PROJEKT

Multitenantné operačné centrum kybernetickej bezpečnosti riešené ako otvorená cloudová služba s prvkami strojového učenia

V14 funkčná infraštruktúra

KPB5 – Realizácia riešenia

typ – dokumentácia

mesiac odovzdania – M19

Financované Európskou úniou
NextGenerationEU

PLÁN OBNOVY

ÚRAD VLÁDY
SLOVENSKEJ REPUBLIKY

Obsah

1. ELK Stack 14
Opis systému 14
Dokumentácia systému 14
Inštalácia - variant Docker 15
Základná konfigurácia 17
Integrácie 22
2. Shuffle 22
Opis systému 22
Dokumentácia systému 23
3. Wazuh 23
Opis systému 23
Dokumentácia systému 23
Inštalácia - variant Quickstart 24
Základná konfigurácia 24
Integrácie 25
4. Greenbone Community edition 25
Opis systému 25
Dokumentácia systému 26
Inštalácia - variant Docker 26
Základná konfigurácia 33
5. OpenCTI 33
Opis systému 34
Dokumentácia systému 34
6. MISP 34
Opis systému 34
Dokumentácia systému 34
7. GLPI 34
Opis systému 34
Dokumentácia systému 35
Inštalácia - variant Docker (Compose) 35
Základná konfigurácia 37
Integrácie 38
8. NetBox 39
Opis systému 39
Dokumentácia systému 39

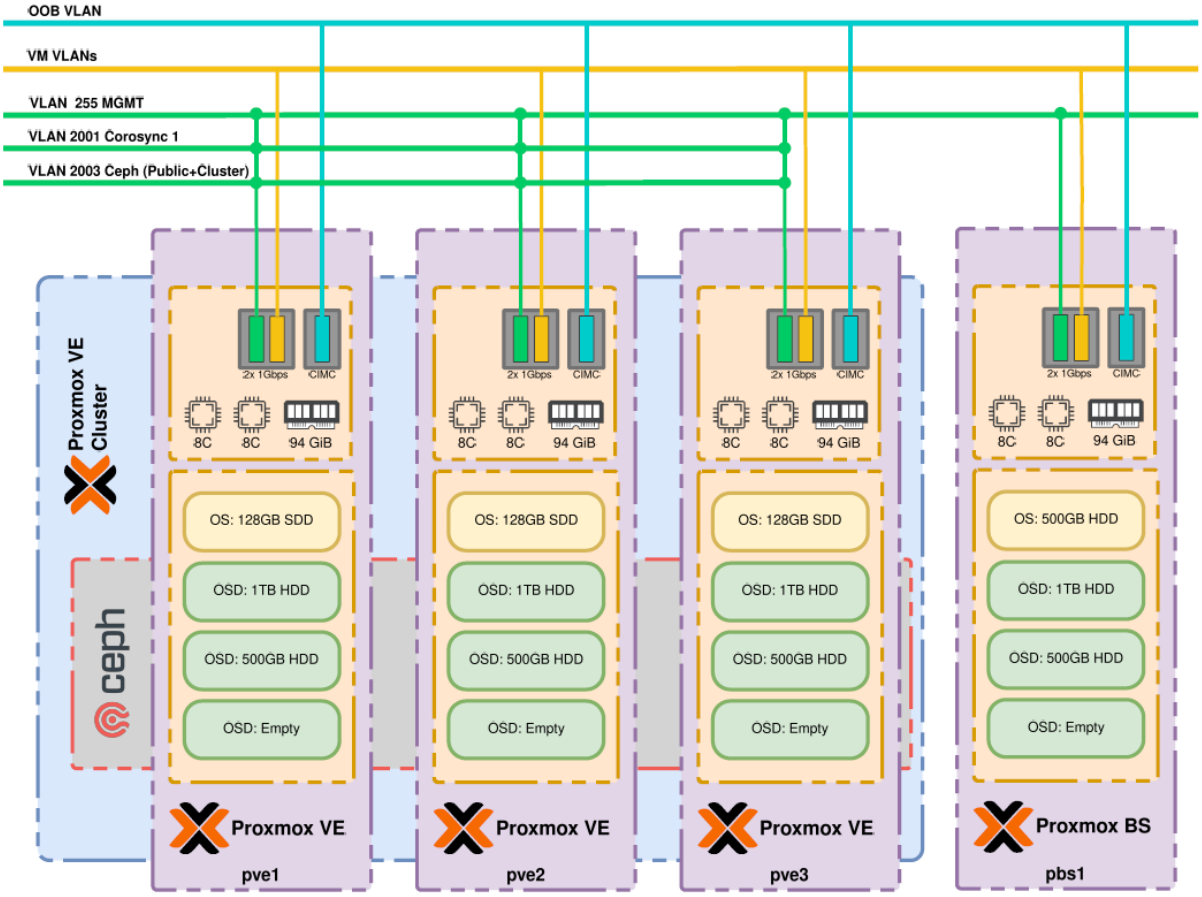
9. DFIR IRIS – Ticketing 39
Opis systému 39
Dokumentácia systému 39
10. DFIR IRIS – Evidencia, správa incidentov 39
Opis systému 40
Dokumentácia systému 40
11. Kibana (ELK) 40
Opis nástroja 40
Dokumentácia systému 40
Inštalacia - variant Docker 41
Základná konfigurácia 42
Integrácie 42
12. Wiki.js 43
Opis systému 43
Dokumentácia systému 43
Inštalácia - variant Docker 43
Základná konfigurácia 45
Integrácie 45

0.1 POPIS INFRAŠTRUKTÚRY KUBERNETES

Kubernetes je platforma s otvoreným zdrojovým kódom určená na automatizované nasadzovanie, škálovanie a správu kontajnerizovaných aplikácií - v praxi sa aj preto najčastejšie stretáva s označením, že ide o aplikačný orchestrátor. Na rozdiel od viac tradičných spôsobov nasadenia kontajnerov v rámci jediného výpočtového servera - najčastejšie pomocou nástroja Docker - ide v prípade technológie Kubernetes o zoskupenie viacerých takýchto výpočtových uzlov pod jednu abstraktnú rovinu. Tieto uzly tvoria klaster, ktorý zabezpečuje beh aplikácií nielen s vysokou dostupnosťou, ale aj napr. s dynamickým škálovaním na základe aktuálnej záťaže danej aplikácie vytvorením viacerých replík. Aj pokiaľ sa používateľ rozhodne nepoužiť funkcionalitu viacerých replík tej istej aplikácie, Kubernetes umožňuje rýchle zotavenie aplikácie aj v prípade, že pri ľubovoľnom uzle klastra príde ku výpadku - v tom prípade kontroléry Kubernetes uzla zodpovedné za jednotlivé časti nasadenia zabezpečia, že jednotlivé aplikácie budú znovu spustené na stále dostupných uzloch klastra. Kubernetes tak vo výsledku rozširuje možnosti kontajnerizačných démonov o viac komplexnú správu kontajnerovej infraštruktúry.
Kubernetes využíva deklaratívny model na definovanie a abstrakciu bežiacich komponentov - túto funkcionalitu zabezpečuje pomocou objektov rôznych typov (v Kubernetes terminológii taktiež \code{kind}), kde každý typ má jasne preddefinovanú štruktúru. Typicky sa tieto objekty definujú v súboroch vo formáte \ac{YAML}, ktoré následne používateľ môže do klastra aplikovať pomocou príkazu \code{kubectl apply --filename file.yaml}.

Ako hypervízor bol zvolený Proxmox Virtual Environment. To si vyžadovalo splnenie hneď niekoľkých úloh:
• aktualizácia firmvéru a ovládačov na všetkých fyzických serveroch a konfigurácia Out-of-band (OOB) manažmentu na vzdialenú správu serverov.
• inštalácia PVE platformy na každý z vybraných serverov
• základná konfigurácia sieťovej konektivity uzlov a príprava jednotlivých podsietí pre potrebné služby:
– Podsieť určená na manažment PVE serverov.
– Corosync podsieť (technológia použitá na klastrovanie PVE serverov)
– Ceph podsieť (technológia zodpovedná za úložisko pre Kubernetes klaster
– Podsiete určené pre virtuálne stroje (napr. Kubernetes uzly)
• Vytvorenie PVE klastra z vytvorených a nakonfigurovaných serverov.
• Vytvorenie Ceph klastra v rámci PVE uzlov a nasadenie potrebných démonov. Tie zahŕňajú monitorovacie služby (Monitors), riadiace služby (Managers) a jednotlivých úložných démonov (Object Storage Daemon (OSD)), kde každá inštancia OSD je typicky zodpovedná za ukladanie dát na konkrétny priradený disk.
• Nasadenie zálohovacieho systému v podobe Proxmox Backup Server (PBS) na dedikovaný server. PBS je softvérové riešenie určené na efektívne a inkrementálne zálohovanie virtuálnych strojov a kontajnerov bežiacich v PVE prostredí a poskytuje natívnu integráciu s PVE systémom [17], prečo aj bolo prirodzenou voľbou pri výbere zálohovacieho systému.

Splnenie všetkých týchto úloh viedlo k vytvoreniu nasledujúcej plne funkčnej infraštruktúry
hypervízorov, ktorá mohla byť použitá v ďalších krokoch na implementáciu samotného Kubernetes klastra:



➢ Infraštruktúra virtuálnych strojov - IaaC

Pre nasadenie virtuálnej infraštukrúry strojov, ktoré sú zodpovedné za beh klastra bol zvolený nástroj Terraform od spoločnosti HashiCorp, ktorý umožňuje deklaratívne definovať žiadaný stav infraštruktúry
pomocou konfiguračných súborov vo formáte HashiCorp Configuration Language (HCL). Terraform podporuje široké spektrum poskytovateľov (terminologicky a konfiguračne dostupné ako provider), nad ktorými je schopný žiadané objekty (v tomto prípade primárne virtuálne stroje) nasadiť. To poskytuje vytvorenému riešeniu jednoduchú rozšíriteľnosť do budúcnosti v prípade, že sa rozhodne pre migráciu z Proxmox-u na iný hypervízor, poprípade o rozšírenie klastra v podobe dodatočných virtuálnych strojov, ktoré nie nutne musia byť súčasťou toho istého klastra hypervízorov. To taktiež otvára možnosti pre vytvorenie testovacích klastrov aj mimo takéhoto predvoleného riešenia. Pre prácu s nástrojom Terraform bol vytvorený repozitár, v ktorom bol v rámci práce vytvorený vlastný modul (v terminológii Terraform ako module) pre jednoduché vytváranie viacerých replík v podobe rôznych prostredí. Pre vytvorenie virtuálnych strojov je následne potrebné už iba vyplniť vytvorenú premennú virtual_machines

daného modulu, ktorá reprezentuje zoznam špeciálne definovaných objektov, kde každý objekt reprezentuje jeden virtuálny stroj. Pri použití tohto modulu teda konfigurácia nasadenia vyzerá nasledovne:

```
module "infrastructure" {
source = "../../modules/k8s-proxmox"
virtual_machines = [
{
name = "server1.k8s.kis.fri.uniza.sk"
vm_id = 51151
description = "K8S - PROD"
tags = local.tags
node_name = "pve1"
pool_id = local.pool_id
cores = local.server_cpus
memory = local.server_memory
network_bridge = local.network_bridge
boot_order = local.boot_order
cdrom = {
file_id = local.iso
}
disk = [
{
size = local.disk_size
interface = "scsi0"
}
]
datastore_id = "vd"
ipv4_address = "10.23.153.151/24"
ipv4_gateway = local.ipv4_gateway
dns_domain = local.dns_domain
dns_servers = local.dns_servers
},
{
name = "server2.kis.fri.uniza.sk"
...
```

Kód: Ukážka použitia vytvoreného Terraform modulu

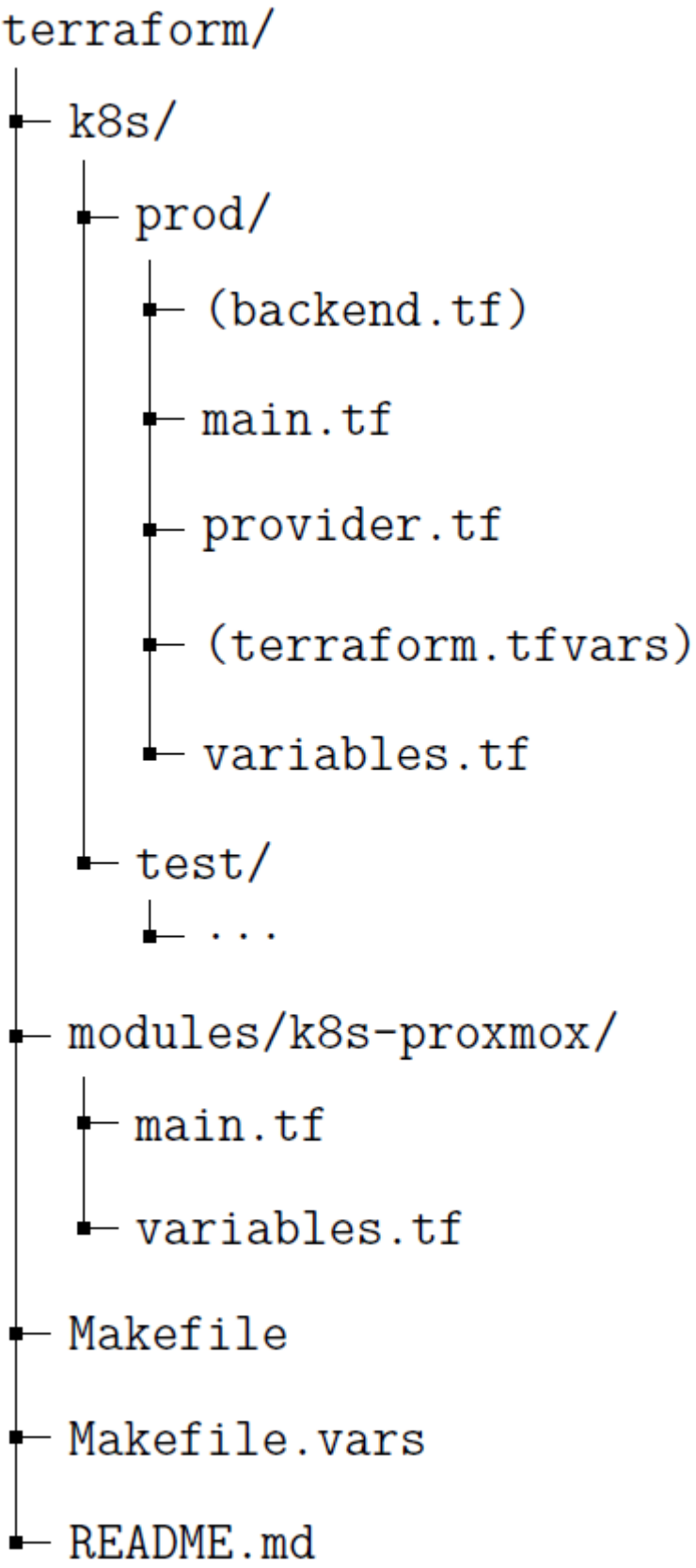
Možno vidieť, že pod sprístupnenou premennou virtual_machines je možné definovať ľubovoľný počet virtuálnych strojov, ktorá zároveň umožňuje definovať ich vlastnosti ako napr. meno (name), unikátny identifikátor v Kubernetes klastri (vm_id), dynamický počet diskov a ich zodpovedajúce veľkosti, sieťové nastavenia a niekoľko ďalších nastavení - podrobný popis je súčasťou dokumentácie daného repozitára.
Pokiaľ ide o premenné local.*, tie boli definované za účelom zníženia duplicitných

informácií pri vytváraní viacerých podobných virtuálnych strojov (ako tak tomu pri Kubernetes klastroch býva). Tie zahŕňajú spoločné informácie pre všetky virtuálne stroje, resp. aj rozdielne informácie na základe úlohy virtuálneho stroja v rámci Kubernetes klastra (control/worker node). Príkladom je počet pridelenej pamäte alebo virtuálnych procesorových jadier (napr. server_cpus a agent_cpus), avšak tento koncept je aplikovateľný na čokoľvek:

```
locals {
tags = ["terraform", "taloslinux"]
pool_id = "K8S"
server_cpus = 4
server_memory = 8192
agent_cpus = 4
agent_memory = 8192
network_bridge = "vlan51"
ipv4_gateway = "10.23.153.1"
dns_domain = "test.k8s.fri.uniza.sk"
dns_servers = [ "8.8.8.8" ]
iso = "iso:iso/talos-v1.9.5-nocloud-amd64-qemuagent.iso"
disk_size = 50
boot_order = ["scsi0","ide0"]
}
```

Kód: Ukážka použitia vytvoreného Terraform modulu - lokálne premenné

- Navrhnutá a vytvorená štruktúra repozitára vyzerá nasledovne:



Pozn.: Súbory umiestnené v zátvorkách - "()" - nie sú súčasťou životného cyklu Git repozitára, nakoľko obsahujú senzitívne premenné, ktoré je potrebné definovať pri inicializácii lokálneho repozitára.

• modules/ - priečinok obsahujúci moduly. Moduly sú v rámci nástroja Terraform opakovateľne použiteľné zoskupenia viacerých Terraform konfigurácii často reprezentujúce určitú časť infraštruktúry. To umožňuje konfiguráciu definovať raz (taktiež s možnosťou parametrizácie), a následne ju opakovane použiť v rôznych častiach projektu/projektov.
Práve na to bol vytvorený modul (priečinok) modules/k8s-proxmox/, ktorý týmto prístupom umožňuje vytvárať rôzne repliky Kubernetes prostredí na testovacie účely.

• k8s/ - obsahuje podpriečinky, ktoré definujú špecifické nasadenie (prostredie) Kubernetes klastra (napr. k8s/prod/ pre produkčné nasadenie alebo k8s-test/ pre testovacie nasadenie).

• k8s/<env>/ - priečinok prostredia (napr. k8s/prod/).

– backend.tf - definuje to, kde je uchovávaný aktuálny stav infraštruktúry (typicky súbor *.tfstate)

34

– provider.tf - definuje poskytovateľa (provider), v konkrétnom prípade špecifický Proxmox poskytovateľ

– variables.tf - deklaruje premenné, ktoré používateľ bude musieť definovať (pomocou súboru terraform.tfvars) po inicializácii lokálneho repozitára. Tieto premenné uchovávajú prihlasovacie údaje do Proxmox klastra, na ktorý majú byť nasadené virtuálne stroje použitého modulu.

– main.tf - hlavný terraform súbor, ktorý používa vyššie definovaný modul modules/k8s-proxmox

ArgoCD poskytuje prehľadné webové rozhranie (ale aj CLI) na zobrazenie a správu nasadených aplikácií, podporu správy viacerých klastrov naraz, taktiež vďaka čomu je aj často preferovaným nástrojom pri vývoji aplikácií. Disponuje aj množstvom doplnkových modulov poskytujúcich dodatočnú funkcionalitu a napojenie na externé systémy (ako napr. integrácia s HashiCorp Vault pre priame pridanie senzitívnych dát).
FluxCD/**ArgoCD** integrácia - Okrem vyššie spomenutých operátorov je však možné použiť aj rôzne integrácie, ktoré sú zabudované do GitOps nástrojov FluxCD a ArgoCD. ArgoCD ponúka napr. vlastný HashiCorp Vault plugin pre priamu integráciu bez dodatočného operátora [16], no je možné zapnúť aj dodatočnú podporu pre KSOPS zmenou ArgoCD konfigurácie (nutné nastaviť kustomize.buildOptions s dodatočnými prepínačmi "–enable-alpha-plugins a –enable-exec

FluxCD volí naopak vzhľadom aj na jeho architektúru viac minimalistický dizajn a prenecháva túto zodpovednosť zväčša na externé systémy - s jednou výnimkou, nakoľko disponuje natívnou a robustnou podporou pre SOPS.

- Správa a nasadzovanie Kubernetes aplikácií

Pre nasadenie dodatočných aplikácií bol ako vhodný zvolený nástroj FluxCD. Pre ten bolo potrebné vytvoriť dodatočný repozitár, ktorý bude obsahovať všetky potrebné YAML manifesty určené pre nasadenie jednotlivých aplikácií. Následne bolo potrebné vytvoriť vhodnú štruktúru, ktorá bude jednoducho rozšíriteľná o ďalšie, nové aplikácie a zároveň umožní jednoduchým a prehľadným spôsobom aplikáciu nielen nasadiť, ale ju aj zo špecifického klastra vymazať. Rovnako je dôležité aby repozitár umožnil jednoduchú rozšíriteľnosť o nové klastre bez zbytočných duplicít, na čo bol primárne využitý nástroj Kustomize. Cieľom bolo taktiež pri použití tohto nástroja oddeliť verziovanie jednotlivých inštalovaných aplikácii na úrovni klastrov - teda aby bolo možné otestovať aktualizácie jednotlivých aplikácii mimo produkčné nasadenie, a po otestovaní tak bezpečne tieto aktualizácie vykonať aj na produkčnom nasadení.

0.2 POPIS INFRAŠTRUKTÚRY DOCKER

Základ infraštruktúry tvorí hypervízor Proxmox VE nasadený priamo na fyzickom serveri (bare-metal). Proxmox využíva kombináciu technológií KVM (Kernel-based Virtual Machine) pre plnú virtualizáciu a LXC pre kontajnerizáciu na úrovni operačného systému. V danom návrhu je použitá výhradne KVM virtualizácia, čím je zabezpečená plná izolácia jednotlivých virtuálnych serverov.

Hypervízor zabezpečuje:

- správu výpočtových zdrojov (CPU, RAM),
- správu diskových úložísk (LVM-thin, ZFS alebo lokálne RAID pole),
- sieťovú virtualizáciu prostredníctvom Linux bridge rozhraní,
- centralizovanú správu a monitoring virtuálnych strojov.

Každý virtuálny server (VM) má pridelené presne definované hardvérové prostriedky, vrátane počtu virtuálnych CPU jadier, veľkosti operačnej pamäte a diskového priestoru. Konfigurácia je navrhnutá tak, aby bolo možné dynamicky škálovať zdroje bez nutnosti reinštalácie operačného systému.

- **2. Operačný systém virtuálnych serverov – AlmaLinux**

Na jednotlivých virtuálnych serveroch je nasadená distribúcia AlmaLinux, ktorá je binárne kompatibilná s Red Hat Enterprise Linux. Tento operačný systém je zvolený z dôvodu:

- vysokej stability a dlhodobej podpory,
- vhodnosti pre serverové a produkčné nasadenia,
- kompatibility s Docker Engine a modernými DevOps nástrojmi.

Základná konfigurácia AlmaLinux serverov zahŕňa:

- minimal installation (bez grafického rozhrania),
- správu balíkov pomocou DNF,
- centralizované aktualizácie bezpečnostných záplat,
- nastavenie firewallu (firewalld alebo nftables),
- zabezpečený prístup cez SSH s využitím kľúčovej autentifikácie.

Každý virtuálny server je koncipovaný ako aplikačný uzol, na ktorom beží Docker Engine a súvisiace aplikačné kontajnery.

---

3. Kontajnerizačná vrstva – Docker Engine

Na AlmaLinux virtuálnych serveroch je nainštalovaný Docker Engine, ktorý zabezpečuje izolovaný beh aplikačných komponentov vo forme kontajnerov. Docker poskytuje jednotné prostredie pre nasadenie aplikácií nezávislé od hostiteľského systému.

Docker Engine zabezpečuje:

- správu životného cyklu kontajnerov,
- sieťovanie medzi kontajnermi pomocou virtuálnych Docker sietí,
- perzistenciu dát prostredníctvom Docker volumes,
- izoláciu procesov, súborového systému a sieťových rozhraní.

Kontajnery sú navrhnuté tak, aby každý poskytoval jednu konkrétnu funkciu (napr. databáza, backendová služba, frontendová aplikácia), čím sa dodržiava princíp mikroservisnej architektúry.

- **4. Orchestrácia aplikácií – Docker Compose**

Spúšťanie a konfigurácia kontajnerov je realizovaná pomocou Docker Compose súborov (docker-compose.yml). Docker Compose umožňuje deklaratívne definovať:

- jednotlivé služby (services),
- použité Docker obrazy,
- sieťové prepojenia medzi službami,
- mapovanie portov,
- perzistentné úložiská,
- konfiguračné premenné prostredia.

Docker Compose súbory sú uložené priamo na virtuálnych serveroch a umožňujú:

- jednoduché nasadenie celého aplikačného stacku jedným príkazom,
- reprodukovateľnosť prostredia (test, staging, produkcia),
- jednoduchú údržbu a aktualizáciu služieb.

Každý aplikačný celok je oddelený do samostatného Compose projektu, čím sa minimalizuje riziko konfliktov medzi jednotlivými nástrojmi a službami.

- **5. Celková architektúra riešenia**

Celá infraštruktúra je navrhnutá ako viacvrstvová architektúra:

- fyzická vrstva (hardvér),
- virtualizačná vrstva (Proxmox VE),
- operačný systém (AlmaLinux VM),
- kontajnerizačná vrstva (Docker Engine),
- aplikačná vrstva (Docker Compose riadené služby).

Takýto prístup zabezpečuje vysokú mieru modularity, škálovateľnosti a jednoduchej správy infraštruktúry. Zároveň umožňuje efektívne oddelenie infraštruktúrnych, systémových a aplikačných zodpovedností, čo je vhodné pre moderné cloud-ready a DevOps orientované prostredia.

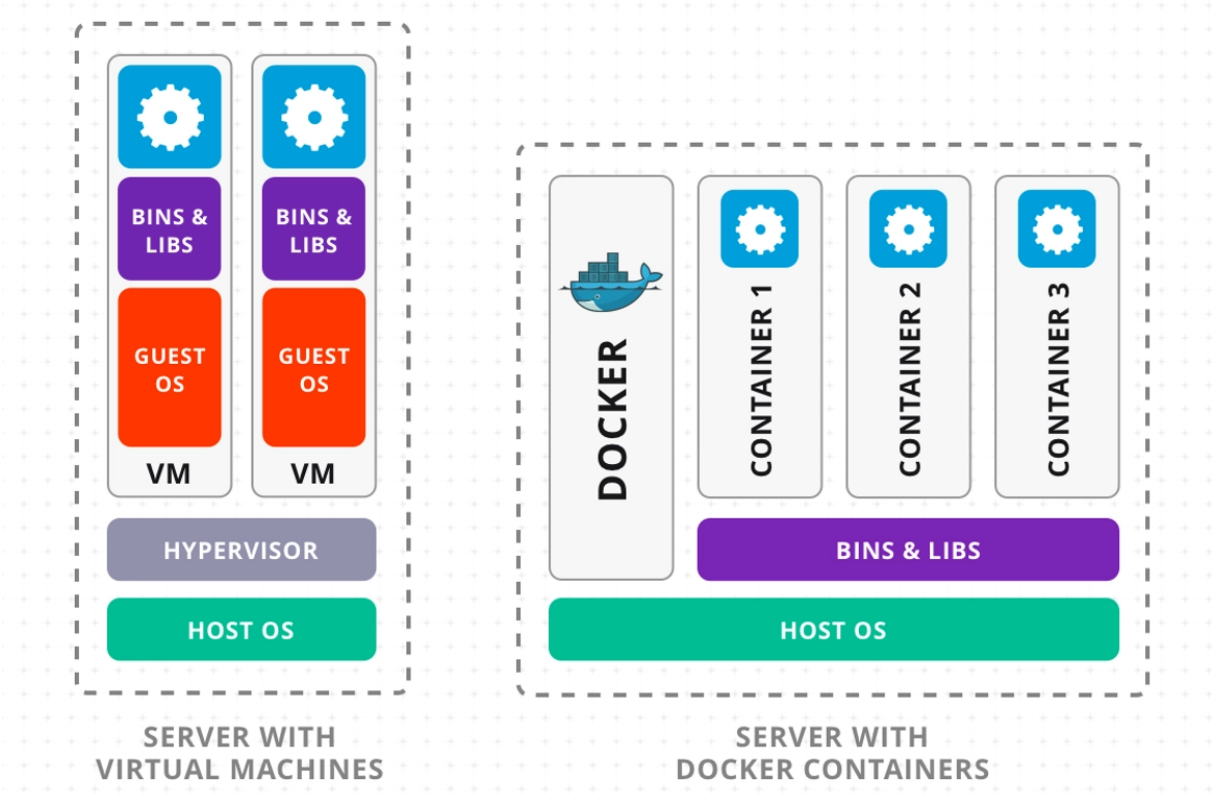


Diagram znázorňujúci topológiu docker infraštruktúry

1. ELK STACK

Typ nástroja: SIEM

Názov nástroja: ELK Stack (Elasticsearch + Logstash/Beats + Kibana)

Verzia: Release 9.2505.1

Opis systému

Centrálna agregácia, indexácia a korelácia logov + vyhľadávanie a analýza

ELK Stack poskytuje rozsiahlu podporu pre zber logov pomocou nástrojov ako Filebeat, Winlogbeat, či Auditbeat. Tieto komponenty umožňujú zber logov z rôznych typov systémov vrátane Windows, Linux, sieťových zariadení a aplikačných služieb. Na úrovni analýzy dát Logstash umožňuje normalizáciu a parsovanie údajov, pričom Elasticsearch poskytuje výkonný indexačný a dotazovací engine. Hoci samotná bezplatná verzia neposkytuje pokročilé ML nástroje, základné alerting mechanizmy je možné nastaviť cez Watcher (alebo Alerting plugin pri OpenSearch variante). Bezpečnostné funkcionality ako TLS medzi komponentmi a RBAC sú k dispozícii až v platených edíciách, čo predstavuje obmedzenie v open-source prostredí. Kibana však poskytuje intuitívne a rozšíriteľné dashboardy s množstvom vizualizačných možností a filtrami.

Dokumentácia systému

Neoficiálna docker compose dokumentácia: https://github.com/deviantony/docker-elk

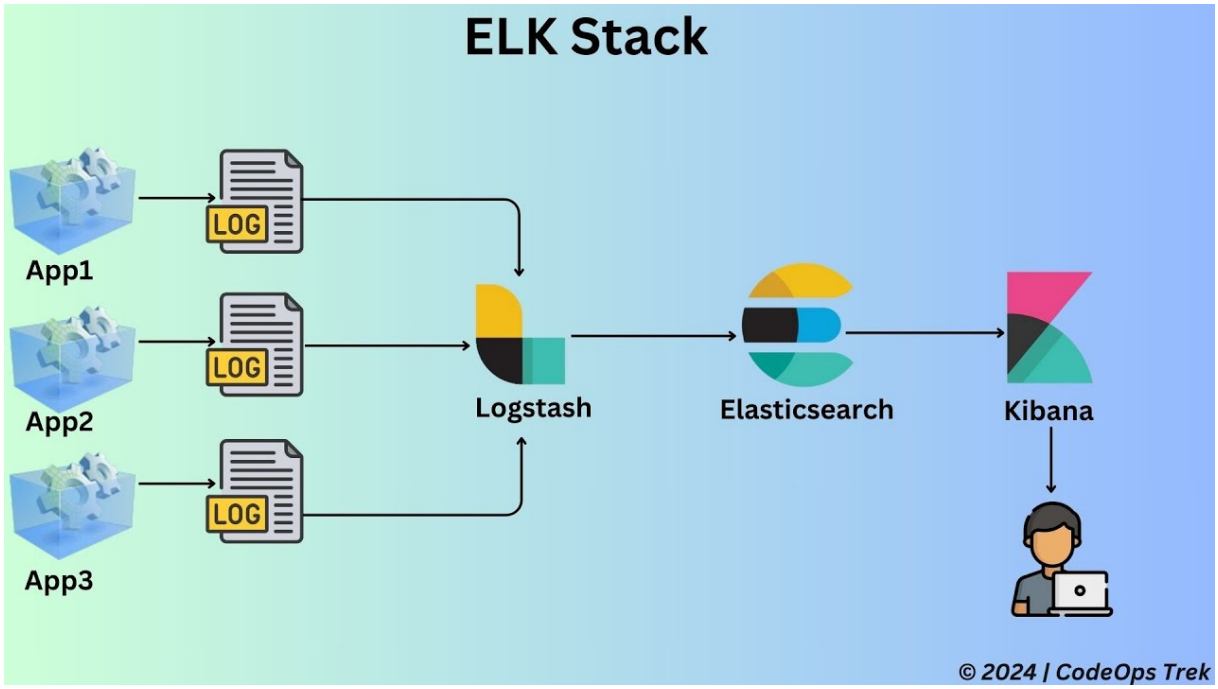
Požiadavky na HW:

8 vCPU

16GiB RAM

500GB disk (SSD, podľa nastavení retencie údajov)

Schémy



Inštalácia - variant Docker

```
      services:
        elasticsearch:
          build:
            context: elasticsearch/
            args:
              ELASTIC_VERSION: ${ELASTIC_VERSION}
          volumes:
            -
./elasticsearch/config/elasticsearch.yml:/usr/share/elasticsearch/co
nfig/elasticsearch.yml:ro,Z
            - elasticsearch:/usr/share/elasticsearch/data:Z
          ports:
            - 9200:9200
            - 9300:9300
          environment:
            node.name: elasticsearch
            ES_JAVA_OPTS: -Xms512m -Xmx512m
            # Bootstrap password.
            # Used to initialize the keystore during the initial
startup of
            # Elasticsearch. Ignored on subsequent runs.
```

```
            ELASTIC_PASSWORD: ${ELASTIC_PASSWORD:-}
            # Use single node discovery in order to disable
production mode and avoid bootstrap checks.
            # see:
https://www.elastic.co/guide/en/elasticsearch/reference/current/boot
strap-checks.html
            discovery.type: single-node
          networks:
            - elk
          restart: unless-stopped
        logstash:
          build:
            context: logstash/
            args:
              ELASTIC_VERSION: ${ELASTIC_VERSION}
          volumes:
            -
./logstash/config/logstash.yml:/usr/share/logstash/config/logstash.y
ml:ro,Z
            - ./logstash/pipeline:/usr/share/logstash/pipeline:ro,Z
          ports:
            - 5044:5044
      #       - 50000:50000/tcp
            - 514:50000/tcp
      #       - 50000:50000/udp
            - 514:50000/udp
            - 9600:9600
          environment:
            LS_JAVA_OPTS: -Xms256m -Xmx256m
            LOGSTASH_INTERNAL_PASSWORD:
${LOGSTASH_INTERNAL_PASSWORD:-}
          networks:
            - elk
          depends_on:
            - elasticsearch
          restart: unless-stopped
        kibana:
```

```
        build:
          context: kibana/
          args:
            ELASTIC_VERSION: ${ELASTIC_VERSION}
        volumes:
          -
./kibana/config/kibana.yml:/usr/share/kibana/config/kibana.yml:ro,Z
        ports:
          # - 5601:5601
          - 80:5601
        environment:
          KIBANA_SYSTEM_PASSWORD: ${KIBANA_SYSTEM_PASSWORD:-}
          SERVER_PUBLICBASEURL: http://elk.innosoc.brainit.tech
        networks:
          - elk
        depends_on:
          - elasticsearch
        restart: unless-stopped
    networks:
      elk:
        driver: bridge
    volumes:
      elasticsearch:
```

Zmeny v docker compose: Zmenené niektoré sieťové porty – viď zakomentované riadky vyššie.

Základná konfigurácia

Konfigurácia nástroja

ELK Stack, obsahuje konfiguračný skript ktorý správne nastaví adresy a potrebnú autentifikáciu medzi službami. Väčšinu konfigurácie som potom nechali v predvolených nastaveniach. Zvyšok konfigurácie potom už pozostáva len prispôsobeniu požiadaviek alebo pluginy a nastavenie retencie údajov.

Konfigurácia logstash pre syslog z Fortigate 200-F
logstash/pipeline/fortigate.conf

```
    input {
      # Listen on the CONTAINER port 50000/udp, mapped from host
port 514/udp
      # If you chose TCP on FortiGate, change 'udp' to 'tcp' here.
      udp {
        port => 50000 # <<< This is the CONTAINER port mapped in
docker-compose.yml
        codec => plain { charset => "UTF-8" }
        type => "fortigate" # Add a type for easier filtering later
        tags => ["fortigate", "udp"]
      }

      # --- OR if you chose TCP on the FortiGate ---
      # tcp {
      #   port => 50000 # <<< CONTAINER port
      #   codec => line { charset => "UTF-8" } # Use 'line' codec
for TCP
      #   type => "fortigate"
      #   tags => ["fortigate", "tcp"]
      # }
    }

    filter {
      # Only process messages tagged as fortigate AND likely
containing key-value pairs
      if "fortigate" in [tags] and [message] =~ /type=/ {

        # Parse the FortiGate key-value format
        kv {
          source => "message"
          field_split => " "
          value_split => "="
          trim_key => "\""
          trim_value => "\""
          # prefix => "fortigate_" # Optional: Uncomment to prefix
all fields
        }
```

```
          # Convert known numeric fields
          mutate {
            convert => {
              "rcvdbyte" => "integer"
              "sentbyte" => "integer"
              "duration" => "integer"
              "proto" => "integer" # Protocol number (e.g., 6 for
TCP)
              "srcport" => "integer"
              "dstport" => "integer"
              "policyid" => "integer"
              # Add other common numeric fields: devid, vd,
sessionid, count, etc.
            }
            # Remove quotes from potentially problematic fields if kv
didn't catch them
            gsub => [
              "srcip", '"', '',
              "dstip", '"', '',
              "action", '"', '',
              "service", '"', ''
              # Add others as needed
            ]
          }

          # Parse timestamp from 'date' and 'time' fields
          if [date] and [time] {
            mutate {
              add_field => { "event_timestamp_str" => "%{date}
%{time}" }
            }
            date {
              match => [ "event_timestamp_str", "yyyy-MM-dd HH:mm:ss"
] # Verify this format matches your logs!
              target => "@timestamp"
```

```
            # timezone => "Your/Fortigate/Timezone" # IMPORTANT:
Set if FortiGate isn't UTC and doesn't include offset
          }
          mutate {
            remove_field => [ "event_timestamp_str" ]
          }
        }

        # GeoIP Enrichment (Optional, requires plugin install - see
below)
        # if [srcip] {
        #   geoip {
        #     source => "srcip"
        #     target => "geoip_src"
        #     # database => "/path/inside/container/to/GeoLite2-
City.mmdb" # Needs GeoIP database mounted or built into image
        #   }
        # }
        # if [dstip] {
        #   geoip {
        #     source => "dstip"
        #     target => "geoip_dst"
        #     # database => "/path/inside/container/to/GeoLite2-
City.mmdb"
        #   }
        # }

        # Optional: Rename fields for ECS compatibility if desired
        # mutate {
        #   rename => { "srcip" => "source.ip" }
        #   rename => { "dstip" => "destination.ip" }
        #   # ... more renames
        # }

      } else {
        # Optionally drop messages that don't look like FortiGate
logs if other
```

```
          # things might arrive on this port (though dedicated port
is better)
          # drop { }
        }
      }

      output {
        elasticsearch {
          # Use the service name 'elasticsearch' as host because they
are on the same 'elk' docker network
          hosts => ["http://elasticsearch:9200"]
          index => "fortigate-%{+YYYY.MM.dd}" # Daily indices

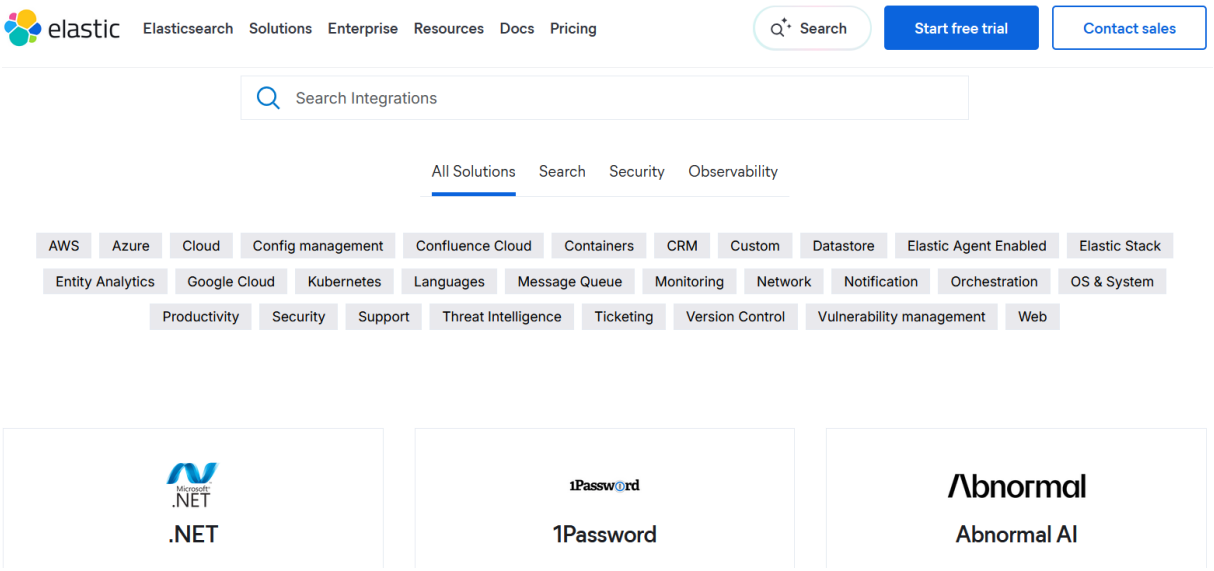
          # === Authentication ===
          # Use the internal user and password defined in your .env
file and passed to the container
          user => "logstash_internal"
          password => "${LOGSTASH_INTERNAL_PASSWORD}" # Logstash
reads this from its environment

          # === Index Lifecycle Management (Recommended for
Production) ===
          # Setup an ILM policy in Kibana first (e.g., named
'fortigate_policy')
          # ilm_enabled => true
          # ilm_rollover_alias => "fortigate"
          # ilm_policy => "fortigate_policy" # Make sure this policy
exists in Elasticsearch/Kibana
          # ilm_pattern => "000001"
        }

        # Optional: Output to container logs for debugging during
setup
        # Remove or comment out for production.
        # stdout {
        #   codec => rubydebug
        # }
      }
```

Integrácie

Elastic má obrovské množstvo a možnosti integrácií:



https://www.elastic.co/integrations/data-integrations

V stacku je potom taktiež logstash. Logstash je po správnej konfigurácií toho ako má spracovávať konkrétne syslog logy možné integrovať s akýmkoľvek systémom ktorý odosiela syslog. Príkladom takejto integrácie je jednoduchá integrácia ktorú sme si napísali pre prijímanie logov z firewallu Fortigate.

2. SHUFFLE

Typ nástroja: SOAR

Názov nástroja: Shuffle

Verzia: Release 2.1.0

Opis systému

Orchestrácia a automatizácia reakcií (playbooky)

Shuffle je ľahký open-source SOAR nástroj s moderným webovým rozhraním, ktorý umožňuje vizuálnu tvorbu workflowov. Je možné prepojenie na API rôznych nástrojov (Elastic, Graylog, VirusTotal, Slack, Jira, atď.). Výhodou je jednoduchosť nasadenia (Docker) a flexibilita pri budovaní vlastných akcií. Nepodporuje však taký rozsah IR funkcionality ako TheHive a nemá nativne pripravený incident manažment model.

Shuffle SOAR vyniká jednoduchosťou tvorby a správy playbookov. Umožňuje flexibilnú integráciu pomocou REST API a konektorov. Na druhej strane mu chýba hlbší incident manažment, ale je výborný ako doplnkový orchestrátor pre rýchle workflowy.

Dokumentácia systému

Officialna dokumentácia:

- https://shuffler.io/docs/about
- https://github.com/shuffle/shuffle

3. WAZUH

Typ nástroja: EDR

Názov nástroja: Wazuh

Verzia: Version 4.14

Opis systému

Endpoint monitoring/detekcie a základná reakcia + posielanie udalostí do SIEM

Wazuh integruje komponenty pre SIEM, EDR a FIM (File Integrity Monitoring). Zber dát je možný prostredníctvom agentov, ktorí podporujú logy z operačných systémov, súborov, auditných mechanizmov a aplikačných serverov. Okrem toho dokáže prijímať logy zo Syslog, čo umožňuje centralizovaný zber údajov aj z pasívnych zariadení bez agenta. Wazuh poskytuje vlastný korelačný engine, s množstvom pravidiel založených na MITRE ATT&CK. Tieto pravidlá sú pravidelne aktualizované komunitou. Webové UI podporuje RBAC, správu úloh a prepojenie s Kibana dashboardmi. Jeho hlavnou silou je kombinácia EDR vlastností s prehľadným SIEM, čo ho robí výhodným riešením pre menšie tímy.

Dokumentácia systému

Officialna dokumentácia:

- https://documentation.wazuh.com/current/installation-guide/index.html
- https://hub.docker.com/u/wazuh

Požiadavky na HW:

4 vCPU, 8 GB RAM, 100+ GB Disk

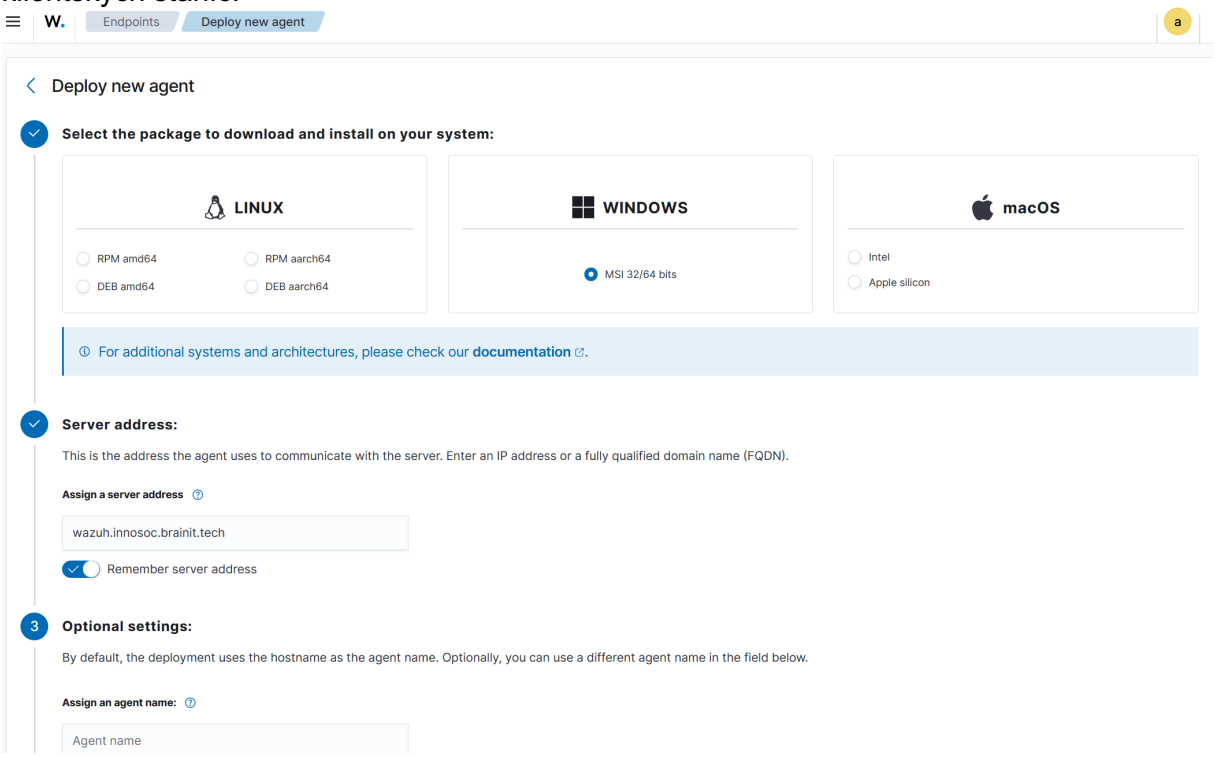
Inštalácia - variant Quickstart

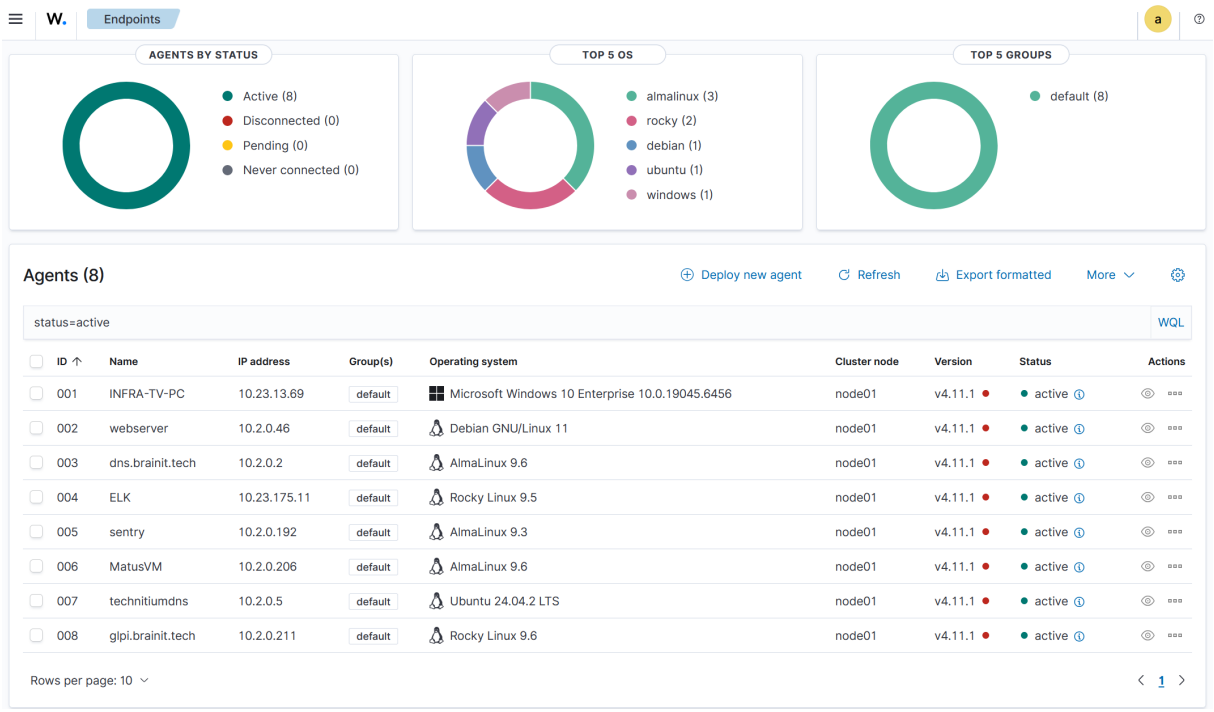
Nainštalovaný na rocky linux 9 VM podľa oficiálnej quickstart dokumentácie
https://documentation.wazuh.com/current/quickstart.html

Základná konfigurácia

Konfigurácia nástroja:

Wazuh má vstavaný systém na poskytovanie inštalátora wazuh agent a onboarding klientskych staníc:





Integrácie

Wazuh je možné integrovať s rôznymi SIEM systémami vrátane elastic search, bol preto integrovaný do ELK stacku: https://documentation.wazuh.com/current/integrations-guide/index.html

https://documentation.wazuh.com/current/integrations-guide/elastic-stack/index.html

4. GREENBONE COMMUNITY EDITION

Typ nástroja: Skener zraniteľností

Názov nástroja: Greenbone Community Edition (GVM) / OpenVAS

Verzia: Version 24.10

Opis systému

Pravidelné skenovanie infra/servisov a reporting zraniteľností

GCE je bezplatná open-source verzia riešenia od Greenbone Networks, založená na nástroji OpenVAS. Poskytuje vysokokapacitné skenovanie, rozsiahle databázy zraniteľností cez komunitný GVMD feed, moderné webové rozhranie GSA a podporu pre plánovanie, alerty aj automatizáciu. Podporuje štandardy ako CVE, CPE, a umožňuje generovanie reportov vo formátoch CSV, XML, PDF. Možnosti integrácie zahŕňajú GMP protokol, SSH, LDAP, RADIUS a iné. GCE umožňuje aj master-senzor architektúru, čo je výhodné pre rozsiahle distribuované prostredia.

Dokumentácia systému

Officialna dokumentácia:

- https://greenbone.github.io/docs/latest/
- https://hub.docker.com/u/greenbone

Požiadavky na HW:

4 vCPU, 8 GB RAM, 60 GB Disk

Schémy

Inštalácia - variant Docker

```
services:
  vulnerability-tests:
    image: registry.community.greenbone.net/community/vulnerability-
tests
    environment:
      FEED_RELEASE: "24.10"
      KEEP_ALIVE: 1
    volumes:
      - vt_data_vol:/mnt

  notus-data:
    image: registry.community.greenbone.net/community/notus-data
    environment:
      KEEP_ALIVE: 1
    volumes:
      - notus_data_vol:/mnt

  scap-data:
    image: registry.community.greenbone.net/community/scap-data
    environment:
      KEEP_ALIVE: 1
    volumes:
      - scap_data_vol:/mnt
```

```
cert-bund-data:
  image: registry.community.greenbone.net/community/cert-bund-data
  environment:
    KEEP_ALIVE: 1
  volumes:
    - cert_data_vol:/mnt

dfn-cert-data:
  image: registry.community.greenbone.net/community/dfn-cert-data
  environment:
    KEEP_ALIVE: 1
  volumes:
    - cert_data_vol:/mnt
  depends_on:
    cert-bund-data:
      condition: service_healthy

data-objects:
  image: registry.community.greenbone.net/community/data-objects
  environment:
    FEED_RELEASE: "24.10"
    KEEP_ALIVE: 1
  volumes:
    - data_objects_vol:/mnt

report-formats:
  image: registry.community.greenbone.net/community/report-formats
  environment:
    FEED_RELEASE: "24.10"
    KEEP_ALIVE: 1
  volumes:
    - data_objects_vol:/mnt
  depends_on:
    data-objects:
      condition: service_healthy
```

```
gpg-data:
  image: registry.community.greenbone.net/community/gpg-data
  volumes:
    - gpg_data_vol:/mnt

redis-server:
  image: registry.community.greenbone.net/community/redis-server
  restart: on-failure
  volumes:
    - redis_socket_vol:/run/redis/

pg-gvm:
  image: registry.community.greenbone.net/community/pg-gvm:stable
  restart: on-failure
  volumes:
    - psql_data_vol:/var/lib/postgresql
    - psql_socket_vol:/var/run/postgresql

gvmd:
  image: registry.community.greenbone.net/community/gvmd:stable
  restart: on-failure
  volumes:
    - gvmd_data_vol:/var/lib/gvm
    - scap_data_vol:/var/lib/gvm/scap-data/
    - cert_data_vol:/var/lib/gvm/cert-data
    - data_objects_vol:/var/lib/gvm/data-objects/gvmd
    - vt_data_vol:/var/lib/openvas/plugins
    - psql_data_vol:/var/lib/postgresql
    - gvmd_socket_vol:/run/gvmd
    - ospd_openvas_socket_vol:/run/ospd
    - psql_socket_vol:/var/run/postgresql
  depends_on:
    pg-gvm:
      condition: service_started
    scap-data:
```

```
        condition: service_healthy
      cert-bund-data:
        condition: service_healthy
      dfn-cert-data:
        condition: service_healthy
      data-objects:
        condition: service_healthy
      report-formats:
        condition: service_healthy

  gsa:
    image: registry.community.greenbone.net/community/gsa:stable
    restart: on-failure
    ports:
      - 127.0.0.1:9392:80
    volumes:
      - gvmd_socket_vol:/run/gvmd
    depends_on:
      gvmd:
        condition: service_started
  # Sets log level of openvas to the set LOG_LEVEL within the env
  # and changes log output to /var/log/openvas instead /var/log/gvm
  # to reduce likelyhood of unwanted log interferences
  configure-openvas:
    image: registry.community.greenbone.net/community/openvas-
scanner:stable
    volumes:
      - openvas_data_vol:/mnt
      - openvas_log_data_vol:/var/log/openvas
    command:
      - /bin/sh
      - -c
      - |
        printf "table_driven_lsc = yes\nopenvasd_server =
http://openvasd:80\n" > /mnt/openvas.conf
```

```
        sed "s/127/128/" /etc/openvas/openvas_log.conf | sed
's/gvm/openvas/' > /mnt/openvas_log.conf
        chmod 644 /mnt/openvas.conf
        chmod 644 /mnt/openvas_log.conf
        touch /var/log/openvas/openvas.log
        chmod 666 /var/log/openvas/openvas.log


  # shows logs of openvas
  openvas:
    image: registry.community.greenbone.net/community/openvas-
scanner:stable
    restart: on-failure
    volumes:
      - openvas_data_vol:/etc/openvas
      - openvas_log_data_vol:/var/log/openvas
    command:
      - /bin/sh
      - -c
      - |
        cat /etc/openvas/openvas.conf
        tail -f /var/log/openvas/openvas.log
    depends_on:
      configure-openvas:
        condition: service_completed_successfully


  openvasd:
    image: registry.community.greenbone.net/community/openvas-
scanner:stable
    restart: on-failure
    environment:
      # `service_notus` is set to disable everything but notus,
      # if you want to utilize openvasd directly, remove
`OPENVASD_MODE`
      OPENVASD_MODE: service_notus
      GNUPGHOME: /etc/openvas/gnupg
      LISTENING: 0.0.0.0:80
    volumes:
```

```
      - openvas_data_vol:/etc/openvas
      - openvas_log_data_vol:/var/log/openvas
      - gpg_data_vol:/etc/openvas/gnupg
      - notus_data_vol:/var/lib/notus
    # enable port forwarding when you want to use the http api from
your host machine
    # ports:
    #   - 127.0.0.1:3000:80
    depends_on:
      vulnerability-tests:
        condition: service_healthy
      notus-data:
        condition: service_healthy
      configure-openvas:
        condition: service_completed_successfully
      gpg-data:
        condition: service_completed_successfully
    networks:
      default:
        aliases:
          - openvasd


  ospd-openvas:
    image: registry.community.greenbone.net/community/ospd-
openvas:stable
    restart: on-failure
    hostname: ospd-openvas.local
    cap_add:
      - NET_ADMIN # for capturing packages in promiscuous mode
      - NET_RAW # for raw sockets e.g. used for the boreas alive
detection
    security_opt:
      - seccomp=unconfined
      - apparmor=unconfined
    command:
      [
```

```
      "ospd-openvas",
      "-f",
      "--config",
      "/etc/gvm/ospd-openvas.conf",
      "--notus-feed-dir",
      "/var/lib/notus/advisories",
      "-m",
      "666",
    ]
  volumes:
    - gpg_data_vol:/etc/openvas/gnupg
    - vt_data_vol:/var/lib/openvas/plugins
    - notus_data_vol:/var/lib/notus
    - ospd_openvas_socket_vol:/run/ospd
    - redis_socket_vol:/run/redis/
    - openvas_data_vol:/etc/openvas/
    - openvas_log_data_vol:/var/log/openvas
  depends_on:
    redis-server:
      condition: service_started
    gpg-data:
      condition: service_completed_successfully
    configure-openvas:
      condition: service_completed_successfully
    vulnerability-tests:
      condition: service_healthy
    notus-data:
      condition: service_healthy

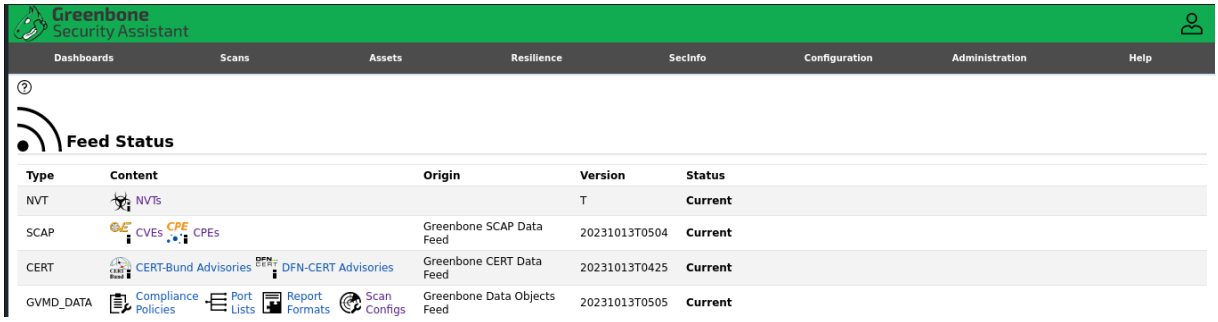
gvm-tools:
  image: registry.community.greenbone.net/community/gvm-tools
  volumes:
    - gvmd_socket_vol:/run/gvmd
    - ospd_openvas_socket_vol:/run/ospd
  depends_on:
```

```
    - gvmd
    - ospd-openvas

volumes:
  gpg_data_vol:
  scap_data_vol:
  cert_data_vol:
  data_objects_vol:
  gvmd_data_vol:
  psql_data_vol:
  vt_data_vol:
  notus_data_vol:
  psql_socket_vol:
  gvmd_socket_vol:
  ospd_openvas_socket_vol:
  redis_socket_vol:
  openvas_data_vol:
  openvas_log_data_vol:
```

Základná konfigurácia

Konfigurácia nástroja:
Len aktualizácia databáz CVE, pridanie hostov.



5. OPENCTI

Typ nástroja: CTI

Názov nástroja: OpenCTI

Verzia: Version 6.9.0

Opis systému

Správa a korelácia threat intelligence, obohacovanie alertov/incidentov

Dokumentácia systému

Officialna dokumentácia:

- https://docs.opencti.io/latest/
- https://github.com/OpenCTI-Platform/opencti
- https://hub.docker.com/u/opencti

6. MISP

Typ nástroja: CTI

Názov nástroja: MISP

Verzia:

Opis systému

Evidencia, zdieľanie a distribúcia IOCs/events, feedy pre detekcie

Dokumentácia systému

Officialna dokumentácia:

- https://www.misp-project.org/documentation/
- https://misp.github.io/MISP/

7. GLPI

Typ nástroja: Asset management

Názov nástroja: GLPI

Verzia: 10.0.16

Opis systému

Evidencia aktív, CMDB/ITSM evidencia, väzba na vlastníkov a služby

Dokumentácia systému

Officialna dokumentácia:

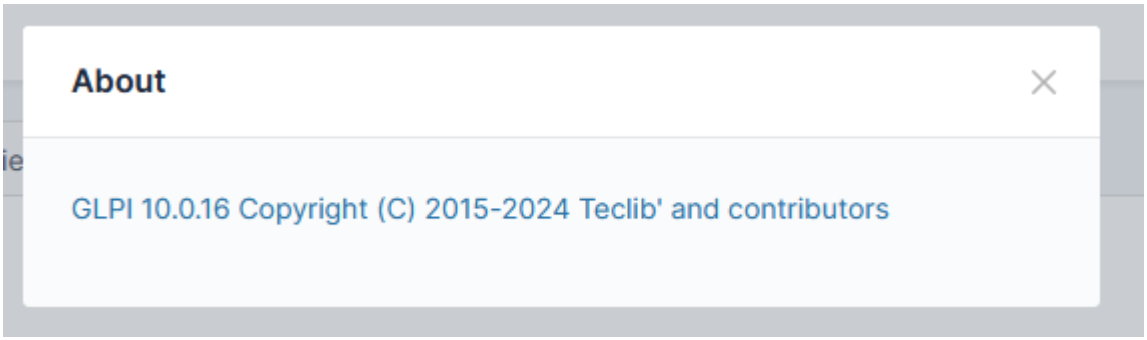
- https://glpi-plugins.readthedocs.io/en/latest/
- https://hub.docker.com/r/glpi/glpi

Požiadavky na HW:

2 vCPU, 4 GB RAM, 30 GB Disk

Schémy (N/A)

Inštalácia - variant Docker (Compose)



Verzia: GLPI 10.0.16

Docker-compose.yml (zhodný s oficiálnou dokumentáciou, pridaný NGINX ako HTTPS reverse proxy pre SSL/TLS)

```
services:
  glpi:
    image: "glpi/glpi:latest"
    restart: "unless-stopped"
    volumes:
      - "./storage/glpi:/var/glpi:rw"
    env_file: .env # Pass environment variables from .env file to
the container
    depends_on:
      db:
        condition: service_healthy
    ports:
      - "80:80"
  db:
    image: "mysql"
```

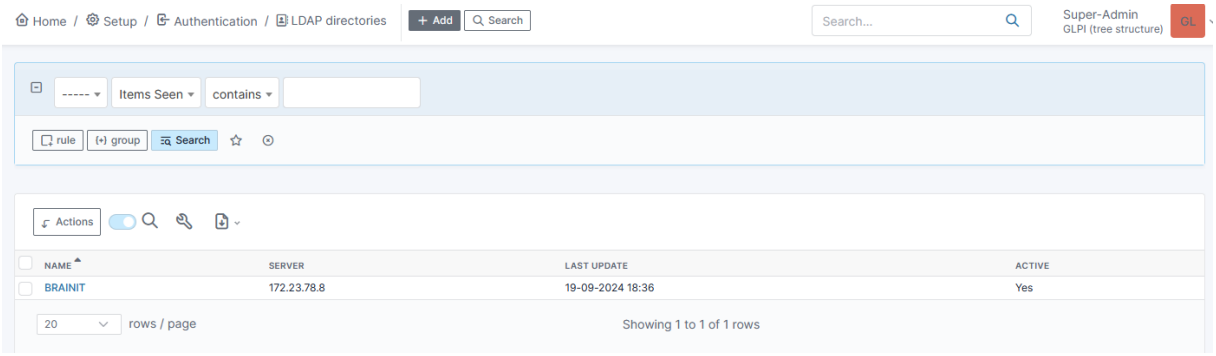
```
    restart: "unless-stopped"
    volumes:
       - "./storage/mysql:/var/lib/mysql"
    environment:
      MYSQL_RANDOM_ROOT_PASSWORD: "yes"
      MYSQL_DATABASE: ${GLPI_DB_NAME}
      MYSQL_USER: ${GLPI_DB_USER}
      MYSQL_PASSWORD: ${GLPI_DB_PASSWORD}
    healthcheck:
      test: mysqladmin ping -h 127.0.0.1 -u $$MYSQL_USER --
password=$$MYSQL_PASSWORD
      start_period: 5s
      interval: 5s
      timeout: 5s
      retries: 10
    expose:
      - "3306"
  nginx-https-proxy:
    image: nginx:latest
    restart: always
    ports:
      - "443:443"
    volumes:
      - ./storage/nginx/conf.d/:/etc/nginx/conf.d/:ro
      - ./storage/nginx/certs/:/etc/nginx/certs/:ro
    environment:
      - TIMEZONE=Europe/Bratislava
      - NGINX_HOST=glpi.brainit.tech
      - NGINX_PORT=443
```

Súbor .env

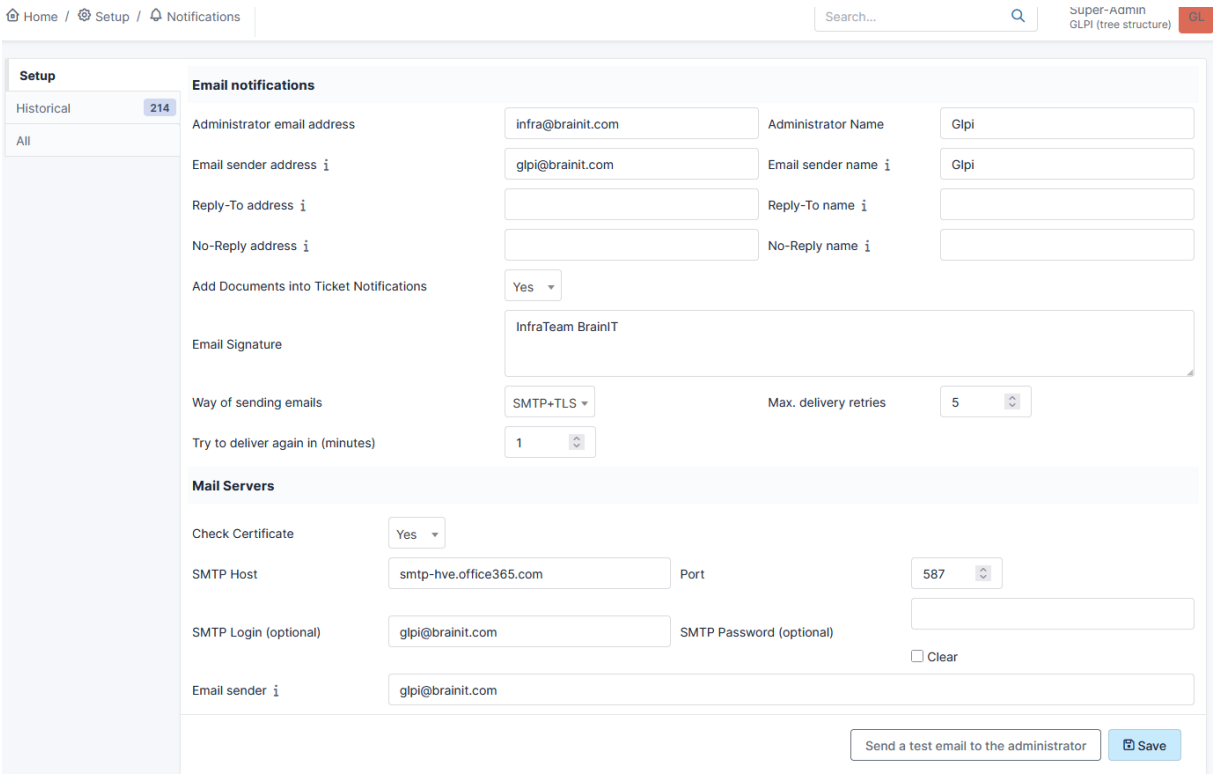
```
GLPI_DB_HOST=db
GLPI_DB_PORT=3306
GLPI_DB_NAME=glpi
GLPI_DB_USER=glpi
```

GLPI_DB_PASSWORD=glpi

Základná konfigurácia

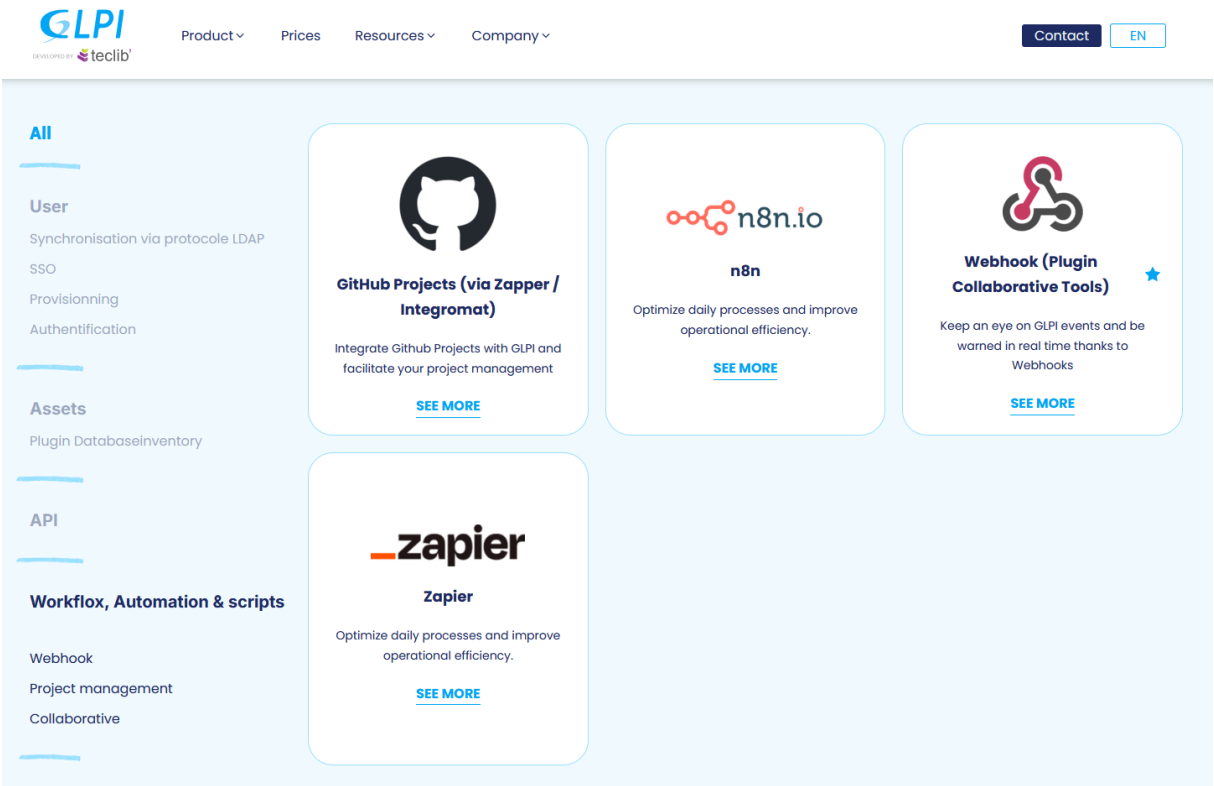


Autentifikácia cez AD/LDAP pre doménu brainit.tech

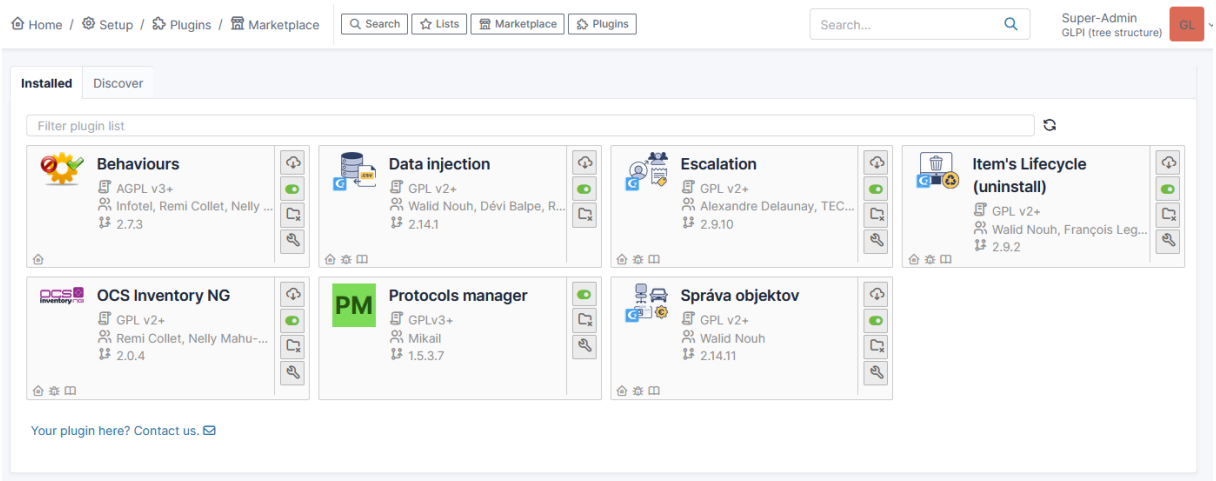


Nastavenie emailových notifikácií ako glpi@brainit.com

Integrácie



Viac o možnostiach integrácií na: https://www.glpi-project.org/en/integrations/



Zoznam pluginov v našej inštancii:

https://github.com/InfotelGLPI/behaviors - pridanie povinných polí, vynucovanie formátovania údajov, ...

https://plugins.glpi-project.org/#/plugin/datainjection – import dát vo formáte CSV

https://github.com/pluginsGLPI/escalade – ticket management a eskalácia

https://github.com/pluginsglpi/uninstall – pridanie inventárnych stavov pre vyradzovanie

https://github.com/pluginsGLPI/genericobject – pridávanie vlastných typov objektov

https://github.com/pluginsGLPI/ocsinventoryng – pre synchronizáciu s „OCS Inventory NG“

8. NETBOX

Typ nástroja: Asset management / IPAM/DCIM

Názov nástroja: NetBOX

Verzia:

Opis systému

Zdroj pravdy pre IP rozsahy, zariadenia, lokality, topológiu (podklad pre skeny a monitoring)

Dokumentácia systému

Officialna dokumentácia:

https://github.com/netbox-community/netbox

9. DFIR IRIS – TICKETING

Typ nástroja: Ticketing

Názov nástroja: DFIR IRIS

Verzia:

Opis systému

Správa ticketov a úloh SOC (triage, eskalácie, SLA)

Dokumentácia systému

Officialna dokumentácia:

- https://docs.dfir-iris.org/latest/

10. DFIR IRIS – EVIDENCIA, SPRÁVA INCIDENTOV

Typ nástroja: Evidencia, správa incidentov

Názov nástroja: DFIR IRIS

Verzia:

Opis systému

Správa ticketov a úloh SOC (triage, eskalácie, SLA)

Dokumentácia systému

Officialna dokumentácia:

- https://docs.dfir-iris.org/latest/

11. KIBANA (ELK)

Typ nástroja: Reportovanie

Názov nástroja: Kibana (ELK)

Verzia: 9.2.3

Opis nástroja

Vizualizácie, dashboardy a reporty pre SOC a management

Dokumentácia systému

Officialna dokumentácia:

- https://github.com/kibana-community/kibana3

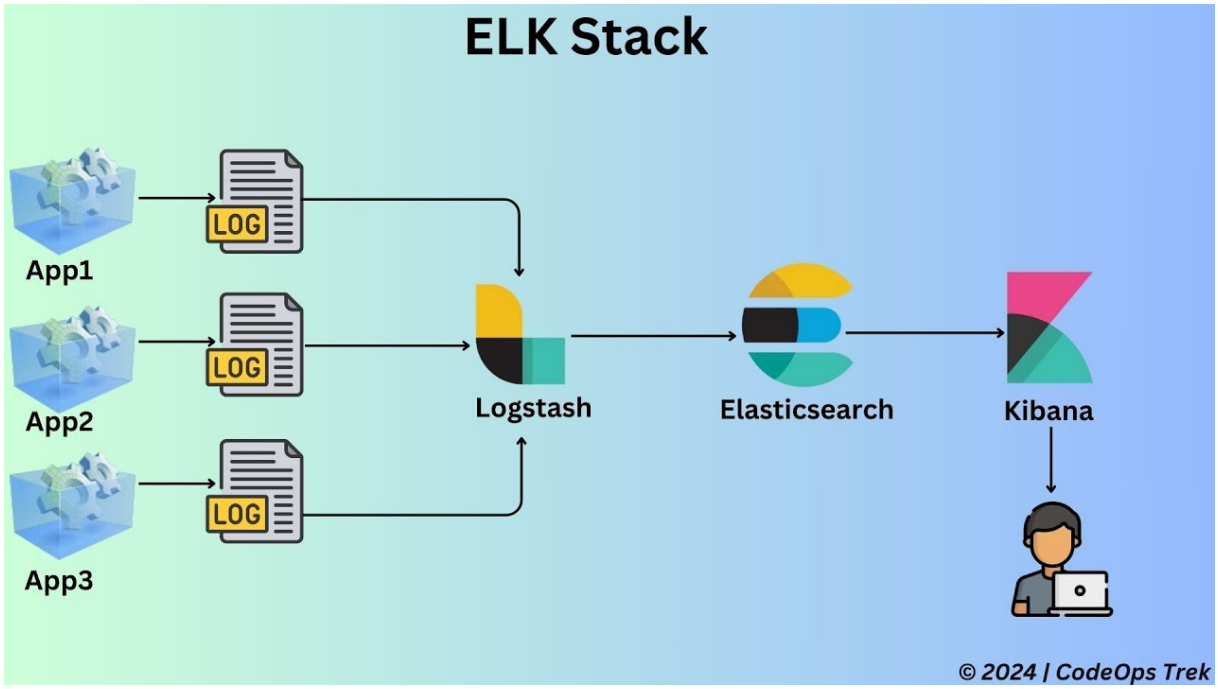
Neoficiálna docker compose dokumentácia:

https://github.com/deviantony/docker-elk

Požiadavky na HW

2 vCPU, 8 GB RAM, 50 GB Disk

Schémy



Inštalacia - variant Docker

```
services:
  kibana:
    build:
      context: kibana/
      args:
        ELASTIC_VERSION: ${ELASTIC_VERSION}
    volumes:
      -
./kibana/config/kibana.yml:/usr/share/kibana/config/kibana.yml:ro,Z
    ports:
      # - 5601:5601
      - 80:5601
    environment:
      KIBANA_SYSTEM_PASSWORD: ${KIBANA_SYSTEM_PASSWORD:-}
      SERVER_PUBLICBASEURL: http://elk.innosoc.brainit.tech
    networks:
      - elk
    depends_on:
```

```
      - elasticsearch
    restart: unless-stopped
networks:
  elk:
    driver: bridge
volumes:
  elasticsearch:
```

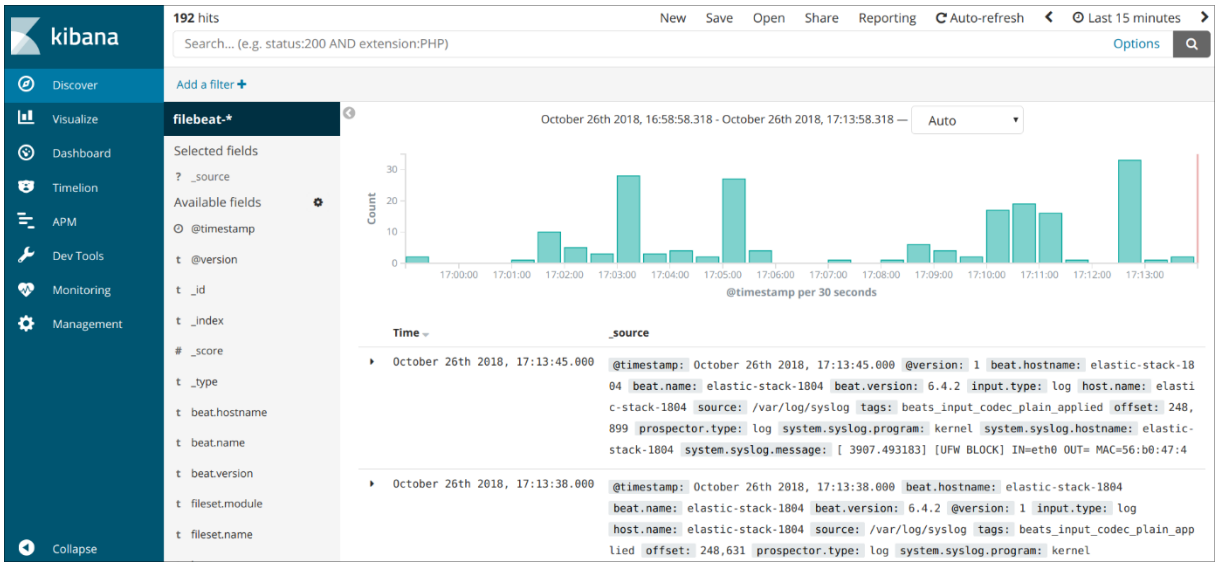
Zmeny v docker compose:

- Toto je samotná Kibana extrahovaná z celého ELK stacku (1. ELK Stack)
- Zmenené niektoré sieťové porty – viď zakomentované riadky vyššie.

Základná konfigurácia

Konfigurácia nástroja:

V prípade že je súčasťou ELK Stacku, obsahuje konfiguračný skript ktorý správne nastaví adresy a potrebnú autentifikáciu medzi službami. V opačnom prípade je len potrebné nastaviť adresy a autentifikáciu manuálne podľa oficiálnej dokumentácie.



Integrácie

Kibana je len frontend ktorý sa spravidla integruje najmä len s elasticsearch. Elasticsearch má potom ďalšie široké možnosti ďalšej integrácie.

12. WIKI.JS

Typ nástroja: Znalostná databáza

Názov nástroja: Wiki.js

Verzia: 2.5

Opis systému

Znalostná báza (runbooky, postupy, playbooky, onboarding)

Slúži pre dokumentáciu interných systémov a postupov.

Dokumentácia systému

Officialna dokumentácia:

- https://docs.requarks.io
- https://github.com/requarks/wiki
- https://hub.docker.com/r/requarks/wiki

Požiadavky na HW:

CPU: 2 jadrá

RAM: 1GB

Storage: 1GB

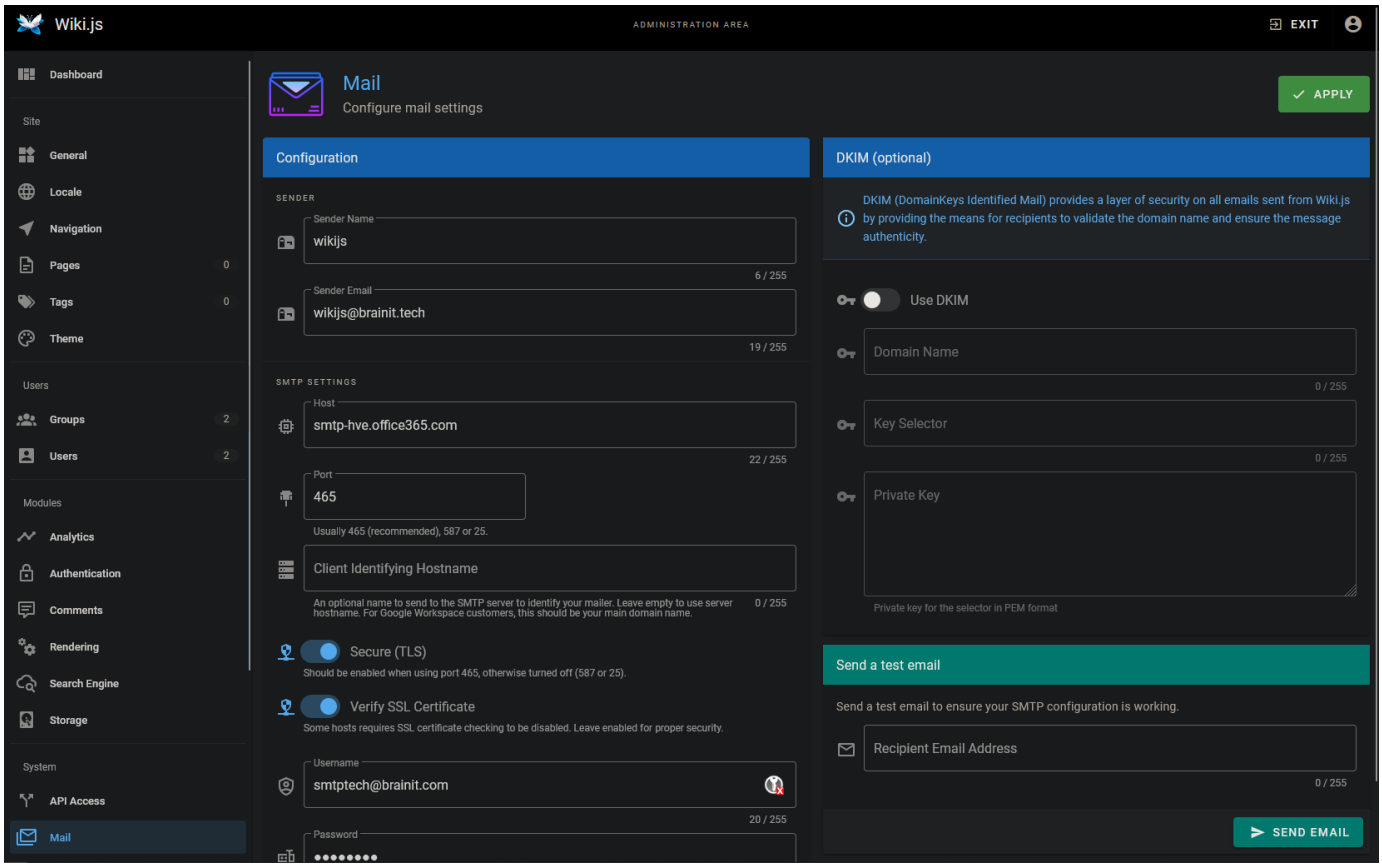
Inštalácia - variant Docker

Docker-compose.yml

```
services:
  db:
    image: postgres:15-alpine
    environment:
      POSTGRES_DB: wiki
      POSTGRES_PASSWORD: wikijsrocks
      POSTGRES_USER: wikijs
    logging:
      driver: none
    restart: unless-stopped
    volumes:
      - db-data:/var/lib/postgresql/data
  wiki:
```

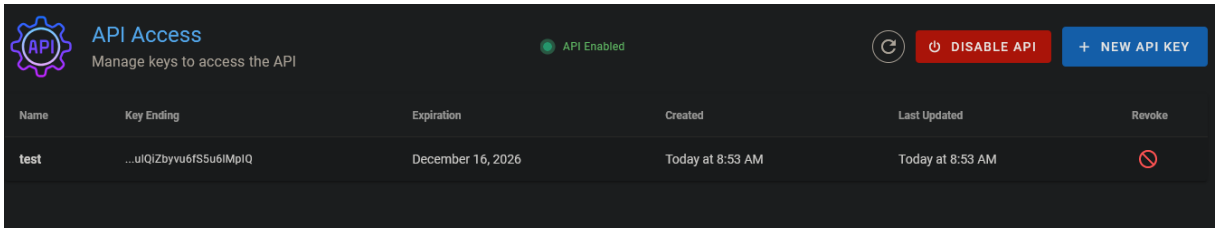
```
    image: ghcr.io/requarks/wiki:2
    depends_on:
      - db
    environment:
      DB_TYPE: postgres
      DB_HOST: db
      DB_PORT: 5432
      DB_USER: wikijs
      DB_PASS: wikijsrocks
      DB_NAME: wiki
    restart: unless-stopped
    ports:
      - "80:3000"
volumes:
  db-data:
```

Základná konfigurácia

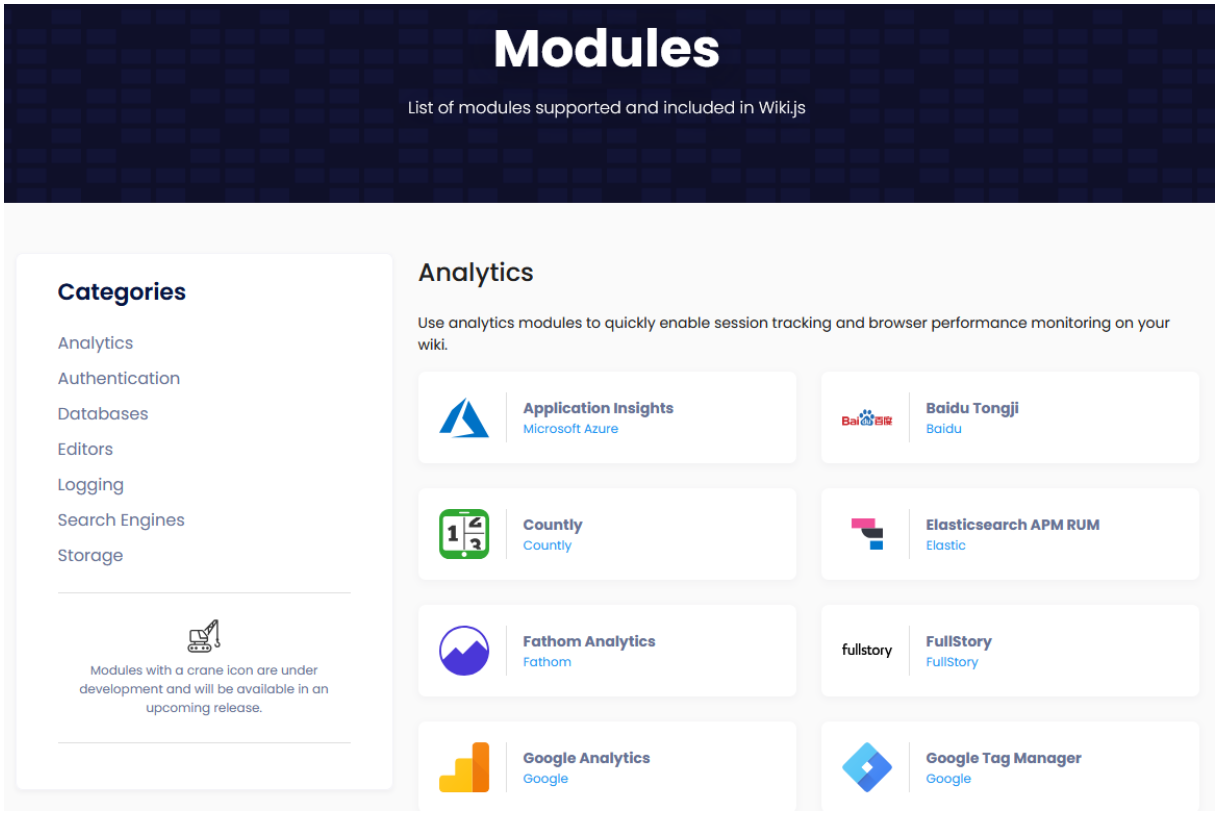


Konfigurácia SMTP

Integrácie



Možnosť integrácie cez API kľúče



Iné možnosti integrácií... https://js.wiki/modules