Curs 9
Administrarea bazelor de date pe Linux

9.1 Fundamentele Linux pentru administrarea bazelor de date

Pentru a administra eficient baze de date pe servere moderne, este esențial să înțelegem conceptele de bază ale sistemului de operare **Linux**.

Majoritatea serverelor de baze de date din mediul enterprise și cloud rulează pe Linux datorită stabilității, securității și performanței sale ridicate. Un administrator de baze de date (DBA) trebuie, așadar, să stăpânească comenzile și noțiunile elementare ale acestui sistem de operare pentru a putea instala, configura, optimiza și depana SGBD-urile.

Așa cum subliniază un ghid dedicat, există un set de comenzi Linux de bază „pe care trebuie să le cunoști ca administrator de sistem sau de baze de date" pentru a gestiona rapid și eficient orice server Linux[1][2].

Rolul sistemului de operare în gestionarea SGBD-urilor

Linux, ca orice sistem de operare de tip Unix, gestionează resursele fizice (procesor, memorie, stocare, rețea) și oferă **abstracții** (fișiere, procese, socket-uri etc.) folosite de procesele aplicațiilor – inclusiv de serverele de baze de date. Un fapt important este că în Linux „totul este tratat ca un fișier" – de la dispozitivele de stocare la pipe-uri și conexiuni de rețea[3] – ceea ce înseamnă că interacțiunea SGBD cu resursele trece prin sistemul de fișiere și serviciile kernel-ului. De exemplu, un fișier de baze de date pe disc, un port TCP pe care ascultă serverul sau un socket UNIX folosit pentru conexiuni locale sunt toate reprezentate ca fișiere în sistemul Linux. Prin urmare, administrarea BD implică în mare măsură administrarea corectă a resurselor la nivel de OS: alocarea memoriei (evitarea **swap**-ării excesive care poate degrada performanța), gestionarea CPU (prioritatea proceselor), configurarea stocării (sisteme de fișiere optimizate, parametri kernel precum cache-ul de disc etc.) și configurarea rețelei (porturi, firewall, latență).

Distribuții Linux populare pentru servere de baze de date:

În funcție de mediul folosit, un DBA poate întâlni diferite distribuții Linux – de exemplu **Red Hat Enterprise Linux (RHEL)** sau CentOS/AlmaLinux (des întâlnite în mediul enterprise, inclusiv pentru Oracle DB), **Ubuntu Server/Debian** (popular pentru MySQL/MariaDB, PostgreSQL etc., mai ales în cloud), sau distribuții specializate (SUSE Linux Enterprise, Oracle Linux ș.a.).

Deși există diferențe la nivel de manager de pachete (rpm vs apt) sau structura unor fișiere de configurare, fundamentele administrării și comenzile de bază rămân aceleași. Administratorul trebuie să cunoască sistemul de fișiere standard al Linux (ex. directoare precum /etc – pentru configurări, /var – pentru date variabile și log-uri, /var/lib – unde deseori se stochează datele BD, /usr – pentru aplicații instalate, /home etc.) precum și modul în care serviciile sunt gestionate (vezi mai jos **systemd**).

Familiarizarea cu **shell-ul** (interpretorul de comenzi, de obicei bash) este esențială, întrucât majoritatea operațiunilor de administrare se fac via terminal, adesea de la distanță prin SSH.

Comenzi esențiale de linie de comandă

Pentru administrarea bazelor de date pe Linux, un set de comenzi de bază sunt absolut necesare. Aceste comenzi acoperă gestionarea fișierelor și directoarelor, vizualizarea și controlul proceselor, administrarea pachetelor și serviciilor, precum și diagnosticarea stării sistemului. Tabelul de mai jos prezintă câteva categorii importante de comenzi și utilitatea lor:

1) **Navigare și gestionare fișiere/directoare:**

ls (listarea conținutului unui director),
cd (schimbarea directorului curent),
pwd (afișarea directorului curent),
mkdir (crearea de directoare noi),
cp (copiere fișiere),
mv (mutare sau redenumire fișiere),
rm (ștergere fișiere sau directoare).

Exemplu: **mkdir -p /backup/db** crează un arbore de directoare pentru backup BD,
rm -rf /backup/db/test șterge recursiv directorul de test (atenție la utilizarea cu drepturi de root!).

2) **Vizualizarea conținutului fișierelor text:**

cat (afișează conținutul unui fișier direct în terminal),
more/less (permite paginarea conținutului pentru fișiere lungi),
tail (afișează ultimele N linii dintr-un fișier – foarte util pentru a vedea ultimele mesaje dintr-un log)
și opțiunea -f la tail (**tail follow**, pentru a vedea în timp real fluxul de mesaje ce se adaugă la un log).
De pildă, tail -f /var/log/mysql/error.log va afișa continuu noile erori apărute în log-ul serverului MySQL.

3) **Căutarea și procesarea textului:**

grep este comandă vitală pentru a căuta un șir de caractere sau un model regex în unul sau mai multe fișiere.
Spre deosebire de **find** (care caută fișiere în sistem după nume sau alte atribute), grep caută un text în interiorul fișierelor[4].

Exemple:
grep -i "error" /var/log/mysql/error.log va lista toate liniile (ignorând capitalizarea) care conțin cuvântul "error" în fișierul de log, ajutând la identificarea rapidă a problemelor;
grep -R "SELECT" /var/lib/mysql ar căuta recursiv în toate fișierele din directorul specificat (poate cod SQL salvat sau log-uri de interogări) șirul "SELECT".
De asemenea, comanda **find** poate fi folosită pentru sarcini de mentenanță, de exemplu găsirea și ștergerea backup-urilor mai vechi de o anumită perioadă:
find /backup/logs -type f -mtime +35 -print > backupfiles.log & (găsește fișiere mai vechi de 35 zile din /backup/logs și le listează într-un fișier de ieșire)[5],
sau **find /backup/logs -type f -mtime +7 -exec rm -f {} \;** pentru a șterge direct fișierele mai vechi de 7 zile.
Astfel de comenzi sunt utile pentru a evita umplerea discului cu backup-uri sau log-uri vechi.

4) **Gestionarea permisiunilor și proprietarilor:**

Linux este un sistem multiutilizator, iar fișierele și directoarele au setări de **permisiuni** (citire, scriere, execuție) pentru trei categorii: **proprietarul fișierului**, **grupul asociat** și **ceilalți utilizatori**[6][7].
Comenzile **chmod (change mode)** și **chown (change owner)** sunt folosite pentru a modifica aceste permisiuni.
De exemplu, **chmod 750 script.sh** acordă proprietarului drepturi de citire-scriere-execuție (7 = 4+2+1) și grupului doar citire-execuție (5 = 4+1), fără niciun acces pentru alți utilizatori (0)[8][9].

În contextul SGBD, este important ca fișierele de date și directoarele de lucru ale bazei de date să aparțină utilizatorului de sistem sub care rulează serviciul BD (ex.: postgres pentru PostgreSQL, mysql pentru MySQL/MariaDB) și să aibă permisiuni restrictive pentru a preveni accesul neautorizat.
O practică bună este evitarea rulării serverelor de baze de date ca **root**, ci utilizarea conturilor dedicate cu privilegii limitate, configurând permisiunile fișierelor corespunzător (de exemplu, directoarele cu date ale BD – adesea în /var/lib/mysql sau /var/lib/postgresql – să fie accesibile doar de către utilizatorul SGBD și de root). În cazul în care permisiunile nu sunt setate corect, serviciul de BD poate refuza să pornească sau poate întâmpina erori de acces la fișiere.

5) **Administrarea utilizatorilor și proceselor:**

useradd, userdel, usermod – administrarea conturilor de utilizator;
passwd – pentru gestionarea parolelor (ex. expirarea unei parole, forțarea schimbării la următoarea autentificare etc.).

De multe ori, instalarea unui SGBD creează automat contul de utilizator dedicat (ex.: pachetul PostgreSQL creează utilizatorul postgres).
Ca administrator, trebuie să știi să ajustezi drepturile sau să adaugi utilizatorul BD în anumite grupuri dacă este nevoie (ex.: grupul dba ș.a.).
Pentru gestionarea **proceselor**,
comanda **ps -aux** listează toate procesele rulând,
pgrep <nume> caută procese după nume (de ex. pgrep -af postgres),
iar **kill** sau **killall** trimite semnale pentru oprirea proceselor (ex.: kill -TERM <PID> pentru o oprire grațioasă a procesului cu PID dat, sau kill -9 pentru terminare forțată – de evitat pe procese de SGBD, deoarece echivalează cu o oprire necontrolată).

Comanda **top/htop** (detaliată în secțiunea următoare) poate fi folosită și pentru a omorî un proces selectat (în htop apăsând tasta k se poate termina procesul curent selectat[10]).

6) **Instalarea și gestionarea pachetelor software:** În funcție de distribuție, se folosesc managerul de pachete apt (Debian/Ubuntu) sau yum/dnf (RHEL/CentOS/Fedora) etc. Pentru un DBA, aceasta este important la instalarea SGBD-ului și a utilitarelor auxiliare. Ex.: sudo apt install postgresql pentru a instala PostgreSQL pe Ubuntu, respectiv sudo yum install mariadb-server pe un CentOS. De asemenea, instrumente precum htop, netstat (net-tools), sysstat (care conține iostat) pot necesita instalare: de exemplu, sudo apt install htop net-tools sysstat[11][12].

7) **Servicii și sistemul de init (systemd):**

În distribuțiile moderne, **systemd** este sistemul de init care pornește automat serviciile la boot și permite control uniform asupra acestora prin comanda unificată systemctl. Administrarea unui server BD implică pornirea, oprirea sau repornirea serviciului de baze de date și configurarea lui să pornească automat.

Comenzi uzuale: systemctl start <serviciu>, systemctl stop <serviciu>, systemctl restart <serviciu>, systemctl status <serviciu> – unde <serviciu> este numele unității service (ex.: mysql.service, postgresql.service etc.).

Spre exemplu, pentru a porni serverul MySQL pe o distribuție Ubuntu/Debian, comanda este sudo systemctl start mysql (pe RHEL/Fedora numele serviciului este mysqld)[13][14]. Comanda nu afișează nimic dacă reușește, de aceea se folosește systemctl status mysql pentru a verifica că serviciul este *active/running*[15].

Pentru pornirea automată la boot: systemctl enable mysql. În distribuțiile vechi sau situații fără systemd, se poate folosi și comanda tradițională service mysql start (sau scriptul din /etc/init.d/), dar acestea sunt în mare parte *alias*-uri către systemd în sistemele recente[16][17].

Notă: *Este crucial ca DBA-ul să știe să repornească rapid serviciul BD în caz de probleme*, să știe *unde se află fișierele de configurare* (de regulă în /etc/, ex: /etc/mysql/my.cnf, /etc/postgresql/<versiune>/main/postgresql.conf) și să inspecteze log-urile dacă serviciul nu pornește (ex.: comanda journalctl -u mysql.service arată ultimele mesaje ale serviciului MySQL în systemd journal, sau log-urile specifice din /var/log).

Considerații de securitate și bune practici:

În administrarea Linux pentru baze de date, trebuie aplicate principiile minimei permisiuni și securizării la nivel de OS. Astfel, se va asigura că doar porturile necesare sunt deschise (ex. portul 5432 pentru PostgreSQL, 3306 pentru MySQL), restul fiind filtrate de firewall (UFW/iptables). Accesul la server se face securizat (SSH cu chei publice, autentificare cu factor multiplu dacă e posibil). Actualizările de securitate ale OS-ului și SGBD-ului se aplică regulat (apt update && apt upgrade, etc.). De asemenea, monitorizarea resurselor și a autentificărilor (log-urile /var/log/auth.log sau /var/log/secure) face parte din responsabilitatea administratorului.

Concluzie:

Stăpânirea SO Linux reprezintă un atu esențial pentru orice specialist în baze de date. Cunoașterea comenzilor de administrare și a modului în care SO interacționează cu SGBD-ul permite diagnosticarea rapidă a problemelor (fie că e vorba de un fișier de date cu permisiuni greșite, de lipsa spațiului pe disc, de o limită atinsă la numărul de descriptori de fișier, sau pur și simplu de un proces blocat ce trebuie oprit). Un administrator de baze de date eficient va combina cunoștințele de SGBD cu cele de Linux pentru a asigura o funcționare optimă și neîntreruptă a sistemelor de baze de date.

9.2 Instrumente Linux pentru monitorizarea și gestionarea bazelor de date

Administrarea bazelor de date (mai ales a celor **distribuite**, cu multiple noduri) necesită o atenție deosebită asupra **performanței și disponibilității** sistemului.

Linux pune la dispoziție numeroase instrumente de monitorizare *out-of-the-box*, care permit examinarea în timp real a utilizării resurselor (CPU, memorie, disc, rețea) fără a necesita interfețe grafice sau instalări complexe[18][19].

Monitorizarea proactivă ajută administratorul să identifice anomalii sau potențiale probleme înainte ca acestea să devină critice, permițând optimizări și intervenții la timp[18].

În continuare vom prezenta cele mai utile utilitare de sistem pentru monitorizarea și gestionarea bazelor de date pe Linux, organizate pe categorii de resurse:

9.2.1. Procese, CPU și memorie – top/htop:

Comanda tradițională **top** este adesea primul punct de apel pentru a obține o imagine de ansamblu a stării sistemului. Ea *afișează într-un mod actualizat continuu lista proceselor active, împreună cu consumul de **CPU** și **memorie RAM** al fiecărui proces, timpul de rulare, numele utilizatorului proprietar etc*[20].

În partea de sus a ecranului top se regăsesc statistici globale precum sarcina medie (load average), numărul total de procese, memoria liberă și swap-ul folosit.

Pentru administrarea unei baze de date, aceste informații sunt valoroase: putem vedea dacă procesul SGBD (ex: mysqld, postgres, mongod etc.) consumă excesiv CPU sau memorie, dacă sistemul începe să facă swap (ceea ce ar indica memorie insuficientă) sau dacă există multe procese concurente care ar putea impacta performanța.

Îmbunătățind top, utilitarul **htop** oferă o interfață colorată și interactivă, cu funcții avansate și controale intuitive (navigare cu taste săgeți, filtre). htop *arată de exemplu grafic încărcarea pe fiecare nucleu de CPU și utilizarea memoriei, suportă scroll orizontal/vertical prin lista de procese și sortare rapidă după diferiți parametri*[21]. Deși nu este instalat implicit peste tot, htop poate fi adăugat ușor (ex. sudo apt install htop pe Ubuntu) și poate înlocui top ca instrument de bază[22].

În htop, administratorul poate sorta procesele după memoria consumată (tasta **M**), după utilizarea CPU (**P**), poate căuta un proces după nume (tasta /) sau chiar trimite semnale (kill) direct selectând un proces și apăsând **F9** sau **k**[23].

Un alt avantaj este că htop afișează numărul de **fire (threads)** al proceselor (importante la SGBD-uri multi-thread cum e MySQL) și permite **filtrarea** proceselor după utilizator – de pildă puteți vedea rapid doar procesele utilizatorului postgres.

Cu ajutorul lui top/htop, un DBA *poate identifica situații precum: procesul bazei de date consumă constant 100% CPU (posibil query neoptimizat), memorie aproape epuizată (necesitatea de a aloca mai mult RAM sau de a ajusta parametri SGBD), sau prezența unor procese "zombie"/blocate.*

9.2.2. Rețea – conexiuni și trafic – netstat/ss și tcpdump:

În contextul bazelor de date, monitorizarea conexiunilor de rețea este esențială, mai ales pentru sisteme distribuite sau baze de date accesate de la distanță.

Utilitarul clasic **netstat** (Network Statistics) oferă *informații versatile despre conexiunile TCP/UDP active, porturile pe care serverul ascultă, statisticile interfețelor de rețea și tabelele de rutare*[24].

Cu netstat -a putem lista toate conexiunile și porturile deschise, cu netstat -plnt putem vedea toate porturile TCP de ascultare împreună cu PID-ul și numele procesului care ascultă (de folos pentru a verifica dacă serviciul de BD a pornit și ascultă pe portul așteptat, de exemplu 5432)[25].

De asemenea, netstat -p -at listează conexiunile TCP cu indicarea proceselor asociate[24], iar netstat -s sau -st afișează statistici pe protocoale (număr de pachete, erori etc.)[26].

Exemplu practic: Rulând netstat -an | grep 3306 pe un server MySQL putem vedea adresele IP conectate și starea conexiunilor (ESTABLISHED, TIME_WAIT etc.), ceea ce ajută la depistarea unui posibil client care inundă serverul cu conexiuni.

Merită menționat că pe sistemele moderne pachetul net-tools (care conține netstat) nu este instalat implicit – dacă netstat nu există, se poate instala (sudo apt install net-tools) sau se poate folosi comanda echivalentă **ss**, care face parte din pachetul iproute2 și oferă funcționalitate similară, într-un mod mai rapid și eficient. De exemplu, ss -tulpn arată porturile TCP/UDP ascultate și procesele corespunzătoare (similar cu netstat -plnt).

Pentru o analiză mai detaliată, *sniffer*-ul **tcpdump** permite capturarea pachetelor de rețea în timp real. Cu tcpdump -i eth0 port 5432 am putea monitoriza traficul către un server PostgreSQL, filtrând pachetele după port. tcpdump poate fi util în scenarii de debugging de rețea sau de securitate (ex. detectarea unui volum neobișnuit de handshake-uri TCP SYN care ar putea indica un atac DoS[27][28]).

*Totuși, în utilizarea de zi cu zi a DBA-ului, netstat/ss sunt suficiente pentru a **verifica conectivitatea** și a vedea dacă există clienți conectați sau conexiuni în așteptare.*

9.2.3 Discuri și I/O – iostat/iotop:

Performanța subsistemului de stocare este adesea factorul critic în aplicațiile de baze de date, mai ales în medii cu volume mari de date sau tranzacții intensive. **iostat** (parte din pachetul **sysstat**) este un instrument simplu, dar puternic, pentru a obține rapid o vedere de ansamblu asupra utilizării **I/O** (input/output) la nivel de sistem și pe fiecare dispozitiv de stocare[29]. Rulând iostat, se afișează întâi statistici medii de CPU (inclusiv procentul de timp CPU în mod idle și mai ales %iowait – procentul de timp CPU în care acesta stă inactiv dar are operații de I/O în așteptare, un indicator important al **gâtuirii pe I/O**), urmate de statistici pe fiecare disc: transferuri pe secundă (tps), rata de citire și scriere (Blocuri sau KB pe secundă), volumul total citit și scris de la pornirea sistemului etc[29]. Un %iowait ridicat combinat cu un throughput de disc saturat sugerează că baza de date este limitată de performanța discului (posibil soluții: RAID mai performant, SSD/NVMe, optimizare de interogări pentru a citi/scrie mai puțin, mai mult RAM pentru caching etc.). iostat este util și pentru a observa distribuția încărcării între mai multe dispozitive (de exemplu, dacă avem separat un disc pentru jurnale de tranzacție și unul pentru fișierele de date, se poate vedea dacă unul devine bottleneck).

Pentru monitorizare continuă, iostat -x 5 (extended stats la fiecare 5 secunde) oferă o actualizare periodică.

Pe lângă iostat, utilitarul **iotop** oferă o interfață interactivă stil *top* care arată exact ce procese generează trafic de I/O la un moment dat[30]. După instalare (sudo apt install iotop), iotop listează procesele și coloane cu KB/s citite și scrise de fiecare, procent swap-in și IO (un fel de procent din timpul de I/O total). Este foarte util dacă discul „lucrează” intens și dorim să știm *cine* anume provoacă asta – de exemplu, dacă un backup rulează,

vom vedea procesul de backup scriind masiv; sau dacă SGBD-ul consumă I/O intens, putem corela cu o anumită operație SQL în curs.

Notă: Pentru a rula iotop e nevoie de permisiuni root (sau să fie în grupul sudo), deoarece accesează /proc//*io. Atât iostat, cât și iotop ajută la identificarea situațiilor de tip I/O-bound** (spre deosebire de top care ne arată CPU-bound).

În plus, comanda df -h (disk free) trebuie folosită regulat pentru a verifica spațiul pe partiții – nimic nu oprește mai abrupt o bază de date decât lipsa spațiului liber pe volumul de date sau de log-uri. DBA-ul trebuie să monitorizeze spațiul și să aibă alerte sau proceduri de curățare (ex. ștergerea logurilor vechi, arhivarea backup-urilor) înainte ca discul să se umple complet.

9.2.4. Memorie și swap – free, vmstat:

Deși htop/top afișează memoria utilizată, uneori este util un instantaneu rapid cu comanda free -m (afișează memorie totală, folosită, liberă, buffere/cache, swap în megabytes).

Trebuie înțeles că Linux folosește memoria liberă pentru **cache de disc** (buffere/cache), ceea ce este benefic pentru performanța BD. Astfel, memorie „liberă" foarte mică nu e neapărat rea, atâta timp cât majoritatea este la cache (care se eliberează la nevoie).

Interesant de urmărit este și **swap-ul**: ideal, pe un server de baze de date swap-ul folosit ar trebui să fie 0 sau foarte redus, deoarece folosirea intensivă a swap indică insuficient RAM și duce la încetiniri drastice (swap-ul implică I/O pe disc mult mai lent decât RAM).

Comanda **vmstat 1** (virtual memory stat) oferă o actualizare pe secundă a unor parametri cheie: procese în așteptare, memorie liberă, swap in/out, IO wait, într-un mod concis. Dacă se observă activitate constantă de swap-in/swap-out, este un semn de alarmă. Soluția poate fi optimizarea configurației SGBD pentru a consuma mai puțin RAM sau adăugarea de memorie fizică.

Un articol de bune practici Linux menționa că este recomandat să existe totuși un spațiu de swap minim configurat, chiar dacă suficient RAM, deoarece prezența unui swap configurat (dar nefolosit intens) nu afectează performanța, dar absența lui poate cauza probleme la epuizarea completă a RAM[31][32].

Totuși, folosirea swap-ului ar trebui monitorizată și limitată.

9.2.5. Procese de bază de date și fișiere deschise – lsof:

Un alt instrument util este **lsof** (list open files), care poate arăta ce fișiere a deschis un anumit proces.

Pentru un SGBD, putem folosi lsof -p <PID> (unde PID este al procesului principal al bazei de date) pentru a vedea toate fișierele pe care le are deschise: fișiere de date .db/.ibd, fișiere de jurnal, socket-ul de comunicație, librării partajate etc.

Aceasta poate fi util, de exemplu, pentru a verifica dacă un anumit fișier de date este într-adevăr folosit sau dacă, în cazul în care ștergem un fișier din greșeală, procesul încă îl ține deschis (situație în care datele rămân accesibile până la închiderea procesului). lsof -i :<port> este de asemenea util pentru a vedea ce proces ascultă pe un port (ex: lsof -i :27017 pentru MongoDB).

9.2.6. Monitorizarea jurnalelor (log monitoring):

În administrarea oricărui server, inclusiv a bazelor de date, analiza fișierelor de jurnal este esențială. Linux oferă comenzi precum tail -f (menționată mai sus) pentru a urmări logurile în timp real, dar și posibilități de a **filtra** logurile după criterii.

O tehnică comună este combinarea grep cu tail sau cat pentru a extrage doar liniile relevante.

De exemplu: *grep "ERROR" /var/log/postgresql/postgresql-14-main.log | tail -100* pentru a vedea ultimele 100 de erori în logul Postgres.

Pentru monitorizare continuă, există unelte dedicate precum watch (care poate rula periodic o comandă, ex: watch -n 5 "netstat -an | grep 5432 | wc -l" ar afișa la fiecare 5 secunde numărul de conexiuni pe portul 5432), sau soluții mai avansate (integrate cu syslog, ELK stack etc.).

Studiu de caz – monitorizare într-un sistem distribuit:

Să ne imaginăm un cluster de baze de date replicat (ex.: un master și două replici MySQL). Administratorul va folosi unelte de mai sus pe fiecare nod Linux pentru a urmări starea: cu htop vede încărcarea pe master versus replici (poate masterul e mult mai solicitat CPU), cu iostat verifică dacă unul din noduri are discul suprasolicitat (poate replica întârzie din cauza I/O lent), cu netstat verifică numărul de conexiuni de replicare și de clienți activi pe fiecare nod, iar din loguri (tail -f pe /var/log/mysql/mysql.log) monitorizează eventualele erori de replicare. Dacă ar exista un trafic anormal de mare între noduri, ar putea folosi tcpdump pentru a captura pachetele și a analiza ce fel de tranzacții se propagă. Toate aceste date împreună dau o imagine holistică a stării clusterului. Un exemplu real al importanței monitorizării la scară este compania Apple, care a raportat operarea a peste **75.000 de noduri Cassandra** (un sistem de baze de date NoSQL distribuit) ce stochează peste 10 petabytes de date, cu clustere individuale de peste 1000 de noduri și milioane de operații pe secundă[33]. La acest nivel, uneltele de monitorizare automată și tablourile de bord devin obligatorii, însă chiar și atunci, cunoștințele de bază despre top, iostat, netstat etc. rămân fundamentale pentru a înțelege și verifica rapid starea unui anumit nod din cluster.

Rezumat secțiunea 9.2: Linux furnizează gratuit și implicit numeroase instrumente cu ajutorul cărora un administrator poate **supraveghea performanța** bazelor de date și a serverelor pe care acestea rulează. Fie că este vorba de observarea proceselor și consumul de resurse cu htop, de inspectarea conexiunilor de rețea cu netstat, sau de identificarea blocajelor de I/O cu iostat, aceste utilitare constituie „trusa de unelte” zilnică a oricărui SysAdmin/DBA. Familiaritatea cu ele și interpretarea corectă a datelor oferite reprezintă cheia pentru diagnostic rapid și tuning eficient. În plus, multe dintre aceste comenzi pot fi **scriptate și automatizate**, subiect pe care îl vom aborda în secțiunea următoare.

9.3 Automatizarea sarcinilor de administrare folosind Bash și Python

O caracteristică a administrării moderne a sistemelor (inclusiv a bazelor de date) este tendința de a **automatiza** sarcinile repetitive sau complexele, pentru a reduce erorile umane și a crește eficiența. În loc să efectueze manual operații de mentenanță în fiecare zi, un administrator priceput va scrie scripturi care să facă aceste operații automat, eventual la ore prestabilite sau la declanșarea anumitor evenimente. Linux, prin filozofia sa de a oferi instrumente scriptabile și

interfață text uniformă, este un mediu ideal pentru automatizare. Două abordări principale sunt frecvent folosite: scriptingul shell (de obicei cu **Bash**) și scriptingul în limbaje de nivel înalt precum **Python**. Ambele au avantajele lor și, adesea, se completează reciproc.

Automatizare cu Bash (Shell Scripting)

Shell-ul Bash (Bourne Again Shell) este interpretorul de comenzi implicit pe majoritatea distribuțiilor Linux și oferă un limbaj de scripting incorporat.

Un script Bash este practic o serie de comenzi Linux puse într-un fișier text executabil, posibil cu structuri de control (if, for, while), variabile și funcții.

Avantajul Bash este că are acces direct la toate comenzile sistemului, fără nevoie de biblioteci externe – practic orice comandă care poate fi executată manual poate fi pusă într-un script.

Pentru un administrator de BD, scripturile shell sunt utile pentru sarcini precum:

1) **backup-uri automate**
2) **rotirea log-urilor**
3) **monitorizare simplă a serviciilor**
4) **migrarea datelor**
5) **instalări și configurări repetabile**, etc.

Exemplu: script de backup automat MySQL.

Să considerăm necesitatea realizării unui backup zilnic al unei baze de date MySQL. Manual, ar trebui în fiecare zi să rulăm comanda mysqldump și să stocăm rezultatul. Un script Bash poate realiza acest lucru automat.

Urmatorii pași ar acoperi un asemenea script:

(1) definirea parametrilor de conectare (utilizator, parolă, nume BD),
(2) definirea locației de backup,
(3) generarea unui nume de fișier bazat pe data curentă (timestamp),
(4) execuția comenzii mysqldump și redirecționarea ieșirii în fișier,
(5) comprimarea fișierului rezultat (ex. cu gzip),
(6) ștergerea backup-urilor mai vechi de X zile pentru a elibera spațiu.

Un astfel de script a fost prezentat într-un articol recent, subliniind importanța **automatizării backup-urilor** pentru siguranța datelor[34]. Un fragment relevant din acel script arată crearea directorului de backup dacă nu există și execuția propriu-zisă a dump-ului urmată de compresie:

```
# Creare director backup dacă nu există
mkdir -p $BACKUP_DIR

# Efectuare backup și compresie
mysqldump -u$DB_USER -p$DB_PASSWORD $DB_NAME >
$BACKUP_DIR/$DB_NAME-$TIMESTAMP.sql
gzip $BACKUP_DIR/$DB_NAME-$TIMESTAMP.sql
echo "Backup complet: $BACKUP_DIR/$DB_NAME-$TIMESTAMP.sql.gz"
```

[35][36]

De asemenea, scriptul include opțional o comandă find pentru a șterge fișierele mai vechi de o anumită perioadă (exemplu: find $BACKUP_DIR -type f -name "*.gz" -mtime +7 -exec rm {} \; pentru ștergerea backup-urilor mai vechi de 7 zile)[37].

Odată scris și testat, scriptul poate fi programat cu cron – de exemplu, adăugând în crontab o linie precum 0 0 * * * /path/catre/script_backup.sh care va rula scriptul în fiecare zi la miezul nopții[38]. Astfel, backup-ul devine complet automatizat, iar administratorul primește notificări sau poate verifica log-urile pentru a se asigura că totul a decurs corect.

Alte aplicații Bash în administrarea BD:

- *Monitorizare simplă a serviciului:*

Un script ar putea verifica dacă procesul bazei de date este în execuție (folosind pgrep sau systemctl status) și, dacă detectează că a picat, să încerce repornirea serviciului și eventual să trimită o alertă (prin mail sau alt canal).

- *Automatizare în medii distribuite:*

Folosind Bash împreună cu comanda ssh, se pot executa comenzi pe noduri remote, permițând, de pildă, unui script central să colecteze metrici de pe mai multe servere. Exemple: un script care pe toate nodurile de replicare colectează rezultatul la SELECT lag FROM pg_stat_replication (în Postgres) sau la SHOW SLAVE STATUS (în MySQL) și raportează dacă vreo replică are întârziere mare.

- *Gestionarea utilizatorilor sau permisiunilor în lot:*

Dacă trebuie creați mai mulți utilizatori SQL sau aplicate scripturi SQL de mentenanță, un script Bash poate citi dintr-un fișier o listă de comenzi SQL și să le aplice în serie (folosind clienții de linie de comandă precum mysql -e "QUERY" sau psql -c "QUERY").

- *Automatizarea migrărilor*:

Ex. exportul datelor dintr-o bază și importul în alta, cu transformări intermediare folosind utilitare de text (sed/awk) – se pretează bine la Bash pipeline.

P entru a scrie scripturi Bash complexe, este nevoie de cunoașterea elementelor de sintaxă (structuri de decizie if/then, bucle for/while, substituția comenzii, variabile de mediu etc.).

Există resurse abundente, precum **Advanced Bash-Scripting Guide**[39], care oferă atât elemente de bază cât și trucuri avansate de scripting. În practică, scripturile trebuie testate riguros, mai ales când efectuează acțiuni critice (backup, ștergeri).

O bună practică este includerea de logare (comanda logger sau redirecționarea ieșirilor către fișiere de log) și notificări în caz de eșec (ex: dacă un backup eșuează, scriptul poate trimite un email folosind comanda mail sau un webhook).

Automatizare cu Python

Python a devenit o alegere populară pentru automatizarea administrării de sistem, complementând sau chiar înlocuind uneori scripturile shell, datorită sintaxei clare și a bibliotecilor bogate disponibile.

Un articol recent (2025) notează că Python este unul din cele mai populare limbaje printre administratorii de sistem datorită simplității și flexibilității sale, permițând automatizarea de la sarcini repetitive până la monitorizare și configurare complexă[40].

Python vine cu avantaje precum: gestionarea facilă a erorilor, portabilitate (rulează ușor pe Windows/Linux), posibilitatea de a folosi protocolul SSH, API-uri web etc., lucruri ce pot fi mai anevoioase în Bash.

Pentru administrarea bazelor de date, Python poate fi utilizat în câteva moduri principale:

1) **Scripturi de administrare a sistemului cu Python:**

Folosind bibliotecile standard sau externe, Python poate realiza tot ce face un script Bash, dar cu o structură de cod adesea mai robustă.

Biblioteci comune includ:

- *modulul os* (inclus în standard) – pentru operații de sistem de fișiere, variabile de mediu, rularea de comenzi de bază[41].
- *modulul subprocess* – pentru a executa comenzi shell direct din Python și a captura rezultatele[41] (ideal când nu există o bibliotecă Python dedicată pentru o anumită acțiune).
- *modulul shutil* – pentru operații cu fișiere/directoare (copiere, mutare, ștergere)[42][43].
- biblioteca psutil (third-party) – oferă acces la informații de sistem ca utilizarea CPU, memorie, disc, procese, similar cu comenzi ca top/vmstat, dar direct în Python[44][45].
- *biblioteca socket* (standard) – pentru operații la nivel de rețea (ex: se poate verifica dacă un port e deschis încercând o conexiune)[44].
- *biblioteca paramiko* (third-party) – implementare de SSH în Python, utilă pentru a automatiza comenzi pe servere remote via SSH (în loc să folosim manual ssh în Bash)[46].

Aceste biblioteci permit crearea unor scripturi Python care să monitorizeze **resursele sistem** (ex: un script care la fiecare 5 secunde loghează CPU, memorie și spațiu liber – practic un mini-monitor, folosind psutil pentru CPU/memorie/disk și eventual psutil.net_io_counters pentru rețea), să **gestioneze fișiere** (mutarea automată a unor dump-uri de pe SSD pe un HDD extern etc.), să **gestioneze utilizatori și permisiuni** (Python poate apela comenzi useradd sau poate edita direct fișierele de configurare), sau să **configureze interfețe de rețea** (prin socket sau pachete precum netmiko pentru echipamente).

2) **Automatizarea operării pe bazele de date cu Python:**

Python se evidențiază în special prin abilitatea de a folosi **biblioteci de conectare la SGBD** (ex: psycopg2 pentru PostgreSQL, mysql-connector-python sau PyMySQL pentru MySQL/MariaDB, cx_Oracle pentru Oracle, pymongo pentru MongoDB etc.).

Astfel, Python poate fi folosit nu doar pentru administrarea sistemului de operare ci și pentru a **automatiza acțiuni direct în baza de date**.

De exemplu:

- Un script Python poate conecta la baza de date și rula automat interogări de sănătate (health checks): ex. verificarea versiunii, verificarea existenței unor indexuri, executarea unor proceduri de mentenanță.
- Se pot genera rapoarte periodice: un script zilnic care se conectează la BD, extrage metrici (număr de utilizatori noi, tranzacții, mărimea bazei) și le salvează într-un fișier CSV sau trimite pe email echipei.
- Automatizarea managementului utilizatorilor de BD: de ex., la onboarding/offboarding al unui dezvoltator, un script Python ar putea adăuga/elimina acel user atât la nivel de sistem Linux (creare cont Unix dacă are nevoie de acces SSH) cât și la nivel de SGBD (creare cont SQL și acordare privilegii), asigurând consistență.
- Orchestrarea backup-urilor și a operațiilor de restaurare: Python poate apela utilitare de backup (mysqldump, pg_dump) similar cum am făcut în Bash, dar poate și verifica integritatea backup-ului (de ex. comparând mărimi, sau încercând o restaurare într-o instanță de test), ceea ce devine un flux de backup mai inteligent.

Exemplu practic: monitorizare personalizată cu Python. Să zicem că dorim un script care să ne alerteze dacă serverul de baze de date devine supraîncărcat. În loc să instalăm un software complex de monitorizare, putem scrie în Python câțiva pași:

- Folosim psutil pentru a citi periodic CPU utilizat (%) și memoria liberă (%)[47][48].
- Folosim psutil sau subprocess să obținem și load average (de exemplu os.getloadavg() dă valorile).
- Ne conectăm la SGBD (ex. cu psycopg2 la Postgres) și executăm o interogare precum SELECT count(*) FROM pg_stat_activity (numărul de conexiuni curente) sau citim din vreo vedere statistică internă (de exemplu, în Postgres pg_stat_database are informații despre tranzacții, în MySQL SHOW GLOBAL STATUS).
- Pe baza acestor date, dacă oricare depășește un prag (CPU > 90% pentru 5 minute, memorie liberă < 10%, >100 conexiuni active etc.), scriptul poate trimite o alertă – fie pe email (se poate folosi modulul smtplib pentru a trimite email), fie prin integrare cu un API (Slack, Teams – există biblioteci), sau chiar să declanșeze automat acțiuni (ex: să pornească o replică de rezervă sau să blocheze un IP care face prea multe conexiuni).
- Scriptul poate rula ca demon (folosind un job systemd) sau poate fi chemat din cron la fiecare X minute.

Acest tip de **automatizare personalizată** e util când nu avem soluții enterprise de monitorizare la dispoziție sau vrem ceva rapid adaptat nevoilor noastre. Python excelează aici, având atâtea librării la dispoziție.

3) **Automatizarea taskurilor complexe și DevOps:** În medii foarte complexe, se folosesc adesea instrumente precum *Ansible*, *Chef*, *Puppet* pentru a automatiza configurarea serverelor (infrastructură ca cod). Merită menționat că **Ansible** folosește la bază Python (modulele sale interne sunt scrise în Python) și permite rularea unor module de baze de date (ex: modul Ansible mysql_db pentru a crea BD sau postgres_user pentru a crea utilizatori Postgres, etc.). Așadar, cunoașterea Python ajută și la înțelegerea și eventual extinderea acestor instrumente.

Exemplu: generarea de scripturi de configurare. Un scenariu comun în DevOps este că, având un fișier YAML/JSON de configurare, se doresc aplicați anumite setări. De exemplu, stocăm lista de baze de date ce trebuie create la inițializarea unui server nou, cu utilizatorii și parolele lor. Un script Python poate parsa acest fișier și executa secvențial comenzi SQL de creare a bazelor de date și utilizatorilor, fie direct printr-o bibliotecă de conectare la SGBD, fie generând un script .sql final. Un articol exemplificativ arată cum Python poate genera automat instrucțiuni CREATE TABLE pornind de la date într-un fișier Excel, scutind astfel munca manuală a DBA-ului la traducerea unui model în DDL[49]. Acest tip de automatizare reduce erorile de dactilografiere și asigură consistență.

Comparativ Bash vs Python: Ambele abordări au meritele lor. Bash este excelent pentru **simple glue tasks** – lipirea unor comenzi existente, manipulare rapidă a fișierelor de configurare prin sed/awk, scrierea de one-liners. Python devine mai eficient pe măsură ce logica devine mai complexă (bucla, condiții multiple, procesare de date) sau când trebuie folosite *APIs* și structuri de date mai complexe (liste, dicționare etc.). De multe ori, soluția optimă este un hibrid: scripturi Python care *apelează* comenzi shell acolo unde e convenabil (prin subprocess), sau scripturi Bash care încorporează bucăți de Python pentru lucruri complicate (de exemplu folosind utilitarul python -c "…"). Important este ca un administrator să fie confortabil cu ambele – să poată scrie un shell script de 20 de linii pentru o sarcină rapidă, dar și un script Python de 200 de linii pentru un daemon de monitorizare, după necesitate.

Exercițiu mental: Gândiți-vă la o sarcină plictisitoare pe care a trebuit să o faceți manual de mai multe ori în administrarea bazelor de date (spre exemplu, verificarea zilnică a spațiului liber și trimiterea unui raport echipei). Cum ați automatiza-o? Ați folosi un script Bash simplu cu df -h și mail sau ați prefera Python cu shutil.disk_usage() și modulul smtplib? Ambele pot atinge scopul – alegerea ține de preferințe, de complexitatea cerințelor (formatul emailului, logica de decizie) și de ecosistemul deja prezent (dacă deja există infrastructură Python, e logic să continuați acolo).
Concluzie secțiunea 9.3: Automatizarea cu Bash și Python reprezintă un pas natural și necesar spre administrarea eficientă a bazelor de date, mai ales în mediile distribuite sau cu cerințe de disponibilitate ridicată. Prin scriptare, administratorii pot asigura că acțiunile de rutină (backup, verificări, curățări, raportări) se desfășoară **consistent**, la timp și fără erori umane[34]. Python aduce un plus de putere și flexibilitate, integrându-se bine cu ecosistemele moderne DevOps, iar Bash rămâne cola care unește utilitarele Linux într-un întreg. Combinând aceste unelte cu planificatoare precum **cron** (sau systemd timers) pentru execuție periodică, se poate ajunge la un mediu aproape auto-administrat, în care intervenția manuală se rezumă la aspecte neprevăzute sau decizii de ordin superior.

Bibliografie

[1] [2] [4] [5] [6] [7] [8] [9] Basic Linux Commands
https://support.dbagenesis.com/linux/basic-linux-commands
[3] How to Linux for SQL Server DBAs — Part 1 - Simple Talk
https://www.red-gate.com/simple-talk/databases/sql-server/database-administration-sql-server/how-to-linux-for-sql-server-dbas-part-1/
[10] [21] [23] [51] How To Use Top, Netstat, Du, & Other Tools to Monitor Server Resources | DigitalOcean
https://www.digitalocean.com/community/tutorials/how-to-use-top-netstat-du-other-tools-to-monitor-server-resources
[11] [12] [18] [19] [20] [22] [24] [25] [26] [27] [28] [29] [30] The Best Command Line Tools for Linux Performance Monitoring
https://bitlaunch.io/blog/the-best-command-line-tools-for-linux-performance-monitoring/
[13] [14] [15] [16] [17] How to Start MySQL Server {+ How to Stop and Restart it}
https://phoenixnap.com/kb/start-mysql-server
[31] Linux Performance: Almost Always Add Swap Space - LinuxBlog.io
https://linuxblog.io/linux-performance-almost-always-add-swap-space/
[32] How big should the swap partition be? - Server Fault
https://serverfault.com/questions/19007/how-big-should-the-swap-partition-be
[33] Apache Cassandra | Apache Cassandra Documentation
https://cassandra.apache.org/_/case-studies.html
[34] [35] [36] [37] [38] Automating MySQL Database Backups with a Bash Script: A Step-by-Step Guide | by Jude Gabriel | Medium
https://medium.com/@innovativejude.tech/automating-mysql-database-backups-with-a-bash-script-a-step-by-step-guide-a4b5cdbebf77
[39] Advanced Bash-Scripting Guide
https://tldp.org/LDP/abs/html/

[40] [41] [42] [43] [44] [45] [46] [47] [48] How to automate system administration with Python - GeeksforGeeks
https://www.geeksforgeeks.org/python/how-to-automate-system-administration-with-python/
[49] Python Database Automation: Effortlessly Generate CREATE TABLE ...
https://medium.com/@balakrishna0106/python-database-automation-effortlessly-generate-create-table-statements-from-excel-49e043988e9f
[50] Automate the Boring Stuff with Python
https://automatetheboringstuff.com/
[52] [53] Top 10 Linux Commands for SQL Server DBAs
https://www.mssqltips.com/sqlservertip/4816/top-10-linux-commands-for-sql-server-dbas/
[54] Is Your System Talking Back? A Guide to Red Hat Linux Monitoring ...
https://dev.to/axisinfo_0a61830e06c3c950/-is-your-system-talking-back-a-beginners-guide-to-red-hat-linux-monitoring-tools-1eh3