

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.

# Acknowledgements

# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introducere</b>                                   | <b>4</b>  |
| 1.1      | Context și motivație . . . . .                       | 4         |
| 1.2      | Problema . . . . .                                   | 4         |
| <b>2</b> | <b>Fundamentare teoretică</b>                        | <b>6</b>  |
| <b>3</b> | <b>Specificațiile proiectului</b>                    | <b>7</b>  |
| 3.1      | Placa de dezvoltare gazdă - Raspberry Pi 3 . . . . . | 7         |
| 3.2      | Workflow-ul normal pentru tipărire . . . . .         | 8         |
| 3.3      | Colectarea și stocarea datelor . . . . .             | 8         |
| <b>4</b> | <b>Design și implementare</b>                        | <b>9</b>  |
| 4.1      | Arhitectură high-level . . . . .                     | 9         |
| 4.2      | Componente software . . . . .                        | 10        |
| 4.2.1    | Sistemul de operare gazdă - BuruX . . . . .          | 10        |
| 4.2.2    | CUPS . . . . .                                       | 11        |
| 4.2.3    | Elastic Stack . . . . .                              | 12        |
| 4.3      | Comunicarea între module . . . . .                   | 17        |
| <b>5</b> | <b>Rezultate</b>                                     | <b>18</b> |
| <b>6</b> | <b>Concluzii</b>                                     | <b>19</b> |
| <b>7</b> | <b>Glosar de termeni</b>                             | <b>20</b> |
| <b>8</b> | <b>Referințe</b>                                     | <b>21</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 preîntâmpina anumite defectiuni. 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.

si/sau ompaniei Océ. In ace4asta interfata grafica se vor afisa graficele necesare si se vor putea crea panouri de control dedicate. Solutia trebuie sa fie extensibila, sa scaleze pe mai multe tipuri de imprimanta. In cazul in care anumite informatii nu pot fi obtinute in mod direct din imprimanta atunci acestea trebuie sa poata fi generate prin analiza formatelor intermediare de tip rastu pe care rasterizatoarele le trimit catre imprimante. Implementarea solutiei trebuie sa respecte urmatoarele cerinte:

1. Sa nu expuna nicio informatie proprietara Océ
2. Se fie facuta intr-un limbaj de nivel inalt
3. Sa permita extensii cu costuri minime pentru modelele viitoare de imprimante
4. Sa implice modificari minime la nivelul infrastructurii clentilor
5. Sa anonimizeze informatiile confidentiale

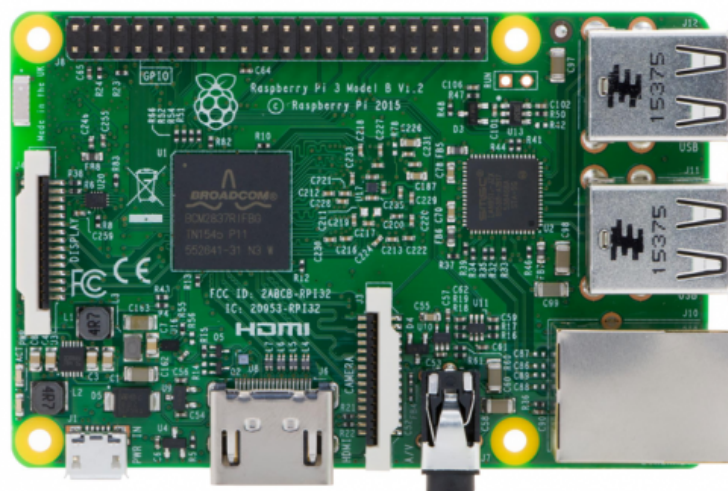
# Fundamentare teoretică

# Specificațiile proiectului

## 3.1 Placa de dezvoltare gazdă - Raspberry Pi 3

Probabil cea mai populara placa de dezvoltare cu microprocesor, Raspberry Pi 3 are urmatoarele specificatii:

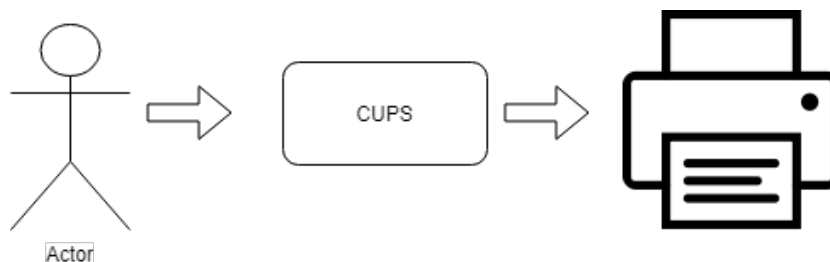
- procesor Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz
- 1GB LPDDR2 SDRAM
- placa de retea 2.4GHz and 5GHz IEEE 802.11 b/g/n/ac
- Gigabit Ethernet over USB 2.0
- extended 40 pin GPIO header
- full-size HDMI
- 4 porturi USB 2.0
- slot de card micro SD
- 5V @ 2.5A intrare alimentare





## 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

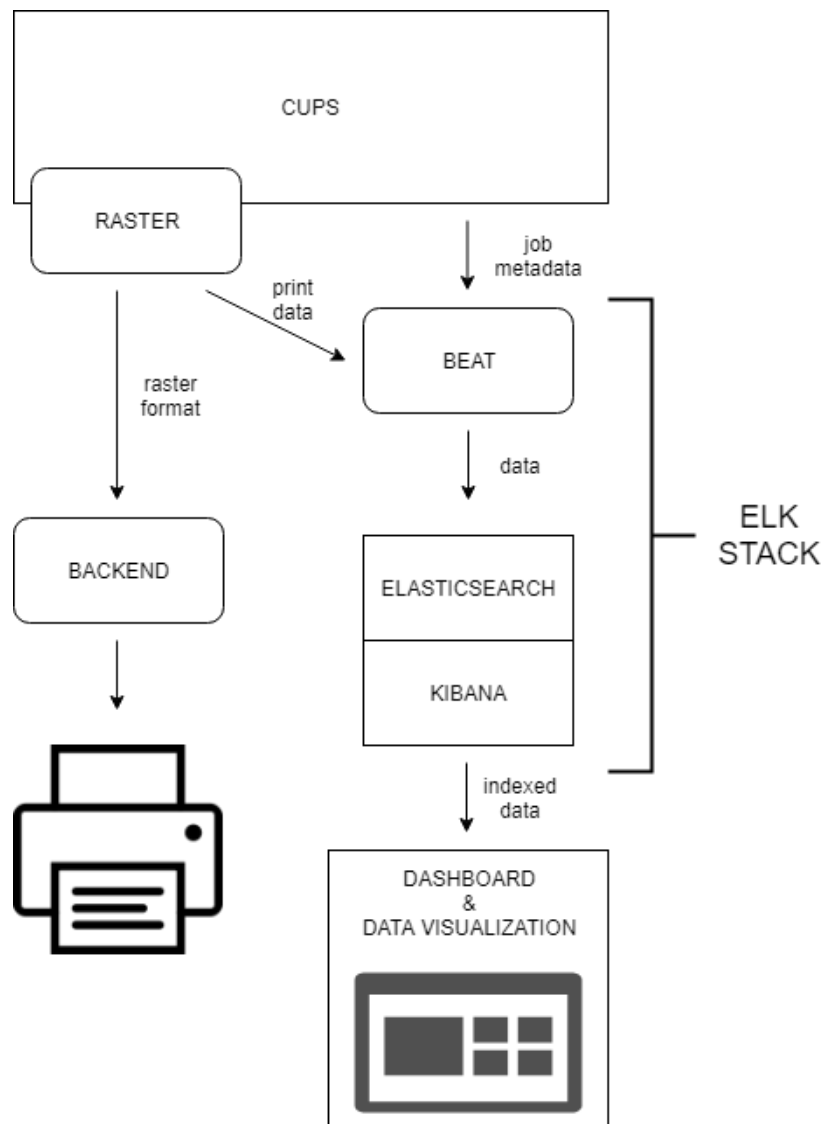
## 3.3 Colectarea și stocarea datelor

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 parasesc niciodată siguranța nucleului sistemului de operare. După 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 a trebuit 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



## 4.2 Componente software

### 4.2.1 Sistemul de operare gazdă - BuruX

Folosind CUPS ca backbone pentru sistem, s-a facut evidenta nevoie folosirii unui sistem de operare Unix-like care sa il gazduiasca. Pe considerente de familiaritate s-a ales folosirea unui sistem de operare bazat pe nucleul Linux. Nucleul Linux este un nucleu monolitic, adica lucreaza in intregime separat de restul sistemului si in mod administrator. Driverile se pot incarca in memoria de lucru la utilizare, de unde sunt sterse ulterior. Nucleul ofer facilitati 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

Distributia a fost compilata pentru ahritectura ARM, specifica procesorului Broadcom BCM2837 cu 4 nuclee ARM Cortex-A53, spre a fi instalata pe o placa Raspberry Pi 3. Procesorul si memoria de 1GB ale placii sunt mai mult decat suficiente pentru a rule sistemul de operare si componentele mentionate mai sus. Versiunea de kernel folosita este 4.14, cu niste patch-uri suplimentare care sa permita folosirea modulelor Bluetooth si Wi-Fi ale placii. In testele efectuate, sistemul de operare este perfect stabil si consuma foarte putine resurse comparativ cu o distributie de Linux full blown. Compilarea completa a sistemului de operare se face, pe o masina relativ veche, in sub 25 minute. Nucleul linux este principala componenta a sistemului de operare GNU/Linux. Pentru use-case-ul de fata, am ales sa nu instalez un desktop environment sau interfata grafica pe sistem, pe considerente de reducere a resurselor consumate. Distributia de Linux folosita a fost compilata special pentru acest uz, drept urmare majoritatea functiilor au fost eliminate, ramanand doar cu cele strict esentiale pentru functionarea CUPS:

- 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 si aplicare de patch-uri a fost folosit Buildroot - o unealta care automatizeaza procesul de building al unui mediu Linux complet si bootabil, folosind cross-compilation pentru a putea servi multiple platforme. Buildroot isi poate build-ui propriul toolchain, crea un root file system, compila o imagine de Linux kernel si genera un boot loader pentru sistemul dorit. Acesta functioneaza pe baza unor fisiere de configuratii, numite defconfigs, care contin informatii 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 si musl. Intern, Buildroot foloseste Kconfig pentru sistemul de configuratie, acesta oferind facilitati precum o interfata cu meniu, ocuparea de dependente, optiunea de 'Help' contextual. Intregul proces de build se construiesc in jurul pachetelor descarcate automate in functie de defconfig. Aceste pachete pot include aplicatii, utilitati, biblioteci, etc. Rezultatul final este un root file system care poate fi copiat pe un mediu bootabil si folosit as-is. Buildroot face tot procesul de compilare, build si deployment usor de inteles si accesibil oricui.

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

## 4.2.2 CUPS

### Common UNIX Printing System

Aparut in 1999, CUPS este un sistem modular de printing open-source pentru sistemele de operare Unix-like. Acesta permite unui calculator sa se comporte ca un print server, devenind un host care accepta job-uri de la clienti, le proceseaza si le trimite catre imprimante. CUPS este format dintr-un spooler, un scheduler, un sistem de filtre care transforma datele dintr-un format in altul, si un sistem de backend-uri care realizeaza comunicarea cu imprimantele.

Workflow-ul standard CUPS este urmatorul: un job ajunge in scheduler, acesta il trimite catre unul sau mai multe filtre spre a fi convertit in alt format, apoi datele ajung intr-un backend de unde sunt trimise spre printer. Sistemul foloseste extensiv PostScript si rasterizarea datelor ca formate intelese de imprimante.

### Scheduler-ul CUPS

Scheduler-ul CUPS implementeaza Internet Printing Protocol si ofera o interfata web-based pentru managementul imprimantelor, job-urilor, configuratii server si documentatie. Pentru accesarea functionalitatilor de management si configuratie, un utilizator trebuie sa se autentifice. O alta parte foarte importanta a scheduler-ului este modulul de logging. Acesta inregistreaza 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 in sistem.

### Sistemul de filtre CUPS

Sistemul de filtre al CUPS poate procesa o varietate de formate. Acesta converteste datele unui print-job intr-un limbaj/format final printr-un sistem de filtre, folosind MIME types pentru identificarea formatelor. Procesul de filtrare incepe prin determinarea tipului de date de intrare, folosind MIME databases. Printre filtrele default ale CUPS se numara: raster to PCL, raster to ESC/P, raster to Dymo, raster to ZPL. Initial, am incercat modificare scheduler-ului, mai exact a modulului de logging, dar informatiile pe care le puteam obtine din acesta erau insuficiente pentru cerintele proiectului, in modulul de logging fiind expuse numai date despre utilizatori, job-uri sau status. Considerand nevoia de a obtine date despre pixeli, culoare, media, pagina, etc. s-a facut necesara accesarea informatiilor disponibile in sistem de filtre al CUPS, mai exact in unitatea de raster.

Dupa ce am identificat informatiile de care am nevoie, le-am trimis catre exterior printr-un HTTP POST, apoi printr-un named pipe. Folosind un apel de sistem catre comanda `cURL` pentru a trimite cererea HTTP catre Beat, s-ar fi creat un numar de procese greu de controlat si care ar fi generat overhead pe care il puteam evita. O a doua alternativa ar fi fost folosirea unei biblioteci, de exemplu `libcurl`, pentru a inlocui apelurile de sistem, dar aceasta alternativa ar fi introdus o dependenta suplimentara in CUPS, ceea ce ar fi facut codul mai putin portabil si ar fi incalcat una din cerintele initiale ale proiectului. In final, am folosit un named pipe pentru transmiterea datelor, acesta fiind usor de accesat si generand overhead minim, iar in plus, datele nu sunt expuse in retea, aceasta interfata rezidand in kernelul sistemului de operare.

## Backend-urile CUPS

Backend-urile CUPS sunt folosite pentru a trimite datele catre imprimante. CUPS pune la dispozitie 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 stiva de aplicatii care permite extragerea, procesarea si vizualizarea datelor. In majoritatea cazurilor, acesta este gasit sub numele de ELK Stack (Elasticsearch - Logstash - Kibana). Pentru aceasta teza nu a fost folosit Logstash, fiind inlocuit cu un Elastic Beater ca unealta de colectare a datelor. Un Beat implementeaza interfata Beater si aduna date spre a le trimite catre Kibana sau, in cazul de fata, Elasticsearch.

#### Beat

Beat-urile au fost adaugate recent la stack-ul ELK. Un Beat are responsabilitatea sa colecteze date si sa le trimita spre procesare sau vizualizare. Acesta trebuie sa implementeze in Golang o interfata numita Beater interface, care contine metode de Start, Run si Stop. Aceasta lucrare foloseste un Beat pentru a primi date de la raster-ul, raster care face parte din sistemul de filtre mentionat anterior. Beat-ul primeste raster data, face operatii de procesare si sanitization minimale, apoi trimite datele catre Elasticsearch. In implementarile conventionale de Beat, datele sunt transmise la intervale periodice de timp catre Elasticsearch sau Kibana. Implementarea Beat-ului din aceasta lucrare trimite date catre Elasticsearch atunci cand primeste informatii noi de la raster. `cBeat` (Beater-ul implementat pentru aceasta lucrare) urmatoarele componente principale:

- Metodele interfetei Beater
  - metoda **Run**
    - \* contine bucla principala a aplicatiei
    - \* in interiorul buclei se transmit 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 cand Beat-ul e semnalat sa se opreasca
- \* 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

### • Receptionarea datelor din CUPS

- citirea din named pipe a evenimentelor se face prin deschiderea FIFO-ului ca orice alt fisier 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", pipePath, err)
6 } else if err != nil {
7     log.Fatalf("Error while opening named pipe '%s': %s",
      pipePath, err)
8 }
9 defer p.Close()

```

Listing 4.3: deschidere named pipe - golang

- FIFO-ului i s-a adaugat un mecanism de watch care ne notifica atunci cand s-a efectuat orice scriere in 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

- functia care supravegheaza pipe-ul a fost lansata intr-o go rutina in 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 citeste din pipe si 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 in CUPS se urmareste generarea unor structuri de date echivalente in 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 eficienta a datelor, se pastreaza encodarea diferitor optiuni de tiparire in numere intregi
- generarea de cod Go din fisierele header care contin structurile CUPS presupune parcurgerea lor
- rezultatul final trebuie sa fie un fisier .go compilabil care contine structurile echivalente
- s-a folosit un algoritm similar cu un automate care parcurge fisierul
- automatul salveaza fiecare intrare intr-un dictionar de dictionare de siruri de caractere

```

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

```

Listing 4.8: instantiere dictionar - golang

- fiecare intrare corespunde unui enum, cu dictionarul corespunzator
- intrarile din dictionarul corespunzator reprezinta prechile de intregi - sir de caractere
- intregul modul de generare de cod a fost scris in limbajul Go

```

1 type HWResolution_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     HWResolution HWResolution_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     HWResolution HWResolution_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 populeaza structurile mentionate la punctul anterior pe baza header-ului CUPS
- atunci cand se receptioneaza un eveniment, acesta ajunge in Beat sub forma de string
- evenimentul este procesat pe baza dictionarelor, intregii fiind inlocuiti cu stringuri
- dupa prelucrare, evenimentul este serializat ca JSON si este trimis catre 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: functia pentru procesarea mesajelor - golang

## Elasticsearch

Elasticsearch este un motor de cautare bazat pe Apache Lucene. Ofer clienti disponibili in majoritatea tehnologiilor si este cel mai popular motor de cautare enterprise. Beneficiile Elasticsearch includ: cautare foarte scalabila, aproape real-time, suportul mai multor clienti, platforma distribuita. Este considerat de multi backbone-ul stack-ului ELK. Informatiile extrase de Beat ajung in Elasticsearch, iar acesta cauta tipare si indexeaza datele spre a le trimite catre Kibana.

## **Kibana**

Kibana este un plugin open source pentru vizualizare de data, care pune la dispozitie capabilitati de vizualizare ale elementelor indexate intr-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 si le foloseste la generarea de grafice si vizualizari relevante pentru utilizator.

Atat Elasticsearch, cat si Kibana sunt hostate pe o masina (Debian) in cloud pe platforma Scaleway. Masina are o configuratie minimala, stack-ul fiind unul robust si eficient nu are nevoie de multe resurse. Administrarea masinii se face strict prin SSH, iar interfata web Kibana este expusa spre exterior. Prin intermediul aplicatiei web se face si configurarea graficelor si dashboard-ului Kibana.

## **4.3 Comunicarea între module**

# Rezultate

# Concluzii

# Glosar de termeni

- CUPS - Common Unix Printing System
- PDF -
- GNU -
- Linux -
- OSX -
- BSD -
- kernel -
- Avahi -
- golang -
- clang -
- Elastic stack - ELK stack -
- Kibana -
- Elasticsearch -

## Referințe

