COLEGIO DE CIENCIAS E INGENIERÍAS

INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN

Entregable I del Proyecto Integrador

Tutor: Roberto Andrade

Autor: José Contreras

Quito – Ecuador

2025

1. Título del Proyecto:

   Orquestador de Agentes de IA para gestión de incidentes de ciberseguridad

2. Resumen de actividades realizadas:

   1. Revisión del estado del arte: se realizó una investigación contemplada en los artículos científicos de los últimos 5 años a través de Elicit. Se identificaron las principales arquitecturas (SIEM/SOAR) dedicadas a la gestión de incidentes de ciberseguridad a nivel de SOC. Entre sus principales limitaciones se encuentra el costo de las licencias propietarias, la rigidez en la personalización y la necesidad de atención humana. En contraste, los enfoques recientes que combinan LLMs, RAG y automatización de respuesta muestran mejoras en clasificación de ataques y reducción de tiempos de triaje. Con base en ello, se justifica una arquitectura orientada a agentes más accesible, automatizable y explicable para la gestión de incidentes en un SOCS.

   2. Definición inicial de un esquema canónico de eventos (CEC) y taxonomías de severidad: a través de la revisión del estado del arte se identificaron 8 campos canónicos que permiten una ingesta de datos genérica respecto de la fuente, priorizar los incidentes, actuar en respuesta, y explicar o auditar en función de métricas medibles. El CEC incluye los campos: event_id, timestamp, source, rule_id, severity (1–5), asset_id, confidence (0–1) y raw_event. Esta habilita ingestión heterogénea con bajo costo de mapeo, priorización consistente y trazabilidad forense. Para la severidad, se adoptó una escala normalizada de 1 a 5 (1=informativo, 5=crítico).

   3. Implementación de prototipo de agente de monitoreo: el agente se encuentra en proceso de construcción en n8n, busca ingerir y normalizar eventos para luego publicarlos en una cola teniendo así un flujo encadenado y auditable. Inicia con un disparador (Manual/Schedule) que recupera eventos desde un dataset de prueba (y, en etapas posteriores, desde APIs o archivos). Un nodo de transformación que aplica el mapeo al CEC, garantizando que cada elemento cuente con event_id, timestamp, source, severity, asset_id, confidence y raw_event. Luego, una etapa de persistencia almacena los eventos normalizados (CSV/DB) junto con un registro de auditoría para trazabilidad. Finalmente, el flujo publica los eventos normalizados hacia el siguiente eslabón (análisis y priorización) mediante un sub-workflow o una cola lógica; todo ello protegido por un “kill-switch” (variable AUTO_MODE) que asegura que, en esta fase de prototipo, las ejecuciones sean controladas y reproducibles sin activar acciones destructivas. Con este diseño, el agente de monitoreo proporciona datos consistentes, medibles y listos para evaluación.

      Con el fin de proporcionar una visión modular y escalable de la arquitectura planteada, se ha desarrollado un modelo bajo el estándar C4, que permite describir el sistema en distintos niveles de detalle: lógico, de desarrollo y de despliegue. Este enfoque facilita tanto la comprensión general del sistema como la reproducibilidad técnica del prototipo.

Nivel Lógico (C1 – System Context)

A continuación, se describen las principales interacciones entre los agentes que conforman el sistema:

- Agente de Monitoreo: encargado de la ingesta de datos provenientes de distintas fuentes (SIEM, registros de red). Este agente normaliza los eventos en un Esquema Canónico de Eventos (CEC), elimina duplicados y persiste los resultados.

- Agente de Análisis: recibe los eventos normalizados y ejecuta procesos de correlación mediante la matriz MITRE ATT&CK, enriquecimiento con indicadores de compromiso (IOC), y clasificación de incidentes mediante modelos de lenguaje apoyados en LangGraph y RAG. Asigna un nivel de confianza y severidad a cada evento.

- Agente de Respuesta: implementa playbooks (manual de ejecución) de automatización (bloqueo de IP, cuarentena de endpoints, cierre de puertos o reinicio de credenciales) mediante n8n. Para incidentes de mayor criticidad se contempla la supervisión de un operador humano (Human-in-the-Loop).

- Agente Orquestador: consolida las métricas operacionales (tiempo de detección, tiempo de respuesta, tasa de falsos positivos), coordina el flujo Monitor–Analizar–Responder y genera tableros ejecutivos y operativos de seguimiento.

La interacción entre agentes es secuencial, con un ciclo de retroalimentación en el cual las métricas obtenidas alimentan nuevamente al agente de monitoreo, cerrando así un proceso iterativo y medible. A continuación, se muestra un diagrama con la arquitectura preliminar que muestra las interacciones de los agentes.

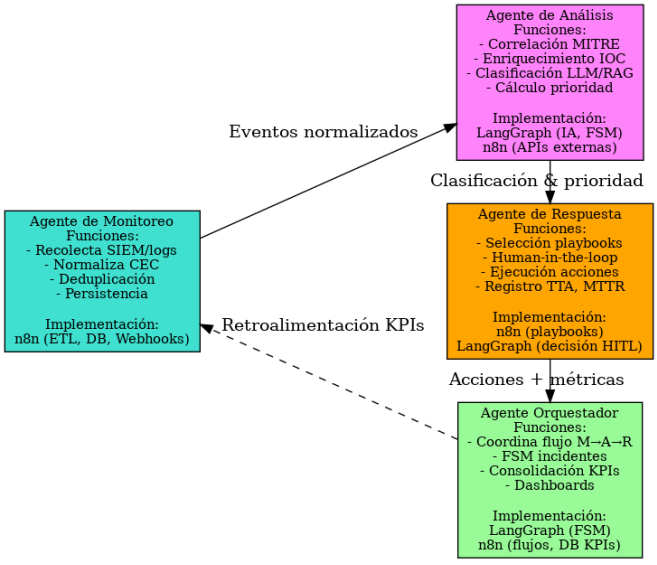

Figura 1. Esquema preliminar de arquitectura de Agentes

Nivel de Desarrollo (C2 – Container Diagram)

En el nivel de desarrollo se especifican las tecnologías y componentes que permiten la implementación de cada agente:

- Agente de Monitoreo: implementado en n8n con nodos de *Schedule Trigger* (para ejecución periódica), *HTTP Request* (extracción de datos), transformación en JavaScript (mapeo al CEC) y persistencia en Postgres.

- Agente de Análisis: combina la ejecución de correlación de reglas en n8n con la coordinación de flujos en LangGraph, enriquecimiento con APIs de reputación de IOC y clasificación de incidentes mediante un esquema LLM/RAG.

- Agente de Respuesta: integra playbooks en n8n (bloqueo de IP, cuarentena de endpoint, cierre de puertos, reseteo de credenciales). Para incidentes críticos se emplea LangGraph como capa de decisión HITL (Human-in-the-Loop).

- Agente Orquestador: utiliza LangGraph para consolidar KPIs a través de máquinas de estados finitos (FSM), Postgres como Data Mart de métricas y n8n para generar tableros integrables con sistemas de visualización (podrían ser Grafana o Metabase).

Este nivel evidencia la integración de tres componentes fundamentales: n8n para la orquestación de flujos, LangGraph para la coordinación de agentes con soporte de IA, y Postgres como repositorio de datos y métricas.

Nivel de Despliegue (C3 – Deployment Diagram)

En este nivel de despliegue, procederemos a tratar la infraestructura actual y la proyectada para fases posteriores:

- Fase de prototipo local: el sistema se ejecuta en contenedores Docker que alojan tanto n8n como Postgres. Los agentes de análisis se conectan a APIs externas y a modelos LLM/RAG. Los tableros operativos se generan de forma local.

A continuación, se presenta un flujograma que representa el proceso abstracto que se desea automatizar e integrar mediante n8n y LangGraph.

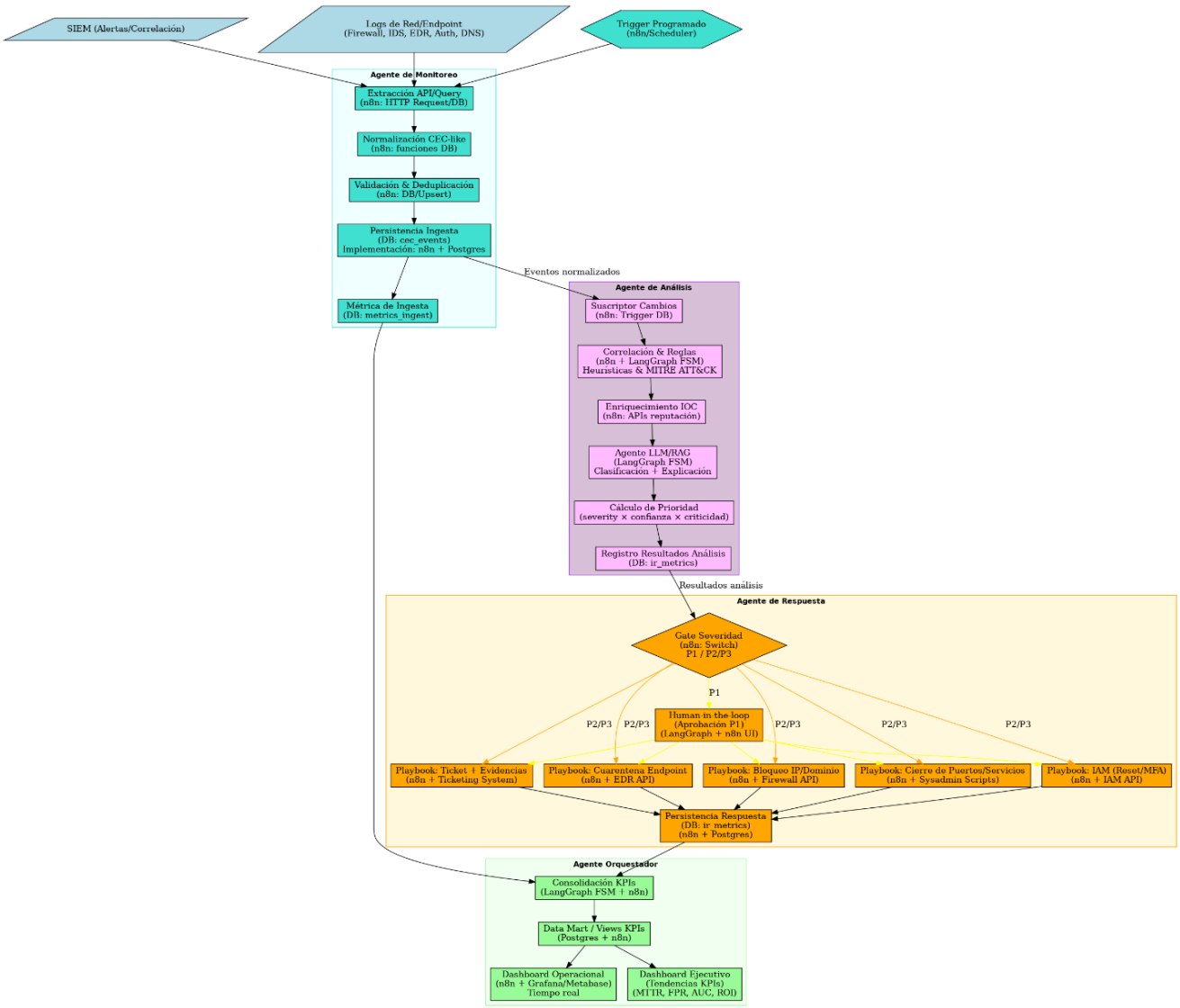

Figura 2. Flujograma general

- Fase futura (despliegue híbrido/nube): se prevé, según posibilidad y pertinencia migrar a un esquema híbrido donde:
    - n8n se ejecute en contenedores orquestados (Docker Compose o Kubernetes).
    - Postgres se traslade a un servicio gestionado en la nube (ej. AWS RDS).
    - LangGraph se ejecute en servidores cloud para permitir la coordinación distribuida.
    - Los dashboards ejecutivos se publiquen en infraestructura web accesible desde el SOC.

Este despliegue progresivo garantiza escalabilidad, modularidad y adaptabilidad frente a diferentes entornos de operación (local, nube o híbrido).

Nivel de Código (C4 – Code Diagram)

A continuación, se presentan las configuraciones, hasta el momento, de los distintos nodos del agente de monitoreo. Este agente se encuentra en fase preliminar de construcción en n8n. El diagrama es:

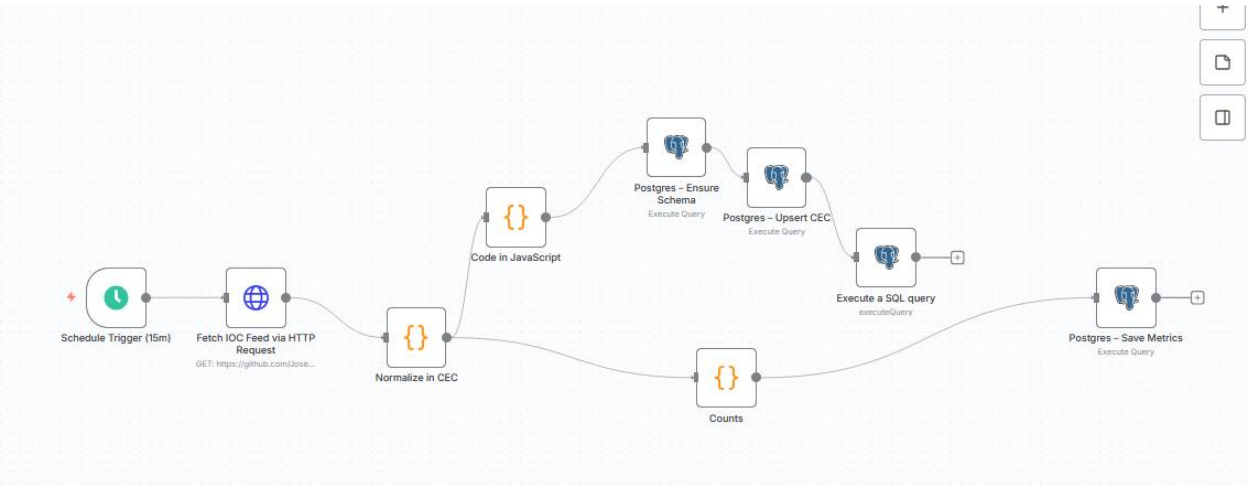

Figura 3. Diagrama preliminar de flujo agente de monitoreo en n8n

Nodo Trigger

El nodo de disparo presenta la siguiente configuración en n8n:

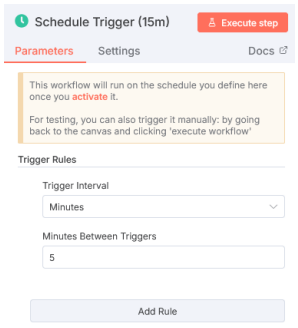

Figura 4. Configuración nodo disparador

El nodo disparador en n8n pide configurar el formato de tiempo para el intervalo, es decir, horas, minutos, etc. Y también despliega un textfield para la cantidad de tiempo para el intervalo en el que se ejecutará este disparador.

Nodo de Fetch

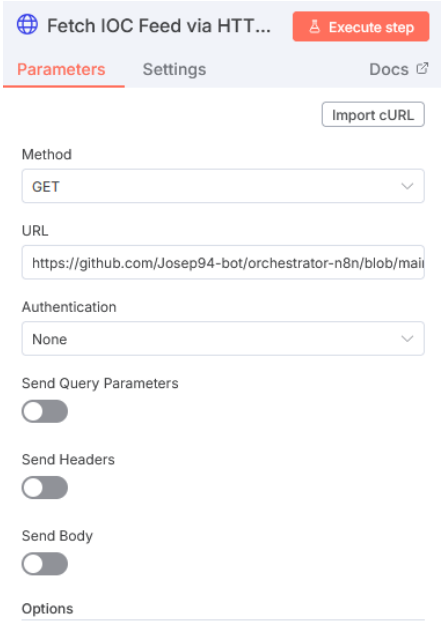

Figura 5. Nodo Fetch/ HTTP request

El nodo de Fetch incluye configuraciones para ejecutar operaciones como Delete, Get, Head, Post entre otras, esto nos permite como en este caso obtener información del link del repositorio de github en donde se encuentra el archivo de muestra a ingerir.

Nodo de Normalización en CEC

Este nodo nos permite ejecutar código JSON con el propósito de modificar el archivo ingerir y normalizar la información en términos de las variables del CEC. A continuación, se muestra el código.

```
function toEpoch(ts){ try { return Math.floor(new Date(ts).getTime()); } catch(e){ return Date.now(); } }
return items.map((item, idx) => {
  const e = item.json || {};
  const indicator = e.indicator || e.ip || e.domain || e.url || e.hash || e.host || e.asset_id;
  const ts = e.timestamp || e.created || e.date || new Date().toISOString();
  const sevRaw = (typeof e.severity === 'number') ? e.severity : (typeof e.score === 'number' ? e.score : 3);
  let confidence = 0.5;
  if (typeof e.confidence === 'number') confidence = Math.max(0, Math.min(1, e.confidence));
  else if (typeof e.reputation === 'number') confidence = Math.max(0, Math.min(1, e.reputation/100));
  const event_id = e.event_id || e.id || null;
  return { json: {
    event_id,
    timestamp: ts,
    ts_epoch: toEpoch(ts),
    source: e.source || 'otx',
    rule_id: e.rule_id || e.signature_id || e.rule || null,
    severity: Math.max(1, Math.min(5, Math.round(sevRaw))),
    asset_id: e.asset_id || indicator || 'unknown',
    confidence,
    raw_event: e
  }};
```

Figura 6. Código JSON de Nodo Normalización

Nodo Contador

Este nodo mantiene la función principal de acumular y reportar la cantidad de elementos que han sido recibidos por el nodo. Es de importancia para el funcionamiento del nodo Postgres -save metris-.

Figura 7. Código JSON de Nodo Counts

Nodo Postgres – Ensure Schema

```
-- 1) Asegurar esquema (idempotente)
CREATE TABLE IF NOT EXISTS public.cec_events (
  event_id       text PRIMARY KEY,
  ts_epoch       text NOT NULL,
  source         text NOT NULL,
  rule_id        text,
  severity       integer NOT NULL CHECK (severity BETWEEN 1 AND 5),
  asset_id       text NOT NULL,
  confidence     double precision NOT NULL CHECK (confidence >= 0 AND confidence <= 1),
  raw_event      jsonb NOT NULL,
  inserted_at    timestamptz NOT NULL DEFAULT now()
);

-- Índice único adicional por "bucket" de 1 minuto para evitar duplicados semánticos
-- cuando distintas fuentes generen IDs diferentes para el mismo hecho.
CREATE UNIQUE INDEX IF NOT EXISTS cec_events_uniq_semantic
ON public.cec_events (
  source,
  asset_id,
  COALESCE(rule_id,''),
  date_trunc('minute', ts_epoch)
);

```

Figura 8. Código JSON de nodo Postgres Ensure -Schema

Este script SQL crea y asegura la tabla cec_events en PostgreSQL para almacenar eventos normalizados bajo el Esquema Canónico de Eventos (CEC). La estructura incluye campos clave como identificador único (event_id), marca temporal (ts_epoch), fuente, regla, severidad (1–5), activo, nivel de confianza (0–1) y el evento crudo en formato JSONB, junto con un registro automático de inserción. Además, incorpora un índice único semántico basado en fuente, activo, regla y ventana de tiempo de un minuto, lo que permite evitar duplicados generados por distintas fuentes para un mismo hecho, garantizando idempotencia, coherencia y trazabilidad de los datos.

Nodo Postgres – Upsert CEC

```
INSERT INTO public.cec_events (
    event_id, ts_epoch, raw_event, source, rule_id, severity, asset_id, confidence
)
VALUES (
    COALESCE($1, md5($4 || '-' || $2)), -- event_id
    $2 ,                                -- ts_epochs
    $3::jsonb,                          -- raw_event
    $4,                                 -- source
    $5,                                 -- rule_id
    $6,                                 -- severity
    $7,                                 -- asset_id
    $8                                  -- confidence
)
ON CONFLICT (event_id) DO UPDATE SET
    ts_epoch = EXCLUDED.ts_epoch,
    raw_event = EXCLUDED.raw_event,
    source = EXCLUDED.source,
    rule_id = EXCLUDED.rule_id,
    severity = EXCLUDED.severity,
    asset_id = EXCLUDED.asset_id,
    confidence = EXCLUDED.confidence,
    inserted_at = now(); -- Corregido de 'updated_at' a 'inserted_at'
```

Figura 9. Código JSON Postgres – Upsert CEC

El código siguiente inserta registros en la tabla cec_events aplicando un esquema de upsert idempotente: si el event_id no se proporciona, se genera automáticamente con un hash (md5) a partir de la fuente y la marca temporal, garantizando unicidad. Los datos incluyen el evento crudo en formato JSONB, su origen, regla asociada, severidad, activo y nivel de confianza. En caso de conflicto por un event_id ya existente, la instrucción actualiza todos los campos del registro con los nuevos valores y refresca la marca de inserción con now(). De este modo, se asegura que cada evento se mantenga actualizado y consistente en la base de datos, evitando duplicados y preservando trazabilidad.

Métricas iniciales de validación

En esta fase temprana se establecieron indicadores base que servirán como línea de referencia:

Métrica	Descripción
Número de eventos procesados	Total de eventos ingeridos por ejecución

Tiempo de ingesta	Tiempo promedio desde trigger hasta persistencia
% de normalización correcta	Eventos conformes al CEC / total procesados
Errores de validación	Eventos descartados por formato incorrecto

Tabla 1. Métricas de Evaluación

Estos indicadores permitirán comparar el rendimiento del sistema en futuras etapas, cuando se integren los agentes de análisis y respuesta.

3. Secciones o capítulos del documento final desarrollados

    1. Introducción
    2. Estado del Arte

4. Revisión y firma del tutor del proyecto

    Yo, Roberto Andrade, profesor de la carrera de Ingeniería en Ciencias de la Computación, hago constar que he revisado y, por lo tanto, apruebo las actividades realizadas durante este período de trabajo. Por otra parte, considero que el avance del proyecto integrador es adecuado y se corresponde con el cronograma definido en el documento de planificación.

Firmado electrónicamente por:
ROBERTO OMAR ANDRADE PAREDES
Validar únicamente con FirmaEC

Fdo: Roberto Andrade

Quito, 3 de Octubre de 2025