



Monitoring the great value in your Big Data

HANDS-ON WALKTHROUGH · PART 2

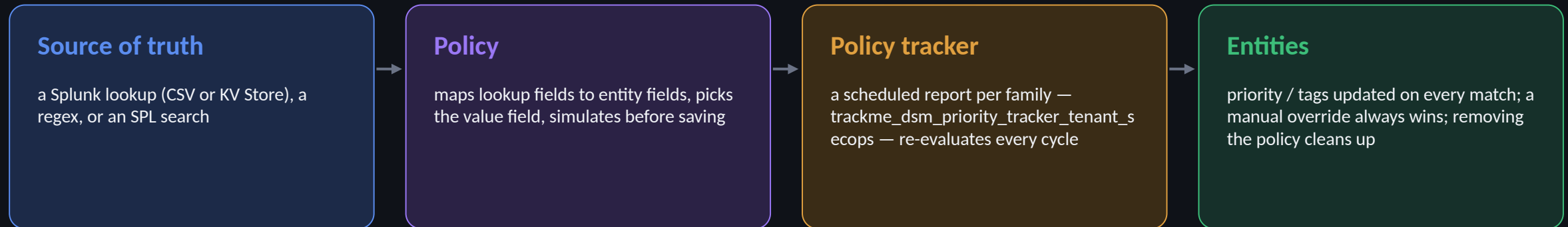
Classify with policies

A tenant where every entity is “medium” is not a monitoring posture. Two lookup-based policies — priority and tags — and the same tenant now knows what matters and who owns it.

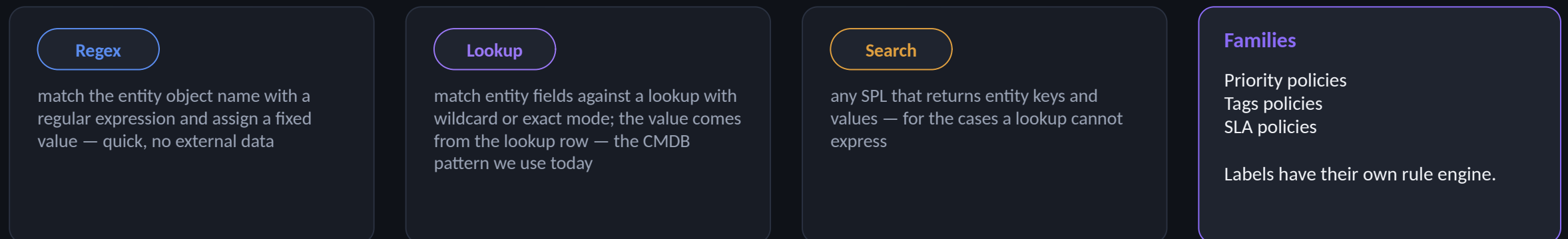


Why policies — discovery finds everything, not what matters

Priority, tags, labels and SLA are how you tell TrackMe which feeds deserve a page at 2 AM, and which team to call. Setting them one entity at a time does not survive contact with a real estate.



Three policy types, three families



Where — Tenant Home → Actions → Manage: priority policies

Actions

Open in search

Refresh all

Artificial Intelligence

AI Routines

Discovery & detections - Hybrid trackers

Execute: hybrid tracker

Manage: hybrid trackers

Create: hybrid tracker

Discovery & detections - Elastic sources

Execute: elastic tracker

Manage: dedicated elastic sources

Manage: shared elastic sources

Create: elastic source

Features

Manage: Impact score

Manage: CMDB integration

Manage: Alert enrichment

Manage: ML Outliers scope

Manage: Default delay

Manage: blocklists

Manage: lagging classes

1

Manage: priority policies

Manage: sla policies

Priority policies

What are priority policies used for

2

You can use policies to automatically influence the priority definition for TrackMe entities.

• **Regex policies:** Match entities using regular expressions against the entity object name and assign a fixed priority.

0 priorities policy currently

Deselect all

Remove selected

Save changes

Type	Account	Policy ID	Pattern / Lookup	Priority
3 No priority policies found. Click "Create new policy" to add one.				

Close

Execute tracker

3

Create new policy

1

Features section

Every classification feature is tenant-scoped and lives in the same menu: impact score, default delay, blocklists, lagging classes, priority / SLA / tags policies, logical groups, labels.

2

What are ... used for

Each policies modal opens with a short explanation of the policy types — useful during a session.

3

Create new policy

Also here: “Execute tracker” to apply policies right now instead of waiting for the schedule.

The lookup we use — example_cmdb_indexes_simple

index	priority	tags
siem-cloud-*	medium	cloud
siem-firewall-*	critical	net, firewall
siem-identity-*	critical	identity
siem-infra-*	high	infra
siem-osnix-*	low	os
siem-web-*	critical	web

...10 rows. One row per index family, wildcards in the key — this is what a CMDB export usually looks like.

TrackMe for Splunk · Hands-on: classify with policies

docs.trackme-solutions.com

Step 1 — Lookup based: pick the transform, preview the content

Priority policies: create new policy

Policy name:

cmdb-priority-by-index

1

type:

☒ Lookup based ☐ Regex matching ☐ Search based (Splunk SPL)

Splunk deployment:

Select the Splunk deployment where the lookup will be executed. Use "local" for the local instance or select a configured remote account.

local

2

1: Select lookup transform

example_cmdb_indexes_simple (csv)

Fields: index, priority, tags

3

Preview lookup content (10 sample rows)

index	priority	tags
siem-cloud-*	medium	cloud
siem-firewall-*	critical	net,firewall
siem-identity-*	critical	identity
siem-infra-*	high	infra
siem-osnix-*	low	os
siem-syslog-*	medium	syslog
siem-web-*	critical	web
siem-wineventlog-*	high	os

4

Step 2: Field mappings

Cancel

Add this new policy

- Policy name + type**
cmdb-priority-by-index · Lookup based. The other radios switch the form to regex or SPL.
- Splunk deployment**
Where the lookup is executed — local, or a remote account when the tenant monitors a remote Splunk.
- Lookup transform**
Any lookup definition visible to TrackMe; fields are read from it (index, priority, tags).
- Preview lookup content**
Ten sample rows, straight from the lookup — check the shape of the keys before mapping.

The preview is where the wildcards become visible: siem-cloud-*, siem-firewall-* ... one row covers every region of that family.

Step 2 — Field mapping and match mode

Priority policies: create new policy

> Preview lookup content (10 sample rows)

Step 2: Field mappings
Map lookup fields to entity fields. At least one mapping is required.

1 →

Add mapping

2 Add condition block (OR)
Entity matches if it satisfies block 1 OR any additional block. Within a block, all conditions must match (AND).

Match mode:
Wildcard (supports *, ?)

3 **Wildcard is much slower than Exact.** Wildcard matching cannot be indexed and is evaluated row by row, so it can be dramatically slower (a 5,000-row lookup can take a couple of minutes), whereas Exact is hash-indexed and near-instantaneous. Use Wildcard only if your lookup values actually contain * or ? patterns — otherwise switch back to Exact.

Step 3: Priority field
Select the field in the lookup that contains priority values.

4

Step 4: Priority value mappings (optional)
If your lookup uses custom priority values (e.g. "P1", "P2"), map them to TrackMe priorities. If the lookup already uses TrackMe priority values (low, medium, high, critical, pending), skip this step. Mapping keys support wildcards (*) and (?), e.g. tier1-* → critical — an exact key always wins over a wildcard.

☐ Enable foreign priority value mappings

Cancel Add this new policy

1 index → data_index

Lookup field on the left, entity field on the right. Entity fields include data_index, data_sourcetype, object, alias, tags... Add mappings for AND conditions.

2 Condition blocks (OR)

A second block matches alternative rows — an entity matches if any block matches, all conditions inside a block must match.

3 Match mode: Wildcard

Required here because the keys contain *. Wildcard is evaluated row by row; Exact is hash-indexed. Use Exact whenever the lookup has literal values.

4 Step 3 — priority field

The lookup column that carries the value: priority.

Simulate before you save — the whole tenant, classified

Priority policies: create new policy

Step 3: Priority field

Select the field in the lookup that contains priority values.

priority

Step 4: Priority value mappings (optional)

1

lookup uses custom priority values (e.g. "P1", "P2"), map them to TrackMe priorities. If the lookup already uses TrackMe priority values (low, medium, high, critical, pending), skip this step. Mapping keys support wildcards (*) and (?), e.g. tier1-* → critical — an exact key always wins over a wildcard.

☐ Enable foreign priority value mappings

Executed in 2.39 seconds

2

Run policy simulation

3

```
{ [-]
  lookup_name: "example_cmdb_indexes_simple",
  lookup_rows_count: 10,
  kvstore_collection_entities_count: 115,
  entities_matched_count: 115,
  entities_matched: [ [+] ],
  priority_distribution: { [-]
    medium: 28,
    high: 26,
    critical: 49,
    low: 12
  },
  field_mappings: { [+] },
  priority_field: "priority",
}
```

4

Cancel

Add this new policy

1 Step 4 — value mappings

Optional: if the lookup says P1 / tier1-*, map those to critical / high... Exact key wins over wildcard. Not needed with TrackMe values.

2 Run policy simulation

Executes the lookup against the tenant's KV Store in 2.4 s — no entity is touched.

3 priority_distribution

Every entity matched; the distribution shows the split by priority. entities_matched lists every one of them.

4 Add this new policy

A confirmation asks for an update comment — it goes in the audit trail with the policy.

The simulation is the sales moment: the audience sees the whole tenant re-classified from one CMDB export before anything is written.

TrackMe for Splunk · Hands-on: classify with policies

docs.trackme-solutions.com

Apply — the policy tracker, on schedule or right now

Priority policies

What are priority policies used for

- You can use policies to automatically influence the priority definition for TrackMe entities.
- Regex policies:** Match entities using regular expressions against the entity object name and assign a fixed priority.
- Lookup policies:** Match entities against a Splunk lookup (CSV or KV Store) using configurable field mappings, with priority values driven from the lookup content.
- Search policies:** Match entities using a custom Splunk SPL search query with configurable field mappings, with priority values driven from the search results.
- When a policy matches one or more entities, TrackMe will ensure that the priority matches the definition of the policy.
- If multiple policies match the same entities, the higher level of priority always wins.

1 priority policy currently

Type	Account	Policy ID	Pattern / Lookup	Priority
Lookup	local	cmdb-priority-by-index	example_cmdb_indexes_simple Index = data_index wildcard matching	from lookup

Close

Execute tracker + Create new policy

1 One policy in the table

type lookup · account local · cmdb-priority-by-index · example_cmdb_indexes_simple, index → data_index, wildcard matching · priority from lookup. Edit in place with the pencil.

2 Execute tracker

Opens “Priority policies: execute tracker” for the report trackme_dsm_priority_tracker_tenant_secops → Run tracker now. Otherwise the scheduler applies it on its next cycle.

3 Execution events

trackmesplkpriority.py logs lookup_policies_count, kvstore_collection_entities_count, entities_updated_count (every entity) and entities_failures_count (no failure), with the run time.

Priority policies: execute tracker

Execute the priority tracker for this tenant and component. The tracker will process all policies and update entities accordingly.

Report trackme_dsm_priority_tracker_tenant_secops

Run tracker now

Events (auto-refresh 15s)

Time	Event
9/8/2026 4:53:06 PM	2026-09-08 15:53:06,238 INFO trackmesplkpriority.py generate 333 tenant_id="secops", component="dsm", The tracker has terminated successfully, run_time="3.146", response={"tenant_id": "secops", "action": "success", "run_time": 0.623, "kvstore_lookup_collection": "trackme_dsm_priority_tracker_tenant_secops", "priority_policies_no_records": 1, "regex_policies_count": 0, "lookup_policies_count": 1, "search_policies_count": 0, "kvstore_collection_entities_count": 115, "entities_updated_count": 115, "entities_unchanged_count": 0, "entities_failures_count": 0, "entities_deleted_count": 0, "error_messages": []}

Close

Every family has its own tracker: trackme_dsm_priority_tracker_..., trackme_dsm_tags_tracker_..., trackme_dsm_sla_tracker_.... They are ordinary scheduled reports — visible in the Splunk scheduler and, if you monitor TrackMe with TrackMe, in WLK.

Tags policy — same lookup, same mapping, one more field

Tags policies: create new policy

> Preview lookup content (10 sample rows)

Step 2: Field mappings

Map lookup fields to entity fields. At least one mapping is required.

index → data_index

Add mapping

+ Add condition block (OR)

An entity matches if it satisfies block 1 OR any additional block. Within a block, all conditions must match (AND).

Match mode:

Wildcard (supports *, ?)

Wildcard is much slower than Exact. Wildcard matching cannot be indexed and is evaluated row by row, so it can be dramatically slower (a 5,000-row lookup can take a couple of minutes), whereas Exact is hash-indexed and near-instantaneous. Use Wildcard only if your lookup values actually contain * or ? patterns — otherwise switch back to Exact.

1 Step 3: Tags field

Select the field in the lookup that contains tag values.

tags

2 Step 4: Tags separator

If the tags field contains multiple tags in a single value, specify the separator character. Leave as comma if tags are comma-separated, or change to match your lookup format.

Run policy simulation

Executed in 1.94 seconds

Run policy simulation

Simulation results:

```
{ [-]
  kvstore_collection_entities_count: 115,
  entities_matched_count: 115,
  entities_matched: [ [+ ] ],
  entities_tags_matched_count: 10,
  entities_tags_matched: [ [+ ] ],
  result_summary: "The lookup has matched 115 entities and 10 tags",
  error_messages: []
}
```

Cancel Add this new policy

Tags policies: create new policy

Wildcard is much slower than Exact. Wildcard matching cannot be indexed and is evaluated row by row, so it can be dramatically slower (a 5,000-row lookup can take a couple of minutes), whereas Exact is hash-indexed and near-instantaneous. Use Wildcard only if your lookup values actually contain * or ? patterns — otherwise switch back to Exact.

Step 3: Tags field

Select the field in the lookup that contains tag values.

tags

Step 4: Tags separator

If the tags field contains multiple tags in a single value, specify the separator character. Leave as comma if tags are comma-separated, or change to match your lookup format.

Run policy simulation

Executed in 1.94 seconds

Run policy simulation

Simulation results:

```
{ [-]
  kvstore_collection_entities_count: 115,
  entities_matched_count: 115,
  entities_matched: [ [+ ] ],
  entities_tags_matched_count: 10,
  entities_tags_matched: [ [+ ] ],
  result_summary: "The lookup has matched 115 entities and 10 tags",
  error_messages: []
}
```

Cancel Add this new policy

1 Step 3 — tags field

The lookup column with the tag values: tags.

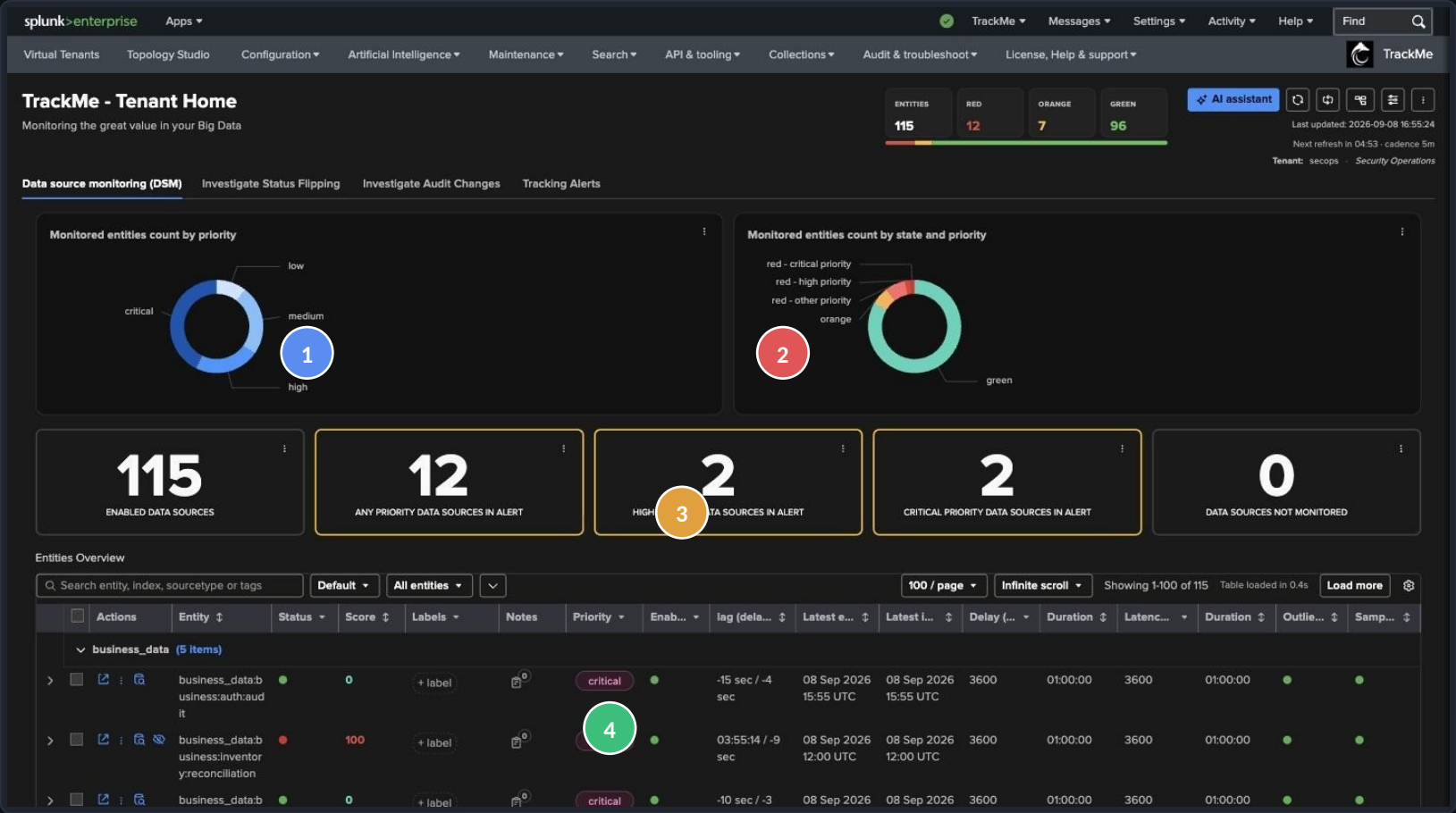
2 Step 4 — separator

net,firewall is two tags — the separator (comma by default) splits multi-valued cells.

3 Simulation

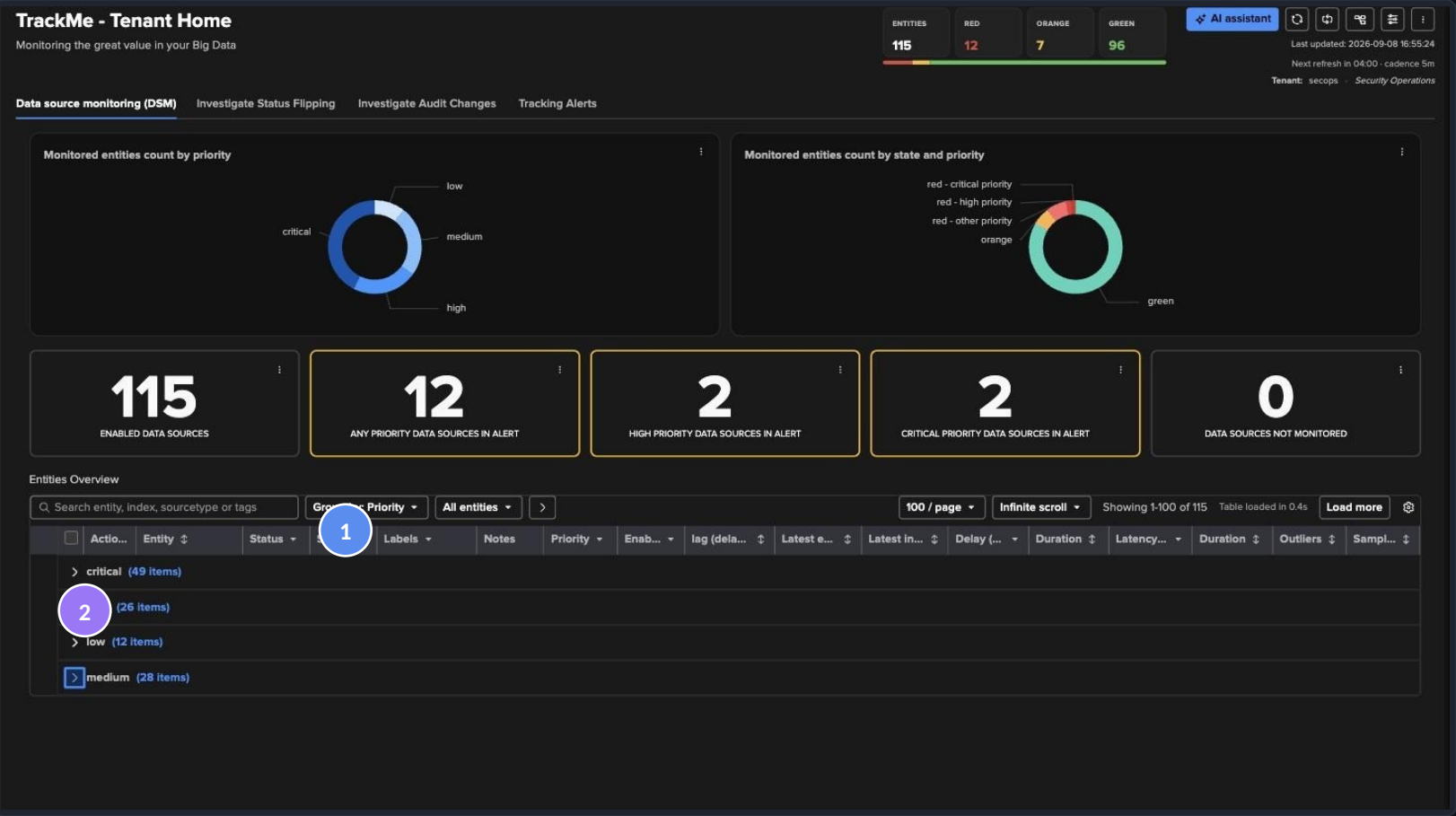
The simulation reports the entities matched and the tags found — then Add, then Execute tracker → Run tracker now, exactly as for priority.

Result — the same tenant, now with a point of view



- 1 Count by priority**
One blue disc became four — critical, high, medium, low: the CMDB's view of the estate.
- 2 State × priority**
Red now splits into red-critical, red-high, red-other. The same red entities — but now you know which ones matter.
- 3 High and critical in alert**
The single-stats that were 0 in Part 1 are populated. This is what an on-call filter looks like.
- 4 Priority chips**
Every row carries critical / high / medium / low — assigned by the policy, overridable per entity.

Group by: Priority — the on-call view



1 Grouping selector

Default · No grouping · Index · State · Priority · Sourcetype · Tags · Labels · Anomaly reason. Remembered per user.

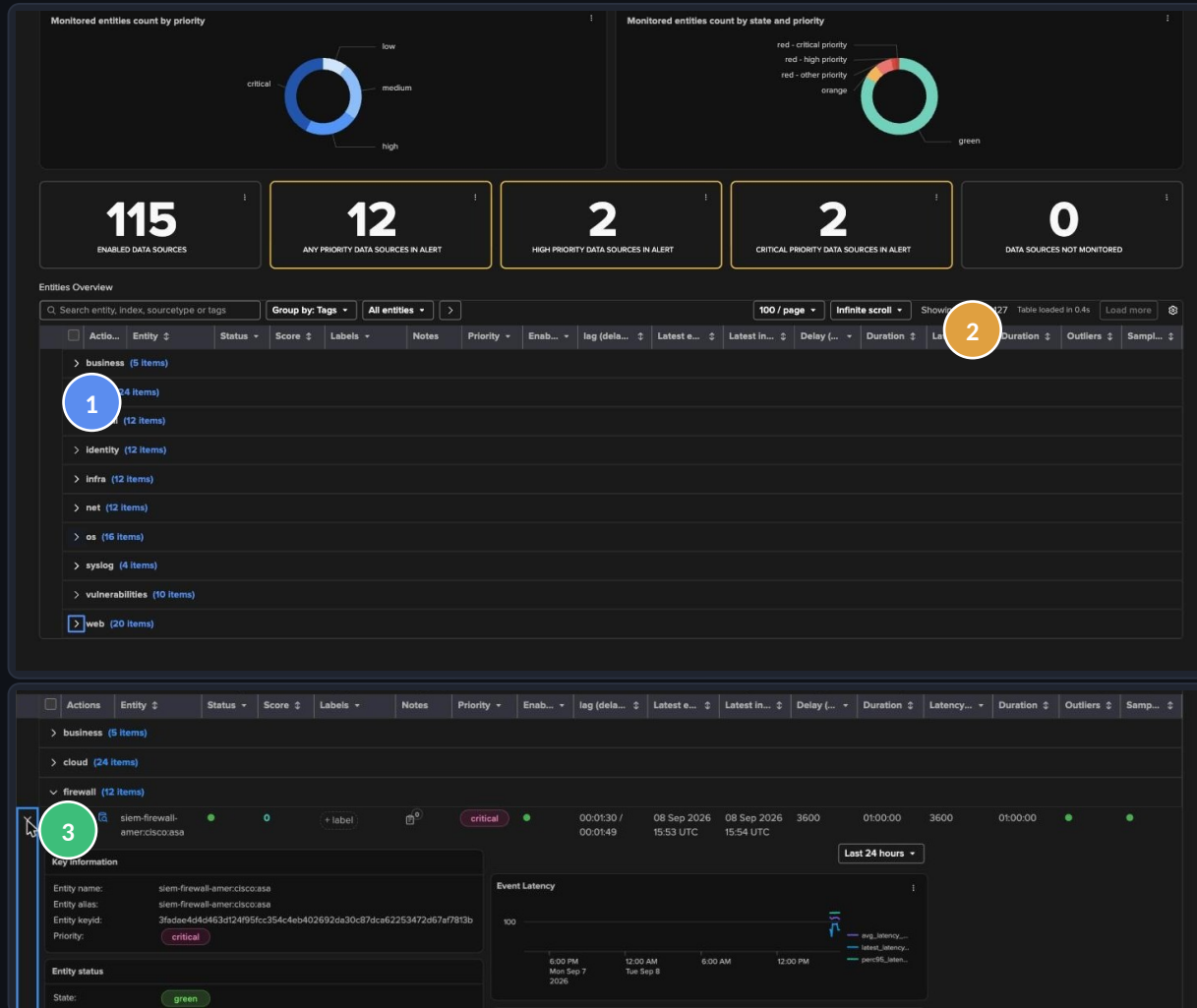
2 Four folds

critical · high · low · medium, each with its count. Expand a group to work through it; combine with the state filter to see “critical and red” only.

Also on the Virtual Tenants page

The secops card now shows its high-priority count, and the deployment-level single-stat “high priority entities in alert” is populated — the number that goes on a NOC screen or in a Virtual Group.

Group by: Tags — the ownership view



1 One fold per tag

business · cloud · firewall · identity · infra · net · os · syslog · vulnerabilities · web. Tags come from the CMDB; nobody typed them.

2 More rows than entities

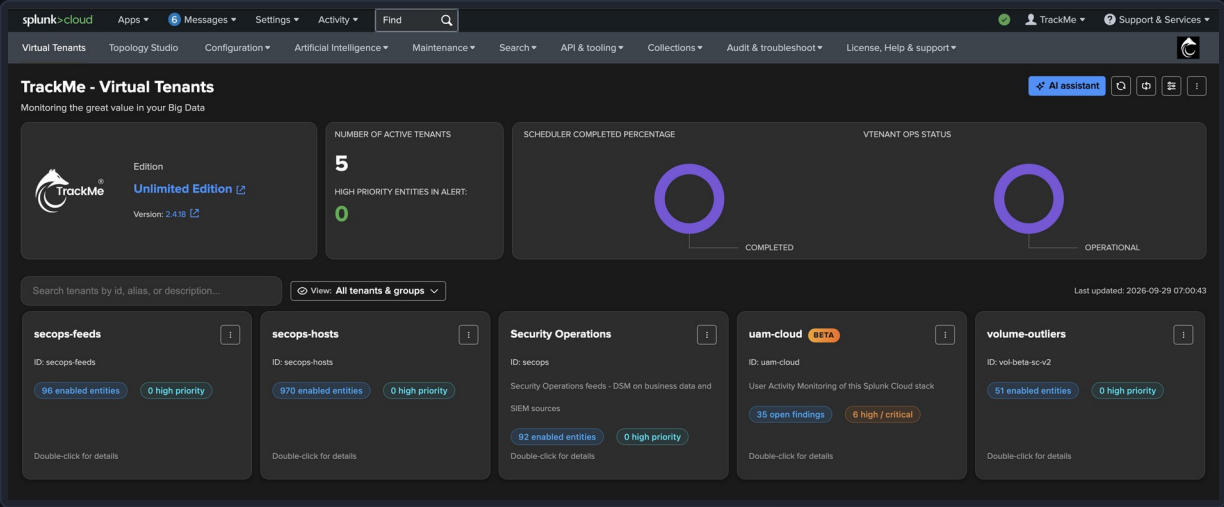
Not a bug: siem-firewall-* carries net, firewall, so those entities appear under both tags. Tags are multi-valued.

3 Inside a group

Expand firewall: siem-firewall-amer:csco:asa is critical (from the priority policy), green, delay 1m30. The inline detail shows key information, status, thresholds and the 24 h charts.

Tags also drive: the search box (type a tag), Virtual Group filters (tags=firewall), ML outliers scope, Topology Studio aggregates, and alert routing.

Where we are



In this part

- **Two policies** priority and tags, both lookup-based on one CMDB export with wildcard keys
- **Simulated** every entity matched, distribution checked before saving
- **Applied** policy trackers run on demand: every entity updated, no failure
- **Read** the tenant by priority and by tag — on-call view, ownership view

What made it work

One source of truth

a CMDB-style lookup drives both priority and tags — wildcard keys, one row per index family

Simulate first

the priority distribution is checked before anything is written to an entity

Trackers apply, overrides win

policy trackers re-evaluate every cycle; a manual override always wins; removing a policy cleans up