Unified Enterprise Logging Strategy

1. Core Principles

- **Centralized**: All application logs flow into a single logging platform.
- **Structured**: Logs use structured formats (JSON) instead of free text.
- **Searchable & Understandable**: Easy querying for developers, BAs, and managers.
- **Scalable & Performant**: Handles high log volume with minimal application overhead.
- **Flexible**: Works across Java, C#, Kubernetes, and future platforms.

2. Recommended Architecture

Applications → Log Agents → Central Log Platform → Dashboards & Alerts

a. Application Logging (Producers)

- **Java (Kubernetes)**
  - Use **Logback / Log4j2** with **JSON logging**.
  - Log to **stdout/stderr** only (Kubernetes best practice).
- **C# (.NET)**
  - Use **Serilog** or **NLog** with JSON output.
  - Stop logging directly into SQL Server tables.

Common Log Schema (important):

- `timestamp`
- `logLevel`
- `serviceName`
- `environment`
- `traceId / correlationId`

- `userId / businessId` (where applicable)
- `message`
- `errorStack` (if any)

b. Log Collection (Agents)

- Deploy **Fluent Bit** or **Filebeat**:
  - As a **DaemonSet** in Kubernetes.
  - As a lightweight service on Windows servers for C# apps.
- Responsibilities:
  - Parse JSON logs
  - Enrich logs (host, pod, namespace, app version)
  - Forward logs reliably (buffering + backpressure)

c. Central Log Platform (Single Source of Truth)

Industry-standard options:

- **ELK / OpenSearch stack** (Elasticsearch/OpenSearch + Kibana/OpenSearch Dashboards)
- **Splunk** (enterprise, higher cost)
- **Cloud-native** alternatives (Azure Monitor / Log Analytics if on Azure)

Capabilities:

- Full-text and structured search
- Time-range filtering
- Aggregations (counts, trends, error rates)
- Role-based access control

3. Observability Enhancements (Best Practice)

- **Correlation IDs**
    - Generate once per request and propagate across services.
    - Enables end-to-end tracing in logs.
- **Log Levels Discipline**
    - `INFO`: business-relevant events
    - `WARN`: recoverable issues
    - `ERROR`: production failures
    - Avoid excessive `DEBUG` in prod
- **Retention Policy**
    - Hot logs (7–14 days)
    - Warm/archive logs (30–90 days)
    - Automatic rollover to control cost

4. Usability for Different Stakeholders

Developers

- Advanced query language
- Stack traces
- Correlated logs across services

Business Analysts (BAs)

- Predefined dashboards:
    - Failed transactions
    - Processing time by business flow
    - Volume trends
- Human-readable business fields (orderId, customerId)

Managers

- High-level KPIs:
    - Error rate
    - System health
    - SLA/SLO indicators
- Alerts instead of raw logs

5. Performance & Scalability

- Async, non-blocking logging in apps
- No direct DB writes from applications
- Horizontal scaling of log collectors and storage
- Backpressure handling to protect applications during spikes

6. Migration Strategy (Practical)

1. Define **common log schema** and standards.
2. Enable JSON logging in Java and C# apps.
3. Deploy Fluent Bit/Filebeat.
4. Stand up central log platform.
5. Gradually decommission SQL Server–based logging.
6. Build shared dashboards and alerts.

7. Outcome

- One centralized, searchable logging system
- Faster production issue investigation
- Reduced operational overhead
- Logs that serve **developers, BAs, and managers** equally well
- Future-proof, cloud-native logging architecture

If you want, I can:

- Tailor this to **Azure-only**, **on-prem**, or **hybrid**
- Provide **sample JSON log formats**
- Recommend **exact tools** based on cost and scale