ZERO TO HERO · ARCHITECTS & DATA ENGINEERS · TECHNICAL WORKSHOP

28 SLIDES · SOURCED, NOT GUESSED

Microsoft GraphRAG. Use cases × Azure build × Governance.

Vector RAG retrieves chunks. GraphRAG reasons over relationships. This workshop covers when that difference is worth the cost — and exactly which Azure services you need to run it under real enterprise controls.

```
$ pip install graphrag && graphrag init --root ./ragtest
> # then read slide 02 before you run a single index job
```

Every number in this deck is cited to a primary Microsoft source.
microsoft.github.io/graphrag · learn.microsoft.com · microsoft.com/research

SLIDE 02 · READ THIS BEFORE YOU BUDGET A PROJECT

VERBATIM FROM THE REPO

The GraphRAG library is in maintenance mode. That is not a reason to skip it — it's a reason to architect differently.

GITHUB.COM/MICROSOFT/GRAPHRAG · README WARNING

"GraphRAG is a research project that explores the functional use of graphs to form a targeted context for question answering. Since our first release in July 2024 the capabilities of frontier models have changed dramatically... **This project is largely in maintenance mode, and won't be accepting new PRs or implementing new features.** *We'll perform bug fixes and dependency updates as appropriate, particularly to address CVEs as they arise."*

Also stated: "the provided code serves as a demonstration and is not an officially supported Microsoft offering."

WHAT THIS CHANGES FOR YOU

01 · Treat the library as a technique, not a platform.
The *method* — entity extraction, Leiden communities, community summaries — is the durable asset. The Python package is a reference implementation.

02 · No enterprise SLA, no roadmap, no new features.
Do not put an unsupported research package on the critical path of a regulated production system without owning that risk explicitly.

03 · The supported path moved to first-party services.
Azure AI Search agentic retrieval, Graph in Microsoft Fabric (GA), and Cosmos DB knowledge graphs now carry the enterprise contract. Slides 08–11 compare all three against the OSS route.

Sources: github.com/microsoft/graphrag README (maintenance-mode warning + "not an officially supported Microsoft offering") · microsoft.github.io/graphrag · microsoft.com/research/project/graphrag.

LOCAL VS GLOBAL QUESTIONS

Baseline RAG returns isolated chunks. It cannot connect the dots and it cannot summarise a whole corpus.

STAY ON VECTOR RAG WHEN…

The answer sits in one chunk.

```
Q: What wattage does this product require?
Q: What was revenue for FY24?
Q: How do I sign up for this service?
Q: What is the SLA on tier 3 support?
```

Why vector is correct here: a single top-*k* similarity hit contains the whole answer. Graph construction adds cost and latency for zero retrieval gain.

Cost anchor: vectorising the full text of *The Wizard of Oz* cost **$0.0056** — negligible next to graph build cost.

GO GRAPHRAG WHEN…

The answer must be synthesised.

```
Q: What are the key themes of this report?
Q: List every product my API gateway supports.
Q: Which contractors worked at this site, and
    what certification level does each hold?
```

Two failure classes GraphRAG fixes:
(a) Connecting the dots — traversing disparate facts via shared attributes to reach a new synthesised insight.
(b) Holistic summarisation — understanding semantic themes across a large collection, or one very large document.

Sources: microsoft.github.io/graphrag ("GraphRAG vs Baseline RAG" — both failure classes quoted) · techcommunity.microsoft.com "GraphRAG Costs Explained" (example questions + $0.0056 vectorisation cost).

SLIDE 04 · WHERE ORGANISATIONS ACTUALLY DEPLOY IT

PICK YOUR PILOT FROM HERE

Five patterns with a real graph payoff. Each has a "why vector fails" you can defend in a design review.

USE CASE 01

Contract & obligation web.

Question: "Which suppliers have an uncapped indemnity AND a 2027 renewal?"

Why vector fails: the answer exists in no single chunk — it requires joining clauses across hundreds of separate agreements.

USE CASE 02

Incident & root-cause memory.

Question: "What recurring themes link our Sev-1s over 18 months?"

Why vector fails: this is a corpus-level summarisation question. Community summaries answer it; top-*k* chunks cannot.

USE CASE 03

Codebase deep research.

Question: "What breaks if we change this interface?"

Why vector fails: call graphs are literally graphs. Reported in production by CodeAlive for codebase context engineering.

USE CASE 04

Fraud & AML networks.

Question: "Which counterparties share addresses, directors, or devices?"

Why vector fails: ring detection is multi-hop traversal. Microsoft cites security as a customer-success area.

USE CASE 05

R&D and drug discovery.

Question: "Which compounds share a mechanism with this target?"

Why vector fails: requires entity-to-entity reasoning across papers. Microsoft names drug discovery and news analysis as proven areas.

THE DISQUALIFIER

Don't graph an FAQ.

If 90% of your traffic is single-fact lookup, GraphRAG is the wrong spend.

Test to run in week 1: take 50 real user questions. Count how many need >1 document to answer. Under 20% → stay on vector + semantic ranker.

Sources: github.com/microsoft/graphrag/discussions/322 (Microsoft: "customers successful in news analysis, drug discovery, security") · reddit.com/r/Rag CodeAlive production report · techcommunity.microsoft.com GraphRAG cost article (contractor-certification example).

WHERE 95% OF COST LIVES

Four stages. Every one is an LLM call per chunk — which is exactly why indexing dominates the bill.

STAGE 1 · CHUNK

TextUnits.

`default chunk = 1,200 tokens`

Slice the corpus into analysable units. TextUnits are also the citation anchor — every claim traces back to one.

The lever that matters most: chunk size. Microsoft reports that dropping from **1,200 → 600** lets GPT-4o-mini reach performance similar to GPT-4-Turbo at a fraction of the cost.

Smaller chunks: better extraction recall, more LLM calls.
Larger chunks: fewer calls, more missed entities.

Benchmark this on a sample before you size the project.

STAGE 2 · EXTRACT

Nodes & edges.

`few-shot prompt, per chunk`

The LLM returns, per chunk:

Entities — `entity_name`, `entity_type`, `entity_description`.

Relationships — `source`, `target`, description, plus a numeric `relationship_strength`.

Claims — key assertions with source references.

This is the expensive stage. Microsoft: "these LLM calls are what drives the majority of the cost."

Tune these prompts — see the Prompt Tuning Guide. Out-of-the-box results are explicitly not optimal.

STAGE 3 · CLUSTER

Leiden communities.

`hierarchical, no LLM needed`

Hierarchical clustering of the graph using the Leiden technique (arXiv 1810.08473).

Produces a **community hierarchy** — nested groups of related entities at multiple resolutions.

Why it matters: the hierarchy is what makes corpus-wide questions tractable. You query summaries at the right zoom level instead of reading everything.

Good news: this stage is graph algorithms, not inference. Cheap and deterministic.

STAGE 4 · SUMMARISE

Community reports.

`bottom-up, LLM per community`

Generate a summary for every community and its constituents, **bottom-up** through the hierarchy.

These summaries are the artefact Global Search reads at query time.

Second-largest cost centre. Every community at every level gets an LLM pass.

Architectural consequence: summaries are precomputed, so they go *stale* when documents change. Slide 26 covers the re-indexing strategy.

Sources: microsoft.github.io/graphrag ("The GraphRAG Process" — Index steps + Leiden arXiv 1810.08473) · techcommunity.microsoft.com "GraphRAG Costs Explained" (extraction prompt, 1,200 default chunk, 600-chunk finding, "majority of the cost").

GLOBAL · LOCAL · DRIFT · BASIC

Four retrieval modes ship in the library. Choosing wrong costs 10× and answers worse.

MODE 01 · GLOBAL SEARCH

Whole-corpus themes.

Reads: community summaries.

Answers: "What are the main themes?" "List all X across the corpus."

Mechanism: map-reduce over community reports at a chosen hierarchy level.

Cost profile: most expensive per query — it fans out across many summaries.

Latency (MSFT test): ~20–24s, with ~10–15s retrieval before streaming can begin.

MODE 02 · LOCAL SEARCH

Entity neighbourhood.

Reads: a specific entity, then fans out to neighbours and associated concepts.

Answers: "Tell me everything about supplier X and who it connects to."

Mechanism: anchor on entities matched from the question, then traverse.

Cost profile: far cheaper than Global — bounded fan-out.

Most enterprise traffic lands here. Default to Local; escalate to Global.

MODE 03 · DRIFT SEARCH

Local + community context.

Reads: entity neighbourhood *plus* the community information around it.

Answers: entity questions where the surrounding theme changes the answer.

Mechanism: same fan-out as Local, but enriched with community context — a middle ground between Local and Global.

When to use: Local returns technically-correct but contextually-thin answers.

MODE 04 · BASIC SEARCH

Plain vector top-*k*.

Reads: the vector index. No graph traversal at all.

Answers: the single-fact lookups from slide 03.

Why it's in the library: Microsoft's own docs say to use it "for those times when your query is best answered by baseline RAG."

Design implication: you need a **router** in front of these four modes. Sending every question to Global Search is the #1 way teams burn budget on GraphRAG.

Sources: microsoft.github.io/graphrag ("Query" section — Global / Local / DRIFT / Basic definitions verbatim) · techcommunity.microsoft.com "GraphRAG Costs Explained" (20–24s query latency, 10–15s retrieval before streaming).

MSFT RESEARCH FIGURES

LazyGraphRAG defers the expensive work to query time — indexing drops to vector-RAG economics.

THE MECHANISM · WHY IT'S "LAZY"

No up-front summarisation.

```
Full GraphRAG   index: extract + cluster + summarise ALL
                query: read precomputed summaries

LazyGraphRAG    index: lightweight graph only (NLP-based)
                query: summarise on demand, best-first
```

What Microsoft Research states: LazyGraphRAG "needs no prior summarization of source data, avoiding prohibitive up-front indexing costs" and is "inherently scalable in cost and quality."

The architectural trade: you move spend from a one-time capital cost (indexing) to a recurring operating cost (per query). That is often the right trade for a corpus that changes frequently — and the wrong one for a static corpus with very high query volume.

THE PUBLISHED FIGURES

Indexing cost: identical to vector RAG, and stated as **0.1%** of full GraphRAG indexing cost.
That is the single most important number on this slide.

Query cost: reported as more than **700× lower** than GraphRAG global search, while matching or exceeding answer quality.

Quality claim: Microsoft Research reports a single flexible query mechanism that "substantially outperform[s] a diverse range of specialized query mechanisms."

Workshop caveat — say this out loud: LazyGraphRAG is a *research result* from Microsoft Research, not a supported product SKU. Treat these figures as directional evidence for a pilot, and re-benchmark on your own corpus before you commit a budget line.

Sources: microsoft.com/en-us/research/blog/lazygraphrag-setting-a-new-standard-for-quality-and-cost (0.1% indexing cost, 700× lower query cost, "no prior summarization", outperformance claim) · Microsoft Research LinkedIn post (scalability statement).

MAX CONTROL · MAX OWNERSHIP

Path A · OSS library, self-hosted on Azure. You own the pipeline, and you own the maintenance.

STAND IT UP · MINIMUM VIABLE PATH

```
$ pip install graphrag
$ graphrag init --root ./ragtest
# edit settings.yaml → point at Azure OpenAI
# api_base, api_version, deployment_name
# auth_type: azure_managed_identity

$ graphrag prompt-tune --root ./ragtest # do NOT skip
$ graphrag index --root ./ragtest
$ graphrag query --root ./ragtest \
    --method local --query "..."
```

Where it runs in production: the indexer is a batch job, not a service. Run it on **Azure Container Apps Jobs** or **AKS**, triggered by an event or schedule; write the parquet outputs to **ADLS Gen2**.

Accelerator: `Azure-Samples/graphrag-accelerator` gives you an API-wrapped deployment to benchmark your own corpus quickly — also a research sample, not a supported product.

THE HONEST SCORECARD

✓ Strengths

· Full control of extraction prompts, entity types, chunking.
· Reproducible offline pipeline — parquet artefacts you can inspect and diff.
· No dependency on preview REST APIs.
· Best fit for research, one-off deep analysis, and air-gapped work.

✗ What you now own yourself

· **Security trimming.** The library has no document-level ACL model. Nothing stops a user retrieving an entity extracted from a document they cannot read.
· Incremental updates, orchestration, retries, monitoring.
· CVE response on an unsupported dependency tree.
· Graph storage + serving layer for low-latency queries.

Verdict for a regulated enterprise: excellent for the **discovery pilot** — prove the graph earns its cost on your corpus. Do not make it the long-term production retrieval plane unless you are prepared to staff it like an internal product. The ACL gap alone is disqualifying for multi-tenant knowledge.

Sources: microsoft.github.io/graphrag/get_started (CLI flow, prompt tuning "strongly recommend") · github.com/microsoft/graphrag (maintenance mode, "not an officially supported Microsoft offering") · github.com/Azure-Samples/graphrag-accelerator.

GA IN 2026-04-01 REST

Path B · Agentic retrieval on Azure AI Search. Multi-query planning replaces most of what you wanted the graph for.

THE FOUR-STEP PIPELINE

```
01 initiation   app calls knowledge base · retrieve action
                 + query + conversation history

02 planning     LLM decomposes into focused subqueries
                 (skipped at reasoning effort = minimal)

03 execution    subqueries run IN PARALLEL, each
                 semantically reranked (L2)

04 synthesis    merged content + refs + activity log
```

Components you configure: a **knowledge base** (orchestrates), one or more **knowledge sources** (indexed or remote), a **search index** with a semantic configuration, the built-in **semantic ranker**, and an **Azure OpenAI** model for planning.

Reasoning effort is the cost dial: `minimal` (no LLM planning) · `low` (default) · `medium`. It is set on the knowledge base.

Also exposed as MCP — `knowledge_base_retrieve` — so agent frameworks can call it directly.

WHY THIS IS THE ENTERPRISE DEFAULT

01 · Security is built in, not bolted on.
Four native document-level access-control approaches, enforced *inside* the search pipeline at query time. This is the single biggest gap in Path A. Slides 20–21 detail it.

02 · It solves the multi-hop problem differently.
Instead of precomputing a graph, it decomposes the question and runs parallel reranked subqueries. Microsoft's stated target: "questions with multiple asks," context-dependent follow-ups, and typos.

03 · No stale-summary problem.
Nothing is precomputed at corpus level, so a changed document is correct as soon as the indexer runs. No community-summary rebuild.

Trade-offs to state honestly: agentic retrieval "adds latency compared to a single-query pipeline"; billing moves to a **token** model across two services; portal access and several features remain **preview**; and it is available only in **select regions** with tier-dependent limits.

Sources: learn.microsoft.com/azure/search/search-agentic-retrieval-concept (4-step workflow, components table, reasoning effort, latency + region caveats, GA in 2026-04-01 REST with portal preview-only) · agentic-retrieval-how-to-retrieve (MCP knowledge_base_retrieve).

FABRIC GRAPH IS GA

Path C · persist the graph as a first-class asset — for traversal, analytics and visualisation, not just retrieval.

OPTION C1 · GRAPH IN MICROSOFT FABRIC

Native, GA, OneLake-resident.

What it is: an integrated graph data management, analytics and visualisation service inside Fabric — described by Microsoft as a horizontally scalable, native graph solution, now **generally available**.

Why architects care: the graph lives beside your lakehouse data in OneLake, under the same workspace governance — not in a bespoke parquet folder owned by one team's Python job.

The GraphRAG-relevant feature: *graph-powered AI reasoning* (preview) — you can add **graph models or querysets as knowledge sources in a Fabric data agent**. That is a first-party graph-grounded answering path with no custom retrieval code.

Best fit: you already have structured relationship data — customers, accounts, assets, transactions, org hierarchy — and you want graph reasoning over it plus your documents.

Note the split: Fabric Graph itself is GA; graph-powered AI reasoning is preview. Do not conflate the two in a steering-committee slide.

OPTION C2 · AZURE COSMOS DB KNOWLEDGE GRAPH

Operational, low-latency serving.

What it is: Microsoft documents building AI knowledge graphs on **Cosmos DB for NoSQL**, positioning Cosmos DB as a unified AI database spanning document, vector, key-value, graph and table models.

Reference implementation: **CosmosAIGraph**, which implements the **OmniRAG** pattern — detect the intent, then retrieve from the graph, the vector store, or direct database lookup as appropriate.

Why OmniRAG matters more than the storage choice: it is the routing discipline from slide 06 expressed as an architecture. One retrieval strategy per question type, chosen at runtime — that is what keeps a graph deployment affordable.

Best fit: a customer-facing app needing single-digit-ms graph lookups, global distribution, and elastic scale.

Caveat: you still need to *build* the graph. Cosmos DB stores and serves it; it does not extract entities from your PDFs. Pair C2 with Path A or a Fabric pipeline for construction.

Sources: learn.microsoft.com/fabric/graph/overview + Fabric community blog "Graph in Fabric (Generally Available)" and "Graph-powered AI reasoning (Preview)" · learn.microsoft.com/azure/cosmos-db/gen-ai/cosmos-ai-graph · github.com/azurecosmosdb/cosmosaigraph (OmniRAG).

SLIDE 11 · THE RECOMMENDATION, ON ONE PAGE

TAKE THIS TO THE REVIEW

Start at Path B. Add a graph only where B provably falls short.

DIMENSION	A · OSS GRAPHRAG	B · AGENTIC RETRIEVAL ⋆	C · FABRIC / COSMOS GRAPH
Support status	Maintenance mode · not an officially supported offering	GA in 2026-04-01 REST · portal still preview	Fabric Graph GA · graph AI reasoning preview
Doc-level security	**You build it. No native ACL model.**	**Native · 4 approaches, query-time enforced**	Workspace / RBAC governance of the store
Cost shape	Heavy one-time indexing · cheap thereafter	Token-based per query, two services	Capacity / RU + construction pipeline
Freshness	Weakest · summaries go stale, rebuild needed	**Strongest · nothing precomputed corpus-wide**	Good · incremental graph upserts
Corpus-wide "themes"	**Strongest · community summaries are purpose-built**	Good via subquery fan-out, not true global synthesis	Depends on modelling effort
Multi-hop traversal	**Native**	Approximated by parallel subqueries	**Native · plus analytics & visualisation**

RECOMMENDED SEQUENCE · ZERO TO HERO

Step 1 — Build Path B properly: AI Search + integrated vectorization + semantic ranker + agentic retrieval, with document-level security on from day one. This alone answers most "hard" questions. **Step 2 —** Run Path A offline on a 100–500 doc sample to measure the delta on your genuinely global questions. **Step 3 —** Only if the delta is material, promote the graph into Path C as a governed, incrementally-maintained store — and route to it selectively (OmniRAG).

Synthesised from the primary sources cited on slides 02, 07, 08, 09 and 10. Status labels reflect those sources as read for this workshop — re-verify GA/preview state at learn.microsoft.com before committing a design.

SLIDE 12 · THE TARGET ARCHITECTURE, SIX LAYERS

ALL TRAFFIC PRIVATE

Every arrow crosses a private endpoint. Every hop carries the caller's identity.

LAYER 1 **Sources**	SharePoint · ADLS Gen2 · Blob · OneLake Azure SQL · Confluence · ServiceNow	**Governance starts here, not later.** Permissions and sensitivity labels must already exist at source — the pipeline propagates them, it cannot invent them.
LAYER 2 **Ingest + enrich**	AI Search indexer + skillset Document Intelligence · PII Detection skill	**Crack → chunk → mask → embed.** ACL and label metadata carries through as index fields; if the skillset chunks, they move to index projections.
LAYER 3 **Graph build**	Container Apps Job / AKS batch → entities, relationships, communities	**Optional layer — only if slide 11 step 2 justified it.** Batch, not a service. Idempotent, restartable, cost-capped per run.
LAYER 4 **Stores**	AI Search index (text + vectors + ACLs) Fabric Graph / Cosmos DB · ADLS artefacts	**Two stores, one truth.** The search index is authoritative for permissions; the graph store is authoritative for topology. Never let them disagree on identity.
LAYER 5 **Retrieval**	Knowledge base + knowledge sources agentic retrieval · semantic ranker · router	**The router is the cost-control point.** Classify each question, then pick basic / local / global / graph. This is the OmniRAG discipline from slide 10.
LAYER 6 **Orchestration**	Microsoft Foundry agents · App Service APIM · Entra ID · App Insights	**On-behalf-of, end to end.** The user's Entra token must reach the retrieve call — never a service principal with blanket read on the whole index.

Sources: learn.microsoft.com/azure/search/search-agentic-retrieval-concept (components) · search-document-level-access-overview (ACL fields, index projections when chunking) · search-security-best-practices (private endpoints) · cosmosaigraph OmniRAG.

SLIDE 13 · EXACTLY WHAT TO PROVISION

FROM ZERO · NOTHING ASSUMED

Nine required services, four optional. Mandatory ones first.

#	SERVICE	ROLE IN THIS BUILD	CONFIG THAT MATTERS ON DAY 1	WHY YOU CAN'T SKIP IT
01	**Azure AI Search**	Index, vectors, semantic ranker, knowledge base	Standard tier or higher · semantic config · permission filters enabled	The only component that enforces doc-level ACLs at query time
02	**Azure OpenAI**	Embeddings · query planning · extraction · answers	Separate deployments per role · per-deployment TPM quotas	Splitting deployments is how you cap indexing spend
03	**ADLS Gen2 / Blob**	Source documents + graph artefacts	Hierarchical namespace ON · POSIX ACLs on directories	ADLS Gen2 is the only source with native file-level ACL ingestion
04	**Microsoft Entra ID**	Identity for users, services and query tokens	Groups over individual users · OBO flow in the app	Microsoft's guidance: prefer group access for manageability
05	**Managed identity**	Service-to-service auth, no secrets	**System-assigned** on the search service	User-assigned is *not* supported for Purview label extraction
06	**Azure AI Language**	PII detection + masking in the skillset	Attached to the AI Search skillset as PII Detection skill	Masking must happen *before* text reaches the index or the LLM
07	**VNet + Private Endpoints**	Network isolation for every PaaS hop	PE per service · public network access disabled · Private DNS zones	Required for Foundry agents to query the index securely
08	**Microsoft Foundry**	Agent runtime · Foundry IQ knowledge layer	Private Link configured · knowledge base attached	Agentic retrieval is the knowledge layer behind Foundry IQ
09	**App Insights / Monitor**	Token spend, latency, activity-log capture	Log the retrieval activity log per request	The activity log is Microsoft's own recommended cost-control tool
OPT	**Purview · Fabric Graph · Cosmos DB · Container Apps Jobs**	Purview for classification/DLP/lineage · Fabric Graph or Cosmos DB when you promote the graph to a governed store (slide 10) · Container Apps Jobs to run the OSS indexer as batch (slide 08)		

Sources: learn.microsoft.com/azure/search — search-document-level-access-overview (system-assigned MI requirement, group-over-user guidance, ADLS Gen2 ACLs) · search-agentic-retrieval-concept (components, activity log for cost control) · cognitive-search-skill-pii-detection · search-security-best-practices · foundry/how-to/configure-private-link · foundry/agents/concepts/what-is-foundry-iq.

DO THIS BEFORE ANY DATA

Landing zone and identity first. Retrofitting security onto a live index means a full rebuild.

PROVISIONING ORDER · CLI SKELETON

```
# 1 · network first — everything lands inside it
az network vnet create -g rg-kg -n vnet-kg \
    --address-prefix 10.20.0.0/16
az network vnet subnet create -g rg-kg --vnet-name vnet-kg \
    -n snet-pe --address-prefixes 10.20.1.0/24

# 2 · storage WITH hierarchical namespace (non-negotiable)
az storage account create -g rg-kg -n stkgdocs \
    --sku Standard_ZRS --enable-hierarchical-namespace true \
    --public-network-access Disabled

# 3 · search service + SYSTEM-assigned identity
az search service create -g rg-kg -n srch-kg \
    --sku standard --identity-type SystemAssigned \
    --public-network-access disabled

# 4 · RBAC, not keys — disable key auth entirely
az search service update -g rg-kg -n srch-kg \
    --auth-options aadOrApiKey --aad-auth-failure-mode http401

# 5 · private endpoints per service, then Private DNS zones
# privatelink.search.windows.net / .blob.core.windows.net
# privatelink.openai.azure.com
```

FIVE DECISIONS THAT ARE EXPENSIVE TO REVERSE

01 · System-assigned managed identity on the search service.
Microsoft states user-assigned identities are *not* supported for Purview sensitivity-label extraction — the service's own identity must hold those permissions.

02 · Groups, never individual users, in ACLs.
Microsoft's explicit guidance for ACL-secured content: "use group access over individual user access for ease of management." Individual-user ACLs will not survive re-orgs.

03 · Hierarchical namespace ON at storage creation.
This cannot be toggled later without migration, and it is the prerequisite for POSIX-like ACL ingestion via the ADLS Gen2 indexer.

04 · Decide your API version now.
Document-level permissions and Purview labels require preview REST (2026-05-01-preview) or prerelease SDKs. Pin it, and record it as accepted preview risk.

05 · Region choice = residency choice.
Agentic retrieval is only in select regions; that list must intersect with your residency requirement (slide 24) *before* you provision.

Sources: learn.microsoft.com/azure/search/search-document-level-access-overview (system-assigned MI, group-over-user, 2026-05-01-preview requirement) · search-how-to-managed-identities · search-security-rbac · search-agentic-retrieval-concept (select regions) · storage/blobs/data-lake-storage-introduction.

INDEXER + DATA SOURCE

Pull model, incremental by default. Set change detection now or you will re-index everything, forever.

DATA SOURCE + INDEXER · THE SHAPE

```
// POST /datasources?api-version=2026-05-01-preview
{
  "name": "ds-kg-docs",
  "type": "adlsgen2",
  "credentials": { "connectionString":
    "ResourceId=/subscriptions/.../stkgdocs;" },
  // incremental — never full rebuild on schedule
  "dataChangeDetectionPolicy": {
    "@odata.type": "#Microsoft.Azure.Search.
      HighWaterMarkChangeDetectionPolicy",
    "highWaterMarkColumnName": "metadata_storage_last_modified"
  },
  // tombstones — deletes must propagate
  "dataDeletionDetectionPolicy": {
    "@odata.type": "#Microsoft.Azure.Search.
      NativeBlobSoftDeleteDeletionDetectionPolicy"
  }
}

// indexer: enable the enrichment cache to avoid
// re-running expensive skills on unchanged docs
"cache": { "storageConnectionString": "...",
        "enableReprocessing": true }
```

CHUNKING · THE HIGHEST-LEVERAGE DIAL

Use integrated vectorization. Text Split skill + an embedding skill inside the skillset means AI Search chunks and vectorises for you — no external pipeline to maintain, and chunk boundaries stay consistent between index and query.

The one number to test on your corpus: chunk size. Microsoft's own reporting on graph extraction found that dropping from **1,200 → 600 tokens** let GPT-4o-mini reach performance comparable to GPT-4-Turbo at a fraction of the cost. Run both and measure; do not inherit a default.

Critical, easy-to-miss rule: if your skillset chunks documents, permission metadata **moves from indexer field mappings to index projections**. Miss this and chunk-level rows come back unfiltered — the security model silently fails open at exactly the granularity you retrieve at.

Also: use Document Intelligence for scanned PDFs and complex tables before splitting — bad extraction upstream cannot be fixed by better retrieval downstream.

Sources: learn.microsoft.com/azure/search — search-how-to-create-indexers + search-how-to-index-sql-database (dataChangeDetectionPolicy / high water mark) · enrichment-cache-how-to-manage · search-document-level-access-overview ("if your skillset chunks documents… ACL fields move to index projections") · techcommunity.microsoft.com GraphRAG cost article (1,200→600 chunk finding).

MASK BEFORE INDEX

Masking happens in the skillset — before text is embedded, indexed, or sent to any extraction LLM.

PII DETECTION SKILL · SKILLSET FRAGMENT

```
{
  "@odata.type":
    "#Microsoft.Skills.Text.PIIDetectionSkill",
  "name": "pii-mask",
  "context": "/document/pages/*",

  // confidence floor — tune per jurisdiction
  "minimumPrecision": 0.8,

  // replace, don't just flag
  "maskingMode": "replace",
  "maskingCharacter": "*",

  // scope: only what you must redact
  "categories": [ "Person", "PhoneNumber",
    "Email", "CreditCardNumber", "IsraelNationalID" ],

  "inputs": [
    { "name": "text", "source": "/document/pages/*" }
  ],
  "outputs": [
    // feed THIS into the embedding skill,
    // never the raw /document/pages/*
    { "name": "maskedText", "targetName": "safeText" },
    { "name": "piiEntities", "targetName": "piiFound" }
  ]
}
```

FIVE RULES FOR THE ENRICHMENT CHAIN

01 · Wire the masked output downstream, explicitly.
The skill produces `maskedText`, but nothing forces you to use it. If your embedding skill still reads the raw page, you have vectorised PII and the masking is decorative.

02 · Graph extraction is a PII amplifier.
Entity extraction is designed to pull out people, orgs and identifiers, then write them into descriptions and community summaries. Unmasked PII spreads into artefacts no DLP scan is watching.

03 · Note the preview status.
The PII Detection skill has been documented as preview. Record it in your risk register alongside the preview REST dependency from slide 14.

04 · Keep `piiEntities` as an audit field.
Store what was found (not the value) so you can prove coverage to an auditor and measure detection drift over time.

05 · Masking is not a permission model.
Redaction protects the *content*; it does not decide *who may retrieve the document*. That is slides 20–21. You need both.

Sources: learn.microsoft.com/azure/search/cognitive-search-skill-pii-detection (skill definition, maskingMode, minimumPrecision, categories, maskedText/piiEntities outputs) · learn.microsoft.com/azure/ai-services/language-service/personally-identifiable-information/overview.

RUN AS BATCH, CAP THE SPEND

Treat graph build as a governed batch job, not a library call. It is the only stage that can bankrupt a pilot.

SETTINGS.YAML · THE PARTS THAT COST MONEY

```yaml
models:
  default_chat_model:
    type: azure_openai_chat
    # cheap model + small chunks beats big model
    deployment_name: gpt-4o-mini
    auth_type: azure_managed_identity
    # protect your TPM quota from the batch job
    tokens_per_minute: 150000
    concurrent_requests: 12

chunks:
  # the single biggest cost lever (slide 05)
  size: 600
  overlap: 100

extract_graph:
  # constrain types — fewer, sharper entities
  entity_types: [organization, person, contract,
    obligation, site, certification]
  # each gleaning = another LLM pass per chunk
  max_gleanings: 1

community_reports:
  # summarising every level is rarely worth it
  max_length: 1500
```

FIVE OPERATIONAL CONTROLS

01 · Always prompt-tune first.
Microsoft's docs "strongly recommend" it and state that out-of-the-box results "may not yield the best possible results." Auto-tuning adapts prompts to your domain — run it before the full index.

02 · Benchmark on a sample, then extrapolate.
Microsoft's published rough guide: cost per word. Their worked example — 30,000 words × $0.0000113/word with GPT-4o-mini ≈ **$0.34**. Use your own sample; their tables are explicitly "a very rough starting point."

03 · Separate the Azure OpenAI deployment.
Give the indexer its own deployment and TPM quota so a runaway batch cannot starve your live chat traffic.

04 · Version every index run.
Write artefacts to `/graph/v{n}/` in ADLS. Never overwrite in place — you need to A/B a prompt change and roll back.

05 · Mind the version discipline.
Run `graphrag init --force` between minor versions; use the migration notebook between major ones to avoid re-indexing. Back up prompts first — it overwrites them.

Sources: microsoft.github.io/graphrag (prompt tuning "strongly recommend", versioning + migration notebook guidance) · techcommunity.microsoft.com "GraphRAG Costs Explained" ($0.0000113/word GPT-4o-mini, 30k-word $0.34 worked example, "very rough starting point" caveat) · microsoft.com/research GraphRAG auto-tuning.

THE COST-CONTROL POINT

One knowledge base, several knowledge sources, and a router in front of everything.

KNOWLEDGE SOURCE + KNOWLEDGE BASE

```
// 1 · knowledge source — what content is in scope
PUT /knowledgeSources/ks-contracts
{
  "kind": "searchIndex",
  "searchIndexParameters": { "searchIndexName": "idx-kg" },
  // carry Purview labels into the pipeline
  "ingestionPermissionOptions": [ "sensitivityLabel" ]
}

// 2 · knowledge base — orchestrates the retrieval
PUT /knowledgeBases/kb-enterprise
{
  "knowledgeSources": [
    { "name": "ks-contracts" },
    { "name": "ks-incidents" }
  ],
  // THE cost dial: minimal | low (default) | medium
  "retrievalReasoningEffort": "low",
  "models": [{ "kind": "azureOpenAI",
    "azureOpenAIParameters": { "deploymentId": "gpt-4o-mini" } }]
}

// 3 · query WITH the user's token — never a service SP
POST /knowledgeBases/kb-enterprise/retrieve
  x-ms-query-source-authorization: <user Entra token>
```

THE ROUTER · WHERE BUDGETS ARE WON OR LOST

```
classify(question) →
  single_fact     → effort=minimal (no LLM plan)
  multi_ask       → effort=low (default)
  comparative     → effort=medium
  corpus_theme    → graph global search (if built)
  entity_deep_dive → graph local / DRIFT
```

Why this matters more than any other tuning: reasoning effort directly determines whether an LLM planning call happens at all. At `minimal`, planning is skipped and queries go straight to the sources. Most enterprise traffic is single-fact — routing it to `minimal` is free money.

Microsoft's own cost-control guidance, applied here: reduce the number of knowledge sources to lower fan-out; lower the reasoning effort; organise content (curated summaries, tables) so answers need fewer sources; and read the returned **activity log** to see exactly which subqueries ran against which source.

Sources: learn.microsoft.com/azure/search — agentic-knowledge-source-overview (ingestionPermissionOptions / sensitivityLabel) · agentic-retrieval-how-to-create-knowledge-base · search-agentic-retrieval-concept (reasoning effort levels incl. minimal skipping planning; "Tips for controlling costs"; activity log) · search-query-access-control-rbac-enforcement (x-ms-query-source-authorization).

ON-BEHALF-OF, END TO END

The app's only security job: carry the user's identity all the way to the retrieve call.

THE IDENTITY CHAIN · GET THIS EXACTLY RIGHT

```
# 1 · user signs in to your app (Entra ID)
user → App Service / SPA → id_token + access_token

# 2 · app exchanges for a downstream token (OBO)
POST /oauth2/v2.0/token
  grant_type = urn:ietf:params:oauth:grant-type:jwt-bearer
  assertion   = <user access_token>
  scope       = https://search.azure.com/.default

# 3 · TWO tokens go to AI Search on every query
POST /knowledgeBases/kb-enterprise/retrieve
  Authorization: Bearer <app identity>
    # → needs Search Index Data Reader
  x-ms-query-source-authorization: <USER token>
    # → drives per-document trimming

# 4 · AI Search performs two checks, in order
# a. can this APP read the index?
# b. which docs may THIS USER see?

# 5 · grounding data → your LLM → answer + citations
```

FIVE APP-LAYER REQUIREMENTS

01 · Two tokens, two purposes.
The app's own credential grants index access (*Search Index Data Reader*). The user token in the header decides which documents come back. Sending only the app token returns everything — a silent data-leak class.

02 · Cite from returned references only.
The retrieve response carries source references and an activity log. Build citations from those — never let the LLM invent a document name it did not receive.

03 · Surface sensitivity labels in the UI.
Label metadata is returned per reference and at response level — use it to render a classification banner, not just to filter silently.

04 · Log the activity log.
Persist which subqueries ran, against which sources, with what token usage. This is your cost attribution and your audit trail in one artefact.

05 · Add your own content safety.
Microsoft is explicit that implementing responsible-AI mitigations — metaprompts, content filters, safety systems — remains your responsibility.

Sources: learn.microsoft.com/azure/search — search-document-level-access-overview (two-check model, Search Index Data Reader, responsible-AI responsibility statement) · search-query-access-control-rbac-enforcement · agentic-retrieval-how-to-retrieve (references, activity log, sensitivityLabelInfo) · entra/identity-platform/access-tokens.

SLIDE 20 · DOCUMENT-LEVEL SECURITY — CHOOSE YOUR APPROACH

PICK BY DATA SOURCE

Four native approaches. Only one is GA — that shapes your rollout.

APPROACH	HOW IT DECIDES ACCESS	USE IT WHEN	LIMITATION TO PLAN AROUND	STATUS
Security filters string comparison	App passes a user/group identity string; an OData filter excludes non-matching docs.	Custom identity system, non-Microsoft security framework, or **any push-model index**.	Plain string matching — no nested group resolution, no Entra semantics. You maintain the strings.	**GA · API-agnostic** works on any version
POSIX-like ACL / RBAC scopes Entra-native	Entra principal from the query token is compared to permission metadata on documents.	Content in **ADLS Gen2** or Azure Blob with existing ACL / RBAC assignments.	ACLs apply to ADLS Gen2 dirs/files; for plain blobs, RBAC scope preservation is **container-level only**.	**Preview** REST + prerelease SDKs
Purview sensitivity labels policy-driven	Indexer extracts labels; at query time the label + Entra token + Purview policy decide access.	Content already governed by **Microsoft Information Protection** policies.	**Single-tenant only**; requires RBAC auth + system-assigned MI; Autocomplete/Suggest unavailable on those indexes.	**Preview** REST + SDKs only
SharePoint in M365 ACLs source-native	Indexer extracts SharePoint permission metadata for query-time checks.	Content from SharePoint libraries, lists, ASPX site pages.	Unique-permission items sync incrementally per run; **inherited** changes need an explicit refresh.	**Preview**

THE ARCHITECT'S READ

Three of the four are preview. If you cannot ship on preview APIs, your GA-safe path is **security filters** with a push-model index — you resolve Entra groups yourself via the Graph SDK and write them as strings. That is more code, but it is generally available and API-agnostic.
Whatever you choose, choose one per source and document it — mixing models across sources in one index is how permission bugs become invisible.

Source: learn.microsoft.com/azure/search/search-document-level-access-overview — "Approaches for document-level access control" + "Choose an approach" tables, incl. container-level RBAC caveat for blobs, Purview single-tenant + Autocomplete/Suggest limitation, and SharePoint inherited-permission refresh behaviour (2026-05-01-preview).

SLIDE 21 · THE FAILURE MODE NOBODY TESTS FOR

THE SYNC LAG IS REAL

Revoking access at the source does not revoke it in the index. Enforcement reads *synchronised metadata*.

WHAT HAPPENS ON EVERY QUERY

```
01 extract user, group and scope claims from the
    token in x-ms-query-source-authorization

02 compare those claims to permission metadata
    stored alongside the indexed documents
    (ACL entries · RBAC scopes · Purview labels · SP ACLs)

03 return ONLY documents whose synchronised
    metadata grants this caller access
```

Why this beats post-query trimming: filtering happens *inside* the search pipeline, before scoring. Microsoft's stated benefits — no custom nested-group resolution code, better performance than loading large result sets and trimming in your app, and reuse of Entra as the single source of truth.

Anti-pattern this replaces: retrieving top-50 then filtering in application code. That returns fewer than 50 permitted results and leaks counts, ranks and existence.

THE PERMISSION-SYNC LAG · PLAN FOR IT

Microsoft states it plainly: permission changes in the source system — Entra group membership, ADLS ACLs, Purview label assignments, SharePoint ACLs — are reflected in results *only after* that metadata is synchronised to the index via an indexer run, push-API update, or Purview refresh.

Also documented: the preview API "can't modify access permissions that were set outside" of it, and "a timing lag occurs before [it] recognizes changes to those access or permission restrictions."

Three controls to implement:

· Define and publish an **RTO for revocation** — how fast must a removal take effect? Set indexer frequency to match.

· On urgent revocation, trigger an **on-demand indexer run** or push the updated doc — do not wait for the schedule.

· For SharePoint, remember inherited-permission changes need an **explicit refresh**; only unique-permission items pick up incrementally.

Source: learn.microsoft.com/azure/search/search-document-level-access-overview — "Enforce document-level permissions at query time" (3-step claim comparison), "Benefits of document-level access control", the permission-synchronisation paragraph, and the 2026-05-01-preview timing-lag notice · search-indexer-sharepoint-access-control-lists.

REUSE, DON'T REBUILD

If your data is already labelled in Purview, AI Search can enforce those labels at query time.

SENSITIVITY-LABEL ENFORCEMENT · SETUP ORDER

01 · Label the documents first. Labels must be applied *before* indexing so the system can recognise and preserve them during ingestion.

02 · Enable a system-assigned managed identity on the search service. User-assigned is not supported — the service's own identity must hold the elevated Purview permissions.

03 · Have a tenant global admin or privileged role admin grant access so the search service can authenticate to Purview and extract label metadata.

04 · Configure index, data source and indexer with the 2026-05-01-preview REST API (or an SDK that supports label ingestion). Supported sources: Blob, ADLS Gen2, SharePoint in M365, OneLake.

05 · If the index is chunked, project the label to every chunk row. Without that projection, chunk-level references are *not* filtered.

06 · Attach the user's Entra token in `x-ms-query-source-authorization` on each query.

WHAT PURVIEW DOES — AND DOESN'T — GIVE YOU

✓ **Label-aware retrieval, everywhere.** Labels surface through two paths: the direct query API, and knowledge sources / agentic retrieval — which return per-reference `sensitivityLabelInfo` plus response-level metadata so agents and chat apps get the same label-aware filtering.

✓ **Auditable elevated read.** Administrators can issue elevated-read requests for auditable investigations — a legitimate break-glass path rather than an undocumented workaround.

✗ **Hard constraints to design around.** Single-tenant scenarios only; RBAC authentication required; preview support via REST and SDKs only; and **Autocomplete and Suggest are unavailable** on Purview-enabled indexes.

Lineage & DLP caveat: label enforcement is what AI Search implements. Broader Purview DLP and lineage over your graph artefacts is your own scanning and cataloguing work — don't assume the retrieval layer covers it.

Source: learn.microsoft.com/azure/search/search-document-level-access-overview — "Pattern for Microsoft Purview sensitivity labels (preview)", "Where sensitivity labels are surfaced", incl. system-assigned MI requirement, admin grant step, chunk projection requirement, single-tenant + Autocomplete/Suggest limitations, elevated read · search-indexer-sensitivity-labels.

NO PUBLIC SURFACE

A private endpoint per service, public access disabled, and Private DNS wired correctly — or nothing resolves.

PRIVATE ENDPOINT + DNS ZONE INVENTORY

service	private DNS zone
AI Search	privatelink.search.windows.net
Azure OpenAI	privatelink.openai.azure.com
Blob / ADLS	privatelink.blob.core.windows.net
+ dfs endpoint	privatelink.dfs.core.windows.net
AI Language	privatelink.cognitiveservices.azure.com
Cosmos DB	privatelink.documents.azure.com
Key Vault	privatelink.vaultcore.azure.net
Foundry	per configure-private-link guidance

```
# then lock the front door
az search service update -g rg-kg -n srch-kg \
    --public-network-access disabled

# and verify from inside the VNet, not your laptop
nslookup srch-kg.search.windows.net
> must return a 10.x private IP, not a public one
```

FIVE THINGS THAT BREAK IN PRIVATE MODE

01 · Indexers lose reach. An indexer on a private search service cannot reach a private data source over the public internet. You need a shared private link / private-endpoint path for indexer-to-source traffic — budget for it.

02 · Foundry agents need the PE explicitly. Microsoft's guidance: ensure the search service has a private endpoint on your VNet so the agent can query the index securely.

03 · The portal stops working. With public access disabled, portal-based index management and the chat playground need a jumpbox, Bastion or VPN inside the VNet.

04 · Private DNS is the usual culprit. Most "it worked yesterday" failures are a missing zone link to the VNet, not a firewall rule. Verify resolution from inside the network.

05 · Preview connections can leave the boundary. Microsoft warns the 2026-05-01-preview supports connections to other Microsoft and third-party services, whose use "might result in data processing or storage outside of the Azure compliance boundary."

Sources: learn.microsoft.com/azure/search/search-security-best-practices (private endpoint architecture) · learn.microsoft.com/azure/foundry/how-to/configure-private-link (PE required for agent-to-index queries) · search-document-level-access-overview (preview cross-boundary data-flow warning) · github.com/Azure-Samples/azure-search-openai-demo deploy_private docs.

DECIDE BEFORE PROVISIONING

Residency is a region decision made once. Getting it wrong means rebuilding the index, not reconfiguring it.

THREE LEVERS, INCREASING STRICTNESS

LEVER 1 · Region pinning.
Deploy every service in one geography and keep it there. Necessary but *not sufficient* — some Azure services store limited usage data or service traces centrally, so read the per-service residency documentation rather than assuming.

LEVER 2 · Azure OpenAI Data Zones.
Data Zones constrain where inference is processed. The **European Union Data Zone** spans Azure OpenAI regions located within EU member states — useful when you need regional processing guarantees without pinning a single region.

LEVER 3 · EU Data Boundary.
A geographically defined boundary within which Microsoft commits to store and process Customer Data and personal data. Microsoft has stated it now includes **AI data processing residency** — delivering on end-to-end AI data-processing commitments in Europe.

THE RESIDENCY CHECKLIST FOR THIS WORKLOAD

01 · Intersect residency with feature availability.
Agentic retrieval is available in *select regions* only. Confirm your required region appears in the support list **before** you commit an architecture — this is the most common late-stage blocker.

02 · Audit the preview cross-boundary risk.
Microsoft explicitly warns that preview connections to other Microsoft and third-party services "might result in data processing or storage outside of the Azure compliance boundary."

03 · Own the boundary decision explicitly.
Microsoft states it is *your* responsibility to manage whether data flows outside your organisation's compliance and geographic boundaries, and to provision appropriate permissions and approvals.

04 · Remember the graph is derived personal data.
Entity descriptions and community summaries are new artefacts generated *from* your source documents. They inherit the residency and retention obligations of the source — and they are easy to forget in a DPIA because no one filed them as a data store.

Sources: learn.microsoft.com/privacy/eudb/eu-data-boundary-learn · azure.microsoft.com/blog "Microsoft strengthens sovereign cloud capabilities" (EU Data Boundary incl. AI data processing residency) · azure.microsoft.com/blog Azure OpenAI Data Zones (EU Data Zone scope) · azure.microsoft.com/explore/global-infrastructure/data-residency · learn.microsoft.com/azure/search/search-region-support + search-document-level-access-overview (preview boundary + customer responsibility statements).

SLIDE 25 · COST ENGINEERING AND TOKEN QUOTAS

WORKED EXAMPLE, CITED

Two bills, two units. Model both, or you will be surprised by one.

MICROSOFT'S OWN WORKED EXAMPLE · AGENTIC RETRIEVAL

```
assumptions (gpt-4o-mini)
  2,000 agentic retrievals
  3 subqueries per plan     → ~6,000 queries
  50 chunks reranked / subquery → 300,000 chunks
  500 tokens per chunk      → 150M rerank tokens
  2,000 input tokens / conversation
  350 output tokens / plan

resulting cost
  AI Search reranking          $3.30
  AOAI input (4M tokens)       $0.60
  AOAI output (700K tokens)    $0.42
  ──────────────────────────────────
  total                        $4.32
```

Read the shape, not the number. ~76% of that cost is **reranking inside AI Search**, not the LLM. Cutting LLM planning alone barely moves it — cutting *fan-out* (fewer sources, fewer subqueries, fewer chunks reranked) is what actually reduces the bill.

Billing model: AI Search bills *tokens* for subquery execution and semantic ranking, with a monthly free allowance on the free plan and pay-as-you-go on standard. Azure OpenAI bills separately for planning and answer synthesis.

SIX CONTROLS · APPLY ALL OF THEM

01 · Consolidate knowledge sources. Microsoft's guidance: fewer indexes lowers fan-out and token volume. Every extra source multiplies every query.

02 · Lower reasoning effort per intent. Route single-fact questions to `minimal` — planning is skipped entirely. Reserve `medium` for genuinely comparative asks.

03 · Curate content for retrievability. Microsoft suggests organising content — curated summaries, tables — so answers need fewer sources and documents.

04 · Mine the activity log. It shows which subqueries hit which sources with what parameters — reissue them and compare against API-reported usage to find waste.

05 · Separate TPM quotas. Indexing, planning and chat get their own Azure OpenAI deployments so a batch job cannot starve production.

06 · Alarm on cost-per-question. Track it as a product SLO. A regression here is a design bug, not a finance problem.

Sources: learn.microsoft.com/azure/search/search-agentic-retrieval-concept — "Availability and pricing", "Example: Estimate costs" (all figures above quoted from that worked example) and "Tips for controlling costs" · techcommunity.microsoft.com "GraphRAG Costs Explained" (indexing-side cost model). Rates change — re-check Azure pricing pages before budgeting.

THE OPERATIONAL TAX

Vector chunks update per document. Community summaries don't — that asymmetry is your biggest ops risk.

BLAST RADIUS OF ONE CHANGED DOCUMENT

```
artefact                 invalidated?   cost
─────────────────────────────────────────────
chunks + vectors         that doc only  cheap
ACL / label metadata     that doc only  cheap
entities + relations     that doc only  moderate
community structure      POSSIBLY ALL   expensive
community summaries      POSSIBLY ALL   expensive

# why: Leiden clustering is global. New edges can
# redraw community boundaries, so summaries above
# the changed node may no longer be accurate.
```

The practical policy most teams land on:

· Vector + ACL layer → **incremental, continuous** (indexer schedule).
· Graph entities → **incremental append** per new document.
· Communities + summaries → **periodic full rebuild** (nightly / weekly), versioned to `/graph/v{n}/`, promoted only after evaluation passes.

State it as an SLA: "single-fact answers are current within minutes; corpus-level thematic answers are current as of the last graph build." Users can accept that — they cannot accept silent staleness.

THE MECHANICS YOU NEED TO KNOW

01 · Reset means full reprocess. Microsoft is explicit: if you reset an indexer, *all* documents in the search corpus are reprocessed. Never reset casually on a large corpus — it re-runs every billable skill.

02 · Use the enrichment cache. It lets unchanged documents skip expensive skill re-execution. Without it, every schema tweak becomes a full-cost re-enrichment of the whole corpus.

03 · Reset selectively when you must. AI Search supports resetting an indexer, specific skills, or individual documents — prefer the narrowest scope that fixes the problem.

04 · Deletions must propagate — including derived data. A deletion-detection policy removes the document from the index, but entity descriptions and community summaries generated from it may still quote its content. For right-to-erasure requests, deleting the source document is *not* sufficient — you must rebuild the affected graph artefacts. Put this in your DSR runbook.

Sources: learn.microsoft.com/azure/search — search-howto-run-reset-indexers (reset indexer / skills / individual documents) · enrichment-cache-how-to-manage ("if you reset an indexer, all documents in the search corpus are reprocessed") · search-howto-reindex · microsoft.github.io/graphrag (Leiden hierarchical clustering, bottom-up community summaries).

KILL ON SIGHT

The graph looks impressive in a demo and degrades quietly in production. These are why.

ANTI-PATTERN 01 · THE BIG ONE

Ignoring entity resolution.

Symptom: "Acme Corp", "Acme Corporation" and "ACME" become three nodes. Each holds part of the truth; every answer is partial.

Cause: in Microsoft GraphRAG, entities are matched primarily *by name*. Extraction always spawns duplicates.

Fix: a dedicated resolution pass — alias tables, normalisation, embedding-based matching.

ANTI-PATTERN 02

Security bolted on later.

Symptom: the pilot works, then security review kills it because any user can retrieve any entity.

Why it's fatal: permission metadata must be present *at ingestion*. Adding ACLs later means re-indexing the corpus — and re-extracting the graph.

Fix: phase 1, not phase 7. Slide 14.

ANTI-PATTERN 03

Global Search for everything.

Symptom: every question costs the same as a corpus-wide synthesis, and takes 20+ seconds.

Cause: no router. The library ships four modes precisely because one mode cannot serve all traffic.

Fix: classify intent first, then pick basic / local / DRIFT / global. Slide 18.

ANTI-PATTERN 04

Skipping prompt tuning.

Symptom: generic entity types, thin descriptions, a graph that answers nothing domain-specific.

Cause: running default prompts. Microsoft's own docs "strongly recommend" tuning and warn out-of-the-box results may not be best.

Fix: `graphrag prompt-tune` before the first full index — it's the cheapest quality win available.

ANTI-PATTERN 05

Indexing the whole lake.

Symptom: a five-figure indexing bill and a noisy graph full of boilerplate entities from email footers and templates.

Cause: "index everything" instead of curating a corpus for one answerable question set.

Fix: scope to a domain. Benchmark cost per word on a sample first (slide 17).

ANTI-PATTERN 06

No evaluation set.

Symptom: "the graph feels better" — with no evidence, and no way to detect regression after a prompt or model change.

Fix: 50–100 real questions with agreed answers, scored before and after every change. Without this you cannot justify the graph's cost, or safely upgrade a model.

Sources: memgraph.com/blog/how-microsoft-graphrag-works-with-graph-databases (entities matched primarily by name — the resolution limitation) · arxiv.org/html/2510.14271v1 (denoising LLM-generated KGs via entity resolution) · microsoft.github.io/graphrag (prompt-tuning recommendation, four query modes) · learn.microsoft.com/azure/search/search-document-level-access-overview.

SLIDE 28 · THE ROLLOUT CHECKLIST

18 BOXES · THEN SHIP

Tick all eighteen and you have a governed knowledge platform, not a demo.

DECIDE · BEFORE ANY BUILD

☐ 01 50 real questions collected; >20% need multiple documents to answer

☐ 02 Path chosen per slide 11 — B first, graph only on proven delta

☐ 03 Maintenance-mode risk of the OSS library formally accepted or avoided

☐ 04 Region intersects residency requirement *and* agentic-retrieval availability

☐ 05 Preview-API dependencies recorded in the risk register

☐ 06 Cost per word benchmarked on a real sample of your own corpus

SECURE · NON-NEGOTIABLE

☐ 07 One access-control approach chosen per data source, and documented

☐ 08 System-assigned managed identity on the search service

☐ 09 ACLs on Entra **groups**, never individual users

☐ 10 User token passed via `x-ms-query-source-authorization` on every query

☐ 11 Permission metadata projected to **chunk level** where the skillset splits

☐ 12 PII masked in the skillset, and the masked output wired downstream

☐ 13 Private endpoints on every service; public access disabled; DNS verified

☐ 14 Revocation RTO defined, and indexer cadence set to meet it

OPERATE · KEEP IT ALIVE

☐ 15 Router in place; reasoning effort mapped per intent (`minimal` for single-fact)

☐ 16 Incremental change detection + deletion detection + enrichment cache enabled

☐ 17 Freshness SLA published: facts in minutes, themes as of last graph build

☐ 18 Evaluation set of 50–100 scored questions wired into every change; activity log persisted for cost attribution and audit

☐ + DSR runbook covers derived artefacts — deleting a source document does not erase the entities and summaries built from it

Primary sources for this deck: github.com/microsoft/graphrag · microsoft.github.io/graphrag · microsoft.com/research (GraphRAG, LazyGraphRAG) · learn.microsoft.com/azure/search (agentic retrieval, document-level access, indexers, PII skill) · learn.microsoft.com/fabric/graph · learn.microsoft.com/azure/cosmos-db/gen-ai · learn.microsoft.com/privacy/eudb. GA/preview status and pricing change — re-verify before committing a design.