1. Exercise Overview

Build a containerized backend composed of multiple FastAPI services and an AWS Lambda function. The system must support:

1. **PDF Service**: PDF upload, text extraction, and metadata storage.
2. **RAG Module**: Document-chunking, Pinecone indexing, and query endpoint (including agent-metrics submission).
3. **AWS Service**: DynamoDB + S3 CRUD, invoking the RAG Module.
4. **Agent-Metrics Lambda**: Receives and stores query metrics in DynamoDB.

All parts should run together (e.g., via Docker Compose). Candidates are responsible for all implementation details, including authentication, error handling, and cloud configuration.

---

2. Repository s Project Structure

Create a repository named backend-recruitment-exercise, organized into the following folders (implementations to be written by the candidate):

- **pdf_service/**
  Handles PDF uploads, text extraction, and metadata.
- **rag_module/**
  Handles document chunking, embedding, Pinecone indexing, RAG queries, and metrics submission.
- **aws_service/**
  Handles DynamoDB + S3 CRUD for document metadata and invokes the RAG Module's endpoints.
- **metrics_lambda/**
  Contains the AWS Lambda package that writes agent-metrics to DynamoDB.
- **tests/**
  Unit tests for each component.
- **docker-compose.yml**
  Orchestrates all services (and LocalStack if used).
- **README.md**
  Instructions for setup, environment, and usage.

---

3. PDF Service

Purpose:
Allow users to upload PDFs, extract their text, and store both raw files and metadata.

High-Level Requirements:

- Store raw PDFs (locally or in S3).

- Extract full text from each uploaded PDF.
- Store extracted text and metadata (e.g., doc_id, filename, upload timestamp).

Endpoints (base path: /pdf/):

1. **POST /pdf/upload**
    - Accept one or more PDF files.
    - Return a JSON array of newly created doc_id values and associated metadata.
2. **GET /pdf/documents/{doc_id}**
    - Return metadata and extracted text for the specified doc_id.
3. **GET /pdf/documents**
    - Accept pagination parameters (page, limit).
    - Return a paginated list of all document metadata.

---

4. RAG Module

Purpose:
Manage document chunking, embed those chunks in Pinecone, perform similarity-based retrieval, generate answers via an LLM, and send query metrics to the Lambda.

High-Level Requirements:

- Break existing documents into chunks of configurable size.
- Create embeddings for each chunk and upsert them into a Pinecone index.
- Expose an endpoint to index one or multiple document_id values.
- Expose a query endpoint to:
    - Accept a set of document_ids and a question.
    - Retrieve relevant chunks from Pinecone, invoke an LLM to generate an answer, compute metrics (e.g., tokens consumed/generated, response time, confidence), and send those metrics to the Lambda.
    - Return the generated answer along with the computed metrics.

Endpoints (base path: /rag/):

1. **POST /rag/index**
    - Request body: JSON list of one or more document_id values.
    - Indexes each document's chunks into Pinecone.
    - Responds with per-document status (success/failure).
2. **POST /rag/query**

- Request body: JSON containing:

```
json

CopyEdit

{

 "document_ids": ["<doc_id1>", "<doc_id2>", ...],

 "question": "<text>"

}
```

- Returns a JSON object containing:
  - run_id (unique ID for this query)
  - answer (LLM output)
  - tokens_consumed
  - tokens_generated
  - response_time_ms
  - confidence_score
- Internally, this endpoint must also send the metrics payload to the Agent-Metrics Lambda.

---

5. AWS Service

Purpose:
Manage document metadata in DynamoDB, store or reference raw PDFs in S3, and forward indexing/query requests to the RAG Module.

High-Level Requirements:

- Maintain a DynamoDB table (DocumentsMetadata) keyed by doc_id, storing filename, upload_timestamp, and any optional fields.
- Upload raw PDFs to an S3 bucket and store the S3 key or URL in DynamoDB.
- Forward indexing and query requests to the RAG Module's endpoints.

Endpoints (base path: /aws/):

1. **POST /aws/documents**
   - Request body: JSON with required fields (e.g., doc_id, filename, optional tags).
   - Creates an item in DynamoDB's DocumentsMetadata table.
2. **GET /aws/documents/{doc_id}**
   - Returns the DynamoDB item for doc_id.

3. **PUT /aws/documents/{doc_id}**
    - Request body: JSON with fields to update (e.g., tags).
    - Updates the DynamoDB item for doc_id.
4. **DELETE /aws/documents/{doc_id}**
    - Deletes the item from DynamoDB.
    - Optionally deletes the corresponding PDF from S3.
5. **POST /aws/documents/{doc_id}/index**
    - Triggers the RAG Module's POST /rag/index for that doc_id.
    - Returns a status indicating success or failure.
6. **POST /aws/query**
    - Request body: JSON containing document_ids and question.
    - Forwards the request to the RAG Module's POST /rag/query endpoint.
    - Returns the RAG Module's response (answer + metrics).

---

6. Agent-Metrics Lambda

Purpose:
Receive query metrics (tokens, response time, confidence, etc.) and store them in a DynamoDB table (AgentMetrics).

High-Level Requirements:

- Create a DynamoDB table named AgentMetrics with:
    - **Partition Key:** run_id (string)
    - **Sort Key:** timestamp (string, ISO 8601 UTC)
    - Attributes: agent_name, tokens_consumed, tokens_generated, response_time_ms, confidence_score, status, and any additional metadata.
- Build a Python-based AWS Lambda function (store_agent_metrics) that:
    - Receives an event containing a JSON payload with:
        - run_id
        - agent_name
        - tokens_consumed
        - tokens_generated
        - response_time_ms
        - confidence_score

        - status
    - Writes a new item to the AgentMetrics table using run_id as the partition key and the current UTC timestamp as the sort key.
    - Returns a JSON response indicating success or failure.
- The Lambda must have an IAM role allowing dynamodb:PutItem on the AgentMetrics table.

---

7. Dockerization s Deployment

Purpose:
Containerize each FastAPI service and (optionally) the Lambda package, so they can run together via Docker Compose (or separately).

High-Level Requirements:

1. **PDF Service Dockerfile**
    - Builds the FastAPI app that listens on port **8000**.
    - Installs dependencies and starts Uvicorn.
2. **RAG Module Dockerfile**
    - Builds the FastAPI app that listens on port **8001**.
    - Installs dependencies (Pinecone SDK, LLM SDK, HTTP client).
    - Reads environment variables for PINECONE_API_KEY, PINECONE_ENV, LLM_API_KEY, CHUNK_SIZE, TOP_K, METRICS_LAMBDA_URL.
3. **AWS Service Dockerfile**
    - Builds the FastAPI app that listens on port **8002**.
    - Installs AWS SDK dependencies (e.g., boto3).
    - Reads environment variables for AWS_REGION, DYNAMODB_TABLE_DOCUMENTS, S3_BUCKET.
4. **Metrics Lambda Packaging**
    - Use a build process (e.g., a multi-stage Dockerfile) to install dependencies into a package directory and create a function.zip.
    - Alternatively, candidates may deploy the Lambda directly to AWS without containerization, but must document the process.
5. **Docker Compose (Optional)**
    - Define services for pdf_service, rag_module, aws_service, and a local stub for metrics_lambda (if not deploying to AWS).

- Optionally include a LocalStack service to simulate DynamoDB, S3, and Lambda for local testing.
- Assign ports:
  - pdf_service → 8000
  - rag_module → 8001
  - aws_service → 8002
  - metrics_lambda → 9000 (if local)
  - localstack → 4566

---

8. Testing s Validation

Unit Tests (using pytest or similar)
Candidates must supply tests covering:

1. **PDF Service**
   - Upload a sample PDF, verify that a valid doc_id is returned.
   - Retrieve metadata and extracted text for that doc_id.
2. **RAG Module**
   - Test chunking logic on a long text: ensure correct chunk sizes/counts.
   - Test embedding logic by mocking Pinecone or the embedding API.
   - Test the /rag/query endpoint by stubbing Pinecone and the LLM: verify that the response includes run_id, answer, tokens_consumed, tokens_generated, response_time_ms, and confidence_score.
   - Verify that an HTTP call is made to the Lambda's URL with the correct metrics payload.
3. **AWS Service**
   - Test CRUD operations against DynamoDB (using moto or LocalStack):
     - Create, retrieve, update, and delete items in DocumentsMetadata.
   - Test that POST /aws/documents/{doc_id}/index invokes the RAG Module's /rag/index endpoint (mock the HTTP call).
   - Test that POST /aws/query invokes the RAG Module's /rag/query endpoint and returns its result.
4. **Metrics Lambda**
   - Test that, given a valid JSON event, the Lambda writes one item to AgentMetrics (using moto or LocalStack to mock DynamoDB).

Manual Testing (Postman or cURL)
Candidates should include sample requests in their README for:

1. **PDF Service**
    - POST /pdf/upload (multipart PDF upload).
    - GET /pdf/documents/{doc_id}.
    - GET /pdf/documents?page=1Climit=10.
2. **RAG Module**
    - POST /rag/index with a list of document_ids.
    - POST /rag/query with document_ids + question.
    - Verify the returned JSON includes run_id, answer, tokens_consumed, tokens_generated, response_time_ms, and confidence_score.
    - Confirm that the metrics Lambda receives the correct payload.
3. **AWS Service**
    - POST /aws/documents with JSON containing doc_id, filename, and optional tags.
    - GET /aws/documents/{doc_id}.
    - PUT /aws/documents/{doc_id} to update fields.
    - DELETE /aws/documents/{doc_id}.
    - POST /aws/documents/{doc_id}/index.
    - POST /aws/query with document_ids + question.
4. **Metrics Lambda (Local)**
    - POST http://localhost:9000/metrics (or AWS URL) with:

json

CopyEdit

```
{
 "run_id": "sample-run-1",
 "agent_name": "RAGQueryAgent",
 "tokens_consumed": 50,
 "tokens_generated": 20,
 "response_time_ms": 150,
 "confidence_score": 0.90,
 "status": "completed"
```

}

- Verify that an item appears in the AgentMetrics table.

G. Evaluation Criteria

Area	Weight	Details
PDF Service	15%	PDF upload, text extraction, pagination, and metadata storage.
RAG Module (Pinecone + Metrics)	25%	Document chunking, embedding, Pinecone upserts, similarity search, LLM integration, correct metrics calculation and forwarding to Lambda.
AWS Service (DynamoDB + S3 + RAG Orchestration)	20%	Proper table schema, S3 usage, IAM configuration, invoking RAG Module from AWS endpoints.
Metrics Lambda (DynamoDB Integration)	15%	Lambda correctness in storing metrics into DynamoDB with the proper schema.
Dockerization s Deployment	10%	Each service has a Dockerfile, can run via Compose, LocalStack or real AWS integration.
Testing s Documentation	10%	Unit tests exist and pass, README clearly explains setup, environment, and usage.
Total	**100%**	

- **Error Handling:** All endpoints should return appropriate HTTP status codes (e.g., 400 for bad input, 404 for missing resources).
- **Code Quality:** Clean, idiomatic Python; clear separation of concerns.
- **Security:** No hardcoded secrets; all API keys and AWS credentials must be provided via environment variables.
- **Git Practices:** Meaningful, atomic commits; clear branch strategy.

---

10. Submission Instructions

1. **Repository Link**
   Push your solution to a public or private repository and share the link. Ensure all code, configurations, and instructions are included.
2. **README.md**
   At the repository root, include:
   - **Prerequisites:** Required Python version, Docker, AWS CLI, etc.
   - **Environment Variables:**

        - **pdf_service:** Variables for PDF storage (e.g., local path or S3 endpoint).
        - **rag_module:** PINECONE_API_KEY, PINECONE_ENV, LLM_API_KEY, CHUNK_SIZE, TOP_K, METRICS_LAMBDA_URL.
        - **aws_service:** AWS_REGION, DYNAMODB_TABLE_DOCUMENTS=DocumentsMetadata, S3_BUCKET=<bucket-name>, AWS credentials or LocalStack endpoints.
        - **metrics_lambda:** METRICS_TABLE=AgentMetrics, AWS credentials or LocalStack endpoint.
    - **Service Setup:** For each folder (pdf_service, rag_module, aws_service, metrics_lambda), describe:
        - How to build and run using Docker.
        - How to configure and start LocalStack (if used).
        - How to set up real AWS resources:
            - Create DynamoDB tables (DocumentsMetadata, AgentMetrics) with correct key schemas.
            - Create an S3 bucket.
            - Deploy the Lambda function and obtain its API Gateway URL.
    - **Docker Compose:** Commands to start all services (e.g., docker-compose up --build).
    - **Example Requests:** For each endpoint, provide at least one cURL or Postman example.
3. **Tests**
    - Place tests in a tests/ folder. Name files using the test_*.py convention.
    - Provide instructions in README on how to run them (e.g., pytest --maxfail=1 -q).
4. **Demo Evidence**
    - Include a short text walkthrough in the README with screenshots or logs demonstrating each major workflow:

1. PDF upload → text extraction → retrieval.
2. RAG indexing → query → metrics verification.
3. AWS Service CRUD → indexing → query.
    - Optionally, provide a 2–3 minute unlisted video link.

---

11. Note to Candidates

You have full freedom to choose your own names, folder structures, and implementation details. Feel free to replace or extend any component (e.g., adding new endpoints, experimenting with different embedding models, improving deployment scripts) to highlight your creativity and skills. As long as all core requirements are met, you are encouraged to demonstrate your unique approach and additional knowledge.

For submission C queries: careers@penteai.com | Video: Unlisted in YouTube or drive link