SQLBITES.NET

Governance as Code

Triage an inherited Fabric tenant.

Jonathan Stewart · SQLLocks / SQLBites

About Me

25+ Years in Data & Analytics

Helping enterprises migrate to and scale on Microsoft Fabric - from architecture and data modeling to Power BI and production governance.

→ **Creator/Founder: SQLBites Community**

Building the next generation of data practitioners.

→ **Migrating Enterprises to Fabric**

Real production migrations across healthcare, finance, retail, and manufacturing.

→ **Hundreds of Technical Talks Globally**

International speaker on data architecture and analytics engineering.

Sound BI

WHAT WE'LL COVER

Govern an inherited tenant, without the dashboard *as code.*

CHAPTER ONE

THE KEYS YOU WERE HANDED

01

Day-one triage, not a dashboard.

FW: Power BI admin

"Adding you as Fabric admin effective today. Sarah's last day was Friday. She said everything is documented in the wiki." The wiki has four pages. The newest is from 2023.

You've been handed the keys. No documentation. No handoff call.

What you actually inherited

Unknown workspaces

Owned by people who left. Nobody can name them all.

1

Refreshes into the void

Schedules firing for content nobody reads.

2

Spend with no name

Capacity burning with no owner attached.

3

The auditor's question

"Who has access to what?" - and one day they WILL ask.

4

Four questions. One hour.

01	**What's actually in here?**	*Inventory: workspaces, reports, models.*
02	**What hasn't been touched in months?**	*Usage: real views, not last-modified dates.*
03	**What's consuming capacity with no owner?**	*Refreshes and orphans burning money.*
04	**Who can see what - including outside the org?**	*The access question that gets people fired.*

This session answers all four. With code, against my real tenant.

Microsoft already built a report for this

Admin Monitoring workspace

Auto-provisioned for Fabric administrators. No setup.

1

Feature Usage and Adoption

Plus the Content Sharing report.

2

Inventory page

Active vs. Inactive, 30-day window.

3

Give Microsoft credit. Then find where the picture ends.

Pages
Overview
Analysis
Inventory

Feature Usage and Adoption Report | Inventory

Report last refreshed: 5/26/2026 4:17:54 PM (UTC)

Navigate to: Overview page | Analysis page | Activity Details page | Item Details page

Filters

Capacity name: All

Workspace name: All

Item name: All

Item type: All

Activity status: All

Last modified by: All

Apply filters

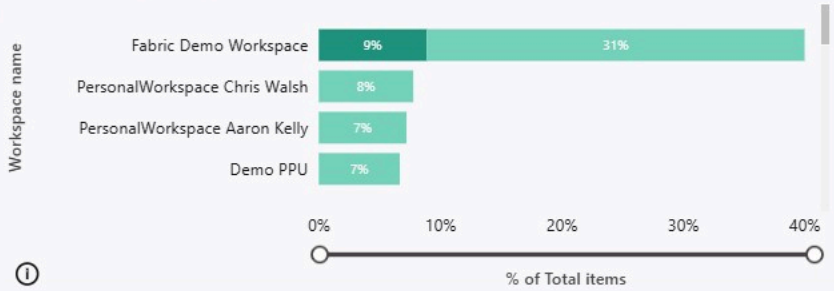
Reset filters

Active workspaces	Total items	Active items	Inactive items
6	180	28	152

Inventory status by Workspace

Right-click a visual element to drill through

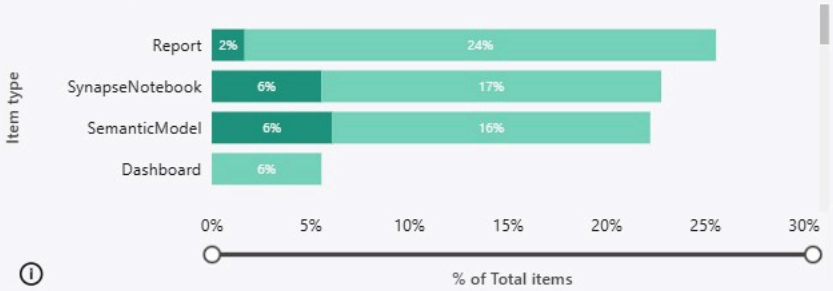
Activity status: Active Inactive



Inventory status by Item type

Right-click a visual element to drill through

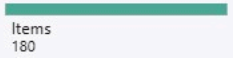
Activity status: Active Inactive



Visualize data across multiple dimensions

Right-click a visual element to drill through

Item type



Learn about this visual

Pages

Overview

Analysis

Inventory

Feature Usage and Adoption Report | Analysis

Report last refreshed: 5/26/2026 4:17:54 PM (UTC)

Navigate to: Overview page | Inventory page | Activity Details page

Filters

Date

4/25/2026 — 5/25/2026

Capacity name

All

Workspace name

All

Operation

All

User type

All

User (UPN)

All

Apply filters

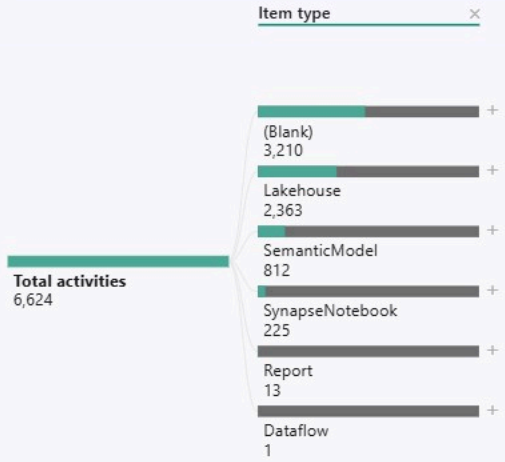
Reset filters

Total activities and users by Date

Right-click a visual element to drill through

Visualize data across multiple dimensions

Right-click a visual element to drill through



Learn about this visual

Filters

85%

Pages
Overview
Analysis
Inventory

Feature Usage and Adoption Report | Overview

Report last refreshed: 5/26/2026 4:17:54 PM (UTC)

Navigate to: Analysis page | Inventory page | Activity Details page

Filters

Date: 4/25/2026 – 5/25/2026

Capacity name: All

Workspace name: All

Operation: All

User type: All

User (UPN): All

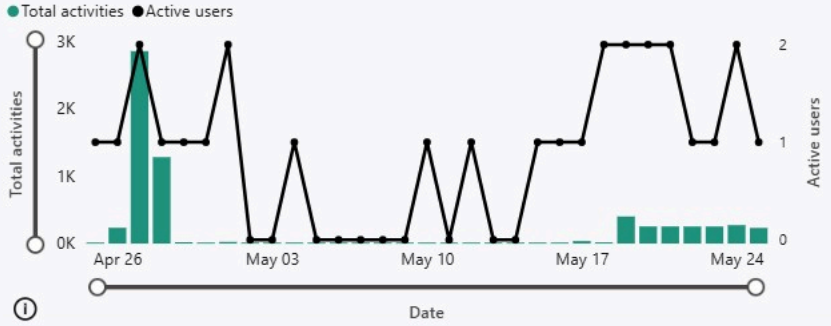
Apply filters

Reset filters

Total activities	Active users	Active capacities	Active workspaces
6,624	4	0	6

Total activities and users by Date

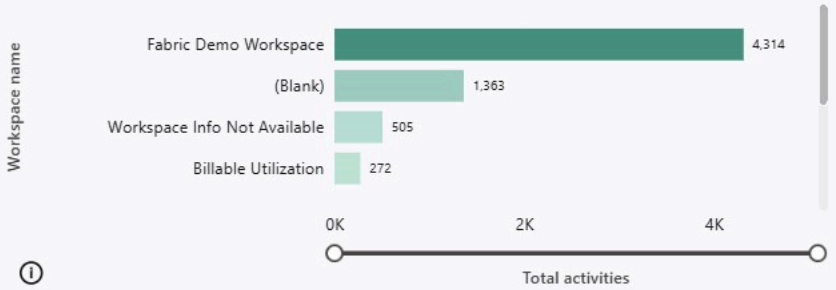
Right-click a visual element to drill through



Capacity | Workspace

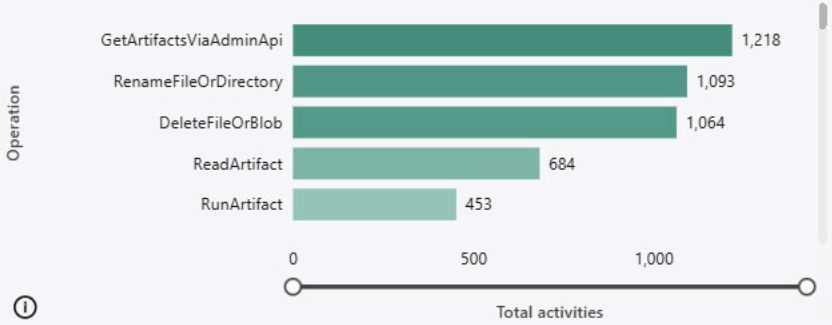
Most active Workspaces

Right-click a visual element to drill through



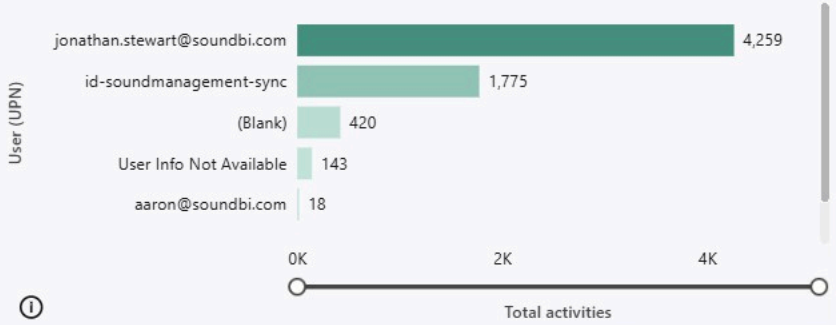
Most active Operations

Right-click a visual element to drill through



Most active Users

Right-click a visual element to drill through



Pages
Overview
Analysis
Workspace
Details

Content Sharing Report | Overview

Report last refreshed: 5/26/2026 4:17:59 PM (UTC)

Navigate to: Analysis page | Workspace page | Details page

Filters

Capacity name: All

Workspace name: All

Item type: All

Item name: All

Last modified by: All

Domain name: All

Apply filters

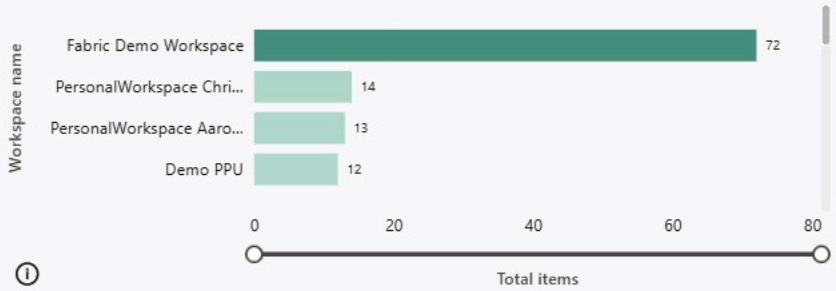
Reset filters

Total items	Total domains	Total capacities	Total workspaces
180	--	7	20

Capacity | Workspace

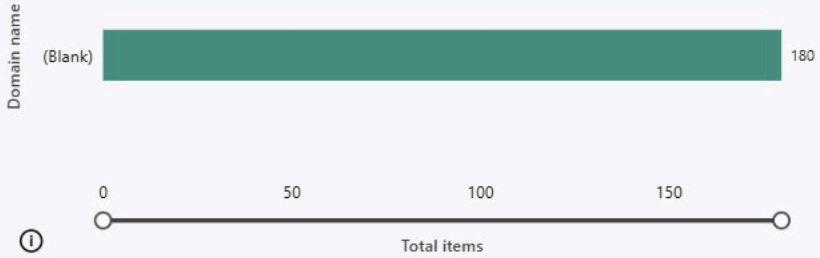
Total items by Workspace

Right-click a visual element to drill through



Total items by Domain

Right-click a visual element to drill through



Endorsement | Sensitivity

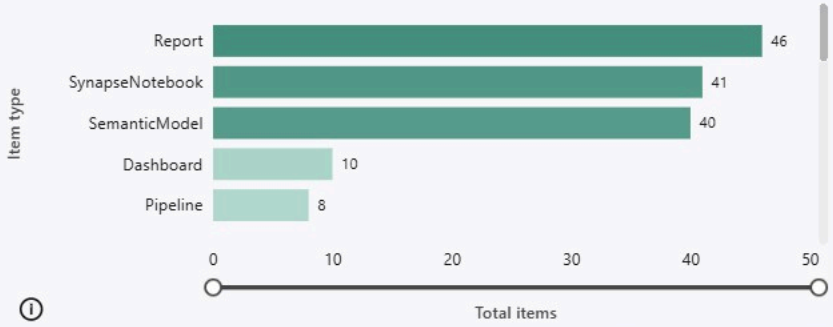
Total items by Endorsement

Right-click a visual element to drill through

Unendorsed items: 180

Total items by Item type

Right-click a visual element to drill through



The picture ends here.

847 reports

Inactive, and that is all the report knows.

It cannot tell you which to delete first, who owns them, or how to notify them at scale, every Monday, without opening a browser.

Microsoft got you to the question. We are going to answer it.

Governance as code, not governance as a dashboard

The Admin Monitoring workspace shows you the picture. The APIs are the camera. Once you own the camera, you decide what to photograph, how often, and what to do when something looks wrong.

CHAPTER TWO

FOUR ENDPOINTS, DEEPLY

02

Auth, autopsies, throttles, and one crash.

Four endpoints. That's it.

Everything the Admin Monitoring workspace knows comes from these four calls.

```
GET /v1.0/myorg/admin/groups → workspace inventory
GET /v1.0/myorg/admin/reports → all reports, all workspaces
GET /v1.0/myorg/admin/datasets → all semantic models
GET /v1.0/myorg/admin/activityevents → who opened what, when
```

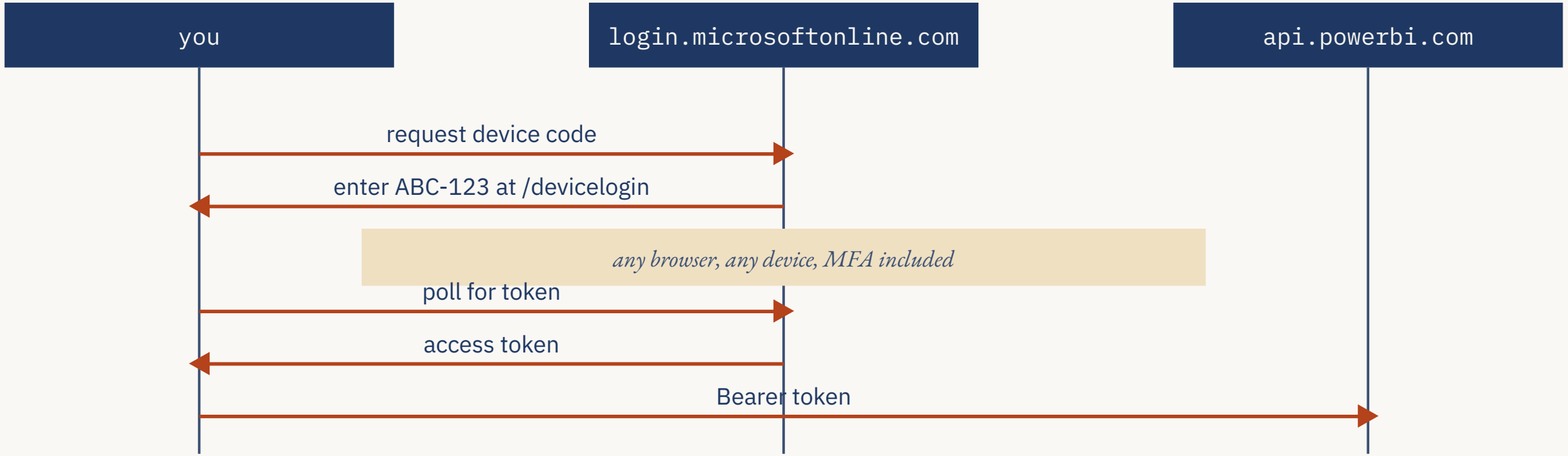
WHY IT MATTERS

Two bonus endpoints join in section 3 for the access question.

Four endpoints is the anchor. Everything else is a join.

How auth actually works

Device-code flow: what Admin Monitoring does for you behind the scenes.



Device flow exists because terminals can't pop a browser. The browser leg, and the MFA, is yours.

The five lines that get a token

People photograph this slide. The scope string is the teaching point.

```
app = msal.PublicClientApplication(CLIENT_ID, authority=AUTHORITY)
flow = app.initiate_device_flow(
    scopes=["https://analysis.windows.net/powerbi/api/.default"])
print(flow["message"])   # microsoft.com/devicelogin, enter code
token = app.acquire_token_by_device_flow(flow)["access_token"]
```

WHY IT MATTERS

analysis.windows.net/powerbi/api = the Power BI service itself, not Graph, not Azure.
.default = every permission already consented on this app.

Wrong resource is the #1 silent auth failure: token acquires fine, API returns 401.

The device code, live

No password in the script. The browser leg is yours.

```
zsh · triage_scanner.py

$ python triage_scanner.py --days 28
To sign in, open a web browser to
  https://microsoft.com/devicelogin
and enter the code  DR7K2QHTB
(waiting for you to authenticate...)
token acquired. Pulling inventory...
```

WHY THIS IS THE RIGHT DEFAULT

1. **No secret in the script.**
   The app never holds a password. You sign in, in your own browser.
2. **MFA happens where it should.**
   Your normal sign-in, conditional access and all.
3. **Headless-ready on Monday.**
   Swap to a service principal plus Key Vault and the prompt disappears.

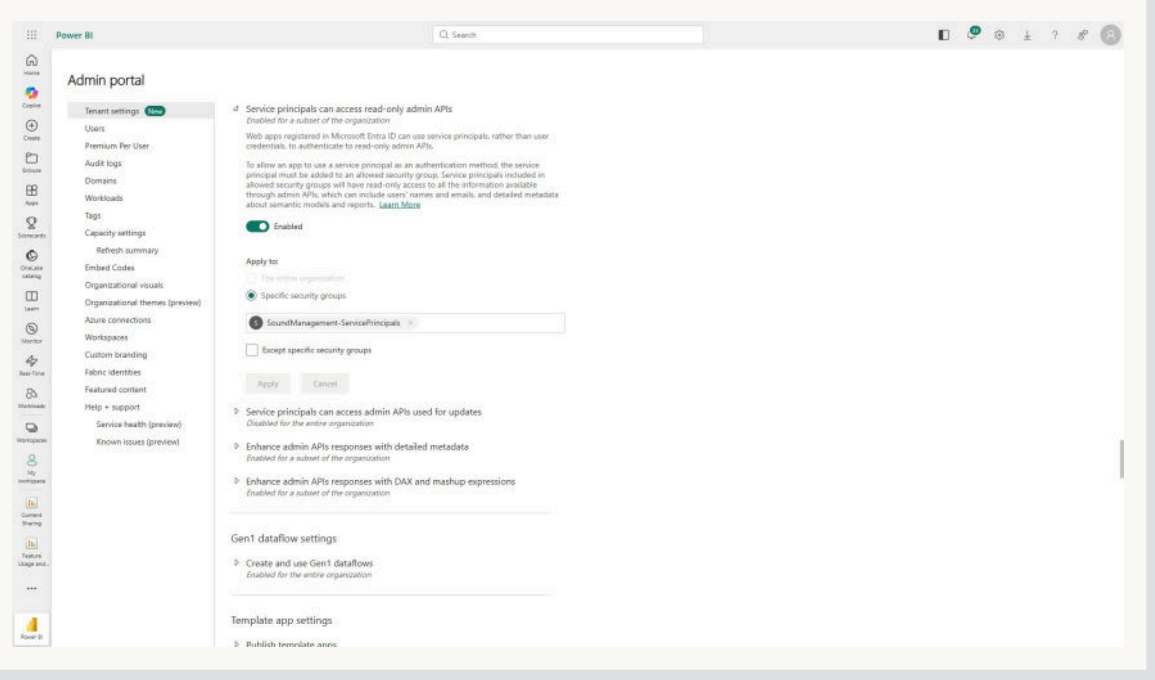
> *Device code today, service principal Monday. Same four endpoints either way.*

Production auth: not your account

	DEVICE CODE (TODAY)	SERVICE PRINCIPAL (MONDAY)
IDENTITY	You	An app registration
SURVIVES YOU LEAVING	No	Yes
MFA PROMPTS	Every run	Never
SECRET MANAGEMENT	None	Key Vault

The tenant setting that unlocks it

An app registration with NO API permissions, plus this one tenant setting.



WHY THIS IS SAFE

1 The setting IS the grant.

The read-only admin API toggle, scoped to a security group.

2 No admin role on the app.

The registration holds zero API permissions.

3 Revocable in one click.

Drop the app from the group; access is gone.

App registration + security group + this setting. The service principal never gets an admin role; the setting is the grant.

The run starts

Real terminal output, my tenant, captured live.

```
zsh · triage_scanner.py

$ python triage_scanner.py --days 7
Pulling inventory...
  49  workspaces
  48  reports
  40  datasets
  4  refreshables
Pulling 7 days of activity (continuation-token paged)...
```

WHAT YOU'RE LOOKING AT

1. Real output, my tenant.
   Not a mock. The API answered 90 seconds ago.
2. 49 here. Yours says 847.
   The code does not care how big the tenant is.
3. One command. Four counts.
   No portal, no clicking, no waiting on a refresh.

> *Every number in this deck is from this run. That sentence is the credibility of the whole talk.*

Three endpoints. One field each.

ENDPOINT	THE FIELD THAT MATTERS	WHY IT MATTERS
/ADMIN/GROUPS	id	The join key every later call resolves against. Personal workspaces fail the owner lookup.
/ADMIN/REPORTS	id (= ReportId)	What activity events join on. 'Modified' is not 'viewed': that gap IS the zero-view list.
/ADMIN/DATASETS	configuredBy	Who owns the refresh. A departed owner's name lingers after they leave.

Three fields are the entire join graph behind all four lists.

Pagination: the part everyone skips

Thirty seconds that save you from silently triaging half your tenant.

```
zsh · triage_scanner.py

Pulling 7 days of activity (continuation-token paged)...
  day -1 (2026-06-11): 18053 events
  day -2 (2026-06-10): 20897 events
  day -3 (2026-06-09): 23277 events
  day -4 (2026-06-08): 24746 events
  day -5 (2026-06-07): 25383 events
  day -6 (2026-06-06): 25659 events
  day -7 (2026-06-05): 25968 events
```

WHY THIS MATTERS

1. Two paging schemes, one tenant.

   $top/$skip for inventory, continuation tokens for activity.

2. $top caps at 5000.

   Run it on 12,000 workspaces and you confidently report 5,000.

3. Loop until the token is gone.

   Skip it and you silently triage half the tenant.

> *Silent truncation is worse than an error.*

Autopsy: /admin/activityevents

The talk's data engine, and the weirdest endpoint of the four.

```
GET .../admin/activityevents
      ?startDateTime='2026-06-09T00:00:00'
      &endDateTime='2026-06-09T23:59:59'
```

WHY IT MATTERS

start and end MUST be the same UTC day.

History reaches back ~28 days, no further. 5-10k events per page plus a continuation token.

The next two slides are its sharp edges.

The continuation-token loop

The loop the docs don't draw.

```
for each UTC day (max ~28 back):
    GET /activityevents?startDateTime=...&endDateTime=...
    while continuationToken in response:
        GET /activityevents?continuationToken='...'
```

WHY IT MATTERS

Drop the inner loop and you silently lose events.

A report someone opened 40 times becomes 'zero-view.' You delete it. They notice.

False positives in a deletion list are how governance projects die.

The throttle budget

200 requests per hour on direct admin REST calls. My real run's budget:

```
inventory pulls:            4 calls
27 days x ~1–2 pages:     ~35 calls
owner joins (cached):     ~30 calls
                          ----------
                          ~70 calls   (35% of one hour)
```

WHY IT MATTERS

One run fits easily. Three full debug runs in an hour and you're throttled.

Develop against ONE day. Run full scans once.

What an event actually looks like

Real event from the real run. Somewhere in this log, the scanner is watching itself.

zsh · triage_scanner.py

```
sample event: {
  "CreationTime": "2026-06-09T00:00:24",
  "Operation": "GetWorkspacesViaAdminApi",
  "Workload": "PowerBI",
  "ResultStatus": "Succeeded",
  "Activity": "GetWorkspacesViaAdminApi",
  "WorkspaceId": "00000000-..."
}
```

WHAT IT TEACHES

1. The scanner watches itself.

   Its own GetWorkspaces call shows up here. The audit API audits its callers.

2. PascalCase, not the docs.

   Activity, WorkspaceId, UserId. Casing the docs get wrong.

3. Verify against real data.

   This is why the scanner prints the raw event on stage.

> *Check field names against real responses, never the documentation.*

Docs vs reality

I check field names against real data, not documentation.

THE DOCS IMPLY	THE TENANT RETURNS
01 camelCase fields	01 PascalCase: Activity, ReportId, UserId, WorkspaceId
02 Workspace names on events	02 IDs only - the name needs a join to /groups
03 Uniform event shape	03 Shape varies by Activity type - code defensively with .get()

vs.

A diff table from Tuesday's run beats a live print statement.

When I rehearsed this, it crashed.

The real traceback, captured tonight. Day -28 is past the retention window: the API returns 400, not an empty list.

```
  day -27 (2026-05-16): 31325 cumulative events
requests.exceptions.HTTPError: 400 Client Error: Bad Request
  .../activityevents?startDateTime='2026-05-15T00:00:00'...
```

WHY IT MATTERS

The original build crashed five minutes in, BEFORE writing a single CSV.

A live demo proves the happy path once. A traceback proves you walked the unhappy path.

The fix: boundaries are not errors

Two lessons in one except block.

```
except requests.HTTPError as e:
    if e.response is not None and e.response.status_code == 400:
        print(f"  day -{d}: outside retention window - "
              f"stopping with {len(events)} events")
        break    # keep everything collected so far
    raise        # anything else is a real error
```

WHY IT MATTERS

The retention window is a boundary, not a failure.

Never catch broadly, or you mask the real 401s and 500s.

Expected boundaries get handled. Real errors still raise.

CHAPTER THREE

THE FOUR LISTS

Zero-view, orphaned, dead refresh, exposed.

03

Four lists. One run.

39

Zero-view reports

No views in 27 days.

4

Orphaned datasets

No report, or no owner.

0

Dead refreshes

Schedules all healthy.

4

Exposed

Org-wide or public.

Every count is a CSV with names and emails. A list is an action; a number is not.

The join that matters

Four responses, one triage engine. This is the slide people photograph.

```
/groups ----------+
                  | names, state, capacity
/reports ---------+ → anti-join on viewed ReportIds → ZERO-VIEW
                  |
/activityevents --+   (ViewReport events, token-paged)
                  |
/datasets --------+ → no report on top OR no configuredBy → ORPHANED
                  |
/refreshables ----+ → lastRefresh.status = Failed → DEAD REFRESH
```

The 12 lines

Everything before this slide was plumbing. The analysis is two expressions.

```
viewed = {e["ReportId"] for e in events
          if e.get("Activity") == "ViewReport"}

zero_view = [r for r in reports if r["id"] not in viewed]
```

WHY IT MATTERS

This is the Admin Monitoring Inventory page, built from scratch.
An anti-join on a set.

The full run

31,325 events. 28 days in one screenshot, including the graceful 400 stop you just learned about.

```
  day -1  (2026-06-11): 18053 cumulative events
  ...
  day -27 (2026-05-16): 31325 cumulative events
  day -28 (2026-05-15): outside retention window (400) - stopping

=== TRIAGE SUMMARY ===
Zero-view reports:          39
Orphaned datasets:           4
Dead refresh schedules:      0
```

Output 1: Zero-view reports - 39 of 48

REPORT	WORKSPACE	MODIFIED	ADMIN_EMAIL
TEST	Surgio Test	2021-08-05	j*****@...
COVID DASHBOARD V0.1	PersonalWorkspace A.K.	2021-11-05	a****@...
...44 MORE ROWS...			

Covid Dashboard v0.1

Last modified November 2021. Views in the activity window: zero. This was URGENT once. Someone built it on a weekend. Nobody ever decommissioned it.

Urgency creates content. Nothing destroys it. That asymmetry is why tenants only grow.

Output 2: Orphaned datasets

The criteria

No reports exist on top, OR configuredBy is blank (the owner left).

1

My tenant: 4 real orphans

On a 40-dataset tenant. Scale that ratio to the 847-workspace tenant in your head.

2

The gotchas the docs won't give you

DataflowsStaging* are SYSTEM items

Fabric's own staging lakehouse/warehouse show up as 'no reports' orphans. Allowlist them, or your cleanup list tells you to delete Fabric itself.

1

Personal workspaces fail the owner lookup

GetGroupUsersAsAdmin errors on them. Handle '(lookup failed)' instead of crashing.

2

This slide exists because of rehearsal, not docs. It is the difference between a script and a tool.

Output 3: Dead refresh schedules

Refreshables where lastRefresh.status = Failed and no active owner. My tenant returned ZERO - all four refreshables healthy. The shape when it isn't zero:

```
| item             | workspace   | last_status | configured_by             |
| Daily Sales Load | Finance Ops | Failed      | departed.user@contoso.com |
```

WHY IT MATTERS

Zero is the goal state. An empty list on a slide is a teaching moment; a live empty list is dead air. Failed refreshes burn capacity on every retry and alert no one.

"Who has access to what?"

Three lists tell you what's DEAD. The fourth list tells you what's EXPOSED.

List 4a: Who is in your workspaces

Same endpoint as the owner join - the responses are already cached. The fourth list costs almost zero extra API calls.

```
GET .../admin/groups/{id}/users     # per workspace, cached

external = [u for u in users
            if "#EXT#" in (u.get("identifier") or "")
            or not u.get("emailAddress","").endswith("@yourdomain.com")]
```

WHY IT MATTERS

Flag: guests with Admin/Member rights, service principals nobody remembers, workspaces with NO admin at all.

List 4b: Shared with the ENTIRE org

My tenant, live: one hit. A billing report, org-wide - and the right is ReadRESHARE: anyone can pass it on.

```
GET .../admin/widelySharedArtifacts/linksSharedToWholeOrganization

{ "displayName": "Billable Utilization",
  "artifactType": "Report",
  "accessRight": "ReadReshare",
  "shareType": "Link",
  "sharer": { "emailAddress": "a****@..." } }
```

WHY IT MATTERS

The sharer email makes the conversation specific: this person, this report.

Org-wide is the FLOOR, because everyone can re-share.

List 4c: Published to the public internet

Every Publish-to-web embed in the tenant: anyone on the internet, no auth, search-indexed. My tenant, live: TWO hits.

```
GET .../admin/widelySharedArtifacts/publishedToWeb

Covid Dashboard v0.2          PublishToWeb
Covid Dashboard v0.2 mobile   PublishToWeb
```

WHY IT MATTERS

Remember Covid Dashboard v0.1, dead since ~~2021~~? Its successors are on the public internet RIGHT NOW. Nobody told me. I found out from this API call, two days ago.

If this endpoint returns anything you didn't expect, it is today's work, not Monday's.

The fourth list, assembled - 16 seconds of API calls

CHECK	ITEM	SEVERITY
NO ADMIN	DUMMY_ pipeline ghost workspace (Feb 2023)	fix this week
ORG-WIDE SHARE LINK	Billable Utilization	review
PUBLISHED TO WEB	Covid Dashboard v0.2	TODAY
PUBLISHED TO WEB	Covid Dashboard v0.2 mobile	TODAY

Beyond ViewReport: the events worth watching

ACTIVITY	WHAT IT TELLS YOU
EXPORTREPORT	data leaving the building
SHAREREPORT / SHAREDASHBOARD	sprawl, as it happens
DELETEREPORT	your cleanup actually happening
ADDGROUPMEMBERS	access drift, daily
*ASADMIN OPERATIONS	who ELSE is running admin scans

The poor man's SIEM

Filter + webhook. Daily. A monitoring system in eight lines.

```
alerts = [e for e in events
          if e.get("Activity") in WATCHLIST
          and e.get("UserId") not in EXPECTED]
if alerts:
    requests.post(TEAMS_WEBHOOK_URL, json=summary(alerts))
```

WHY IT MATTERS

It's not Sentinel - don't oversell it.

But on day 30 of an inherited tenant, 'I get a Teams ping when someone exports from Finance' is more than the previous admin ever had.

CHAPTER FOUR

MONDAY MORNING

Owners, notify, schedule.

04

The CSV is the deliverable

01	**Who owns these workspaces?**	*A count is a report. A list with names is an action.*
02	**How do you tell them?**	*At scale, without writing 39 emails by hand.*
03	**How does this run without you?**	*Governance that requires you is governance that stops when you're busy.*

Four CSVs: zero_view, orphaned, dead_refresh, access_exposure. Not the terminal output. The files.

Step 1: Find the owners

Every row gets an admin email. Real quirks from the real run included.

```
r = requests.get(f"{BASE}/groups/{workspace_id}/users", headers=headers)
admins = [u for u in r.json()["value"]
          if u["groupUserAccessRight"] == "Admin"]
```

WHY IT MATTERS

Service-principal admins return a GUID identifier, not an email.
Cache lookups (one per workspace, not per row) or you eat the throttle.

Not a count - a list with names attached.

Step 2: Notify at scale

Option A: Teams webhook, ten lines. Option B: Power Automate - CSV to HTTP trigger to email, no code.

```
payload = {"text": f"**Inactive report cleanup needed**\n\n{table_markdown}"}
requests.post(TEAMS_WEBHOOK_URL, json=payload)
```

WHY IT MATTERS

Screenshot of the real posted card replaces the live post: same proof, zero on-stage webhook risk.

Step 3: Schedule it

OPTION	EFFORT	BEST FOR
GITHUB ACTIONS (CRON YAML)	Low	Teams with a repo
FABRIC NOTEBOOK (SCHEDULED)	Medium	Fabric tenants - output to Lakehouse
WINDOWS TASK SCHEDULER	Minimal	Inherited tenant, no capacity yet

Scanner → Lakehouse table → Power BI report → Admin Monitoring, but yours.

CHAPTER FIVE

TRIAGE TO TREND

One run is a snapshot. History is a derivative.

05

One run is triage. Scheduled runs are HISTORY.

What went stale THIS month?

That wasn't stale last month.

1

Is the orphan count trending?

Up or down - is governance winning?

2

Did the notifications WORK?

The only metric that justifies the system.

3

A single scan answers 'what is dead today?' An appended Lakehouse table answers the questions that matter.

The week-over-week diff

The resurrected set is the safety mechanism nobody builds.

```
this_week = set(zero_view_now["report_id"])
last_week = set(zero_view_prev["report_id"])

newly_dead  = this_week - last_week   # just crossed the line
resurrected = last_week - this_week   # someone is using it again
```

WHY IT MATTERS

resurrected = reports that were on last week's deletion list and came BACK to life. Exactly the false positive the 28-day window creates. The diff catches it.

Same pattern as everything else in this talk - left as homework.

The Monday scorecard

Four numbers, every Monday, zero browsers opened.

```
TENANT HEALTH — week of June 8
  Zero-view reports:   39  (down 7)
  Orphaned datasets:    4  (no change)
  Dead refreshes:       0  (no change)
  Public exposures:     2  (Covid v0.2 + mobile — REVOKE)
```

WHY IT MATTERS

This card IS the talk's promise delivered: Admin Monitoring, but yours, in your channel, on your cadence, with your definitions.

When you outgrow the four endpoints

Metadata scanning APIs: built for 10,000+ workspaces. Same auth, same pattern, batch-shaped.

```
POST .../admin/workspaces/getInfo       # request a scan
GET  .../admin/workspaces/scanStatus/{id}
GET  .../admin/workspaces/scanResult/{id}
```

WHY IT MATTERS

Returns items, lineage, endorsements, sensitivity labels.
Incremental: getModifiedWorkspaces since last scan.

The four endpoints are the teaching tool. Scanner APIs are the industrial tool.

The 30-day playbook

WEEK	DO	DELIVERABLE
1	Run the scanner. Allowlist system items. Check publishedToWeb FIRST.	Four CSVs + the exposure answer
2	Owner join. Notify admins of their zero-view lists.	The email trail
3	Second run. Week-over-week diff. Archive candidates with sign-off.	Resurrected-list check
4	SP auth + schedule. Lakehouse history. Monday scorecard live.	Governance that runs without you

CHAPTER SIX

OWN THE CAMERA

Limits, one live command, the code.

06

What this doesn't tell you

01	**Activity history is ~28 days**	*'Zero views' means zero in the window, not ever. Day 29 is an HTTP 400, not an empty day.*
02	**The scanner has no delete button**	*It surfaces lists. A human decides.*
03	**configuredBy is inconsistent**	*Absent on datasets never configured for refresh. Test before acting.*
04	**System items appear as orphans**	*Allowlist DataflowsStaging*.*
05	**Admin APIs need Fabric or Premium**	*And admin rights. This doesn't run on free capacity.*
06	**No API for 'views last year'**	*Usage beyond 28 days needs the scheduled history you just built.*

Be honest with your data. Trust is earned on this slide, not the 50 before it.

Once you own the camera, you decide.

The Admin Monitoring workspace shows you the picture. The APIs are the camera.

[LIVE] One command before you go

No tenant. No auth. No API calls. Three CSVs in two seconds.

```
python triage_scanner.py --demo-data
```

WHY IT MATTERS

You can do this TONIGHT on the train with zero credentials.
Point it at your tenant Monday with two config lines.

Get the code

The command you just watched is the first line.

```
github.com/sqllocks/triage-scanner

pip install msal requests pandas
python triage_scanner.py --demo-data   # tonight, on the train
python triage_scanner.py --days 7     # Monday, your tenant
```

WHY IT MATTERS

Questions → sqllocks@sqlbites.com - SQLBites.net

Questions? Let's Talk.

Q.01

Q

Your Inherited Tenant

Q.02

Q

Service Principal Setup

Q.03

Q

The Access Sweep

Q.04

Q

Scheduling + Lakehouse History