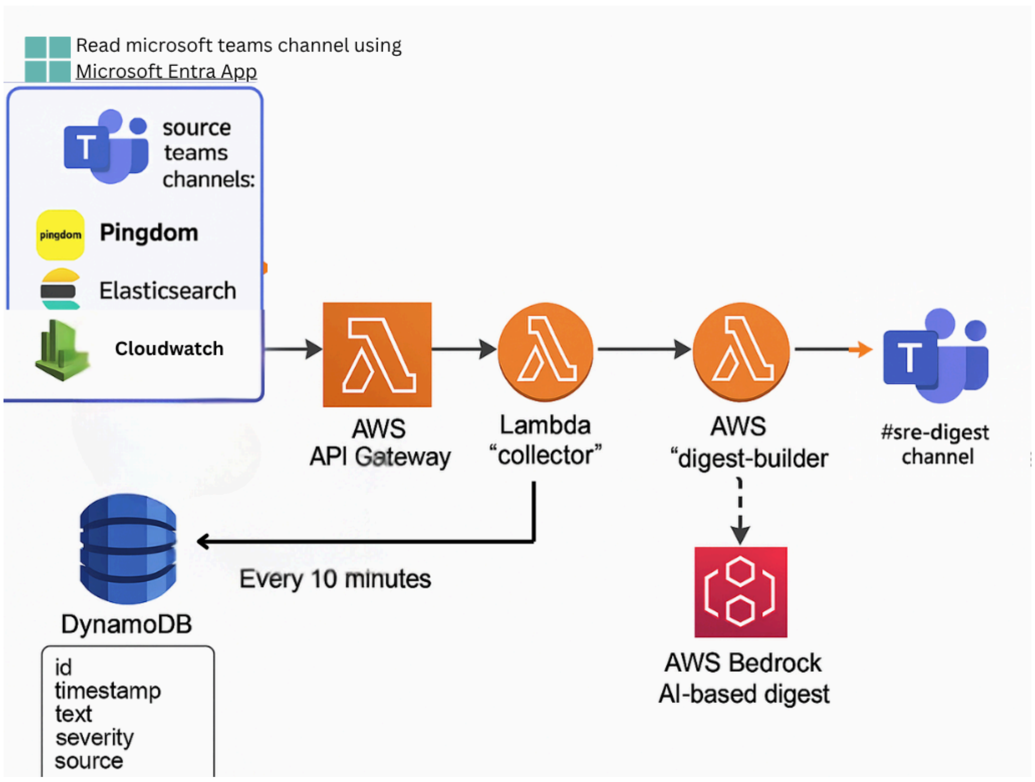
SRE Teams Digest



We will build a **Teams-first, config-driven SRE digest**. The **only ingestion source is Microsoft Teams channels** (what SREs actually read).

We consume posts via **Microsoft Graph change notifications**, **normalize & dedupe** incidents across **PagerDuty / Elastic / Pingdom**, summarize every **10 minutes** with **Bedrock**, and post Markdown to **#sre-digest**. **Phase-1 scope = GovMeetings incidents only**.

The design is reusable for other products with configuration only.

> Access requirement: We must have access to a Microsoft Entra ID tenant (formerly Azure AD) to register an app with Graph application permissions and grant admin consent. If we cannot hold permissions ourselves, IT can host the app in their tenant and give us the client credentials.

- **What channels & message types are in scope?**
  - Phase-1: `alerts_prod` (Pingdom incidents), `production-support` (Elastic/PagerDuty incidents; **exclude CRs**), `mon_prod_crit_govmeetings` (PagerDuty incidents), `mon_prod_crit_Swagit` (Elastic incidents for GovMeetings-Swagit).
  - Default out (can be enabled later): `mon_prod_warn_govmeetings` (warnings), `mon_prod_crit_OneMeeting`, `mon_prod_warn_OneMeeting`, `SRE_Reporting`, `TSE-Support-govMeetings`, CRs and On-call lookup forms.
- **What are the sources and where do we ingest from?**
  - Upstream systems: **PagerDuty, Elastic, Pingdom**.

    - **Ingestion source of truth: Microsoft Teams channels** via Graph change notifications (Teams-first approach). We parse PD/Elastic/Pingdom **as posted in Teams**.
- **Do we use Incoming Webhooks?**
    - **No** for reading. We **only** use **Graph change notifications** to ingest messages from Teams.
    - For posting, prefer **Graph** `ChannelMessage.Send`. Incoming Webhooks are legacy and may be retired.
- **Is Teams-first the only viable design?**
    - Not the only one, but it best matches what SREs see (and covers ICT-owned Pingdom without cross-team work).
    - Later we can add **upstream** webhooks (PD/Pingdom/Elastic) into the same pipeline for lower latency—design is compatible.
- **How do we include only GovMeetings incidents and avoid noise?**
    - Channel scoping + content filters (PD service, Elastic rule group, Pingdom check names).
    - Phase-1 excludes **warnings** and **CRs**.
    - 10-min batching + dedupe reduces notification volume.
- **How do we dedupe across multiple paths (PD→Teams, Elastic→PD→Teams)?**
    - **IncidentKey** = PD Incident ID | Elastic Rule ID+Instance | Pingdom Check ID; fallback to hash(title, links, ±60s).
    - Maintain state OPEN→ACK→RESOLVED within the digest window.
- **How will other products (GoAccess/GoService) onboard?**
    - **Config-driven** mapping: product → channels → filters → parser set. **No code change**.
- **What access is required to start?**
    - A **Microsoft Entra ID (Azure AD)** tenant where we can: register an app, create a client secret, and grant **admin consent** for `ChannelMessage.Read.All` and `ChannelMessage.Send`.
    - If direct access is not possible, an IT/Global Admin can **create the app on our behalf** and share the **tenant ID, client ID, client secret**.

Phase plan (with best practices)

Phase	Action	Best-practice details
1 – Entra app	Register **Microsoft Entra app**(application permissions)	`ChannelMessage.Read.All` (read), `ChannelMessage.Send` (post), optional `Team.ReadBasic.All` for metadata. Use client-credentials; store secret in **AWS Secrets Manager**.
2 – Change notifications	Create **Graph subscriptions** for each channel	`POST /v1.0/subscriptions` with `resource=/teams/{tid}/channels/{cid}/messages`, `notificationUrl` → AWS. Set `expirationDateTime` to max allowed (~**3 days**); auto-renew daily via EventBridge.
3 – Webhook endpoint	**API Gateway + Lambda (collector)**	On validation, immediately echo `validationToken` and **return 200 within 5s**. On payloads, normalize and push to **DynamoDB** (or SQS) asynchronously.
4 – Storage & dedupe	**DynamoDB** with composite keys	Keys: `messageId` and `incidentKey`. Columns: `source, product, severity, status, service, env, links[], notables[], receivedAt`. TTL: **24–48h**.
5 – Digest builder	**EventBridge (*/10)→ Lambda**	Query last 10 min, group by product/severity, dedupe by `incidentKey`, call **Bedrock** with structured prompt, post to **#sre-digest** using Graph send. Thread updates within the same digest window.

Phase	Action	Best-practice details
6 – Observability	CW Logs, Metrics, Alarms, X-Ray	Alarms on webhook 5xx >1%, Graph 429/401 spikes, subscription renew failures, posting errors, Bedrock timeouts. DLQ for failed posts.

Channel inventory (from screenshots)

Channel	Product focus	Source(s)	Include Phase-1?	Notes
`alerts_prod`	Mixed Ops	**Pingdom**	**Yes (incidents)**	Host/URL/Description/timestamp; Check URL to my.pingdom.com.
`production-support`	All products	**Elastic Workflows**, some **PagerDuty**, **CRs**	**Yes (incidents)**	Ignore **CR-xxxxx**; focus `INCIDENT` /PD cards.
`mon_prod_crit_govmeetings`	GovMeetings	**PagerDuty**	**Yes**	PD card: Incident ID, Service, Assignee, Status.
`mon_prod_crit_Swagit`	GovMeetings (Swagit)	**Elastic**	**Yes**	Wowza VOD HTTP 500 errors.
`mon_prod_warn_govmeetings`	GovMeetings	**Elastic**	**Default: No**	Warnings (storage %, "No host data"); enable only if requested.
`mon_prod_crit_OneMeeting`	OneMeeting	**PagerDuty**	No	For extensibility testing later.
`mon_prod_warn_OneMeeting`	OneMeeting	**Elastic**	No	Includes "Recovered" messages (state updates).

Common summary schema (Bedrock prompt contract)

```
{
  "source": "pagerduty|elastic|pingdom",
  "product": "GovMeetings",
  "service": "string",
  "environment": "prod|stage|...",
  "severity": "sev1|sev2|sev3|crit|warn",
  "status": "open|ack|resolved",
  "title": "string",
  "short_description": "string",
  "timestamp_original": "iso8601",
  "timestamp_teams": "iso8601",
  "incident_key": "pd#|elastic#|pingdom#",
  "links": ["url"],
  "notables": ["host=...", "threshold=...", "observed=..."]
}
```

Acceptance criteria (Phase-1)

- Teams-first ingestion via **Graph change notifications**; **no** Teams Incoming Webhook ingestion.
- Only **GovMeetings incidents** appear in the digest; **warnings/CRs excluded**.
- Digest includes **source, severity, product, service, env, timestamps, IDs, links, status, assignee (if available)**.
- **Dedupe** across PD/Elastic/Pingdom; state updates reflected once within window.
- **Config-only** onboarding for one additional product (smoke test with OneMeeting).
- Post to **#sre-digest** using **Graph** `ChannelMessage.Send`.

Rollout plan & timeline

Week 1: Entra app, Secrets, collector webhook, subscriptions for 3 channels, Dynamo schema.

Week 2: Digest Lambda, Bedrock prompt, posting to #sre-digest, alarms & dashboards, UAT with real traffic.

Go/No-Go: 2-day soak; then enable Swagit channel and (optionally) warnings if approved.

Risks & mitigations

- **Graph throttling (429)/latency** → retry with backoff; async processing; alarm on spikes.
- **Legacy webhook posts in channels** → parser handles Workflows + older formats; track migration.
- **Duplicate paths (PD→Teams vs Elastic→PD→Teams)** → IncidentKey + hash fallback.
- **Subscription expiry** → EventBridge renew job; alert on renewal failures.
- **Noisy Production-Support** → strict GovMeetings filters + incident-only rule.

Open items for manager confirmation

1. Include **warnings** (`mon_prod_warn_govmeetings`) in Phase-1 or keep **incidents only**?
2. Proceed with **Graph Send** for posting (vs. incoming webhooks)?
3. Any additional channels to include now?

Appendix – Implementation notes

- **Naming:** Microsoft **Entra ID** is the new name for **Azure Active Directory (Azure AD)**.
- **IDs:** Use Teams “Get link to channel” to extract `teamId` and `channelId`.
- **Subscriptions:** Max TTL ~**3 days**; store active subscription IDs and auto-renew daily.
- **Validation:** Echo `validationToken` immediately; heavy work async via SQS/Dynamo.
- **IAM:** Lambdas limited to required actions (Dynamo CRUD, EventBridge rule, Bedrock `InvokeModel/InvokeAgent`, Secrets read).
- **Data retention:** Dynamo TTL 24–48h; CloudWatch log retention 30–90 days.
- **Testing:** Use a sandbox Team/channel to replay sample payloads and verify dedupe and formatting before prod cutover.