eBook

# Building a Security Lakehouse

A modular reference architecture for modernizing detection, response, and security operations using Databricks

databricks

# Contents

# Executive Summary

This blueprint outlines how Databricks can serve as the analytical backbone for modern security operations centers (SOCs), starting with SIEM augmentation and evolving toward a lakehouse-centric detection platform. It enables teams to offload expensive queries, enrich and analyze telemetry at scale, and support real-time detection and investigation workflows without replacing their existing stack.

# 1 Motivation

Security operations teams are facing growing pressure to process more data, detect advanced threats and respond faster, all while managing cost and operational complexity. Traditional SIEM platforms often fall short due to high ingestion costs, rigid schemas and limited scalability. These constraints make it difficult to apply modern analytics, retain historical visibility or integrate machine learning (ML) into security workflows.

The Databricks security lakehouse provides a flexible, scalable foundation for modernizing security operations. It enables teams to ingest, normalize, enrich and analyze telemetry from identity systems, endpoints, cloud services and network infrastructure using an open and governed architecture. Rather than replacing existing tools, Databricks integrates with SIEM, SOAR and case management systems to improve detection, investigation and automation outcomes.

This blueprint presents a modular reference architecture for building a lakehouse-centric security operations platform. Many organizations begin by offloading high-volume telemetry or running pilot detections in Databricks, then expand their footprint over time as the platform demonstrates value across scale, speed and flexibility.

2

# Core Use Cases

| USE CASE | DESCRIPTION | EXAMPLE OUTCOME |
|---|---|---|
| **Detection Engineering** | Build SQL- and Python-based detection logic using normalized telemetry | Correlate access anomalies across Okta, VPN, AWS |
| **Log Cost Optimization** | Store raw logs in Databricks and forward only enriched alerts to the SIEM | Reduce Splunk ingest volume by 80% |
| **Historical Threat Hunting** | Query months of data across sources using Delta Lake | Detect lateral movement over a 12-month window |
| **Contextual Enrichment** | Join logs with asset, identity and threat intel data | Flag anomalous process execution on high-value endpoints |
| **Compliance and Reporting** | Create structured, repeatable reports for auditors, regulators and leadership | Automate generation of PCI, ISO or executive risk dashboards |
| **AI-Augmented Triage** | Use LLMs and AI agents to analyze, summarize and prioritize alerts | LLM adds context and narrative to login anomalies |

3

# Reference Architecture Overview

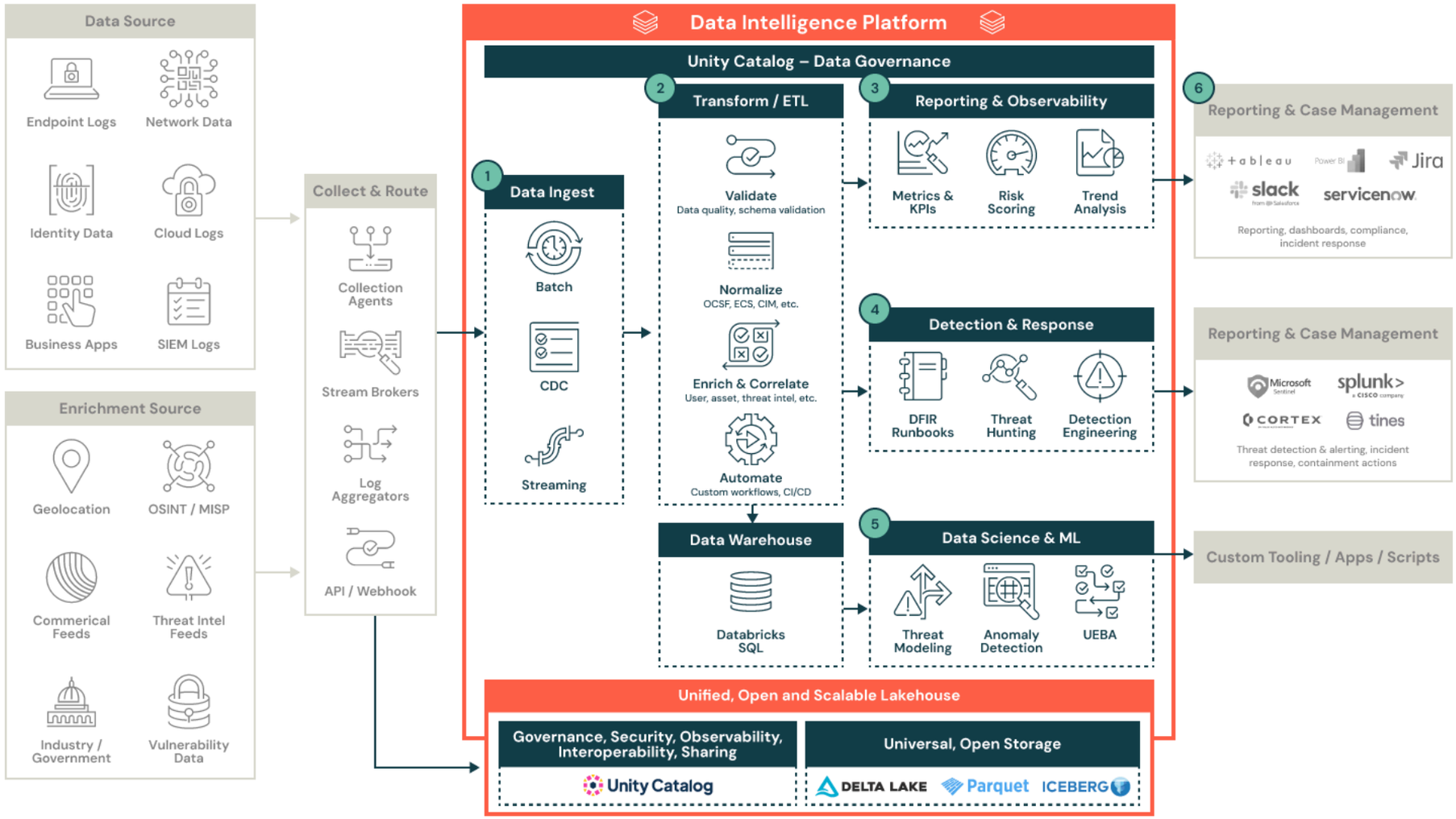



Figure 1. Databricks security reference architecture

1 **Collect and route:** Telemetry and threat intelligence are collected using agents, brokers, aggregators and APIs. This step supports structured and unstructured formats and enables both real-time and batch ingestion. Data sources include endpoint logs, identity events, cloud telemetry, vulnerability data and commercial threat feeds.

2. **Transform and enrich:** Raw data is validated for quality and schema alignment, then normalized and enriched for operational use. Enrichment can include user, device, application, threat intel and asset context. Schema models such as OCSF and ECS help standardize fields for analysis and automation.

3. **Reporting and observability:** Security analysts and stakeholders use structured data to build dashboards, visualize trends and track key metrics. These outputs help teams monitor coverage, identify gaps and meet compliance and audit requirements.

4. **Detection and response:** Detection engineers create and manage rules for identifying threats and suspicious behavior. Alerts feed into incident response workflows, DFIR runbooks and threat hunting investigations. Teams can orchestrate playbooks or push alerts to case management systems.

5. **Data science and ML:** Security teams can apply machine learning models for advanced analytics and detection. Use cases include threat modeling, behavioral baselining, anomaly detection and user entity behavior analytics. These models improve prioritization and reduce false positives.

6. **Integrate with external systems:** Curated alerts, dashboards and context are delivered to downstream platforms such as SIEM, SOAR and reporting and ticketing systems. This allows Databricks to serve as the analytical backbone while preserving familiar analyst workflows in tools like Splunk, Sentinel, Jira and ServiceNow.

4

# Step-by-Step Implementation Guide

This section outlines a modular, repeatable process for building a security lakehouse with Databricks. Each step aligns to a core function within the modern security operations lifecycle. The approach supports phased adoption and can be implemented alongside existing SIEM and SOAR tooling.

| STEP | TITLE | OUTCOME |
|---|---|---|
| 0 | Define Goals | Clear alignment on use cases, metrics and success criteria across stakeholders |
| 1 | Inventory and Prioritize Data | A structured log source inventory based on detection value, volume and coverage gaps |
| 2 | Ingest and Normalize | Raw telemetry is ingested, parsed and transformed into normalized Delta tables using the medallion architecture |
| 3 | Author and Operationalize Rules | Detection rules are written in SQL or Python, version-controlled and deployed to generate alerts |
| 4 | Route Alerts and Integrate With SOC Tools | High-quality alerts and enriched context are delivered to SIEM, SOAR or case management systems |
| 5 | Support Triage and Investigation | Analysts pivot across identity, cloud and process telemetry using dashboards and notebooks |
| 6 | Deliver Security Outcomes | Metrics, dashboards and compliance reports are shared with leadership and auditors; risk scores are generated for prioritization |

## Prerequisite: Define Goals

**Objective:** Establish clear business and operational goals that guide the technical design of your modern security analytics platform. This will ensure alignment across teams and provide a foundation for effective implementation.

**Outcome:** A well-defined set of objectives tied to business value and organizational priorities that will inform architecture, telemetry strategy and detection development.

| | |
|---|---|
| **Key Principles** | ▪ A strong implementation starts with clearly defined problems<br>▪ Goals should be tied to measurable improvements: detection coverage, cost reduction, analyst efficiency<br>▪ Cross-functional input (security, data engineering, operations) is essential |
| **Implementation Guidance** | ▪ Document the "as is" SOC architecture, including log sources, tools and workflows<br>▪ Identify core pain points: ingest cost, false positives, long triage cycles, poor historical visibility<br>▪ Define measurable outcomes: e.g., reduce SIEM ingest by 30%, enable 12-month hunting, add ML detections<br>▪ Create a business justification statement to support funding and executive sponsorship |

Before building pipelines or writing detection logic, clarify your organization's goals:

- What are the key security or operational problems you need to address?
- Which data sources are most critical for visibility and detection?
- Where are the pain points in your current process or tooling?
- What is your "as is" state versus your "to be" state?

## Step 1: Inventory and Prioritize Data

**Objective:** Inventory, score and prioritize log sources to align your ingestion strategy with the highest-impact use cases.

**Outcome:** A prioritized list of log sources with scoring rationale that informs your ingestion, normalization, enrichment and detection plans.

| | |
|---|---|
| **Key Principles** | ▪ Not all telemetry is equal. Prioritize based on detection value, threat coverage, regulatory requirements and investigation needs.<br>▪ Use top-down mapping from business risk to telemetry coverage<br>▪ Apply structured evaluation to avoid reactive or ad hoc ingestion decisions |
| **Guidance** | ▪ Document the current-state security architecture, including log sources, tooling, ownership and data flow<br>▪ Identify core challenges such as high ingestion cost, excessive false positives, long triage cycles or limited historical context<br>▪ Define measurable objectives such as reducing telemetry overload, extending retention windows or enabling specific detection use cases<br>▪ Create a business case that aligns data priorities with threat coverage, compliance needs and operational efficiency |
| **Best Practices** | ▪ Start with identity, endpoint and cloud logs because they provide the broadest ATT&CK and detection surface coverage<br>▪ Document each source's schema, format, expected volume, retention requirements and current gaps in coverage |

## A NOTE ON HIGH-VALUE DATA SOURCES

Ingesting everything can be costly and operationally complex. Focus first on high-value data sources that support core detection use cases, reduce blind spots and enrich other signals.

The table below highlights common telemetry types that are frequently prioritized.

| DATA SOURCE | WHY IT MATTERS |
|---|---|
| Windows Security Logs (AD/DC) | Core for identity-based detections. Supports brute-force, lateral movement and privilege abuse detection. |
| Firewall and Network Logs | Captures ingress/egress activity and indicators of exfiltration or lateral movement. |
| EDR Logs | Provides high-fidelity process telemetry. Essential for detecting malware, suspicious behavior and post-exploitation activity. |
| DNS Logs | Identifies command-and-control activity, domain generation algorithms and suspicious domain usage. Complements endpoint and network visibility. |
| Cloud Audit Logs (e.g., AWS CloudTrail, Azure Activity) | Tracks IAM changes, API usage, resource creation and misconfigurations. Crucial for cloud-native threat detection and compliance monitoring. |

## Step 2: Ingest and Normalize

**Objective:** Establish scalable, schema-enforced pipelines that convert raw logs into structured tables suitable for detection and investigation.

**Outcome:** A governed, scalable pipeline that turns raw telemetry into structured, queryable and reliable inputs for detection and analytics.

| | |
|---|---|
| **Key Principles** | ▪ Ingest first, transform later. Land raw data in Bronze tables to preserve auditability and enable replay.<br>▪ Enforce schemas early to support downstream reliability<br>▪ Follow the medallion architecture to separate raw, normalized and analytical outputs for flexible downstream use |
| **Guidance** | **Ingestion methods:**<br>▪ Use **Auto Loader** for cloud-native logs (S3, GCS, ADLS)<br>▪ Use **Structured Streaming** for real-time sources (Kafka, syslog)<br>▪ Support **batch file** drops or **REST API** ingestion for legacy sources<br>**Normalization approach:**<br>▪ Parse and flatten nested structures<br>▪ Standardize core fields: `event_time`, `src_ip`, `user_name`, `event_type`<br>▪ Add metadata fields: `source_type`, `parser_version`, `workspace_id`<br>▪ Register normalized (Silver) tables in Unity Catalog for governance, discovery and access control |

| | |
|---|---|
| **Guidance** | **Schema design:**<br>▪ Align with OCSF where feasible, but prioritize practical execution<br>▪ Version schemas and store normalization logic in Git<br>**Parser example:** CloudTrail pipeline using PySpark to explode Records array, add ingest metadata and write to `cloudtrail_bronze` |
| **Best Practices** | ▪ Keep Bronze in raw format with minimal transformation<br>▪ Drive all detection logic off Silver<br>▪ Use comments and table annotations to improve discoverability |

## GOVERNANCE AND LINEAGE

Effective security data pipelines require clear ownership, access control and traceability across raw, normalized and enriched datasets. Databricks supports these requirements through Unity Catalog, which provides fine-grained access control, audit logging and data lineage tracking at the table, column and query levels.

**Unity Catalog allows teams to:**

- **Restrict access** to sensitive log data based on user roles (e.g., least-privilege access to identity logs or process telemetry)
- **Segment environments** by business unit, region or tenant using catalogs, schemas and table naming conventions
- **Track lineage** from raw (Bronze) to normalized (Silver) to enriched or alert (Gold) tables, ensuring traceability of detections and investigations
- **Enforce auditability** of user queries, schema changes and pipeline runs for compliance and incident response

Governance considerations also include:

- **Masking sensitive fields** like passwords, email addresses or internal hostnames
- **Tagging data** for classification (e.g., PII, regulated logs, critical infrastructure)
- **Controlling LLM access** if using generative AI or triage agents downstream

While Unity Catalog provides robust primitives to support all of the above, a detailed governance strategy depends on your security model, regulatory requirements and data maturity. A full deep dive into Unity Catalog configuration, identity federation and lineage visualization is beyond the scope of this blueprint. However, any deployment should treat governance as a first-class design requirement, not a bolt-on.

## INGESTION CONSIDERATIONS

Databricks supports several ingestion strategies depending on the source type and latency requirements:

| SOURCE TYPE | METHOD | NOTES |
|---|---|---|
| Cloud audit logs (e.g., AWS CloudTrail, GCP Audit Logs) | Autoloader and Cloud Storage | Ideal for landing structured JSON/CSV/Parquet from S3, GCS or ADLS. Supports schema inference and incremental load. |
| Streaming logs (e.g., Kafka, syslog, Fluentd) | Structured Streaming | Use for near real-time ingestion. Integrate with Kafka, Kinesis or custom forwarders. |
| On-premises or legacy logs | Batch File Upload or REST API | Secure file transfers via cloud storage or intermediary tools that forward logs to cloud object stores. |

> Please refer to the Appendix for additional technical details and example code.

## MEDALLION ARCHITECTURE FOR CYBERSECURITY LOG DATA

A medallion architecture organizes your data pipelines into layered stages (Bronze, Silver and Gold) to support scalable, maintainable log processing for security use cases.

### 1 Bronze layer (raw ingest)

This layer stores raw logs as landed from the source with minimal transformation. It preserves the original format for auditability and replay.

**Examples:**

- Raw JSON from AWS CloudTrail
- Okta logs from an S3 drop
- syslog ingested via Kafka

**Best practices:**

- Partition by date and source
- Add metadata fields: `ingestion_time`, `source_file`, `log_source`

### 2 Silver layer (normalized and enriched)

This layer contains structured tables ready for detection logic and correlation. Raw data is parsed, cleaned and normalized to a consistent schema, and optionally enriched.

**Examples:**

- `cloud_activity`
  normalized CloudTrail and Azure logs
- `auth_events`
  login successes, failures, MFA
- `endpoint_processes`
  Windows or EDR logs (normalized)

**Best practices:**

- Align schemas to OCSF or internal standards
- Flatten nested structures and unify field names
- Add enrichment fields (e.g., `geo_country`, `user_role`, `asset_type`)

### 3 Gold layer (detection and analytics outputs)

This layer stores alert artifacts, summary dashboards and risk-scored entities. It supports downstream consumption by SOCs, SIEMs or business intelligence tools.

**Examples:**

- `alerts_table` (detection results with severity, rule_id, status)
- `user_risk_scores` (aggregated risk by user or account)
- `incident_timeline_views` (stitched events across data sources)

**Best practices:**

- Include detection metadata (e.g., rule version, confidence)
- Maintain alert lifecycle fields (e.g., `created_time`, `status`, `last_updated`)
- Create query-optimized views for dashboards or BI tools

## Step 3: Author and Operationalize Detections

**Objective:** Implement detection-as-code by writing, versioning and executing scalable, high-fidelity detection logic on normalized security data in Databricks.

**Outcome:** A modular, scalable detection pipeline that supports both real-time and retrospective alerting with explainable logic and full metadata.

**Key Principles**

- Author detections on Silver tables
- Include metadata in every alert
- Treat detection logic as code: testable, versioned and modular

| | |
|---|---|
| **Guidance** | **Authoring approaches:**<br>▪ SQL: Ideal for prototyping and writing simple correlation rules. SQL Pipeline Syntax provides familiar constructs to users who prefer SPL/KQL.<br>▪ PySpark: Supports complex, multisource or behavioral detections using structured logic and joins<br>**Execution models:**<br>▪ Batch: Scheduled via Lakeflow Jobs<br>▪ Streaming: Real-time detection via Lakeflow Declarative Pipelines and Structured Streaming<br>▪ Hybrid: Score-based or sliding window models<br>**Alert output schema:**<br>▪ Required fields: `rule_id`, `rule_name`, `event_time`, `severity`, `confidence`, `source_table`, `raw_event`, `status`<br>▪ Store alerts in `gold.alerts_table` partitioned by `alert_time` |
| **Best Practices** | ▪ Maintain detection logic in Git with CI/CD pipelines for versioning and testing<br>▪ Parameterize rules for multi-tenant environments<br>▪ Tag detection rules with MITRE technique IDs if applicable |

## Step 4: Route Alerts and Integrate With SOC Tools

**Objective:** Establish reliable, bidirectional workflows between Databricks and existing SOC platforms to support alert routing, case enrichment and full-loop integration.

**Outcome:** Databricks serves as the centralized intelligence layer for detection and analytics while integrating with SIEM, SOAR and case management systems to preserve existing triage workflows and enhance downstream investigation and response.

| | |
|---|---|
| **Key Principles** | ▪ Detection happens in Databricks; triage happens in downstream SOC platforms<br>▪ Routing must be reliable, scalable and aligned with existing workflows<br>▪ Alerts should include all metadata needed for prioritization and investigation<br>▪ All interactions should follow consistent schema standards and tagging conventions |
| **Guidance** | **Outbound: Route alerts and insights**<br>▪ Write all alerts to a Gold-layer Delta table (e.g., `alerts_table`) partitioned by `alert_time`<br>▪ Include standard fields: `rule_id`, `severity`, `confidence`, `alert_context`, `src_ip`, `user_id`, `event_time`, `raw_event_link`<br>▪ Implement routing workflows using batch (Lakeflow Jobs) or streaming (Structured Streaming):<br>  ▪ **Splunk:** HEC endpoint via Python notebook or REST API<br>  ▪ **Sentinel:** Log Analytics API or Event Hub<br>  ▪ **ServiceNow:** REST API for ticket creation<br>  ▪ **Slack/Teams:** Webhooks for ChatOps alerts<br>  ▪ **Delta Sharing:** For MSSPs, partners or multi-tenant delivery |

| | |
|---|---|
| **Guidance** | **Inbound: Ingest alerts and incidents**<br>▪ Common ingestion methods:<br>  ▪ **Splunk:** REST API, HEC export, JDBC or S3 export<br>  ▪ **Sentinel:** Azure Data Export to Event Hub or REST API<br>  ▪ **Elastic/QRadar:** Kafka or JSON file drop<br>▪ Normalize alert schema to match internal `alerts_table`<br>▪ Tag all imported data with `source_system`, `ingest_time` and `source_type`<br>▪ Schedule periodic ingestion using Lakeflow Jobs |
| **Best Practices** | ▪ Include raw event links in outbound alerts for fast pivoting during investigations<br>▪ Normalize and store external alerts using the same table structures as internal detections<br>▪ Use deduplication logic and alert state tracking to avoid noisy or repeated notifications<br>▪ Monitor integration health using dashboards and alert on ingestion failures or latency spikes<br>▪ Treat integrations as modular building blocks — enable phased adoption per tool or team |

## Step 5: Support Investigation and Triage

**Objective:** Provide analysts with a flexible, scalable investigation environment that enables rich context, deep pivoting and collaborative analysis without relying on legacy SIEM interfaces.

**Outcome:** Analysts can investigate alerts directly in Databricks using dashboards, structured queries and parameterized notebooks. This accelerates triage, reduces noise and enables deeper visibility across identity, process, network and cloud data sources.

| | |
|---|---|
| **Key Principles** | ▪ Every alert must include enough context to answer: Who? What? When? Where? How?<br>▪ Analysts need flexible interfaces tailored to their workflows, including dashboards, notebooks and SQL queries<br>▪ Investigation workflows should be repeatable, scalable and collaborative |
| **Guidance** | Build dashboards in Databricks SQL for:<br>▪ Alert trend analysis<br>▪ Entity frequency, geo dispersion and behavior heat maps<br>▪ Alert-to-source drilldowns<br><br>Create templated triage notebooks:<br>▪ Parameterize by alert_id or user_id<br>▪ Join with logs from identity, process, cloud and network sources<br>▪ Visualize timelines, graphs and contextual metadata |

| | |
|---|---|
| **Guidance** | Integrate enrichment sources:<br>▪ Asset inventory, user role, CMDB<br>▪ Threat intelligence and GeoIP<br>▪ Historical behaviors and peer baselines<br><br>Enable collaborative triage:<br>▪ Analysts can annotate notebooks with markdown summaries<br>▪ Output can be handed off to IR or exported as case evidence |
| **Best Practices** | ▪ Govern investigation data using Unity Catalog for access control and lineage<br>▪ Standardize notebook templates to accelerate onboarding and reduce cognitive load<br>▪ Audit notebook usage to support compliance and quality control |

## Step 6: Deliver Security Outcomes

**Objective:** Translate detection logic, enriched telemetry and investigation insights into actionable outputs for reporting, compliance and executive visibility.

**Outcome:** A set of structured, shareable outputs that support downstream consumption across compliance, risk, executive reporting and operational dashboards. These artifacts provide evidence of control effectiveness, drive cross-functional decision-making and close the feedback loop from detection to measurable business impact.

| | |
|---|---|
| **Key Principles** | ▪ Security outcomes should be visible, measurable and tied to organizational risk and compliance goals<br>▪ Outputs must be accessible to multiple personas, including security analysts, compliance teams and business stakeholders<br>▪ Standardized reporting structures reduce manual effort, increase consistency and improve audit readiness |
| **Guidance** | Dashboards and KPIs — build dashboards for:<br>▪ Alert volumes, suppression rates and response-time metrics<br>▪ Threat coverage by log source, rule or MITRE technique<br>▪ SOC KPIs such as mean time to detect and mean time to respond<br><br>Create templated reports for:<br>▪ Control coverage and policy alignment (e.g., NIST, PCI, ISO 27001)<br>▪ Data access and lineage for regulated logs (via Unity Catalog)<br>▪ Evidence of detection logic, alerting pipelines and case response actions |

| | |
|---|---|
| **Guidance** | Generate aggregate scores for:<br>▪ Users, hosts and accounts based on alert activity and behavioral anomalies<br>▪ Asset exposure based on vulnerability, threat intel and access patterns<br>▪ Trends over time to track control effectiveness and response maturity<br><br>Deliver curated outputs to:<br>▪ SIEM or SOAR tools for triage and automation<br>▪ BI platforms (e.g., Power BI, Tableau) for executive reporting<br>▪ Ticketing systems (e.g., Jira, ServiceNow) for remediation tracking |
| **Best Practices** | ▪ Tailor dashboards and reports to audience-specific needs (e.g., SOC, risk, executive, compliance)<br>▪ Use Declarative Pipelines or scheduled Lakeflow Jobs to automate report generation<br>▪ Store historical reporting artifacts in Delta for auditability and trend analysis<br>▪ Align outputs to the same schema and logic used in detection for consistency and traceability |

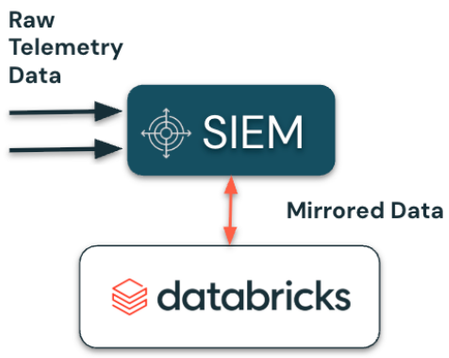
5

# Deployment Models

Building a security lakehouse does not require an overnight migration. Organizations adopt Databricks as their detection and analytics layer in phases that balance legacy constraints with future-state architecture goals. These models provide a practical approach for aligning Databricks implementation with operational maturity, data strategy and security priorities.

**Key takeaway:** These models are not mutually exclusive. Many customers begin with parallel pipelines for a specific log type or detection use case, then expand Databricks footprint over time, ultimately shifting from a SIEM-centric model to a lakehouse-first SOC.

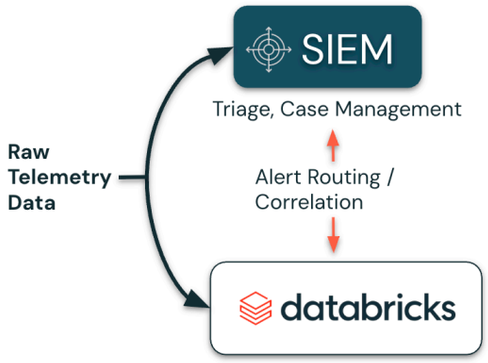
## Model 1: Visibility Overlay (Legacy-Centric SOC)




Databricks receives a copy of telemetry already routed to the SIEM. It is used primarily for cost analysis, historical hunting and proof-of-value detection development.

| BEST FOR: | BENEFITS: | LIMITATIONS: |
|---|---|---|
| ▪ Teams testing detection-as-code or ML models<br>▪ Use cases like log retention, performance tuning or shadow detections | ▪ Simple to implement<br>▪ No disruption to analyst workflows<br>▪ Enables cost benchmarking and detection quality comparison<br>▪ Often used as a “Phase 1” in modernization efforts | ▪ Databricks is not involved in real-time detection or triage<br>▪ No cost reduction in the SIEM platform<br>▪ Difficult to correlate or enrich across sources outside the SIEM |

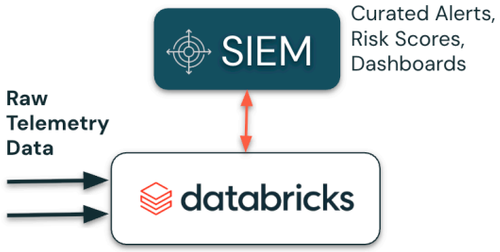
## Model 2: Dual Processing (Integrated Analytics Architecture)




Telemetry is split at the source or collection layer. Databricks and the SIEM receive data in parallel. Detection logic runs in both systems, but Databricks leads enrichment, correlation and large-scale analytics.

| BEST FOR: | BENEFITS: | LIMITATIONS: |
|---|---|---|
| ▪ Teams evaluating Databricks for advanced detection or ML use cases<br>▪ Organizations migrating off of SIEM features over time<br>▪ Use cases like UEBA, anomaly detection and alert enrichment<br>▪ Environments with mixed ownership (e.g., security owns SIEM, data team owns Databricks) | ▪ Enables offloading of high-cost queries and detections<br>▪ Supports complex analytics and detection-as-code in Databricks<br>▪ Allows selective routing to reduce SIEM license usage | ▪ Requires duplicate pipelines and potential schema alignment between systems<br>▪ Increases operational complexity and tool fragmentation<br>▪ Risk of detection drift if logic isn't centrally maintained |

## Model 3: Lakehouse-Centric SOC (Modernized Detection Backbone)




Databricks is the telemetry system of record. The SIEM becomes a downstream consumer of curated alerts, enrichment or specific log subsets.

| BEST FOR: | BENEFITS: | LIMITATIONS: |
|---|---|---|
| ▪ Organizations prioritizing open data architectures and cloud-native security pipelines<br>▪ MSSPs or global organizations with centralized detection but distributed SOC operations<br>▪ Use cases requiring advanced triage, risk scoring or ML-driven workflows | ▪ Centralizes detection, enrichment and investigation in one platform<br>▪ Reduces SIEM volume and dependency<br>▪ Enables analytics across broader time windows and data domains | ▪ Requires architectural alignment and buy-in across teams<br>▪ May require custom integration with legacy tools<br>▪ Still emerging in terms of packaged integrations |

6

# Common Pitfalls (and How to Avoid Them)

This section highlights the most common pitfalls and how to proactively mitigate them. Avoiding these pitfalls improves adoption, reduces operational drag and ensures that SIEM augmentation delivers measurable outcomes across cost, coverage and analyst experience.

| PITFALL | WHY IT MATTERS | HOW TO AVOID IT |
|---|---|---|
| **Skipping Schema Enforcement** | Detection logic becomes brittle, joins break and pipelines degrade over time | Define schemas for all Silver-layer tables; flatten and standardize fields during normalization; enforce types using Delta and Unity Catalog |
| **Writing Detections on Raw Logs** | Raw JSON changes frequently and lacks structure, making detections fragile | Normalize all telemetry to Silver tables; isolate parsing from detection logic; treat Silver tables as detection contracts |
| **Overengineering Initial Schemas** | Waiting for perfect schemas delays deployment and adoption | Start with minimal viable schemas; iterate; use schema versioning and annotations |
| **No Observability on Pipelines** | Pipeline failures can go unnoticed, creating blind spots in detection coverage | Use Declarative Pipelines expectations and metrics; log failures; monitor detection and ingestion health via dashboards |
| **Inconsistent Parser Logic Across Environments** | Parser drift leads to inconsistent detection results and harder debugging | Store parser logic in Git; tag records with `parser_version`; promote through CI/CD |

| PITFALL | WHY IT MATTERS | HOW TO AVOID IT |
|---|---|---|
| **Weak Governance and Access Controls** | Sensitive security data without access controls creates risk and slows adoption | Use Unity Catalog for access policies; mask sensitive fields; audit access and query history |
| **Assuming SIEM Replacement** | Creates friction with SOC teams and overpromises platform capabilities | Position Databricks as a detection and enrichment engine; maintain SIEM/SOAR integrations; emphasize signal quality improvement |

7

# Resources and Tooling References

This section provides links, references and tools to support the successful implementation of a security lakehouse with Databricks. It includes official documentation, open standards and sample assets for faster onboarding and execution.

## Databricks resources

- Data Intelligence for Cybersecurity eBook
- Databricks Auto Loader (AWS, Azure, GCP)
- Declarative Pipelines (AWS, Azure, GCP)
- Unity Catalog (AWS, Azure, GCP)
- Structured Streaming (AWS, Azure, GCP)
- Databricks SQL dashboards (AWS, Azure, GCP)

## Detection engineering and security content

- MITRE ATT&CK Framework
- Open Cybersecurity Schema Framework (OCSF)
- CloudTrail Log Event Reference
- Elastic Common Schema (ECS)

## Useful tools and libraries

- Library requests for alert routing via REST
- GeoLite2 (MaxMind)
- JQ for JSON flattening
- Jupyter Markdown reference

**For any implementation questions, engagement requests or support, contact your Databricks Solutions Architect or the Field Engineering Cybersecurity team.**

# Appendix

## Sample Silver Tables

| TABLE | DESCRIPTION |
|---|---|
| `auth_events` | Login successes, failures and MFA interactions |
| `cloud_activity` | Cloud API actions (e.g., CloudTrail, Azure, GCP) |
| `endpoint_processes` | Process creation, execution, parent-child relationships |
| `dns_queries` | Query name, source IP, domain classification |
| `network_flows` | Source/destination IP, port, protocol, bytes in/out |

## Parser and Schema Normalization

Normalization is critical to enabling scalable, reusable detections and efficient investigation workflows. It involves transforming raw logs into structured, query-friendly Delta tables with consistent field names, formats and types across sources.

### RECOMMENDED APPROACH: BUILD WITH NOTEBOOKS AND LAKEFLOW JOBS

We recommend building normalization logic using standard Databricks Notebooks and automating it with Lakeflow Jobs or Declarative Pipelines for repeatable, scalable execution.

**This pattern gives you:**

- Full control over parser logic (JSON flattening, regex, type coercion)
- Support for custom enrichment (e.g., GeoIP, asset tagging)
- Clear separation between raw ingestion (Bronze) and normalized tables (Silver)

**Suggested development pattern**

1. Create a raw landing table (Bronze) using Auto Loader or a streaming source
2. Build a notebook that:
   - Reads raw data
   - Applies schema validation
   - Performs field extraction, transformation and enrichment
   - Writes to a normalized Delta table (Silver)
3. Schedule the notebook using Lakeflow Jobs or Declarative Pipelines with event-driven or time-based triggers
4. Add table comments and register in Unity Catalog for discoverability and access control

## Parser Design Recommendations

- Parse and flatten all nested JSON structures at ingest
- Extract a consistent `event_time` field using UTC and handle time zone shifts
- Standardize field names across similar sources (e.g., `user_name`, `src_ip`, `dest_ip`, `event_type`)
- Tag records with metadata fields such as `source_type`, `log_vendor`, `parser_version` and `workspace_id`

## PARSER EXAMPLE: CLOUDTRAIL TO CLOUDTRAIL_BRONZE TABLE

This example demonstrates how to build a basic CloudTrail ingestion pipeline using PySpark on Databricks. The script reads raw AWS CloudTrail logs in JSON format from an S3 bucket, explodes the Records array to isolate individual events, and adds operational metadata such as ingestion timestamp and source file path. The resulting flattened events are written to a Bronze Delta table (kristin.aws_anomaly.cloudtrail_bronze) for persistent storage and further processing.

This is a common pattern for staging raw telemetry before normalization and enrichment.

```
# Set up paths
s3_path = 's3://my-s3-bucket/aws_cloudtrail/'
bronze_table_name = 'kristin.aws_anomaly.cloudtrail_bronze'

# Read CloudTrail logs from S3
df_raw = spark.read.schema(cloudTrailSchema).json(s3_path)

# Explode the Records array to get individual events
df_bronze = df_raw.select(explode("Records").alias("record")) \
   .select("record.*")

# Add metadata columns after exploding
df_bronze = df_bronze.withColumn("ingest_timestamp", current_timestamp()) \
   .withColumn("source_file", input_file_name())

df_bronze.write.format('delta') \
   .mode('append') \
   .option('mergeSchema', 'true') \
   .saveAsTable(bronze_table_name
```

## About Databricks

Databricks is the data and AI company. More than 10,000 organizations worldwide — including Block, Comcast, Condé Nast, Rivian, Shell and over 60% of the Fortune 500 — rely on the Databricks Data Intelligence Platform to take control of their data and put it to work with AI. Databricks is headquartered in San Francisco, with offices around the globe, and was founded by the original creators of Lakehouse, Apache Spark™, Delta Lake and MLflow. To learn more, follow Databricks on LinkedIn, X and Facebook.

databricks