Cybersecurity Threat-Hunting & Autonomous SecOps

(Draft)
Thiago Horiuchi Allue
thiago.allue@hotmail.com

GitHub repository: thiago-h-a/multi-agent-security-ai

A complete threat-hunting pipeline rarely fits in a single prompt. Like the Research-Assistant case, our SecOps study case decomposes the mission into **seven specialized agents** stitched together with LangGraph—each shouldering a single cognitive load while passing a shared `HuntState`. The result is traceable, testable, and open to surgical extension (loops, side branches, retries).

Cybersecurity Threat-Hunting & Autonomous SecOps **1**
1. Glossary 3
2. Why Seven Agents? 3
3. Collector Node 4
4. Intel Agent 4
5. Hypothesis Agent 5
6. Query-Builder Agent 5
7. Detector Node 5
8. Correlator Node 6
9. Responder Node 6
10. Global Error-Handling & Logging 6
11. Data-Lineage at a Glance 6
12. Extension Ideas 7
13. Takeaways 7

1. Glossary

Term	Description
HuntState	The shared state object (a dataclass) passed between nodes; holds `messages`, `evidence`, `alerts`, `story`, etc.
Command	The return type from each agent node that tells the graph what to do next (e.g., `goto="next_node"`), and what data to update.
Evidence	A dictionary inside `HuntState` used to store intermediate and derived data throughout the hunt (e.g., raw logs, enriched events).
Alerts	A list of anomalies or findings detected during the hunt (usually by the `detector_node`).
Story	A final summary message or incident narrative generated by the `responder_node`.
ESQL	Elasticsearch's SQL-like query language, used in querying logs.
CTI Feed	Cyber Threat Intelligence feed (external tool) that provides IP, hash, or domain reputation data.
SOAR	Security Orchestration, Automation, and Response platform. Agents may trigger SOAR actions (e.g., isolate host, raise ticket).
End Node	Special graph node that terminates the workflow.
Try/Except	Python error handling used around each node to ensure failures are logged but don't crash the graph.
Red Arrow (🟥)	In visual diagrams, represents an error path or graph termination.
Green Arrow (🟩)	In diagrams, represents normal forward progression in the hunt.

2. Why Seven Agents?

Hunting in real-world SOCs involves **collection, enrichment, reasoning, search, detection, context-joining, and human reporting**. Packing all of that into one LLM prompt invites incoherence and brittle prompts. Instead, splitting the work yields:

Challenge	Dedicated agent	Benefit
Raw-data diversity	**Collector Node**	Vendor-specific collectors live here, keeping the graph vendor-agnostic.
Intelligence context	**Intel Agent**	Central spot to swap CTI feeds or add local allow-lists.
Formulating “what could go wrong?”	**Hypothesis Agent**	Keeps creative reasoning isolated from query syntax details.
Translating ideas to data-store language	**Query-Builder Agent**	Single source of truth for Elasticsearch, Splunk, SQL, …
Separating signal from noise	**Detector Node**	Pure statistical / ML logic; easy to A/B test.
Multi-alert stitching	**Correlator Node**	Groups alerts so analysts see “an incident,” not a pager-storm.
Human story & orchestration	**Responder Node**	Decouples narrative tone and SOAR actions from detection code.

Modularity means each agent can be tuned, rewritten, or AB-tested without perturbing the others—an operational super-power.

---

3. Collector Node

- **Responsibility:** ingest raw telemetry (`messages → evidence["raw"]`).
- **Failures:** network timeouts, malformed logs.
- **Safeguards:** `try/except`, log and short-circuit to **End** so one broken source never blocks the hunt.

---

4. Intel Agent

- **What it adds:** CTI look-ups, hash/IP reputation → `evidence["enriched"]`.
- **Key design decisions**
    - **Rate-limit awareness**—batch queries.

    - **Caching**—hot IoCs return often; memoize for speed and cost.

- **Extensibility:** drop in YARA-match service, geolocation tags, or internal threat scores.

---

5. Hypothesis Agent

- **Creative engine.** Looks at enriched logs and poses candidate "stories" (e.g., *"credential-stuffing from TOR exit nodes"*).
- **Output:** list of hypothesis dicts (`evidence["hypotheses"]`).
- **Prompt pattern:** "Given {enriched_events}, enumerate plausible threats; return JSON[{query, rationale}]".
- **Why separate?** Enables prompt engineering without touching data connectors.

---

6. Query-Builder Agent

- **Mission:** turn human ideas into ESQL strings (`evidence["queries"]`).
- **Best practice:** keep a unit-test corpus—every hypothesis template ↔ expected query—to prevent silent prompt drift.
- **Potential loop:** If Elastic returns ≥ `N` warnings ("unknown field"), feed back to Hypothesis Agent for rewrite.

---

7. Detector Node

- **Decision point #1** (see *Decision & Branching Logic* diagram).
    - `len(alerts) == 0` → short-circuit **End**.
    - `> 0` → continue.
- **Swappable scoring**—today a one-liner; tomorrow a deep-learning outlier model, same signature.

- **Logging:** `logger.info` for path taken; `logger.error` on exception, continue safe path.

---

8. Correlator Node

- **Purpose:** dedupe and group related alerts → `evidence["incident"]`.
- **Heuristics:** same user, asset, TTP, or time-window.
- **Low-risk:** pure Python; no network calls → minimal failure surface.

---

9. Responder Node

- **Final mile:**
    - Build analyst-friendly narrative (`story["summary"]`).
    - Optional SOAR hooks (contain host, create ticket).
- **Guarantee:** *some* summary—even if previous steps bailed—so the analyst is never left wondering.

---

10. Global Error-Handling & Logging

- **Green solid arrows** = happy path.
- **Red dashed** = exception recovery.
- Every agent is a **circuit-breaker**: catch → log → safe `Command(goto="end")`.
- Centralised `logging.basicConfig` in `graph.build`: timestamps + levels for unified Kibana dashboards.

---

11. Data-Lineage at a Glance

Timeline diagram shows exact key populated at each hop.
`messages → raw → enriched → hypotheses → queries → alerts → incident → story`
Analysts (or auditors) can replay any run by persisting state snapshots; great for incident post-mortems.

---

12. Extension Ideas

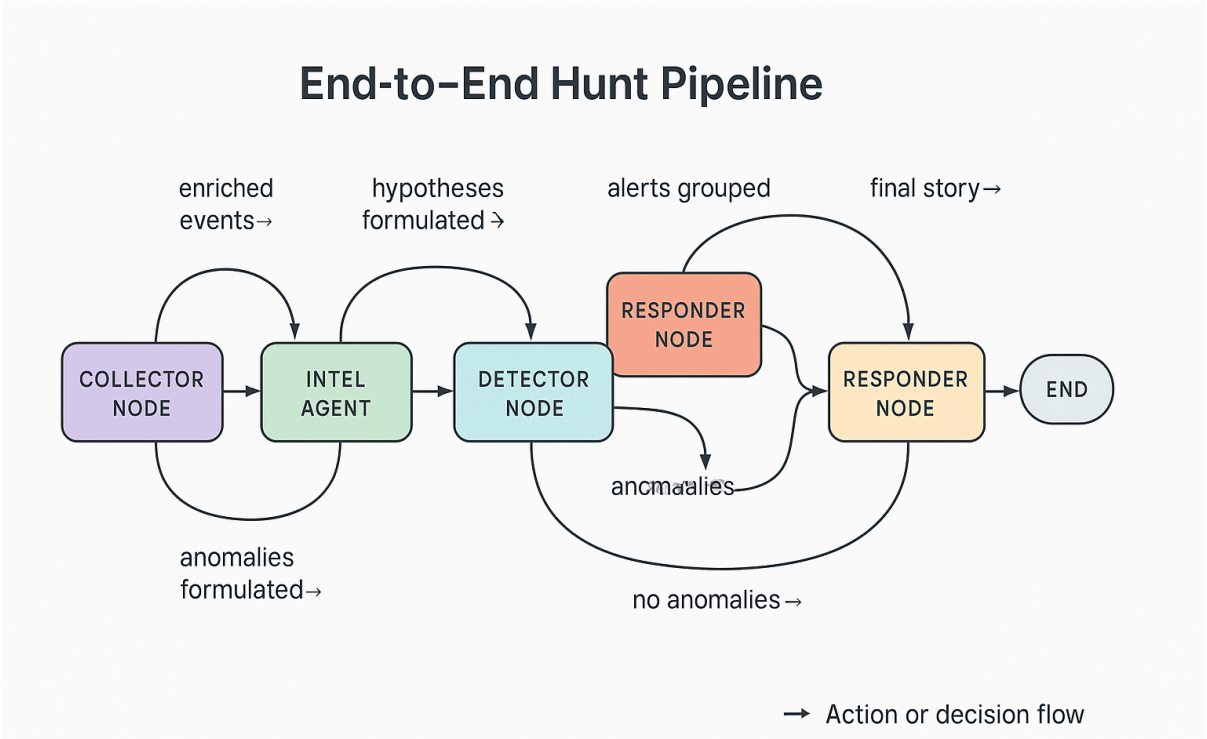
1. **Auto-Tuning Loop** – If Responder sees “false positive” feedback, route back to Detector with new thresholds.
2. **Fact-Check Subgraph** – Verify intel assertions against OSINT before alerting.
3. **Parallel Branches** – Run Windows-specific and Linux-specific hypotheses concurrently, merge at Correlator.
4. **Confidence Scoring** – Attach a probability to each incident; only SOAR when ≥ 0.8.

---

13. Takeaways

- **Separation of concerns** keeps prompts short and intentions clear.
- **Error containment** ensures partial outages don’t torpedo the hunt.
- **Typed state + unit tests** make the graph CI-friendly.
- **Observable logging** turns the pipeline into an explainable, auditable asset—crucial for SOC trust.

In short, the seven-agent Hunt-Graph offers a **robust template** for industrial-grade SecOps automation—easy to evolve, easy to debug, and ready for advanced loops when your maturity demands it.



[LLM Generated Image]

How to read the diagram 💡

Step	From → To	Arrow label	What happens	Resulting HuntState update
1	Collector Node → Intel Agent	**raw events →**	Raw telemetry is forwarded for enrichment.	`evidence["raw"]` populated
2	Intel Agent → Hypothesis Agent	**enriched events →**	Threat-intel context is added (IPs, hashes, etc.).	`evidence["enriched"]`
3	Hypothesis Agent → Query-Builder Agent	**hypotheses formulated →**	High-level hunting ideas (e.g., “failed logins from TOR”) are produced.	`evidence["hypotheses"]`
4	Query-Builder Agent → Detector Node	**ESQL queries →**	Concrete data-store queries are generated.	`evidence["queries"]`
5a	Detector Node → Correlator Node	**anomalies detected →**	If anomalies ≥ 1, they’re promoted to alerts.	`alerts` list filled

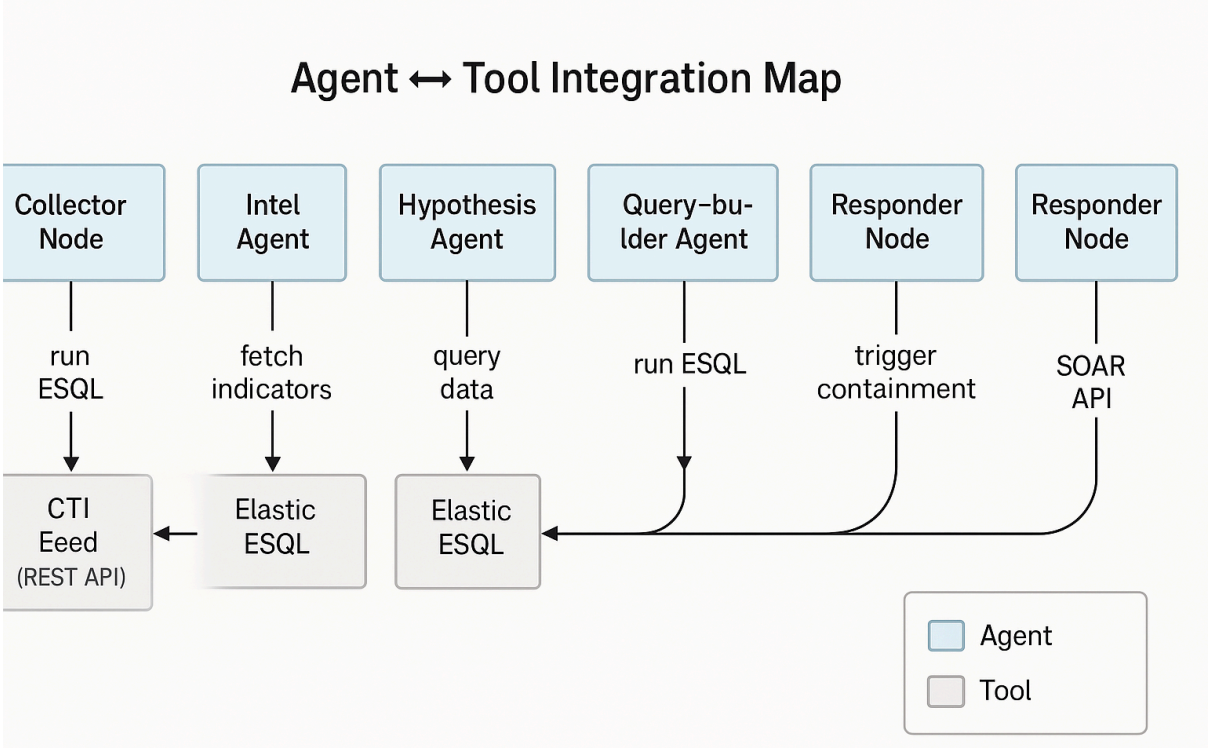
5b	Detector Node → End	**no anomalies** →	Zero anomalies short-circuit the hunt.	Graph terminates
6	Correlator Node → Responder Node	**alerts grouped** →	Alerts are bundled into an incident object.	`evidence["incident"]`
7	Responder Node → End	**final story** →	Human-readable summary is attached.	`story["summary"]`

The **curved split arrows** out of the **Detector Node** make the decision logic explicit:

- *Anomalies present* → continue the pipeline.
- *No anomalies* → clean exit.

All other arrows are simple “data-in → data-out” flows.

Together the chart and table show *who* talks to *whom*, *what* each hop represents, and *how* shared state mutates end-to-end.



[LLM Generated Image]

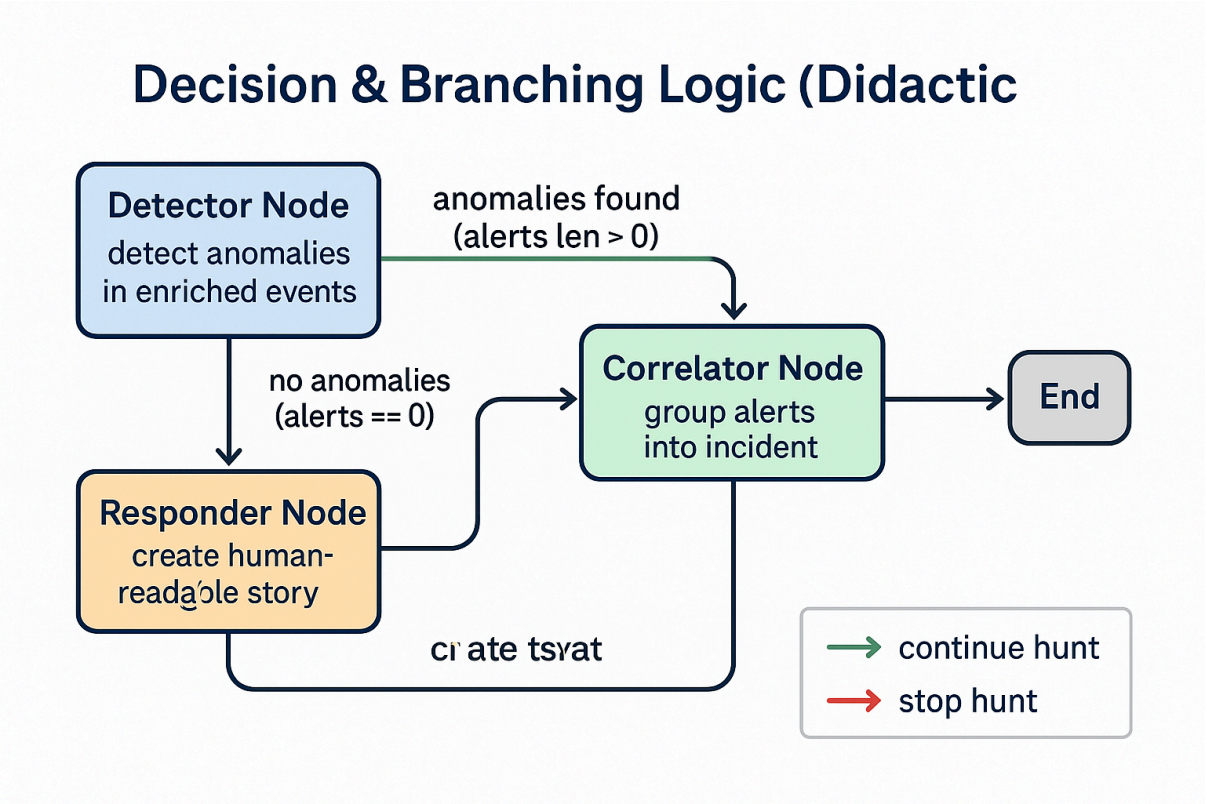
Agent ↔ Tool Integration Map (explanation)

Agent	External tool(s) called	Arrow label in diagram	Purpose
Collector Node	Elastic ESQL	**run ESQL**	Pulls raw log events directly from Elastic to seed the hunt.
Intel Agent	CTI Feed (REST)	**fetch indicators**	Enriches raw events with IP / hash reputation data.
Hypothesis Agent	Elastic ESQL	**query data**	Performs quick look-ups to validate a hypothesis before formal queries are built.
Query-Builder Agent	Elastic ESQL	**run ESQL**	Translates hypotheses into fully-fledged ESQL queries.
Detector Node	SOAR API	**trigger containment**	If anomalies are severe, calls SOAR playbook (e.g. isolate host) while continuing the hunt.
Correlator Node	*No external tool*	—	Purely groups alerts; internal logic only.
Responder Node	SOAR API	**trigger containment** (second call)	Final response actions (ticketing, email, orchestration).

Legend

- Pastel boxes = Agents (code you wrote)
- Grey boxes = External services / tools
- Solid arrows = Data or command flow, labeled with the action
- Tools may receive calls from multiple agents; the CTI feed is sourced only from Intel Agent, while Elastic ESQL is used by three agents.

Together, the diagram and table clarify exactly **which agent hits which external system and why**, helping you trace dependencies or mock integrations in tests.

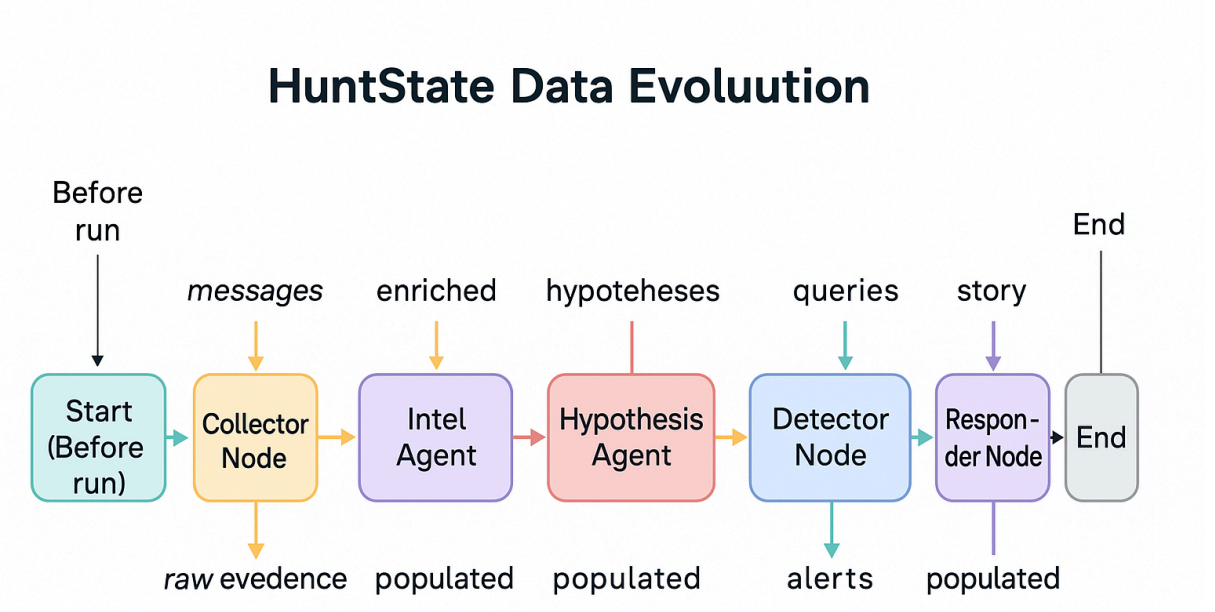


[LLM Generated Image]

Quick read of the branching diagram

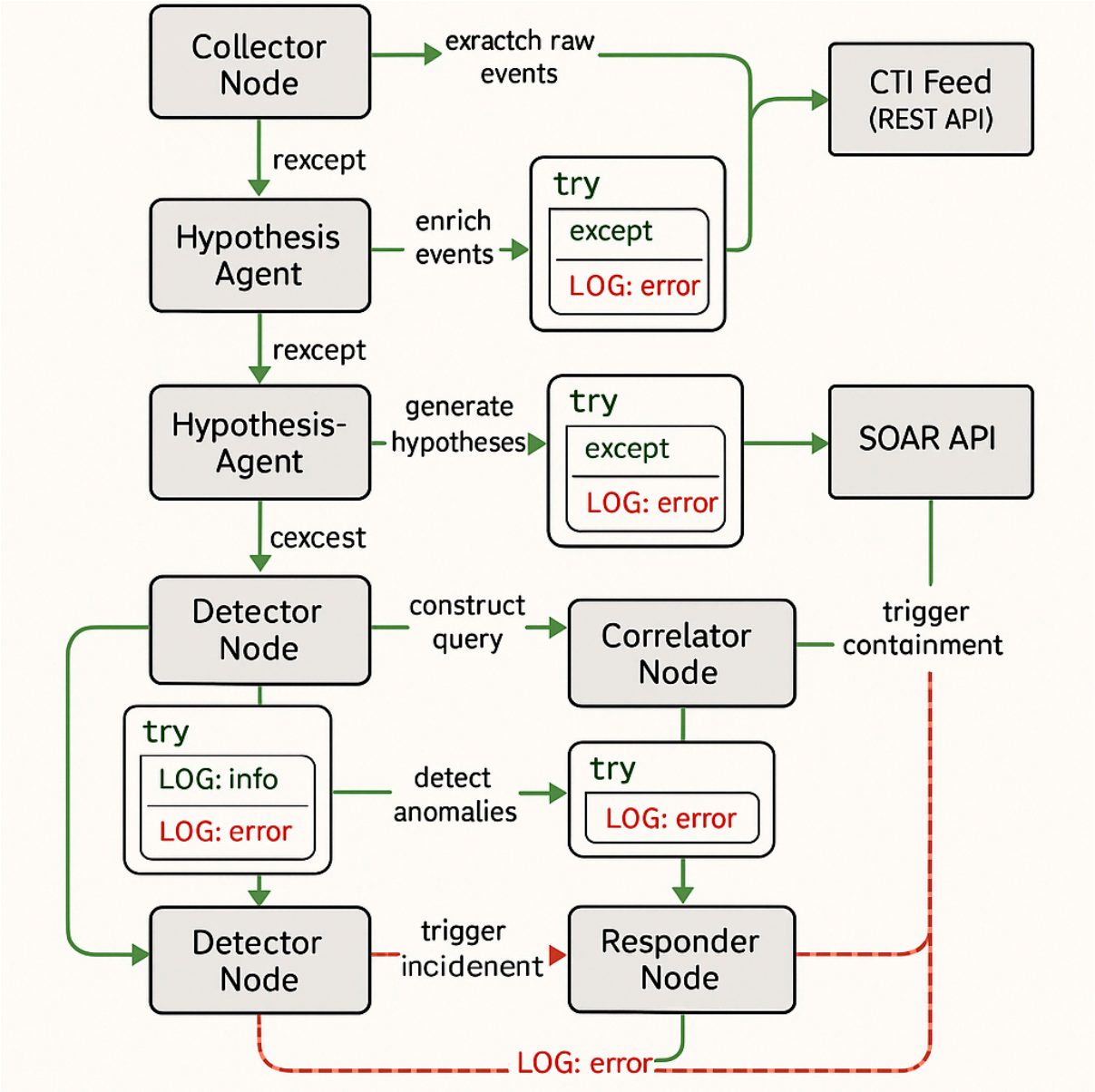
- **Detector Node**
  - `alerts == 0` → straight to **Responder Node** → **End** (hunt stops early).
  - `alerts > 0` → **Correlator Node**.
- **Correlator Node**
  - Bundles alerts into an **incident** → **Responder Node**.
- **Responder Node**
  - Writes human-readable summary (empty or populated incident) → **End**.

Legend: **green arrow = continue**, **red arrow = terminate**.



[LLM Generated Image]

Stage	New / updated HuntState key(s)	What the agent adds
Before run	—	`messages` list only (user input).
Collector Node	`evidence["raw"]`	Copies raw event blobs from `messages`.
Intel Agent	`evidence["enriched"]`	Adds CTI metadata to each event.
Hypothesis Agent	`evidence["hypotheses"]`	High-level hunt ideas.
Query-Builder Agent	`evidence["queries"]`	Concrete ESQL strings for Elastic.
Detector Node	`alerts` list	Anomalies pulled from query results.
Correlator Node	`evidence["incident"]`	Bundled incident object.
Responder Node	`story` dict	Human-readable summary text.
End	— (no further changes)	Graph terminates.



[LLM Generated Image]

Legend—read the red vs. green paths

Arrow color	Meaning	Typical `logger` call
Green solid	Normal data flow (`try` succeeds)	`logger.info(...)`
Red dashed	Exception path → next safe node / End	`logger.error(...)` in `except`

Every agent/tool pair is wrapped in `try/except`; on error, a red dashed arrow shows where execution resumes, keeping the pipeline alive while the failure is logged.