Extended SecOps: Chronicle-Sniffer

A scalable Wireshark-to-SecOps pipeline on Google Cloud Platform.

Designed for robust, event-driven network traffic capture and security analytics.

Scalable and Reliable Services 2025

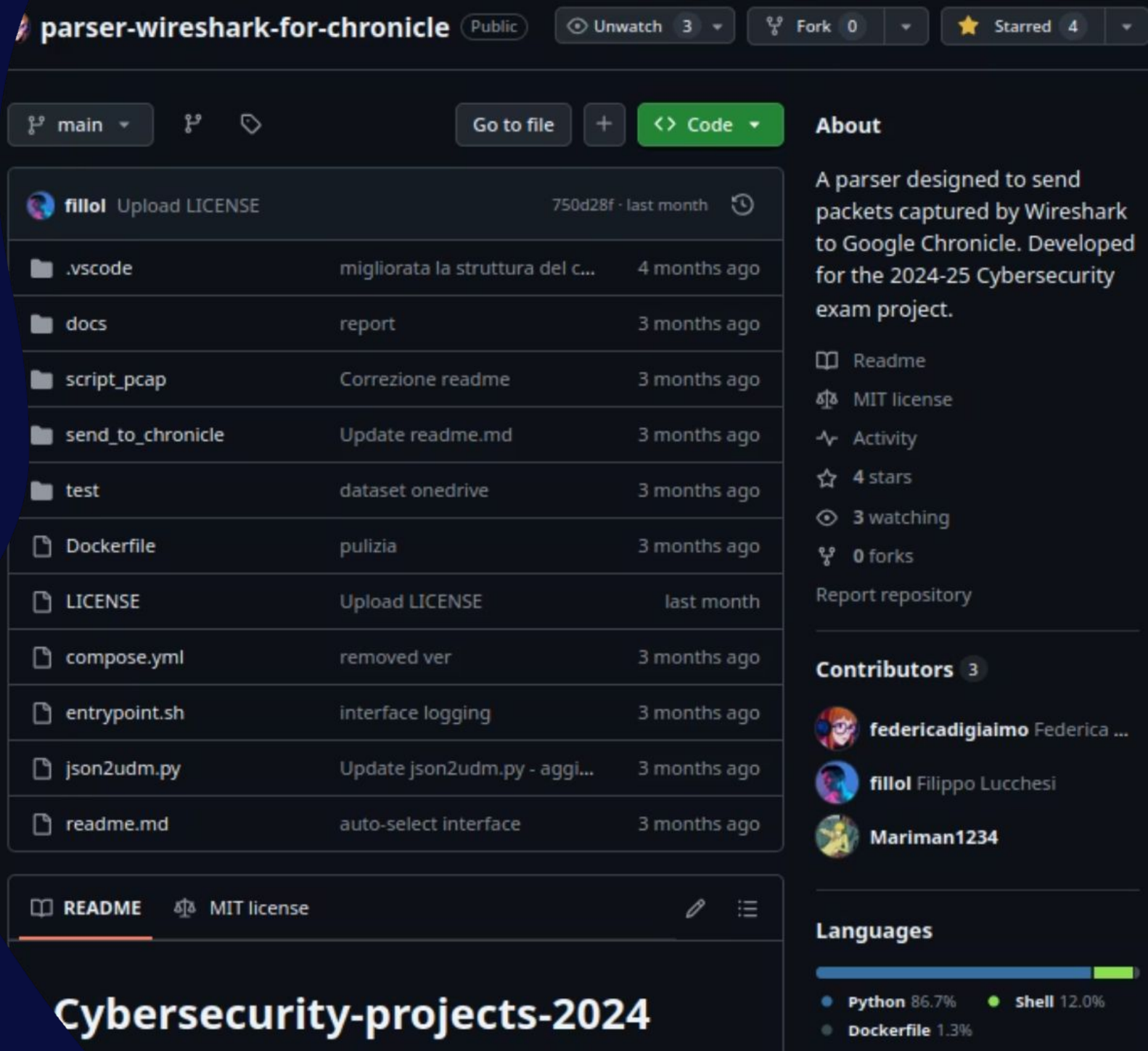
Filippo Lucchesi

Origins of the project

Autonomous proposal derived from my **Cybersecurity** exam's project "**Parser Wireshark for Chronicle**"

Enhanced with **cloud-native** architecture and enterprise-grade security operations capabilities

Not anymore a local script, now a complete **extension to Google SecOps**



Key Features & Enhancements

Hybrid Capture Model

Dockerized tshark sniffer for on-premises deployment with automatic PCAP rotation and cloud upload.

Serverless Processing

GCP Cloud Run service transforms network data into Unified Data Model format.

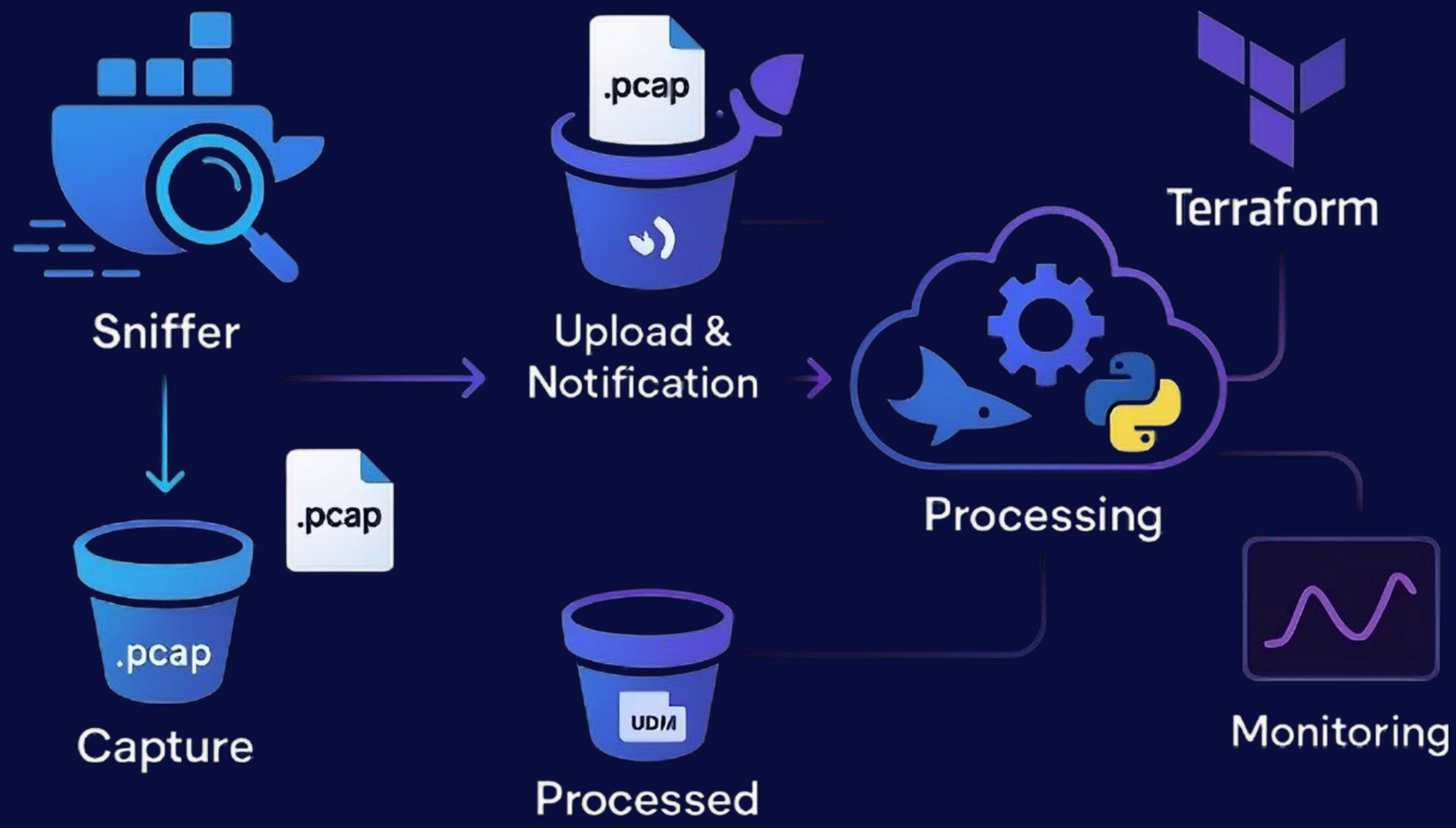
Optimized Transformation

Streaming JSON parsing handles massive outputs within Cloud Run's memory constraints.

Secure by Design

Implements IAM least-privilege principles and OIDC-authenticated invocations.

Architecture Overview



Architecture Overview

Capture & Notify

Sniffer container **runs** tshark, rotates PCAPs, **uploads** to GCS, and **notifies** Pub/Sub

Trigger & Process

Cloud Run service **downloads** PCAP, **converts** to JSON, **maps** to UDM format, and **uploads** results

Error Handling

Dead-letter **topics** capture failed messages after multiple retries

Observability

All logs sent to Cloud **Logging** with metrics available in Cloud **Monitoring**

Terraform Modules

1 **cloudrun_processor**

Deploys PCAP processor as Cloud Run **service** with resource limits and health probes

2 **gcs_buckets**

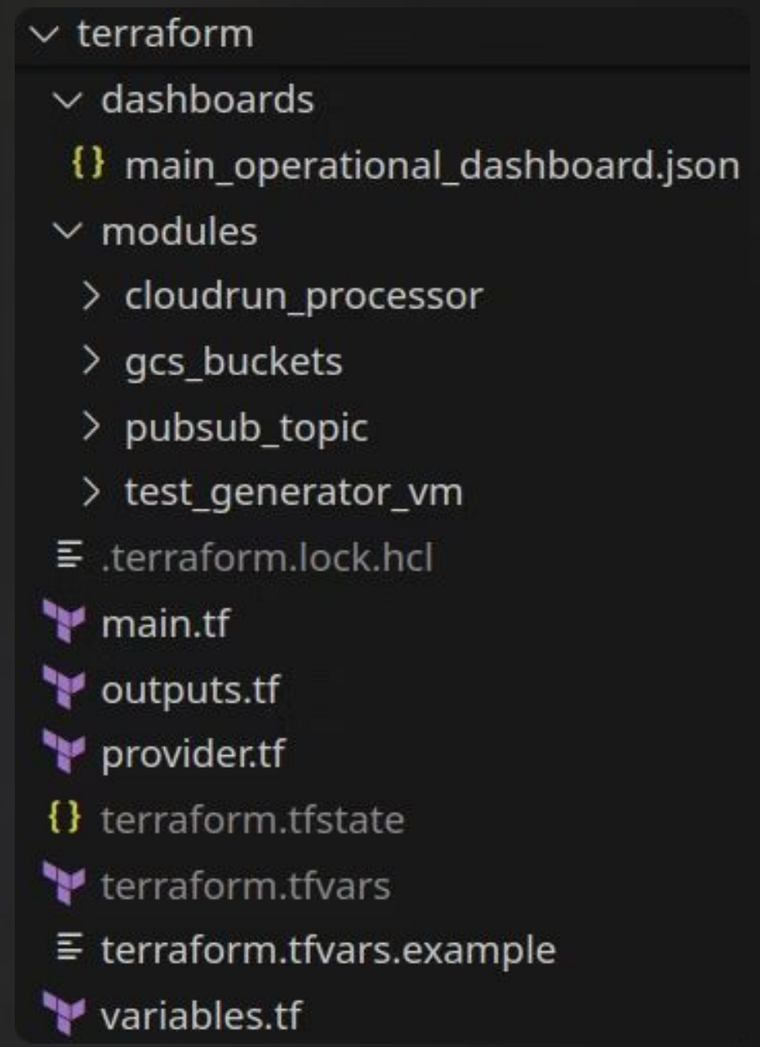
Provisions two GCS buckets: one for **incoming** raw PCAP files and another for **processed** UDM files

3 **pubsub_topic**

Creates Pub/Sub topic for **notifications** and **dead-letter** topic for failed messages

4 **test_generator_vm**

Optional GCE instance to **simulate** on-premises **environment** for **testing**



Terraform Implementation

Standard Structure

Follows Terraform **best practices** with main.tf, outputs.tf, and variables.tf files for root and all four **submodules**.

Variable Definition

Specific configurations defined in **terraform.tfvars** following official standards and guidelines.

Modular Design

Each submodule maintains consistent file organization enabling **easy maintenance** and reusable infrastructure **components**.

Configuration Management

Clear **separation** between variable **declarations** and actual values ensures secure and flexible **deployment** across environments.

Terraform.tfvars

terraform > terraform.tfvars > [ ] ssh_source_ranges > 0

```
gcp_project_id = "gruppo-2"
gcp_region     = "europe-west8"
gcs_location   = "EU"
base_name      = "chronicle-sniffer"

incoming_pcap_bucket_name = "chronicle-sniffer-incoming-pcaps"
processed_udm_bucket_name = "chronicle-sniffer-processed-udm"

processor_cloud_run_image = "europe-west8-docker.pkg.dev/gruppo-2/chronicle-sniffer/pcap-processor:latest"
sniffer_image_uri         = "fillol/chronicle-sniffer:latest" //"europe-west8-docker.pkg.dev/gruppo-2/chronicle-sniffer/onprem-sniffer:latest"

test_vm_zone                    = "europe-west8-b" // (o -a, -c)
test_vm_sniffer_id              = "my-custom-gce-sniffer-id"
allow_unauthenticated_invocations = false

cloud_run_max_concurrency = 10
cloud_run_cpu             = "1000m"
cloud_run_memory          = "2Gi" // S'interrompeva durante il processing con "512Mi"

alert_notification_channel_id = ""
ssh_source_ranges             = ["xxxxxxxxxxxxx/32"]
// enable_bucket_versioning  = true
// cmek_key_name             = "" // Lascia vuoto per usare Google-managed keys
```

All "personalized code" is defined in a **tfvars** file, fooling Terraform's guidelines

Sniffer Container Implementation

Dockerfile Setup

Based on Google Cloud SDK with tshark, procps, and iproute2 installed.

Network Capture

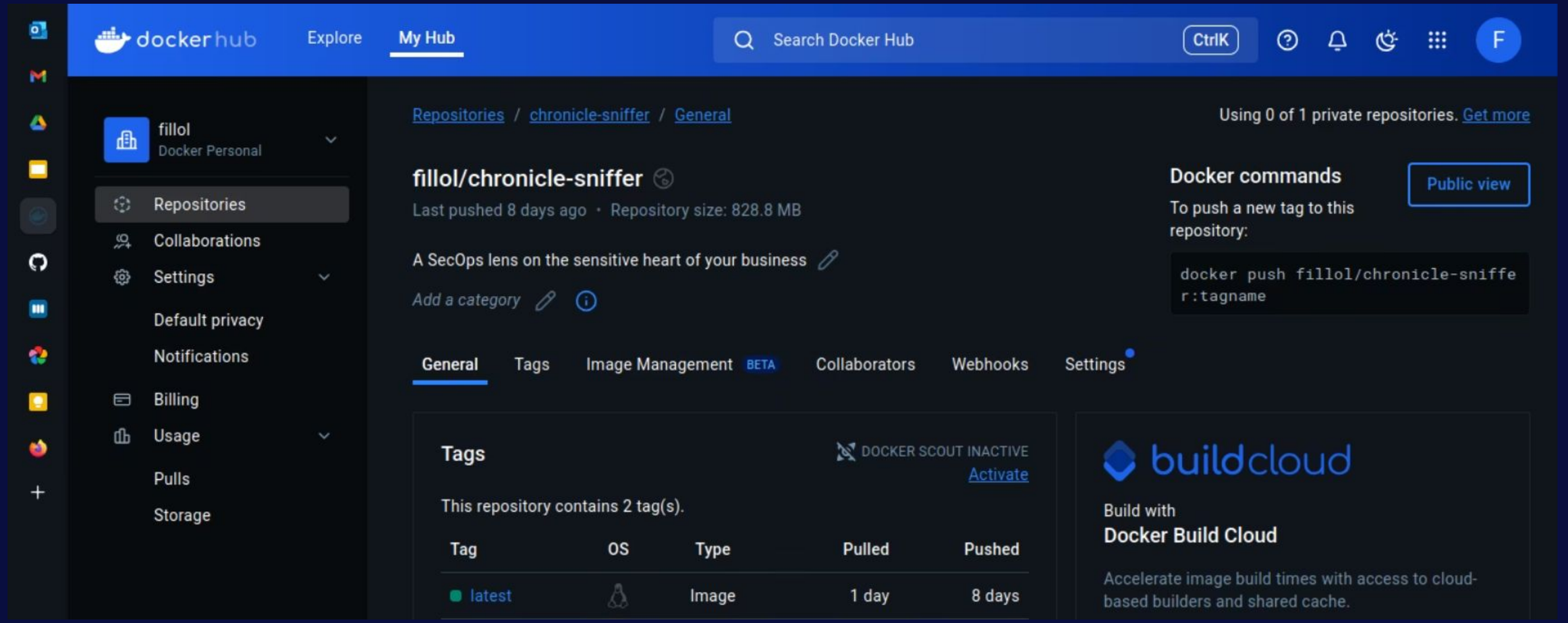
Detects active interface and runs tshark with configurable rotation settings.

Cloud Integration

Uploads completed PCAPs to GCS and publishes notifications to Pub/Sub.

The container includes heartbeat monitoring and graceful shutdown handling.

Sniffer: easy deployment

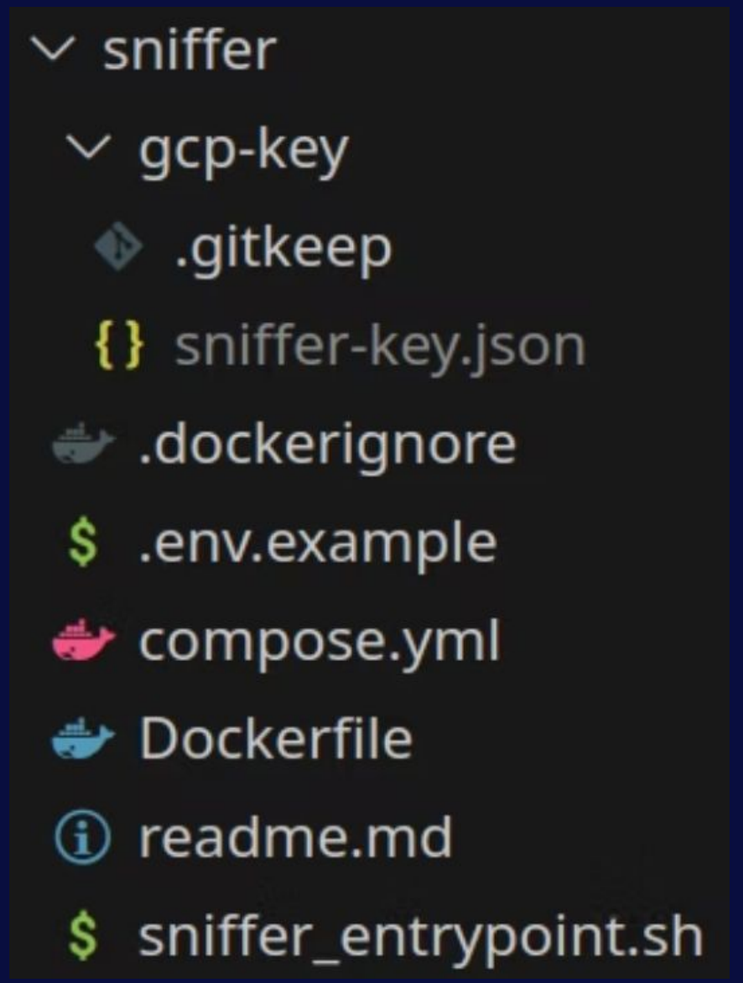


On-prem Container is public on DockerHub:

Sniffer: structure

Directory hierarchy:

- Service account credentials: **key.json**
- **.env** example (to compile your own)
- **readme** for sniffer documentation/usage

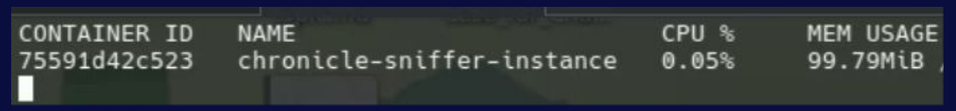


Sniffer: Dockerfile

```
FROM gcr.io/google.com/cloudsdktool/google-cloud-cli:alpine

RUN apk add --no-cache \
        tshark      \
        procps      \
        iproute2

WORKDIR /app
RUN mkdir captures gcp-key

COPY sniffer_entrypoint.sh .
RUN chmod +x sniffer_entrypoint.sh

ENTRYPOINT ["/app/sniffer_entrypoint.sh"]
```

Lightweight **Alpine** Image...

For a very **light** container:



(output of: `$ docker stats`)

Sniffer: simplest way to start on edge

```yaml
services:
  ▷ Run Service
  sniffer:
    image: fillol/chronicle-sniffer:latest
    build:
      context: .
    container_name: chronicle-sniffer
    restart: unless-stopped
    env_file:
      - .env
    network_mode: "host"
    cap_add:
      - NET_ADMIN
      - NET_RAW
    volumes:
      - ./gcp-key:/app/gcp-key:ro
      - ./captures:/app/captures
```

Docker compose.yaml

to start the container on premises

People only need to compile .env and run:

```
$ docker compose up -d
```

Docker .env

```
GCP_PROJECT_ID=IL_TUO_ID_PROGETTO_GCP
INCOMING_BUCKET=NOME_BUCKET_PCAP_IN     # Dall'output TF
PUBSUB_TOPIC_ID=ID_TOPIC_PUBSUB         # Dall'output TF (es. projects/...)
SNIFFER_ID="my-edge-location-01"        # Identificatore univoco per questa istanza dello sniffer
# ROTATE=-b filesize:5120               # Decommenta se vuoi personalizzare la rotazione
```

Sniffer: interface

At startup, the container automatically searches for a valid network interface to sniff on.
(if not provided)

It does this by iterating through **/sys/class/net** and excluding common virtual ones.

```
# Auto-detect active network interface if not explicitly set.
# This loop tries to find a suitable non-loopback, non-docker, etc., interface that is 'up'.
if [ -z "$INTERFACE" ]; then # Only auto-detect if INTERFACE is not already set by env var
    echo "(ID: $SNIFFER_ID) Network interface not specified. Searching for active network interface..."
    detected_iface_found=false
    while true; do # Loop for retrying detection
        for iface_path in /sys/class/net/*; do
            iface_name=$(basename "$iface_path")
            # Skip common virtual or loopback interfaces. Add others if needed.
            case "$iface_name" in lo|docker*|br-*|tun*|veth*|wg*|virbr*|kube-ipvs*) continue ;; esac
            # Check if the interface operational state is "up".
            if [[ -f "$iface_path/operstate" && $(< "$iface_path/operstate") == "up" ]]; then
                # Further check: Ensure it has an IP address (more robust check for "active")
                # This requires 'ip' command from 'iproute2' package
                if command -v ip >/dev/null && ip addr show dev "$iface_name" | grep -qw inet; then
                    INTERFACE_NAME_ONLY=$iface_name
                    INTERFACE="-i $iface_name" # Set tshark interface option.
                    echo "(ID: $SNIFFER_ID) Active network interface found and selected: $INTERFACE_NAME_ONLY"
                    detected_iface_found=true
                    break 2 # Break out of both loops (inner for, outer while).
                fi
            fi
        done
        if [ "$detected_iface_found" = false ]; then
            echo "(ID: $SNIFFER_ID) No suitable active interface found with an IP address. Retrying in 10 seconds..."
            sleep 10
        fi
    done
else
    # If INTERFACE was set via environment variable, use it directly.
    # Assume it's just the name, so prefix with '-i'.
    INTERFACE_NAME_ONLY=$INTERFACE
    INTERFACE="-i $INTERFACE"
    echo "(ID: $SNIFFER_ID) Using specified network interface: $INTERFACE_NAME_ONLY"
fi
```

Sniffer: entrypoint

```
find "$CAPTURE_DIR" -maxdepth 1 -name "${FILENAME_BASE}_*.pcap*" -type f | while read -r pcap_file_path; do
    current_pcap_file_basename=$(basename "$pcap_file_path")

    # Skip if this is the file tshark is currently writing to.
    if [[ -n "$active_file" && "$current_pcap_file_basename" == "$active_file" ]]; then
        # echo "[DEBUG] Skipping active file: $current_pcap_file_basename" # Optional debug
        continue
    fi
    # Skip if this file has already been processed.
    if is_processed "$current_pcap_file_basename" "${processed_files[@]}"; then
        # echo "[DEBUG] Skipping already processed file: $current_pcap_file_basename" # Optional debug
        continue
    fi

    echo "[$(date +'%Y-%m-%dT%H:%M:%SZ')] (ID: $SNIFFER_ID) Detected completed file: $current_pcap_file_basename"
```

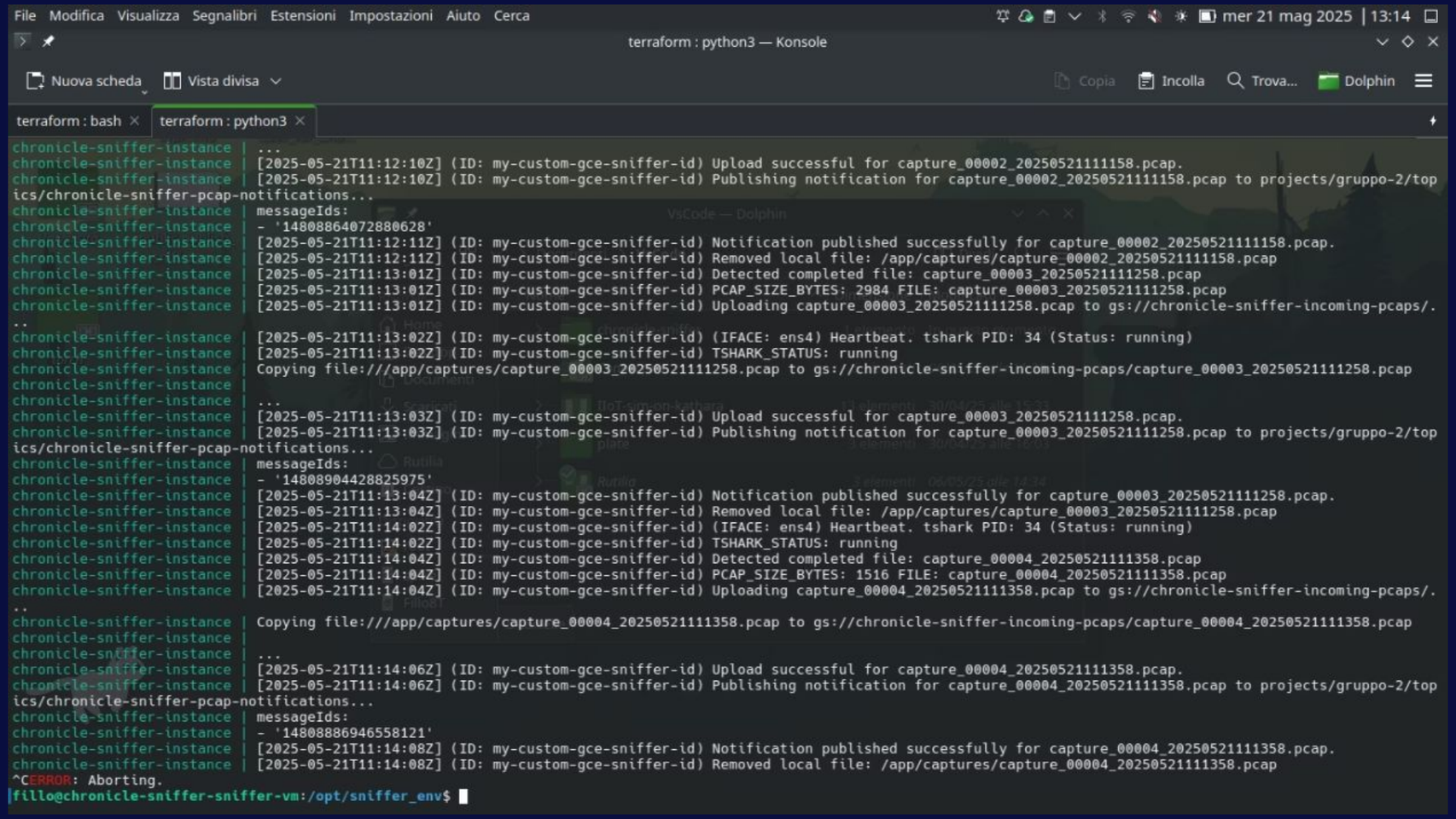
To find completed pcap...

Sniffer: entrypoint

```
    # Get file size
    file_size_bytes=$(stat -c%s "$pcap_file_path")
    echo "[$(date +'%Y-%m-%dT%H:%M:%SZ')] (ID: $SNIFFER_ID) PCAP_SIZE_BYTES: $file_size_bytes FILE: $current_pcap_file_basename"

    # 1. Upload the completed .pcap file to Google Cloud Storage.
    echo "[$(date +'%Y-%m-%dT%H:%M:%SZ')] (ID: $SNIFFER_ID) Uploading $current_pcap_file_basename to gs://${INCOMING_BUCKET}/..."
    if gcloud storage cp "$pcap_file_path" "gs://${INCOMING_BUCKET}/" --project "$GCP_PROJECT_ID"; then
        echo "[$(date +'%Y-%m-%dT%H:%M:%SZ')] (ID: $SNIFFER_ID) Upload successful for $current_pcap_file_basename."

        # 2. Publish a notification to Pub/Sub with the filename.
        echo "[$(date +'%Y-%m-%dT%H:%M:%SZ')] (ID: $SNIFFER_ID) Publishing notification for $current_pcap_file_basename to ${PUBSUB_TOPIC_ID}..."
        if gcloud pubsub topics publish "$PUBSUB_TOPIC_ID" --message "$current_pcap_file_basename" --project "$GCP_PROJECT_ID"; then
            echo "[$(date +'%Y-%m-%dT%H:%M:%SZ')] (ID: $SNIFFER_ID) Notification published successfully for $current_pcap_file_basename."
            processed_files+=("$current_pcap_file_basename") # Add to processed list.
            # 3. Remove the local .pcap file after successful upload and notification.
            rm "$pcap_file_path"
            echo "[$(date +'%Y-%m-%dT%H:%M:%SZ')] (ID: $SNIFFER_ID) Removed local file: $pcap_file_path"
        else
            echo "[$(date +'%Y-%m-%dT%H:%M:%SZ')] (ID: $SNIFFER_ID) Error: Failed to publish notification for $current_pcap_file_basename. Will retry."
            # File is not removed or added to processed_files, will retry on next loop iteration.
        fi
    else
        echo "[$(date +'%Y-%m-%dT%H:%M:%SZ')] (ID: $SNIFFER_ID) Error: Failed to upload $current_pcap_file_basename to GCS. Will retry."
        # Upload failed; will retry on next loop iteration.
    fi
done
```

And upload them to GCP

Sniffer: running...



Output of: `$docker compose logs -f` on test-VM

Sniffer: to ensure consistency...

```
graceful_shutdown() {
  echo "[$(date +'%Y-%m-%dT%H:%M:%SZ')] (ID: $SNIFFER_ID) Received termination signal. Shutting down tshark and heartbeat..."
  # Terminate tshark process
  if kill -0 $TSHARK_PID 2>/dev/null; then
    echo "[$(date +'%Y-%m-%dT%H:%M:%SZ')] (ID: $SNIFFER_ID) Sending SIGTERM to tshark (PID: $TSHARK_PID)..."
    kill -TERM $TSHARK_PID
    wait $TSHARK_PID # Wait for tshark to finish
    echo "[$(date +'%Y-%m-%dT%H:%M:%SZ')] (ID: $SNIFFER_ID) tshark terminated."
  else
    echo "[$(date +'%Y-%m-%dT%H:%M:%SZ')] (ID: $SNIFFER_ID) tshark (PID: $TSHARK_PID) already stopped."
  fi
  # Terminate heartbeat process
  if kill -0 $HEARTBEAT_PID 2>/dev/null; then
    echo "[$(date +'%Y-%m-%dT%H:%M:%SZ')] (ID: $SNIFFER_ID) Sending SIGTERM to heartbeat (PID: $HEARTBEAT_PID)..."
    kill -TERM $HEARTBEAT_PID
  fi
  echo "[$(date +'%Y-%m-%dT%H:%M:%SZ')] (ID: $SNIFFER_ID) Sniffer shutdown complete."
  exit 0
}
```

```
# Function for sniffer heartbeat, runs in background
send_heartbeat() {
    while true; do
        # Log current tshark status along with heartbeat
        local tshark_status="stopped"
        if kill -0 $TSHARK_PID 2>/dev/null; then # Check if tshark process exists
            tshark_status="running"
        fi
        echo "[$(date +'%Y-%m-%dT%H:%M:%SZ')] (ID: $SNIFFER_ID) (IFACE: $INTERFACE_NAME_ONLY) Heartbeat. tshark PID: $TSHARK_PID (Status: $tshark_status)"
        echo "[$(date +'%Y-%m-%dT%H:%M:%SZ')] (ID: $SNIFFER_ID) TSHARK_STATUS: $tshark_status" # Explicit log for TSHARK_STATUS metric
        sleep 60 # Send heartbeat every 60 seconds
    done
}
```

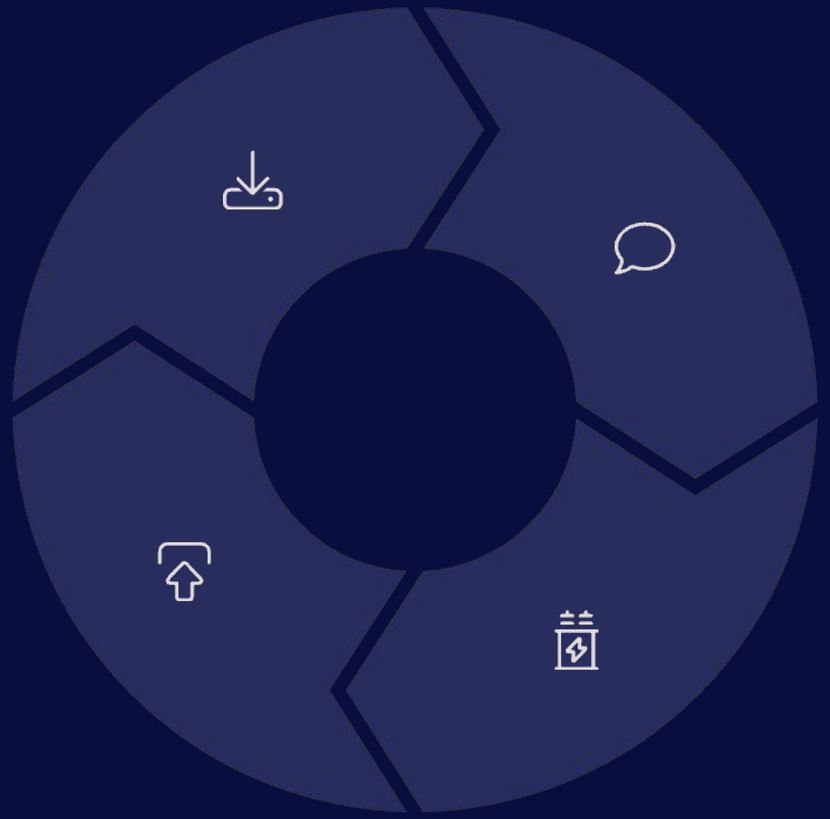
Sniffer: IAM

```
// --- IAM Permissions ---
// Configures IAM policies to grant necessary permissions to Service Accounts.

// IAM for Sniffer Service Account (used via downloaded key):
// Grants permission to write .pcap files to the incoming GCS bucket and to publish notifications to the Topic.

resource "google_pubsub_topic_iam_member" "sniffer_sa_pubsub_publisher" {
  project = var.gcp_project_id
  topic   = module.pubsub_topic.topic_id
  role    = "roles/pubsub.publisher"
  member  = "serviceAccount:${google_service_account.sniffer_sa.email}"
}

resource "google_storage_bucket_iam_member" "sniffer_sa_gcs_writer" {
  bucket = module.gcs_buckets.incoming_pcap_bucket_id
  role   = "roles/storage.objectCreator" // Allows creating objects in the bucket
  member = "serviceAccount:${google_service_account.sniffer_sa.email}"
}
```

Cloud Run Processor Implementation

Receive & Download

Flask app receives Pub/Sub notifications and downloads PCAP from GCS.



Convert to JSON

Executes tshark to transform PCAP into raw JSON representation.

Upload Results

Uploads processed UDM JSON to output bucket and logs metrics.

Transform to UDM

Streams JSON with ijson library to create standardized UDM format.

Processor: Dockerfile

```
FROM python:3.9-slim

RUN apt-get update && apt-get install -y --no-install-recommends \
    tshark \
    && rm -rf /var/lib/apt/lists/*

COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

WORKDIR /app
COPY processor_app.py .
COPY json2udm_cloud.py .

ENV PYTHONUNBUFFERED=1

CMD exec gunicorn --bind :${PORT:-8080} --workers 1 --threads 8 --timeout 600 processor_app:app
```

Copying **2 main files**: the **entrypoint** and the cybersecurity project's **parser**

Processor.py (Cloud Run entrypoint)

Cloud Run waked up by **Pub/Sub** notification (*when a pcap is available*)

```
# 1. Download pcap from GCS
logging.info(f"Downloading gs://{INCOMING_BUCKET_NAME}/{pcap_filename} to {local_pcap_path}")
active_storage_client.bucket(INCOMING_BUCKET_NAME).blob(pcap_filename).download_to_filename(local_pcap_path)
logging.info(f"Download complete for {pcap_filename}.") # Confirmation for success metric
```

First raw conversion to JSON...

```
# 2. Convert pcap to JSON (tshark)
logging.info(f"Converting {local_pcap_path} to JSON...")
tshark_command = ["tshark", "-r", local_pcap_path, "-T", "json"]
with open(local_json_path, "w") as json_file:
    process = subprocess.run(tshark_command, stdout=json_file, stderr=subprocess.PIPE, text=True, check=True)
logging.info(f"tshark conversion successful: {local_json_path}")
if process.stderr: logging.warning(f"tshark stderr: {process.stderr.strip()}")
```

Processor.py

Then the UDM-compliant conversion with **json2udm.py** (from **cybersecurity** project)

```python
# 3. Convert JSON to UDM (json2udm_cloud.py)
logging.info(f"Converting {local_json_path} to UDM: {local_udm_path}")
udm_script_command = ["python3", "/app/json2udm_cloud.py", local_json_path, local_udm_path]
process = subprocess.run(udm_script_command, capture_output=True, text=True, check=True)
logging.info(f"UDM conversion script done for {pcap_filename}.") # Confirmation
if process.stdout: logging.info(f"json2udm_cloud.py stdout: {process.stdout.strip()}")
if process.stderr: logging.warning(f"json2udm_cloud.py stderr: {process.stderr.strip()}")

if not os.path.exists(local_udm_path) or os.path.getsize(local_udm_path) == 0:
    logging.error(f"UDM file {local_udm_path} missing or empty post-conversion for {pcap_filename}.")
    return "Internal Server Error: UDM generation failed.", 500 # Retry
```

And the upload to completed bucket

```python
# 4. Upload UDM JSON to GCS
logging.info(f"Uploading {local_udm_path} to gs://{OUTPUT_BUCKET_NAME}/{udm_output_filename}")
active_storage_client.bucket(OUTPUT_BUCKET_NAME).blob(udm_output_filename).upload_from_filename(local_udm_path)
logging.info(f"Upload complete for {udm_output_filename}.") # Confirmation

processing_end_time = datetime.now(timezone.ut
processing_duration_seconds = (processing_end
logging.info(f"PROCESSING_DURATION_SECONDS: {processing_duration_seconds:.3f} FILE: {pcap_filename}")
```

```
(variable) processing_duration_seconds: float
```

Json2udm_cloud.py

```
with open(json_file_path, 'rb') as f_json:
    # `ijson.items(f_json, 'item')` assumes the JSON is an array of objects at the root.
    # 'item' tells ijson to yield each element of that root array.
    json_packet_iterator = ijson.items(f_json, 'item')
    for packet_data_dict in json_packet_iterator:
        udm_event = convert_single_packet_to_udm(packet_data_dict)
        udm_events_list.append(udm_event)
        processed_packet_count += 1
        # Check if the generated UDM event was an error event
        if "PacketProcessingError" in udm_event.get("event", {}).get("metadata", {}).get("product_name", ""):
            error_event_count += 1

logging.info(f"Successfully converted {processed_packet_count} packets from JSON to UDM format for file {os.path.basename(json_file_path)}.")
logging.info(f"UDM_PACKETS_PROCESSED: {processed_packet_count} FILE: {os.path.basename(json_file_path)}")

if error_event_count > 0:
    logging.warning(f"{error_event_count} packets encountered processing errors [...] {os.path.basename(json_file_path)}.")
    logging.warning(f"UDM_PACKET_ERRORS: {error_event_count} FILE: {os.path.basename(json_file_path)}")
```

Uses a "small version" of old script on a packet at time, reading them as a stream

From Local Batch to Cloud-Native Streaming

Original Approach

- Load entire JSON file into memory
- Process all packets at once
- Only local file operations
- Slower

Cloud Improvements

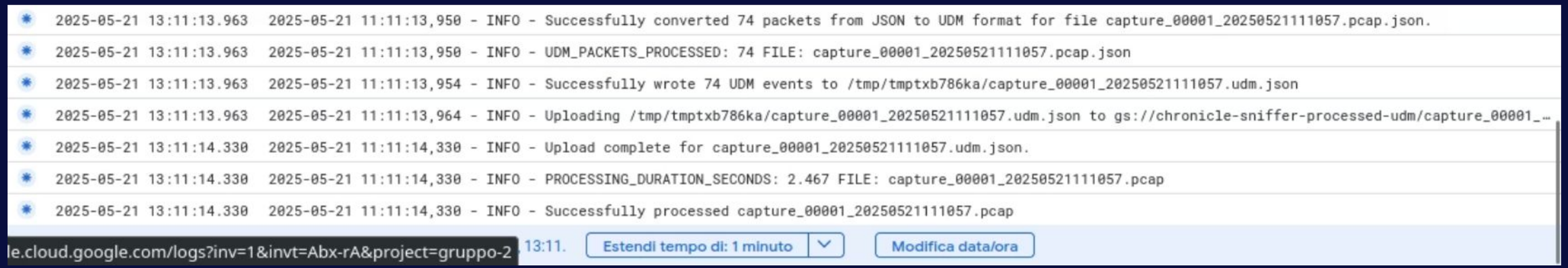
- Stream processing with ijson
- Packet-by-packet handling
- Enhanced error recovery
- Refined UDM structure

These adaptations transformed a local tool into a scalable, cloud-native component.

Json2udm_cloud.py: output

```
# Outputting to a single file. The previous script had `write_to_multiple_files` which isn't needed here,
# as the next step in a GCP environment would likely be to upload this single file to GCS or send its contents via API to Chronicle
try:
    # Ensure output directory exists, especially if output_file_path includes subdirectories
    output_directory = os.path.dirname(output_file_path)
    if output_directory and not os.path.exists(output_directory): # Check if dirname is not empty
        os.makedirs(output_directory, exist_ok=True)
        logging.info(f"Created output directory: {output_directory}")

    with open(output_file_path, "w") as f_out: # 'w' for text mode, as json.dump expects string data
        json.dump(udm_event_list_result, f_out, indent=4) # Pretty print for readability

    if udm_event_list_result: # Only log success if events were actually written
        logging.info(f"Successfully wrote {len(udm_event_list_result)} UDM events to {output_file_path}")
    else:
        # This case covers ijson errors or if the input JSON was empty/invalid leading to no UDM events
        logging.warning(f"No UDM events were generated (or stream parsing failed). Wrote an empty list to {output_file_path}.")

except Exception as e_write:
    # This is a critical error if we can't even write the output.
    logging.critical(f"CRITICAL ERROR: Failed to write UDM output to '{output_file_path}': {e_write}", exc_info=True)
    sys.exit(1) # Exit with error if output fails
```

Processor: GCP logs



Cloud Run Log's Snippet on Google CLoud Platform

Processor: resource definition

```
// Cloud Run Processor Module:
// Deploys the PCAP processing application as a Cloud Run service.
// Configures the service with necessary environment variables, resources, and scaling settings.
module "cloudrun_processor" {
  source                = "./modules/cloudrun_processor"
  project_id            = var.gcp_project_id
  region                = var.gcp_region
  service_name          = "${var.base_name}-processor"
  image_uri             = var.processor_cloud_run_image
  service_account_email = google_service_account.cloud_run_sa.email
  env_vars = {
    INCOMING_BUCKET = module.gcs_buckets.incoming_pcap_bucket_id
    OUTPUT_BUCKET   = module.gcs_buckets.processed_udm_bucket_id
    GCP_PROJECT_ID  = var.gcp_project_id
  }
  max_concurrency = var.cloud_run_max_concurrency
  cpu_limit       = var.cloud_run_cpu
  memory_limit    = var.cloud_run_memory
}
```

a snippet of main.tf

Processor & Pub/Sub: IAM

```
// Cloud Run Invocation Permissions:
// Controls who can invoke the Cloud Run processor service.
// By default, restricted to OIDC-authenticated Pub/Sub calls via the Cloud Run SA.
// Optionally allows unauthenticated invocations if var.allow_unauthenticated_invocations is true

resource "google_cloud_run_v2_service_iam_member" "allow_unauthenticated" {
  count      = var.allow_unauthenticated_invocations ? 1 : 0
  project    = var.gcp_project_id
  name       = module.cloudrun_processor.service_name
  location   = module.cloudrun_processor.service_location
  role       = "roles/run.invoker"
  member     = "allUsers" // Allows public access if enabled
  depends_on = [module.cloudrun_processor]
}

resource "google_cloud_run_v2_service_iam_member" "allow_pubsub_oidc_invoker" {
  count      = !var.allow_unauthenticated_invocations ? 1 : 0
  project    = var.gcp_project_id
  name       = module.cloudrun_processor.service_name
  location   = module.cloudrun_processor.service_location
  role       = "roles/run.invoker"
  member     = "serviceAccount:${google_service_account.cloud_run_sa.email}" // Allows invocation
  depends_on = [module.cloudrun_processor, google_service_account.cloud_run_sa]
}
```

var `allow_unauthenticated_invocations` :

Authenticated Access

- Default behavior
- Pub/Sub calls verified through Cloud Run service account.
- OIDC token validation

Unauthenticated Access

- Public endpoint access
- Simplified testing

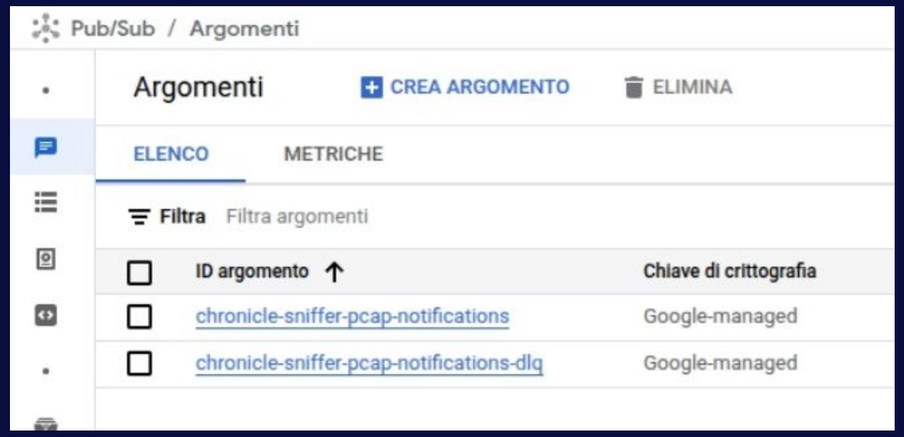
Pub/Sub topics

```
// --- Pub/Sub Subscription ---
// Creates a push subscription to the PCAP notifications topic.
// This subscription delivers messages to the Cloud Run processor service endpoint.
// Configured with OIDC authentication (if var.allow_unauthenticated_invocations is false)
// and a dead-letter policy for unprocessable messages.
resource "google_pubsub_subscription" "processor_subscription" {
  project              = var.gcp_project_id
  name                 = "${var.base_name}-processor-sub"
  topic                = module.pubsub_topic.topic_id // Main topic for PCAP notifications
  ack_deadline_seconds = 600                         // Time allowed for Cloud Run to process a message

  push_config {
    push_endpoint = module.cloudrun_processor.service_url // Cloud Run service URL
    dynamic "oidc_token" {
      for_each = !var.allow_unauthenticated_invocations ? [1] : []
      content {
        service_account_email = google_service_account.cloud_run_sa.email // SA used for OIDC token generation
        audience              = module.cloudrun_processor.service_url     // Audience for the OIDC token
      }
    }
  }

  dead_letter_policy {
    dead_letter_topic     = module.pubsub_topic.dlq_topic_id // DLQ topic for failed messages
    max_delivery_attempts = 5                                // Max retries before sending to DLQ
  }

  depends_on = [
    module.cloudrun_processor,
    google_cloud_run_v2_service_iam_member.allow_unauthenticated,
    google_cloud_run_v2_service_iam_member.allow_pubsub_oidc_invoker,
    google_service_account_iam_member.pubsub_sa_token_creator_for_cloud_run_sa,
    module.pubsub_topic // Ensures DLQ topic exists if the module creates it
  ]
}
```

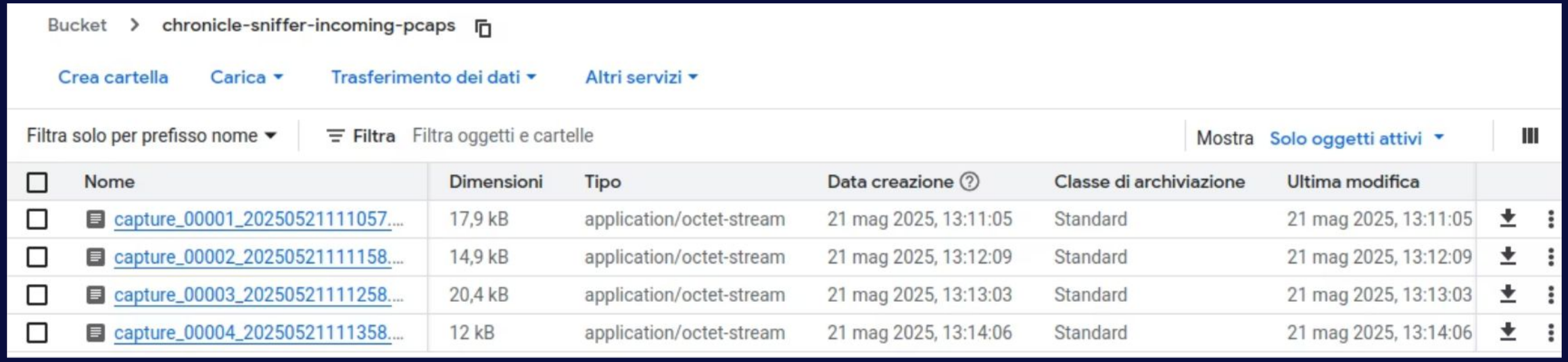
Normal push

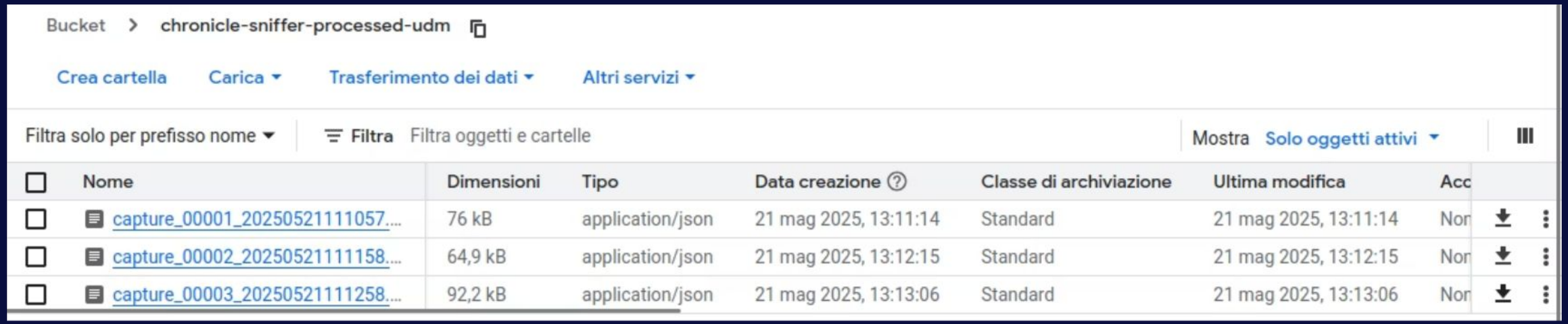
&

Dead Letter Queue



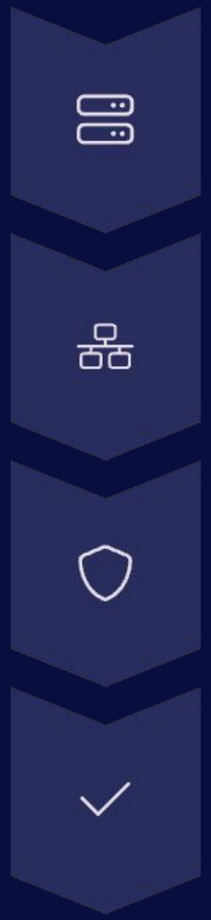
GCP: buckets


Bucket > chronicle-sniffer-incoming-pcaps
Crea cartella
Carica
Trasferimento dei dati
Altri servizi
Filtra solo per prefisso nome
Filtra
Filtra oggetti e cartelle
Mostra
Solo oggetti attivi
Nome
Dimensioni
Tipo
Data creazione
Classe di archiviazione
Ultima modifica
capture_00001_20250521111057.... 17,9 kB application/octet-stream 21 mag 2025, 13:11:05 Standard 21 mag 2025, 13:11:05
capture_00002_20250521111158.... 14,9 kB application/octet-stream 21 mag 2025, 13:12:09 Standard 21 mag 2025, 13:12:09
capture_00003_20250521111258.... 20,4 kB application/octet-stream 21 mag 2025, 13:13:03 Standard 21 mag 2025, 13:13:03
capture_00004_20250521111358.... 12 kB application/octet-stream 21 mag 2025, 13:14:06 Standard 21 mag 2025, 13:14:06


Bucket > chronicle-sniffer-processed-udm
Crea cartella
Carica
Trasferimento dei dati
Altri servizi
Filtra solo per prefisso nome
Filtra
Filtra oggetti e cartelle
Mostra
Solo oggetti attivi
Nome
Dimensioni
Tipo
Data creazione
Classe di archiviazione
Ultima modifica
Acc
capture_00001_20250521111057.... 76 kB application/json 21 mag 2025, 13:11:14 Standard 21 mag 2025, 13:11:14 Nor
capture_00002_20250521111158.... 64,9 kB application/json 21 mag 2025, 13:12:15 Standard 21 mag 2025, 13:12:15 Nor
capture_00003_20250521111258.... 92,2 kB application/json 21 mag 2025, 13:13:06 Standard 21 mag 2025, 13:13:06 Nor

TestVM: validates pipeline through realistic simulation

This Terraform-defined component replicates **on-premises edge environments** where the sniffer operates. It uses **tcpreplay** to replay PCAP files and **hping3** to simulate network attacks like SYN floods.



VM Initialization

Startup script configures Docker environment with GCP credentials and network settings

Traffic Generation

Tools like tcpreplay and hping3 create controlled network scenarios for testing

Attack Simulation

SYN flood attacks test sniffer response to anomalous traffic patterns

End-to-End Validation

Complete pipeline testing from packet capture to UDM format conversion

TestVM: Resouce definition

```
// Provisions a GCE instance to simulate an on-premises sniffer environment for testing purposes.
// The startup script prepares the VM with Docker and pulls the sniffer image.
module "test_generator_vm" {
  source            = "./modules/test_generator_vm"
  project_id        = var.gcp_project_id
  zone              = var.test_vm_zone
  vm_name           = "${var.base_name}-sniffer-vm"
  ssh_source_ranges = var.ssh_source_ranges

  attached_service_account_email = google_service_account.test_vm_sa.email
  startup_script_path            = "${path.module}/modules/test_generator_vm/startup_script_vm.sh"

  sniffer_image_uri_val       = var.sniffer_image_uri
  sniffer_gcp_project_id_val  = var.gcp_project_id
  sniffer_incoming_bucket_val = module.gcs_buckets.incoming_pcap_bucket_id
  sniffer_pubsub_topic_id_val = module.pubsub_topic.topic_id // Assumes module.pubsub_topic outputs 'topic_id' (full path)
  sniffer_id_val              = var.test_vm_sniffer_id

  depends_on = [
    google_service_account.sniffer_sa, // Sniffer SA should exist before VM setup references it implicitly via outputs
    module.gcs_buckets,
    module.pubsub_topic
```

a snippet of main.tf

Easy testing

Output of:

```
┌──(fillo@spike)-[~/Documenti/chronicle-sniffer/terraform]
└─$ terraform apply
```

All required values are printed, thanks to the definitions in outputs.tf

```
Apply complete! Resources: 31 added, 0 changed, 0 destroyed.

Outputs:

cloud_run_service_account_email = "chronicle-sniffer-run-sa@gruppo-2.iam.gservicea
generate_sniffer_key_command = "gcloud iam service-accounts keys create ../sniffer
incoming_pcap_bucket_id = "chronicle-sniffer-incoming-pcaps"
processed_udm_bucket_id = "chronicle-sniffer-processed-udm"
processor_cloud_run_service_url = "https://chronicle-sniffer-processor-[redacted]
pubsub_subscription_id = "projects/gruppo-2/subscriptions/chronicle-sniffer-proces
pubsub_topic_id = "projects/gruppo-2/topics/chronicle-sniffer-pcap-notifications"
sniffer_service_account_email = "chronicle-sniffer-snfr-sa@gruppo-2.iam.gserviceac
test_generator_vm_ip = "[redacted]"
test_generator_vm_name = "chronicle-sniffer-sniffer-vm"
test_vm_service_account_email = "chronicle-sniffer-testvm-sa@gruppo-2.iam.gservice
test_vm_sniffer_setup_instructions = <<EOT
```

Additionally, you'll find a small “how-to” guide for testing the project with the TestVM:

```
INSTRUCTIONS FOR THE SNIFFER ON THE TEST VM ('chronicle-sniffer-sniffer-vm'):
-----------------------------------------------------------------------------------------------
The base environment on the VM has been prepared by the startup script:
- Docker and Docker Compose are installed.
- The sniffer Docker image ('fillol/chronicle-sniffer:latest') has been pulled.
- Configuration files for the sniffer are in '/opt/sniffer_env' on the VM.
  This includes a base 'docker-compose.yml' and a 'docker-compose.override.yml'
  which configures volumes and network settings specifically for the VM.

To run the sniffer, follow these steps:

1.  PREPARE THE SNIFFER'S SERVICE ACCOUNT KEY (on YOUR LOCAL MACHINE):
    Run the gcloud command shown in the Terraform output named 'generate_sniffer_key_command'.
    This command will create (or overwrite) 'key.json' for the Service Account 'chronicle-sniffer-snfr-sa@gruppo-2.iam.gserviceaccount.com'.
    (The command will be similar to: gcloud iam service-accounts keys create ../sniffer/gcp-key/key.json --iam-account=chronicle-sniffer-snfr-sa@gruppo-2.ia

2.  ACCESS THE TEST VM VIA SSH (from YOUR LOCAL MACHINE):
    gcloud compute ssh --project gruppo-2 --zone europe-west8-b chronicle-sniffer-sniffer-vm

3.  COPY THE SA KEY TO THE VM (from a NEW terminal on YOUR LOCAL MACHINE):
    The VM's startup script has created the directory '/opt/gcp_sa_keys/sniffer' with open permissions.
    Copy the 'key.json' file (generated in step 1) to this directory on the VM, ensuring it is named 'key.json':

    gcloud compute scp ../sniffer/gcp-key/key.json chronicle-sniffer-sniffer-vm:/opt/gcp_sa_keys/sniffer/key.json --project gruppo-2 --zone europe-west8-b

    (Note: If you encounter permission issues with 'gcloud compute scp' directly to /opt/, you can first copy the key to the VM's home directory:
       gcloud compute scp ../sniffer/gcp-key/key.json chronicle-sniffer-sniffer-vm:~/key.json --project gruppo-2 --zone europe-west8-b
     Then, connect to the VM via SSH (step 2) and move the file:
       sudo mv ~/key.json /opt/gcp_sa_keys/sniffer/key.json
     Ensure the final path on the VM is '/opt/gcp_sa_keys/sniffer/key.json'.)

4.  START THE SNIFFER (inside the SSH session on the VM):
    Navigate to the sniffer's environment directory and use Docker Compose:
    cd /opt/sniffer_env
    sudo docker-compose up -d

5.  CHECK SNIFFER LOGS (inside the SSH session on the VM):
    sudo docker logs chronicle-sniffer -f
    You should see logs indicating the activation of the SA 'chronicle-sniffer-snfr-sa@gruppo-2.iam.gserviceaccount.com' and tshark starting its capture.

6.  GENERATE NETWORK TRAFFIC ON THE VM FOR TESTING (inside the SSH session):
    ping -c 20 google.com
    curl http://example.com

7.  VERIFY THE PIPELINE:
    *   Sniffer logs: Look for GCS upload and Pub/Sub publish messages.
    *   GCS Bucket 'chronicle-sniffer-incoming-pcaps': .pcap files should appear.
    *   Cloud Run logs for 'chronicle-sniffer-processor': Notification received and processing messages.
    *   GCS Bucket 'chronicle-sniffer-processed-udm': .udm.json files should appear.

8.  TO STOP THE SNIFFER (inside the SSH session on the VM):
    cd /opt/sniffer_env
    sudo docker-compose down

9.  TO CLEAN UP ALL GCP RESOURCES (from YOUR LOCAL MACHINE, in the terraform directory):
    terraform destroy
    (Remember to also delete the local SA key file './key.json').
```

Observability and Monitoring

Cloud Logging

Structured logs from sniffer and processor centralized in Google Cloud Logging.

Log-Based Metrics

Custom metrics track heartbeats, file operations, packet counts, and errors.

Operational Dashboard

Comprehensive view of pipeline health using Monitoring Query Language (MQL).

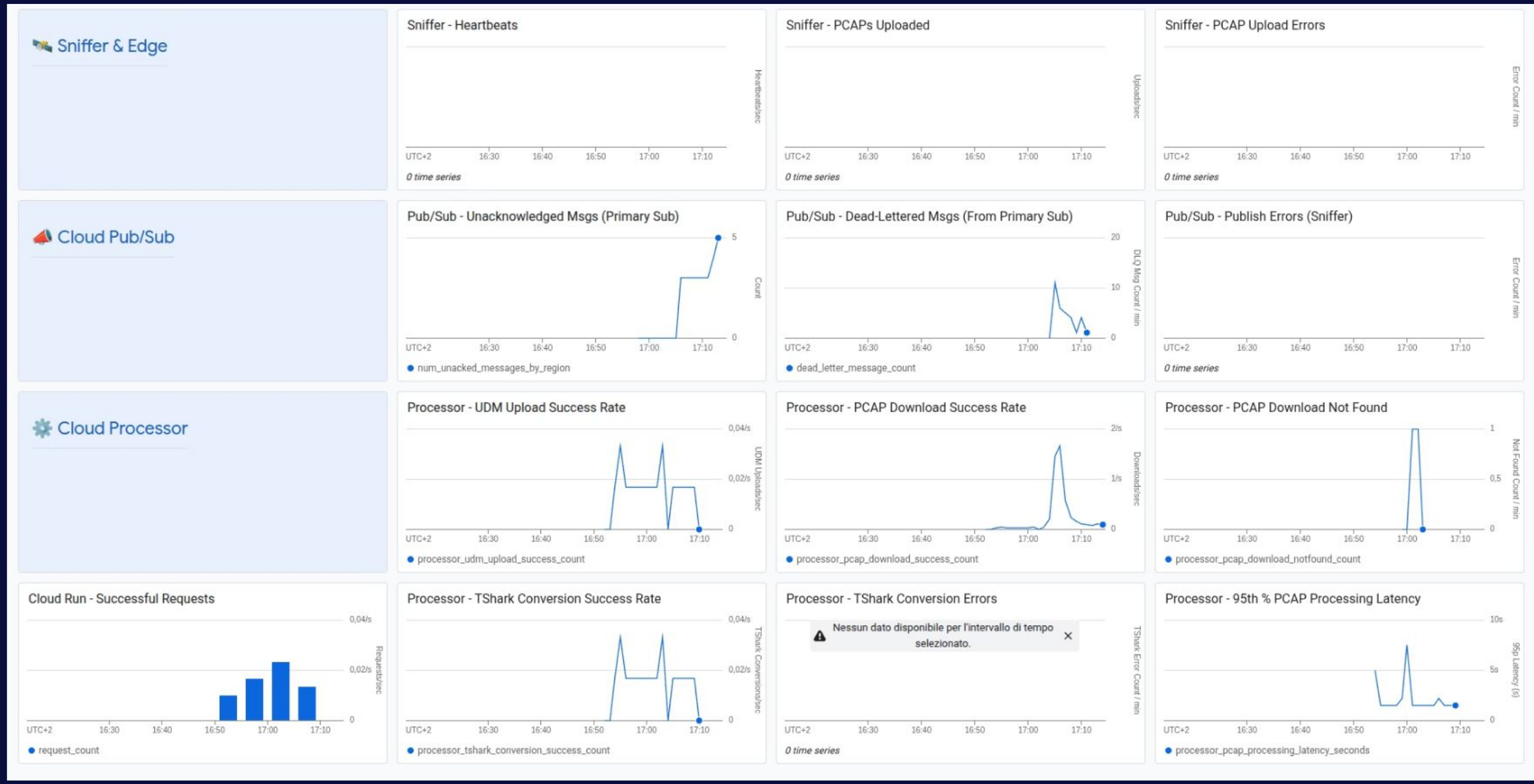
Log metrics in Terraform

```
// Processor TShark Conversion Success Metric: Counts successful tshark conversions.
resource "google_logging_metric" "tshark_conversion_success_processor" {
  project     = var.gcp_project_id
  name        = "processor_tshark_conversion_success_count"
  filter      = "resource.type=\"cloud_run_revision\" AND textPayload=~\"INFO - tshark conversion successful:\""
  description = "Counts successful TShark conversions by the processor."

  metric_descriptor {
    metric_kind  = "DELTA"
    value_type   = "INT64"
    unit         = "1"
    display_name = "Processor TShark Conversion Success"
  }
}
```

an example of metric defined in terraform's main.tf

Operational Dashboard

Deployable via terraform, exported in JSON



Operational Dashboard: resource definition

```
// --- Cloud Monitoring Dashboard ---
// Defines the operational dashboard using a JSON template file.
// This dashboard visualizes the custom metrics and standard GCP service metrics.
resource "google_monitoring_dashboard" "main_operational_dashboard" {
  project = var.gcp_project_id
  dashboard_json = templatefile("${path.module}/dashboards/main_operational_dashboard.json", {
    cloud_run_processor_service_name = module.cloudrun_processor.service_name,
    pubsub_processor_subscription_id = google_pubsub_subscription.processor_subscription.name
    // Add other variables here if the dashboard template needs them.
  })

  depends_on = [ // Ensure metrics are created before the dashboard attempts to use them.
    google_logging_metric.sniffer_heartbeat_metric,
    google_logging_metric.pcap_files_uploaded_metric,
    // google_logging_metric.processor_udm_packets_processed_metric, // RIMOSSO
    google_logging_metric.pcap_upload_errors_metric,
    // google_logging_metric.sniffer_tshark_status_running_count,    // RIMOSSO
    // google_logging_metric.pcap_file_size_metric,                  // RIMOSSO
    google_logging_metric.pubsub_publish_errors_metric,
    google_logging_metric.pcap_download_success_processor,
    google_logging_metric.pcap_download_notfound_processor,
    google_logging_metric.tshark_conversion_success_processor,
    google_logging_metric.tshark_conversion_error_processor,
    // google_logging_metric.udm_packet_processing_errors,           // RIMOSSO
    google_logging_metric.udm_upload_success_processor,
    google_logging_metric.processor_pcap_latency
  ]
}
```

Security Considerations

This pipeline applies **defense-in-depth security** with least privilege IAM and granular service account permissions.

Network isolation safeguards edge sniffers from unauthorized access. Containers are scanned before deployment, and data transmissions are encrypted for integrity.

The system features least-privilege IAM, **OIDC-secured** invocations, secure key management, and protected storage. Firewall rules limit access to specific IP ranges (for test-vm access).

The Test Generator VM supports **threat simulation** using hping3 and tcpreplay for validating detection.

Comprehensive logging audits security events. Dead letter queues prevent data loss.

One last (complete) Overview

Educational Value & Cloud-Native Principles

Managed Services

Leveraging GCP's services reduces operational overhead

Infrastructure as Code

Terraform ensures consistent, repeatable deployments

Scalability & Decoupling

Event-driven architecture with independent components

Security & Observability

OIDC authentication and comprehensive monitoring

Project Conclusion & Key Outcomes

This project successfully demonstrated the practical implementation of modern cloud-native principles, culminating in a **scalable**, **secure**, and **observable** packet capture to Chronicle pipeline

Core Achievements & Demonstrations:

Cloud-Native Architecture: From local batch to optimized streaming

Resilient Design: Event-driven (Pub/Sub) for decoupling; Serverless (Cloud Run) for auto-scaling

Efficient Resource Management: `ijson` streaming for cloud memory adaptation

Industry Best Practices: Security (Least Privilege IAM) & end-to-end monitoring

Thank You.

Project by **Filippo Lucchesi**