PROJECT

Multi-tenant cybersecurity operations centre designed
as an open cloud service with machine learning elements

V14 functional infrastructure

KPB5 – Implementation of the solution

type – documentation

month of delivery – M19

Funded by the European Union
NextGenerationEU

RECOVERY AND RESILIENCE PLAN

GOVERNMENT OFFICE OF THE SLOVAK REPUBLIC

Contents

1. ELK Stack 14
System description 14
System documentation 14
Installation - Docker variant 15
Basic configuration 17
Integrations 21
2. Shuffle 22
System description 22
System documentation 22
3. Wazuh 23
System description 23
System documentation 23
Installation - Quickstart variant 23
Basic configuration 23
Integrations 24
4. Greenbone Community edition 25
System description 25
System documentation 25
Installation - Docker variant 25
Basic configuration 32
5. OpenCTI 33
System description 33
System documentation 33
6. MISP 33
System description 33
System documentation 33
7. GLPI 34
System description 34
System documentation 34
Installation - Docker (Compose) variant 34
Basic configuration 36
Integrations 37
8. NetBox 38
System description 38
System documentation 38
9. DFIR IRIS – Ticketing 39

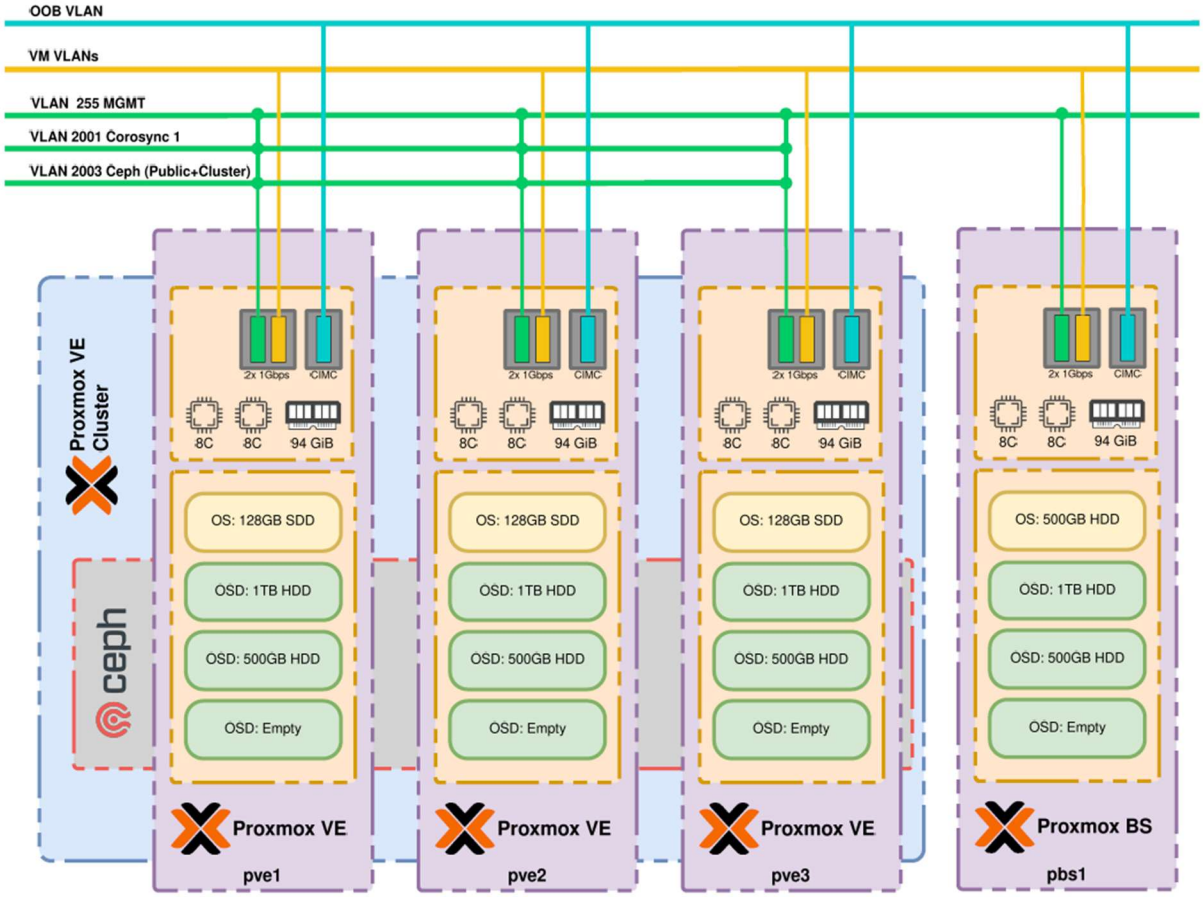
System description 39
System documentation 39
10. DFIR IRIS – Recording and management of incidents 39
System description 39
System documentation 39
11. Kibana (ELK) 39
Tool description 40
System documentation 40
Installation - Docker variant 40
Basic configuration 41
Integrations 42
12. Wiki.js 42
System description 42
System documentation 42
Installation - Docker variant 43
Basic configuration 44
Integrations 44

0.1 DESCRIPTION OF THE KUBERNETES INFRASTRUCTURE

Kubernetes is an open-source platform designed for the automated deployment, scaling and management of containerized applications – which is also why, in practice, it is most often referred to as an application orchestrator. Unlike more traditional ways of deploying containers within a single compute server – most often using the Docker tool – Kubernetes groups several such compute nodes under a single abstract layer. These nodes form a cluster that runs applications not only with high availability but also, for example, with dynamic scaling based on the current load of the given application by creating multiple replicas. Even if the user decides not to use the functionality of multiple replicas of the same application, Kubernetes enables rapid recovery of the application even if any node of the cluster fails – in that case, the controllers of the Kubernetes node responsible for the individual parts of the deployment ensure that the individual applications are restarted on the cluster nodes that are still available. As a result, Kubernetes extends the capabilities of containerization daemons with more comprehensive management of the container infrastructure.
Kubernetes uses a declarative model to define and abstract running components – it provides this functionality by means of objects of various types (in Kubernetes terminology also \code{kind}), where each type has a clearly predefined structure. Typically, these objects are defined in files in \ac{YAML} format, which the user can then apply to the cluster using the command \code{kubectl apply --filename file.yaml}.

Proxmox Virtual Environment was chosen as the hypervisor. This required several tasks to be completed:
• updating the firmware and drivers on all physical servers and configuring Out-of-band (OOB) management for remote server administration.
• installing the PVE platform on each of the selected servers
• basic configuration of the network connectivity of the nodes and preparation of the individual subnets
for the required services:
– Subnet intended for the management of the PVE servers.
– Corosync subnet (technology used for clustering the PVE servers)
– Ceph subnet (technology responsible for the storage for the Kubernetes cluster
– Subnets intended for virtual machines (e.g. Kubernetes nodes)
• Creating a PVE cluster from the created and configured servers.
• Creating a Ceph cluster within the PVE nodes and deploying the necessary daemons. These include monitoring services (Monitors), management services (Managers) and the individual storage daemons (Object Storage Daemon (OSD)), where each instance of OSD is typically responsible for storing data on a specific assigned disk.
• Deploying a backup system in the form of Proxmox Backup Server (PBS) on a dedicated server. PBS is a software solution designed for efficient and incremental backup of virtual machines and containers running in the PVE environment and provides native integration with the PVE system [17], which is why it was the natural choice when selecting the backup system.

The completion of all these tasks led to the creation of the following fully functional infrastructure
of hypervisors, which could be used in the next steps to implement
the Kubernetes cluster itself:



➢ Virtual machine infrastructure - IaaC

For the deployment of the virtual infrastructure of machines responsible for running the cluster,
the selected tool Terraform from HashiCorp, which makes it possible to declaratively define the desired state of the infrastructure
using configuration files in the HashiCorp Configuration Language (HCL) format.
Terraform supports a wide range of providers (terminologically and in configuration available as provider), on which it is able to deploy the desired objects (in this case primarily virtual machines). This gives the created solution simple
extensibility in the future in the event that a decision is made to migrate from Proxmox to another
hypervisor, or to extend the cluster in the form of additional virtual machines,
which need not necessarily be part of the same hypervisor cluster. This also opens up possibilities for creating test clusters even outside such a default solution.
For working with the Terraform tool, a repository was created in which, as part of
the work, a custom module (in Terraform terminology a module) was created for the simple creation of multiple replicas in the form of different environments. To create virtual

machines, it is then only necessary to fill in the created variable virtual_machines of the given module, which represents a list of specially defined objects, where each object represents one virtual machine. When this module is used, the deployment configuration
therefore looks as follows:

```
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"
...
```

Code: Example of using the created Terraform module

It can be seen that under the exposed variable virtual_machines it is possible to define
any number of virtual machines, which at the same time makes it possible to define their properties
such as the name (name), a unique identifier in the Kubernetes cluster (vm_id), a dynamic

number of disks and their corresponding sizes, network settings and several other settings - a detailed description is part of the documentation of the given repository.
As regards the local.* variables, they were defined in order to reduce duplicate information when creating multiple similar virtual machines (as is usually the case with Kubernetes clusters). They include information common to all virtual machines, or also differing information based on the role of the virtual machine within the Kubernetes cluster (control/worker node). An example is the amount of allocated memory or virtual processor cores (e.g. server_cpus and agent_cpus); however, this concept is applicable to anything:

```
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"]
}
```

Code: Example of using the created Terraform module – local variables

- The designed and created repository structure looks as follows:

```
terraform/
├── k8s/
│   ├── prod/
│   │   ├── (backend.tf)
│   │   ├── main.tf
│   │   ├── provider.tf
│   │   ├── (terraform.tfvars)
│   │   └── variables.tf
│   └── test/
│       └── ...
├── modules/k8s-proxmox/
│   ├── main.tf
│   └── variables.tf
├── Makefile
├── Makefile.vars
└── README.md
```

Note: Files placed in parentheses - "()" - are not part of the life cycle of the Git repository, as they contain sensitive variables that need to be defined during initialization of the local repository.

• modules/ - a folder containing modules. Within the Terraform tool, modules are reusable groupings of multiple Terraform configurations, often representing a certain part of the infrastructure. This makes it possible to define a configuration once (also with the option of parameterization) and then reuse it in different parts of the project/projects.

It is precisely for this purpose that the module (folder) modules/k8s-proxmox/ was created, which with this approach makes it possible to create various replicas of Kubernetes environments for testing purposes.

• k8s/ - contains subfolders that define a specific deployment (environment) of the Kubernetes cluster (e.g. k8s/prod/ for the production deployment or k8s-test/ for the test deployment).

• k8s/<env>/ - environment folder (e.g. k8s/prod/).

– backend.tf - defines where the current state of the infrastructure is stored (typically a *.tfstate file)

34

– provider.tf - defines the provider (provider), in this particular case a specific Proxmox provider

– variables.tf - declares the variables that the user will have to define (using the terraform.tfvars file) after initialization of the local repository. These variables store the login credentials for the Proxmox cluster to which the virtual machines of the module used are to be deployed.

– main.tf - the main terraform file, which uses the module defined above modules/k8s-proxmox

ArgoCD provides a clear web interface (as well as a CLI) for displaying and managing deployed applications and supports managing multiple clusters at once, which is also why it is often the preferred tool in application development. It also has a number of add-on modules providing additional functionality and connection to external systems (such as integration with HashiCorp Vault for directly adding sensitive data).

FluxCD/**ArgoCD** integration - In addition to the operators mentioned above, it is, however, also possible to use various integrations that are built into the GitOps tools FluxCD and ArgoCD. ArgoCD offers, for example, its own HashiCorp Vault plugin for direct integration without an additional operator [16], but it is also possible to enable additional support for KSOPS by changing the ArgoCD configuration (kustomize.buildOptions must be set with the additional switches "–enable-alpha-plugins and –enable-exec

FluxCD, on the other hand, also owing to its architecture, opts for a more minimalist design and mostly leaves this responsibility to external systems - with one exception, as it has native and robust support for SOPS.

- Management and deployment of Kubernetes applications

FluxCD was chosen as a suitable tool for deploying additional applications. For it, an additional repository had to be created that would contain all the YAML manifests needed to deploy the individual applications. It was then necessary to create a suitable structure that would be easily extensible with further, new applications and would at the same time allow an application not only to be deployed but also to be deleted from a specific cluster in a simple and clear way. It is equally important that the repository allows easy extension with new clusters without unnecessary duplication, for which the Kustomize tool was primarily used. Another aim in using this tool was to separate the versioning of the individual installed applications at cluster level – that is, so that updates of individual applications could be tested outside the production deployment and, once tested, safely applied to the production deployment as well.

0.2 DESCRIPTION OF THE DOCKER INFRASTRUCTURE

The infrastructure is based on the Proxmox VE hypervisor deployed directly on a physical server (bare-metal). Proxmox uses a combination of KVM (Kernel-based Virtual Machine) technologies for full virtualization and LXC for operating-system-level containerization. In the given design, KVM virtualization is used exclusively, which ensures full isolation of the individual virtual servers.

The hypervisor provides:

- management of compute resources (CPU, RAM),
- management of disk storage (LVM-thin, ZFS or a local RAID array),
- network virtualization by means of Linux bridge interfaces,
- centralized management and monitoring of virtual machines.

Each virtual server (VM) is allocated precisely defined hardware resources, including the number of virtual CPU cores, the size of the main memory and the disk space. The configuration is designed so that resources can be scaled dynamically without the need to reinstall the operating system.

- **2. Operating system of the virtual servers – AlmaLinux**

The individual virtual servers run the AlmaLinux distribution, which is binary-compatible with Red Hat Enterprise Linux. This operating system was chosen because of its:

- high stability and long-term support,
- suitability for server and production deployments,
- compatibility with Docker Engine and modern DevOps tools.

The basic configuration of the AlmaLinux servers includes:

- minimal installation (without a graphical interface),
- package management using DNF,
- centralized updates of security patches,
- firewall configuration (firewalld or nftables),
- secured access via SSH using key-based authentication.

Each virtual server is conceived as an application node running Docker Engine and the related application containers.

---

3. Containerization layer – Docker Engine

Docker Engine is installed on the AlmaLinux virtual servers and ensures isolated running of application components in the form of containers. Docker provides a uniform environment for deploying applications that is independent of the host system.

Docker Engine provides:

- management of the container life cycle,
- networking between containers using virtual Docker networks,
- data persistence by means of Docker volumes,
- isolation of processes, the file system and network interfaces.

The containers are designed so that each provides one specific function (e.g. database, backend service, frontend application), thereby adhering to the principle of microservice architecture.

- **4. Application orchestration – Docker Compose**

Containers are started and configured using Docker Compose files (docker-compose.yml). Docker Compose makes it possible to define declaratively:

- the individual services (services),
- the Docker images used,
- network connections between services,
- port mapping,
- persistent storage,
- configuration environment variables.

The Docker Compose files are stored directly on the virtual servers and enable:

- simple deployment of the entire application stack with a single command,
- reproducibility of the environment (test, staging, production),
- simple maintenance and updating of services.

Each application unit is separated into its own Compose project, which minimizes the risk of conflicts between the individual tools and services.

- **5. Overall architecture of the solution**

The entire infrastructure is designed as a multi-layer architecture:

- physical layer (hardware),
- virtualization layer (Proxmox VE),
- operating system (AlmaLinux VM),
- containerization layer (Docker Engine),
- application layer (services managed by Docker Compose).

Such an approach ensures a high degree of modularity, scalability and ease of infrastructure management. It also enables effective separation of infrastructure, system and application responsibilities, which is suitable for modern cloud-ready and DevOps-oriented environments.

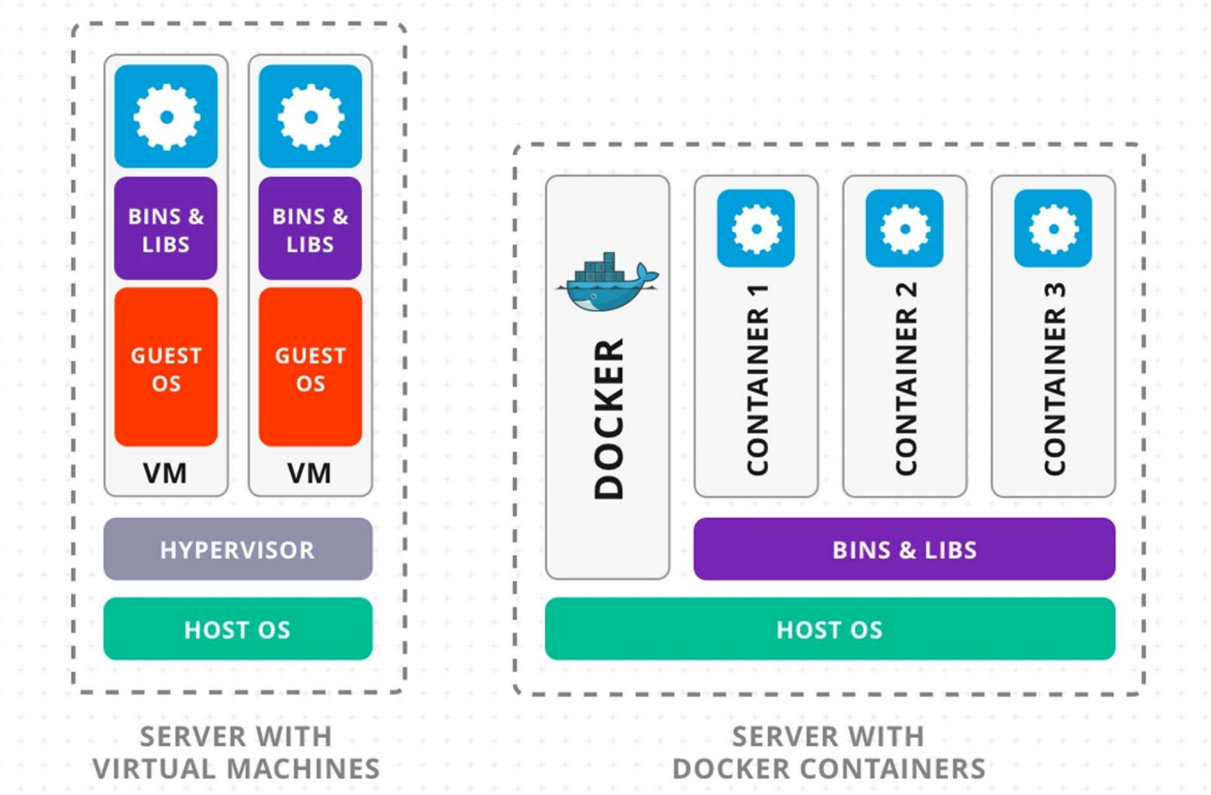


Diagram showing the topology of the Docker infrastructure

1. ELK STACK

Tool type: SIEM

Tool name: ELK Stack (Elasticsearch + Logstash/Beats + Kibana)

Version: Release 9.2505.1

System description

Central aggregation, indexing and correlation of logs + search and analysis

ELK Stack provides extensive support for log collection using tools such as Filebeat, Winlogbeat or Auditbeat. These components enable log collection from various types of systems, including Windows, Linux, network devices and application services. At the data analysis level, Logstash enables normalization and parsing of data, while Elasticsearch provides a powerful indexing and query engine. Although the free version itself does not provide advanced ML tools, basic alerting mechanisms can be set up via Watcher (or the Alerting plugin in the OpenSearch variant). Security features such as TLS between components and RBAC are only available in the paid editions, which represents a limitation in an open-source environment. Kibana, however, provides intuitive and extensible dashboards with numerous visualization options and filters.

System documentation

Unofficial docker compose documentation: https://github.com/deviantony/docker-elk

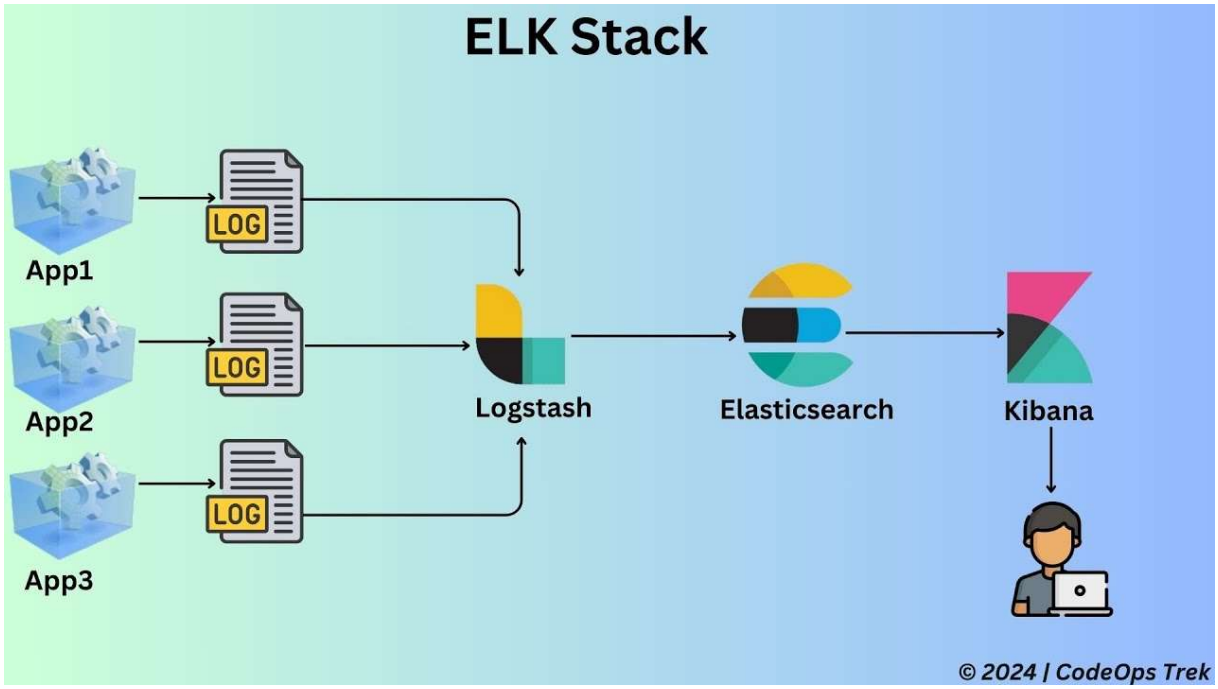
HW requirements:

8 vCPU

16GiB RAM

500GB disk (SSD, depending on the data retention settings)

Diagrams



Installation – Docker variant

```
      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:
```

Changes in docker compose: Some network ports have been changed – see the commented-out lines above.

Basic configuration

Tool configuration

ELK Stack contains a configuration script that correctly sets the addresses and the necessary authentication between the services. We then left most of the configuration at the default settings. The rest of the configuration then consists only of adapting to requirements or plugins and setting data retention.

Logstash configuration for syslog from 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

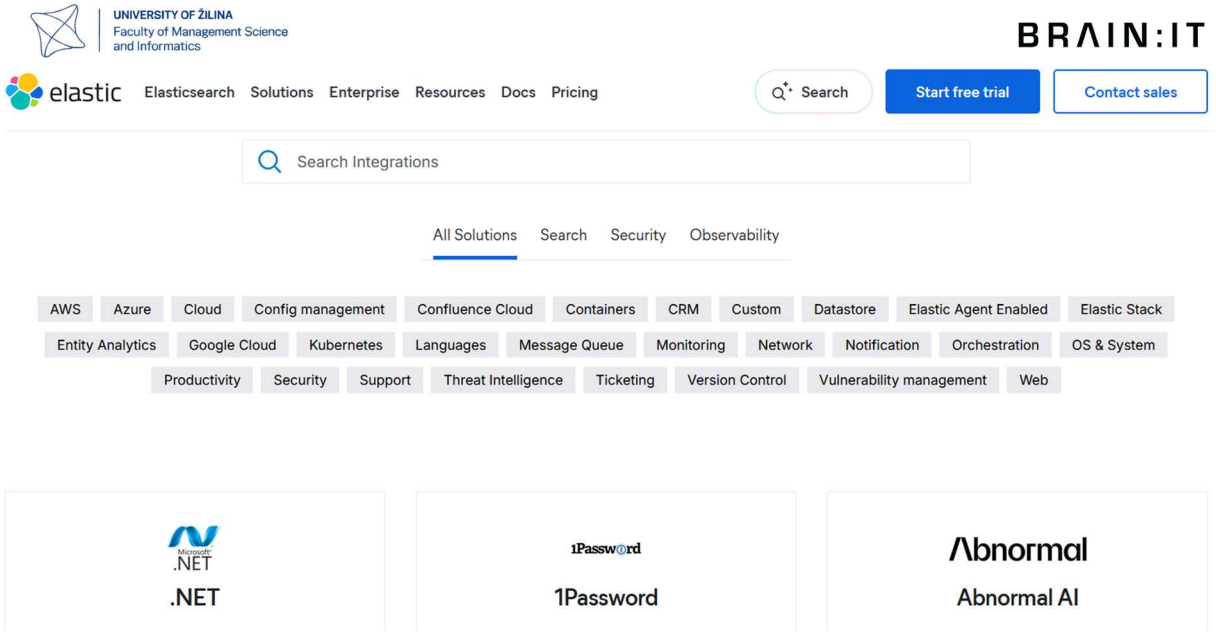

        # === Authentication ===
        # Use the internal user and password defined in your .env
file and passed to the container
        user => "logstash_internal"
        password => "${LOGSTASH_INTERNAL_PASSWORD}" # Logstash
reads this from its environment


        # === 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
      # }
    }
```

Integrations

Elastic has a huge number of integrations and integration options:



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

The stack also includes Logstash. Once Logstash has been correctly configured with how it should process specific syslog logs, it can be integrated with any system that sends syslog. An example of such an integration is the simple integration that we wrote for receiving logs from the Fortigate firewall.

2. SHUFFLE

Tool type: SOAR

Tool name: Shuffle

Version: Release 2.1.0

System description

Orchestration and automation of responses (playbooks)

Shuffle is a lightweight open-source SOAR tool with a modern web interface that enables visual creation of workflows. It is possible to connect to the APIs of various tools (Elastic, Graylog, VirusTotal, Slack, Jira, etc.). Its advantages are simplicity of deployment (Docker) and flexibility in building custom actions. However, it does not support such a range of IR functionality as TheHive and does not have a natively prepared incident management model.

Shuffle SOAR excels in the simplicity of creating and managing playbooks. It enables flexible integration using REST API and connectors. On the other hand, it lacks deeper incident management, but it is excellent as a complementary orchestrator for fast workflows.

System documentation

Official documentation:

Funded by the European Union
NextGenerationEU
RECOVERY AND RESILIENCE PLAN
GOVERNMENT OFFICE OF THE SLOVAK REPUBLIC

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

3. WAZUH

Tool type: EDR

Tool name: Wazuh

Version: Version 4.14

System description

Endpoint monitoring/detections and basic response + sending events to the SIEM

Wazuh integrates components for SIEM, EDR and FIM (File Integrity Monitoring). Data collection is possible via agents that support logs from operating systems, files, audit mechanisms and application servers. In addition, it can receive logs from Syslog, which enables centralized data collection also from passive devices without an agent. Wazuh provides its own correlation engine with numerous rules based on MITRE ATT&CK. These rules are regularly updated by the community. The web UI supports RBAC, task management and connection to Kibana dashboards. Its main strength is the combination of EDR features with a clear SIEM, which makes it an advantageous solution for smaller teams.

System documentation

Official documentation:

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

HW requirements:

4 vCPU, 8 GB RAM, 100+ GB Disk

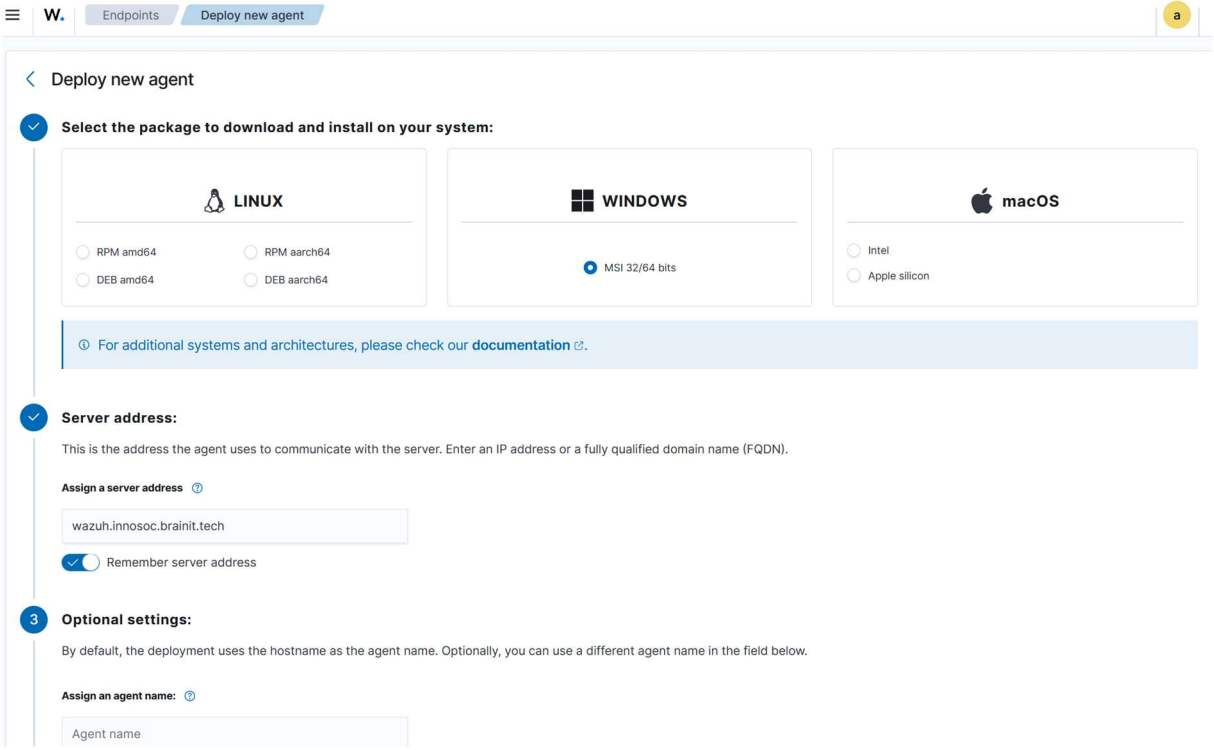
Installation – Quickstart variant

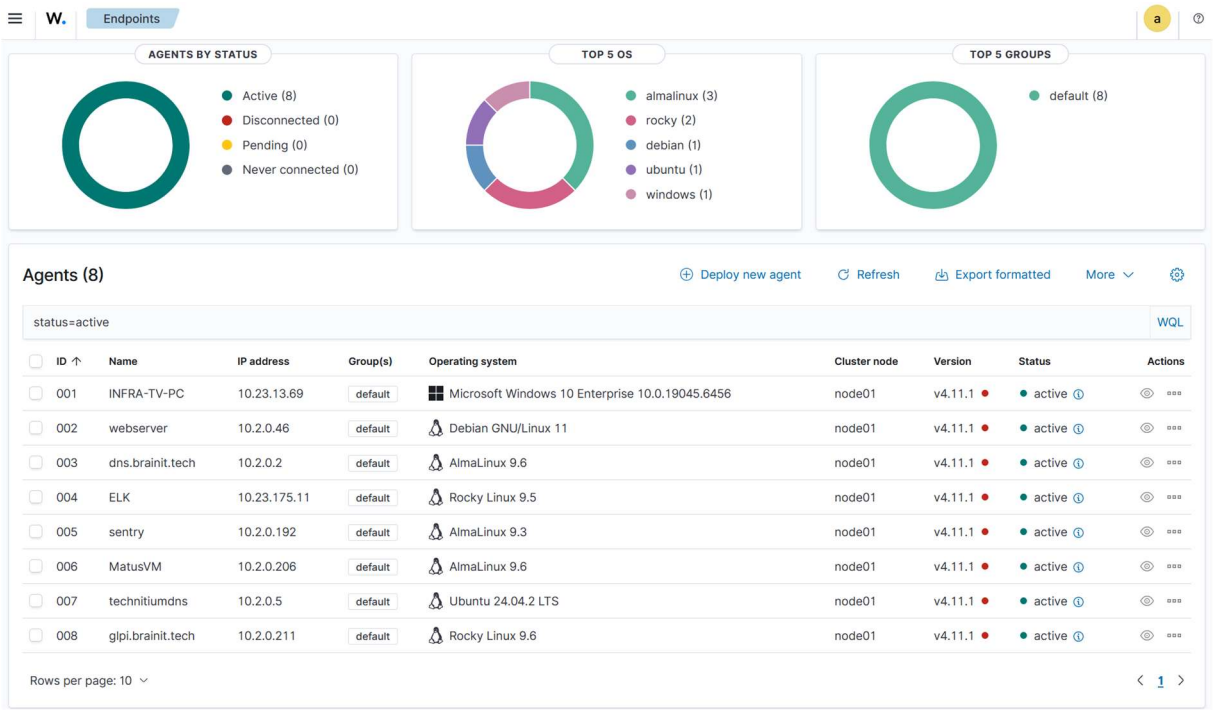
Installed on a Rocky Linux 9 VM according to the official quickstart documentation https://documentation.wazuh.com/current/quickstart.html

Basic configuration

Tool configuration:

Wazuh has a built-in system for providing the Wazuh agent installer and onboarding client workstations:





Integrations

Wazuh can be integrated with various SIEM systems including elastic search; it was therefore integrated into the ELK stack:
https://documentation.wazuh.com/current/integrations-guide/index.html

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

Funded by the European Union
NextGenerationEU
RECOVERY AND RESILIENCE PLAN
GOVERNMENT OFFICE OF THE SLOVAK REPUBLIC

4. GREENBONE COMMUNITY EDITION

Tool type: Vulnerability scanner

Tool name: Greenbone Community Edition (GVM) / OpenVAS

Version: Version 24.10

System description

Regular scanning of infrastructure/services and vulnerability reporting

GCE is the free open-source version of the solution from Greenbone Networks, based on the OpenVAS tool. It provides high-capacity scanning, extensive vulnerability databases via the community GVMD feed, the modern GSA web interface and support for scheduling, alerts and automation. It supports standards such as CVE and CPE, and enables the generation of reports in CSV, XML and PDF formats. Integration options include the GMP protocol, SSH, LDAP, RADIUS and others. GCE also enables a master-sensor architecture, which is advantageous for large distributed environments.

System documentation

Official documentation:

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

HW requirements:

4 vCPU, 8 GB RAM, 60 GB Disk

Diagrams

Installation – Docker variant

```
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:
```

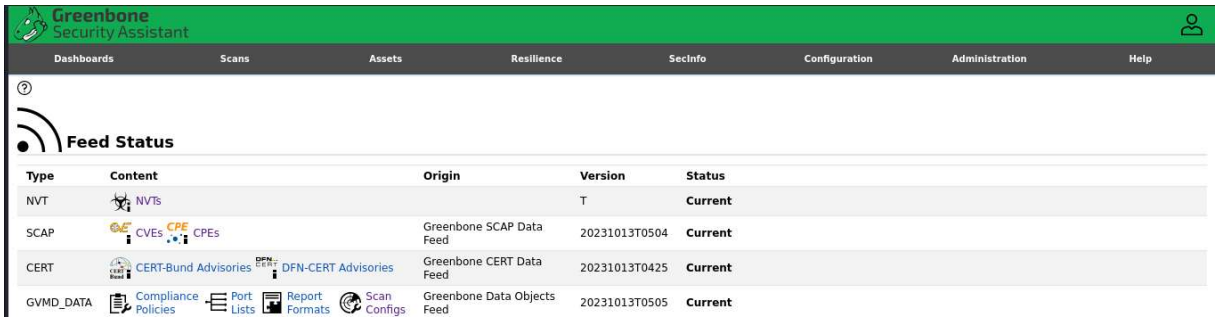
```
        condition: service_healthy
      notus-data:
        condition: service_healthy

  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:
```

Basic configuration

Tool configuration:
Only updating of the CVE databases, adding hosts.



5. OPENCTI

Tool type: CTI

Tool name: OpenCTI

Version: Version 6.9.0

System description

Management and correlation of threat intelligence, enrichment of alerts/incidents

System documentation

Official documentation:

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

6. MISP

Tool type: CTI

Tool name: MISP

Version:

System description

Recording, sharing and distribution of IOCs/events, feeds for detections

System documentation

Official documentation:

- https://www.misp-project.org/documentation/

- https://misp.github.io/MISP/

7. GLPI

Tool type: Asset management

Tool name: GLPI

Version: 10.0.16

System description

Asset records, CMDB/ITSM records, links to owners and services

System documentation

Official documentation:

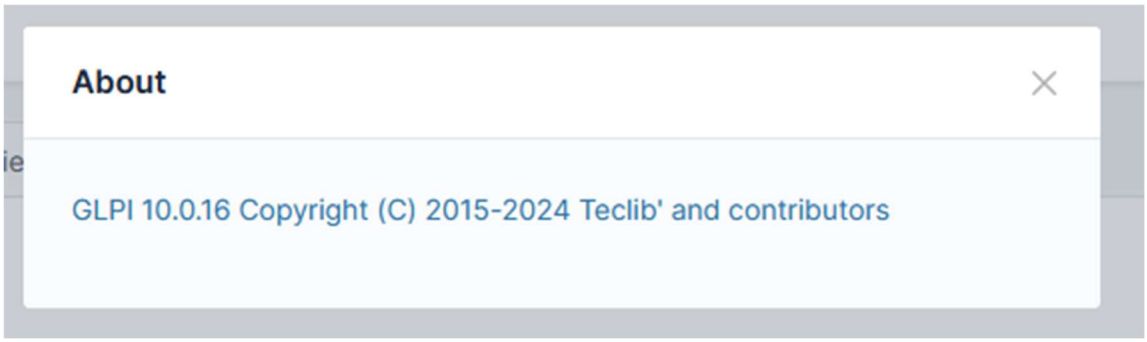
- https://glpi-plugins.readthedocs.io/en/latest/
- https://hub.docker.com/r/glpi/glpi

HW requirements:

2 vCPU, 4 GB RAM, 30 GB Disk

Diagrams (N/A)

Installation – Docker (Compose) variant



Version: GLPI 10.0.16

Docker-compose.yml (identical to the official documentation, with NGINX added as an HTTPS reverse proxy for SSL/TLS)

```
services:
  glpi:
    image: "glpi/glpi:latest"
    restart: "unless-stopped"
    volumes:
```

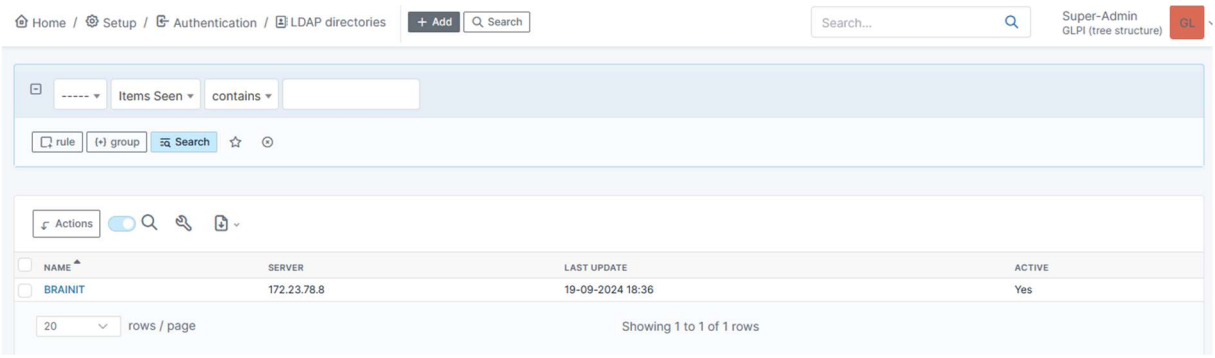
```
      - "./storage/glpi:/var/glpi:rw"
    env_file: .env # Pass environment variables from .env file to
the container
    depends_on:
      db:
        condition: service_healthy
    ports:
      - "80:80"
  db:
    image: "mysql"
    restart: "unless-stopped"
    volumes:
       - "./storage/mysql:/var/lib/mysql"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

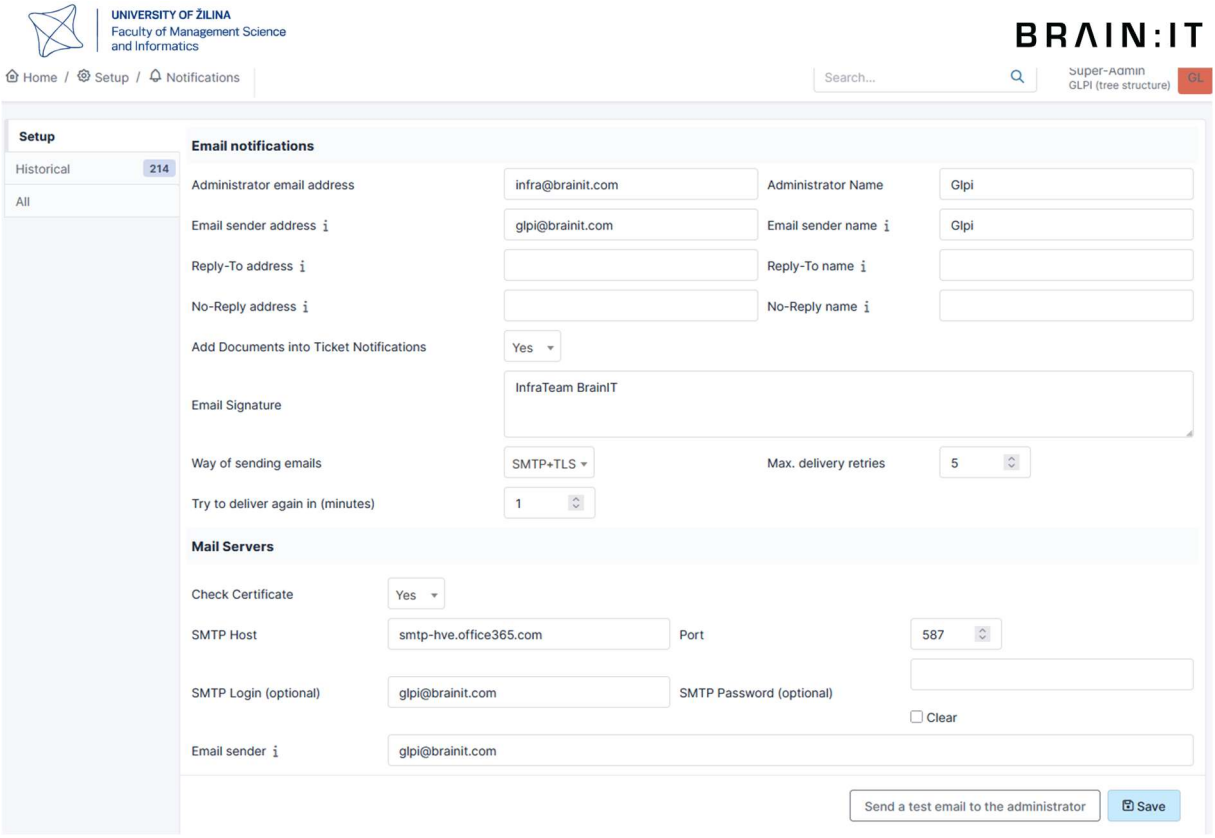
The .env file

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

Basic configuration

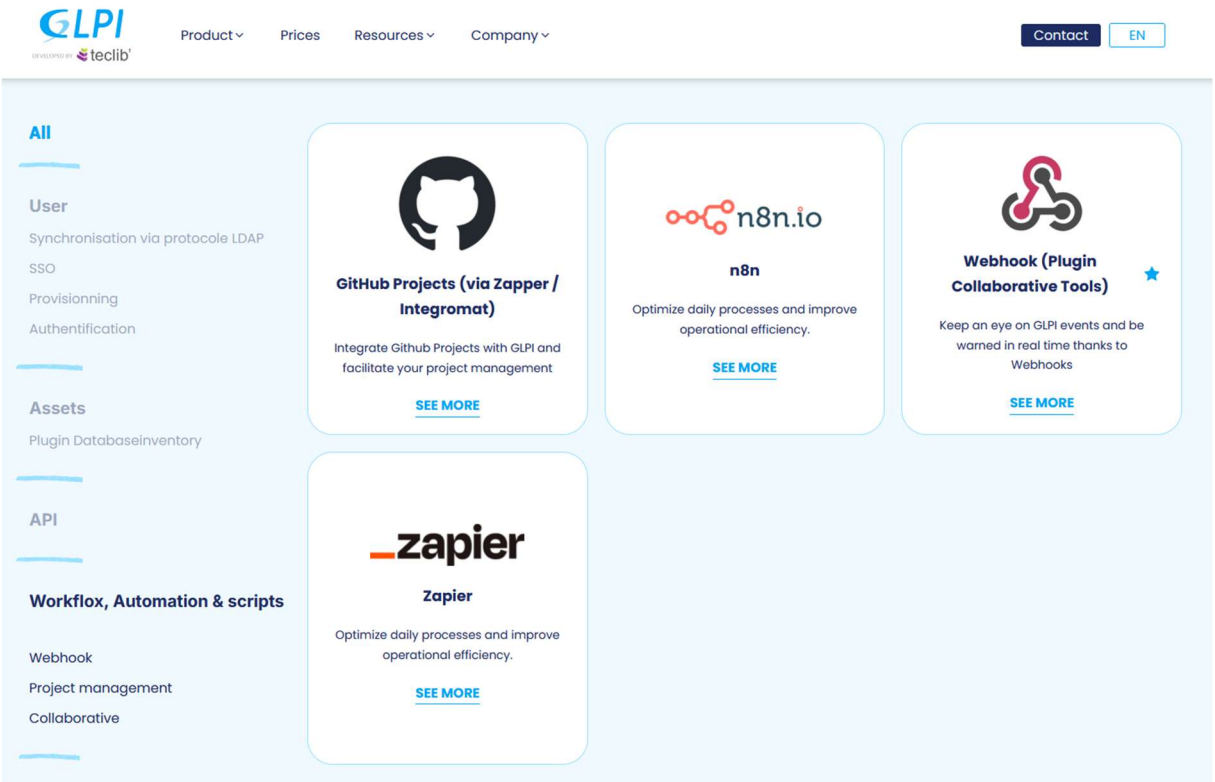


Authentication via AD/LDAP for the brainit.tech domain



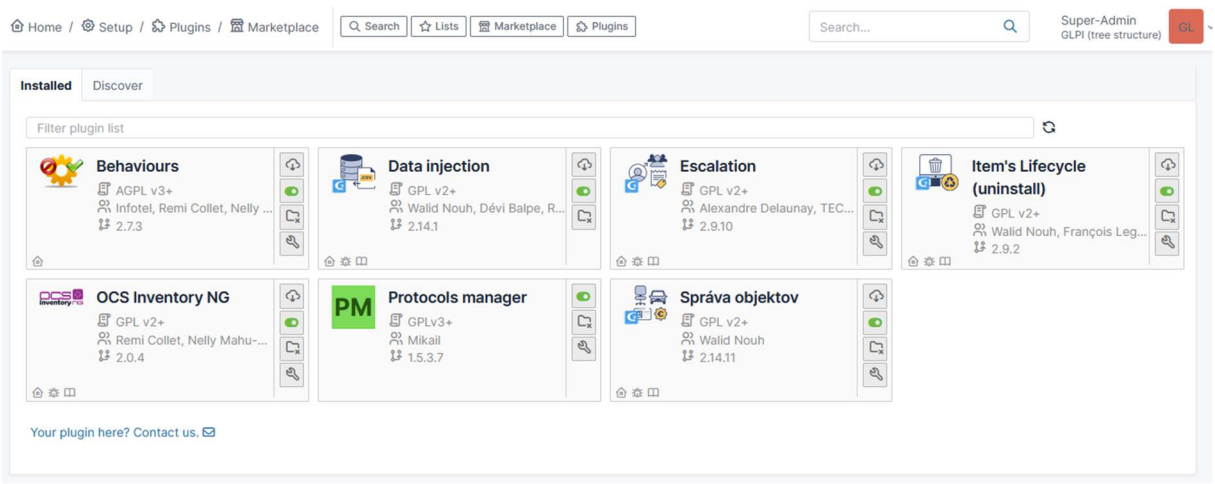
Setting up e-mail notifications as glpi@brainit.com

Integrations



Funded by the European Union
NextGenerationEU
RECOVERY AND RESILIENCE PLAN
GOVERNMENT OFFICE OF THE SLOVAK REPUBLIC

More on integration options at: https://www.glpi-project.org/en/integrations/



List of plugins in our instance:

https://github.com/InfotelGLPI/behaviors – adding mandatory fields, enforcing data formatting, ...

https://plugins.glpi-project.org/#/plugin/datainjection – data import in CSV format

https://github.com/pluginsGLPI/escalade – ticket management and escalation

https://github.com/pluginsglpi/uninstall – adding inventory states for decommissioning

https://github.com/pluginsGLPI/genericobject – adding custom object types

https://github.com/pluginsGLPI/ocsinventoryng – for synchronization with “OCS Inventory NG”

8. NETBOX

Tool type: Asset management / IPAM/DCIM

Tool name: NetBOX

Version:

System description

Source of truth for IP ranges, devices, sites, topology (basis for scans and monitoring)

System documentation

Official documentation:

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

9. DFIR IRIS – TICKETING

Tool type: Ticketing

Tool name: DFIR IRIS

Version:

System description

Management of SOC tickets and tasks (triage, escalations, SLA)

System documentation

Official documentation:

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

10. DFIR IRIS – INCIDENT RECORDS AND MANAGEMENT

Tool type: Incident records and management

Tool name: DFIR IRIS

Version:

System description

Management of SOC tickets and tasks (triage, escalations, SLA)

System documentation

Official documentation:

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

11. KIBANA (ELK)

Tool type: Reporting

Tool name: Kibana (ELK)

Version: 9.2.3

Tool description

Visualizations, dashboards and reports for the SOC and management

System documentation

Official documentation:

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

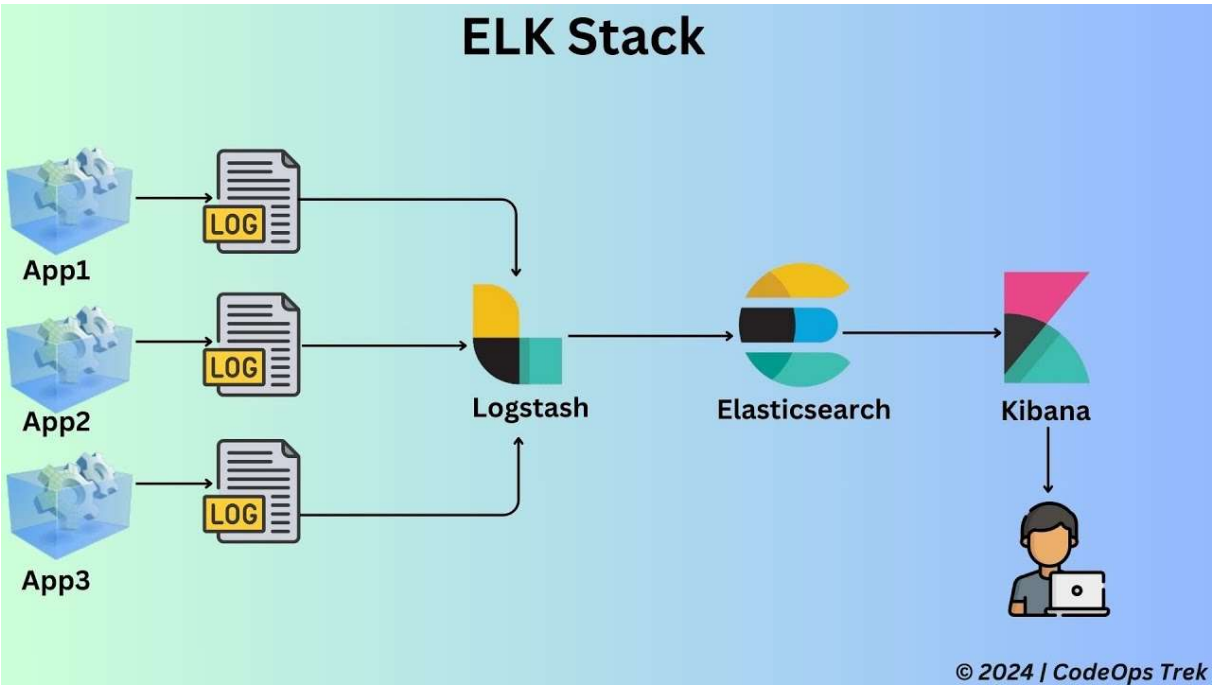
Unofficial docker compose documentation:

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

HW requirements

2 vCPU, 8 GB RAM, 50 GB Disk

Diagrams



Installation – Docker variant

```
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:
```

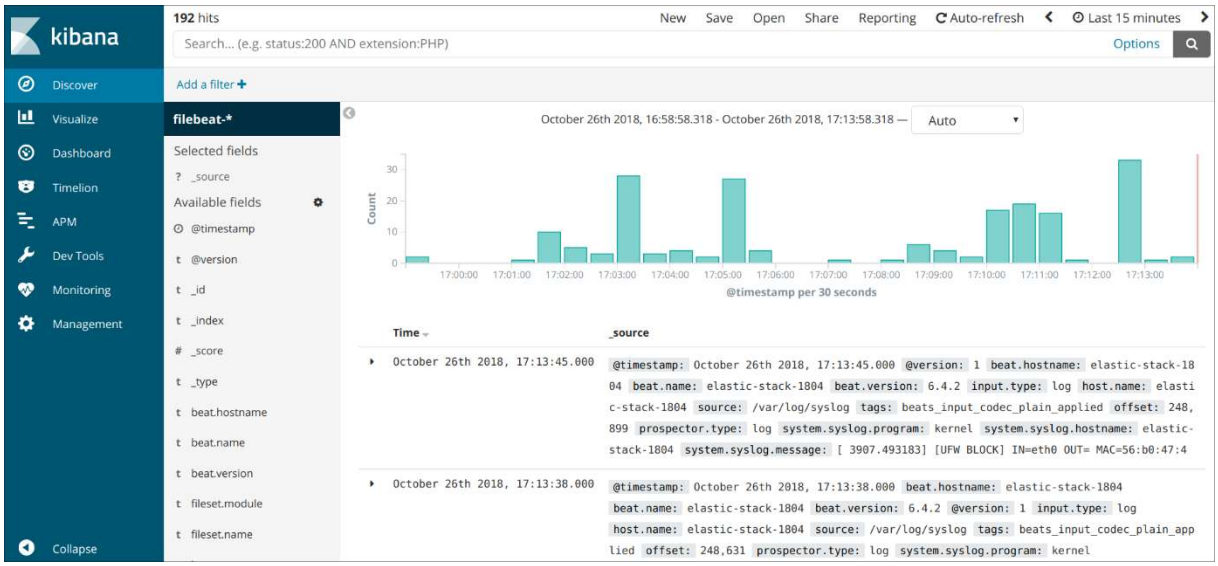
Changes in docker compose:

- This is Kibana itself, extracted from the complete ELK stack (1. ELK Stack)
- Some network ports changed – see the commented-out lines above.

Basic configuration

Tool configuration:

If it is part of the ELK Stack, it contains a configuration script that correctly sets the addresses and the necessary authentication between the services. Otherwise, it is only necessary to set the addresses and authentication manually according to the official documentation.



Integrations

Kibana is only a frontend, which as a rule is integrated mainly only with Elasticsearch. Elasticsearch then has further extensive options for further integration.

12. WIKI.JS

Tool type: Knowledge database

Tool name: Wiki.js

Version: 2.5

System description

Knowledge base (runbooks, procedures, playbooks, onboarding)

It is used for documenting internal systems and procedures.

System documentation

Official documentation:

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

HW requirements:

CPU: 2 cores

RAM: 1GB

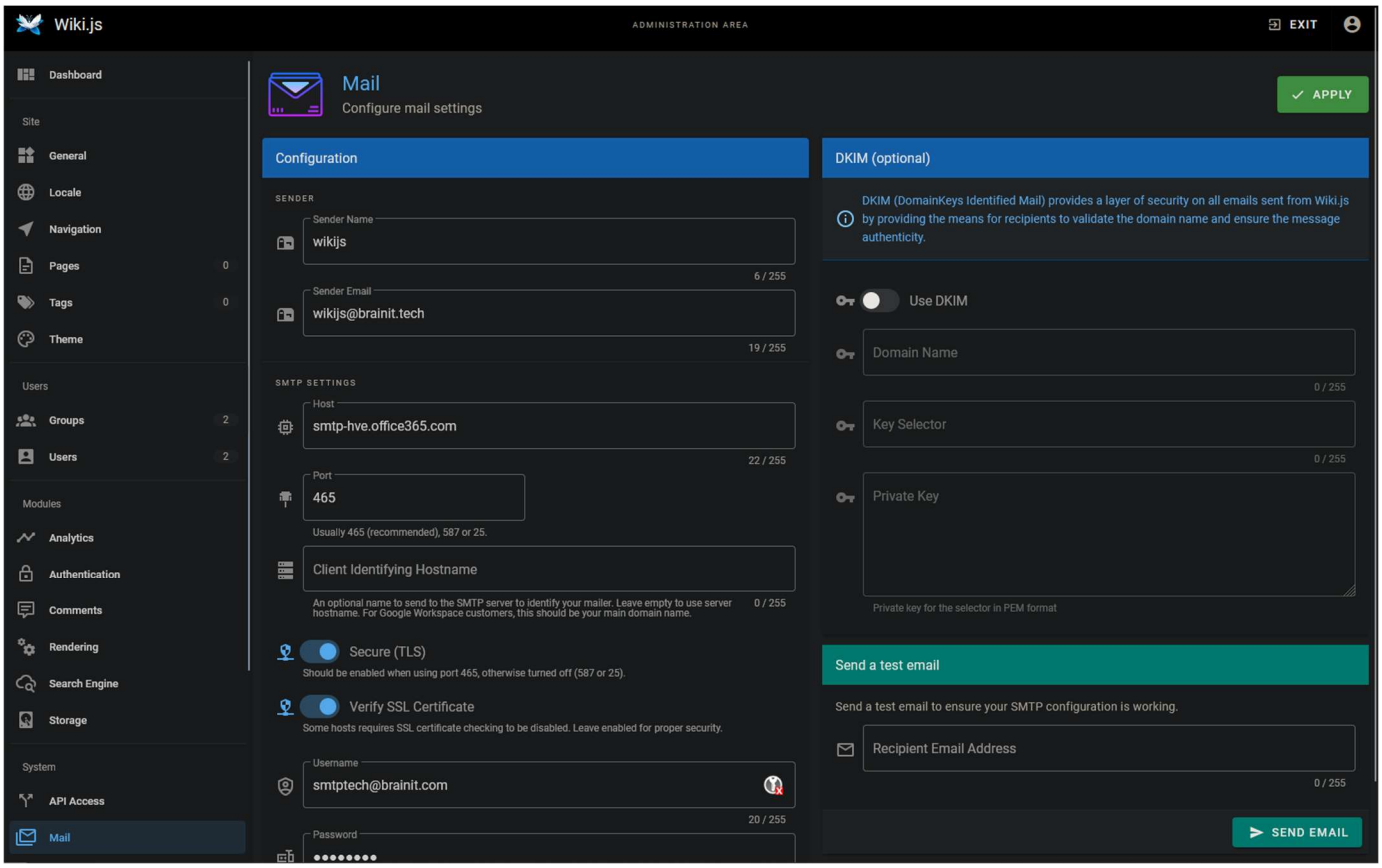
Storage: 1GB

Installation – Docker variant

Docker-compose.yml

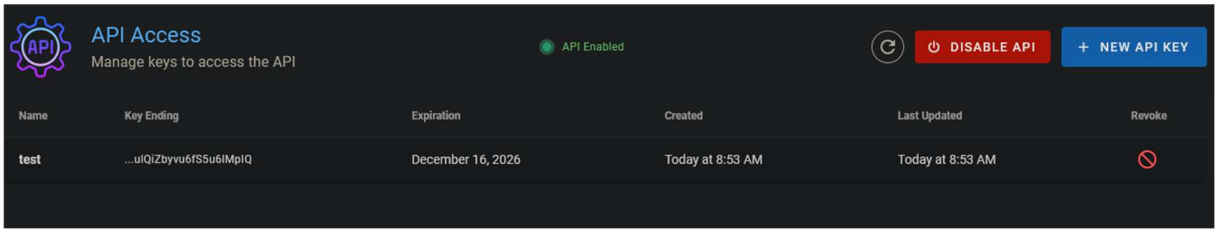
```
services:
  db:
    image: postgres:15-alpine
    environment:
      POSTGRES_DB: wiki
      POSTGRES_PASSWORD: wikijsrocks
      POSTGRES_USER: wikijs
    logging:
      driver: none
    restart: unless-stopped
    volumes:
      - db-data:/var/lib/postgresql/data
  wiki:
    image: ghcr.io/requarks/wiki:2
    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:
```

Basic configuration

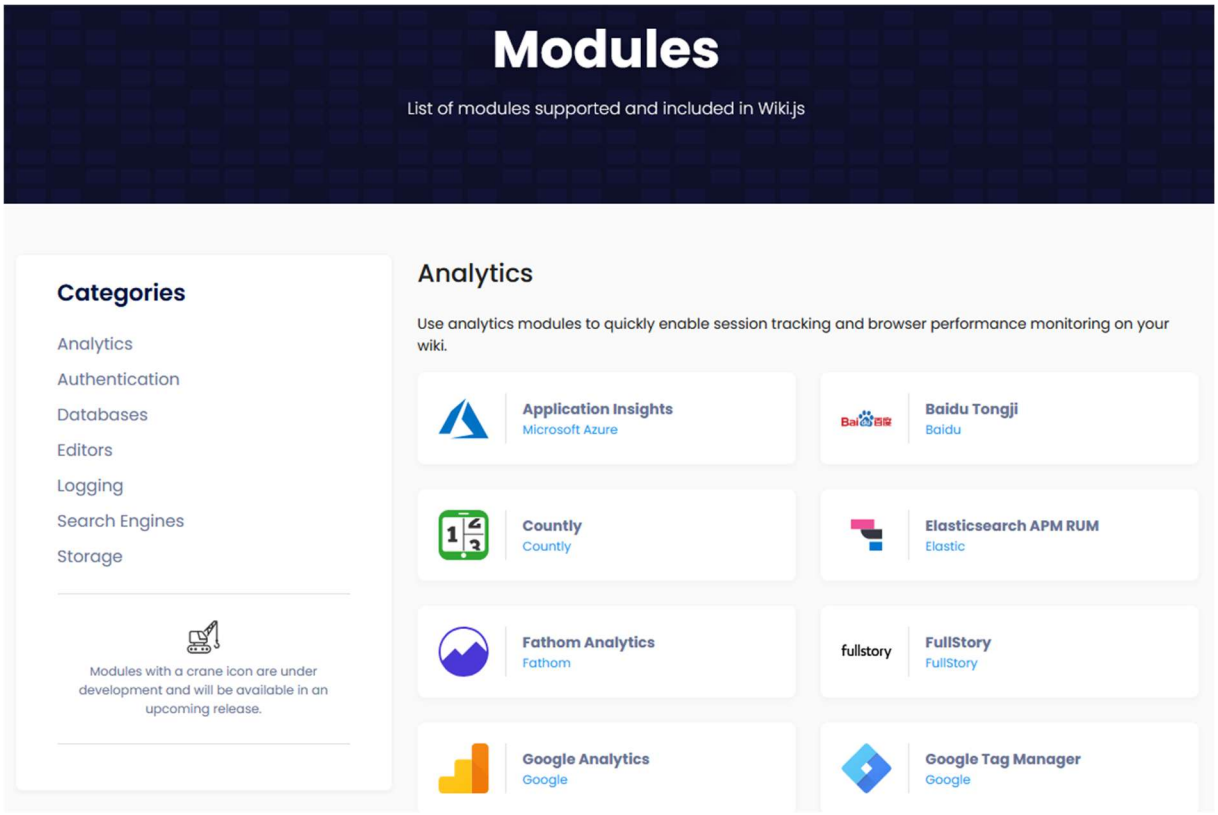


SMTP configuration

Integrations



Integration option via API keys



Other integration options... https://js.wiki/modules