

Colectarea și analiza datelor din procesul de tipărire folosind tehnologii distribuite

Lucrare de licență

Universitatea Politehnica Timișoara



Andrei Buruntia
andreiburuntia@gmail.com

Coordonator: sl. dr. ing. Dan Cosma

20-05-2018

Rezumat

Proiectul urmărește extragerea și analiza datelor din procesul de tipărire într-un mod cât mai puțin invaziv, bazându-se pe sistemul de printing open-source CUPS și folosind stack-ul ELK pentru agregarea, indexarea și vizualizarea datelor. Informațiile relevante sunt extrase din CUPS, trimise spre un endpoint local, acest endpoint trimițându-le către o mașină în cloud spre a fi indexate. Pe baza datelor indexate se construiesc grafice.

S-a considerat prioritară creerea unei platforme cap-coadă care să nu restricționeze în vreun fel workflow-ul normal al unui utilizator.

Lucrarea arată potențialul folosirii tehnologiilor și sistemelor open-source la rezolvarea problemelor din ecosistemul corporate prin modificări și îmbunătățiri aduse acestora.

Declarație pe propria răspundere

Subsemnatul Buruntia Andrei-Cristian, legitimat cu CI seria TZ nr. 080265, CNP 1950819165656, autorul lucrării ”Sistem de colectare și analiză a datelor din procesul de tipărire”, elaborată în vederea susținerii examenului de finalizare a studiilor de licență organizat de către Facultatea de Automatică și Calculatoare din cadrul Universității Politehnica Timișoara, sesiunea iunie a anului universitar 2018, luând în considerare conținutul art. 39 din RODPI – UPT, declar pe proprie răspundere, că această lucrare este rezultatul propriei activități intelectuale, nu conține porțiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislației române și a convențiilor internaționale privind drepturile de autor.

Semnătura

Data

Cuprins

1	Introducere	9
1.1	Context și motivație	9
1.2	Problema	9
1.3	Analiza studiului actual în domeniul problemei	10
2	Fundamente teoretice	11
2.1	Arhitectura client-server	11
2.2	Sisteme distribuite	11
2.3	Considerații IPC	12
2.4	Nucleul sistemului de operare	12
2.5	Grafică raster	14
2.6	CUPS	14
3	Specificațiile proiectului	15
3.1	Placa de dezvoltare gazdă - Raspberry Pi 3	15
3.2	Workflow-ul normal pentru tipărire	16
3.3	Colectarea și stocarea datelor	16
4	Design și implementare	17
4.1	Arhitectură high-level	17
4.2	Componente software	18
4.2.1	Sistemul de operare gazdă - BuruX	18
4.2.2	CUPS	19
4.2.3	Elastic Stack	20
4.3	Datele extrase	25
4.4	Build și deployment	28
4.5	Generare de date	28
4.6	Limbajul de interogare Lucene	32
4.7	Limbajul de interogare Elasticsearch	33
4.8	Platforma Scaleway	34
5	Concluzii	36
6	Glosar de termeni	38
7	Referințe	41

Introducere

1.1 Context și motivație

Océ, compania în care lucrez, activează în domeniul de printing, fiind subsidiar Canon. În companie se pune problema analizei și colectării datelor din procesul de imprimare. La propunerea companiei, am încercat să abordez această problemă. Mai jos este prezentată mai pe larg problema, iar lucrarea urmărește propunerea unei idei de implementare a unui sistem care să o rezolve. În această documentație se va încerca detalierea asupra anumitor aspecte ale lucrării: dificultățile întâlnite, probleme de comunicare, constraint-uri de orice fel, etc.

1.2 Problema

Analiza și colectare de statistici despre procesul de imprimare

Pentru a putea optimiza costurile și procedeele de tipărire în cazul imprimantelor de format mare se dorește analiza și colectarea de statistici referitoare la:

1. Cantitatea de cerneala folosită
2. Tipul de material pe care se face tipărire
3. Informații despre tipul de culoare
4. Informații despre metoda de finisare (capsare, copertare, lacuire. etc)

Aceste informații trebuie agregate și afișate sub forma unor grafice care pot fi folosite de către utilizatori cât și de către compania Océ. Utilizatorii vor avea informații legate de costuri și timpi de execuție. Compania Océ poate folosi aceste informații în mod anonimizat pentru a culege informații legate de calitatea tipăririi și a procesului cât și pentru a analiza în mod proactiv procesul de tipărire și a colecta informații despre modul în care imprimantele se comporta în timp și a preveni anumite defecțiuni. Deoarece baza de imprimante este relativ mare și nu toate imprimantele oferă în mod nativ informații despre consumabilele și cantitatea de cerneala folosită se dorește ca aceste informații să fie colectate în mod transparent și de la imprimantele mai puțin evaluate. Pentru aceasta ar trebui ca aceste informații să fie colectate într-un mod cât mai transparent, fără a afecta funcționarea normală a imprimantei și fără a afecta firmware-ul deja instalat. Informațiile astfel colectate vor fi afișate într-o interfață web accesibilă utilizatorilor finali.

și/sau companiei Océ. I această interfața grafica se vor afișa graficele necesare și se vor putea crea panouri de control dedicate. Soluția trebuie să fie extensibilă, să scaleze pe mai multe tipuri de imprimantă. În cazul în care anumite informații nu pot fi obținute în mod direct din imprimantă atunci acestea trebuie să poată fi generate prin analiza formatelor intermediare de tip rastru pe care rasterizatoarele le trimit către imprimante. Implementarea soluției trebuie să respecte următoarele cerințe:

1. Să nu expună nicio informație proprietară Océ
2. Să fie făcută într-un limbaj de nivel înalt
3. Să permită extensii cu costuri minime pentru modelele viitoare de imprimante
4. Să implice modificări minime la nivelul infrastructurii clienților
5. Să anonimizeze informațiile confidențiale

1.3 Analiza studiului actual în domeniul problemei

Din cercetările noastre, nu există o modalitate non-invazivă de a colecta date de natură non-personală despre tipărire. Există, însă, sistemul CUPS, care dispune de tot ce este nevoie în proiect în afară de capacitățile de colectare de date. Există, de asemenea, unelte pentru agregarea și vizualizarea diferitor tipuri de date.

Fundamente teoretice

2.1 Arhitectura client-server

Modelul client-server este o structură pentru aplicațiile distribuite care împarte load-ul între furnizori de resurse sau servicii (serve) și clienți, care au nevoie de serviciile sau resursele respective. În general, comunicarea între servere și clienții se realizează printr-o rețea de calculatoare sau pe hardware separat, dar nu este neîntâlnit ca serverul și clientul să fie pe aceeași mașină. Un server rulează unul sau mai multe programe care împart resursele clienților. Un client nu își expune resursele, dar cere resurse de la server. Clienții inițiază în general conexiunea sau sesiunea și așteaptă ca cererile lor să fie abordate și să primească răspuns de la server. Orice sistem care oferă facilități de network printing folosește arhitectura client-server.

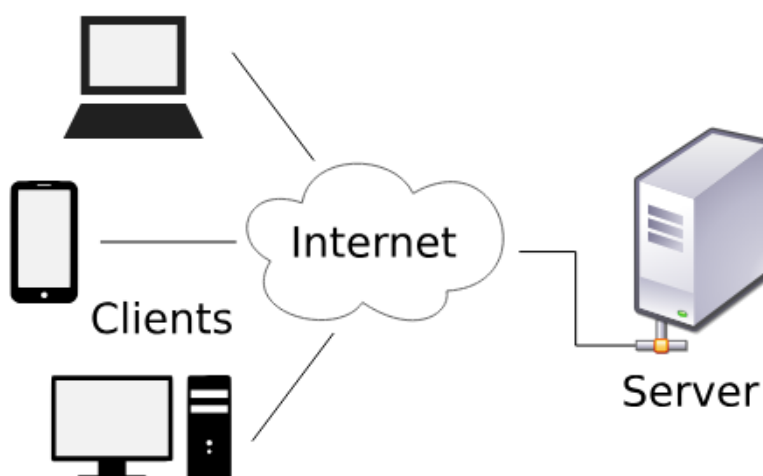


Figure 2.1: Arhitectură client-server

Caracteristica client-server descrie o relație de cooperare în aplicație. Componenta server expune o funcție sau un serviciu pentru unul sau mai mulți clienți. Clientul trebuie doar să înțeleagă răspunsul bazându-se pe un protocol prestabilit.

2.2 Sisteme distribuite

Scopul programării distribuite este împărțirea funcționalităților sau a modulelor spre a fi rulate în locații diferite. În proiectarea unui sistem distribuit trebuie tratate cazurile

în care o componenta cade. Structura sistemului nu se cunoaște în avans, la fel ca infrastructura de comunicație. Orice componentă/calculator are doar o viziune limitată a întregului sistem. Modelul client-server este o implementare specifică a acestui paradigmă.

2.3 Considerații IPC

Inter-process communication se referă la mecanismele unui sistem de operare prin care pune la dispoziție proceselor pe care le găzduiește modalități de a comunica unele cu altele și de a trimite date de la unele la altele. În mod normal, aplicațiile pot folosi IPC sub forma modelului client-server, clientul cerând și serverul furnizând date. Multe aplicații pot fi și client și server, așa cum este de multe ori cazul în distributed computing. Metodele de implementare ale IPC sunt împărțite pe categorii bazate pe cerințe software (performanță sau modularitate), circumstanțe de sistem, etc. IPC este foarte important pentru procesul de proiectare a microkernelurilor și nanokernelurilor. Microkernelurile reduc numărul de funcționalități pe care le expune kernelul. Acele funcționalități sunt obținute ulterior via IPC. Un microkernel folosește IPC mult mai intens față de un kernel monolitic convențional.

Există numeroase abordări și implementări pentru IPC:

- fișier pe disc
- semnal
- socket
- Unix domain socket
- coadă de mesaje
- pipe
- named pipe sau FIFO
- etc

Sistemele POSIX, dar și Windows, includ majoritatea acestor mecanisme.

2.4 Nucleul sistemului de operare

Kernelul este programul care stă la baza sistemului de operare și are control asupra întregului sistem. La majoritatea sistemelor de operare, nucleul este printre primele programe care se încarcă în memorie la start-up, imediat după bootloader. Este responsabil cu I/O, traducerea stream-urilor de date de la aplicații în instrucțiuni pentru procesor, arbitrează procese, alocă spațiu pentru utilizator și alte funcții de bază ale sistemului de operare. În mod normal, nucleul se încarcă într-o zonă specială și separată de memorie, protejată de accese ale aplicațiilor sau a componentelor mai puțin critice ale sistemului. Toate lucrurile critice sistemului au loc în kernel space, iar programele non-critice au loc în user space. Separând astfel user data de kernel data, se previne interferența și instabilitatea. Interfața nucleului este un nivel de abstractizare low-level. Apelurile de la procese către nucleu se numesc apeluri de sistem sau system calls.

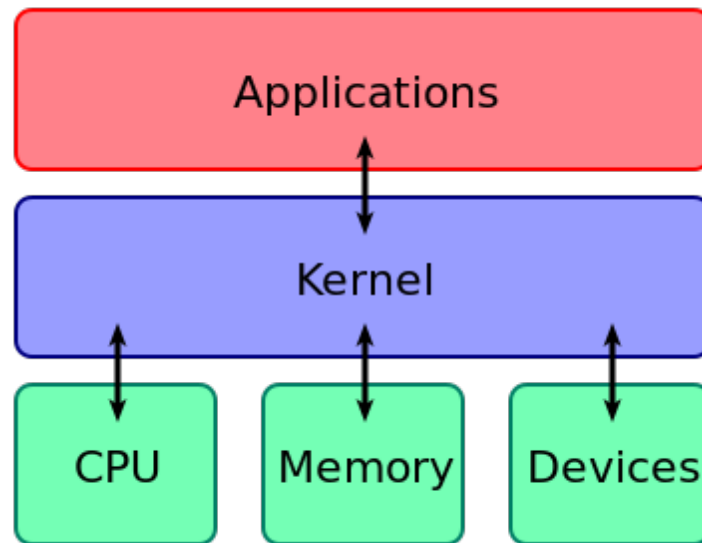


Figure 2.2: Diagramă care ilustrează locul nucleului in stiva sistemului

Există diferite abordări de proiectare a nucleelor, printre care monolithic kernel, care rulează toate instrucțiunile de sistem în același spațiu de adresare sau microkernel, mai modular, care rulează multe din procesele critice în user space.

Scopul principal al nucleului este, deci, de a partaja resursele sistemului, incluzând:

- CPU
- RAM
- dispozitive I/O
- management resurse
- management memorie
- management dispozitive
- apeluri sistem

Nucleul Linux, folosit în proiect, este unul de tip monolitic, fiind extrem de rapid și versatil. Proiectarea unui kernel fiabil s-a dovedit dificilă și adesea foarte costisitoare ca timp. Folosirea unui kernel consacrat și fiabil și construirea sistemului peste acesta reprezintă un compromis optim de timp și resurse. Nucleul Linux este cel mai popular kernel monolitic Unix-like și pe baza lui s-a construit familia de sisteme de operare GNU/Linux, care conține mii de distribuții. Datorită capabilităților sale multi-CPU, nucleul Linux și-a consolidat poziția pe piața server și enterprise.

Pentru compilarea și configurarea sistemului de operare gazdă pentru acest proiect s-a folosit Buildroot. Detalii despre acest proces, dar și despre sistemul de operare în sine se va elabora la capitolul de implementare, mai exact la secțiunea 4.2.1.

2.5 Grafică raster

În grafica pe calculator, bitmap-urile sau graficile rasterizate sunt reprezentări grafice sub forma unor matrici de puncte, vizibile pe un monitor, imprimabile sau vizibile pe alte medii de vizualizare. Există multe formate pentru stocarea fișierelor de acest fel.

Un bitmap (single-bit raster) corespunde bit cu bit unei imagini afișate pe ecran. O imagine raster este, de regulă, caracterizată de lungime, lățime și înălțime/adâncime a pixelilor, adică numărul de biți per pixel (adesea întâlnită în literatură ca și color depth).

În industria de printing, graficile raster sunt cunoscut ca "contones" (de la "continuous tones"). Opusul contones sunt așa numitele "line works", implementate ca grafici vectorizate în sistemele digitale. Imaginile vectorizate se pot rasteriza și viceversa folosind software. În ambele cazuri se pierde o cantitate mică de informație.

Imaginile raster sunt dependente de rezoluție, deci nu se pot scala în sus fără să se piardă calitatea aparentă. În contrast, imaginile vectorizate se pot scala atât cât pot fi randate de dispozitiv. Uzurile imaginilor rasterizate includ fotografii și alte aplicații în domenii practice, în timp ce imaginile vectorizate se folosesc la design grafic și typesetting.

2.6 CUPS

Common Unix Printing System este o platformă de imprimare care se ocupă cu orice ține de imprimante și imprimare, de la administrare și expunerea lor în rețea, la trimiterea de job-uri către cozi (imprimante). În CUPS se pot adăuga imprimante fie din utilitarul din linia de comandă, fie din interfața web. Odată adăugată, o imprimanta devino o 'coadă' (queue) și este expusă pe rețea. Felul în care funcționează CUPS și alte detalii se găsesc la capitolul de implementare.

Specificațiile proiectului

3.1 Placa de dezvoltare gazdă - Raspberry Pi 3

Probabil cea mai populara placa de dezvoltare cu microprocesor, Raspberry Pi 3 are următoarele specificații:

- procesor Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- 1GB LPDDR2 SDRAM
- placa de rețea 2.4GHz and 5GHz IEEE 802.11 b/g/n/ac
- Gigabit Ethernet over USB 2.0
- extended 40 pin GPIO header
- full-size HDMI
- 4 porturi USB 2.0
- slot de card micro SD
- 5V @ 2.5A intrare alimentare

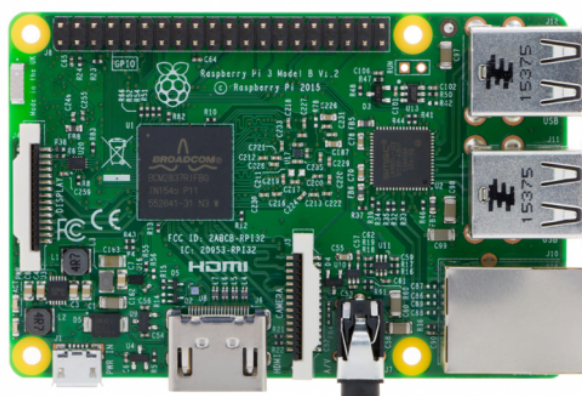


Figure 3.1: Placa de dezvoltare Raspberry Pi 3

3.2 Workflow-ul normal pentru tipărire

Un scenariu normal de printare prin CUPS presupune trimiterea pur și simplu a unui job de print către CUPS, iar acesta se va ocupa de restul. Un aspect important este abstractizarea imprimantelor în spatele 'cozilor' CUPS și se urmărește ca acest aspect să rămână neschimbat. CUPS asignează fiecărei imprimante una sau mai multe 'cozi' (queues), iar acestea apar ca imprimante pe rețea pentru sistemele Unix (OSX, GNU/Linux, BSD, etc.). Proiectul urmărește extragerea cât mai multor informații din procesul normal de printare CUPS, dar și menținerea modificărilor asupra CUPS la un nivel minim. O cerință importantă este ca workflow-ul normal de printare să rămână neschimbat, iar modificările aduse pentru colectarea de date să impacteze cât mai puțin platforma.

Modificările aduse CUPS trebuie să se respecte următoarele cerințe:

- să nu adauge overhead sistemului
- să nu expună informație proprietară Océ
- să nu modifice funcționarea modulelor CUPS
- să expună într-o manieră ușor de interceptat informațiile relevante
- să respecte licența Apache v2.0 aferentă CUPS

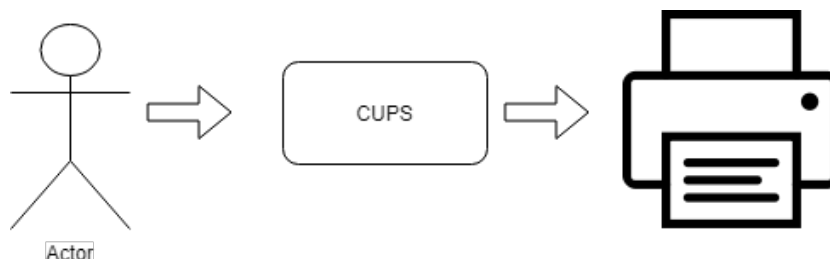


Figure 3.2: Workflow-ul normal

3.3 Colectarea și stocarea datelor

Modificările aduse CUPS urmăresc exclusiv colectarea datelor de imprimare și trimiterea lor către agregator. Colectarea datelor se face într-o manieră cât mai puțin invazivă, conform cerințelor anterior menționate. Se urmărește extragerea datelor spre a fi procesate, nu stocarea lor propriu zisă. Datele sunt transmise printr-un named pipe, prin urmare acestea nu părăsesc niciodată siguranța nucleului sistemului de operare. Lupa ce datele sunt recepționate și procesate de către agregator, acestea nu sunt salvate pe disc sau în alt mediu de stocare persistentă. Singurul loc în care datele persistă este în Elasticsearch, dar este importantă mențiunea că datele ajunse în Elasticsearch sunt deja procesate, iar în procesul de procesare pot fi eliminate orice informații nedorite.

Un alt criteriu important pe care proiectul trebuie să îl respecte este respectarea confidențialității datelor. În niciun punct sistemul nu colectează date cu caracter personal, nici măcar numele utilizatorului, jobului sau mașina de pe care a fost trimis un print job.

Design și implementare

4.1 Arhitectură high-level

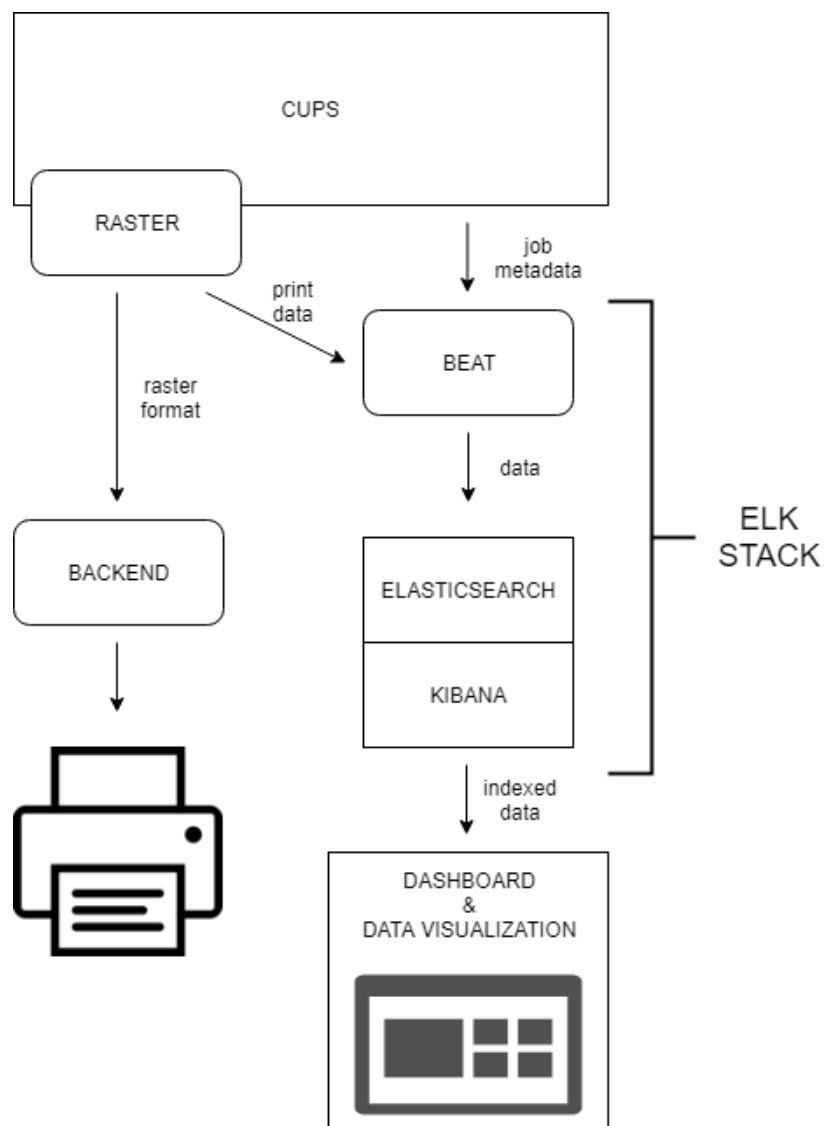


Figure 4.1: Arhitectura sistemului

4.2 Componente software

4.2.1 Sistemul de operare gazdă - BuruX

Folosind CUPS ca backbone pentru sistem, s-a făcut evidenta nevoie folosirii unui sistem de operare Unix-like care sa îl găzduiască. Pe considerente de familiaritate s-a ales folosirea unui sistem de operare bazat pe nucleul Linux. Nucleul Linux este un nucleu monolitic, adică lucrează în întregime separat de restul sistemului și în mod administrator. Driverile se pot încarca în memoria de lucru la utilizare, de unde sunt șterse ulterior. Nucleul ofer facilități precum:

- multitasking
- suport pentru memorie virtuala
- suport avansat pentru TCP/IP
- sistem de sunet
- pana la 1 miliard de procese simultane
- management avansat si eficient al memoriei
- suport pentru sistem multi-procesor

Distribuția a fost compilata pentru arhitectura ARM, specifica procesorului Broadcom BCM2837 cu 4 nuclee ARM Cortex-A53, spre a fi instalata pe o placa Raspberry Pi 3. Procesorul și memoria de 1GB ale plăcii sunt mai mult decât suficiente pentru a rula sistemul de operare și componentele menționate mai sus. Versiunea de kernel folosita este 4.14, cu niște patch-uri suplimentare care sa permită folosirea modulelor Bluetooth și Wi-Fi ale plăcii. În testele efectuate, sistemul de operare este perfect stabil și consuma foarte puține resurse comparativ cu o distribuție de Linux full blown. Compilarea completa a sistemului de operare se face, pe o mașina relativ veche, în sub 25 minute. Nucleul linux este principala componenta a sistemului de operare GNU/Linux. Pentru use-case-ul de față, am ales sa nu instalez un desktop environment sau interfața grafica pe sistem, pe considerente de reducere a resurselor consumate. Distribuția de Linux folosită a fost compilată special pentru acest uz, drept urmare majoritatea funcțiilor au fost eliminate, rămânând doar cu cele strict esențiale pentru funcționarea CUPS și transmiterea datelor:

- facilitati elementare de networking - Avahi
- o selectie minimala de unelte si compilatoare - clang, golang
- sistemul de printing CUPS cu dependentele aferente
- functionalitate de remote access - SSH

Pentru compilare și aplicare de patch-uri a fost folosit **Buildroot** - o unealta care automatizează procesul de building al unui mediu Linux complet și bootabil, folosind cross-compilation pentru a putea servi multiple platforme. Buildroot își poate build-ui propriul toolchain, crea un root file system, compila o imagine de Linux kernel și genera un boot loader pentru sistemul dorit. Acesta funcționează pe baza unor fișiere de configurații, numite defconfigs, care conțin informații relevante pentru procesul de build (versiune de

kernel, arhitectura, target file system, etc.). Sunt suportate mai multe biblioteci de C, printre care GNU C Library, uClibc și musl. Intern, Buildroot folosește Kconfig pentru sistemul de configurație, acesta oferind facilități precum o interfață cu meniu, ocuparea de dependente, opțiunea de 'Help' contextual. Întregul proces de build se construiește în jurul pachetelor descărcate automate în funcție de defconfig. Aceste pachete pot include aplicații, utilități, biblioteci, etc. Rezultatul final este un root file system care poate fi copiat pe un mediu bootabil și folosit as-is. Buildroot face tot procesul de compilare, build și deployment ușor de înțeles și accesibil oricui.

Pe parcurs, a apărut problematica alegerii unui sistem de init. Buildroot pune la dispoziție implicit BusyBox - o implementare a unui program rudimentar de init, suficient pentru majoritatea aplicațiilor.

Cu toate că inițial s-a dorit folosirea CUPS pe un Raspberry Pi cu un sistem de operare special conceput, acest lucru a devenit ulterior opțional, datorită ușurinței de instalare pentru testarea sistemului pe un calculator normal. Majoritatea testelor și dezvoltării s-au efectuat pe un sistem cu arhitectură x64 sub o distribuție de Debian Jessie. Pentru instalarea pe un sistem de operare customizat este nevoie de compilare specială pentru arhitectura respectivă. O alternativă mai simplă este folosirea unei distribuții Debian-based, adică Raspbian pentru Raspberry Pi.

4.2.2 CUPS

Common UNIX Printing System

Apărut în 1999, CUPS este un sistem modular de printing open-source pentru sistemele de operare Unix-like. Acesta permite unui calculator să se comporte ca un print server, devenind un host care accepta job-uri de la clienți, le procesează și le trimite către imprimante. CUPS este format dintr-un spooler, un scheduler, un sistem de filtre care transformă datele dintr-un format în altul, și un sistem de backend-uri care realizează comunicarea cu imprimantele.

Workflow-ul standard CUPS este următorul: un job ajunge în scheduler, acesta îl trimite către unul sau mai multe filtre spre a fi convertit în alt format, apoi datele ajung într-un backend de unde sunt trimise spre printer. Sistemul folosește extensiv PostScript și rasterizarea datelor ca formate înțelese de imprimante.

Scheduler-ul CUPS

Scheduler-ul CUPS implementează Internet Printing Protocol și oferă o interfață web-based pentru managementul imprimantelor, job-urilor, configurații server și documentație. Pentru accesarea funcționalităților de management și configurație, un utilizator trebuie să se autentifice. O altă parte foarte importantă a scheduler-ului este modulul de logging. Acesta înregistrează evenimente legate de acces, erori sau page log. Alte module importante ale scheduler-ului sunt: modulul MIME (multipurpose internet mail extensions), modulul PPD (PostScript printer description), modulul care se ocupa cu management-ul dispozitivelor disponibile în sistem.

Sistemul de filtre CUPS

Sistemul de filtre al CUPS poate procesa o varietate de formate. Acesta convertește datele unui print-job într-un limbaj/format final printr-un sistem de filtre, folosind MIME types pentru identificarea formatelor. Procesul de filtrare începe prin determinarea tipului de

date de intrare, folosind MIME databases. Printre filtrele default ale CUPS se număra: raster to PCL, raster to ESC/P, raster to Dymo, raster to ZPL. Inițial, am încercat modificare scheduler-ului, mai exact a modulului de logging, dar informațiile pe care le puteam obține din acesta erau insuficiente pentru cerințele proiectului, în modulul de logging fiind expuse numai date despre utilizatori, job-uri sau status. Considerând nevoia de a obține date despre pixeli, culoare, media, pagina, etc. s-a făcut necesara accesarea informațiilor disponibile în sistem de filtre al CUPS, mai exact în unitatea de raster. Lupa ce am identificat informațiile de care am nevoie, le-am trimis către exterior printr-un HTTP POST, apoi printr-un named pipe. Folosind un apel de sistem către comanda `cURL` pentru a trimite cererea HTTP către Beat, s-ar fi creat un număr de procese greu de controlat și care ar fi generat overhead pe care îl puteam evita. O a doua alternativa ar fi fost folosirea unei biblioteci, de exemplu `libcurl`, pentru a înlocui apelurile de sistem, dar aceasta alternativa ar fi introdus o dependentă suplimentară în CUPS, ceea ce ar fi făcut codul mai puțin portabil și ar fi încălcat una din cerințele inițiale ale proiectului. În final, am folosit un named pipe pentru transmiterea datelor, acesta fiind ușor de accesat și generând overhead minim, iar în plus, datele nu sunt expuse în rețea, aceasta interfață rezidând în kernelul sistemului de operare.

Backend-urile CUPS

Backend-urile CUPS sunt folosite pentru a trimite datele către imprimante. CUPS pune la dispoziție multe backend-uri: paralel, serial, USB, cups-pdf, PDF virtual printing, JetDirect, LPD, SMB, etc.

4.2.3 Elastic Stack

Beater - Elasticsearch - Kibana

Elastic Stack este o stivă de aplicații care permite extragerea, procesarea și vizualizarea datelor. În majoritatea cazurilor, acesta este găsit sub numele de ELK Stack (Elasticsearch - Logstash - Kibana). Pentru aceasta lucrare nu a fost folosit Logstash, fiind înlocuit cu un Elastic Beater ca unealta de colectare a datelor. Un Beat implementează interfața Beater și adună date spre a le trimite către Kibana sau, în cazul de față, Elasticsearch.

Beat

Beat-urile au fost adăugate recent la stack-ul ELK. Un Beat are responsabilitatea să colecteze date și să le trimită spre procesare sau vizualizare. Acesta trebuie să implementeze în Golang o interfață numită Beater interface, care conține metode de Start, Run și Stop. Aceasta lucrare folosește un Beat pentru a primi date de la raster-ul, raster care face parte din sistemul de filtre menționat anterior. Beat-ul primește raster data, face operații de procesare și sanitization minimale, apoi trimite datele către Elasticsearch. În implementările convenționale de Beat, datele sunt transmise la intervale periodice de timp către Elasticsearch sau Kibana. Implementarea Beat-ului din aceasta lucrare trimite date către Elasticsearch atunci când primește informații noi de la raster. cBeat (Beater-ul implementat pentru aceasta lucrare) următoarele componente principale:

- Metodele interfetei Beater
 - metoda Run

- * contine bucla principala a aplicatiei
- * in interiorul buclei se transmite evenimente catre restul ELK stack
- * un eveniment este trimis encodat ca JSON catre Elasticsearch
- * mai jos este exemplificata metoda Run din exemplul oferit de Elastic
- * se observa cum evenimentele sunt trimise la intervale periodice de timp

```

1 func (bt *Countbeat) Run(b *beat.Beat) error {
2 logp.Info("countbeat is running! Hit CTRL-C to stop it.")
3
4 bt.client = b.Publisher.Connect()
5 ticker := time.NewTicker(bt.config.Period)
6 counter := 1
7 for {
8 select {
9 case <-bt.done:
10 return nil
11 case <-ticker.C:
12 }
13
14 event := common.MapStr{
15 "@timestamp": common.Time(time.Now()),
16 "type": b.Name,
17 "counter": counter,
18 }
19 bt.client.PublishEvent(event)
20 logp.Info("Event sent")
21 counter++
22 }
23 }

```

Listing 4.1: exemplu metoda Run - golang

– metoda Stop

- * este chemata atunci când Beat-ul e semnalat sa se oprească
- * mai jos este exemplificata metoda Stop

```

1 func (bt *Countbeat) Stop() {
2 bt.client.Close()
3 close(bt.done)
4 }

```

Listing 4.2: exemplu metoda Stop - golang

• Recepționarea datelor din CUPS

- citirea din named pipe a evenimentelor se face prin deschiderea FIFO-ului ca orice alt fișier de pe disk, folosind modulul os al Golang

```

1 pipePath = "/tmp/cupsbeat"
2 if p, err = os.Open(pipePath); os.IsNotExist(err) {
3 log.Fatalf("Named pipe '%s' does not exist", pipePath)
4 } else if os.IsPermission(err) {
5 log.Fatalf("Insufficient permissions to read named pipe '%s': %s",
6 pipePath, err)
7 } else if err != nil {
8 log.Fatalf("Error while opening named pipe '%s': %s", pipePath,
9 err)
10 }

```

```

8 }
9 defer p.Close()

```

Listing 4.3: deschidere named pipe - golang

- FIFO-ului i s-a adăugat un mecanism de watch care ne notifica atunci când s-a efectuat orice scriere în pipe

```

1 c := make(chan notify.EventInfo, MAX_CONCURRENT_WRITERS)
2 notify.Watch(pipePath, c, notify.Write|notify.Remove)

```

Listing 4.4: watch pentru named pipe - golang

- funcția care supraveghează pipe-ul a fost lansata într-o go rutina în metoda Run

```

1 go func(){
2     readPipe()
3 }()

```

Listing 4.5: apel readPipe in go rutina separata - golang

- ca mecanism IPC s-a folosit un go channel intre green thread-ul care citește din pipe și cel al metodei principale

```

1 var messageQueue = make(chan Message_s_t)

```

Listing 4.6: instantiere canal - golang

• Generarea structurilor necesare prelucrării datelor

- pe baza enum-urilor definite în CUPS se urmărește generarea unor structuri de date echivalente în limbajul Go

```

1 typedef enum cups_adv_e      /***** AdvanceMedia attribute values
   *****/
2 {
3     CUPS_ADVANCE_NONE = 0,    /* Never advance the roll */
4     CUPS_ADVANCE_FILE = 1,    /* Advance the roll after this file */
5     CUPS_ADVANCE_JOB = 2,     /* Advance the roll after this job */
6     CUPS_ADVANCE_SET = 3,     /* Advance the roll after this set */
7     CUPS_ADVANCE_PAGE = 4     /* Advance the roll after this page */
8 } cups_adv_t;

```

Listing 4.7: exemplu enum CUPS - C

- pentru transmiterea eficienta a datelor, se păstrează encodarea diferitor opțiuni de tipărire în numere întregi
- generarea de cod Go din fișierele header care conțin structurile CUPS presupune parcurgerea lor
- rezultatul final trebuie sa fie un fișier .go compilabil care conține structurile echivalente
- s-a folosit un algoritm similar cu un automate care parcurge fișierul
- automatul salvează fiecare intrare într-un dicționar de dicționare de șiruri de caractere

```
1 var maps = make(map[string]map[string]string)
```

Listing 4.8: instantiere dictionar - golang

- fiecare intrare corespunde unui enum, cu dicționarul corespunzător
- intrările din dicționarul corespunzător reprezintă perechile de întregi - șir de caractere
- întregul modul de generare de cod a fost scris în limbajul Go

```
1 type HWRResolution_t struct {
2     Hr1 int
3     Hr2 int
4 }
5
6 type ImaginBoundingBox_t struct {
7     Ibb1 int
8     Ibb2 int
9     Ibb3 int
10    Ibb4 int
11 }
12
13 type PageSize_t struct {
14     Ps1 int
15     Ps2 int
16 }
17
18 type Message_t struct{
19     CutMedia int
20     Duplex int
21     HWRResolution HWRResolution_t
22     ImagingBoundingBox ImaginBoundingBox_t
23     InsertSheet int
24     Orientation int
25     NumCopies int
26     PageSize PageSize_t
27     Tumble int
28     CupsWidth int
29     CupsHeight int
30     CupsBitsPerColor int
31     CupsBitsPerPixel int
32     CupsColorOrder int
33     CupsColorSpace int
34     CupsNumColors int
35 }
36
37 type Message_s_t struct{
38     CutMedia string
39     Duplex int
40     HWRResolution HWRResolution_t
41     ImagingBoundingBox ImaginBoundingBox_t
42     InsertSheet int
43     Orientation string
44     NumCopies int
45     PageSize PageSize_t
46     Tumble int
47     CupsWidth int
48     CupsHeight int
49     CupsBitsPerColor int
```

```

50  CupsBitsPerPixel  int
51  CupsColorOrder   string
52  CupsColorSpace   string
53  CupsNumColors    int
54 }

```

Listing 4.9: structuri de date pentru generator/procesor date - golang

• Prelucrarea datelor

- la compilare, se populează structurile menționate la punctul anterior pe baza header-ului CUPS
- atunci când se recepționează un eveniment, acesta ajunge în Beat sub forma de string
- evenimentul este procesat pe baza dicționarilor, întregii fiind înlocuiți cu stringuri
- după prelucrare, evenimentul este serializat ca JSON și este trimis către Elasticsearch

```

1 func processMsg(msg string) Message_s_t{
2     var maps = cups_itf.Maps
3     fmt.Println(maps["Duplex"][0])
4     logp.Info("PROCESSING STRING:")
5     logp.Info(msg)
6     var message Message_t
7     var message_s Message_s_t
8     if isJSON(msg){
9         logp.Info("VALID JSON")
10        json.Unmarshal([]byte(msg), &message)
11        message_s.CutMedia = maps["CutMedia"][strconv.Itoa(message.
CutMedia)]
12        message_s.Duplex = message.Duplex
13        message_s.HWResolution = message.HWResolution
14        message_s.ImagingBoundingBox = message.ImagingBoundingBox
15        message_s.InsertSheet = message.InsertSheet
16        message_s.Orientation = maps["Orientation"][strconv.Itoa(
message.Orientation)]
17        message_s.NumCopies = message.NumCopies
18        message_s.PageSize = message.PageSize
19        message_s.Tumble = message.Tumble
20        message_s.CupsWidth = message.CupsWidth
21        message_s.CupsHeight = message.CupsHeight
22        message_s.CupsBitsPerColor = message.CupsBitsPerColor
23        message_s.CupsBitsPerPixel = message.CupsBitsPerPixel
24        message_s.CupsColorOrder = maps["cupsColorOrder"][strconv.Itoa(
message.CupsColorOrder)]
25        message_s.CupsColorSpace = maps["cupsColorSpace"][strconv.Itoa(
message.CupsColorSpace)]
26        message_s.CupsNumColors = message.CupsNumColors
27        logp.Info("CREATED NEW STRUCT")
28    }
29    return message_s
30 }

```

Listing 4.10: funcția pentru procesarea mesajelor - golang

Elasticsearch

Elasticsearch este un motor de căutare bazat pe Apache Lucene. Oferă clienți disponibili în majoritatea tehnologiilor și este cel mai popular motor de căutare enterprise. Beneficiile Elasticsearch includ: căutare foarte scalabilă, aproape real-time, suportul mai multor clienți, platforma distribuită. Este considerat de multi backbone-ul stack-ului ELK. Informațiile extrase de Beat ajung în Elasticsearch, iar acesta caută tipare și indexează datele spre a le trimite către Kibana.

Kibana

Kibana este un plugin open-source pentru vizualizare de date, care pune la dispoziție capacități de vizualizare ale elementelor indexate într-un cluster Elasticsearch. Un utilizator poate construi diferite tipuri de grafice bazate pe datele indexate de Elasticsearch, iar pe baza acestor grafice se pot construi dashboard-uri configurabile. Kibana preia datele indexate de la Elasticsearch și le folosește la generarea de grafice și vizualizări relevante pentru utilizator.

Stat Elasticsearch, ca și Kibana sunt hostate pe o mașină (Debian) în cloud pe platforma Scaleway. Mașina are o configurație minimală, stack-ul fiind unul robust și eficient nu are nevoie de multe resurse. Administrarea mașinii se face strict prin SSH, iar interfața web Kibana este expusă spre exterior. Prin intermediul aplicației web se face și configurarea graficelor și dashboard-ului Kibana. Pe mașina host este instalat nginx care redirecțiază orice acces HTTP către portul Kibana (9300).

4.3 Datele extrase

Proiectul urmărește extragerea datelor relevante pentru consum de cerneală, media, spațiu de culori, etc.

Majoritatea acestor informații sunt disponibile în sistemul de filtre CUPS menționat mai sus, mai exact în modulul de rasterizare. Aceste informații apar chiar în procesul de rasterizare.

CUPS determină în momentul dinaintea tipăririi dacă este nevoie sau nu de rasterizare în funcție de mai mulți factori:

- tipul și capacitățile cozii spre care se trimite job-ul
daca imprimanta suportă sau nu PostScript, etc.
- tipul de fișier imprimat
imagine raster, imagine vector, text, PDF, etc.
- coada respectivă are sau nu implementare de rasterizor/filtru CUPS
pentru cozile fără filtru implementat în CUPS se folosește rasterizorul CUPS

Odată ce CUPS a determinat folosirea unui filtru, datele despre fișier ajung în modulul respectiv. Modulul extrage pixel data și formează o imagine din pixeli. Imaginea din pixeli respectă constrângerile graficii raster:

- e formată dintr-o matrice de puncte (sau mai multe matrici suprapuse)

- are un color space
- are rezoluție

În continuare vor fi enumerate și explicate câteva dintre informațiile disponibile în CUPS.

CutMedia

CutMedia este o setare pentru imprimarea cu tăiere și poate avea oricare din valorile:

- CUPS_CUT_NONE - fără tăiere
- CUPS_CUT_FILE - tăiere după fișier
- CUPS_CUT_JOB - tăiere după job
- CUPS_CUT_SET - tăiere după set
- CUPS_CUT_PAGE - tăiere după pagină

Opțiunea este relevantă pentru imprimantele cu cutter sau pentru configurațiile de imprimantă și cutter în secvență.

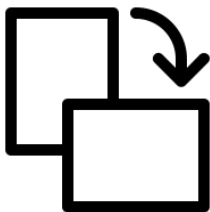
Duplex

Duplex este setarea care îi spune imprimantei dacă tipărirea trebuie efectuată pe ambele fețe ale paginii sau nu. Evident, coada spre care merge acest job trebuie să suporte această funcționalitate.

Hardware Resolution

HWResolution dă rezoluție hardware exprimată pe două dimensiuni.

Orientation



Describe orientarea dorită pentru tipărire. Opțiuni:

- CUPS_ORIENT_0 - fără rotire
- CUPS_ORIENT_90 - rotire în sens trigonometric
- CUPS_ORIENT_180 - pagina este întoarsă vertical
- CUPS_ORIENT_270 - rotire în sens orar

PageSize

Mărimea paginii pe două dimensiuni.

NumCopies

Numărul de copii.

BitsPerColor

Biți per culoare, cunoscută în literatură ca bit depth, este numărul de biți folșiți pentru la indicarea culorii unui singur pixel. Cele mai folosite valori sunt 32 și 24 biți.

BitsPerPixel

Biți per pixel sau bpp denotă numărul de biți pe un singur pixel. Acesta depinde de numărul de biți pe culoare. Numărul total de combinații este, evident, 2^{bpp} . Tabelul de mai jos ilustrează câte combinații se pot face cu diferite valori pentru bpp.

biți per culoare	culori
1 bpp	2 culori
2 bpp	4 culori
4 bpp	16 culori
6 bpp	64 culori
8 bpp	256 culori
10 bpp	1024 culori
16 bpp	65536 culori
24 bpp	16.7 milioane culori
32 bpp	4294 milioane culori

ColorSpace

Un color space este o organizare specifică a culorilor. În combinație cu profiling-ul fizic al dispozitivelor, permite reproducerea exactă a culorilor și reprezentarea lor cât mai aproape de adevăr. Un model de culoare este un model matematic abstract și descrie felul în care culorile sunt reprezentate în tuple de numere (triplu în cazul RGB sau quadruplu la CMYK). Adesea color space e folosit pentru a desemna un color model. Spre exemplu, multed color space-uri se bazează pe modelul de culoare RGB, nu există color space RGB singular.

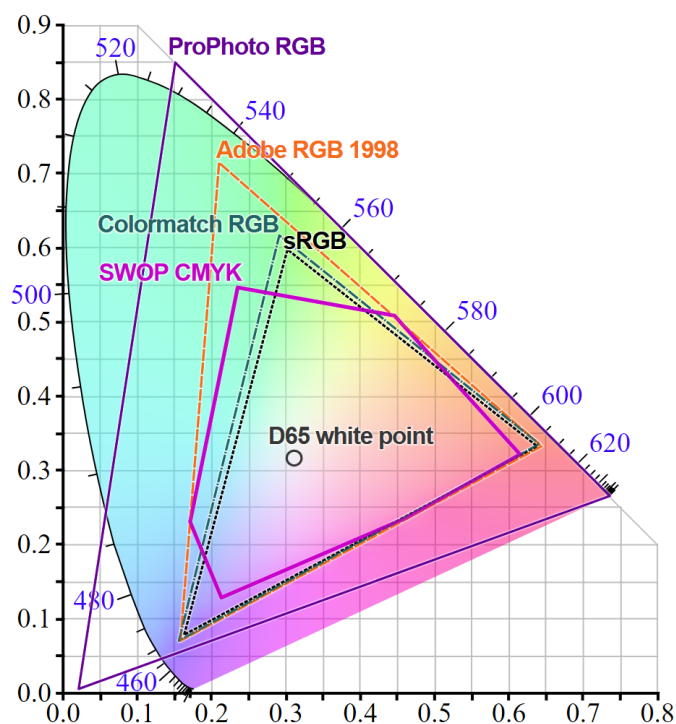


Figure 4.2: Spatii de culoare

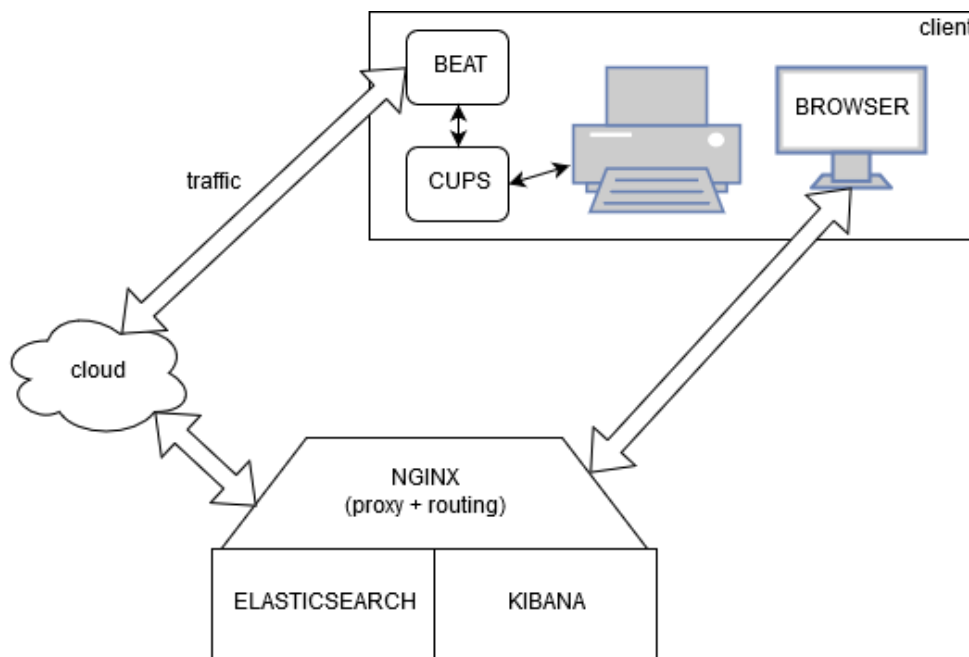


Figure 4.3: comunicarea CUPS - Beat - Elasticsearch - Kibana - Utilizator

Toate acestea, dar și alte informații ajută compania să evalueze mai bine piața. Printre lucrurile de înaltă relevanță pentru companie se numără informații despre cum sunt folosite, concret, imprimantele: ce se tipărește, în ce fel se tipărește, ce color space, ce dimensiuni, câte tăieturi se fac pe un job, ce folding, etc.

4.4 Build și deployment

Build-ul CUPS presupune compilarea fișierelor C din proiect și linkeditarea corespunzătoare. Apple pune la dispoziție un Makefile pentru build. Build-ul Beat se face prin compilarea fișierelor Go aferente, și el având un Makefile. Deployment-ul CUPS se face local, pe mașina de unde se urmărește extragerea informațiilor. Acest lucru se realizează prin copiere binarelor în locația corespunzătoare sau prin instalarea unui pachet .deb, .rpm, etc. generat pentru fiecare platformă în parte. Kibana și Elasticsearch sunt deployed pe o mașină în cloud, pe platforma Scaleway. Pe mașină este instalat nginx, care oferă capabilități de proxy și routing elementare. Dacă se introduce adresa mașinii într-un browser, nginx va redirecta direct către Kibana. Fiecare grafic din dashboard este configurabil. Configurația include tipul de metrice folosite, tipul de metoda de agregare, axele, câte metrice sunt folosite, câte și care câmpuri sunt folosite, etc.

4.5 Generare de date

Dat fiind că datele de tipărire care se pot colecta într-un timp scurt sunt limitate, s-a decis implementarea unui program care să simuleze tipărirea prin generarea de tichete JSON asemănătoare celor trimise de CUPS, dar cu valori arbitrare pentru fiecare câmp, evident cu constraint-urile aferente. Implementarea a fost făcută în Python3. Programul parcurge enum-urile C ale CUPS menționate anterior și generează pe baza lor un JSON. Acesta este trimis către Beat, spre a fi prelucrat ca orice tichet venit din CUPS. Stub-ul ajută la

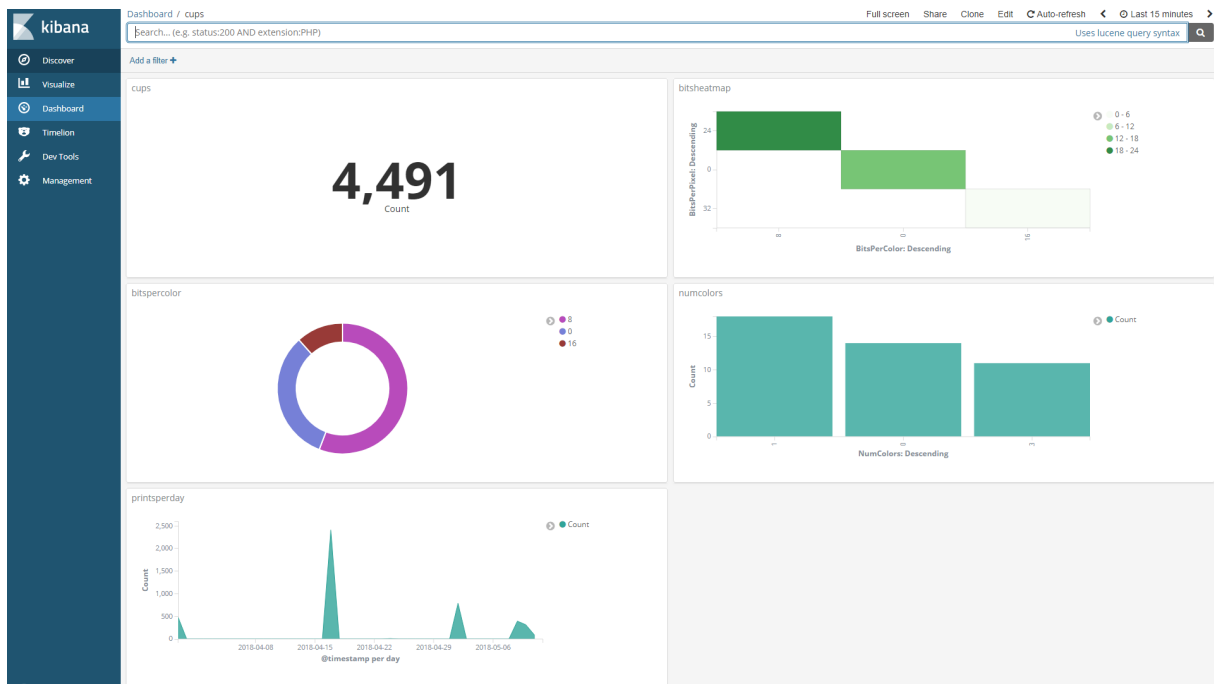


Figure 4.4: dashboard-ul Kibana intr-o fereastră de browser

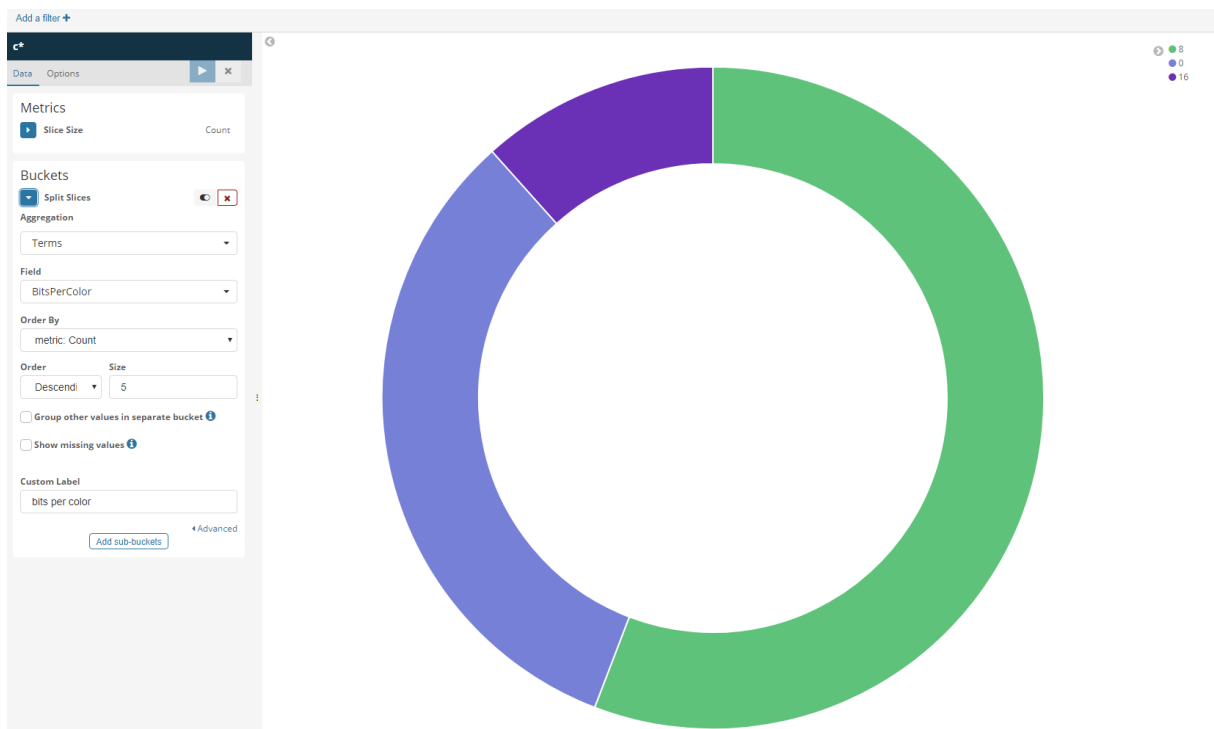


Figure 4.5: dashboard-ul Kibana intr-o fereastră de browser

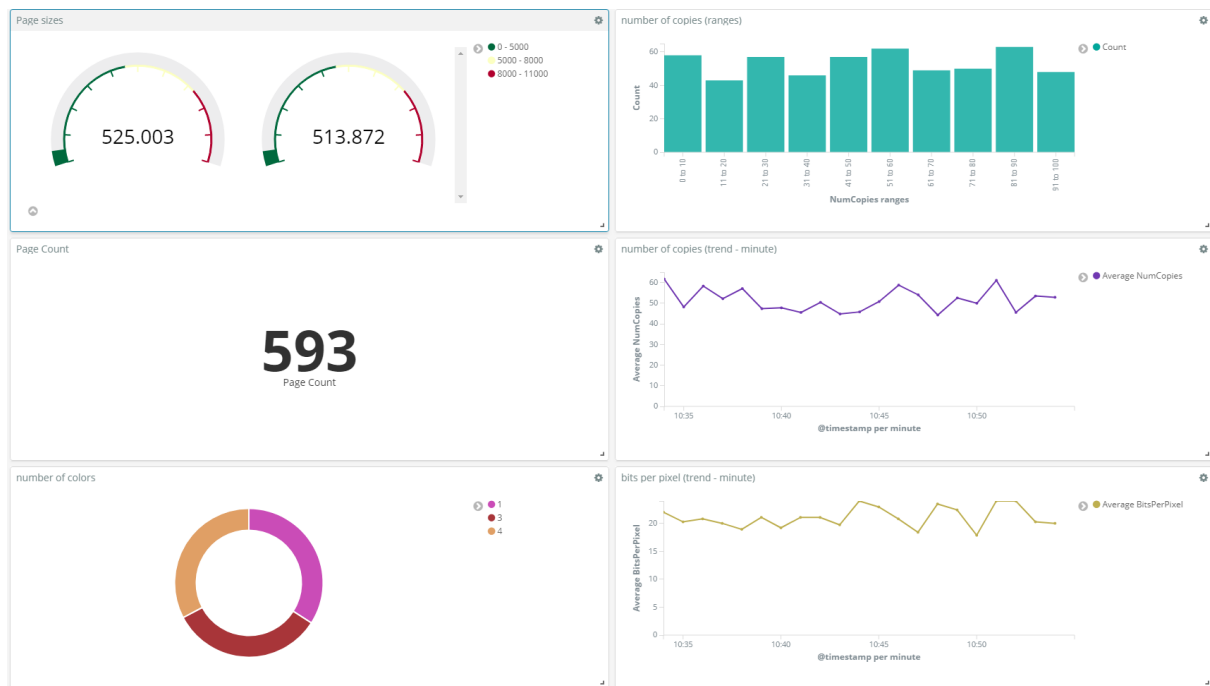


Figure 4.6: exemple de grafice in Kibana; de la dreapta la stanga, de sus in jos: page size, intervale de numere de copii, page count, trend in timp cu numere de copii, pie chart cu numar de culor si trend in timp cu bits per pixel

colectarea de date și exemplificarea funcționalității dashboard-ului Kibana. Generatorul de tichete este parametizat cu constraint-uri si valori rezonabile pentru fiecare camp din JSON.

```

1 # CONSTANTS
2 class Constraints(object):
3     def __init__(self):
4         self.cupsBitsPerColor = [8, 16, 32]
5         self.cupsBitsPerPixel = [8, 24, 32]
6         self.cupsColorOrder = [0, 1, 2]
7         self.cupsColorSpace = [0, 1, 2, 3, 4, 5]
8         self.CutMedia = [0, 1, 2, 3, 4]
9         self.Duplex = [0]
10        self.HW_RES_MAX = 8096
11        self.IBB_MAX = 1024
12        self.InsetSheet = [0]
13        self.cupsNumColors = [1, 3, 4]
14        self.NUM_COPIES_MAX = 100
15        self.Orientation = [0, 1, 2, 3]
16        self.PAGE_SIZE_MAX = 8096
17        self.Tumble = [0]
18        self.CUPS_HEIGHT_MAX = 1024
19        self.CUPS_WIDTH_MAX = 1024
20
21 const = Constraints()
22
23 def randomize(x):
24     return randint(0,x)
25
26 class Blank(object):

```

```

27     pass
28
29 special = ['HW_RES_MAX', 'IBB_MAX', 'NUM_COPIES_MAX', 'PAGE_SIZE_MAX', '
    CUPS_HEIGHT_MAX', 'CUPS_WIDTH_MAX']
30
31 def get_populated_class():
32     _json = Blank()
33     fields = const.__dict__
34     for k in fields:
35         if k in special:
36             pass
37         else:
38             setattr(_json, k, fields[k][randomize(len(fields[k])-1)])

```

Listing 4.11: cazul general dupa care se populeaza tichetul - Python

În cazurile speciale, câmpurile sunt adaugate individual, unul câte unul. Aceste câmpuri sunt cele desemnate de membrii listei 'special' definite mai sus.

```

1 # set HWResolution
2     res = Blank()
3     setattr(res, 'hr1', randomize(const.HW_RES_MAX))
4     setattr(res, 'hr2', randomize(const.HW_RES_MAX))
5     setattr(_json, 'HWResolution', res.__dict__)
6
7 # set ImagingBoundingBox
8     ibb = Blank()
9     setattr(ibb, 'ibb1', 0)
10    setattr(ibb, 'ibb2', 0)
11    setattr(ibb, 'ibb3', randomize(const.IBB_MAX))
12    setattr(ibb, 'ibb4', randomize(const.IBB_MAX))
13    setattr(_json, 'ImagingBoundingBox', ibb.__dict__)
14
15 # set NumCopies
16    setattr(_json, 'NumCopies', randomize(const.NUM_COPIES_MAX))
17
18 # set PageSize
19    ps = Blank()
20    setattr(ps, 'ps1', randomize(const.PAGE_SIZE_MAX))
21    setattr(ps, 'ps2', randomize(const.PAGE_SIZE_MAX))
22    setattr(_json, 'PageSize', ps.__dict__)
23
24 # set cupsHeight
25    setattr(_json, 'cupsHeight', randomize(const.CUPS_HEIGHT_MAX))
26
27 # set cupsWidth
28    setattr(_json, 'cupsWidth', randomize(const.CUPS_WIDTH_MAX))

```

Listing 4.12: cazurile speciale, tratate individual - Python

```

1 {
2     "HWResolution":{
3         "hr1":3272,
4         "hr2":670
5     },
6     "InsetSheet":0,
7     "PageSize":{
8         "ps2":361,
9         "ps1":6210
10    },

```

```

11  "CutMedia":2,
12  "Orientation":3,
13  "cupsColorOrder":1,
14  "Tumble":0,
15  "Duplex":0,
16  "ImagingBoundingBox":{
17    "ibb4":819,
18    "ibb3":863,
19    "ibb2":0,
20    "ibb1":0
21  },
22  "cupsBitsPerColor":8,
23  "cupsWidth":736,
24  "cupsHeight":163,
25  "NumCopies":43,
26  "cupsBitsPerPixel":24,
27  "cupsNumColors":3,
28  "cupsColorSpace":1
29 }

```

Listing 4.13: exemplu de output pentru programul de mai sus - JSON

Tichetul este citit de Beat, iar valorile numerice ale câmpurilor sunt traduse în siruri de caractere corespunzătoare din enum-urile CUPS. Această abordare permite inclusiv testarea infrastructurii de transmitere și colectare de date.

4.6 Limbajul de interogare Lucene

O interogare este descompusă sintactic în termeni și operatori. Termenii, la rândul lor, se împart în termeni simplii și fraze. Un termen simplu este un cuvânt, iar o frază este un grup de cuvinte. Mai mulți termeni pot fi combinați folosind operatori pentru a forma o interogare mai complexă. Analizor folosit pentru indexare se va folosi de termeni și fraze.

Câmpuri

Lucene suportă date sub forma câmpurilor. Atunci când se efectuează o interogare, sunt specificate câmpuri sau se folosesc câmpuri implicite. Numele câmpurilor sunt specifice fiecărei implementări. Pentru a căuta după un câmp se va folosi numele câmpului urmat de ":", apoi valoarea.

```
1 title:"The Right Way" AND text:go
```

Listing 4.14: exemplu de căutare după câmpuri - Lucene query

Căutări wildcard

Lucene suportă wildcards cu unul sau mai multe caractere în căutări.

- pentru căutare wildcard cu un singur caracter se va folosi simbolul "?"

```
1 te?t
```

Listing 4.15: căutare wildcard cu un singur caracter - Lucene query

- pentru căutare wildcard multi-caracter se va folosi simbolul "*"


```
1 test
```

Listing 4.16: căutare wildcard multi-caracter - Lucene query

Căutări fuzzy

Sunt suportate căutările fuzzy bazate pe algoritmi Levenshtein Distance sau Edit Distance. Pentru a efectua o căutare fuzzy se va folosi simbolul " " la finalul unui termen. După Lucene 1.9 a fost adăugat un parametru suplimentar cu valoare între 0-1 care reprezintă similitudinile cu termenul căutat, pe o scară de la 0 la 1. Implicit se folosește 0.5.

```
1 test~0.9
```

Listing 4.17: căutare fuzzy - Lucene query

Căutări pe intervale

Permit găsirea acelor intrări care satisfac o condiție de genul A mai mic decât x mai mic decât B. Pentru interval închis, deschis se folosesc acolade, respectiv paranteze drepte.

```
1 mod_date:[20020101 TO 20030101]
2
3 title:{Aida TO Carmen}
```

Listing 4.18: căutări pe intervale - Lucene query

Elevarea relevanței unui termen

Lucene permite ridicarea relevanței unui termen folosind operatorul special urmat de nivelul dorit. Cu cât nivelul este mai mare, cu atât termenul va fi mai relevant în interogare.

```
1 "jakarta apache" ^4 "Apache Lucene"
```

Listing 4.19: elevarea relevanței - Lucene query

Operatori

Operatorul implicit pentru conjuncție este SAU. Alți operatori sunt: SI, +, NOT, -.

4.7 Limbajul de interogare Elasticsearch

Kibana acceptă atât interogări Lucene, cât și interogări Elasticsearch. Oricare interogare Elasticsearch este scrisă sub forma unui JSON. Câmpul query din obiect desemnează tipul interogării.

```
1 GET /bank/_search
2 {
3   "query": {
4     "bool": {
5       "must": { "match_all": {} },
6       "filter": {
7         "range": {
```

```

8      "balance": {
9          "gte": 20000,
10         "lte": 30000
11     }
12 }
13 }
14 }
15 }
16 }

```

Listing 4.20: exemplu interogare Elasticsearch - JSON

Interogarea din listing-ul de mai sus va returna toate conturile cu balance cuprins între 20000 si 30000. Se observă cum interogarea conține un *match_all* si un *range*, ambele indicând tipuri de interogări.

Implicit, interogările returnează documentul JSON întreg ca parte a câmpului *_source*. Dacă se dorește ca documentul să nu fie returnat, putem cere doar anumite câmpuri din obiect. Peste o interogare se pot aplica filtre si agregări. Elasticsearch este un produs atât simplu cât si complex. În general se folosește împreună cu un RESTful API.

4.8 Platforma Scaleway

Pentru deployment s-a ales platforma Scaleway pe considerente practice si economice. Ea oferă performanțe similare platformelor mainstream la o fracțiune din preț. Scaleway folosește masini ARM-based. Principalul avantaj față de alte platforme este că pornind de la pachetul de bază, toate includ stocare SSD, cu opțiune pentru NVMe SSD. Scaleway ofera root access complet pe un server dedicat. Performanțele sunt constante si predictibile, spre deosebire de servere virtuale. Platforma oferă flexibilitate pentru plata serviciilor, taxarea fiind făcută dinamic în funcție de timpul folosit. Pe lângă asta, Scaleway este printre singurele companii mari de hosting europene, având servere in Amsterdam si Paris. Latența din Timisoara este de sub 40 ms. Administrarea se face printr-o platforma web.

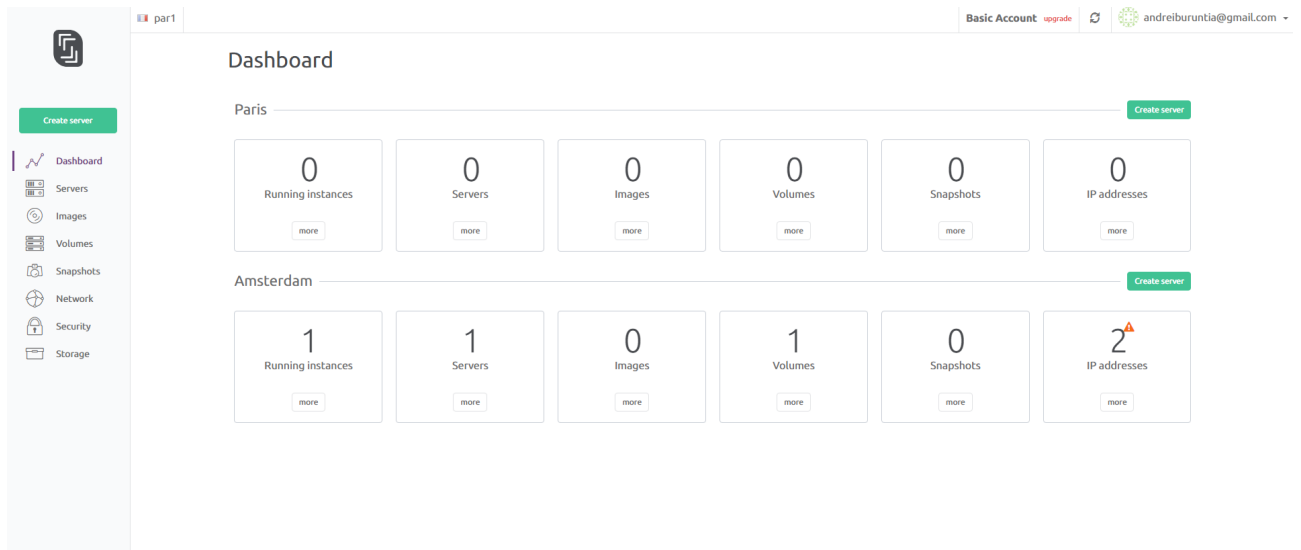


Figure 4.7: Dashboard-ul Scaleway

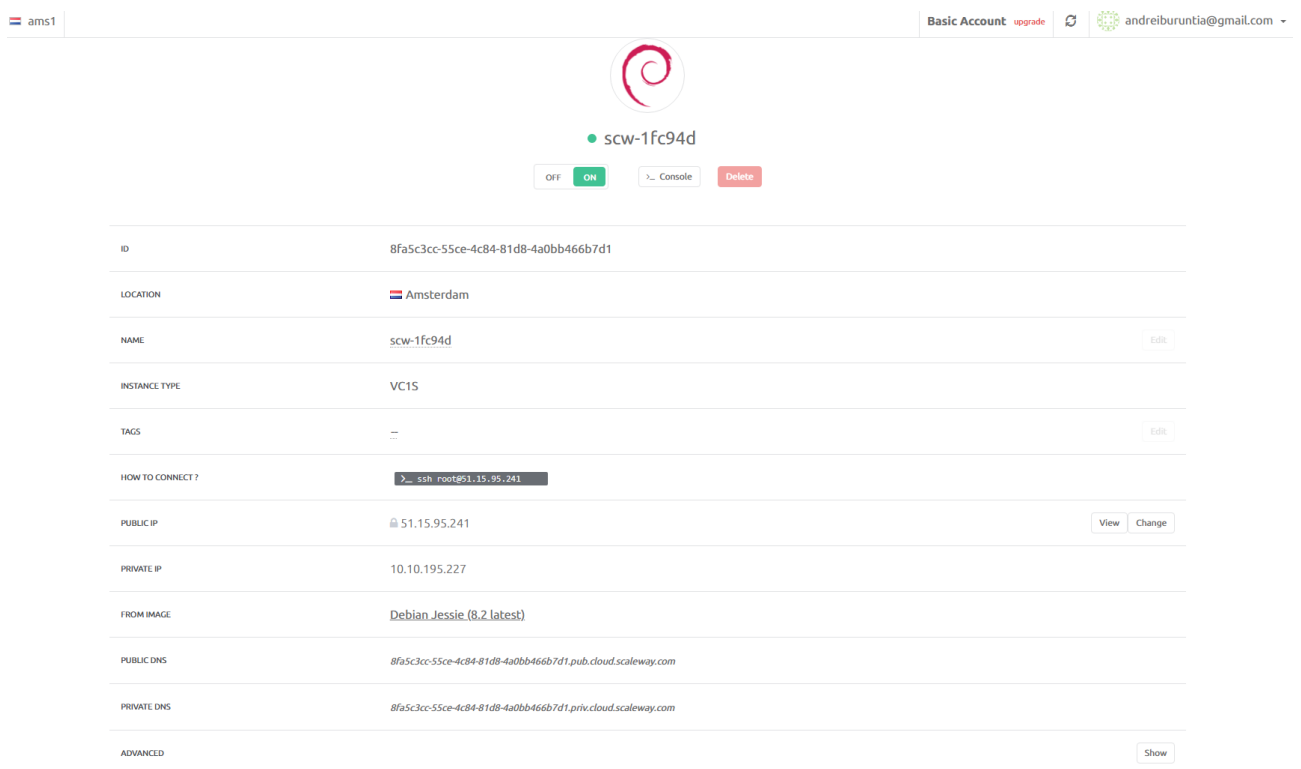


Figure 4.8: Dashboard-ul Scaleway - administrarea unei instanțe de server

Concluzii

Am urmărit și realizat crearea unui sistem de tipărire care trimite date către un agregator spre a fi procesate în scopul generării de analytics. Prin modificările aduse CUPS, workflow-ul normal nu este modificat în vreun fel, nu apare overhead semnificativ, nu se modifica nici o interfață internă sau externă CUPS și extragerea datelor se face într-o manieră elegantă, non-invazivă. Platforma Kibana și facilitățile ei satisfac cerințele legate de agregarea și vizualizarea datelor colectate din CUPS. Informațiile proprietare Océ nu sunt expuse. Folosirea limbajului Go permite extinderea ulterioară a proiectului cu un cost relativ redus și facilitează posibilitatea folosirii pachetelor și a funcțiilor unui limbaj high-level. Modificările la nivelul clienților sunt aproape inexistente, aceștia trebuind doar să folosească versiunea modificată de CUPS, care, după cum a fost menționat mai sus, nu alterează comportamentul sau aspectul acestuia. Informațiile colectate sunt suta la suta anonime, nefiind trimise date confidențiale legate de utilizator sau mașina acestuia.

Folosirea ELK stack și CUPS a scăzut considerabil timpul de lucru, scutind dezvoltatorul de implementarea unor astfel de sisteme.

Scalabilitatea proiectului este, în teorie, mărginită doar de capacitățile de stocare de date și generarea graficelor. O mașină cu un singur nucleu x64, 1 GB de memorie și conexiune de 100Mb/s s-a descurcat în toate testele efectuate. Un eveniment encodat în JSON are dimensiuni de ordinul sutelor de bytes (de obicei mult sub 1kB). Pe 10 GB de stocare persistentă se pot salva peste 10^7 astfel de intrări. Având în vedere disponibilitatea spațiului de stocare în zilele noastre, numărul de mai sus respectă orice cerință rezonabilă. Sistemul poate colecta date de la oricâte Beat-uri de pe oricâte mașini distincte.

Câteva lucruri de luat în considerare sunt:

- este evidentă utilitatea folosirii tehnologiilor open-source și clădirea peste sistemelor 'tried and tested'
- am remarcat avantajele limbajului Go atunci când se dorește folosirea capacităților unui limbaj de nivel înalt, dar și păstrarea funcționalităților low-level oferite de limbaje precum C
- stack-ul ELK este unul robust, solid și eficient
- uneltele IPC puse la dispoziție de nucleul Linux (named pipe) sunt suficient de robuste și eficiente pentru a fi folosite cu tehnologii moderne
- abundența dispozitivelor și gradul de dezvoltare rapid a Internetului vor permite colectarea datelor de orice fel

- am remarcat importanța abilității CUPS de a comunica și rezolva o multitudine de imprimante, fără ca utilizatorul să facă vreun efort
- am remarcat versatilitatea și fezabilitatea modelului client-server, folosit aici în două instanțe cu constringent-uri și parametrii diferiți
- în zilele noastre, un sistem poate colecta, procesa și transmite cu succes date în unități de timp de ordinul sutelor de microsecunde sau zecilor de milisecunde
- am remarcat flexibilitatea unei licențe permissive precum Apache v2.0
- contribuțiile mele strict tehnice la realizarea acestui proiect includ:
 - instalarea și configurarea stack-ului ELK în cloud folosind platforma Scaleway
 - dezvoltarea, configurarea, compilarea și deployment-ul sistemului de operare BuruX bazat pe kernelul Linux
 - modificarea, compilarea și deployment-ul sistemului de printing CUPS
 - conceperea unei arhitecturi/infrastructuri de comunicație
 - dezvoltarea Beater-ului
 - inițializarea și configurarea canalelor de comunicație
 - dezvoltarea de scripturi și programe care să ajute la build sau deployment
 - dezvoltarea de programe care să joace rolul anumitor componente din sistem (stubbing)
 - testarea modulară
 - testarea întregului sistem
 - învățarea tehnologiilor, conceptelor, limbajelor de programare aferente realizării fiecărui din punctele anterior menționate
- contribuții de altă natură includ studiul problemei, propunerea și selecția unor soluții fiabile, etc.
- CUPS poate fi compilat pentru diferite arhitecturi, acest lucru crescând portabilitatea sistemului
- Beat-ul nu trebuie să fie pe același sistem cu CUPS, named pipe-ul putând fi înlocuit cu orice mecanism de transmisie de date (HTTP POST, sockets, etc.)
- pornind de la ultimele două puncte, putem spune că întregul sistem este portabil și flexibil, putând fi instalat pe orice mașină pentru care se poate compila CUPS
- dashboard-ul poate fi accesat de oriunde, atâta timp cât porturile sunt deschise
- datele extrase ajută compania să înțeleagă cum se folosesc imprimantele: câte tăieturi, ce tip de folding, ce tip de culori, câtă cerneală, etc.

Glosar de termeni

- CUPS - Common Unix Printing System
- PDF - Portable Document Format
- GNU - GNU Not Unix (un acronim recursiv)
- Linux - familie de sisteme de operare bazate pe nucleul cu același nume
- OSX - sistem de operare desktop dezvoltat de Apple
- BSD - Berkeley Software Distribution - sistem de operare Unix-like
- kernel - nucleul unui sistem de operare
- Avahi - o implementare de zeroconf networking
- Golang - limbaj de programare dezvoltat de Google; compilatorul pentru limbajul Go
- clang - compiler front end pentru backend-ul LLVM
- Elastic stack - ELK stack - stiva de aplicatii Elastic
- nginx - web server dezvoltat de Igor Sysoev
- IPC - inter process communication

Figuri

2.1	Arhitectură client-server	11
2.2	Diagramă care ilustrează locul nucleului in stiva sistemului	13
3.1	Placa de dezvoltare Raspberry Pi 3	15
3.2	Workflow-ul normal	16
4.1	Arhitectura sistemului	17
4.2	Spații de culoare	27
4.3	comunicarea CUPS - Beat - Elasticsearch - Kibana - Utilizator	28
4.4	dashboard-ul Kibana intr-o fereastră de browser	29
4.5	dashboard-ul Kibana intr-o fereastră de browser	29
4.6	exemple de grafice in Kibana; de la dreapta la stanga, de sus in jos: page size, intervale de numere de copii, page count, trend in timp cu numere de copii, pie chart cu numar de culor si trend in timp cu bits per pixel . . .	30
4.7	Dashboard-ul Scaleway	35
4.8	Dashboard-ul Scaleway - administrarea unei instanțe de server	35

Listings

4.1	exemplu metoda Run - golang	21
4.2	exemplu metoda Stop - golang	21
4.3	deschidere named pipe - golang	21
4.4	watch pentru named pipe - golang	22
4.5	apel readPipe in go rutina separata - golang	22
4.6	instantiere canal - golang	22
4.7	exemplu enum CUPS - C	22
4.8	instantiere dictionar - golang	23
4.9	structuri de date pentru generator/procesor date - golang	23
4.10	functia pentru procesarea mesajelor - golang	24
4.11	cazul general dupa care se populeaza tichetul - Python	30
4.12	cazurile speciale, tratate individual - Python	31
4.13	exemplu de output pentru programul de mai sus - JSON	31
4.14	exemplu de căutare după câmpuri - Lucene query	32
4.15	căutare wildcard cu un singur caracter - Lucene query	32
4.16	căutare wildcardmulti-caracter - Lucene query	33
4.17	căutare fuzzy - Lucene query	33
4.18	căutari pe intervale - Lucene query	33
4.19	elevarea relevanței - Lucene query	33
4.20	exemplu interogare Elasticsearch - JSON	33

Referințe

- <https://www.scaleway.com/faq/general/>
- Linux Online (2008). "Linux Logos and Mascots". Archived from the original on August 15, 2010. Retrieved August 11, 2009.
- Torvalds, Linus (January 5, 1992). "Release notes for Linux v0.12". Linux Kernel Archives.
- "The Linux Kernel Archives: Frequently asked questions". kernel.org. September 2, 2014. Archived from the original on September 5, 2015. Retrieved September 4, 2015.
- "What Is Linux: An Overview of the Linux Operating System". Linux Foundation. April 3, 2009. Archived from the original on August 13, 2011. Retrieved August 15, 2011.
- Tung, Liam (July 27, 2017). "Raspberry Pi: 14 million sold, 10 million made in the UK — ZDNet". ZDNet.
- Upton, Eben (14 March 2018). "Raspberry Pi 3 Model B+ on Sale at USD35". Raspberry Pi Blog. Raspberry Pi .
- "Distributed Application Architecture" (PDF). Sun Microsystem. Archived from the original (PDF) on 6 April 2011. Retrieved 2009-06-16.
- Benatallah, B.; Casati, F.; Toumani, F. (2004). "Web service conversation modeling: A cornerstone for e-business automation". IEEE Internet Computing. 8: 46. doi:10.1109/MIC.2004.1260703.
- Roch, Benjamin (2004). "Monolithic kernel vs. Microkernel" (PDF). Archived from the original (PDF) on 2006-11-01. Retrieved 2006-10-12.
- Silberschatz, Abraham; James L. Peterson; Peter B. Galvin (1991). Operating system concepts. Boston, Massachusetts: Addison-Wesley. p. 696. ISBN 0-201-51379-X.
- Ball, Stuart R. (2002) [2002]. Embedded Microprocessor Systems: Real World Designs (first ed.). Elsevier Science. ISBN 0-7506-7534-9.
- Deitel, Harvey M. (1984) [1982]. An introduction to operating systems (revisited first ed.). Addison-Wesley. p. 673. ISBN 0-201-14502-2.

- Denning, Peter J. (December 1976). "Fault tolerant operating systems". ACM Computing Surveys. 8 (4): 359–389. doi:10.1145/356678.356680. ISSN 0360-0300.
- Shaw, Alan C. (1974). The logical design of Operating systems. Prentice-Hall. p. 304. ISBN 0-13-540112-7.
- Tanenbaum, Andrew S. (1979). Structured Computer Organization.
- Davidson and Fraser; The Design and Application of a Retargetable Peephole Optimizer; ToPLaS v2(2) 191-202 (April 1980)
- Rainer Leupers, Peter Marwedel. "Retargetable Compiler Technology for Embedded Systems"

