zurück # CrowdSec Sync & Metabase Dashboard

Dieses Setup synchronisiert automatisch alle CrowdSec-Alerts (inkl. Entscheidungen) in eine eigene MySQL-Datenbank und visualisiert sie über Metabase.

—

🔧 Komponenten

- **CrowdSec** – erkennt Bedrohungen & verwaltet Entscheidungen - **MySQL/MariaDB** – speichert Alerts & Decisions - **Python-Script** – sync.py überträgt Daten via CrowdSec Local API - **Docker** – orchestriert `<DB_USER>-syncer` und Metabase - **Metabase** – Web-Dashboard zur Analyse

—

📁 Verzeichnisstruktur

``` /opt/docker/<DB_USER>-syncer/ ├── app/ │ └── sync.py ├── Dockerfile └── docker-compose.yml ```

—

🐳 Dockerfile

```
FROM python:3.11-slim

WORKDIR /app
COPY ./app /app

RUN pip install --no-cache-dir requests mysql-connector-python schedule

CMD ["python", "sync.py"]
```

—

📄 docker-compose.yml

```yaml version: „3.8“

services:

<DB_USER>-syncer:
  build: .
  container_name: <DB_USER>-syncer
  restart: unless-stopped
  networks:
    - docker_backend
  depends_on:
    - <DB_USER>
```

networks:
```

```
docker_backend:
  external: true
```

```

—

## □ app/sync.py

```python import requests, mysql.connector, schedule, time from datetime import datetime

API_KEY = „DEIN_API_KEY“ API_URL = „http://<DB_USER>:8080/v1/alerts?include_decisions=true“

DB_CONFIG = {

```
  "host": "<MYSQL_HOST>",
  "user": "<DB_USER>",
  "password": "<DB_PASSWORD>",
  "database": "<DB_USER>"
```

}

def sync():

```
  print(f"[{datetime.now()}] Starte Sync...")
  try:
      response = requests.get(API_URL, headers={"X-Api-Key": API_KEY})
      if response.status_code != 200:
          print("API Fehler:", response.status_code, response.text)
          return
```

```
      data = response.json()
      conn = mysql.connector.connect(**DB_CONFIG)
      cursor = conn.cursor()
```

```
      for alert in data:
          cursor.execute("""
              INSERT IGNORE INTO alerts (
                  id, scenario, scenario_hash, scenario_version, message,
                  created_at, source_scope, source_value, machine_id,
events_count, remediation
              ) VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)
          """, (
              alert.get("id"),
              alert.get("scenario"),
              alert.get("scenario_hash"),
              alert.get("scenario_version"),
              alert.get("message"),
              alert.get("created_at"),
              alert.get("source", {}).get("scope"),
```

```
                alert.get("source", {}).get("value"),
                alert.get("machine_id"),
                alert.get("events_count"),
                alert.get("remediation", True)
            ))
```

```
            for decision in alert.get("decisions", []):
                cursor.execute("""
                    INSERT IGNORE INTO decisions (
                        id, origin, type, scope, value, duration, start_at,
stop_at,
                        simulated, alert_id
                    ) VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)
                """, (
                    decision.get("id"),
                    decision.get("origin"),
                    decision.get("type"),
                    decision.get("scope"),
                    decision.get("value"),
                    decision.get("duration"),
                    decision.get("start_at"),
                    decision.get("stop_at"),
                    decision.get("simulated", False),
                    alert.get("id")
                ))
```

```
        conn.commit()
        print(f"{cursor.rowcount} Einträge synchronisiert.")
        cursor.close()
        conn.close()
```

```
    except Exception as e:
        print("Fehler:", e)
```

schedule.every(5).minutes.do(sync) sync()

while True:

```
    schedule.run_pending()
    time.sleep(60)
```

```

—

## □ SQL Tabellenstruktur

```sql – alerts CREATE TABLE IF NOT EXISTS alerts (

```
id INT PRIMARY KEY,
scenario VARCHAR(255),
```

```
scenario_hash VARCHAR(255),
scenario_version VARCHAR(64),
message TEXT,
created_at TIMESTAMP,
source_scope VARCHAR(64),
source_value VARCHAR(64),
machine_id VARCHAR(64),
events_count INT,
remediation BOOLEAN
```

);

– decisions CREATE TABLE IF NOT EXISTS decisions (

```
id INT PRIMARY KEY,
origin VARCHAR(64),
type VARCHAR(64),
scope VARCHAR(64),
value VARCHAR(64),
duration VARCHAR(64),
start_at TIMESTAMP,
stop_at TIMESTAMP,
simulated BOOLEAN,
alert_id INT,
FOREIGN KEY (alert_id) REFERENCES alerts(id) ON DELETE CASCADE
```

); ```

—

□ Beispiel-Metabase-Abfragen

```sql – Anzahl Bans pro Tag SELECT DATE(created_at) AS Tag, COUNT(*) AS Anzahl FROM alerts GROUP BY Tag ORDER BY Tag DESC; ```

```sql – Top IPs mit meisten Alerts SELECT source_value AS IP, COUNT(*) AS Hits FROM alerts GROUP BY source_value ORDER BY Hits DESC LIMIT 10; ```

—

□ Ergebnis

- Metabase zeigt strukturierte CrowdSec-Daten - Dashboards für Angriffsquellen, Szenarien und Zeitverlauf - Automatisierte, zyklische Datenaktualisierung

—

□ Tipps

- Achte auf gültige API-Keys (`cscli bouncers add …`) - Container-Rebuild bei Änderungen: `docker compose down && docker compose up -d –build` - Metabase-URL:

`https://<DB_USER>.mash4077.dedyn.io`

—

Lars Weiß 05.06.2025 11:45