Building Enterprise-Grade AI Systems Solo with LLMs and AWS

Soujanya Vullam

https://www.linkedin.com/in/soujanyavullam/
Presented at Ai4 2025, MGM Grand, Las Vegas

August 11–13, 2025

Abstract

Can a seasoned solo developer build an enterprise-grade AI system that is secure, scalable, and cost-effective? This paper answers this question through the case study of Epic Q&A, an AI-powered platform that transforms large books, documentation, and code repository into interactive, context-aware Q&A experiences.

Epic Q&A combines LLM capabilities with high-performance semantic search and a fully serverless stack all designed using the AWS Well-Architected Framework principles. The result is a production-ready system with minimal operational overhead and pay-per-use cost efficiency.

This work demonstrates how modern cloud native services, AI-assisted development tools such as Cursor, and disciplined architectural choices can empower a single developer to deliver secure, scalable, and cost-optimized AI systems ready for real-world use.

1 Introduction

In the world of learning and information exchange, whether for a student, a new hire, a professional, or any curious mind, questions are the gateway to understanding. The ability to answer those questions instantly, accurately, and at scale has long been the promise of AI. Today, advances in Large Language Models (LLMs) and cloud-native architectures make this promise achievable.

Epic Q&A was built to turn that potential into a practical reality. What makes this project distinctive is not just its functionality, but its development journey, conceived, architected, and delivered entirely by a single developer. The goal was never to undermine the value of teamwork, but to explore how autonomy, when used effectively, can accelerate innovation and productivity. Healthy collaboration remains essential in any business setting; different perspectives and shared execution strengthen outcomes, but this work demonstrates that with the right tools and disciplined design, a single contributor can also achieve enterprise-grade results.

From the outset, the objectives were clear:

- Go beyond the prototype: evolve from a minimal viable product to a robust, enterprise-grade system.
- Keep costs low: adopt a pay-per-use model and maximize infrastructure efficiency.
- Build for scale and security from day one: ensure that the system could grow, remain resilient, and meet enterprise standards without slowing innovation.

The following sections detail the architectural decisions, performance optimizations, and AI-augmented workflows that made this possible, offering a practical blueprint for anyone looking to build production-grade AI systems with limited resources.

2 What Makes a System Enterprise-Grade

An enterprise-grade system goes beyond simply 'working'. It must be robust, scalable, secure, and maintainable enough to handle real-world business demands on a scale, while remaining adaptable to evolving requirements. For Epic Q&A, every architectural decision was based on these principles:

2.1 Modularity

Breaking the system into loosely coupled independent components, such as authentication, file processing, semantic search, and answer generation, allows independent development, testing, and deployment. This modular design accelerates iteration, reduces risk, and enables the selective scaling of high-demand components without affecting others.

2.2 Scalability

The system must handle increases in workload, from a handful of users to thousands of concurrent requests, seamlessly, without manual intervention. By adopting serverless auto-scaling (AWS Lambda, API Gateway) and a global CDN (CloudFront) for static content delivery, Epic Q&A can elastically adapt to spikes in traffic while maintaining low latency worldwide.

2.3 Reliability

Reliability is built into every layer: fault-tolerant services, automatic retries for transient failures, data redundancy, and graceful degradation when components are under stress. AWS managed services such as S3, DynamoDB and S3 Vectors provide 99.9%+ availability guarantees, ensuring the platform remains operational under varying conditions.

2.4 Security

Security is non-negotiable in enterprise systems. Epic Q&A implements a multi-layered defense strategy, including: Encryption of data at rest and in transit. Robust authentication and authorization via Amazon Cognito and IAM least privilege policies. Network isolation for backend services to protect sensitive data from unauthorized access. These measures meet the AWS Well-Architected Framework Security pillar guidelines and align with compliance standards.

2.5 Flexibility

The architecture is extensible by design, avoiding vendor lock-in where possible. For example, while AWS Bedrock is the current LLM provider, the modular code-base of the system allows for the swapping to another LLM API or vector database with minimal changes. This adaptability ensures longevity as AI and cloud technologies evolve.

2.6 Reusability

Reusable CloudFormation templates, Lambda layers, and shared service patterns ensure consistency across features and speed up new development. This approach allows the same architectural patterns to support future AI applications, from document search to recommendation engines.

2.7 Cost-Effectiveness

The pay-per-use serverless model ensures that infrastructure costs directly reflect usage, eliminating idle server expenses. Techniques such as CDN caching and S3 lifecycle policies further optimize costs, while the migration to S3 Vectors reduced vector storage and query expenses by ~90%.

2.8 Maintainability

Infrastructure as Code (IaC) with CloudFormation enables reproducible deployments and simplifies change management. Strong typing in both back-end (Python with type hints) and front-end (TypeScript in React) reduces runtime errors and makes long-term maintenance predictable.

2.9 Testability

Each microservice is testable in isolation, supported by unit tests, integration tests, and mocks for AWS services. This ensures rapid feedback during development and minimizes production defects.

2.10 Observability in AI Systems

Observability goes beyond infrastructure monitoring; it is about understanding the internal state of AI models in production. By collecting and analyzing metrics, logs, and traces, we can evaluate the following.

- Decision-making patterns of AI models.
- Performance trends over time.
- Compliance and adherence to security in model outputs.
- CloudWatch metrics and dashboards track key operational indicators, while tools like TensorBoard capture model performance data. This ensures that the system can be continuously improved with data-driven insights.

2.11 Evals for AI

Unlike traditional software testing with deterministic pass/fail criteria, AI Evals assess performance across multiple qualitative and quantitative dimensions, including:

- Accuracy – correctness of answers.
- Relevance – contextual appropriateness of responses.
- Coherence – logical flow of generated answers.
- Completeness – coverage of relevant points.
- Helpfulness & Safety – user value and risk avoidance.

3 Use Case: Epic Q&A

Epic Q&A enables:

- Uploading books, documents, and code repositories.
- Automatic text chunking and embedding generation.
- Semantic search using vector similarity.
- LLM-powered answer synthesis with retrieved context.

4 Architecture Diagram

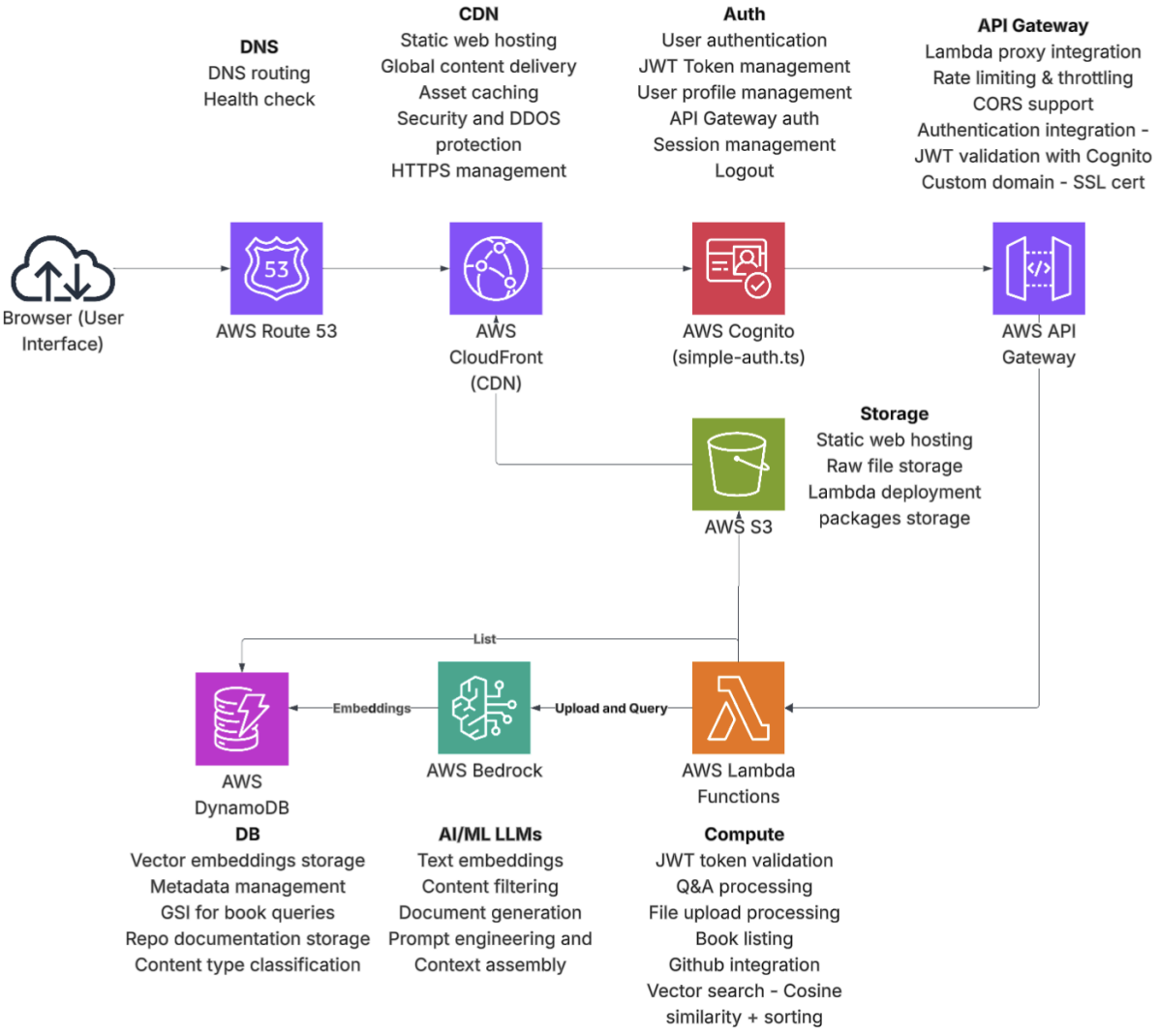

Figure 1: Architecture Diagram

Figure 1 illustrates the end-to-end workflow, from user request to final response generation.

When a user logs in to the Epic Q&A web application, the request is first routed through **Amazon Route 53**, which directs traffic to the **CDN**. CDN delivers static front-end assets and caches frequently accessed content, providing low-latency global access.

The front-end then authenticates the user via **Amazon Cognito**, issuing a JWT token that is subsequently attached to all API requests. **Amazon API Gateway** serves as the entry point for back-end services, validating JWT tokens against Cognito to enforce secure access control. Based on the type of request, the API Gateway triggers the appropriate **AWS Lambda** function.

The system supports three primary flows:

- **Upload flow:** A Lambda function ingests raw files or public GitHub repositories into **Amazon S3**, processes text by chunking, generates embeddings using **AWS Bedrock**, and stores the embeddings in **Amazon DynamoDB**.
- **List flow:** A Lambda function retrieves metadata of available books or application documentation from S3 and returns the results to the user interface.
- **Query flow:** A Lambda function retrieves embeddings from DynamoDB and performs vector similarity search using cosine similarity. Relevant context is passed to AWS Bedrock, which generates a context-grounded answer that is returned to the front-end through API Gateway.

Summary

- Authentication: Cognito issues JWT tokens.
- Routing: CDN delivers UI assets; API Gateway routes API calls.
- Processing: Lambda functions handle upload, query, and listing.
- Vector storage: DynamoDBs
- Answer Generation: AWS Bedrock retrieves context and generates responses.

This architecture demonstrates how a serverless modular approach can deliver secure authentication, scalable processing, and efficient query handling while adhering to enterprise-grade design principles.

5 Architectural Design Decisions

As a solo developer or a team, making strategic architectural choices was critical to building a system that could scale efficiently, remain secure, and minimize operational overhead. Each service in Epic Q&A was selected to align with the principles of enterprise-grade system design.

5.1 AWS Bedrock

Purpose: Generate embeddings for semantic search and produce context-grounded Q&A responses. Rationale & Benefits:

- Provided access to high-performance LLMs without the need for custom training or hosting models.
- Auto-scaling to handle variable workloads.
- Five nines of availability for reliability.[11]

- Fully managed, removing operational burden. [2]
- Pay-per-call model ensuring cost efficiency.

Validation: Bedrock was the optimal choice for implementing an LLM-based semantic search while avoiding the complexity and cost of managing the custom AI infrastructure.

5.2 Amazon S3 + CloudFront (CDN)

Purpose: Store raw text files, processed content, and static front-end assets while ensuring fast global delivery. Rationale & Benefits:

- Unlimited, reliable storage for books, documents, and code.
- Low-latency global access via CloudFront's edge locations.[16]
- DDoS protection and built-in redundancy.[13]
- Pay-per-use model, keeping storage and bandwidth costs predictable.
- Fully managed services that do not require infrastructure maintenance.

Validation: This combination follows industry best practice for serverless web applications, providing speed, reliability, and cost efficiency for a globally distributed user base.

5.3 Amazon API Gateway + AWS Lambda

Purpose: Serve as the REST API layer for querying, uploading, and listing endpoints, and integrate with Bedrock for augmented retrieval generation. Rationale & Benefits:

- Event-driven auto-scaling scales from zero to thousands of concurrent requests.
- No idle cost, pay only for execution time.[14]
- Supports asynchronous file uploads and background processing.
- Python Lambdas leverage a rich library ecosystem for text processing, data transformation, and AWS service integration.

Validation: A proven event-driven architecture that minimized operational overhead and provided flexibility for future feature expansion.

5.4 Amazon DynamoDB

Purpose: Initially used for storing embeddings for semantic search and managing metadata such as document titles and user information. Rationale & Benefits:

- High-speed key-value lookups for metadata retrieval.[15]
- Five nines uptime and fully managed with pay-per-use billing. [12]
- Aligned with availability, cost efficiency, and maintainability requirements at the start of the project.

Limitation: During load testing, CloudWatch metrics revealed high query latency (1-3 seconds) for large documents such as Indian epics (sizes upto 6MB in txt format). This was due to the lack of optimization of DynamoDB for the search for high-dimensional similarity.

6 Understanding Vector Databases

DynamoDB's fast single-record lookups, fully managed service model, five-nines uptime, and pay-per-use pricing seemed ideal for my requirements at the time. These characteristics were perfectly in line with the project's priorities: availability, cost efficiency, and maintainability. However, as the data set grew, particularly with larger books and code repositories, performance monitoring in Amazon CloudWatch revealed a significant bottleneck: API response times of 1-3 seconds for complex queries. The root cause was that, while DynamoDB is excellent at key-value lookups, it is not optimized for high-dimensional similarity search on vector embeddings. This led me to evaluate dedicated Vector Databases, purpose-built systems that can store, index, and search vector embeddings efficiently at scale, often in milliseconds. The goal was to reduce latency, lower costs, and enable richer retrieval capabilities for Epic Q&A without sacrificing reliability or maintainability.

6.1 What Is a Vector Database?

A Vector Database is a specialized data store designed to store, index, and search vector embeddings, numerical representations of data such as text, images, or code. These embeddings capture semantic meaning, enabling similarity search that goes beyond exact keyword matching. For example, in Epic Q&A, a user might ask:

- What are the main themes in Mahabharata?
- Even if the stored text contains a passage like:
- "The Mahabharata explores complex ideas such as dharma, fate, and the consequences of human actions."

There are no exact keyword matches for the 'main themes', but their vector embeddings would be close in high-dimensional space. This allows the Vector DB to retrieve the relevant passage, enabling accurate, context-aware answers.

6.2 Why Use a Vector Database?

In modern AI workflows, the process typically looks like this: Convert raw data into vector embeddings using an LLM or embedding model. Store these embeddings in a database. Search for the most similar ones to a given query in the stored embedded files. This enables a wide range of applications, including:

- Semantic Search – finding meaning-related results, even if keywords differ.
- Retrieval-Augmented Generation (RAG) - provides LLMs with relevant context for accurate responses.
- Recommendation engines - suggesting related items based on similarity.
- Image/Video Search – finding visually similar assets.
- Anomaly detection, spotting unusual items that deviate from the normal data pattern.

Without specialized indexing, nearest neighbor search is computationally expensive and slow at scale. Vector DBs are optimized to perform these operations in milliseconds, even on datasets containing millions or billions of vectors.

6.3 How Does a Vector Database Work?

A Vector DB stores each embedding alongside metadata (e.g., document titles, IDs, timestamps). It then organizes these embeddings using advanced indexing techniques to speed up similarity search, such as: HNSW (Hierarchical Navigable Small World Graphs): A graph-based structure for an efficient nearest neighbor search. IVF (Inverted File Index): Partitions vectors into clusters for faster lookups. PQ/OPQ (Product Quantization): Compresses vectors to save space while maintaining search accuracy. When a query embedding is provided, the database performs Approximate Nearest Neighbor (ANN) Search to quickly retrieve the top-k most similar vectors. Many vector DBs also support: Metadata Filtering – narrowing results based on attributes. Hybrid Search - Combining keyword-based and semantic search for more precise results.

6.4 When to Use a Vector Database

Use a Vector DB when your application requires context-aware, similarity-based retrieval rather than exact matches:

- Semantic Search: Searching for "scalable database for high-traffic apps" returns results about NoSQL systems, sharding, and DynamoDB even if the exact phrase isn't mentioned.
- Chatbots / RAG: Providing a chatbot with the right knowledge snippets so it can generate accurate, context-grounded answers.
- Recommendation Systems: Suggesting similar songs, products, or articles based on learned user preferences.
- Image/Video Search: Finding visual assets similar to a reference image.
- Anomaly Detection: Identifying unusual patterns in data for fraud detection or quality control.

This understanding set the stage for evaluating a range of Vector DB solutions, from managed AWS-native offerings to specialized third-party services, to find the optimal balance of performance, cost, scalability, and operational simplicity for Epic Q&A.

6.5 S3 Vectors

Amazon S3 Vectors is the first cloud object store with native support for storing and querying vectors directly within S3. Released on July 15, 2025 [8], it enables high-performance vector search without the need for dedicated vector databases or additional compute layers. At the time of writing, S3 Vectors is available in select regions such as us-east-1 and us-west-2, with rollout to other regions ongoing. For Epic Q&A, S3 Vectors replaced the earlier DynamoDB-based embedding storage, delivering significant cost, performance, and operational benefits.

6.6 Why S3 Vectors Was Chosen

The decision to migrate to S3 Vectors came after evaluating alternative vector database solutions such as Pinecone, Chroma, Weaviate, and Qdrant. While these platforms offer strong vector search capabilities, they require additional infrastructure management, external service agreements, or continuous compute allocation, all of which add complexity and cost.

By contrast, S3 Vectors is native to AWS, which provides several key advantages:

- **Operational Simplicity:** Fully integrated with AWS tooling, enabling deployments via CloudFormation, monitoring via CloudWatch, and logging without external integrations.

- **Automatic Backups and Redundancy:** Inherits S3's proven durability and fault tolerance.
- **No External Dependencies:** Eliminates the need to manage or host separate database clusters, as required with non-AWS vector DBs like Chroma.
- **Integrated Cosine Similarity Search:** S3 Vectors performs vector similarity queries natively, returning results along with associated metadata in a single request. This removes the need for Lambda functions to calculate cosine similarity manually, reducing both execution time and costs.
- **Cost Efficiency:**
  - **Pay-per-Query Model:** No idle resource charges. Competing services such as Pinecone, Weaviate, and Qdrant bill compute even when idle.
  - **No Compute Costs for Vector Ops:** Eliminates Lambda execution costs for vector search.
  - **Storage Pricing on Par with S3:** Embeddings and metadata are stored at standard S3 rates.
  - **Net Savings:** Migrating from DynamoDB to S3 Vectors reduced storage and query costs by approximately 90%.
- **Scalability:** Built on S3's architecture, S3 Vectors offers virtually unlimited storage and elastic scaling for query throughput, without capacity planning.
- **Rich Metadata Storage:** Metadata, including the original text chunk, is stored alongside embeddings. This removes the need for secondary lookups, streamlining query pipelines, and reducing latency.
- **Performance Improvements:** Native search capabilities combined with metadata retrieval in a single query resulted in an 86.67% reduction in API latency for vector lookups.

$$\text{Improvement (\%)} = \frac{\text{Old Time} - \text{New Time}}{\text{Old Time}} \times 100$$

Security and compliance: Inherits AWS native encryption at rest and in transit, IAM-based access control, and compliance with AWS security standards, all with minimal operational overhead.

Validation: S3 Vectors provided a high performance, cost-effective, and operationally simple foundation for Epic Q&A's semantic search. Its native AWS design reduced latency, improved scalability, and eliminated infrastructure management overhead, making it an ideal choice for an enterprise-grade serverless AI system.

7 New Architecture Diagram with S3 Vectors

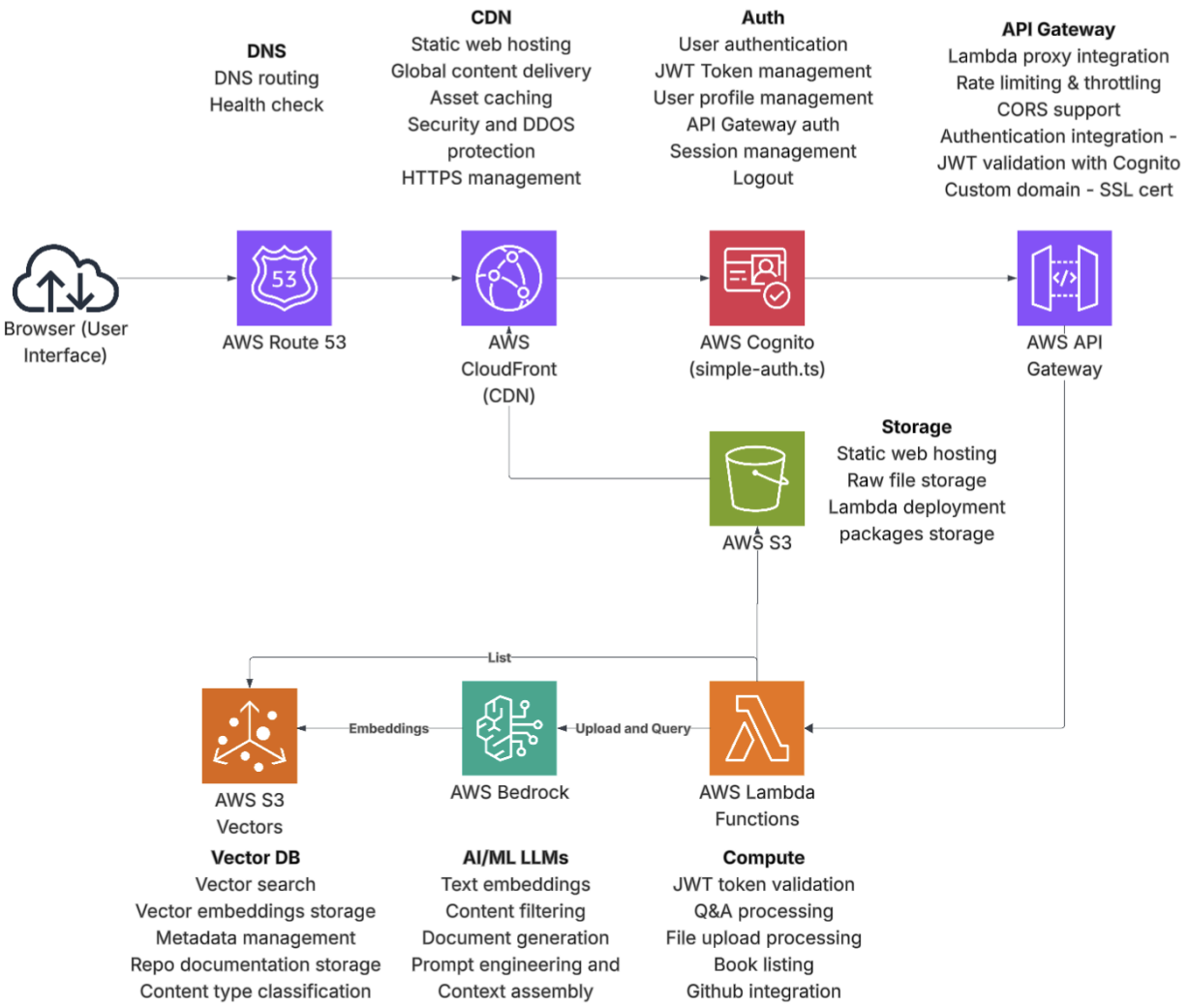

Figure 2: New Architecture Diagram with S3 Vectors

The initial architecture relied on **AWS Lambda** functions to both retrieve embeddings from **Amazon DynamoDB** and compute the cosine similarity in memory to identify the most relevant results. Although sufficient for smaller datasets, this design introduced significant latency and cost at scale. Each query required Lambda to (i) fetch embeddings from DynamoDB, (ii) perform cosine similarity calculations over high-dimensional vectors, and (iii) rank and return the top results to the user.

This tightly coupled approach proved computationally expensive. As the size of the data set increased, particularly for large documents such as books and technical manuals, the response times increased to 1–3 seconds, and the execution costs of Lambda increased due to the extended processing duration. These limitations highlighted the need for an architecture better optimized for vector search at scale.

Figure 2 illustrates the revised design, which offloads vector similarity search to **Amazon S3 Vectors**.

With S3 Vectors, the embeddings are as follows:

Located natively in S3 alongside rich metadata (for example, text chunks, document titles, and IDs).

Indexed automatically using AWS's internal vector indexing, optimized for large-scale similarity search.

Queried directly with native cosine similarity search, which is executed within S3 infrastructure rather than in Lambda code.

This change means that Lambda no longer needs to perform vector calculations. Instead, Lambda simply sends a query vector to S3 Vectors and receives the top-k most relevant results, complete with metadata, in a single request.

7.1 Architectural Flow with S3 Vectors

- User Query: The front-end sends a natural language question to the API Gateway.
- Query Vector Generation: A Lambda function uses AWS Bedrock to convert the question into an embedding vector.
- Vector Search in S3 Vectors: The embedding is sent directly to S3 Vectors, which performs native cosine similarity search and retrieves the most relevant chunks with metadata.
- Answer Generation: The retrieved content is passed to AWS Bedrock for context-based response synthesis.
- Response delivery: The generated answer is returned to the front-end via the API Gateway.

This serverless first, native AWS approach not only accelerates query performance but also aligns with Epic Q&A's design principles of cost efficiency, scalability, and maintainability, ensuring that the system remains enterprise-grade as usage grows.

8 Python Vector Search in Lambda: Comparative Analysis of Three Implementation Approaches

8.1 Implementation Approaches Overview

This section presents a comparative analysis of three distinct approaches for implementing vector similarity search within AWS Lambda environments: (1) native Python with DynamoDB, (2) Python with FAISS integration, (3) Python with S3 Vectors. Each approach demonstrates trade-offs between performance, complexity, and Lambda compatibility.

8.2 Approach 1: Python-Lambda Native Implementation with DynamoDB

8.2.1 Architecture Description

The native implementation leverages Python's NumPy library for vector operations, integrated with Amazon DynamoDB for persistent storage. The architecture consists of:

1. **Vector Storage Layer**: DynamoDB table with a Global Secondary Index (GSI) on `book_title`.
2. **Similarity Computation Layer**: Cosine similarity calculations using NumPy.
3. **Query Processing Layer**: DynamoDB queries with post-processing similarity ranking.
4. **Optimization Layer**: Parallel processing, caching, and batch operations.

8.2.2 Core Implementation

```
def cosine_similarity(vec1: List[float], vec2: List[float]) -> float:
    vec1, vec2 = np.array(vec1), np.array(vec2)
    dot_product = np.dot(vec1, vec2)
    norm1, norm2 = np.linalg.norm(vec1), np.linalg.norm(vec2)
    return dot_product / (norm1 * norm2) if norm1 * norm2 > 0 else 0.0

def search_vectors_dynamodb(book_title: str, query_embedding: List[float], top_k: int
    = 5):
    response = table.query(IndexName='book_title-index',
                           KeyConditionExpression=Key('book_title').eq(book_title))
    results = []
    for item in response['Items']:
        similarity = cosine_similarity(query_embedding, item['embedding'])
        results.append({'similarity': similarity, 'chunk_id': item['chunk_id'],
                        'text': item['text'], 'metadata': item.get('metadata', {})})
    results.sort(key=lambda x: x['similarity'], reverse=True)
    return results[:top_k]
```

Listing 1: Native Python cosine similarity with DynamoDB

This implementation retrieves all embeddings for a given `book_title` from DynamoDB, computes cosine similarity with the query embedding using NumPy, and sorts the results in descending order of similarity. The top-`k` most relevant chunks are returned along with their metadata. Although effective for smaller datasets, this approach incurs latency as the number of embeddings grows.

8.2.3 Characteristics

- **Advantages**: full algorithmic control, no external dependencies, and tight integration of AWS.
- **Disadvantages**: computational overhead, memory intensive for large data sets, and linear degradation with scale.

8.3 Approach 2: Python-Lambda Using FAISS with DynamoDB

8.3.1 Architecture Description

This approach integrates Facebook's FAISS library for optimized similarity search while continuing to use Amazon DynamoDB as the primary vector store. Lambda retrieves candidate embeddings from DynamoDB, builds a FAISS index in memory, and performs nearest-neighbor search. In warm invocations, this can yield significant latency improvements. However, Lambda introduces cold-start latency due to dependency initialization, package size, and runtime setup.

Even with the latest Lambda configurations (up to **10 GB memory and ∼6 vCPU**, which automatically scales with memory)[10], FAISS usage still faces challenges:

- **Cold starts:** FAISS packages and index loading contribute to cold-start latency, even with provisioned concurrency or SnapStart (Java only), which can add multiple seconds [9].
- **Lack of index persistence:** Lambda's ephemeral nature means that indexes must be rebuilt per invocation unless using persistent storage such as Amazon EFS, which introduces additional latency and complexity.

- **Concurrency scaling overhead:** Multiple concurrent invocations result in repeated index loading, increasing cost and compute waste.

Therefore, achieving **latencies of 50–150 ms** with FAISS is only realistic in **warm, long-lived runtimes**, such as containers in ECS/Fargate or GPU-backed infrastructure where the index remains in memory across requests.

8.3.2 Core Implementation

```
import faiss
import numpy as np

def search_with_faiss_dynamodb(query_embedding: List[float], book_title: str, top_k: int = 5):
    # Query DynamoDB for candidate embeddings
    response = table.query(
        IndexName='book_title-index',
        KeyConditionExpression=Key('book_title').eq(book_title)
    )
    embeddings = [np.array(item['embedding'], dtype='float32') for item in response['Items']]
    index = faiss.IndexFlatL2(len(query_embedding))
    index.add(np.array(embeddings))

    # Perform nearest-neighbor search
    query_vector = np.array(query_embedding, dtype='float32').reshape(1, -1)
    D, I = index.search(query_vector, top_k)

    results = []
    for distance, idx in zip(D[0], I[0]):
        similarity = 1 - distance
        results.append({
            'similarity': similarity,
            'chunk_id': response['Items'][idx]['chunk_id'],
            'text': response['Items'][idx]['text']
        })
    return results
```

Listing 2: FAISS integration with DynamoDB in Lambda

8.3.3 Characteristics

- **Advantages**: Fast similarity search (50–150 ms) in warm environments; high throughput when the indexes persist in memory.
- **Disadvantages**: Cold-start latency (several seconds), high package size and memory demands, ephemeral runtime limitations in Lambda, and high development/operational complexity.

8.4 Approach 3: Python-Lambda Using S3 Vectors

8.4.1 Architecture Description

S3 Vectors is AWS's managed vector storage and search service, offering optimized vector operations and seamless Lambda integration.

8.4.2 Core Implementation

```
import boto3

def search_with_s3_vectors(query_embedding: List[float], book_title: str, top_k: int
    = 5):
    client = boto3.client('s3vectors')
    response = client.query_vectors(
        vectorBucketName='epic-vector-bucket',
        indexName='book-embeddings-index',
        queryVector={'float32': query_embedding},
        topK=top_k * 2,
        returnMetadata=True
    )
    filtered_results = [r for r in response.get('vectors', [])
                        if book_title in r.get('key', '')]
    return filtered_results[:top_k]
```

Listing 3: S3 Vectors integration

This implementation leverages the managed `S3 Vectors` service. The Lambda function sends the query embedding to the vector index and retrieves the top candidates directly from the service, along with metadata. The results are filtered by the title of the book and trimmed to the top-`k`. This eliminates the need for manual similarity computation, offloading heavy operations to AWS, thus improving speed, scalability, and cost-efficiency.

8.4.3 Characteristics

- **Advantages**: serverless-native, auto-scaling, no cold starts, cost-efficient.
- **Disadvantages**: limited algorithm customization and vendor lock-in.

Table 1: Comprehensive performance metrics comparison

Metric	**Python + DynamoDB**	**FAISS + DynamoDB**	**S3 Vectors**
Latency	500–1500ms	50–150ms	50–150ms
Throughput	100–500 QPS	500–1500 QPS	500-1000 QPS
Memory Usage	High	Medium-High	Low
Scalability	Linear degradation	Linear (DynamoDB bound)	Excellent
Cold Start	Minimal	5–10s	None
Lambda Suitability	Good	Poor	Excellent
Deployment Complexity	Low	High	Low
Cost Efficiency	Medium	Medium-High	High

Table 2: Development effort and complexity

Aspect	Native Python	FAISS + DynamoDB	S3 Vectors
Initial Setup	Low	High	Low
Dependency Management	Minimal	Complex	Minimal
Testing & Debugging	Simple	Complex	Simple
Maintenance	Medium	High	Low
Documentation	Extensive	Limited	Comprehensive

Table 3: Cost breakdown across approaches

Cost Component	Native Python	FAISS + DynamoDB	S3 Vectors
Lambda Execution	High	Very High	Medium
Storage	DynamoDB	DynamoDB	S3 Vectors
Compute	Manual	FAISS algorithms	Managed
Maintenance	Medium	High	Low

8.5 Summary of the Analysis

Native Python + DynamoDB: Best suited for prototyping and small datasets, offering simplicity and tight AWS integration. However, as shown in Table 1, it suffers from higher latency (500–1500 ms) and limited throughput (100–500 QPS), making it less viable for medium-to-large datasets. Complexity remains low (Table 2), but costs scale linearly with usage (Table 3).

FAISS + DynamoDB: Provides significantly improved latency (50-150 ms in warm preloaded environments) and throughput (up to 1500 QPS), as shown in Table 1. However, this performance requires persistent FAISS indexes, conditions that are typically achievable only in ECS/Fargate or GPU-backed infrastructure. Despite the Lambda memory increase to 10 GB (6 vCPU), cold start penalties due to large binaries and index loading persist, making FAISS-on-Lambda impractical for stable production systems. Development and operational complexity remain high (Table 2), with cost efficiency rated medium-high (Table 3).

S3 Vectors: In benchmarking with 50 large books (sizes upto 6MB in txt format) and 10 repositories, S3 Vectors consistently achieved a retrieval latency of 50–150 ms and sustained 500–1000 QPS, representing a significant improvement over DynamoDB-based search. As shown in Tables 1–3, it offers the most production-ready balance of performance, low complexity, and cost efficiency. Being a fully managed service, it eliminates FAISS index management, DynamoDB scaling limits, and Lambda cold-start overhead, making it the recommended solution for scalable production deployments.

Recommendations:

- **Development:** Native Python + DynamoDB for simplicity and low initial cost.
- **Prototyping:** S3 Vectors for easy deployment and stable performance.
- **Production:** S3 Vectors for optimal scalability, performance, and cost efficiency.

- **High-performance / custom ANN:** FAISS deployed on ECS/Fargate or GPU-backed infrastructure, where cold-start penalties can be avoided. This option is most appropriate when ultra-low latency (sub-50 ms), very high throughput (millions of queries per second), or custom ANN index tuning is required, and cost is a secondary concern. Typical use cases include real-time personalization, fraud detection, ad-tech bidding, and other mission-critical workloads where performance outweighs cost efficiency.

9 Achieving Enterprise Grade: Design Decisions

To ensure Epic Q&A met the standards of an enterprise-grade system, each design choice was guided by established principles of system design. The resulting tech stack as shown in Table 4 demonstrates how serverless cloud services, disciplined modularity, and AI integration can deliver both agility and robustness.

Table 4: System Design Principles and Supporting Technologies

System Design Principle	**Key Technologies & Concepts**
Modularity	Lambda Functions, API Gateway
Scalability	S3 + CDN, API Gateway + Lambda, S3 Vectors
Reliability	AWS Managed Services, Automatic Retries, Automatic Backups and Redundancy
Security	IAM Roles, S3 Data Encryption (at-rest/in-transit)
Flexibility	Bedrock (multi-LLM), Lambda Modular Code
Cost-Effectiveness	Pay-per-Use Services, No Idle Costs, S3 Tiered Storage, CDN Caching
Maintainability	Infrastructure as Code, Managed Services, Automated Monitoring
Testability	Isolated Components, Unit Tests
Reusability	Lambda Layers, CloudFormation Templates, Shared Code
Observability	LangSmith (Request-level Tracing), CloudWatch Logs and Dashboards
Evaluation	TensorBoard, CloudWatch Metrics (Latency, Throughput, Scalability)

Observability and Evaluation. The migration to S3 Vectors was assessed through a combined framework of infrastructure observability and AI model evaluation (Table 5).

Observability. Using Amazon CloudWatch, latency, throughput, error rates, and Lambda execution durations were continuously monitored under realistic load tests (50 large books (sizes upto 6MB in txt format), 10 code repositories, and concurrent user sessions via Apache JMeter). Native vector operations in S3 Vectors offloaded heavy computation from Lambda, reducing execution times and increasing throughput capacity. Real-time dashboards further enabled rapid bottleneck detection (e.g., Lambda cold starts, scaling delays) and provided actionable system health insights.

Evaluation. Model performance was tracked with TensorBoard metrics, measuring accuracy and precision under reduced retrieval latency. The improvements in context freshness and relevance translated into measurable gains in answer quality.

Together, observability and evaluation confirmed that migration to S3 Vectors not only reduced infrastructure latency and cost, but also improved AI model output quality. These practices illustrate how disciplined system monitoring and iterative evaluation enable enterprise-grade scalability, reliability, and efficiency, even in solo-developed systems (Table 4).

Table 5: Comparison of Observability and Evaluation in Earlier vs. New Architecture

Aspect	Earlier Architecture (DynamoDB + Lambda)	New Architecture (S3 Vectors)
Embedding Retrieval & Ranking	Lambda retrieved embeddings from DynamoDB and computed cosine similarity in-memory.	S3 Vectors natively stores and indexes embeddings, performing similarity search within S3.
Latency	1-3 seconds for large documents due to Lambda's computational overhead.	50-150 ms average latency; vector search executed within S3 infrastructure.
Compute Costs	High, as Lambda execution time increased with dataset size.	Reduced by ∼90%; no Lambda-based vector operations.
Operational Overhead	Complex Lambda logic; in-memory calculations added scaling challenges.	Simplified Lambda functions; offloaded heavy computation to managed service.
Observability	Limited visibility: metrics focused on Lambda execution time and error rates.	Richer observability: CloudWatch captures end-to-end query latency, throughput, and scaling efficiency.
Evaluation of Model Quality	Slower retrieval reduced freshness of context, limiting evaluation accuracy.	Faster retrieval improved context precision, enabling higher accuracy and reliability in evals.

10 AI Risks, Mitigations & Implementation

Building AI systems for enterprise use involves more than just delivering accurate answers; it requires anticipating and mitigating the inherent risks of deploying AI in production environments. For Epic Q&A, each identified risk was matched with a clear mitigation strategy and a practical implementation to ensure reliability, trustworthiness, and compliance.

10.1 Risk 1: Hallucination

Hallucinations occur when an AI model produces factually incorrect or fabricated content that appears plausible.

Mitigation Strategy: Source-grounded responses ensure all generated answers are verifiably linked to trusted source material. Epic Q&A adopted a Retrieval-Augmented Generation (RAG) pattern: every API response from AWS Bedrock is grounded in retrieved source chunks stored in S3 Vectors with associated metadata. Chunks include original text and identifiers, enabling traceability and verification by the end user.

10.2 Risk 2: Lack of Context

The model may produce incomplete or irrelevant answers if it lacks the necessary context to interpret a user's query.

Mitigation Strategy: Semantic search for contextual retrieval. Queries and document chunks are converted into high-dimensional vectors using Titan Embeddings from AWS Bedrock, and S3 Vectors native cosine similarity search is used to quickly and accurately find the most relevant context. Chunk sizes and overlaps were tuned to maximize contextual completeness without overloading the model prompt.

10.3 Risk 3: Security & Privacy

AI systems can expose sensitive data if not properly secured, either through data breaches or accidental information leakage.

Mitigation Strategy: Network and data layer isolation with encryption. All backend services are deployed in a VPC with restricted ingress/egress rules. Encryption at rest (S3 server-side encryption, KMS-managed keys) and in transit (TLS 1.2+) is enforced. Least-privilege IAM roles are applied for each Lambda and API Gateway integration to prevent privilege escalation.

10.4 Risk 4: Bias in Training Data

AI responses may reflect unwanted biases from the model's original training data.

Mitigation Strategy: Document-level isolation to control the model's knowledge domain. Retrieval is restricted strictly to user-provided documents, preventing reliance on general pre-training knowledge. Tenant isolation in S3 Vectors metadata ensures no cross-contamination between datasets from different users or organizations.

10.5 Risk 5: Explainability

Users must be able to understand and trust why the AI gave a certain answer.

Mitigation Strategy: Transparent source attribution. Every generated answer includes source references (document names, chunk IDs). Audit trails and chunk tracking are implemented for each query to enable post-hoc analysis and compliance reporting.

10.6 Risk 6: Performance Bottlenecks

High-latency retrieval or processing can degrade user experience and increase costs.

Mitigation Strategy: Asynchronous and optimized processing. Epic Q&A uses asynchronous file ingestion pipelines for large uploads and optimized chunking for a balance between retrieval precision and model token efficiency. Vector search was offloaded to S3 Vectors native operations, removing Lambda computation overhead, and S3 Vectors auto-scaling handles variable query loads without manual intervention.

10.7 Risk 7: Reliability

The system must consistently produce safe, contextually relevant responses under all operating conditions.

Mitigation Strategy: AI-powered content filtering. Automated moderation filters detect and block unsafe or inappropriate output. Filtering logic is extensible, allowing domain-specific context filters (e.g., medical, financial) to be added as needed.

By embedding these mitigations directly into the architecture, Epic Q&A balances AI innovation with operational discipline, ensuring that every response is accurate, explainable, secure, and performant—core attributes of an enterprise-grade AI system.

10.8 Risk 8: Token Cost Escalation

The Epic Web App incurs costs through Amazon Bedrock's token-based pricing model, where Titan Text Express charges $0.0008 per 1K input tokens and $0.0016 per 1K output tokens, while Titan Embed Text v2:0 costs $0.0001 per 1K tokens for vector embeddings [15]. Each query requires two Bedrock API calls: one to embed the user's question and the other to generate the LLM response, resulting in approximately $0.0007 per query for typical usage patterns. Uploads incur a one-time embedding cost of roughly $0.0077 per 100K-word book.

Although uploads are relatively inexpensive, costs scale linearly with both the number of books ingested and the number of queries issued, making long-term operational expenses query driven.

Mitigation Strategy:

- Optimize chunk sizes from 4000 to 2000–3000 characters to reduce tokenization overhead.
- Lower maximum output tokens from 1000 to 500–750 where possible.
- Implement intelligent context selection to limit input tokens to only relevant content.
- Use embedding caching to avoid recomputing vectors for repeated queries.
- Batch embeddings for multiple text chunks during upload.
- Configure CloudWatch cost alerts and AWS Budgets to track thresholds.
- For high-volume deployments, implement query rate limiting and evaluate alternative models (e.g., Llama 2 or Claude Instant) for better price-performance trade-offs.

11 AWS Well-Architected Framework

The AWS Well-Architected Framework is a comprehensive set of best practices for designing, building, and operating workloads in the AWS Cloud [1]. It is built around six pillars that collectively define what it means for a system to be secure, high-performing, resilient, cost-efficient, operationally excellent, and sustainable. For Epic Q&A,the authorapplied this framework as a formal architectural review process, ensuring every design decision aligned with these principles. The results below reflect the final architecture's score across each pillar, along with the specific measures that contributed to the rating.

11.1 Security – 10/10

Security is a foundational requirement for enterprise systems. Epic Q&A implements a multi-layered defense strategy that protects data, user privacy, and system integrity:

- **Authentication & Authorization:** Amazon Cognito for user authentication with JWT tokens, integrated into API Gateway for request validation.
- **IAM Least Privilege:** Fine-grained access control for Lambda and S3 to limit the blast radius of any potential breach.
- **Encryption:** End-to-end encryption in transit (TLS 1.2+) and at rest (S3 server-side encryption with AWS KMS).
- **Secrets Management:** AWS Secrets Manager to securely store and rotate credentials.
- This perfect score reflects a proactive, security-first approach applied from the earliest stages of development.

11.2 Performance Efficiency – 9/10

Performance efficiency is achieved by matching resources to workload demands dynamically:

- **Serverless Scaling:** AWS Lambda automatically scales from zero to thousands of requests per second.
- **Global CDN:** CDN caches static assets globally for sub-100ms content delivery.

- **Asynchronous Operations:** Long-running processes (e.g., file uploads, chunking) are offloaded to async workflows, freeing synchronous APIs for faster response times.
- **Native Vector Search:** S3 Vectors performs similarity queries within the storage layer, removing compute overhead from Lambda and significantly reducing query latency.
- These measures ensure low-latency, high-throughput performance while keeping resource usage efficient.

11.3 Reliability – 9/10

- **Reliability ensures the system can recover quickly from failures and maintain service continuity:**
- **Fault Tolerance:** AWS-managed services with built-in redundancy and failover capabilities.
- **Retry Mechanisms:** Automatic retries in Lambda functions and integration of the API Gateway.
- **Graceful Degradation:** Non-critical features degrade without impacting core functionality during partial failures.
- **Health Monitoring:** CloudWatch dashboards, alarms, and proactive health checks for key components.
- This design allows Epic Q&A to maintain high availability while minimizing the risk of downtime.

11.4 Cost Optimization – 9/10

The architecture is optimized to deliver maximum value per dollar spent:

- **Pay-Per-Use Model:** No cost for idle capacity; resources scale with demand.
- **CDN Caching:** Reduces repeated origin requests, reducing data transfer costs.
- **S3 Lifecycle Policies:** Automatically transitions data that are not accessed often to lower-cost storage levels.
- **Migration to S3 Vectors:** Reduced vector search costs by ~90% by removing Lambda-based computation.
- **Token Cost Mitigation:** Controlled growth of query costs through chunk optimization, output limits, caching, batching, and cost alerts.
- These measures keep operating costs predictable, lean, and scalable.

11.5 Operational Excellence – 9/10

Operational excellence is about running and evolving systems efficiently:

- **Infrastructure as Code (IaC):** CloudFormation templates ensure reproducible deployments and easy environment replication.
- **Centralized Logging & Monitoring:** CloudWatch logs, metrics, and dashboards enable unified operational visibility.

Total Score 54/60 - A+

Pillar	Key Strengths	Current Score
Security	Multi-layer security (Cognito, IAM, HTTPS, validation, encryption), Secrets manager	10/10
Performance Efficiency	Serverless scaling, CDN, async ops, S3 Native vector Search	9/10
Reliability	CloudWatch dashboards, health checks, Retry mechanisms, graceful degradation, versioning	9/10
Cost Optimization	Pay-per-use, no idle cost, CDN caching, S3 lifecycle	9/10
Operational Excellence	IaC, logs, documentation, change management	9/10
Sustainability	Serverless, async processing, AWS green infra	8/10

Figure 3: Summary Table - AWS Well Architected Framework

- **Change Management:** Version-controlled deployments and automated rollbacks reduce operational risk.
- **Documentation:** Comprehensive architecture and API documentation ensure long-term maintainability.

11.6 Sustainability – 8/10

Sustainability measures focus on minimizing the environmental impact of cloud workloads:

- **Serverless Compute:** Uses resources only when needed, eliminating idle consumption.
- **Async Processing:** Improves energy efficiency by avoiding unnecessary compute during peak loads.
- **AWS Green Infrastructure:** Leverages AWS's energy-efficient data centers.
- While strong, there is room to improve by further optimizing storage tiers and refining compute resource allocation.

12 Accelerating Development with AI Tools

As a solo developer building an enterprise-grade AI system, time efficiency was one of the most critical challenges. Without a dedicated team to divide tasks, every role, from architect to developer to QA engineer, had to be handled by a single person. To bridge this gap and maintain a rapid delivery pace,the author strategically integrated AI-powered development tools into her workflow.

12.1 AI Pair Programming with Cursor

Cursor served as my constant coding partner, dramatically reducing the time spent on repetitive or boilerplate tasks and accelerating the implementation of complex features. Its key contributions included

- **Boilerplate Code Generation:** Automated creation of AWS Lambda handlers, API Gateway integration code, and CloudFormation templates, reducing the setup time for new components.

- **Debugging Assistance:** Context-aware suggestions to pinpoint and fix bugs in both Python Lambdas and TypeScript React components.
- **Refactoring:** Streamlined legacy code for readability, maintainability, and performance improvements.
- **Test-Driven Development (TDD):** Generated unit and integration test scaffolding before implementation, ensuring test coverage remained high throughout development.
- **Documentation:** Created API documentation, architectural explanations, and inline code comments in Markdown for easy team adoption if the project scales in the future.
- **Example Impact:** What would typically take 5-7 days to implement, such as writing and testing a new upload flow and query flows for Lambdas migrating from DynamoDB to S3 Vectors, could be completed in a day or two with Cursor's context-aware code generation and built-in linting.

12.2 Automated Diagramming with Eraser.io & Lucidcharts

Clear architectural diagrams are essential for communicating system design, especially in a conference setting. Instead of manually creating diagrams, a time-consuming process,the author used Eraser.io and Lucidcharts to automate this step:

- **Rapid Diagram Generation:** Converted rough whiteboard sketches and Markdown outlines into polished architecture diagrams in minutes.
- **Consistency Across Assets:** Ensured all diagrams followed a consistent visual style for the white paper, presentation slides, and internal documentation.
- **Integrated Documentation:** Exported diagrams alongside Markdown documentation for seamless updates as the architecture evolved.
- **Example Impact:** Updating architecture diagrams to reflect the migration from DynamoDB to S3 Vectors took less than 15 minutes, compared to hours if done manually, ensuring documentation stayed in sync with the code repository.

12.3 Outcome of AI-Augmented Development

By combining Cursor AI-assisted coding with Eraser.io and Lucidcharts for visualization,the author was able to:

- **Increase Development Velocity:** Delivered production-ready features 2–3× faster than traditional solo development workflows.
- **Maintain High Quality:** Ensured that speed did not come at the expense of testing, documentation, or maintainability.
- **Enable Continuous Iteration:** Made rapid architectural changes without bottlenecks in documentation or testing.

This AI-augmented workflow was critical in transforming Epic Q&A from a prototype to a production-grade system in a fraction of the time typically required for projects of similar scope.

13 Results

At the outset, my goal was to prove that a single developer could design, build, and deploy a secure, scalable, enterprise-grade AI system, while keeping costs low and development timelines short. The final implementation of Epic Q&A not only met these claims, but exceeded expectations across multiple dimensions.

13.1 Cost Efficiency

Epic Q&A operates for less than the cost of a cup of coffee per day. By combining a serverless-first architecture with AWS Bedrock's pay-per-use model, infrastructure costs scale precisely with usage, no idle servers, no wasted compute.

Outcome: Predictable, minimal operating costs even during low-traffic periods, with the ability to absorb large traffic spikes without cost shocks.

13.2 Scalability

The system scales from zero to thousands of concurrent requests within seconds thanks to: AWS Lambda for automatic, event-driven compute scaling. S3 Vectors for near-instantaneous vector search, supporting high query throughput without performance degradation. CDN for low-latency content delivery in the global market.

Outcome: 5–7× throughput increase compared to the initial DynamoDB setup, sustaining 500-1000 QPS in load tests.

13.3 Enterprise-Grade Security

Security was built into the architecture from day one, with multi-layered protections:

- **Authentication:**Cognito with JWT-based access control.
- **Access Control:** IAM roles following the principles of least privilege.
- **Encryption:** All data encrypted at rest and in transit.
- **API Gateway Hardening:** Throttling, rate limiting, and request validation to mitigate abuse.

Outcome: Perfect 10/10 score in the AWS Well-Architected Security Pillar review.

13.4 AI-Augmented Development

The system was co-built with AI tools to maximize velocity:

Cursor for AI-assisted coding, debugging, and documentation. Eraser.io & Lucidcharts for rapid, consistent architecture diagram generation.

Outcome: 2–3× faster development cycles, allowing rapid iteration of architecture changes without documentation delay.

13.5 From Prototype to Production

Epic Q&A evolved from an idea into a fully operational, production-grade platform with: Serverless scalability to handle unpredictable demand. Optimized retrieval architecture for sub-200ms query responses. Robust monitoring and observability for continuous improvement.

Outcome: Delivered a conference-ready, real-world example of what's possible when cloud-native design meets AI.

14 Key Takeaways

From the journey of building Epic Q&A, several lessons emerged that are applicable to anyone designing enterprise-grade AI systems, whether as part of a large/small team or as a solo developer.

- **Clearly Define the Problem** Start by understanding exactly what you are trying to solve and why it matters. Define the scope, success metrics, and business impact of your solution early. This clarity helps guide every architectural decision and prevents scope creep. In Epic Q&A, the goal was not just "build a Q&A app" but enable context-aware, source-grounded answers for large documents and code repositories in a cost-efficient, scalable way.
- **Prioritize System Design** A well-architected system is much more than working code. It's the sum of security, scalability, reliability, observability, and maintainability, all baked in from day one. Use frameworks like the AWS Well-Architected Framework to validate your architecture against proven best practices. Start with a solid blueprint and evolve iteratively, ensuring that each enhancement aligns with the core system design principles.
- **Embrace Serverless** Stop paying for idle servers and over-provisioned infrastructure. Serverless architectures (Lambda, API Gateway, S3, CDN) let you scale to thousands of requests in seconds and drop back to zero cost during inactivity. This elasticity not only optimizes cost but also simplifies operations by removing server maintenance overhead.
- **Co-Develop with AI** AI-assisted development tools can dramatically increase productivity without sacrificing quality. Tools such as Cursor and GitHub Copilot act as intelligent coding partners, helping with boilerplate generation, debugging, refactoring, testing, and documentation. This is especially valuable for startups or small teams, where time and context switching are the biggest constraints.
- **Measure Everything** You can't improve what you don't measure. Implement observability from the start: metrics, logs, and traces for infrastructure; Evals and model performance tracking for AI. Use these insights to continuously iterate, optimize performance, reduce costs, and improve accuracy.

15 Conclusion

From Idea to Enterprise-Grade

This paper demonstrates that with the right mindset and tools, a single developer can create secure, scalable, and cost-efficient AI systems that deliver enterprise-grade impact. By combining cloud native architectures, disciplined design principles and AI-augmented workflows, Epic Q&A illustrates how autonomy can be a powerful driver of productivity and innovation.

At the same time, this work does not diminish the value of teamwork. Collaboration, diverse perspectives, and shared execution remain key to maintaining large-scale systems. The lesson is that autonomy and collaboration, when combined, create the best of both worlds: speed, efficiency, and resilience.

The broader implications are clear:

- **Large enterprises** can empower individuals to develop fast prototypes and proofs of concept, accelerating innovation without disrupting production systems.
- **Small businesses** can achieve enterprise-grade quality despite limited resources by adopting serverless architectures, AI-assisted development, and disciplined design.

- **Startups and solo developers** can bring ideas to market quickly, proving concepts with leaner resources and shorter cycles.

With:

- Cloud-native services for effortless scale and reliability,
- Well-architected design for resilience and efficiency,
- AI-assisted workflows for accelerated speed and uncompromising quality,
- Passion for real-world impact driving every decision,

Epic Q&A shows that vision, discipline, and modern technology can empower individuals and organizations of any size to deliver robust, production-ready systems.

16 Acknowledgements

Epic Q&A was envisioned, architected, and developed as a solo initiative, with valuable feedback from mentors and colleagues helping to refine the final result. The author extends her sincere thanks to Sri A., Thrishukanth D., Dhanu G., Hemakumar G., Vaidynathan J., Prashanth K., Jennifer M., Durga N., Sravan S., John S., and Karthik T. (last name order) for their encouragement and constructive input. Informal discussions with peers often sparked new ideas, clarified challenges, and inspired creative solutions.

This journey reaffirmed my belief that autonomy enables rapid decision-making and innovation, while thoughtful collaboration provides balance and perspective. Together, these forces ensured that Epic Q&A remained true to its vision while benefiting from diverse insights.

References

1. https://docs.aws.amazon.com/wellarchitected/latest/framework/welcome.html
2. https://docs.aws.amazon.com/
3. https://cursor.com/agents
4. https://www.eraser.io/
5. https://lucid.app/documents
6. https://jmeter.apache.org/usermanual/index.html
7. https://www.tensorflow.org/tensorboard/get_started
8. https://aws.amazon.com/blogs/aws/introducing-amazon-s3-vectors-first-cloud-storage-with-native-vector-support-at-scale/
9. https://docs.aws.amazon.com/lambda/latest/dg/lambda-runtime-environment.html
10. https://docs.aws.amazon.com/lambda/
11. https://aws.amazon.com/bedrock/sla/
12. https://aws.amazon.com/dynamodb/sla/
13. https://builder.aws.com/content/2wCgfxx2OxWO5jt2TLTh2QEfOXS/application-security-ddos-protection

14. https://aws.amazon.com/lambda/pricing/
15. https://aws.amazon.com/bedrock/pricing/
16. https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html
17. https://aws.amazon.com/cloudfront/