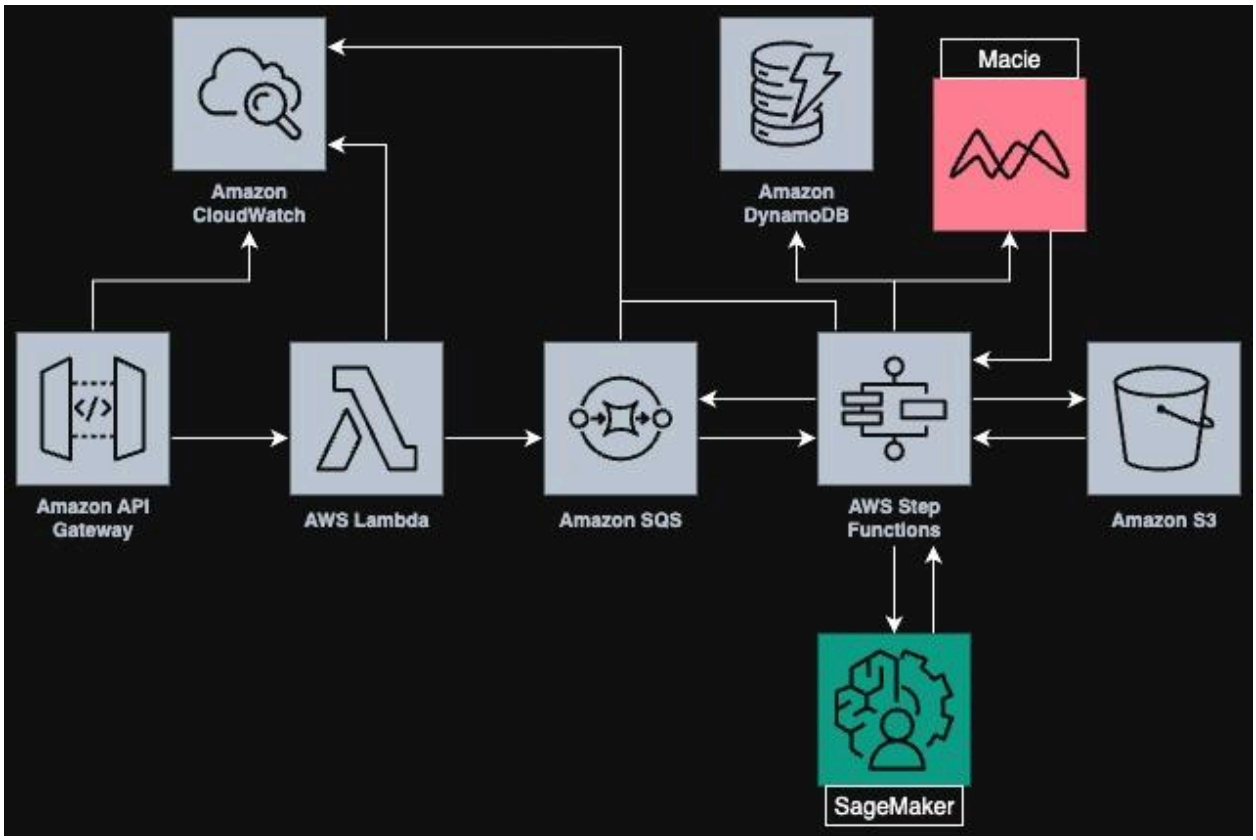
System Design: Extraction Service

a. High-Level Architecture

The system is a serverless, queue-based pipeline deployed in AWS GovCloud, using managed services and open-source LLMs to ensure compliance and scalability. Key components:

- **API Gateway (REST API):** Entry point for job submissions (e.g., POST /jobs with config and source path). Authenticated via IAM roles for multi-tenancy.
- **AWS Lambda (Job Manager):** Handles job creation, validation, and queueing. Stores job metadata in DynamoDB.
- **Amazon SQS (Job Queue):** Decouples job submission from processing. Standard queue for extraction tasks, dead-letter queue for failures.
- **AWS Step Functions:** Orchestrates the workflow (crawl → extract → deliver). Ensures idempotency and state tracking.
- **AWS S3 (Storage):** Stores input docs (S3/SharePoint sync), output JSON, and logs. Encrypted with SSE-KMS.
- **Amazon SageMaker (LLM Inference):** Hosts open-source LLM (e.g., BERT or LLaMA from Hugging Face) for metadata extraction. Deployed in GovCloud with custom endpoints.
- **AWS Macie:** Scans for PII before processing; flags sensitive docs for review.
- **Amazon CloudWatch & X-Ray:** Metrics, logs, and traces for observability. Custom dashboards for job status.
- **AWS SDK (SharePoint Connector):** Crawls SharePoint via GovCloud-compatible APIs (assumes OAuth or federated access).

Architecture Diagram (Conceptual):



Rationale: Serverless minimizes ops overhead and scales automatically. SQS/Step Functions ensure reliable, idempotent processing. SageMaker supports open-source LLMs, avoiding third-party APIs. All services are GovCloud-compliant.

b. Data Model

Job Metadata (DynamoDB):

json

```
{
 "job_id": "uuid-1234", // Partition key
 "tenant_id": "agency-xyz", // Sort key for multi-tenancy
 "status": "pending | running | completed | failed",
 "config": {
   "fields": ["author_names", "publish_date", "abstract_summary",
"code_snippets"],
   "source_type": "s3 | sharepoint",
   "source_path": "s3://bucket/prefix | https://sharepoint-url",
   "output_path": "s3://bucket/output"
 },
```

```
 "created_at": "2025-09-18T10:02:00Z",
 "updated_at": "2025-09-18T10:03:00Z",
 "doc_count": 1000,
 "failed_docs": ["doc1.pdf", "doc2.pdf"]
}
```

Output JSON (S3 per doc):

json

```
{
 "doc_id": "doc1.pdf",
 "job_id": "uuid-1234",
 "tenant_id": "agency-xyz",
 "metadata": {
   "author_names": ["John Doe", "Jane Smith"],
   "publish_date": "2023-01-01",
   "abstract_summary": "This paper discusses AI advancements...",
   "code_snippets": ["def example(): ..."],
   "pii_flagged": false
 },
 "status": "success | failed",
 "error": "null | parsing_failed"
}
```

Rationale: DynamoDB for low-latency job tracking; S3 for scalable, durable output storage. JSON schemas are simple, extensible, and normalized for downstream use.

c. Sequence Diagrams

1. Create Job → Crawl → Extract → Deliver Results

1. User via API Gateway submits POST /jobs with config and source path.
2. Lambda job manager validates request, stores job in DynamoDB, pushes to SQS.
3. Step Functions picks up job, triggers Lambda (Crawler) to list docs from S3/SharePoint.
4. For each doc:
    - Macie scans for PII; flags if sensitive.
    - Lambda pulls doc from S3/SharePoint.
    - SageMaker extracts metadata based on config.
    - Lambda saves JSON to S3 output path.
5. Step Functions updates job status in DynamoDB.
6. API Gateway returns job status or notifies user via SNS email.

2. Retries, Backoff, Idempotency

1. If extraction fails, SQS retries 3x with exponential backoff of 1s, 2s, 4s.
2. Step Functions ensures idempotency via job_id/doc_id checks.
3. Failed docs move to dead-letter queue; logged in CloudWatch.
4. Lambda retry handler analyzes failures, updates DynamoDB, alerts if threshold exceeded.

d. Scaling Strategy

- **100k Docs:** Process in batches. SQS handles backpressure by buffering tasks.
- **Concurrency:** Lambda auto-scales up to GovCloud limits, ~1000 concurrent. SageMaker endpoints scale via instance count.
- **Backpressure:** SQS visibility timeout prevents overload; Step Functions throttles if SageMaker hits rate limits.
- **Rate Limits:** API Gateway throttles at 100 req/s; configurable per tenant.
- **Cost Caps:** Monitor via CloudWatch Budgets; cap SageMaker instances at 10 to stay under $1k/100k docs.
- **Optimization:** Pre-warm SageMaker endpoints; use spot instances for non-critical jobs.

e. Observability & Ops Runbook

Observability:

- **Metrics via CloudWatch:** Processed docs, succeeded/failed, p50/p95 latency.
- **Logs:** CloudWatch Logs for Lambda, SageMaker, Step Functions. Structured JSON with job_id/doc_id.
- **Traces:** X-Ray for end-to-end request tracking.
- **Dashboards:** CloudWatch Dashboard showing throughput, error rate, cost, and job status.

Runbook:

- **SLOs:** 99% API uptime, <1% failed extractions, <24h for 100k docs.
- **Alerts:** CloudWatch Alarms for >1% error rate, >$500 spend, or >5min p95 latency.
- **Actions:**
    - High error rate → Check DLQ, scale SageMaker instances.
    - Cost overrun → Reduce batch size, use spot instances.
    - Job stuck → Restart Step Functions state machine; inspect logs.