

Thesis Title

Universitatea Politehnica Timișoara



Andrei Buruntia  
andreiburuntia@gmail.com

Advisor: conf. dr. ing. Dan Cosma

20-02-2018

# Abstract

În zilele noastre există o multitudine de imprimante și soluții de printing disponibile, fiecare cu interfața și setul ei de funcționalități. Această teză urmărește construirea unei platforme universale care să permită ascunderea imprimantelor după un strat de abstractizare suplimentar, oferind o singură interfață și posibilitatea de a furniza unei companii de printing date relevante sub formă de grafice.

Adresarea problemei anterior menționată necesită depășirea anumitor obstacole, trei dintre care fiind: abstractizarea imprimantei și a funcțiilor ei, integrarea imprimantelor moderne de orice fel, realizarea unor rapoarte închegate și coerente cu date extrase din procesul de printare, care să prezinte relevanță pentru utilizator, ideal permițând utilizatorului să își creeze propria interfață, după nevoile și preferințele lui.

În consecință, lucrarea mea propune o soluție bazată pe CUPS, care se ocupă de comunicațiile și controlul imprimantelor și de expunerea lor pe rețea. Acest lucru permite acoperirea unei game largi de imprimante, pastrearea complexității la un nivel relativ redus și accesarea informațiilor lower-level.

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 răspundere

Subsemnatul Buruntia Andrei-Cristian, legitimat cu CI seria TZ nr. 080265, CNP 1950819165656, autorul lucrării TITLU, 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.

---

Semnătură

---

Data



# Cuprins

<b>1</b>	<b>Introducere</b>	<b>6</b>
1.1	Context și motivație . . . . .	6
1.2	Problema . . . . .	6
<b>2</b>	<b>Fundamentare teoretică</b>	<b>8</b>
<b>3</b>	<b>Specificațiile proiectului</b>	<b>9</b>
3.1	Placa de dezvoltare gazdă - Raspberry Pi 3 . . . . .	9
3.2	Workflow-ul normal pentru tipărire . . . . .	10
3.3	Colectarea și stocarea datelor . . . . .	10
<b>4</b>	<b>Design și implementare</b>	<b>11</b>
4.1	Arhitectură high-level . . . . .	11
4.2	Componente software . . . . .	12
4.2.1	Sistemul de operare gazdă - BuruX . . . . .	12
4.2.2	CUPS . . . . .	13
4.2.3	Elastic Stack . . . . .	14
4.3	Build și deployment . . . . .	19
4.3.1	Stubbing . . . . .	21
4.3.2	Limbaajul de interogare Lucene . . . . .	23
4.3.3	Limbaajul de interogare Elasticsearch . . . . .	25
4.3.4	Platforma Scaleway . . . . .	25
<b>5</b>	<b>Rezultate</b>	<b>27</b>
<b>6</b>	<b>Concluzii</b>	<b>28</b>
<b>7</b>	<b>Glosar de termeni</b>	<b>30</b>
<b>8</b>	<b>Referințe</b>	<b>33</b>

# 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 cerneală 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 comportă 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 cerneală 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. În această interfață grafică 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 imprimantă. 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 imprimantă
4. Să implice modificări minime la nivelul infrastructurii clienților
5. Să anonimizeze informațiile confidențiale



# Fundamentare teoretică

# 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

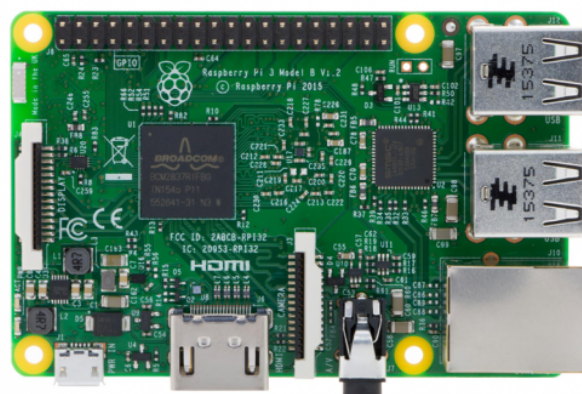


Figure 3.1: Placa de dezvoltare Raspberry Pi 3

## 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

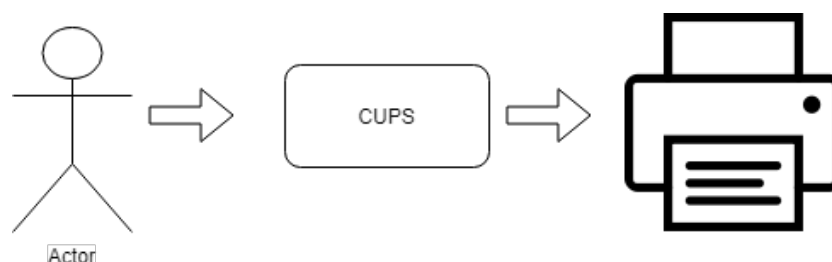


Figure 3.2: Workflow-ul normal

## 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 persista 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

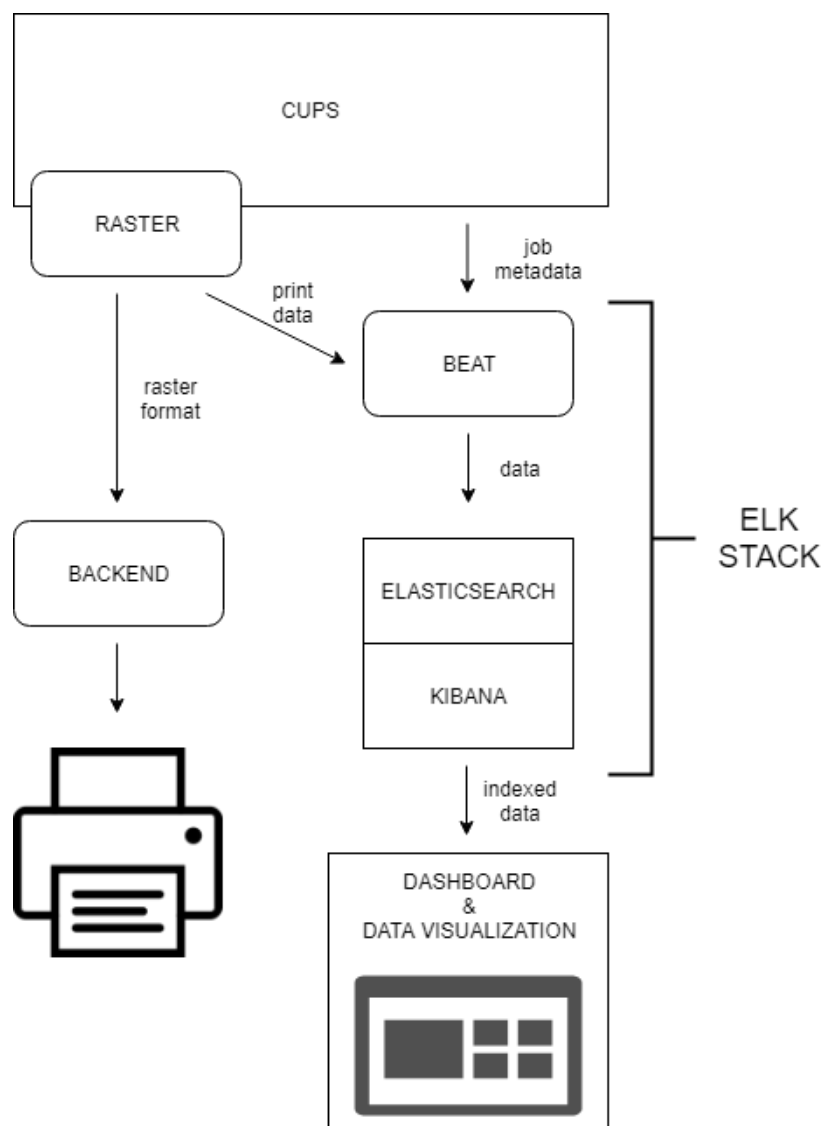


Figure 4.1: Arhitectura sistemului

## 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 oferă facilități precum:

- multitasking
- suport pentru memorie virtuală
- suport avansat pentru TCP/IP
- sistem de sunet
- până la 1 miliard de procese simultane
- management avansat și eficient al memoriei
- suport pentru sistem multi-procesor

Distribuția a fost compilată pentru arhitectura ARM, specifică procesorului Broadcom BCM2837 cu 4 nuclee ARM Cortex-A53, spre a fi instalată pe o placă 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 folosită este 4.14, cu niște patch-uri suplimentare care să permită folosirea modulelor Bluetooth și Wi-Fi ale plăcii. În testele efectuate, sistemul de operare este perfect stabil și consumă foarte puține resurse comparativ cu o distribuție de Linux full blown. Compilarea completă a sistemului de operare se face, pe o mașină relativ veche, în sub 25 minute. Nucleul Linux este principala componentă a sistemului de operare GNU/Linux. Pentru use-case-ul de față, am ales să nu instalez un desktop environment sau interfața grafică 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 cele strict esențiale pentru funcționarea CUPS:

- facilități elementare de networking - Avahi
- o selecție minimală de unelte și compilatoare - clang, go lang
- sistemul de printing CUPS cu dependentele aferente
- funcționalitate de remote access - SSH

Pentru compilare și aplicare de patch-uri a fost folosit Buildroot - o unealtă 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, compilează o imagine de Linux kernel și generează 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.

## 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 numără: raster to PCL, raster to ESC/P, raster to Dymo, raster to ZPL. Inițial, am încercat modificarea 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 necesară 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 alternativă ar fi fost folosirea unei biblioteci, de exemplu `libcurl`, pentru a înlocui apelurile de sistem, dar aceasta alternativă ar fi introdus o dependență 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 teză 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

- \* conține bucla principală a aplicației
    - \* în interiorul buclei se transmit evenimente către restul ELK stack
    - \* un eveniment este trimis encodat ca JSON către Elasticsearch
    - \* mai jos este exemplificată metoda Run din exemplul oferit de Elastic
    - \* se observă cum evenimentele sunt trimise la intervale periodice de timp

```

1 func (bt *Countbeat) Run(b *beat.Beat) error {
2 logp.Infof("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.Infof("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 }
11 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 lansată într-o go rutină în metoda Run

```

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

```

Listing 4.5: apel readPipe în go rutină separată - golang

- ca mecanism IPC s-a folosit un go channel între 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 {
4     CUPS_ADVANCE_NONE = 0,      /* Never advance the roll */
5     CUPS_ADVANCE_FILE = 1,      /* Advance the roll after this file */
6     CUPS_ADVANCE_JOB = 2,       /* Advance the roll after this job */
7     CUPS_ADVANCE_SET = 3,       /* Advance the roll after this set */
8     CUPS_ADVANCE_PAGE = 4       /* Advance the roll after this page */
9 } cups_adv_t;

```

Listing 4.7: exemplu enum CUPS - C

- pentru transmiterea eficientă a datelor, se păstrează encodarea diferitelor opțiuni de tipărire în numere întregi
- generarea de cod Go din fișierele header care conțin structurile CUPS presupune parcurgerea lor
- rezultatul final trebuie să 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 dicționar - golang

- fiecare intrare corespunde unui enum, cu dicționarul corespunzător
- intrările din dicționarul corespunzător reprezintă perechi 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_s.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.

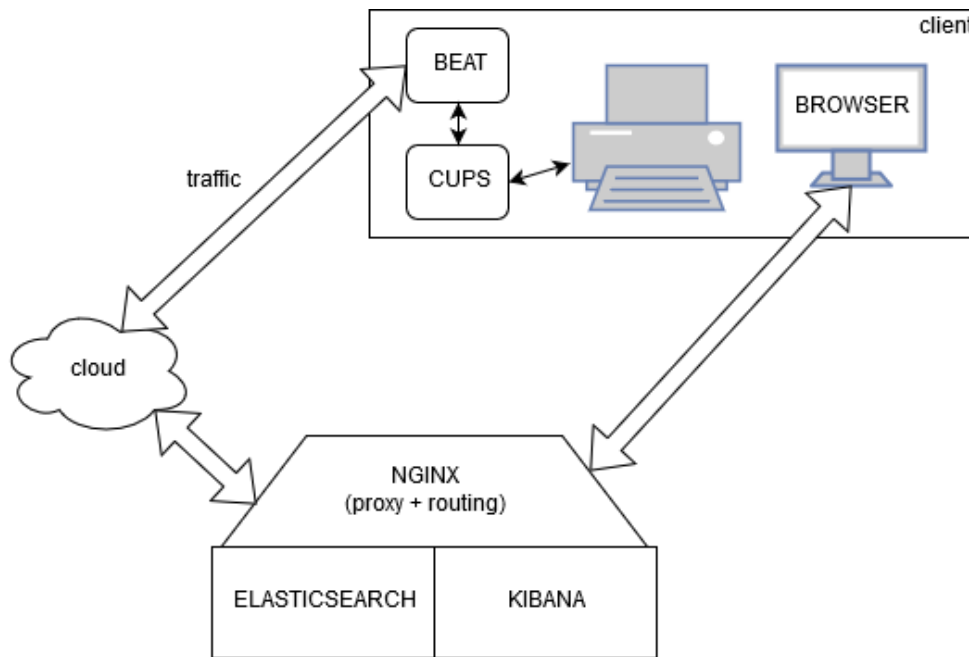


Figure 4.2: comunicarea CUPS - Beat - Elasticsearch - Kibana - Utilizator

## Kibana

Kibana este un plugin open-source pentru vizualizare de data, care pune la dispoziție capabilități 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 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 redirecționa 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.

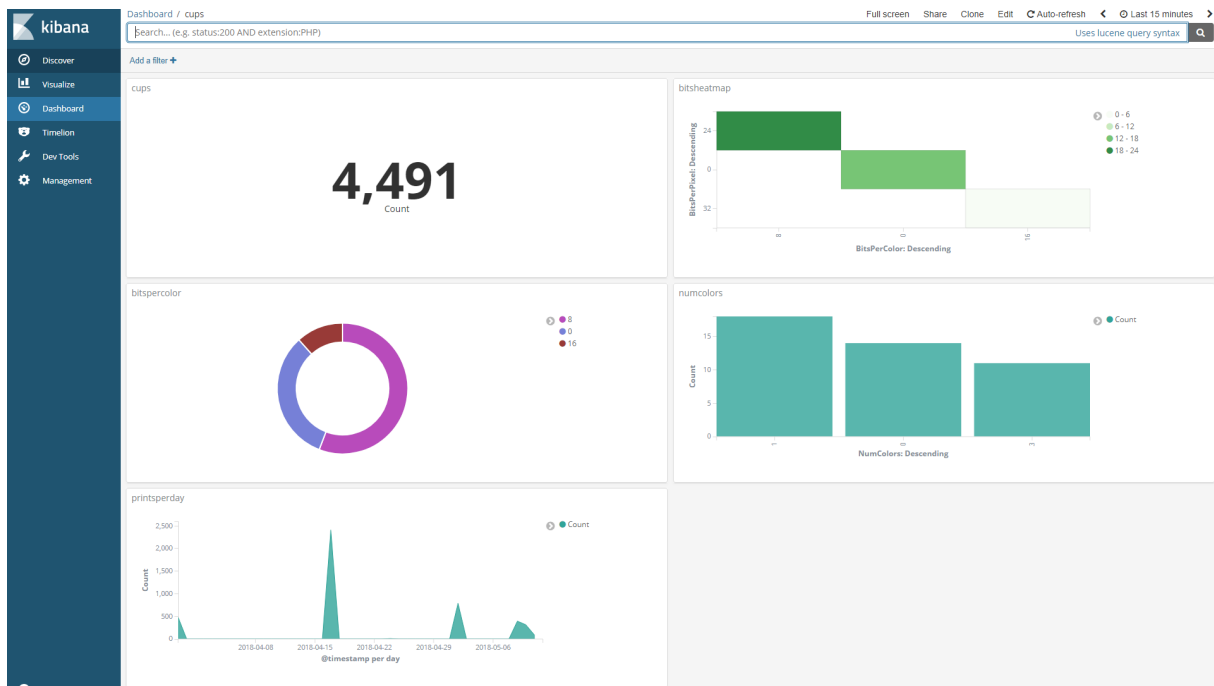


Figure 4.3: dashboard-ul Kibana intr-o fereastră de browser

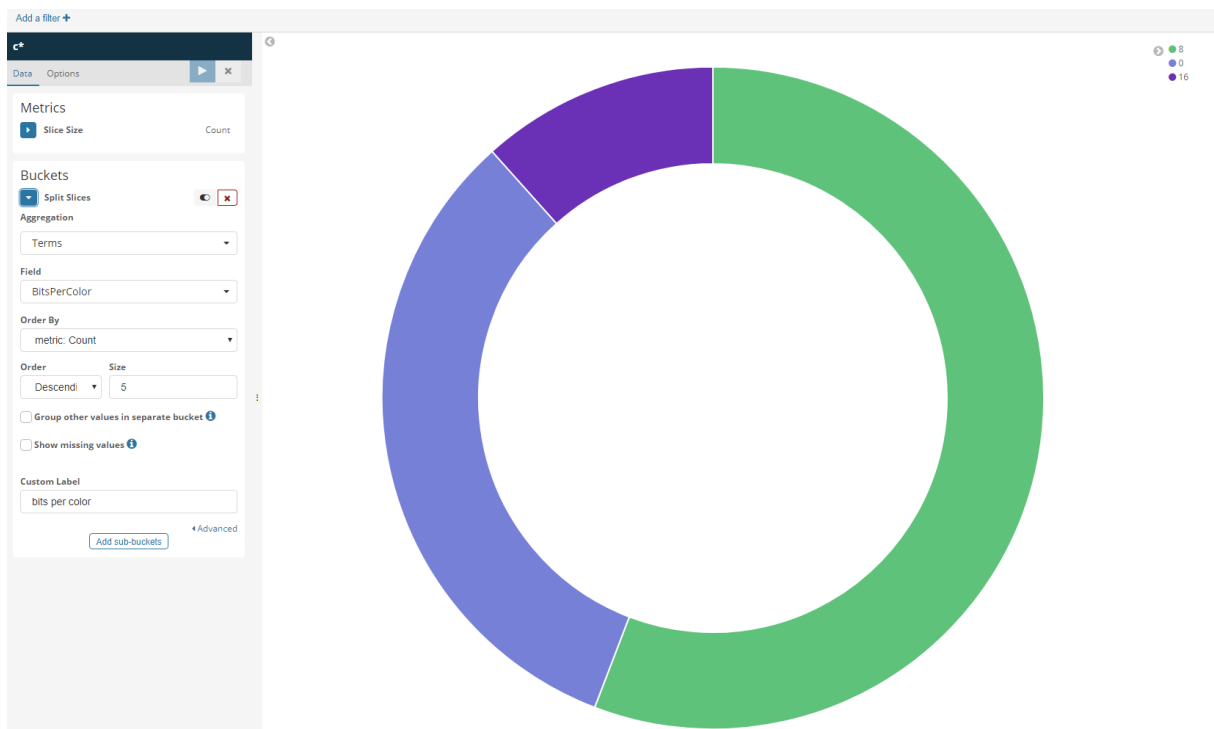


Figure 4.4: dashboard-ul Kibana intr-o fereastră de browser

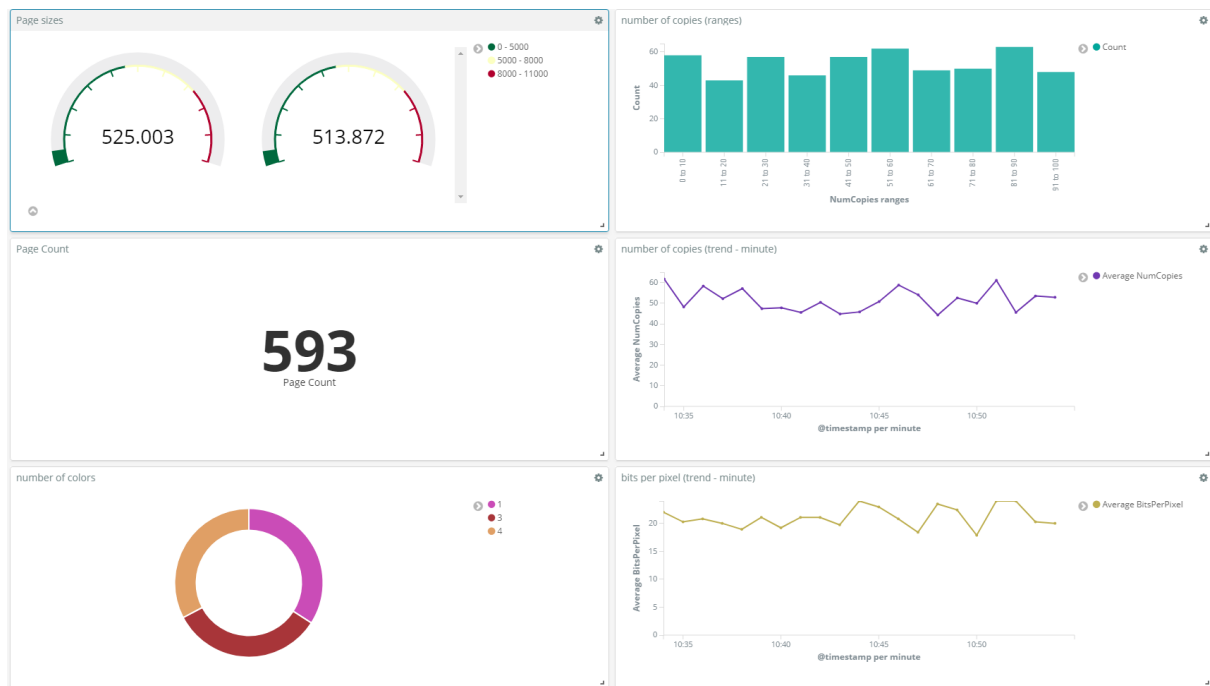


Figure 4.5: exemple de grafice in Kibana; de la dreapta la stanga, de sus in jos: page size, intervale de numere de copii, page count, trend in timp cu numere de copii, pie chart cu numar de culor si trend in timp cu bits per pixel

### 4.3.1 Stubbing

Dat fiind ca datele de tipărire care se pot colecta intr-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 in Python3. Programul parcurge enum-urile C ale CUPS menționate anterior si generează pe baza lor un JSON. Acesta este trimis către Beat, spre a fi prelucrat ca orice tichet venit din CUPS. Stub-ul ajuta la colectarea de date și exemplificarea funcționalității dashboard-ului Kibana. Generatorul de tichete este paremetrizat 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 HWRresolution
2     res = Blank()
3     setattr(res, 'hr1', randomize(const.HW_RES_MAX))
4     setattr(res, 'hr2', randomize(const.HW_RES_MAX))
5     setattr(_json, 'HWRresolution', 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.

### 4.3.2 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 te*t
```

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, -.

### 4.3.3 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 și 30000. Se observă cum interogarea conține un ***match\_all*** și 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 și agregări. Elasticsearch este un produs atât simplu cât și complex. În general se folosește împreună cu un RESTful API.

### 4.3.4 Platforma Scaleway

Pentru deployment s-a ales platforma Scaleway pe considerente practice și economice. Ea oferă performanțe similare platformelor mainstream la o fracțiune din preț. Scaleway folosește mașini 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 oferă root access complet pe un server dedicat. Performanțele sunt constante și 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 în Amsterdam și Paris. Latența din Timisoara este de sub 40 ms. Administrarea se face printr-o platforma web.

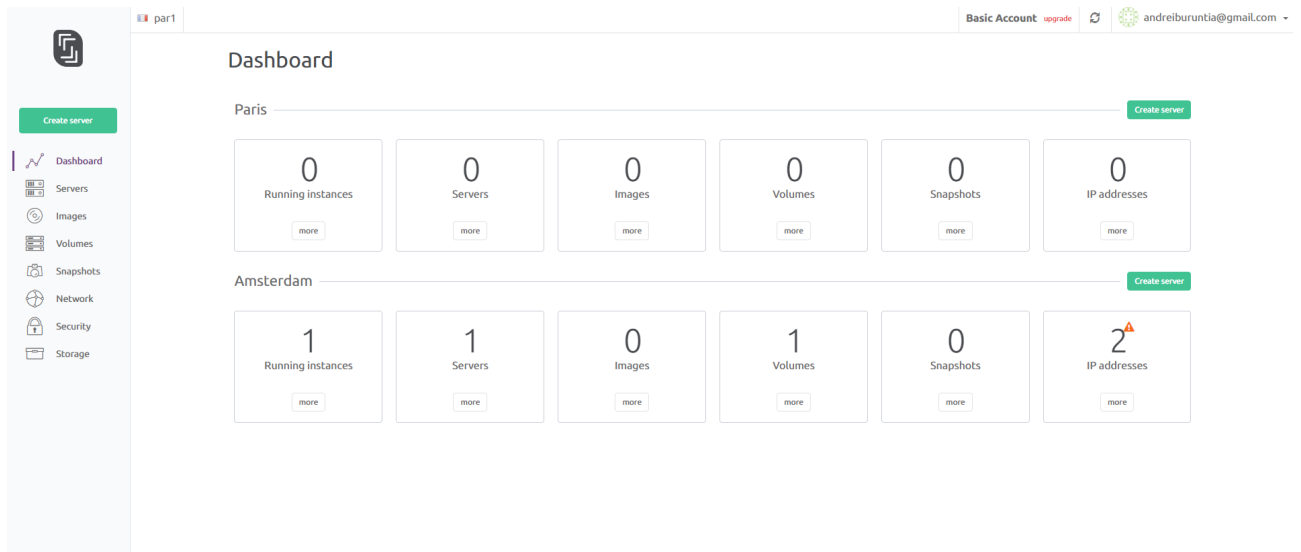


Figure 4.6: Dashboard-ul Scaleway

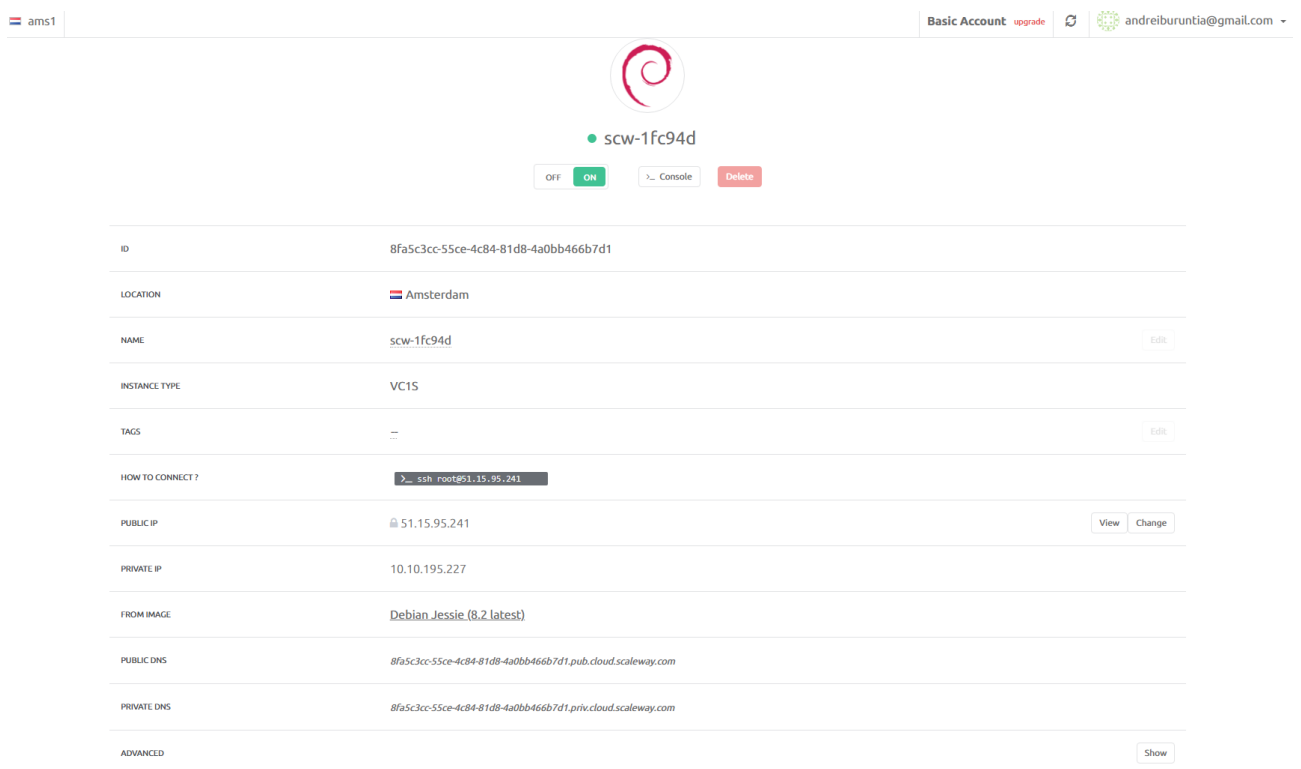


Figure 4.7: Dashboard-ul Scaleway - administrarea unei instanțe de server

# Rezultate

S-a 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 codat 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.

# Concluzii

- se face evidenta utilitatea folosirii tehnologiilor open-source și clădirea peste sistemelor 'tried and tested'
- se remarcă avantajele limbajului Go atunci când se dorește folosirea capabilităților 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
- este de menționat cât de importantă a fost abilitatea CUPS de a comunica și rezolva o multitudine de imprimante, fără ca utilizatorul să facă vreun efort
- se remarcă versatilitatea și fezabilitatea modelului client-server, folosit aici în două instanțe cu constraint-uri și parametri 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
- se remarcă 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 fiecarui din punctele anterior menționate

- contribuții de altă natură includ studiul problemei, propunerea și selecția unor soluții fiabile, 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

3.1	Placa de dezvoltare Raspberry Pi 3 . . . . .	9
3.2	Workflow-ul normal . . . . .	10
4.1	Arhitectura sistemului . . . . .	11
4.2	comunicarea CUPS - Beat - Elasticsearch - Kibana - Utilizator . . . . .	19
4.3	dashboard-ul Kibana intr-o fereastră de browser . . . . .	20
4.4	dashboard-ul Kibana intr-o fereastră de browser . . . . .	20
4.5	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 . . .	21
4.6	Dashboard-ul Scaleway . . . . .	26
4.7	Dashboard-ul Scaleway - administrarea unei instanțe de server . . . . .	26



# Listings

4.1	exemplu metoda Run - golang . . . . .	15
4.2	exemplu metoda Stop - golang . . . . .	15
4.3	deschidere named pipe - golang . . . . .	15
4.4	watch pentru named pipe - golang . . . . .	16
4.5	apel readPipe in go rutina separata - golang . . . . .	16
4.6	instantiere canal - golang . . . . .	16
4.7	exemplu enum CUPS - C . . . . .	16
4.8	instantiere dictionar - golang . . . . .	16
4.9	structuri de date pentru generator/procesor date - golang . . . . .	17
4.10	functia pentru procesarea mesajelor - golang . . . . .	18
4.11	cazul general dupa care se populeaza tichetul - Python . . . . .	21
4.12	cazurile speciale, tratate individual - Python . . . . .	22
4.13	exemplu de output pentru programul de mai sus - JSON . . . . .	22
4.14	exemplu de căutare după câmpuri - Lucene query . . . . .	23
4.15	căutare wildcard cu un singur caracter - Lucene query . . . . .	24
4.16	căutare wildcardmulti-caracter - Lucene query . . . . .	24
4.17	căutare fuzzy - Lucene query . . . . .	24
4.18	căutari pe intervale - Lucene query . . . . .	24
4.19	elevarea relevanței - Lucene query . . . . .	24
4.20	exemplu interogare Elasticsearch - JSON . . . . .	25

# 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 .

