Data Storage

Data Engineering on Microsoft Azure

KNOWLEDGE CHEATSHEET

DESIGN THE ANALYTICAL STORE

- **Lake** — Files and folders
- **Warehouse** — Tables and keys
- **Partitions** — Scan and retain
- **Resilience** — Replicate and recover

01 Design and implement data storage · 15– 20%

Lake structure and file formats

- **ADLS Gen2** adds a hierarchical namespace and directory ACLs to Blob Storage. Organize raw, validated and curated data separately so ingestion, transformation and serving have clear boundaries.
- Place the **security boundary** first in a folder path, followed by source and time: department/source/year/month/day. This simplifies inherited permissions and lets date-filtered workloads narrow their scans.
- **Parquet** is typed, compressed, columnar storage for analytical projections and predicate pushdown. **Avro** is a row-oriented serialization format with an embedded schema. Neither is a database index for arbitrary point lookups.
- **CSV/TSV** require agreement on delimiters, headers, quoting and types. **JSON** handles nested records but adds parsing overhead. Preserve numeric and timestamp types when producing curated datasets.
- Avoid thousands of **tiny files** and excessive partition cardinality. Compact files into practical batch sizes; use compression supported by the downstream engine, such as Snappy with Parquet.

Retention and storage resilience

- **Hot** suits frequent access. **Cool** lowers capacity cost for infrequent online access. **Archive** is offline and requires rehydration; it cannot satisfy an immediate-read requirement.
- **Lifecycle management** applies tiering or deletion rules to matching blobs. Match container/prefix filters and age conditions to retention needs; enable **last access time tracking** before using access-based conditions.
- Evaluate retrieval charges, minimum retention charges and **account/redundancy support** before selecting a tier. Premium storage does not provide the same tiering options as standard storage.
- **LRS** keeps replicas in one location; **ZRS** spreads synchronous replicas across zones. **GRS/GZRS** add asynchronous replication to another region; **RA** variants expose secondary reads before failover.
- Zonal availability and **regional recovery** are separate requirements. Secondary writes require failover. **Last Sync Time** helps assess geo-replication lag and possible data loss against the recovery point objective.

Data Storage

Data Engineering on Microsoft Azure

KNOWLEDGE CHEATSHEET

DESIGN THE ANALYTICAL STORE

- **Lake** — Files and folders
- **Warehouse** — Tables and keys
- **Partitions** — Scan and retain
- **Resilience** — Replicate and recover

Warehouse distribution

- A **dedicated SQL pool** spreads table data across **60 distributions** for parallel processing. Distribution, indexing and partitioning are independent design decisions.
- Use **HASH** for large tables with a well-distributed key. Prefer a frequently joined key with many distinct values; repeated NULLs, low-cardinality flags and hot values can cause skew.
- Compatible **hash join keys and data types** can colocate joins and reduce movement. A date used to filter a narrow period can concentrate work in few distributions; assess both join locality and balance.
- **REPLICATE** makes a small table available on every compute node, avoiding movement for dimension joins. It suits small, relatively stable dimensions; frequent changes or scaling can require cache rebuilds.
- **ROUND_ROBIN** loads staging data quickly without a distribution-key decision. Joins commonly require shuffles. A **round-robin heap** is a useful transient landing table before loading the analytical model.

Indexes and partition lifecycle

- **Clustered columnstore** compresses large analytical tables and favors broad scans and aggregates. A **heap** favors staging loads; a rowstore **clustered index** can suit selective range or point access.
- Partition by a column used in common filters or retention operations, often a **date**. Include a matching **WHERE** predicate to enable partition elimination; partitioning alone does not reduce every query's I/O.
- Avoid overpartitioning a columnstore table. Around **one million rows per distribution per partition** is a useful density guideline, not an eligibility rule. Sixty million evenly spread rows yield about one million per distribution.
- **RANGE RIGHT** assigns a boundary value to the higher partition; **RANGE LEFT** assigns it to the lower partition. Start-of-month boundaries with RANGE RIGHT naturally describe calendar months.
- **Partition switching** moves data through metadata operations. Align schemas, distributions and partition boundaries; switch old data out and truncate/drop the staging table. For replacement, empty or switch out the destination partition first.

Data Storage

Data Engineering on Microsoft Azure

KNOWLEDGE CHEATSHEET

DESIGN THE ANALYTICAL STORE

- **Lake** — Files and folders
- **Warehouse** — Tables and keys
- **Partitions** — Scan and retain
- **Resilience** — Replicate and recover

Dimensional modeling and history

- Define the **fact grain** first: one event, transaction line or snapshot per row. Store measures and dimension keys in facts; keep descriptive attributes in denormalized **dimensions**.
- A **business key** identifies the source entity. A **surrogate key** identifies a warehouse row or historical version. An integer **IDENTITY** can generate surrogate keys; do not depend on values being gap-free or consecutive.
- **SCD Type 0** preserves the original value. **Type 1** overwrites it without history. **Type 2** adds a new version row. **Type 3** keeps limited history in extra current/previous columns.
- For **Type 2**, retain the business key, assign each version a surrogate key, and maintain effective start/end dates and optionally a current flag. Expire the old version and insert the replacement; facts reference the applicable version.
- Use a shared **date dimension** for fiscal/calendar attributes and role-playing order, due and ship dates. A self-referencing **manager key** models employee hierarchies and supports joins to manager attributes.

Querying lake data

- **Serverless SQL** queries lake files on demand; a **dedicated SQL pool** hosts provisioned warehouse tables. Project needed columns and restrict paths/partitions to reduce scans and serverless processing cost.
- **OPENROWSET** provides ad hoc file access. Specify format and parsing options; its **WITH** clause supplies column names and explicit types instead of relying on inference.
- Use **wildcard paths**, **filepath()** and **filename()** to select partitions or group records by source file. Recursive patterns and external-table folder behavior depend on the access method; use explicit paths when scope matters.
- An **external data source** defines the location and optional credential; an **external file format** describes encoding. An **external table** adds a reusable SQL schema over files without importing them.
- For line-delimited **JSON**, read each document as text through OPENROWSET, then apply **OPENJSON** with a typed schema. Match row delimiters to document boundaries; arrays need explicit expansion.
- Synapse **shared metadata** exposes supported Spark lake tables to serverless SQL asynchronously. Prefer compatible Parquet tables and allow synchronization delay. **Lake database templates** supply table definitions and relationships grouped by business area.

Data Storage / Data Processing

Data Engineering on Microsoft Azure

KNOWLEDGE CHEATSHEET

FROM INGESTION TO ANALYTICS

Ingest Batch or events → **Transform** SQL or Spark → **Persist** Lake or warehouse → **Serve** Queries and reports

- An **external Hive metastore** shares catalog metadata between compatible Spark environments; configure the metastore connection and underlying storage access separately. Shared table definitions do not themselves grant access to the files.
- Storage **query acceleration** filters rows and projects columns in supported CSV/JSON blobs before transferring results. This is a storage read optimization, separate from warehouse indexing and lake partition pruning.

02 Develop data processing · 40– 45%

Processing services and SQL loads

- **Data Factory/Synapse pipelines** orchestrate movement and compute. **Spark/Databricks** execute distributed transformations and custom code. **Stream Analytics** provides managed SQL-like processing for event streams.
- **ETL** transforms before loading; **ELT** loads staging data before warehouse transformations. Use **COPY INTO** or supported **PolyBase** loads for parallel ingestion into dedicated SQL pools.
- **CTAS** creates an internal table from a query and can change distribution or storage design. Build and validate a replacement, then perform a short name swap when minimizing disruption.
- **CETAS** creates an external table and writes query results to lake files. A normal external-table definition stores metadata only. Recreate unsupported external-table schema changes rather than assuming ordinary ALTER TABLE behavior.
- Validate file schema, column sizes, headers and encoding before bulk loads. **Reject thresholds** control tolerable bad rows; quarantine failures instead of silently discarding business data. Use explicit transactions for batch atomicity.

Factory objects and runtimes

- A **linked service** defines a connection or compute service; a **dataset** describes the data location/shape. An **activity** performs work, a **pipeline** groups activities, and a **trigger** starts runs.
- **Azure integration runtime** supports managed data movement and mapping data flows. Select the required region for residency constraints; the factory's location alone does not establish where all processing occurs.
- A **self-hosted integration runtime** reaches supported private or on-premises systems. Install it on supported Windows hosts with network access to the source, even when the source database runs on Linux.

Data Processing

Data Engineering on Microsoft Azure

KNOWLEDGE CHEATSHEET

FROM INGESTION TO ANALYTICS

Ingest Batch or events → **Transform** SQL or Spark → **Persist** Lake or warehouse → **Serve** Queries and reports

- Add self-hosted runtime **nodes** for availability and throughput. Tune concurrent jobs against CPU, memory and network capacity; when a copy uses self-hosted runtime, Azure data-flow core settings do not enlarge that copy engine.
- **Azure-SSIS integration runtime** executes SSIS packages. Mapping data flows use managed Spark compute; notebook/JAR activities invoke their target compute rather than converting a self-hosted runtime into Spark.

Copy and incremental ingestion

- Use **binary copy** for unchanged files without parsing their contents. Choose format-aware copy for column mapping or format conversion; a file extension alone does not define its actual format.
- **PreserveHierarchy** retains relative paths; **FlattenHierarchy** places files in one target folder; **MergeFiles** combines compatible source files. Consolidation reduces small-file overhead but one giant file can constrain parallel reads.
- For an incremental **high watermark**, read the previous committed value and capture a new upper bound. Copy rows within those bounds, then persist the new watermark only after a successful load.
- Use reliable **change tracking/CDC** when updates or deletes cannot be identified safely from a modified timestamp. Design retries and destination writes to be **idempotent** so reprocessing does not duplicate data.
- Filter file ingestion by known **time folders**, naming patterns or modification bounds. Tune **parallelCopies**, source partitioning and Azure **DIUs** against source, sink and network limits; more compute cannot remove a throttled endpoint.

Trigger selection and recovery

- A **schedule trigger** follows a wall-clock recurrence. Configure time zone, interval and calendar times deliberately; daylight saving behavior depends on the schedule and frequency.
- A **tumbling window trigger** maintains contiguous, nonoverlapping processing windows. It supports backfill, retries and dependencies; use window start/end parameters for deterministic incremental slices.
- A tumbling-window **delay** waits after a window ends before starting processing, allowing late files to arrive without changing window boundaries. **maxConcurrency** limits simultaneously running windows.
- A **self-dependency** waits for earlier windows to succeed. For hourly windows, an offset of minus one hour with a one-hour dependency size targets the preceding window.
- A **storage event trigger** reacts to supported blob creation/deletion through **Event Grid**. Filter container/path events and pass file parameters into the pipeline; it does not continuously process records inside a growing file.

Data Processing

Data Engineering on Microsoft Azure

KNOWLEDGE CHEATSHEET

FROM INGESTION TO ANALYTICS

Ingest Batch or events → **Transform** SQL or Spark → **Persist** Lake or warehouse → **Serve** Queries and reports

Pipeline control and results

- Use a parent pipeline with **Execute Pipeline** activities for ordered workflows. Run independent ingestions in parallel, then wait for required dimensions before loading facts. Configure whether the parent waits for child completion.
- **Lookup** retrieves configuration rows; **Get Metadata** retrieves properties such as size, existence or child items. **ForEach** iterates the resulting array and can run iterations concurrently.
- Use **parameters** for run inputs and **variables** for mutable pipeline state. Avoid changing shared pipeline variables inside parallel ForEach iterations; pass per-item values to child pipelines instead.
- **Script** can execute SQL and return result sets. **Lookup** can retrieve a supported procedure's result set. A **Stored Procedure** activity does not automatically expose returned rows as pipeline output.
- Dependencies can be **Succeeded, Failed, Completed or Skipped**. A handled activity failure can yield a successful pipeline; evaluate the full dependency graph. Use an explicit **Fail** activity when recovery must still report failure.
- Use a **Web activity** for supported REST orchestration, such as invoking another service's pipeline. Supply the correct endpoint, authorization and parameters; calling an API does not by itself wait for remote processing to finish.

Mapping data flow transformations

- **Select** chooses or renames columns. **Derived Column** computes or casts fields, including explicit ingestion timestamps. **Aggregate** groups rows; a selected column is not automatically retained through aggregation.
- **Flatten** unrolls nested arrays into rows. **Pivot** rotates values into columns with aggregation; **Unpivot** converts columns into rows. Choose these according to shape, not file extension.
- Use **Conditional Split** to route valid and invalid rows to separate sinks. Check string lengths and conversion rules before SQL writes; retain rejected records with enough context to investigate them.
- Use **Exists/Lookup/Join** patterns to identify new or changed rows. **Alter Row** assigns insert, update, delete or upsert policies; configure sink keys and permitted operations for those policies to take effect.
- **Schema drift** accepts evolving columns but weakens design-time assumptions. Apply explicit types and validation at curated boundaries. Parameterize data-flow compute from metadata when workload sizes vary substantially.

Data Processing

Data Engineering on Microsoft Azure

KNOWLEDGE CHEATSHEET

FROM INGESTION TO ANALYTICS

Ingest Batch or events → **Transform** SQL or Spark → **Persist** Lake or warehouse → **Serve** Queries and reports

Spark and Delta Lake

- Read data into a **DataFrame**, transform it, then write to the target. **select/filter/groupBy/orderBy** express projections, predicates, aggregates and ordering; cast values explicitly before numeric calculations.
- Parse a JSON string with **from_json** and a schema; use **explode** for array elements and **alias** for output names. Spark **PIVOT** summarizes category values into columns.
- **Delta Lake** adds a transaction log, ACID operations and schema controls to Parquet files. Use append for new rows and **MERGE** for keyed changes; deduplicate the source before an ambiguous multi-match merge.
- A **Type 2 MERGE** needs staged logic to expire changed current rows and insert new versions. A simple matched UPDATE alone preserves no historical row.
- Use **partitionBy** for storage partitions aligned to common filters. Separate append, full overwrite and selective partition replacement deliberately; an unrestricted overwrite can remove data outside the incoming batch.
- Notebook language commands include **%python, %sql, %scala and %r**. Pipeline notebook parameters override values in a designated parameter cell. A Synapse Spark SQL-pool connector can write DataFrames to dedicated SQL tables.

Stream ingestion and windows

- **Event Hubs** ingests event streams; **IoT Hub** adds device-facing capabilities. A **partition key** keeps related events on one partition, with ordering scoped to that partition.
- In Stream Analytics, **stream inputs** carry ongoing events; **reference inputs** provide finite lookup data from supported Blob or SQL sources. Date/time path patterns can select refreshed reference files.
- **TIMESTAMP BY** selects application event time. A **tumbling window** is fixed and nonoverlapping, so an event belongs to one window; use it for distinct interval totals.
- A **hopping window** has a window size and hop interval. Fifteen-minute windows with five-minute hops produce overlapping running summaries. Equal window and hop sizes behave as tumbling windows.
- A **sliding window** produces results when window contents change. A **session window** groups activity separated by an inactivity timeout, subject to its maximum duration.
- Use **geospatial functions** for point, distance and polygon tests. Built-in **AnomalyDetection_SpikeAndDip** and change-point functions analyze time-series anomalies. Unsupported input encodings can require a custom deserializer.

Data Processing

Data Engineering on Microsoft Azure

KNOWLEDGE CHEATSHEET

FROM INGESTION TO ANALYTICS

Ingest Batch or events → **Transform** SQL or Spark → **Persist** Lake or warehouse → **Serve** Queries and reports

Stream state and event time

- Bound **stream-to-stream joins** with a time relationship such as DATEDIFF so state can expire. **LAG/LAST** look back within a duration and partition key; **TopOne** returns a ranked record within its grouping/window.
- Stream Analytics **late arrival** and **out-of-order** policies determine whether timestamps are adjusted or events are dropped. Larger tolerances can increase latency and retained state.
- Spark **Structured Streaming** incrementally maintains a result table. **Append** emits newly finalized rows, **Update** emits changed rows, and **Complete** emits the whole result; actual sink support constrains the choice.
- For event-time aggregations, a Spark **watermark** tracks progress and bounds old state. Append output generally waits for a window to be finalized. It does not guarantee unlimited acceptance of late events.
- Keep a durable **checkpoint location** for offsets and state. **Auto Loader** incrementally discovers cloud files with schema tracking; configure schema evolution and rescued data rather than repeatedly listing and reprocessing every file.

Source control, links and lineage

- With **Git integration**, develop and save in feature branches, review a pull request, merge into the collaboration branch, then **publish**. A saved or merged change does not automatically update scheduled live runs.
- Git can preserve work with validation issues; publishing requires valid artifacts. **adf_publish** holds generated ADF deployment templates by default. **CI/CD** deploys parameterized artifacts to test and production environments.
- **Synapse Link for Cosmos DB** exposes an analytical store for supported SQL/Spark analytics without consuming ordinary transactional query throughput. Enable Link and the relevant container's analytical store.
- Cosmos **transactional TTL** and **analytical TTL** govern different stores. Analytical schema modes handle mixed types differently; do not assume a type change has the same representation as a conventional SQL column.
- **Synapse Link for SQL** supports near-real-time replication into dedicated SQL pools. **Microsoft Purview** scans discover schemas/classifications; supported pipeline executions generate operational lineage. Search the catalog to inspect an asset's schema and upstream/downstream relationships.

Security & Operations

Data Engineering on Microsoft Azure

KNOWLEDGE CHEATSHEET

OPERATE THE DATA PLATFORM

- **Identity** — Authenticate
- **Permissions** — Authorize
- **Telemetry** — Diagnose
- **Tuning** — Improve throughput

03 Secure, monitor, and optimize data storage and processing · 30– 35%

Identities and external credentials

- **Microsoft Entra ID** authenticates users, groups and applications; interactive authentication can enforce MFA. Map groups to contained database users and database roles for maintainable least-privilege SQL access.
- A **managed identity** avoids embedded credentials for service access. A system-assigned identity follows its resource; a user-assigned identity has an independent lifecycle. Grant both data permissions and network reachability.
- **Credential passthrough** uses the signed-in user's identity for storage requests; a **service principal** uses an application identity. Select a supported runtime/access mode and apply storage permissions to the actual requesting principal.
- A **SAS** delegates specific storage operations for a limited scope and interval. Account keys grant broad access; prefer identity-based authorization and keep required secrets in **Key Vault**.
- A **database scoped credential** supplies external authentication for a data source. A database master key can protect stored credential secrets. In serverless SQL, permitted Entra access can also use caller identity passthrough.
- For supported ad hoc serverless access without a data source, a **server-level credential** can supply storage authorization. SQL permission to query and storage permission to read are separate; grant required credential REFERENCES privileges narrowly.

Lake permissions and inheritance

- **Azure RBAC** grants data access at scopes such as account or container. **Storage Blob Data Reader** reads; **Data Contributor** reads/writes/deletes. A management **Contributor** role does not itself grant Entra-based blob data access.
- Use container-scoped roles for whole-container access and **ACLs** for directory/file granularity. A broader RBAC data grant can authorize access before ACL evaluation; ACLs do not subtract permissions already granted by RBAC.
- For ACL-based file access, grant **execute** on the container root and each ancestor directory. **Read** on a file permits reading; **write** permits writing/appending. Directory **read + execute** permits listing and traversal.
- Creating or deleting child items generally requires **write + execute on the parent directory**, with execute on ancestors. File write permission alone does not grant the right to delete its directory entry.

Security & Operations

Data Engineering on Microsoft Azure

KNOWLEDGE CHEATSHEET

OPERATE THE DATA PLATFORM

- **Identity** — Authenticate
- **Permissions** — Authorize
- **Telemetry** — Diagnose
- **Tuning** — Improve throughput

- An **access ACL** controls the existing item. A directory's **default ACL** is a template for newly created descendants; changing it does not retroactively update existing children. Use recursive updates deliberately.
- An ACL **mask** caps effective permissions for named users/groups and the owning group. Assign access to security groups where possible, then check effective permissions, inheritance and every ancestor path.

Network, encryption and secrets

- **Private endpoints** provide private IP connectivity through Private Link. Plan managed virtual networks, endpoint approval, DNS and public-access settings together; identity permissions do not bypass a firewall.
- For public SQL endpoints, **firewall IP rules** can allow an on-premises egress range. Private access from on premises needs a reachable private network path; a virtual network rule alone does not create that path.
- **TDE** encrypts dedicated SQL storage and associated logs/backups at rest without application query changes. **TLS/secure transfer** protects traffic in transit; these solve separate protection requirements.
- A **customer-managed key** gives key-lifecycle control. Synapse workspace double encryption uses a Key Vault key in addition to service-managed encryption; protect the vault with **soft delete and purge protection** and grant the service identity key access.
- Monitor **Key Vault audit logs** and preserve key availability during recovery. Resolve secrets through authorized linked services or notebook utilities; workspace access, Key Vault authorization and storage authorization remain separate checks.

SQL security and sensitive data

- Grant **SELECT ON SCHEMA** to a database role for read-only access to that schema, then add the group-backed database user. Schema permissions cover relevant current and future objects without database-wide read access.
- **Column-level security** grants access only to permitted columns. **Row-level security** uses a predicate function and **CREATE SECURITY POLICY** to filter supported internal-table rows according to execution context.
- Protect external files through **storage permissions** and supported SQL access boundaries. Do not assume an external-table definition supports every security feature of an internal dedicated SQL table.

Security & Operations

Data Engineering on Microsoft Azure

KNOWLEDGE CHEATSHEET

OPERATE THE DATA PLATFORM

- **Identity** — Authenticate
- **Permissions** — Authorize
- **Telemetry** — Diagnose
- **Tuning** — Improve throughput

- **Dynamic data masking** changes query output for users without unmask privileges; stored data remains unchanged. Default, email and partial masks serve presentation needs. Masking alone does not prevent inference by a determined querying user.
- **Sensitivity classifications** label confidential columns; they do not deny access. Combine labels with **SQL auditing** to record who queried sensitive data, and use authorization or appropriately designed tokenization for stronger access restrictions.

Pipeline and Spark observability

- Use pipeline **activity runs** for status, duration, input/output and error details. **Annotations** group/filter pipelines by purpose; user properties expose useful run context such as a source table.
- **Diagnostic settings** export pipeline, activity and trigger logs to Log Analytics for KQL, trends and longer retention. Configure retention before the required history expires; subscription Activity Log does not replace execution logs.
- With resource-specific ADF tables, query **ADFPipelineRun**, **ADFActivityRun** and **ADFTriggerRun** for their respective levels. Legacy destination mode uses **AzureDiagnostics**. Inspect the selected table schema before writing queries.
- Separate copy/data-flow **queue time**, transformation time and **sink processing time**. A PathNotFound error points to path generation, timing or absent input; extra retries cannot create a missing source file.
- Use Synapse Studio **Monitor > Apache Spark applications** and the Spark UI for jobs, stages and executors. Databricks cluster events and driver/executor/init-script logs diagnose execution failures; **audit logs** track cluster and job management actions.
- Route supported application metrics and logs to **Log Analytics** using authorized monitoring integration. Build log/metric alerts and attach an **action group** for notifications or automated response.

Warehouse performance diagnostics

- Use **sys.dm_pdw_exec_requests** for request state and duration, **sys.dm_pdw_request_steps** for execution steps, and **sys.dm_pdw_waits** for resource waits. Grant **VIEW DATABASE STATE** where required for monitoring.
- Inspect **sys.dm_pdw_nodes_db_partition_stats** or **DBCC PDW_SHOWSPACEUSED** in the dedicated pool for uneven row distribution. **sys.dm_pdw_nodes_tran_database_transactions** helps investigate rollback and transaction-log activity across nodes.
- Check **sys.pdw_nodes_column_store_row_groups** for total/deleted rows and rowgroup quality. **ALTER INDEX ... REBUILD** can improve fragmented columnstore storage; ensure load batches and memory also support dense rowgroups.
- Maintain **statistics** after major loads or distribution changes. **Materialized views** precompute supported joins/aggregates; **result set caching** reuses eligible repeated query results and is invalidated by relevant changes.

Security & Operations

Data Engineering on Microsoft Azure

KNOWLEDGE CHEATSHEET

OPERATE THE DATA PLATFORM

- **Identity** — Authenticate
- **Permissions** — Authorize
- **Telemetry** — Diagnose
- **Tuning** — Improve throughput

- Use **result_cache_hit** in execution requests to check result reuse. **Cache hit percentage/used percentage** describe the adaptive columnstore cache, a separate layer; they do not directly report result set cache hits.
- Larger **resource classes** allocate more memory per query but reduce concurrency. Evaluate CPU, I/O, tempdb, queueing and cache alongside **DWU** utilization before scaling; workload groups/classifiers can isolate important loads.

Spark and Delta optimization

- **Cache/persist** datasets reused by multiple Spark actions when they fit the chosen storage level. Inspect the Spark UI for skewed tasks, shuffle volume and spills before adding workers.
- **OPTIMIZE** compacts Delta files. **Z-Ordering** colocates related values to improve data skipping; **dynamic file pruning** can skip files during selective joins. Partition pruning, file skipping and caching address different costs.
- Use **job compute** for scheduled batch runs and **all-purpose compute** for interactive work. **Autoscaling** adjusts workers; **auto-termination** stops idle compute. Instance **pools** reduce startup delays but idle instances can still incur cloud costs.
- Keep compute configuration reproducible. **Pinning** an eligible terminated all-purpose cluster preserves its configuration without requiring it to keep running. Match runtime, access mode and library support to the workload.

Streaming capacity and recovery

- Monitor Stream Analytics **SU utilization**, **Backlogged Input Events** and **watermark delay** together. Increasing backlog or delay can reflect insufficient processing, input throttling or an underprovisioned output sink.
- Separate **deserialization errors**, output errors and late/out-of-order event counts from capacity problems. Fix malformed inputs or sink failures before assuming that additional streaming units will restore throughput.
- Preserve compatible partition keys through **input, query and output** for end-to-end parallelism. Repartition with **PARTITION BY** when needed; explicitly repartitioned stream joins need compatible keys and partition counts.
- Event Hubs **partitions** enable parallel consumers; throughput/capacity units still bound ingress and egress. **Auto-inflate** raises provisioned throughput within its configured limit. Partition-count changes depend on service tier and can alter key routing.
- For regional streaming resilience, deploy independently recoverable processing in separate regions and plan input/output recovery. Use **consumer groups**, checkpoints and idempotent sinks to control replay and duplicate processing.
- A dedicated SQL **restore point** restores a new pool; resume a paused source when an active operation is required. Pause an unused restored copy to reduce compute cost while retaining its storage; pausing is not a backup strategy.