom PC računaru pokrenemo program sa računara Commodor 64. Da li koristimo emulator ili simulator?
2. Koje su prednosti korišćenja hipervizora tipa 1?
3. Navedite primere tri hipervizora tipa 2.
4. Koji su motivi korišćenja virtuelizacije na nivou OS, kao što je chroot jail?
5. Nasuprot velikog broja prednosti virtuelizacije stoji ozbiljan nedostatak: postojanje jedinstvene tačke otkaza. Na koji način bi se mogao prevazići?
6. Na koji način radi zlonamerni hipervizor?

4.11. Praktični primeri virtuelizacije

Virtuelizacija na lokalnim mašinama

VirtualBox je besplatan alat za virtuelizaciju otvorenog koda, razvijen od strane Oracle-a. On omogućava kreiranje i upravljanje virtuelnim mašinama (VM) na vašem računaru, što znači da može pokretati više operativnih sistema (npr. Linux, Windows, macOS) istovremeno na jednoj fizičkoj mašini. Idealan je za testiranje, razvoj i učenje jer dozvoljava izolaciju različitih okruženja bez uticaja na host sistem.

VMware Workstation je komercijalni alat za virtuelizaciju koji omogućava pokretanje više operativnih sistema na jednoj fizičkoj mašini. Razvijen od strane VMware-a, ovaj softver je popularan među profesionalcima i IT administratorima zbog svoje pouzdanosti, performansi i naprednih funkcija. VMware Workstation podržava širok spektar operativnih sistema, uključujući Windows, Linux i macOS (kao gostujući sistem na macOS hostovima putem Fusion verzije).

U poređenju sa VirtualBox-om, VMware Workstation nudi bolju integraciju sa enterprise alatima, optimizovanu upotrebu hardverskih resursa i napredne funkcije poput snapshot-a, kloniranja, mrežne simulacije i integracije sa VMware vSphere-om. Postoje dve verzije:

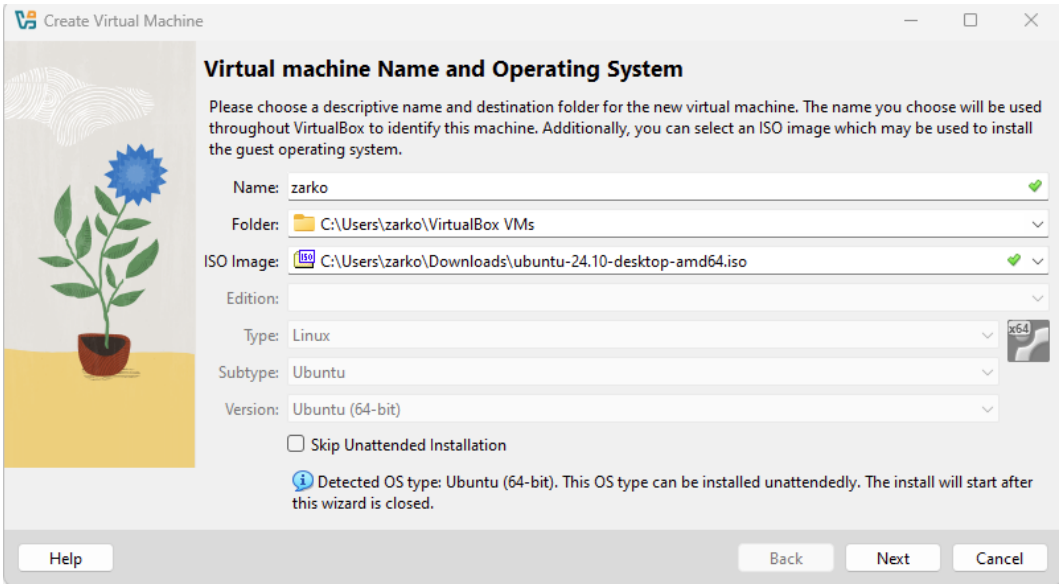
- **VMware Workstation Player** (besplatna za ličnu upotrebu, sa ograničenim funkcijama)
- **VMware Workstation Pro** (besplatna za ličnu upotrebu; plaćena verzija sa naprednim funkcijama za profesionalce)

VMware Workstation je idealan za razvoj, testiranje aplikacija, IT administraciju i simulaciju mrežnih okruženja.

U narednim koracima sledi pokretanje lokalne virtuelne mašine.

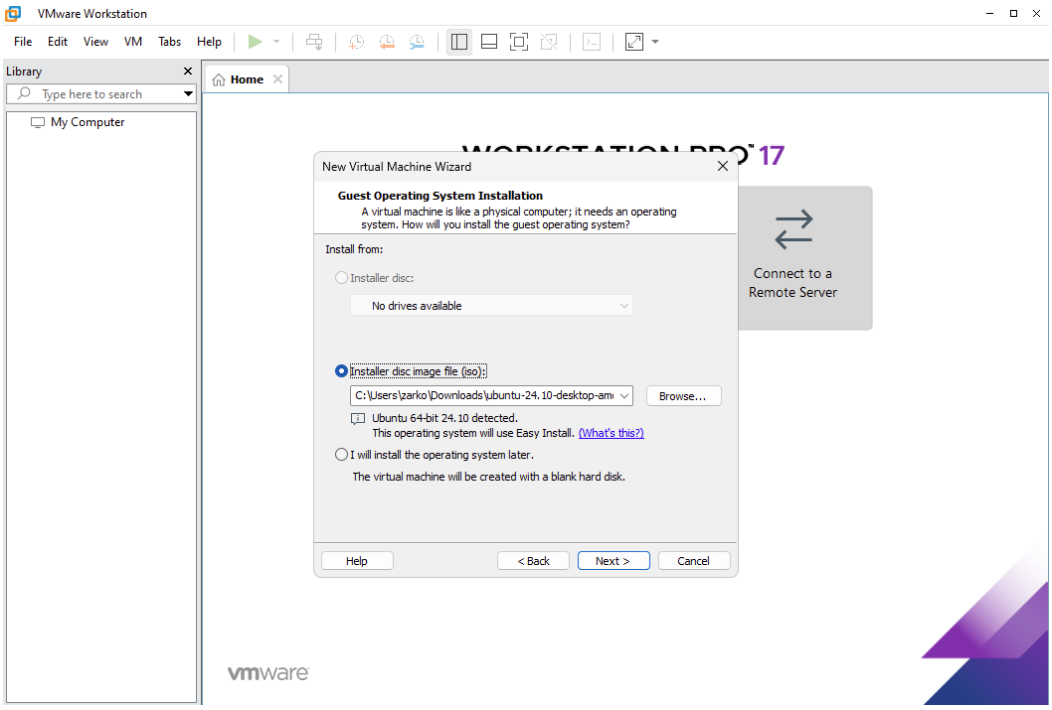
Potrebno je preuzeti željeni softver za virtuelizaciju u instalirati ga na host sistem. Zatim je potrebno da se, pre pokretanja virtuelne mašine, preuzme operativni sistem koji će biti instaliran. Za svrhe vežbe koristi se Ubuntu koji je moguće preuzeti sa zvaničnog web sajta u ISO formatu. Kada se preuzme, potrebno je podesiti pokrenuti softver za virtuelizaciju i napraviti novu virtuelnu mašinu.

Kod VirtualBox-a potrebno je izabrati opciju New, uneti naziv virtuelne mašine, zatim izabrati lokaciju gde će ona biti sačuvana i odabrati ISO file (Slika 7). Nakon što se odabere ISO file, sekcija sa operativnim sistemom trebalo bi automatski da prepozna koji sistem se instalira; u slučaju da ne prepozna, potrebno je ručno uneti parametre sistema.



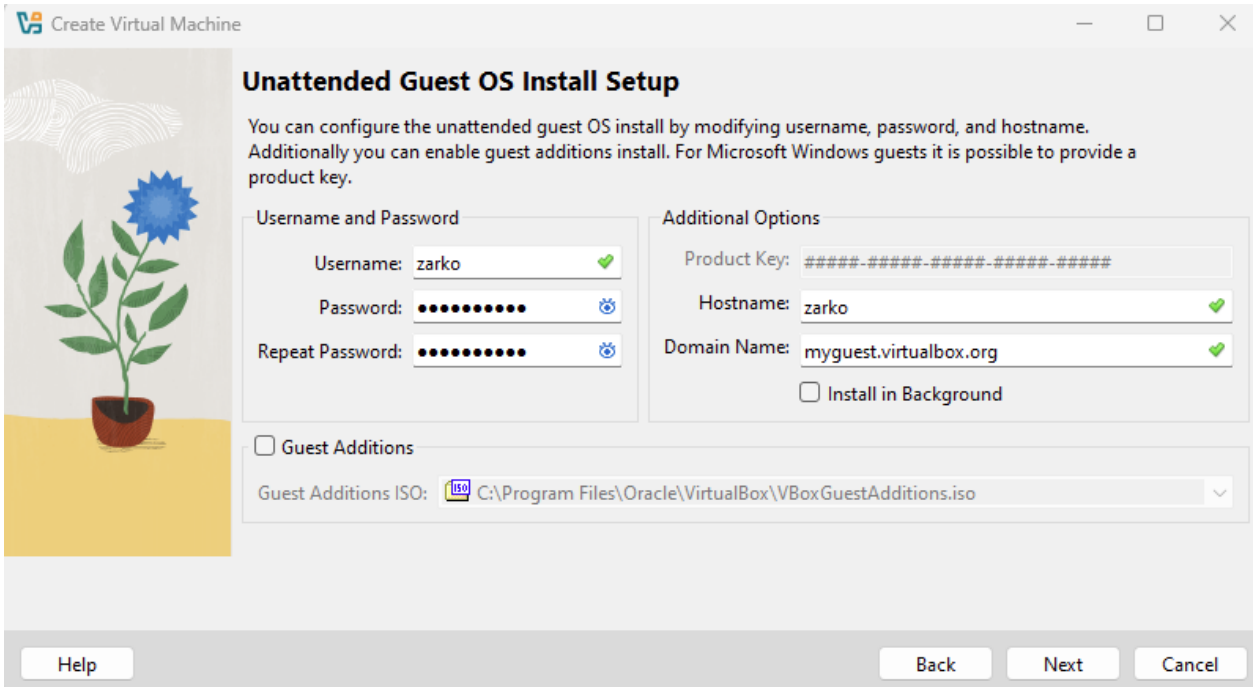
Slika 7 Kreiranje nove virtuelne mašine u VirtualBox-u

U slučaju VMWare Workstation Pro potrebno je ići na file, pa new virtual machine i izabrati Typical setup. Zatim se bira ISO image i takođe vrši automatska detekcija operativnog sistema (slika 8).

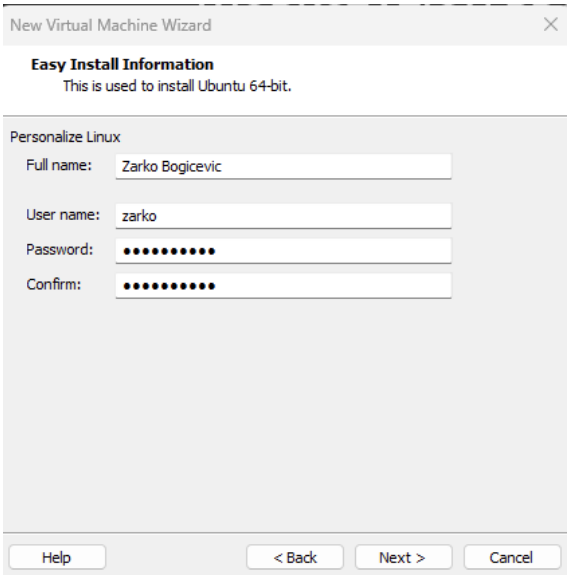


Slika 8 Kreiranje nove virtuelne mašine u VmWare Workstation Pro

Nakon toga se popunjavaju prikazani parametri (slika 9) ili Easy install u slučaju VMWare (Slika 10).

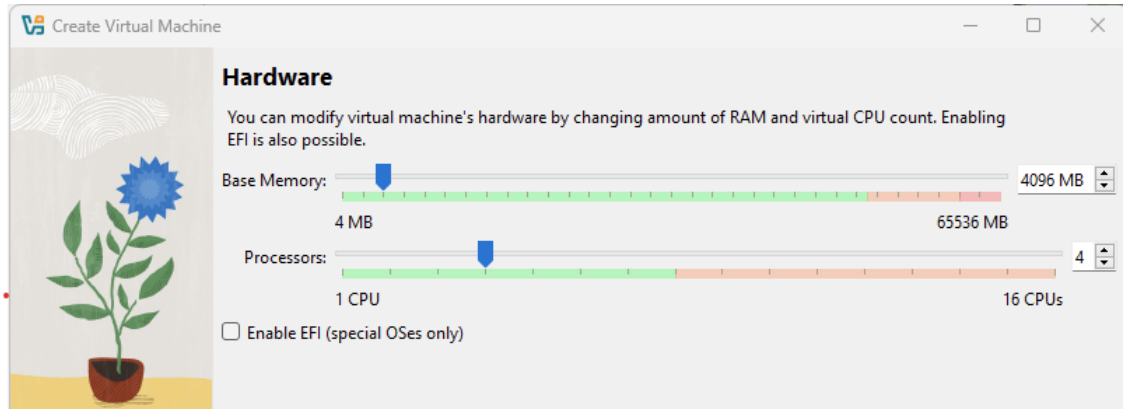


Slika 9 Podešavanje korisnika - VirtualBox

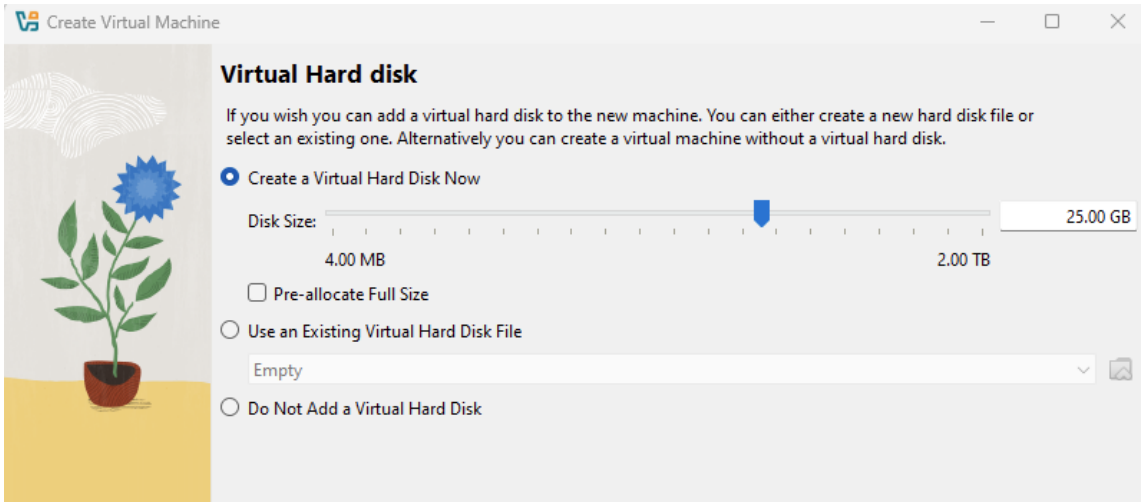


Slika 10 Podešavanje korisnika u Easy Install modu – VmWare Workstation Pro

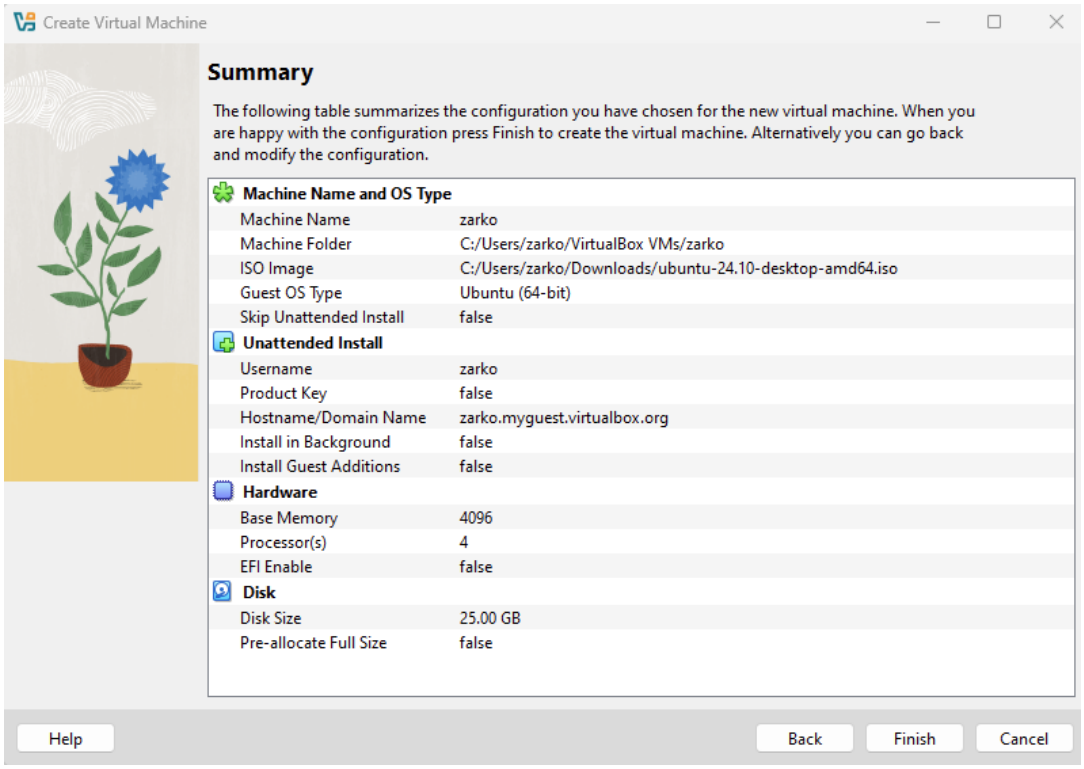
Dalje, potrebno je dodeliti resurse virtuelnoj mašini (Slika 11 i 12). Bitno je dodeliti onoliko resursa koliko je potrebno za zadatak, ne ispod minimalne količine preporučene od strane zvanične dokumentacije, ali važno je i ostaviti dovoljno resursa kako bi host mašina mogla nesmetano da funkcioniše.



Slika 11 Dodela resursa procesora i RAM memorije - VirtualBox

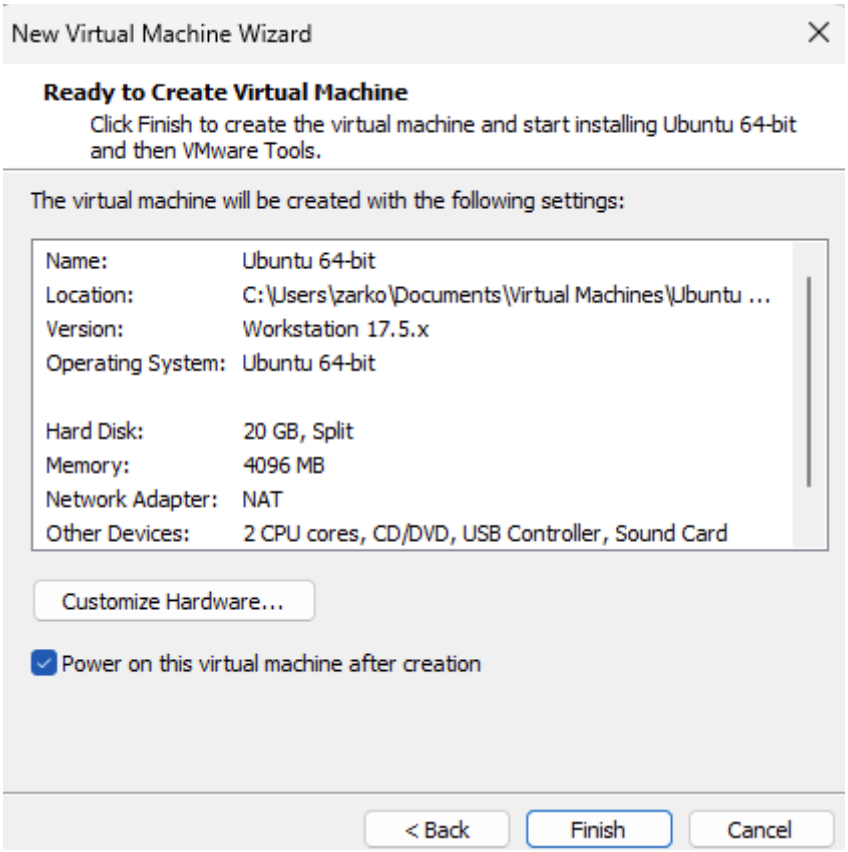


Slika 12 Dodela prostora na disku - VirtualBox



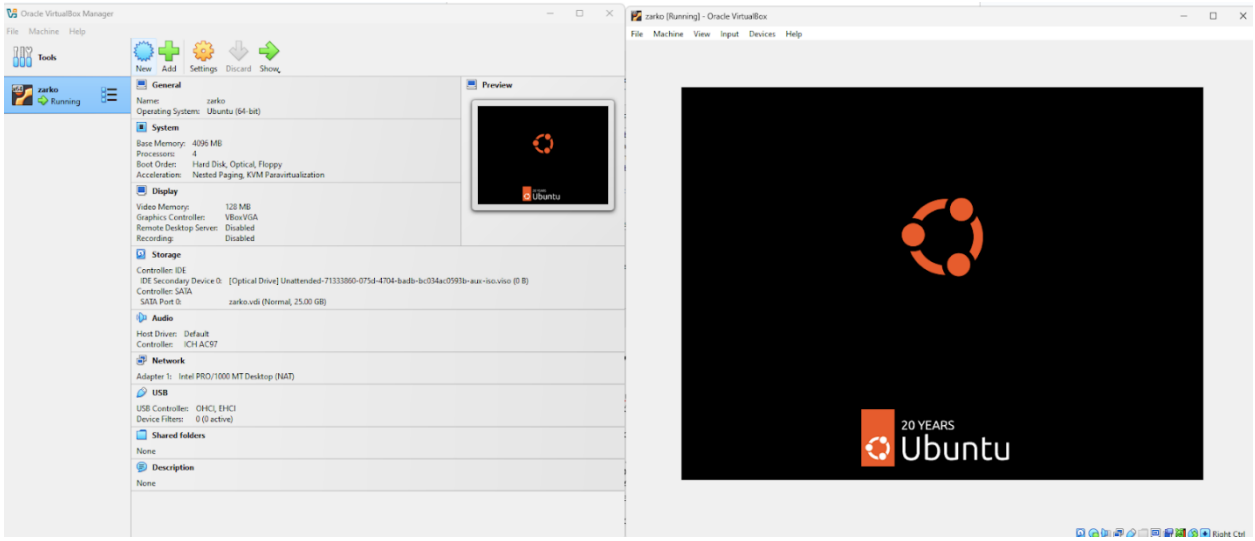
Slika 13 Pregled dodeljenih resursa - VirtualBox

VMWare Workstation će dodeliti automatski potrebne resurse, međutim, moguće je uraditi korekciju dodeljenih resursa klikom na Customize Hardware (Slika 14).

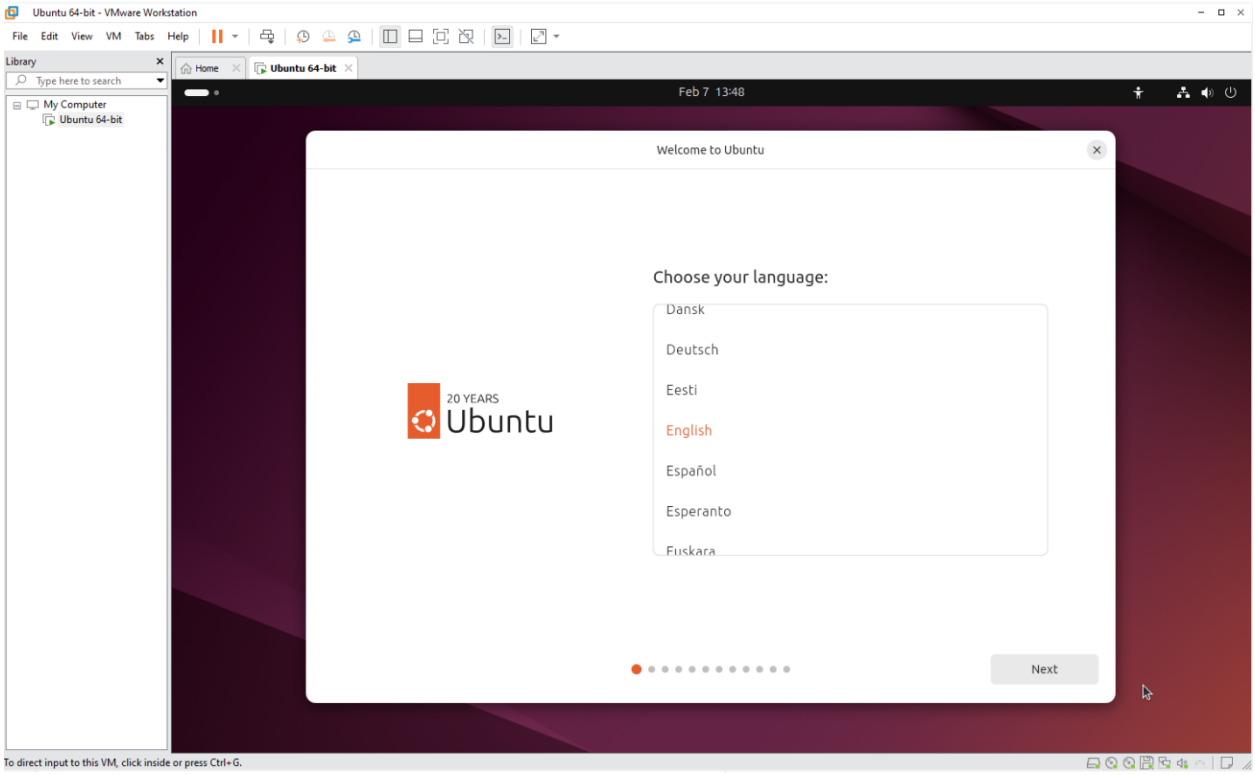


Slika 14 Pregled automatski dodeljenih resursa - VMWare Workstation Pro

Nakon svih podešavanja može se pokrenuti virtuelna mašina (Slika 15 i 16).



Slika 15 Pokretanje virtuelne mašine - VirtualBox



Slika 16 Pokretanje virtuelne mašine - VMWare Workstation Pro

Nakon pokretanja virtuelne mašine potrebno je pratiti instrukcije na ekranu i pomoću njih dovršiti instalaciju Ubuntu operativnog sistema na virtuelnoj mašini.

Virtuelizacija u oblaku: AWS EC2

Kao što je već pomenuto, **AWS EC2** (Elastic Compute Cloud) je web servis koji omogućava pravljenje i upravljanje virtuelnim mašinama (instancama) u AWS oblaku. EC2 instance su skalabilne, fleksibilne i plaćaju se po potrošnji (**On Demand**), što ih čini idealnim za različite scenarije: od jednostavnih web servera do kompleksnih distribuiranih sistema. Postoje i načini plaćanja kao što su dugoročni najam tzv. rezervisane instance **(Reserved Instances – RI)** po znatno povoljnim cenama u periodu od jedne ili tri godine, kao i **Spot instance** koje koriste neiskorišćene kapacitete u AWS infrastrukturi i omogućavaju korisnicima da ih koriste uz značajno nižu cenu u poređenju sa On Demand instancama.

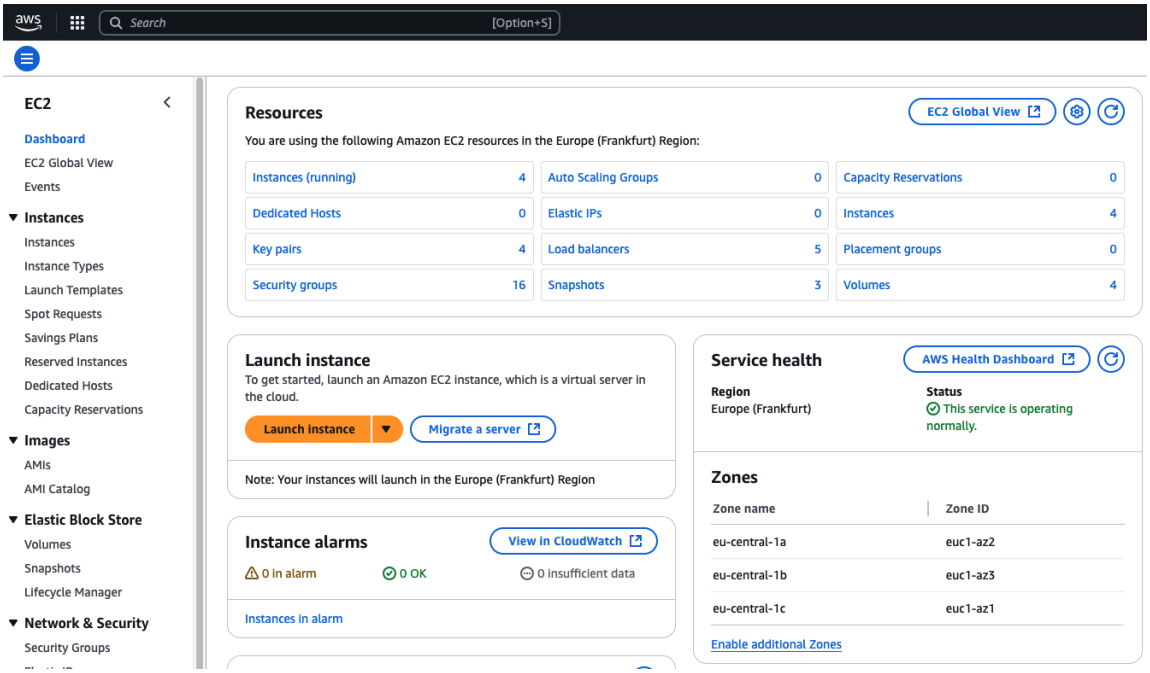
Spot instance mogu biti i do **90% jeftinije** od On-Demand instanci, ali AWS može da ih preuzme (ugasi) u bilo kom trenutku ako im je potreban kapacitet za druge korisnike koji plaćaju punu cenu.

Neke od karakteristika i prednosti EC2 instanci su:

1. **Skalabilnost** – Instance se mogu lako dodavati ili uklanjati prema potrebama aplikacije (horizontalno i vertikalno skaliranje).
2. **Plaćanje prema potrošnji** – Plaća se samo za vreme korišćenja instanci.
3. **Fleksibilnost** – Podrška za različite operativne sisteme, CPU i memorijske konfiguracije, i vrste skladišta.
4. **Automatizacija i orkestracija** – Integracija sa Auto Scaling-om, Elastic Load Balanser-om i drugim AWS servisima.
5. **Globalna dostupnost** – Instanciranje u različitim AWS regionima i Availability Zonama za visoku dostupnost.
6. **Visoka sigurnost** – Integracija sa IAM, VPC, Security Groups i KMS za upravljanje pristupom i enkripcijom.
7. **Integracija sa drugim AWS servisima** – Lako povezivanje sa RDS, S3, CloudWatch i drugim servisima.
8. **Podrška za različite tipove opterećenja** – Postoje instance optimizovane za procesorsku snagu, velike količine memorije, brzo skladište i GPU obradu.
9. **Automatsko bekapovanje** – Snapshot-i i AMI (Amazon Machine Images) omogućavaju brzu replikaciju i obnovu instanci u slučaju bilo kakvog otkazivanja.
10. **Hybrid i on-premises integracija** – Može se koristiti zajedno sa lokalnom infrastrukturom pomoću AWS Outposts i VPN konekcija.

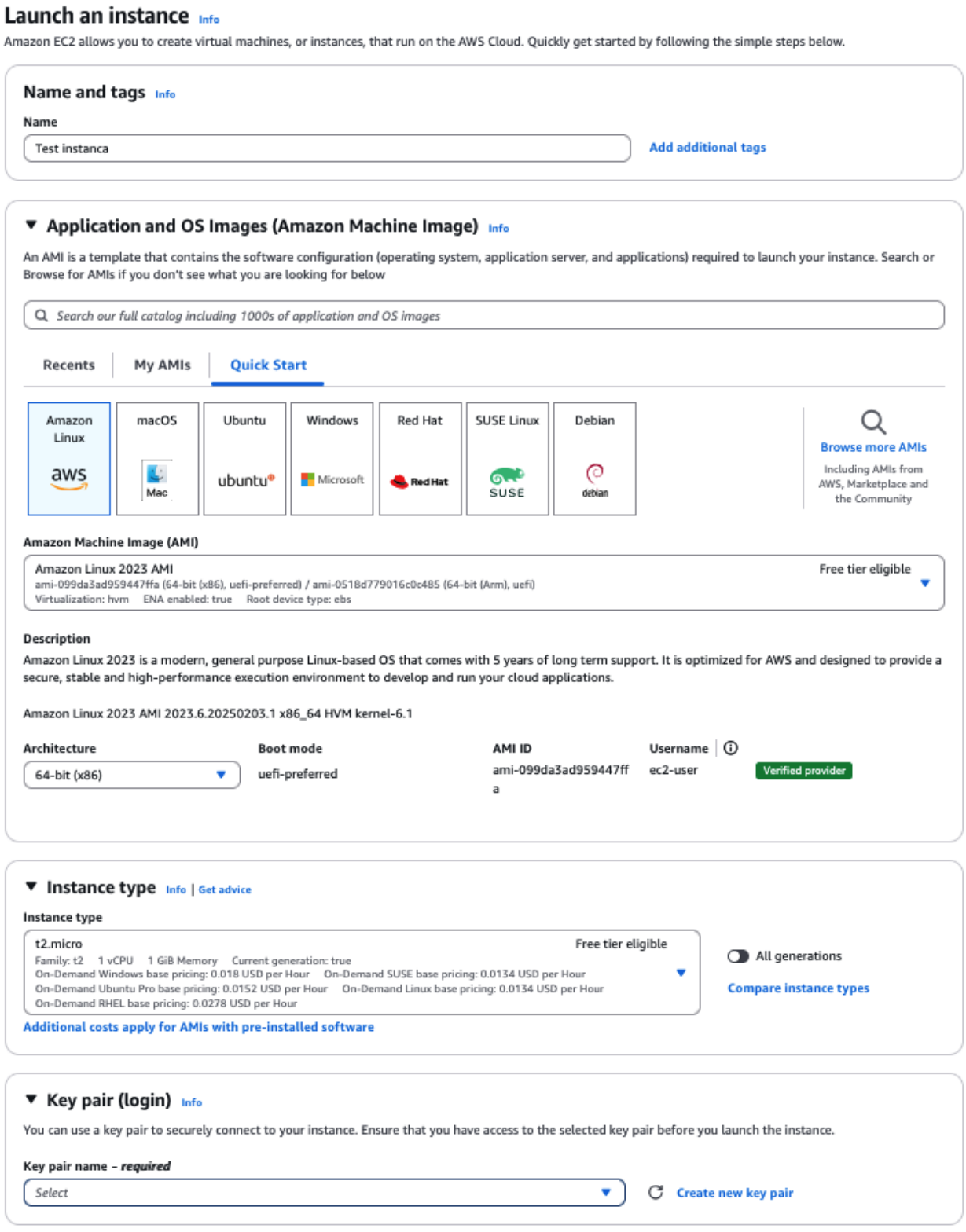
U narednim koracima sledi pokretanje EC2 instance koje podrazumeva da već postoji AWS nalog i da se njemu može pristupiti.

1. Prijavite se na AWS Console.
2. U pretrazi servisa ukucati **EC2** i otvoriti EC2 dashboard (kontrolnu tablu) (Slika 17).

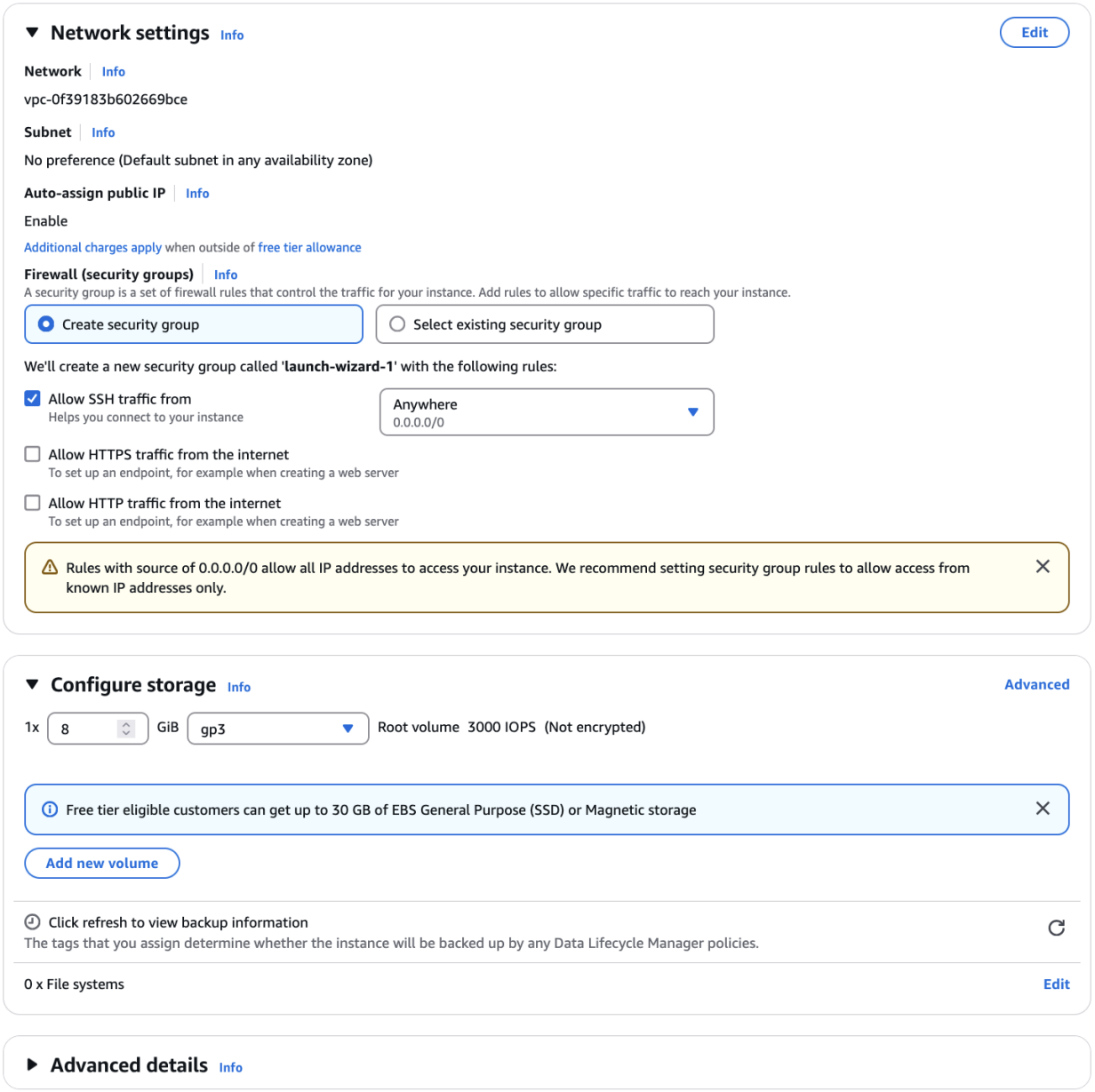


Slika 17 Prikaz EC2 kontrolne table

1. Zatim je potrebno kliknuti na Launch Instance dugme.
2. Nakon toga potrebno je popuniti sledeće podatke (Slika 18 i 19)
    0. Naziv instance – Test instanca ili slično
    1. Slika (image) operativnog sistema – Amazon Linux, Ubuntu ili neki drugi
    2. Arhitektura – 64 bit (x86) ili 64 bit (ARM)
    3. Tip instance – t2.micro ili neki drugi manji tip instance
    4. Key Pair – Potrebno je generisati novi ključ ili dodeliti postojeći radi kasnijeg povezivanja na EC2 instancu.
    5. Network settings – Podesiti Firewall, tj. security group da dozvoljava saobraćaj sa porta 22 klikom na dugme Allows SSH traffic from i u drop down meniju pored izabrati Anywhere ili My Ip (bezbednija opcija)
    6. U Configure storage sekciji se može ostaviti već podešena veličina od 8GB gp3 Storage.
    7. Nije potrebno vršiti izmene u Advanced details sekciji.



Slika 18 Prvi prikaz podešavanja EC2 instance



Slika 19 Drugi prikaz podešavanja EC2 instance

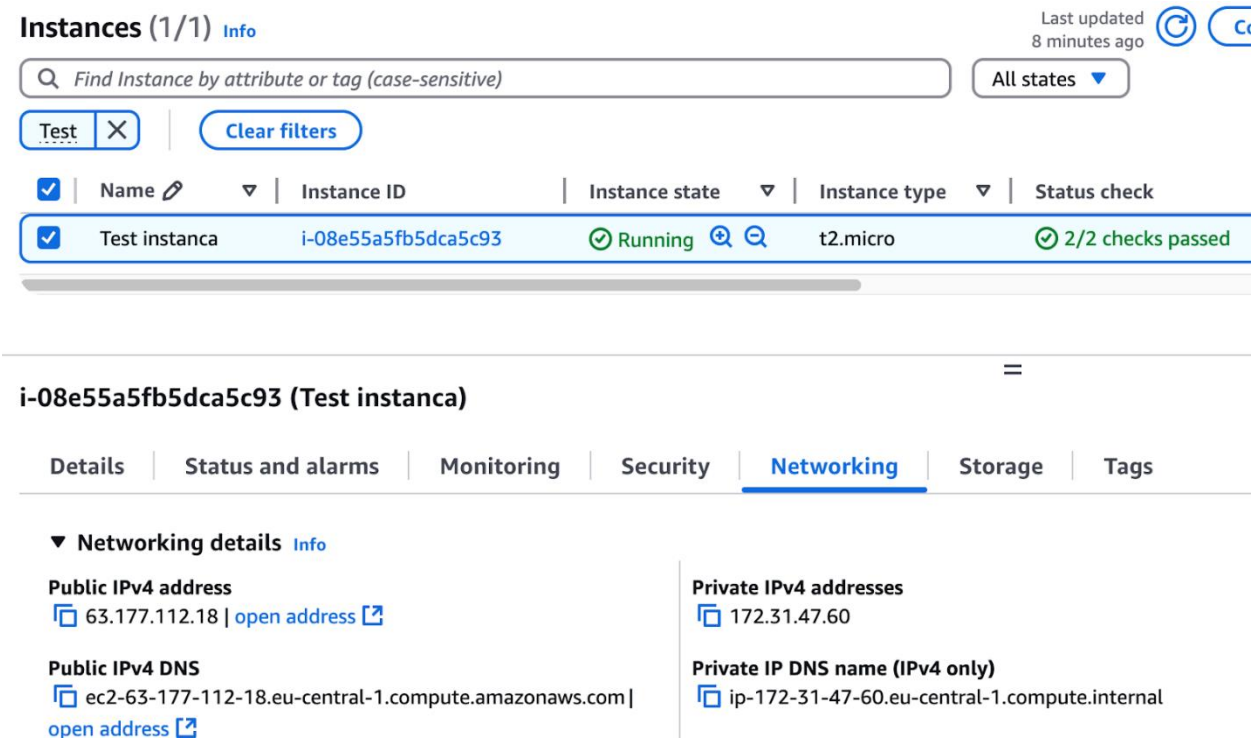
1. U desnom panelu Summary pod brojem instanci ostaviti 1 i kliknuti na dugme Launch Instance.

Nakon par minuta čekanja instanca ce kompletirati proveru statusa (status check) i preći iz statusa Initializing, što označava da se instanca priprema i podiže željeni image, u status 2/2 checks passed, što označava da su system status check i instance status check ispunjeni i instanca spremna za povezivanje.

Kada se odabere Test instanca, u okviru prozora ispod se mogu videti svi njeni detalji, kao što su detalji mreže i ip adresa, identifikator instance, status i mnogi drugi podaci. Na security tabu se mogu videti podešavanja vezana za firewall i koji saobraćaj je dozvoljen, na kojim portovima i sa kojih izvora.

Povezivanje na EC2 instancu se vrši kopiranjem javne IP adrese iz Networking taba (Slika 20) i korišćenjem programa za SSH konekciju uneti prikazanu IP adresu, port (22) i dodati ključ, tj. Key pair koji je u koracima pri pravljenju EC2 instance napravljen, preuzet na lokalnu mašinu i dodeljen toj EC2

instanci. Na Windows operativnim sistemima mogu se koristiti programi kao što je Putty, dok na MacOs i Linux sistemima se može direktno iz terminala pristupiti komandom za SSH. Primer SSH komande: ssh -i ~/Downloads/moj-kljuc.pem ec2-user@63.177.112.18



Slika 20 Prikaz kartice za umrežavanje i javne IP adrese

Ukoliko je napravljena Windows instanca, postoji mogućnost povezivanja pomoću RDP (Remote Desktop Protocol) na tu instancu.

Neki od scenarija upotrebe EC2 instanci:

- **Web serveri i hostovanje aplikacija** – Pokretanje web sajtova, API servisa, backend aplikacija.
- **Big Data i analitika** – Obrada velikih podataka sa EMR (Elastic MapReduce) ili Spark-om.
- **AI/ML i HPC (High-Performance Computing)** – Korišćenje GPU optimizovanih instanci za trening modela i simulacije.
- **IoT i edge computing** – Procesiranje IoT podataka pre slanja dalje.
- **Dev/Test okruženja** – Privremene instance za razvoj i testiranje aplikacija.
- **Gaming serveri** – Pokretanje online servera za igre sa niskom latencijom i visokim mrežnim protokom.
- **CI/CD i automatski build sistemi** – Korišćenje instanci za kompajliranje i testiranje koda.
- **VPN i sigurnosne aplikacije** – Postavljanje VPN servera i sigurnosnih rešenja.

AWS EC2 pripada free tier uslugama (besplatna upotreba) na AWS-u u određenim konfiguracijama i do godinu dana od datuma pravljenja AWS naloga. Ove instance su obeležene sa dodatnim natpisom „Free tier” i mogu se koristiti bez skrivenih troškova i pogodne su za eksperimentisanje.

Najbolje prakse za virtuelizaciju

1. **Optimizacija resursa**
    - **Ne preterivati sa dodelom resursa** – Dodela virtuelnim mašinama (VM) samo onoliko RAM-a i CPU-a koliko je neophodno. Prekoračenje može dovesti do usporenja host mašine. Primer: Ako host ima 8 GB RAM-a, ne dodeljivati više od 4 GB po VM.
    - **Praćenje iskorišćenosti resursa** – Koristiti alate kao što su **htop** (Linux) ili **Task Manager** (Windows) za praćenje potrošnje resursa. U oblaku (npr. za AWS EC2), koristiti **CloudWatch** za monitoring ili u EC2 Dashboard za izabranu instancu pratiti Monitoring tab.
2. **Bezbednost**
    - **Redovni backup-i kroz snapshot-e**:
        - **Lokalna virtuelizacija (VirtualBox/VMware)** – Snapshot-ovi omogućavaju brzo vraćanje na prethodno stanje u slučaju greške. Primer: Pre instalacije novog softvera, napraviti snapshot VM-a.
        - **Cloud virtuelizacija (AWS EC2)** – Koristiti **Amazon Machine Images (AMI)** za čuvanje backup-a instanci.
    - **Ograničavanje pristupa**:
        - **Lokalno** – Konfigurisati firewall na host mašini da blokira nepotrebne portove.
        - **Cloud** – Koristiti security grupe u AWS-u da se dozvoli pristup samo sa poznatih IP adresa. Primer: Otvoriti port 22 (SSH) samo za svoju IP adresu (kao iz prethodne vežbe).
    - **Ažuriranje softvera**:
        - Redovno primenjivanje security patcheve na VM-ovima i host mašini. Na cloud okruženjima samo raditi ažuriranje sistema, ostalo je u rukama cloud provajdera.

3. **Kontrola troškova i resursa**
    - **Gašenje neiskorišćenih instanci** – U **lokalnoj virtuelizaciji** isključiti VM kada se ne koristi da bi se oslobodili resursi. U **oblaku** koristiti **EC2 Instance Scheduler** za automatsko gašenje instanci tokom noći ili vikendom, dok za testne instance koristiti **t2.micro** (besplatni tier) ili minimalne veličine da podrže aplikaciju.
    - **Korišćenje „Reserved Instances“ (RI)** – AWS nudi do **72% popusta** za dugoročne rezervacije instanci (1 ili 3 godine).
    - **Korišćenje „Spot instances“** – Idealne su za scenarije gde aplikacija može podneti prekide (big data, batch processing, transcoding itd.). Korišćenjem Spot instanci može se uštedeti do 90% u odnosu na On Demand instance.
    - **Optimizacija skladišta** – Ukloniti nepotrebne snapshot-ove i image-ove kako ne bi došlo do dodatnih troškova skladištenja.

Zadatak

Pokrenuti Ubuntu server, instalirati i podesiti web server (Nginx/Apache) i hostovati statičnu web stranicu:

- Na virtuelnoj mašini (VirtualBox ili VMWare)
- Na AWS EC2 instanci.

5. Kontejneri i orkestracija kontejnera

Ciljevi ovog poglavlja su upoznavanje sa konceptom kontejnera, tehnologijama kontejnerizacije i fundamentima Docker-a.
Po savladavanju ovog poglavlja student će moći da:

- objasni koncept kontejnera i prednosti u odnosu na virtuelizaciju
- navede popularne tehnologije kontejnerizacije
- opiše arhitekturu Docker-a
- objasni koncept orkestracije kontejnera na primeru Kubernetes-a

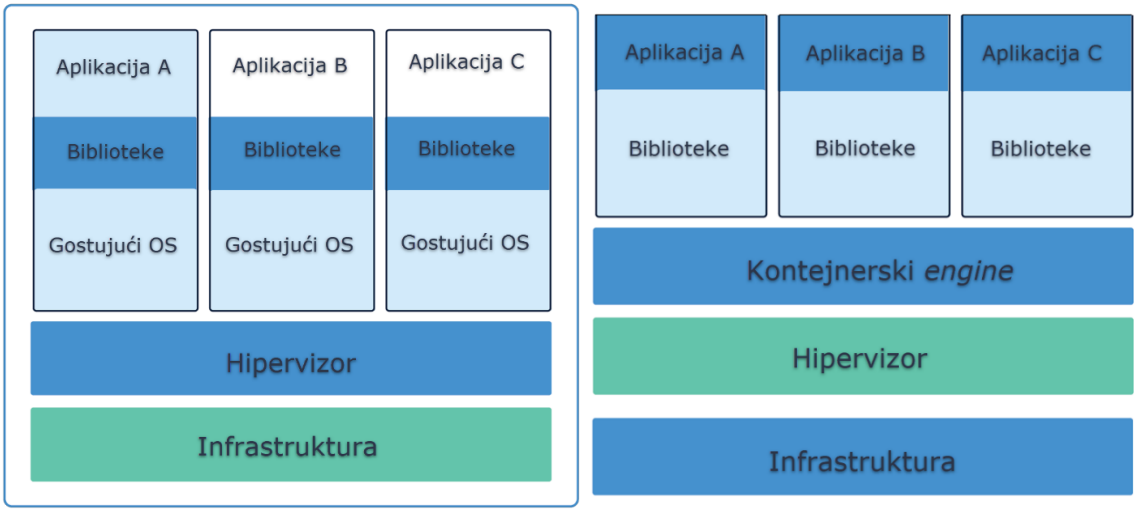
Student će umeti da:

- konfiguriše jednostavne Docker kontejnere
- objasni razlike između Podman-a i Docker-a
- konfiguriše različite oblike orkestracije kod AWS-a

U prethodnom poglavlju predstavljeni su fundamenti virtualizacije, kao tehnologije koja je revolucionarizovala korišćenje računara. Mogućnost pokretanja kompletnog izolovanog okruženja, koje se ponaša kao zaseban računar, lakoća kloniranja mašina i visok stepen iskorišćenja hardvera svojstva su koja su neposredno podstakla „klaudizaciju“ IT-a. Jednostavna migracija računarskih resursa po principu lift&shift omogućila je brojnim kompanijama da optimizuju svoje troškove i u dobroj meri izmeste servere iz svojih prostorija.

Sa druge strane, korišćenje kompletnih virtuelnih mašina pokazalo se kao dodatno opterećenje sa stanovišta razvoja i održavanja softvera, naročito u svetlu popularizacije servisno orijentisane arhitekture. Aplikacija koja se razvija i testira na mašinama programera u nekom trenutku se uvodi u produkciono okruženje na serverima dobavljača usluga (oblak ili konvencionalni deljeni hosting), gde je neophodno formirati identično okruženje da bi sve funkcionisalo. Ukoliko je u pitanju već formiran server, sa aplikacijama koje već funkcionišu, upitno je da li će moći da se formira okruženje za novu aplikaciju bez remećenja rada postojećih. Ako stare aplikacije zahtevaju jednu verziju programskog jezika i određenih biblioteka, a nova aplikacija koristi drugu, dati server možda neće moći da se koristi. Moguće rešenje je formiranje posebne virtuelne mašine sa okruženjem koje je podešeno za novu aplikaciju. Naravno, takva mašina će zahtevati posebne resurse, koje treba optimizovati. Za potrebe jedne aplikacije se instalira kompletan operativni sistem, koji je zatim neophodno i održavati. Nova mašina znači i novu površinu za eventualne napade itd.

Rešenje koje se ispostavilo kao praktično jeste kontejnerizacija aplikacija. Kontejnerizacija podrazumeva vid virtualizacije na nivou operativnog sistema, gde su kontejneri nezavisne celine u smislu aplikacija i pratećeg okruženja (*runtime*, programski jezici, biblioteke), ali je jezgro operativnog sistema zajedničko za sve kontejnere. U prethodnom poglavlju objašnjen je koncept chroot-a kao opcije za kreiranje kontejnera. Na slici Slika 21 dato je poređenje virtualizacije i kontejnerizacije.



Slika 21 Konvencionalna virtualizacija (levo) i kontejnerizacija (desno)

Kontejnerizacijom se ostvaruje:

- Visoka portabilnost. U kontejner se „pakuju“ aplikacija i sve ono što je aplikaciji potrebno za rad: biblioteke, kod, runtime. Kod chroot-a je za izolovanje aplikacije bilo potrebno ručno prekopirati sve potrebne fajlove i direktorijume, dok je kod modernih kontejnera proces pojednostavljen i automatizovan.
- Efikasnost. Veći broj kontejnera može se pokrenuti na istom hardveru, bez potrebe za instalacijom više operativnih sistema. Servis koji upravlja kontejnerizacijom troši mnogo manje resursa od hipervizora.
- Izolacija. Kontejneri mogu koristiti potpuno različite verzije programskih jezika; to neće dovesti ni do kakvog konflikta i veći broj kontejnera može raditi potpuno nezavisno ili sarađivati, u zavisnosti od arhitekture aplikacije. Kontejneri se u svrhu bezbednosti mogu konfigurisati sa minimumom pristupa osnovnom sistemu, kao i uzajamno, ponovo u zavisnosti od potreba aplikacije i bezbednosnih zahteva.
- Skalabilnost. Sami kontejneri nemaju stanje, tj. oni sadrže gradivne elemente aplikacije koja obrađuje i čuva podatke, ali sami podaci su u bazi, dok je kontejner nezavisan. To omogućava orkestraciju većeg broja kontejnera (ili jedinica koje sadrže više kontejnera) i skaliranje o potrebi.
- Konzistentnost. Eliminiše se sindrom „na mojoj mašini radi“ – kada u razvojnom okruženju softver radi, a u produkciji ne. Kada se razvija u kontejneru i taj kontejner iskopira u produkciono okruženje, gotovo sigurno[8] će raditi besprekorno.
- Brži razvoj i pogodnost za servisno orijentisanu arhitekturu. Čitav ciklus razvoja softvera je ubrzan i olakšan. Kontejneri su posebno pogodni za razvoj mikroservisa, gde je aplikacija izdeljena na male servise, koje je moguće nezavisno razvijati i održavati. Svakom takvom servisu može da odgovara jedan kontejner.

Postoji više tehnologija kontejnerizacije, kao što su: Podman, LXC, OpenVZ, CRI-O.

Podman je alat za upravljanje kontejnerima razvijen od strane Red Hat-a, koji kombinuje jednostavnost Docker-a sa naprednim bezbednosnim funkcijama. Za razliku od Docker-a, Podman ne zahteva daemon (pozadinski proces sa root privilegijama), što ga čini bezbednijim i lakšim za integraciju u okruženja sa

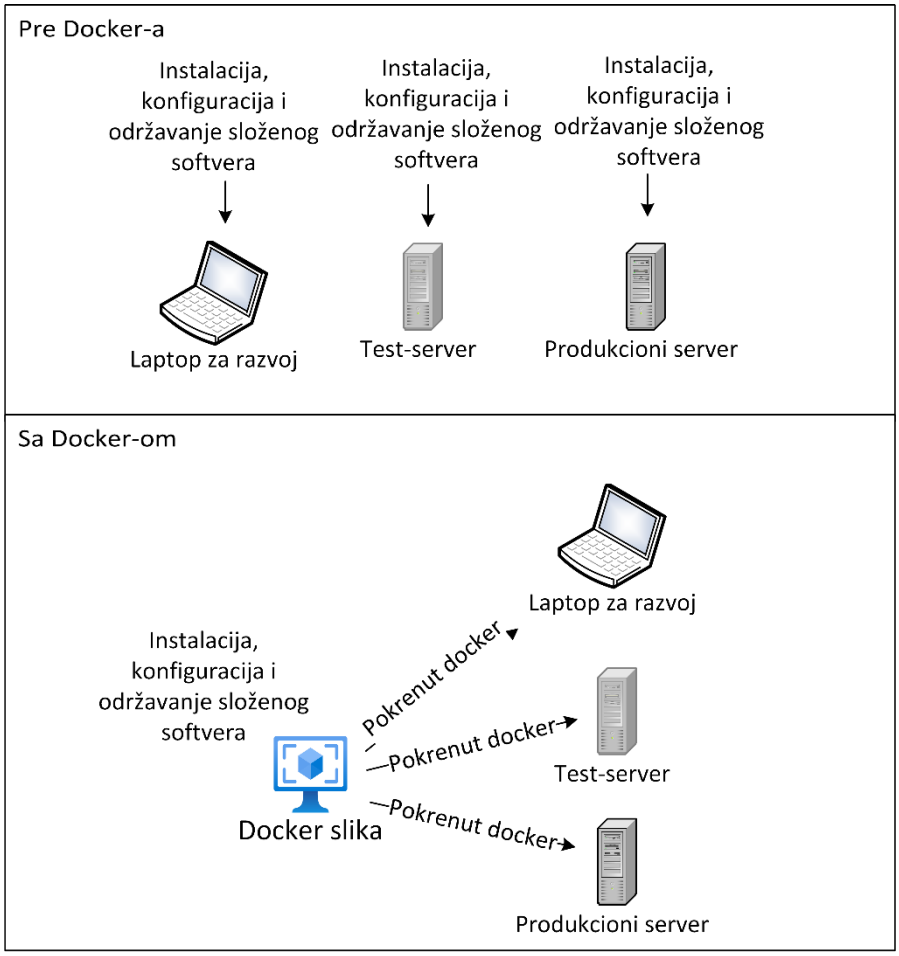
[8] Postoji i moderan sindrom „na mom kontejneru radi“, kada je neophodno istražiti uzroke nefunkcionalnosti kontejnera u produkcionom okruženju, ali su takvi slučajevi retki ukoliko se obrati pažnja pri kreiranju i preuzimanju kontejnera.

striktnim sigurnosnim polisama. Ključna prednost Podman-a je mogućnost pokretanja kontejnera u rootless modu – korisnici mogu da upravljaju kontejnerima bez administratorskih privilegija, smanjujući rizik od zloupotrebe. Iako je kompatibilan sa Docker komandama, Podman podržava i naprednije scenarije kao što su Kubernetes integracija i alate poput Buildah (za pravljenje image-a) i Skopeo (za upravljanje kontejner registrima). Ova fleksibilnost čini ga idealnim izborom kako za razvojne timove tako i za produkcijske sisteme koji traže balans između performansi i bezbednosti. Nedavna promena u polisi Docker-a koja je ograničila upotrebu u preduzećima dodatno je stimulisala proces usvajanja Podman-a.

Napomena: Kontejneri i aplikacije kao Docker i Podman najbolje rade na Linux operativnim sistemima, ali postoji podrška za Windows i MacOS. U slučaju korišćenja na Windows operativnim sistemima, potrebno je da WSL (Windows Subsystem for Linux) bude uključen.

U ovoj knjizi će se primeri uglavnom zasnivati na najpopularnijem rešenju, Docker-u.

Razvoj softvera se umnogome pojednostavio uz kontejnere. Kontejner je portabilna jedinica koja može raditi na bilo kom sistemu, npr. developeri mogu da koriste Windows sa Docker servisom za kreiranje kontejnera, a produkcioni server može biti Red Hat. Uz kontejnerizaciju se isti kontejner koristi za različite faze u razvoju (Slika 22).



Slika 22 Faze razvoja softvera pre docker-a (deo iznad) i sa docker-om (deo ispod)

5.1. Mehanizmi kontejnerizacije

Kako je već pomenuto, izolovana izvršna instanca se naziva kontejner. Slika *(image)* kontejnera je šablon na osnovu kojeg je kreiran kontejner. To je statička celina, koju je potrebno postaviti na server,

a zatim je aktivirati kroz *engine* za kontejnerizaciju *(container engine)*. Ovaj *engine* je ključni deo platforme za kontejnerizaciju.

Više kontejnera može deliti resurse, kao što je disk, i komunicirati međusobno putem standardne interprocesne komunikacije. Ovakve celine, koje su najčešće na istoj IP adresi, nazivaju se *pod*-ovi. U narednim sekcijama ovog poglavlja biće više reči o *pod*-ovima.

Ukoliko je potrebno, moguće je aktivirati više instanci jednog kontejnera. Takva potreba se javlja kod povećanog opterećenja, gde je potrebno više resursa za opsluživanje korisnika. Ovakve instance se nazivaju replike. Moguće je kreirati više instanci unapred, tako da budu spremne za izvršavanje. Ovakve grupe su poznate kao klasteri kontejnera.

Paket-menadžer

Kada se aplikacija kontejnerizuje, to znači da se vrši priprema podrške za aplikaciju i zavisnih paketa, tako da aplikacija može nesmetano da funkcioniše na nivou kontejnera. U kreiranju ovakvog nezavisnog paketa glavnu ulogu igra kontejnerski paket-menadžer. Paket-menadžer uključuje alate za kreiranje, označavanje (tagovanje) i predaju kontejnerskih slika (imidža) u registar kontejnera, kao i kreiranje i upravljanje samim slikama i njihovim zavisnostima. Često uključuju alat za verzionisanje i omogućavaju korišćenje šablona za konfiguracione fajlove koji definišu samu sliku.

5.2. Docker

U nastavku će principi kontejnerizacije biti objašnjeni na primeru Dockera. Druge tehnologije kontejnerizacije koriste iste osnovne principe, dok pojedini detalji mogu značajno varirati, sa ciljem poboljšanja bezbednosti, veće kontrole nad konfiguracijom i drugo.

Osnovni koncepti

Docker koristi Linux-jezgro. To mu omogućava da izvorno radi na Linux distribucijama, tako što neposredno koristi jezgro samog hosta. Za rad na Windowsu neophodna je virtualizacija tako što se koristi npr. Hyper-V i na njemu dalje pokreće Linux-jezgro, koje čini osnovu virtuelizacije Dockera. Sličan pristup je i kod MacOS-a[9]. Posledica se ogleda u boljim performansama Docker-a koji radi na Linux-u. Takođe, moguće je pokretati kontejnerizovane varijante Linux distribucija – na Linux hostu. Na primer, Ubuntu se, sa pratećim aplikacijama, može pokrenuti kao kontejner koji nema svoj sopstveni kernel, već „ispod haube“ koristi kernel Linux hosta.

Osnovne komponente Dockera su:

- Docker engine: Jezgro dockera, rukuje stvaranjem kontejnera i njihovim upravljanjem.
- Docker Image: Read-only šablon koji se koristi za stvaranje kontejnera. Sadrži kod i zavisne komponente.
- Dockerfile: Skript koji sadrži instrukcije za stvaranje Docker slike (image).

Slika (imidž) kontejnera je statička celina koja pokretanjem postaje kontejner kao što izvršni program statično stoji na disku dok se ne pokrene. Slike se postavljaju u tzv. registre, koji mogu biti javni ili privatni. Najpoznatiji javni registar slika jeste Docker Hub (hub.docker.com). U okviru ovog registra moguće je pronaći brojne slike sortirane prema oblastima i spremne za korišćenje. Slike tipskih servisa kao što su npr. memcashed, redis, nginx mogu se preuzeti sa Docker hub-a i iskoristiti za kreiranje sopstvenog softverskog ekosistema. Nakon instalacije i podešavanja na primer slike blog-platforme, ona se može sačuvati kao nova slika i postaviti javno na docker hub (ili čuvati u privatnom registru).

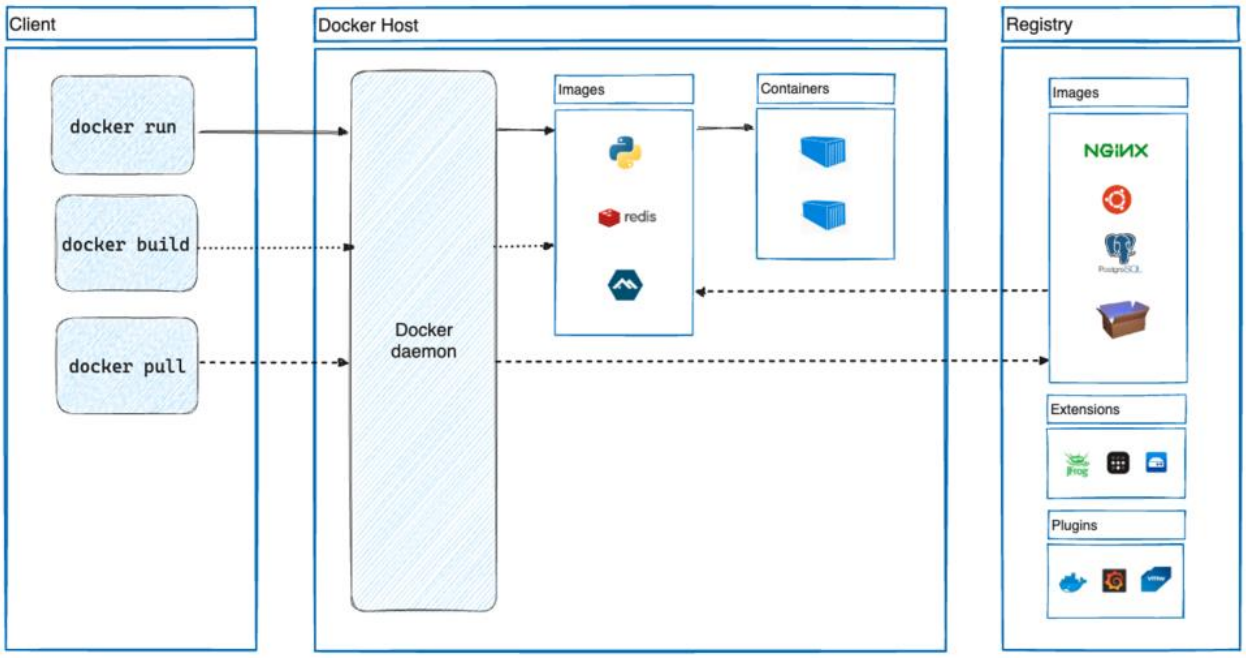
[9] Kod MacOS-a se koristi Hypervisor Framework, https://developer.apple.com/documentation/hypervisor

Naravno, pri preuzimanju slika i njihovom korišćenju, treba koristiti proverene slike i eventualno ih testirati. Na primer, može se dogoditi da je slika konfigurisana tako da se pored osnovne aplikacije instalira i kriptomajner. Kada korisnik rasporedi takvu sliku, odnosno pokrene kontejner, ovakav majner počinje da radi u pozadini i troši resurse servera u korist napadača, tj. autora slike.

Arhitektura Dockera

Docker funkcioniše po principu klijent-server. Klijent komunicira sa Docker serverom *(Docker daemon)* i navodi ga kako da izgradi, pokrene i distribuira Docker kontejnere. Klijent i server mogu biti na istom sistemu ili na posebnim sistemima. Komunikacija se odvija putem posebnog API-ja.

Na slici Slika 23 prikazana je osnovna arhitekura Dockera. Osnovni zadatak klijenta je da preuzme („povuče“, *pull*) sliku iz Docker registra i pokrene kontejner na Docker hostu. Odgovarajuće komande za izgradnju, preuzimanje i pokretanje slike jesu docker build, docker pull i docker run.



Slika 23 Arhitektura Dockera (preuzeto sa Wikipedije, Creative commons licenca)

Na Docker-hostu je pokrenut Docker servis *(daemon)*, na njemu su smeštene Docker slike i pokrenuti kontejneri.

Ovi kontejneri su podrazumevano izolovani. Pri pokretanju se definiše da li se i šta deli između kontejnera međusobno, odnosno između kontejnera i hosta.

Struktura Docker slike

Polazna tačka kontejnera je slika. Sama slika je izgrađena od više slojeva. Na ovaj način slika (i kasnije kontejner) je robusnija i fleksibilnija. Da bi se kreirala slika, potrebno je zadati određene komande u docker fajlu. Svaka komanda kreira jedan sloj docker slike.

Na primer[10]:

```
# syntax=docker/dockerfile:1
```

[10] Primer je preuzet iz oficijelne Docker dokumentacije https://docs.docker.com/

```
FROM ubuntu:22.04

LABEL org.opencontainers.image.authors="org@example.com"

COPY . /app

RUN make /app

RUN rm -r $HOME/.cache

CMD python /app/app.py
```

Prva instrukcija definiše verziju sintakse. Zatim se naredbom FROM definiše osnovna slika, u ovom slučaju to je Ubuntu. To znači da će se naredne naredbe zasnivati na okruženju Ubuntu. Naredba LABEL menja metapodatke slike i ne kreira novi sloj same slike. Komanda COPY kopira podatke iz aktivnog direktorijuma klijenta u direktorijum /app slike i na taj način kreira novi sloj. Naredba RUN izgrađuje aplikaciju koristeći komandu *make* i zapisuje rezultat u novi sloj.

Docker i skladište podataka

Podrazumevano, kontejner čuva podatke u upisivom sloju kontejnera. Nakon isključivanja kontejnera podaci se gube. Takođe, ovaj upisivi sloj je tesno povezan sa host-mašinom na kojoj radi kontejner i podatke je teško migrirati na drugo mesto. Upisivanje u ovaj sloj iziskuje poseban drajver skladišta. Time se unosi dodatno posredovanje i smanjuju performanse.

Postoje dva načina trajnog čuvanja podataka, tako da i nakon isključivanja kontejnera podaci ostaju dostupni:

- Docker particije *(volumes)*
- Vezano montiranje *(bind mounts)*

Još jedna mogućnost jeste čuvanje podataka u memoriji hosta. U slučaju Linux hosta reč je o tmpfs fajl-sistemu, koji faktički čuva podatke u memoriji (a ne na disku) i pogodan je kod velikih količina podataka koje se obrađuju, ali koje se ne čuvaju trajno.

Docker particije jesu najbolji način za čuvanje podataka i podrazumevaju poseban deo fajl-sistema hosta, koji ekskluzivno koristi Docker. Lokacija particija je /var/lib/docker/volumes

Vezano montiranje omogućava čuvanje bilo gde na fajl-sistemu hosta. I drugi procesi, ne samo Docker, mogu pristupati podacima čuvanim na ovaj način.

Svaki od oblika skladišta ima svoje namene. Docker particije se biraju kod deljenja podataka između više kontejnera, pružaju bolje performanse i bezbednost. Vezano montiranje je od značaja kada je potrebno da kontejner i host dele podatke, bilo da je u pitanju konfiguracija ili čitav projekat, koji se razvija na hostu i testira na kontejneru.

Docker i umrežavanje

Kod virtualnih mašina mreža se može konfigurisati na različite načine, od toga da virtualna mašina dobija interni opseg adresa i povezuje se na internet putem hosta (koji se ponaša kao ruter) – NAT povezivanje, do bridge mreže, gde je virtuelna mašina ravnopravno povezana kao i host i dobija isti opseg IP adresa.

Kod kontejnera, takođe su na raspolaganju različite mrežne funkcionalnosti realizovane kroz odgovarajuće drajvere. Podrazumevani je *bridge*. Pomoću ovog drajvera mogu se definisati segmenti

kojima pripadaju zadati kontejneri i takvi kontejneri su onda umreženi međusobno, a izolovani od drugih kontejnera (na istom hostu).

Drajver *overlay* omogućava povezivanje kontejnera, koji se nalaze na različitim hostovima, kroz šifrovanu vezu. Koristi se kod orkestriranih grupa kontejnera, pod Docker *Swarm*-om ili pod Kubernetes. (Više reči o orkestraciji biće kasnije u poglavlju.)

Drajver host omogućava povezivanje hosta i kontejnera i dostupnost kontejnera kroz sam host. Host i kontejner imaju istu IP adresu, a servisu kontejnera se pristupa preko porta. Faktički, servis kontejnera sluša na portu hosta. Ako je npr. pokrenut web-server unutar kontejnera i zadata mreža kao host, kada se pristupi IP adresi hosta i portu 80, otvara se web-strana kontejnera. Ovakav pristup je preporučljiv kada su potrebne maksimalne performanse mreže, pa se izbegava preusmeravanje portova.

Čest slučaj u praksi je upravo preusmeravanje portova pri kreiranju kontejnera. Na ovaj način se određeni port hosta preusmerava na port kontejnera.

Postoje još dve opcija kod izbora drajvera, vrlo specifične namene, te će samo biti pomenute. Drajver macvlan omogućava prikazivanje kontejnera kao fizičkih uređaja, koji imaju samo MAC adrese i mogu se raspoređivati u VLAN-ove. Na kraju, za potpunu izolaciju može se koristiti opcija network none pri pokretanju kontejnera, gde se kreira samo *loopback* uređaj i nema komunikacije ni sa hostom, ni sa drugim kontejnerima.

5.3. Orkestracija kontejnera

Upravljanje jednim kontejnerom ne zahteva posebne alate i može se izvoditi kroz pojedinačne komande. Kontejner se pokrene uz definisane portove koje izlaže i spreman je da obrađuje zahteve. Ukoliko iz nekog razloga „padne“, ponovo se pokrene iz komandne linije i „peške“ istražuje razlog pada itd. Ideja kontejnera je da se kreira portabilno izolovano okruženje, koje između ostalog omoućava skaliranje servisa. To podrazumeva veći broj kontejnera koji učestvuju u pružanju određenog servisa, pri čemu je važno automatizovati raspoređivanje (deployment), umrežavanje i uopšte upravljanje kontejnerima, a naročito u smislu dodavanja kontejnera za potrebe povećanog opterećenja. Svi ovi postupci spadaju u orkestraciju kontejnera. Na raspolaganju je više ovakvih alata, a u nastavku će biti reči o Docker Swarm-u i Kubernetes.

Docker Swarm

Docker Swarm je alat za orkestraciju, kojim je moguće kreirati klaster Docker nodova (poznat i kao Swarm) i na njemu rasporediti servise. Nije resursno zahtevan i omogućava postavljanje projekta za kratko vreme. U klasteru postoje nodovi-worker-i i nodovi-menadžeri. Menadžer je zadužen za samu organizaciju nodova-workera na kojima su pokrenuti servisi. Na primer, ako neki nod-worker postane nedostupan, nod-menadžer dodeljuje njegov posao drugom nodu-workeru. Uobičajeno je da postoji više nod-menadžera: takav sistem je robusniji. Menadžer-nodovi se dogovaraju o akcijama koristeći poseban algoritam (tzv. raft konsenzus).

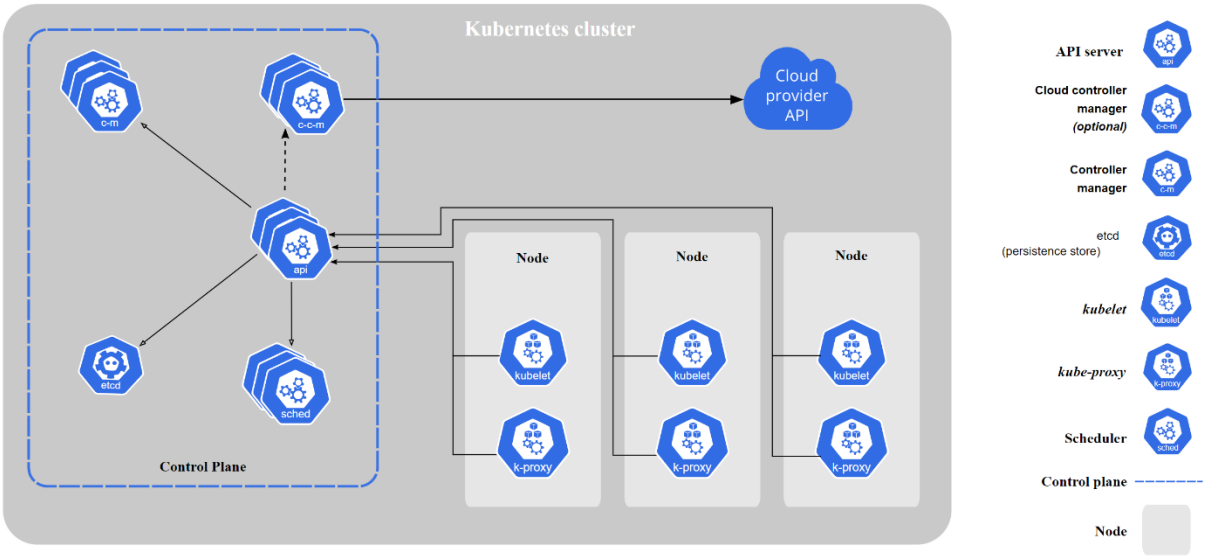
Kubernetes

Kubernetes, poznat i kao K8s, jeste najpopularniji alat za orkestraciju kontejnera. Njime se originalnim svojstvima Docker-a dodaju elementi kojim se osnažuju performanse i skalabilnost. (Treba naglasiti da se Kubernetes može koristiti i sa drugim tehnologijama za kontejnerizaciju. Pored Dockera, najčešći izbor je Podman. U ovoj knjizi će fokus biti na sprezi Kubernetes i Dockera.)

Neke od najvažnijih funkcija koje obezbeđuje Kubernetes su:

- Vidljivost servisa i raspodela opterećenja *(load balancing)*. Servis postaje izložen putem IP adrese ili simboličkog imena. Ukoliko je saobraćaj visokog intenziteta, K8s može da izvrši raspodelu opterećenja i donese stabilnost celokupnog sistema.
- Orkestracija skladišta. K8s omogućava montiranje proizvoljnog skladišta (lokalni disk, skladište u oblaku itd.).
- Automatizovano uvođenje i uklanjanje kontejnera. Ukoliko se pojavi potreba, novi kontejneri se aktiviraju ili se postojeći uklanjaju uz dodelu njihovih resursa aktivnim kontejnerima.
- Samo-isceljenje. K8s restartuje kontejnere koji su otkazali, zamenjuje kontejnere, ukida one koji ne reaguju na proveru ispravnosti i ne obznanjuje ih klijentima dok se ne poprave.
- Upravljanje tajnama i konfiguracijom. K8s omogućava čuvanje osetljivih podataka, kao što su lozinke, OAuth tokeni i SSH ključevi. Postavljanje i ažuriranje ključeva obavlja se bez ponovne izgradnje imidža kontejnera.
- Horizontalno skaliranje. Aplikacija se skalira naviše ili naniže putem komande, grafičkog interfejsa ili automatski, na osnovu opterećenja procesora.

Na slici 24 dat je prikaz arhitekture Kubernetes-a. U nastavku će biti predstavljeni osnovne komponente. Koncept je veoma sličan Docker Swarm-u.



Slika 24 Arhitektura Kubernetes-a (preuzeto sa kubernetes.io)

Klaster je skup sistema (bilo fizičkih, bilo virtuelnih) i drugih elemenata infrastrukture koje koristi Kubernetes za kontejnerizovane aplikacije.

Nod kontrolne ravni: sistem koji raspoređuje opterećenja, upravlja nodovima-radnicima, sprovodi politike pristupa i sinhronizuje promene u klasteru. Elementi noda su servisi kube-apiserver, etcd, kube-scheduler i kube-controller-manager i oni vode računa o raspodeli opterećenja, skaliranju, samooporavku, očuvanju stanja i delegaciji zadataka nodovima-radnicima (worker nodes).

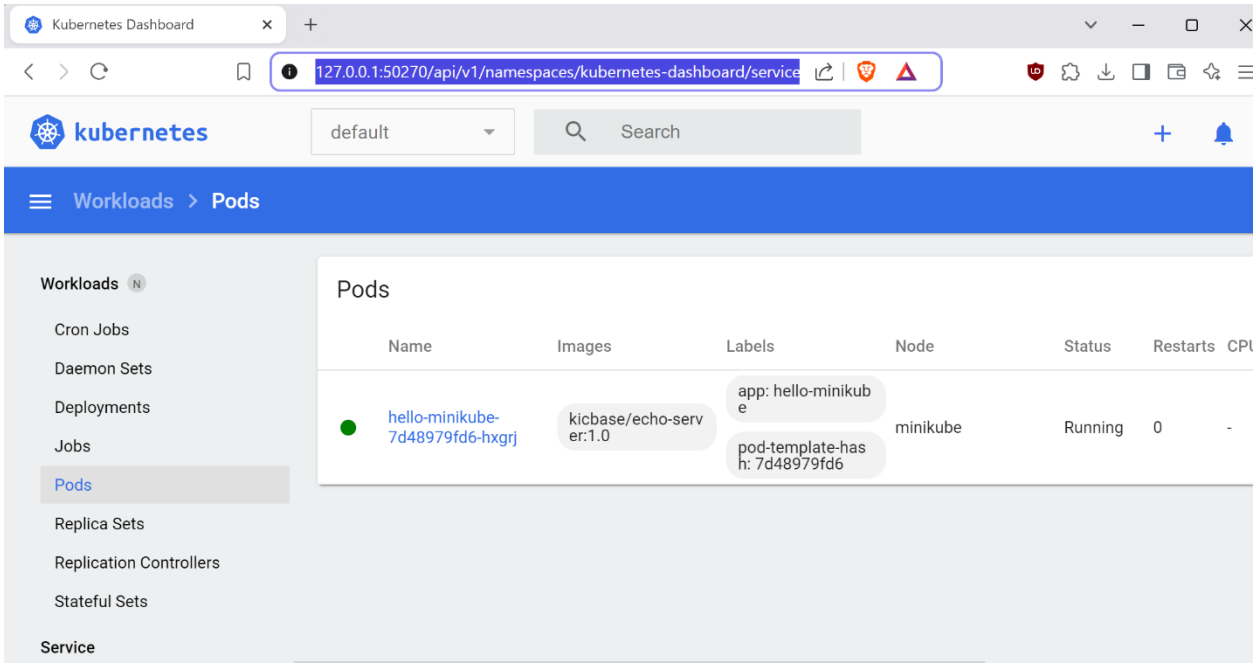
Nod-radnik izvršava zadatke dobijene od noda kontrolne ravni putem servisa kubelet. Ovaj servis je odgovoran i za proveru ispravnosti, ograničenje upotrebe resursa i izveštavanje kontrolnom nodu, tačnije servisu apiserver. Servis kube-proxy omogućava dostupnost za spoljašnje zahteve.

Prostor imena (namespace) omogućava logičku podelu klastera u particije, čime se omogućava izolacija više korisnika u sklopu jednog klastera.

Minikube

Minikube predstavlja projekat koji omogućava realizaciju i testiranje Kubernetes-a na lokalnom računaru, korišćenjem samo jedne virtualne mašine. Podržan je Docker, ali i HyperV i VirtualBox, kao podloga za virtualizaciju. Instalacijom i pokretanjem minikube-a, jedan čvor predstavlja i kontrolni i radni nod.

Na slici 25 prikazana je kontrolna tabla pokrenutog minikube-a. Više informacija o instalaciji i korišćenju minikubea može se pronaći na oficijelnoj stranici: https://minikube.sigs.k8s.io/docs/



Slika 25 Kontrolna tabla Minicube-a

Pitanja za proveru i obnavljanje znanja

1. U čemu je osnovna razlika između kontejnera i virtuelne mašine?
2. Na koji način se ostvaruje elastičnost servisa upotrebom kontejnera?
3. Da li za rad sa Dockerom mora da se koristi Linux?
4. Koje su to osnovne komponente Docker-a?
5. Na koji način se mogu trajno sačuvati podaci koji nastanu korišćenjem kontejnera?
6. Navedite tri funkcije Kuberenetes-a.
7. Koja je namena programa Minikube?

5.4. Praktični primeri kontejnerizacije

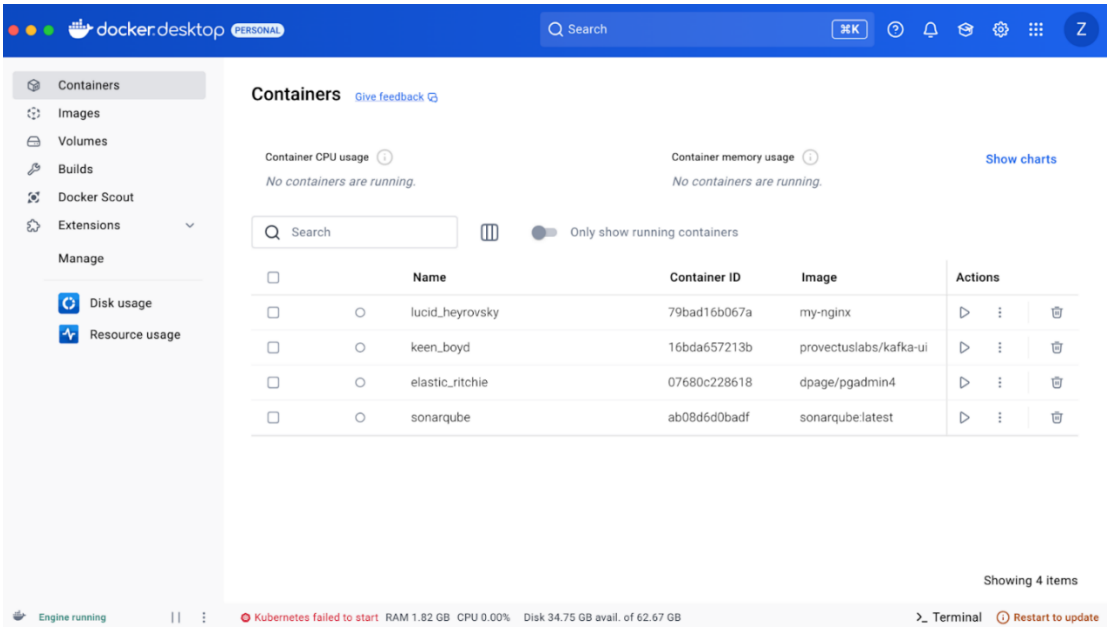
U predstojećim primerima biće korišćeni Docker i Podman.Instalacija Docker-a

Linux (Ubuntu/Debian):

```
sudo apt update && sudo apt install docker.io
sudo systemctl enable --now docker
```

Windows i macOS: Preuzima se Docker Desktop sa zvaničnog sajta.

Docker, pored CLI *(Command Line Interface)*, poseduje i Docker Desktop aplikaciju koja se može koristiti radi lakšeg pregleda i dodatnih podešavanja (Slika 26).



Slika 26 Docker Desktop aplikacija

Slika 27 Podešavanje dodele resursa Docker-u

U okviru aplikacije se mogu podesiti i resursi koje Docker može koristiti za svoje potrebe (Slika 27). Ovo je bitno kako Docker ne bi zauzeo previše resursa host-mašine, a ukoliko se pokreće nekoliko Docker image-a istovremeno, da bi se mogle proširiti količine dodeljenih resursa.

Korisne komande

- Lista aktivnih kontejnera i lista svih kontejnera na mašini:
  docker ps i **docker ps -a**

- Zaustaviti i pokrenuti kontejner
  docker stop my-nginx i **docker start my-nginx**
- Obrisati kontejner i obrisati image
  docker rm my-nginx i **docker rmi nginx**
- Pročitati logove kontejnera
  docker logs my-nginx
- Nasilno zaustaviti kontejner
  docker kill my-nginx

Primer 1: Pokretanje prvog kontejnera

U ovom primeru pokreće se kontejner, koji je zasnovan na Nginx image-u, podešava se da radi u pozadini, dodeljuje ime i mapiraju portovi (host:kontejner).

1. Preuzeti Nginx image
   docker pull nginx:latest
2. Pokretanje kontejnera
   docker run -d -p 8080:80 --name my-nginx nginx
   - **-d** - Detach mode (radi u pozadini).
   - **-p 8080:80** - Mapira port 80 kontejnera na port 8080 hosta.
3. Proveriti da li kontejner radi otvaranjem http://localhost:8080 u browseru.

Primer 2: Pravljenje Dockerfile-a za jednostavnu web stranicu

1. Potrebno je napraviti jednostavnu html web stranicu, na primer index.html sa sledećim sadržajem:

index.html > html

```
<!DOCTYPE html>
<html>
<body>
    <h1>Hello, World!</h1>
</body>
</html>
```

1. Zatim je potrebno napraviti Dockerfile u istom folderu kao i index.html stranicu. Dokcerfile će imati sledeći sadržaj:

Dockerfile > ...

```
FROM nginx:1.27.4-alpine
COPY index.html /usr/share/nginx/html
EXPOSE 80
CMD ["nginx-debug", "-g", "daemon off;"]
```

Objašnjenje sadržaja Dockerfile-a:

- FROM nginx:1.27.4-alpine – Određuje bazni image na kojem će se graditi kontejner. U ovom slučaju, koristi se zvanični nginx image verzije 1.27.4, zasnovan na Alpine Linux distribuciji. Alpine Linux je ultra laka Linux distribucija (oko 5MB) koja čini image lakšim i bržim za preuzimanje i pokretanje.
- COPY index.html /usr/share/nginx/html – Kopira fajl index.html iz lokalnog direktorijuma (gde se nalazi Dockerfile) u direktorijum /usr/share/nginx/html unutar

kontejnera. Direktorijum u koji kopira je /usr/share/nginx/html jer Nginx podrazumevano služi statičke fajlove iz ovog direktorijuma. Ako se ovde postavi index.html, on će biti dostupan na osnovnoj putanji (/) web servera.

- EXPOSE 80 – Obaveštava Docker da će kontejner slušati na portu 80. Ovo je port na kojem Nginx podrazumevano radi. Sam EXPOSE samo dokumentuje koji port se koristi, ali ga ne otvara. Da bi se omogućio pristup sa hosta, mora se mapirati port prilikom pokretanja kontejnera (npr. -p 8080:80).
- CMD ["nginx-debug", "-g", "daemon off;"] – Definiše komandu koja će se izvršiti kada se kontejner pokrene. U ovom slučaju, pokreće se nginx-debug (debug verzija Nginx-a, tj. log nivo podešen na debug) sa opcijom -g "daemon off;" koja sprečava Nginx da se pokrene u pozadini (daemon mode). Umesto toga, Nginx će raditi u prvom planu, što je neophodno za kontejnere jer Docker očekuje da glavni proces radi u prvom planu.

Sada je potrebno napraviti izgradnju *(build)* image-a na osnovu instrukcija koje su upisane u Dockerfile. Build se može izvršiti komandom:

docker build -t my-nginx (Slika 28).

```
zarkobogicevic@192 Vezba 1 % docker build -t my-nginx .
[+] Building 8.1s (8/8) FINISHED                                                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                              0.0s
 => => transferring dockerfile: 150B                                                                                              0.0s
 => [internal] load metadata for docker.io/library/nginx:1.27.4-alpine                                                            5.0s
 => [auth] library/nginx:pull token for registry-1.docker.io                                                                      0.0s
 => [internal] load .dockerignore                                                                                                 0.0s
 => => transferring context: 2B                                                                                                   0.0s
 => [internal] load build context                                                                                                 0.0s
 => => transferring context: 156B                                                                                                 0.0s
 => [1/2] FROM docker.io/library/nginx:1.27.4-alpine@sha256:b471bb609adc83f73c2d95148cf1bd683408739a3c09c0afc666ea2af0037aef      3.0s
 => => resolve docker.io/library/nginx:1.27.4-alpine@sha256:b471bb609adc83f73c2d95148cf1bd683408739a3c09c0afc666ea2af0037aef      0.0s
 => => sha256:df7b6963d5252424319dc265539db380d75bd4e40e112a83f8eaf09d1a9cb909 2.50kB / 2.50kB                                    0.0s
 => => sha256:525fa81b865c3bef8743265945df1859f8f0cb06a4f71aacbcb54f2fbd5a57d8 11.24kB / 11.24kB                                  0.0s
 => => sha256:7881d666ae1e7e1d2c967e53593819b6503e5d686dba95a109421cbfc51450da 1.78MB / 1.78MB                                    1.0s
 => => sha256:b471bb609adc83f73c2d95148cf1bd683408739a3c09c0afc666ea2af0037aef 10.36kB / 10.36kB                                  0.0s
 => => sha256:52f827f723504aa3325bb5a54247f0dc4b92bb72569525bc951532c4ef679bd4 3.99MB / 3.99MB                                    0.9s
 => => sha256:f256ac5fd451ec18939846ff7ed223dd1765a9f82fda67245d30931b74b78c2a 625B / 625B                                        0.6s
 => => sha256:a967eb288a8baec4bc92d2b26d5bb3aae030792d2204023b8917218ff4c1db80 952B / 952B                                        0.8s
 => => sha256:1232554cb43555bbe242a0624c2112090e184d8aaacced677bd08b6f97ae4a27 402B / 402B                                        1.0s
 => => extracting sha256:52f827f723504aa3325bb5a54247f0dc4b92bb72569525bc951532c4ef679bd4                                         0.1s
 => => sha256:08df7581e762b82e26bda5775a6b9836af9e5864099fb02f529ca40d0c8ec1cb 1.21kB / 1.21kB                                    1.4s
 => => extracting sha256:7881d666ae1e7e1d2c967e53593819b6503e5d686dba95a109421cbfc51450da                                         0.0s
 => => sha256:8b08ab47bab0bd7dbf941ba6ef384fb95ebe848fda99f72c41a059d86e825e5f 15.88MB / 15.88MB                                  2.7s
 => => sha256:0422927a03f94f34f1a992ea153be7d24ca83113863ff4c7d17dd64c2e6f562b 1.40kB / 1.40kB                                    1.3s
 => => extracting sha256:f256ac5fd451ec18939846ff7ed223dd1765a9f82fda67245d30931b74b78c2a                                         0.0s
 => => extracting sha256:a967eb288a8baec4bc92d2b26d5bb3aae030792d2204023b8917218ff4c1db80                                         0.0s
 => => extracting sha256:1232554cb43555bbe242a0624c2112090e184d8aaacced677bd08b6f97ae4a27                                         0.0s
 => => extracting sha256:08df7581e762b82e26bda5775a6b9836af9e5864099fb02f529ca40d0c8ec1cb                                         0.0s
 => => extracting sha256:0422927a03f94f34f1a992ea153be7d24ca83113863ff4c7d17dd64c2e6f562b                                         0.0s
 => => extracting sha256:8b08ab47bab0bd7dbf941ba6ef384fb95ebe848fda99f72c41a059d86e825e5f                                         0.2s
 => [2/2] COPY index.html /usr/share/nginx/html                                                                                   0.0s
 => exporting to image                                                                                                            0.0s
 => => exporting layers                                                                                                           0.0s
 => => writing image sha256:2aa19291064b3032f42c710f7429aa98902a1d6f9e71d9b92b2c54b5e9258376                                      0.0s
 => => naming to docker.io/library/my-nginx                                                                                       0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/ystyoxuyczmjtu0yf1qfvm7pd

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
zarkobogicevic@192 Vezba 1 %
```

Slika 28 Prikaz docker build procedure

Iz logova prikazanih na slici može se zaključiti da je build procedura uspešna; ukoliko dođe do problema, oni će biti prikazani u logovima. Nakon uspešnog build-a može se pokrenuti kontejner sledećom komandom (Slika 29):

docker run -d -p 8080:80 my-nginx

```
zarkobogicevic@192 Vezba 1 % docker run -d -p 8080:80 my-nginx
79bad16b067a5bb791993cfc05db2eb47d96e7066b59951fea744a6c1ebfbcaa
zarkobogicevic@192 Vezba 1 % docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED         STATUS         PORTS                   NAMES
79bad16b067a   my-nginx   "/docker-entrypoint.…"   3 seconds ago   Up 2 seconds   0.0.0.0:8080->80/tcp   lucid_heyrovsky
```

Slika 29 Prikaz pokretanja i provere da li kontejner radi

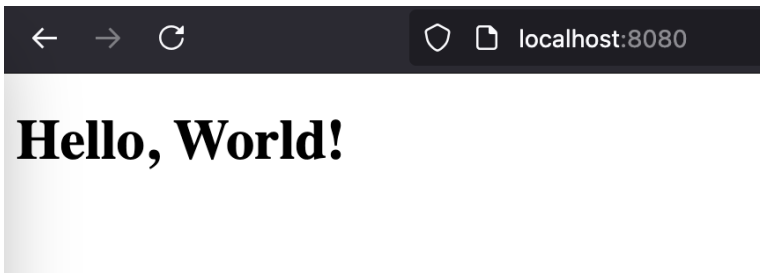
Kao što je prikazano na slici Slika 29, sa **docker ps** se može utvrditi da li je kontejner pokrenut, kao i njegovi parametri.

Komandom docker logs možemo proveriti logove unutar kontejnera. Nakon komande potrebno je uneti ime kontejnera (NAMES) ili identifikator (CONTAINER ID), kao što je prikazano na slici Slika 30. U ovom primeru ime kontejnera nije uneto pri pokretanju, pa je docker sam dodelio ime lucid_heyrovsky.

```
zarkobogicevic@192 Vezba 1 % docker logs 79bad16b067a
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2025/02/11 07:18:30 [notice] 1#1: using the "epoll" event method
2025/02/11 07:18:30 [notice] 1#1: nginx/1.27.4
2025/02/11 07:18:30 [notice] 1#1: built by gcc 14.2.0 (Alpine 14.2.0)
2025/02/11 07:18:30 [notice] 1#1: OS: Linux 6.10.14-linuxkit
2025/02/11 07:18:30 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2025/02/11 07:18:30 [notice] 1#1: start worker processes
2025/02/11 07:18:30 [notice] 1#1: start worker process 30
2025/02/11 07:18:30 [notice] 1#1: start worker process 31
2025/02/11 07:18:30 [notice] 1#1: start worker process 32
2025/02/11 07:18:30 [notice] 1#1: start worker process 33
2025/02/11 07:18:30 [notice] 1#1: start worker process 34
2025/02/11 07:18:30 [notice] 1#1: start worker process 35
2025/02/11 07:18:30 [notice] 1#1: start worker process 36
2025/02/11 07:18:30 [notice] 1#1: start worker process 37
2025/02/11 07:18:30 [notice] 1#1: start worker process 38
2025/02/11 07:18:30 [notice] 1#1: start worker process 39
```

Slika 30 Pregled docker logova za nginx kontejner

Pristupanjem adresi **http://localhost:8080** kroz web pretraživač možemo videti da se stranica index.html uspešno prikazuje od strane nginx kontejnera (Slika 31).



Slika 31 Pokrenut nginx kontejner prikazuje index.html stranicu

Primer 3: Docker Compose

Kreirajmo docker-compose.yml za aplikaciju sa Wordpress-om i MySQL bazom podataka.

Slika 32 Definicija docker-compose koja pokreće Wordpress i MySQL

U okviru prikazanog docker-compose fajla (Slika 32) definisano je sledeće:

- **Version: '3.8'** – Određuje verziju Docker Compose sintakse. Verzija 3.8 je stabilna i podržava sve potrebne funkcionalnosti.
- **image: wordpress:latest** – Koristi zvanični WordPress image (najnovija verzija).
- **container_name: my-wordpress** – Daje ime kontejneru (my-wordpress).
- **restart: always** – Osigurava da se kontejner automatski restartuje u slučaju pada.
- **ports: "8080:80"** – Mapira port 80 unutar kontejnera (gde WordPress radi) na port 8080 hosta.
- **Environment** – Postavlja promenljive okruženja za povezivanje na bazu podataka:
  - WORDPRESS_DB_HOST - Ime MySQL servisa (db).
  - WORDPRESS - DB_USER, DB_PASSWORD, DB_NAME - Korisnički podaci za bazu.
- **volumes: wordpress_data:/var/www/html** – Čuva WordPress fajlove (teme, pluginove itd.) u Docker volumenu wordpress_data.
- **depends_on: db** – Osigurava da se WordPress pokreće tek nakon što je MySQL baza spremna.
- **image: mysql:9.2.0** – Koristi MySQL verziju 9.2.0
- **container_name: my-mysql** – Daje ime kontejneru (my-mysql).
- **restart: always** – Automatski restartuje kontejner u slučaju pada.
- **Environment** – Postavlja promenljive okruženja za konfiguraciju baze:
  - MYSQL_ROOT_PASSWORD - Lozinka za root korisnika.

- MYSQL_DATABASE - Ime baze (wordpress).
- MYSQL_USER, MYSQL_PASSWORD – Korisnički podaci za WordPress.

- **volumes: db_data:/var/lib/mysql**: Čuva podatke baze u Docker skladištenju db_data.

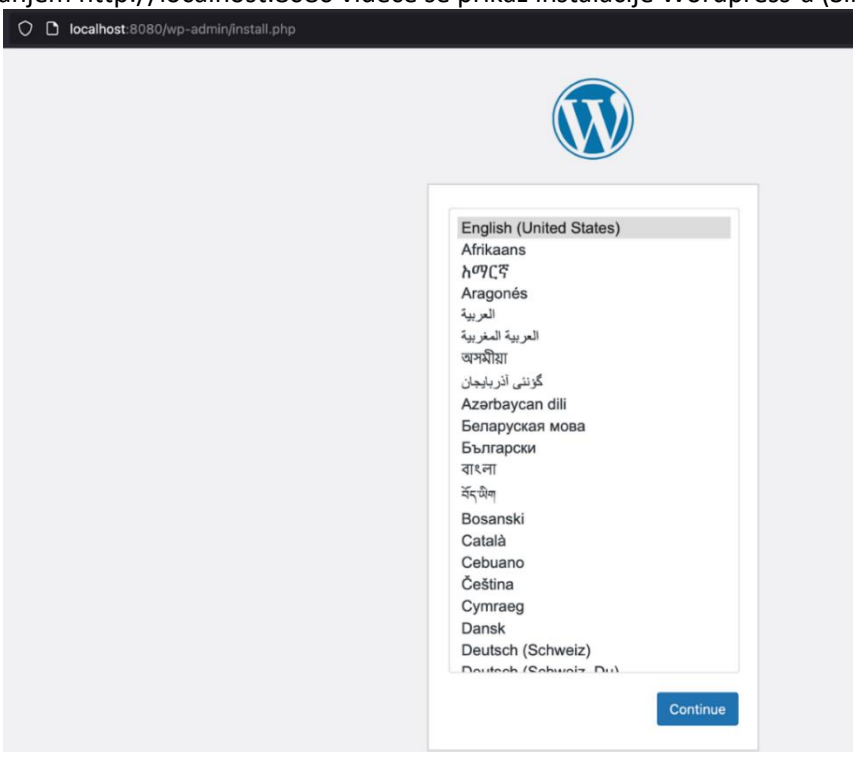
Nakon pravljenja fajla sledi pokretanje pomoću komande:

docker-compose up -d (Slika 33)

```
zarkobogicevic@192 Vezba 1 % docker-compose up
WARN[0000] /Users/zarkobogicevic/Documents/Vezba 1/docker-compose.yaml: the attribute `version` is obsolete, it will be ignored, please remove it to avoid poten
tial confusion
[+] Running 11/11
 ✔ db Pulled                                                                                                                                          33.3s
   ✔ 63067de277cc Pull complete                                                                                                                       21.3s
   ✔ 9aa9fba22b5d Pull complete                                                                                                                       21.3s
   ✔ ea43bb80aedd Pull complete                                                                                                                       21.3s
   ✔ ec26acc907bb Pull complete                                                                                                                       21.4s
   ✔ a59d170d9107 Pull complete                                                                                                                       21.4s
   ✔ 2734671063c0 Pull complete                                                                                                                       21.5s
   ✔ 4244cf9e5e42 Pull complete                                                                                                                       22.0s
   ✔ 8db433bbf7dd Pull complete                                                                                                                       22.0s
   ✔ d27d767a60a7 Pull complete                                                                                                                       30.9s
   ✔ ba4d9af659b4 Pull complete                                                                                                                       30.9s
[+] Running 5/5
 ✔ Network vezba1_default          Created                                                                                                            0.0s
 ✔ Volume "vezba1_wordpress_data"  Created                                                                                                            0.0s
 ✔ Volume "vezba1_db_data"         Created                                                                                                            0.0s
 ✔ Container my-mysql              Created                                                                                                            0.1s
 ✔ Container my-wordpress          Created                                                                                                            0.1s
Attaching to my-mysql, my-wordpress
my-mysql       | 2025-02-11 07:49:11+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 9.2.0-1.el9 started.
my-wordpress   | WordPress not found in /var/www/html - copying now...
my-mysql       | 2025-02-11 07:49:11+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
my-mysql       | 2025-02-11 07:49:11+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 9.2.0-1.el9 started.
my-mysql       | 2025-02-11 07:49:12+00:00 [Note] [Entrypoint]: Initializing database files
my-mysql       | 2025-02-11T07:49:12.042290Z 0 [System] [MY-015017] [Server] MySQL Server Initialization - start.
my-mysql       | 2025-02-11T07:49:12.042887Z 0 [System] [MY-013169] [Server] /usr/sbin/mysqld (mysqld 9.2.0) initializing of server in progress as process 80
my-mysql       | 2025-02-11T07:49:12.046429Z 1 [System] [MY-013576] [InnoDB] InnoDB initialization has started.
my-mysql       | 2025-02-11T07:49:12.198350Z 1 [System] [MY-013577] [InnoDB] InnoDB initialization has ended.
my-wordpress   | Complete! WordPress has been successfully copied to /var/www/html
my-wordpress   | No 'wp-config.php' found in /var/www/html, but 'WORDPRESS_...' variables supplied; copying 'wp-config-docker.php' (WORDPRESS_DB_HOST WORDPRESS_D
B_NAME WORDPRESS_DB_PASSWORD WORDPRESS_DB_USER)
my-wordpress   | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.18.0.3. Set the 'ServerName' directive global
ly to suppress this message
my-wordpress   | AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.18.0.3. Set the 'ServerName' directive global
ly to suppress this message
my-wordpress   | [Tue Feb 11 07:49:12.671469 2025] [mpm_prefork:notice] [pid 1] AH00163: Apache/2.4.57 (Debian) PHP/8.2.16 configured -- resuming normal operatio
ns
my-wordpress   | [Tue Feb 11 07:49:12.671500 2025] [core:notice] [pid 1] AH00094: Command line: 'apache2 -D FOREGROUND'
my-mysql       | 2025-02-11T07:49:12.744986Z 6 [Warning] [MY-010453] [Server] root@localhost is created with an empty password ! Please consider switching off th
e --initialize-insecure option.
my-mysql       | 2025-02-11T07:49:14.093686Z 0 [System] [MY-015018] [Server] MySQL Server Initialization - end.
my-mysql       | 2025-02-11 07:49:14+00:00 [Note] [Entrypoint]: Database files initialized
my-mysql       | 2025-02-11 07:49:14+00:00 [Note] [Entrypoint]: Starting temporary server
```

Slika 33 Prikaz pokretanja docker kontejnera u okviru docker-compose

Pristupanjem http://localhost:8080 videće se prikaz instalacije Wordpress-a (Slika 34).



Slika 34 Učitavanje Wordpress-a u okviru docker-compose

Komandom docker ps možemo se uveriti da je docker u pozadini pokrenuo dva nezavisna kontejnera za MySQL i Wordpress sa opisanim konfiguracijama (Slika 35).

```
zarkobogicevic@192 Vezba 1 % docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED          STATUS          PORTS                    NAMES
90e6415b9cbe   wordpress:latest   "docker-entrypoint.s…"   15 minutes ago   Up 15 minutes   0.0.0.0:8080->80/tcp     my-wordpress
3e75ad263409   mysql:9.2.0        "docker-entrypoint.s…"   15 minutes ago   Up 15 minutes   3306/tcp, 33060/tcp      my-mysql
```

Slika 35 Prikaz kontejnera iz docker-compose konfiguracije

Nakon toga možemo zaustaviti sve korišćenjem sledeće komande (Slika 36):

docker-compose down

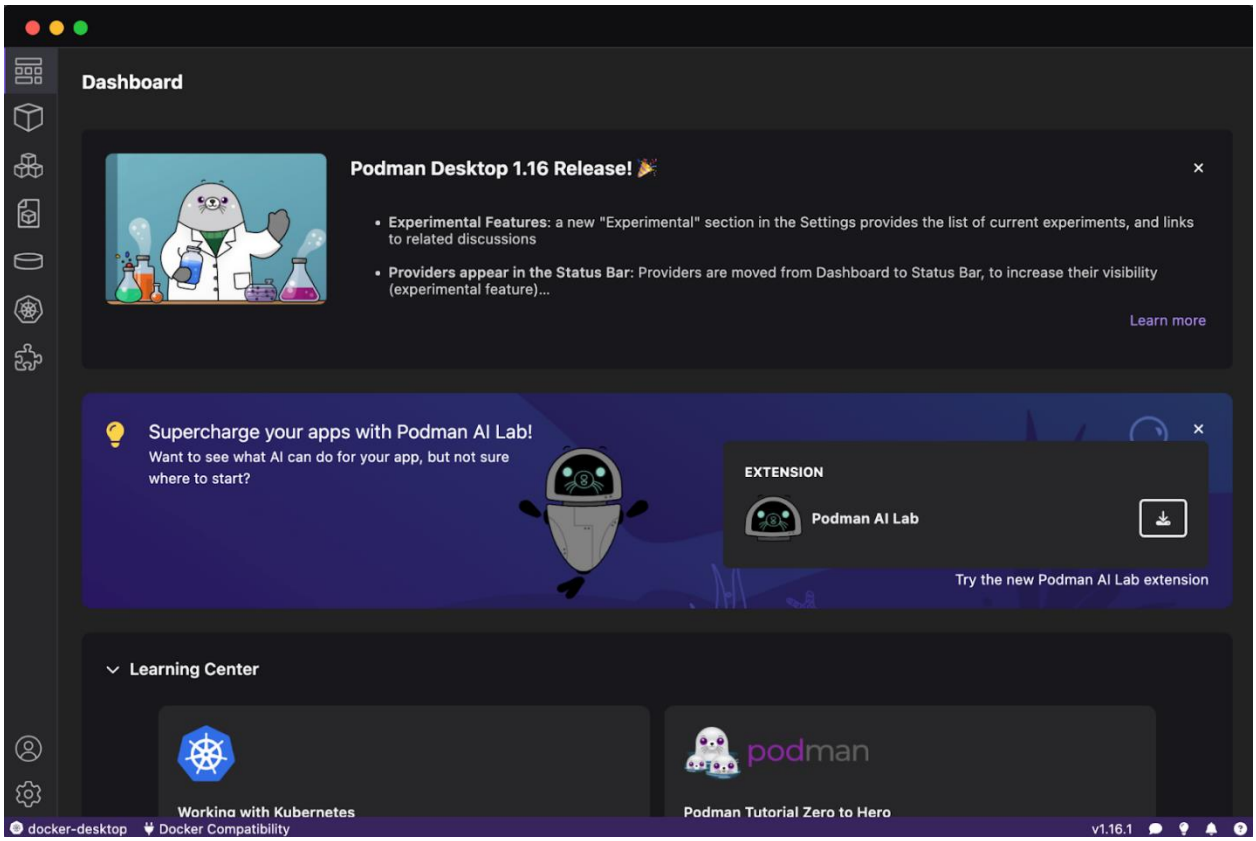
```
zarkobogicevic@192 Vezba 1 % docker-compose down
WARN[0000] /Users/zarkobogicevic/Documents/Vezba 1/docker-compose.yaml: the attribute `version` is obsolete, it will be ignored, please
 remove it to avoid potential confusion
[+] Running 3/3
 ✔ Container my-wordpress  Removed                                                                                    1.1s
 ✔ Container my-mysql      Removed                                                                                    1.5s
 ✔ Network vezba1_default  Removed                                                                                    0.1s
```

Slika 36 Zaustavljanje svih kontejnera i komponenti definisanih u okviru docker-compose

Instalacija Podman-a

- **Ubuntu/Debian**:
  sudo apt install podman
- **Windows i macOS**: Preuzima se Podman sa zvaničnog sajta.

Podman takođe poseduje Podman Desktop aplikaciju (Slika 37).



Slika 37 Podman Desktop aplikacija

Korisne komande

1. Lista aktivnih kontejnera i lista svih kontejnera na mašini:
   podman ps i **podman ps -a**
2. Zaustaviti i pokrenuti kontejner
   podman stop my-nginx i **podman start my-nginx**
3. Obrisati kontejner i obrisati image
   podman rm my-nginx i **podman rmi nginx**
4. Pročitati logove kontejnera
   podman logs my-nginx
5. Nasilno zaustaviti kontejner
   podman kill my-nginx

Primer: Pokretanje kontejnera u Podman-u

Pre pokretanja prvog kontejnera potrebno je uraditi inicijalizaciju podman mašine komandom:

podman machine init

Time se preuzima Linux okruženje u okviru kog se izvršavaju kontejneri (Slika 38).

```
podman machine init
Looking up Podman Machine image at quay.io/podman/machine-os:5.3 to create VM
Getting image source signatures
Copying blob cc337dac08f3 done   |
Copying config 44136fa355 done   |
Writing manifest to image destination
cc337dac08f317e9141e55f324d3af74f5d72f39c797f8d1f396948854fa655d
Extracting compressed file: podman-machine-default-arm64.raw: done
Machine init complete
To start your machine run:

        podman machine start
```

Slika 38 Init Podman mašine

Zatim je potrebno da se podman mašina startuje komandom (Slika 39):

podman machine start

```
podman machine start
Starting machine "podman-machine-default"

This machine is currently configured in rootless mode. If your containers
require root permissions (e.g. ports < 1024), or if you run into compatibility
issues with non-podman clients, you can switch using the following command:

        podman machine set --rootful

API forwarding listening on: /var/folders/dd/_ffb9mw97_z6yynd82tfm4fc0000gn/T/podman/podman-machine-default-api.sock

The system helper service is not installed; the default Docker API socket
address can't be used by podman. If you would like to install it, run the following commands:

        sudo /opt/homebrew/Cellar/podman/5.3.2_1/bin/podman-mac-helper install
        podman machine stop; podman machine start

You can still connect Docker API clients by setting DOCKER_HOST using the
following command in your terminal session:

        export DOCKER_HOST='unix:///var/folders/dd/_ffb9mw97_z6yynd82tfm4fc0000gn/T/podman/podman-machine-default-api.sock'

Machine "podman-machine-default" started successfully
```

Slika 39 Pokretanje Podman mašine

Pokreće se Nginx (ista sintaksa kao Docker) (slika 40):
podman run -d -p 8080:80 --name my-nginx nginx

Slika 40 Pokretanje nginx kontejnera pomoću Podman-a

Glavne karakteristike

- Podman ne zahteva daemon (*daemonless*).
- Podman podržava **rootless** kontejnere (bez sudo privilegija).
- Komande su identične Docker-u (podman build, podman-compose itd.).

Uporedna analiza: Docker vs. Podman

Poređenje dve platforme za kontejnerizaciju dato je u tabeli 3.

Tabela 3 Uporedni prikaz svojstava Dockera i Podmana

Svojstvo	**Docker**	**Podman**
Arhitektura	Zahteva demon (pozadinski proces)	Bez demona (daemonless)
Root privilegije	Često zahteva sudo	Može raditi bez sudo (rootless)
CLI kompatibilnost	Vlasnički CLI	Docker-kompatibilan CLI
Bezbednost	Demon radi sa root privilegijama	Bezbedniji (nema demona sa root pravima)
Kubernetes podrška	Docker Swarm	Integracija sa Kubernetes
Build alat	Docker Build	Buildah (odvojen alat)
Registri	Docker Hub (podrazumevani)	Podman podržava više registara
Volume management	Docker Volumes	Podman Volumes
Networking	Docker Network	Podman Network
Orkestracija	Docker Compose	Podman Compose
Performanse	Brz, ali daemon može usporiti	Brz (nema daemona)
Zajednica i podrška	Velika zajednica, dobra dokumentacija	Manja zajednica, ali postoji podrška od Red Hat-a
Upotreba u produkciji	Široko korišćen	Sve više korišćen u enterprise okruženjima

Zadatak

1. Kreirajte Dockerfile za Node.js aplikaciju koja služi statički HTML fajl ili aplikaciju po želji.
2. Pokrenite PostgreSQL kontejner i povežite ga sa aplikacijom koristeći docker-compose.yml.
3. Eksperimentišite sa Podman-om: pokrenite isti kontejner koristeći Podman.

Kontejneri i oblak

AWS nudi više usluga za upravljanje kontejnerima, omogućavajući timovima da koriste Docker i Kubernetes u potpuno upravljanim okruženjima. Evo pregleda ključnih AWS usluga:

1. Amazon ECS (Elastic Container Service)

ECS je AWS-ov upravljani *(managed)* servis za pokretanje kontejnera. On upravlja klasterima EC2 instanci ili koristi Fargate za serverless izvršavanje.

Ključne karakteristike:

- Integracija sa Docker-om: Možete koristiti standardne Docker image-e.
- Automatsko skaliranje: ECS može automatski skalirati broj instanci ili taskova.
- Laka integracija sa AWS alatima: Load balancer, IAM, CloudWatch itd.

Kada koristiti ECS?

- Ako je potrebno jednostavno upravljanje Docker kontejnerima bez kompleksnosti Kubernetes-a.
- Ako se koristi AWS ekosistem i poželjna je potpuna integracija.

2. Amazon EKS (Elastic Kubernetes Service)

EKS je AWS-ov upravljani servis za pokretanje Kubernetes klastera. On upravlja Kubernetes kontrolnom ravni (control plane), dok korisnik upravlja worker čvorovima (EC2 ili Fargate).

Ključne karakteristike:

- Potpuna podrška za Kubernetes: Koriste se standardni Kubernetes manifesti i alati.
- Integracija sa AWS alatima: IAM za RBAC, ALB za ingress itd.
- Podrška za Fargate: Bez-serversko izvršavanje za Kubernetes pod-ove.

Kada koristiti EKS?

- Ako se već koristi Kubernetes i potrebno je potpuno upravljano okruženje.
- Ako je potrebna fleksibilnost Kubernetes-a sa AWS integracijama.

3. AWS Fargate

Fargate je serverless platforma za pokretanje kontejnera. Ona eliminiše potrebu za upravljanjem EC2 instancama – samo se definišu kontejneri, a AWS upravlja infrastrukturom.

Ključne karakteristike:

- Bez servera, tj. serverless: Nema potrebe za upravljanjem instancama.
- Integracija sa ECS i EKS: Može se koristiti Fargate za pokretanje taskova (ECS) ili pod-ova (EKS).

- Automatsko skaliranje: Fargate automatski dodaje resurse prema potrebi.

Kada koristiti Fargate?

- Ako je fokus na kontejnerima, a ne na infrastrukturi.
- Ako je promenljivo opterećenje i poželjno automatsko skaliranje.

Uporedna analiza data je u tabeli 4.

Tabela 4 Uporedni prikaz svojstava AWS-ovih servisa za kontejnerizaciju

Karakteristika	**ECS**	**EKS**	**Fargate**
Upravljanje infrastrukturom	Upravljanje EC2 instancama	Upravljanje EC2 instancama	Serverless (AWS upravlja infrastrukturom)
Orkestracija	Docker	Kubernetes	Docker (ECS) ili Kubernetes (EKS)
Integracija sa AWS alatima	Da	Da	Da
Fleksibilnost	Ograničena (samo Docker)	Visoka (Kubernetes)	Ograničena (serverless)
Cena	Niža (samo EC2 troškovi)	Viša (EC2 + EKS upravljanje)	Viša (serverless model)

Primer 1: Pokretanje Docker kontejnera na ECS

Za pokretanje kontejnera na ECS potrebno je izvršiti naredne korake:

1. Kreirajte Dockerfile za vašu aplikaciju.

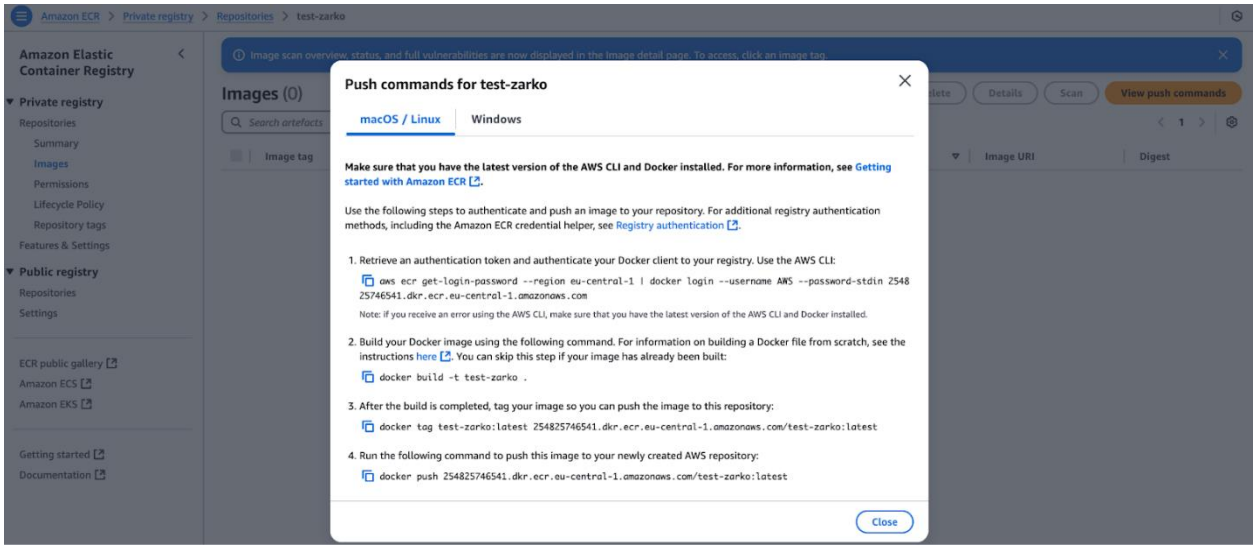
   Ovde se može iskoristiti Dockerfile sa prethodnih vežbi.

1. Izvršite push slike na **Amazon ECR** (Elastic Container Registry).

   Kako bi se objavio image na ECR potrebno je prvo napraviti repozitorijum za taj image u okviru ECR (Slika 41).

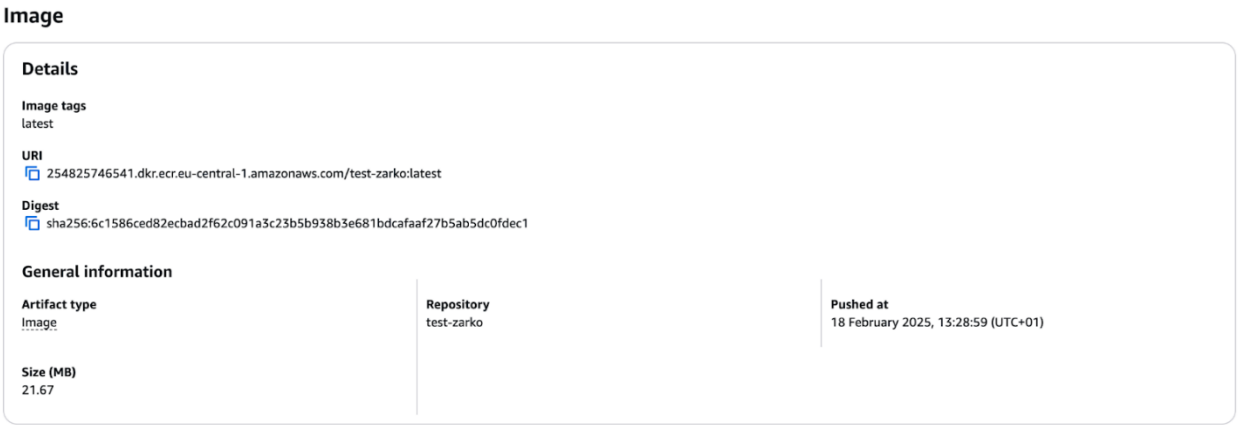
Slika 41 Kreiranje repozitorijuma na ECR

Zatim pristupiti repozitorijumu i aktivirati opciju view push commands za pregled komandi koje je potrebno izvršiti kako bi se image objavio (na primer, slika 42):



Slika 42 Pregled komandi kako bi se objavio image na AWS ECR repozitorijumu

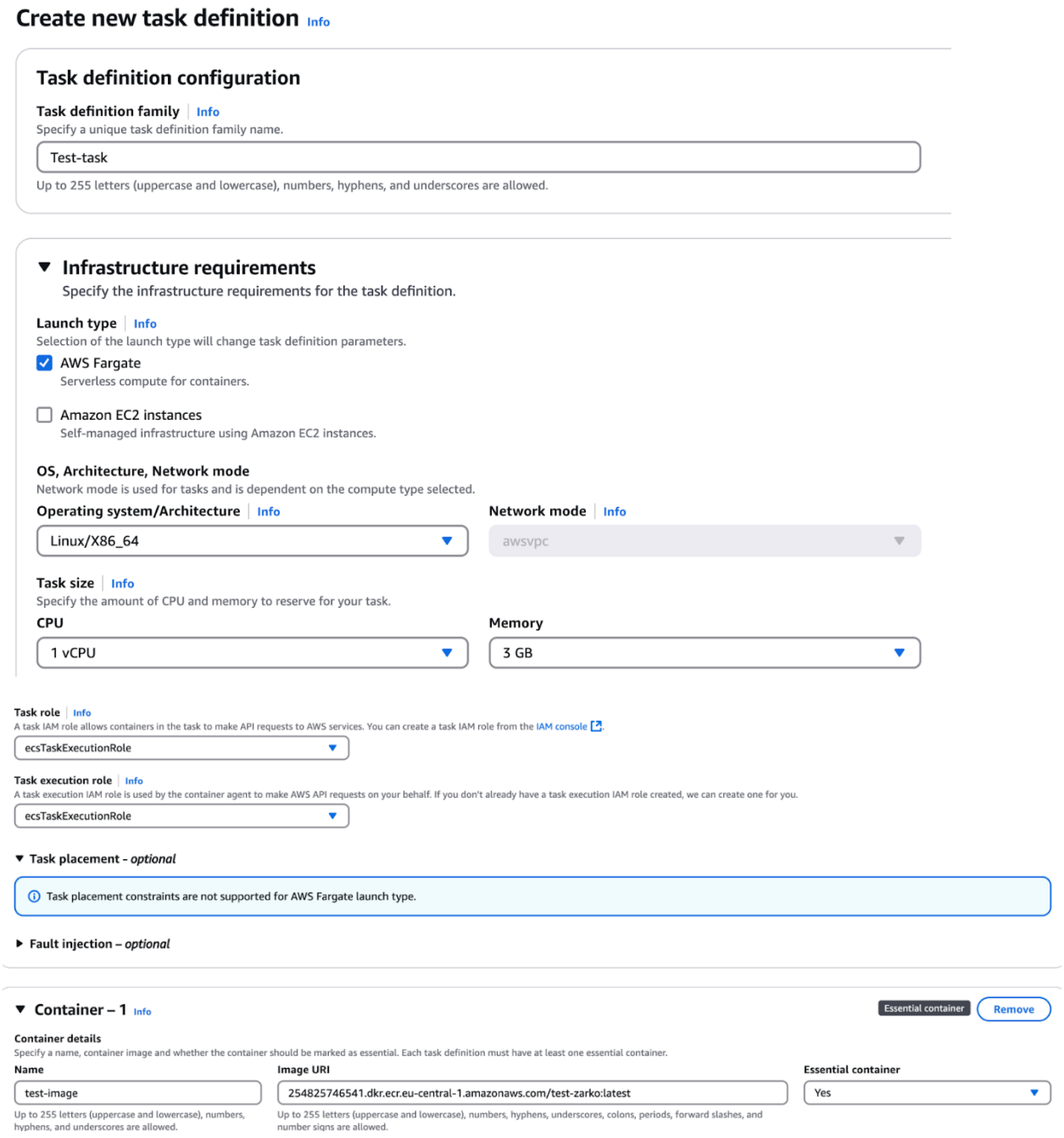
Nakon izvršenih komandi image se pojavljuje u okviru repozitorijuma i može mu se pristupiti kroz AWS interfejs (Slika 43).



Slika 43 Image objavljen na AWS ECR repozitorijum

1. Definišite ECS task definiciju (koju sliku i resurse korist itd.).

   Zatim je potrebno definisati ECS Task kroz ECS servis u okviru AWS-a. U bočnom meniju se bira Task definitions i tu se precizira konfiguracija. Na slici Slika 44 je prikazana konfiguracija kao primer za prethodni image.



Slika 44 Koraci konfiguracije taska kroz ECS servis

1. Pokrenite ECS servis koji će upravljati taskovima.

Primer 2: Pokretanje Docker kontejnera na EKS

1. Kreirajte Kubernetes manifeste (npr. deployment.yaml, service.yaml).

   Deployment fajl definiše sve parametre pod-a i potrebne resurse koje će Kubernetes klaster obezbediti.

! service.yaml > { } spec > { } selector

io.k8s.api.core.v1.Service (v1@service.json)

```yaml
apiVersion: v1
kind: Service
metadata:
  name: test-zarko-service  # Ime Service-a
  labels:
    app: test-zarko  # Label za lakše upravljanje
spec:
  type: LoadBalancer  # Tip Service-a (LoadBalancer izlaže aplikaciju na javnu IP adresu)
  ports:
  - port: 80          # Port na kojem Service sluša
    targetPort: 80     # Port na kojem kontejner sluša
    protocol: TCP      # Protokol (TCP je podrazumevani za HTTP)
  selector:
    app: test-zarko  # Label za selektovanje Pod-ova iz Deployment-a
```

Service fajl definiše način pristupa pod-u i njegove mrežne karakteristike, kao i balansiranje saobraćaja preko više pod-ova (u ovom slučaju 3) i jednu pristupnu tačku.

! service.yaml > { } spec > { } selector

io.k8s.api.core.v1.Service (v1@service.json)

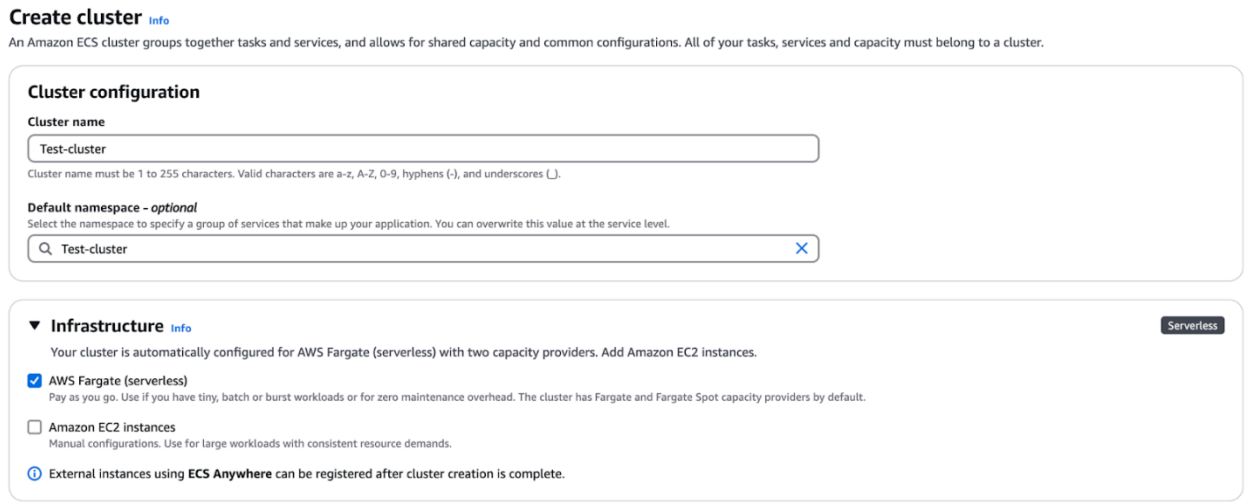
```yaml
apiVersion: v1
kind: Service
metadata:
  name: test-zarko-service  # Ime Service-a
  labels:
    app: test-zarko  # Label za lakše upravljanje
spec:
  type: LoadBalancer  # Tip Service-a (LoadBalancer izlaže aplikaciju na javnu IP adresu)
  ports:
  - port: 80          # Port na kojem Service sluša
    targetPort: 80     # Port na kojem kontejner sluša
    protocol: TCP      # Protokol (TCP je podrazumevani za HTTP)
  selector:
    app: test-zarko  # Label za selektovanje Pod-ova iz Deployment-a
```

1. Pristupite EKS klaster-u.
2. Primenite manifeste:
   kubectl apply -f service.yaml
   kubectl apply -f deployment.yaml

Primer 3: Serverless kontejneri sa Fargate

1. Podesite ECS ili EKS klaster sa Fargate podrškom.

   U okviru ECS servisa na AWS-u, potrebno je u sekciji Clusters definisati novi cluster sa podešavanjem stavke infrastructure na opciji AWS Fargate *(serverless)*(slika 45).



Slika 45 Podešavanje Fargate-a

1. Definišite task ili pod koji će se pokretati na Fargate (kao iz primera 1).
2. Fargate automatski dodaje resurse prema potrebi.

6. Skladišta i baze podataka

Ciljevi ovog poglavlja su podsećanje na konvencionalne tipove skladišta i upoznavanje sa modernim načinima čuvanja podataka, uz fokus na infrastrukturu i fajl-sisteme namenjene računarstvu u oblaku.
Po savladanom poglavlju, student će moći da:

- navede elemente infrastrukture za skladištenje podataka
- objasni koncept kompleksnijih sistema kao što su nizovi diskova, RAID, SAN i NAS
- objasni specifičnosti zahteva fajl-sistema u oblaku
- objasni princip rada obrade MapReduce
- navede osnovne principe fajl-sistema GFS i HDFS
- navede opcije baza podataka dostupne u oblaku
- navede oblike skladišta dostupne kod AWS-a
- navede oblike baza podataka kod AWS-a

Student će umeti da:

- izabere odgovarajuće skladište u AWS-u
- samostalno konfiguriše skladište tipa S3 bucket u AWS-u
- samostalno konfiguriše skladišta tipa EBS i EFS u AWS-a
- samostalno konfiguriše baze podataka u AWS-u

Pouzdano čuvanje i efikasno dobavljanje podataka jesu imperativi računarstva uopšte, a računarstvo u oblaku samo donosi nove nivoe apstrakcije, dok su ciljevi isti, odnosno zahtevniji. U prethodnom poglavlju dat je pregled tehnologija data-centra oblaka i u tom kontekstu i neke osnovne informacije o tehnologijama skladišta.

6.1. Tehnologije skladištenja

U nastavku će prvo detaljnije biti predstavljene hardverske tehnologije skladištenja. Korak dalje ide se sa virtuelizacijom, koja otvara vrata za sva očekivanja stavljena pred računarstvo u oblaku.

Hard-diskovi

Prvi komercijalni hard-diskovi predstavljeni su od strane IBM-a 1956. godine. Tadašnji principi čuvanja podataka zasnovani na magnetizmu sreću se i decenijama kasnije kod PC računara, ali i danas u serverima data-centara.

Hard-disk uređaji (HDD) čuvaju podatke na metalnim namagnetisanim površinama (platters – „tanjirima“). Svaka takva kružna površina izdeljena je na koncentrične prstenove – staze (tracks). Glava disk-uređaja lebdi iznad trake i očitava podatke sa nje. Glava se može kretati od najdalje unutrašnje ka najdaljoj spoljašnjoj traci, dok se disk okreće, kako bi zahtevani deo došao tačno ispod glave za čitanje/upis. Najmanja jedinica koju čita ili piše glava hard-disk uređaja je sektor i obično je veličine 512 bajtova.

U praksi, ne čita se pojedinačni sektor, već obično nekoliko susednih sektora, koji čine tzv. klaster. Klaster je jedinica koju adresira operativni sistem i kada se šalje zahtev za podacima, čita se ceo klaster. Hard-diskovi su često usko grlo u modernim računarskim sistemima jer poseduju mehaničke delove koji ne mogu da prate brzinu ostalih komponenti računara.

SSD

Solid State Drive predstavlja tehnologiju poluprovodničke memorije (kolokvijalno se često nazivaju SSD disk, iako tehnički ne postoji disk). Zahvaljujući tome što nemaju mehaničkih delova, ovakve memorije su znatno brže i za čitanje i za upis.

Magnetne trake

Magnetne trake su medijum koji se pojavio još pre magnetnih diskova i predstavljali su naslednika bušenih kartica. U pitanju je medijum sa sporim pristupom, međutim i dalje nalazi primenu u modernim računarskim sistemima. Traka omogućava sekvencijalni pristup: namotana je i premotava se u odnosu na glavu za čitanje/upis. Kad se traka pozicionira, brzine prenosa su slične kao kod hard-diska. Takozvane biblioteke traka mogu da čuvaju petabajte ili čak egzabajte (milion terabajta) podataka. Namenu nalaze pre svega u arhiviranju podataka, u situacijama kada je potrebna veliki kapacitet, ali nije neophodno da se podacima brzo pristupa.

Magnetne trake ne samo da ne stagniraju, već se razvijaju. Nove tehnologije zapisivanja omogućavaju drastičan porast gustine zapisanih podataka, čime raste i kapacitet jedinica za skladištenje. Povećanje kapaciteta hard-diskova ne može da isprati dinamiku rasta proizvedenih podataka, te su magnetne trake izbor za čuvanje svih onih ogromnih količina podataka koje zahtevaju da budu arhivirane, a nije neophodno da im se brzo pristupa. IBM je jedna od firmi koje su najviše uključene u razvoj ovih tehnologija. Godine 2020. zajedno sa FujiFilm-om, IBM je razvio metodu zapisa 27 puta gušćeg nego kod postojećih traka[11].

Glavna tehnologija koja dominira u sferi magnetnih traka je Linear Tape-Open (LTO) standard, koji je razvijen od strane konzorcijuma kompanija IBM, HPE i Quantum. LTO trake poslednje generacije (LTO-9, iz 2021. godine) pružaju kapacitet do 18 TB bez kompresije i do 45 TB sa kompresijom (2,5:1). LTO-9 omogućava brzinu prenosa do 400 MB/s bez kompresije. Trajnost podataka je preko 30 godina. LTO podržava hardversko šifrovanje uz standard AES-256.

Cena magnetne traje je povoljnija od hard-diska, potrošnja električne energije je manja, a medijum je generalno bezbedniji, naročito kad se ima u vidu da se sami kertridži sa trakama mogu lako ukloniti i skladištiti u potpunosti oflajn. Na kraju, otpornost je veća nego kod HDD i SSD memorija, nema specijalnih mehaničkih delova i samo trošenje je minimalno.

Interesantan je slučaj iz 2011. godine, kada je usled baga došlo do brisanja mejlova oko 40 hiljada korisnika gmail-a[12]. Ni posedovanje kopija na hard-diskovima na više nezavisnih lokacija nije pomoglo jer je bag „pokvario“ i te kopije. Na kraju, mejlovi su vraćeni iz rezervne kopije (bekapa) čuvane na magnetnim trakama.

Nizovi diskova

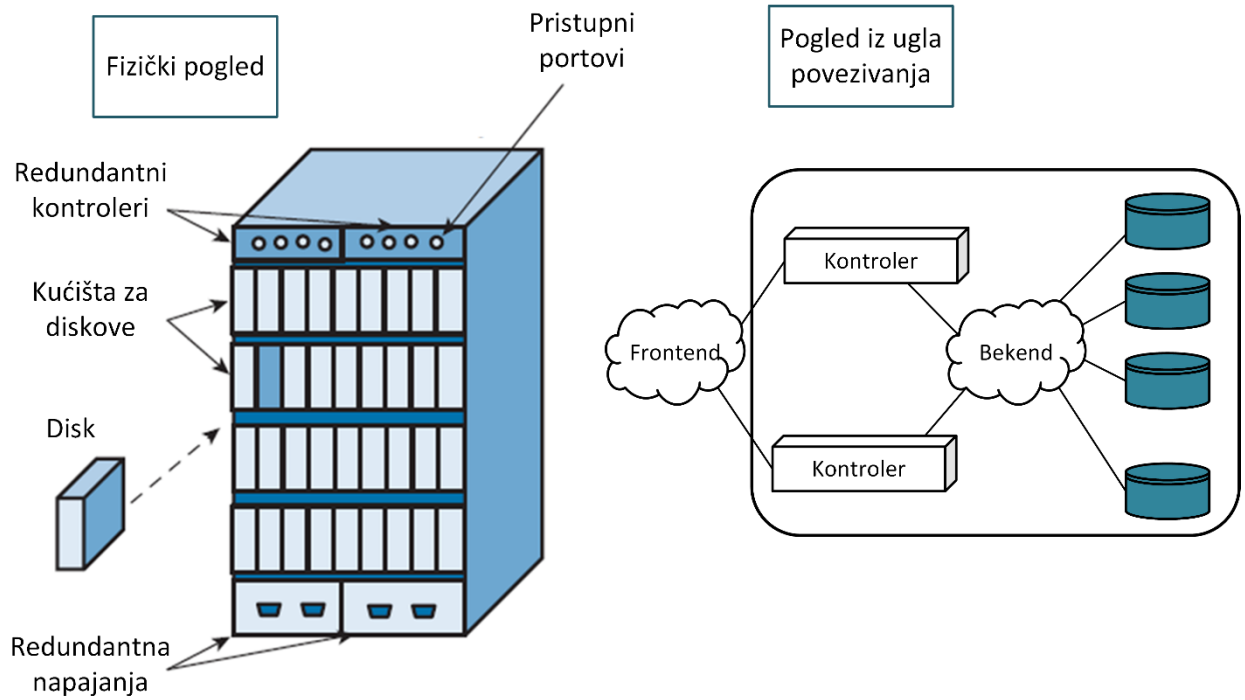
U konvencionalnom PC računaru diskovi su neposredno povezani na matičnu ploču, međutim, kod velikih sistema, gde je neophodna pouzdanost i modularnost, diskovi se u data centrima organizuju u nizove *(disk arrays).* Ciljevi ovakve organizacije jesu razdvajanje servera i skladišta, kao i povećanje pouzdanosti i olakšavanje održavanja.

Nizovi diskova su zaokružene celine, koje imaju svoje kontrolere, napajanje i RAM memoriju; na neki način liče na nezavisne „skladišne-servere“ sa jednom funkcijom: čuvanje podataka. Povezivanje sa serverima se vrši putem portova – sa jedne strane se povezuju diskovi na kontrolere, a sa druge serveri (Slika 46).

[11] https://research.ibm.com/blog/tape-density-record

[12] https://www.techmonitor.ai/hardware/data-centres/magnetic-tape-storage-is-seeing-cloud-go-back-to-the-future-for-its-archival-data-needs

Na kraju, nizovi diskova su osnova za SAN i NAS oblike skladišta, koji se danas koriste u data-centrima i faktički su sklopovi koji nose podatke modernog oblaka (više o SAN/NAS kasnije u poglavlju).



Slika 46 Komponente niza diskova

Pristup podacima u mirovanju

U terminologiji oblaka podaci se mogu razmatrati u dva slučaja: podaci koji se prenose (data in transit) i podaci koji su pohranjeni, čuvani, u mirovanju (data in rest). U nastavku će biti fokus na pristupu podacima u mirovanju, uz nadovezivanje na prethodno opisane tehnologije, koje neposredno doprinose trajnom čuvanju podataka.

Kako je navedeno u prethodnim sekcijama, hard-disk je podeljen na jedinice poznate kao sektori. I drugi medijumi imaju ovakve jedinice i one se mogu jedinstveno nazvati blokovi.

Aplikacije pristupaju podacima na višem nivou apstrakcije, uostalom kao i korisnici. Najpoznatiji vid je fajl. To je imenovani skup zapisa koji čini jednu celinu. Obično ga čini niz susednih blokova. Pored samih podataka koje aplikacija koristi, fajlovi imaju i metapodatke: dozvole pristupa, vremena kreiranja, pristupa i menjanja fajla itd.

Kod baza podataka, relevantna jedinica jeste zapis (record). Tehnički, tabele relacionih baza podataka se takođe čuvaju u fajlovima na disku, ali sam pristup na nivou sistema za upravljanje bazama podataka (npr. MySQL, PostgreSQL) „vidi“ zapis i prilikom izvršavanja upita jedinica koja figuriše jeste zapis.

Zavisno od toga na kom nivou se razmatra pristup, da li na nivou bloka, fajla ili zapisa, može se govoriti o različitim tehnologijama pristupa.

Pristup na nivou bloka izvodi se povezivanjem skladišta putem različitih interfejsa. Kod PC računara za interne diskove danas se najviše koristi SATA (serial ATA), dok se ranije koristio PATA (parallel ATA). Za povezivanje eksternih diskova u opticaju je najviše USB, dok je ranije aktuelan bio i FireWire.

Za povezivanje većeg broja diskova najpopularniji standard je bio SCSI – Small Computer Systems Interface (čita se skazi). SCSI definiše ne samo fizički interfejs, već i protokole koji se poštuju prilikom prenosa podataka na relaciji server – skladišni uređaj. Skladišni uređaj može biti pojedinačni disk, ali i RAID niz (više o RAID-u kasnije). Upravljanje prenosom vrši Host Bus Adapter (HBA), koji je obično realizovan kao kartica u serveru ili deo matične ploče. Skladišni uređaji su kaskadno povezani kablom i komunikacija je poludupleks. To znači da u jednom trenutku samo jedan uređaj može da prenosi podatke datom magistralom. Svaki uređaj može imati više jedinica za skladištenje, npr. više logičkih particija ili virtuelnih diskova i svaka takva jedinica ima jedinstven LUN (logical unit number). Kada HBA inicira prenos, navodi magistralu, target (skladišni uređaj) i LUN i time posebno definiše konkretnu lokaciju za prenos.

Danas se u data-centrima koristi poboljšana varijanta SCSI-a, SAS – Serial Attached SCSI. SAS omogućava veće brzine prenosa, povezivanje više hiljada uređaja (SCSI omogućava samo 15), kao i veće dužine kablova.

RAID

Skraćenica RAID znači Redundant Array of Independant Disks[13] i to je tehnologija za grupisanje većeg broja diskova sa ciljem dobijanja na pouzdanosti i brzini prenosa. Da bi se postavio RAID, potrebno je da postoji poseban RAID kontroler, koji upravlja formiranjem RAID-a i komunikacijom.

Hardverski RAID podrazumeva da se čitavi fizički diskovi tretiraju kao pojedinačni i kao takvi grupišu u odgovarajuća RAID polja. Sve operacije vezane za kontrolu vrši hardver.

Kod softverskog RAID-a, RAID polja se formiraju na nivou operativnog sistema. Glavni procesor upravlja RAID-om i zbog toga je ova varijanta sporija. Sa druge strane, veća je fleksibilnost jer delovi fizičkih diskova mogu da se tretiraju kao jedinice za formiranje RAID-a.

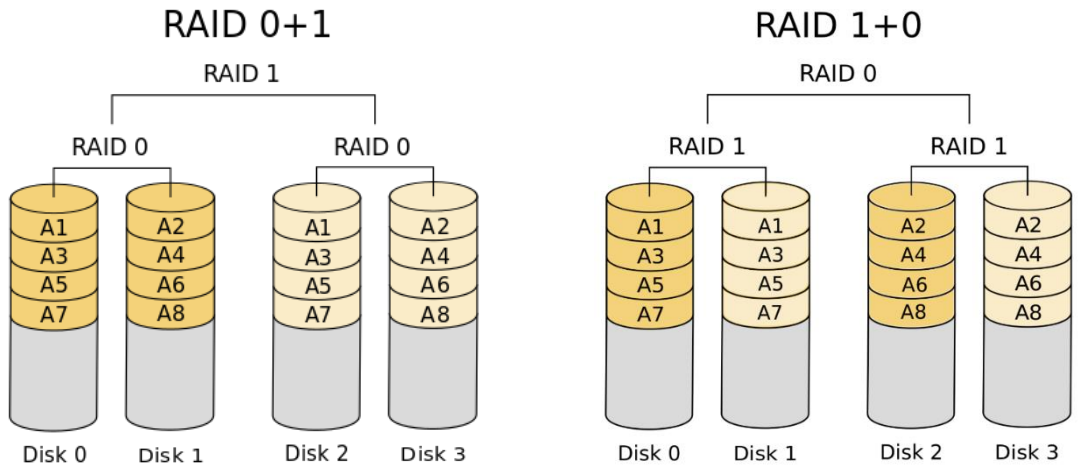
Dva osnovna RAID polja su RAID 0 i RAID 1. RAID 0 je poznat i kao *„striping“.* Zahtevaju se najmanje dva diska. Podaci se mogu upisivati paralelno na oba diska, čime se dobija na brzini. Podaci se dele u „štrafte“ (stripes) veličine do 1 MB. Takvi delovi podataka se upisuju na više diskova (u najjednostavnijoj implementaciji na dva) paralelno[14]. Kapacitet ovog RAID-a je zbir kapaciteta dva diska.

Raid 1 (mirroring) podrazumeva upisivanje istih podataka na dva diska, tako da oba sadrže identične podatke. Time se dobija na pouzdanosti – otkazom jednog diska, drugi zadržava podatke. Ukoliko je podržan hot-swap, disk se može brzo zameniti i tada se ponovo formira RAID polje sa novim, praznim diskom koji se postepeno „puni“ podacima i u nekom trenutku polje ponovo sadrži dva diska sa istim podacima. Kapacitet polja RAID 1 je veličine jednog diska, tj. dvostruko je manji od ukupnog kapaciteta diskova. Takođe, RAID1 je relativno spor.

Korišćenjem 4 diska može se formirati kombinacija RAID polja 0 i 1 i to 10 ili 01. Kod RAID 10, prvo se vrši *striping* podataka na dva diska, a onda se sa ta dva diska vrši mirroring. Kod RAID-a 01 je obrnuto, prvo se dupliraju podaci, a onda se „strajpuju“. Na slici Slika 47 je prikazan princip.

[13] Inicijalno skraćenica je predstavljala *Redundant Array of Inexpensive disks* i fokus je bio na korišćenju jeftinih diskova.

[14] Tehnički, ne upisuju se doslovno paralelno, ali se dobija na brzini određenom dozom paralelizma koju je moguće ostvariti. Dok jedan disk vrši upis, kontroler može da pripremi podatke za drugi disk, time ostvarujući delimičnu paralelizaciju.



Slika 47 RAID polja 0+1 (levo) i 1+0 (desno)

Popularan tip RAID-a je RAID5, koji pravi kompromis između pouzdanosti, performansi i broja korišćenih diskova. Koristi se najmanje tri diska na koje se beleže podaci po principu strajpinga, ali se na iste diskove smeštaju i blokovi parnosti. Kada jedan disk otkaže, na osnovu blokova parnosti preostalih diskova mogu se rekonstruisati podaci. Blok parnosti je faktički XOR operacija (ekskluzivno ili, operator $\oplus$) nad dva bloka; ako su blokovi podataka A i B, onda se na treći disk upisuje P ($P=A \oplus B$). Ukoliko bilo koji od tri diska otkaže, nedostajući podatak se dobija od preostala dva, npr. ako otkaže disk sa podatkom A, A se dobija kao $B \oplus P$, sa preostala dva diska.

Pored navedenih tipova RAID polja, na raspolaganju su i druga (od 2 do 10). Za potrebe konteksta ove knjige predstavljena polja su dovoljna.

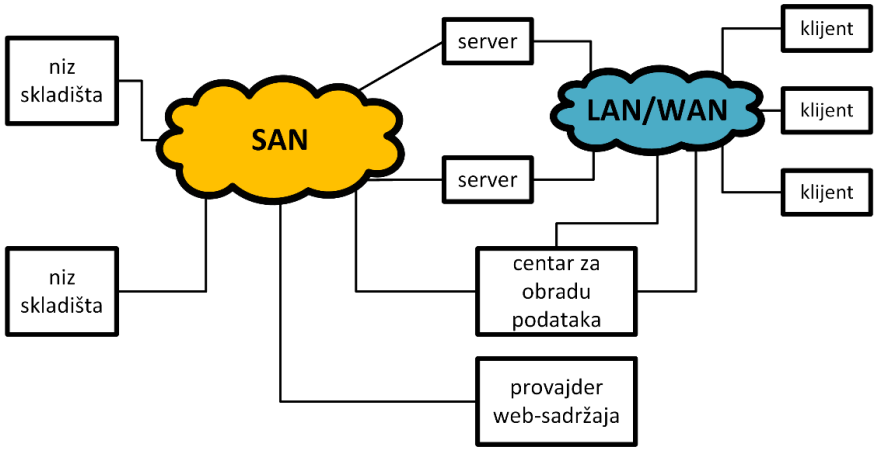
Bez obzira na koji tip RAID-a će pasti izbor, i dalje postoji jedinstvena tačka otkaza (SOF – single point of failure), a to je sam RAID kontroler. Podaci sa diskova koji su otkazali mogu se nadoknaditi, ali kvarom kontrolera svi diskovi postaju nedostupni. Dalje usavršavanje skladišta ka nezavisnom robusnom rešenju podrazumeva korišćenje više kontrolera na takav način da otkazom jednog drugi može da preuzme čitav posao, bez značajnog pada performansi.

Na kraju, važno je obezbediti redundantnost i na nivou procesora (znači najmanje dva procesora), kao i na nivou mreže (dva mrežna interfejsa – jedan kao rezervni).

SAN

SAN (Storage Area Network) se definiše kao mreža sa primarnim ciljem prenosa podataka između računarskog sistema (servera) i skladišta. Pored komunikacione infrastrukture, obuhvata i sloj upravljanja (Slika 48). SAN je specijalizovana mreža visoke propusnosti koja eliminiše tradicionalnu posvećenu vezu servera i skladišta. Nasuprot tome, omogućava vezu svih sa svima. Server može pristupati skladištu, još bolje: više servera može pristupati jednom skladištu. Takođe, serveri mogu pristupati jedni drugima i na kraju skladišta mogu pristupati jedni drugima, na primer za potrebe kreiranja rezervnih kopija, bez uplitanja servera i trošenja njihovih resursa.

Jedan server više nije ograničen na određeni kapacitet za čuvanje podataka, koji je definisan neposrednom povezanošću na sam server.



Slika 48 Tipična SAN konfiguracija

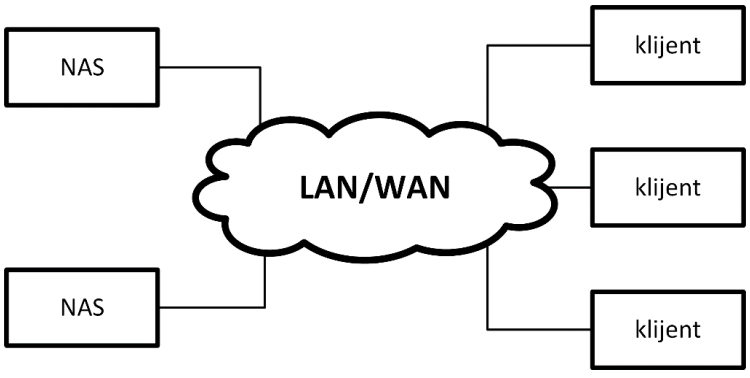
Korisno je uporedno predstaviti ova dva značajna vida skladištenja: NAS i SAN. U tabeli 5 dat je uporedni pregled ove dve distribuirane skladišne tehnologije (Tate et al, 2017).

Tabela 5 - Uporedni prikaz tehnologija NAS i SAN

NAS – Network Attached Storage	SAN – Storage Area Network
Podaci se čuvaju na nuvou fajla	Podaci se čuvaju na nivou bloka
Osnovni medijum: Ethernet	Osnovni medijum: fibre channel (optika)
Skladište se prikazuje kao deljeni folder	Skladište se prikazuje kao neposredno povezano
Povoljan	Skup
Zavisi od mreže (LAN-a)	Ne zavisi od mreže
Ne zahteva nikakve arhitekturne izmene	Zahteva izmene u arhitekturi

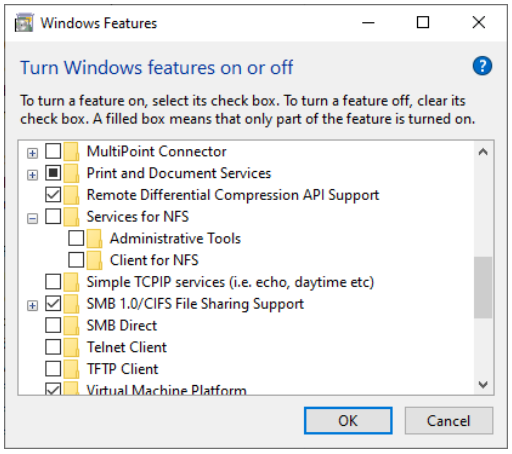
6.1. Fajl-sistem

Fajl-sistem predstavlja način organizacije podataka na diskovima. Obuhvata metode za pristup podacima, opisuje koji se metapodaci koriste, kako se fajlovi grupišu i pretražuju itd. Na PC računarima i izolovanim serverima u upotrebi su često samo lokalni fajl-sistemi, na primer kod Windowsa NTFS, kod Linuxa ext3, ext4, btrfs, XFS itd. Kod kompleksnih sistema neophodno je povezivanje skladišta sa većim brojem servera i definisanje komunikacije putem mreže. Skladišta povezana putem mreže poznata su kao NAS (Network Attached Storage). Ovakva skladišta na sebi imaju fajl sisteme poznate kao Network FS – NFS (slika 49). Pored NFS-a, treba pomenuti i iSCSI – Internet SCSI, interfejs koji omogućava povezivanje diskova putem mreže tako da se ponašaju kao lokalni diskovi povezani SCSI-jem.



Slika 49 Koncept NFS-a

NFS uopšteno predstavlja distribuirani fajl-sistem koji omogućava korišćenje skladišta preko mreže (npr. Etherneta). Poznata implementacija NFS-a upravo se naziva samo NFS i podržana je kod Linux/UNIX sistema. Kada je neki direktorijum servera podeljen putem NFS-a, NFS klijent može da ga „montira“ (mount) u svoj fajl sistem i koristi u skladu sa dozvolama. Windows može takođe koristiti NFS, ali je neophodno da se instalira odgovarajuća podrška (Slika 50 – Services for NFS).



Slika 50 Podrška za NFS u Windows-u

Nasuprot NFS-u, kod Windowsa je izvorno razvijan CIFS (Common Internet File System). Ponovo, postoji podrška i u „drugom taboru“. Tačnije, razvijena je kompletna implementacija CIFS-a otvorenog koda – za Linux, poznata kao Samba. Ona omogućava olakšanu integraciju Windows i Linux sistema.

I NFS i CIFS podržavaju autentifikaciju klijenta i servera kroz Windows ili Linux mehanizme, a oba podržavaju Kerberos.

Fajl-sistemi u oblaku

Fajl-sistem (ili sistem datoteka) predstavlja logičku organizaciju podataka u sklopu određenog skladišta. Fajl-sistem definiše maksimalnu veličinu fajlova, način adresiranja i povezivanja blokova, metode za pretraživanje, sistem dozvola pristupa, eventualne mehanizme oporavka podataka itd. Kod konvencionalnih računara i servera izbor fajl-sistema je značajan. Trivijalan primer: FAT32 ne može da čuva fajlove veće od 4GB – ako je potrebno nekome dati na kopiranje video-zapis od 4,5 GB, neophodno je koristiti drugi fajl-sistem, npr. exFAT. Takođe, za specijalne namene, npr. čuvanje baze podataka, XFS može dati bolje performanse pri konkurentnom pristupu, nego ext4. Kod računarstva u oblaku zahtevi idu korak dalje i iziskuje se razvoj i upotreba posebnih fajl-sistema, koji mogu da odgovore na zahteve oblaka. Dakle, u upotrebi su konvencionalni fajl-sistemi; na primer, ako kreiramo virtuelnu mašinu sa Windows Serverom, izabraćemo NTFS, a sa Ubuntuom ext4 ili btrfs, ali se u određenim situacijama koriste i specifični fajl-sistemi, kadri da odgovore na zahteve intenzivnih aplikacija i čuvanja velikih količina podataka.

U nastavku će prvo biti reči o konvencionalnim fajl-sistemima, a zatim će biti prezentovani primeri „cloud-native“ fajl-sistema. Pre svega će akcenat biti na Linux fajl-sistemima, s obzirom na to da je većina servera sa sistemima Linux/UNIX, kao i da je Linux u osnovi većine implementacija oblaka.

Kod razmatranja fajl-sistema za opštu namenu na računarima i serverima, izbor obično pada na stare, dobro poznate sisteme i oni se podrazumevano i nude kod Windowsa i Linuxa. Kod Windowsa je već dve decenije unazad to NTFS, dok je kod Linuxa danas u pitanju ext4, a izbor može biti i ext3, btrfs, XFS itd. Svi moderni fajl-sistemi imaju podršku za važnu fukciju, a to je *journaling*; to nije bio slučaj kod FAT32 ili ext2 fajl-sistema. Journaling se izvodi najčešće na nivou metapodatka: sve akcije nad

podacima se prvo beleže u poseban dnevnik – journal, pa tek onda izvode. Ukoliko dođe do prekida u radu računara usred operacije, na primer kopiranja ili brisanja podataka, sistem se može vratiti u konzistentno stanje. Kad računar proradi, na osnovu zapisa u dnevniku sve operacije će se vratiti unazad i podaci će biti usaglašeni ili će pak operacija biti dovršena. Dnevnik podataka je danas *„must have“* za sve sisteme opšte namene: lične računare, web-servere prosečnih veb-sajtova, fajl servere u manjim organizacijama i slično. Sa druge strane, kod kritičnih sistema treba dalje razmotriti izbor FS. U tabeli 6 dat je uporedni prikaz nekoliko modernih fajl-sistema i njihova svojstva koja mogu biti kriterijumi za izbor.

Tabela 6 Uporedni prikaz modernih fajl-sistema

	FAJL-SISTEM				
Svojstvo	NTFS	EXT4	BTRFS	XFS	ZFS
Platforma	Windows	Linux	Linux	Linux	Linux, BSD, Solaris
Maks. veličina fajla [15]	16 TB[16]	16 TB	16 EB	8 EB	16 EB
Maks. vel. particije	8 PB	1 EB	16 EB	8 EB	16 EB
Snapshot	Ne	Ne	Da	Ne	Da
Podrška za RAID	Ne	Ne	Da	Ne	Da
Kompresija	Da	Ne	Da	Ne	Da
Integritet podataka	Osnovno (kontrolne sume metapodataka)	Osnovno	Kompletno (sume metapodataka i podataka)	Osnovno	Kompletno
Skalabilnost	Srednja	Visoka	Visoka	Vrlo visoka	Visoka
I/O performanse[17]	Visoka (opšta upotreba)	Visoka (opšta upotreba)	Srednje (zbog podške za COW)	Visoke (intenzivno pisanje)	Srednje (RAM/CPU intenzivne)
Zahtevi za resursima (CPU. RAM)	Niski	Niski	Srednji	Niski	Visoki
Scenario za upotrebu	Windows okruženja, opšta namena	Linux okruženja, opšta namena	Napredne Linux konfiguracije, potreba za snapshot-ima	Linux serveri visokih performansi	Kritični sistemi i zahtevan integritet podatka
Ugrađeno šifrovanje	Da	Ne	Eksperimentalno	Ne	Da
Defragmentacija	Da	Retko potrebna	Nepotrebna	Da (postoje alati)	Nepotrebna

Već su navedeni i distribuirani fajl-sistemi, kao što su NFS i CIFS. Ovakvi fajl-sistemi su na neki način korak dalje ka specijalnim sistemima za oblak.

[15] Veličina fajla zavisi od veličine klastera, tako da su date neke okvirne vrednosti.
[16] TB – terabajt, 1000 gigabajta; PB – petabajt, 1000 terabajta; EB – egzabajt, 1000 petabajta
[17] Navedene su performanse za najčešće scenarije korišćenja datih FS.

6.2. Specijalni fajl-sistemi u oblaku

Pre nego budu opisani neki od popularnih fajl-sistema u oblaku, biće analizirane potrebe koje se, u odnosu na konvencionalno lokalno i distribuirano računarstvo, postavljaju pred oblak.

Jedno od osnovnih svojstava oblaka jeste postojanje više korisnika (engl. *tenants* – stanari). Kod javnog oblaka ti korisnici su različite firme, kod privatnog oblaka, korisnici su odeljenja iste firme. U oba slučaja treba imati u vidu da korisnici ne mogu da veruju jedni drugima i neophodno je izvršiti izolaciju. Takođe, korisnici ne mogu u potpunosti da veruju ni dobavljaču usluga u oblaku! U nekim situacijama, kod veoma osetljivih podataka, kao što su medicinski ili finansijski zapisi, korisnici ne smeju da veruju dobavljaču, te postoje dodatna ograničenja vezana za čuvanje ovakvih podataka u oblaku. Moderni fajl-sistemi u oblaku moraju da budu koncipirani tako da odgovore na ove zahteve. Veliki broj korisnika-tenanta i veliki broj krajnjih korisnika koji su hostovani kod korisnika usluga u oblaku, uz činjenicu da se dinamički taj broj stalno povećava (registruju se novi), dalje vodi ka problematici vezanoj za upravljanje identitetima. Neophodno je omogućiti mehanizam koji će da hijerarhijski uredi prostor identiteta, tako da dobavljač delegira upravljanje svakom korisniku-tenantu, a onda će svaki taj korisnik-tenant da vodi računa o svojim krajnjim korisnicima. U ovom zadatku neophodna je podrška fajl-sistema.

MapReduce

Da bi se kvalitetnije i potpunije objasnila funkcija specijalnih fajl-sistema, potrebno je predstaviti nove paradigme paralelnog i distribuiranog procesiranja. Jedna od glavnih ovakvih paradigmi jeste programski model MapReduce. Svrha ovog modela je da omogući obradu velikih količina podataka u efikasnom maniru, distribuirajući opterećenje na više čvorova. Izvorno, ovaj model je kreiran od strane Google-a 2004. godine sa ciljem obrade ogromnih količina podataka, koje su sakupljane sa Web-a, radi realizacije algoritma PageRank[18].

Ukratko, poenta MapReduce-a je da se obrada velikog skupa podataka rasporedi na više čvorova, a zatim se objedini: prvo se izvršava „Map“, funkcija koja deli obradu u distribuiranom okruženju i proizvodi među-rezultat, a zatim „Reduce“, funkcija koja spaja međurezultate i formira konačan rezultat obrade. MapReduce je model koji obezbeđuje apstrakciju izražavanja izračunavanja i podrazumeva biblioteke koje vode računa o paralelizaciji, tako da programer ne mora da brine o rešavanjima stanja trke, da prati opterećenja čvorova itd. U praksi postoje različiti okviri kojima se implementira MapReduce, a najpoznatiji je Apache Hadoop. U nastavku će na primeru biti objašnjene osnove MapReduce-a, sa ciljem približavanja fajl-sistema koji se zahtevaju za ovakav tip obrade.

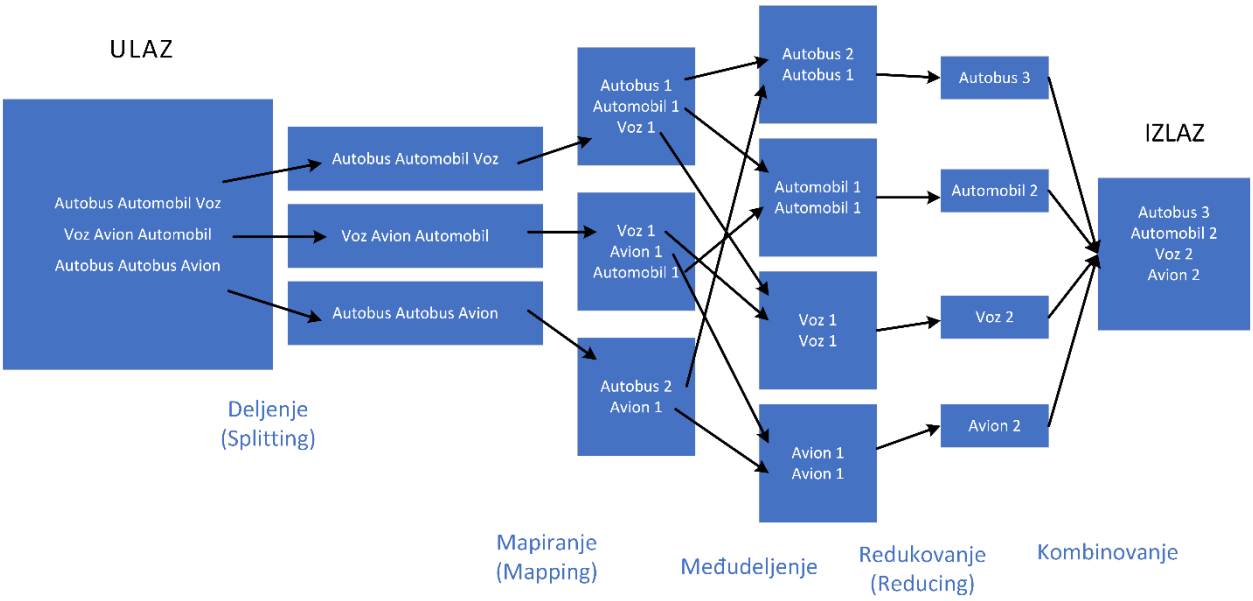
Pretpostavimo da imamo veliki fajl sa tekstom i da nam je cilj da izbrojimo pojave svake reči. Dakle, očekivani krajnji rezultat bi bio fajl u kojem bi se nalazio niz parova reč-broj pojava. Naizgled, ovo nije naročito zahtevan zadatak. Kako bi se rešio „peške“, tj. linearno u jednoj liniji obrade? Reči bi bile učitavane od početka fajla, na primer u jedan dvodimenzionalni niz, tzv. rečnik. Kad se pojavi nova reč, moralo bi da se proveri da li se već nalazi u nizu: ako se ne nalazi, dodao bi se nov element u niz i pridružio broj 1. Ako već postoji data reč, onda bi se broj pojava uvećao za 1; na primer, imali bismo [klaud,2]. Na kraju bismo imali sve reči u nizu sa svojim brojačem.

Sa algoritmom MapReduce, ceo tekst se raspoređuje (faza Mapper) na više čvorova. Sada svaki čvor za svoj deo podataka vrši funkciju prebrojavanja pojava reči. Naravno, ista reč se može pojaviti na dva čvora. U praksi se preporučuje da broj mapper-čvorova bude najviše 100.

[18] PageRank je Google-ov algoritam kojim se određuje rangiranje web-sajtova u rezultatima pretrage.

U narednoj fazi se podaci grupišu (mešaju – *shuffle*) tako da se na posebne čvorove šalju iste reči. Na kraju se u fazi reduce broje sve pojave i dobija konačan rezultat.

Na slici 50 je ilustrovan ovaj proces.



Slika 51 Proces obrade kod MapReduce-a

Na početku se ceo ulaz deli na tri segmenta i dodeljuje trima čvorovima (Slika 51). Ti čvorovi prebrojavaju pojave reči paralelno i time se štedi na vremenu obrade. Kreiraju se parovi (reč, broj pojava). Zatim se vrši Shuffle faza, gde novim čvorovima (u ovom slučaju 4) svaki od tri čvora šalje odgovarajuće reči, tako da se jedna reč dodaje uvek jednom čvoru. Na kraju se, u fazi Reduce, vrši sabiranje pojava. U datom primeru, u fazi Shuffle se sve pojave reči BUS čuvaju na istom čvoru, upravo da bi taj čvor posle mogao da ih kompletno obradi. Više o varijantama MapReduce može se pročitati kod Marinescu-a (2022).

GFS

Google fajl-sistem (GFS) nastao je još 90-ih godina i koristio se kao fajl-sistem u oblaku do 2010[19]. Osnovna ideja bila je kreiranje robusnog distribuiranog fajl-sistema koji bi omogućio visoku propusnost (čak i na račun kašnjenja), a koji bi se izvršavao na klasterima konvencionalnih računara.

U Google-u su inženjeri i istraživači pratili korišćenje podataka i na osnovu opaženih potreba, kreirali nov fajl-sistem. Pri planiranju su koristili sledeće zaključke praćenja:

- Skalabilnost i pouzdanost su kritična svojstva sistema, moraju se uvažiti od početka, bolje nego u kasnijim fazama projektovanja.
- Velika većina fajlova su veličine nekoliko gigabajta do nekoliko stotina terabajta.
- Najčešća operacija je dodavanje (append) postojećem fajlu; proizvoljno upisivanje u fajl je vrlo retka pojava.
- Sekvencijalno čitanje je uobičajeno.

[19] Danas je u upotrebi Colossus. Više informacija o ovom sistemu može se pročitati na zvaničnom blogu Google-a: https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system

- Korisnici obrađuju podatke „na veliko” i nisu mnogo zabrinuti za vreme odziva.
- Model treba da bude relativno jednostavan, da ne opterećuje dodatno programere aplikacija.

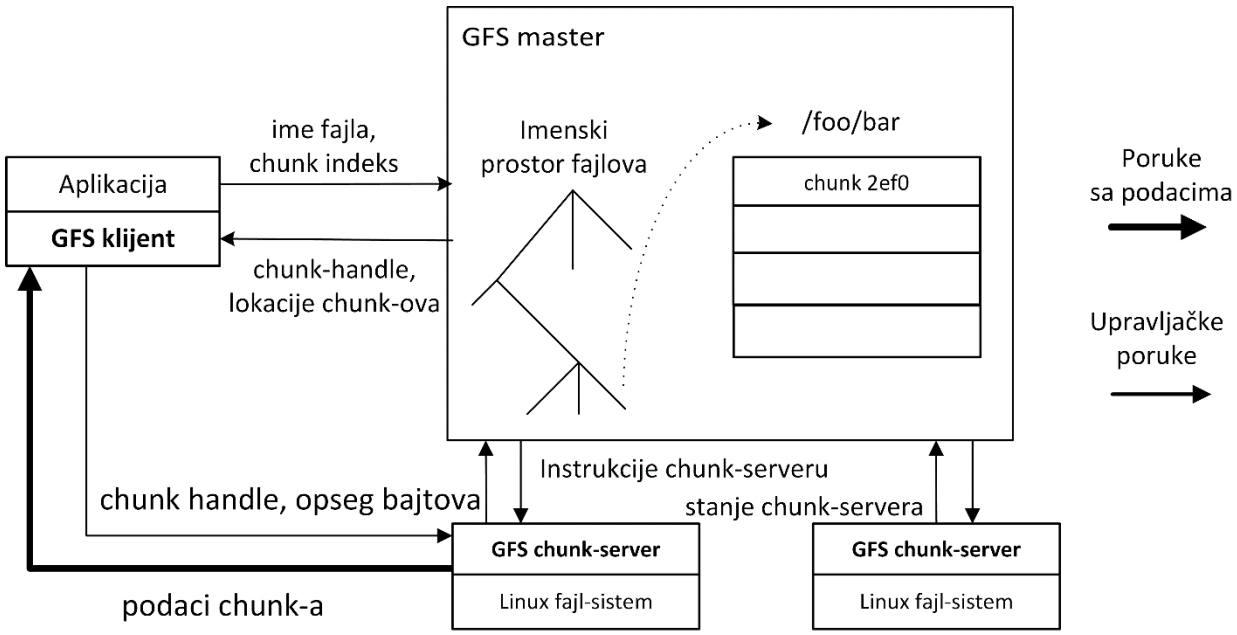
Pred implementaciju GFS-a postavljeni su sledeći zahtevi (Ghemawat et al, 2003):

- Segmentirati fajlove u velike delove;
- Ugraditi atomsku uperaciju „append“ tako da više aplikacija konkurentno može da dodaje podatke istom fajlu;
- Izgraditi klaster oko visokopropusne mreže (pre nego oko mreže niskog kašnjenja). Odvojiti kontrolne od korisničkih podataka. Uvećati propusnost slanjem više uzastopnih zahteva putem TCP-a. Iskoristiti topologiju mreže slanjem podataka najbližem čvoru;
- Eliminisati keširanje na strani klijenta;
- Osigurati konzistentnost usmeravanjem kritičnih operacija nad fajlovima preko master-čvora, koji kontroliše ceo sistem, ali minimizovati njegovu uključenost u pristupanju fajlovima;
- Ugraditi podršku za *check-point*-e i mehanizme brzog oporavka;
- Ugraditi efikasne mehanizme za uklanjanje nepotrebnih podataka (garbage collecting).

U GFS-u fajlovi su kolekcije segmenata fiksne veličine, od 64 MB zvanih *chunk* (komad) i prilikom kreiranja fajla svaki chunk dobija jedinstveni chunk-handle, identifikator. Svaki chunk ima i 32-bitni kontrolni zbir (checksum). Svaki chunk se replicira na više čvorova – taj broj se može zadati.

Poenta veličine komada od čak 64 MB je u tome što se očekuje čuvanje velikih fajlova, tako da je interna fragmentacija minimalna. Takođe, ovim se povećava verovatnoća da se više podataka nalazi u istom chunk-u, tako da se više zahteva može opslužiti kroz pozivanje istog chunk-a.

Arhitektura GFS-a data je na slici Slika 52.



Slika 52 Arhitektura GFS klastera

Fajlovi, odnosno chunk-ovi u kojima se oni čuvaju, nalaze se na više chunk-servera. U osnovi se nalazi Linux fajl-sistem, dok je chunk-server softverska nadgradnja. Serveri su povezani putem brze optičke mreže. Orkestraciju svih servera vrši master, koji čuva metapodatke i daje početne informacije (chunk handle za tražene chunk-ove) aplikaciji. Zatim aplikacija preuzima date chunk-ove sa pojedinačnih servera. Na nivou samog mastera vrši se neka vrsta journaling-a: evidencija svih operacija u jednom log-fajlu, što otvara mogućnost za praćenje i reakciju u slučaju defekta.

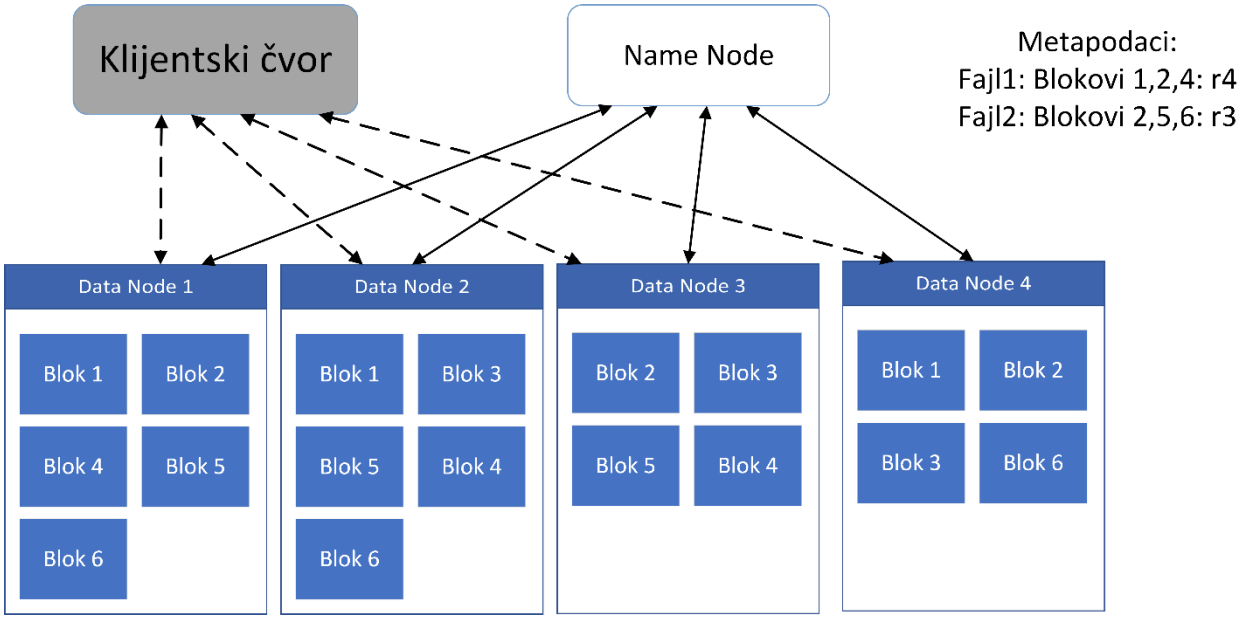
Postupci dodavanja podataka (append) su mnogo jednostavniji od izmene. U mnogim slučajevima upravo ovakav model je prirodan. Na primer, kod logova, svaki novi podatak (log-zapis) se dodaje, a stari se ne menjaju, oni su fiksirani i dokumentuju neki trenutak u vremenu. Kod MapReduce-a takođe nema izmena podataka, već se dobijeni podaci dalje tretiraju čitanjem, odnosno na kraju se upisuju rezultati u nove chunk-ove. Ovakav model korišćenja podataka je Write Once, Ready Many (Piši jednom, čitaj više puta). Ako je neophodno izmeniti neki podatak, ne ide se redom po replikama da bi se vršila prepisivanja novim podacima, već se ponovo vrši dodavanje (append) novih chunk-ova podataka, koji onda predstavljaju novu verziju postojećih; takvi chunk-ovi se onda u evidenciji metapodataka povezuju sa fajlom, dok se stari chunk-ovi na neki način arhiviraju.

Kako je prethodno napomenuto, GFS se odlično uklapa uz model MapReduce, tačnije samo planiranje i projektovanje GFS-a je izvršeno, imajući u vidu MapReduce. U nastavku slede svojstva koja povezuju GFS i MapReduce:

- GFS podrazumevano deli fajl na delove iste veličine, koji su pogodni za raspoređivanje na više čvorova, na kojima se dalje paralelno obrađuju.
- Kod GFS-a se chunk-ovi čuvaju u više primeraka (replika). Kod algoritma MapReduce, za obradu se bira čvor na kojem se već nalaze chunk-ovi ili eventualno čvor koji je veoma blizu, tako da se štedi na vremenu i opterećenju mreže.
- Postojanje većeg broja replika (najčešće 3) doprinosi robusnosti sistema: ukoliko određeni čvor otkaže, operacija se nastavlja na čvoru koji ima drugu repliku. Pošto master-čvor vodi evidenciju o svim chunk-ovima, može da iskoordiniše preraspodelu na alternativne čvorove sa replikama.
- MapReduce radi sa podacima koji se ne menjaju, tako da GFS omogućava dobre performanse i izbegavaju se komplikacije koje sa sobom nosi konkurentni pristup.
- Kod GFS-a master-čvor vodi računa o koordinaciji metapodataka i praćenju chunk-ova, tako da sam MapReduce može da bude fokusiran na obradu.
- GFS dobro skalira – ukoliko je potreban veći kapacitet, mogu se jednostavno dodati novi serveri.

HDFS

GFS je vlasnički fajl-sistem. Inspirisan njime, kreiran je Hadoop Distributed File System – HDFS. Otvorenog je koda i zasnovan na Javi. Razlike u odnosu na GFS nisu naročito značajne. Koriste se chunk-ovi od 128 MB i kod HDFS-a se oni nazivaju blokovi. Centralni čvor se naziva NameNode i metapodaci se čuvaju u memoriji (a ne na disku). Konceptualni prikaz HDFS-a dat je na sliciSlika 53:



Slika 53 Hadoop Distributed File System - HDFS

Kada klijentski čvor treba da kreira nov fajl, komunicira sa imenskim čvorom (name-node-om). Ovaj čvor onda:

- Dodaje ime novog fajla metapodacima.
- Ustanovljava novi broj bloka za fajl.
- Određuje listu čvorova podataka u kojima će blok biti sačuvan.
- Plasira tu informaciju nazad ka klijentskom čvoru.

Klijentski čvor kontaktira prvi čvor za podatke koji je naveden od strane imenskog čvora (name node) i započinje upisivanje fajla na taj čvor za podatke. Istovremeno, klijentski čvor šalje čvoru za podatke listu ostalih čvorova za podatke koji će replicirati blok. Kako čvor za podatke prima podatke od klijentskog čvora, on kontaktira sledeći čvor za podatke sa liste i počinje da šalje podatke tom čvoru radi replikacije. Taj drugi čvor za podatke zatim kontaktira sledeći čvor sa liste i proces se nastavlja tako što se podaci prenose kroz sve čvorove za podatke koji čuvaju blok.

Kada je prvi blok zapisan, klijentski čvor može dobiti broj sledećeg bloka i listu čvorova za podatke od čvora imena za naredni blok. Kada je ceo fajl upisan, klijentski čvor obaveštava imenski čvor da je fajl zatvoren. Važno je napomenuti da ni u jednom trenutku nijedan deo fajla nije bio prenet na čvor imena. Ovo pomaže u smanjenju protoka podataka ka imenskom čvoru kako bi se izbegla zagušenja koja bi mogla usporiti performanse sistema.

Slično, ako klijentski čvor treba da pročita fajl, on kontaktira imenski čvor kako bi zatražio listu blokova povezanih s tim fajlom i čvorova za podatke koji ih drže. S obzirom na to da se svaki blok može nalaziti na više čvorova za podatke, za svaki blok klijent pokušava da ga preuzme sa čvora za podatke koji mu je najbliži na mreži. Koristeći ove informacije, klijentski čvor direktno čita podatke sa svakog od tih čvorova.

Periodično, svaki čvor za podatke komunicira sa imenskim čvorom. Čvorovi za podatke šalju izveštaje o blokovima i tzv. heartbeat signale. Izveštaj o blokovima se šalje svakih 6 sati i obaveštava imenski čvor o tome koji se blokovi nalaze na tom čvoru za podatke. Heartbeat signal se šalje na svake 3 sekunde. Ovaj signal omogućava čvoru imena da zna da je čvor za podatke još uvek dostupan.

Ako čvor za podatke doživi kvar, na primer zbog hardverskog otkaza, nestanka struje i slično, čvor imena neće primiti heartbeat signal od tog čvora za podatke. Kao rezultat toga, čvor imena zna da ne treba da uključi taj čvor za podatke u liste koje šalje klijentskim čvorovima za čitanje ili upisivanje fajlova. Ako izostanak heartbeat signala uzrokuje da blok ima manje replika nego što je željeni broj, čvor imena može naložiti nekom aktivnom čvoru za podatke da započne replikaciju tog bloka na drugi čvor za podatke.

Zajedno, komponente HDFS-a čine moćan, ali visoko specijalizovan distribuirani sistem fajlova koji dobro funkcioniše za specijalizovane zahteve obrade podataka u aplikacijama za „velike podatke“ (Big Data).

HDFS i MapReduce predstavljaju okosnicu modernih sistema za analitiku u paradigmi Big Data, Hadoop (Sandhu, 2022).

6.3. Baze podataka

Baze podataka su nezaobilazna komponenta bilo kakvog modernog softvera. Sa „klaudizacijom“ aplikacija, baze se sele u oblak i postaju deo integralne infrastrukture zasnovane na uslugama u oblaku. Kao i kod drugih usluga, svojstva oblaka i kod baza pružaju prednosti brzog skaliranja, lakšeg obezbeđivanja rezervnih kopija i replikacije baze i uopšte rukovanja velikim količinama strukturiranih i nestrukturiranih podataka, kao i dodatne opcije zaštite podataka.

Kada je reč o bazama podataka u oblaku, u opticaju su dva rešenja. Kod prvog, reč je o konvencionalnim sistemima za upravljanje bazama podataka, koji se izvršavaju na virtualnoj infrastrukturi, po modelu infrastrukture kao usluge (IaaS), sa svim svojstvima koje inače ovakvi sistemi imaju na posebnim računarima ili serverima. Baze podataka — kao što su MySQL, PostgreSQL ili MongoDB — hostuju se na virtuelnim mašinama u oblaku. U ovom modelu:

- Korisnik ima punu kontrolu nad serverskim okruženjem baze.
- Softver baze mora biti ručno instaliran, podešen i održavan.
- Bekapovanje, nadogradnje, podešavanje performansi i skaliranje su odgovornost korisnika.

Ovaj model često biraju timovi koji zahtevaju specifične konfiguracije, imaju aplikacije koje zavise od starijih sistema ili žele potpunu fleksibilnost. Pogodan je i za migraciju postojećih sistema u oblak bez promene arhitekture aplikacije.

Međutim, ovakav pristup zahteva više operativnog rada i znanja. Skaliranje kapaciteta ili obezbeđivanje visoke dostupnosti uglavnom podrazumeva ručne intervencije, balansiranje opterećenja i upravljanje replikama. Tradicionalni modeli baza u oblaku jesu pogodne za migraciju, kada se kompletni serveri prenose po sistemu lift&shift na virtuelne mašine.

Drugo rešenje jeste baza podataka kao servis (DBaaS), gde se koriste upravljane usluge: korisnik ne instalira sistem za upravljanje bazama već koristi gotove usluge dostupne u oblaku i treba samo da kreira bazu, pristupi joj, izvrši upite itd. Nema nikakvog posla vezanog za administraciju samog sistema za upravljanje. Glavne karakteristike su:

- Bez brige o infrastrukturi: Dobavljač obezbeđuje automatsko podešavanje, skaliranje, bekapovanje, ažuriranje i oporavak.
- Plaćanje prema korišćenju: Naplata se vrši po kapacitetu skladištenja i broju operacija, često uz mogućnost automatskog skaliranja.
- Brza implementacija i integracija: Baza se može postaviti za nekoliko minuta pomoću šablona i grafičkih interfejsa.

Upravljane baze u oblaku podržavaju i SQL i NoSQL modele. Suprotno uvreženom mišljenju da su upravljane usluge pogodne samo za NoSQL, moderne platforme podržavaju širok spektar relacionih i nerelacionih baza.

Primeri upravljanih usluga na AWS-u:

- Amazon RDS (Relational Database Service): Podržava SQL baze kao što su MySQL, PostgreSQL, Oracle i SQL Server. Automatski upravlja bekapovima, nadogradnjama, skaliranjem i replikacijom. Omogućava više modela implementacije (jedna dostupnost zona, više zona, read-replica).
- Amazon Aurora: Visokoperformantna, cloud-native baza kompatibilna sa MySQL i PostgreSQL. Aurora nudi veću skalabilnost i otpornost od klasičnih baza, uz zadržavanje poznatog SQL modela.
- Amazon DynamoDB: Upravljana NoSQL baza podataka sa ključ-vrednost i dokument modelom. Obezbeđuje nisku latenciju, automatsko skaliranje i ugrađene sigurnosne mehanizme poput enkripcije i IAM kontrole pristupa.

U tabeli 7 sumirani su kriterijumi za izbor pristupa: tradicionalna baza ili upravljana baza.

Tabela 7 Kriterijumi za izbor baze podataka

Svojstvo	**Tradicionalne baze u VM**	**Upravljane cloud-native baze**
Kontrola	Puna	Ograničena na konfiguraciju
Održavanje	Ručno	Automatizovano
Skalabilnost	Ručno	Automatski
Optimizacija troškova	Ručna	Ugrađena u servis
Tipični slučajevi upotrebe	Nasleđene aplikacije, fleksibilnost	Moderne aplikacije, brz razvoj

Pitanja za proveru i obnavljanje gradiva

1. Da li se magnetne trake i dalje koriste za skladištenje podataka?
2. Koji interfejsi omogućavaju povezivanje većeg broja diskova?
3. Da li putem RAID-a može da se ostvari istovremeno i brzina i pouzdanost? Obrazložite.
4. Objasnite razliku između SAN-a i NAS-a.
5. Koji su to specifični zahtevi fajl-sistema oblaka u odnosu na konvencionalne sisteme?
6. Kako se odvijaju faze MapReduce obrade?
7. Kada se menjaju podaci kod GFS-a, da li se stari podaci na disku prepisuju novim? Objasnite.
8. Kakve opcije postoje kod migracije MySQL baze u oblak?

6.4. Skladišta i AWS

Skladištenje podataka u cloudu (Cloud Storage) predstavlja model skladištenja podataka gde se oni čuvaju na serverima u cloudu. Tim serverima se pristupa uglavnom putem interneta, uz mogućnosti pravljenja virtuelnih privatnih mreža između kancelarija ili on-premise data centara i clouda. Organizacije i pojedinci mogu koristiti lokalne servere ili fizičke diskove za čuvanje podataka, kao i

cloud skladištenje koje im omogućava fleksibilnost, skalabilnost i sigurnost, bez potrebe za održavanjem hardverske infrastrukture.

Ključne prednosti skladištenja u oblaku:

- **Dostupnost i skalabilnost** – Servisi za skladištenje podataka su dostupni globalno i može se povećavati kapacitet prema potrebama korisnika.
- **Visoka pouzdanost** – Cloud provajderi, kao AWS, štite podatke neuporedivom bezbednošću i skladištem dizajniranim za 99,999999999 (11x9) izdržljivosti i otpornosti na više zona dostupnosti. Podaci su dostupni aplikacijama uz otpornost podataka obezbeđenu kroz opcije backup-a i recovery-ja u AWS i lokalnim (on-premise) okruženjima.
- **Bezbednost** – Cloud provajderi štite podatke na više načina, uz pomoć napredne enkripcije i kontrole pristupa kada su smešteni (at rest), kao i enkripcije u prenosu podataka (in transit).
- **Efikasno upravljanje troškovima** – Ne postoje inicijalna ulaganja u hardver i plaća se samo onoliko koliko se koristi; zavisno od tipa skladištenja, može i plaćati na osnovu količine pristupanja podacima.

AWS servisi za skladištenje podataka

AWS nudi različite servise za skladištenje podataka, zavisno od toga kakav je workload, pa samim tim su i servisi optimizovani.

AWS S3

Amazon S3 (Simple Storage Service) je objektno skladištenje namenjeno za čuvanje velikih količina podataka, sa više klasa skladištenja i mogućnostima verzionisanja. S3 može skalirati do ogromnih količina podataka sa visokim nivoom izdržljivosti, dostupnosti i performansi. Podacima koji se nalaze na S3 se može pristupiti sa bilo kog mesta putem interneta, preko Amazon konzole i jako dobrog S3 API.

Neke od ključnih karakteristika S3 skladišta:

- Buckets – Podaci se čuvaju u tzv bucket-ima. Svaki bucket može da sačuva neograničenu količinu nestrukturiranih podataka.
- Elastična skalabilnost – S3 nema ograničenja za količinu prostora za skladištenje. Objekat može biti veličine do 5TB.
- Fleksibilna struktura podataka – Svaki objekat se identifikuje pomoću jedinstvenog ključa i mogu se koristiti metapodaci da bi se organizovali podaci u okviru samih bucket-a.
- Preuzimanje podataka – Podaci se mogu lako deliti unutar ili van organizacije. Postoji mogućnost deljenja sa drugim AWS nalozima, kao i generisanje *pre-signed* URL-a za deljenje podatka na određeno vreme. Podaci mogu biti podešeni i da su javno dostupni za pregled i preuzimanje što može biti pogodno za hosting statičkih web sajtova.
- Dozvole – Dodela dozvola može biti na nivou bucket-a ili objekta unutar bucket-a kako bi se imala potpuna kontrola nad podacima. Postoji i kontrola akcija nad podacima, tako da se može ograničiti brisanje, prepisivanje i ostale akcije nad podacima.
- API – S3 API, dostupan kao REST i SOAP interfejs, odavno je industrijski standard i integrisan je sa mnogim drugim servisima, SDK-ovima itd.

S3 podžava i klase (nivoe) skladištenja podataka, oni se mogu primeniti na nivou bucket-a, dela bucket-a (određene putanje), kao i samog objekta. Postoji i mogućnost podešavanja pravila (polisa) životnog ciklusa podataka koja bi automatski premeštala objekte između nivoa, na osnovu definisanih pravila.

Glavne klase skladištenja su:

- **S3 Standard** – 99.999999999% dostupnost i trajnost garantovani na više uređaja u više objekata i dizajnirani su da istovremeno podnose gubitak 2 zone dostupnosti.
- **S3 IA** – **Infrequently Accessed** – Za podatke kojima se pristupa ređe, ali zahtevaju brzi pristup kada je to potrebno.
- **S3 One Zone IA** – Jeftinija opcija za podatke kojima se retko pristupa, ali ne postoje u više zona dostupnosti.
- **S3 Intelligent Tiering** – Dizajniran za optimizaciju troškova automatskim premeštanjem podataka na najisplativiji nivo skladištenja.
- **S3 Glacier** – Sigurna izdržljiva i jeftina klasa skladišta za arhiviranje podataka. Vreme preuzimanja podesivo od minuta do sati.
- **S3 Glacier Deep Archive** – Prihvatljivo je vreme vraćanja podataka od 12 sati.

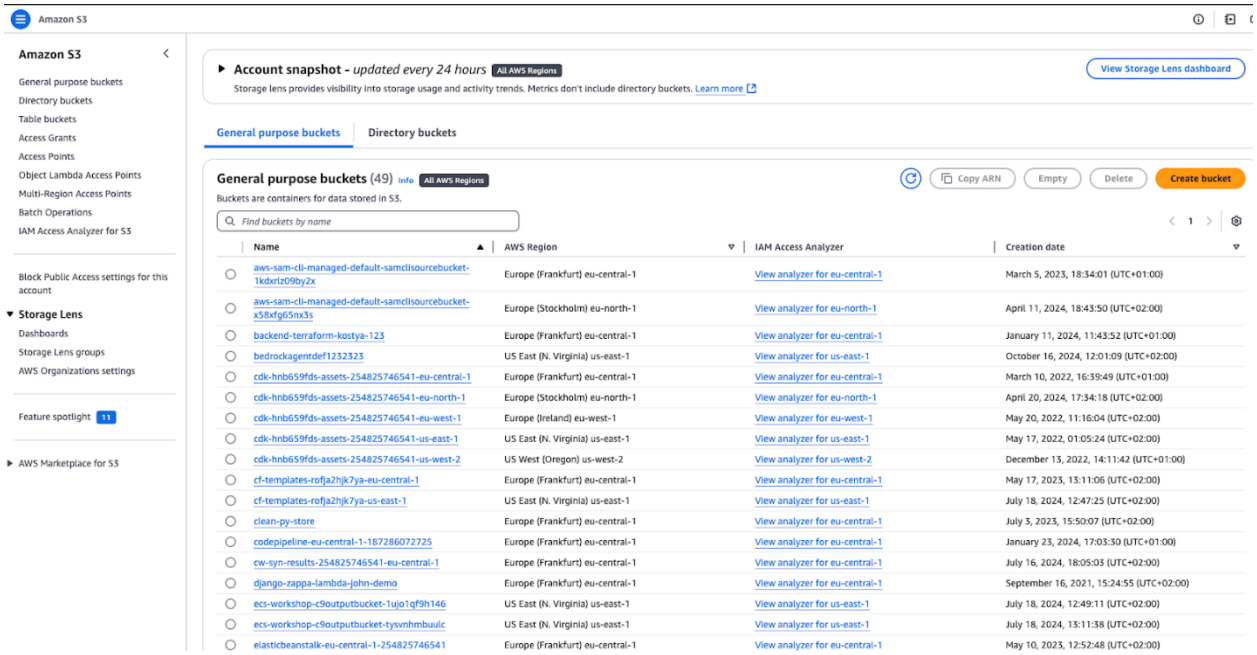
Tabela 8 Prikaz S3 nivoa skladištenja i njihovih karakteristika

	S3 Standard	S3 Intelligent-Tiering	S3 Standard-IA	S3 One Zone-IA	S3 Glacier	S3 Glacier Deep Archive
Dizajnirano za izdržljivost	99.999999999% (11 devetki)	99.999999999% (11 devetki)	99.999999999% (11 devetki)	99.999999999% (11 devetki)	99.999999999% (11 devetki)	99.999999999% (11 devetki)
Dizajnirano za dostupnost	99.99%	99.9%	99.9%	99.5%	99.99%	99.99%
Dostupnost SLA	99.9%	99%	99%	99%	99.9%	99.9%
Dostupne zone	≥3	≥3	≥3	1	≥3	≥3
Minimalna naplata kapaciteta po objektu			128KB	128KB	40KB	40KB
Minimalna naplata trajanja skladištenja		30 dana	30 dana	30 dana	90 dana	180 dana

Naknada za preuzimanje			po preuzetom GB	po preuzetom GB	po preuzetom GB	po preuzetom GB
Latencija prvog bajta	milisekunde	milisekunde	milisekunde	milisekunde	nekoliko minuta ili sati	nekoliko sati
Prelazi životnog ciklusa	Da	Da	Da	Da	Da	Da

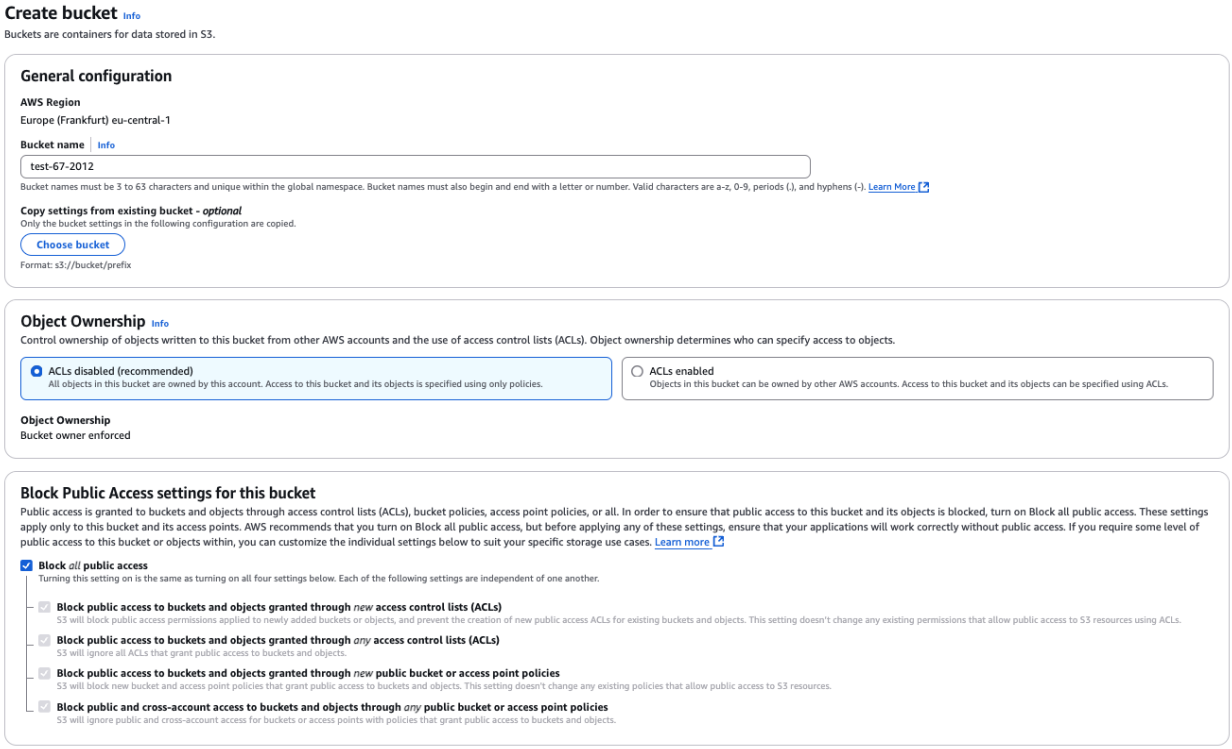
U narednim koracima (slika Slika 54) je prikazano pravljenje jednog S3 bucket-a kao i podešavanje parametara pristupa.

Potrebno je odabrati S3 servis na Search bar-u ili prikazu svih servisa.

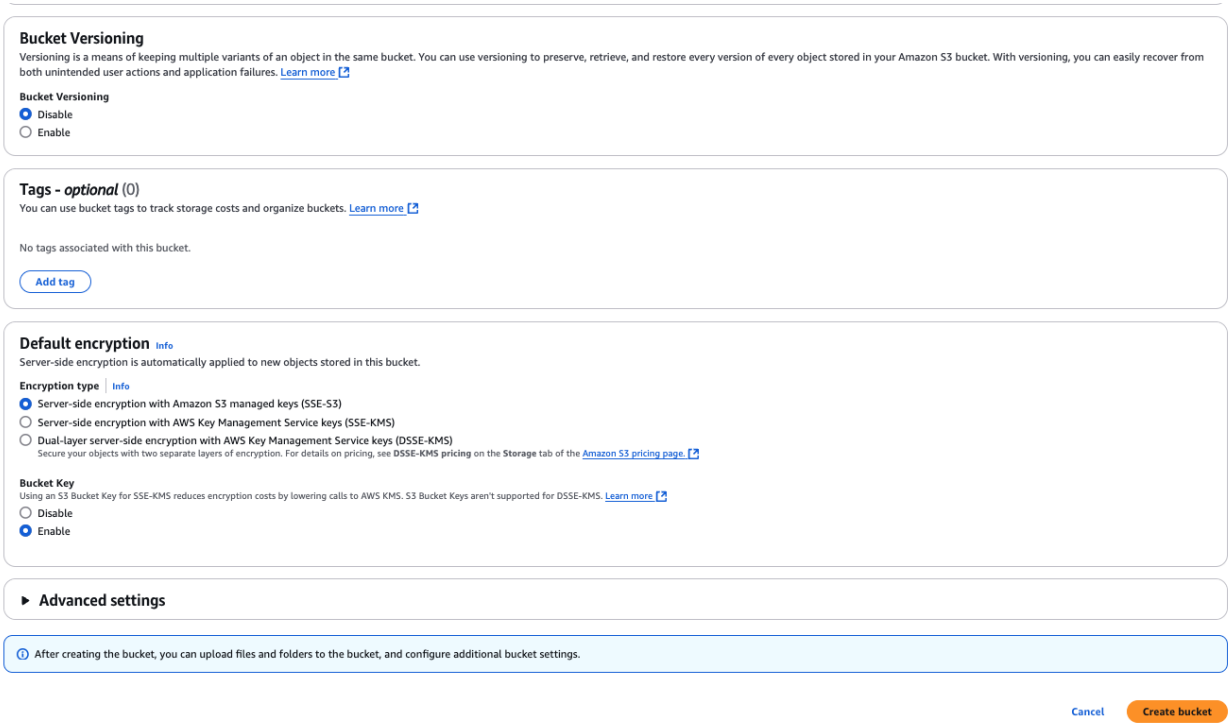


Slika 54 Prikaz S3 interfejsa na AWS konzoli

Zatim se ide na Create bucket i popunjavaju polja (Slika 55 i Slika 56). U ime bucket-a je bitno uneti jedinstveno ime i dva bucket-a ne mogu imati isto ime, ne samo u okviru jednog AWS naloga, već globalno; kao primer jedinstvenih imena se obično koristi kombinacija naziva koji ima smisla i nastavka koji može biti nasumično generisan hash ili identifikator AWS naloga, na primer data-bucket-12345abcdef ili data-bucket-254825746541 (identifikator naloga). U prikazanom primeru je korišćen broj indeksa kao nastavak na ime.



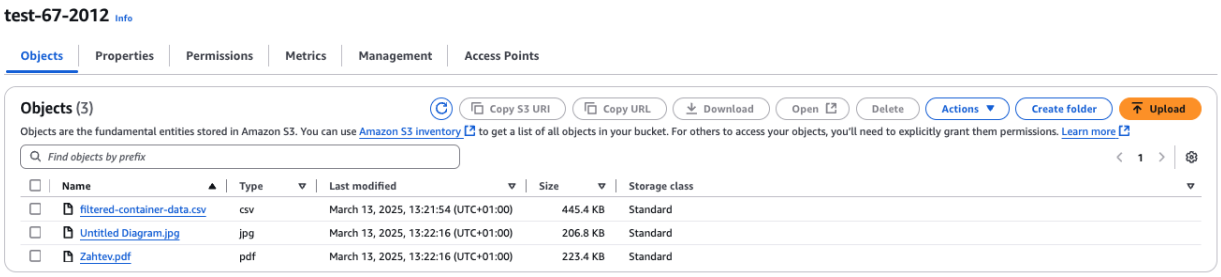
Slika 55 Kreiranje S3 bucket-a – prvi deo ekrana



Slika 56 Kreiranje S3 bucket-a – drugi deo ekrana

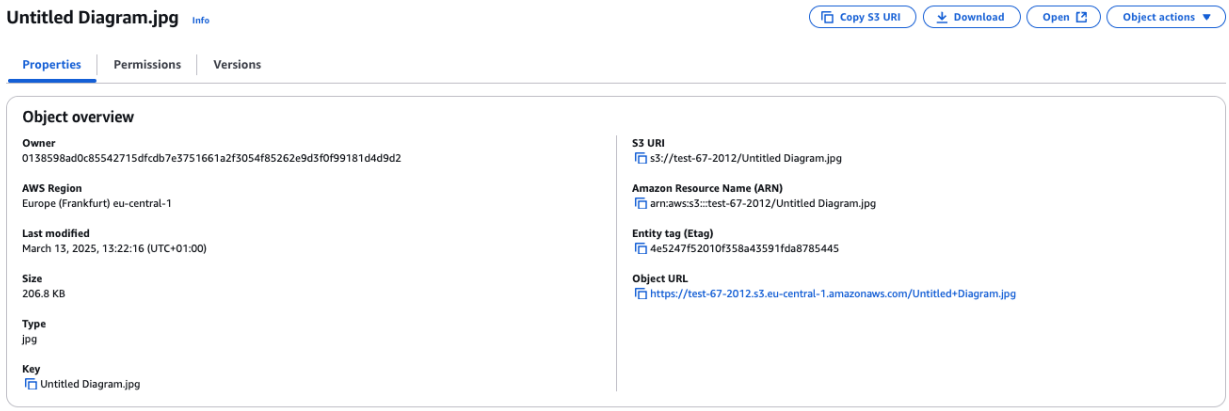
Nakon podešavanja parametara potrebno je napraviti bucket klikom na Create bucket. Ukoliko je ime jedinstveno, trebalo bi da se dobije poruka koja potvrđuje pravljenje bucket-a.

Na slici Slika 57 su prikazana tri fajla koja su uploadovana na S3 bucket. Svaki fajl ima svoje metapodatke, kao i parametre u S3.



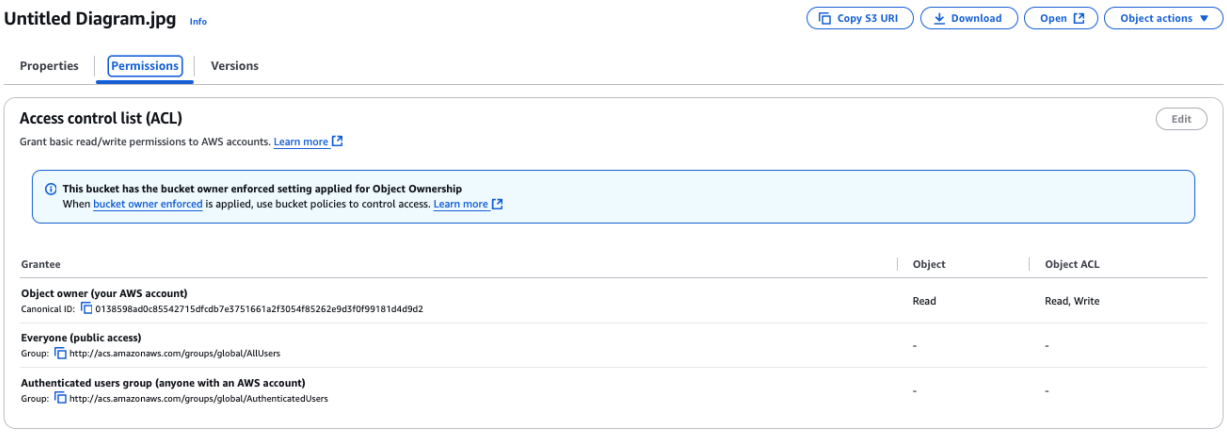
Slika 57 Fajlovi postavljeni na S3 bucket

Klikom na bilo koji od fajlova (objekata) mogu se dobiti dodatne informacije o njemu (slika 58).



Slika 58 Detalji fajla u S3 bucket-u

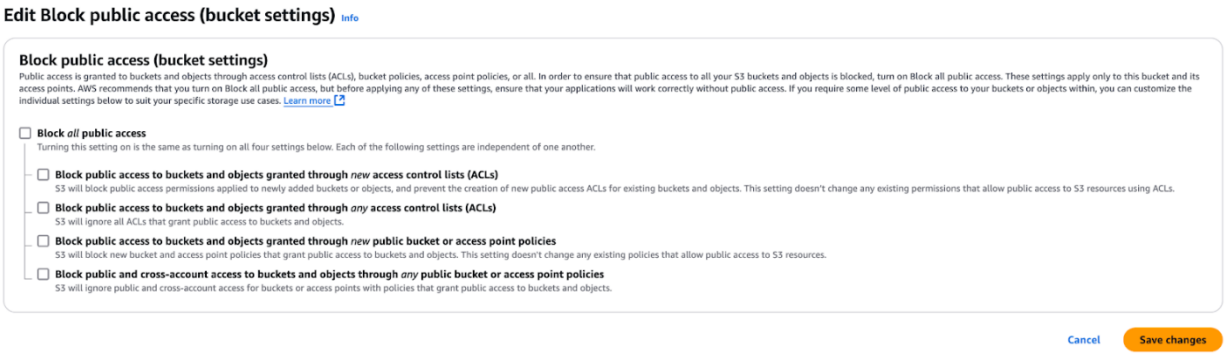
U okviru tabova tog objekta se vidi kartica sa dozvolama i verzijama (Slika 59); ukoliko je *bucket versioning* uključen, isti fajl može se postavljati više puta, svaka verzija će biti sačuvana i postojaće mogućnost pristupa prethodnim verzijama, dok će podrazumevano biti preuzeta verzija koja je poslednja, tj. najnovija.



Slika 59 Prava pristupa nad fajlom u S3 bucket-u

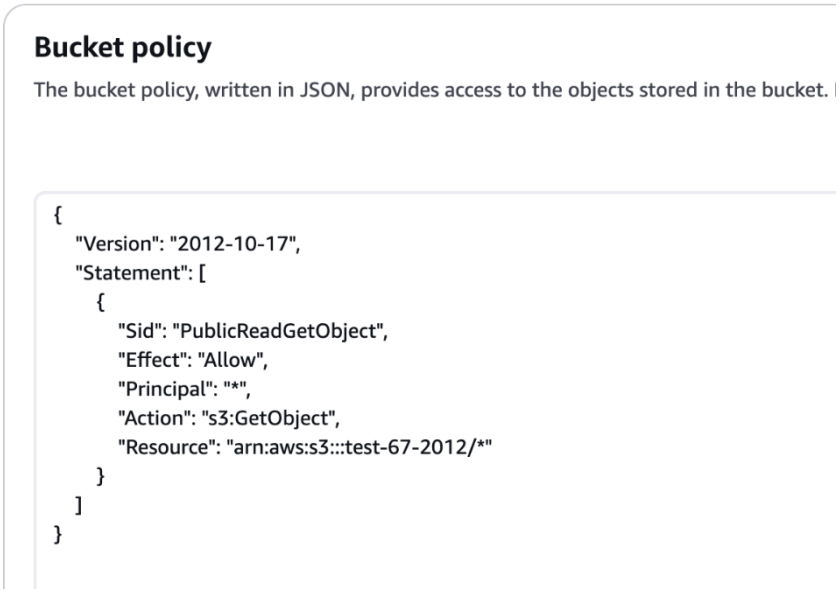
S3 ima još jednu jako zanimljivu mogućnost, a to je sposobnost da se hostuju statički web-sajtovi na samom S3 bez potrebe za dodatnim serverima, već se jednostavno može koristiti S3 uz određenja podešavanja za pristup i pravila rutiranja.

U okviru Permissions taba u S3 bucket podešavanjima potrebno je isključiti sva ograničenja na Block public access (bucket settings) kako bi se omogućio javni pristup (Slika 60).



Slika 60 Isključivanje ograničenja javnog pristupa na S3 bucket-u

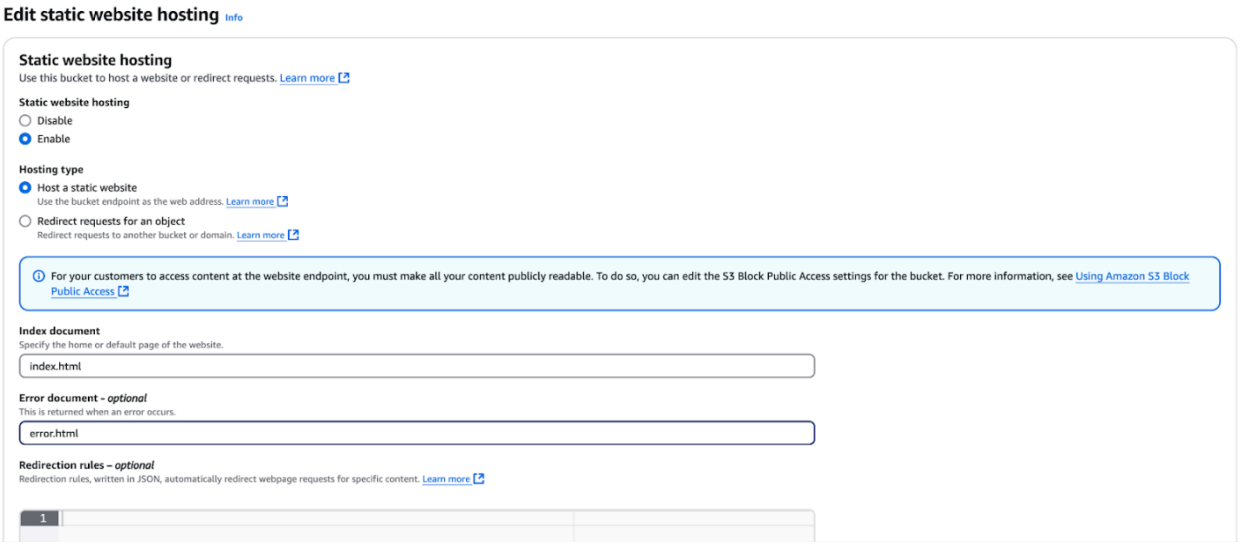
Zatim, na podešavanju Bucket policy (odmah ispod prethodnog podešavanja; Slika 61) potrebno je dodati polisu koja će osigurati da je javni pristup moguć samo za čitanje objekata. Ova polisa je standardna i može se naći u okviru AWS dokumentacije i prekopirati u polje.



Slika 61 S3 polisa za javni pristup čitanju objekata

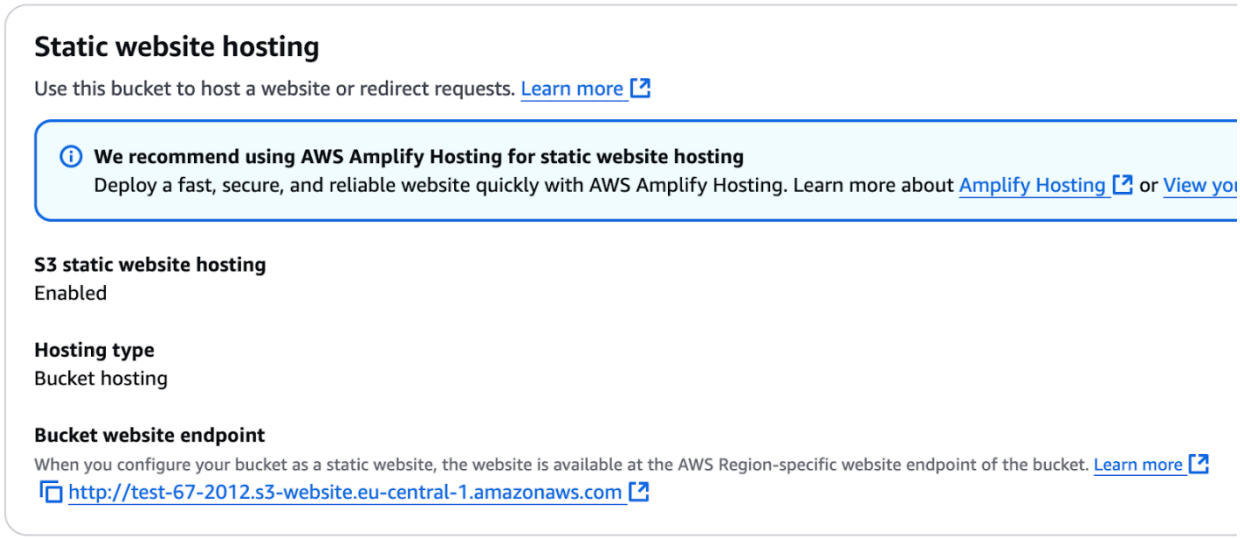
Napomena: U okviru Resource dela polise potrebno je kopirati ARN od bucket-a u kom se podešava. Nakon ovoga treba sačuvati podešavanje polise.

Sledeći korak je pristupanje Properties tabu i na samom dnu stranice pronaći Static website hosting opciju. U okviru podešavanja, potrebno je popuniti naziv index i error strana (ukoliko error postoji, moguće je upisati da u slučaju greške vrati na index stranu), kao i pravila redirekcije ukoliko ona postoje (Slika 62).

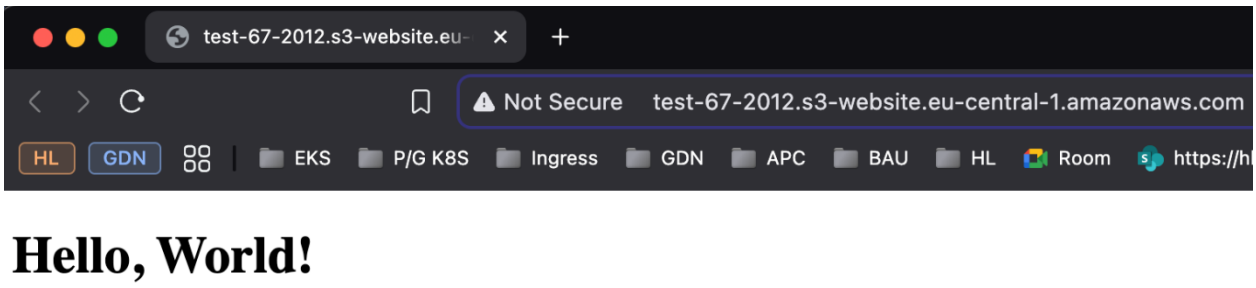


Slika 62 Static website hosting podešavanje u okviru S3 servisa

Nakon toga, na S3 možemo izvršiti upload index.html, kao i ostalih stranica web sajta. Link za pristup web sajtu se nalazi u Properties tabu, na samom dnu, nakon podešavanja static website hostinga (Slika 63).



Slika 63 Podešen static website hosting i URL do pristupa



Slika 64 Pristup index stranici putem URL-a od S3 bucket-a

Pristupom adresi se prikazuje web sajt ukoliko su sva podešavanja pravilno unesena (Slika 64). U slučaju pristupa stranici koja ne postoji ili pogrešnom podešavanju error stranice, dobiće se 404 Not found greška.

Napomena: Neki browseri mogu pokušati da rutiraju HTTP request u HTTPS, što sam S3 ne podržava jer nema SSL sertifikat vezan za sajt. Potrebno je proveriti da sigurno pristupate HTTP URL-u. Ukoliko je potrebno dodati sertifikat i napraviti secure website hostovan u S3, to se može uraditi pomoću CloudFront servisa, kao i ACM (AWS Certificate Manager) za generisanje i automatsko obnavljanje SSL sertifikata.

AWS EBS

Elastična blok memorija (EBS) je izdržljiva i trajna memorija tipa bloka koja se može povezati na EC2 instance radi dodatnog skladištenja. Za razliku od standardnih EC2 prostora za skladištenje koji su pogodni za čuvanje privremenih podataka, EBS su veoma pogodni za osnovne i dugoročne podatke. EBS su specifični za zone dostupnosti i mogu biti pridruženi samo instancama unutar iste zone dostupnosti.

EBS se može kreirati sa EC2 interfejsa u AWS konzoli, kao i u koraku pravljenja EC2 instance, kao što je prikazano u prethodnom poglavlju. Kada se napravi EBS sa EC2, on se pravi u istoj zoni dostupnosti kao i EC2 instanca, međutim, kada se naprave nezavisno, korisnici mogu da izaberu zonu dostupnosti u kojem je EBS potreban.

Neke od EBS karakteristika:

- **Skalabilnost**: Veličina i mogućnosti EBS-a skaliraju prema potrebama.
- **Backup**: Korisnici mogu da prave snapshotove EBS diskova. Oni se mogu zakazati da se automatski prave u određenom intervalu ili ručno praviti. Skladište se u S3 kako bi smanjili troškove čuvanja snapshotova. Snapshotovi su inkrementalni i mogu se kopirati kako bi se napravio EBS disk u drugoj zoni dostupnosti.
- **Enkripcija**: EBS nudi šifrovanje diskova pomoću AWS servisa za šifrovanje (KMS – Key Management Service), vrlo lako se uključuje pri samom pravljenju EBS diska. Snapshotovi koji nastanu od šifrovanih diskova su istim tim ključem šifrovani.
- **Troškovi:** EBS se naplaćuje prema rezervisanom prostoru za taj konkretan disk, bez obzira na zauzetost tog rezervisanog prostora.

EBS diskovi su nezavisni od EC2 instanci na koje se povezuju i podaci na njima ostaju nepromenjeni ukoliko dođe do gašenja ili restarta EC2 instance na koju je povezan. Jedan EBS volume može biti povezan samo na jednu EC2 instancu, dok jedna EC2 instanca može imati više EBS diskova povezanih na sebe. Diskovi su vezani za zonu dostupnosti u kojoj su napravljeni i mogu se samo povezati na EC2 instance iz iste zone dostupnosti. Postoji replikacija podataka EBSa; međutim, ukoliko dođe do pada zone dostupnosti, svakako će pristup EBS disku biti nemoguć.

Tipove EBS diskova možemo svrstati u SSD i HDD. SSD diskovi su pogodni za „sitnije“ podatke gde je brzina pristupa i broj IOPS značajan; ovaj tip skladištenja je pogodan za glavne, tj root diskove EC2 instanci. HDD je pogodniji za „krupnije” podatke i sporiju obradu; on se ne može koristiti kao glavni disk EC2 instance.

Tipovi SSD diskova:

- General Purpose SSD (GP2/3) – Vreme pristupa u milisekundama, može da pruži visok IOPS (16 000), tipičan propusni opseg ide od 128Mbps do 170Gbps.
- Provisioned IOPS SSD (IO1/2) – Korisnici pri pravljenju definišu koji im je IOPS potreban. Veličine idu od 4TB do 16TB s maksimalnom IOPS od 20 000.

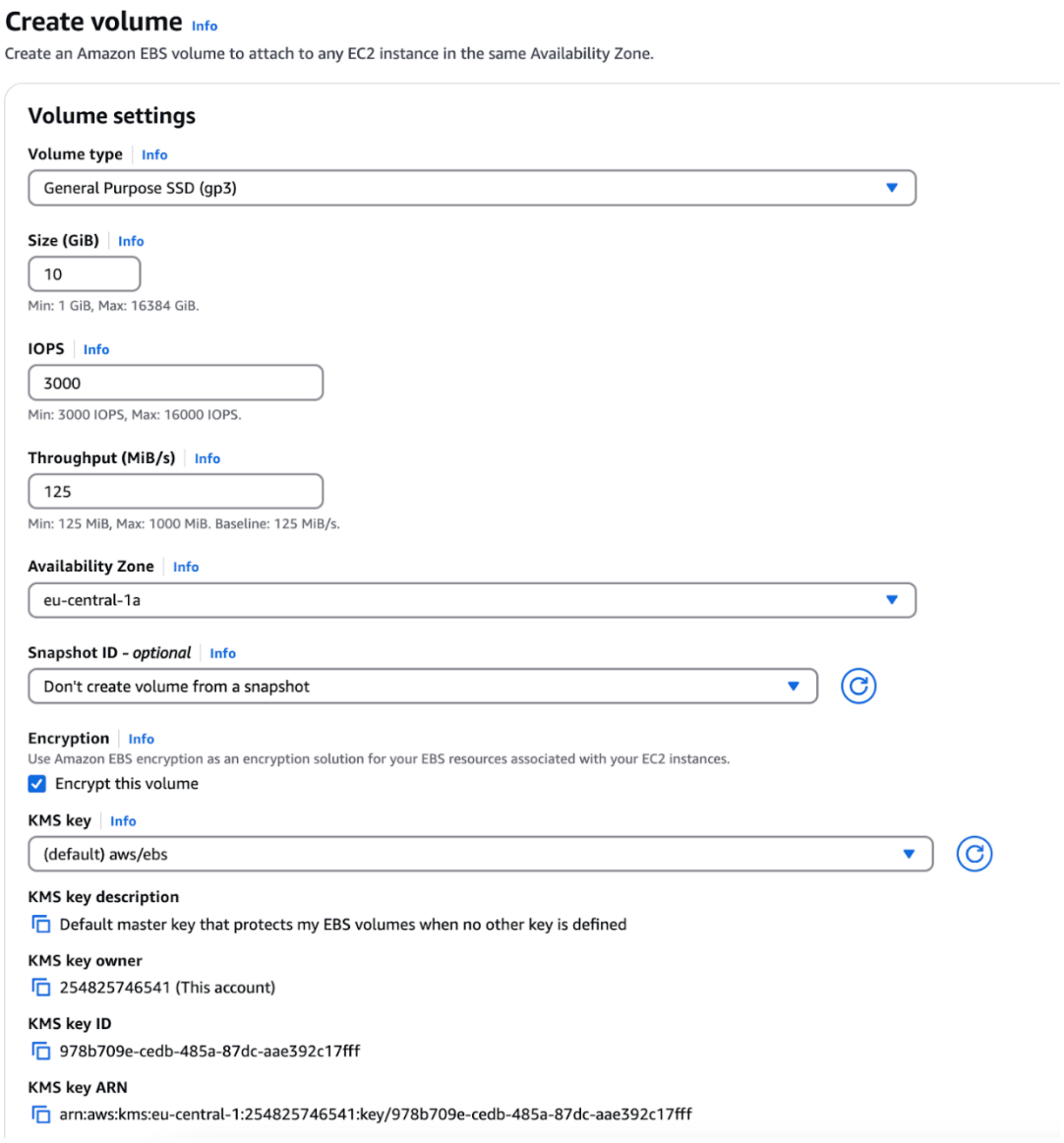
Tipovi HDD diskova:

- Cold HDD (SC1) – Ovo je najjeftiniji tip EBS diskova, pogodan za veće podatke kojima se ne pristupa često.
- Throughput optimized HDD (ST) – Pogodan za veće podatke kojima se često pristupa, podesiv od 250 Mbps do 500 Mbps.

Postoji i opcija Magnetic (Previous generation) koja predstavlja originalnu opciju EBS diskova, a ujedno i najsporiju i najjeftiniju. Idealno služe za najmanje bitne podatke kojima se retko pristupa. Mogu biti veličine od 1GB to 1TB i performanse se ne povećavaju sa veličinom diska već ostaju iste, oko 200 IOPS i 40 – 90Mbps.

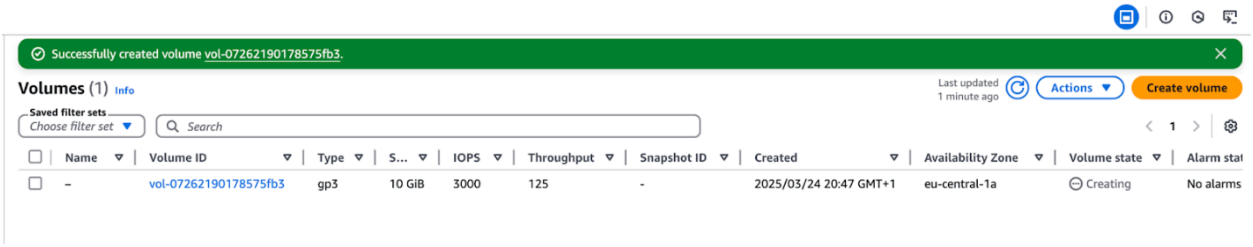
U narednim koracima je prikazano pravljenje jednog EBS diska (Slika 65).

Potrebno je pristupiti AWS konzoli i izabrati EC2 u listi servisa. Zatim u bočnom meniju, ispod Elastic Block Store, izabrati Volumes. Tu se nalazi prikaz svih EBS diskova; kako biste napravili novi, potrebno je izabrati opciju Create volume.



Slika 65 Podešavanje GP3 diska

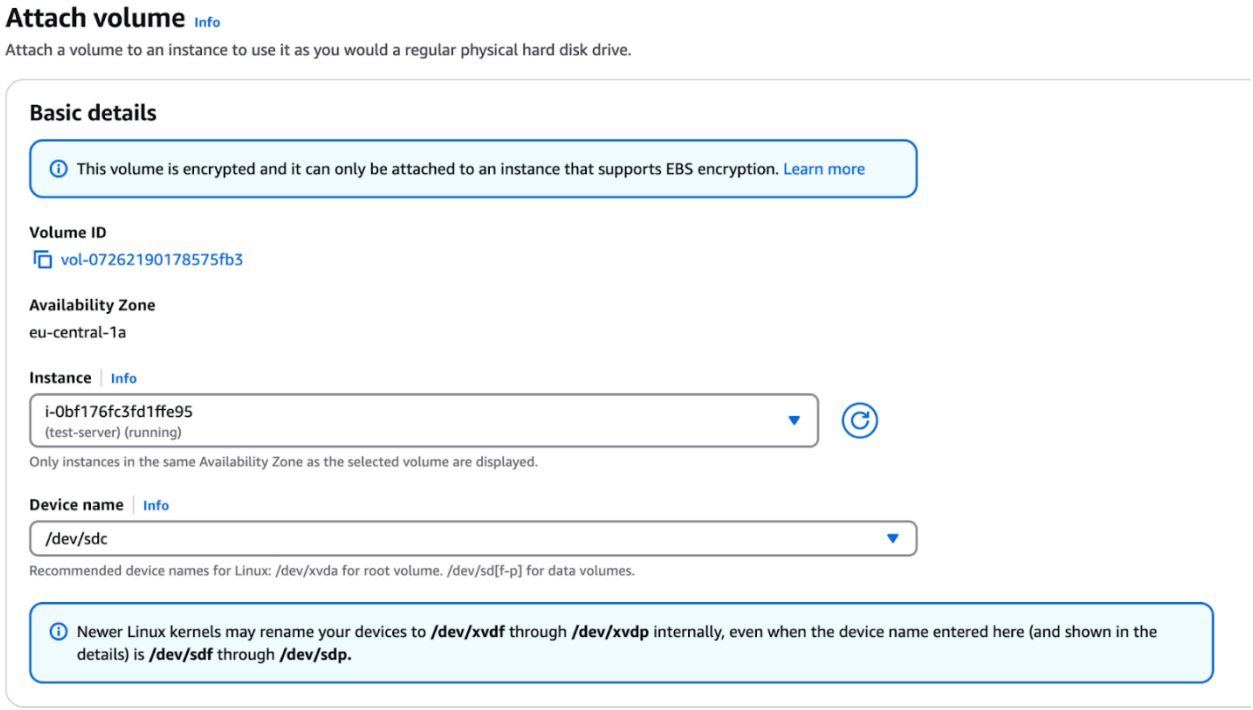
Nakon pravljenja diska potrebno je sačekati da se *volume state* prebaci u Available (Slika 66).



Slika 66 Volume u pripremi (Creating state)

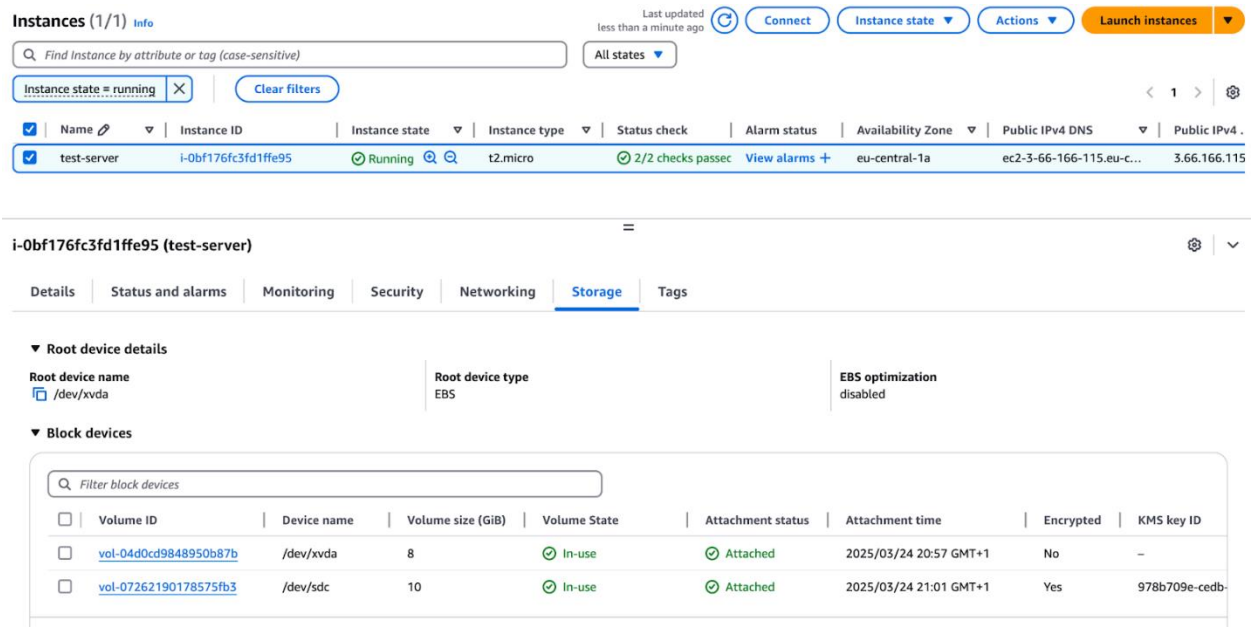
Kada status postane „Available", to znači da je disk spreman za povezivanje na EC2 instancu. To možemo uraditi kroz Actions meni. Napomena: Pogledati prethodno poglavlje u kom je opisano kako pokrenuti EC2 instancu.

Odabirom kreiranog EBS diska i klikom na Actions i Attach Volume dolazimo do interfejsa za povezivanje EBS diska i EC2 instance (Slika 67).



Slika 67 Povezivanje EBS sa EC2 instancom

Ukoliko se ne prikazuje EC2 instanca u opcijama ispod Instances, potrebno je proveriti da li su EC2 instanca i EBS volume u istoj zoni dostupnosti. Zatim se ide na Attach Volume i momentalno se povezuje EBS sa EC2. Povezanost se može proveriti kroz Volumes ili kroz Instance interfejs i Storage tab kao što je prikazano na slici Slika 68.



Slika 68 Prikaz povezanog EBS sa EC2

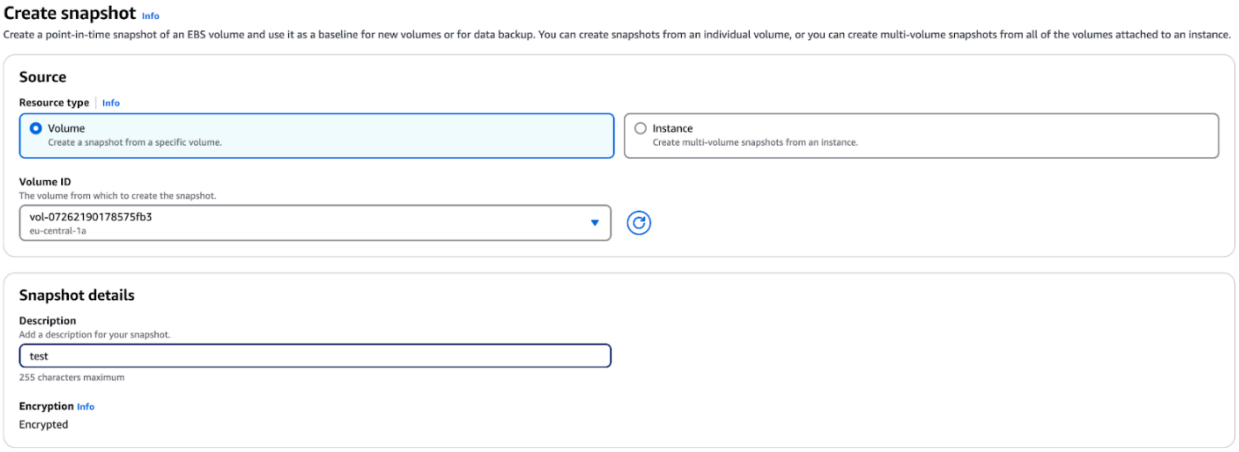
Povezivanjem putem SSH ili Instance Connect opcije možemo proveriti da li se vidi disk na samoj instanci.

```
[root@ip-172-31-28-6 ~]# lsblk
NAME      MAJ:MIN RM SIZE RO TYPE MOUNTPOINTS
xvda      202:0    0   8G  0 disk
├─xvda1   202:1    0   8G  0 part /
├─xvda127 259:0    0   1M  0 part
└─xvda128 259:1    0  10M  0 part /boot/efi
xvdc      202:32   0  10G  0 disk
[root@ip-172-31-28-6 ~]#
```

Slika 69 Prikaz EBS diska povezanog na EC2

Na slici Slika 69 se vidi 10 GB disk povezan na EC2 instancu (xvdc).

Prelaskom na Snapshots može se pokrenuti proces pravljenja snapshota diska; potrebno je odabrati disk od kog se pravi i uneti opis (Slika 70).



Slika 70 Pravljenje Snapshota EBS diska

Nakon odabira opcije Create Snapshot potrebno je sačekati par minuta dok proces ne bude gotov i snapshot status ne pređe u Completed (Slika 71).

Slika 71 Spreman Snapshot EBS diska

Nakon toga može se dalje manipulisati snapshotom – on se može se kopirati, premestiti se u arhivu, mogu se praviti novi diskovi, itd.

AWS EFS

EFS spada u kategoriju skladištenja fajlova. EFS je sistem za skladištenje na nivou fajlova, u potpunosti upravljan od strane AWS-a. Može mu se istovremeno pristupati sa više EC2 instanci, kao i drugih AWS servisa. Baš kao i AWS EBS (Elastic Block Store), EFS je posebno dizajniran za aplikacije koje zahtevaju veliki protok podataka (high throughput) i nisku latenciju.

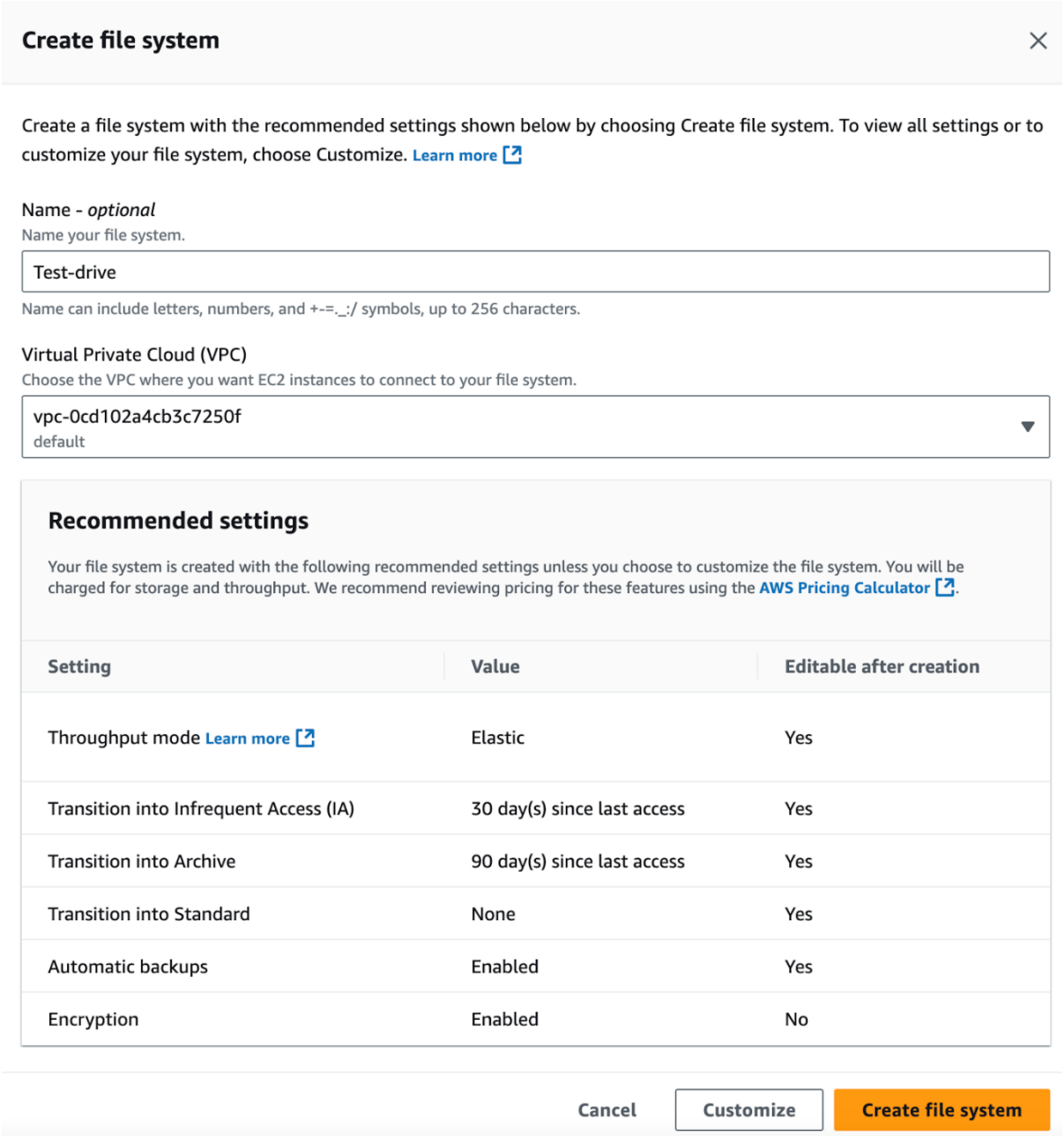
EFS se može napraviti pomoću EFS servisa, gde će biti napravljen unutar određene regije i raspoređen kroz više zona dostupnosti kako bi se postigla visoka dostupnost i otpornost sistema. Možete izabrati EFS sistem u zavisnosti od količine IOPS koje su planirane za korišćenje.

Neke karakteristike EFS su:

- Upravljan servis (Managed) – Nema potrebe za održavanjem jer AWS automatski upravlja skaliranjem, dostupnošću i održavanjem.
- Skaliranje – Automatski se prilagođava kapacitet i nema potrebe za definisanjem limita
- Deljenje pristupa – Istovremeno može više servisa, kao EC2, da se poveže na njega korišćenjem NFS protokola.
- Visoka dostupnost – Podaci se automatski raspoređuju u više zona dostupnosti u okviru AWS regiona, što ujedno povećava i zaštitu od gubitka podataka.
- Visoke performanse – General purpose koji je pogodan za većinu podataka, kao i Max I/O koji je pogodan za aplikacije koje zahtevaju visok protok i paralelne operacije.
- Niska latencija – Pristup fajlovima je veoma brz.

- Bezbednost – Podrška za enkripciju na nivou skladišta i u toku prenosa podataka. Koristi IAM za kontrolu pristupa i POSIX dozvole. Kroz podešavanja Virtual Private Cloud (VPC) mreže može se osigurati za privatni pristup.
- Integracija – lako se integriše sa drugim AWS servisima kao što su EC2, Lambda, ECS i EKS (servisi za kontejnerizaciju) itd.
- Troškovi – Plaća se samo prostor koji se koristi.

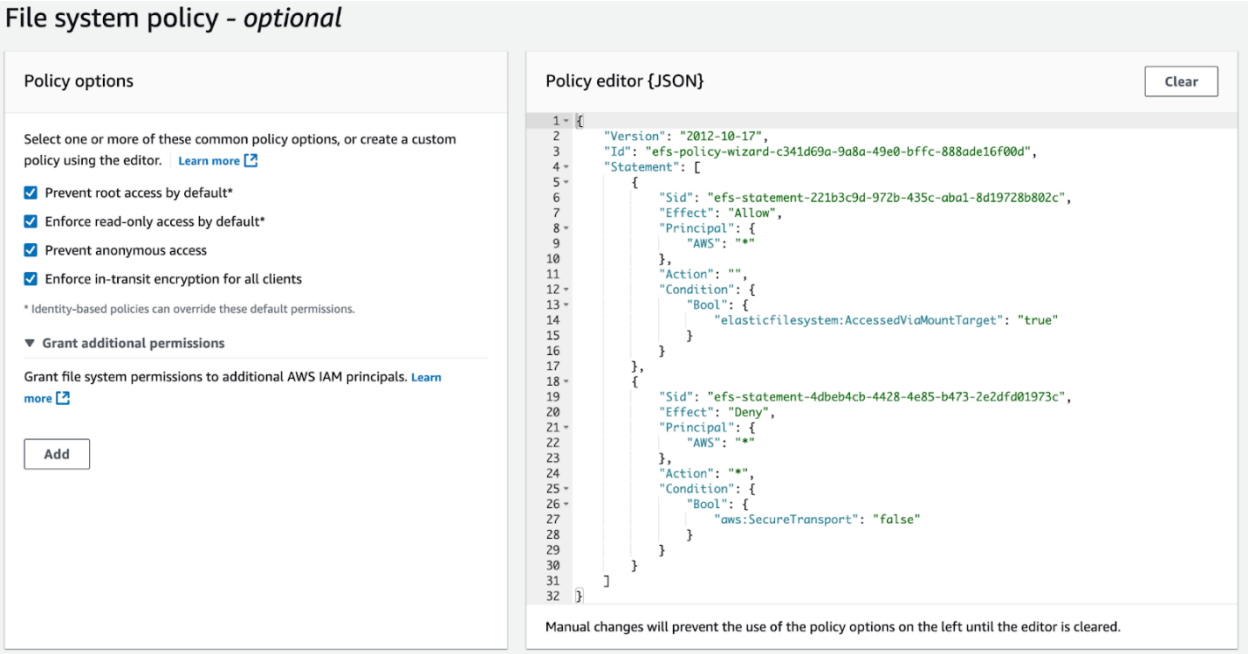
U narednih nekoliko koraka je prikazano pravljenje EFS. Kroz AWS konzolu možemo pristupiti EFS servisu i u okviru njega izabrati opciju Create file system (Slika 72).



Slika 72 Kreiranje EFS-a

Odabirom opcije Customize možemo izmeniti preporučena podešavanja. Jako je bitno obratiti pažnju na enkripciju jer nju nije moguće izmeniti nakon pravljenja EFS, što znači da ako je EFS bez enkripcije, onda se mora napraviti novi i izvršiti migracija na njega kako bi kasnije imao enkripciju. Podešavanja

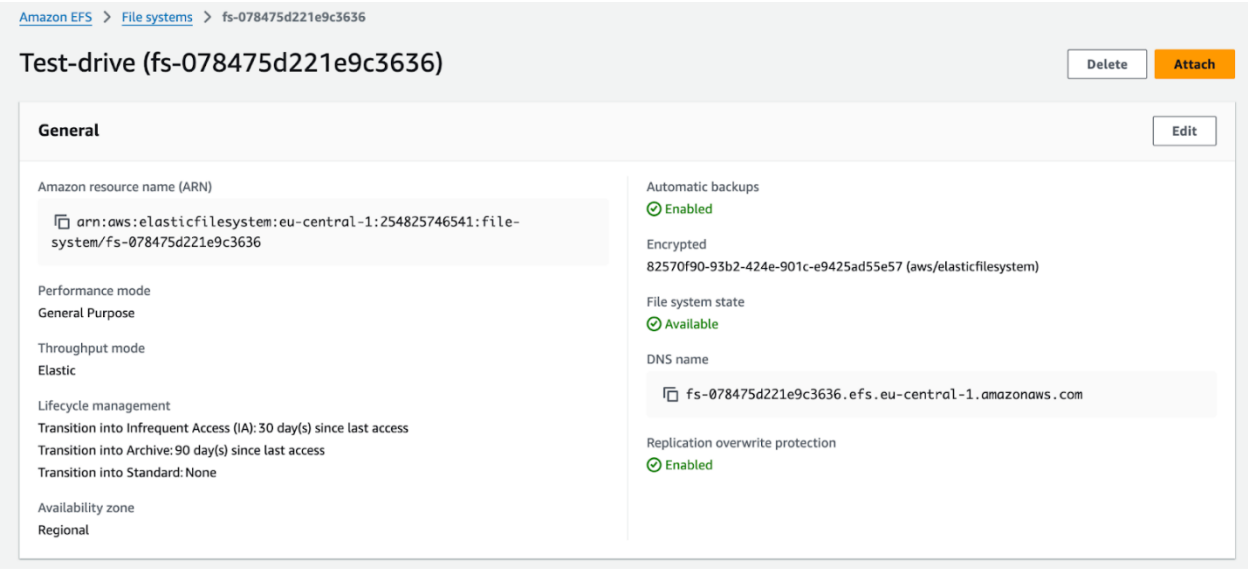
brzine prenosa, IOPS, mreže itd. se nalaze u Customized podešavanjima. Takođe, u okviru njih moguće je definisati i polisu pristupa fajl-sistemu (Slika 73), kao, na primer, da je zabranjen root pristup, da je EFS read-only, da je enkripcija u prenosu podataka obavezna itd.



Slika 73 Podešavanje polise EFS

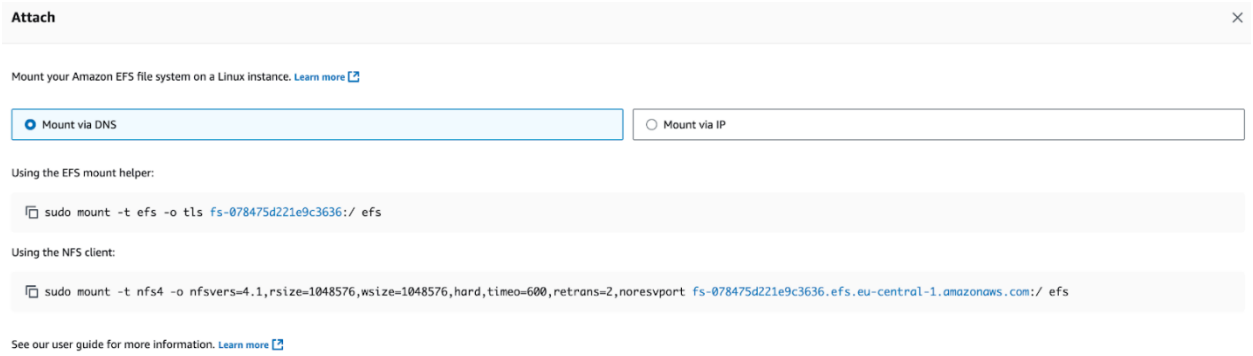
Nakon podešavanja može se izabrati opcija Create. Brzo nakon toga disk postaje dostupan za pristup i njegov status prelazi iz Creating u Available.

U pregledu EFS možemo pogledati njegova podešavanja, vršiti monitoring, proveriti veličinu, mrežna podešavanja (IP adrese po subnetu), kao i pogledati njegovu DNS vrednost koja je bitna za povezivanje na EFS (Slika 74).



Slika 74 Pregled EFS

Ukoliko se izabere opcija Attach, mogu se pogledati instrukcije za povezivanje na EFS putem DNS ili putem IP adrese (Slika 75).



Slika 75 Načini povezivanja na odabrani EFS

EFS se isto može povezati na Lambde; u tom slučaju je potrebno definisati access point sa pravima pristupa, a zatim u okviru interfejsa Lambde dodati file system i podesiti da se povezuje na taj EFS access point. Samim tim, kad god se Lambda izvrši, imaće pristup file systemu EFS automatski.

AWS FSx, Storage Gateway i Snow Family

Amazon FSx

AWS FSx je upravljana (managed) usluga fajl servera koja omogućava pokretanje i korišćenje fajl sistema visokih performansi. Za razliku od EFS, koji podržava samo NFS protokol, FSx nudi integraciju sa Windows operativnim sistemima preko SMB protokola, iako u podvrijanti ima podršku i za NFS protokol.

FSx for Windows file server koristi SMB protokol i dizajniran je za upotrebu sa Windows file sistemima; posebno je značajan ako se koriste Windows ACL pravila i potrebno je da se primene.

FSx for Lustre je izuzetno brz fajl sistem koji služi za HPC (High Performance Compute) računare i pogodan je za analitiku. Posebno je značajan kod upotrebe za mašinsko učenje, simulacije itd.

Karakteristike Amazon FSx:

- **Jednostavnost**: Omogućava brzo i jednostavno pokretanje potpuno upravljanog i visokopouzdanog fajl sistema.

- **Visoka dostupnost**: Amazon FSx nudi različite opcije implementacije kako bi odgovarao zahtevima vašeg radnog opterećenja po pitanju dostupnosti i izdržljivosti. FSx for Windows File Server omogućava implementacije u jednoj ili više dostupnosnih zona (single-AZ ili multi-AZ), u zavisnosti od potreba aplikacije. FSx for Lustre omogućava izbor između privremenog (scratch) ili trajnog (persistent) skladišta, u zavisnosti od toga da li se podaci obrađuju kratkoročno ili dugoročno.

- **Bezbednost**: Amazon FSx automatski enkriptuje podatke na disku (at rest) i u prenosu (in transit). Za kontrolu mrežnog pristupa fajl sistemima korisnika, omogućava pokretanje fajl sistema unutar Amazon Virtual Private Cloud (Amazon VPC). Usklađen je sa najvišim bezbednosnim standardima i ima sertifikate koji potvrđuju usklađenost sa **ISO, PCI-DSS, SOC, HIPAA** standardima. Integracija FSx-a sa AWS Backup omogućava centralizovano upravljanje bekapovima.

- **Troškovi**: Nudi širok spektar SSD i HDD opcija skladišta. Kompresija podataka pomaže u smanjenju potrošnje skladišta. Za dodatnu optimizaciju troškova, Amazon FSx for Windows

File Server uklanja redundantne podatke, dok FSx for Lustre omogućava izbor „scratch“ fajl sistema kako bi bili dodatno smanjeni troškovi za privremenu obradu podataka.

- **Laka integracija**: FSx se može integrisati sa drugim AWS uslugama kao što su Amazon S3, CloudWatch, CloudTrail, AWS KMS itd. FSx fajl sistemi mogu se povezati i sa uslugama kao što su Amazon SageMaker, Amazon WorkSpaces, Amazon ECS, Amazon EKS, AWS Batch…

AWS Storage Gateway

AWS Storage Gateway je hibridni servis skladišta u oblaku koji pomaže preduzećima da povežu svoje lokalne aplikacije sa AWS storage servisima. AWS Storage Gateway dolazi u više opcija, u zavisnosti od tipa upotrebe i načina skladištenja podataka, i omogućava jednostavan prenos podataka između lokalnog okruženja organizacije i AWS-a. Firme mogu da iskoriste skladištenje u oblaku za backup podataka, arhiviranje, oporavak od katastrofa i migraciju podataka, dok istovremeno čuvaju često korišćene podatke lokalno radi brzog pristupa. Ovaj hibridni pristup omogućava da se podaci bezbedno skladište u oblaku, a da lokalne aplikacije imaju laki pristup podacima putem poznatog interfejsa.

Tipovi AWS Storage Gateway servisa:

- Amazon S3 File Gateway – Omogućava migraciju podataka u oblak tako što organizacija može da čuva fajlove aplikacija i backup image kao objekte u Amazon S3. Amazon S3 File Gateway podržava prenos podataka pomoću SMB ili NFS protokola, uz lokalno keširanje. To znači da ovaj gateway omogućava brz lokalni pristup i garantuje dostupnost podataka.
- Amazon FSx File Gateway – Za korisnike Windows fajl sistema, omogućava lokalni pristup fajl deljenju koristeći Amazon FSx for Windows File Server. Ovaj gateway je pogodan za organizacije koje žele da koriste Windows fajl deljenje (SMB) bez potrebe za održavanjem lokalnih Windows servera. Takođe se lako integriše sa poslovnim aplikacijama.
- Volume Gateway – iSCSI blok skladište za lokalne aplikacije, dok se podaci zapravo čuvaju u oblaku. Idealan je za aplikacije kojima treba skalabilno blok skladište, ali žele i da smanje troškove. Dostupan je u dva režima rada:
    - Cache mode – Lokalni podaci se keširaju, a glavno skladište je u S3.
    - Stored mode – Podaci se primarno skladište lokalno, a bekap se vrši u S3.
- Tape Gateway – Omogućava organizacijama da zamene fizičke trake sa virtuelnim trakama u oblaku (AWS). Umesto fizičkih tape uređaja na lokaciji, koristi se virtuelna traka koja se kešira lokalno i skladišti u Amazon S3. Podržava sve glavne softvere za bekapovanje i omogućava jednostavnu integraciju sa postojećim tokovima rada.

AWS Snow Family

Kako bi omogućio prenos velikih količina podataka u oblak i iz oblaka bez oslanjanja na računarsku mrežu, AWS je razvio Snow Family – grupu fizičkih uređaja. Ovi uređaji su idealni za masovan prenos podataka i lokalno skladištenje.

Za slučajeve upotrebe poput naprednog mašinskog učenja i analize video snimaka u visokoj rezoluciji u okruženjima bez mrežne povezanosti, uređaj Snowball Edge Compute Optimized nude 52 virtuelna CPU-a, 42 TB blok ili objektnog prostora, kao i opciono GPU ubrzanje. Za ekstremno velike količine podataka, AWS Snowmobile koristi šleper sa kontejnerom, koji fizički transportuje podatke. Ova usluga je korisna za migraciju podataka, oporavak od katastrofa, selidbu i gašenje lokalnih data centara, prikupljanje podataka na udaljenim lokacijama itd.

Uređaji iz Snow porodice su pravljeni da budu robusni, prenosivi i bezbedni. Generalno su dizajnirani da rade u teškim uslovima rada i da omoguće bezbedan i brz prenos podataka.

Prednosti AWS Snow Family:

- Jednostavno upravljanje i monitoring
- Lokalna obrada podataka (on-board computing)
- Enkripcija podataka
- Otpornost na neovlašćen pristup (anti-tamper) i vizuelna evidencija pokušaja manipulacije (tamper-evident)
- Praćenje uređaja od kraja do kraja
- Sigurno brisanje podataka nakon prenosa

Karakteristike skladištenja u AWS-u

Tipovi skladištenja u AWSu se mogu podeliti na:

- **Blok skladište (Block Storage)** – npr. EBS
- **Objektno skladište (Object Storage)** – npr. S3
- **Fajl sistemsko skladište (File Storage)** – npr. EFS, FSx
- **Hibridno i offline skladištenje** – npr. Storage Gateway i Snow Family

Neke od ključnih prednosti cloud skladištenja u odnosu na on-premise rešenja:

1. **Skalabilnost i fleksibilnost** – Skladište se lako prilagođava rastu poslovanja, omogućavajući dinamičnu alokaciju resursa.
2. **Smanjeni troškovi** – Nema potrebe za kupovinom i održavanjem fizičkog hardvera, a korisnici plaćaju samo resurse koje koriste.
3. **Povećana sigurnost** – Implementirana enkripcija podataka, kontrola pristupa i compliance sa standardima kao što su GDPR i HIPAA.
4. **Visoka dostupnost i pouzdanost** – Cloud infrastruktura osigurava redundanciju i automatske backup mehanizme.
5. **Brza integracija i automatizacija** – Ponuda API i alata za automatizaciju, što olakšava upravljanje skladištenjem i ubrzava workflow-ove.

Bezbednost i upravljanje skladištem u cloudu:

- **Enkripcija i zaštita podataka** – AWS Key Management Service (KMS) za upravljanje ključevima za enkripciju i enkripcija podataka. Enkripcija "at rest" i "in transit".
- **Politike pristupa i kontrola korisnika** – AWS IAM polise za kontrolu pristupa skladišnim servisima kao i princip najmanjih privilegija (Least Privilege Principle).
- **Backup strategije i disaster recovery** – AWS Backup, S3 Glacier za dugoročno čuvanje podataka. Multi-region backup i replikacija.

Cloud nudi i optimizaciju troškova skladištenja kroz razne mogućnosti, kao što su biranje klase skladištenja koja je potrebna (EBS, S3), polise za životni ciklus podataka (S3 lifecycle policies), kompresija i deduplikacija.

Praktični zadaci za studente

<u>Zadatak 1: Kreiranje i upravljanje S3 bucketom</u>

Cilj: Kreirati S3 bucket, učitati fajlove, podesiti dozvole i testirati verzionisanje.
Zahtevi:

- Napraviti S3 bucket koristeći AWS Management Console ili CLI;
- Uploadovati tri fajla i podesiti različite permisije (public/private);
- Omogućiti verzionisanje i testirati kako funkcioniše kada se fajl zameni.

Zadatak 2: Hostovati web sajt na S3

Cilj: Kreirati S3 bucket, uploadovati stranice za web sajt, podesiti da je hostovan sa S3.
Zahtevi:

- Napraviti S3 bucket koristeći AWS Management Console ili CLI;
- Napraviti index.html i error.html web stranice;
- Uploadovati web stranice na S3;
- Podesiti resource policy;
- Podesiti javan pristup fajlovima;
- Podesiti web site hosting opciju u okviru S3.

Zadatak 3: Konfiguracija i upotreba EBS volumena

Cilj: Kreirati EC2 instancu i povezati EBS volumen, zatim formatirati disk i koristiti ga.
Zahtevi:

- Kreirati EC2 instancu i dodati dodatni EBS volumen od 10 GB;
- Formatirati disk i uvezati ga na EC2 instancu;
- Testirati trajnost podataka tako što će se EC2 instanca restartovati.

Zadatak 4: EFS povezivanje na više EC2

Cilj: Napraviti dve EC2 instance, napraviti EFS i sa jedne instance upisati podatke na EFS, a sa druge pročitati.
Zahtevi:

- Napraviti dve EC2 instance;
- Napraviti EFS;
- Povezati EFS sa obe instance;
- Izvršiti upis i čitanje sa EC2 instanci.

6.2. Baze podataka i AWS

U savremenim sistemima podaci su osnova aplikacije, od *checkout* transakcije u e-trgovini, preko telemetrije IoT uređaja, do kataloga proizvoda i preporuka. Tradicionalne, *self-managed* baze daju finu kontrolu, ali nose i visoku cenu – izdavanje zakrpa, bekapi, visoka dostupnost, *failover*[20], nadzor, kapacitetno planiranje i noćne intervencije. AWS managed servisi preuzimaju veliki deo operativnog tereta i nude izgrađene šablone za visoku dostupnost, bezbednost i skaliranje.

Izbor tipa baze prvenstveno određuje model podataka i način (obrazac) pristupa:

- Relacioni model i stroge transakcije (ACID) – **RDS/Aurora**.

[20] Failover kod baza podataka je mehanizam automatskog prebacivanja rada sistema na rezervni (backup) server ili instancu baze u slučaju da primarna baza padne ili postane nedostupna.

- Fleksibilniji, ključ-vrednost (key-value), dokument model, horizontalno skaliranje, milisekundska latencija pri velikoj količini upita – **DynamoDB**.

- Keširanje i izuzetno niska latencija u memoriji – **MemoryDB/ElastiCache**.

- Dokumenti (JSON), graf relacije, vremenske serije itd. – specijalizovani servisi (DocumentDB, Neptune, Timestream, QLDB).

Amazon RDS

Amazon RDS (Relational Database Service) je upravljani servis koji pojednostavljuje postavljanje, održavanje i skaliranje relacionih baza (PostgreSQL, MySQL/MariaDB, Oracle, SQL Server, uz Auroru kao poseban, optimizovan *engine*). Kod RDS-a AWS se brine o životnim ciklusima instanci, rezervnim kopijama, patchevima i sledu standardnih operativnih zadataka, dok korisnik dizajnira šemu, piše SQL i razvija aplikaciju. RDS je orijentisan na OLTP tip opterećenja i strukturisane podatke, pa je čest izbor kao zamena za on-premise DB bez promene modela i SQL semantike.

Multi-AZ pravi rezervnu instancu u drugoj AZ (unutar iste regije) i sinhrono replicira transakcije. U slučaju kvara, RDS vrši automatski failover na sekundarnu instancu bez promena na aplikacionom nivou (DNS endpoint ostaje stabilan, tj. isti). Multi-AZ je mehanizam za izdržljivost i kontinuitet rada, a ne za skaliranje čitanja.

***Read* replike** su asinhrone kopije primarne baze koje služe za horizontalno skaliranje čitanja, na primer, API endpointi koji su opterećeni čitanjem, servisi za izveštavanje itd. Replike mogu biti u istoj AZ, drugoj AZ, pa čak i u drugom regionu (cross-region), mogu se promovisati u nezavisne instance prilikom migracija/incidenta, ali podrazumevano nisu bekap niti cilj failovera.

RDS sprovodi **automatske bekape** unutar definisanog prozora održavanja, koji se definiše kroz konfiguraciju u svrhe patchovanja i backupovanja, i omogućava **Point-in-Time Recovery (PITR)** u okviru retention perioda. Pored toga, korisnici prave **ručne snapshots**, pune kopije trenutnog stanja koje se čuvaju u S3 i ostaju dostupne i nakon brisanja instance, uz mogućnost deljenja sa drugim AWS nalozima i kreiranja novih instanci iz snapshota (uvek sa novim endpointom). Pre brisanja produkcione instance preporučuje se finalni snapshot.

RDS podržava **encryption at rest** (KMS) i **in transit** (TLS). Kada je enkripcija uključena, enkriptovani su storage, bekapi, snapshots i replike. Važno ograničenje: aktivna (ne-enkriptovana) instanca se ne može na licu mesta enkriptovati, već se pravi snapshot, enkriptuje, pa se iz njega podiže nova enkriptovana instanca. Replike nasleđuju status enkripcije primarne instance.

RDS omogućava **vertikalno skaliranje** računarske klase (uz kratak prekid), kao i **povećanje** dodeljenog prostora (smanjenje prostora nije podržano). Promene se mogu primeniti odmah ili u okviru prozora održavanja; storage proširenje se obično dešava online, uz moguće privremeno pogoršanje performansi. Za čitanja se koristi horizontalno skaliranje preko read replika.

Praktična napomena: U planiranju kapaciteta uvek uračunati safety margin, jer je smanjenje prostora komplikovano i često znači da se mora kreirati nova instanca i migracija.

U praksi RDS se oslanja na:

- **Parameter/Option Group** za fino podešavanje *engines*, na primer work_mem u Postgresu, InnoDB parametri u MySQLu;

- **Maintenance window** za patcheve i minor nadogradnje;

- **CloudWatch/Performance Insights/Enhanced Monitoring** za metrike kao pto su llatencija, IOPS, buffer cache hit ratio, čekanja po tipovima, top SQL itd;
- **Automated minor version upgrades** uz kontrolu kad se desi;
- **Automatski failover** u Multi-AZ i preporučene **runbook** procedure, na primer, validacija aplikacione konekcije, health checkovi, read replica lag nadzor.

Amazon Aurora

Aurora je AWS-ov *engine* za relacione baze kompatibilan sa MySQLom i PostgreSQLom, ali sa drugačijim slojem skladištenja. Ključne razlike u odnosu na klasični RDS *engine* su:

- **Distribuiran storage** sa **6 kopija podataka u 3 AZ-a**, čime se izoluje compute od skladišnog sloja, a time se ubrzava recovery i smanjuje rizik od storage kvara.
- **Performanse** su tipično do 5 puta veće u odnosu na standardni MySQL i do 3 puta veće u odnosu na standardni PostgreSQL, uglavnom zahvaljujući optimizacijama enginea i IO.
- **Skaliranje čitanja** preko velikog broja Aurora read replika sa niskom latencijom.
- **Aurora Serverless v2** omogućava granularno automatsko podešavanje kapaciteta u skladu sa opterećenjem, što dozvoljava eliminisanje viška u trenucima slabijeg opterećenja (noću, vikendima itd.).
- **Global Database** (multi-region) za nisku latenciju čitanja globalnim korisnicima i brzi oporavak u slučaju regionalnog incidenta.

Aurora je idealna kada je potrebno SQL/ACID i relacije, ali se traže elastičnost i performanse iznad onoga što dobijaju standardni enginei u RDSu.

Amazon DynamoDB

DynamoDB je potpuno upravljana NoSQL baza (key-value i dokumenti) sa izuzetno niskom latencijom na hiljade i milione upita u sekundi, bez servera i bez administracije instanci. Svi podaci su automatski replikovani preko više AZ-ova, skaliranje i kapacitet se prilagođavaju opterećenju (on-demand) ili se eksplicitno planiraju (provisioned). DynamoDB je oslonac za internet-scale servise, gaming sesije, IoT tokove podataka, metapodatke i keš-backende.

U centru baze je **primarni ključ** i može se koristiti na dva načina:

- **Partition key** – raspoređuje stavke po particijama;
- **Partition + sort key** – omogućava upite i grupe događaja pod istim particionim ključem.

Umesto normalizacije, kao u RDBMS-u, u DynamoDB-u se šema modeluje **po obrascima pristupa** *(query first)*. Često se denormalizuje kako bi jedan upit vratio sve što interfejsu ili mikroservisu treba. Za dodatne preglede dodaju se **LSI** – lokalni indeksi nad istim partition ključem, ali sa drugim *sort* ključem i **GSI**, koji su globalni indeksi sa novim *partition/sort* ključem.

DynamoDB podržava dva režima rada:

- **On-Demand** – Plaća se po zahtevu, idealno za nepredvidiv saobraćaj i PoC-eve[21].
- **Provisioned** – Unapred se zadaju **RCU/WCU** (Read/Write Capacity Units), uz **auto scaling**. Predvidljiv trošak i dobra kontrola SLOa.

Čitanja mogu biti **eventual** ili **strongly consistent**. Za ultra nisku latenciju dostupan je DynamoDB Accelerator (**DAX**) koji je in-memory keš kompatibilan sa DynamoDB APIem.

Global Tables pružaju multi-region replikacije, sa konflikt-free modelom last writer wins po timestampu.

Integracije i operacije sa DynamoDB:

- **DynamoDB Streams** – Svaka promena (insert/update/delete) je događaj (event), tipično se obrađuje event sa Lambdom za reakcije, projekcije, audit zapis ili replikaciju;
- **Transactions** – Podržane su ACID transakcije nad više stavki/tabela, ograničenja postoje, ali za mnoge slučajeve su prihvatljiva;
- **PartiQL** – SQL like sintaksa za čitanje/pisanje nad tablama;
- **TTL** – Za automatsko uklanjanje stavki kao što su keš, sesije, zastareli podaci;
- **Backups i PITR** – Kontinuirani bekap i vraćanje na tačku u vremenu;
- **Sigurnost** - IAM za fine grained kontrolu, KMS enkripcija at rest, TLS in transit.

Tipični obrasci upotrebe kod DynamoDB su:

- **User session store** – Primarni ključ je user-id ili session-id, a TTL održava čistoću tabele;
- **Event feed/telemetrija** – Compound ključ, na primer device-id i timestamp, za vremenske preglede i range upite;
- **Shopping cart/inventory** – Konzistentno čitanje i atomic counteri, streams za rezervacije i saglasnost stanja;
- **Feature flags/profilisanje** – Prosta key-value sa vrlo niskom latencijom.

RDS ili DynamoDB

Model podataka i fleksibilnost šeme. Ako je potrebna relacija između podataka (primarni-spoljni ključ), JOIN-ovi, složeni upiti, *window* funkcije i bogat SQL, RDS/Aurora je od izbor. Ako je primarno čitanje po ključu, podela entiteta po particijama da se modeluju putanje do podataka unapred, DynamoDB daje jednostavnost i brzinu.

[21] PoC – Proof of Concept, testna, ograničena varijanta nekog sistema (u ovom slučaju baze), koja služi za isprobavanje tehnologije ili pristupa, pre prelaska u produkciju.

Transakcije i konzistentnost. RDS nudi klasične ACID transakcije i snažan konzistentni model. DynamoDB ima transakcije, ali se tipično oslanjaju na jednostavnije per item operacije, idempotenciju i dizajn bez JOINova.

Skaliranje
Aurora/RDS skaliraju vertikalno (veća klasa) i horizontalno za čitanje (read replike), write skaliranje je i dalje ograničeno jednom writer instancom (Aurora poboljšava write performanse, ali konceptualno ostaje jedan writer). DynamoDB horizontalno skalira i za čitanje i za pisanje po delovima tabele (particije), što je jako korisno za vrlo visoka opterećenja.

Operativni profil
RDS ostavlja brigu o klasama instanci, patch prozorima, ponekad kratkom prekidu kod promene veličine instance i storage. DynamoDB je serverless, što znači da nema ni instance, ni patch prozore, a kapacitet je brojka (on-demand/provisioned) i IAM polisa.

Trošak
Za stabilna, predvidiva, kompleksna relational workload opterećenja RDS/Aurora je često najbolje rešenje, posebno sa rezervacijama/RI i skaliranjem read replika. Kod promenljivog i nepredvidivog opterećenja sa milijardama jeftinih zahteva, DynamoDB on-demand bude jednostavniji i često jeftiniji, sve dok je model pristupa dobro dizajniran.

Ekosistem i alatke.
Ako je potreban standardan SQL, JDBC/ODBC, BI alati itd., RDS/Aurora pojednostavljuju integracije. Ako treba cloud native event driven arhitektura sa Lambdom, Streams, globalnim replikacijama i ekstremnim QPS, DynamoDB je bolji izbor.

Ostale baze u AWS

Amazon MemoryDB for Redis/ValKey Redis-kompatibilan, **in-memory** i **durable**, za razliku od klasičnog keša, sa replikacijom u više AZ i perzistencijom. Podržava Redis OSS i ValKey. Odličan kada se zahteva latencija u stotinama mikrosekundi uz garantovanu trajnost, na primer keš koji ne sme da se izgubi, rang liste, sesije sa jakim SLA. Dobara baza ispred RDS/DynamoDB za najčešće pristupane ključeve.

Amazon ElastiCache – **Redis/Memcached** Klasični keš (ne i baza) za read heavy pristupe. Redis nudi strukture (sorted sets, lists), Memcached je jednostavan, linearan i lak za sharding. Ako trajnost nije prioritet i mogu da se rekonstruišu podaci iz izvora, ElastiCache je često najisplativiji za smanjenje latencije i rasterećenje baze.

Amazon DocumentDB – **kompatibilan sa MongoDB API** Dokumenti (JSON), dinamična šema, popularan u aplikacijama koje već koriste MongoDB biblioteke. Naglasak je na kompatibilnosti API-ja, upravljanosti i jednostavnoj skali čitanja.

Amazon Keyspaces for Apache Cassandra Upravljani Cassandra kompatibilan servis. Za vrlo velike, raspodeljene, write heavy upotrebe sa globalnim footprintom i poznatim Cassandra obrascima pristupa (particioni ključevi i kolone za vremenske serije).

Amazon Neptune Graf baza (Property Graph/Gremlin i RDF/SPARQL) za relacije prvog reda – društveni graf, detekcija prevara, preporuke po relaciji. Optimizovana za graf upite na velikoj skali.

Amazon QLDB Ledger baza sa kriptografski proverljivim, neizmenjivim dnevnikom promena. Za sisteme gde je audibilnost događaja kritična, na primer transakcioni lanci, sledljivost.

Amazon Timestream Baza za vremenske serije, automatska tiering strategija sa vrućim i hladnim podacima, idealna za IoT metrike, aplikativni monitoring i analitičke upite nad vremenskim oknima.

Praktična zapažanja, optimizacije i zamke

Za RDS/Aurora, dimenzionisanje **IO i skladišta (gp3/io2)** utiče na latenciju transakcija, tako da se predlaže posmatranje p95/p99[22] i waits (IO, locks) kako bi se optimizovale performanse. **Connection pool** (PgBouncer/HikariCP) je obavezan u visoko konkurentnim Java/Python/Node servisima i može se lako „isprazniti", što stvara ozbiljan problem, pogotovu u postavkama sa mikroservisima gde dosta replika otvara konekcije ka bazi. **Read Your Writes** znači ne preusmeravati sveže čitanje odmah na repliku, osim ako to nije problem domena. **Uputstvo za migraciju** kod RDS: napravi se snapshot, zatim restore (novi endpoint), test, *cutover* – prebacivanje na RDS (DNS/connection string). Uvek se pre brisanja formirafinalni snapshot.

Za DynamoDB modelovanje po upitima, početi od access patterns (ko, šta, kojim redosledom čita/piše). Izbeći vruće particije, dobar parittion key ravnomerno raspoređuje saobraćaj. Troškovi kod DynamoDB mogu vrlo lako izmaći kontroli zbog lošeg pristupa, pažljivo planirati bazu, upite, GSI, velike stavke itd. DynamoDB Steams u kombinaciji sa Lambdom su često izuzetno dobro rešenje za reaktivnu arhitekturu, ali je potrebno kontrolisati idempotenciju i retry logiku.

Kod izbora baze, ukoliko je potrebna SQL sa JOIN-ovima, složenim agregacijama tabela, ACID, sledi RDS/Aurora. Ako je key-value store, horizontalno skaliranje, latencija u milisekundama sa globalnim pristupom, tu je idealan DynamoDB. Isto tako, ako je potreban izuzetno brz i trajan keš, koristiti MemoryDB, a ako trajnost nije ključna, onda ElastiCache. Za čuvanje JSON dokumenata, sa Mongo API-jem tu je DocumentDB. Za Graf odgovara Neptune, za vremenske serije podataka (metrika) idealan je Timestream, dok za nepromenljivi dnevnik odgovara QLDB.

Zaključak

RDS, uključujući Auroru, i DynamoDB nisu konkurencija već komplement, osiguravaju da na raspolaganju postoji i klasična relacijska pouzdanost i cloud native horizontalna elastičnost. Dodavanjem specijalizovanih servisa, MemoryDB/ElastiCache, DocumentDB, Keyspaces, Neptune, QLDB, Timestream, dobija se skup iz kog se može sklopiti arhitektura tačno po meri poslovnog domena.

Primer

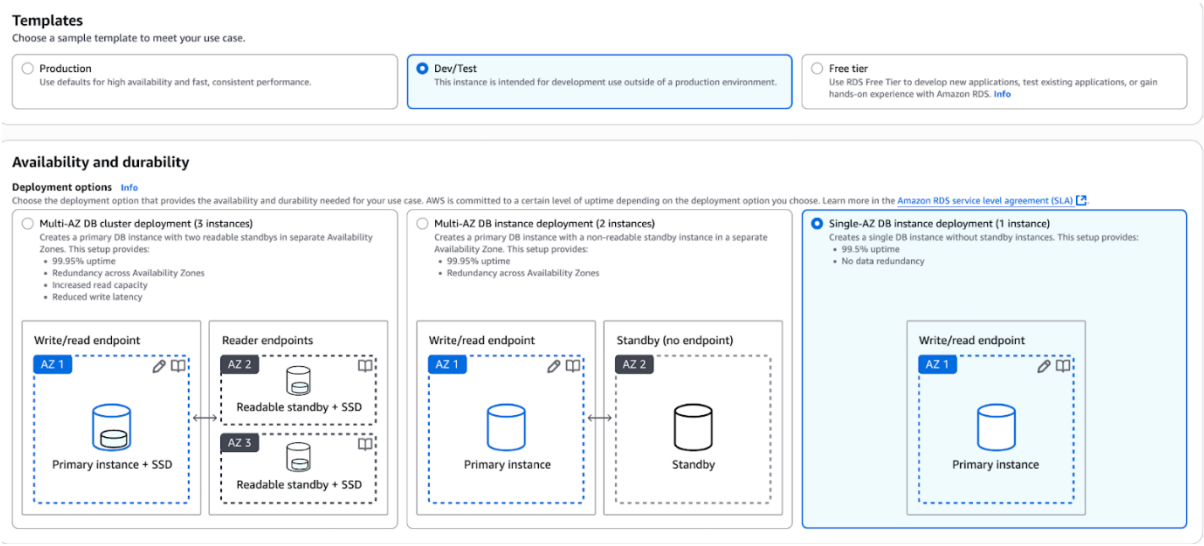
RDS baza

U ovom primeru prikazaće se proces kreiranja relacione baze u okviru Aurora and RDS servisa. Potrebno je u AWS konzoli izabrati istoimeni servis. Zatim izabrati opciju Databases sa leve strane i ići na opciju Create database. Postoje dva načina koje AWS nudi; Standard create sa svim opcijama i Easy create gde će AWS ponuditi opcije i preporučena podešavanja zavisno od toga da li se pravi Dev/Test baza ili Production baza. Za svrhe ovog primera ostati na Standard create. Za tip baze može se odabrati odgovarajuća baza, za svrhe ovog primera koristi se PostgreSQL opcija.

U okviru *engine version* ostaviti podrazumevani engine ili odabrati neki od starijih podržanih ukoliko to aplikacija zahteva.

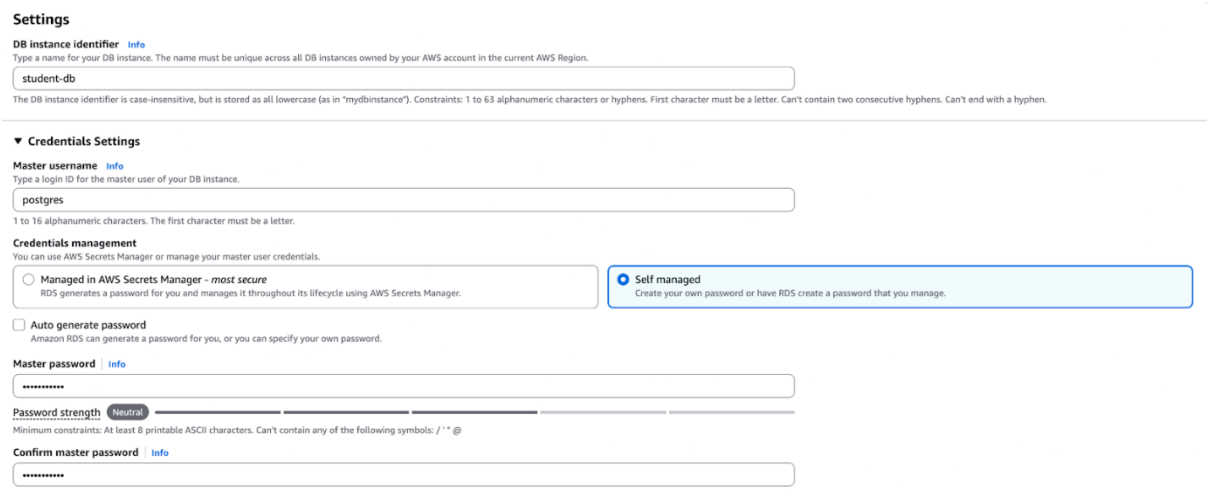
[22] p95 i p99 predstavljaju metrike performansi za određen procenat uzoraka. Ako je p95 = 250 ms, to znači da 950 od 1000 zahteva ima vreme odziva manje od 250 ms. Ako je p99 = 400 ms, to znači da 990 od 1000 zahteva traje manje od 400 ms, ali onih 1% zahteva traje duže — možda i nekoliko sekundi.

U okviru Templates sekcije odabrati Free tier ili Dev/Test jer Production automatski pravi klaster baza gde postoji *failover* replika i *reader endpoint* instanca (Slika 76). Kod Multi-AZ se pravi rezervna kopija bez endpointa u slučaju otkazivanja primarne, dok se kod Single-AZ opcije pravi samo jedna baza sa klasičnim endpointom.



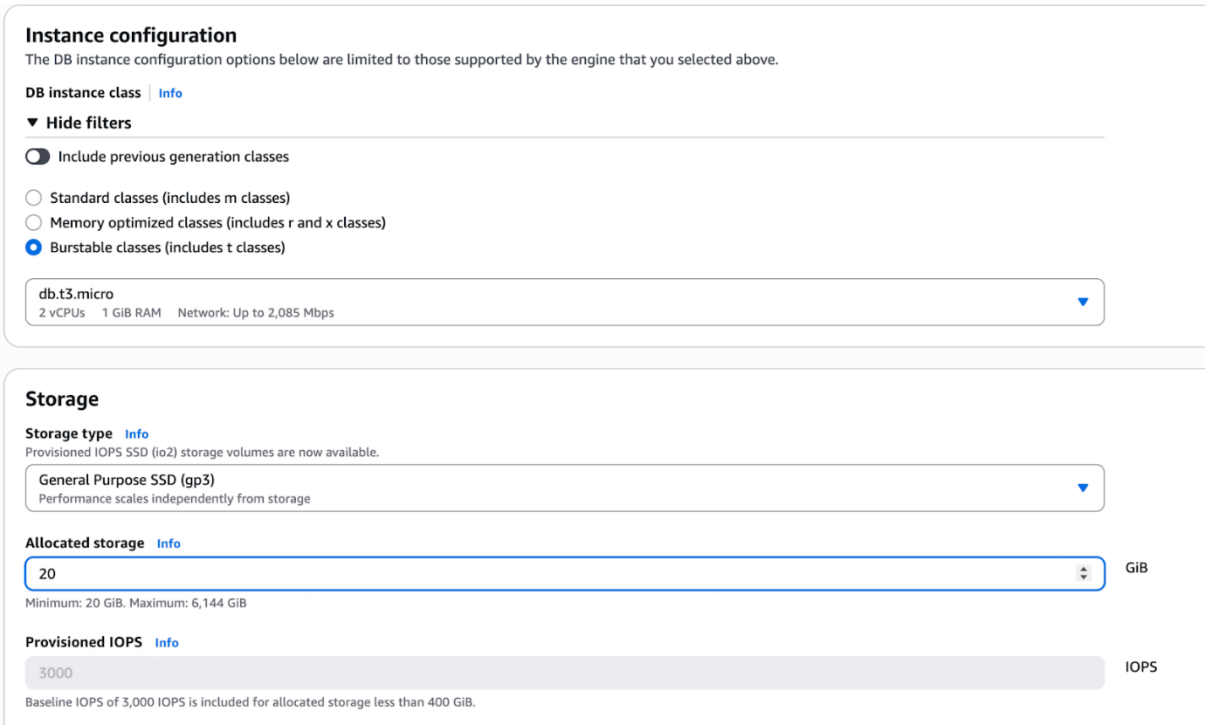
Slika 76 Odabir Template i Availability i durability opcija

Nakon toga potrebno je dati naziv bazi, tj. Identifier, i odabrati kako se upravlja kredencijalima; preporučena opcija je koristiti upravljane tajne preko Secrets Manager servisa, koji takođe nudi automatsku rotaciju tih kredencijala ka bazi, što je jako korisno i obavezno za produkcijske sisteme. Postoji i opcija Self Managed koja nije preporučena, ali za svrhe neke Dev/Test baze pojednostavljuje stvari. U Self managed sekciji obavezno sačuvati Master username i Master password (Slika 77).



Slika 77 Podešavanje kredencijala baze

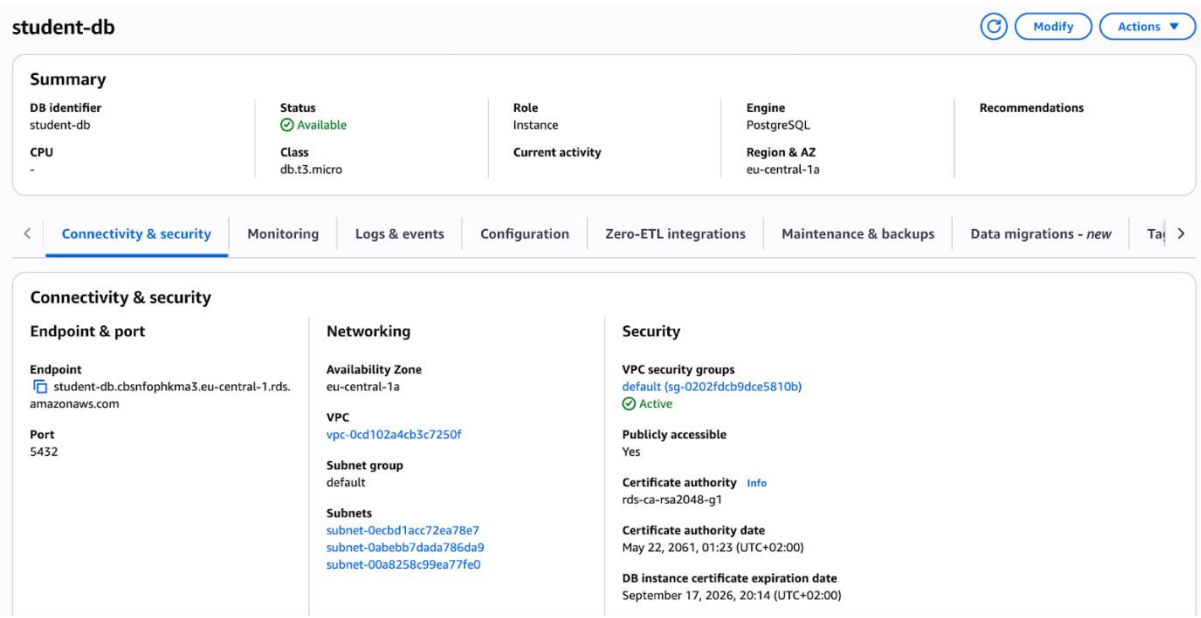
U okviru Instance configuration potrebno je uneti koja klasa instanci će biti korišćena zavisno od očekivanog opterećenja ka bazi. Za svrhe primera izabrana je jedna od manjih instanci — burstable klasa, db.t3.micro sa 2 vCPU i 1GiB RAMa. Za storage sloj se bira gp3 sa minimalnih 20GiB prostora, bitna napomena je da je ovo najmanja količina storage za bazu, uz to da storage može da raste, ali ne i da se smanjuje, tako da je u ovom koraku bitno da se ne uradi over provisioning (Slika 78).



Slika 78 Konfiguracija instance i skladišta

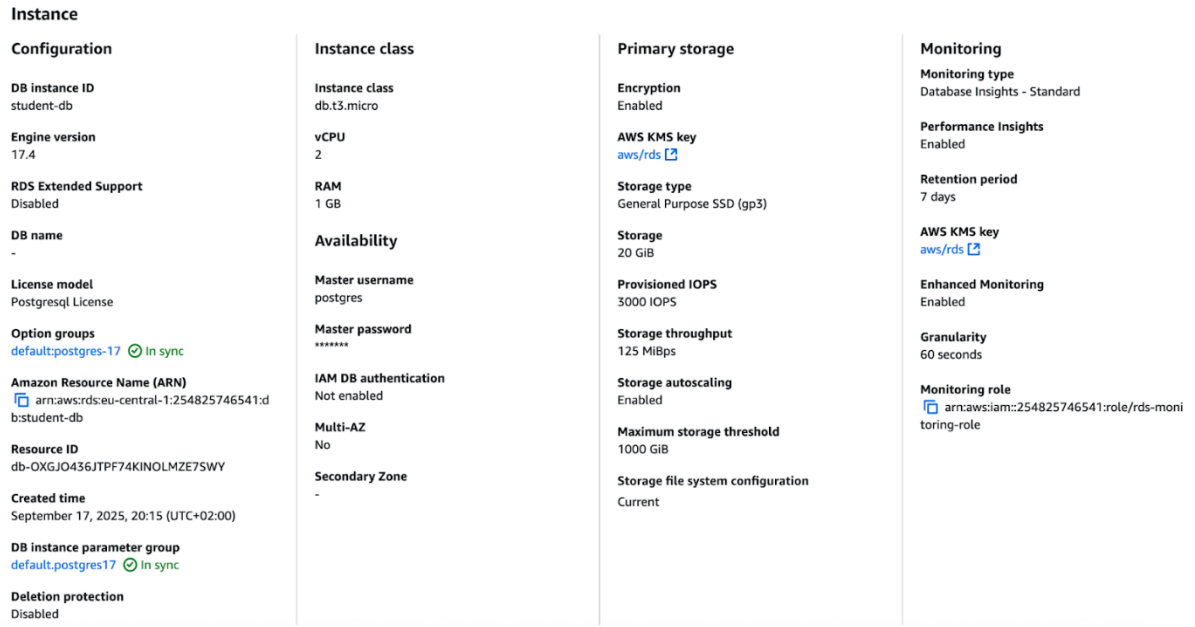
U Connectivity delu podešavanja ostaviti sve po default podešavanjima, s tim što je potrebno izabrati Yes na Public access opciji kako bi bila moguća konekcija ka bazi preko javnog interneta, uz napomenu da se u produkcijskim okruženjima nikad ne vrši konekcija ka bazi javno, već isključivo preko VPN ili proxy konekcija. U Additional configuration je moguće promeniti i default port na kom baza prihvata konekcije; u ovom slučaju to je port 5432 za Postgre baze. Ostatak podešavanja ostaviti istim i na dnu stranice ići na Create database dugme.

Potrebno je sačekati par minuta dok baza ne pređe u Available Status. Nakon toga može se odabrati opcija View connection details ili ući na panel baze i kopirati Endpoint & port parametre (Slika 79).



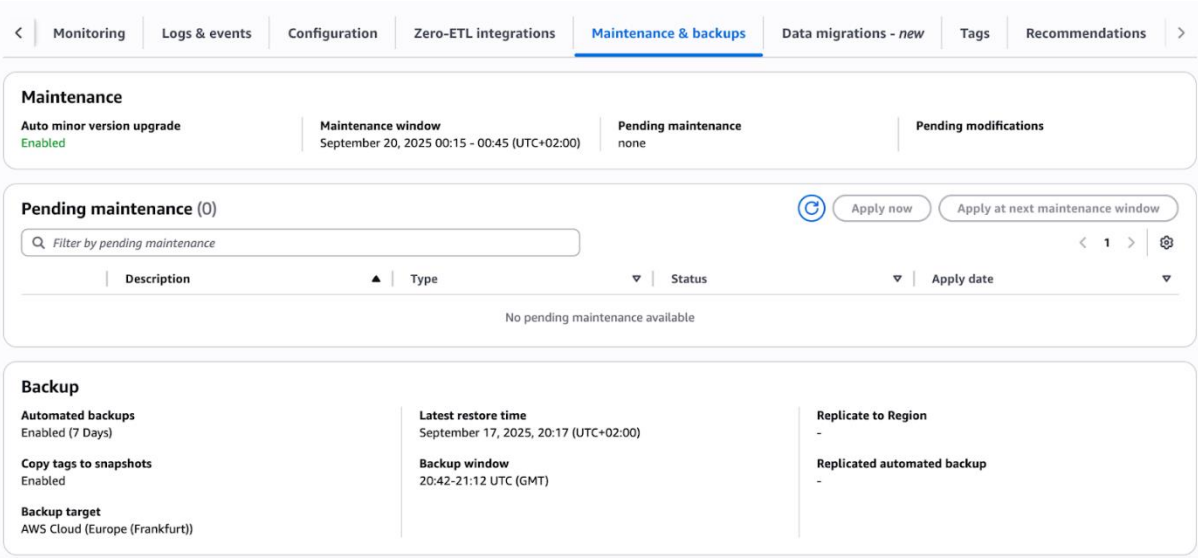
Slika 79 Prikaz parametara aktivne baze

U Configuration tabu mogu se videti sva podešavanja koja su prethodno uneta a vezana su za samu konfiguraciju baze (Slika 80).



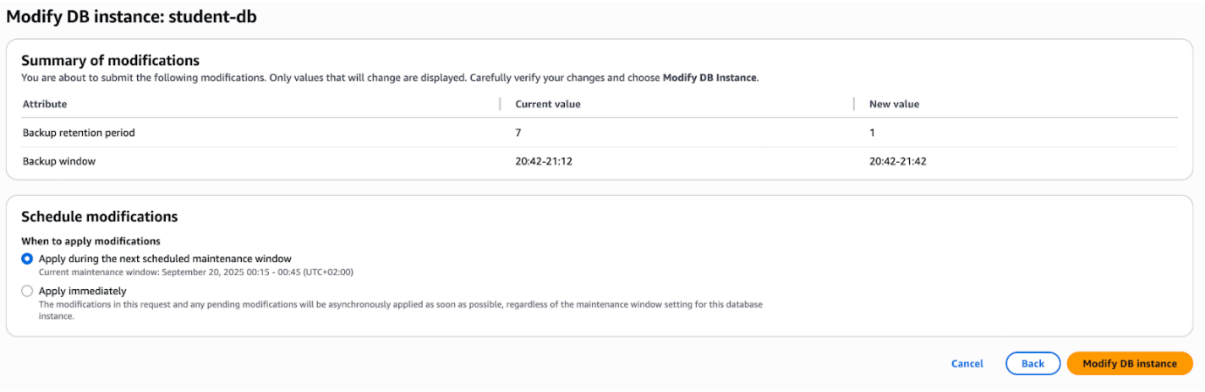
Slika 80 Pregled konfiguracije baze

Maintenance & backup prikazuju kada je maintenance window, koji se može podešavati, kao i da li ima Pending maintenance akcija, kao na primer patch za bazu. U okviru Backup sekcije se vidi konfiguracija i retencija backupa.



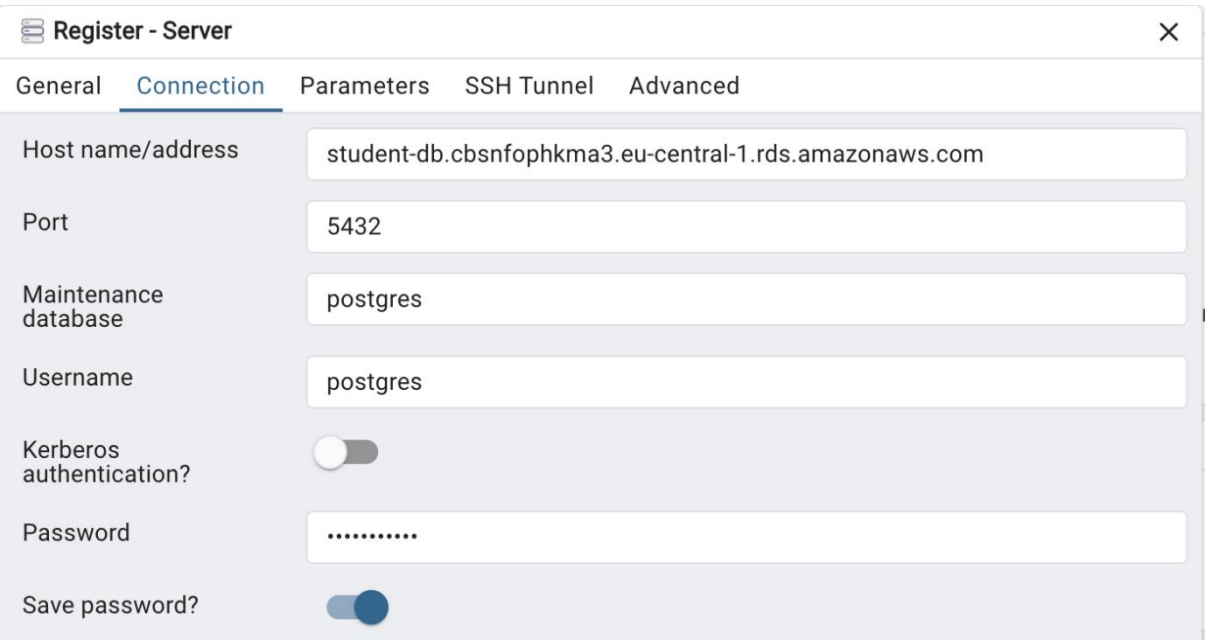
Slika 81 Maintenance i Backup

Ukoliko se ide na Modify opciju baze, moguće je promeniti neke od parametara, kao i promeniti tip instance baze. Kada se naprave izmene i odabere Continue opcija, AWS će ponuditi ta te promene izvrši odmah (ukoliko je neophodno) ili u okviru standardnog narednog Maintenance window-a (ukoliko promene nisu hitne) (Slika 82).

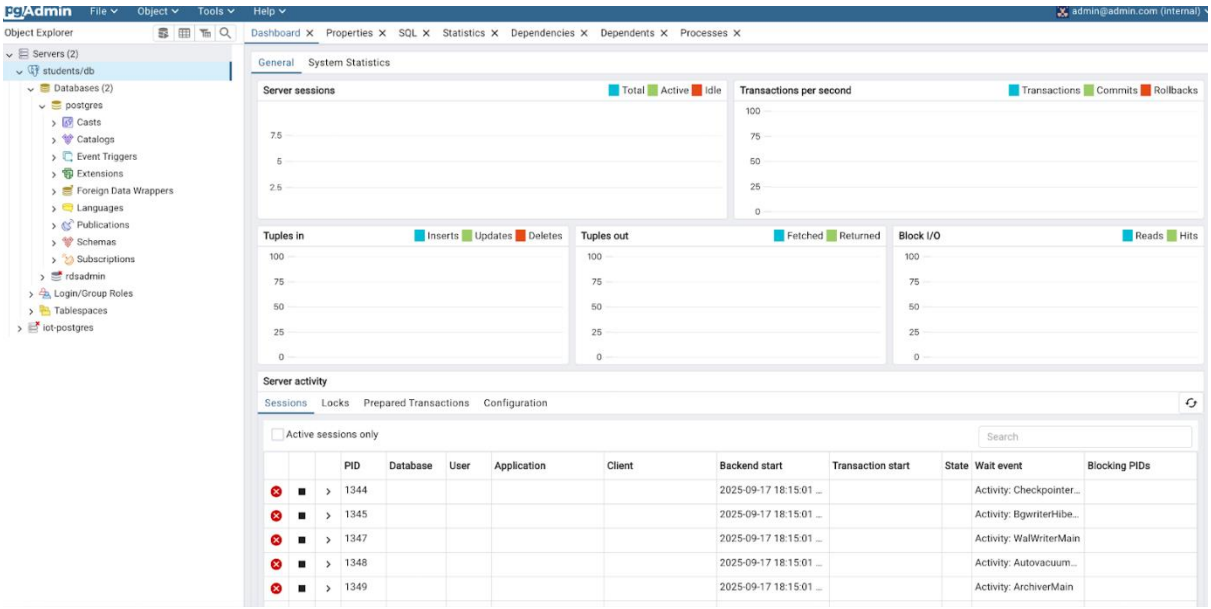


Slika 82 Prikaz pregleda željenih izmena sa odabirom kada ih izvršiti nad bazom

Unošenjem parametara u pgAdmin softver (Slika 83 i Slika 84) moguće je ostvariti konekciju ka bazi i vršiti dalje promene nad bazom, tabelama itd.



Slika 83 Povezivanje na bazu uz pomoć pgAdmin alata

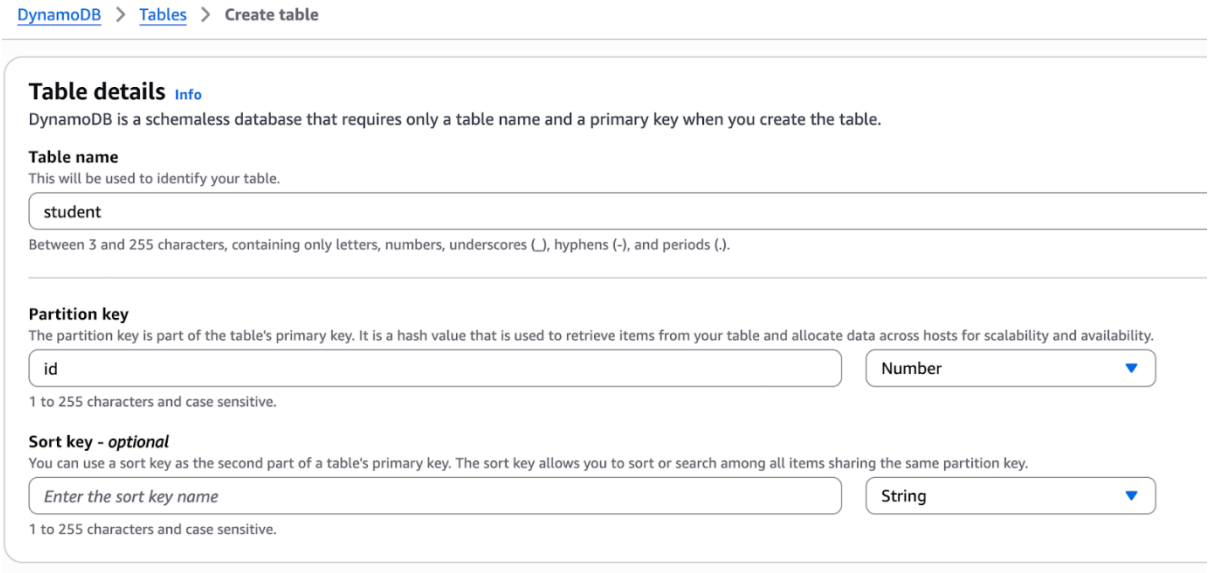


Slika 84 Pregled baze nakon povezivanja

Napomena: Ukoliko povezivanje nije uspešno, potrebno je proveriti sve parametre baze, security grupu koja je vezana za bazu, kao i Inbound port i da je otvorena konekcija na odgovarajući port - inbound na 5432 port sa svoje IP adrese ili sa 0.0.0.0/0.

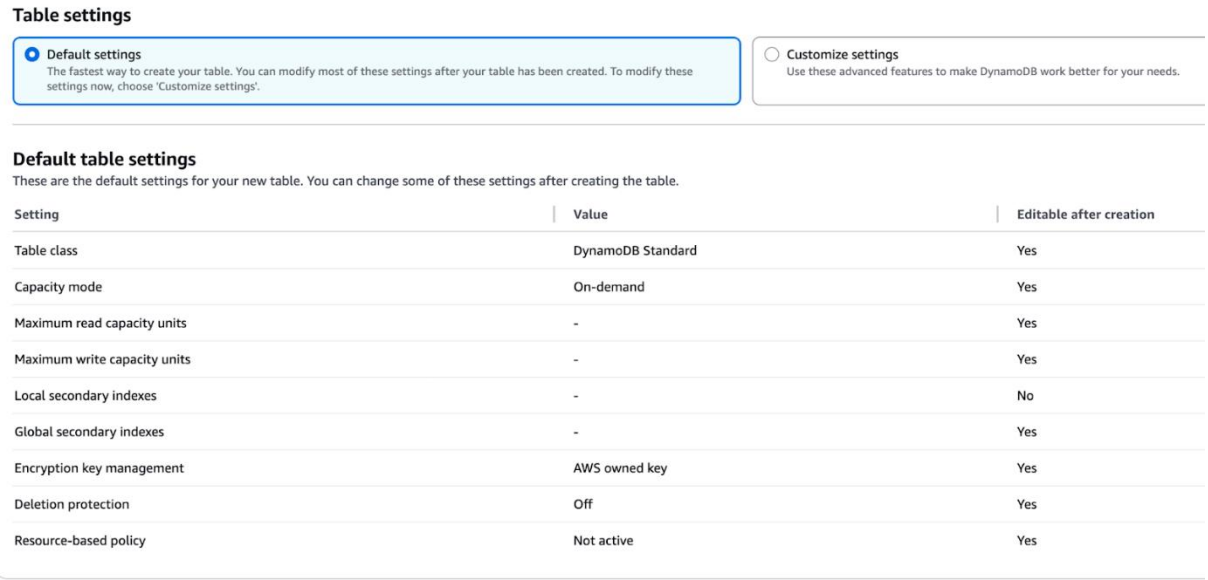
DynamoDB

U okviru DynamoDB servisa potrebno je odabrati opciju Create table, a zatim se daje ime tabeli; u svrhu primera daće se ime student. Partition key može biti String, Number ili Binary; obično se koristi number kao neki inkrementalni id ili string kao uvid ili neki drugi jedinstveni identifikator. Zatim postoji opcija i sort key-a (ključa za sortiranje) koja se neće koristiti za primer (Slika 85).



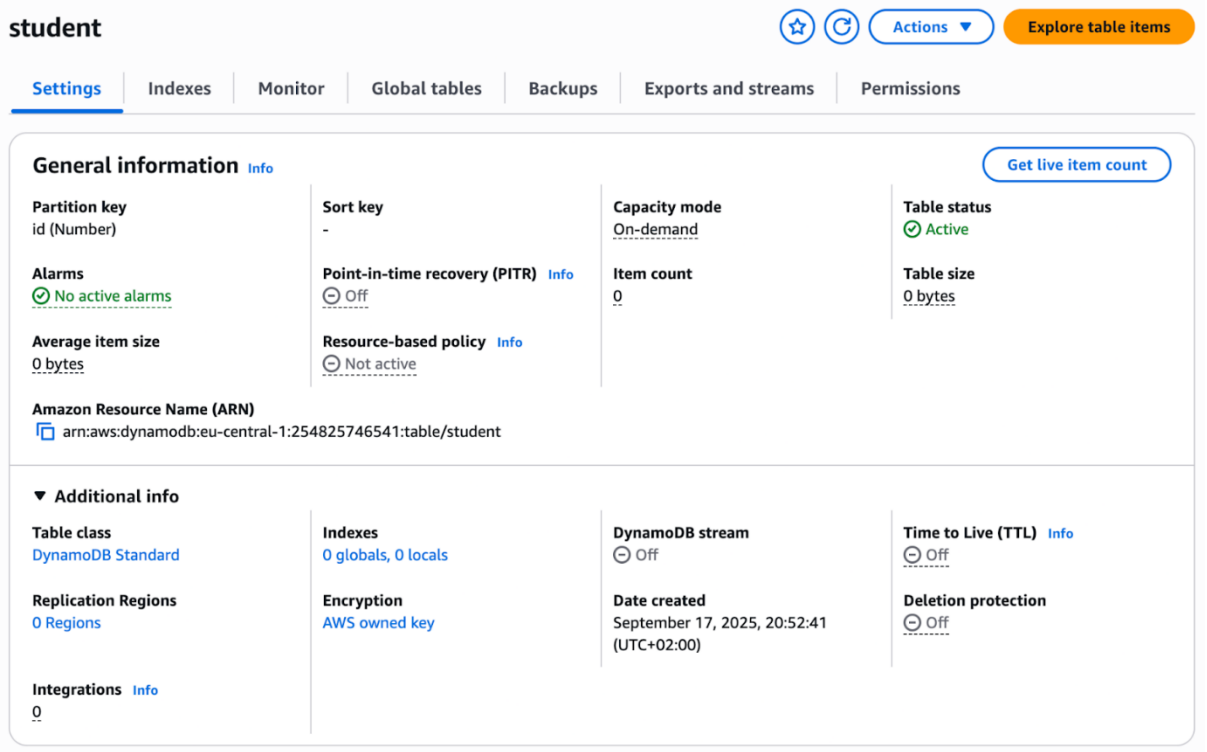
Slika 85 Podešavanja detalja tabele

U prikazu Table settings moguće je ostaviti Default settings ili Customize settings koji dozvoljavaju podešavanje bitnih parametara kao tip tabele, on-demand ili provisioned capacity, RCU i WCU, ključ za enkripciju tabele itd. Većinu ovih opcija je moguće izmeniti nakon pravljenja tabele.



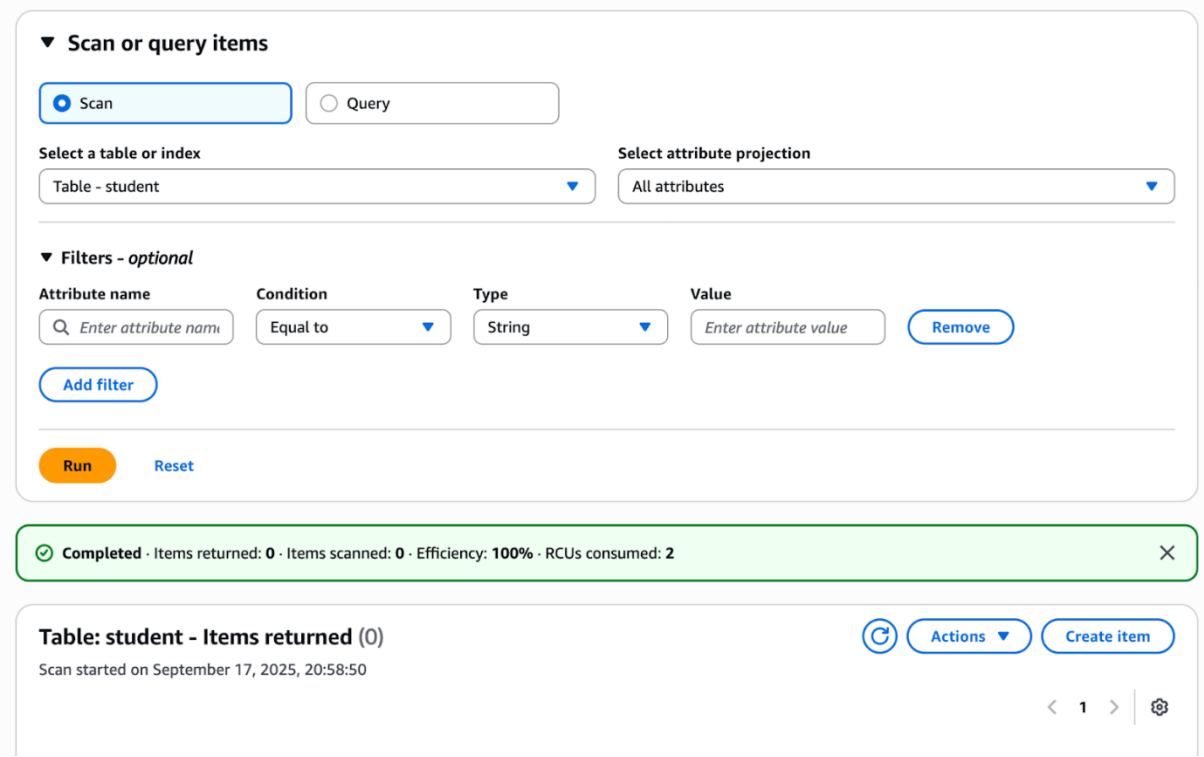
Slika 86 Podešavanje parametara tabele

Nakon toga se bira opcija Create table i posle par minuta tabela postaje dostupna (Slika 87).



Slika 87 Pregled parametara tabele student

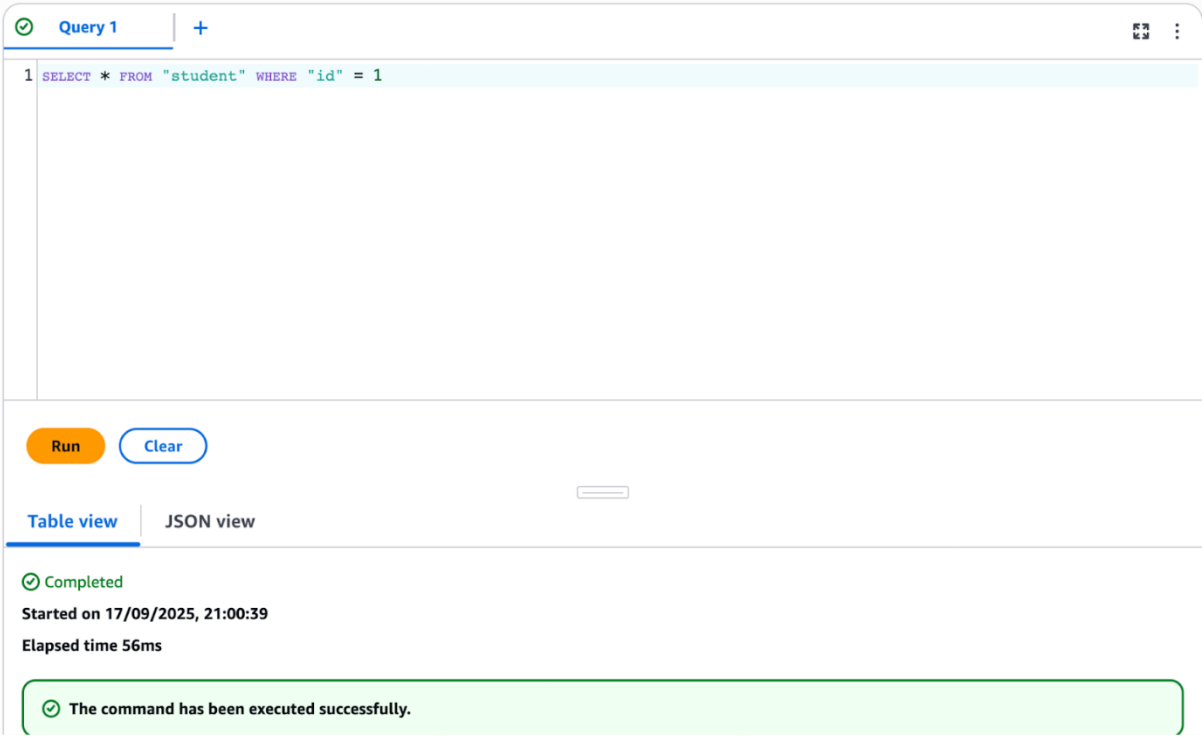
Nakon toga se može izvršiti povezivanje na tabelu uz pomoć više programskih jezika i AWS SDK. AWS nudi i pregled tabele kroz AWS konzolu pomoću Explore items dela, gde se može ručno pregledati sadržaj tabele (Slika 88).



Slika 88 Explore items u okviru AWS konzole

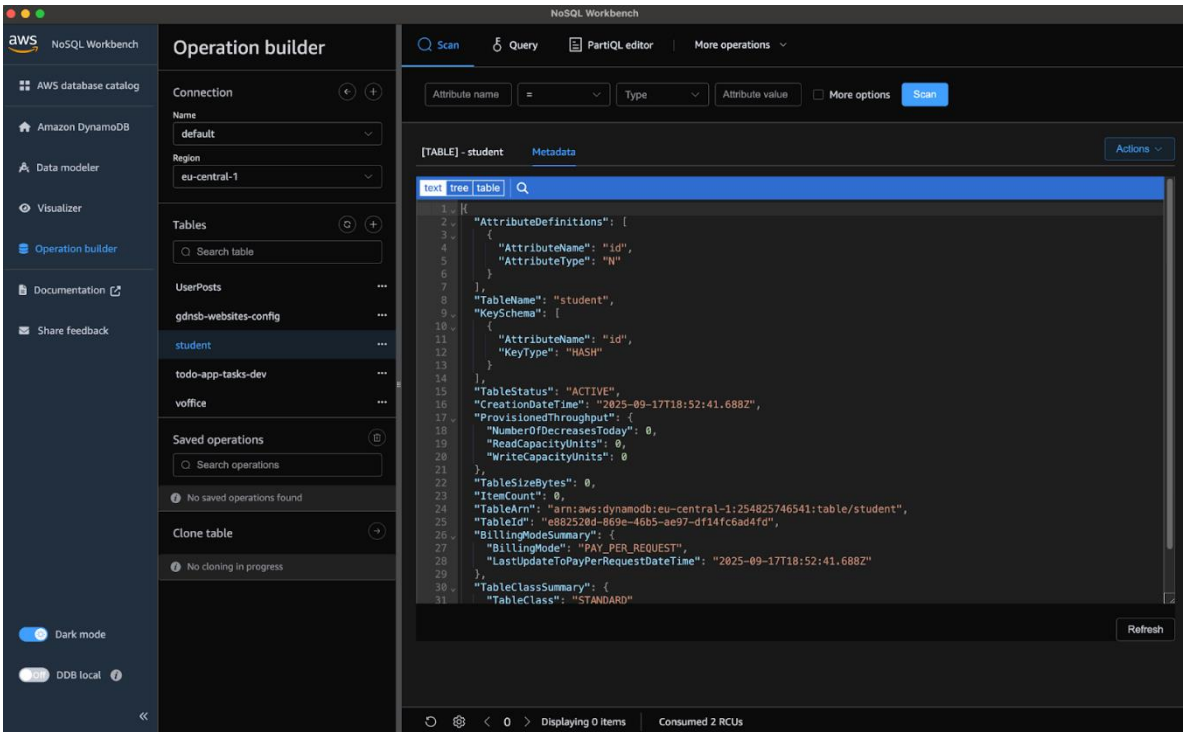
U ovom delu može se pregledati sadržaj tabele i mogu se vršiti Scan i Query uz odgovarajuće filtere za podatke.

Takođe, tu je i PartiQl editor koji dozvoljava upotrebu SQL-likih jezika za upite ka tabeli (Slika 89).



Slika 89 PartiQl editor za SQL like upite ka tabeli

Postoji i pristup pomoću lokalnog softvera koji zahteva da je urađen login na AWS pomoću parametara za programski pristup i obezbeđen pristup nalogu na kom se nalazi tabela — NoSQL Workbench (Slika 90).



Slika 90 Pristup Tabeli pomoću NoSQL Workbench alata

Zadaci

1. Napraviti Aurora RDS sa minimalnom cluster konfiguracijom i povezati se na cluster endpoint.
2. Napraviti DynamoDB tabelu i izvršiti CRUD operacije koristeći AWS SDK za programski jezik po želji.

7. Umrežavanje u oblaku

Cilj poglavlja je upoznavanje sa mrežnom infrastrukturom i sticanje znanja o principima funkcionisanja fizičke i virtuelne mreže u računarstvu u oblaku, kao i ovladavanje veštinama za konfiguraciju mreže u okruženju AWS.
Po savladanom poglavlju, student bi trebalo da može da:

- objasni specifičnosti mreže u oblaku
- opiše način organizacije mreže u oblaku (virtuelne i fizičke)
- objasni osnovne principe mrežnih protokola korišćenih u oblaku
- opiše prevođenje adresa kod VL2 infrastrukture

Student bi trebalo da ume da:

- konfiguriše virtuelni privatni oblak u AWS-u
- konfiguriše mrežne servise u AWS-u

Kod PC računara komunikacija između komponenti je brza zahvaljujući brzim magistralama koje su ugrađene u samom računaru. Kod System-On-Chip rešenja, integracija je još veća pa su kašnjenja u komunikaciji zanemarljiva jer su sve komponente: procesor, memorija i I/O „na licu mesta“.

Za dizajn oblaka ključno je obezbediti robusnost i skalabilnost, što dalje vodi ka distribuiranoj obradi, gde su podaci čuvani u različitim skladišnim sistemima, koji su fizički udaljeni. Kod aplikacija koje zahtevaju veliki broj upisa i čitanja neophodno je maksimalno smanjiti kašnjenje, što dalje iziskuje optimizaciju čitave infrastrukture korišćene za umrežavanje čvorova. Treba imati u vidu perspektivu ovakve mreže:

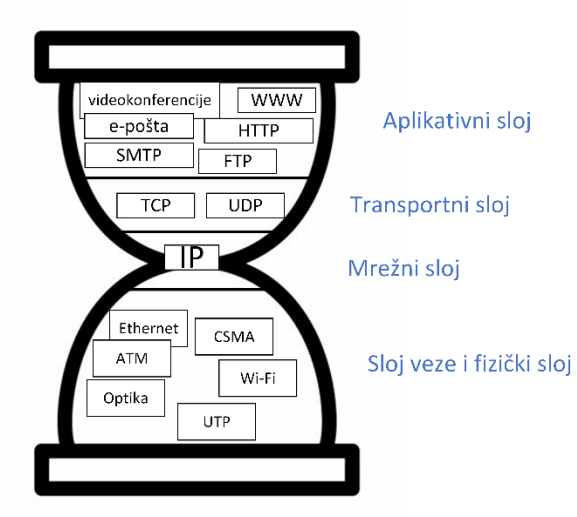
- Infrastruktura oblaka podrazumeva hiljade servera, skladišta i svičeva koji su međusobno povezani u data-centru.
- Sam data-centar je povezan sa spoljnim svetom putem brze internet infrastrukture, koja dalje prosleđuje IP pakete.
- Unutar data-centra postoji jedan čitav „mali internet“, kompleksna mreža, koja zahteva dobru povezanost, otpornost i dobro reagovanje na moguće otkaze.
- Interna mreža data-centra oblaka mora da podrži virtualizaciju i zahteve vezane za visoku fleksibilnost sistema koji obitavaju u oblaku, kao i softversko upravljanje mrežama.

U nastavku će prvo ukratko biti izložene osnove modernih računarskih mreža, sa posebnim osvrtom na elemente koji su od važnosti za oblak, a zatim će biti dat prikaz mrežnih tehnologija koje se koriste u data-centrima oblaka.

7.1. Osnove modernih mreža: Konvencionalni internet

Konvencionalni internet kakav se, uz određene adaptacije, koristi već više od 5 decenija unazad podrazumeva paketsku mrežu zasnovanu na protokolskom steku TCP/IP. Internet je tzv. *best-effort* mreža, IP paketi se šalju sa jednog kraja mreža na drugu bez garancije same mreže da će biti isporučeni. Zadatak infrastrukture je da se maksimalno „potrudi“ da paket stigne do odredišta, ali ništa više od toga. Sve ostale zadatke moraju obaviti pošiljalac i primalac.

Osnovna jedinica podataka na internetu jeste pomenuti IP paket. IP je skraćenica od Internet Protocol i predstavlja jedan od najznačajnijih mrežnih protokola. Protokoli su raspoređeni po slojevima i na taj način su postavljeni na svoje „dužnosti“. Poznati slojeviti modeli su OSI (Open System Interconnection) i TCP/IP, a u nastavku će biti reči o ovom poslednjem. Zbog intenzivne oslonjenosti kompletnog interneta na IP, može se konstatovati da konvencionalni internet ima oblik peščanog sata (Slika 91).



Slika 91 Paradigma interneta kao peščanog sata

Mreža je složen sklop i podrazumeva fizičko povezivanje, softversku podršku i ispunjavanje različitih zahteva koji su neophodni da bi aplikacija na kraju mogla nesmetano da funkcioniše sa podacima koje dobije.

TCP/IP sadrži 4 sloja[23]:

- Sloj pristupa mreži: Bavi se fizičkim i osnovnim logičkim aspektima umrežavanja, interfejsima, kabliranjem, signalima, okvirima podataka, konkurentnim pristupom itd. Postoji veliki broj protokola i standarda koji operišu na ovom sloju.
- Internet-sloj. Bavi se prenosom paketa kroz mrežu, što uključuje adresiranje, „cepkanje“ paketa na manje delove (fragmentaciju) i rutiranje. Faktički jedan protokol – IP – obavlja sav posao na ovom sloju, kako je to i predstavljeno u „modelu peščanika“.
- Transportni sloj vrši funkciju povezivanja aplikacija različitih krajnjih uređaja, na primer klijenta i servera. Protokol TCP pruža pouzdanu uslugu sa zasnivanjem veze, dok UDP nije pouzdan i ne zasniva vezu.
- Aplikativni sloj se bavi aplikacijama i najbliži je korisniku. Na ovom sloju funkcioniše niz otvorenih protokola, kao što su HTTP, FTP, BittTorrent, kao i veliki broj vlasničkih protokola za prenos multimedije, videokonferensing itd.

Adresiranje

Kada se vrši slanje podataka sa jednog uređaja na drugi (bili oni virtualni ili ne), potrebno je definisati ko šalje i kome se šalje da bi se dati podatak usmerio na pravo odredište. Krajnji uređaji (računari, serveri, mobilni uređaji itd.) unutar lokalne mreže, u kojoj su povezani unutar jedne prostorije ili jednog kompleksa, za adresiranje koriste MAC (Media Access Control) adrese. One se u praksi gotovo nikada ne menjaju i jednoznačno određuju uređaj, tačnije njegov mrežni interfejs. Na primer, ako ruter ima 8 priključaka na mrežu, svaki od njih imaće posebnu MAC adresu. MAC adresa se čuva u 6 bajtova, u heksadecimalnom formatu, u kojem prva polovina označava proizvođača, a druga dalje određuje sam

[23] U publikacijama i literaturi mogu se naći raznovrsne varijacije sa 4 ili 5 slojeva; u ovoj knjizi ostaje se pri četvoroslojnoj, jednostavnoj, ali preciznoj formulaciji.

interfejs.
Npr. E4-C7-67-4B-30-7F.

Paket poslat u lokalnoj mreži, npr. gde su uređaji spojeni na jedan svič, relativno lako nalazi odredište. Kada paket izađe iz okvira lokalne mreže, sadrži IP zaglavlje i opremljen je IP adresama (izvorišta i odredišta) i na osnovu ovih adresa ruteri usmeravaju paket između više mreža. U upotrebi su adrese verzije 4 i 6. IP v4 podrazumeva da se koristi protokol verzije 4 i da su IP adrese veličine 32 bita (4 bajta). Takve adrese se zapisuju dekadno u grupama po 4 broja, na primer 192.168.1.5. Svaki broj može imati vrednost 0 – 255. Kod verzije 6, adrese su znatno veće, imaju 128 bitova i zapisuju se heksadecimalno, na primer 2600:1f18:22ba:8c00:ba86:a05e:a5ba:00FF

IP adrese računara koji su u istoj mreži, podudaraju se u prvom delu IP adrese. Kada se računaru (ili pojedinačnom interfejsu, ako ih ima više) zadaje IP adresa, zadaje se i tzv. mrežna maska. Ona određuje koji deo IP adrese je zajednički za računare u istoj mreži. Na primer, u nekoj mreži postavke mreže mogu biti: 91.187.132.11/24. To znači da u navedenoj 32-bitnoj IP adresi 24 bita (to su u dekadnom zapisu upravo prva tri broja) identifikuju mrežu. Taj prvi deo se naziva maska ili prefiks, a ostali brojevi su pojedinačni identifikatori računara. Dati računar sa adresom 91.187.132.20 je prema tome u istoj mreži kao i 91.187.132.11 i može se reći da je njihova komunikacija direktnija; paketi se ne moraju slati preko posebnih rutera u druge mreže, koje su možda u vlasništvu drugih institucija itd.

Sa druge strane, prefiksom se definiše veličina određene mreže: duži prefiks – manja mreža. Kada se adresira mreža, deo koji je određen za računare postavlja se na nulu (sve binarne nule, ako bi se gledao binarni zapis). Na primer, mreža 10.5.5.0/24 jeste mreža kod koje će svi uređaji imati adrese koje počinju sa 10.5.5, a zatim će se poslednji broj definisati u rasponu 1– 254, tako da je ukupan broj uređaja u ovakvoj mreži preko 250. Treba imati u vidu da se u praksi prva adresa (10.5.5.1) dodeljuje ruteru na koji su povezani svi uređaji, a poslednja, 255, predstavlja *broadcast*. Kada uređaj pošalje nešto na tu adresu, to je isto kao da je poslao svim uređajima u mreži. Pored ovih posebnih adresa, u virtualnim mrežama u oblaku, mogu biti izdvojene i druge adrese specijalne namene, koje se ne dodeljuju „običnim" uređajima, već imaju posebnu namenu.

Privatne adrese. Pojedini opsezi IP adresa su definisani kao ne-rutabilni, tj. ove adrese se ne mogu pojaviti van lokalnih mreža. Najpopularniji rang privatnih adresa su 192.168.0.0/16, koje koristi većina kućnih rutera, odnosno manjih lokalnih mreža. Uglavnom je praksa da se koristi jedna podmreža iz ovog ranga, na primer 192.168.1.0/24. Drugi opsezi su mreža 10.0.0.0/8 (to znači sve adrese koje počinju sa 10) i 172.16.0.0-172.31.255.255. Ukoliko računari sa ovakvim adresama treba da pristupe internetu, odnosno da „izađu" iz lokalne mreže tako što će poslati IP paket na neku javnu adresu van lokalne mreže, neophodno je da u nekom trenutku njihova izvorišna IP adresa, budući da je privatna i da nije jedinstvena, bude zamenjena jedinstvenom adresom rutera. Ovu funkciju vrši NAT, o kojem će biti više reči nešto kasnije.

MAC adrese funkcionišu na niskom nivou, nivou pristupa mreži, dok se IP adrese koriste u komunikaciji između više mreža. Za komunikaciju između *aplikacija* kroz mrežu koriste se portovi. Oni predstavljaju transportnu adresu aplikacije koja je locirana na određenoj IP adresi. Kada IP paket stigne do odredišta, dakle do određene IP adrese, važno je da se podaci koje nosi isporuče odgovarajućoj aplikaciji koja radi na toj IP adresi. Pošto na jednom računaru, koji ima jednu IP adresu, može raditi više aplikacija koje komuniciraju sa mrežom, neophodno je odrediti koja aplikacija tačno dobija dati podatak. Kako je već pomenuto, na transportnom sloju aktivni su UDP i TCP. Aplikacija koristi ili jedan ili drugi i onda „cilja" odgovarajući port, na primer web-server „sluša" na TCP portu 80 i klijent će uputiti dati paket ka tom portu. Na raspolaganju je 65.536 portova, tj. od 0 do 65.535. Portovi od 0 do 1023 mogu biti korišćeni

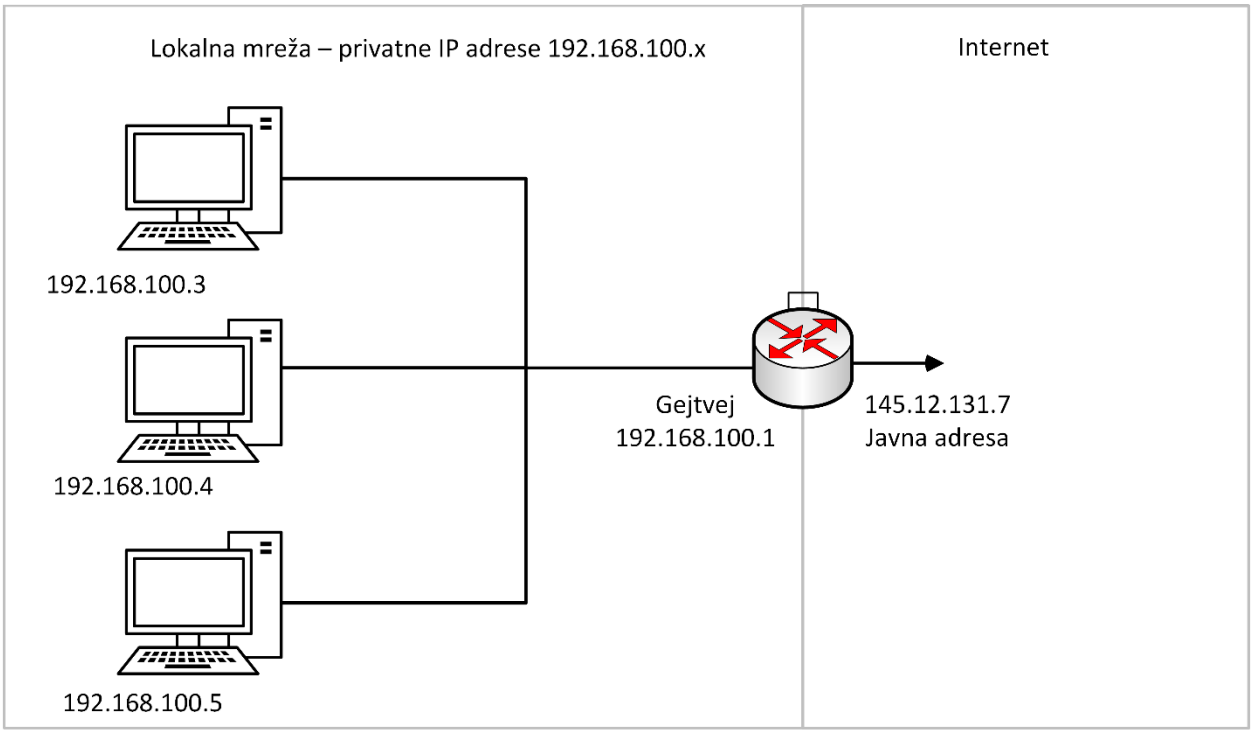
samo ako dati servis ili aplikacija imaju administratorske privilegije. Viši portovi mogu biti korišćeni kroz regularne naloge.

Portovi za aplikacije koje treba nesmetano da funkcionišu moraju biti otvoreni. Ukoliko je u upotrebi zaštitni softver, npr. fajervol, dati portovi moraju se otvoriti da bi aplikacija mogla biti kontaktirana.

Rutiranje je postupak usmeravanja paketa kroz mrežu. Ruter je uređaj koji ima više interfejsa i povezan je na više mreža. Kada dobije paket na jednom interfejsu, mora da „zna“ na koji interfejs da prosledi taj paket da bi on stigao u odgovarajuću mrežu. Kada na svom putu paket stigne do rutera koji je povezan na njegovu odredišnu mrežu, takav ruter će ga na kraju proslediti konkretnom uređaju koji treba da primi paket. Kako ruter „zna“ na koji interfejs da prosledi paket? Postoje različiti algoritmi i protokoli koje ruteri koriste da međusobno „razgovaraju“ i razmene svoja saznanja o različitim mrežama na koje su povezani, tj. da upoznaju topologiju mreže. Nakon dobijanja informacija od drugih rutera, svaki pojedinačni ruter je u stanju da formira svoju tabelu rutiranja, koja govori o tome kom ruteru se šalje paket sa određenom odredišnom adresom. Upravo na osnovu tabele rutiranja, koja je opet formirana na osnovu saznanja o topologiji mreže, formira se i interna tabela prosleđivanja. U ovoj tabeli je definisano koji paketi se prosleđuju na koji interfejs. Na primer, ako IP adresa dolaznog paketa počinje sa 91.187, prosleđuje se na interfejs eth3. Ukoliko neki paket nema eksplicitno definisanu akciju u tabeli prosleđivanja, onda se upućuje na tzv. *last resort* rutu. Jednostavno, jedan interfejs je odvojen za prosleđivanje ukoliko ne postoji specifično navedeno šta raditi sa određenim paketom.

NAT

Internet je inicijalno zamišljen tako da svaki uređaj ima jedinstvenu, javnu IP adresu. Međutim, relativno brzo je došlo do nestašice IP adresa (verzije 4). Rešenje je pronađeno u tome da uređaji mogu imati IP adrese koje globalno nisu jedinstvene, ali tako da ne pristupaju internetu direktno, već preko uređaja koji imaju jedinstvene adrese. Ovakav princip prevođenja privatnih adresa u javne jeste NAT – Network Address Translation. Na slici Slika 92 dat je koncept NAT-a.



Slika 92 Primer NAT-ovane mreže

Kada računar sa privatnom adresom 192.168.100.3 treba da kontaktira bilo koji računar iz iste mreže, komunikacija se obavlja jednostavno, bez uplitanja rutera (sa adresom 192.168.100.1). Međutim, kada računar 192.168.100.3 treba da pošalje paket nekom serveru spolja, na primer, korisnik želi da poseti sajt eucenje.ftn.kg.ac.rs, koji se nalazi na adresi 91.187.132.18, onda se postupak odvija preko rutera. Ruter ima jedan interfejs koji je povezan na lokalnu mrežu sa privatnom adresom iz te mreže, npr. 192.168.100.1 i drugi interfejs koji ima javnu adresu, na primer 145.12.131.7. Ruter, koji u ovom slučaju igra i ulogu tzv. NAT-box-a, će u svojoj internoj evidenciji ubeležiti da je računar 192.168.100.3 uputio paket ka odredištu 91.187.132.18 i onda će u paketu zameniti IP adresu pošiljaoca sa svojom adresom (145.12.131.7). Paket dalje biva usmeren kroz mrežu i stiže do servera. Server sada treba da odgovori na zahtev, na primer da izda neku web-stranu. On upućuje paket koji ima izvorišnu adresu servera i odredišnu adresu rutera, dakle javnu adresu 145.12.131.7. Server „ne zna“ za računar koji mu je inicijalno poslao zahtev, njegova adresa se nije nalazila u IP paketu; server „vidi“ samo javne adrese. Kada ruter-NAT primi ovaj paket, na osnovu svoje evidencije poslatih zahteva proslediće odgovor baš računaru sa kojeg je upućen zahtev, odnosno posećen sajt eucenje.ftn.kg.ac.rs. Ruter će pred samo slanje zameniti adresu odredišta privatnom adresom računara-pošiljaoca i na taj način će se „zatvoriti krug“.

NAT omogućava da veliki broj računara može da pristupa internetu kroz jednu IP adresu. Sami računari imaju proizvoljne lokalne adrese; važno je da pripadaju istom opsegu i da su privatne. Ovakvi računari su zaštićeni od direktnog pristupa spolja, jer na internetu nisu vidljivi – vidljiv je samo ruter. Sa druge strane, ukoliko je potrebno da omogućimo nekom servisu lokalne mreže da bude vidljiv spolja, to se može ostvariti tako što se odgovarajući portovi rutera usmere ka datom računaru ili se pojedini računari postave u tzv. demilitarizovanu zonu.

NAT je od naročitog značaja za virtuelne mreže koje se kreiraju u oblaku. Ako se određen broj virtuelnih mašina poveže u jednu privatnu mrežu, potrebno je definisati NAT kao oblik pristupa. NAT je definisan kao servis i može omogućavati pristup ka internetu ili samo pristup ka drugim unutrašnjim mrežama.

DNS (Domain Name System) još jedan je od ključnih servisa na internetu. On obezbeđuje povezivanje simboličkih i numeričkih adresa. Niz DNS servera na distribuiran način čuva podatke o domenskim imenima. Nadležni serveri za domen org, čuvaju adrese svih servera koji su nadležni za adrese na domenu org, a ti serveri dalje čuvaju IP adrese koje su vezane za konkretne servere. Npr. za domen wikipedia.org jedan od nadležnih servera je ns0.wikimedia.org. Taj server na kraju čuva par www.wikipedia.org – 185.15.59.224. Kada klijent treba da kontaktira www.wikipedia.org, prvo se dobavlja IP adresa sajta kroz DNS upite ka DNS serverima, a onda se zahtev upućuje koristeći datu IP adresu. Ovo je elementarna funkcija DNS-a – DNS server ima u tzv. A zapisu ubeležen par IP-adresa – simboličko ime i daje ga klijentima kojima je to potrebno. Postoje i druge vrste zapisa, kao što je npr. zapis NS, koji čuva IP adresu nadležnog servera za neki domen, zatim CNAME, koji čuva drugo ime (alias) postojećeg simboličkog imena, MX, koji čuva IP adresu mejl-servera za određeni domen, PTR, koji čuva reverzni zapis itd.

Pored osnovnih funkcija vezanih za razrešavanje imena u IP adrese, DNS može da učestvuje i u raspoređivanju opterećenja (load balancingu). Servis može da bude pokrenut na više servera i tada za jedan upit DNS server može dati kao odgovor IP adresu prvog servera, a već za naredni može dati drugu IP adresu tako da se zahtevi rasporede.

Slojevitost interneta i autonomni sistemi

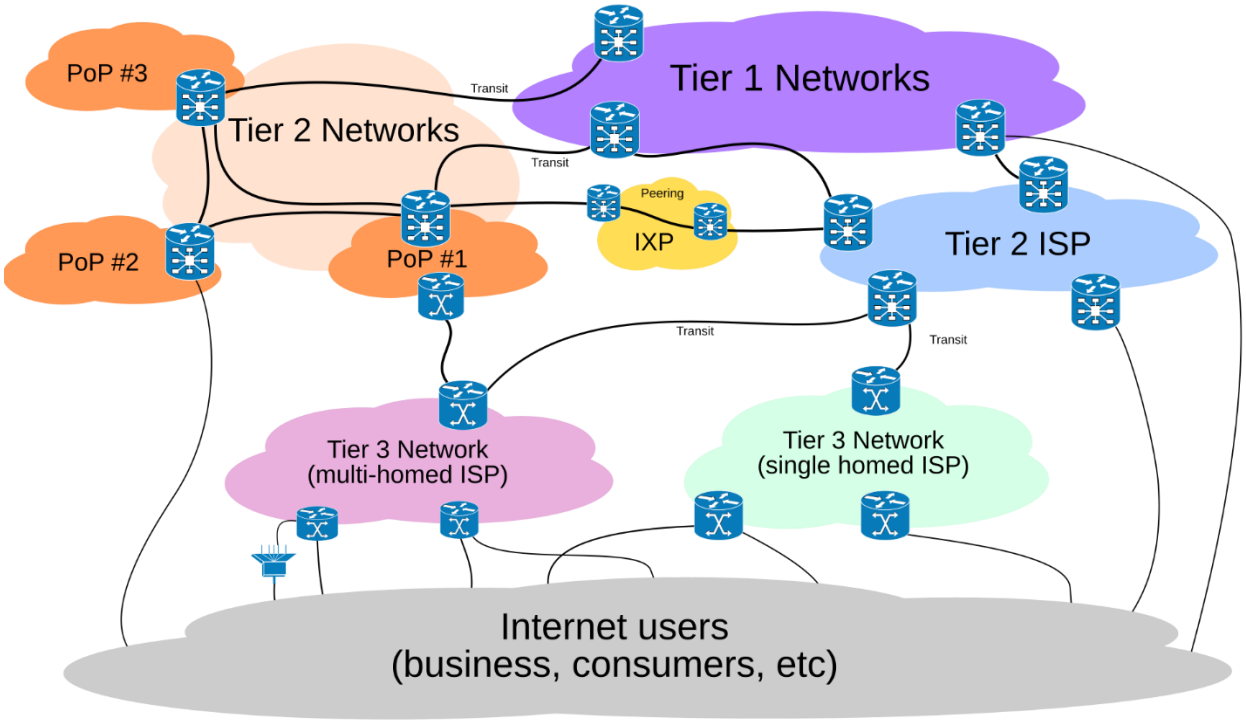
Do sada je bilo reči o lokalnim mrežama i internetu, bez ulaženja u detalje kako je sam internet organizovan. Uobičajeno je da korisnici imaju internet provajdera preko kojeg se povezuju na internet

i tu se završava čitav postupak. Zaista, za krajnjeg korisnika nije od velikog značaja šta se „krije“ iza provajdera i kako je sam provajder povezan na internet. Sa druge strane, infrastruktura u oblaku jeste jedan značajan element interneta i korisno je postaviti ga u određen, detaljnije objašnjen koncept.

Veliki sistemi, kao što su multinacionalne kompanije i akademske mreže jesu tzv. autonomni sistemi (AS). Svaki ovakav sistem je povezan na druge sisteme koristeći principe koji su usaglašeni među AS. Sa druge strane, unutar svakog autonomnog sistema vlasnik može da definiše proizvoljno pravila rutiranja i uopšte upravljanja mrežom. Dobavljači usluga u oblaku su uglavnom autonomni sistemi. Kada neki uređaj iz jednog autonomnog sistema komunicira sa uređajem iz drugog autonomnog sistema, neophodno je da se u nekom trenutku zahtev prosledi iz jednog AS u drugi. Čitav AS (kao u primeru sa lokalnim mrežama) ima ruter koji ga povezuje sa drugim AS. Pojedini AS mogu razmenjivati podatke besplatno i to je tzv. peering. U nekim slučajevima je potrebno da podaci prođu kroz jedan AS da bi stigli do drugog i takav prolazak se plaća. Reč je o tranzitu.

Najniži, treći sloj upotrebljavaju mali internet-provajderi. Oni moraju da realizuju povezanost koristeći usluge viših nivoa provajdera i to se plaća. Na drugom sloju su provajderi koji koriste viši nivo, ali mogu da komuniciraju i kroz peering, tako da više autonomnih sistema funkcioniše na istom nivou i to rade „u kompenzaciji“, tj. ne plaća se posebno.

Sloj 1 (tier 1) obuhvata provajdere koji ne plaćaju saobraćaj. Preko njih se obavlja veliki deo globalnog saobraćaja, a oni međusobno komuniciraju kroz peering (Slika 93).



Slika 93 Prikaz međupovezanosti interneta (Image by Ludovic.ferre, creative commons)

Mreže iz različitih slojeva su povezane kroz tzv. Point of Presence (POP), a na nivou peeringa mogu postojati posebna internet čvorišta, tzv. IXP (Internet Exchange Points).

6.3. Softverski definisane mreže

Rutiranje je jedna od ključnih operacija na internetu. Tradicionalno rutiranje podrazumeva da se IP paketima i logikom rutiranja upravlja na istom mestu. Ruteri, kao uređaji koji rade na mrežnom sloju, formiraju tabele rutiranja i na osnovu njih prosleđuju IP pakete. Algoritmi koji upravljaju formiranjem tih tabela definišu se na nivou autonomnog sistema i ruteri samostalno kreiraju tabele prosleđivanja.

Među konvencionalnim algoritmima rutiranja su RIP (Routing Information Protocol) i OSPF (Open Shortest Path First). Ono što je zajedničko za ova dva protokola jeste to što ruteri samostalno međusobno razmenjuju informacije o svojoj percepciji mreže, na osnovu kojih formiraju tabele rutiranja. Na primer, ruter A može da izmeri ping do svojih susednih rutera, a zatim tu informaciju (zajedno sa ostalim, ako ih ima) prosledi svojim susedima. Kada svi ruteri postupe na taj način, nakon izvesnog vremena, svi će imati podatke o vremenu kašnjenja između svih suseda. Na osnovu tih podataka kreiraće tabele rutiranja sa najmanjom ukupnom udaljenošću do svakog rutera. Ovaj pristup karakterističan je za algoritam Distance Vector Routing, na kojem se bazira RIP.

S druge strane, ruter može da izmeri kašnjenje do svojih suseda i tu informaciju prosledi svim ruterima istovremeno (koristeći tehniku *flooding*). Na taj način, svaki ruter dobija podatke o svim kašnjenjima u mreži. Ovako funkcioniše OSPF.

Poenta oba pristupa jeste u tome da ruteri sami donose odluke o rutiranju. Za generisanje i ažuriranje tabela rutiranja neophodno je određeno vreme. Sve operacije odvijaju se na mrežnom sloju i nisu povezane s drugim funkcijama, osim rutiranja.

Softverski definisane mreže (SDN) su nastale kao odgovor na ograničenja tradicionalnih mrežnih arhitektura. Ključni razlozi koji su doveli do razvoja SDN-a uključuju:

1. **Složenost upravljanja mrežama**

   Tradicionalni mrežni uređaji (ruteri, svičevi) imaju ugrađene fiksne kontrole i funkcije za upravljanje mrežnim saobraćajem, što otežava njihovu konfiguraciju i prilagođavanje promenama u mreži. Ovo je posebno problematično u velikim i dinamičkim okruženjima, gde su prisutne česte promene topologije ili opterećenja.

2. **Ograničena fleksibilnost**

   Kod tradicionalnog pristupa, pravila rutiranja i prosleđivanja implementirana su na svakom uređaju pojedinačno. Ovakva rigidna struktura otežava uvođenje novih servisa ili promena jer su istovremeno neophodne intervencije na mnogim uređajima.

3. **Eksplozivan rast mrežnih uređaja i servisa**

   Pojava računarstva u oblaku, IoT uređaja i povećanje broja korisnika i aplikacija zahtevali su veću skalabilnost i mogućnost dinamičkog upravljanja mrežama, što tradicionalne arhitekture nisu mogle efikasno da pruže.

4. **Potencijal za automatizaciju**

   Povećana potreba za brzim i automatizovanim upravljanjem mrežnim resursima motivisala je razvoj tehnologija koje omogućavaju centralizovano upravljanje mrežama.

Kod softverski definisanih mreža razdvojeni su slojevi podataka i upravljanja. Podacima se bave ruteri, dok je upravljanje centralizovano i izvodi se softverski, od strane kontrolera.

Ruteri dobijaju instrukcije od kontrolera kako da postupe sa paketima, to je server koji odlučuje. Svaki ruter od kontrolera dobija tabelu tokova. Tok *(flow)* je jedinica kojom se upravlja u SDN-u. Može se definisati na osnovu različitih parametara, npr. svi paketi koji pristižu sa određene adrese i određenog porta mogu se tretirati kao jedan tok i takav tok se može proslediti na određeni izlaz rutera. Nad tokovima se mogu sprovoditi različite operacije. Osnovna je prosleđivanje, ali tok i svi njegovi paketi mogu biti blokirani (funkcija fajervola) ili, ako ispunjavaju određeni uslov, prosleđeni samom kontroleru na dodatnu analizu.

Prvi široko prihvaćeni protokol softverski definisanih mreža je OpenFlow, koji je korišćen od strane velikih dobavljača usluga u oblaku. Danas su u upotrebi fleksibilnija rešenja, kao što je VXLAN.

U dosadašnjem izlaganju uglavnom je bilo reči o ruterima, kao uređajima koji rukuju IP paketima. Konvencionalna kategorizacija uređaja podrazumeva da ruteri funkcionišu na sloju mreže (L3), dok svičevi rade na nižem, sloju veze (L2). U modernim infrastrukturama, razlika između ova dva uređaja je maglovita jer i svičevi mogu da rade na mrežnom sloju. U tom smislu, u daljem izlaganju će se i naziv svič odnositi na napredne, L3 svičeve, koji „razumeju“ IP pakete.

Fajervol

Fajervol ili zaštitna barijera jeste uređaj ili softver koji štiti pojedinačni server ili čitavu mrežu. Zaštita se pre svega vrši na osnovu filtriranja paketa po osnovu različitih parametara koji se nalaze u zaglavlju, na primer IP adresa, TCP port itd. Fajervol sa praćenjem stanja (statefull) vodi evidenciju o zasnovanim TCP konekcijama i to može koristiti kao kriterijum za odobrenje ili zabranu određenog saobraćaja.

Napredni, tzv. fajervoli naredne generacije, podržavaju i funkcije duboke inspekcije paketa, prepoznavanja aplikacija, izolovanja sumnjivog saobraćaja, integraciju sa drugim bezbednosnim rešenjima itd.

7.2. Mrežna infrastruktura oblaka

Korisnici usluga u oblaku pristupaju brojnim uslugama preko svog internet linka, odnosno VPN-a, koristeći određen web-interfejs ili API. U sklopu svog pretplatničkog plana usluga, korisnik-klijent ima mogućnost da kreira virtuelne mašine, umrežava ih, formira virtuelni privatni oblak, definiše gejtvej za taj oblak, konfiguriše adrese, dozvoljava ili brani određeni saobraćaj, konfiguriše balansiranje opterećenje itd. Faktički, sve ove operacije se izvode nad virtuelizovanim uređajima. Usluge kao što su CloudFront, Lambda, Forecast i mnoge druge jesu apstrakcije ispod kojih se nalaze određene virtuelne mašine koje, opet, funkcionišu na realnom hardveru. Taj hardver se nalazi u data-centrima, umrežen je i mora da pruži visoke performanse, da bude robustan i skalabilan da bi apstrakcije koje klijent vidi i koristi mogle efikasno da vrše svoje funkcije.

Na primer, klijent želi da instalira aplikaciju koja će da dobro skalira i u tom smislu koristi autoskaliranje. To znači da će se, ako opterećenje dostigne određen prag, pokrenuti dodatne virtuelne mašine. Originalna grupa virtuelnih mašina je formirana da pripada istoj virtuelnoj mreži. Dakle, same virtuelne mašine imaju svoje IP adrese na nivou virtuelne mreže i oblaka. Takve virtuelne mašine se nalaze na određenim fizičkim serverima – verovatno na potpuno različitim i razdvojenim mašinama, koje imaju svoje IP adrese (nazovimo ih realnim IP adresama). Kada se pokrenu nove virtuelne mašine, one se nalaze na drugim fizičkim mašinama, koje su u data-centru povezane u mrežu sa drugim fizičkim mašinama. Ekstremniji slučaj je da mogu biti u potpuno različitom data-centru. Sve ove umrežene mašine su vidljive klijentu u jedinstvenoj mreži, sa IP adresama koje pripadaju istoj podmreži, vidljive su jedna drugoj, a u fizičkoj infrastrukturi su „raštrkane“.

Fizička (tzv. underlay) mreža ima zadatak da podrži servise vidljive (tzv. overlay) mreže:

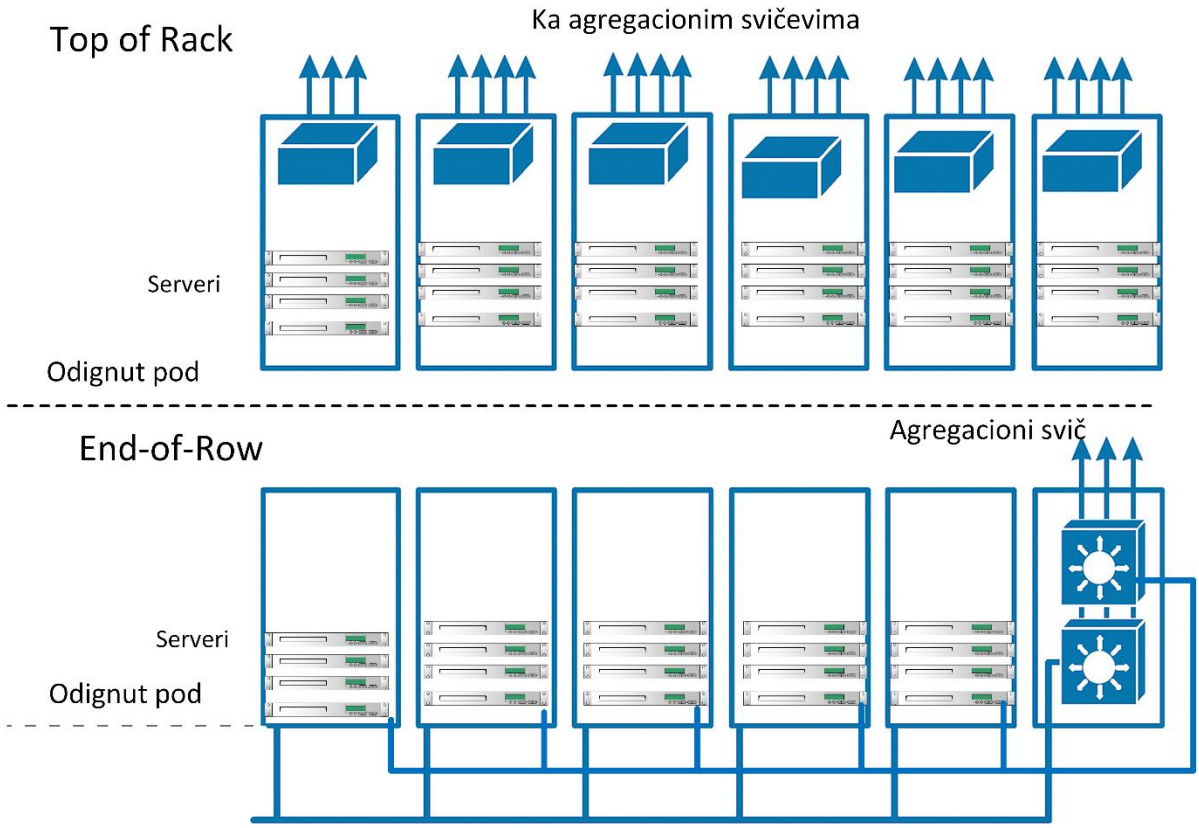
- u smislu performansi
- u smislu izolovanja korisnika i usluga
- u smislu skaliranja

U nastavku će biti predstavljene „underlay“ tehnologije umrežavanja koje se koriste u modernim data centrima oblaka. Kao i kod svake druge tehnologije, počeci su bili ograničeni, a sa narastanjem potreba razvijana su nova rešenja. Da bi se stekao uvid u evoluciju pristupa umrežavanju, biće dati prikazi početaka.

Fizička organizacija mrežne opreme u data-centru oblaka

Data-centar oblaka kompleksna je struktura koja pored računarskih resursa obuhvata i komunikacioni sistem i sistem napajanja.

Osnovne jedinice organizacije povezanih računarskih sistema jesu rack-ormani. Unutar jednog ormana može se nalaziti veći broj servera i skladišta. Ormani mogu biti međusobno povezani tzv. ToR (top of the rack) svičevima, gde je svaki orman opremljen svičem, koji je nadalje povezan na naredni nivo svičeva (svičevi agregacije). Ovaj pristup omogućava lako proširenje i iziskuje manje kablova. Drugi pristup je povezivanje svih servera direktno na agregacione svičeve *(end-of-row).* Ovaj pristup je manje fleksibilan, ali troši manje energije i zahteva manji broj svičeva. Obe varijante prikazane su na sliciSlika 94.



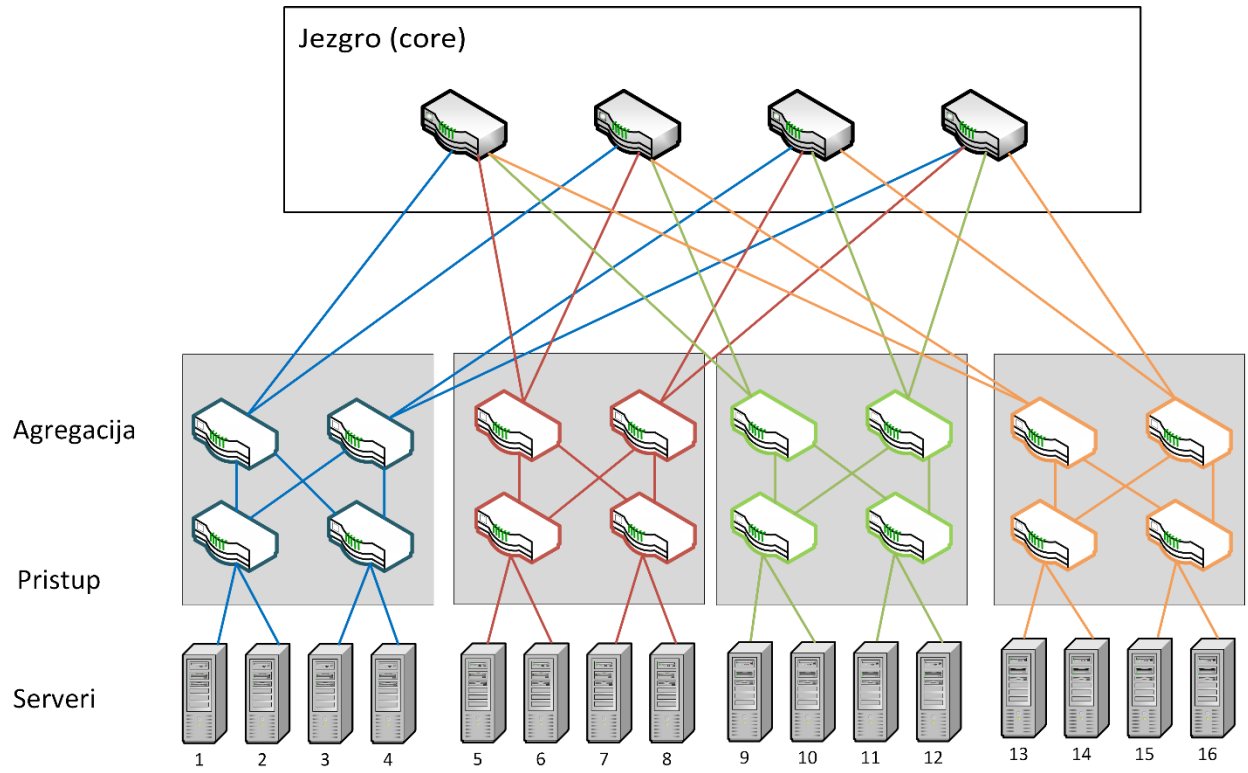
Slika 94 Pristupi organizaciji servera u data-centru: Top of Rack i End-of-Row

Da bi ovakva struktura mogla da skalira i bude isplativa, sačinjena je od tzv. *commodity* hardvera. To znači da se odgovarajuće komponente serijski proizvode, mogu se naći relativno povoljno na tržištu i nije potrebno posebno ih projektovati i proizvoditi specijalizovanim postupcima.

Fat Tree Network

Računarski sistemi se ne povezuju neposredno, serijski, već se ToR svičevi dalje umrežavaju sa narednim nivoom svičeva, tzv. agregacionim slojem, koji sadrži manji broj svičeva, a objedinjuje sve povezane sisteme. Takođe, međusobno povezivanje svičeva realizuje se putem više linkova. Ovakva struktura je poznata kao Fat Tree Network (Slika 95). Postoje tri sloja putem kojih je izvedeno potpuno umrežavanje. Sloj pristupa predstavlja ToR svičeve, koji su dalje povezani na agregacione svičeve, a sami agregacioni svičevi jedne grupe su povezani međusobno. Dalje, sloj agregacije je povezan na jezgro (core), gde se nalaze glavni svičevi. (Struktura je slična globalnoj internet konfiguraciji, sa slojevima provajdera.)

Topologija Fat Tree je u suštini varijanta tzv. Clos mreže, koja je korišćena u telefonskoj infrastrukuri.



Slika 95 Arhitektura Fat Tree mreže

Grupa pristupnih i agregacionih svičeva naziva se *pod*. Na slici Slika 95 prikazana su četiri poda (od po 4 sviča). U datoj strukturi se nalaze tri sloja. Core-sloj ima 4 sviča. Svi svičevi imaju po 4 porta. Svaki pristupni svič je povezan sa 2 krajnja servera i dva agregaciona sviča. Svaki pod je povezan na 4 servera. Ako se proračun generalizuje i broj portova označi sa k, ukupan broj servera je $N = k \cdot \frac{k^2}{4} = \frac{k^3}{4}$

Fat Tree Network obezbeđuje više ciljeva za oblak:

- Skalabilnost — Ukoliko je potrebno povezati dodatne servere, mogu se dodati novi svičevi ili eventualno ugraditi svičevi sa više portova.
- Raspodela opterećenja i robusnost — Povezanost između bilo koja dva servera u data-centru je ostvarena preko više linija. Time se pre svega dobija na ravnomernosti opterećenja — uklanjaju se uska grla tako što se saobraćaj usmerava kroz različite linije. Takođe, povećana je

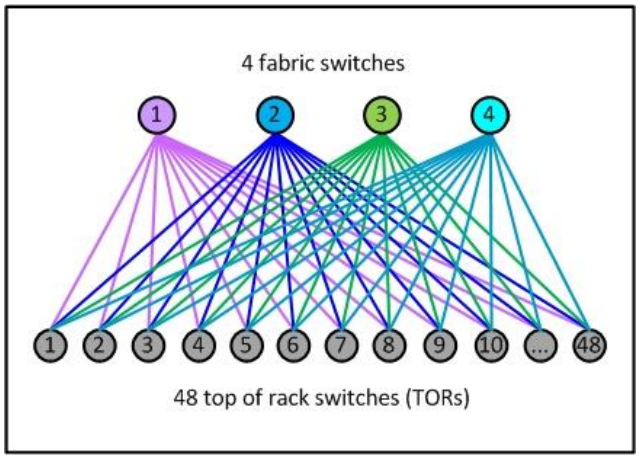
otpornost na otkaz; ukoliko otkaže jedna linija, drugom će se preneti podaci dok se ne otkloni kvar.

U ovom trenutku značajno je pomenuti pojam *oversubscription*. U pitanju je termin koji se može sresti u različitim delatnostima, npr. u avio-prevozu. Podrazumeva da je rezervisani kapacitet veći od mogućnosti samog resursa. Konkretno, u umrežavanju, primer bi mogao biti sledeći: dva sviča su povezana linkom od 1 Gbps, a na svaki od njih su povezana po dva računara (svaki na link od 1 Gbps). Ako bi sada svi računari komunicirali, link između dva sviča ne bi bio dovoljan. Dva računara bi iziskivala ukupno 2 Gbps saobraćaja ka drugom sviču, a link između svičeva podržava 1 Gbps. Kaže se da je odnos oversubscription-a u ovom slučaju 2:1. Oversubscription je uobičajen u mrežama, ali i generalno kod rezervacije resursa. Poenta je u tome da je relativno mala verovatnoća da dati računari istovremeno komuniciraju maksimalnom brzinom. Ipak, visoke vrednosti odnosa oversubscription-a povlače veću mogućnost zagušenja, te je potrebno dobro planirati konfiguraciju.

Veliki sistemi, kao što su Google, AWS, Microsoft, dizajnirali su specifične arhitekture za svoje mreže. One se suštinski zasnivaju na Fat Tree Network.

Studija slučaja — Facebook mrežna infrastruktura oblaka

Facebook je uspostavio kao „jedinicu“ svoje mreže — *pod*, pri čemu se jedan pod sastoji iz 48 ToR svičeva i 4 tzv fabric-sviča (Slika 96)[24].

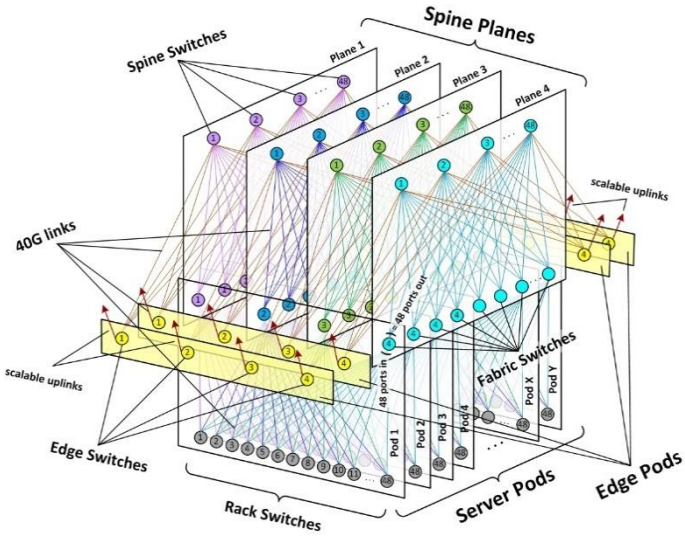


Slika 96 Pod — jedinica mreže

Kompletna mreža je organizovana u četiri ravni-kičme. Fabric-svičevi su povezani na svaku ravan (Slika 97). Facebook je u svrhe realizacije svoje mreže razvio sopstvene varijante svičeva[25].

[24] https://engineering.fb.com/2014/11/14/production-engineering/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/

[25] Detaljniji opis Facebook-ove mrežne arhitekture dat je na Facebook-ovom zvaničnom blogu: https://engineering.fb.com/2014/11/14/production-engineering/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/



Slika 97 Logička šema mrežne infrastrukture Facebook-a

7.3. Protokoli umrežavanja u oblaku

Infrastruktura data-centra oblaka podržava konvencionalne protokole. Međutim, izuzetno intenzivan saobraćaj i zahtevi vezani za virtuelizaciju i fleksibilnost, iziskuju kreiranje novih protokola, adaptacije postojećih i, na kraju, primenu postojećih protokola na drugačiji način.

VLAN

VLAN — Virtual LAN jeste tehnologija koja omogućava logičko razdvajanje mreže računara povezanih na jedan svič. Pored standardnih elemenata zaglavlja postavlja se identifikator VLAN-a, tako da se saobraćaj za različite logičke mreže odvaja. (Ovaj postupak se naziva tagovanje, koristi se protokol 802.1q i virtuelne mreže se razlikuju prema različitom VLAN ID-u.) Na primer, na sviču portovi od 1 do10 mogu pripadati jednom VLAN-u, a portovi od 11 do 24 drugom. Saobraćaj između ova dva VLAN-a je time razdvojen. Naravno, moguće je povezati više svičeva, tako da ova konfiguracija pokrije veći broj računara. Ovakav pristup je generalno pogodan za organizacije i manje konvencionalne data centre. Međutim, ograničenje od 4094 mreže može biti ozbiljna prepreka za velike sisteme. Takođe, VLAN je tesno vezan za niže slojeve, dok se kod oblaka umrežavanje pomera daleko od fizičkog i teško je vršiti izmene ako je VLAN osnovni mehanizam za organizaciju.

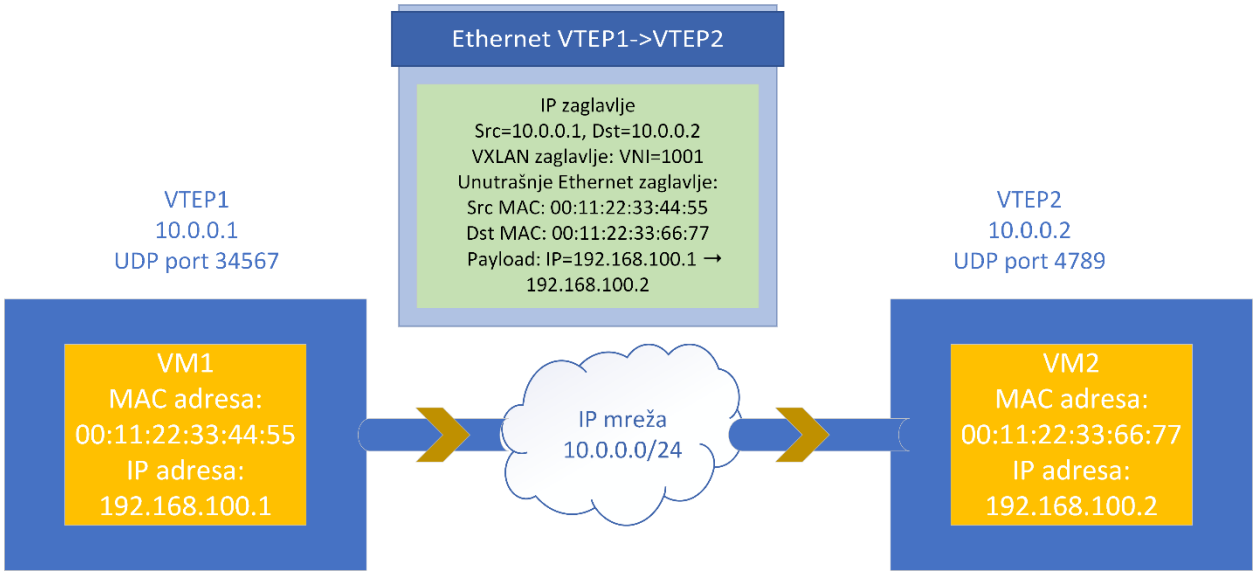
VXLAN

Virtual eXtensible Local-Area Network — VXLAN se može zamisliti kao veoma fleksibilna varijanta VLAN-a[26]. On omogućava formiranje 16 miliona mreža, čime se u potpunosti pokriva skaliranje. VXLAN funkcioniše tako što enkapsulira ramove u UDP datagrame, čime se saobraćaj „izdiže“ na više nivoe OSI-a i usmerava kroz različite podmreže. Virtuelni serveri koji se nalaze na potpuno različitim fizičkim serverima ponašaju se kao da su u istoj lokalnoj mreži. Za označavanje logičkih mreža koristi se VNI — Virtual Network Identifier, slično VLAN ID-u kod konvencionalnih VLAN mreža.

Mobilnost je omogućena primenom koncepta VxLAN kranje tačke tunela (eng. VxLAN tunnel endpoint — VTEP). VTEP može da bude bilo koji mrežni uređaj koji ima mogućnost obrade VxLAN paketa.

[26] VXLAN je opisan u standardu RFC 7348.

Kada VTEP uređaj primi paket koji je namenjen lokalnom računaru, ovaj uređaj vrši dekapsulaciju paketa i lokalnom računaru prosleđuje samo originalni ram, u virtuelnoj mreži definisanoj VNI-brojem. Na taj način, ceo proces se apstrahuje i lokalni računar „vidi“ samo običan ram (Slika 98).



Slika 98 VXLAN enkapsulacija

Protokoli za izbegavanje petlji

Petlja (loop) dovodi do beskonačnog kruženja ramova i tzv. *broadcast storm*-a. U konvencionalnim mrežama, petlje se izbegavaju uz pomoć protokola Spanning Tree (STP).

Ovaj automatizovani protokol funkcioniše tako što svičevi umesto topologije koja sadrži petlje, kreira topologiju razgranatog stabla. Ovakva topologija, jednom uspostavljena, koristi se konstantno dalje u mrežnom saobraćaju. Ako eventualno dođe do otkaza nekog sviča, potrebno je određeno vreme da se razmene poruke STP-a i formira novo razgranato stablo.

Dok je STP potpuno zadovoljavajući za konvencionalne lokalne mreže, kod infrastrukture oblaka ispoljavaju se pojedini nedostaci. Svičevi koji su pri vrhu razgranatog stabla lako mogu postati preoterećeni. Ukidanje petlji rešava problem kruženja ramova, međutim, takođe i ukida alternativne putanje koje bi mogle da relaksiraju saobraćaj (ako bi postojao mehanizam da se koriste naizmenično). Takođe, otkaz sviča dovodi do prekida rada mreže, bar dok se ne rekonfiguriše. U tom smislu za potrebe data-centara razvijeni su novi protokoli: TRILL, QFabric i VCS.

TRILL (Transparent Interconnection of Lots of Links) kombinuje tehnike sloja veze i mrežnog sloja (2. i 3. sloja) OSI-a. Radi slično kao link-state routing, samo što umesto IP mreža tretira VLAN-ove. Svaki svič poznaje put ka drugom sviču; ove informacije se dodaju u zaglavlje TRILL paketa. TRILL zahteva nove sviceve koji podržavaju TRILL.

VCS (Virtual Chasis Fabric) omogućava logičko ujedinjenje više svičeva, tako da se posmatraju kao jedan veliki svič. Ovim se pojednostavljuje upravljanje u ubrzava komunikacija između svičeva.

QFabric produbljuje ideju VCS-a i formira jedan „super-svič“ od svih svičeva data-centra. Rezultat je taj da se server, gde god se nalazio, ponaša kao da je direktno povezan na drugi server.

VCS i QFabric „poravnavaju“ mrežu, tako da ona gubi kompleksnost, putanje se izračunavaju unapred i postupak usmeravanja je značajno ubrzan.

Protokoli mrežnog sloja

Dok se protokoli 2. sloja jednostavnije koriste — uglavnom se sve automatski konfiguriše, kod protokola 3. sloja situacija je nešto složenija.

Za mreže srednje veličine, konvencionalni protokoli rutiranja, kao što je OSPF, i dalje su upotrebljivi. No, kod većih mreža, često se menja topologija i OSPF-u je potrebno više vremena da ažurira tabele rutiranja. Kod većih mreža, takođe, sami ruteri mogu biti preopterećeni intenzivnim slanjem informacija o linkovima.

Kod većih mreža izbor pada na varijante BGP-a i na softverski definisane mreže.

BGP

Border Gateway Protocol — BGP jeste vektorski protokol koji nije izvorno namenjen rutiranju unutar autonomnih sistema, već naprotiv, između autonomnih sistema (AS). Zadatak BGP-a je da pomogne u definisanju najbolje putanje. Takva putanja ne mora biti najkraća, već najbolja, tj. mora poštovati politiku zadatu od strane administratora sistema. Ukoliko postoji više putanja ka odredištu, BGP će na osnovu metapodataka o opcijama izabrati onu koja je najpovoljnija. BGP, iako mu je osnovna funkcija vezana za mrežni sloj, komunicira preko viših slojeva i koristi TCP port 179 za prosleđivanje informacija. Jedna od ključnih struktura podataka koja se održava u BGP-u jeste AS Path — to je lista autonomnih sistema kroz koje je prošao paket. Ako ruter u tom nizu uoči svoj ID, neće prihvatiti taj paket da ne bi proizveo petlju.

Kod rutiranja u oblaku, BGP se koristi slično kao kod rutiranja između AS, s tim što umesto AS postoje individualni serveri (realni ili virtuelni serveri i virtuelni privatni oblaci — VPC).

Rutiranje tokova

Tok (flow) predstavlja logičku celinu podataka koja se razmenjuje između dva kraja komunikacije. Pogledajmo primer video-striminga. Niz paketa koji sadrže slike šalju se sukcesivno između računara i servera. Taj niz paketa ne mora biti ravnomeran. Ukoliko kapacitet mrežne veze varira, može se smanjiti rezolucija videa ili *bitrate* i slati manja ili ređa količina podataka. Ovakvi paketi, iako pripadaju istom toku, ne moraju se kretati ravnomerno kroz mrežu. Sa druge strane, u jednom toku mogu se identifikovati grupice paketa koje se kreću sukcesivno, jedna za drugom, uz malo kašnjenje. Ovakvi delovi toka nazivaju se *flowlet*-i.

Kada se saobraćaj u infrastrukturi Fat Tree šalje višim slojevima, postoje dva suprotstavljena zahteva. Prvi je da se ne šalje sav saobraćaj istim linkom, već da se naizmenično koriste različiti linkovi, kako ne bi došlo do preopterećenja. Sa druge strane, izmena putanje usred jednog flowlet-a može dovesti do „rastezanja“ flowlet-a, jer jedan paket se kreće jednom putanjom, a drugi drugom i to može dovesti do posledica po aplikaciju. Dakle, ideja je da se rasterete linkovi, ali ne po cenu „cepanja“ flowlet-a. Naravno, poseban je izazov identifikovati *flowlet*-e, jer oni variraju od aplikacije do aplikacije.

Jedan od protokola kojim se definiše korišćenje pojedinačnih putanja jeste ECMP — Equal Cost Multi-Path. Ovaj protokol balansira više putanja tako što distribuira pakete ravnomerno na različite putanje. ECMP se primenjuje na agregacionom sloju i na core-sloju. Prvi pristup jeste da ECMP šalje pakete naizmenično, jedan na prvi, a naredni na drugi link. Ovakav pristup je najravnomerniji u teoriji. Međutim, na taj način se mogu lako razdvojiti paketi jednog toka (flow-a), tako da se upute različitim putanjama i to dalje može dovesti do toga da takvi paketi završe u različitim redovima (queue) uređaja,

što opet dalje može izazvati pristizanje na odredište promenjenim redosledom. Epilog je da aplikacija može u nekom trenutku da odbaci pakete i da nastane zagušenje i pad brzine prenosa. Dakle, u data-centru oblaka je veoma značajno da paketi koji pripadaju istom flow-u ili, još specifičnije, paketi koji pripadaju grupici paketa istog flow-a (pomenuti flowlet-i) budu upućeni istom putanjom. Uzrok je u tome što je dužina linkova unutar data-centra relativno mala i kašnjenje usled prostiranja može biti za red veličine manje od kašnjenja usled čekanja u redu.

Kod konvencionalnog ECMP-a usmeravanje se reguliše na nivou toka (flow-a) tako što se svi paketi sa istim IP adresama i portovima izvorišta i odredišta i istim transportnim protokolom smatraju delovima istog toka i usmeravaju na isti izlazni port rutera. Ne upoređuju se sve ove vrednosti posebno, već se od njih kreira brojčana heš vrednost (koja je uvek ista, ako su ove vrednosti iste) i onda se port određuje kao ostatak pri celobrojnom deljenju sa brojem portova rutera. Na primer, ako su adrese konekcije 192.168.1.10 i 10.0.0.20, i transportni portovi 12345 i 443 TCP-a (oznaka 6 se koristi za TCP), onda se sve ove brojčane vrednosti predstave kao binarne, spoje i izračuna heš od njih. Takav broj se dekadno podeli sa brojem portova rutera (neka ruter ima 4 porta). U ovom slučaju heš funkcijom se dobija broj 2768209332. Podeljen sa 4 daje ostatak 0, znači da se ovakav paket upućuje na port 0 (to je prvi port rutera; portovi se označavaju od 0 do 3).

Unapređenje se dobija kada se nizovi paketa istog toka, između kojih postoji značajnije kašnjenje — flowleti, tretiraju kao posebne celine. Mehanizam je sličan kao kod usmeravanja tokova, samo što se naredna grupa paketa (flowlet) tretira kao posebna jedinica i usmerava na drugi port (iako je heš isti).

VL2 infrastuktura

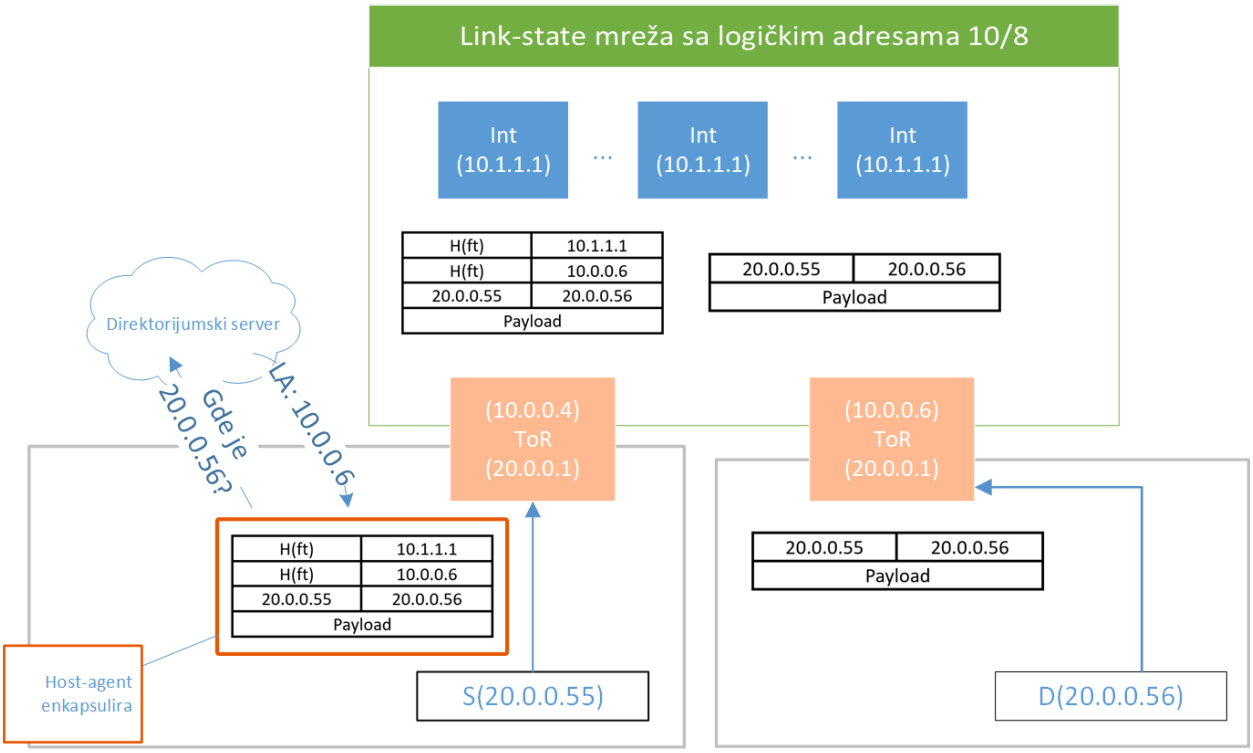
VL2 je preteča Azure, Microsoft-ove mrežne infrastrukture oblaka. VL2 je koncipiran imajući u vidu nedostatke postojećih rešenja. A postojeća rešenja u tom trenutku (2009. godine) bila su zasnovana na Fat Tree mreži, sa kruto povezanom logičkom i fizičkom organizacijom mreže, nedovoljnom otpornošću na greške i nefleksibilnim servisima. Virtualni LAN-ovi su i dalje ključni za organizaciju mreže, a performanse mreže variraju u odnosu na poziciju servera unutar mreže.

Ustanovljeno je da je kombinacija većeg broja jednostavnijih i manjih svičeva bolja nego kombinacija većih svičeva, jer korišćenjem većih svičeva (sa više portova), ako dođe do kvara, često bude zahvaćen čitav svič, dovodeći do otkaza većeg broja servera.

Virtuelizacija računarskih sistema dovodi do toga da se virtuelni serveri iste logične vrituelne mreže mogu naći fizički bilo gde u data-centru. Potrebno je razdvojiti adresne prostore lokalnih adresa (LA), koje pripadaju fizičkoj mreži i aplikativnih adresa (AA), koje su vezane za virtuelnu mašinu i servis koji pruža. I ako se virtuelna mašina instancira na drugoj lokalnoj adresi, njena AA mora ostati ista. Tu na scenu stupa direktorijum adresa, koji vrši povezivanje LA i AA. Naravno, ako dođe do promene, podaci u ovom direktorijumu se moraju brzo ažurirati. Ovakvo svojstvo, da se svaki server može dodeliti svakom servisu, naziva se agilnost.

Primer — prevođenje adresa u oblaku

U nastavku je prikazan proces adresiranja aplikacije u oblaku na primeru mreže VL2 (Slika 99).



Slika 99 VL2 arhitektura mreže

Koriste se dve varijante IP adresa: mrežna infrastruktura funkcioniše putem lokacijski zasnovanih adresa (LA), svi svičevi imaju zadate ove adrese i koristi se link-state algoritam kojim se oglašavaju samo ove adrese. U ovom slučaju, u pitanju su adrese 10.X.X.X. Aplikacije koriste aplikacijske adrese (AA), koje se ne menjaju, bez obzira na migraciju servera ili podizanje instanci na drugom serveru. Vezu između LA i AA održava direktorijumski servis. Čim se instancira AA adresa, kreira se mapiranje u LA-AA direktorijumu. Fizički — virtuelni serveri sa svojim AA adresama u primeru sa slike su locirani na posebnim delovima mreže. Logički — ponašaju se kao da su na istom sviču. Kada servis treba da pošalje paket drugom servisu u istoj logičkoj mreži, šalje ARP paket (broadcast) za datom AA adresom. VL2 agent koji radi na datom hostu uzima ovaj zahtev i formira zahtev ka tabeli direktorijuma. Dobija odgovarajuću LA adresu ToR sviča koji odgovara odredištu. Agent kešira ovaj podatak, da bi ga brže koristio ako mu zatreba kasnije. U ovom slučaju direktorijumski servis dostavlja mapiranje 20.0.0.56 u 10.0.0.6.

Agent sada „pakuje“ podatke u nekoliko slojeva. Prvi sloj se adresira na logičko odredište (20.0.0.56), zatim se ovakav paket dalje „pakuje“ u zaglavlje koje sadrži adresu odredišnog ToR sviča — 10.0.06 i tom zaglavlju odgovara heš (H(ft)) kojim se jednoznačno određuje tok (heširaju se IP adrese, protokol i port). Na kraju, ovakav paket se „pakuje“ u zaglavlje koje sadrži adresu agregacionog sviča — 10.1.1.1 i odgovarajući heš. Agregacioni svičevi svi imaju iste IP adrese, čime se pojednostavljuje organizacija. Kada se paket usmerava na 10.1.1.1, a postoji više svičeva sa tom adresom, takvo slanje se naziva anycast. Određuje se jedan od svičeva (na osnovu toka, koji je opet određen hešom, tako da paketi istog toka idu preko istog rutera, kako je to opisano u sekciji protokola ECMP). Kada agregacioni svič primi paket, odstranjuje deo zaglavlja, čita adresu 10.0.0.6 i usmerava paket ka ToR sviču odredišne mreže. Ovaj svič je neposredno povezan na server 20.0.0.56 i dostavlja paket odredišnom virtuelnom serveru.

CDN — Content Delivery Network

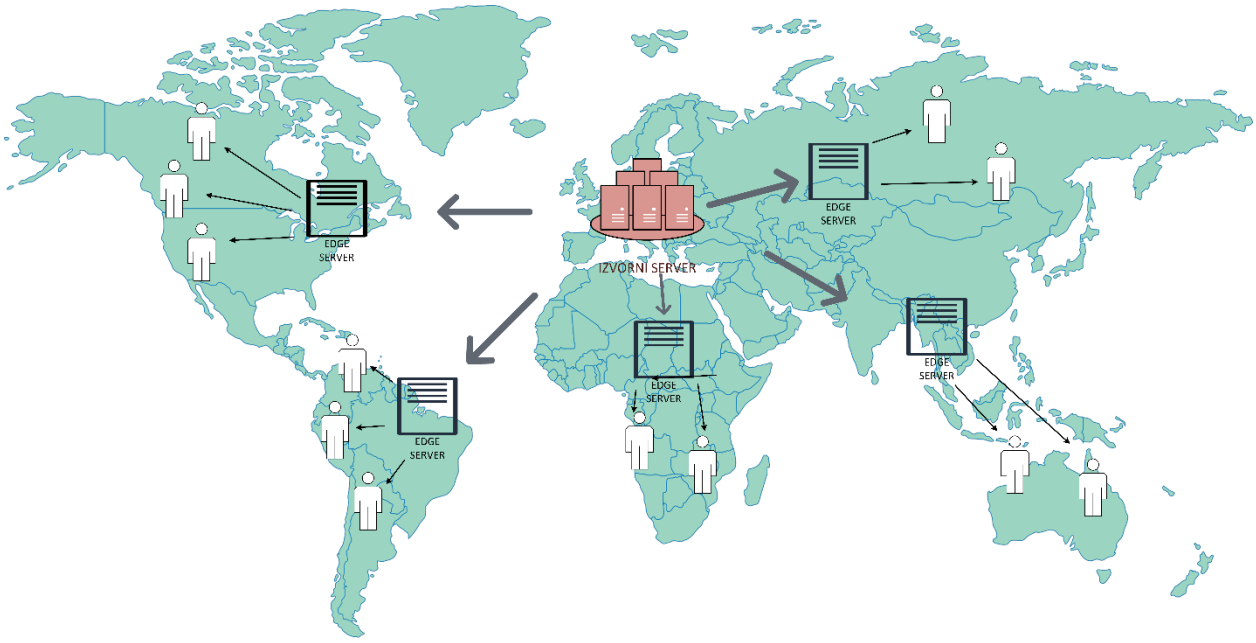
U prethodnim sekcijama dat je prikaz funkcionisanja unutrašnje mreže data-centra. Moglo se videti iz primera da je saobraćaj unutar data-centra veoma intenzivan i da je jedan od ciljeva smanjiti kašnjenja optimizacijom saobraćaja i držanjem paketa koji pripadaju istoj logičkoj grupi u jednoj putanji.

Kada klijent pošalje zahtev, jedan deo kašnjenja odnosi se na saobraćaj i obradu unutar data-centra. Drugi deo kašnjenja vezan je za putovanje paketa do i od data-centra. Ukoliko je klijent fizički udaljen od data-centra, povećava se dužina puta koji paket treba da pređe, kao i broj rutera koji posreduju u saobraćaju. Klijent koji se nalazi u Tokiju može očekivati značajnu razliku u odzivu kada komunicira sa data-centrom u Beogradu i kada komunicira sa data-centrom u Seulu.

CDN (Content Delivery Network) je mreža distribuiranih servera koji sadrže replike sadržaja i služe za isporuku veb sadržaja prema svakom zahtevu sa odgovarajućeg serverskog čvora, na osnovu geografske lokacije odakle je zahtev potekao i lokacije serverskog čvora sa sadržajem. Ova usluga je posebno efikasna u ubrzavanju isporuke veb sadržaja tokom visokog saobraćaja u mreži.

Alternativno, može se reći da je CDN sistem međusobno povezanih servera koji brzo isporučuju sadržaj korisnicima širom sveta putem interneta. Brza isporuka sadržaja postiže se dupliranjem sadržaja na više servera koji se nalaze na različitim lokacijama, a zatim se sadržaj usmerava korisnicima sa tih servera kada je potreban.

Pravi ili originalni izvor bilo kog sadržaja poznat je kao izvor sadržaja ili originalni server (origin server). CDN nudi brzu i pouzdanu isporuku sadržaja preko kanala sa malom propusnošću tako što kešira i replicira sadržaj na više serverskih čvorova koji su strateški raspoređeni širom svet (Slika 100). Ovi dodatni serverski čvorovi, koji se nalaze na različitim geografskim lokacijama, poznati su kao edge serveri.



Slika 100 CDN

CDN, nakon što primi sadržaj sa izvornog (origin) servera, replicira ga na edge keš servere. Svaki zahtev za sadržajem automatski se preusmerava na najbliži server (u mrežnom vremenu, a ne u geografskim kilometrima). Drugim rečima, posetilac sajta iz Njujorka može preuzeti isti sadržaj sa servera u Njujorku, dok posetilac iz Singapura dobija sadržaj sa servera u Singapuru.

Postoje dve politike upravljanja sadržajem: potpuna replikacija sajta i delimična replikacija sajta. Potpuna replikacija sajta se primenjuje kada je sajt statičan. Tada se sav sadržaj sajta replicira na jedan ili više odgovarajućih edge-servera sa origin-servera. Strategija delimične replikacije sajta se primenjuje na sajtove u kojima je ugrađen nestatičan sadržaj. Tada se statički delovi veb stranica repliciraju sa origin servera na edge-servere, dok nestatičan sadržaj biva isporučen na osnovu strategije upravljanja sadržajem.

U tehnici keširanja, politike ažuriranja keša, zajedno sa održavanjem keša, od velike su važnosti za upravljanje sadržajem preko CDN-a. Potrebno je odgovarajuće planiranje i implementacija kako bi se održala doslednost i integritet sadržaja na replikama postavljenim na različitim ivičnim serverima. Postoje različite politike ažuriranja sadržaja koje se primenjuju na replikovani sadržaj, poput ažuriranja pokrenutih promenama sadržaja, ažuriranja na zahtev i periodičnih ažuriranja. Dizajneri CDN sistema koriste neke od ovih politika ažuriranja (ili kombinaciju više njih) u svom sistemu.

Kroz rutiranje zahteva, korisnički zahtevi se usmeravaju na optimizovano najbliži edge-server preko CDN-a koji može najbolje da usluži zahtev. Ovde se udaljenost ne meri u kilometrima. Odluka kombinuje više parametara, poput vrste zahtevanog resursa, lokacije korisnika, trenutnog stanja mreže i opterećenja edge-servera.

U početnim danima CDN-a, isporuka sadržaja se zasnivala isključivo na pull tehnologiji. Kada god bi neki korisnik zatražio novi sadržaj, taj sadržaj bi se „povukao“ (pull) sa origin servera na neki server sadržaja blizu lokacije korisnika. U slučaju već keširanog sadržaja, softver bi proverio da li je to najnovija verzija, a u suprotnom bi najnoviji sadržaj bio dobavljen sa origin servera. Time bi se ubrzala isporuka sadržaja tom korisniku ili korisnicima iz okolnih geografskih lokacija pri kasnijim zahtevima.

Kod sadržaja koji se često menja, pull-mehanizam nije praktičan, jer iziskuje stalne provere kod origin-servera. Da bi se rešio ovaj problem, koristi se push-mehanizam. Uz HTTP push, sadržaj se automatski distribuira ili „gura" na povezane edge-servere kada god bude dodat na origin server ili kada god bude promenjen. Ova tehnika je efikasnija za veće fajlove, jer takvi fajlovi zahtevaju više vremena za isporuku. Ako lokalna kopija ili najnovija verzija tih fajlova nisu sačuvane u CDN-u, korisnicima će biti potrebno više vremena da ih preuzmu sa origin servera.

U pull mehanizmu, najnovija verzija sadržaja se ne prenosi automatski na ivične servere. To se dešava tek kada neki korisnik podnese zahtev za sadržajem. Tek tada, ažurirana verzija sadržaja se skladišti na ivičnom serveru tokom isporuke tog sadržaja.

Ovaj mehanizam se koristi u keširanju veb sadržaja tokom internetskog pregleda. Kada korisnik pregleda neki interesantan sadržaj, browser povlači sadržaj sa servera i kešira ga koristeći privremeno skladište internetskih fajlova. Ovi sačuvani fajlovi sadrže informacije o prethodno posećenim veb stranicama i fajlovima, kao što su grafike prikazane na stranici. Ovaj mehanizam ubrzava učitavanje stranica u kasnijem vremenu. Međutim, veb keširanje je efikasno samo kada se sadržaj dokumenta ne menja. Za dokumente koji se često menjaju, ova tehnika postaje manje efikasna. U takvim slučajevima, klijent mora svaki put proveriti izvorni server sadržaja kada korisnik pokuša da pristupi sadržaju i ponovo povuče ažurirane delove.

Da bi se rešio ovaj problem, koristi se *push* mehanizam. Uz HTTP push, sadržaj se automatski distribuira ili „gura" na povezane ivične servere kada god bude dodat na origin server ili kada god bude ažuriran ili promenjen. Ova tehnika je efikasnija za veće fajlove, jer takvi fajlovi zahtevaju više vremena za isporuku. Ako lokalna kopija ili najnovija verzija tih fajlova nisu sačuvane u CDN-u, korisnicima će biti potrebno više vremena da ih preuzmu sa origin servera.

Pitanja za proveru i obnavljanje znanja

1. U čemu se razlikuju MAC i IP adrese?
2. Na koji način NAT-ruter „zna“ šta da radi sa paketom koji dolazi spolja?
3. Osnovni zadatak DNS-a je da poveže korišćenu simboličku adresu sa IP adresom. Koje još funkcije ima DNS?
4. Kako može biti organizovana *underlay* mreža u data-centru oblaka?
5. Šta se sve dobija korišćenjem Fat Tree mreže?
6. U čemu je razlika između VLAN-a i VXLAN-a?
7. Kako ECMP rutira tokove u data-centru?
8. Na kom principu funkcioniše CDN i šta se njime ostvaruje?

7.4. Umrežavanje i AWS

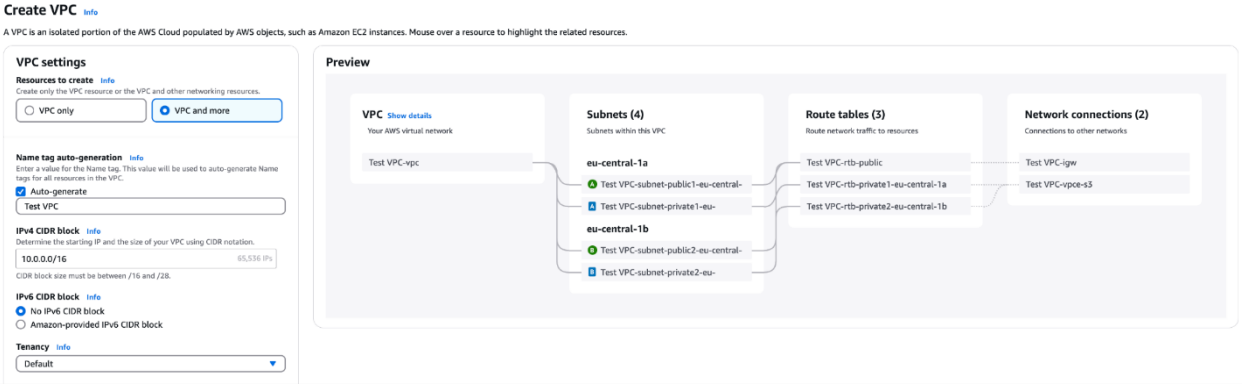
Radi sigurne komunikacije i povezivanja resursa u oblaku, AWS nudi nekoliko servisa i komponenti za upravljanje VPC (Virtual Private Cloud). Razumevanje ovih servisa je ključno za građenje skalabilnih, sigurnih, visoko dostupnih i otpornih aplikacija u oblaku.

U okviru ove sekcije se pokriva pregled ključnih AWS servisa koji čine mrežnu arhitekturu u oblaku. Počev od VPC, preko direktnih sigurnih veza sa infrastrukturom u prostoru organizacije, DNS servisa, kontrole pristupa, kao i globalnu isporuku sadržaja. Neophodno je sve komponente razumeti i imati jasnu sliku o mrežnoj infrastrukturi kako bi se isplanirali i postavili resursi koji će moći bezbedno i pozudano da komuniciraju međusobno.

AWS VPC

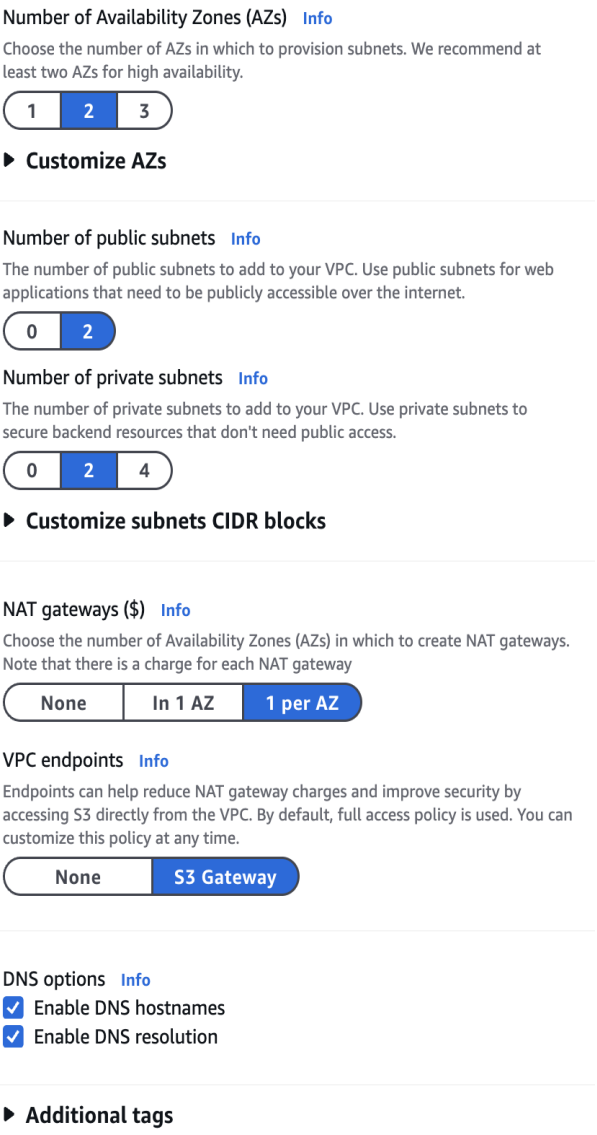
AWS VPC je izuzetno kompleksan servis koji u svom domenu ima jako puno podservisa koji mu pomažu da funkcioniše i rutira saobraćaj. Kako je dosta korisnika imalo problema sa postavkama svog VPC, AWS je napravio pristupačan način kreiranju, kao i preporučena podešavanja za produkcijska okruženja, olakšavajući pravljenje i rutiranje. VPC je komponenta koja funkcioniše na nivou jednog regiona, u okviru jednog ili više zona dostupnosti (preporučeno je minimum 2 za produkcijska okruženja). Postoji podrazumevani VPC koji AWS napravi uz svaki nalog, međutim, uglavnom se prave posebni VPC, a podrazumevani se ne dira.

VPC se formira tako što se pristupi VPC servisu u oviru AWS konzole. Na vrhu stranice postoji dugme Create VPC i potrebno je kliknuti na njega. Ostaviti izabranu opciju VPC and more i pogledati koje sve resurse će AWS napraviti (Slika 101).



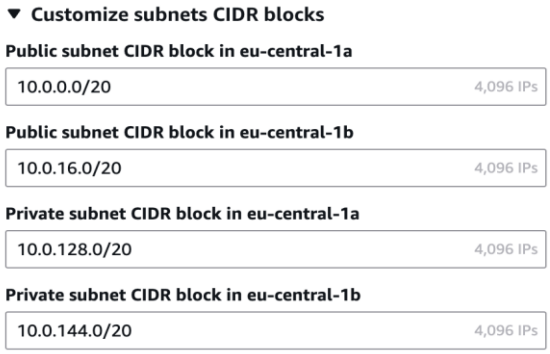
Slika 101 ‘Create VPC’ opcija sa potrebnim mrežnim resursima

Opcija „VPC and more“ (Slika 102) je veoma pogodna jer će napraviti i dodatne komponente koje su potrebne i optimalne da bi VPC pravilno funkcionisao. Na osnovu imena Test VPC planirani su resursi za pravljenje koji ukazuju na to da je VPC podešen na 2 zone dostupnosti, samim tim i 4 podmreže (subnets), od čega 2 javne i 2 privatne, kao i tabele rutiranja unutar VPC-a, jedna za public i dve tabele za private. Takođe vidimo da se planira pravljenje igw (internet gateway) kako bi VPC imao izlaz na internet i jedan endpoint za pristup S3 (privatni endpoint sa internim rutiranjem unutar AWS mreže).



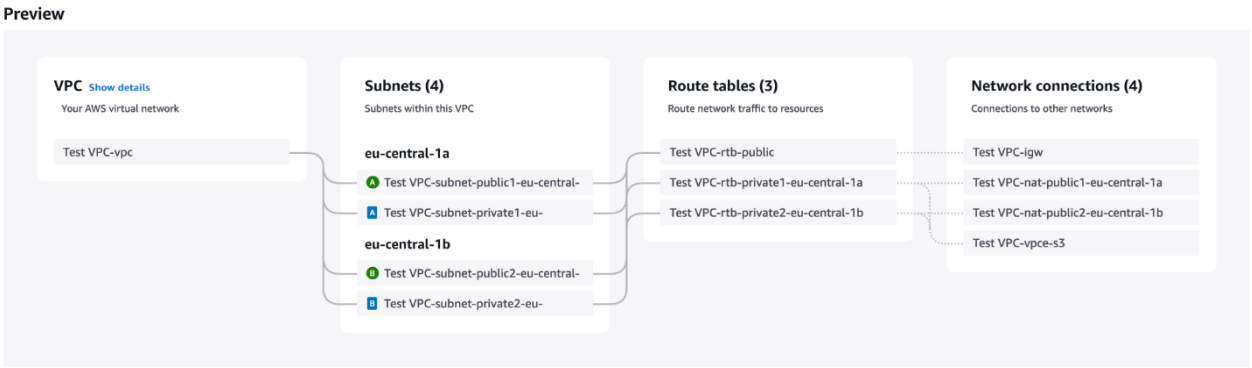
Slika 102 Dodatna podešavanja VPC-a

Za produkciona okruženja minimalna podešavanju su kao sa slike. Potrebno je imati bar 2 zone dostupnosti kako bi infrastruktura opstala u slučaju otkazivanja jedne zone; takođe, ide se na minimum 2 javne i 2 privatne podmreže, kao i 1 NAT gateway po zoni dostupnosti. Za ostala okruženja, kao što su dev i testna, moguće je smanjiti ove resurse kako bi se uštedelo; na primer, samo jedan NAT gateway u jednoj zoni dostupnosti i tabela rutiranja koja bi sve privatne podmreže rutirala na taj NAT. Takođe, moguće je i dodatno uraditi podešavanje podmreža i njihovih opsega opcijom Customize subnets and CIDR blocks (Slika 103).



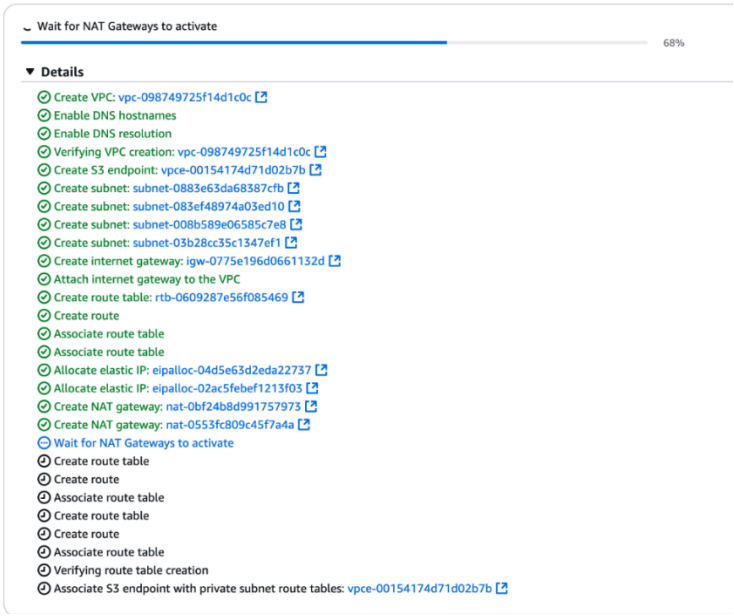
Slika 103 Podešavanje podmreža

S obzirom na to da se podmreže ne mogu izmeniti nakon pravljenja VPC-a, predlaže se da se naprave veće podmreže kako se kasnije ne bi došlo u situaciju sa nedostatkom IP adresa koja može biti jako izazovna za rešavanje. Nakon ovih podešavanja, na početak se može vratiti i pregled VPC-a kako bi se utvrdili svi resurse i rutiranja koja će AWS napraviti (Slika 104).



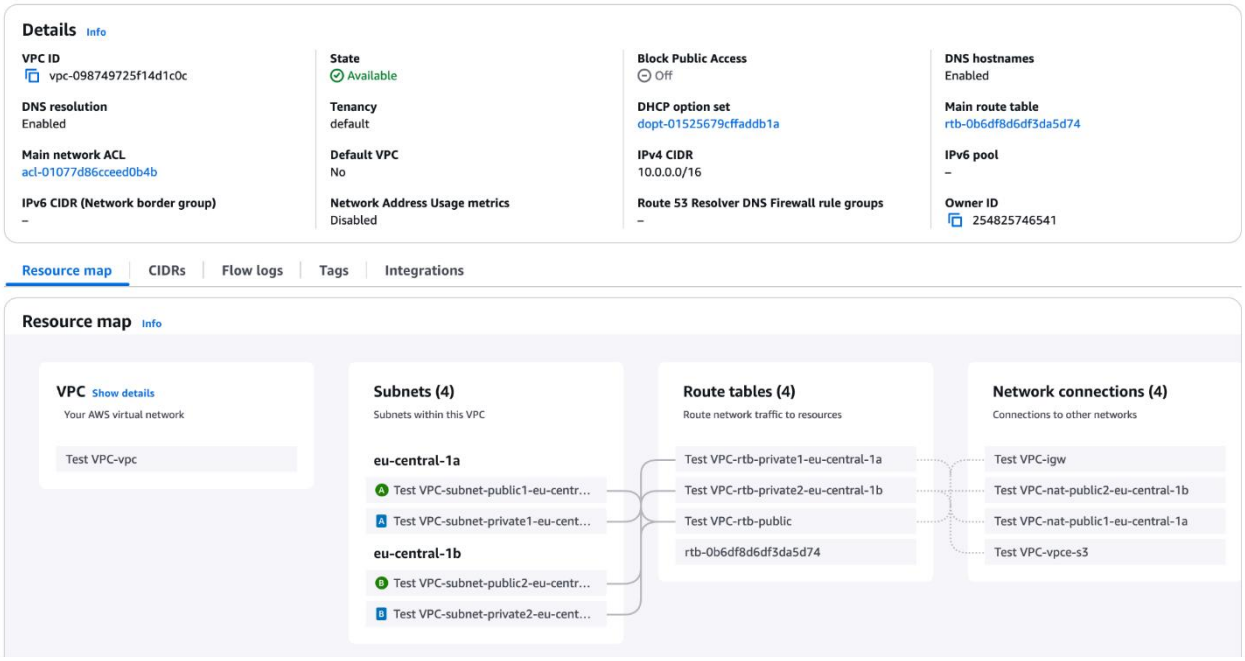
Slika 104 Pregled resursa koji će se napraviti prema unetoj konfiguraciji

Nakon ovih podešavanja možemo izabrati opciju Create VPC i sačekati dok se svi resursi ne naprave (Slika 105).



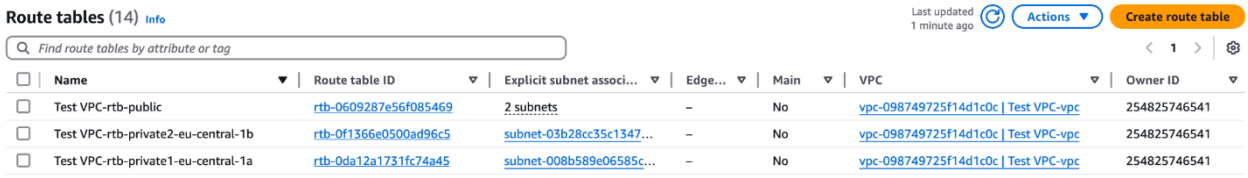
Slika 105 Pravljenje VPC resursa

AWS će na osnovu konfiguracije napraviti sve potrebne resurse i prikazati njihove identifikatore. AWS NAT Gateway može potrajati pri pravljenju, pa samim tim ne treba prekidati proces, već sačekati da se resursi naprave i aktiviraju. Po uspešnom pravljenju VPC-a možemo ići na opciju View VPC i tu pogledati sve resurse, kao na slici Slika 106.



Slika 106 Kreiran VPC i prikaz njegovih resursa

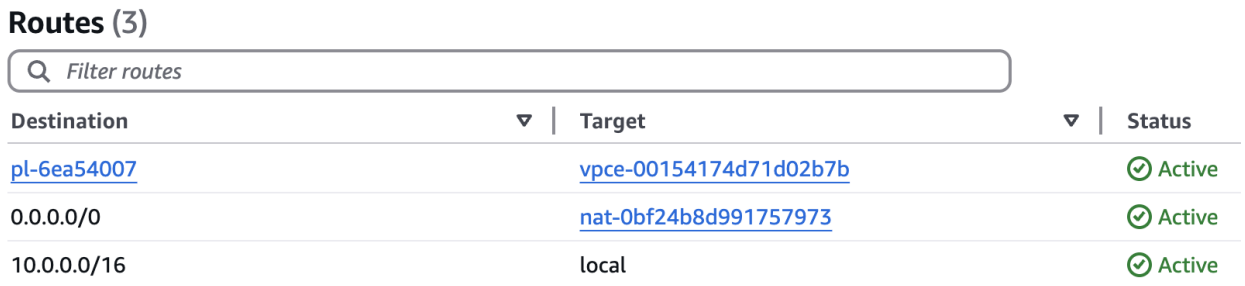
Pregledom tabela rutiranja možemo malo jasnije razumeti kako se kreće saobraćaj u okviru samog VPC-a i njegovih podmreža (subneta) (Slika 107).



Slika 107 Prikaz tabela rutiranja (route table)

Pregledom tabela rutiranja utvrđujemo sledeće:

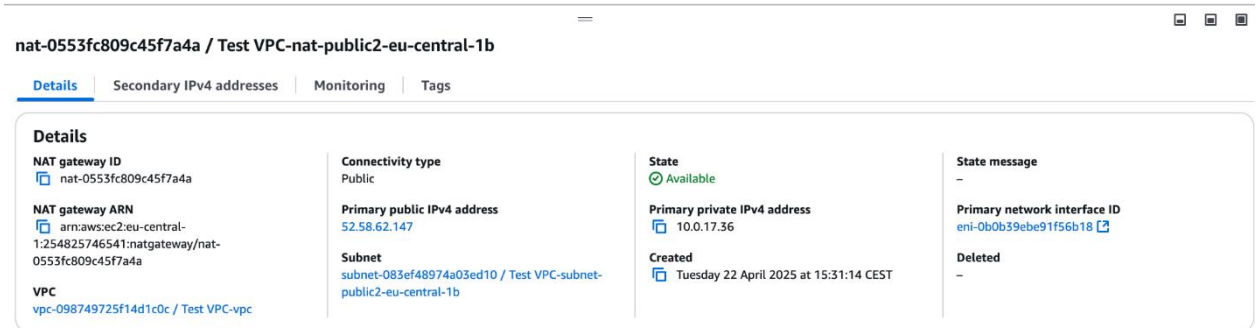
- Test VPC-rtb-public — Ova tabela rutira javni saobraćaj ka 0.0.0.0/0 adresi pomoću resursa sa oznakom igw, što ukazuje na internet gateway. Takođe ima pravilo 10.0.0.0/16, što ukazuje da rutira sav interni saobraćaj u okviru VPC-a. Povezan je sa dva subneta — 10.0.0.0/20 i 10.0.16.0/20, što ukazuje na javne subnete i znači da rutira saobraćaj za ta dva subneta.
- Test VPC-rtb-private1/2-eu-central-1a/b — Dve tabele rutiranja koje služe da rutiraju saobraćaj privatnih subneta (10.0.144.0/20 i 10.0.128.0/20) ka internoj VPC mreži (10.0.0.0/16), kao i prema odgovarajućem NAT gatewayu (0.0.0.0/0 saobraćaj ide kroz NAT) i VPC endpointu za S3.



Slika 108 Primer tabele rutiranja privatnog subneta

Ono što definiše da li je subnet privatan ili ne nije njegovo ime, niti išta slično tome, već način na koji se vrši rutiranje saobraćaja, što se može lako uveriti pregledom kako se rutira odlazni saobraćaj ka javnom internetu (0.0.0.0/0); da li je to kroz internet gateway (igw) — public subnet, ili kroz NAT gateway — private subnet (Slika 108).

Internet gateway i **NAT gateway** su upravljane komponente od strane AWS i oko njih nema nikakvog posebnog održavanja niti upravljanja. NAT gateway (Slika 109) ima neke dodatne parametre, propusni opseg, skaliranje itd. Pogodan je i za korišćenje u slučajevima gde je potrebna statička IP adresa kao izlazna iz VPC-a jer sve komponente iza NAT gateway ka javnoj mreži imaju istu IP adresu — tu koju ima NAT gateway.



Slika 109 Prikaz NAT gateway

Podmreže (subnets), njihovi opsezi, kao i broj preostalih IP adresa mogu se pogledati u subnets meniju. Kod VPC endpointa, kao što je endpoint za S3 koji je napravljen pri kreiranju VPC-a, postoje dva načina pristupa — jedan je definisan kroz *route tables,* gde je rutiranje saobraćaja iz određenih subneta ka tim privatnim endpointima samo moguće, dok je drugi način kroz polise, što znači da samo određeni resursi mogu da koriste taj endpoint. Podrazumevana polisa dozvoljava svim resursima da ga koriste, dok postoji mogućnost ograničavanja pristupa na određene S3 ili određenim IAM roles (slike Slika 110Slika 111).

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow-access-to-specific-IAM-role",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "ArnEquals": {
          "aws:PrincipalArn": "arn:aws:iam::111122223333:role/role_name"
        }
      }
    }
  ]
}
```

Slika 110 AWS endpoint polisa za dozvolu pristupa samo određenoj IAM role

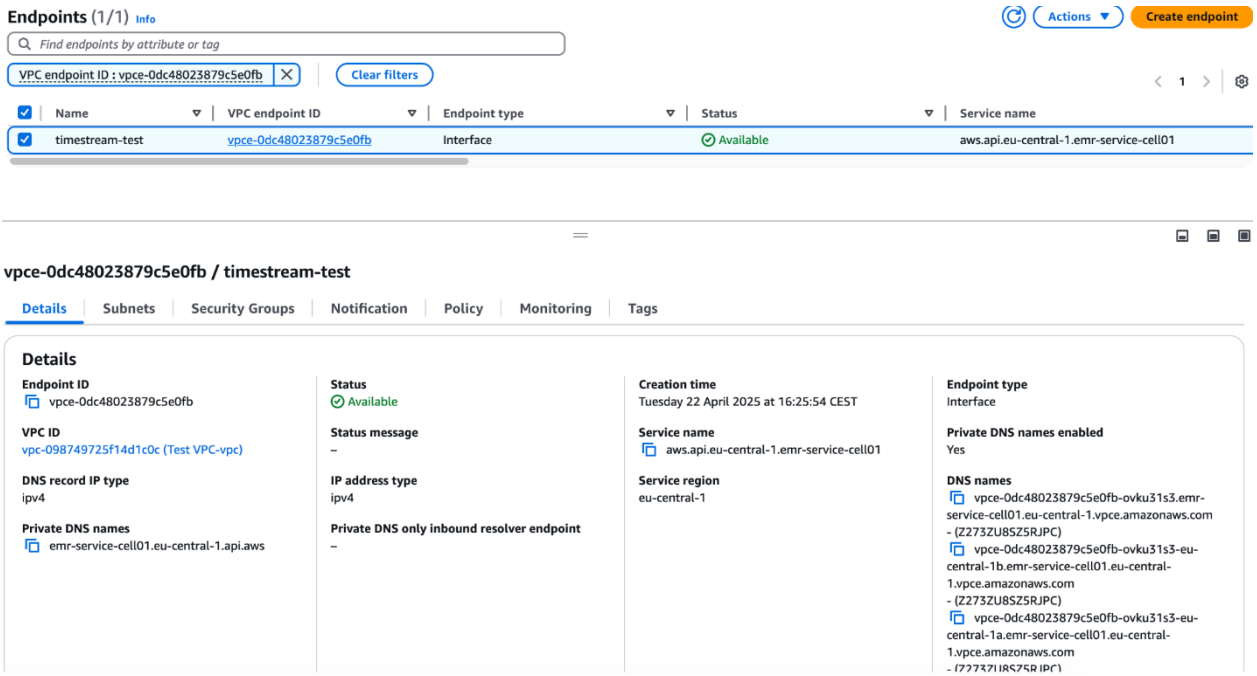
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow-access-to-specific-bucket",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
         "s3:ListBucket",
         "s3:GetObject",
         "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::bucket_name",
        "arn:aws:s3:::bucket_name/*"
      ]
    }
  ]
}
```

Slika 111 AWS endpoint polisa za određene akcije nad određenim S3

AWS ima dva tipa **VPC endpointa** za AWS servise:

- **Interface** — Služi za većinu AWS servisa, mora da napravi network interfejs u okviru VPC, zauzima IP adresu i naplaćuje se po satu i količini saobraćaja, ka njemu se može rutirati pomoću IP adrese ili DNS vrednosti.
- **Gateway** — Važi samo za S3 i DynamoDB, rutira saobraćaj pomoću tabela rutiranja, automatski skalira i nema dodatnih troškova.

Primer *interface* endpointa je prikazan na slici Slika 112.



Slika 112 Interface VPC endpoint za servis AWS Timestream

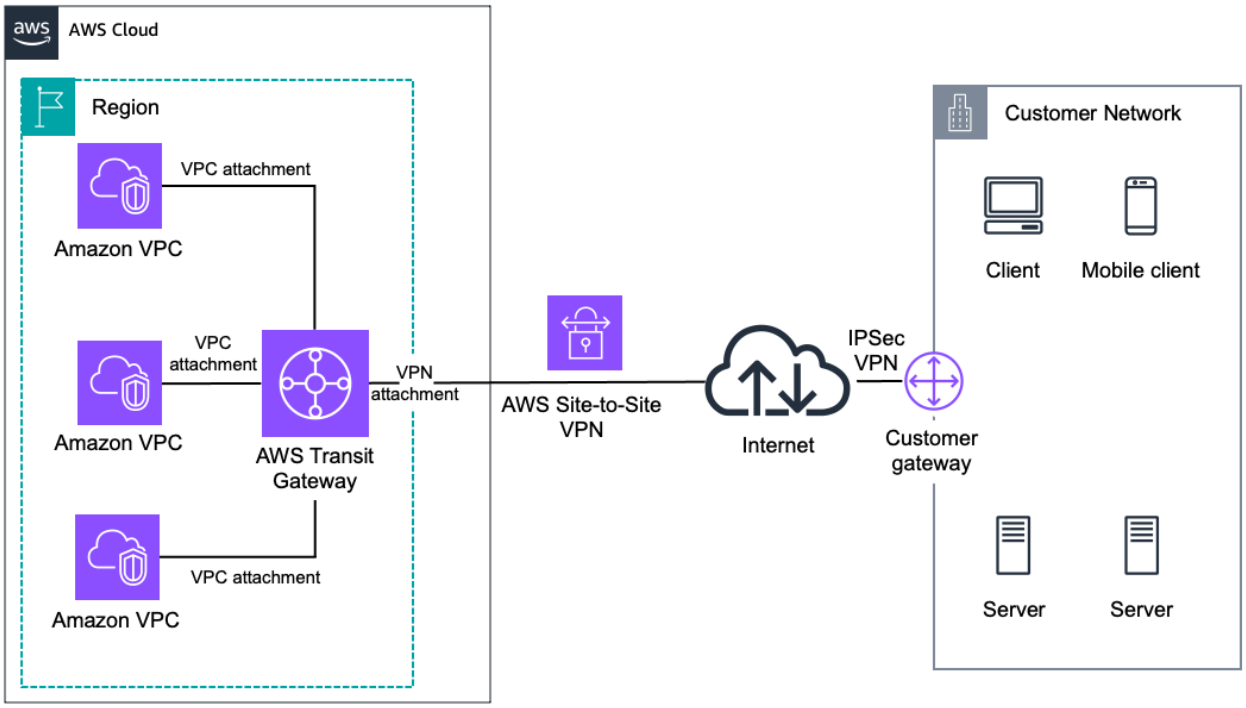
Sa slike se može zaključiti da je Interface endpoint za AWS Timestream servis, da ima dodeljenu DNS vrednost i koja je ta vrednost kako bi se koristila u aplikacionom kodu. Poseduje povezanost sa subnetima i ima svoju security grupu, kao i polisu.

AWS VPN je servis koji služi za povezivanje on-premise mreža ka AWS VPC-u. Koristi IPsec tunele preko interneta, dobar je za bezbednu, enkriptovanu i point to point ili site to site konekciju.

AWS VPC Peering služi za direktno povezivanje dva VPC-a, koristi privatne IP adrese i konekciju niske latencije, nema mogućnosti naprednog rutiranja, ali je dobar za direktno konekciju dva VPC-a.

AWS Transit Gateway služi kao centralni hub za povezivanje više VPC-a i on-premise mreža, podržava transitive rutiranje, veoma je skalabilan i efikasan. Idealan je za velike mrežne arhitekture gde ima više VPC-a i on-premise mreža.

Na slici Slika 113 data je arhitektura koja kombinuje navedene servise.



Slika 113 Primer kombinacije AWS Transit Gateway i AWS VPN konekcija i on-prem infrastrukture

Za kontrolu mrežnog saobraćaja u okviru VPC-a AWS ima dva servisa koji pomažu:

- **NACL** — Network Access Control List
- **Security Groups**

NACL su liste koje se dodaju na svaki subnet (podmrežu) u okviru AWS VPC-a. Kada saobraćaj ulazi i napušta subnet, pravila u okviru NACL se proveravaju i određuje se da li je taj saobraćaj dozvoljen ili blokiran. NACL je stateless, pa je samim tim potrebno saobraćaj eksplicitno dozvoliti. NACL se ponaša kao vrsta firewalla na nivou subneta; kontroliše sav dolazi i odlazni saobraćaj tako što prolazi kroz listu pravila definisanih u okviru sebe, od najnižeg ka najvišem, podržava allow i deny pravila.

Security Groups (SG) je stateful firewall-i koji rade na nivou instance. Kontrolišu dolazni i odlazni saobraćaj i podržavaju samo allow pravila, kao i automatsko dozvoljavanje povratnog saobraćaja. Automatski se vezuju za mrežne interfejse EC2 instanci, lambdi i ostalih AWS servisa koji ih imaju.

Kada uporedimo **NACL** i **Security groups,** postaje jasno u čemu je ključna razlika:

- SG (bezbednosne grupe) su stateful, dok su Network ACLs (NACL) stateless. To znači da ako se omogući dolazni saobraćaj kroz SG, povratni saobraćaj je automatski dozvoljen; kod NACL-a to nije slučaj.
- SG-ovi funkcionišu na nivou instance (EC2), dok se NACL-ovi primenjuju na nivou subnet-a.
- SG-ovi dozvoljavaju saobraćaj, ali ne podržavaju eksplicitno odbijanje. NACL-ovi mogu da dozvole ili eksplicitno odbiju saobraćaj po pravilima.
- SG-ovi su lakši za upravljanje kada je potrebna detaljna kontrola pristupa za pojedinačne instance.

Koristiti Security Groups kada je potrebna kontrola pristupa na nivou instanci uz stateful ponašanje, dok su NACL-ovi pogodniji za širu kontrolu na nivou subnet-a, posebno kada je potrebno eksplicitno odbijanje saobraćaja.

AWS Route 53

DNS (Domain Name System) je globalni, distribuirani sistem nalik imeniku, koji prevodi tekstualne adrese u IP adrese. Prevodilac funkcioniše tako što ima hijerarhijsku strukturu — root — **TLD** (Top Level Domain), nadležni (autoritativni) serveri za konkretne zone i keširanje odgovora uz **TTL** (Time To Live) vrednosti. Što je TTL kraći, brže se mogu uvesti promene u DNS vrednostima i brže se detektuje prekid, ali isto tako se povećava broj DNS upita i samim tim troškovi vezani za DNS.

Route 53 je Amazonov servis koji ima ulogu DNS servera za domene kojim upravlja, uz mogućnost kupovine domena. Potrebno je razumeti da Route 53 ne obavlja razrešavanje za krajnje korisnike - taj posao pripada resolver-ima korisnikovog provajdera (ISP) - već daje konačan odgovor samo za imena kojima upravlja. Odgovori koje daje mogu biti jednostavni (jedna IP adresa) ili pametni tako što zavise od zdravlja servisa, geografske lokacije korisnike, latencije itd. Upravo u toj pametnoj funkcionalnosti leži razlog zašto se Route 53 često koristi u modernim, višeregionalnim i visoko dostupnim arhitekturama.

Na najnižem nivou Route 53 održava hosted zone koji sadrži DNS zapise i metapodatke za određeni domen. Za svaku zonu se dobija skup NS (Name Server) zapisa koji se delegiraju kod registra domena kako bi globalni DNS znao da odgovore na određeni domen potraži kod Route 53 autoritativnih servera. Routing politika se primenjuje u trenutku kada se pristupi serveru i vraća se odgovor. Ako je uključen health check, Route 53 će izbeći slanje korisnika na nezdrave adrese.

Postoje dva tipa hosted zona:

- **Javne hosted zone** — Vidljive su celom internetu i koriste se za javne sajtove i API. Na primer, ftn.kg.ac.rs
- **Privatne hosted zone** — Vidljive su isključivo interno, unutar jednog ili više VPC-a, koriste se za mikroservise, interne service, baze podataka i druge usluge koje ne izlaze na internet. Na primer, internal.ftn.kg.ac.rs

Tipovi DNS unosa koje servis podržava:

- **A** — Mapira ime hosta na IPv4 adresu.
- **AAAA** — Mapira ime hosta na IPv6 adresu.
- **CAA** — Navodi koje sertifikacione vlasti smeju da izdaju TLS sertifikate za domen (npr. issue, iodef).
- **CNAME** — Pravi alias jednog imena ka drugom imenu i ne može stajati na apex-u zone.
- **DS** — Zapis u nadređenoj zoni koji sadrži otisak KSK ključa potčinjene zone i gradi DNSSEC lanac poverenja.
- **HTTPS** — SVCB-varijanta za HTTPS koja omogućava „service binding“ i opcionalno aliasiranje uz parametre (npr. ALPN, ECH) radi efikasnijeg povezivanja.
- **MX** — Definiše mejl servere i njihov prioritet za prijem pošte za domen.
- **NAPTR** — Primenjuje regex pravila za preusmeravanje/resoluciju servisa (često uz SRV) u scenarijima poput SIP/ENUM.
- **NS** — Određuje autoritativne name servere za zonu ili delegira podzonu na druge servere.

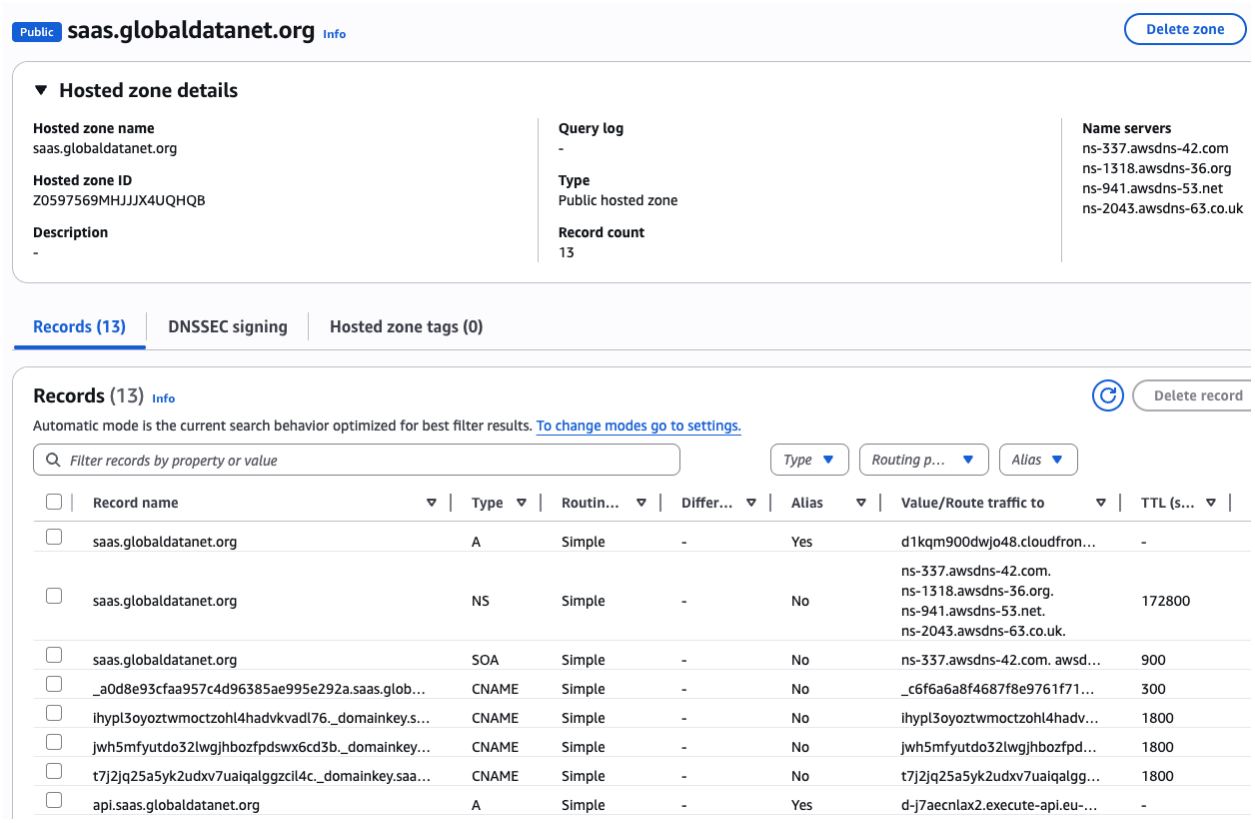
- **PTR** — Koristi se u „reverse DNS“ zonama (in-addr.arpa/ip6.arpa) da mapira IP adresu nazad na ime.
- **SOA** — „Start of Authority“ zapis sa ključnim parametrima zone (primarni NS, serijski broj, intervali osvežavanja).
- **SPF** — Istorijski tip za SPF pravila za e-poštu; danas se preporučuje objava kroz TXT, ali se i dalje sreće zbog kompatibilnosti.
- **SRV** — Navodi prioritet, težinu, port i cilj hosta za određeni servis/protokol (npr. _sip._tcp).
- **SSHFP** — Objavljuje otiske javnih SSH ključeva (algoritam i heš) za DNSSEC, validiranu proveru identiteta SSH servera.
- **SVCB** — Generički „service binding“ zapis za deklarisanje cilja i parametara servisa, osnova za moderni HTTPS tip.
- **TLSA** — Vezuje TLS sertifikat ili CA (heš) za domen preko DANE, omogućavajući verifikaciju putem DNSSEC-a.
- **TXT** — Nosi proizvoljan tekst; najčešće se koristi za verifikacije i email politike (SPF/DMARC/DKIM).

Route 53 može periodično da proverava zdravlje endpointa putem HTTP/HTTPS ili TCP provera. Na osnovu rezultata provere zdravlja može automatski uticati na DNS odgovor, na primer, može da izbaci endpoint koji je neispravan i da odgovara samo ispravnim dok se neispravan endpoint ne popravi, a zatim ga automatski vrati u listu odgovora. Još jedna mogućnost je automatsko alarmiranje pomoću CloudWatch-a (servis za logove i metrike) kako bi se incident ispratio.

Route 53 ima sledeće polise rutiranja saobraćaja:

- **Simple** — Vraća jedan ili više zapisa bez dodatne logike ili uslovnog odlučivanja.
- **Weighted** — Raspodeljuje saobraćaj između više zapisa prema unapred zadatim procentima.
- **Latency based** — Šalje korisnika na endpoint u AWS regionu sa najmanjom mrežnom latencijom prema njegovoj lokaciji.
- **Failover** — Prebacuje saobraćaj na sekundarni endpoint ako primarni postane nedostupan prema health checku.
- **Geolocation** — Bira odgovor na osnovu geografske lokacije korisnika (zemlja ili kontinent).
- **Geoproximity** — Usmerava saobraćaj ka resursima koji su geografski najbliži i omogućava podešavanje „bias“-a za preusmeravanje opterećenja.
- **Multi-Value Answer** — Vraća do osam zdravih IP adresa za isti DNS naziv radi jednostavne raspodele opterećenja.

Route 53 je zamišljen da se direktno integriše sa ključnim AWS servisima. Za CloudFront se najčešće koristi ALIAS na apex, tako što se dobija globalna distribucija za sadržaj i keširanje, a korisnici uvek idu do najbliže edge lokacije, tj edge lokacije sa najmanjom latencijom. Koristeći load balansere dobijaju se stabilni DNS nazivi koji iza sebe imaju dinamične resurse, kao što su instance i kontejneri (ECS/EKS). Za service kao API Gateway i Lambda mogu se mapirati domeni kroz ALIAS zapise, dok S3 static website hosting zahteva da se unesu S3 website endpoint, a ne ime bucketa.



Slika 114 Primer Public DNS Hosted Zone i njenih unosa

Na slici Slika 114 se mogu videti primeri DNS Hosted Zone za domen saas.globaldatanet.org. Ove unose je moguće proveriti i kroz druge alate, kao što su nslookup, dig itd. jer je u pitanju javni DNS.

AWS Load Balancing

Load balancer se može zamisliti kao pametna raskrsnica koja rutira saobraćaj namenjen za aplikacije, koje su u ovom slučaju odredišta, i posao im je da svaki dolazni zahtev rasporedi na jedan ili više servera u pozadini. Ti serveri mogu biti instance, kontejneri, lambda funkcije itd. Load balancer će iz mrežnog saobraćaja automatski da ukloni resurse koji nisu zdravi i da prilagodi opterećenje na preostale resurse bez prekida rada, a ujedno automatski da doda resurse koji postanu zdravi i započne rutiranje saobraćaja ka njima.

AWS pruža Load Balancing kroz familiju usluga koje naziva Elastic Load Balancing (ELB). Tipovi Load Balancera koje AWS nudi:

- **Application Load Balancer (ALB)** — Radi na aplikacionom sloju (HTTP/HTTPS), razume samu semantiku protokola i može da donosi odluke na osnovu hostname, path, header i query parametara upita. Samim tim podržava napredne funkcionalnosti pametnog rutiranja koji mogu biti jako korisni za API i web aplikacije.
- **Network Load Balancer (NLB)** — Radi na transportnom sloju (TCP/UDP/TLS) i optimizovan je za ekstremne performanse, što podrazumeva milione istovremenih konekcija, nisku latenciju itd. Pogodno je što može da terminira TLS ukoliko ima potrebe za skidanje enkripcije sa aplikacije ili da propusti neizmenjen TLS do klijenta, tj. cilja.
- **Gateway Load Balancer (GWLB)** — Specijalizovan je za ubacivanje mrežnih uređaja. Pomaže da se lako rasporedi, skalira i upravlja virtuelnim uređajima trećih strana. Pruža jedan gateway

za distribuciju saobraćaja između više virtuelnih uređaja, dok ih istovremeno skalira, na osnovu potražnje.

Nekada je postojao i Classic Load Balancer, ali se danas koristi samo zbog kompatibilnosti.

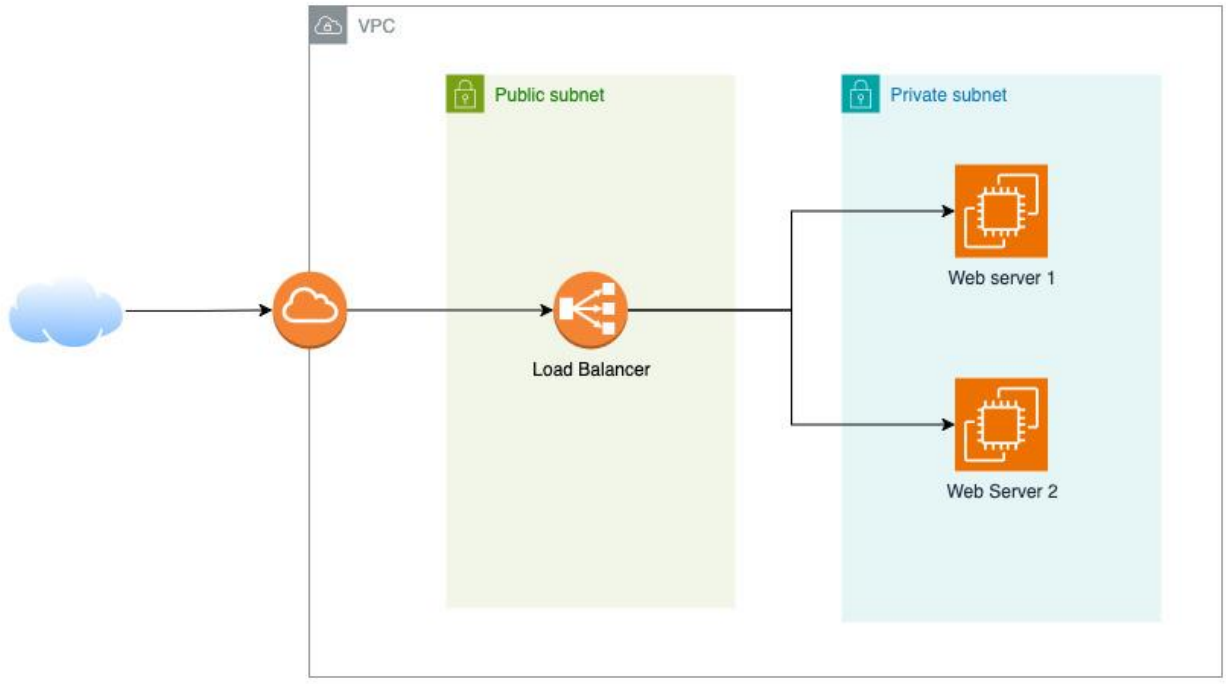
Kako bi se lakše razumeo rad Load Balancera, može se videti primer jednog HTTP zahteva. Polazi se od DNS zahteva (Route 53), zatim se dobija ime load balancera (DNS vrednost) i uspostavlja se veza ka definisanom listeneru (na primer HTTPS na portu 443). Listener primenjuje pravila rutiranja i odlučuje gde će da prosledi zahtev, tj. u koju target grupu. **Target grupa** je logički skup ciljeva kao što su EC2 instance, IP adrese, kontejneri, lambde itd. koji dele ista pravila provere zdravlja (health check).

Health check je provera stanja svakog cilja i može se konfigurisati na više načina; na primer, da li cilj na putanji /health vraća status code 200 u razumnom roku. Ako ta provera ne uspe (prespor odgovor ili nema odgovora), automatski se taj cilj sklanja iz grupe i saobraćaj se više ne rutira na taj cilj. U pozadini AWS upravlja i skaliranjem samog load balancera, pa se infrastruktura ispred aplikacije širi i skuplja po potrebi, tj. kako se saobraćaj menja.

Kod ALB postoji mogućnost rutiranja saobraćaja po sadržaju. Na primer, ako je zahteva na api.fakultet.com, on se rutira ka jednom mikroservisu, dok se saobraćaj sa test.fakultet.com rutira ka drugom mikroservisu. Isto tako može se vršiti rutiranje prema parametrima putanje; /v1 se usmeri ka staroj verziji aplikacije, dok /v2 se usmerava ka novoj. Takođe, može se integrisati sa **Web Access Firewall (WAF)** kako bi se postigla zaštita od web pretnji, poseduje integracije sa CloudWatch metrikama (latencija, broj 4xx/5xx grešaka itd.).

Vežba

U narednom delu biće prikazan primer rutiranja pomoću Load Balancera prema dijagramu sa slikeSlika 115.



Slika 115 Prikaz rutiranja pomoću load balancera za vežbu

Za svrhe prikaza rada, potrebno je napraviti dve EC2 instance, jednu nazvati web-instance-1, a drugu web-instance-2, na kojima će biti instaliran Ubuntu, web server i opsluživaće po web stranicu.

Kada status instanci pređe u running i status check bude 2/2, potrebno je povezati se na njih i pristupiti konzoli (SSH konekcija ili preko AWS Session Manager) i izvršiti sledeće komande.

apt-get update -y

apt-get install -y nginx

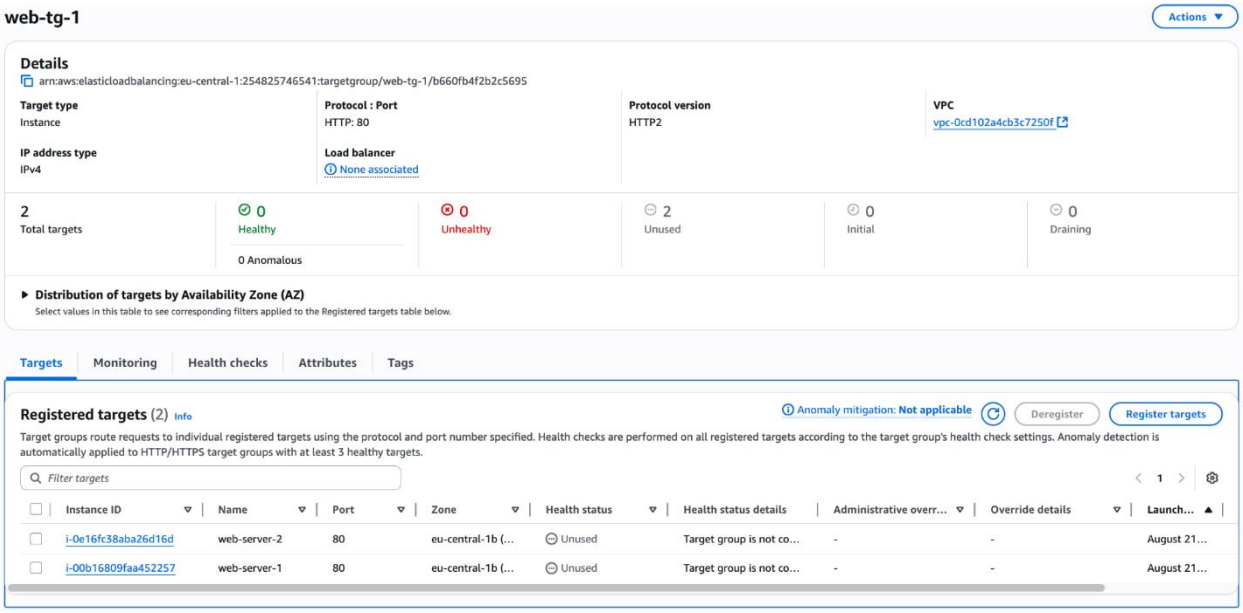
echo "I am web instance 1" | tee /var/www/html/index.html

systemctl enable --now nginx

Uz promenu teksta "I am web instance 1" na drugoj instanci u "I am web instance 2".

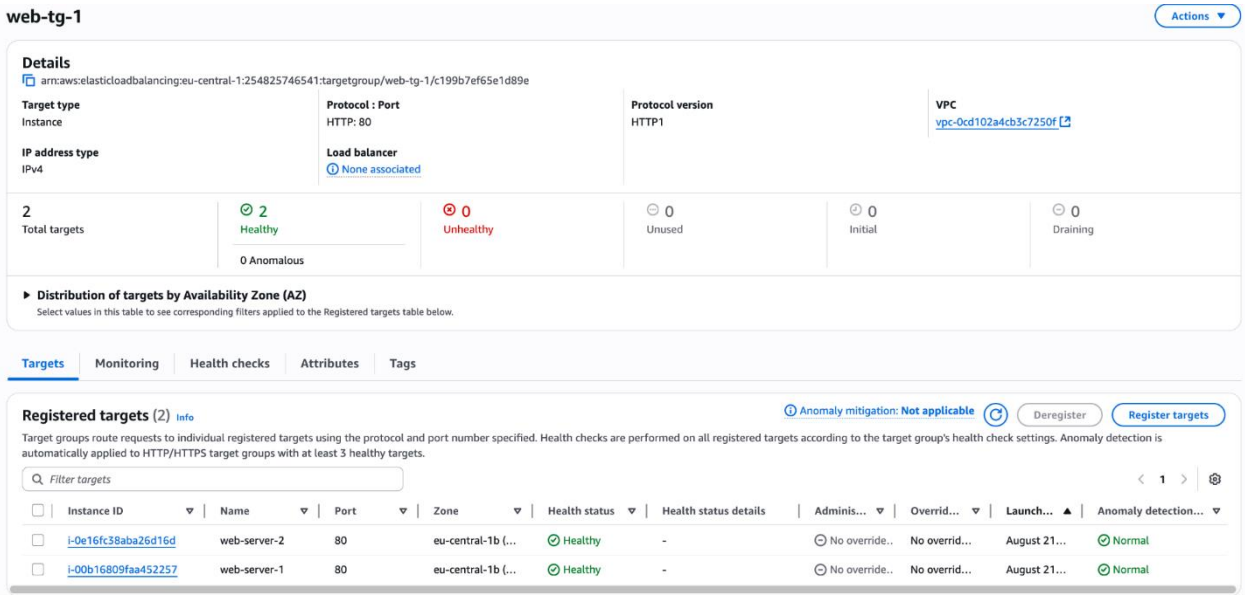
Kopiranjem javne IP adrese instanci možemo izvršiti proveru da li je sve ispravno konfigurisano, uz napomenu da je potrebno koristiti http:// ispred IP adrese, jer moderni web browseri automatski ispravljaju http zahtev na https. Na primer, http://3.122.254.188 i prikaz bi trebalo da bude tekst koji je ubačen.

Zatim, potrebno je pristupiti delu EC2 konzole, u okviru Load Balancing i napraviti novi Target Group. U okviru Target Group podesiti da je instances, dodeliti ime (na primer web-tg-1), izabrati HTTP protokol na portu 80 i ostaviti Health check kao HTTP na / putanji. U okviru Register targets odabrati obe EC2 instance i napraviti Target group. Ako je sve urađeno kako treba, dobiće se prikaz kao na slici Slika 116

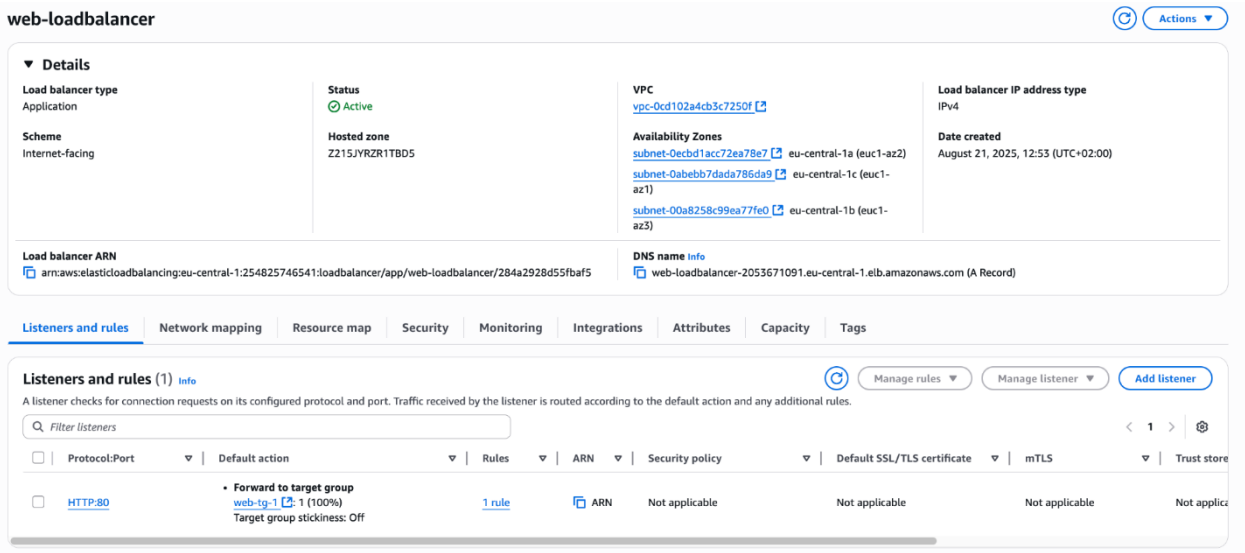


Slika 116 Registrovanje EC2 instanci u okviru Target grupe

Sada se prelazi na Load Balancers i pravi Application Load Balancer. Potrebno je dati ime, izabrati da je internet-facing i podesiti da je u okviru VPC-a, zatim izabrati da je u dve availability zone (kako bi se postigla visoka dostupnost), podesiti istu security grupu kao EC2 instance radi lakše konfiguracije i postarati se da je na toj grupi otvoren port 80 za sav saobraćaj. U okviru podešavanja Listeners and routing, podesiti HTTP protokol, port 80 i "Forward to" podesiti na prethodnog kreiranu target grupu — web-tg-1. Potrebno je sačekati kako bi se napravio load balancer i registrovali targeti. Sve je spremno kada status load balancera bude Ready i instanci u Target grupi bude Healthy (slike Slika 117 i Slika 118).



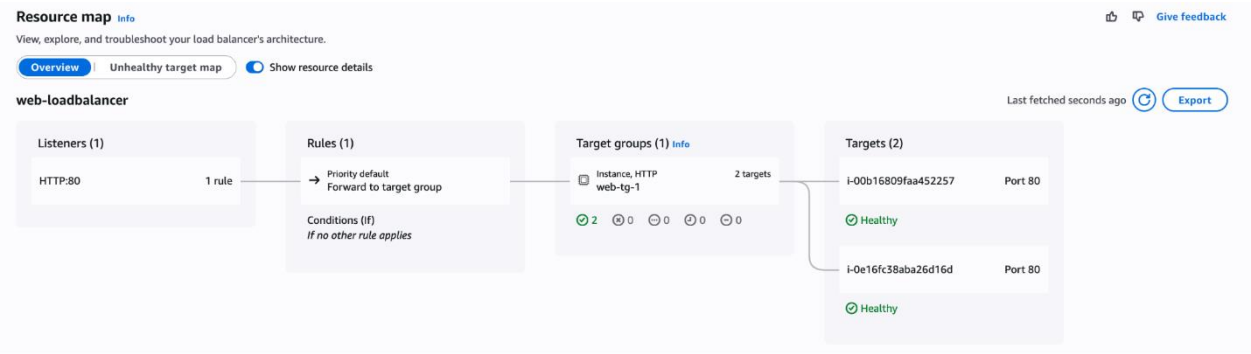
Slika 117 Prikaz EC2 instanci u Healthy stanju i spremnih za prihvat saobraćaja



Slika 118 Prikaz spremnog Load Balancera i njegove konfiguracije

U okviru detalja Load balancera postoji njegova DNS vrednost (DNS name na slici Slika 118). Tu vrednost kopirati i uneti u browser; pristupom će se pojaviti jedna od dve EC2 instance koje se mogu prepoznati po tekstu web stranice. Ukoliko se više puta osveži stranica, vrednost koju prikazuje će se naizmenično menjati između web instance 1 i 2. Tako znamo da se algoritam za rutiranje saobraćaja izvršava i na koju od dve instance rutira.

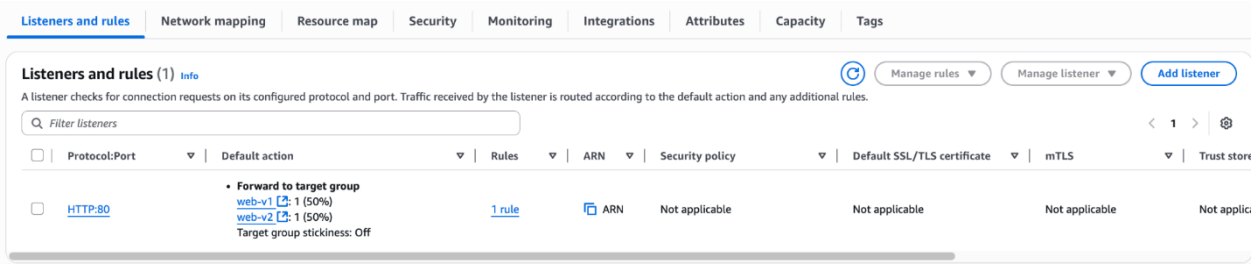
U okviru Load Balancera postoji Resource map deo; ukoliko je sve povezano kako treba, prikaz će biti kao na slici Slika 119.



Slika 119 Resource Map povezanog Load Balancera i ostalih resursa

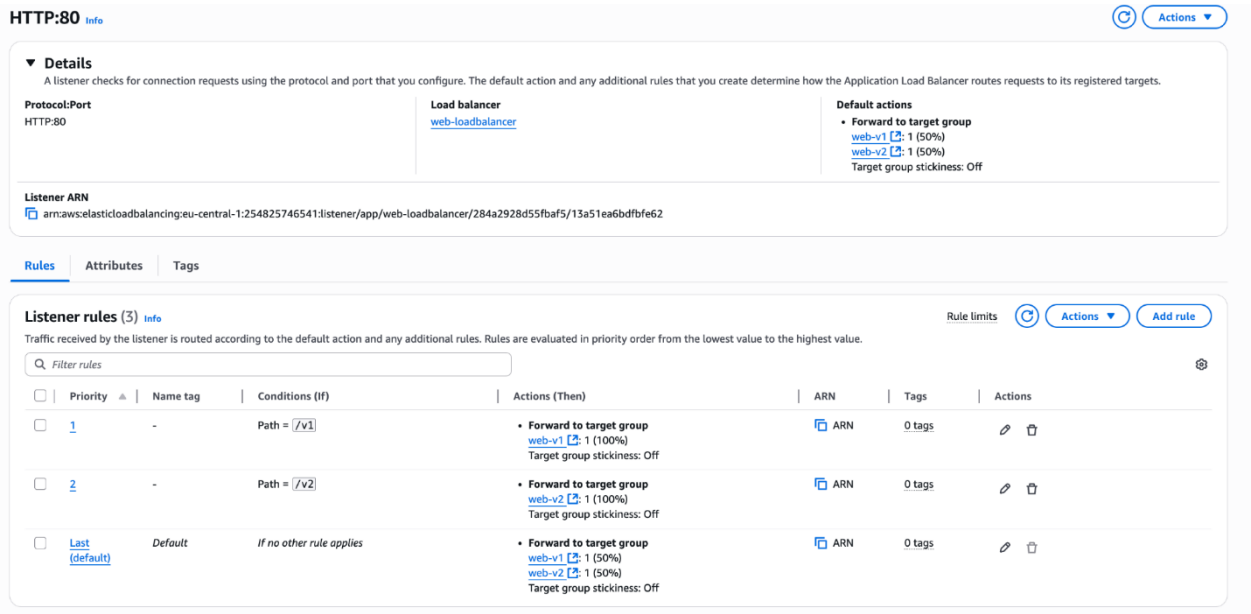
U slučaju rutiranja prema putanji, kao /v1 i /v2, potrebno je napraviti novi target group i u njega uključiti samo prvu web instancu (nazvati ga web-v1), isto tako uraditi i za drugu target grupu i drugu web instancu; sva ostala podešavanja uraditi kao pre, tako da sada postoje web-v1 i web-v2 target grupe koje sadrže odgovarajuće EC2 instance.

Zatim u okviru postojećeg web-loadbalancer ukloniti postojeći „listeners and rules" i dodati novi listener na portu 80 gde postoje dve target grupe — web-v1 i web-v2 (Slika 120).



Slika 120 Prikaz konfiguracije sa dve grupe na portu 80

Zatim je potrebno uneti pravila u okviru listenera; pravila će biti: ukoliko je putanja /v1, usmerava se na target grupu web-v1, i ako je /v2, na web-2, kao na slici Slika 121.



Slika 121 Konfiguracija pravila rutiranja na osnovu putanje

Kada se pristupi DNS vrednosti, može se dobiti stranica bilo kog od ova dva servera, dok na putanjama /v1 i /v2 dobijaju odgovarajući serveri. Zbog same konfiguracije u okviru servera, na tim putanjama se dobija 404 Not Found od web servera jer stranice ne postoje, ali se može videti da je saobraćaj u okviru metrika rutiran na njih; takođe, može se napraviti stranica pod tom putanjom i proveriti.

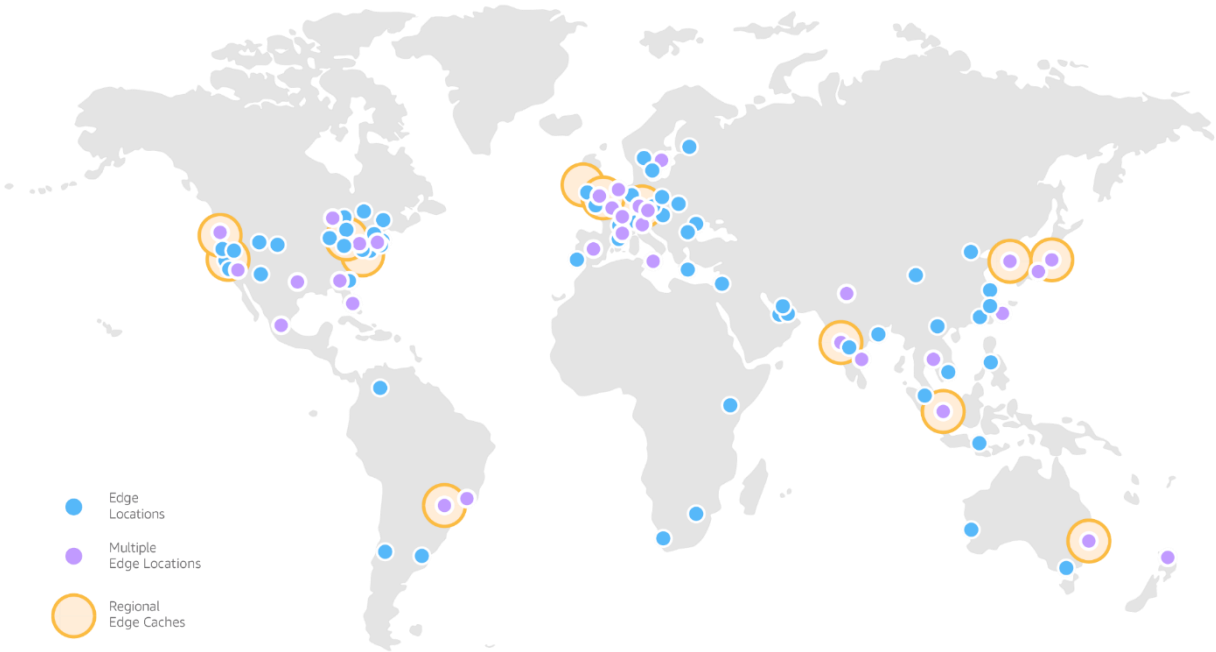
Kada se sve podesi, load balancer postaje kontrolna mrežna komponenta dostupnosti i performansi tako što reguliše distribuciju saobraćaja, kontroliše kvarove, a promene izvršava bez ometanja krajnjih korisnika.

Zadatak

Hostujte sopstvenu aplikaciju na EC2, u okviru VPC-a, uz pomoć load balancera i podeste rutiranje i *health check*-ove.

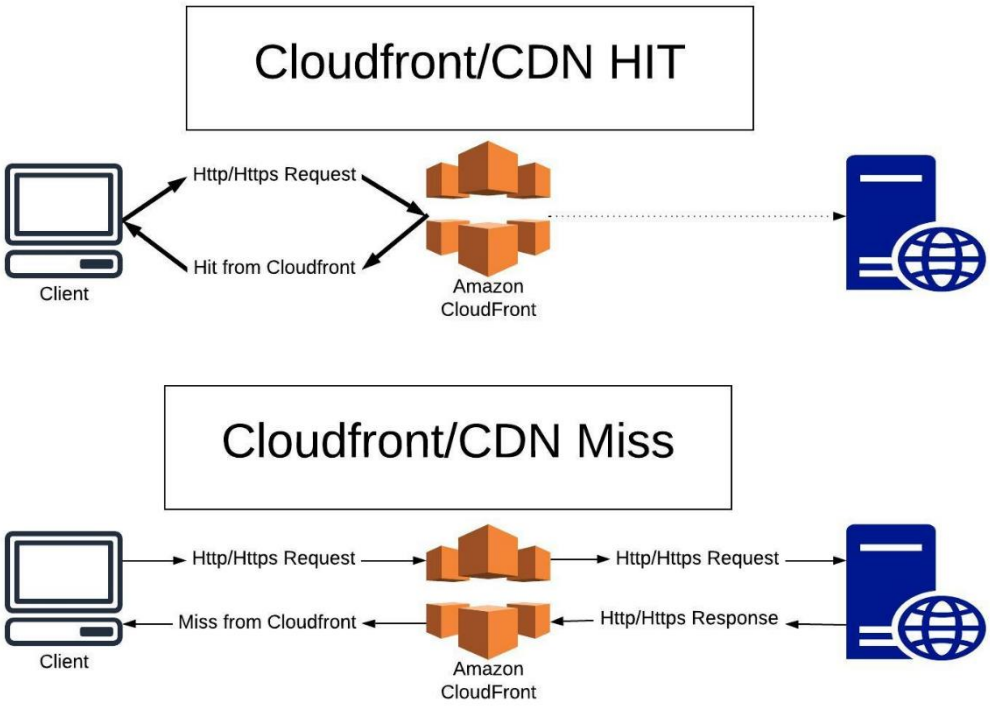
AWS CloudFront

Ako se CloudFront posmatra kao mrežna komponenta, on pripada globalnoj mreži koja približava sadržaj samim korisnicima. CloudFront je CDN sistem sa edge lokacijama koje su raspoređene širom sveta, sa ciljem da statičke i dinamičke resurse isporuči brzo, pouzdano i bez opterećivanja izvorišta (origin) (Slika 122).



Slika 122 Mapa lokacija CloudFront mreže

CloudFront, pored toga što je rešenje za statički saobraćaj, odlično služi u za dinamičke sadržaje. Keš može biti kratak ili čak nepostojeći, za video strimove se kombinuje sa MediaPackage/MediaStore, dok za velika preuzimanja uspeva da rasparčava sadržaj kako bi se lakše preuzeo. Ukoliko dođe do problema sa izvorom, CloudFront može da posluži keširani (zastareli) objekat tokom problema, time pružajući kontinuitet dok se problem ne reši (Slika 123).



Slika 123 Prikaz rada keširanja kod CloudFronta

Dodatne performanse se mogu postići i pomoću kompresije koju CloudFront nudi automatski kroz opsluživanje gzip formata. CloudFront podržava i invalidaciju keširanog sadržaja kroz ručni okidač iako kod velike količine mrežnog saobraćaja to treba raditi sa oprezom ili samo invalidaciju određenih putanja, kako ne bi došlo do naglog opterećenja izvora.

CloudFront ima i podršku za funkcije na samom edgu kroz CloudFront Functions u okviru kojih se mogu integrisati lagane JS rutine koje se izvršavaju i transformišu reques/response, kao i URL, normalizaciju headera, A/B rutiranje i jednostavne redirekcije. Kada su potrebne naprednije funkcionalnosti na edge-u, kao što je kompleksna logika, personalizacija, validacija tokena itd. koristi se Lambda@Edge sa punim okruženjem.

CloudFront podržava direktne integracije sa drugim AWS servisima, kao što su:

- Route 53
- WAF
- ACM (AWS Certificate Manager)
- S3
- Kinesis
- CloudWatch

Kao izvori, mogu se podesiti S3, EC2, Load Balancer, drugi HTTP izvori itd.

Uz sve ove mogućnosti, treba uzeti u obzir da je CloudFront onoliko dobar koliko i sama njegova konfiguracija i praćenje najboljih praksi. Loše keš polise, preterano grananje, masovne invalidacije itd. mogu pojesi sve benefit CloudFronta.

Praćenje mreže u oblaku

U oblaku je mreža drugačija od tradicionalne i definisana je softverom (VPC, rute, bezbednosne grupe itd.), dinamična je i menja se zajedno sa aplikacijom. Praćenje takve mreže ima svoje izazove i više se fokusira na tok samog saobraćaja, latencije, gubitak paketa, pokušaje pristupa i šta se promenilo u samoj konfiguraciji. Kod AWS to možemo posmatrati kroz tri jako bitna parametra:

- Metrike
- Logovi
- Tracing

U sredini svega ovoga je AWS CloudWatch (CW) koji skladišti metrike (CW Metrics), podiže alarme (CW Alarms) i prikazuje kontrolne table (CW Dashboards). CW Logs prikuplja tekstualne logove sa raznih servisa, Logs Insights omogućava da se koristi jezik nalik SQL da se brzo pretraže metrike i primeni filtriranje, kao i da se vrše napredne funkcije nad samim logovima.

AWS GuardDuty se bavi bezbednošću samih naloga, opterećenja, podataka i vrši inteligentnu detekciju pretnji uz pomoć Ai/ML servisa, uzima u obzir VPC tokove, DNS zapise i druge izvore kako bi pronašao sumnjive akcije kao skeniranje portova, neobična kretanja podataka i poznate šeme napadanja. **VPC Flow Logs** su zapisi o tokovima saobraćaja koji je prihvaćen ili odbijen, uz informaciju o broju bajtova, čvorova, putanji, izvoru i odredištu, portovima i protokolima. Ove logove je moguće skladištiti u CloudWatch Logs ili u S3 pa pomoću dodatnih servisa kao što je Athena raditi analizu i S3 koristiti kao Big Data warehouse.

Kada se posmatra **Application Load Balancer,** postoje zapisi dnevnika pristupa (access logs), kao što su statusni kodovi, vremena, target grupe, latencije (p50, p90, p99), količine 4xx i 5xx grešaka itd. Kod **Network Load Balancera** na transportnom sloju prikazuju se metrike o konekcijama i greškama. Ovi logovi su jako bitni jer sav saobraćaj bi idealno išao kroz Load Balancer.

Kod servisa kao **WAF**, koji stoje ispred Load Balancera ili CloudFronta, šalju se detaljni logovi odbijenih upita. Za vidljivost nad DNS saobraćajem zaduženi su **Route 53 DNS Firewall** i **Resolver Query Logging** koji daju vidljivost nad DNS saobraćajem.

Za dijagnostiku topologije i pravila postoje „interaktivni“ alati, kao što je **VPC Reachability Analyzer** koji simulira put od izvora do odredišta i jasno pokazuje gde se paket zaustavlja (rute, NACL, security grupa, firewall). Kada se proverava usaglašenost, **Network Access Analyzer** traži neželjene puteve, kao što je put do RDS baze iz javnog interneta.

Napredna forenzika često zahteva i pregled samih paketa. Tu služi **VPC Traffic Mirroring** koji kopije paketa sa mrežnih interfejsa (ENI) šalje ka komponenti koja vrši analizu. Treba reći da je je kao servis skup i da ga treba pažljivo koristiti, ali isto tako je nezamenljiv kada se traže retke greške ili sumnjivi tokovi koji se prikrivaju.

Monitoring mreže u AWS nije jedan alat, već skup alata i dobrih praksi. Izborom pravih metrika i logova, centralizacijom i automatizacijom dobijamo mrežu koja je transparentna i predvidiva. Kada se posmatraju kontrolne table metrika, ukrste sa logovima i analizama putanje, problemi postaju jasni.

Zadatak

Napraviti API Gateway, u okviru njega povezati Lambdu ili Application Load Balancer koji će opsluživati aplikaciju ili statičku web stanicu na EC2. Zatim uključiti detaljno logovanje INFO i ERROR logova na API Gateway i ispratiti jedan API poziv od API Gateway do aplikacije i nazad.

8. Upravljanje resursima u oblaku: planiranje, skaliranje i balansiranje opterećenja

Ciljevi ovog poglavlja jesu upoznavanje sa principima planiranja resursa u oblaku i mehanizmima za skaliranje i balansiranje opterećenja.
Student po savladanom poglavlju može da:

- poredi specifičnosti planiranja kod različitih modela usluge
- navede oblike skaliranja resursa u oblaku
- navede oblike balansiranja opterećenja u oblaku
- analizira kriterijume balansiranja opterećenja u oblaku

Student ume da konfiguriše skaliranje i balansiranje resursa u AWS okruženju u oblaku.

Koncept računarstva u oblaku prati primamljiva teza o postojanju neograničenih resursa koji se nalaze „tamo negde". Naravno, čak i najveći dobavljači usluga u oblaku nemaju neograničene resurse. Postoji veliki zajednički skup *(pool)* resursa koji su raspoloživi, međutim, taj pool je ograničen, a resursi koštaju, tako da je potrebno kalkulisati koliko resursa je zaista potrebno, uz svest o svim prednostima i nedostacima statičkog i dinamičkog obezbeđivanja resursa.

8.1. Planiranje kapaciteta

U oblaku se planiranje kapaciteta vrši na dva nivoa. Na prvom nivou, svaki korisnik usluge u oblaku vrši svoje vlastito planiranje kapaciteta. Na drugom nivou, dobavljač usluge analizira kapacitetske zahteve svih korisnika zajedno i pravi konačno ukupno planiranje kapaciteta. U tom smislu, korisnici usluga u oblaku mogu se podeliti u dve grupe:

- IaaS korisnici obično planiraju i rezervišu fiksni kapacitet resursa koji će koristiti u određenom periodu. To ih podstiče da obave svoje vlastito planiranje kapaciteta, ali samo za virtuelne resurse koje konzumiraju.
- SaaS i PaaS korisnici obično biraju model upotrebe resursa koji se dinamički meri. Stoga je njihova deklaracija o zahtevima za usluge, zabeležena u ugovorima o nivou usluge (SLA), vrlo važna za dobavljača u procesu planiranja kapaciteta.

Međutim, za sve vrste/nivoe korisnika usluga u oblaku, planiranje kapaciteta se potpuno razlikuje od scenarija u kojem nije korišćen oblak. Oni nisu ograničeni predviđanjem tačnih fizičkih potreba resursa za svoje aplikacije, što je kritičan zadatak. Pošto su resursi dostupni na zahtev u oblaku, planiranje kapaciteta nije previše izazovno za korisnike i mogu početi sa umerenim procenama resursa.

Međutim, ako korisnik usluge na bilo kom nivou oblaka (bilo da je IaaS, PaaS ili SaaS) ne može tačno da proceni buduće zahteve za resursima i stvarna potražnja potroši mnogo više resursa nego što je planirano, model naplate može neočekivano povećati iznos naplate prema potrošnji. Zato korisnici moraju upravljati kapacitetom u skladu sa svojim budžetskim ograničenjima i planirati svoju sopstvenu konačnu potrošnju.

Jedna važna stvar koju treba razumeti prilikom razmatranja planiranja kapaciteta je tip i opseg odgovornosti za određeni nivo dobavljača usluga, kao i za korisnika usluga. Krajnji korisnici, koji koriste samo SaaS uslugu, kupci su dobavljača usluga iz SaaS kategorije. Provajder SaaS usluga može, zauzvrat, biti kupac nekog dobavljača usluga iz PaaS kategorije, ako nije nezavisni dobavljač računarstva. Slično tome, provajder PaaS usluga je ili nezavisni dobavljač, ili kupac provajdera usluga iz IaaS kategorije.

Uzmimo primer tri različite kompanije koje su dobavljači različitih oblačnih usluga prema nekom ugovoru. Kompanija A — dobavljač IaaS usluga, kompanija B — provajder PaaS usluga i kompanija C —

provajder SaaS usluga. Kupci kompanije C su korisnici aplikacija (krajnji korisnici računarstva). Oni mogu izabrati tri načina da obaveste kompaniju C o svojim zahtevima za kapacitetom.

Slučaj 1 Mogu proceniti tačne poslovne zahteve i obavestiti C.

Slučaj 2 Prenose celu odgovornost za podršku svojim zahtevima na kompaniju C.

Slučaj 3 Mogu izabrati srednji put, gde će poslovni zahtevi koji prelaze određeni (procenjeni) nivo biti podržani od strane C.

Izbor i zahtevi moraju biti navedeni kao ugovor u SLA (Service Level Agreement — ugovor o uslugama), kako bi C mogao proceniti zahteve za kapacitetom i preduzeti odgovarajuće mere za njihovo obezbeđivanje. Kompanija C, nakon što prikupi sve SLA ugovore sa svojim korisnicima, mora proceniti buduće zahteve na sličan način. Oni takođe imaju tri slična izbora kao i njihovi korisnici.

Kompanija B ima dve opcije za održavanje kapaciteta resursa kako bi podržala svoj poslovni zahtev. Sa pravilnom procenom, B može držati nekoliko dodatnih virtuelnih servera (koje joj obezbeđuje A) kao rezervne resurse, kako bi podržala neočekivano povećanje opterećenja i upravljala kapacitetom. Druga opcija je da direktno prenesu odgovornost za planiranje kapaciteta na kompaniju A, verujući u njihove sposobnosti. Takođe, mogu izabrati srednji put, tako što će upravljati planiranjem kapaciteta do određenog nivoa, a zatim preneti odgovornost na dobavljača infrastrukturnih usluga u ekstremnim slučajevima. Međutim, ugledni dobavljač usluga mora samostalno preuzeti odgovornost, bez oslanjanja na druge.

Upravljanje fizičkim infrastrukturnim resursima je odgovornost dobavljača IaaS usluga A. Oni nemaju opciju da prenesu zadatak planiranja kapaciteta na druge dobavljače usluga, jer je pružanje fizičkih resursa krajnja odgovornost IaaS provajdera A.

Dakle, zadatak planiranja kapaciteta zahteva odgovarajuću procenu budućih zahteva na svakom sloju oblačnih usluga. Iako korisnici SaaS i PaaS usluga ne mogu direktno učestvovati u aktivnosti planiranja kapaciteta, oni bi trebalo da procene poslovnu potražnju i unapred obaveste svoje odgovarajuće osnovne slojeve o budućoj potražnji. Svaki sloj mora biti dobro obavešten o mogućim budućim zahtevima kako bi se pripremio.

Ako kompanija „B" ostane neinformisana (od svojih viših slojeva) i ne bude mogla da podrži zahteve za resursima iznad svog plana, verovatno će A biti u nemogućnosti da podrži dodatno opterećenje u mnogim slučajevima, ako više njihovih korisnika (kao što je B) ne uspe da to uradi. Dobavljač IaaS usluga treba da bude viđen kao poslednja linija odbrane, i odgovornost za obezbeđivanje neograničenih resursa ne sme biti preneta na njih.

Koraci planiranja kapaciteta su sledeći:

Korak 1. Određivanje očekivane potražnje – U prvom koraku procesa planiranja kapaciteta dobavljač usluga mora pažljivo ispitati očekivane obrasce ukupnog korišćenja resursa, jer se oni menjaju tokom određenog vremenskog perioda.

Korak 2. Analiza trenutnog odgovora na opterećenje – Dobavljač zatim mora analizirati dostupne kapacitete resursa svog sistema i kako aplikacije reaguju na opterećenje (ili preopterećenje) pri trenutnom kapacitetu, kako bi se identifikovala potreba za dodatnim kapacitetima koje treba dodati.

Korak 3. Razumevanje vrednosti sistema – Na kraju, dobavljač usluga mora biti svestan vrednosti sistema za poslovanje, kako bi znao kada dodavanje dodatnih kapaciteta donosi vrednost i kada to nije slučaj.

8.2. Skaliranje

U IT žargonu često se može čuti da neki softver ili servis „dobro skalira“ ili na primer „MS ACCESS je odličan za male lokalne baze, ali loše skalira“. Skaliranje je upravo jedno od osnovnih svojstava računarstva u oblaku, koje omogućava veliku fleksibilnost i održavanje performansi. Sa strane korisnika, sistem koji dobro skalira jeste onaj koji podržava povećani broj korisnika, bez pada performansi. Da bi se opslužio veći broj korisnika, sistem iznutra mora takođe da skalira, a to povlači povećanje kapaciteta.

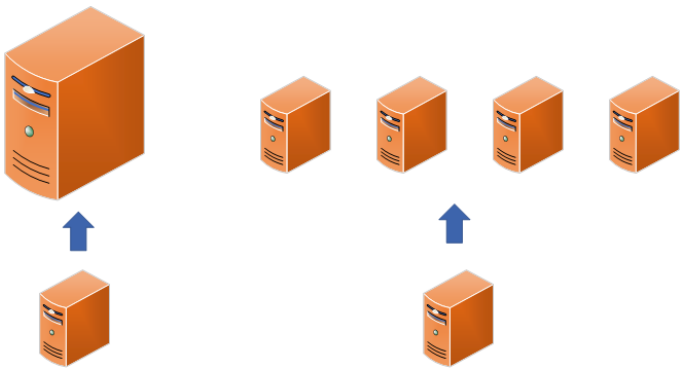
Skaliranje se može definisati kao svojstvo sistema, modela ili funkcije koja opisuje njegovu sposobnost rasta ili smanjenja kada je to potrebno. U računarskom kontekstu, skaliranje predstavlja sposobnost sistema ili aplikacije da efikasno obrađuje promenljiv nivo opterećenja bez izazivanja situacija u kojima nedostatak resursa ometa performanse ili višak resursa povećava troškove obrade.

Jednostavno rečeno, skaliranje se definiše kao sposobnost proširivanja (ili smanjenja) kako bi se prilagodio rastu (ili padu) i ispunili poslovni zahtevi[27]. Arhitektura sistema ili aplikacije može se nazvati skalabilnom ako se njene performanse poboljšavaju dodavanjem novih resursa, a to poboljšanje je proporcionalno dodatom kapacitetu.

Sa pojmom skaliranja tesno je povezano svojstvo elastičnosti. Elastičnost podrazumeva brzo — često automatsko skaliranje u realnom vremenu — meru koliko brzo sistem može da skalira.

Uopšteno, postoje dva tipa skaliranja (Slika 124). Jedan je vertikalno skaliranje. Podrazumeva povećanje pojedinih resursa, na primer, ukoliko imamo jedan server i postoje zahtevi za više memorije, moguće je dodati još RAM-a u server. Svakako, postoje ograničenja samog hardvera u tome koliko još memorije ili diskova je moguće dodati.

Sa druge strane, kod horizontalnog skaliranja, dodaju se nove kompletne instance. U prethodnom primeru, umesto dodavanja memorije ili diskova, dodaje se kompletan server koji zatim preuzima deo opterećenja. Horizontalno skaliranje je podrazumevana strategija u oblaku. Prednosti su što ne postoji „single point of failure“ (npr. u prethodnom primeru server je jedinstven i njegovim otkazom, gubi se usluga), tako da je otpornost na greške veća. Takođe, skaliranje se vrši na zahtev, prema potrebama. Cena horizontalnog skaliranja je potreba za posebnim mehanizmom koji će balansirati opterećenja prema skaliranim resursima. Horizontalno skaliranje je dugoročno rešenje za skaliranje.



Slika 124 Vertikalno i horizontalno skaliranje

Osnovni nedostatak horizontalnog skaliranja jeste povećana kompleksnost upravljanja, jer je u pitanju više servera koje treba koordinisati u distribuiranom okruženju.

[27] U praksi, najčešće se pod terminom skaliranje podrazumeva skaliranje naviše, tj. povećanje resursa.

Kod dinamičkog skaliranja, sistem se može prilagođavati tokom rada bez potrebe za ponovnim pokretanjem ili prekidanjem bilo koje usluge. Ovaj ključni zadatak dinamičke promene kapaciteta može se izvršiti na dva načina:

Ručnim putem: Sistem se može skalirati tokom rada izvršavanjem odgovarajućih komandi putem aplikacionog interfejsa.

Automatski: Ovakvo skaliranje sistema se implementira pomoću programa koji automatski prilagođavaju kapacitet sistema posmatrajući stvarnu potražnju.

Mogućnost ručnog prilagođavanja kapaciteta sistema tokom njegovog rada predstavlja veliki izazov za bilo koje računarsko okruženje. Međutim, prava snaga skaliranja u oblaku leži u mogućnosti automatskog skaliranja. U tom slučaju nije potrebna ljudska interakcija jer se sistem može samostalno prilagođavati ili promeniti veličinu.

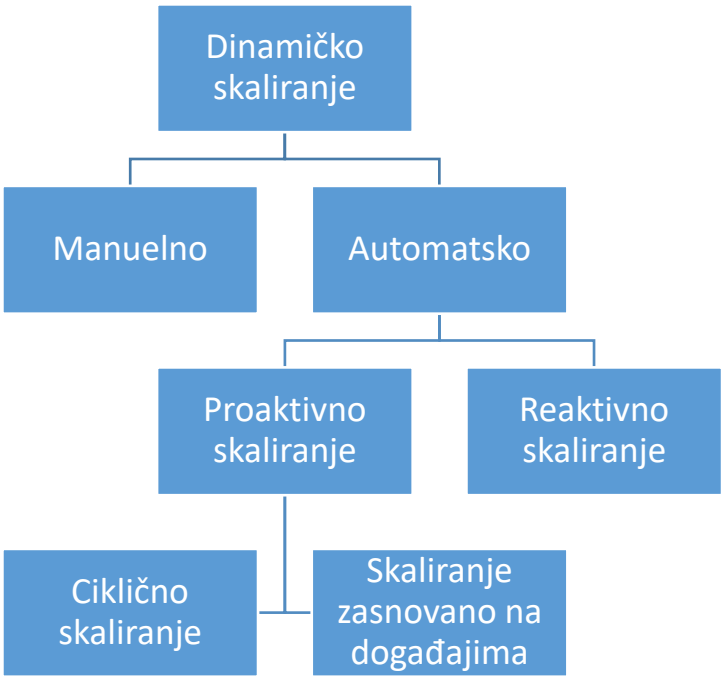
Dinamičko automatsko skaliranje obično se naziva autoskaliranjem, poznatim i kao skaliranje u oblaku. Autoskaliranje se može implementirati na dva različita načina:

- Skaliranje na osnovu unapred definisanog rasporeda, poznato kao proaktivno skaliranje.
- Skaliranje na osnovu trenutne stvarne potražnje, poznato kao reaktivno skaliranje.

Proaktivno skaliranje. Zahtevi aplikacije za resursima u principu variraju u toku vremena. Na primer, sajt za e-kupovinu se retko posećuje rano ujutru, a korporativne aplikacije se najviše koriste pri kraju radnog vremena. Striming platforme (npr. Netflix, Disney) najveće opterećenje imaju tokom večernjih sati, naročito vikendom, a sportske aplikacije (npr. za praćenje rezultata uživo) najaktivnije su tokom velikih sportskih događanja, danima kada se igraju npr. utakmice Lige šampiona. Kada je, kao u datim primerima, očekivano povećanje ili smanjenje opterećenja poznato unapred, postavlja se unapred programiran plan koji automatski prilagođava kapacitet resursa. Kada se skaliranje odvija periodično, u redovnim intervalima i unapred definisanom dobu dana, reč je o proaktivnom cikličnom skaliranju. Kod situacija kod kojih je porast opterećenja vezan za pojedinačne događaje (npr. početak promotivne kampanje), vrši se skaliranje na osnovu događaja. U oba scenarija proaktivnog skaliranja važno svojstvo je da se ne čeka da dođe do povećanog opterećenja, već se ono predviđa i resursi rezervišu unapred.

Reaktivno skaliranje. Kada sistem trenutno reaguje na promene zahteva, bilo dodavanjem ili uklanjanjem kapaciteta, reč je o reaktivnom skaliranju. Kada iskorišćenje nekog resursa dostigne određeni prag, sistem automatski skalira. Unapred je potrebno definisati pragove u odnosu na koje se vrši skaliranje, a kasnije sistem skalira samostalno. Reaktivno skaliranje naizgled može delovati kao najbolje rešenje, kojim se u drugi plan stavljaju sve druge opcije. Međutim, ovaj vid skaliranja ne isključuje planiranje resursa i oslanjanje isključivo na reaktivno skaliranje; može imati posledice na ukupne performanse i dostupnost usluge i aplikacija. Na primer, ako opterećenje poraste veoma naglo i pređe prag, koji je relativno nizak, mora proći neko vreme da bi se pokrenuli dodatni resursi i tu može doći do prekida u radu ili čak gubitaka podataka. Na drugoj strani, preniski pragovi mogu dovesti do preranog pokretanja resursa, koji onda prave trošak, a nisu dovoljno iskorišćeni.

Na slici Slika 125 sumarno su prikazani načini dinamičkog skaliranja.



Slika 125 Klasifikacija dinamičkih oblika skaliranja

U praksi je čest slučaj kombinacije više oblika skaliranja: proaktivnih i reaktivnih. Na primer, ako je potražnja za resursima neke aplikacije visoka vikendom, definiše se rezervacija više resursa vikendom — to je ciklično proaktivno skaliranje. Ali, u slučaju da se spontano pojavi veći broj zahteva drugim danima, postavlja se prag opterećenja čijim prelaskom se pokreću novi resursi — to je reaktivno skaliranje. Naposletku, ukoliko dođe do vanredne situacije, u smislu napada na aplikaciju, treba računati na manualno skaliranje, kojim se posebno zadaje kapacitet resursa.

U uvodnom poglavlju navedena je kategorija hibridnog oblaka, kao rešenja koje kombinuje privatni i javni oblak. Hibridni oblaci su često vezani upravo za problem skaliranja. Kada postojeća infrastruktura privatnog oblaka ne može da obradi nagle skokove opterećenja, radno opterećenje se prenosi na javni oblak. Kada se opterećenje aplikacije smanji, resursi javnog oblaka se oslobađaju i sistem se vraća na interno okruženje. Na taj način, radno opterećenje prelazi između spoljnog i internog hostovanja kako bi se prilagodilo promenljivim zahtevima za resursima, bez ikakvog saznanja korisnika. Preopterećenje privatnog oblaka koji dovodi do hibridizacije i skaliranja poznato je kao „provala oblaka” *(cloud bursting)*.

8.3. Balansiranje opterećenja

Uz odgovarajuće planiranje kapaciteta u okviru računarstva u oblaku, veliki *pool* virtualizovanih resursa stvara iluziju beskonačnog izvora računarskih resursa. U prethodnim sekcijama razmatralo se kako aplikacije često moraju da skaliraju kako bi upravljale promenljivim opterećenjem, kao i kako se resursi dinamički alociraju kako bi se zadovoljili zahtevi skaliranja u oblaku. Takođe je naglašeno da je horizontalno skaliranje bolje pripremljeno za skaliranje velikih razmera u distribuiranoj okolini, poput oblaka.

Nakon dobijanja alociranih ili dealociranih dinamičkih resursa, horizontalno skaliranje pruža elastičnost računarstvu u oblaku. Međutim, ova fleksibilnost donosi i veliku složenost implementacije. Balansiranje opterećenja predstavlja mehanizam putem kojeg se ove karakteristike ostvaruju. Osim toga, budući da postoji više resursa dostupnih za opsluživanje određenog tipa zahteva, u distribuiranom računarskom okruženju postaje neophodno ravnomerno rasporediti te zahteve među

dostupnim resursima kako nijedan od njih ne bi postao preopterećen i time narušio performanse celog sistema. Upravo logika ili algoritam u toku izvršavanja, zaduženi za distribuciju zahteva, moraju biti sposobni da ravnomerno rasporede opterećenje.

Pravilno osmišljena arhitektura distribucije i balansiranja opterećenja smanjuje neiskorišćenost resursa, poboljšava performanse sistema i umanjuje troškove obrade.

Kod implementacije balansiranja u oblaku, balanser opterećenja, poseban server kome je to osnovni zadatak, raspoređuje zahteve ka više instanci, na osnovu određenog algoritma.

Postoji više pristupa balansiranju opterećenja.

Statičko balansiranje. Kod ovog pristupa balansiranje se vrši isključivo na osnovu zahteva, a ne na osnovu poznavanja stanja opterećenja postojećeg sistema. Faktički, cilj je da se zahtevi ravnomerno rasporede, bez obzira na to koliko opterećenja će oni izazvati kod servera na koje se raspoređuju.

Dinamičko balansiranje. Dinamički pristup ne raspoređuje opterećenje među resursima u zavisnosti od unapred definisanih pravila. Planiranje zadataka obično se vrši na osnovu trenutnog stanja resursa ili čvorova (servera). Prednost korišćenja dinamičkog balansiranja opterećenja je u tome što, ukoliko bilo koji čvor postane potpuno opterećen ili otkaže, to neće dovesti do zaustavljanja celokupnog sistema.

Ovaj pristup se implementira primenom mehanizma povratne sprege, gde se resursi kontinuirano nadgledaju. Čvorovi balanseru opterećenja periodično šalju informacije o opterećenju i statusu, i ukoliko bilo koji čvor prestane da odgovara tokom rada, balanser opterećenja prestaje da mu prosleđuje saobraćaj. Ova specifična karakteristika je u računarstvu u oblaku nasleđena sa mrežnih platformi zasnovanih na grid-tehnologiji. Međutim, ovaj pristup zahteva komunikaciju u realnom vremenu preko mreže, što dodaje saobraćaj sistemu.

Implementacija tehnike se vrši na dva odvojena načina: distribuirani i nedistribuirani načini. U distribuiranom pristupu balansiranju opterećenja, zadatak balansiranja opterećenja obavljaju svi čvorovi (serveri) prisutni u sistemu. To znači da svi čvorovi sistema dele odgovornost za balansiranje opterećenja kroz međusobnu komunikaciju. Deljenje zadatka može se ponovo vršiti na dva načina: kao kooperativni i nekooperativni zadaci deljenja. Kada svi čvorovi rade zajedno paralelno kako bi postigli isti cilj, kao što je poboljšanje ukupnog vremena odgovora čvorova, tehnika se naziva kooperativno balansiranje opterećenja. Inače, ako čvorovi rade odvojeno kako bi postigli neki lokalni cilj, tehnika se naziva nekooperativno balansiranje opterećenja. U nekooperativnom pristupu, različiti čvorovi mogu imati različite ciljeve; jedan može biti fokusiran na vreme odgovora, dok je drugi fokusiran na isporuku sistema.

Distribuirani dinamički pristup balansiranju opterećenja delegira zadatak balansiranja opterećenja svim čvorovima (serverima) prisutnim u distribuiranom sistemu.

U nedistribuiranom pristupu balansiranja opterećenja, zadatak balansiranja opterećenja upravlja jedan pojedinačni čvor ili klaster čvorova. Kada se upravlja od strane jednog čvora, pristup se naziva centralizovano balansiranje opterećenja. U ovom pristupu, jedan čvor, nazvan centralni čvor, upravlja zadatkom balansiranja opterećenja celokupnog sistema. Nakon toga svi ostali čvorovi sistema moraju direktno komunicirati sa ovim centralnim čvorom kako bi dobili dodelu zadataka.

Kada zadatkom balansiranja opterećenja upravlja klaster čvorova, pristup se naziva poludistribuirano balansiranje opterećenja. Kod ovog pristupa, ceo sistem je podeljen na više delova, a svaki deo ima grupu ili klaster čvorova. U svakoj particiji ili grupi čvorova, zadatak balansiranja opterećenja se održava

na centralizovan način. Jedan čvor u svakoj grupi dobija zadatak da balansira opterećenje unutar grupe, a svi ovi čvorovi zajedno upravljaju balansiranjem opterećenja celokupnog sistema na saradnički način.

U nedistribuiranom dinamičkom pristupu balansiranja opterećenja ne učestvuju svi čvorovi distribuiranog sistema. Jedan ili nekoliko čvorova dobija zadatke.

Čvorovi koji se bave balansiranjem opterećenja u distribuiranom sistemu moraju pratiti status jedni drugih putem komunikacije porukama. Ovo generiše dodatni saobraćaj u sistem i uvodi dodatno opterećenje. Distribuirani pristup dinamičkom balansiranju opterećenja generiše više poruka nego nedistribuirani pristup, jer je potrebno više interakcije među čvorovima balansa opterećenja u distribuiranom pristupu. S druge strane, u distribuiranom pristupu, sistem ne trpi veće probleme čak i ako jedan ili više čvorova balansa opterećenja u sistemu ne uspe, jer više balansa opterećenja radi zajedno.

Centralizovani pristup balansiranju opterećenja razmenjuje vrlo malo poruka da bi doneo odluku. Međutim, ovaj pristup može uvesti druge pretnje za sistem. Pošto je čitav proces balansiranja opterećenja sistema zavisan od jednog čvora, može doći do problema sa uskim grlom. Takođe, postoji mogućnost pojave jedne tačke kvara (single point of faliure) jer proces balansiranja opterećenja može da trpi ako centralni čvor ne uspe. Generalno, da bi se rešile takve situacije, sistemi čuvaju rezervni balans opterećenja koji je konfigurisan da preuzme kontrolu. Centralizovani pristup je prikladniji za manje mreže.

Prilikom realizacije balansiranja opterećenja, u obzir se uzimaju različiti parametri:

- Iskorišćenje resursa: Svi resursi moraju se iskoristiti na planirani način tako da se nijedan resurs ne preoptereti i ugrozi uslugu.
- Vreme odziva: Sam balanser opterećenja mora da reaguje brzo i što pre preusmeri opterećenje na odgovarajući čvor.
- Ispravnost sistema: Balanser mora imati informaciju o tome da li eventualno neki čvor ima problem i ne može da odgovara na zahteve. Onda tom čvoru ne prosleđuje zadatke.
- Skaliranje: Balanser omogućava skaliranje čvorova koji odgovaraju na zahteve.

Pitanja za proveru i obnavljanje znanja

1. Da li dobavljači usluga u oblaku mogu imati svog dobavljača? Obrazložite.
2. Kako glase koraci planiranja kapaciteta?
3. Koji tip skaliranja dominira kod oblaka: vertikalno ili horizontalno? Zašto?
4. Uzmite kao primer sistema koji treba da skalira platformu za e-učenje, na kojoj su video predavanja, šalju domaći, obaveštenja i postavljaju resursi za učenje, proveru znanja itd. Kako biste definisali skaliranje?
5. Na koji način se može skalirati u slučaju korišćenja privatnog oblaka?
6. Kako funkcioniše dinamičko balansiranje? Da li ovakvim balansiranjem može upravljati više čvorova?

8.4. Autoskaliranje u AWS

Autoskaliranje je sposobnost sistema da automatski prilagođava kapacitet resursa u skladu sa opterećenjem. U AWSu to ce najčešće povezuje sa **EC2 Auto Scaling Group** (**ASG**), koja automatski dodaje ili uklanja EC2 instance na osnovu metrika kao što su CPU, mrežni saobraćaj, broj zahteva po targetu iza load balancera itd. i raspoređuje ih preko više Availability Zone, obnavlja neispravne instance i održava željeni kapacitet.

Pored EC2 ASGa, postoji i **Application Auto Scaling** za druge servise, kao što su **ECS** (broj taskova), **DynamoDB** (RCU/WCU), **Aurora** (read replike), **Kinesis** (shardovi), **OpenSearch** (nodovi) itd. **Lambda** skalira sama na osnovu zahteva, dok se u **EKS** svetu autoskaliranje često radi preko **Karpentera** ili **Cluster Autoscalera**. Za potrebe demonstracije i razumevanja koncepta, EC2 Auto Scaling + Load Balancer je najintuitivniji primer koji će biti pokriven u ovom poglavlju.

Launch Template je recept za instancu, sadrži AMI, tip, user-data (skripta koja se izvršava pri startu), security grupe, key pair itd. **Auto Scaling Group (ASG)** drži minimalan, željeni i maksimalni broj instanci (minimum, desired and maximum), raspoređuje ih na više AZ-ova, vodi health check, zamenjuje neispravne i sprovodi scaling polise.

Scaling polisa definiše kada i koliko da doda ili ukloni kapacitet. **Target tracking** cilja neku metriku, na primer 100 zahteva po targetu. **Step scaling** definiše procenat i korak skaliranja, na primer, ako CPU average metrika pređe 60%, dodati još jednu instancu, ako pređe 80%, dodati još dve itd. **Scheduled scaling** definiše schedule po kom će instance skalirati, recimo od 9h do 17h radnim danima držati 3 instance, van toga smanjiti na 1. **Health checks** sprovodi EC2 ili ELB, kao i **termination policy** koji definiše kako se bira koju instancu će da ugasi. **Cooldown / warm-up** daju novim instancama vremena da se zagreju pre sledeće odluke. **Mixed Instances/Spot** omogućava kombinovanje tipova instanci i kombinaciju On-Demand/Spot radi uštede. **Capacity Rebalancing** menja Spotove kad postanu „rizični“. **Warm pools** čuvaju prethodno pokrenute instance u pripravnosti radi bržeg skaliranja na gore.

Primer

Cilj ovog primera je da se na adresi Load Balancera prikazuje „Hello from …“, a kada se generiše opterećenje, ASG automatski podiže nove instance, ravnomerno ih raspoređuje i posle smanjenja opterećenja vraća se na minimalu.

Kako bi se ovaj primer realizovalo potrebne su sledeće AWS usluge:

- Application Load Balancer — podešen na HTTP
- Target Group
- Auto Scaling Group — min 1, desired 1, max 4 na 2 AZ
- Target tracking — po metrikama ALB (RequestCountPerTarget)

User-data skripta za Amazon Linux 2

Skripta instalira Apache HTTP server, napravi index.html sa Instance ID i AZ i pokrene servis:

```
#!/bin/bash
yum update -y
yum install -y httpd
TOKEN=`curl -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-ttl-seconds: 21600"`
INSTANCE_ID=$(curl -H "X-aws-ec2-metadata-token: $TOKEN" http://169.254.169.254/latest/meta-data/instance-id)
AZ=$(curl -H "X-aws-ec2-metadata-token: $TOKEN" http://169.254.169.254/latest/meta-data/placement/availability-zone)
echo "<h1>Hello from $INSTANCE_ID in $AZ</h1>" > /var/www/html/index.html
systemctl enable httpd
systemctl start httpd
```

Napomena: IP adresa 169.254.169.254 je AWS-ova zauzeta adresa koja sadrži metapodatke instance.

Koraci

VPC i mreža — Koristiti default VPC sa najmanje 2 public subneta u različitim AZ-ovima (npr. eu-central-1a i 1b).

Security group — Napraviti dve grupe, dozvoliti HTTP/80 iz 0.0.0.0/0. Za EC2 dozvoliti HTTP/80 samo od SG-a ALB-a (bolje nego 0.0.0.0/0). Ukoliko je potrebno, mogu se koristiti iste SG radi jednostavnosti primera.

Target Group — Napraviti tip Instances, protokol HTTP/80, health check path /.

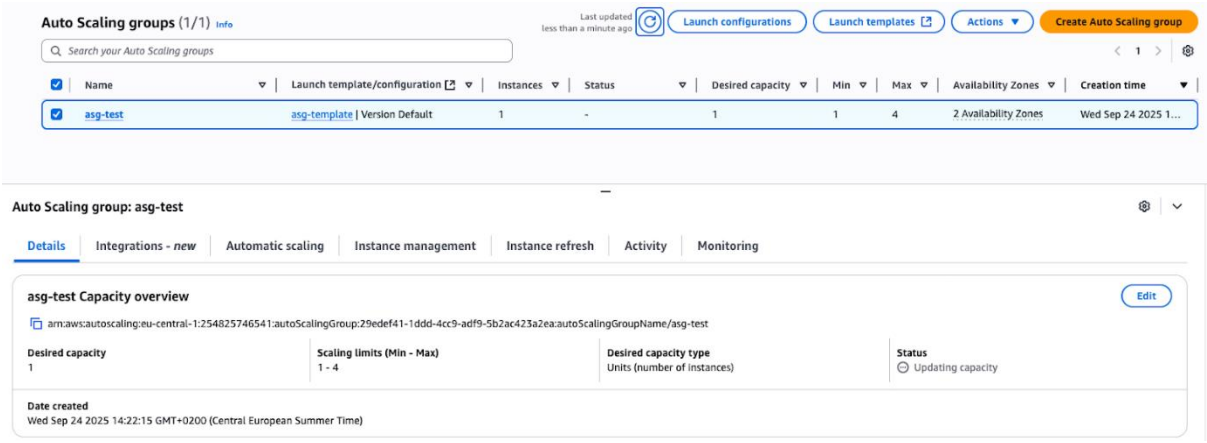
ALB — Application Load Balancer, public subnets (2 AZ), prikači target grupu kao default rule.

Launch Template — Amazon Linux 2023, t3.micro, SG (EC2), staviti user-data iznad, key pair po želji ili ne staviti uopšte.

Auto Scaling Group — Koristiti Launch Template, odabrati 2 AZ, min=1, desired=1, max=4, health check tip ELB, prikači target grupu. Povezati na postojeći load balancer.

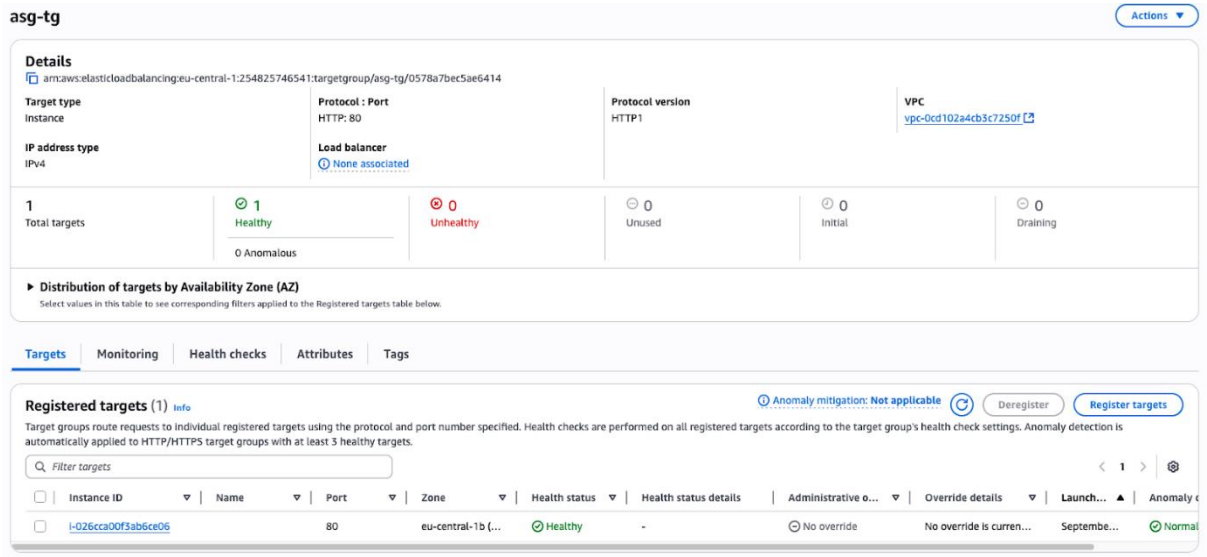
Scaling politika — Target tracking na metrikama ALB-a, odabrati Application Load Balancer request count per target, odabrati target group i ostaviti target na 50. To znači da će sistem ciljati ~50 zahteva po instanci; kad pređe, dodaće novi target (instancu), kad padne dovoljno, vrati se na manje.

Nakon svih ovih podešavanja potrebno je sačekati da Auto Scaling Group pokrene jednu EC2 instancu, što se može pratiti iz EC2 pregleda instanci (Slika 126).



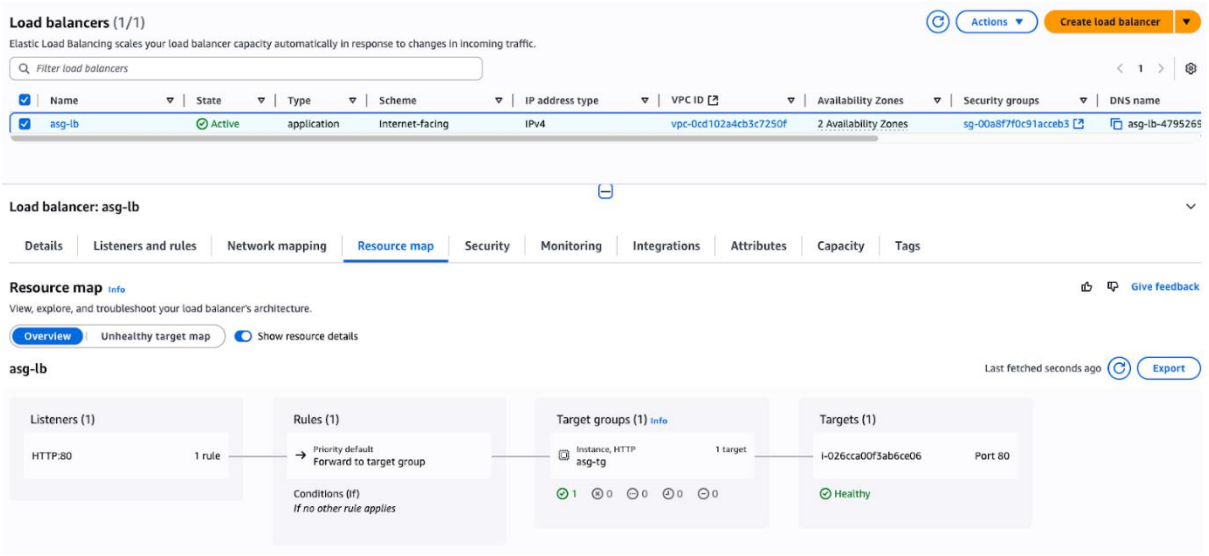
Slika 126 AutoScaling grupa podešava broj EC2 instanci

Nakon uspešnog pokretanja EC2 instance, ona će se pojaviti u okviru same Target Grupe kao dostupna (Slika 127).



Slika 127 Automatski registrovana nova instanca u okviru Target Grupe

Nakon toga, kroz Load Balancer meni se može pogledati Resource Map koji jasno pokazuje kako se kreće saobraćaj i kako je sve povezano (Slika 128).



Slika 128 Prikaz Resource Map kod Load Balancera

Pristupanjem DNS vrednosti Load Balancera dobijamo hostovanu stranicu na EC2 od strane Apache HTTP servera (Slika 129).

Slika 129 Prikaz hostovane stranice na EC2

Kako bi se postiglo automatsko skaliranje na više instanci, potrebno je povećati opterećenje nad Load Balancerom. Postoji više načina da se ovo uradi; jedan od njih je programski: pisanjem skripti koje će u paraleli da otvaraju stranicu; drugi je pomoću alata kao Apache Jmeter; treći je pokrenuti jednu novu EC2 instancu i pomoću nje stvoriti opterećenje. U svrhu primera koristiće se treći pristup.

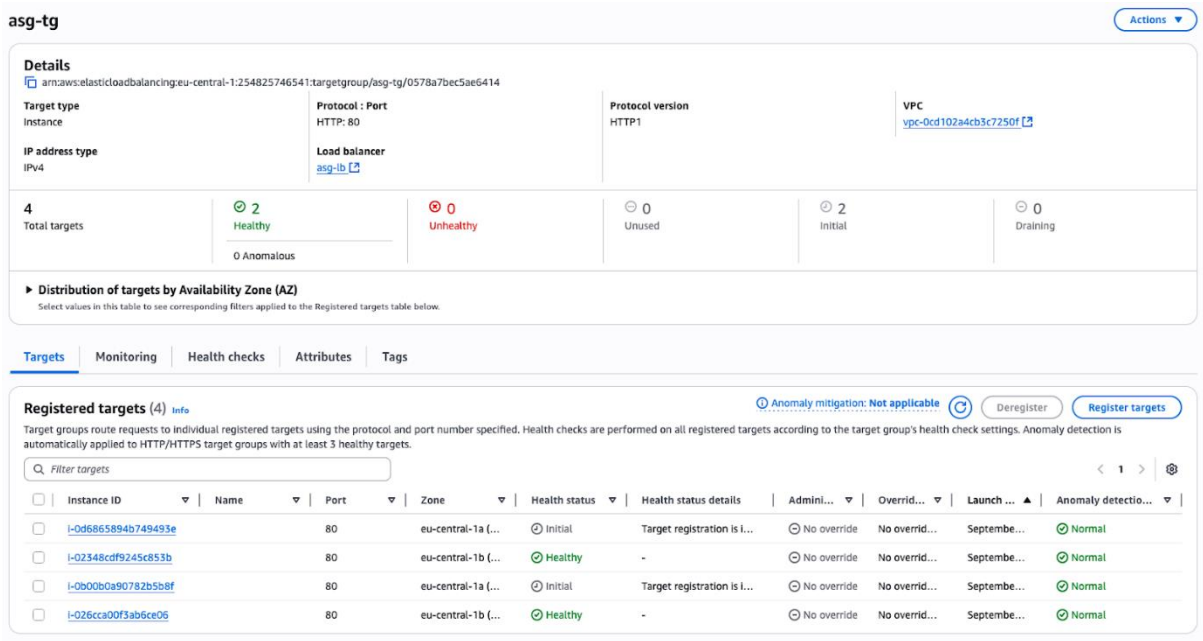
Potrebno je pokrenuti malu instancu, kao što je t3.micro, u istom VPC-u i instalirati ApacheBench koristeći sledeću komandu:

sudo yum install -y httpd-tools

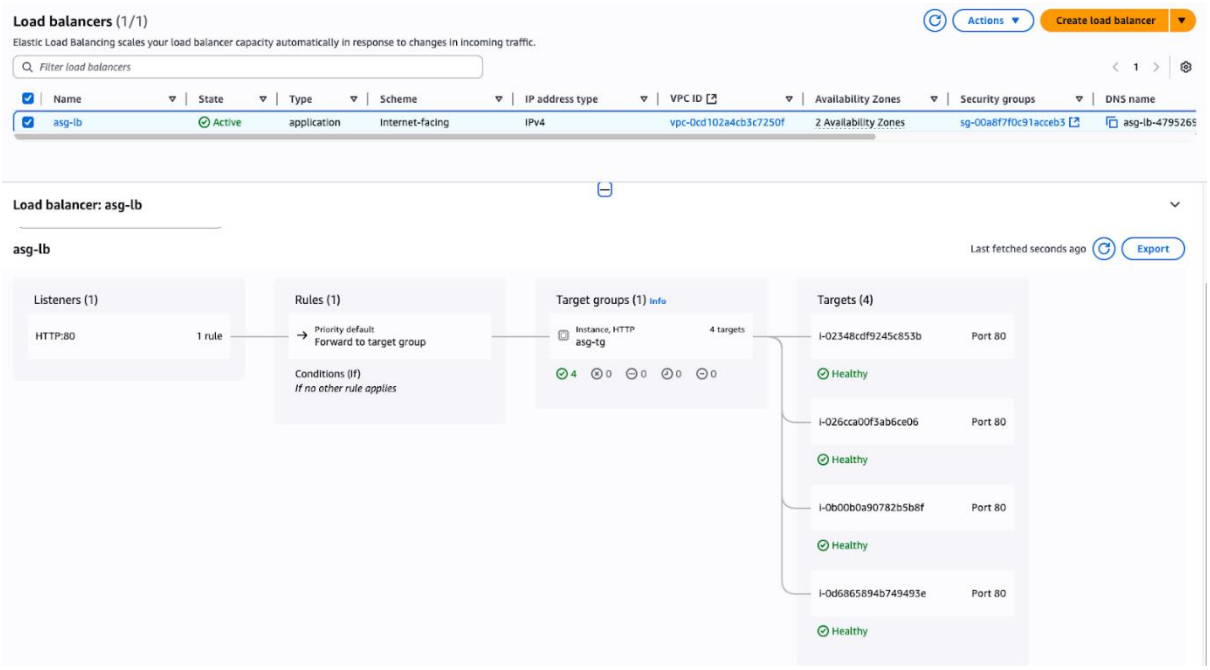
Zatim je potrebno opteretiti server, zameniti ALB-DNS-IME sa pravom vrednošću kopiranom iz Load Balancer opisa pod DNS.

ab -n 400000 -c 100 http://ALB-DNS-IME/

Dok test radi, otvoriti EC2/Auto Scaling u konzoli i posmatrati kako desired/actual capacity rastu, a u **Target Group** se vide nove instance (slike Slika 130 i Slika 131). Kada test prestane i prođe cooldown/warm-up, ASG će smanjiti broj instanci nazad.



Slika 130 AutoScaling Grupa dodaje nove instance



Slika 131 Load Balancer Resource Map prikazuje nove EC2 instance kao targete

Ako se koriste ALB metrike za scaling, Application Load Balancer request count per target je često intuitivniji od CPU-a za HTTP API, jer CPU može varirati sa tipom zahteva, dok je broj zahteva direktno vezan za opterećenje. Ako se koristi **CPU** za scaling, obavezno postaviti **instance warm-up** da novi čvorovi dobiju vremena da pokrenu servise pre sledeće odluke; u suprotnom može doći do konstantnog paljenja i gašenja EC2 instanci. **Security grupe** su odlična linija odbrane i preporučeno je da ALB bude otvoren ka internetu, a backend instance izložene samo ALB-u. Uvek testirati **scale-in** (ne samo scale-out); neprijatno iznenađenje je da kad se sve prošiti kako treba, a ne umanji posle, često problem bude cooldown, health ili policy koji je zaboravljen.

Zadatak

1. Pokrenuti ceo demo po koracima primera, u trenutku kada postoje 4/4 instance, simulirati kvar na jednoj od instanci tako što će se zaustaviti HTTP server komandom systemctl stop httpd. Isto tako simulirati kvar gašenjem EC2 instance koristeći komandu Terminate.
2. Podesiti AutoScaling Grupu da skalira na osnovu CPU load kada pređe 50% i testirati.

9. Bezbednost i oblak

Cilj ovog poglavlja jeste razvijanje svesti o dimenzijama bezbednosti u oblaku, upoznavanje sa najvažnijim pretnjama i merama zaštite, kao i sa organizacionim aspektima bezbednosti informacija u oblaku.
Po savladanom poglavlju, student će znati da:

- navede odgovornosti za bezbednost u oblaku u odnosu na model usluga
- nabroji najvažnije pretnje u oblaku
- objasni koncepte zaštite podataka u prenosu, obradi i mirovanju
- analizira proces detekcije upada

Bezbednost računarstva u oblaku je aktuelna tema od samih početaka. Računari su sami po sebi podložni napadima i malicioznim programima, a činjenica da se podaci kod oblaka obrađuju i čuvaju „tamo negde“ prirodno nameće dodatna pitanja, naročito kada su u pitanju javni oblaci.

U nastavku će biti obrađena tematika bezbednosti sa stanovišta dobavljača usluga u oblaku i sa stanovišta korisnika (klijenta) računarstva u oblaku. Bezbednost je upravo aspekt kod kojeg je ključan tzv. model deljene odgovornosti. Kod servera koji su lokalni *(on premise)* nema deljene odgovornosti — sve je na samoj instituciji ili eventualno spoljnim saradnicima koji se brinu o sistemi. U oblaku, stvari su malo komplikovanije. Ovde je mesto da se napomene suptilna razlika između termina bezbednost oblaka *(cloud security)* i bezbednost **u** oblaku *(security in the cloud),* koji se često mogu sresti u stranoj literaturi. Bezbednost oblaka se odnosi upravo na bezbednost infrastrukture, softvera, mrežne opreme i objekata koji čine oblak. Bezbednost u oblaku je bezbednost svega onoga što korisnik (klijent) stvara i koristi u oblaku i mehanizme zaštite, na primer, da podaci budu šifrovani s kraja na kraj, da se ograniči pristup nekom skladištu, da se definišu uloge korisnika itd.

U uvodnom poglavlju data je klasifikacija usluga u oblaku. Na jednom kraju spektra je infrastruktura kao usluga (IaaS), gde dobavljač stavlja na raspolaganje infrastrukturu, a sve ostalo je na klijentu (briga o operativnom sistemu, bibliotekama, arhitekturi aplikacije, bazi podataka, autentifikaciji itd.). U ovakvom scenariju na klijentu je i veliki deo odgovornosti vezane za bezbednost. Sve komponente koje su pod nadležnošću klijenta, jesu i sa strane zaštite pod njegovom odgovornošću. Na drugom kraju spektra usluga je softver kao usluga (SaaS). Ovde je na klijentu manji deo odgovornosti, prevashodno vezano za autentifikaciju i način korišćenja softvera. Sve ostalo: bezbednost infrastrukture, operativnog sistema, middleware-a (npr. baze podataka i biblioteka koje se nalaze između aplikacije i operativnog sistema), pratećih servisa i biblioteka jeste odgovornost dobavljača usluga.

U modelu deljene odgovornosti (tabela 9) svetlijom bojom (žutom) prikazani su elementi za koje je odgovoran korisnik (klijent), tamnijom (svetloplava) oni za koje odgovornost dele dobavljač i klijent i najtamnijom (plavom) elementi za čiju bezbednost je odgovoran dobavljač.

Bezbednost za cilj ima minimizaciju rizika — minimizaciju loših stvari koje mogu da se dese. Tematikom identifikacije i vrednovanja rizika bavi se posebna disciplina: procena rizika (risk assessment). Kada su rizici identifikovani, na raspolaganju je jedna od sledeće četiri opcije:

- Izbeći rizik. U informacionoj bezbednosti to obično znači da se sistem potpuno isključi. Tada više nema rizika, ali ni koristi od rada sistema.
- Ublažiti rizik. Rizik i dalje postoji, ali se preduzimaju dodatne mere kako bi se smanjila verovatnoća da će se nešto loše desiti ili umanjile posledice ako se ipak dogodi. Na primer, može se odlučiti da se skladište manje osetljivi podaci kako bi, u slučaju kompromitovanja, posledice bile manje ozbiljne.

- Preneti rizik. Plaća se nekome drugom da upravlja situacijom, tako da rizik postaje „tuđi problem“. Ovo je često slučaj sa oblakom, gde se na dobavljača usluga u oblaku prenose mnogi rizici upravljanja nižim nivoima sistema.
- Prihvatiti rizik. Nakon procene ukupnog nivoa rizika i koristi od nastavka aktivnosti, može se odlučiti da se formalno zabeleži postojanje rizika, dobije saglasnost svih zainteresovanih strana i nastavi dalje.

Tabela 9 Model deljene odgovornosti u oblaku

Lokalna infrastruktura (on premises)	Infrastruktura kao usluga (IaaS)	Platforma kao usluga (PaaS)	Softver kao usluga (SaaS)
Bezbednost pristupa	Bezbednost pristupa	Bezbednost pristupa	Bezbednost pristupa
Bezbednost aplikacije	Bezbednost aplikacije	Bezbednost aplikacije	Bezbednost aplikacije
Bezbednost middleware-a	Bezbednost middleware-a	Bezbednost middleware-a	Bezbednost middleware-a
Bezbednost operativnog sistema	Bezbednost operativnog sistema	Bezbednost operativnog sistema	Bezbednost operativnog sistema
Mrežna bezbednost	Mrežna bezbednost	Mrežna bezbednost	Mrežna bezbednost
Bezbednost virtualizovane infrastrukture	Bezbednost virtualizovane infrastrukture	Bezbednost virtualizovane infrastrukture	Bezbednost virtualizovane infrastrukture
Bezbednost fizičke infrastrukture	Bezbednost fizičke infrastrukture	Bezbednost fizičke infrastrukture	Bezbednost fizičke infrastrukture

Najvažnije pretnje u oblaku

Više svetski priznatih organizacija se bavi bezbednošću u oblaku i bezbednošću oblaka. Cloud Security Alliance (CSA) je vodeća u ovoj oblasti; to je organizacija posvećena istraživanju o bezbednosti oblaka, unapređenju svesti i sertifikaciji u ovoj oblasti. U svojim redovnim izveštajima CSA objavljuje koje su to najznačajnije pretnje, slično kao što OWASP radi za web-aplikacije. U izveštaju iz 2024[28] stoji:

1. Pogrešna konfiguracija. Može dovesti do brojnih problema: narušavanja poverljivosti ili gubitka podataka, smanjenja performansi aplikacije, kršenje zakonskih regulativa itd. Pogrešna konfiguracija može obuhvatati upravljanje tajnama, nekontrolisane portove, isključeno praćenje događaja, otvoren pristup skladištima itd.
2. Upravljanje identitetom i pristupom. Neodgovarajuća IAM strategija lako dovodi do narušavanja poverljivosti i prekida rada elemenata infrastrukture u oblaku, a posredno i do pada reputacije.
3. Nebezbedni interfejsi i API vode kao narušavanju poverljivosti, posebno bekenda aplikacije, curenja kredencijala i otkaza delova infrastrukture.
4. Neodgovarajući izbor i implementacija bezbednosne strategije. Strategija prethodi dizajnu i predstavlja visoki nivo apstrakcije, koji obuhvata brojne planove, vezane za sam model usluge, podršku više zona dostupnosti itd. Neodgovarajuća strategija komplikuje dizajn, duplira posao

[28] https://cloudsecurityalliance.org/artifacts/top-threats-to-cloud-computing-2024

i iziskuje naknadne intervencije, a može rezultovati u neslaganju sa regulativama i dovesti do pada reputacije kompanije.

5. Nebezbedni resursi treće strane. U razvoju aplikacija koje se izvršavaju u oblaku često se koriste biblioteke treće strane, naročito otvorenog koda. U pitanju je lanac snabdevanja (supply chain) kod kojeg je dovoljno da jedna karika (jedna linija koda) bude sporna da bi čitav sistem bio kompromitovan.
6. Nebezbedan razvoj softvera
7. Slučajno otkrivanje podataka oblaka

Identifikacija i klasifikacija podataka

Da bi se podaci uspešno zaštitili, neophodno je prvo utvrditi koji se sve podaci čuvaju i na kom su nivou poverljivosti.

Primer nivoa klasifikacije podataka

Svaka organizacija je drugačija, ali sledeća pravila pružaju dobru i jednostavnu polaznu tačku za procenu vrednosti podataka, a samim tim i rizika od njihovog kompromitovanja:

Nizak nivo ili javni podaci

Iako informacije u ovoj kategoriji mogu ili ne moraju biti namenjene javnosti, njihovo objavljivanje ne bi imalo značajan uticaj na organizaciju. Evo nekoliko primera:

- Javne IP adrese servera;
- Podaci iz aplikacionih logova koji ne sadrže lične podatke, tajne ili vredne informacije za napadače;
- Materijali za instalaciju softvera bez tajnih podataka ili drugih vrednih informacija za napadače.

Umeren nivo ili privatni podaci

Ove informacije ne bi trebalo da budu otkrivene van organizacije bez odgovarajućih ugovora o poverljivosti (NDA). U mnogim slučajevima (posebno u većim organizacijama), ovakvi podaci bi trebalo da budu dostupni unutar organizacije samo na principu „potrebno je znati". U većini organizacija, većina podataka spada u ovu kategoriju. Primeri uključuju:

- Detaljne informacije o dizajnu informacionih sistema, koje bi mogle biti korisne napadačima;
- Podaci o zaposlenima, koji bi mogli poslužiti napadačima za fišing ili pretekstualne napade;
- Rutinske finansijske informacije. Podaci poput naloga za kupovinu ili naknada za putne troškove mogu se koristiti, na primer, za zaključivanje o mogućim akvizicijama.

Visok nivo ili poverljivi podaci

Ove informacije su ključne za organizaciju, a njihovo otkrivanje moglo bi prouzrokovati značajnu štetu. Pristup ovim podacima treba da bude strogo kontrolisan uz višestruke zaštitne mere. U nekim organizacijama, ovaj tip podataka naziva se „krunski dragulji“. Primeri uključuju:

- Informacije o budućoj strategiji ili finansijski podaci koji bi mogli doneti značajnu prednost konkurenciji;
- Poslovne tajne, poput recepta za popularno gazirano piće:
- Tajne koje omogućavaju potpunu kontrolu, poput pristupnih kredencijala za infrastrukturu u oblaku:
- Osetljive informacije poverene na čuvanje, poput finansijskih podataka klijenata;

- Bilo koje druge informacije čije bi kompromitovanje moglo postati vest.

9.1. Regulativa i pomoć u klasifikaciji podataka

Zakoni i industrijski propisi mogu uticati na način klasifikacije određenih informacija. Na primer, Opšta uredba o zaštiti podataka Evropske unije (GDPR) postavlja različite zahteve za obradu ličnih podataka, pa bi se u ovom sistemu moglo odlučiti da se svi lični podaci klasifikuju kao podaci „umerenog“ rizika i u skladu sa tim bi se vršila zaštita. Standard sigurnosti podataka u industriji platnih kartica (PCI DSS) najverovatnije bi zahtevao da se podaci o vlasnicima kartica klasifikuju kao „visok“ rizik ako postoje u datoj infrastrukturi.

Takođe, postoje cloud servisi koji mogu pomoći u klasifikaciji i zaštiti, prikupljaju ili štite podatke kako bi razumeli sistem klasifikacije i pravilno ga primenjivali.

Zaštita podataka u oblaku

Šifrovanje predstavlja osnovnu metodu zaštite podataka – kako u konvencionalnom računarstvu, tako i u oblaku. Mogu se definisati tri kategorije podatka koji se štite:

- podaci u prenosu (na primer između klijenta i oblaka ili između delova oblaka)
- podaci u obradi (konkretni podaci koji se u datom trenutku obrađuju od strane procesora ili čuvaju u memoriji)
- podaci u mirovanju (podaci koji su sačuvanu na disku — u bazi podataka, u vidu fajla itd.)

Zaštita podataka u prenosu obuhvata kriptografske i druge tehnike kojima se štite podaci koji se prenose između klijenta i oblaka ili između delova infrastrukture oblaka. Kriptografske metode su ugrađene u protokole kao što su TLS i SSH, kao i VPN (Virtual Private Networks), kod kojih omogućavaju zaštitu poverljivosti, ali i autentifikaciju poruke i neporicanje. O detaljima će biti reči kasnije u jednoj od podsekcija.

Zaštita podataka u obradi predmet je tzv. poverljivog računarstva. Iziskuje se posebna hardverska podrška, a cilj je izolovati podatke između više stanara (tenanta) koji koriste istu infrastrukturu. Na taj način eventualni maliciozni kod ili bag ne bi mogao da izazove npr. curenje podataka, tako da jedan stanar dođe do podataka drugog stanara. Tehnologije koje se pominju u ovom kontekstu su Intel SGX/TGX, AMD SEV i IBM Z Pervasive Encryption.

Zaštita podataka u mirovanju

Šifrovanje se sprovodi nekim od standardnih algoritama. Međutim, pravi izazov je čuvanje ključeva, jer su podaci sigurni onoliko koliko je siguran ključ. Kod konvencionalnih infrastruktura za čuvanje ključeva koristi se HSM — hardverski modul. To je skupo rešenje, koje svoje varijante ima i u oblaku.

Najjednostavniji način upravljanja ključevima podrazumeva generisanje ključa, šifrovanje podataka tim ključem, skladištenje ključa u sistem za upravljanje ključevima (KMS) i zapisivanje informacije o tome koji ključ je korišćen za šifrovanje. Međutim, ovaj pristup ima dva problema:

- Veliko opterećenje KMS-a — Ako svaki fajl ima poseban ključ, KMS mora da skladišti ogroman broj ključeva i omogućava njihovo brzo preuzimanje.
- Sigurno brisanje podataka — Da bi se trajno izbrisali podaci, mora postojati sigurnost da je KMS trajno uklonio ključ ili se moraju prepisati svi šifrovani podaci, što može trajati satima ili danima.

Bolji pristup: Višeslojna enkripcija

Zbog ovih problema, koriste se dva nivoa ključeva:

- Ključ za enkripciju podataka (DEK) — direktno šifruje podatke.
- Ključ za enkripciju ključeva (KEK) — koristi se za šifrovanje DEK-a i skladišti se u KMS-u.

DEK ključevi se čuvaju uz šifrovane podatke, ali su „umotani" KEK-om. Kada je potrebno dešifrovanje, KEK ostaje u KMS-u, dok se DEK privremeno otvara i koristi, ali se nikada ne skladišti u nešifrovanom (otvorenom) obliku.

Upravljanje identitetima i pristupom

Koncept upravljanja identitetima i pristupom, poznat u originalu kao IAM — Identity and Access Management jeste jedan od stubova bezbednosti oblaka. Ukradeni i zloupotrebljeni kredencijali jesu jedna od najvećih pretnji u oblaku i uspostavljanjem efikasnog sistema upravljanja pristupom i definisanje politika dodela prava mogu umnogome da doprinesu bezbednosti. Dva osnovna elementa, identitet i pristup, upravo odgovaraju postupcima autentifikacija i autorizacije, te se može konstatovati da IAM služi realizaciji ova dva segmenta.

Kod tradicionalne infrastrukture, jednostavnije je definisati perimetar, granicu infrastrukture, i to pre svega u fizičkom smislu. Tačno je poznato koji serveri su povezani na koji svič, nalozi su centralizovani, mrežne barijere jasno odvajaju unutrašnju mrežu od spoljašnje i mnogo je jednostavnije pratiti korisnike i resurse. Ukoliko dođe do incidenta, jednostavnije je pristupiti infrastrukturi i moguće je lokalno dobiti informacije. Kod oblaka, sve se odvija onlajn, postoji princip deljene odgovornosti između klijenta i dobavljača usluga, a pristupi (i potencijalni napadi na resurse) zasnivaju se na API pozivima. U mikroservisnoj arhitekturi potrebno je definisati prava pristupa ne samo za korisnike, već i za servise.

Multifaktorska autentifikacija jeste obavezna kod oblaka. Podrazumeva da se za pristup koriste korisničko ime i lozinka uz jedan dodatni vid prijave, kao što je SMS kod, vremenski limitiran token i sl.

Zajednički ID-jevi (Shared ID) su identiteti kojima pristupa više osoba koristeći istu lozinku ili druge akreditive, poput ugrađenog root ili Administrator naloga na sistemu. Upravljanje ovim ID-jevima može biti izazov u okruženju oblaka, baš kao i u tradicionalnoj infrastrukturi.

Gde god je to moguće, svaki korisnik ili alat treba da ima sopstveni identifikator — ID, koji ne deli ni sa kim drugim. Mnogi sistemi omogućavaju korisnicima da preuzmu privilegovanu ulogu ili koriste poseban ID sa višim privilegijama za određene aktivnosti, poput sudo komande u Unix-olikim sistemima.

Ako ipak moraju da se koriste zajednički ID-jevi, važno je da se uvek može tačno identifikovati koja osoba (ili automatizovani alat) ih koristi. Kada se koristi deljeni ID, poput root naloga, sistem na kojem se koristi ne može razlikovati ko ga je tačno upotrebio. Zato je neophodno imati poseban proces i alat za izdavanje zajedničkih akreditiva i njihovu promenu nakon što se vrate. Ovi alati se obično nazivaju sistemi za upravljanje privilegovanim pristupom (PAM) ili upravljanje privilegovanim identitetima (PIM).

Federacija identiteta i Single Sign On (SSO) su važni koncepti u upravljanju identitetu. Federacija podrazumeva da korisnik (ili alat, servis) ima jedan identitet koji može koristiti na više sistema. Sami sistemi međusobno uspostavljaju poverenje i time se korisnik autentifikovan na jednom sistemu prihvata na drugom. Na primer, korisnici se autentifikuju putem Azure AD (Active Directory) i onda bez dodatnog prijavljivanja koriste servise SalesForce, AWS i tako dalje. Mnogi veb-sajtovi podržavaju SSO

autentifikaciju — korisnik se prijavljuje „preko Google-a“, tj. autentifikuje se sa svojim Google nalogom i pristupa sajtu, bez potrebe za kreiranjem posebnog naloga. Pored Google naloga, često se koriste i Microsoft i Facebook.

Treba naglasiti da, iako je autentifikacija pomoću lozinke i dalje veoma popularna, postoje tendencije prelaska na *passwordless* autentifikaciju — autentifikaciju bez lozinki. Umesto lozinki, koje su teške za pamćenje, koriste se biometrija ili uređaji za čuvanje kriptografskih podataka. Više o FIDO2, jednom od standarda iz ove oblasti, može se pogledati na https://fidoalliance.org/fido2/

9.2. Uloge

Mnogi provajderi cloud usluga nude **uloge (roles)** ili **pouzdane profile (trusted profiles)**, koji su slični **zajedničkim identitetima (shared IDs)** po tome što korisnik preuzima određenu ulogu, obavlja akcije koje ta uloga dozvoljava, a zatim napušta ulogu. Ovo se malo razlikuje od tradicionalne definicije uloge, gde je uloga skup **dozvola (permissions)** ili **prava pristupa (entitlements)** dodeljenih određenom korisniku ili grupi.

Za razliku od zajedničkog identiteta, rola nije pun identitet, to je status koji može preuzeti drugi identitet koji ima dozvolu da koristi datu rolu. U tom slučaju, identitet dobija privremene kredencijale za datu rolu.

Prednosti korišćenja rola su:

- **Dodatni sloj bezbednosti** — Korisnici ili servisi moraju eksplicitno preuzeti ulogu za obavljanje privilegovanih operacija, čime se prati princip **najmanjih privilegija (least privilege)**.
- **Privremene privilegije** — Većinu vremena korisnik **ne može** obavljati visoko privilegovane radnje, osim ako **preuzme odgovarajuću ulogu**.
- **Praćenje aktivnosti** — Sistem beleži svaki zahtev za preuzimanje uloge, omogućavajući administratorima da prate **ko je imao koju ulogu u određenom trenutku** i kako je koristio svoje privilegije.

Uloge, opet, nisu samo za ljude. Osim korisnika, i **komponente u okruženju oblaka** mogu preuzeti role. Na primer:

- **Virtuelne mašine (VMs)** mogu preuzeti određenu rolu prilikom kreiranja i koristiti njene privilegije za obavljanje specifičnih zadataka.
- **Serverless funkcije (npr. AWS Lambda, Azure Functions)** mogu imati dodeljene role koje im omogućavaju pristup drugim servisima **bez fiksnih lozinki ili API ključeva**.

Iako role podsećaju na grupe, postoje ključne razlike. Grupa obuhvata određene korisnike, ali sama grupa ne sadrži informaciju šta joj je omogućeno, tj. kakav pristup kojim resursima ima. Takođe, grupa je fiksna. Rola, sa druge strane, sadrži definiciju šta omogućava, a posebno se povezuje sa korisnikom. Na kraju, role, za razliku od grupa, mogu preuzeti i servisi.

IAM ciklus

Za rukovanje procesima vezanim za upravljanje identitetom i pristupom, potrebno je definisati sve faze aktivnosti vezanih za IAM, počevši od samog inicijalnog zahteva korisnika. U tom smislu može se govoriti o životnom ciklusu IAM.

Entitet (korisnik ili servis) upućuje zahtev aplikaciji. Pojedine aplikacije mogu da dozvole pristup i bez posebnog naloga, anonimno, ali je preporuka da se uspostavi neki osnovni oblik identiteta. Pojedini sistemi dodeljuju podrazumevano jedan poseban – guest nalog, za korisnike koji nemaju definisan identitet (na primer, ovu opciju imaju Windows, Moodle). Ako bi želeo da kreira identitet, korisnik može samostalno da se registruje ili će morati da prođe deo provere, koju će obaviti ovlašćeno lice. To lice može biti administrator kompanije, koji će proveriti određene podatke i „ručno“ odobriti kreiranje identiteta. Kada se kreira identitet, on dobija određena prava. U narednom koraku se vrši autentifikacija, proverava se ko je korisnik (uz pomoć imena/lozinke, uz pomoć dvofaktorske autentifikacije itd.). Vrši se određena aktivnost, koja se prati — monitoring je neraskidivo povezan sa IAM. Na kraju se uklanja dozvola i završava aktivnost.

Različiti dobavljači usluga u oblaku imaju različite nazive usluge IAM. Kod Amazona je u pitanju AWS IAM, kod Microsoft Azure-a to je Azure Active Directory, a kod Google-a i IBM-a to je Cloud IAM.

O AWS IAM-u je bilo više reči u Poglavlju 3.4.5.

Mrežna bezbednost u oblaku

Već je naglašeno da je kod oblaka nemoguće uspostaviti jasan perimetar, kao što je to kod konvencionalnih mreža. Perimetar podrazumeva liniju koja jasno odvaja unutrašnji, zaštićeni i spoljašni deo mreže ili sistema. Kod oblaka, granice poverenja nisu ni jasne, niti očigledne. Na primer, ukoliko se koristi baza podataka kao usluga (DBaaS), da li se ona nalazi unutar ili van perimetra? Ako postoji više instalacija u više zona dostupnosti, koja od njih se računa kao „u perimetru“, a koja van? Na sve to, treba imati u vidu koji model isporuke se razmatra. Infrastruktura kao usluga (IaaS) je najbliža konvencionalnim okruženjima, sa visokim ingerencijama klijenta u postavljanju, konfiguraciji i zaštiti mreže. Na drugom kraju spektra su bezserverske instalacije (serverless), koje uključuju AWS Lambdu, OpenWhisk i druge, kod kojih kontrole mreže često uopšte ne postoje.

Koncepti i mehanizmi zaštite u oblaku

Jedan od najvažnijih koncepata, koji važe uopšteno, ali i u kontekstu umrežavanja, jeste nulto poverenje (Zero Trust). Ono podrazumeva model koji se zasniva na principu da nijedan korisnik, uređaj ili servis ne bi trebalo automatski da bude smatran pouzdanim, čak i ako se nalazi unutar mrežne organizacije. Stalna i dosledna primena nultog poverenja znači da određivanje pripadnosti perimetru više nije ključno, jer se svi proveravaju. (Na primer, čak i veza sa „localhost-om“ se šifruje.) Takođe, mreža se deli na veći broj mikrosegmenata; to je mnogo jednostavnije u okruženju oblaka, nego u fizičkim tradicionalnim mrežama. Različite komponente se odvajaju u posebne podmreže i štite barijerama.

Primena belih i crnih lista je koncept koji se takođe intenzivno koristi i u oblaku. Princip bele liste jeste takav da se pristup odobrava samo entitetima koji su eksplicitno navedeni u listi. Svim ostalim je zabranjen. Kod crne liste, pristup je podrazumevano otvoren, a brani se eksplicitnim unošenjem spornih entiteta u crnu listu. Konvencionalne liste se formiraju na osnovu IP adresa, ali kod modernih mreža (kod kojih korisnici često menjaju adrese) takav pristup nije pogodan. Kod oblaka, dodatno, treba imati u vidu da se mnogi sistemi generišu i uništavaju dinamički, tako da je teško definisati liste sa konkretnim adresama, već treba uneti određen blok adresa koji je dozvoljen. Dalje, treba imati u vidu da vidljiva adresa klijenta često jeste adresa proksi-servera. Ako na primer zabranimo tu adresu, time ćemo potencijalno zabraniti pristup mnogim drugim korisnicima, koji su korektni, ali pristupaju preko istog proksija. Treba napomenuti, da pored IP adresa liste u principu mogu koristiti i protokole, geolokaciju, identitete itd.

Ovde je prikladno pomenuti i DMZ (demilitarizovanu zonu). Ovaj koncept je standardan kod konvencionalnih mreža, a koristi se i kod oblaka, dok ga pojedini modeli koriste podrazumevano (nema potrebe dodatno podešavati). Osnovni koncept jeste da se u datoj zoni postavljaju servisi koji su dostupni javnosti, uz osnovno filtriranje zahteva, npr. load-balanser, web-server sa statičkim sadržajem, proksi. Već drugi servisi, backend, kao što su aplikativni server, baza podataka i drugi, nalaze se iza naredne barijere i nisu neposredno dostupni spolja, već servisi iz DMZ-a mogu da im pristupaju, uz kontrolu pristupa.

Proksi-serveri su posrednici u komunikaciji između klijenta i servera. Postoje dve osnovne vrste: forward proxy i reverse proxy. Prevashodna namena jeste optimizacija saobraćaja: ukoliko jedan klijent zahteva sadržaj koji je drugi klijent već zahtevao, taj sadržaj je keširan u forward-proksi serveru i odatle se brže dostavlja, bez potrebe da se ponovo traži od originalnog servera. Dakle, forward-proksi je zajednički za više klijenata koji ga koriste. Kod reverse-proksija klijent šalje zahtev serveru sa sadržajem i proksi nije vidljiv za njega (klijent ne mora ni na koji način da ga definiše kod sebe). Zahtev preuzima proksi-server i onda odlučuje kojem će serveru sa sadržajem da ga prosledi. Na ovaj način rasterećuju se serveri. Klijent ima doživljaj posete jednom te istom serveru, iako zahtevi mogu završiti na različitim serverima sa sadržajem, odnosno uslugom. Sa bezbednosnog aspekta, forward-proksi se obično upotrebljavaju tako što se definišu pravila pristupa, npr. koji URL su dozvoljeni ili koji tipovi fajlova se mogu preuzimati itd. Reverse-proksi sami po sebi pružaju zaštitu time što ublažavaju eventualne probleme sa ranjivim protokolima, tako što ne prosleđuju direktno zahteve, već se potencijalni napad završava na njima. Na primer, ako postoji ranjivost u TLS protokolu koja daje pristup podacima u memoriji servera, ne ostvaruje se pristup „pravim“ podacima — onome što backend server ima, već samo memoriji proksija. Takođe, proksi može odbacivati pogrešno formatirane pakete i dr.

Zaštita podataka u prenosu

Standard u zaštiti podataka u prenosu jeste TLS (ranije SSL) — Transport Layer Security. TLS štiti poverljivost i integritet podataka, a omogućava i potvrdu identiteta servera (a po potrebi i klijenta). U nastavku će biti sažeto objašnjen princip rada TLS-a. Za više detalja vredi pogledati (Stallings, 2014).

- Postoje dva ključa, jedan je privatni i nalazi se na serveru, ne otkriva se. Kod oblaka postoji poseban menadžer ključevima koji ga čuva. Javni ključ je objavljen, dostupan i koristi se za šifrovanje.
- Klijent šalje saobraćaj šifrovan javnim ključem, a server ga dešifruje privatnim ključem.
- Za proveru integriteta koriste se posebni, dodatni algoritmi, koji se paralelno koriste uz šifrovanje.
- Klijent preuzima javni ključ iz sertifikata. Sertifikat sadrži različite informacije o serveru, kao i javni ključ, koji je potpisan od strane tela, koje je od poverenja. Tako klijent može da „zna“ da sertifikat pripada baš tom serveru i da slobodno može da koristi javni ključ iz sertifikata. Na ovaj način se potvrđuje identitet servera.
- Ukoliko je potrebno, i klijent može da poseduje sertifikat, kojim potvrđuje svoj identitet. Ovo je ređi scenario.

Protokol TLS ne koristi jedan utvrđeni algoritam za šifrovanje, već se server i klijent dogovaraju o skupu različitih algoritama koje će koristiti. Vremenom, pojedini algoritmi se mogu pokazati kao nedovoljno bezbedni i važno je takve algoritme izuzeti iz izbora. Na kraju, treba koristiti poslednje verzije samog TLS protokola (trenutno to je 1.3).

VPN i bastion host su važni mehanizmi za pre svega administrativni pristup zaštićenim, izolovanim delovima mreže, kao što je VPC. VPN podrazumeva komunikaciju kroz zaštićeni kanal, slično kao što

radi TLS, samo što radi na nižem nivou od aplikacija. Potrebno je konfigurisati VPN pristup na serveru u internoj mreži kojoj se pristupa, kao i na klijentu. Bastion-host se koristi u situacijama kada se pored bezbednosti zahtevaju dodatne funkcije, kao što je snimanje saobraćaja. Čitava mreža se podešava tako da saobraćaj mora proći kroz bastion-host. Samim svojim posredovanjem već vrši smanjenje površine za napade (slično kao reverse proksi), a može vršiti i nadzor i prekid sesija.

Detekcija i odgovor na napade

Primena mehanizama zaštite jeste esencijalno za smanjenje rizika. Međutim, savršena bezbednost ne postoji, pojavljuju se napadi koji „prolaze“ i izuzetno je važno ustanoviti da je došlo do napada i primeniti mere za otklanjanje uzroka napada i njegovih posledica. U oblaku, kako je navedeno pri samom početku poglavlja, postoji sistem deljene odgovornosti. Za praćenje zapisa o aktivnostima (logova) i eventualnu detekciju anomalija i bezbednosnih pretnji često je odgovoran dobavljač usluga, naročito ako je reč o Softveru kao usluzi. Sa druge strane, na nivou korisnika, u oblaku su na raspolaganju različiti izvori zapisa, komandi i API poziva kojim se mogu dobiti informacije o bezbednosno interesantnim događajima. Na nižim nivoima, kod infrastrukture kao usluge, korisnik na raspolaganju ima i mehanizme operativnog sistema, midlvera (middleware) i same aplikacije. „Samo“ je potrebno definisati šta se sve prati u moru podataka koji se konstantno generišu. Ne može se pratiti sve, već je neophodno izdvojiti relevantne elemente koji se prate, koji su verodostojni indikatori upada.

Sam dobavljač usluga u oblaku pruža sopstvene mehanizme i alate oblaka kojim se mogu pratiti događaji i okidati određeni alarmi. Svaki dobavljač, očekivano, koristi sopstveni specifični naziv za ovakve alate. Kod Amazona je to AWS CloudTrail, kod Microsoft-a je Azure Monitor, kod IBM-a Cloud Activity Tracker itd. Ovi alati prate API pozive koje obavljaju korisnici, usluge, komandna linija itd. Logovi koje čuva pomažu u bezbednosti i praćenju. Uobičajena praksa je da se koriste i za automatizaciju, npr. ako neko obriše S3 bucket, šalje se alarm. Dalje, ako se npr. uoči intenzivan I/O, a pristup aplikaciji nije u porastu, to može biti signal da je reč o rensomver (ransomware) napadu: maliciozni program šifruje ceo disk i tako se generiše visok I/O.

Pored ovih globalnih alata koji se konfigurišu na nivou korisnika/naloga, različite usluge mogu imati i sopstvene mehanizme za praćenje i izveštavanje; na primer, Kubernetes, baze i druge usluge imaju svoje alate koji se mogu konfigurisati, preusmeriti svoje logove, centralizovati itd. Na kraju, sama aplikacija može generisati svoje sopstvene logove, na osnovu kojih se može vršiti bezbednosna analiza i revizija.

Kada je reč o virtualnim mašinama (kod AWS EC2 instanci), na nivou operativnog sistema korisnici mogu definisati raznovrsne posebno konfigurisane alate za praćenje događaja, detekciju upada putem mreže i slično.

Važno je ustanoviti kakva tačno regulativa važi za dati sistem i datu aplikaciju i u kojem je zakonskom okviru i sa kojim industrijskim zahtevima. Upad nije „samo“ stvar administratora i klijenta, to je krivično delo i jako je važno imati dovoljno informacija da bi se uspešno izvršila forenzika i sprovela istraga. To uključuje format logova, šta sadrži od informacija, kao i koliko dugo se čuvaju.

Jedan od mehanizama praćenja jeste vezan za integritet fajlova. Postoje fajlovi koji se često menjaju i to je potpuno regularno i očekivano, npr. log-fajlovi. Sa druge strane, sistemski fajlovi ne bi trebalo da se menjaju često, pojedini gotovo nikad. Pod Linuxom, fajl /etc/passwd sadrži po jednu liniju za svakog korisnika. Taj fajl se menja retko, samo kada se korisnik dodaje ili uklanja i eventualno kada se promeni neko specifično podešavanje korisnika. Prema tome, korisno je ispratiti promene ovog fajla i proveriti ih. Isto važi za faktički sve fajlove u direktorijumu /etc. Dobro je ispratiti svaku promenu; velika većina

jesu legitimne, ali revizijom se mogu pronaći anomalije — da je npr. otvoren širok pristup FTP serveru, a da to niko od zaposlenih i administratora nije inicirao.

Lično („peške”) čitanje logova je nepraktično i u mnogim slučajevima potpuno neprimenjivo. Stoga su na raspolaganju različiti log-parseri koji služe da iščitaju logove iz više izvora, izvuku potrebne informacije i pronađu korelaciju i prezentuju odgovarajući sažetak eventualnog događaja. Ipak, čak i kada automatizacija identifikuje sumnjive aktivnosti i generiše alarme, konačnu odluku donosi čovek, jer kontekst je ključan za tačno tumačenje događaja.

Primer

Imamo web-aplikaciju na Apache web-serveru, koji generiše sopstvene logove. Imamo log-fajl na serveru (Linux) koji evidentira uspešne i neuspešne autentifikacije (/var/log/auth.log). I na kraju, tu je sistem za detekciju upada sa svojim log fajlom.

Prvo, u log-fajlu web-servera se pojavljuje red:

```
192.0.2.15 - - [07/Apr/2025:12:14:01 +0000] "GET /index.php?page=../../../../etc/passwd HTTP/1.1" 200 1245
```

Ovo je *path-traversal* napad: napadač putem specifične, neregularne URL adrese pokušava da dođe do liste naloga (kroz prikaz fajla /etc/passwd).

Nešto kasnije, u fajlu /var/log/auth.log se pojavljuje red:

```
Apr  7 12:17:43 myserver sshd[2971]: Accepted password for www-data from 192.0.2.15 port 54832 ssh2
```

Ovaj red govori da se korisnik sa nalogom www-data (a to je sistemski nalog pod kojim radi process web-servera) uspešno prijavio. Ovo nije regularan događaj, jer se sistemski korisnici ne prijavljuju.

Na kraju u logu sistema za detekciju upada pojavljuje se red:

```
[**] [1:2010937:2] ET POLICY PE EXE or DLL Windows file download [**]

[Priority: 1]

04/07-12:19:00.147202 192.0.2.15 -> 192.168.1.10
```

To znači da se dogodila akcija koja je kršenje politike sistema: preuzimanje izvršnog fajla.

Podrazumevano, samo bi sistem za detekciju upada generisao upozorenje. Poslao bi ga na mejl ili SMS-om. Prva dva izvora podrazumevano ne reaguju na logove, samo ih generišu. Ako bi se postavili parseri koji bi okidali upozorenja i za logove web-servera i za autentifikacioni log (auth.log), onda bi se mogla dobiti ukupno tri alarma. Korelacijom sva tri alarma, dobija se slika kompletnog napada.

Korelacija se može vršiti pisanjem posebnih skripti, koje će na osnovu regularnih izraza dati određene rezultate detekcije. Za veliki broj logova i velike količine informacija, razvijena su SIEM rešenja. SIEM znači Security Information and Event Manager. U pitanju je čitav sistem koji ima za zadatak da na osnovu raznovrsnih logova, kojima se opskrbljuje (direktno ili posle prethodne agregacije ili filtriranja) izvrši različite analize i dâ rezultate korelacije. Sam SIEM može biti pokrenut u oblaku ili van njega. Čak, mnogi dobavljači usluga u oblaku pružaju upravo SIEM usluge. Primer je AWS Security Hub, koji vrši funkcije SIEM-a i bezbednosne orkestracije, uz integraciju sa drugim AWS uslugama — GuardDuty,

Macie[29], Inspector i korišćenje CloudWatch-a i CloudTrail-a za detekciju i korelaciju. Na kraju, on pruža automatizovane akcije za rešavanje problema. Drugi ovakvi primeri jesu Microsoft-ov Azure Sentinel, koji koristi mašinsko učenje za automatsku detekciju i korelaciju, kao i poseban KQL upitni jezik za naprednu analizu podataka; zatim Google Chronicle, kod kog se može izdvojiti forenzička analiza uz vraćanje podataka unazad i istraživanje pretnji, zatim Splunk Cloud, koji se može integrisati u bilo koji oblak itd.

Kao i kod drugih alata, i kod alata za detekciju može se govoriti o rešenjima koja funkcionišu na mrežnom nivou i o onima koji funkcionišu na krajevima (endpoint). I jedan i drugi pristup imaju svoje prednosti i nedostatke. Ako se koristi mrežna detekcije, to znači da će se koristiti posvećena mašina koja proverava sav mrežni saobraćaj, sa posebnim IDS rešenjem, koja može biti postavljena unutar VPC-a. U konvencionalnim mrežama, potrebno je da se sav saobraćaj koji je potrebno pratiti, na neki način usmerava preko mašine sa IDS-om, što se rešava korišćenjem haba, *mirroring* porta na sviču (koji duplira saobraćaj jednog porta na zadati port) itd. Kod oblaka, pošto je reč o virtualizaciji mreže, nema potrebe za podešavanjem mreže na niskom nivou. Kod AWS-a, na primer, postoji servis Traffic Mirroring, koji omogućava upravo praćenje saobraćaja sa zadatih mrežnih kartica.

Važno je dobro proveriti kako se naplaćuju usluge vezane za bezbednost (kao i bilo koje usluge inače) da troškovi ne bi izmakli kontroli. Neke usluge koje se koriste za bezbednost se naplaćuju vremenski, druge prema količini podataka. Sve to treba imati u vidu pri kreiranju modela troškova i finansijske konstrukcije zaštite, kao i kompletne razvijane infrastrukture.

Kada je napad detektovan, trebalo bi da postoji tim koji reaguje u takvim situacijama — *cloud security incident response team* — CSIRT tim.

To je specijalizovana grupa unutar organizacije koja se bavi bezbednosnim incidentima koji utiču na sisteme i usluge zasnovane na oblaku. Ovaj tim se fokusira na otkrivanje, obuzdavanje, uklanjanje i oporavak od sajber napada u okruženju oblaka, sa ciljem minimizacije štete i vraćanja normalnog poslovanja.

Ključne odgovornosti Cloud CSIRT-a:

- Otkrivanje incidenata: Identifikacija potencijalnih bezbednosnih upada u oblaku pomoću različitih alata za praćenje i analizu.

- Analiza incidenata: Istraživanje i procena prirode i obima otkrivenih incidenata.

- Obuzdavanje: Preduzimanje koraka za ograničavanje uticaja incidenta, kao što je izolacija kompromitovanih sistema ili usluga.

- Uklanjanje: Eliminacija malicioznih aktera i malvera sa pogođenih sistema.

- Oporavak: Vraćanje pogođenih sistema i usluga u njihovo normalno stanje.

[29] AWS Macie je cloud-native servis koji koristi mašinsko učenje za otkrivanje, klasifikaciju i zaštitu osetljivih podataka u AWS okruženju. Specifično je dizajniran za prepoznavanje ličnih podataka, kao što su brojevi kreditnih kartica, brojevi socijalnog osiguranja, ime i prezime i drugi osetljivi podaci koji se nalaze u Amazon S3 (Simple Storage Service) i drugim skladištima podataka u AWS-u.
Glavna funkcionalnost AWS Macie-a je automatsko prepoznavanje, označavanje i zaštita tih podataka, čime se pomaže organizacijama da održavaju usklađenost sa regulativama kao što su GDPR, CCPA i druge zakonske regulative vezane za zaštitu privatnosti podataka.

- Post-incidentna analiza: Analiza incidenta radi identifikovanja slabosti u bezbednosnim kontrolama i implementacija poboljšanja kako bi se sprečili budući incidenti.

- Komunikacija i saradnja: Koordinacija sa drugim timovima unutar organizacije i, po potrebi, sa eksternim zainteresovanim stranama, kao što su dobavljači usluga u oblaku ili organi za sprovođenje zakona.

Kada se upad dogodi, važno je posedovati rezervnu kopiju podataka (bekap). Pošto je potencijalno čitav sistem kompromitovan, bekap mora biti čuvan u potpuno odvojenom okruženju, dakle nipošto u okviru istog naloga u oblaku.

Pitanja za proveru i obnavljanje znanja

1. Kod kog modela usluge dobavljač usluga u oblaku ima najviše odgovornosti za bezbednost?

2. Koje su to četiri opcije upravljanja rizikom?

3. Navedite po jedan primer javnih, privatnih i poverljivih podataka.

4. Šta podrazumeva federacija identiteta?

5. Šta se podrazumeva pod korelacijom bezbednosnih događaja?

10. Raspoređivanje i upravljanje u oblaku

Cilj ovog poglavlja jeste upoznavanje studenata sa praksama raspoređivanja i korišćenja izvornih rešenja u oblaku, kao što su serverless i infrastruktura kao kod.
Po savladanom poglavlju, student će moći da:

- navede strategije raspoređivanja u oblaku
- postavi osnovne elemente alata za konfiguraciju
- konfiguriše jednostavnu infrastrukturu kroz kod
- objasni svojstva serverless servisa i konfiguriše ih

10.1. Strategije i modeli raspoređivanja

U produkcionim sistemima, način na koji se isporučuje nova verzija softvera često je važniji od same verzije. Strategije raspoređivanja (Deployment strategies) su šabloni kojim se nova verzija softvera uvodi uz minimalan rizik i predvidivo ponašanje. Model raspoređivanja (Deployment model) je tehnika tog šablona kod izabrane platforme. AWS nudi više dokazanih strategija, kao što su rolling, blue/green, canary, all-at-once, uz servisne mehanizme koji ih čine praktičnim u EC2/ECS/EKS/Lambda okruženjima, kao i pomoćne alate, kao Route 53, AppConfig, za kontrolu saobraćaja i odvojeno dodeljivanje isporuke od objave funkcionalnosti (kontrolu saobraćaja ka verzijama).

Kratak pregled strategija:

- **All-at-once (recreate)** — Aktivira novu verziju u jednom potezu i odmah preusmerava sav saobraćaj — najbrža, ali i najrizičnija opcija, ima kratak downtime, ali postoji rizik da se aplikacija neće ponašati kako treba zbog naglog skoka ka novoj verziji.
- **Rolling** — Postepeno zamenjuje stare instance novim u talasima; u svakom koraku deo kapaciteta je „nov“, deo je „star“ pa je rizik niži. Na primer, menja 2 po 2 instance, ali ovaj pristup zahteva da su stara i nova verzija aplikacije kompatibilne i ne dovode do problema.
- **Blue/Green** — Uvodi paralelno okruženje (green) uz postojeće (blue), testira se pod realnim uslovima, a zatim jednim korakom preusmeri promet. Rollback procedura je prebacivanje nazad na blue.
- **Canary** — Izlaže mali procenat korisnika (na primer, postepeno 5%, pa 20% itd.) novoj verziji, posmatra metrike i, ako je sve u redu, povećava udeo do 100%.

U praksi, rolling ublažava rizik bez dupliranja celog okruženja, blue/green i canary daju najbolji kvalitet rollbacka jer stara verzija ostaje netaknuta i u spremnom stanju (warm).

Za EC2 aplikacije tipično se koristi in-place, tj rolling ažuriranje koji zamenjuje verzije sa postojećim instancama po grupama *(batch*-evima). Blue/Green pokrene novi skup instanci (green) i zameni stari skup (blue) preusmravanjem saobraćaja iza Load Balancera, sa mogućnošću brzog povratka ukoliko dođe do problema sa novom verzijom.

Servis Route 53 pomaže ako ima više regiona ili postoje nezavisni blue/green ulazi; tada mogu da se koriste težinski Route 53 DNS zapisi koji omogućavaju da se podeli saobraćaj, na primer 90/10, tako što se postepeno dodeljuje saobraćaj novim instancama, pa bi u slučaju problema samo deo korisnika (10%) osetio taj problem i time se sprovede canary (postepen prelaz) ili prelaz na novu stranu.

ECS podržava *rolling* kroz Service scheduler, ali i blue/green kroz CodeDeploy tako što se servis duplira (task set „green“), validira, pa se saobraćaj (ALB target-group) prebacuje po planu (all-at-once, linear, canary), uz automatski rollback ako metrike pokažu problem.

Lambda podržava canary/linear/all-at-once preko verzija i aliasa. Alias omogućava kontrolu nad tim koja je verzija aktivna u određenim okruženjima (test, prod). Alias može da prosledi na primer 5% saobraćaja ka novoj verziji, pa se mera uvećava dok ne stigne do 100%, s mogućnošću trenutnog povratka na staru verziju.

Čak i kad je nova verzija aplikacije objavljena, objava funkcionalnosti može biti odložena ili ograničena kroz *feature flags*[30]. **AWS AppConfig** uvodi kontrolisanu, validiranu promenu konfiguracije i zastavica (flagova) sa mogućnošću ciljanja dela korisnika ili okruženja (više varijanti, pravila, validator), što pomaže da se u produkciji uključi/isključi funkcija bez ponovnog raspoređivanja (deploymenta).

Na ALB-u (Application Load Balancer) se saobraćaj prema više target grupa može težinski deliti (weighted forward) radi canarya unutar istog regiona. Na Route 53 nivou, weighted routing deli saobraćaj između različitih endpointa i regiona, pogodno za cross-region canary/blue-green.

Svaku strategiju je preporučeno uvezati sa CI/CD[31] procedurama i štititi guardrailovima, kao što su CloudWatch alarmi i health check — 4xx/5xx, p95/p99, error rate i obavezni automatski rollback kad kriterijumi nisu ispunjeni. Za Lambda/ECS, CodeDeploy nudi gotove profile za Canary, Linear ili AllAtOnce[32].

Najčešći uzrok „neprijatnih iznenađenja" je **schema drift** kod baza podataka. Podrazumeva razilaženje između očekivane i stvarne strukture podataka, a dešava se kada se izvorna šema (kao što su tabele, kolone, tipovi podataka) iznenada ili neplanski promeni. Ovo može dovesti do grešaka u obradi podataka, prekida u data pipeline-ovima i netačnih analitičkih rezultata, posebno u realnom vremenu. Može se rešiti primenom expand/contract, tako što se prvo doda polje/tabela i omogući kompatibilnost (expand), a zatim raspoređivanje aplikacije koja koristi oba formata, i tek nakon stabilizacije se uklanjaju stare kolone (contract). U međuvremenu je potrebno držati dual-write/dual-read ako je potrebno. Ovaj deo nije specifičan za AWS, ali je neodvojiv od svake produkcijske strategije raspoređivanja.

Preporuke za upotrebu strategija raspoređivanja:

- **All at once** — Demo/POC, niska kritičnost, dopušten kratak prekid rada aplikacije.
- **Rolling (in-place)** — Stabilne aplikacije gde je važno ne duplirati sav kapacitet, rollback je zahtevniji.
- **Blue/green** — Minimalan rizik, rollback, skuplje (dupliran kapacitet na kratko).
- **Canary** — Analitika i fina testiranja u živom sistemu. I dealno uz AppConfig i automatske alarme.

Serverless i kontejnerske platforme omogućavaju da deployment postane vođen telemetrijom. Metrike i tracing govore kada prebaciti više saobraćaja, a automatizacija obavlja prebacivanje umesto čoveka. S vremenom, deployment postaje rutinski, a release postaje planski i kontrolisan.

[30] *Feature flags* – tehnika razvoja softvera, koja omogućava uključivanje i isključivanje opcija softvera, bez raspoređivanja novog koda. Zasniva se na uslovnom aktiviranju koda, zavisno od korisnika ili drugih uslova: ukoliko su neke opcije isključene u konfiguraciji, pojedine opcije softvera se ne aktiviraju.

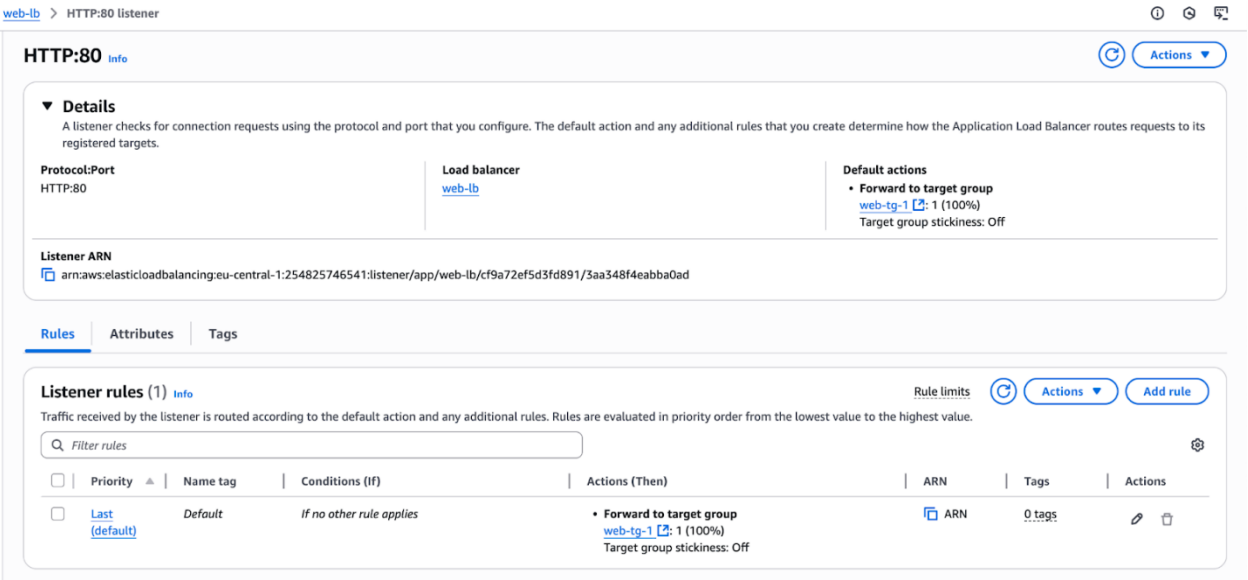
[31] CI/CD: Kontinuirana integracija (Continuous Integration) i Kontinuiraau isporuka (Continuous Delivery), metodologija razvoja softvera koja automatizuje procese izrade, testiranja i isporuke novog koda.

[32] Osnovne strategije implementacije. Canary znači postepen prelaz, uz mali procenat saobraćaja na početku, Linear je brži prelaz, kontinualno se prelazi sve više i više, dok je AllAtOnce prelazak odjednom, najbrži, ali narizičniji.

Primer

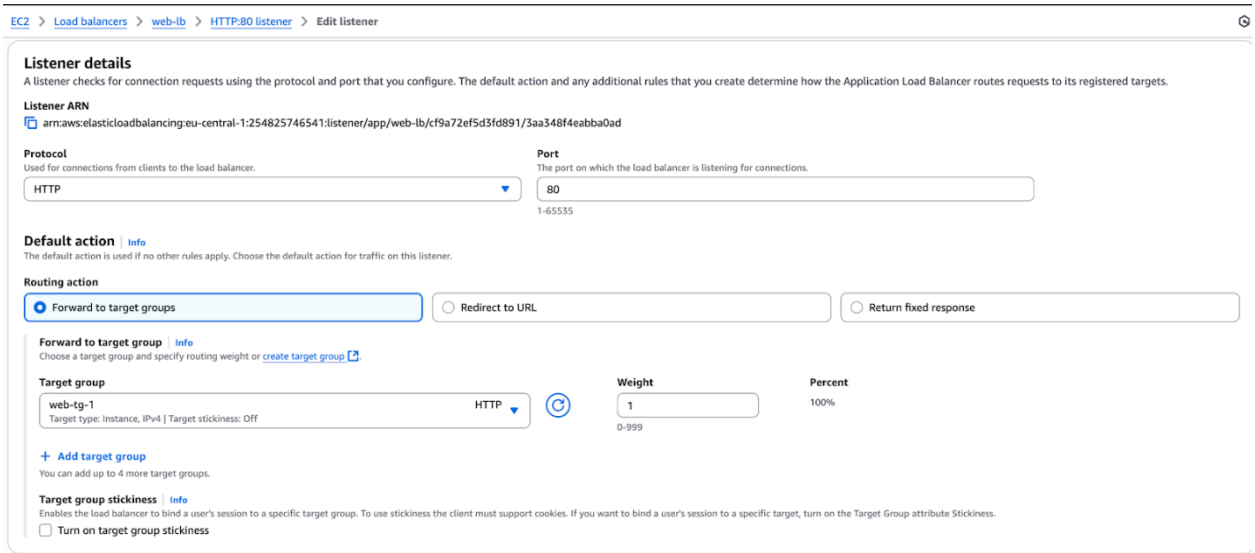
Za svrhe ovog primera može se ponoviti postavka slična vežbi za Load Balancing u okviru poglavlja za mreže. Dakle, napraviti dve web instance, target grupe i napraviti load balancer sa listenerom na portu 80 i ovu vežbu početi od te tačke.

Ako se pogleda početna konfiguracija Application Load Balancera, ona izgleda kao na slici Slika 132.



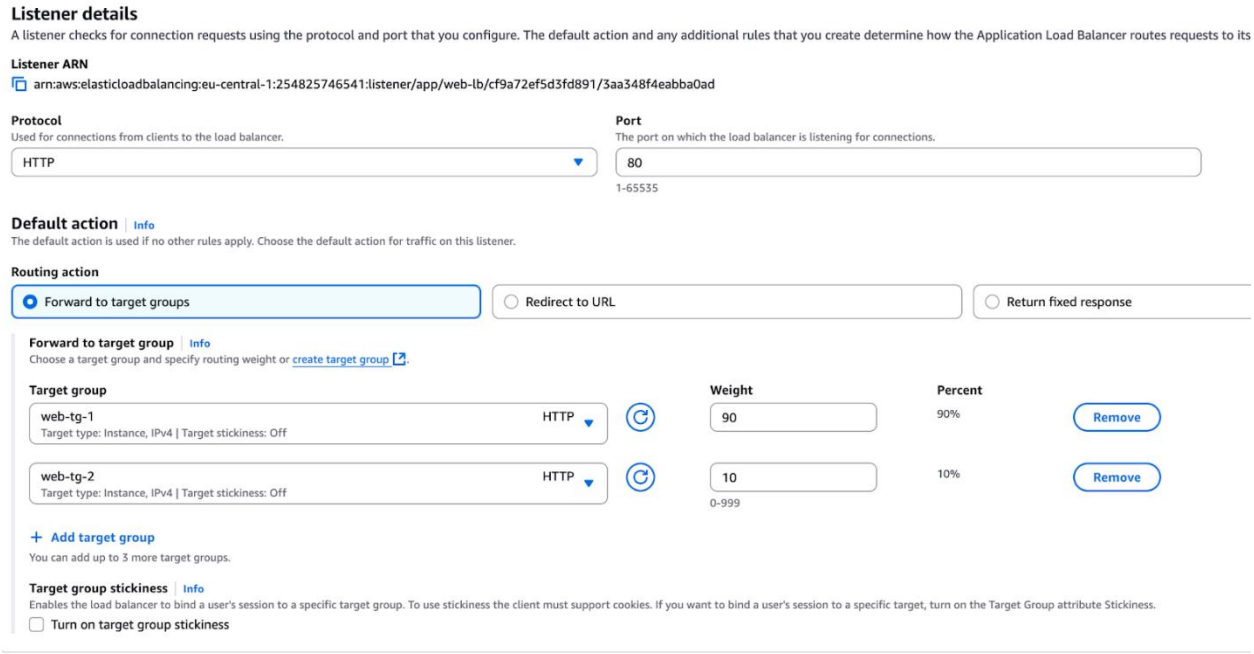
Slika 132 Početna konfiguracija vežbe

Potrebno je urediti podrazumevano listener pravilo tako što će se pod Listener rules odabrati listener i ići na Action i edit rule. Inicijalno se vidi da se sav saobraćaj rutira na jedan target group i stoji da je weight 1, tj. 100% saobraćaja, kao na slici Slika 133.



Slika 133 Početna konfiguracija Listenera

Kako bismo sada dodali web instancu 2, koja sadrži novu verziju aplikacije, potrebno je dodati novi target group odabirom Add target group opcije, a zatim podesiti weight prema željenom stanju. Na primer, kao na slici Slika 134, podešeno je 90 i 10 na target grupama.



Slika 134 Podešen Target Group weight

Ova konfiguracija znači da će 90% saobraćaja otići na originalni target group gde je prethodna verzija softvera, a samo 10% otići na novi target group koji sadrži novu verziju softvera. Ovakvom konfiguracijom može se kontrolisano prebaciti saobraćaj sa prethodne na novu verziju softvera sve dok se ne postigne 100% saobraćaja na web-tg-2; nakon toga može se u potpunosti ukloniti web-tg-1 i sa tim završiti deployment.

Mini zadatak: Pristupiti DNS vrednosti load balancera i kroz browser i raditi refresh dok se ne prikaže web stranica druge EC2 web server instance, zatim korigovati listener rules ponovo na veći procenat prema target group 2 i testirati ponašanje ponovo.

Ovakav deployment je blue/green canary.

Zadatak

1. Testirati neku od preostalih metoda deploymenta. Može se probati canary i sa Lambda funkcijama, rolling update aplikacije uz pomoć AWS Beanstalk ili nešto drugo.

10.2. Alati za upravljanje konfiguracijom

Uloga alata za konfiguraciju (configuration management tools) je da operativni sistem i aplikacioni sloj dovedu u željeno stanje tako što se instaliraju paketi, podešavaju servisi, kreiraju korisnici, drop-in konfiguracije, templatuju fajlovi i obezbedi da su promene **idempotentne** — ponovno pokretanje ne pravi neželjene efekte. U modernim AWS okruženjima, configuration management sprovodi se kao obavezno sa kontejnerima i AMI baking (Packer), a često se uopšte ne diraju podovi/instance jer se konfiguracija isporučuje već upakovana u image.

Ansible je agentless (SSH/WinRM), čitljiv (YAML) i zbog toga omiljen za brze i pregledne playbook-e. **Puppet** i **Chef** koriste agent/server i model deklarativnog stanja, više odgovaraju velikim okruženjima sa centralizovanom kontrolom i compliance-om. **SaltStack** kombinuje brzu orkestraciju i event-driven pristup. Na strani AWS-a, **Systems Manager (SSM)** obezbeđuje Run Command, State Manager, Patch Manager i Distributor, sve agent-based, bez otvaranja SSH-a, sa inventarizacijom i auditom (CloudTrail).

Configuration Management je najbolje koristiti kad:

- Postoje dugovečne mašine (na primer, bastion host, self-hosted broker itd.) koje zahtevaju konfiguraciju nakon pravljenja (post-provisioning);
- Kad se hibridno upravlja i on-prem i cloud serverima;
- Kad je potrebna brza isporuka manjih promena bez rebuilda AMI ili kontejnera.

Za stateless aplikacije u kontejnerima i autoscaling konfiguracijama, primaran pristup je immutable image (Docker/AMI), a Configuration Management ostaje minimalan (telemetrija, agenti).

Primeri

Primer Ansible — Instalacija Nginxa i jednostavne konfiguracije, idempotentno:

```
# site.yml
- hosts: web
  become: true
  tasks:
    - name: Install nginx
      apt: { name: nginx, state: present, update_cache: yes }
    - name: Deploy config
      template: { src: nginx.conf.j2, dest: /etc/nginx/nginx.conf }
      notify: Reload nginx
  handlers:
    - name: Reload nginx
      service: { name: nginx, state: reloaded }
```

Uz dinamički inventory za AWS (EC2 plugin), grupe se formiraju po tagovima (na primer Environment=prod), što je praktično za slojeve (web, app, db).

Primer AWS Systems Manager — Primena CloudWatch agenta preko State Managera, definiše se Association za tagovane instance, SSM Agent bezbedno povlači dokument (SSM Document), instalira i konfiguraciju drži pod kontrolom (verzije, retry, audit).

Dobre prakse:

- Idempotentnost, deklarativni stil i mali, proverljivi playbook;
- Tajne čuvati, rotirati i povlačiti iz SSM Parameter Store i Secrets Manager (nikako u git);
- Testiranje;
- Meriti uspeh kroz telemetriju — svaki Cloud Management korak proizvodi logove i metrike (CloudWatch).

Zadatak

Instalirati Ansible i podići AWS EC2 instancu, zatim pomoću Ansible podesiti konekciju ka EC2 instanci i instalirati web server i podesiti jednostavnu HTML stranicu.

10.3. Infrastuktura kao kod

Infrastruktura kao kod — **Infrastructure as Code (IaC)** tretira infrastrukturu kao verzionisani kod, tako što je željeno stanje opisano deklarativno/imperativno; alat generiše plan promene na osnovu trenutnog (realnog) i željenog stanja, zatim kontrolisano primenjuje promene. Ključne koristi su ponovljivost, code review, audit i automatizacija kroz CI/CD.

Deklarativni i imperativni pristup:

- **Deklarativno** (Terraform, CloudFormation) — Koje je željeno stanje (resursi i njihovi atributi)? Alat izračuna graf zavisnosti i proizvede razliku i plan.
- **Imperativno** (CDK, Pulumi) — Piše se u programskom jeziku (TypeScript, Java, Python itd.), zatim biblioteka generiše deklarativni manifest (npr. CloudFormation) i onda se manifest primenjuje.

Alati koji se koriste u praksi su:

- **Terraform** — multi-cloud, bogat ekosistem provajdera, remote state (S3) i locking (DynamoDB), moduli, workspaces.
- **CloudFormation** — AWS native alat, drift detekcija, StackSets (multi-account/region), change set.
- **AWS CDK (Cloud Development Kit)** — Konstruisanje AWS resursa kroz programski jezik na osnovu kog se generiše CloudFormation.
- **Pulumi** — IaC u opštem jeziku, multi-cloud, stabilan state backend.

Ključni koncepti kod IaC su jako bitni za razumevanje. **State i drift** se odnose na stanje, na primer, Terraform čuva u S3 trenutno stanje i koristi DynamoDB za zaključavanje kako ne bi došlo do toga da dva korisnika ili korisnik i CICD u isto vreme izvršavaju promene, što bi dovelo do drifta. **Moduli i stackovi** tipično služe za enkapsuliranje određenih komponenti, kao što su VPC, mrežne komponente, observability stack, korisnički servisi itd. **Environments** (okruženja — dev, staging, prod) izoluju okruženja i parametre jedne od drugih, kod Terraforma koristi se Workspaces za tu kontrolu, a .tfvars fajlovi za parametre koji su specifični za to okruženje. Na primer, veličina diska vezanog za EC2 instancu neće biti ista u svakom okruženju, pa samim tim može se podesiti kroz .tfvars. To je pogodno, jer postoji izolacija i na nivou AWS naloga prema okruženjima. U GitOps/CI procesima automatski se izvršava formatiranje, validacija i lintovanje[33] koda, zatim security skeniranje (npr. tfsec ili Checkov). Nakon toga se generiše plan kao artefakt, dok se primena (apply) najčešće ostavlja kao ručno odobrenje nakon pregleda i potvrde tačnosti plana. Policy as Code rešenja, kao što su OPA (Open Policy Agent), SCP ili Terraform Sentinel, omogućavaju sprovođenje definisanih pravila — na primer: svaki S3 bucket mora biti enkriptovan, load balancer mora biti privatan itd. Upravljanje tajnama (Secrets) se vrši preko referenci na AWS SSM Parameter Store ili Secrets Manager. Za tajne u formi fajlova preporučuje se korišćenje SOPS-a (Secrets OPerationS) u kombinaciji sa KMS-om (Key Management Service) za enkripciju. Tajne nikada ne treba čuvati kao plain text u Terraform state-u ili Git repozitorijumu.

[33] Lintovanje koda je postupak automatske analize koda kojom se otkrivaju potencijalne greške i propusti u kodu.

Primeri

Terraform

Opis primera: Terraform backend podešen na S3, sa state lockingom u DynamoDB tabeli. Dobavljač (provider) je AWS u eu-central-1 (Frankfurt), varijabla za ime bucket-a, S3 bucket sa uključenim verzionisanjem sačuvanih objekata i server side enkripcijom. Svaki javni pristup S3 bucket-u je blokiran.

```
terraform {
 backend "s3" {
   bucket         = "iac-state-prod"
   key            = "s3/app-bucket.tfstate"
   region         = "eu-central-1"
   dynamodb_table = "iac-state-locks"
   encrypt        = true
 }
 required_version = ">= 1.7.0"
}

provider "aws" {
 region = "eu-central-1"
}

variable "bucket_name" {
 type        = string
 description = "The name of the S3 bucket"
}

resource "aws_s3_bucket" "app" {
 bucket        = var.bucket_name
 force_destroy = false
}

resource "aws_s3_bucket_versioning" "versioning" {
 bucket = aws_s3_bucket.app.id
 versioning_configuration {
   status = "Enabled"
 }
}

resource "aws_s3_bucket_server_side_encryption_configuration" "sse" {
 bucket = aws_s3_bucket.app.id
 rule {
   apply_server_side_encryption_by_default {
     sse_algorithm = "AES256"
   }
 }
}

resource "aws_s3_bucket_public_access_block" "pab" {
 bucket = aws_s3_bucket.app.id

 block_public_acls       = true
 block_public_policy     = true
 ignore_public_acls      = true
 restrict_public_buckets = true
}
```

CloudFormation

Opis primera: CloudFormation jednostavni template za postavljanje EC2 web servera sa security grupom

```
AWSTemplateFormatVersion: 2010-09-09
Description: CloudFormation Template for WebServer with Security Group and EC2 Instance
Parameters:

 LatestAmiId:
   Description: The latest Amazon Linux 2 AMI from the Parameter Store
   Type: 'AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>'
   Default: '/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2'

 InstanceType:
   Description: WebServer EC2 instance type
   Type: String
   Default: t2.micro
   AllowedValues:
     - t3.micro
     - t2.micro
   ConstraintDescription: must be a valid EC2 instance type.

 MyIP:
   Description: Your IP address in CIDR format (e.g. 203.0.113.1/32).
   Type: String
   MinLength: '9'
   MaxLength: '18'
   Default: 0.0.0.0/0
   AllowedPattern: '^(\d{1,3}\.){3}\d{1,3}\/\d{1,2}$'
   ConstraintDescription: must be a valid IP CIDR range of the form x.x.x.x/x.

Resources:
 WebServerSecurityGroup:
   Type: AWS::EC2::SecurityGroup
   Properties:
     GroupDescription: Allow HTTP access via my IP address
     SecurityGroupIngress:
       - IpProtocol: tcp
         FromPort: 80
         ToPort: 80
         CidrIp: !Ref MyIP

 WebServer:
   Type: AWS::EC2::Instance
   Properties:
     ImageId: !Ref LatestAmiId
     InstanceType: !Ref InstanceType
     SecurityGroupIds:
       - !Ref WebServerSecurityGroup
     UserData: !Base64 |
       #!/bin/bash
       yum update -y
       yum install -y httpd
       systemctl start httpd
       systemctl enable httpd
       echo "<html><body><h1>Hello World!</h1></body></html>" > /var/www/html/index.html

Outputs:
 WebsiteURL:
   Value: !Join
     - ''
     - - http://
       - !GetAtt WebServer.PublicDnsName
   Description: Website URL
```

AWS CDK

Opis: Typescript CDK kod za pravljenje S3 bucketa za zabranjenim javnim pristupom

```
import { Stack, StackProps, RemovalPolicy } from 'aws-cdk-lib';
import { Bucket, BlockPublicAccess } from 'aws-cdk-lib/aws-s3';
import { Construct } from 'constructs';

export class StorageStack extends Stack {
 constructor(scope: Construct, id: string, props?: StackProps) {
   super(scope, id, props);

   new Bucket(this, 'AppBucket', {
     blockPublicAccess: BlockPublicAccess.BLOCK_ALL,
     versioned: true,
     removalPolicy: RemovalPolicy.RETAIN
   });
 }
}
```

Opis: Python CDK kod za pravljenje S3 bucketa sa ukljucenim verzionisanjem

```
from aws_cdk import (
   aws_s3 as s3,
   Stack,
)
from constructs import Construct

class MyCdkAppStack(Stack):

   def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
       super().__init__(scope, construct_id, **kwargs)

       s3.Bucket(
           self,
           "MyFirstBucket",
           bucket_name="my-unique-cdk-bucket-123",
           versioned=True
       )
```

CDK generiše CloudFormation, pa se dobija prednost deklarativnog raspoređivanja uz korišćenje programskih jezika.

Testiranje, bezbednost i CICD kod IaC alata:

- **Lint i security** — za terraform, fmt i validate, tflint, tfsec/Checkov, za CFN, cfn-lint, cfn-nag.
- **Plan kao artifact** — U CI se čuva terraform plan i pokazuje se reviewerima. Apply koristi tačno taj plan koji je prošao review. Ovde je bitno napomenuti da plan može da istekne, ukoliko se ne okine apply na vreme, mora se ponovo generisati plan.
- **Environments i promocija** — dev, staging, prod kroz odvojene naloge i parametre. Izbegava se shared state i ručna presipanja.
- **Kontrola troška** — Tagovi resursa (CostCenter, Env), budžeti i alarmi. IaC obezbeđuje konzistentnost i jasnu vlasničku liniju

Tipične zamke i kako ih izbeći:

- **Ručne izmene** — U konzoli ručne izmene dovode do drifta i iznenađenja.
- **Centralni monolitni modul** — Koristiti više manjih, testiranih modula.
- **Tajne u kodu/state-u** — Koristiti SSM/Secrets Manager/SOPS. Proveriti da je Terraform state enkriptovan na storage sloju.

- **Nekontrolisana paralelizacija i zavisnosti** — Prepustiti alatu da gradi DAG, eksplicitno koristiti depends_on samo kad je to neophodno.

Cloud Management i Infrastructure as Code nisu konkurenti, već slojevi istog sistema. IaC gradi stub infrastrukture, kao što su mreža, sigurnost, servisi, dok CM dograđuje OS i aplikaciju gde to ima smisla.

Zadatak

Prvi zadatak: Koristeći Terraform, realizovati sledeći zadatak:

1. Podići S3 state backend i DynamoDB lock tabelu.
2. Napraviti Terraform modul za VPC (CIDR kao varijabla) i pozvati ga iz tri okruženja (dev/stg/prod).
3. Uvesti Checkov/tfsec (može kroz pre-commit hooks)
4. (Opciono) Dodati OPA/Conftest pravilo: svaki resurs mora imati tags = { Owner, Env }

Drugi zadatak: Koristeći CDK u željenom programskom jeziku, uraditi zadatak:

1. Podići VPC okruženje sa private i public subnetima, kao i ostalim resursima.
2. Podići EC2 web server u public subnetu.
3. Podići RDS bazu podataka u private subnetu.
4. Postaviti security grupe tako da sve komponente mogu da komuniciraju.

10.4. Serverless

Serverless nije „magija bez servera”, već model u kome infrastruktura postoji, ali nema ručnog upravljanja, tj. nema upravljanja operativnim sistemom, zakrpama, autoscaling grupama, niti kapacitetnim planiranjem na nivou mašina. Treba razmišljati o događajima (event-ovima) i funkcijama/servisima koji na te događaje reaguju. Naplata je po upotrebi, po pozivu, trajanju izvršavanja, broju zapisa, veličini objekta. Rezilijentnost i skaliranje su podrazumevani, a ostatak logike samostalno se gradi i sklapa kroz upravljane servise (Aluvalu&Maheswari, 2024).

U AWS-u, serverless nije ograničen samo na Lambda funkcije. To je čitav ekosistem servisa:

- Računarstvo - Lambda
- API sloj - API Gateway, AppSync
- Orkestracija - Step Functions
- Događaji - EventBridge
- Redovi (Queue) i obaveštenja - SQS, SNS
- Podaci - DynamoDB, S3, Aurora Serverless, Redshift Serverless, OpenSearch Serverless
- Analitika i obrada tokova - Kinesis, Glue, EMR Serverless
- Bezbednost i identitet - Cognito
- Posmatranje (Observability) - CloudWatch, X-Ray
- Serverless za kontejnere - AWS Fargate

U okviru ovog poglavlja cilj je da se upoznaju osnovni koncepti, a zatim prođu i napredni obrasci koji se stvarno koriste u praksi, kao što su: event-driven arhitekture, idempotentnost, orkestracija vs. koreografija, pouzdanost u asinhronom svetu i optimizacija troška i performansi.

U klasičnom modelu (IaaS) razmišlja se u instancama, dok se u serverless svetu razmišlja u funkcijama i događajima. Funkcija reaguje na HTTP zahtev, na poruku u redu, na fajl u S3, na zapis u streamu, na raspored (cron) itd. Platforma podiže dovoljno radnika (workers), raspodeljuje događaje, čuva retry semantiku i izbacuje neuspehe u DLQ (dead-letter queue) ako zatreba.

Prednosti:

- Brz put od ideje do produkcije
- Automatsko skaliranje
- Fina granularnost troška
- Manje površine za bezbednosne propuste u OS-u

Kompromisi:

- Ograničenja vremena izvršavanja;
- Hladni start (Cold Start) — Servisi se isključuju kada se završi korišćenje, a onda, po novom pozivu, potrebno je ponovo kreirati okruženje za izvršavanje.
- Ograničenja mreže;
- Potreba za asinhronim dizajnom — retry, idempotentnost, redosled.

Ključni serverless servisi u AWS-u

AWS Lambda izvršava funkcije na zahtev. Kada se konfiguriše, odabir količine RAM memorije povećava i količinu CPU dostupnosti. Privremena /tmp memorija je dostupna (konfigurisana), a konkurentnost skalira automatski, uz određene „meke" kvote i zaštite, koje se mogu povećati. Funkcije se mogu pakovati kao ZIP ili kao container image. Pored HTTP-a (API Gateway, ALB), okidači (trigger) su S3, DynamoDB Streams (na osnovu promene u tabeli), SQS (poruke), SNS (notifikacije), EventBridge (raspored/događaji), Kinesis (data stream), Cognito itd.
Neke od naprednih opcija su:

- **Provisioned Concurrency** — Stalan „topli" pool za nisku latenciju, kako bi se prevazišla ograničenja hladnog starta, koja, zavisno od veličine Lambde, mogu biti značajna.
- **Lambda Layers** — Deljeni runtime/dependencije, idealno za pakovanje common paketa koje deli više lambda i biblioteka.
- **Extensions** — Telemetrija, agenti.
- **Function URLs** — Generiše direktan HTTP ulaz, bez potrebe za API Gatewayom ili sličnom komponentom ispred Lambde.

API Gateway nudi REST/HTTP/WebSocket APIe sa integracijama ka Lambda, ALB, direktno ka servisima (SQS, Kinesis itd.) i privatnim NLB-ovima. On pruža mogućnosti throttling, auth, uz pomoć Cognito/JWT i Lambda authorizer, keš, WAF integraciju i detaljne logove kroz CloudWatch.

AppSync je GraphQL managed sloj, resolveri mogu ići direktno ka DynamoDB, Lambda, HTTP i drugim izvorima, podržava real-time subscription kanale. Resolveri se pišu u VTL (Velocity Template Language).

Amazon EventBridge je event bus koji spaja proizvođače (producers) i potrošače (consumers) događaja bez „tvrde" zavisnosti. Pravila (rule) se usklade sa događajima i šalju ih ciljevima kao što su Lambda,

Step Functions, SQS/SNS, API Gateway itd. Tu su i Scheduler (cron) i pipes od SQS/Data steama do cilja uz filtriranje i obogaćenje podataka.

AWS Step Functions orkestrira više koraka poslovnog procesa. Standard Workflows za sporije procese, Express za high-throughput niskih troškova. Integriše se sa servisima, kao što su SNS, SQS, DynamoDB, Glue, EMR, SageMaker itd., map state (fan-out), callback uz task token (čekanje na eksterni signal), i izuzetno jasan model grešaka/retrya.

SQS je pouzdan red (queue), radi u **Standard**, ne garantuje redosled, ali je jeftin i brz, i **FIFO** modu, što znači da ima redosled i exactly-once processing na nivou reda. Lambda i SQS prirodno rade zajedno, Lambda povlači poruke, dozira se veličina batcha i visibility timeout koji definiše za koliko poruka opet da se pojavi u redu, ukoliko red nije dobio signal da je ona već obrađena.

SNS emituje poruke ka više pretplatnika (pub/sub) kao što su HTTP endpoint, SQS, Lambda, e-mail, SMS itd. Tipičan obrazac upotrebe je fan-out na više pretplatnika.

S3 je objektni (object) storage koji nudi verzionisanje, lifecycle, event-e, OAC za CloudFront, server-side enkripciju itd. Trigger S3 ka Lambdi je najčešći u slučajevima obrade fajlova.

DynamoDB je serverless NoSQL baza podataka. Nudi **particioni** (PK — partition key) i **sort** ključ (SK — sort key), **GSI** (Global Secondary Index) za alternativne query patterne. Nudi i **Streams** za reaktivnost — Lambda reaguje na promene u bazi, TTL, transakcije, **on-demand** i **provisioned** kapacitet.

Druge serverless servisi za podatke:

- **Aurora Serverless v2** — relaciona baza podataka, automatsko skaliranje;
- **Redshift Serverless** — data warehousing;
- **OpenSearch Serverless** — pretraga i skladištenje logova;
- **EMR Serverless** — Spark/Hive bez klastera;
- **Kinesis on-demand** — Data streams.

Cognito je korisnički bazen (user pool) koji nudi autentifikaciju, autorizaciju, federaciju (Google/Apple/ADFS), izdavanje JWT[34]-a, napredne mogućnosti sign in i sign up procesa, MFA itd.

CloudWatch nudi metrike, logove, alarme i dashboard, **Logs Insights** za upite sačuvanih logova, **Internet Monitor** za mrežnu degradaciju, **Synthetics** za canary-je itd.

AWS X-Ray — distributed tracing, integracija sa Lambda, API Gateway, Step Functions itd. vizuelizuje putanju requesta i latencije.

Performanse, najbolje prakse i kada serverless nije rešenje

Asinhroni sistemi moraju podneti **ponavljanja** i **reordere**. Šta to znači:

- **Idempotentni handler** — Ako isti događaj stigne dva puta, posledica je kao da je stigao jednom. Tehnike koje pomažu sa tim su deduplikacioni ključevi (SQS FIFO) i DynamoDB

[34] JWT (JSON Web Token) je standard za siguran prenos informacija (kao što su podaci o korisniku) između klijenta i servera putem JSON objekata, najčešće u aplikacijama koje koriste princip stateless (bez stanja) autentifikacije.

conditional writes (attribute_not_exists). Postoje i druge tehnike zavisno od arhitektura i korišćenih servisa.

- **Retry/backoff** — Koristiti podrazumevana ponavljanja (Lambda, EventBridge, Step Functions) i razdvojiti trenutne greške (rate limit) od trajnih (schema invalidna i šalje se u DLQ).

- **Tačnost** — „exactly-once“ je idealan delivery, ali u praksi se postiže at-least-once delivery i idempotentno na potrošaču ili FIFO uz deduplikaciju i transakcije.

Iako serverless nudi visoke performanse, postoji dosta faktora koji utiču na njih i zahtevaju dodatne optimizacije prema opterećenju i količini podataka. Bitni faktori kako bi se postigle optimalne performanse i troškovi su:

- **Hladni start** — Može da se ublaži kroz **Provisioned Concurrency** ili povremenim okidanjem Lambde kako bi ostala u toplom stanju. Bitan je i izbor runtime i veličina memorije do određene mere; više memorije automatski dodeljuje više CPU resursa, što bi značilo kraće vreme izvršavanja, a samim tim to može značiti da su i troškovi manji jer se Lambda brže izvršava. Napomena: Nije uvek rešenje dodati što više resursa; potrebno je razumeti kod i ulogu Lambde. Nekad je faktor koji najviše usporava Lambdu API koji ona poziva i u tim slučajevima povećanje resursa nema nikakav benefit.

- **Batch vs. single** — SQS i Kinesis consumeri obrađuju u paketima radi efikasnosti; ti paketi mogu da variraju od nekoliko paketa odjednom do nekoliko desetina paketa. Kod Kinesisa taj broj može ići i do 100 paketa po obradi, uz napomenu da je to idealno gde se ne zahteva real time obrada, već je prihvativo i near real time (na primer kod IoT uređaja).

- **Skaliranje** — Razumevanje ograničenja konkurentnosti i „burst“ ponašanje je bitno radi konfiguracije potrošača. Preporučeno je koristiti reserved concurrency gde je to moguće. Kod Kinesis strimova se može podesiti veći batch i concurrency po Kinesis Shardu; to dozvoljava značajno veći protok podataka ka Lambdi.

- **Pravilan servis** — Lambda nije za duge, konstantno aktivne poslove, tada je bolje preći na Fargate, ECS ili EKS. Za relacione baze zahtevne po pitanju konekcija, koristiti **RDS Proxy** ili **Aurora Serverless.** Kod Lambdi je jako bitno razumevanje deljenih konekcija ka bazama podataka i pravilno otvaranje/zatvaranje konekcija, jer vrlo lako se može dostići maksimalni broj podržanih konekcija i „ugušiti“ baza podataka. Takođe, nije rešenje otvarati i zatvarati konekciju pri svakom izvršenju jer dolazi do opterećenja CPU resursa baze podataka.

Bitno je razumeti i kada serverless nije pravi izbor i treba koristiti neko drugo rešenje kao što je ECS, Fargate, EKS, EC2 itd. Serverless gubi smisao u sledećim slučajevima:

- **Dugoročni procesi** sa konstantnim CPU opretećenjem (na primer, 24/7 transcoding); u tim slučajevima pre treba koristiti Fargate/ECS/EKS, čak razmotriti upotrebu Spot instanci zavisno od toga da li su odgovarajuće za taj workload.

- **Stateful** sistemi koji zahtevaju stalne TCP konekcije i specifičnu mrežnu topologiju.

- Opterećenja koja zahtevaju specifičan kernel modul ili low-level OS pristup, licenciranje na nivou virtuelnih mašina itd.

- Serverless i kontejneri se odlično kombinuju u dosta slučajeva, kao event-driven front (Lambda, API GW) i podsistem u Fargate za teške poslove, koji mogu zahtevati i preko 10GB RAMa koji je limit na Lambdama.

Kako bi se postigli svi benefiti serverlessa, bitno je pridržavati se najboljih praksi:

- **Jedna funkcija** — Funkcija ima jednu odgovornost. Piše se kao mala funkcija, purpose built, laka za testiranje;

- **Idempotentnost svuda** — DLQ[35] i dobr retry/backoff implementacija;

- **Least privilege** — Dobro kontrolisane IAM polise, tajne u Secrets Manageru ili SSMu, VPC endpointi za pristup servisima umesto NAT-a kad god je moguće;

- **Observability by default** — Strukturirani logovi, X-Ray, metrike i alarmi;

- **IaC (SAM/CDK/Terraform)** — Automatizovani deployment i kontrolisane strategije deploymenta;

- **Merenje troška i latencije** — Redovno raditi tuning memorije i veličine batcheva, pogotovo kod većih izmena Lambde, dodavanja API poziva ka eksternim servisima i promena u količini obrade podataka.

Serverless je **inženjerska disciplina** pre nego što je samo niz servisa. Kad se nauči razmišljanje u događajima (eventima) i malim, idempotentnim koracima, AWS-ov skup servisa (Lambda, API GW, EventBridge, Step Functions, SQS/SNS, DynamoDB, S3…) pretvara se u platformu za brzo, sigurno i jeftino isporučivanje. Uz dobar IaC i observability, dobijate sisteme koji se **skaliraju bez drame** i „graciozno“ podnose greške — baš ono što želite u modernoj produkciji.

Primer

API Gateway, Lambda i DynamoDB integracija

U ovom primeru se posmatra jedan REST API Gateway sa putanjom da upiše i pročita unose u DynamoDB. Ako se podeli u korake, potrebno je uraditi sledeće:

1. Napraviti DynamoDB tabelu;

2. Napraviti IAM role sa dozvolama za upis i čitanje DynamoDB tabele;

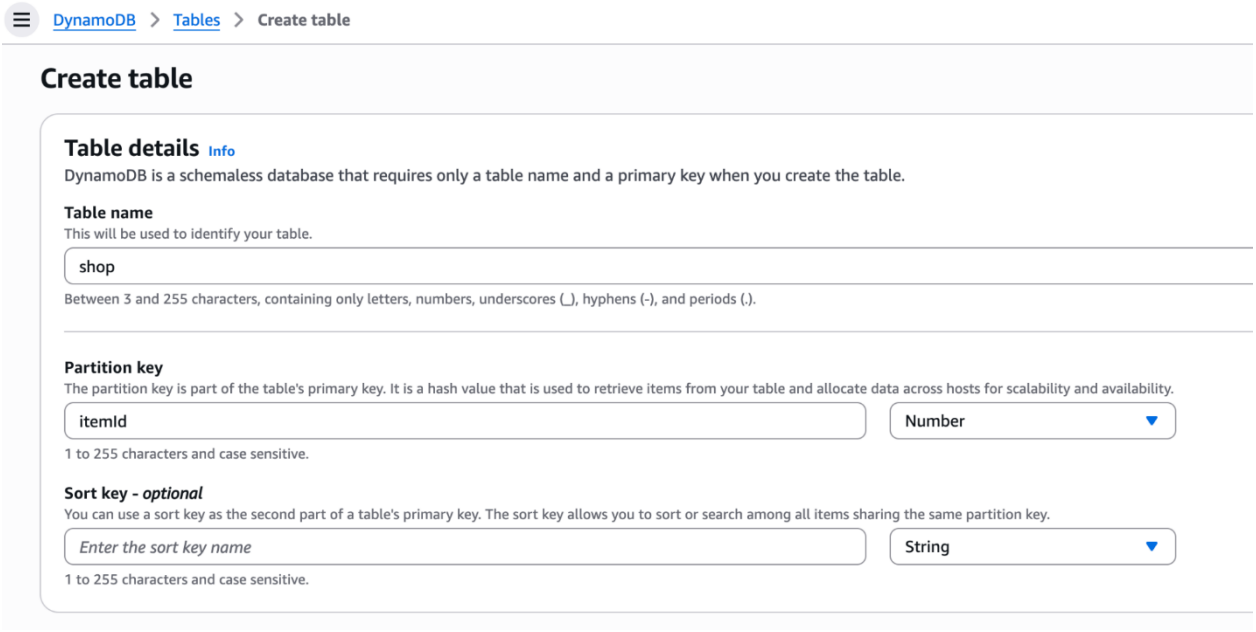
3. Napraviti Lambdu i zakačiti IAM role iz prethodnog koraka;

4. Napraviti API Gateway i definisati pristup Lambdi;

[35] DLQ je skraćenica za Dead Letter Queue (red neisporučivih poruka), poseban red za poruke koje sistem ne može da obradi. Koristi se za zadržavanje poruka koje su neuspešno obrađene iz glavnog reda (iz razloga kao što su greške u formatu, istek vremena ili problemi u prijemu), čime se sprečava blokiranje glavnog toka i omogućava analiza i ponovno slanje tih poruka nakon otklanjanja problema.

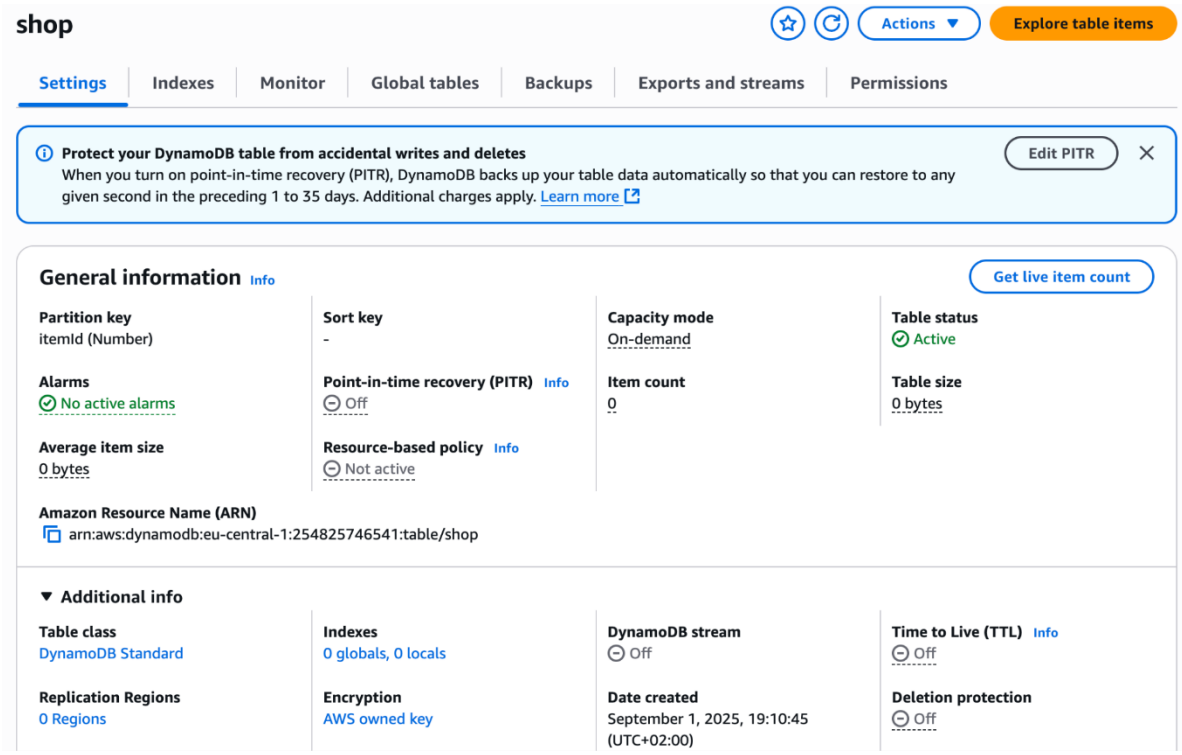
5. Testirati API.

Kako bi se napravila DynamoDB tabela, potrebno je pristupiti DynamoDB u AWS konzoli i odabrati opciju Create table. Zatim u Table name ukucati ime tabele, partition key (primarni ključ) uneti itemId (number), sort key se može preskočiti. Ostatak može ostati na Default settings i ići na Create Table (Slika 135).



Slika 135 Konfiguracija DynamoDB tabele

Nakon toga može se pogledati kako izgleda kreirana tabela (Slika 136) i proveriti status tabele, kao i kopirati njen ARN koji će biti potreban za sledeći korak.

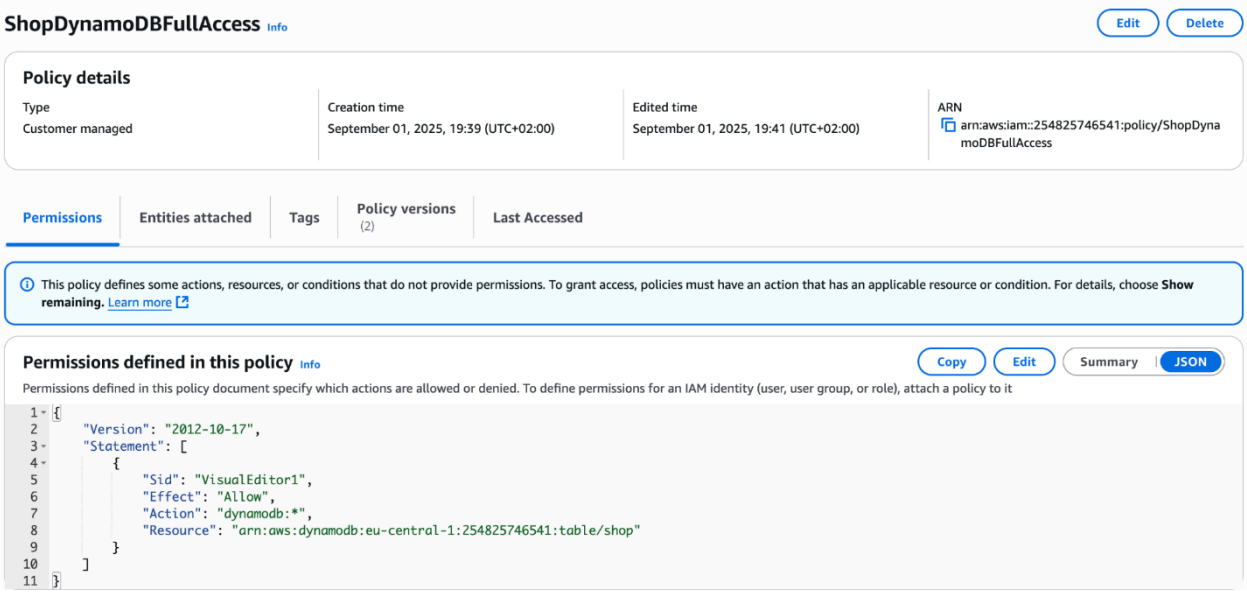


Slika 136 Prikaz napravljene DynamoDB tabele

Zatim je potrebno preći u IAM servis, gde će se napraviti odgovarajuća Lambda i uloga za pristup DynamoDB tabeli.

Za početak, potrebno je definisati IAM polisu, pa prelaskom na Policies ići na Create Policy. U okviru Specify permissions postoje dva načina za definisanje dozvola — Visual i Json. U oba slučaja na kraju će polisa izgledati isto. U ovom primeru koristiće se Visual. U Select a service upisati DynamoDB, izabrati All DynamoDB actions, a zatim u Resources izabrati Specific i pod table ići na Add ARNs i tu kopirati ARN tabele koja je prethodno napravljena i potvrditi. Zatim ići na Next i upisati Policy name, na primer ShopDynamoDBFullAccess. Nakon toga može se dodati Description, a i ne mora, i ići na Create Policy.

U Search naći polisu po imenu ShopDynamoDBFullAccess i pristupiti joj radi provere. Polisa treba da izgleda kao na slici Slika 137.

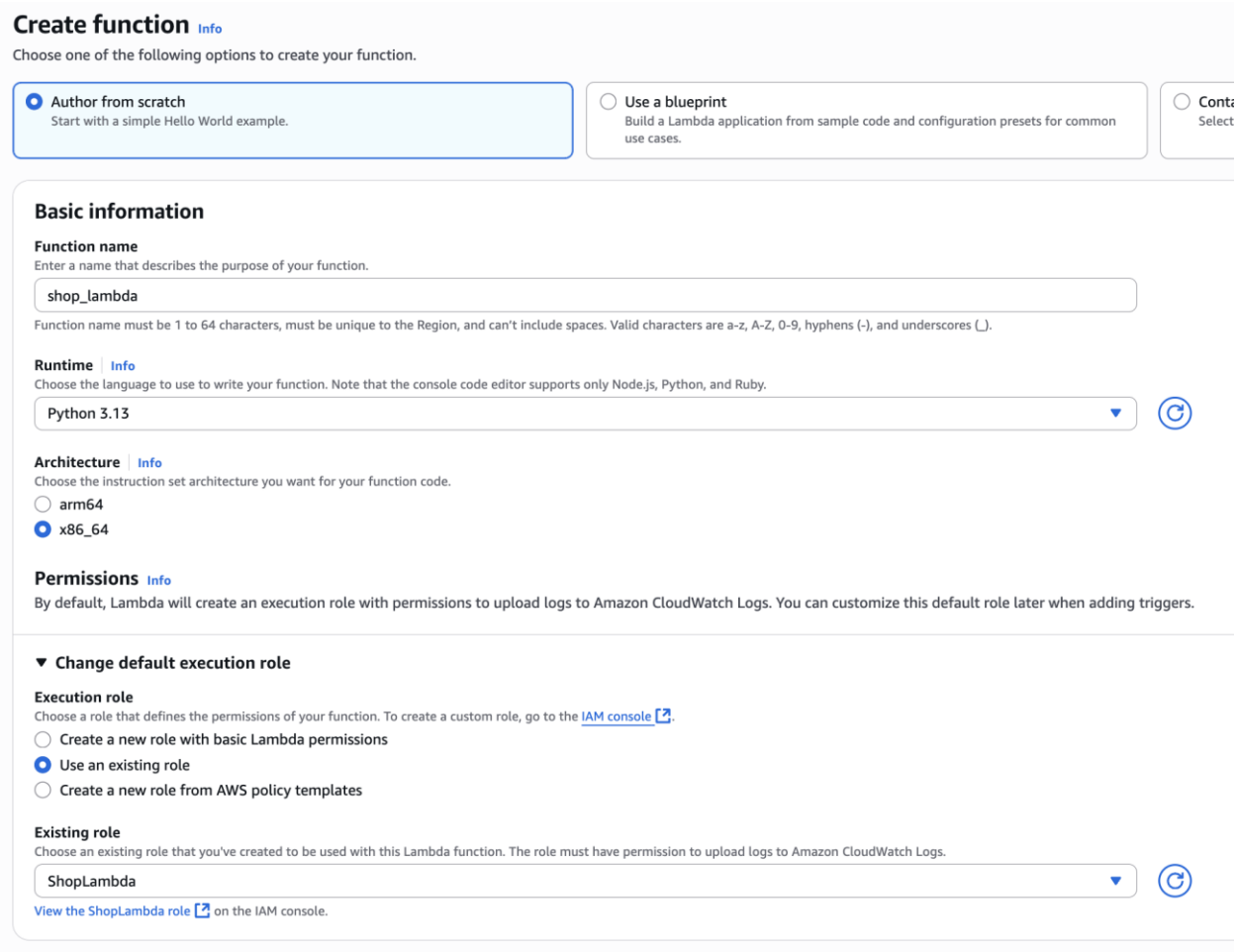


Slika 137 Prikaz ShopDynamoDBFullAccess polise

Napomena: Bez obzira što su akcije ograničene na jednu tabelu, nikada nije dobar savet dozvoliti sva prava pristupa (All actions), već uvek uraditi procenu prava i zadati tačno onoliki scope prava koliki je potreban. Za svrhe vežbe je prihvativo dodeliti sva prava.

Zatim, u okviru IAM izabrati Roles i izabrati Create role. Na sledećem ekranu u Select trusted entity izabrati AWS service i u Use case izabrati Lambda. Zatim se ide na next i u okviru Add permissions je moguće filtrirati kroz Search i dodati ShopDynamoDBFullAccess polisu koja je prethodno napravljena i ići Next. U Role name uneti ime, na primer, ShopLambda. Nakon toga u Roles pronaći ShopLambda role i proveriti da li je sve dodato kako treba u Permissions i Trust relationships.

Odabirom Lambda iz service search bara prelazi se na pravljenje Lambde. Odabrati Create Function i ostaviti Author from scratch, u ime Lambde može se upisati shop_lambda, u Runtime izabrati najnoviju verziju Pythona (u ovom slučaju 3.13). Pod Arhitekturom ostaviti x86 i ići na Change default execution role, izabrati Use an existing role i u meniju ispod izabrati rolu ShopLambda. Zatim ići na Create Function (Slika 138).



Slika 138 Konfiguracija Shop Lambde

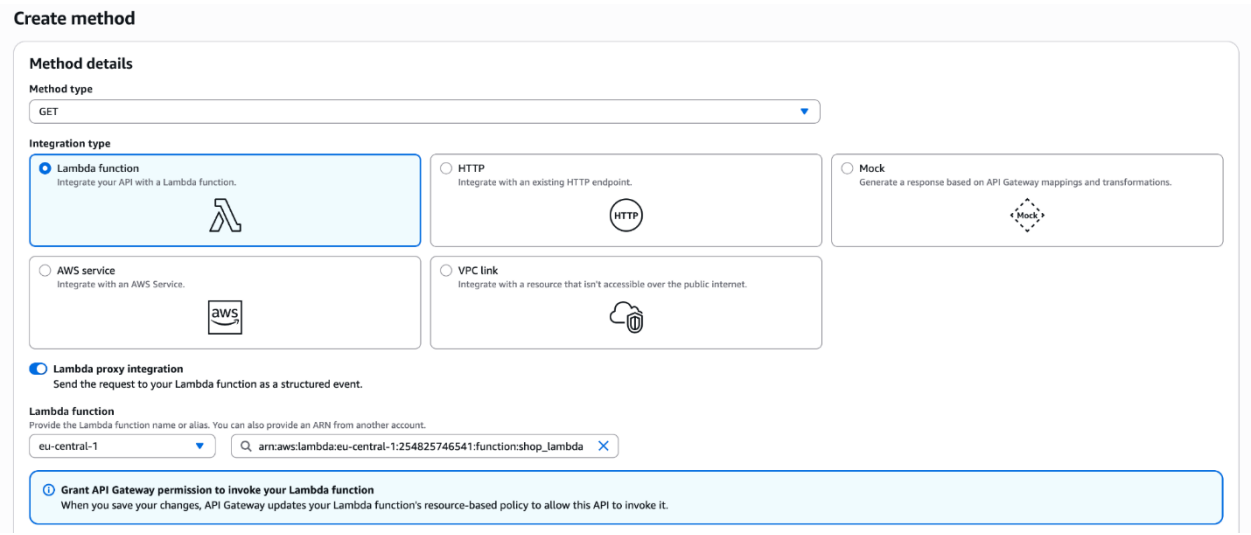
U okviru Lambde, postaviti sledeći Python kod:

```
import json
import boto3
from decimal import Decima

table = boto3.resource("dynamodb").Table("shop")

CORS = {
    "Access-Control-Allow-Origin": "*",
    "Access-Control-Allow-Methods": "GET,POST,OPTIONS",
    "Access-Control-Allow-Headers": "Content-Type",
}

def decimal_encoder(obj):
    if isinstance(obj, Decimal):
        return int(obj) if obj % 1 == 0 else float(obj)
    raise TypeError(f"Object of type {type(obj)} is not JSON serializable")
```

```
def response(code, body):
    return {"statusCode": code, "headers": {"Content-Type": "application/json",
**CORS}, "body": json.dumps(body, default=decimal_encoder)}



def lambda_handler(event, context):
    method = event.get("httpMethod")
    path = (event.get("path") or "").rstrip("/")

    # GET /items → scan all
    if method == "GET" and path == "/items":
        data = []
        scan_args = {}
        while True:
            resp = table.scan(**scan_args)
            data.extend(resp.get("Items", []))
            lek = resp.get("LastEvaluatedKey")
            if not lek:
                break
            scan_args["ExclusiveStartKey"] = lek
        return response(200, data)

    # POST /add  → insert item
    if method == "POST" and path == "/add":
        body = json.loads(event.get("body") or "{}")
        try:
            item = {
                "itemId": int(body["itemId"]),
                "name": str(body["name"]),
                "description": str(body["description"]),
            }
        except (KeyError, ValueError, TypeError):
            return response(400, {"message": "Body must include numeric itemId,
name, description."})
        table.put_item(Item=item)
        return response(201, {"message": "created", "item": item})
    return response(404, {"message": "Not found"})
```

Nakon toga preći na Api Gateway pomoću pretrage servisa.

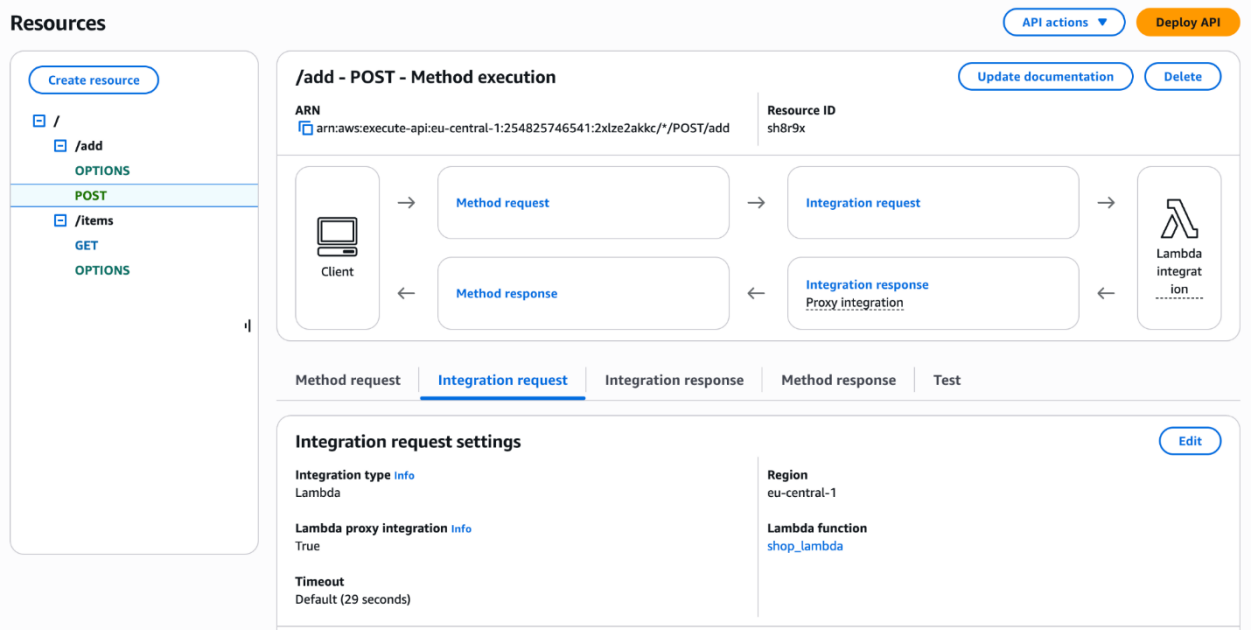
Zatim napraviti novi klikom na Create API, pod REST API odabrati opciju Build. Ostaviti New API, u ime uneti Shop API i ići na Create API. Potrebno je ići na opciju Create Resource, u Resource Path upisati /, a u Resource Name upisati items, potvrditi CORS. Zatim odabrati Items i ići na opciju Create method (Slika 139), gde je Method type GET, integration type Lambda function, uključen Lambda proxy integration i u Lambda function odabrana shop lambda. Nakon toga ići na Create integration.



Slika 139 Podešavanje lambda integracije

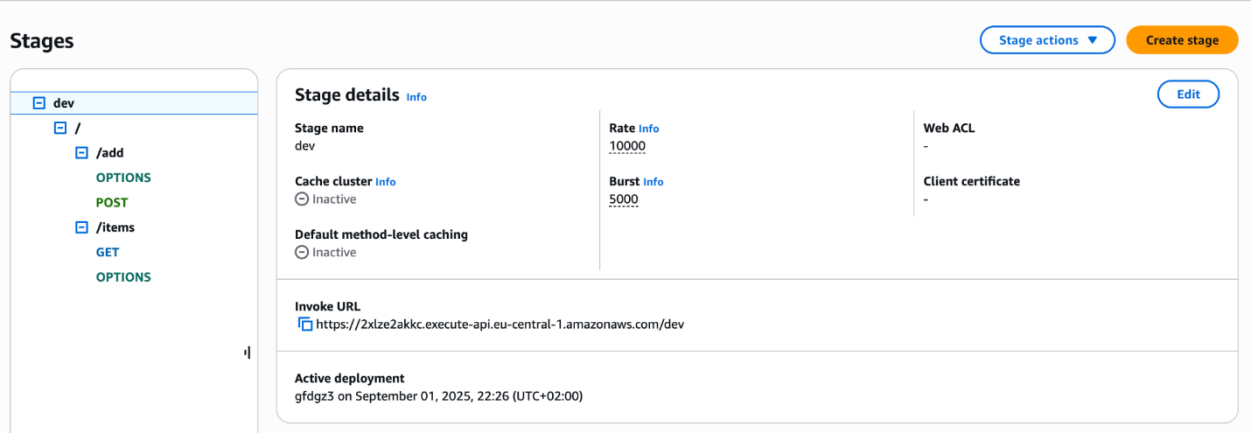
Zatim se dodaje još jedan endpoint, tj. resource - add. Ponovo odabrati / na Resources delu i odabrati Create Resource. Nakon toga popuniti isto kao u prethodnom koraku, samo je sada putanja add. Posle kreiranja odabrati /add putanju i ići na Create method. Za method type odabrati POST, ostatak ostaje isti kao u prethodnom koraku, popunjavaju se parametri za Lambdu.

Krajnji rezultat je prikazan na slici Slika 140.



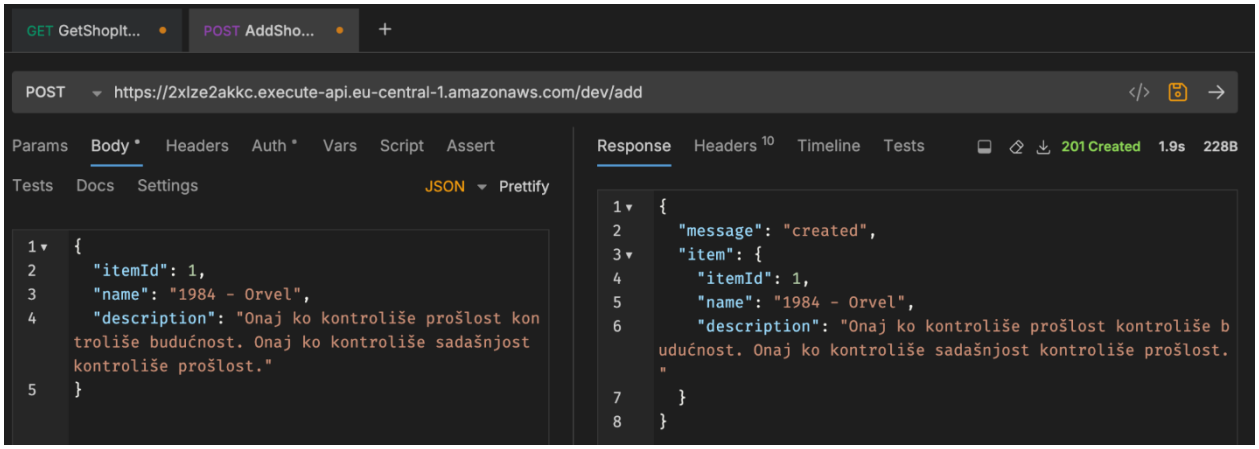
Slika 140 Podešen Shop API Gateway sa minimalnom konfiguracijom

Nakon toga ići na Deploy API, ispod Stage izabrati New Stage, upisati dev u Stage name i ići na deploy. Nakon toga dobija se Invoke URL za taj API (Slika 141).



Slika 141 Objavljen Shop API

Sada se može obaviti testiranje API. Testiranje je moguće izvršiti pomoću CURL komandi ili nekih popularnih alata kao što su Postman ili Bruno. Potrebno je prozvati API na pravilan način i odgovarajućim metodama. To podrazumeva korišćenje Invoke URL prikazanog na slici Slika 141 i dodavanje na path /add i /items zavisno od API poziva koji se pravi. Na primeru ispod (Slika 142) dodata je knjiga na listu stavki (items) koristeći POST metodu na /add endpoint.

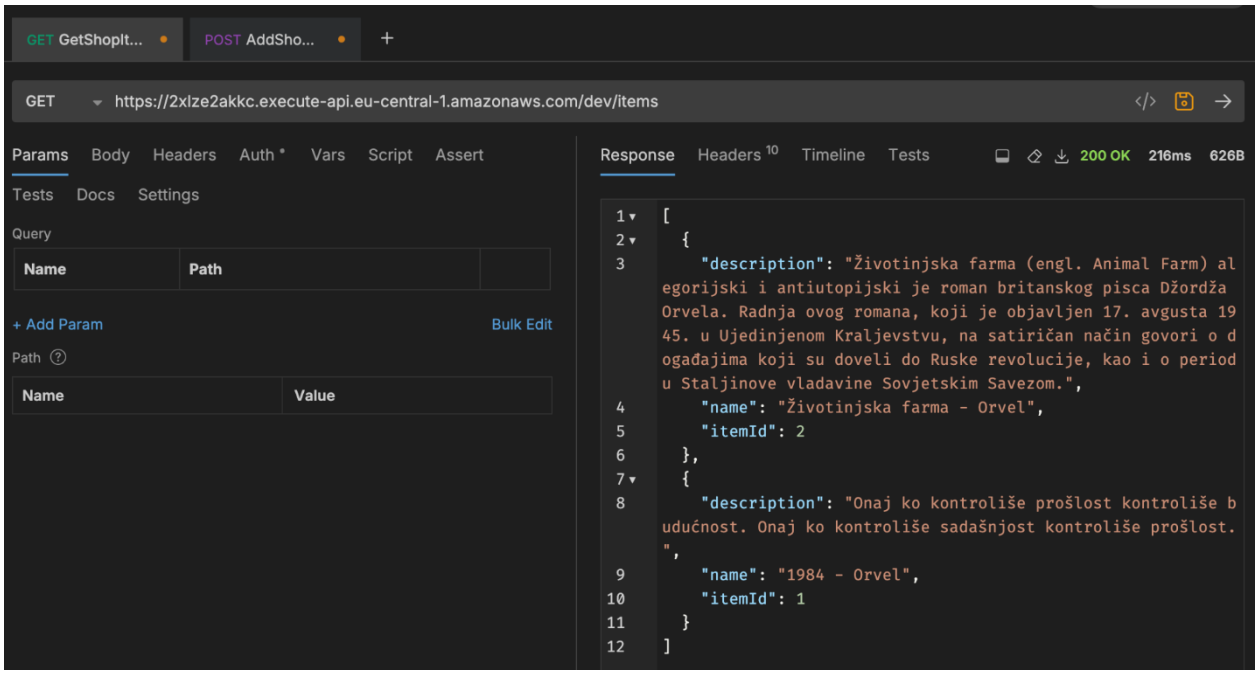


Slika 142 Dodavanje stavke u Shop

Primer dodavanje itema kroz CURL komandu:

```
curl --request POST \
 --url https://2xlze2akkc.execute-api.eu-central-1.amazonaws.com/dev/add \
 --header 'content-type: application/json' \
 --data '{
 "itemId": 2,
 "name": "Životinjska farma - Orvel",
 "description": "Životinjska farma..."
}'
```

Nakon toga, koristeći GET metodu i drugi endpoint, tj. /items može se pogledati šta sve od itema postoji u shopu (Slika 143).

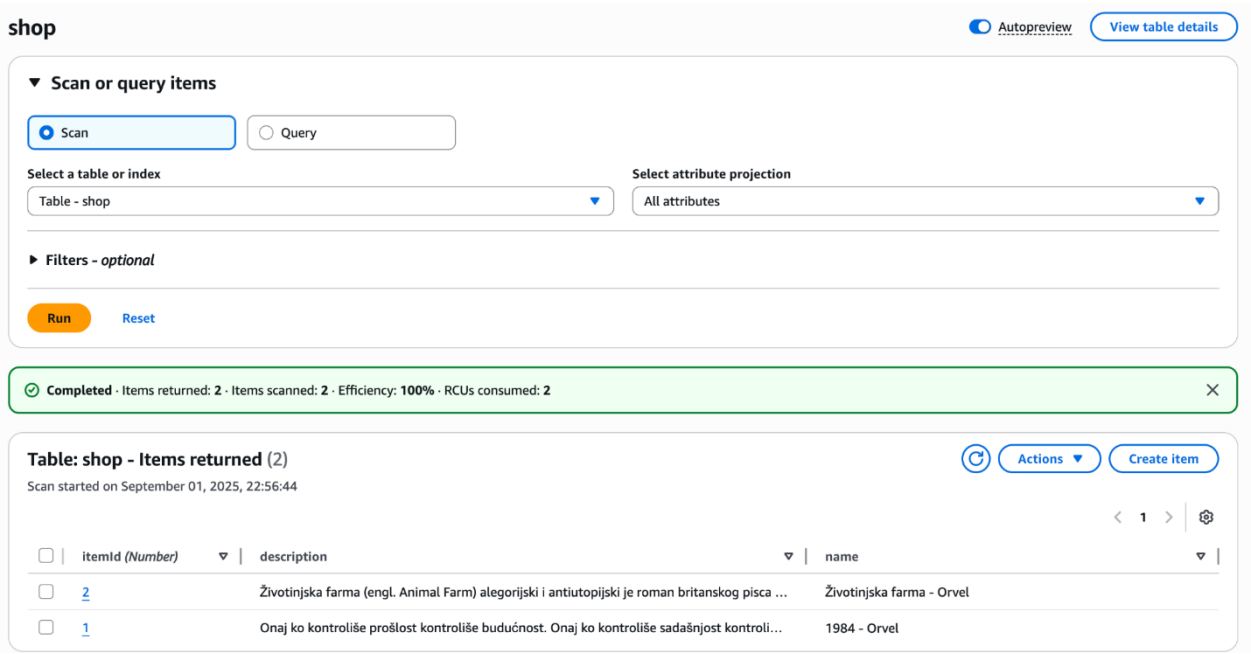


Slika 143 Pregled svih itema u shopu

Primer pregleda svih itema kroz CURL komandu:

```
curl --request GET \
  --url https://2xlze2akkc.execute-api.eu-central-1.amazonaws.com/dev/items
```

Takođe, može se ručno proveriti sadržaj baze (i izmeniti) pristupanjem DynamoDB servisu u okviru AWS konzole (Slika 144).



Slika 144 Prikaz sadržaja DynamoDB baze

Kada se pravi API na AWS Lambdi, postoji nekoliko pristupa, od jednostavnog koda sa ručnim rutiranjem, web framework ili čak jedna funkcija po ruti.

U slučaju **jedne „sirove" Lambda funkcije** to znači da su sve rute unutar funkcije i radi se isto kao u ovom primeru; takav pristup je minimalan, brz za razvoj i za ovakve slučajeve i jako male API implementacije sasvim prihvativ. Međutim, kada broj ruta i metoda po ruti porase, vrlo brzo nastaju komplikacije i postaje teško za održavanje i testiranje.

Kada se koristi pristup „jedna Lambda funkcija po ruti" (npr. getItems, addItem, itd.), dobijaju se bolje **izolovane funkcije** i **preciznije IAM dozvole**, a i **verzionisanje** svake funkcije je jednostavnije. Međutim, nedostaci su: složenije **održavanje** i podešavanje većeg broja funkcija, kao i mogućnost češćih **cold start** problema kod ruta koje se retko pozivaju.

Kod pristupa **Lambda + Web framework** (FastAPI i drugi), koriste se poznati Web frameworci, Lambda se pravi nalik mikro servisu, postoji jasno rutiranje, validacija podataka, automatska dokumentacija, međutim, veće je vreme pokretanja iz cold starta, više zavisnosti i Lambda postaje „teža" pa samim tim zahteva više RAM memorije.

Postoji i opcija **AWS Lambda Powertools;** to nije framework već sklop alata za logovanje, metrike, tracing i slično, ali postoji i podrška za REST API, što omogućuje da funkcioniše slično web framework-ima.

U praksi:

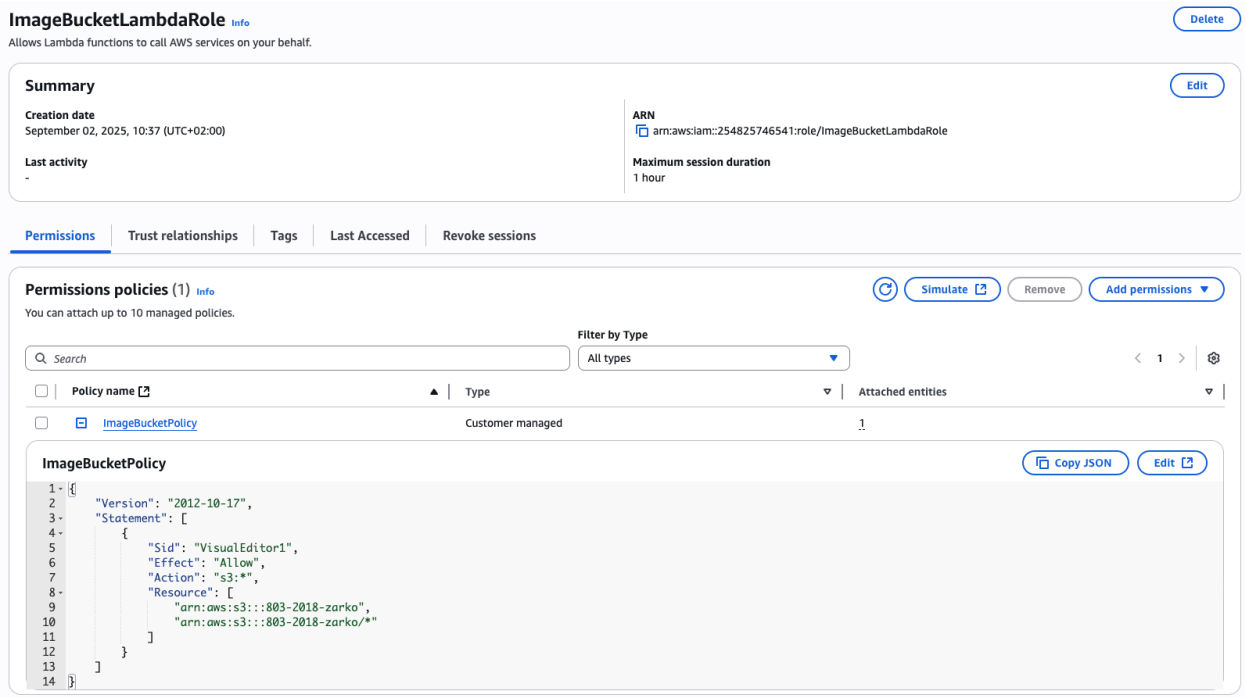
- **Sirova Lambda** funkciju za par ruta ili brze PoCeve
- **FastAPI ili Powertools** ako API raste i potrebna je struktura
- **Jedna Lambda po ruti** kada je bitna bezbednost i precizna kontrola pristupa

S3 Lambda trigger

Ovaj primer pokriva Lambdu koja, kada se uploaduje slika u S3 bucket, konvertuje je u format avatara - 256x256, i uploaduje u /avatars putanju tog bucketa. Bitno je da ima dozvole nad S3 jer kada se dogodi upload, Lambda zapravo prima sam Event koji opisuje šta se desilo i gde, pa samim tim ona mora prvo da povuče taj fajl iz S3, uradi konverziju lokalno i zatim odradi upload nazad na S3.

Prvo je potrebno napraviti S3 bucket prelaskom u S3 servis i dati jedinstveno ime tom bucketu; primera radi, u svrhu ovog zadatka: broj indeksa i ime — 803-2018-zarko. Dovoljno je uneti ime u Bucket name i ne dirati ostala podešavanja, AWS po podrazumevanim podešavanjima blokira javni pristup S3 i koristi SSE enkripciju. Kada se napravi bucket, ući i kopirati iz Properties taba ARN (nalazi se ispod Bucket overview).

U okviru IAM je potrebno napraviti novi Role sa dozvolama za pristup S3. Kako se pravi Polisa i Role može se pogledati u prethodnoj vežbi; na slici Slika 145 je prikazan krajnji rezultat.



Slika 145 IAM Role i policy za Image Lambdu

Sada se može preći na pravljenje Lambda funkcije.

Lambda funkcija će biti Python 3.13, s tim što se sada Lambda oslanja na Pillow biblioteku kako bi transformisala slike, pa samim tim je potrebno da se uz Lambdu spakuje ta biblioteka, tako da će se koristiti Lambda Layer opcija.

Pri pravljenju lambda definisati ime ImageConverter, podesiti Python 3.12 runtime, ostaviti arhitekturu na x86_64 i u execution role odabrati ImageBucketLambdaRole i napraviti Lambdu. U sam kod kopirati sledeće:

```
import os
import json
import urllib.parse
```

```
import boto3
from PIL import Image, ImageOps

s3 = boto3.client("s3")

# Config (tweak via Lambda environment variables if you want)
AVATAR_SIZE = int(os.getenv("AVATAR_SIZE", "256"))
DEST_PREFIX = os.getenv("DEST_PREFIX", "avatars/")

def lambda_handler(event, context):
    print(event)

    # S3 put event → get bucket/key
    record = event["Records"][0]
    bucket = record["s3"]["bucket"]["name"]
    key = urllib.parse.unquote_plus(record["s3"]["object"]["key"])

    # Avoid infinite loop if we're already in the avatars prefix
    if key.startswith(DEST_PREFIX):
        return {"statusCode": 200, "body": json.dumps({"skipped": "already avatar"})}

    # Only process common image types
    ext = key.lower().rsplit(".", 1)[-1] if "." in key else ""
    if ext not in {"jpg", "jpeg", "png", "webp"}:
        return {"statusCode": 200, "body": json.dumps({"skipped": "not an image"})}

    # Paths in /tmp
    src_path = f"/tmp/src.{ext}"
    out_ext = "jpg" if ext == "jpeg" else ext  # keep format (jpeg->jpg)
    out_key = f"{DEST_PREFIX}{key.rsplit('/', 1)[-1].rsplit('.', 1)[0]}_avatar.{out_ext}"
    out_path = f"/tmp/out.{out_ext}"

    print("bucket: ", bucket)
    print("key: ", key)
    print("src_path: ", src_path)
    print("out_ext: ", out_ext)
    print("out_key: ", out_key)
    print("out_path: ", out_path)
```

```
    # Download → resize (square) → upload
    s3.download_file(bucket, key, src_path)

    print("downloaded", src_path)
    with Image.open(src_path) as img:
        # Convert to RGB to avoid issues with PNG/alpha when saving JPEG
        if img.mode not in ("RGB", "L"):
            img = img.convert("RGB")

        # Center-crop + resize to square
        avatar = ImageOps.fit(img, (AVATAR_SIZE, AVATAR_SIZE), Image.LANCZOS)
        avatar.save(out_path, quality=90)

    content_type = {
        "jpg": "image/jpeg",
        "jpeg": "image/jpeg",
        "png": "image/png",
        "webp": "image/webp",
    }[out_ext]

    s3.upload_file(out_path, bucket, out_key, ExtraArgs={"ContentType": content_type})
    print("uploaded", out_path)
    print("uploaded", out_key)
    return {"statusCode": 200, "body": json.dumps({"ok": True, "avatar_key": out_key})}
```

Nakon toga potrebno je da se ide na Deploy, a zatim da se pređe na pakovanje Lambda Layera. Layer se na lokalnoj mašini pravi tako što se biblioteka povuče i zapakuje uz napomenu da je potrebno da je na lokalnoj mašini i Lambdi ista verzija Pythona (3.12) i ista arhitektura (x86_64). Komanda je sledeća:

mkdir -p python

pip install --no-cache-dir Pillow -t python

zip -r layer.zip python

Pomoću ovih komandi se pravi zipovan fajl sa bibliotekom koji je spreman za Layer.

Ukoliko se koristi druga verzija Pythona na lokalnoj mašini ili druga arhitektura (ARM kod SBC ili Mac uređaja), postoje dva pristupa: ili promeniti arhitekturu Lambde na ARM ili pomoću Dockera instalirati kroz kontejner i mounted volume biblioteku. Primer komande:

docker run --rm \

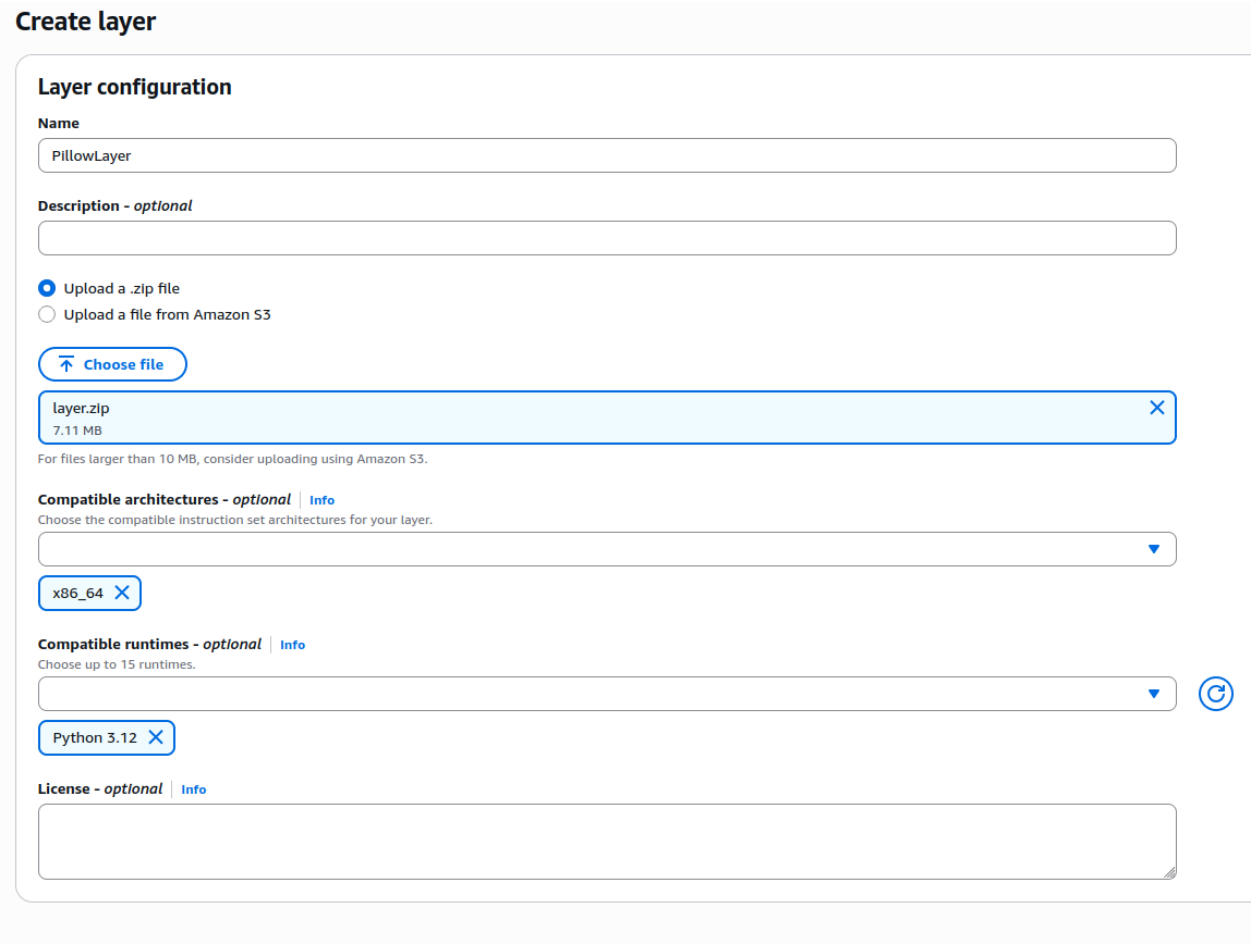
--platform=linux/amd64 \

```
-v "$PWD":/opt_build \
--entrypoint /bin/bash \
public.ecr.aws/lambda/python:3.12 \
-lc "\
  mkdir -p /opt_build/python && \
  python -m pip install --upgrade pip && \
  python -m pip install --no-cache-dir --only-binary=:all: Pillow -t /opt_build/python \
"
```

Zatim treba zipovati sadržaj Python foldera komandom:

zip -r layer.zip python

Sada je potrebno u okviru Lambda servisa u bočnom meniju preći na Layers, pod Additional resources. Dalje treba ići na Create layer i pouniti kao na slici Slika 146; u Upload delu dodati iz lokalne mašine layer.zip fajl koji je prethodno napravljen i ići na Create.

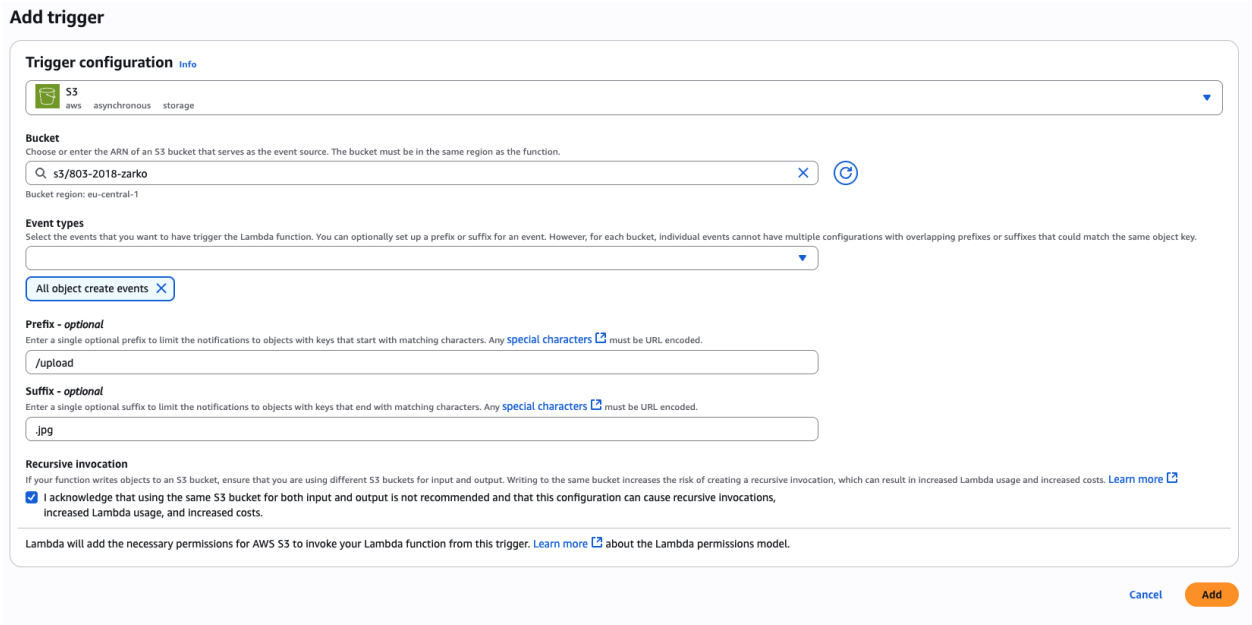


Slika 146 Podešavanje Lambda Layera

Nakon što se Layer napravi, potrebno je vratiti se u functions i u okviru glavnog prikaza Lambde (gde je i sam kod) preći na dno stranice i pod Layers ići na Add a layer. Tu se vidi i opcija da se dodaju AWS

Layeri koji su već spremni; u ovom slučaju odabrati Custom layers, PillowLayer i Version 1. Zatim ići na Add i sačekati da se Lambda ažurira.

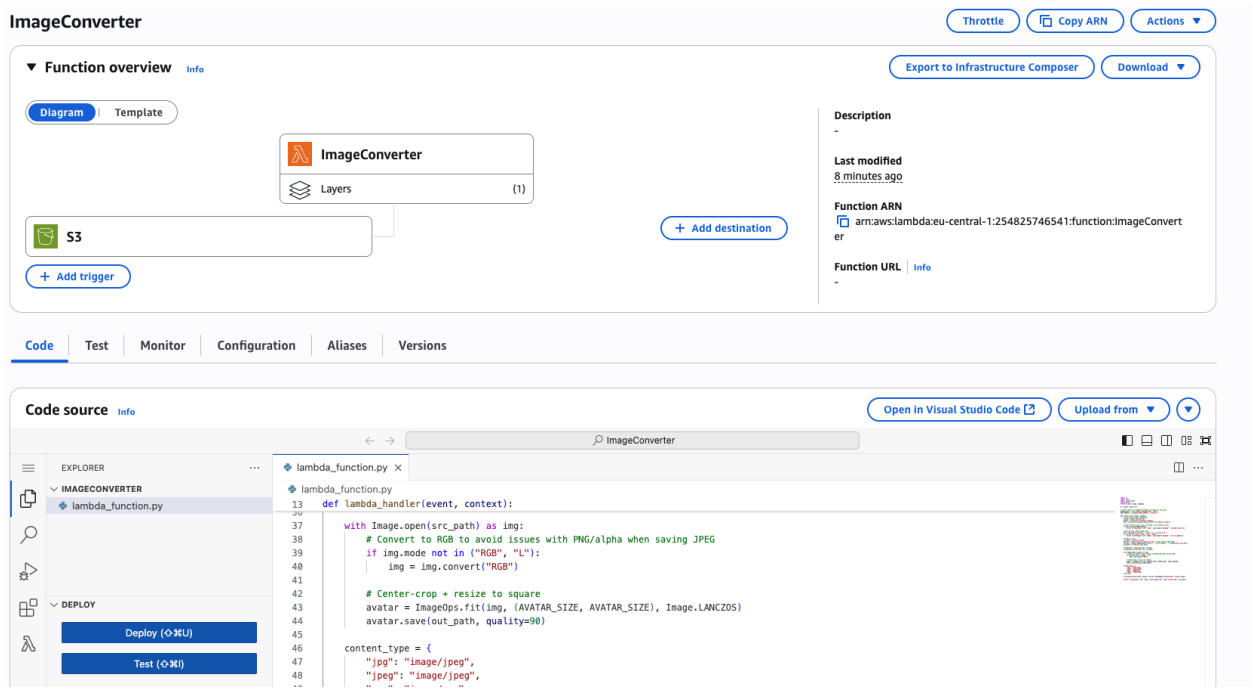
Sada je vreme da se doda lambda trigger. Na početnoj ImageConverter Lambde postoji opcija + Add trigger pa se može već tu ući u proceduru dodavanja, ali postoji i mogućnost dodavanja kroz S3 servis isto tako. U ovom slučaju dodati ga uz + Add trigger oopciju. U select source upisati S3, u Bucket ime prethodno napravljenog Bucketa. Ostatak konfiguracije uneti kao na slici Slika 147.



Slika 147 Konfiguracija S3 triggera

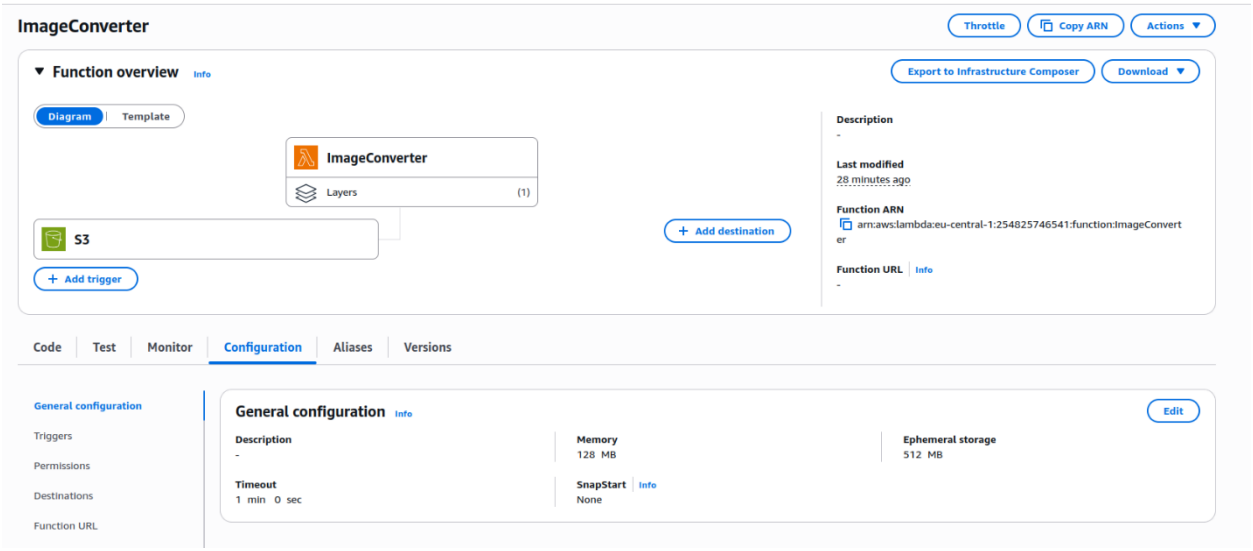
Napomena: Potrebno je definisati Prefix kako ne bi došlo do rekurzivnog okidanja Lambde s obzirom na to da ona sama vrši upload pa bi samim tim ušla u beskonačnu rekurziju konverzija. Samom konfiguracijom definisano da pokupi sve nove objekte iz /upload foldera koji imaju ekstenziju (Suffix) .jpg.

Nakon uspešno dodatog triggera Lambda izgleda kao na slici Slika 148.



Slika 148 Gotova konfiguracija ImageConverter Lambde

U okviru Lambda configuration taba, pa zatim General configuration, potrebno je izmeniti timeout Lambde sa default vrednosti od 3 sekunde na višu vrednost; za slučaj ove Lambde dovoljno je 15 sekundi ili više (Slika 149).



Slika 149 Izmena Timeout vremena na Lambdi

Sada je vreme u S3 da se uploaduje slika. Potrebno je da je slika u odgovarajućem formatu i da se uploaduje u /upload folder (prethodno ga napraviti) u S3 bucketu.

Nakon uspešnog uploada, Lambda bi trebalo u roku od par sekundi da napravi dodatni folder u S3 sa imenom avatars i u okviru njega da smesti konvertovanu sliku sa nastavkom _avatar. Proverom logova Lambde se potvrđuje da je uspešno izvršena (Slika 150).

Slika 150 Logovi izvšenja Lambde

Nova slika se može preuzeti iz S3 bucketa na lokalnu mašinu i pregledom slike i proverom njenih parametara se potvrđuje uspešna konverzija.

Zadaci

Zadatak 1: Napraviti API za beleške sa endpointima za listanje, dodavanje, brisanje i update beleški.

Cilj: Izgraditi REST API za beleške koji ima mogućnost dodavanja novih, brisanje, listanje i update (CRUD) beleški koje se čuvaju u DynamoDB.

Zadatak 2: Lambda Tuning

Cilj: Optimizovati resurse Lambde iz primera 2 (S3 Lambda trigger) ili napraviti novu Lambdu i uraditi optimizacije, kao što su podešavanje Timeout i Memory parametara.

11. Budućnost Računarstva u oblaku

Računarstvo u oblaku je postalo osnova savremene digitalne infrastrukture, omogućavajući skalabilne, fleksibilne i isplative servise u gotovo svakoj industriji. Kako se ulazi u sredinu 2020-ih, razvoj oblikuju postepeno stapanje veštačke inteligencije (AI), edge computinga, 6G mreža, kvantnog računarstva i sve veći zahtevi za održivošću. Ovo poglavlje istražuje trenutnu putanju razvoja računarstva u obliku, tehnologije koje ga transformišu i strateške pravce koji se očekuju do 2030. godine.

11.1. Trendovi u računarstvu u oblaku

Spajanje računarstva u oblaku i veštačke inteligencije menja mogućnosti koje platforme oblaka mogu da ponude. Dobavljači sve češće ugrađuju AI/Machine Learning za optimizaciju radnih opterećenja, prediktivno održavanje, sajber bezbednost i korisničko iskustvo. Ističe se konvergencija AI i 6G kao „transformativni pomak", koji omogućava analitiku u realnom vremenu i autonomno funkcionisanje sistema. Očekuje se da AI-as-a-Service (AIaaS) postane dominantna usluga, omogućavajući čak i malim firmama pristup naprednim AI modelima bez potrebe za stručnim znanjem (Butt&Shah, 2025;.Farhaoui et al, 2025)

Sa sve većim brojem aplikacija koje zahtevaju nisku latenciju, kao što su autonomna vozila, pametne fabrike itd., edge computing postaje nezaobilazan. Budući oblak radiće u hibridnom modelu, tako što će distribuirati inteligenciju između centralizovanih sistema oblaka i decentralizovanih edge čvorova. Naglašava se integracija edge computinga sa cloudom u inteligentnim transportnim sistemima, što poboljšava performanse i obradu podataka. Ovaj hibridni model podržava koncept „Cloud Continuum", gde se resursi distribuiraju između edge, fog i centralnog clouda (Li et al, 2025).

Ekološka održivost postaje ključni pokretač inovacija u cloudu. Data centri, poznati po velikoj potrošnji energije, prolaze kroz temeljne promene. Istraživanja razmatraju inicijative zelenog computinga u Africi, koje uključuju obnovljive izvore energije i optimizaciju putem softverskih rešenja (Siunduh & Otieno, 2025). Veliki cloud provajderi, kao što su AWS i Google Cloud, najavili su postizanje karbonske neutralnosti kroz efikasnije hlađenje, upravljanje resursima uz pomoć AI-a i korišćenje obnovljive energije.

11.2. Tehnologije koje oblikuju cloud

Blockchain širi računarstvo u oblaku na okruženja bez poverenja (trustless). Distribuirani modeli clouda obezbeđuju isporuku servisa (compute, skladištenje, mreža) na geografski udaljenim lokacijama. Blockchain poboljšava transparentnost i integritet podataka u distribuiranom cloudu, naročito u lancima snabdevanja i finansijama. Kombinacija blockchaina i cloud computinga osnova je za oblasti poput DeFi, digitalne zdravstvene evidencije i pametnih ugovora (Ali, 2025). Iako još u ranoj fazi, kvantno računarstvo već utiče na istraživanje i razvoj u cloud industriji. Provajderi poput IBM-a i AWS-a nude kvantne simulatore u okviru svojih cloud servisa. Kvantna infrastruktura clouda omogućiće napredne bezbednosne algoritme i rešavanje složenih problema koji su do sada bili van domašaja (Danish & Siraj, 2025).

Cloud native pristupi sajber bezbednosti se ubrzano razvijaju kako bi odgovorili na sve sofisticiranije pretnje. Napredne platforme za detekciju pretnji koje koriste AI za prevenciju napada u finansijskim sistemima zasnovanim na cloudu već se koriste (Eshra, 2025). Paradigma „confidential computinga", gde su podaci šifrovani čak i tokom obrade, očekuje se da postane standard do 2026. godine, naročito u sektorima zdravstva i finansija.

11.3. Tržišni i strateški pravci

Dobavljači računarstva u oblaku kreiraju specifična rešenja po industrijama, na primer Cloud for Healthcare, Cloud for Retail itd. Ove platforme nude alate, pretreniran AI i prilagođene bezbednosne funkcionalnosti. Ovaj trend „vertikalizacije" omogućava ciljana digitalna rešenja za specifične sektore, čime se ubrzava inovacija (Eshra et al, 2025).

Organizacije se sve više oslanjaju na više dobavljača (multi cloud), zahtevajući veću interoperabilnost. Postoje detaljne analize strategije raspodele zadataka i resursa u multi cloud okruženjima (Awad & Ariffin, 2025). Standardi otvorenog koda, kao što su Kubernetes, Istio itd. olakšavaju prelaz ka neutralnim infrastrukturama bez vendor lock-ina.

Cloud native pristupi, kao što su kontejneri, mikroservisi i serverless computing postaju osnovni način razvoja softvera. Ove tehnologije omogućavaju brzo postavljanje aplikacija, skaliranje po potrebi i smanjeno opterećenje u održavanju. Funkcije kao servis (FaaS) beleže snažan rast i predviđa se trostruko uvećanje do 2027[36].

11.4. Optimizacija raspodele opterećenja

Upravljanje resursima u oblaku zahteva pažljivo balansiranje efikasnosti, pouzdanosti, energetske štednje i poštovanja SLA ugovora, koristeći inteligentno zakazivanje i elastičnu dodelu resursa — posebno u okruženjima sa velikim brojem korisnika i promenljivim opterećenjem. Značaj optimizacije je naročito vidljiv u svetlu povećane potrošnje električne energije zbog sve većih zahteva stavljenih pred računarstvo u oblaku. Predviđanja američkog ministarstva energetike su udvostručenje ili čak utrostručenje potrošnje do 2028. godine (u odnosu na kraj 2024.)[37], a globalne procene su zabrinjavajuće – 2030. godine data-centri će trošiti 13% sveukupne električne energije[38].

Glavna odgovornost za upravljanje resursima u cloud okruženju pripada pružaocima usluga IaaS (Infrastructure as a Service), koji grade i održavaju data centre. Njihova dva ključna cilja su:

1. Pouzdanost i efikasnost infrastrukture — da se resursi koriste optimalno i stabilno.
2. Poštovanje ugovora o nivou usluge (SLA) — što reguliše odnos sa krajnjim korisnicima.

Stabilno i efikasno upravljanje infrastrukturom obuhvata sledeće:

- Balansiranje opterećenja: Ravnomerno raspoređivanje zadataka kako bi se izbeglo preopterećenje nekih servera, a drugi ostali neaktivni.
- Otpornost na greške (fault tolerance): Sistem treba da bude postavljen tako da kvar jednog resursa ne utiče ozbiljno na ceo sistem.
- Ušteda energije: Smanjenjem potrošnje energije smanjuju se troškovi, ali i emisije ugljen-dioksida.

Dobavljač usluga u oblaku može birati između:

- Sveobuhvatnog pristupa koji optimizuje sve faktore zajedno (performanse, energija, balans) ili

[36] https://www.intellectualmarketinsights.com/report/function-as-a-service-market-size-and-share-analysis/imi-008324

[37] https://www.energy.gov/articles/doe-releases-new-report-evaluating-increase-electricity-demand-data-centers

[38] L. de Roucy-Rochegonde and A. Buffard, *AI, Data Centers and Energy Demand: Reassessing and Exploring the Trends*, Ifri Papers, French Institute of International Relations (Ifri), Paris, France, Feb. 2025. ISBN: 979-10-373-1000-2.

- fokusiranja na pojedinačne ciljeve, zavisno od opterećenja sistema (npr. tokom male potražnje akcenat je na štednji energije, a tokom velike na performansama).

Glavni izazov je pronaći ravnotežu između visokih performansi i energetske efikasnosti, a da korisnički kvalitet ne trpi.

U svrhu optimizacije potrošnje resursa, u upotrebi su sve više napredne metode, koje obuhvataju mašinsko učenje i metaheuristike.

U radu predstavljena je optimizacija korišćenja resursa primenom kombinovanih metoda mašinskog učenja, detekcije anomalija i algoritma Particle Swarm Optimization – za izbor najpovoljnije konfiguracije resursa. Predstavljena je kod Deeplaxmija i saradnika (2024). Vrši se horizontalno skaliranje (dodavanje više instanci resursa), vertikalno skaliranje (povećanje kapaciteta jedne instance) i prilagođava konfiguraciju u skladu sa promenama cena dobavljača usluga u oblaku. Model je testiran u realnom okruženju — na platformi MS Azure. Pokazalo se da daje uštede do 85%, kao i da je značajno efikasniji od Azure-ovog servisa autoscale.

11.5. Kvantno računarstvo u oblaku

Kvantno računarstvo koristi principe kvantne mehanike za izvođenje proračuna. Za razliku od klasičnih računara koji koriste bitove (0 ili 1), kvantni računari koriste kubite, koji mogu biti u superpoziciji stanja (istovremeno i 0 i 1). Ovo svojstvo, zajedno sa tzv. kvantnim preplitanjem, omogućava kvantnim računarima da potencijalno rešavaju određene probleme mnogo brže od klasičnih računara. Neki od primera primene su:

- Analiza velikih podataka — efikasna obrada ogromnih skupova podataka.
- Bezbednost — sigurniji prenos i čuvanje podataka uz kvantnu enkripciju.
- Mašinsko učenje (ML/DL) — brža obuka i veća preciznost modela.
- Biotehnologija i farmacija — dizajn lekova i rad sa DNK/genima.

Kao primer značaja kvantnog računarstva, može se izdvojiti uticaj na kriptografiju. Kvantni računari su u stanju da u dostižnom vremenu provale mnoge aktuelne algoritme šifrovanja, te se stoga aktivno radi na zameni novim algoritmima, koji su otporni i na napade uz pomoć kvantnih računara.

Kvantni računari su i dalje izuzetno skupi, te model korišćenja u oblaku predstavlja rešenje koje je isplativo i za dobavljače i za krajnje korisnike. Pionir ovog pristupa jeste IBM, koji je 2016. pokrenuo svoj servis IBM Quantum Experience[39]. Sledio je razvoj programske podrške za korišćenje kvantnih računara u popularnim jezicima. Najpoznatije komercijalne platforme računarstva u oblaku su ubrzo ponudile svoje usluge, te tako Amazon nudi AWS Braket, Microsoft Azure Quantum, Google Quantum Cloud itd.

Iako obećava, kvantno računarstvo u oblaku suočava se s nizom prepreka(Golec et al., 2024):

- Nerazvijene praktične primene — za robote, kvantnu kriptografiju itd.
- Skalabilnost — Teško je povećati broj kvantnih resursa kao u klasičnom oblaku.
- Visoki troškovi proizvodnje — Kvantni računari su i dalje ekstremno skupi.
- Privatnost podataka — Postoje rizici da kvantna snaga ugrozi današnje šifre.
- Bezbednost komunikacije — Potrebni su novi protokoli za siguran prenos podataka.

[39] Danas je aktivna IBM-ova platforma (uz mogućnost besplatnog isprobavanja): https://quantum.cloud.ibm.com/

- Tehnološka nejednakost — Razvijene zemlje će brže usvojiti ovu tehnologiju, što može povećati globalne razlike i uticati na tržište rada.

Računarstvo u oblaku se razvija izvan svoje tradicionalne uloge kao infrastruktura i postaje strateški stub AI razvoja, održivosti i inteligentnih sistema budućnosti. Njegova budućnost leži u decentralizaciji, autonomiji i energetskoj efikasnosti. Kombinovanjem blockchain-a, kvantnog računarstva i edge computinga, cloud postaje osnova sledeće generacije digitalnih inovacija.

Dodatak A: AWS Naplata (Billing)

U AWS-u svaki servis meri potrošnju u svojim jedinicama — vreme procesora, memoriju preko trajanja poziva, gigabajt-mesece skladištenja, broj zahteva, količinu izlaznog saobraćaja — i te brojke se, na kraju meseca, pretvore račun. Ako se u jedan nalog uveže više timova ili okruženja, račun postaje slika kompletne organizacije koja pokazuje koja aplikacija je zahtevna na mreži, ko troši skladište, a ko procesorsko vreme. Razumevanje Billinga je zato kritično za razumevanje; kako bi bilo jasno odakle ti troškovi dolaze i na koji način bi mogli biti optimizovani kako bi se trošak smanjio, takođe, bitno je razumeti i koliko će neki novi feature ili komponenta povući dodatnih troškova kako bi imali odgovarajuća očekivanja i potencijalno planirali naplatu tog novo featurea.

Kako funkcioniše Billing

Na najvišem nivou, plaća se u okviru pojedinačnog naloga ili preko payer naloga u **AWS Organizations** (konsolidovana naplata). Unutar tog okvira AWS vodi fine metrike po servisu, regionu i vrsti potrošnje. Dve stvari su ovde kritične, a to su **označavanje resursa (tagovi)** i **model naplate**. Tagovi (npr. Env=prod, Owner=zarko, CostCenter=EDU) ne znače ništa dok ih ne aktivirate u Billing-u kao **Cost Allocation Tags**, a nakon toga postaju dimenzije u izveštajima. A model naplate je, u osnovi, izbor između plati-koliko-koristiš (**On-Demand**), obaveži se i plati manje (**Savings Plans/Reserved Instances**) ili rizikuj prekide za znatno nižu cenu (**Spot**). U praksi, zrela postavka kombinuje sve troje.

Kada se otvori **Billing & Cost Management** konzola, prikazuje se nekoliko uvida. **Cost Explorer** pretvara potrošnju u grafikone i preseke po servisu, regionu, nalogu, tagu, po danima ili mesecima, sa prognozom (estimate) do kraja perioda. Vide se i različiti pogledi na cenu, **Unblended** — što je realno obračunato tog dana, **Amortized** — koji ravnomerno raspoređuje efekat rezervacija i planova štednje kroz vreme i potrošače, pa i metrike pokrivenosti i iskorišćenja za Savings/RI koji pokazuju koliko sati je pokriveno obavezom i koliko je ta obaveza zapravo korišćena. Tu je i **Anomaly Detection** koji nudi model za uočavanje neuobičajenog skoka, na primer,p+ izlazni promet S3 u porastao 300%, i šalje upozorenje pre nego što račun postane iznenađenje.

Tamo gde grafikoni nisu dovoljni, pomaže **Cost & Usage Report (CUR)**. To je detaljan dnevnik svih jedinica potrošnje, spušten do pojedinačnih linija, koji se sliva u S3. Odatle, uz pomoć Glue i Athene, trošak se može pretvoriti u upite (query) i time se Billing pretvara u analitiku.

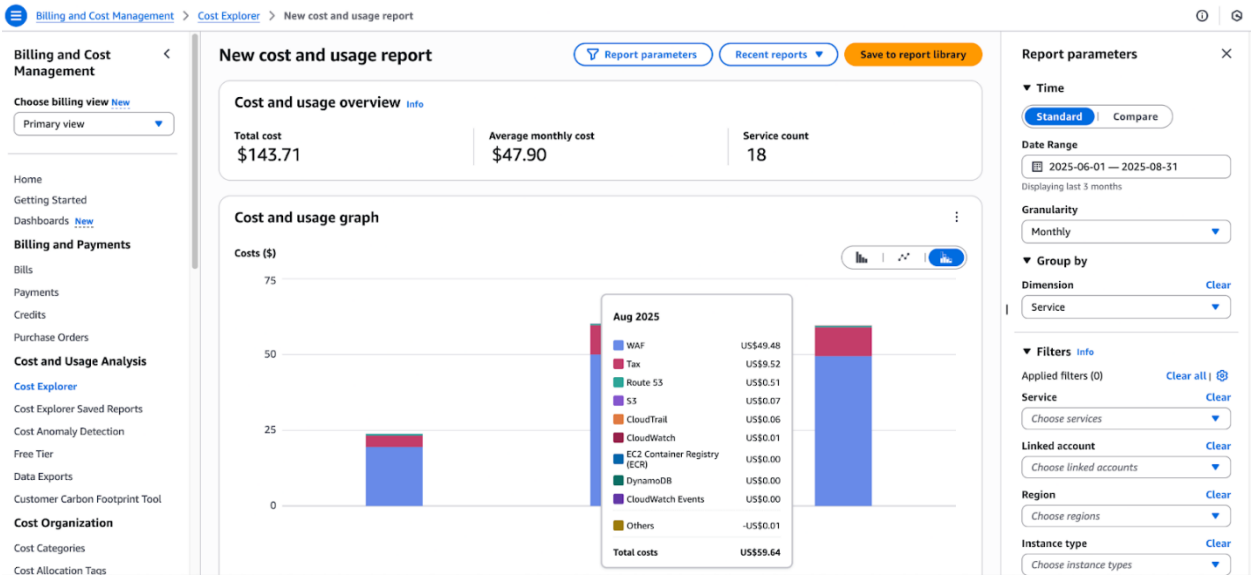
Većina korisnika koji se upuštaju u AWS po prvi put očekuju da će EC2 i RDS biti najkrupniji troškovi, kao što i često jesu. Ali najveća iznenađenja dolaze iz **saobraćaja** i „lepka" između servisa. Izlaz ka internetu (Data Transfer Out) se naplaćuje, pa i mala, stalna „curkanja" vremenom narastu. **NAT Gateway** se plaća i po satu i po gigabajtu, aplikacije koje stalno povlače podatke preko NAT-a lako naprave veliki trošak, i to u servisima koji nisu direktno vidljivi. Zato je izuzetno bitno pametno pravljenje mreže; na primer, ako privatnim funkcijama ili kontejnerima treba S3, DynamoDB ili drugi AWS servis, treba dodati **VPC endpointe** da promet ostane u mreži oblaka (bez NAT-a). Ako se javnosti služi statički ili polustatički sadržaj, **CloudFront** približava odgovore korisnicima i smanjuje egress iz regiona. Ove odluke deluju mrežno, ali završavaju kao linije na računu.

Drugi čest izvor zabune je kako se neki servisi obračunavaju. Za S3, na primer, plaća se gigabajt-mesec, tj. koliko dugo koliko podataka stoji, broj zahteva (PUT/GET/LIST) i egress. Za Lambda se plaća broj poziva i trajanje u zavisnosti od memorije, a za API Gateway kombinaciju poziva i dodataka kao što su keš i web-soketi. To je razlog zašto je korisno već u dizajnu definisati budžete kao što su koliko poziva se očekuje, koliki cache hit, kakav je cilj za p95 latenciju i koliko to košta. Tek tada optimizacije imaju smisla.

Zrela disciplina troška počinje od starta tako što svaki resurs dobija tagove, a budžeti se postavljaju na nivou naloga, tima i ponekad servisa. Budžet nije kazna već ga treba posmatrati kao rani alarm. Ako dev okruženje ima budžet od 50 USD mesečno, a 20. u mesecu stigne upozorenje da je na 80%, to je znak da se implementiraju neke automatizacije kao gašenje dev resursa noću, skraćeni životni ciklusi sandbox resursa itd. Isti princip važi i za Anomaly Detection; sistem pazi na neočekivana kretanja, korisnik proverava hipoteze u Cost Exploreru i, ako treba, u CUR-u.

Sledeći korak je **showback/chargeback**. Ako više timova deli isti *payer* nalog, pošteno je da svako vidi ili plati svoj deo. Tu tagovi i **Cost Categories** daju elegantnu podelu troškova u izveštajima. Na nivou infrastrukture, Savings Plans i rezervacije daju popuste dugoročno predvidivim delovima potrošnje, dok se elastični, povremeni delovi i dalje koriste on-demand ili spot. Ključno je posmatrati **coverage** i **utilization**; nije cilj zakupiti što više obaveze, već zakupiti tačno onoliko koliko se stvarno koristi.

Jako je bitno pridržavati se najboljih praksi i određenih pravila. Jedno od njih je **šta nije u kodu, ne postoji**. Tagove, budžete i alarme uvoditi preko IaC i CI/CD, nipošto ručno. Drugo, **misliti u dimenzijama**, svaki grafikon posmatrati kroz servis, region, tim i okruženje, ista cifra izgleda drugačije u razvoju i u produkciji. Treće, **simptom - hipoteza - dokaz**, kada nešto „poskupi“, Anomaly obaveštava, Cost Explorer daje hipotezu, a CUR daje dokaz.



Slika 151 Cost Explorer — raspodela mesečnog troška po servisima

Kratak primer

Zamisliti tim koji održava studentski portal i odjednom stiže poruka da trošak raste. Anomaly Detection obaveštava da je „Data Transfer Out“ u S3 (eu-central-1) neobično visok. U Cost Exploreru potvrditii skok baš u tom servisu i regionu. CUR, upit (query) kroz Athenu, otkriva da se velike datoteke preuzimaju direktno iz aplikacije (preko NAT-a), ne kroz CloudFront. Rešenje je u tri koraka:

1. Postaviti **CloudFront** iznad S3 uz **OAC** (da se S3 ne može zaobići);
2. Definisati keš politiku koja izdvaja statičke rute sa dugim TTLom ;
3. Dodati **VPC endpoint** ka S3 da se ukloni NAT trošak za server-to-S3 saobraćaj.

Sa ove tri izmene u konfiguraciji troškovi će se sigurno značajno smanjiti.

Dodatak B: AWS Organizations i Landing Zone

U jednom AWS nalogu može se „daleko dogurati“, ali već na prvim ozbiljnijim projektima javlja se potreba za **granulacijom odgovornosti**, jasnim **granicama rizika** i **odvojenim troškovima** po timovima i okruženjima. Tu na scenu stupa ***AWS Organizations***, servis koji skuplja više naloga pod zajedničko upravljanje i obezbeđuje polise, bezbednost i obračun na nivou **cele organizacije**. Na Organizations se prirodno nadovezuje ***AWS Control Tower***, gotov landing zone[40] koji automatizuje početno postavljanje multi-account okruženja, uvodi standardne naloge (log arhiva, audit) i uključuje čuvare *(guardrails)* protiv najčešćih propusta. Zajedno, oni čine jake osnove na kojima ostatak cloud arhitekture može bezbedno da raste.

U pristupu sa jednim AWS nalogom, svi resursi i ljudi dele isti set dozvola, kvota i budžeta. Ako se nešto pogrešno obriše ili previše otvori, ***blast radius*** je ceo nalog. U multi account pristupu, okruženja (dev/stage/prod), poslovne linije, pa čak i visokorizični eksperimenti dobijaju **sopstvene naloge**. Time se pojednostavljuju IAM polise, jasnije se prate troškovi i, što je najvažnije, incidenti ostaju u svom sandboksu.

U korenu je **management/payer** nalog (najčešće zvan **root** nalog). Ispod njega se prave **organizacione jedinice (OU)** koje grupišu naloge po svrsi, na primer, Security, Infrastructure, Shared Services, Sandbox, Workloads/Prod/Qa/Dev itd. Na te OU-ove i pojedinačne naloge kače se tipovi polisa koje Organizations koristi:

- **Service Control Policies (SCP)** — Gornji plafon dozvola koje mogu **zabraniti** klase akcija bez obzira na lokalne IAM dozvole u nalogu (ali ne mogu dati nova prava). Primer: zabranjeno je kreirati javne S3 bukete ili dozvoljeni su samo regioni EU.
- **Tag Policies** — Standardizuju nazive i vrednosti tagova (na primer Owner, CostCenter, Env), da bi izveštavanje i automatske kontrole imali tačne metapodatke.
- **Backup Policies** — Opisuju očekivane rasporede i rokove čuvanja rezervnih kopija u okviru organizacije.
- **AI services opt-out policy** -— Centralno podešavanje da li pojedini AI servisi imaju dozvole da koriste sadržaj za unapređenje modela (gde je primenljivo).

Kroz **delegated administrator** mehanizam dodeljuje pojedinim nalozima ulogu „centralne kontrole“ za određene servise (na primer, neko zadužen za **AWS Config**, **CloudTrail**, **GuardDuty**, **Security Hub**, **Firewall Manager**, **Backup** ili **Cost & Usage** izveštaje) i dobija organizacione preglede i jedinstveno upravljanje.

Za pristup korisnika i timova preporučuje se federacija preko **IAM Identity Center-a** (nekada se zvao AWS SSO), gde se definišu ***permission* setovi** (skupovi dozvola), a korisnici ulaze u ciljne naloge i uloge jednim klikom, bez statičnih pristupnih ključeva. Za deljenje resursa preko naloga, tu je **AWS Resource Access Manager (RAM)**; na primer, centralni *Transit Gateway, License Manager*, deljene Route 53 privatne zone ili KMS ključevi.

[40] *Landing zone* je uređeno i bezbedno polazište u oblaku koje sadrži strukturu naloga, mrežnu arhitekturu, politiku bezbednosti, upravljanje identitetima itd. To je osnova, na kojoj se posle grade konkretne aplikacije, baze, virtuelne mašine itd.

Naravno, moguće je samostalno napraviti i podesiti AWS Organizations, CloudTrail, Config i guardrails, ali **Control Tower** to uradi umesto korisnika, dosledno i proverljivo. On postavlja:

- **Standardne naloge**: *Log Archive* (svi CloudTrail/Config dnevnici „padaju“ u jednu tačku) i *Audit* (nalog za bezbednosni tim, sa read-only uvidom i alatima).
- **Guardrails**: Skup **preventivnih** (SCP) i **detektivnih** (Config pravila) mera: od „zabrani javne S3 buckete“ i „samo odobreni regioni“, do „CloudTrail uvek uključen“ i „KMS ključevi rotirani“.
- **Account Factory**: „Mašina za izradu naloga“ — obrazac kojim standardizujete kako nastaje novi nalog (OU, baseline, mreža, tagovi, inicijalne role).
- **Enrollment**: Mogućnost da postojeće naloge „uvedete“ u kontrolisano okruženje i nad njima primenite guardrails.

Landing Zone Accelerator (LZA)

Landing Zone Accelerator on AWS (LZA) je referentna implementacija iznad Control Tower-a, koja spaja zahteve vezane za bezbednost i usaglašenost (na primer NIST, CIS, ISO) u spreman paket: mrežni *hub-and-spoke*[41], centralni DNS/Resolver, *Firewall Manager* politike, *conformance* pakete za Config, integracije sa Security Hubom, GuardDutyjem i Detectiveom, pa i tipične „shared services“ (centralizovani skeneri, repozitorijumi, ECR mirror itd.). Ako Control Tower predstavlja kostur, LZA dodaje mišiće i reflekse za organizacije kojima compliance nije „nice-to-have“, već uslov postojanja.

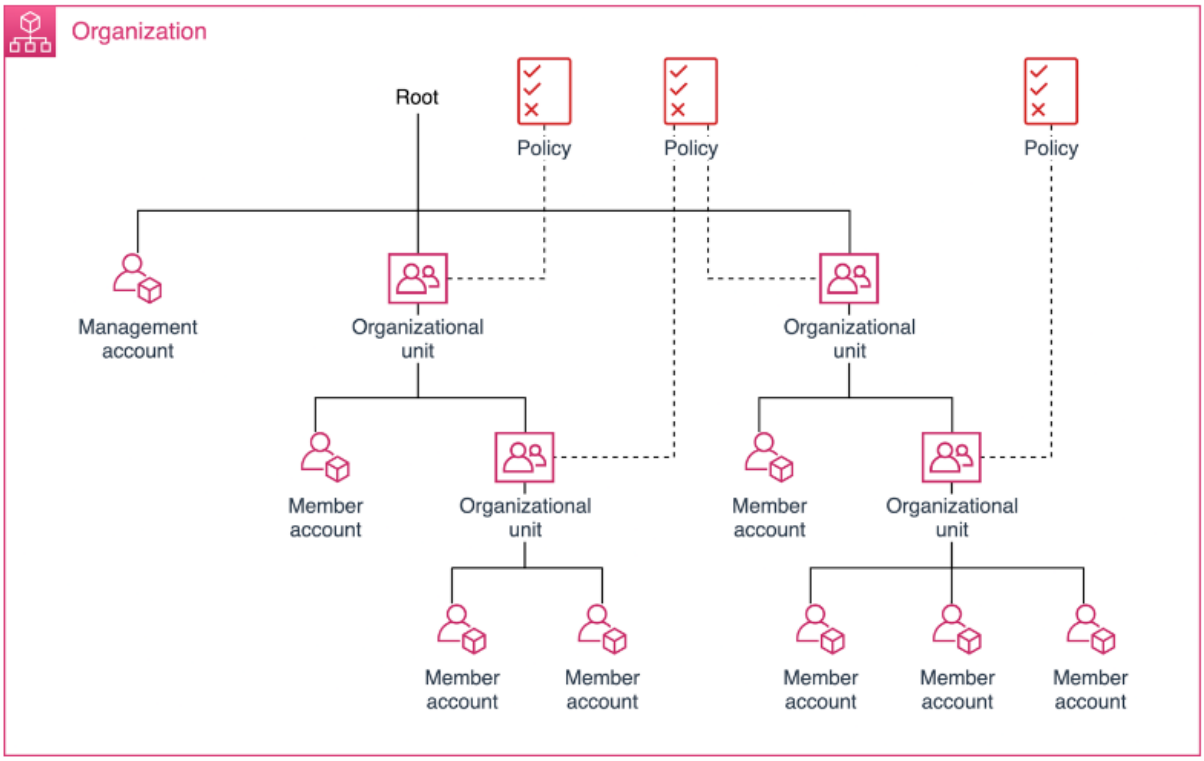
Pored LZA, srodni servisi u domenu organizacione kontrole su **AWS CloudTrail (organization trail)**, **AWS Config Aggregator**, **AWS Firewall Manager** (centralno sprovođenje VPC/ALB/WAF mrežnih polisa), **Security Hub** (skuplja nalaze čitavog bezbednosnog ekosistema), **GuardDuty** (otkriva indikatore), **Audit Manager** (automatizuje prikupljanje dokaza za revizije) i **Service Catalog** (katalog odobrenih šablona koji timovi smeju da podižu).

Kako izgleda dobar OU

Zamisliti fakultet ili firmu sa već dve aplikacije u produkciji i trećom u razvoju. U vrhu je **Management** nalog. Ispod njega OU **Security** sa nalozima Log Archive i Audit. OU **Infrastructure** nosi Network Hub (Transit Gateway, centralni DNS/Resolver), Shared Services (artifakti, ECR, centralni CI) i Monitoring. OU **Workloads** se deli na **Prod** i **NonProd**, svaki sa sopstvenim nalozima po aplikaciji. Konačno, OU **Sandbox** je igraonica sa VPC budžetom. Na Prod OU primenjuju se stroža pravila (samo EU regioni; zabranjeni su „eksperimentalni“ servisi ako ih polisa ne dozvoljava), dok je NonProd liberalniji, ali ipak sa obaveznim logovanjem i enkripcijom.

U takvoj slici, kada tim traži novi nalog za sledeći projekat, **Account Factory** može da ga izbaci iz template-a, već je povezan na centralni log, ima *baseline* mrežu, osnovne IAM role i guardrails. Security tim ne mora svaki put iz početka da proverava da li je CloudTrail uključen i ostali servisi podešeni, već Control Tower i Config to garantuju i nadgledaju.

[41] Razgranati model mreže koji centralizuje bezbednost i decentralizuje posebne mreže, namenjene različitim organizacionim celinama.



Slika 152 Opšta šema jednog OU

Najčešća greška je **jedan nalog za sve** koji se godinama pretvori u džunglu polisa i secret-a. Druga je **ručno pravljenje naloga**, bez Account Factory-a i IaC. Treća je **SCP kao kontrola za sve**; setiti se da su i **Config** i **Firewall Manager** deo kontrole, dok SCP najbolje radi kao preventivni okvir. I najvažnije, **root korisnik** u management nalogu mora biti zaključan, sa MFA, bez access ključeva, i break-glass procedurom.

Sa **Consolidated Billing-om**, svi povezani nalozi doprinose većim popustima *(tiered pricing),* a i dalje se vidi trošak po nalogu/OU/tagu. **Cost Explorer** i **CUR** postaju još bitniji, jer može se raditi showback/chargeback prema timovima. **Anomaly Detection** „hvata izlete“ u bilo kom nalogu, **Budgets** se postavlja i na Sandbox OU i na pojedinačne produkcione naloge. U logičkom sloju, **Organization Trail** i **Config Aggregator** pružaju jedinstven black box, što znači da niko više ne gasi audit na svom nalogu neopaženo.

SCP primeri

Dozvoli samo EU region (u Prod OU-u):

```
{
 "Version": "2012-10-17",
 "Statement": [{
  "Sid": "DenyOutsideEU",
  "Effect": "Deny",
  "Action": "*",
  "Resource": "*",
```

```
   "Condition": {
    "StringNotEquals": { "aws:RequestedRegion": [ "eu-central-1", "eu-west-1", "eu-west-2", "eu-north-1" ] }
   }
 }]
}
```

Zabrani javne S3 buckete, na celoj organizaciji, osim određenim servisima:

```
{
 "Version": "2012-10-17",
 "Statement": [{
   "Sid": "DenyS3Public",
   "Effect": "Deny",
   "Action": [ "s3:PutBucketAcl", "s3:PutBucketPublicAccessBlock", "s3:PutBucketPolicy" ],
   "Resource": "arn:aws:s3:::*",
   "Condition": { "Bool": { "aws:PrincipalIsAWSService": "false" } }
 }]
}
```

Napomena: SCP ne daje dozvole, već samo **ograničava maksimum**. Lokalni IAM i dalje mora da dozvoli ono što treba da radi. Ako SCP nešto ograniči, lokalna IAM polisa to ne može otključati.

LITERATURA

Ali, A. (2025). Advancements and transformative applications of blockchain technology. *Journal of Engineering and Computational Intelligence Review, 3*(1), 36–51 PDF

Aluvalu, R., & Maheswari, U. V. (Eds.). (2024). *Serverless computing: Concepts, technology and architecture*. IGI Global. ISBN-13: 979-8369316825.

Awad, K., Zainol Ariffin, W., Nazri, K. A., Ahmad, M. Z., & Yassen, E. T. (2025). Resource allocation strategies and task scheduling algorithms for cloud computing: A systematic literature review. *Journal of Intelligent Systems, 34*(1), 20240441. https://doi.org/10.1515/jisys-2024-0441

Bhowmik, S. (2017). *Cloud Computing*. Cambridge University Press. ISBN-13: 9781316638101.

Butt, I. N., & Shah, S. (2025). The Convergence of AI and 6G. *Journal of Engineering and Computational Intelligence*. PDF

Coronel, C., & Morris, S. (2018). *Database systems: Design, implementation, & management* (13th ed.). Cengage Learning. ISBN-10: 1337627909.

Danish, M., & Siraj, M. M. (2025). AI and Cybersecurity: Defending Data and Privacy in the Digital Age. *Journal of Engineering and Computational Intelligence Review*, *3*(1), 25-35. PDF

Deeplaxmi, P., Vikas, C., & Medishetti, S. K. (2024). Resource optimization in cloud environment using advanced metaheuristic scheduling algorithm. In *2024 International Conference on Sustainable Communication Networks and Application (ICSCNA)* (pp. 513–520). IEEE. https://doi.org/10.1109/ICSCNA63714.2024.10863932

Erl T. (2024) *Cloud Computing: Concepts, Technology, Security & Architecture,* Pearson, 2024, ISBN 978-0-13-805225-6

Eshra, S. A., Zohora, F. T., Akter, S., Rasul, I., & Hossain, A. (2025). The Role of Threat Intelligence in Preventing Financially Motivated Cyberattacks. *Journal of Engineering and Computational Intelligence Review*, *3*(2), 20-37.. PDF

Farhaoui, Y., et al. (2025). *Intersection of AI, Data Science, and Cloud*. Springer Book. Google Books

Ghemawat, Sanjay, Howard Gobioff, and Shun-Tak Leung (2003). The Google file system. *Proceedings of the nineteenth ACM symposium on Operating systems principles*.

Golec, M., Hatay, E. S., Golec, M., Uyar, M., Golec, M., & Gill, S. S. (2024). Quantum cloud computing: Trends and challenges. *Journal of Economy and Technology, 2*, 190–199. https://doi.org/10.1016/j.ject.2024.05.001

Golightly L, Chang V, Xu QA, Gao X, Liu BS (2022). Adoption of cloud computing as innovation in the organization. *International Journal of Engineering Business Management*;14. doi:10.1177/18479790221093992

Grance, T., & Jansen, W. (2011). *Guidelines on security and privacy in public cloud computing* (NIST Special Publication 800-144). National Institute of Standards and Technology. https://doi.org/10.6028/NIST.SP.800-144

Grande, R., Vizcaíno, A., & García, F. O. (2024). Is it worth adopting DevOps practices in global software engineering? Possible challenges and benefits. *Computer Standards & Interfaces*, 87, 103767. https://doi.org/10.1016/j.csi.2023.103767

Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Partridge, K., & Sengupta, S. (2010). VL2: A scalable and flexible data center network. *Proceedings of the 9th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS 2010)*, 1–12. https://doi.org/10.1145/1594977.1592576

Kavis, M.J. (2014) Architecting the cloud – Design decisions for cloud computing service models (SaaS, PaaS, and IaaS), Wiley, 2014, ISBN 978-1-118-61761-8

Li, H., Sahrani, S., Sarker, M. R., & Xiao, Y. (2025). State-of-the-art on IoV-based deep learning framework for enhanced driving behavior recognition: Recent progress, technology updates, challenges, and future direction. *IEEE Access, 13*, 135969–135989. https://doi.org/10.1109/ACCESS.2025.3594091

Marinescu, D. C. (2022). *Cloud computing: Theory and practice* (3rd ed.). Elsevier. ISBN-13: 978-0323852777.

Mell, P., & Grance, T. (2011). The NIST definition of cloud computing. *NIST Special Publication 800-145, https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf*

O'Brien, C., et al. (2025). Digital Transformation in HRM, *Current Psychology,* Springer

Popek, G. J.; Goldberg, R. P. (1974) Formal requirements for virtualizable third generation architectures. *Communications of the ACM*. ***17*** *(7): 412–421. doi:10.1145/361011.361073. S2CID 12680060*

Sandhu, A. K. (2022). Big data with cloud computing: Discussions and challenges. *Big Data Mining and Analytics,* 5(1), 32–40. https://doi.org/10.26599/BDMA.2021.9020016

Siunduh, E. S., & Otieno, V. M. (2025). Trends in Green Computing in Africa. *International Journal of Scientific Research & Engineering Trends* PDF

Stallings W., *Osnove bezbednosti mreža: aplikacije i standardi,* CET, Beograd, 2014, ISBN 978-86-7991-376-0.

Subashini, S., & Kavitha, V. (2011). A survey on security issues in service delivery models of cloud computing. *Journal of Network and Computer Applications*, 34(1), 1–11, ISSN 1084-8045

Štrumberger i., Bačanin-Džakula N. (2021). *Klaud računarstvo*, Univerzitet Singidunum, Beograd, ISBN: 978-86-7912772-3

Tate, J., Beck, P., Ibarra, H. H., Kumaravel, S., & Miklas, L. (2017, December 19). *Introduction to Storage Area Networks* (9th ed.). IBM Redbooks. ISBN-13: 9780738442884. https://www.redbooks.ibm.com/abstracts/sg245470.html

Wittig, M., & Wittig, A. (2023). *Amazon Web Services in Action: An in-depth guide to AWS* (3rd ed.). Manning Publications. ISBN-13: 978-1633439160.

INDEKS POJMOVA

Indeks slika