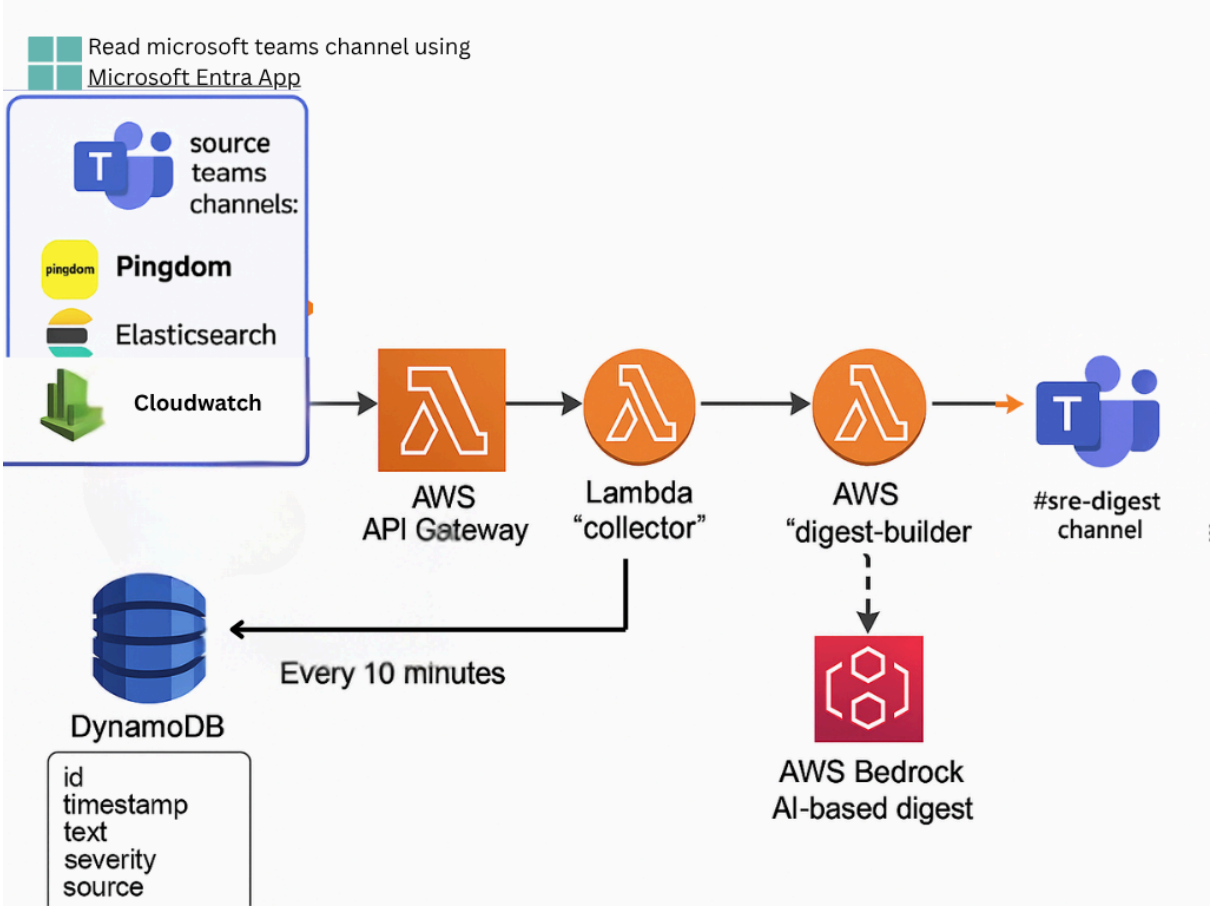
Summary for SRE-Collector Copilot Digest



Phase	Action	Best-practice details
1 – App registration	Create an **Azure AD app**.Grant *application* permissions: `ChannelMessage.Read.All` (read) + `ChannelMessage.Send` (post digest)	Application-only tokens avoid tying the service to any individual user login.
2 – Create subscription	POST to `https://graph.microsoft.com/v1.0/subscriptions` with: `"resource": "/teams/{tid}/channels/{cid}/messages","notificationUrl": "https://api.example.com/graph/webhook","expirationDateTime": {now+3 days}`	Whatever pingdom, cloudwatch alerts and elastic/kibana posts to certain microsoft teams channel such as *#alerts_prod, #production-support, mon_prod_crit_govmeetings, .. etc*

3 – Webhook endpoint	API Gateway + Lambda: • On **validation** call, echo `validationToken`.• On normal payload, parse JSON, push each alert into Dynamo or SQS.	Return *200 OK within 5 s*; Graph drops the call otherwise.
4 – Storage + dedup	DynamoDB table keyed by message ID; TTL = 24 h.	Makes the 10-minute batch Lambda idempotent and supports "seen/in-digest" flags.
5 – Digest Lambda	Triggered every 10 min by EventBridge.• Scan for items `timestamp > now-10min`.• Group & dedup.• Call **Bedrock Agent** (prompt defined earlier).• Post result to **#sre-digest** via incoming-webhook URL.	Keep the webhook URL in AWS Secrets Manager; never hard-code.
6 – Observability	CloudWatch Logs for both Lambdas; create alarm on >1% error rate.Enable X-Ray tracing if latency debugging is needed.	Helps catch Graph throttling (HTTP 429) or posting failures quickly.

TL;DR

1. *Subscribe* to [microsoft teams channels which will have pingdom alerts, elastic and cloudwatch alarms posted] `#alerts_prod, #production-support, mon_prod_crit_govmeetings, .. etc` such as #prod-crit-alerts via Graph change notifications.
2. *Collect* each new message in Dynamo (push).
3. *Batch* every 10 min, summarise with Bedrock, and *post* a tidy Markdown digest back to Teams in ***#SRE-DIGEST***

Channels we will possibly monitor: x***#alerts_prod, #production-support, mon_prod_crit_govmeetings, .. etc***
this will give us information about : message, product info etc for the easy processing of digest.

> 💡 The SRE team now watches one digest channel, refreshed every 10 minutes, with perfectly formatted, deduplicated summaries produced by an LLM—all without manual scrolling through multiple Teams channels or losing alerts in the noise. The pipeline is event-driven, auditable, and extensible to any future alert source that can publish to SNS or Teams.

Design choice :

Original requirement	Key design choices & why they satisfy it
1. "Monitor the Teams `#alerts_prod, #production-support, mon_prod_crit_govmeetings, .. etc` channel (and other alert feeds) automatically."	• We subscribe to `.../teams/{team-id}/channels/{channel-id}/messages` via Microsoft Graph **change-notifications**.• Graph *pushes* every new message to our AWS webhook, so nothing is missed and no polling gaps occur.• Optional webhooks/SNS from CloudWatch, Pingdom, Elastic can be stored in the same DynamoDB table—one unified stream.
2. "Deduplicate and summarise unique incidents every 10 min."	• The **collector Lambda** writes each message with its ID + timestamp to DynamoDB, giving us state to spot duplicates.• An **EventBridge-scheduled "digest-builder" Lambda** runs exactly every 10 minutes, loads only the last 10 min of events, groups & dedups them, then invokes the Bedrock Agent with the structured prompt we created.
3. "Post a concise Markdown digest to *#sre-digest*."	• The Bedrock Agent returns a ready-to-paste Markdown table.• The digest-builder Lambda uses a simple **incoming-webhook** (or Graph `ChannelMessage.Send`) to post that Markdown into **#sre-digest**, so every SRE sees one clean summary instead of dozens of raw alerts.
4. "Reduce noise / alert fatigue."	• Only alerts from the last window are analysed; duplicates (same text & source) are collapsed.• Severity grouping + bullet-style language make the table skimmable in seconds.• By batching into a single message every 10 min, phone notifications drop from *N* per alert to just six per hour.
5. "Leverage AWS Bedrock's LLMs."	• The summarisation prompt is executed via **bedrock:InvokeModel / InvokeAgent** within the digest-builder Lambda, letting us swap models (Claude, Titan) without code changes.• All sensitive tokens (Teams secret, Bedrock invoke rights) are in Secrets Manager and IAM roles, following AWS and Microsoft least-privilege best practices.

Phase 1 : Entra part

We need microsoft entra app from Azure portal using microsoft entra id for registering an entra app, **why is this required?**
because we are using the microsoft graph we want to monitor the messages that are sent from Pingdom, Cloudwatch alerts and Elastic for different products, these messages are sent to microsoft teams channel and we want the ability to read them and store them on AWS DyanmoDB for example for our digest to process this data.

Channels we will possibly monitor: x***#alerts_prod, #production-support, mon_prod_crit_govmeetings, .. etc***
this will give us information about : message, product info etc for the easy processing of digest.

1. Sign in to **Azure portal → Microsoft Entra ID → App registrations → New registration.**• Name: `SREReportingCopilotDigestCollector` • Supported account type: *Single tenant*• Redirect URI: *(leave blank –*

we'll use client-credentials)
2. Certificates & secrets → New client secret → copy the value; store it in AWS Secrets Manager.
3. API permissions → Add permission → Microsoft Graph → Application permissions → search and add `ChannelMessage.Read.All` (least-privilege for channel-message subscriptions). (Microsoft Learn)
4. Click **Grant admin consent** (Global Admin action).
5. Note the **Directory (tenant) ID** and **Application (client) ID** – your Lambda will need them to obtain a token.

Phase 2: Create the change-notification subscription

> We'll subscribe to /teams/{team-id}/channels/{channel-id}/messages so every new alert in #warn-critical is pushed to AWS.
>
> ***#alerts_prod, #production-support, mon_prod_crit_govmeetings, .. etc***

3-A. Gather IDs

1. In Teams UI › … next to the channel › **Get link to channel** → the URL contains
   `https://teams.microsoft.com/l/channel/{channel-id}/{name}?groupId={team-id}&tenantId=...`
   Record **team-id** and **channel-id**.

Phase 3: Putting it together in AWS

1. **API Gateway + Lambda "collector"** → validates token, writes each notification payload to DynamoDB/SQS.
2. **EventBridge schedule (every 10 min)** triggers **Lambda "digest-builder"** → reads last 10 min, calls Bedrock Agent with your prompt, posts digest to #sre-digest via incoming-webhook.

Sample digest output expectation:

🚨 SRE Digest Summary (Auto-updated every 15 min)

Timestamp: 2025-07-31 12:15 UTC

Source Systems: CloudWatch, Pingdom, Elastic

Teams Channels Monitored: production_support, warn, critical, alerts_prod

Products Tracked: 15

Deduplicated Alerts/Incidents: ✅

📦 Product 1 – UserAPI

🔥 Incident RUN-101

Severity: 1

Message: High latency observed in user authentication endpoints

Source: CloudWatch → production_support

Status: In Progress

Occurrences: 3

⚠️ Alert RUN-102

Message: CPU usage > 80% on user-api-prod-node-2

Source: Pingdom → warn

Status: In Progress

Occurrences: 1

📦 Product 2 – BillingService

🔥 Incident RUN-110

Severity: 2

Message: Payment gateway timeouts for 40% of transactions

Source: Elastic → production_support

Status: In Progress

Occurrences: 2

⚠️ Alert RUN-111

Message: /billing/summary API error rate > 5%

Source: CloudWatch → alerts_prod

Status: Resolved

Occurrences: 1

📦 Product 4 – NotificationService

⚠️ Alert RUN-118

Message: Email queue backlog > 10K

Source: Elastic → critical

Status: In Progress

Occurrences: 2

📦 Product 7 – DataWarehouse

🔥 Incident RUN-127

Severity: 3

Message: Nightly ETL job failed on dw-job-node-4

Source: CloudWatch → production_support

Status: In Progress

Occurrences: 1

📦 Product 9 – AnalyticsDashboard

⚠️ Alert RUN-133

Message: Dashboard load time > 10s for 40% of users

Source: Pingdom → warn

Status: In Progress

Occurrences: 4