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 sa își creeze propria interfață, dupa 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 sa nu restricționeze in 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 si îmbunatațiri aduse acestora.

Ackowledgements

Contents

Introducere

1.1 Context și motivație

Océ, compania in care lucrez, activeaza in domeniul de printing, fiind subsidiar Canon. In companie se pune problema analizei si colectarii datelor din procesul de imprimare. La propunerea companiei, am incercat sa abordez aceasta problema. Mai jos este prezentata mai pe larg problema, iar lucrarea urmareste propunerea unei idei de implementare a unu sistem care sa o rezolve. In aceasta documentatie se va incerca detalierea asupra anumitor aspecte ale lucrarii: dificultatile intalnite, probleme de comunicare, constraint-uri de orice fel, etc.

1.2 Problema

Analiza si colectare de statistici despre procesul de imprimare

Pentru a putea optimiza costurile si procedeele de tiparire in cazul imprimamtelor de format mare se doreste analiza si colectarea de statistici referitoare la:

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

Aceste informatii trebuiesc agregate si afisate sub forma unor grafice care pot fi folosite de catre utilizatori cat si de catre compania Océ. Utilizatorii vor avea informatii legate de costuri si timpi de executie. Compania Océ poate folosi aceste informatii in mod anonimizat pentru a culege informatii legate de calitatea tiparirii sia a procesului cat si pentru an analiza in mod proactiv procesul de tiparire si a colecta informatii despre modul in care imprimantele se comporta in timp si a preintamppina anumite defectiuni. Deoarece baza de imprimante este relative mare si nu toate imprimantele ofera in mod nativ informatii despre consumabilele si cantitatea de cerneala folosita se doreste ca aceste informatii sa fie colectate in mod transparent si de la imprimantele mai putin evaluate. Pentru aceasta ar trebui ca aceste informatii sa fie colectate intr-un mod cat mai transparent, fara a afecta functionarea normal a imprimantei si fara a afecta firmware-ul deja instalat. Informatiile astfel colectate vor fi afisate intr-o interfata web accesibila utilizatorilor final

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 tipuiri 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 rastru 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

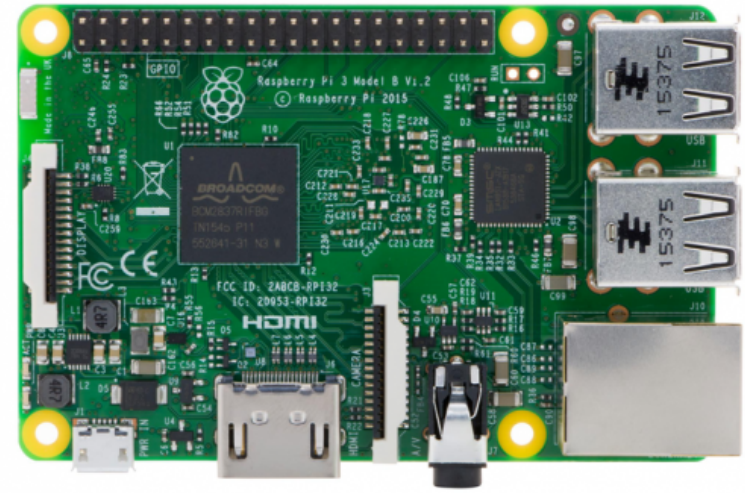
Fundamentare teoretică

Specificațiile proiectului

3.1 Placa de dezvlotare gazdă - Raspberry Pi 3

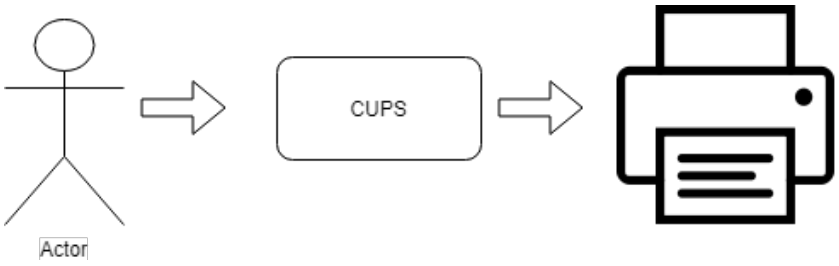
Probabil cea mai populara placa de dezvoltare cu microprocesor, Raspberry Pi 3 are urmatoarele specificatii:

- 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 si simplu a unui job de print catre CUPS, iar acesta se va ocupa de restul. Un aspect important este abstractizarea imprimantelor in spatele 'cozilor' CUPS si se urmareste ca acest aspect sa ramana neschimbat. CUPS asigneaza fiecarei imprimante una sau mai multe 'cozi' (queues), iar acestea apar ca imprimante pe retea pentru sistemele Unix (OSX, GNU/Linux, BSD, etc.). Proiectul urmareste extragerea cat mai multor informatii din procesul normal de printare CUPS, dar si mentinerea modificarilor asupra CUPS la un nivel minim. O cerinta importanta este ca workflow-ul normal de printare sa ramana neschimbat, iar modificarile aduse pentru colectarea de date sa impacteze cat mai putin platforma.



Modificarile aduse CUPS trebuie sa se respecte urmatoarele cerinte:

- sa nu adauge overhead sistemului
- sa nu expuna informatie proprietara Océ
- sa nu modifice functionarea modulelor CUPS
- sa expuna intr-o maniera usor de interceptat informatiile relevante
- sa respecte licenta Apache v2.0 aferenta CUPS

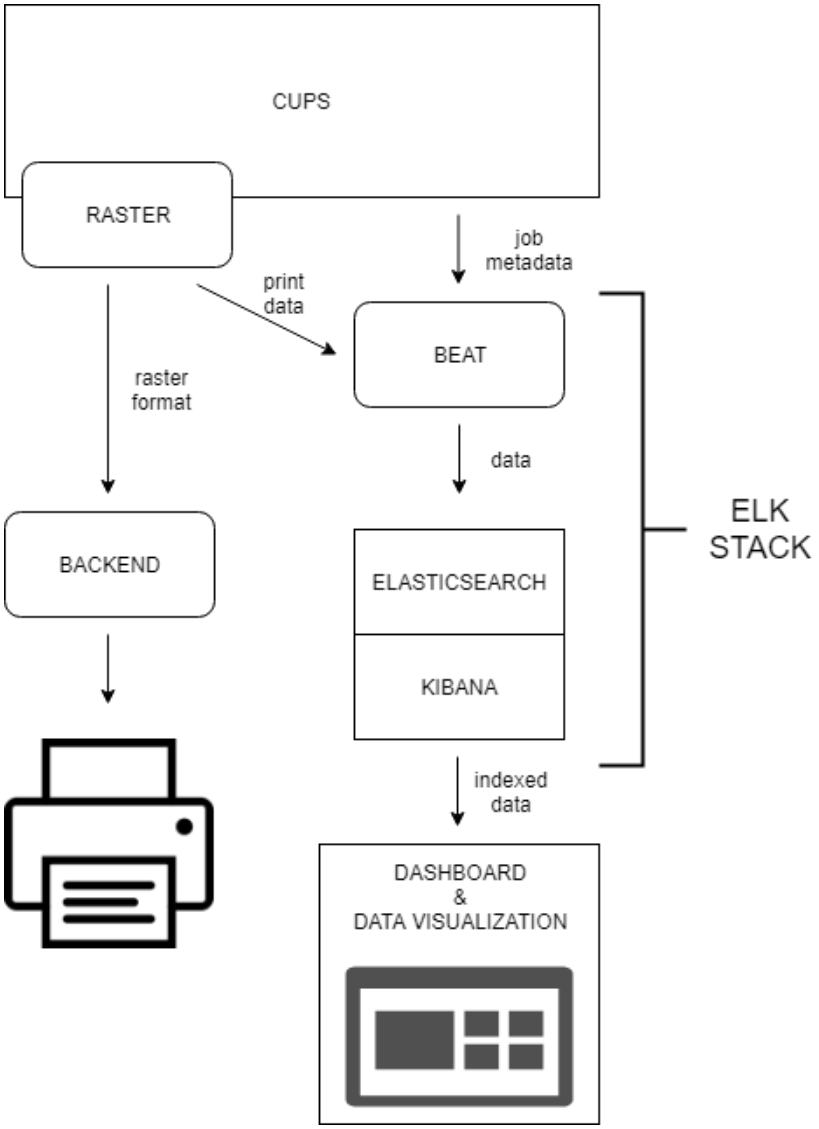
3.3 Colectarea și stocarea datelor

Modificarile aduse CUPS urmaresc exclusiv colectarea datelor de imprimare si trimiterea lor catre agregator. Colectarea datelor se face intr-o maniera cat mai putin invaziva, conform cerintelor anterior mentionate. Se urmareste extragerea datelor spre a fi procesate, nu stocarea lor propriu zisa. Datele sunt transmise printr-un named pipe, prin urmare acestea nu parasesc niciodata siguranta nucleului sistemului de operare. Dupa ce datele sunt receptionate si procesate de catre agregator, acestea nu sunt salvate pe disc sau in alt mediu de stocare persistenta. Singurul loc in care datele persista este in Elasticsearch, dar este importanta mentiunea ca datele ajunse in Elasticsearch sunt deja procesate, iar in procesul de procesare pot fi eliminate orice informatii nedorite.

Un alt criteriu important pe care proiectul a trebui sa il respecte este respectarea confidentialitatii datelor. In niciun punct sistemul nu colecteaza date cu caracter personal, nici macar numele utilizatorului, jobului sau masina 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. Driverele 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 construieste 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 schedeuler-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 utlizatori, 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 transimse 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 evenimentel sunt trimise la intervale periodice de timp

```
func (bt *Countbeat) Run(b *beat.Beat) error {
        logp.Info("countbeat is running! Hit CTRL-C to stop it.")

        bt.client = b.Publisher.Connect()
        ticker := time.NewTicker(bt.config.Period)
        counter := 1
        for {
                select {
                case <-bt.done:
                        return nil
                case <-ticker.C:
                }

                event := common.MapStr{
                        "@timestamp": common.Time(time.Now()),
                        "type":       b.Name,
                        "counter":    counter,
                }
                bt.client.PublishEvent(event)
                logp.Info("Event sent")
                counter++
        }
}
```

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

```
func (bt *Countbeat) Stop() {
        bt.client.Close()
        close(bt.done)
}
```

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

```
pipePath = "/tmp/cupsbeat"
 if p, err = os.Open(pipePath); os.IsNotExist(err) {
     log.Fatalf("Named pipe '\%s' does not exist", pipePath)
} else if os.IsPermission(err) {
    log.Fatalf("Insufficient permissions to read named pipe '\%s': \%s", pipePath, err)
} else if err != nil {
    log.Fatalf("Error while opening named pipe '\%s': \%s", pipePath, err)
}
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

```
c := make(chan notify.EventInfo, MAX_CONCURRENT_WRITERS)
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

```
go func(){
  readPipe()
}()
```

Listing 4.5: apel readPipe in go routina separata - golang

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

```
var messageQueue = make(chan Message_s_t)
```

Listing 4.6: instantiere canal - golang

- Generarea structurilor necesare prelucrarii datelor

  – pe baza enum-urilor definite in CUPS se urmareste generarea unor structuri de date echivalente in limbajul Go

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

```
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

```
type HWResolution_t struct {
  Hr1 int
  Hr2 int
}

type ImaginBoundingBox_t struct {
  Ibb1 int
  Ibb2 int
  Ibb3 int
  Ibb4 int
}

type PageSize_t struct {
  Ps1 int
  Ps2 int
}

type Message_t struct{
  CutMedia int
  Duplex int
  HWResolution HWResolution_t
  ImagingBoundingBox ImaginBoundingBox_t
  InsertSheet int
  Orientation int
  NumCopies int
  PageSize PageSize_t
  Tumble int
  CupsWidth int
  CupsHeight int
  CupsBitsPerColor int
  CupsBitsPerPixel int
  CupsColorOrder int
  CupsColorSpace int
  CupsNumColors int
}

type Message_s_t struct{
  CutMedia string
  Duplex int
  HWResolution HWResolution_t
  ImagingBoundingBox ImaginBoundingBox_t
  InsertSheet int
  Orientation string
  NumCopies int
  PageSize PageSize_t
  Tumble int
  CupsWidth int
  CupsHeight int
  CupsBitsPerColor int
  CupsBitsPerPixel int
  CupsColorOrder string
  CupsColorSpace string
  CupsNumColors int
}
```

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, avesta 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_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: 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 indexat de la Elastisearch 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. Pe masina host este instalat nginx care redirecteaza orice acces HTTP catre portul Kibana (9300).

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 -
- nginx -

Referințe