Qu’est-ce qu’OpenObserve ?

OpenObserve est une plateforme d’observabilité qui permet de surveiller et analyser ce qui se passe dans un système informatique.
Elle sert à rassembler, stocker et visualiser différents types de données comme :

- Les logs (messages produits par les serveurs et applications)
- Les métriques (informations chiffrées comme l’utilisation du processeur ou de la mémoire)
- Les traces (le parcours d’une requête à travers plusieurs services)

À quoi ça sert ?

OpenObserve aide les administrateurs systèmes et les développeurs à :

- Comprendre l’état des serveurs et des applications
- Détecter les erreurs ou les pannes
- Suivre les performances (par exemple si un service devient lent)
- Centraliser toutes les informations au même endroit

Grâce à cela, on peut rapidement savoir ce qui ne va pas dans une infrastructure sans devoir chercher sur chaque machine.

Comment ça fonctionne ?

Des agents (comme Vector ou Fluent Bit) envoient les données (logs ou métrics) vers OpenObserve. Il enregistre ces données dans sa base interne. L’utilisateur peut ensuite les afficher sous forme de graphiques, tableaux ou recherches dans une interface web.

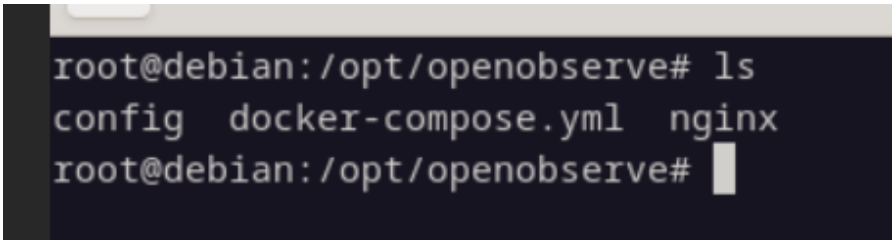
Les configurations mises en places :

Pour mettre en place la solution de monitoring, on a utilisé Docker et Vector, deux outils essentiels pour déployer et gérer l’infrastructure de supervision. Docker nous a permis de lancer facilement plusieurs services dans des conteneurs isolés, comme le serveur de supervision, la base de données et le proxy Nginx. L’utilisation de Docker Compose a simplifié le déploiement en regroupant tous les services dans un seul fichier de configuration, ce qui rend le système plus clair, plus facile à maintenir et plus portable. Grâce à Docker, chaque service fonctionne indépendamment des autres, tout en restant connecté dans un même réseau virtuel.

L’agent Vector, de son coté, a été installé pour assurer la collecte et la transmission des logs vers la plateforme de supervision(vector est installé sur la vm applicatif). Il récupère automatiquement les journaux du système et des conteneurs (comme ceux d’Apache, Nginx ou du système Debian), puis les envoie au serveur d’observation(la vm réseau dans notre cas). Vector peut aussi filtrer, transformer et enrichir les données avant de les transmettre, ce qui permet d’avoir des informations claires et organisées dans l’interface de monitoring.

Son rôle est donc de centraliser les logs provenant de différentes sources afin d'avoir une vision complète de l'activité des machines et de faciliter le diagnostic en cas de problème.

Dans la vm réseau :



docker-compose.yml :

```
version: "3.9"

services:
  postgres:
    image: postgres:15
    container_name: openobserve_postgres
    restart: unless-stopped
    environment:
      POSTGRES_DB: openobserve_db
      POSTGRES_USER: openobserve_user
      POSTGRES_PASSWORD: openobserve_password
    volumes:
      - pg_data:/var/lib/postgresql/data
    networks:
      - internal_net

  openobserve:
    image: public.ecr.aws/zinclabs/openobserve-enterprise:latest
    container_name: openobserve
    depends_on:
      - postgres
    environment:
      - ZO_DATA_DIR=/data
      - ZO_LOCAL_MODE=true
      - ZO_TELEMETRY=false
      - ZO_HTTP_ADDR=0.0.0.0:5080
      - ZO_JSON_LIMIT=209715200
      - ZO_PAYLOAD_LIMIT=209715200
      - ZO_META_STORE=postgres
      -
ZO_META_POSTGRES_DSN=postgres://openobserve_user:openobserve_password@postgres:54
32/openobserve_db
      - ZO_ROOT_USER_EMAIL=admin@nailloux.lan
      - ZO_ROOT_USER_PASSWORD=SuperStrongPassword123
    ports:
      - "5080:5080" # optional
    volumes:
      - o2_data:/data
```

```
      - ./config/config.yaml:/etc/openobserve/config.yaml:ro
    restart: unless-stopped
    healthcheck:
      test: ["CMD-SHELL", "curl -fs http://localhost:5080/health || exit 1"]
      interval: 30s
      timeout: 5s
      retries: 5
    networks:
      - internal_net

  nginx:
    image: nginx:latest
    container_name: openobserve_nginx
    depends_on:
      - openobserve
    ports:
      - "80:80"
    volumes:
      - ./nginx/openobserve.conf:/etc/nginx/conf.d/default.conf:ro
    restart: unless-stopped
    networks:
      - internal_net

volumes:
  o2_data:
    driver: local
  pg_data:
    driver: local

networks:
  internal_net:
    driver: bridge
```

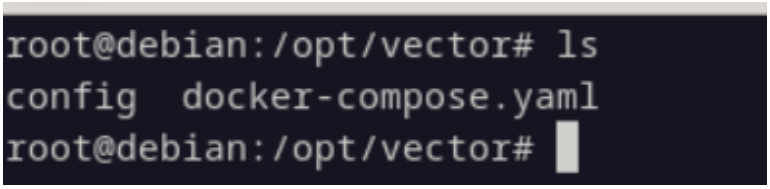
config/config.yaml

```
server:
  http_addr: 0.0.0.0
  http_port: 5080

auth:
  enabled: true
  local:
    enabled: true

telemetry:
  enabled: false

storage:
  data_dir: /data
  meta_store: postgres
```

nginx/openobserve.conf

```
server {
    listen 80;
    server_name monitoring.nailloux.lan;

    # Security headers
    add_header X-Content-Type-Options nosniff;
    add_header X-Frame-Options SAMEORIGIN;
    add_header X-XSS-Protection "1; mode=block";
    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;

    # Proxy to OpenObserve
    location / {
        proxy_pass http://openobserve:5080;
        proxy_http_version 1.1;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_connect_timeout 60s;
        proxy_read_timeout 120s;
    }

    # Optional: custom error pages
    error_page 502 503 504 /50x.html;
    location = /50x.html {
        root /usr/share/nginx/html;
    }
}
```

Dans la vm applicatif:



docker-compose.yaml

```
version: "3.9"

services:
  vector:
    image: timberio/vector:latest-alpine
    container_name: vector
    restart: unless-stopped
    networks:
      - internal_net
    environment:
      # OpenObserve credentials
```

```
      USER: admin@nailloux.lan
      PASSWORD: SuperStrongPassword123
      O2_HOST: monitoring.nailloux.lan
      O2_PORT: 80
    volumes:
      - ./config/vector.yml:/etc/vector/vector.yml:ro
      - /var/log:/var/log:ro
      - /var/run/docker.sock:/var/run/docker.sock
    command: ["--config", "/etc/vector/vector.yml"]
    ports:
      - "9100:9100"
    healthcheck:
      test: ["CMD-SHELL", "curl -fs http://localhost:9100/health || exit 1"]
      interval: 30s
      timeout: 5s
      retries: 5

networks:
  internal_net:
    driver: bridge
```

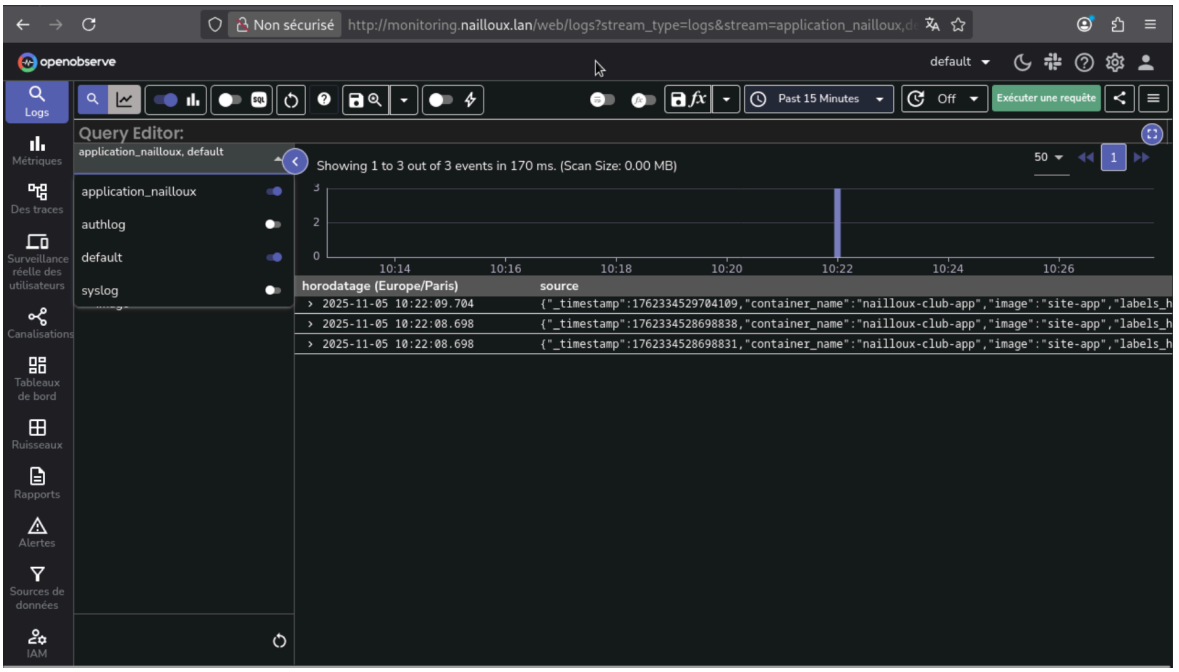
config/vector.yml

```
api:
  enabled: true
  address: "0.0.0.0:9100"

sources:
  files:
    type: file
    include:
      - "/var/log/syslog"
      - "/var/log/auth.log"
    ignore_older_secs: 86400   # ignore fichiers plus vieux que 1 jour
    read_from: beginning
    multiline:
      start_pattern: "^[^\\s]"
      mode: "continue_through"
      condition_pattern: "^\\s"
      timeout_ms: 1000
    max_line_bytes: 32768
    #max_size: 1073741824     # 1 Go

  docker_logs:
    type: docker_logs
    auto_partial_merge: true
    multiline:
      start_pattern: "^[^\\s]"
      mode: "continue_through"
      condition_pattern: "^\\s"
      timeout_ms: 1000

transforms:
```

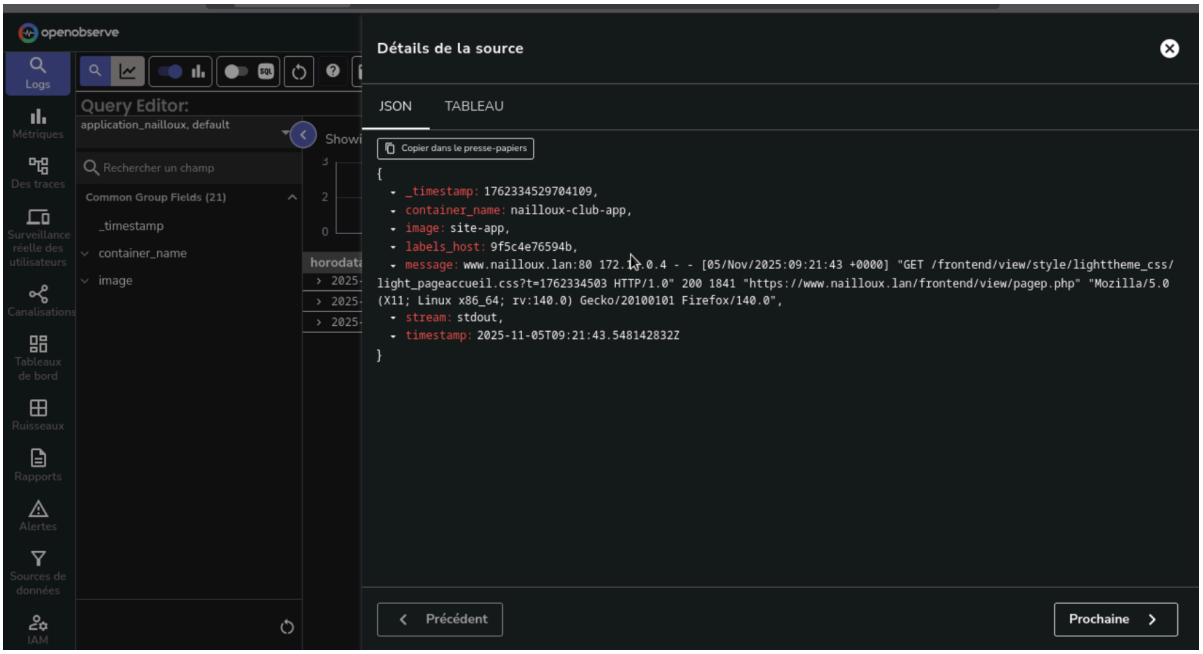
```
  add_labels:
    type: remap
    inputs: [files, docker_logs]
    source: |-
      .labels.host = get_hostname!()
      if exists(.container.name) {
        .labels.service_name = .container.name
        .labels.container = .container.name
      }
      if exists(.container.image) {
        .labels.image = .container.image
      }

  router:
    type: route
    inputs: [add_labels]
    route:
      auth_log: '.file == "/var/log/auth.log"'
      syslog: '.file == "/var/log/syslog"'
      docker_logs: '.source_type == "docker_logs"'

  keep_docker_fields:
    type: remap
    inputs: [router.docker_logs]
    source: |-
      .out = {}
      .out.container_name = .container_name
      .out.image = .image
      .out.message = .message
      .out.timestamp = .timestamp
      .out.stream = .stream
      .out.labels = .labels
      . = .out

# merge_all:
#   type: merge
#   inputs: [router.auth_log, router.syslog, keep_docker_fields]

sinks:
  openobserve_all:
    type: http
    inputs: [router.auth_log, router.syslog, keep_docker_fields]
    uri: "http://monitoring.nailloux.lan/api/default/default/_json"
    method: post
    auth:
      strategy: "basic"
      user: "admin@nailloux.lan"
      password: "9CXqXNGc6Mr3OMFN"
    compression: "gzip"
    encoding:
      codec: "json"
      timestamp_format: "rfc3339"
    request:
      in_flight_limit: 10
```

```
  timeout_secs: 10
  retry_attempts: 10
  retry_backoff_secs: 5
buffer:
  type: disk
  max_size: 1073741824
  when_full: block
healthcheck:
  enabled: true
```

l'interface d'openobserve :



On peut voir les logs et les informations de la vm applicatif (ici on voit les logs du site web nailloux qui est déployé en utilisant docker ->container_name nailloux-club-app) :


openobserve
Logs
Métriques
Des traces
Surveillance réelle des utilisateurs
Canalisations
Tableaux de bord
Ruisseaux
Rapports
Alertes
Sources de données
IAM
Query Editor:
application_nailloux, default
Rechercher un champ
Common Group Fields (21)
_timestamp
container_name
image
Détails de la source
JSON
TABLEAU
Copier dans le presse-papiers
{
_timestamp: 1762334529704109,
container_name: nailloux-club-app,
image: site-app,
labels_host: 9f5c4e76594b,
message: www.nailloux.lan:80 172.18.0.4 - - [05/Nov/2025:09:21:43 +0000] "GET /frontend/view/style/lighttheme_css/light_pageaccueil.css?t=1762334503 HTTP/1.0" 200 1841 "https://www.nailloux.lan/frontend/view/pagep.php" "Mozilla/5.0 (X11; Linux x86_64; rv:140.0) Gecko/20100101 Firefox/140.0",
stream: stdout,
timestamp: 2025-11-05T09:21:43.548142832Z
}
Précédent
Prochaine