Monitorizarea sistemului și a log-urilor pentru Data Hub Software (DHuS)

Sânziana-Aurelia VOICU[1], Florin POP[1,2], Cătălin NEGRU[1], Adrian STOICA[3]

[1] Universitatea Politehnica București

[2] Institutul Național de Cercetare-Dezvoltare în Informatică – ICI București

[3] Terrasigna

sanziana.voicu@stud.acs.upb.ro, florin.pop@cs.pub.ro/florin.pop@ici.ro, catalin.negru@cs.pub.ro, adrian.stoica@terrasigna.com

Rezumat: Multe proiecte în domeniul tehnologiei ajung să fie un eșec, unul dintre motive fiind lipsa de monitorizare. Este pe de-o parte lipsa monitorizării sistemului, care duce la vulnerabilități la nivel hardware sau kernel, iar pe de altă parte lipsa monitorizării serviciilor și aplicațiilor componente, care duce la utilizatori nemulțumiți din punct de vedere al funcționalității. Soluția propune monitorizarea sistemului DHuS, utilizat pentru descărcare de date satelitare, pentru că ne interesează performanța unui sistem ce gestionează date atât de mari și pentru că este folositor de urmărit interesul indivizilor în privința serviciilor oferite. Soluția cuprinde două componente, una de monitorizare sistem și una de monitorizare bazata pe log-uri. Ambele au ca rezultat vizualizarea informațiilor strânse într-o interfață grafică, în ideea că utilizatorul lor poate îmbunătăți părți ale proiectului la următoarea iterație de implementare.

Cuvinte cheie: DHuS, monitorizare, analiza log-urilor, imagini satelitare.

System and log monitoring for Data Hub Software (DHuS)

Abstract: Many technology projects fail to be successful, one of the reasons being the lack of monitoring. It is on one hand the lack of system monitoring, which may cause hardware and kernel vulnerabilities, and on the other hand, the lack of applications and services monitoring, which may cause unsatisfied users with the outcome of the project. We propose the monitoring of DHuS, used for downloading sentinel data, which is important once because the system handles large sets of data and once because it is useful to see the user's interest in the provided services. The solution has two components, one of system monitoring and the other of log monitoring. Both results in sending data to graphical interfaces, hoping that the users of those two components may be able to improve the project based on the information provided.

Keywords: DHuS, monitoring, log analysis, satellite images.

1. Introducere

Monitorizarea, indiferent de domeniul în care se aplică aceasta, constă în observarea și înregistrarea în mod regulat al aspectelor definitorii pentru un sistem. De exemplu, paradigma Internet of Things (IoT) a devenit element de transformare revoluţionară, cantitativă şi calitativă a cunoaşterii şi interacţiunii noastre cu mediul, implică dezvoltarea de soluții și aplicații diverse (oraşe inteligente, case inteligente, monitorizare digitală a sănătăţii, a unor procese industriale sau a poluării mediului etc.) care trebuie monitorizate si analizate (Neagu, 2017).

În cadrul unui proiect software, monitorizarea sistemului este un aspect primordial, întrucât poate detecta și preveni potențiale vulnerabilități, atât în implementarea software, cât și la nivelul componentelor hardware (Fioriti, 2017). De exemplu, trafic inexistent pe o componentă a sistemului poate însemna că nu există conectivitate în rețea către acel punct, puțină memorie RAM liberă semnalează faptul că este nevoie de mai multă, un procent mare de utilizare procesor semnifică nevoia distribuirii muncii pe mai multe mașini sau eficientizarea algoritmului (Wu, 2013).

Un bun indicator pentru performanțele unui sistem sunt fișierele de log provenite de la servere, rețea, dispozitive de securitate și nu în ultimul rând, servicii și aplicații (Alexandru, 2017). Log-urile unei aplicații dedicate să facă ceva specific aduc în plus informații despre domeniul de activitate al proiectului. Aceste log-uri nu sunt atât de relevante pentru dezvoltatori, cât sunt pentru

https://doi.org/10.33436/v30i1y202006

cei ce au inițiat dezvoltarea proiectului și acum urmăresc progresul lui. De asemenea din fișierele de log se pot extrage parametrii specifici pentru contractul de furnizare a serviciilor (Iordache, 2017).

Un sistem de monitorizare pe bază de log-uri poate da informații utile atât dezvoltatorilor sistemului, cât și Agenției Spațiale Europene. În cazul DHuS: numărul de descărcări grupate pe țări, numărul de utilizatori activi în funcție de țara de proveniență sau volumul total de descărcări de la începutul operațiunii.

Soluția conține două componente principale: un sistem de monitorizare a sistemului și un sistem de monitorizare bazat pe log-uri (Talaș, 2017). Monitorizarea sistemului constă în a instala pe fiecare mașină componenta DHuS o aplicație care să investigheze metricile locale despre hardware și despre kernel. La intervale de timp stabilite, un server trebuie să preia și să stocheze metricile respective cu scopul de a fi trimise mai departe către o aplicație care să le expună într-o interfață grafică. În interfața grafică vor fi structurate sub formă de grafice și diagrame intuitive. Componenta de monitorizare a aplicației DHuS bazată pe log-uri aduce pe de-o parte o interfață cu grafice relevante pentru numărul de descărcări sau numărul de utilizatori activi în ultima perioadă, în timp real, precum și un JSON conform specificațiilor din documentul "*Sentinels Rolling Archive User Access, Operations, Maintenance and Evolutions*", la sfârșitul fiecărei zile (Tona, 2018).

Prima secțiune prezintă în introducere motivația și obiectivele implementării unui sistem de monitorizare pentru DHuS. Secțiunea a II-a prezintă componentele software-lui DHuS, precum și structura datelor satelitare, înțelegerea ei fiind folosită mai departe în dezvoltarea sistemului de monitorizare bazat pe log-uri. De asemenea, subcapitolul de abordări existente aduce informații despre soluțiile de monitorizare sistem existente. Secțiunea a III-a prezintă tehnologiile folosite pentru fiecare componentă a soluției în parte, precum și interdependența între acestea. Secțiunea a IV-a aduce o descriere amănunțită a procesului de implementare, fișiere de configurare, script-uri și eventuale rezultate din urma dezvoltării. În Secțiunea a V-a se discută despre rezultatele experimentale si despre eficiența folosirii unei abordări în favoarea alteia, precum și diferențe între rulări cu variație de parametri. Secțiunea a VI-a prezintă concluziile și eventuale dezvoltări ulterioare care să îmbunătățească starea actuală a soluției.

2. Lucrări similare

DHuS furnizează o interfață grafică web care permite găsirea și descărcarea de date satelitare în mod interactiv, precum și o interfață de programare a aplicațiilor, care permite accesul la date în cadrul scripturilor și programelor proprii. Acesta poate fi instalat și configurat în trei modalități, iar acestea, în ordinea performanțelor sunt: ca o singură instanță, în mod front-end/back-end și în mod scalabil 2.0. Ca singură instanță, asigură preluarea de date de la surse externe, administrează cererile utilizatorilor și publică produsele către aceștia. În modul front-end / back-end, sunt folosite cel puțin două instanțe DHuS care se sincronizează folosind protocolul OData. Preluarea datelor din surse externe este făcută de una sau mai multe instanțe DHuS numite back-end. Administrarea cererilor utilizatorilor și publicarea produselor sunt făcute de o singură instanță DHuS numită front-end. Modul scalabil 2.0 presupune existența mai multor factori și instanțe de DHuS, fiecare cu un rol specific, după cum urmează:

- **Proxy**: este folosit pentru a redirecționa cererile utilizatorilor către anumite noduri, bazându-se pe un algoritm round-robin de balansare a consumului procesorului, astfel încât cererile, în ordinea apariției lor, sunt trimise către fiecare nod în parte, iar când se ajunge la sfârșitul nodurilor, o ia de la început. Eficiența acestui algoritm constă în faptul că toate serverele (nodurile) din sistem au aceleași capacități de calcul;
- **Baza de date relațională și baza de date ne-relațională**: sunt instalate ca servicii pe mașini diferite de nodurile care conțin servicii DHuS. Sistemele de gestionare a bazelor de date suportate la momentul actual sunt PostgreSQL pentru cea relațională și Solr pentru cea ne-relațională;
- **Orchestrator pentru baza de date ne-relațională**: este instalat ca serviciu pe o mașină diferită de nodurile care conțin servicii DHuS, asemenea bazelor de date și conține fișiere de configurare pentru Solr;

- **Master**: nodul master DHuS se ocupă cu preluarea și sincronizarea produselor/datelor pe celelalte noduri;
- **Noduri**: sunt mașinile către care proxy-ul trimite cererile utilizatorilor. Este important ca ele să fie sincronizate cu nodul master în ceea ce privește produsele.

DHuS gestionează accesul la produse provenite de la sateliți, ca: Sentinel-1, Sentinel-2, Sentinel-3, care se împart după misiune, tip și instrument (vezi Figura 1).

MISSION	TYPE	INSTRUMENT
S1	GRDF	SARC
S1	GRDH	SARC
S1	GRDM	SARC
S1	OCN_	SARC
S1	RAW_	SARC
S1	SLC_	SARC
S2	MSIL1C	Optical
S2	MSIL2A	Optical
S3	OL_1_EFR___	Optical
S3	OL_1_ERR___	Optical
S3	OL_2_LFR___	Optical
S3	OL_2_LRR___	Optical
S3	SL_1_RBT___	Optical
S3	SL_2_LST___	Optical
S3	SR_1_SRA___	Topography
S3	SR_1_SRA_A_	Topography
S3	SR_1_SRA_BS	Topography
S3	SR_2_LAN___	Topography
S3	SY_2_SYN___	Optical
S3	SY_2_V10___	Optical
S3	SY_2_VG1___	Optical
S3	SY_2_VGK___	Optical
S3	SY_2_VGP___	Optical

Figura 1. Misiune, tip și instrument pentru produsele satelitare

În ceea ce privește soluția de monitorizare a sistemului, există multe variante de astfel de produse pe piață, chiar produse ale echipei de cercetare din UPB: (Ghiț, 2010), (Pop, 2016), (Poenaru, 2018), (Șerbănescu, 2015), (Pop, 2014), (Pop, 2007). De exemplu, Nagios este o alternativă viabilă, open source și populară în domeniu (Imamagic, 2007). Am ales Prometheus deoarece este mai ușor de folosit pentru a investiga containere decât Nagios, care, pentru a avea acces la metrici despre Docker trebuie să ruleze ca și `root`, în condițiile în care un server Nagios rulează în mod normal ca un utilizator „`nagios`”. Nagios este foarte bun pentru aplicații mici, sisteme statice. Prometheus (https://prometheus.io) poate monitoriza inclusiv aplicații care rulează pe un server (oferă informații despre orice, pornind de la numărul de cereri HTTP până la codul răspunsurilor primite în urma unei cereri). Referitor la partea a doua a soluției, bazată pe log-uri, Agenția Spațială Europeană și consorțiul Serco/Gael au definit un document de control al interfației, ce prezintă un standard de JSON pentru toate aplicațiile neoficiale ce folosesc DHuS.

Sentinel Data Dashboard (Sentinel, 2019) este un produs web, cu scopul de a oferi statistici în timp real, referitoare la descărcarea și accesul la datele satelitare (de exemplu, numărul de descărcări pe fiecare misiune și pe fiecare țară de proveniență a utilizatorului în parte). Statisticile sunt actualizate în fiecare zi și dețin informații despre accesul la produse pe platforme ale Agenției Europene Spațiale precum OpenHub, ColHub, IntHub and ServHub. Cum DHuS oferă și o interfață

pentru programatori, datele satelitare descărcate de pe acele platforme nu sunt enumerate (vezi Figura 2).

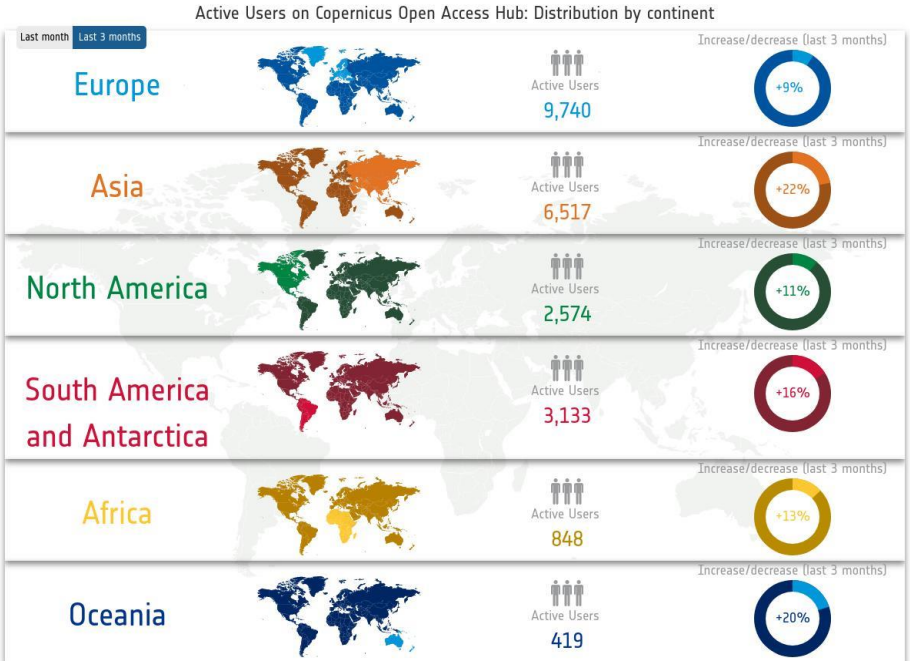

Figura 2. Exemplu de statistică raportată pe Sentinel Data Dashboard

Astfel, Sentinel Collaborative Ground Segment este o inițiativă de a extinde misiunile sateliților în diverse domenii și utilizarea lor de cât mai multe persoane. În țările partenere, există entități care fac așa numite „mirror sites", replici ale altor site-uri web și redistribuie produsele către utilizatori. Pentru acele site-uri ce folosesc DHuS ca software pentru distribuire, se recomandă trimiterea unui JSON, conform documentului, către Sentinel Collaborative Ground Segment, cu o frecvență zilnică, în maximum 12 ore de la încheierea ultimei zile raportate (Gascon, 2017).

3. Soluția propusă pentru monitorizarea sistemului

A. Monitorizare sistem

Am reprodus DHuS în modul scalabil 2.0, folosind Docker, sub forma unei stive de servicii interdependente:

- proxy-ul este reprezentat de un server Nginx;
- baza de date relațională este o instanță MySQL;
- masterul este o aplicație în Python care inserează intrări în baza de date;
- nodurile sunt aplicații în Python și Flask, identice ca funcționalitate.

Pe lângă serviciile care reproduc arhitectura DHuS, am adăugat servicii care au ca scop monitorizarea aplicației:

- Node Exporter pentru expunerea metricilor despre mașina host;
- cAdvisor pentru expunerea metricilor despre containere;
- Prometheus pentru preluarea metricilor de la Node Exporter și de la cAdvisor;
- Grafana pentru vizualizarea metricilor în timp real.

Docker este o platformă pe care dezvoltatorii o folosesc cu scopul de a crea, lansa și rula aplicații folosind containere. Containerele au la bază imagini, iar imaginile sunt create pornind de la un Dockerfile, un fișier ce conține, în ordine, toate comenzile necesare rulării containerului, precum și dependințele care trebuie instalate. Docker Compose este un utilitar folosit pentru a încapsula într-o singură aplicație mai multe containere. Un avantaj pe care îl aduce containerele în

fața mașinilor virtuale este faptul că, acestea păstrează o parte din izolarea mașinilor virtuale, la un cost mai mic față de resursele mașinilor host. Toate subsistemele unei aplicații, încapsulate în containere, rulează pe același host și împart kernel-ul. Versiunea de Docker folosită este 18.09.6, iar imaginile descărcate de pe DockerHub sunt: mysql:5.7, respectiv ultimele variante încărcate pentru prom/prometheus, prom/node-exporter, google/cAdvisor, grafana/grafana, nginx. Nginx este un server web ce poate acționa ca un proxy, balansând traficul din partea utilizatorilor și distribuind în mod egal cererile către toate nodurile din sistem.

În cadrul imaginilor implementate, am ales folosirea limbajului Python împreună cu Flask, un set de funcții și clase pentru dezvoltarea de aplicații web.

Prometheus este un produs software folosit pentru monitorizare, care extrage metrici de la anumiți clienți, numiți target-uri, utilizând HTTP. Deoarece sistemele pot expune aceste metrici în formate diferite, este nevoie de alte aplicații, numite exportatori, care se ocupă cu transformarea metricilor într-un format înțeles de Prometheus.

Am folosit Node Exporter pentru metrici ale mașinii host și cAdvisor pentru metrici ce țin de containerele care rulează la momentul respectiv.

Grafana este un instrument de vizualizare ce permite utilizatorilor să creeze diverse panouri cu grafice, diagrame, pornind de la o sursă de date, în cazul de față Prometheus (vezi Figura 3).

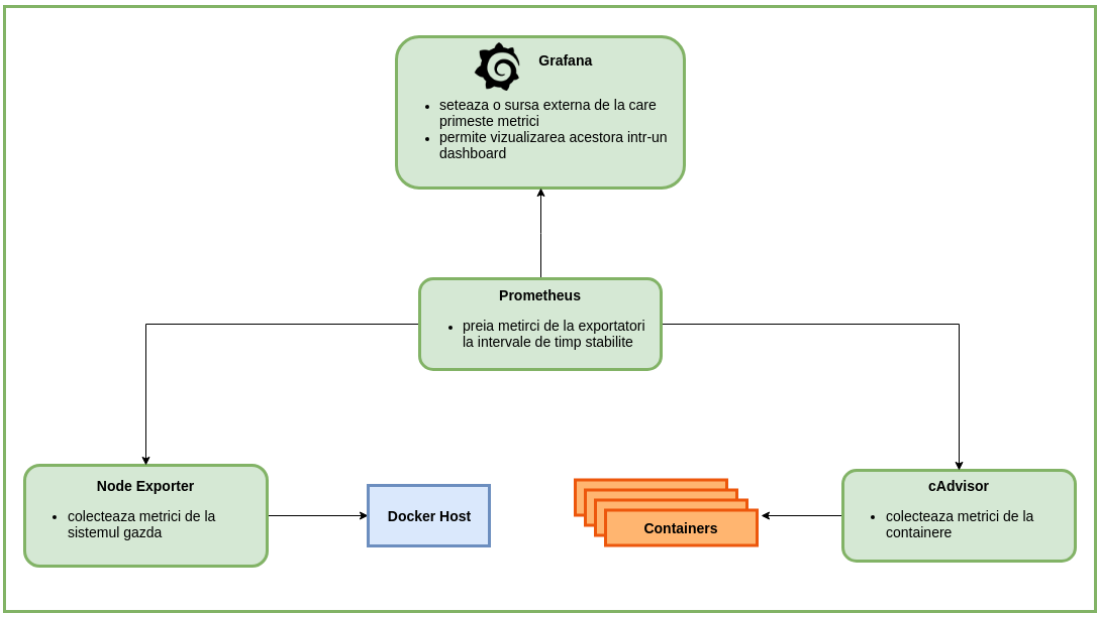

Figura 3. Arhitectura componentei de monitorizare sistem

Figura 3 ilustrează modul de interacțiune al tehnologiilor folosite. Prometheus folosește doi exportatori: cAdvisor și Node Exporter, de la care extrage la intervale de timp regulate metrici, pe care le actualizează constant cu noile valori venite. Node Exporter expune către Prometheus informații despre hostul pe care rulează stiva de containere precum CPU load, trafic de internet, memorie RAM disponibilă/utilizată etc. cAdvisor știe să culeagă aceleași informații, dar distribuite pe fiecare container în parte. Prometheus pune la dispoziție metricile venite de la ambii exportatori către Grafana, care interoghează serverul și vizualizează datele în timp real, atâta timp cât sursa de date nu este oprită.

B. Monitorizare servicii pe bază de log-uri

Logstash este un motor de colectare, procesare și integrare a datelor în alte arhitecturi, funcționând pe bază de evenimente. Are la bază Ruby, motiv pentru care sintaxa din fișierul de configurare seamănă cu acesta și, în plus, pot fi inserate ușor scripuri în Ruby care modifică datele. Logstash funcționează ca un pipeline, de obicei de forma input - filter - output și are în acest sens o gamă largă de module plugin, pachete independente ce se găsesc pe RubyGems.org.

File este modulul pe care l-am folosit pentru partea de input a pipeline-ului. Preia datele (log-urile) dintr-un fișier și se activează atunci când apare o linie nouă la sfârșit. Astfel, atâta timp cât este pornit daemonul de logstash, orice operație de adăugare în fișier este înregistrată.

Grok este un plugin pentru filtrarea datelor. Funcționează pe bază de expresii regulate, oferind deja o gamă de șabloane deja definite. Este folositor întrucât atribuie componentelor unui log etichete, ce pot fi utilizate ulterior. De exemplu, pentru log-urile care conțin informații despre descărcarea produselor, să existe etichete pentru utilizatori, pentru identificatorul produsului etc. Un șablon Grok arată %{SYNTAX:SEMANTIC}, unde SYNTAX este numele dat șablonului, și are în spate o expresie regulată, iar SEMANTIC este eticheta asociată cuvântului care se potrivește cu șablonul.

Jdbc_static filter plugin este folosit pentru filtrarea datelor raportându-se la o bază de date. Presupune conectarea la baza de date și apoi adăugarea de câmpuri noi bazându-se pe interogări ale tabelelor din baza de date.

Elasticsearch este un motor de căutare și o bază de date orientată document. Datele sunt stocate sub forma unor documente JSON și sunt accesibile printr-o interfață web RESTful, folosind protocolul HTTP. Câteva concepte de bază pentru înțelegerea modului în care funcționează sunt:

- un cluster este format din unul sau mai multe noduri. Numele lui este implicit „elasticsearch“;
- un nod este un singur server, ca parte a unui cluster;
- un index este o colecție de documente cu proprietăți (câmpuri) asemănătoare. Într-un cluster se pot afla indexuri;
- un document este un obiect scris în JSON (Java Script Object Notation) și reprezintă unitatea de bază de informație dintr-un index.
- Shards (cioburi) sunt subdiviziuni ale unui index. Fiecare subdiviziune funcționează ca un index total funcțional. Împărțirea în aceste subdiviziuni este importantă deoarece paralelizează căutările în index, îmbunătățind astfel performanța. Subdiviziunile indexului pot fi împărțite pe orice nod, neexistând o regulă pentru această operațiune.
- replicarea se face la nivel de index. Astfel, un index împărțit în subdiviziuni va avea același număr de replici din fiecare. Două replici ale aceleiași subdiviziuni nu pot exista pe același nod, asigurând un mecanism de recuperare în caz de eșec sau distrugere al unui nod. În mod implicit, fiecare index se împarte în 5 subdiviziuni și îi este făcută o singură replică.

Elasticsearch este considerat un motor de căutare puternic deoarece poate să gestioneze cantități uriașe de date cu timp de răspuns minim, bazându-se pe o arhitectură distribuită care poate scala cu până la 1000 de servere.

Kibana este o interfață web construită să funcționeze bine împreună cu Elasticsearch. Poate să ofere vizualizare grafică pentru cantități foarte mari de date și se actualizează în timp real.

HSQLDB (HyperSQL Database Engine) este un sistem de gestionare pentru baze de date relaționale. Este scris în Java și oferă un motor de căutare multi-thread și tranzacțional.

Am folosit Python3 pentru dezvoltare, printre modulele relevante aflându-se următoarele:

- JayDeBeApi: permite conectarea din cod la baze de date folosind JDBC (Java Database Connectivity);
- Elasticsearch: este un client ce oferă conectivitate la interfața REST API a motorului Elasticsearch; sintaxa folosită este foarte asemănătoare cu limbajul de interogare folosit de Elasticseach, dar în plus există și posibilitatea scrierii sub formă de clase pentru programare orientată obiect;
- Datetime: oferă clase și metode pentru manipularea datelor și a timpului, folosit în special pentru că output-ul oferit de Grok este sub formă de string;

- Collections – Counter: subclasa pentru numărarea obiectelor din dicționare;
- Json: face parsarea de JSON-uri ușoară și oferă o funcție de transformare din dicționar;
- în JSON, lucru foarte util în dezvoltarea scriptului de generare JSON cu statistici;
- Cron este un utilitar pentru sistemele de operare Unix care funcționează ca un planificator ce permite rularea unor scripturi la intervale regulate de timp (vezi Figura 4).

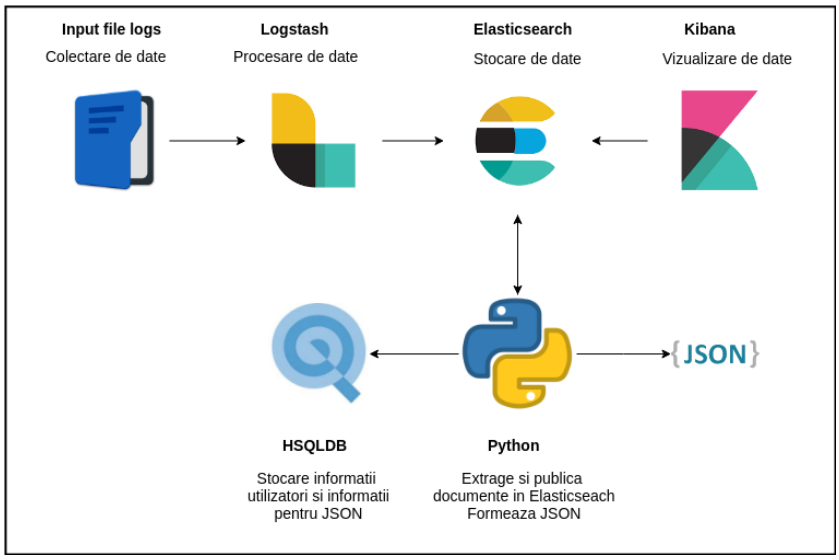

Figura 4. Arhitectura componentei de monitorizare pe bază de log-uri

Conform figurii, am folosit conceptul numit Elastic Stack sau ELK (Elasticsearch, Logstash, Kibana). La baza acestei stive stă faptul că Logtash preia log-urile din diverse surse, le procesează și le trimite la output către Elasticsearch. Elasticsearch stochează datele și le expune pentru Kibana, unde pot fi vizualizate în timp real.

Peste stiva ELK am adăugat un sistem de gestionare de baze de date relaționale, HSQLDB, care stochează utilizatorii sistemului, precum și date pentru construirea JSON-ului final.

În Python am construit datele de testare și am filtrat din Elasticsearch doar acele log-uri importante pentru generarea JSON-ului. De asemenea, am publicat documente noi pentru a afișa în Kibana statistici în plus față de ceea ce oferă deja Elasticsearch atunci când apare un log nou în fișier. De asemenea, utilitarul Cron permite rularea fișierului în Python de generare JSON la intervale definite de timp.

4. Detalii de implementare

A. Monitorizare sistem

Docker Compose este o soluție bună atunci când vine vorba de agregarea mai multor imagini de containere într-o singură aplicație. Compose folosește un fișier numit `docker-compose.yml`, care definește servicii, rețele și volume. Un serviciu în Docker este parte a unei aplicații distribuite mai mari. Ele mai sunt numite și „containere în producție”, fiecare având la bază o imagine: fie una deja existentă în mediul online, fie una definită de utilizator, cel mai frecvent printr-un executabil și un fișier Dockerfile, în urma operației de build. În cadrul stivei create am folosit imagini oficiale pentru Prometheus, Node Exporter, cAdvisor, Grafana, Nginx, MySQL. Nodurile și masterul sunt scripuri de python rulate din Dockerfile-uri.

În ceea ce privește conectivitatea între servicii în Docker, sunt de menționat două tipuri de drivere de rețea: bridge și overlay. Bridge este driverul de rețea implicit dacă nu este un altul specificat. Permite containerelor să comunice unul cu altul, în același timp oferind izolare față de

acelea care nu se află în aceeași rețea. Toate containerele ce rulează în aceeași rețea de tip bridge își expun porturile unul altuia și niciun port către restul internetului. Un alt avantaj al driverului bridge este faptul că asigură funcționalitate de DNS resolver, containerele se pot accesa unul pe altul pe bază de nume sau de alias. Am folosit rețea de tip bridge pentru conectivitatea serviciilor din Docker care se ocupă de monitorizare (Prometheus, Node Exporter, cAdvisor și Grafana) deoarece am considerat că acestea ar trebui să se poată accesa unul pe altul și, din motive de securitate, să nu poată fi accesate din exterior. În plus, pentru monitorizare, nu există necesitatea ca un utilizator extern rețelei să acceseze serviciile.

Overlay este un driver de rețea ce permite comunicarea containerelor de pe hosturi diferite, creând o rețea distribuită. Această rețea stă deasupra celorlalte rețele specifice hosturilor. Am folosit rețea de tip overlay pentru componentele DHuS (baza de date, master, noduri și proxy Nginx) deoarece funcționalitatea acestuia presupune accesare de resurse din exterior, indiferent de rețeaua din care utilizatorul face parte. Porturile pe care pot fi accesate serviciile se setează folosind `ports` și `expose`. Ports activează containerul să asculte cereri atât din exterior (de pe aceeași mașină host sau de pe altă mașină), cât și din interior (serviciile din stiva Docker). Expose activează containerul să asculte doar cererile din interior.

Volumele în Docker îndeplinesc două roluri principale: persistența și posibilitatea împărțirii datelor între containere diferite. În cazul de față, am considerat că nodurile, masterul și baza de date trebuie să împartă aceleași date. Masterul este cel ce populează tabelele, iar nodurile trebuie să poată vedea conținuturile acestora. Am creat un volum comun, numit `dhus_data`.

Un alt aspect important în ceea ce privește volumele, este maparea unui director/fișier de pe host cu un director/fișier de pe container. Sintaxa `host_dir:container_dir` presupune montarea directorului aflat la calea `host_dir` în container la calea `container_dir`. Astfel, dacă un container populează directorul de la calea `container_dir`, atunci schimbările se vor vedea atât local, cât și pe container.

Am creat volumul numit `dhus_data`, volum pe care îl împart toate containerele bazei de date, master și noduri, astfel încât datele populate de către master vor fi vizibile și în noduri. Volumul este atribuit la calea `/var/lib/mysql`.

Pentru Prometheus am atribuit directorul `prometheus` local la `/etc/prometheus`, pentru a urca în container fișierul de configurare prometheus.yml. Etichetele din `docker-compose.yml` setate pentru fiecare serviciu sunt relevante pentru Grafana, astfel încât se va putea distinge care dintre containere fac parte din componentele DHuS și care dintre containere sunt parte a aplicațiilor de monitorizare. Exemplu de etichetă: `org.label-schema.group: "dhus"`. Astfel, atunci când în Grafana, o metrică conține o etichetă diferită de DHuS, ea va fi ignorată dacă scopul este monitorizarea doar a serviciilor care reproduc DHuS. Am folosit un script `wait-for-it.sh` care primește ca argumente serviciul și portul după care se așteaptă, urmat de comanda ce trebuie dată imediat după ce serviciul rulează. Un exemplu este: `command: ["./wait-for-it.sh", "db:3306", "--", "python", "master.py"]`. A fost necesar ca scriptul master.py să nu ruleze decât atunci când baza de date este inițializată întrucât scriptul introduce date în tabele.

Pentru testare, am creat un nod Master care are rol de administrare a bazei de date. Conține un script `init.sql` care este rulat prima dată când este pornită baza de date. Acesta conține crearea unei baze de date, precum și crearea unei tabele. Datele sunt persistente și pentru a putea schimba structura tabelelor din baza de date, fără a scrie un cod dintr-un client, este nevoie de ștergerea volumelor. Scriptul în Python rulat de container inserează în tabela bazei de date valori dintr-o sursă externă (un fișier), imitând astfel comportamentul Masterului DHuS.

Nodes sunt identice ca funcționalitate. Folosind Flask, ele au un endpoint "/Node" asociat unei funcții ce execută o operație simplă.

Nginx are rolul de a împărți cererile clienților în mod egal la nodurile din sistem, astfel, în fișierul de configurare `nginx.conf` am setat `worker_connections=1024`, ce înseamnă numărul maxim de conexiuni simultane, atât din partea altor servere, cât și din partea clienților. Serverele către care Nginx redirecționează cererile se numesc upstream servers. În cazul stivei Docker,

acestea sunt Node1, Node2 și Node3. Scopul este de a demonstra că cererile utilizatorilor sunt redirectate de către serverul Nginx în mod egal. În imagine observăm că traficul prin Nginx este aproximativ egal cu suma traficului din Node1, Node2 si Node3, care sunt și egale, de asemenea.

1) Configurare servicii de extragere metrici

Fișierul de configurare pentru Prometheus se numește `prometheus.yml`. La nivel global am setat `scrape_interval = 15s`, acesta fiind intervalul de timp la care Prometheus să ceară metrici de la serviciile utilizate. Cea mai importantă secțiune din fișier este `scrape_configs`, unde sunt menționate (prin IP și port) exact endpoint-urile de la care se extrag metrici la un interval regulat de timp. De asemenea, se poate suprascrie `scrape_interval` din secțiunea global.

Deoarece toate serviciile de monitorizare se află în aceeași rețea de tip bridge, containerele se pot accesa prin nume. Un exemplu de configurare pentru extragerea metricilor de la Node Exporter este:

```
scrape_configs:
- job_name: 'cadvisor'
scrape_interval: 5s static_configs:
- targets: ['cadvisor:8080']
```

Pentru a verifica dacă Prometheus funcționează și dacă există conectivitate între el și serviciile sale exportatoare, Node Exporter și cAdvisor, accesăm în browser http://localhost:9090/targets (vezi Figura 5).

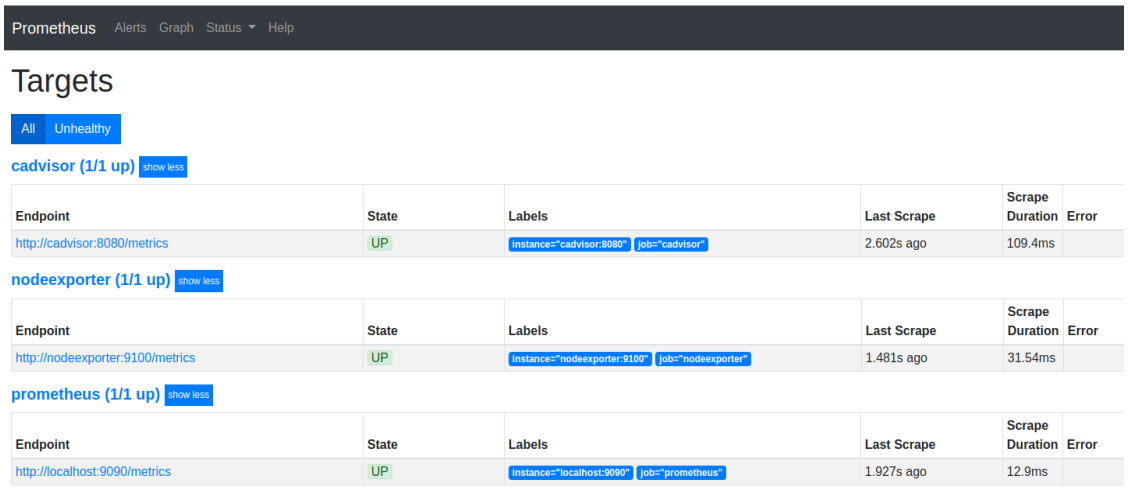

Figura 5. Componentele de la care Prometheus extrage metrici

Pentru cAdvisor și pentru Node Exporter am mapat directoare de pe host care conțin informații necesare pentru metricile căutate de serviciile respective. De exemplu, directoarele `/procs` pentru informații despre procese care rulează sau `/sys` pentru informații referitoare la kernel. cAvisor mapează întregul director root în container, doar cu drepturi de citire, din motive de securitate.

2) Raportare în interfața grafică Grafana

Directorul aferent serviciului Grafana conține un script numit *Zsetup.sh* și două directoare, *dashboards* și *datasources*.

Datasources conține fișierul `Prometheus.json` care specifică adresa de la care Grafana își ia datele pe care le afișează în interfață, la adresa `http://prometheus:9090`.

Interfețele conțin fișiere de tip JSON cu structura panourilor din interfață, precum și cu formulele de calcul ce au ca date de intrare informațiile culese de la Prometheus în timp real. Pentru crearea interfeței am pornit de la unele deja existente pe site-ul aplicației (Leitner, 2017) și am extras doar graficele și diagramele pe care le-am considerat relevante.

Fișierul setup.sh conține cod pentru încărcarea în Grafana a sursei de date și a interfețelor, lucru care poate fi făcut, de asemenea, și din interfață. Din browser, Grafana poate fi accesată la adresa `http://localhost:3000`. Interfața din Grafana redă două tipuri de statistici: despre mașina host și despre containerele componente DHuS. Statisticile despre mașina host sunt următoarele:

- timpul de când rulează, adică ora curentă din care se scade timpul de bootare;
- procentul de timp în care procesorul a fost inactiv, ca medie pe toate nucleele; se folosește `rate()`, o funcție care returnează media valorilor pe secundă raportată la intervalul de timp dat ca parametru, luând prima și ultima valoare din interval;
- numărul de nuclee ale procesorului;
- memoria RAM disponibilă pentru alocarea de noi procese sau pentru alocarea în cadrul proceselor deja existente;
- memoria swap liberă, în cazul în care nu mai este memorie RAM disponibilă;
- cantitatea de memorie externă liberă, indiferent de tipul sistemelor de fișiere existente;
- CPU load mediu, calculat pe o perioadă de 1 minut; load-ul este o măsura a supra-utilizării sau a sub-utilizării procesorului într-un sistem, numărul de procese executate de procesor sau numărul de procese în stare de așteptare;

De exemplu, pe un procesor cu un singur nucleu, 0.5 înseamnă că orice proces care apare va rula direct, 1 înseamnă că deja se află la capacitate maximă, iar peste 1 înseamnă că trebuie să stea la coadă, din moment ce doar un proces poate rula pe un nucleu la un moment dat. În general, ideal este ca load-ul procesorului să fie sub valoarea 0.7.

Pe un procesor cu 2 nuclee, utilizare 100% înseamnă un load de 2.00, iar pe un procesor cu 4 nuclee, utilizare 100% înseamnă un load de 4.00.

- numărul de procese care rulează la un moment dat, respectiv numărul de procese blocate;
- rata instantanee a întreruperilor într-un interval de 5 minute, mai exact se iau în considerare ultimele valori, raportate la intervalul de timp;
- rata utilizării procesorului, ca medie între toate nucleele, în funcție de modul în care rulează, mai exact cât timp dintr-o secundă petrece în fiecare mod de utilizare.

  Modurile posibile ale procesorului sunt:

  - `idle`: procesorul este inactiv, nu este utilizat de niciun program,
  - `iowait`: procesorul așteaptă după operații de intrare/ieșire,
  - `irq`: procesorul petrece timp rulând un cod special pentru întreruperi în modul kernel,
  - `nice`: procesorul petrece timp rulând procese cu valori nice pozitive (prioritate redusă), adică procese care nu sunt o prioritate și pe care procesorul le-ar programa în mod normal atunci când are ocazia,
  - `softirq`: procesorul rulează o întrerupere software, de obicei o modalitate de a comunica cu kernelul sau de a chema apeluri de sistem,
  - `steal`: timp pe care procesorul virtual îl petrece așteptând procesorul de pe host, ocupat cu altă operație; se întâlnește pe mașinile care conțin medii virtualizate,
  - `system`: procesorul rulează în modul kernel,
  - `user`: procesorul rulează în modul utilizator;
- memoria RAM utilizată;
- rata numărului de octeți citiți/scriși la Intrare/Ieșire, adică media valorilor pe secundă, timp de 1 minut;
- rata traficului care intră, respectiv traficul care iese pe interfață eth0 (ethernet), reprezintă media valorilor pe secundă, timp de 1 minut.

Statisticile despre containerele componente DHuS sunt:

- `CPU load`, ca medie pe toate containerele;
- suma memoriei RAM utilizată de containere;
- suma memoriei externe ocupată de containere;
- numărul de containere care rulează se află folosind orice metrică referitoare la containere cu funcția `scalar()` care numără câte rezultate aduce funcția primită ca argument;
- rata utilizării procesorului ca medie între toate nucleele și pe fiecare container în parte;
- suma memoriei cache utilizate pentru fiecare container în parte;
- rata traficului de internet care intră în container pentru fiecare container în parte;
- rata traficului de internet care iese din container pentru fiecare container în parte (vezi Fig. 6).

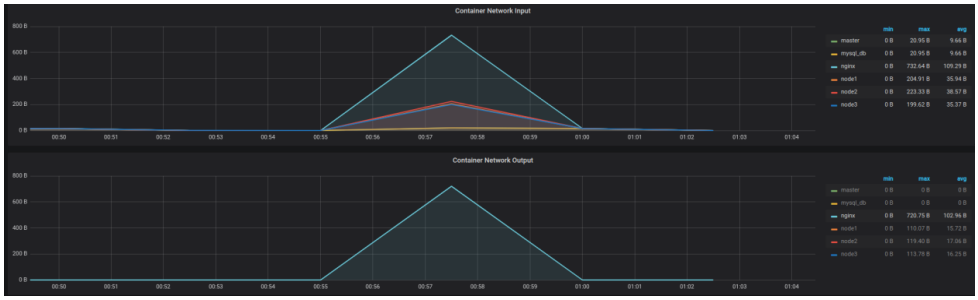

Figura 6. Traficul de internet pe cele trei noduri componente DHuS

B. Monitorizare servicii pe baza de log-uri

Am pornit de la un fișier de log-uri DHuS din luna februarie a acestui an. Are dimensiunea de 1.8 GB și conține 5 961 681 de log-uri. Folosind utilitarul grep, am cercetat fișierul și am ajuns la concluzia că log-urile relevante sunt acelea care specifică că o descărcare de produs s-a efectuat cu succes de un anumit utilizator:

```
[0.13.4-22][2019-02-24 06:25:43,951][INFO] Product '22768885-6aee-4176-
9214-544c8324bfbb'(S1B_IW_RAW__0SDV_20190224T042917_20190224T042950_
015083_01C305_EF1F) download by user 'uvt_sync' completed in 342501ms ->
1561350640 (DownloadActionRecordListener.java:76 - http-nio-8081-exec-1)
```

DHuS este un produs open source și am căutat în instanța Github a proiectului modelul bazei de date utilizate. Am aflat clasă aferentă tabelei User și câmpurile folositoare mai târziu pentru generarea JSON-ului sunt următoarele:

- LOGIN: username-ul utilizatorului,
- COUNTRY: țara cu care utilizatorul s-a înregistrat.

Pentru a genera propriul fișier de log-uri, cu date pentru cel puțin 3 luni în urmă, am avut nevoie de utilizatori și de produse. Am instalat o instanță de HSQLDB și, din interfață grafică Database Manager Swing, am creat tabela User cu cele două câmpuri. Scripturile în Python importante pentru această etapă a soluției sunt:

- `countries_mapping.py`: conține o listă cu toate țările menționate în documentul de specificații, precum și o mapare continent – țară, sub forma unui dicționar al cărei cheie este continentul, iar valoarea este o listă de țări. Are două metode: `random_country()` care returnează o țară la întâmplare, și `get_continent(country)`, care returnează continentul țării date ca parametru;
- `generate_users.py`: generează un număr de username-uri folosind modulul faker din Python, îi asociază fiecărui utilizator o țară aleasă aleator și inserează în baza de date informațiile. Pentru conectarea la baza de date (care trebuie să ruleze) am folosit

modulul jaydebeapi. Username-ul este cheie unică, în cazul în care se încearcă introducerea aceluiași utilizator, se aruncă o excepție, aceasta este prinsă și se trece a următoarea inserare;

- `generate_logs.py`: primul pas a fost generarea aleatorie a datelor în format timestamp și ordonarea lor ascendent, urmând asocierea fiecărei date cu un utilizator și un produs (prin denumire și prin dimensiune). Pentru extragerea produselor, am folosit Logstash pe fișierul de log-uri DHuS, care a repartizat output-ul în Elasticsearch. Am deschis o conexiune pe local la serverul de Elasticsearch și am returnat toate documentele din indexul tocmai creat. Documentele conțin, printre alte informații legate de descărcarea datelor, dimensiunea și denumirea produsului.

Pentru configurarea Logstash, am trecut prin fiecare fază din pipeline-ul input – filter – output. Pentru intput, am folosit File și am dat calea către locul în care se găsește fișierul de log-uri. Pentru filtrare, am folosit Grok și am scris un șablon pentru log-ul de descărcare cu succes de produs, marcând prin etichete părțile importante ale log-ului: data descărcării, tipul de log, id-ul produsului, denumire, utilizator, timp de descărcare și dimensiune produs. De asemenea, orice log care nu se potrivește cu acest șablon, este ignorat și nu apare în output.

Conform documentului de specificații pentru JSON (Pathirannehelage, 2018), se doresc următoarele informații:

- volumul total de descărcări și numărul de utilizatori înregistrați de la începutul operațiunii (03/10/2014);
- numărul de utilizatori activi (care au descărcat un produs în perioada raportată) grupați pe continente și împărțiți pe misiuni (S1, S2, S3 și cumulativ), în ultimele 3 luni (90 de zile), în ultima lună (31 de zile) și de la începutul operațiunii, precum și un procentaj de creștere sau descreștere față de ultima raportare;
- numărul de utilizatori activi grupați pe țări europene, împărțiți pe misiuni și apoi pe instrumente, în ultimele 3 luni, în ultima lună, precum și un procentaj de creștere sau descreștere față de ultima raportare;
- numărul de utilizatori activi pentru fiecare misiune în parte și, cumulativ, zilnic pentru ultima lună și săptămânal pentru ultimele 3 luni;
- numărul de utilizatori pe intervale de număr de descărcări (<10, 10 – 99, 100 – 1000, >1000) pe fiecare misiune în parte și, cumulativ, în ultima lună și în ultimele 3 luni;
- volumul în octeți al produselor descărcate, grupate după tipul produsului și împărțite pe misiuni și apoi pe instrumente, în ultima lună, în ultimele 3 luni; dacă tipul produsului este specificat, atunci instrumentul se lasă null;
- topul primelor 10 țări în ceea ce privește numărul de descărcări, grupate după misiune, apoi după instrument, în ordine alfabetică, în ultimele 3 luni, respectiv în ultima lună, precizând numele țării și numele continentului, dar nu și numărul de descărcări;
- numărul și volumul de descărcări pentru fiecare misiune în parte, zilnic pentru ultima lună, săptămânal pentru ultimele 3 luni și în total;
- numărul total de produse descărcate de la începutul operațiunii pentru fiecare misiune în parte, împărțit la numărul de produse publicate.

În această etapă a implementării soluției am folosit două scripturi în Python:

- `inițial_script.py`: pentru fiecare categorie de raportare ce trebuie făcută, se inserează în tabelul bazei de date tipul raportării (caracteristică care ajută la diferențierea între tipurile de statistici cerute) cu valori standard, 0 ca volum de date descărcate, ca număr de descărcări sau ca număr de utilizatori, pentru fiecare misiune, instrument și tip de produs în parte;
- `generate_json.py`: este scriptul ce trebuie să ruleze periodic, în fiecare zi, în maxim 12 ore de la încheierea ultimei zile raportate. Bazându-se pe tipul raportării, precum și pe caracteristicile fiecărui tip de statistică, creează un JSON și îl pune într-un fișier cu

numele "`collaborative_XX_sentinel_data_yyyymmdd.json`", unde XX = reprezintă id-ul în cadrul Collaborative Ground Segments („ro", „fr" etc.), iar „`yyyymmdd`" este timpul generării fișierului.

Am decis ca scriptul `generate_json.py` să fie rulat în fiecare zi la ora 0:30. Din linia de comandă am rulat „crontab -e", ce permite editarea fișierului în care se scriu reguli:

```
30 0 * * * /usr/bin/python /home/../generate_json.py
```

Raportarea în interfața Kibana presupune ca serverul de Elasticsearch să ruleze, astfel încât să poată avea acces la indexuri. Am definit un index pattern, însemnând că am făcut referință către un index deja existent în Kibana. Apoi, pentru raportarea în aplicație, este nevoie de căutare și filtrare în spațiul datelor. Am adăugat la datele deja existente în Elasticsearch un câmp Country, respectiv un câmp Continent, pentru a putea raporta în funcție de țara și continentul de proveniență a utilizatorilor. Pentru aceasta am folosit `jdbc_static filter plugin`, care extrage țara și continentul din baza de date. O alternativă ar fi fost crearea unui script care să se conecteze la Elasticsearch și să actualizeze datele, însă nu ar fi fost în timp real, precum varianta de mai sus.

În Logstash, timpul de parsare pentru un fișier de aproximativ 8.1 GB și 5 961 681 de linii a fost de aproximativ 15 minute, cu o singură instanță de Logstash. O a doua instanță care rulează pe același fișier va produce împărțirea datelor din fișierul de intrare, reducând astfel munca unui singur nod. În cazul Elasticsearch, modificarea numărului de noduri nu aduce rezultate semnificative, întrucât numărul de log-uri noi într-o zi nu este o valoare mare.

5. Concluzii

În concluzie, monitorizarea unui sistem, în cazul de față a sistemului DHuS, are caracteristici unice în funcție de componentele sistemului și de ceea ce realizează acesta. Am considerat că relevant pentru monitorizarea de sistem este în primul rând, traficul de internet, fiind vorba de un server care redirectează cereri de utilizatori către alte noduri. Apoi, valori de utilizare a procesorului, precum și a memoriei RAM sau memoriei externe, memoria RAM și swap liberă și disponibilă. În plus, pentru mediul în care am decis să testez, și anume Docker, monitorizarea mașinii host este și ea relevantă, cu aceleași metrici menționate mai sus. Monitorizarea de log-uri are un scop precis în cazul DHuS și anume generarea periodică a JSON-ului pentru publicarea online cu scopul de a agrega statistici pentru site-urile membre Collaborative Ground Segments, care pun la dispoziție datele satelitare.

O eventuală dezvoltare ulterioară pentru componenta de monitorizare a sistemului este transmiterea de alerte în cazul în care parametrii pentru un anumit container nu se încadrează în valori normale de funcționare. În acest sens, utilitarul AlertManager se integrează bine cu Prometheus pentru trimitere de alerte.

Pentru un sistem de monitorizare, alertarea este esențială întrucât interfața nu poate fi urmărită în permanență de către cineva. Ideal este ca sistemul să ruleze, și numai atunci când sunt probleme, persoanele care au potențial de rezolvare a problemei să fie chemate (pe mail, Slack). Lucrul cu AlertManager se împarte în două etape: în prima etapă, regulile de alertare sunt scrise în fișierul de configurare aparținând lui Prometheus, urmând ca atunci când se încalcă o regulă, să se trimită o alertă către AlertManager. Cea de-a doua etapă presupune inhibarea sau gruparea alertelor emise la pasul anterior.

În ceea ce privește dezvoltarea componentei de monitorizare log-uri, o dezvoltare ulterioară interesantă ar fi o statistică privind motivul utilizatorilor pentru accesul la datele satelitare. Pentru a selecta produse, se poate desena un poligon care să marcheze zona geografică pentru care se dorește informația. Astfel, în funcție de desenarea sau nu a poligonului, respectiv de descărcarea sau nu a produsului, am ales trei tipuri de motive.

Dacă un utilizator descarcă direct o resursă fără a modifica coordonatele poligonului, probabil că o descarcă pentru testare. Dacă un utilizator doar desenează un poligon, atunci el prezintă interes

pentru resursă. Dacă un utilizator desenează poligonul și descarcă resursa, înseamnă că, probabil, lucrează cu acest tip de date și are nevoie de valori concise.

Cu toate acestea, la momentul actual, log-urile nu conțin informații despre desenarea poligoanelor, deci o dezvoltare în acest sens nu ar putea fi efectuată momentan. De asemenea, monitorizarea pe bază de log-uri ar putea fi extinsă la orice serviciu parte al aplicației. Astfel, orice poate fi monitorizat din punct de vedere al funcționalității.

BIBLIOGRAFIE

1. Alexandru, A., & Coardoș, D. (2017). *BIG DATA–Concepte, Arhitecturi și Tehnologii*. Revista Română de Informatică și Automatică, 27(1), 15-24.
2. Fioriti, V., Chinnici, M. (2017). *Node seniority ranking in networks*. Studies in Informatics and Control, 26(4), 397-402.
3. Gascon, F., Bouzinac, C., Thépaut, O., Jung, M., Francesconi, B., Louis, J., Languille, F. (2017). *Copernicus Sentinel-2A calibration and products validation status*. Remote Sensing, 9(6), 584.
4. Ghit, B., Pop, F., Cristea, V. (2010). *Epidemic-style global load monitoring in large-scale overlay networks*. In 2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, IEEE, 2010, pp. 393-398.
5. Imamagic, E., Dobrenic, D. (2007, June). *Grid infrastructure monitoring system based on nagios*. In Proceedings of the 2007 workshop on Grid monitoring, ACM, pp. 23-28.
6. Iordache, G., Paschke, A., Mocanu, M., Negru, C. (2017). *Service level agreement characteristics of monitoring wireless sensor networks for water resource management* (slas4water). Studies in Informatics and Control, 26(4), 379-386.
7. Leitner, P., Ebner, M. (2017, July). *Development of a dashboard for Learning Analytics in Higher Education*. In International Conference on Learning and Collaboration Technologies. Springer, Cham, pp. 293-301.
8. Neagu, G., Vrejoiu, M.H, Preda, S.A., Stanciu, A. (2017). *Platforme IOT – Soluții actuale și Tendințe de Evoluție.* Revista Română de Informatică și Automatică, 27(3), 5-18.
9. Pathirannehelage, S., Kumarapeli, P., Byford, R., Yonova, I., Ferreira, F., de LUSIGNAN, S. (2018, May). Uptake of a Dashboard Designed to Give Realtime Feedback to a Sentinel Network About Key Data Required for Influenza Vaccine Effectiveness Studies. In MIE (pp. 161-165). Chicago.
10. Poenaru, A., Istrate, R., Pop, F. (2018). AFT: *Adaptive and fault tolerant peer-to-peer overlay -A user-centric solution for data sharing*. Future Generation Computer Systems 80: pp.583-595.
11. Pop, F., Cristea, V. (2007). *Intelligent strategies for dag scheduling optimization in grid environments*. In Proceedings of the 16th International Conference on Control Systems and Computer Science (CSCS16'07), May (pp. 22-25) (2015).
12. Pop, F., Potop-Butucaru, M. eds., (2015). *Adaptive Resource Management and Scheduling for Cloud Computing: Second International Workshop*, ARMS-CC 2015, Held in Conjunction with ACM Symposium on Principles of Distributed Computing, PODC 2015, Donostia-San Sebastián, Spain, July 20, 2015, Revised Selected Papers (Vol. 9438). Springer.
13. Pop, F., Potop-Butucaru, M., (2016). *ARMCO: Advanced topics in resource management for ubiquitous cloud computing: An adaptive approach*, pp.79-81.
14. Sentinel Data Dashboard, Online, Available: https://dashboard.copernicus.eu, Ianuarie 2020.
15. Șerbănescu, V., Pop, F., Cristea, V. and Antoniu, G., (2015). A formal method for rule analysis and validation in distributed data aggregation service. World Wide Web, 18(6), pp.1717-1736.
16. Talaș, A., Pop, F. and Neagu, G., (2017). *Elastic stack in action for smart cities: Making sense of big data*. In 2017 13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP) IEEE, September, pp. 469-476.

17. Tona, C., Bua, R. (2018, April). *Open Source Data Hub System: free and open framework to enable cooperation to disseminate Earth Observation data and geospatial information*. In EGU General Assembly Conference Abstracts (Vol. 20, pp. 13038).
18. Wu, G., Zhang, H., Qiu, M., Ming, Z., Li, J., Qin, X. (2013). *A decentralized approach for mining event correlations in distributed system monitoring*. Journal of parallel and Distributed Computing, Chicago, 73(3), pp. 330-340.

Sânziana-Aurelia VOICU este studentă la master în cadrul Departamentului de Calculatoare și Tehnologia Informației la Universitatea Politehnica din București. Interesele sale generale de cercetare sunt: dezvoltarea de software, proiectarea sistemelor de procesare de date, regăsirea informațiilor, agregarea și vizualizarea datelor.

Sânziana-Aurelia VOICU is Master's Student at the Department of Computer Science and Engineering at the Politehnica University of Bucharest. Her general research interests include: software development, processing system design, information retrieval, data aggregation and data vizualization.

Florin POP este profesor în cadrul Departamentului de Calculatoare și Tehnologia Informației la Universitatea Politehnica din București. De asemenea, lucrează ca cercetător științific gradul I la Institutul Național de Cercetare-Dezvoltare în Informatică (ICI) București. Interesele sale generale de cercetare sunt: sisteme distribuite pe scară largă (proiectare și performanță), grid computing și cloud computing, sisteme peer-to-peer, gestionarea Big Data, agregarea datelor, tehnici de regăsire și clasificare a informațiilor, metode de optimizare Bio-Inspired.

Florin POP is professor at the Department of Computer and Information Technology, the Politehnica University of Bucharest. He also works as a 1st degree scientific researcher at National Institute for Research and Development in Informatics (ICI) Bucharest. His general research interests are: distributed systems (design and performance), grid computing and cloud computing, peer-to-peer systems, Big Data management, data aggregation, information retrieval and classification techniques, Bio-inspired optimization methods.

Cătălin NEGRU este inginer de sistem și cercetător în cadrul Departamentului de Calculatoare al Facultății de Automatică și Calculatoare și membru activ al Laboratorului de Sisteme Distribuite al Universității Politehnica din București. A obținut doctoratul în 2016, interesele sale de cercetare incluzând sisteme distribuite, eficiența energetică, stocare în cloud, sisteme cyber-fizice, GIS. Cercetările sale au dus la publicarea a numeroase lucrări și articole la reviste științifice importante, precum Future Generation Computer Systems, Soft Computing, International Journal of Applied Mathematics and Computer Science și conferințe (ICCP, AQTR, HPCC, CISIS, CSCS etc.). Este implicat în mai multe proiecte de cercetare naționale și internaționale.

Cătălin NEGRU is a system engineer and researcher at the Computer Science Department of the Faculty of Automatic Control and Computers and an active member of Distributed System Laboratory at University Politehnica of Bucharest. He obtained his PhD from the same faculty in 2016. His research interests include distributed systems, energy efficiency, cloud storage, cyber-physical systems, GIS. His research has led to the publishing of numerous papers and articles at important scientific journals, such as Future Generation Computer Systems, Soft Computing, International Journal of Applied Mathematics and Computer Science and conferences (ICCP, AQTR, HPCC, CISIS, CSCS etc.). He is involved in several national and international research projects.

Adrian STOICA este liderul grupului de aplicații software la TERRASIGNA. Interesele sale generale de cercetare includ: procesarea de imagini satelitare, sisteme distribuite, baze de date spațiale, integrare software, managementul serviciilor IT, Big Data. Este manager de proiect / lider tehnic al diferitelor proiecte și activități din cadrul Agenției Spațiale Europene (ESA), a Comisiei Europene (CE) H2020 și a contractelor programului național de cercetare, legate de observarea Pământului, prelucrarea imaginilor prin satelit, date mari, aplicații mobile, aplicații web, baze de date.

Adrian STOICA is the head of Software Application Group at TERRASIGNA. His general research interests include: satelite image processing, distributed systems, spatial databases, software integration, IT Service Management, Big Data. He is project manager/Technical leader of various projects and activities under European Space Agency (ESA), European Commission (EC) H2020 and National Research Programme contracts, related to Earth Observation, Satellite Image Processing, Big Data, Mobile Applications, Web Applications, Databases.