

AWS CLOUD SOC — DETECTION, INVESTIGATION & AUTOMATED RESPONSE

Complete educational lab guide · Public-safe technical edition · October 18, 2026

Type: Personal AWS cloud-security engineering project · **Status:** Core MVP complete; deployed runtime and billing must be checked separately before making current-state claims.

Evidence boundary. Screenshots of the Next.js analyst cockpit are **synthetic offline fixtures**, not live incidents. The October 13, 2026 **84-second automatic rollback** is a separate, historically documented personal-lab AWS acceptance test. Do not mistake mocked **LIVE MUTATION** labels or earlier administrator restores for that result.

Contents

1. Purpose and learning outcomes
 2. What was actually built
 3. Threat model and trust boundaries
 4. Architecture and event paths
 5. AWS lab foundation and least privilege
 6. The six CloudTrail detection heuristics
 7. GuardDuty → EventBridge → Lambda → DynamoDB
 8. Investigation, triage, and analyst console
 9. Human approval, SigV4, and execution permissions
 10. Containment and recovery-first scheduling
 11. Historical acceptance and evidence
 12. Auditing, failure recovery, and limitations
 13. Local reproducibility and safe tests
 14. Troubleshooting matrix
 15. Operational checklist and cost considerations
 16. Interview explanation, glossary, and next steps
-

1. Purpose and learning outcomes

A security-operations center should explain not only **what activity looks suspicious**, but also **how evidence is evaluated, who authorizes response, how dangerous actions are bounded, and whether recovery is proven**. This personal project connects those questions using AWS-native components, Python, Terraform, and a local Next.js read-only analyst cockpit.

It does **not** claim paid SOC employment, a production-grade managed detection-and-response platform, or a real attack against a customer workload. Controlled tests used a disposable, unattached lab network interface.

At the end of this guide, a learner should be able to diagram two detection sources, interpret a case lifecycle, explain least-privilege approval, describe pre-armed rollback, validate synthetic demo behavior, and distinguish a successful automated restoration from emergency manual recovery.

2. What was actually built

Capability	Implementation	Evidence class
CloudTrail API-event hunter	Six deterministic Python management-event heuristics	Source-code verified; can run offline
GuardDuty finding workflow	EventBridge rule, Lambda enrichment, DynamoDB case persistence, SNS notification	Architecture/source described
Case investigation	Severity, matched rule, sanitized evidence, analyst rationale and timeline	Verified local UI in synthetic mode
Response approval	IAM/SigV4 authenticated reviewer, nonce/binding and separated API	Source and historical lab evidence
Containment	SQS dispatch, scoped Lambda EC2 security-group mutation on disposable canary	Historical personal AWS lab
Scheduled recovery	Original baseline saved and EventBridge Scheduler armed before mutation; Lambda restores original group	Historical independent acceptance
Analyst cockpit	Eight-view Next.js 16 dashboard, localhost-only, mock and authenticated read-only adapters	Work-mode browser audit
Quality	Automated Python security tests, mock browser checks and Terraform validation	Historical test/CI records

The project began with read-only detection and gradually added guarded response. The final core outcome was **automatic restoration of the canary's original security group 84 seconds after quarantine verification**, with a **60-second configured one-time schedule**. EventBridge's scheduling and worker execution are not hard real-time guarantees.

3. Threat model and trust boundaries

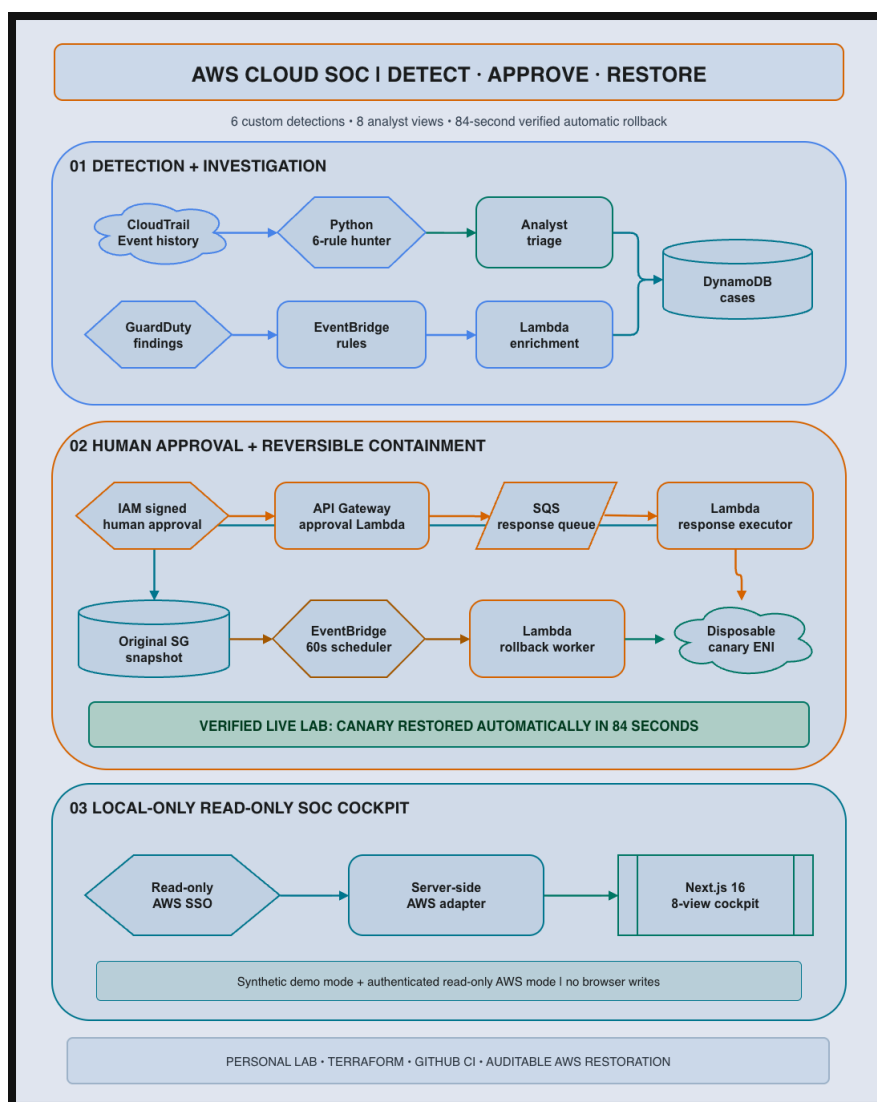
Assets to protect: AWS account permissions, raw event records, response approval tokens, original security-group state, the recovery schedule, audit integrity, and a user's browser session. The highest-risk operation is a live EC2 change; therefore normal use remains **read-only**, and mutation requires an explicit operator-owned workflow outside the browser.

Threats and failures considered: overprivileged API identities; forged or replayed approvals; the wrong account or target resource; logging of SQS receipt handles or nonces; a failed or delayed Scheduler invocation; manual out-of-band configuration changes; confusion between synthetic and live telemetry; leaking account/resource identifiers through screenshots.

Controls: narrow IAM roles, expected-account checks, isolated canary scope, nonce-bound reviewer authorization, zero browser mutation endpoints, durable baseline snapshots, pre-armed one-time rollback, independent recovery enablement, EC2 read-back, and audited manual reconciliation. Append-only audit is **logical and policy-governed**, not cryptographic or physically tamper-proof.

Security is layered. A high-severity case is *evidence for review*, not permission to modify infrastructure. A signed approval authorizes only the defined target and action, while the response executor verifies prerequisites again.

4. Architecture and event paths



Path A — CloudTrail history hunt: a read-only AWS session requests bounded CloudTrail `LookupEvents` management history. `tools/cloudtrail_history_hunter.py` parses event details *in memory*, emits metadata-only findings, and avoids persisting raw event bodies. A human reviews and classifies the resulting findings.

Path B — GuardDuty findings: GuardDuty → EventBridge → Lambda enrichment → DynamoDB case record. SNS can notify the analyst queue. This is separate from the CloudTrail hunter; describing the entire pipeline as a single six-rule real-time detector would be inaccurate.

Response path: escalated case → separate SigV4 reviewer approval → controlled API/state transition → SQS → response executor Lambda → durable baseline snapshot and one-time rollback schedule → EC2 canary SG change → automatic recovery Lambda → read-back and audit.

Visibility path: DynamoDB and bounded AWS read operations → server-side Next.js adapter → a local eight-view dashboard. The **mock adapter** supplies synthetic cases without cloud API calls; the **AWS adapter** is separate and read-only. Browser-originated live approvals and containment are intentionally unavailable.

5. Lab foundation, permissions, and safe setup

Use a **dedicated, personally controlled AWS sandbox**, not a production/customer account. Establish an AWS region, cost alerts and an administrator-approved teardown plan before creating billable resources. Prefer AWS IAM Identity Center sessions and role separation to long-lived access keys.

A practical progression is:

1. Clone or inspect the repository and review `README.md`, `infra/`, `tools/`, `scripts/`, and `apps/soc-console/`.
2. Run unit tests and offline fixtures **without AWS credentials**.
3. Configure the local SOC console in `DATA_SOURCE=mock` mode and confirm the synthetic banner.
4. Inspect Terraform files, IAM scopes and rollback behavior in a plan-only review. A Terraform plan can disclose account-specific metadata: keep plans private.
5. Only if authorized to run your **own** bounded lab: independently approve an unattached disposable canary target, verify rollback arming and recovery paths, and keep explicit manual recovery procedures available.
6. After any live test, verify AWS state read-back and return forward execution to safe mode while preserving recovery capability.

Do not copy IDs, ARNs, keys, nonces, original request files or private CloudWatch messages from another account. In particular, **public documentation should never instruct someone to paste real approval nonces or run unrestricted EC2 mutations**.

6. Six Python CloudTrail detection heuristics

Source implementation: `tools/cloudtrail_history_hunter.py`. The hunter operates on management event **history**. It is not an exhaustive behavior detector, SIEM, network packet sensor or proof of compromise. Its exact rules are:

Rule	Severity	Trigger logic
<code>ROOT_PRINCIPAL_ACTIVITY</code>	High	CloudTrail <code>userIdentity.type == Root</code>
<code>CONSOLE_LOGIN_FAILURE</code>	Medium	<code>ConsoleLogin</code> with <code>responseElements.ConsoleLogin == Failure</code>
<code>ACCESS_DENIED_ATTEMPT</code>	Low	Error code includes <code>AccessDenied</code> , <code>Unauthorized</code> or <code>OperationNotPermitted</code>
<code>CLOUDTRAIL_CONFIGURATION_CHANGE</code>	High	Successful specified CloudTrail configuration mutation API

Rule	Severity	Trigger logic
IAM_PRIVILEGE_MODIFICATION	High	Successful specified IAM policy or trust-policy modification API
S3_SECURITY_POLICY_CHANGE	Medium	Successful bucket policy, ACL or public-access-block change API

The real code defines specific event-name allowlists and checks the AWS service source before firing the latter three rules. A single event can trigger more than one heuristic. Failure/denied signals alone are *not* proof of malicious intent.

Run the hunter safely against a sanitized fixture:

```
cd aws-cloud-soc-automated-response
python3 tools/cloudtrail_history_hunter.py --fixture path/to/sanitized-events.json
```

The fixture must have an `Events` array shaped like CloudTrail lookup results. Live read-only querying additionally requires an expected account ID and AWS profile, but is not necessary for this educational exercise. Never include the real 12-digit ID in screenshots or public transcripts.

Illustrative, not executable finding record:

```
{"rule":"IAM_PRIVILEGE_MODIFICATION","severity":"HIGH", "event":"AttachRolePolicy",
"service":"iam.amazonaws.com", "time":"<redacted UTC>"}
```

Validate by testing rule matching, negative cases, severity classification and suppression of raw `CloudTrailEvent` /identity detail in output.

7. GuardDuty, EventBridge, Lambda, DynamoDB, SNS

GuardDuty produces managed threat-detection findings. **EventBridge** filters/routs matching events. A **Lambda enrichment handler** validates and normalizes selected metadata, then writes a durable **DynamoDB case**. **SNS** can deliver an analyst notification. Each component has a distinct role; separating them makes it easier to audit data minimization and error handling.

In a synthetic exercise, feed a sanitized finding through the offline transformation tests and verify that the resulting case has a stable identifier, event source, severity, matched rule/type, timestamps and analyst-readable summary. Sensitive identifiers and credentials must be scrubbed before showing evidence in a browser.

Operationally, account for duplicate events, eventual consistency, expired cases (TTL), failed Lambda invocations and downstream notification errors. A notification success is not equivalent to a completed investigation or a containment approval.

8. Case investigation and the eight-view cockpit

The local Next.js 16 console contains: **Overview, Investigations, the full Case Dossier, Approval Gate, Response Operations, Forensic Audit, Architecture, and Settings & Posture**. The sidebar exposes seven top-level destinations; the case dossier is reachable through Investigations as the eighth view.

Mock demo:

```
cd aws-cloud-soc-automated-response/apps/soc-console
npm ci
DATA_SOURCE=mock npm run dev -- --hostname 127.0.0.1
```

Open `http://127.0.0.1:3000` and verify `MOCK MODE ACTIVE`, `Synthetic Snapshot (Offline)`, and `READ-ONLY ENFORCED` before recording or presenting. The browser-audited navigation is:

`Overview → Full Cases Explorer → Search Keyword: credential → Apply → Inspect → Full Case Dossier → View in Approval Gate → Response Operations → Forensic Audit`.

The case dossier includes a redacted evidence summary, analyst triage assessment and four-stage lifecycle. **The displayed cases and some history rows are synthetic, even if a mock row bears the word `LIVE` or `VERIFIED`.** They are not the historical October 13 automatic-recovery result.

Known issue as of browser audit: Forensic Audit phase filters change the URL but still show all entries. Do not demonstrate these filters as working. Some Overview/Architecture fixture text still refers to a ten-minute/600-second schedule, and mock windows use a fixed capture reference. Those are stale fixture labels, not a reason to alter the historical acceptance record.

9. IAM reviewer, SigV4 and least privilege

The approval subsystem intentionally separates **investigating** from **authorizing** and **executing**. A triaged/escalated finding may lead to a specific proposal. A dedicated reviewer identity authenticates a signed API request, including a fresh challenge/nonce and request-bound parameters. The backend checks the request and its allowed target; the execution worker separately enforces scoped containment.

Why not a browser Approve button? A local read-only browser must not hold privileged AWS tokens or expose a mutation endpoint. The dashboard's Approval Gate page documents these boundaries rather than executing authorization itself.

In threat modeling, validate: wrong reviewer role, stale nonce, changed parameters, expired approval, duplicate requests, unsupported target, inadequate IAM and attempted browser mutation. Denied execution should remain logged as a failure rather than be silently retried as success.

10. Recovery-first containment sequence

Containment changes are potentially disruptive. The lab therefore implemented a recovery-first sequence:

1. Reviewer authenticates and approves the specific lab-only action; request enters a queue.
2. The response worker confirms account, VPC, target ENI, tags, status and allowed original/quarantine security groups.
3. The worker **reads and durably stores the original security-group configuration**.
4. **Before modifying the ENI**, it arms a one-time `EventBridge Scheduler` rollback and records its identity/deadline.
5. Only then does the worker apply the temporary quarantine security group and verify actual EC2 read-back.

6. At the configured time (with ordinary scheduler/invoke latency), the independent recovery Lambda reads the saved baseline, restores the original group and verifies the EC2 read-back.
7. The durable request and audit trail distinguish success, failure, and emergency administrator restoration.

The configured timer is a **policy**, not a hard real-time SLA. A 60-second configured schedule produced an **84-second measured** interval in the historical acceptance test.

Do not adapt the lab containment mutation commands to real infrastructure without an independent authorization, rollback and recovery plan. This guide deliberately excludes live mutating commands.

11. Historical October 13, 2026 acceptance evidence

The primary source is `docs/phase05/02_AUTOMATED_ROLLBACK_LIVE_ACCEPTANCE.md` in the private engineering repository. Independent historical AWS checks corroborated the request record, Scheduler invocation, recovery Lambda logs, actual EC2 group change and restoration, and durable audit evidence.

Event	Historical UTC time
Baseline snapshot and schedule armed	18:08:44
Quarantine SG verified by EC2 read-back	18:08:45
Configured one-time schedule deadline	18:09:44
Scheduler-triggered recovery Lambda invocation	18:10:05
Original SG restored and verified	18:10:09

Measured containment-to-restoration: 84 seconds. The request was classified `ROLLED_BACK`, with trigger `AUTOMATIC_SCHEDULED_ROLLBACK`, method `WORKER_EC2_MUTATION`, and verified restoration status. No administrator performed the restoration between containment and recovery.

Earlier requests had **emergency administrator restoration** and later reconciliation. Those events are not automated rollback successes. Do not combine the historical records or replace their failure labels with the outcome of the successful test.

12. Audit integrity, failure recovery and limitations

DynamoDB stores response/approval history under authorization and conditional-write policies. The system is designed as a **logical append-only audit**: it is not an immutable cryptographic ledger. An out-of-band administrator who directly changes EC2 state can create a physical-versus-audit discrepancy until reconciliation is performed.

Failure examples and responses: missed schedule → inspect recovery Lambda, scheduler record, DDB request and EC2 read-back; permissions-denied containment → stop and investigate IAM scope; incorrect original snapshot → fail closed before mutation; unauthorized manual restore → verify EC2 state and write a clearly labeled reconciliation event. Manual recovery must remain manual in the audit.

Current limitations: the cockpit is localhost only; mock datasets are fixed demonstrations; audit phase filters have a UI bug; older mock values are not updated to the final one-minute test; some claimed encryption/network controls need IaC verification; source-only CloudWatch SQS-log sanitization was committed but its AWS deployment was not independently confirmed by the browser audit. A final security review must not overstate these boundaries.

13. Testing and reproducibility

Historical final QA documented 244 core Python regression checks, 60 Phase 3B checks, 39 SOC console unit tests, eight mock Playwright browser checks and static/security lint results. These counts describe the reviewed release, not an assertion that every test was re-run at guide-publication time.

Safe commands:

```
cd aws-cloud-soc-automated-response
./scripts/run_checks.sh
bash scripts/phase03b_static_checks.sh
python3 -m tools.privacy_lint
terraform -chdir=infra/phase03/response validate
```

For the cockpit:

```
cd apps/soc-console
npm ci
npm run lint
npm run typecheck
npm test
npm run test:e2e
npm run build
```

Requirements: use a compatible Node/Python runtime and the repo's declared package versions. Check the current `package.json` scripts before running. UI Playwright should be confined to mock mode to avoid AWS changes and costs. Terraform `validate` is static; it does not establish deployed IAM correctness or AWS bill state.

14. Troubleshooting — symptom → evidence → hypothesis → fix → verify

Symptom	Evidence to collect	Next diagnostic step
Dashboard not opening	Terminal exit code, port binding	Confirm Node/npm version, local <code>127.0.0.1:3000</code> , existing process
Unexpected AWS calls in demo	Source-mode banner, env config	Restart with <code>DATA_SOURCE=mock</code> , verify zero outbound reads
Case search seems wrong	Keyword, Apply click, URL state	Reset filters and use verified <code>credential</code> route
	URL parameter + unchanged six entries	Document the known UI bug; do not imply functionality

Symptom	Evidence to collect	Next diagnostic step
Audit phase filter still shows all		
Response remains quarantined	Real EC2 read-back, request status, Scheduler/Lambda logs	Follow independently authorized recovery runbook; preserve failure evidence
<code>botocore</code> missing	Python interpreter path	Activate the correct virtual environment; run <code>python -c "import botocore"</code>
CloudWatch exposes raw SQS payload	Deployed Lambda code hash versus source and logs	Review sanitized source, regression tests and separately authorized deployment
Time window inconsistent	Mock fixture capture timestamp	Explain fixed demo reference rather than implying live chronology

Never change IAM, approve containment, or apply Terraform in response to a cosmetic demo issue. Diagnose in a read-only environment first.

15. Operational closeout, privacy and cost review

Before any public publication: check screenshots for IDs, ARNs, roles, account numbers, IPs, request IDs, nonce values, raw JSON, credential material, and other private details. Cropping alone can leave small identifiers visible. Replace them with opaque masks and inspect the flattened output.

Before any independent new lab test: confirm explicit authorized target, available rollback and manual recovery, correct account/region, planned costs and clean initial state. The final production video deliberately used a historical result so no second mutation test was needed.

After a live test: verify the original group restored; zero orphaned schedules; no unresolved requests; response Lambda execution disabled; recovery remains enabled; Terraform and DDB state consistent. The October 13 acceptance report documented this for that date, not a continuous guarantee afterward.

Costs should be read from current account billing/Cost Explorer when claimed. Do not state that AWS resources cost zero based solely on a mock screen or estimated free-tier fit. CloudTrail, GuardDuty, logs, DynamoDB, Scheduler, S3 and data transfer may incur charges.

16. Interview story, glossary and next actions

30-second explanation: “I built a personal AWS Cloud SOC lab that combines six deterministic CloudTrail heuristics with a separate GuardDuty/EventBridge findings path, Lambda/DynamoDB case management, and an eight-view read-only analyst cockpit. A separate IAM-signed approval flow controls SQS/Lambda containment on an unattached disposable ENI. I configured recovery before changing the interface and verified a historical automatic restoration in 84 seconds, supported by AWS read-back and audit records.”

Describe challenges honestly: rollback delay beyond the configured minute; initial failed/manual tests; source-versus-deployed logging differences; preserving approval separation; preventing UI from implying mock records are live. These constraints show sound engineering judgment.

Glossary: **SOC** — security operations center; **ENI** — elastic network interface; **SG** — security group; **IAM** — identity and access management; **SigV4** — AWS Signature Version 4; **TTL** — time-to-live; **Lambda** — serverless compute; **SQS** — message queue; **DynamoDB** — managed key-value/document store; **EventBridge** — event routing; **Scheduler** — timed invocation service; **Read-back** — independently query actual resource state; **Reconciliation** — align physical changes and audit/request state.

Next steps: review the private implementation and acceptance documents; demonstrate the safe mock cockpit; add sanitized architecture/screenshots to the portfolio; follow through on source/deployed-code audit; evaluate optional SIEM or streaming additions only after core correctness is maintained.

Source provenance: private GitHub repository [HaythamAbouelfaid/aws-cloud-soc-automated-response](#) on its completed Phase V `main` release, final historical acceptance dated Oct 13 2026, and the eight-view browser audit dated Oct 17–18 2026. This guide is explanatory, not a deployment warrant or a disclosure of customer infrastructure.