$ opensearchcon --india --2026 --track security

REAL-TIME THREAT HUNTING ON A BUDGET

OpenSearch *vs.* The Unknown

Turning the search engine you already run into a scrappy, surprisingly powerful security command center: no license key required.

Prerit Munjal
Product & Engineering

GROUPON

$ whoami && why-this-talk

Security isn't just firewalls and alerts.

It's curiosity, context, and catching the weird stuff *before it gets weird.*

Budget

Enterprise SIEMs bill per-GB ingested. Costs scale with your data volume — not with the value you get.

Patience

Months of onboarding, field tuning and license negotiation before you catch a single anomaly.

Fit

Tools feel either overkill for a small team, or underwhelming the day a real threat shows up.

So we asked: **what if the search engine we already run could be the SOC?**

$ cat /etc/company/groupon.yaml

Where I build: Groupon

GROUPON

The marketplace for local experiences — connecting customers with the local merchants who power Main Street.

"Get people offline — through quality local experiences at great value."

MY CORNER OF IT

Data platform · observability · security & org tooling — the plumbing under all of it.

Millions

of local merchants powering Main Street

2008

founded in Chicago · NASDAQ: GRPN

AI-native

rebuilding for agentic commerce — the world this talk lives in

Personal views — not an official statement of Groupon, Inc. · Figures from Groupon's public Q1 2026 results (May 2026).

$ cat invoice.pdf | grep TOTAL

The math that started it all

TYPICAL ENTERPRISE SIEM

$500K–$2M / yr

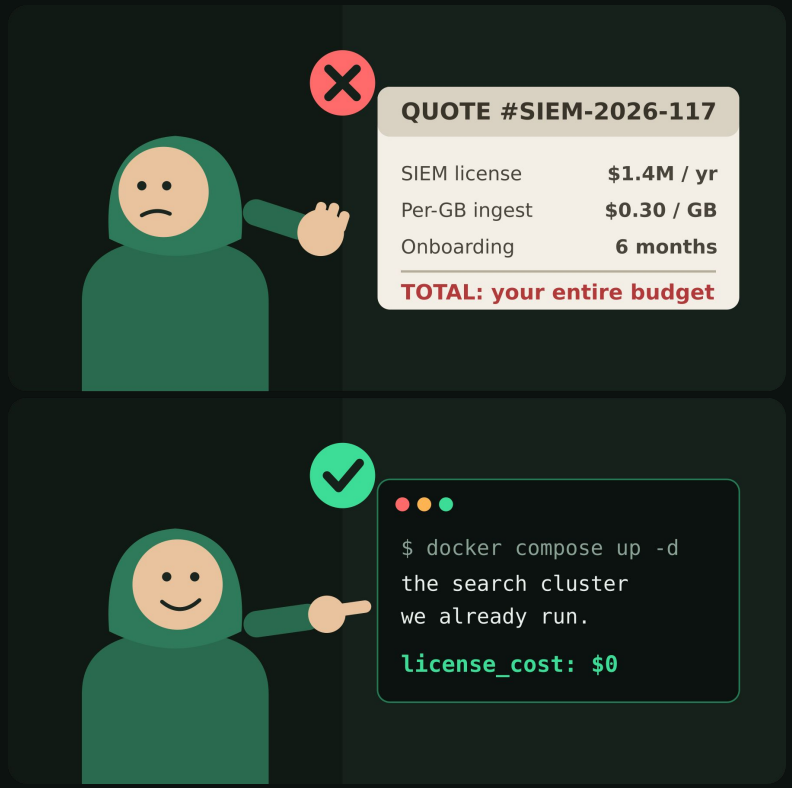
before you've caught a single anomaly · per-GB pricing · vendor lock-in

OPENSEARCH SECURITY ANALYTICS

$0 in licensing

Apache 2.0 · 2,200+ Sigma rules built in · pay only for infra you'd run anyway

Same detection logic. Two very different invoices.



$ cat hunt-plan.md

What you'll walk away with

01

Anomaly-first log schemas

Design logs so the weird stuff is loud — not buried in noise.

02

Detection pipelines

Ingest processors + Security Analytics: Sigma rules, detectors, correlation.

03

Real-time alerting

Monitors that ping Slack the moment something matches — for free.

04

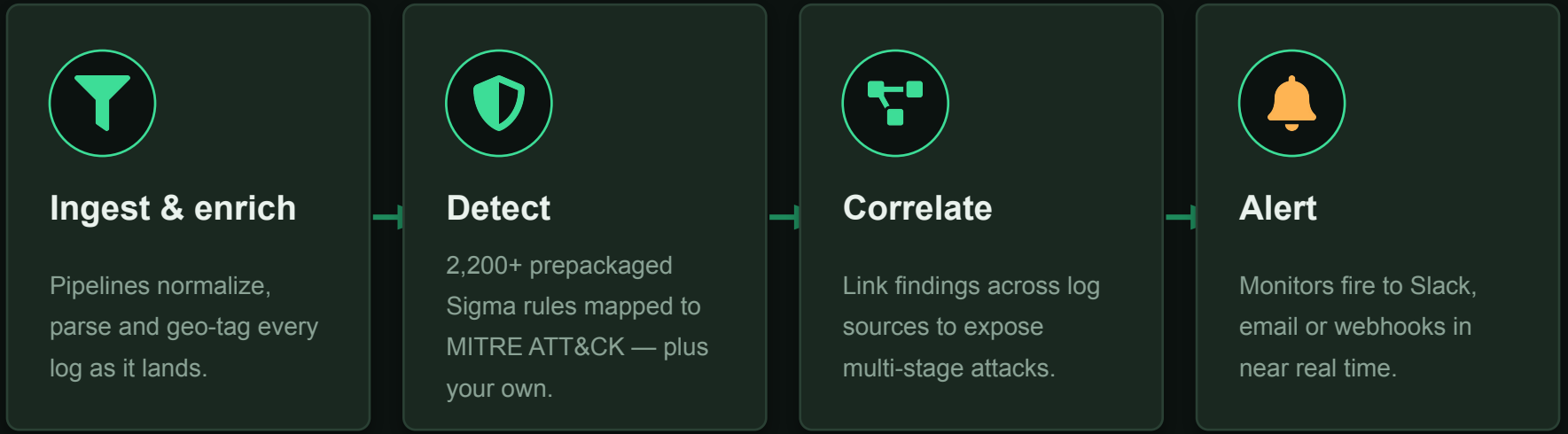
A real incident

The time OpenSearch caught what our cloud provider's tooling missed.

```
$ opensearch-plugin list | grep security-analytics
```

A SIEM hiding inside your search cluster

Security Analytics ships with every OpenSearch distribution. Four native pieces do the heavy lifting:



All open source. All running on the cluster **you may already operate** for logs and search.

$ explain --how-opensearch-stores-data

The data model — and how it's physically stored

logical ▸ distributed ▸ physical

1 · LOGICAL MODEL

what you query against

INDEX

auth-logs

a named collection of documents + a mapping (schema)

DOCUMENT — JSON

client_ip	:	ip
@timestamp	:	date
user	:	keyword
event.action	:	keyword
event.hour	:	integer

FIELD TYPE = ON-DISK SHAPE

keyword → doc values + exact match
text → analyzed inverted index
ip / date → range-optimized

types are ~permanent: changing one means reindexing the whole index.

one index, stored physically ↓

2 · DISTRIBUTED ACROSS NODES

the index is split into shards; each shard is a complete, independent Lucene index. Primaries hold the data; replicas are copies on another node.

routing: hash(_id) % number_of_primaries → picks the shard · number_of_shards is fixed at creation.

inside one shard ↓

3 · INSIDE A SHARD (on disk)

SHARD P0 = one Lucene index

- in-memory buffer (not yet searchable)
- translog — write-ahead log · durability

immutable segments

- segment_1 — mini inverted index
- segment_2 — mini inverted index
- segment_3 — mini inverted index

INSIDE ONE SEGMENT

- **inverted index** — full-text + exact match
  svc-deploy → [d3, d9] user_login → [d3, d7, d12]
- **doc values** — sort + aggregate
  event.hour: [3, 14, 9, 11, 3 …] columnar
- **stored fields** — fetch / return
  _source: { "client_ip":…, "user":… }

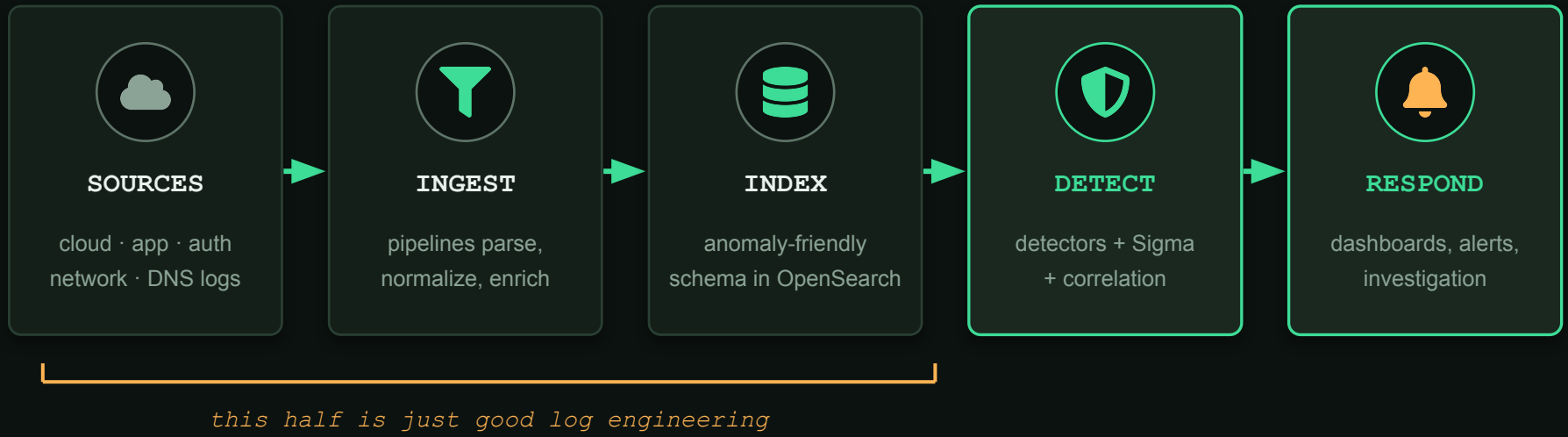
WRITE PATH

index doc → buffer + translog → refresh ~1s → searchable → flush — fsync + truncate → merge — combine + drop

OpenSearch · index → shards (primary/replica) → immutable Lucene segments → inverted index + doc values + stored source

$ draw --pipeline threat-hunting

How it fits together



The trick: get the schema right and detection becomes almost free — every rule turns into a cheap, one-line query.

$ step 1/4 — schema: make anomalies loud

Design logs that make anomalies loud

Geo-enrich every IP

ip2geo adds country, city, ASN at ingest. “Login from a new country” becomes a one-line query.

Stable identity fields

Normalize user, host, action into consistent ECS-style names so rules map cleanly across sources.

Time & baseline fields

Capture hour-of-day and rolling counts so “unusual time” and rate spikes are trivial to spot.

Tag the boring away

Drop health checks and known-good noise at ingest. Less to store, sharper signal, lower bill.

grep-ing raw logs at 2 AM,
vibes-based investigation

index everything raw
and hope a query saves you

enrich at ingest — geo, ASN,
identity, hour-of-day on every doc

drop the boring noise BEFORE it costs you storage — anomalies become one-line queries

$ step 2/4 — enrichment in a few lines

The threat-enrich pipeline

PUT _ingest/pipeline/threat-enrich

```
{
  "processors": [
    { "ip2geo": { "field": "client_ip",
        "datasource": "geo-db",
        "target_field": "geo" } },
    { "grok": { "field": "message",
        "patterns": […] } },
    { "drop": { "if": "ctx.url=='/healthz'" } }
  ]
}
```

ip2geo

Country, city & ASN from any IP — the backbone of “impossible travel” and rare-ASN detection.

grok

Pull structured fields out of messy log lines so Sigma rules have something to match on.

drop

Throw away health-check noise before it ever costs you a rupee of storage.

$ step 3/4 — from log line to finding

From log line to finding

Sigma rules

2,200+ prepackaged, MITRE-mapped detections. Duplicate one in the visual editor and tweak — or write your own YAML.

Correlation engine

Define multi-source scenarios (auth + network) inside a time window. Findings get linked into one attack story.

Anomaly detection

ML Commons learns baselines for login times or request rates, then flags the statistical outliers for you.

rare-asn-odd-hours.yml

```
detection:
  selection: { event.action: user_login, event.outcome: success }
  condition: selection and not known_asn and not office_hours   # level: high
```

A finding isn't always an attack — it isolates an event of interest worth a human's curiosity.

$ step 4/4 — get pinged before you'd ever notice

Get pinged before you'd ever notice

THE ALERT PATH

1. **Detector finds a match**
   A Sigma or custom rule hits an incoming event
2. **Trigger condition evaluates**
   Severity ≥ high, a specific tag, or count over N
3. **Notification dispatched**
   Routed to the right team's channel instantly

DELIVERED TO

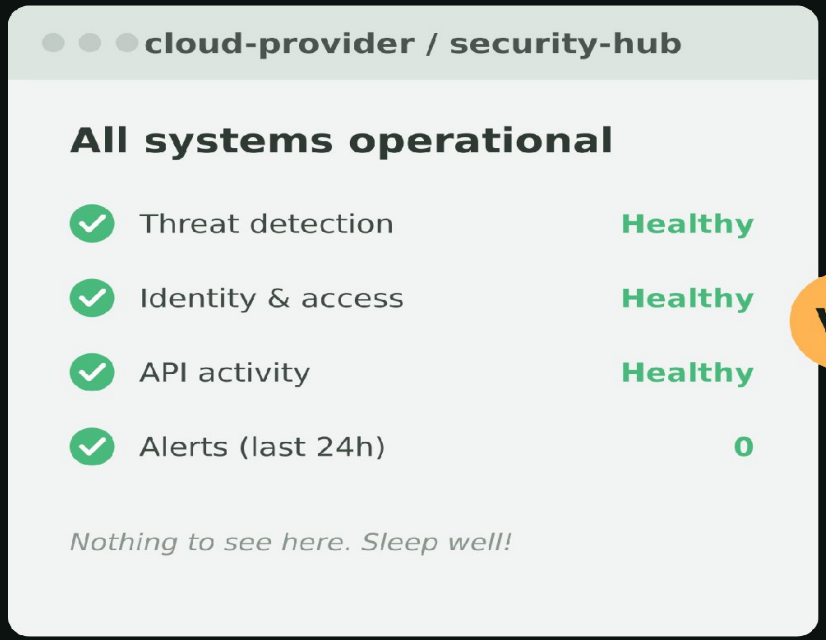
Slack
Channel ping with finding details and a deep link to investigate

Email
For on-call escalation paths and audit trails

Webhooks
PagerDuty, Jira, or your own automation — anything with a URL

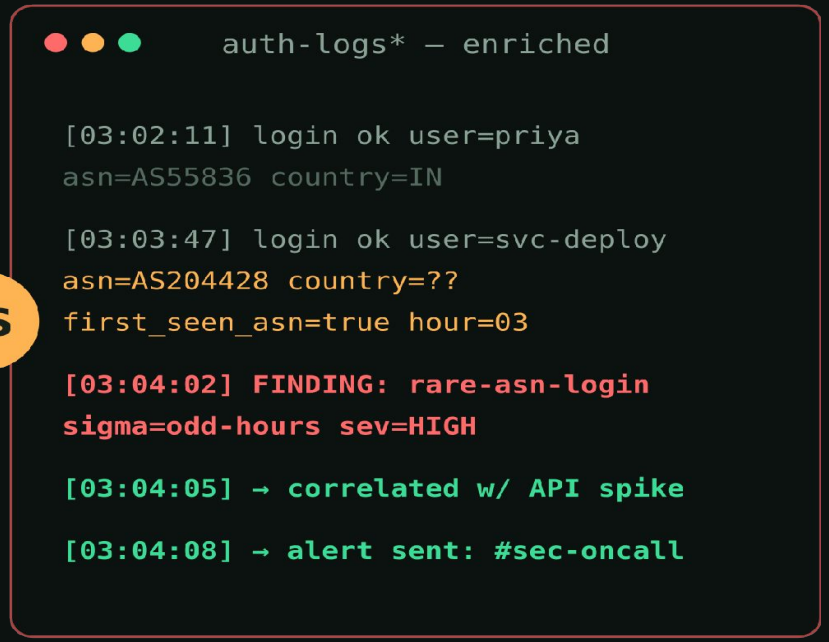
$ tail -f the-night-everything-was-fine.log

03:04 AM. Two views of the same minute.



YOUR CLOUD PROVIDER, 03:04 AM

VS



YOUR OPENSEARCH, SAME MINUTE

$ the payoff — what our cloud provider missed

What our cloud provider missed

Odd login lands

svc-deploy signs in from an ASN we've never seen, at 3 AM. Geo fields attached at ingest.

Finding raised

Custom Sigma rule (rare ASN + odd hours) matches. Provider dashboards: still green.

Correlated

Correlation links the login to anomalous API calls in a 5-min window. One attack story.

Contained

Alert hits Slack with a deep link. Keys revoked in minutes — not at the next audit.

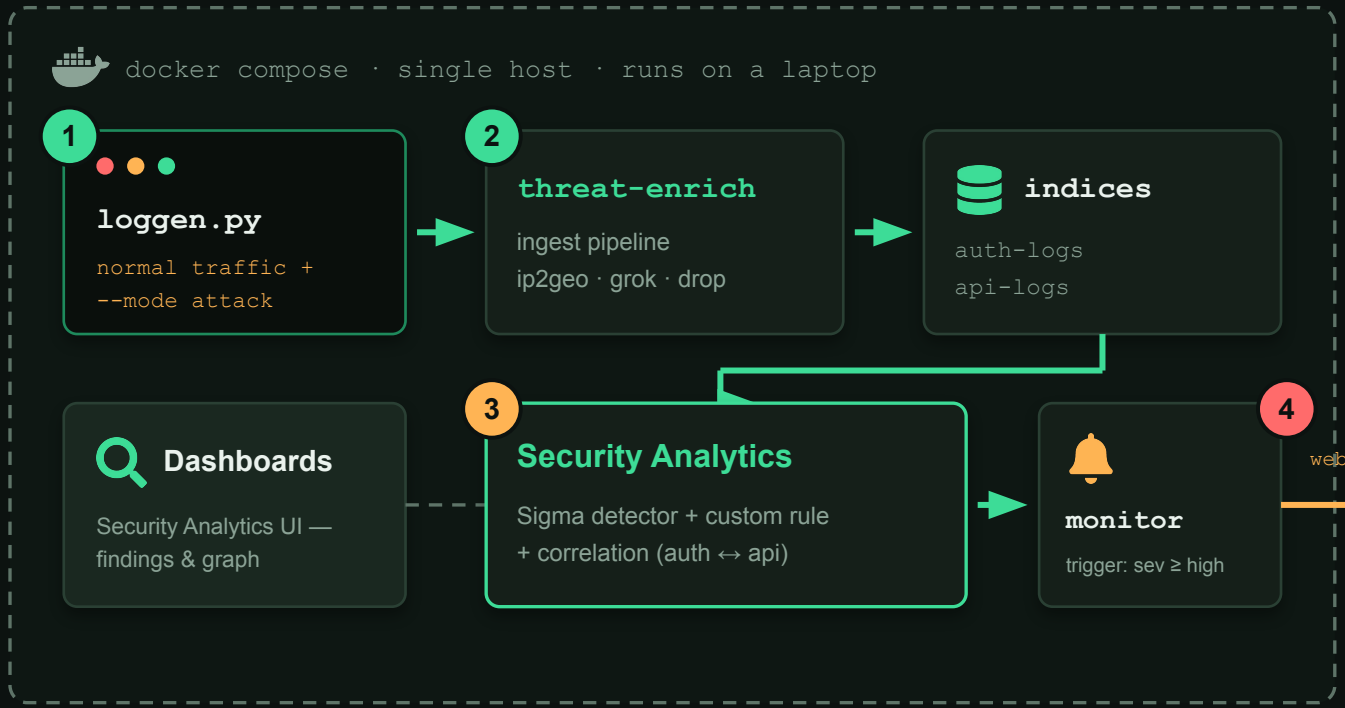
minutes
to contain — not days

$0
in new tooling spend

1
schema decision made it visible

```
$ docker compose up -d  # the whole demo
```

Live demo — the rig



Five beats, one rig.

Everything you'll see is in the public runbook — clone it and replay the attack tonight.

$ ./demo.sh --beats 5

From raw log to Slack ping — in five beats

1	**Stream logs in**	loggen.py tails realistic auth + API traffic into OpenSearch through the threat-enrich pipeline.
2	**Before / after**	_simulate one raw line: watch geo, ASN and parsed fields appear — and a health check get dropped.
3	**Detector fires**	Enable the Sigma detector + the rare-ASN custom rule; run --mode attack; a finding appears.
4	**Correlation graph**	The suspicious login links to odd API calls — one connected story on the correlation graph.
5	**Slack ping**	The monitor fires and the alert lands in #sec-oncall, live, with a deep link back to the finding.

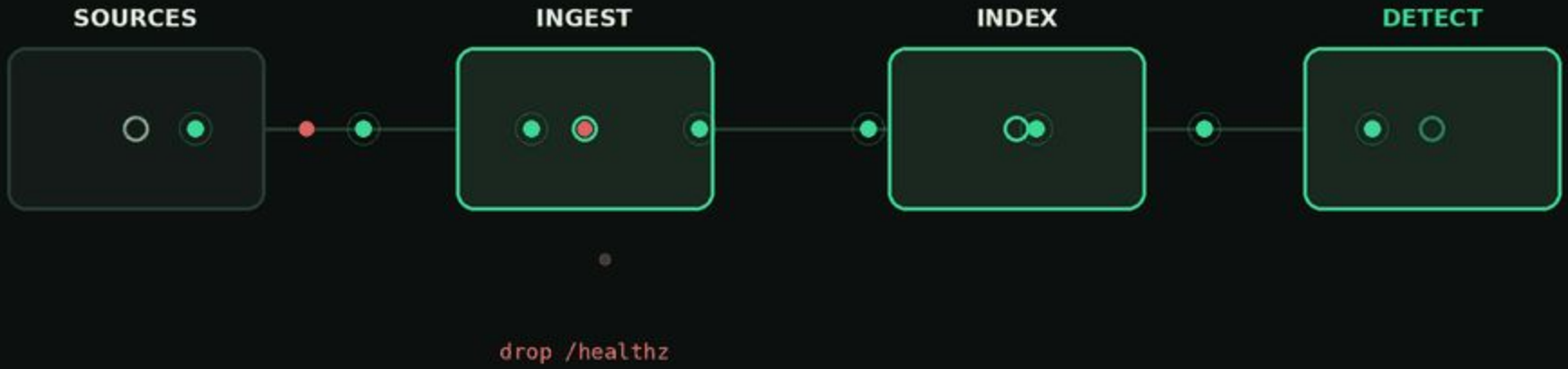

demo — bash

live monitor

auth-logs · every 1m

ALL CLEAR

$ stream --logs | enrich | detect



$ git clone your-weekend-starter-kit

Open-source recipes to steal

Spin it up

docker compose up an OpenSearch + Dashboards stack. Security Analytics is already inside.

Enrich first

Add ip2geo + grok + a drop processor before you write a single detection rule.

Borrow rules

Enable prepackaged Sigma detectors mapped to MITRE; clone & tweak the closest match.

Correlate

Write one correlation rule across two log types — where real attacks reveal themselves.

Wire an alert

One monitor → Slack. Now you find out in seconds, not at the next audit.

Hunt curiously

Schema + dashboards turn “that's weird” into a saved query you can rerun forever.

NO LICENSE KEY REQUIRED

Go catch the weird stuff.

Curiosity + context + open source beats a million-dollar invoice.

Thank you — questions?

Prerit Munjal · Engineering

GROUPON

$ exit 0