GenAI PoC

Serverless Knowledge Assistant

Technical Handover

December 31, 2025

Prepared by:

Ostap Semenchuk (semenchuk.o20@gmail.com)

Executive Summary

Serverless Knowledge Assistant - A PoC that enables engineers to query the AWS Serverless Application Lens whitepaper using natural language via RAG (Retrieval-Augmented Generation).

Project Goal: Validate serverless-first architecture with Infrastructure as Code for cost-effective knowledge retrieval.

Status: Successfully deployed and operational

Functional Architecture

Overview

The solution implements a RAG (Retrieval-Augmented Generation) pipeline that:

- Ingests PDF documents into a vector database
- Accepts natural language queries via REST API
- Retrieves relevant context from the knowledge base
- Generates accurate, context-aware answers using Amazon Bedrock

API Specification

Endpoint: POST `/query`

Request Format:

```
{
  "query": "What are serverless principles?"
}
```

Response Format:

```
{
  "answer": "..."
}
```

Data Flow

```
User Query
    ↓
API Gateway HTTP API (POST /query)
    ↓
Lambda Function (handler.py)
    ↓
Bedrock Knowledge Base (Retrieve API)
    ↓
S3 Vectors (Vector Search)
    ↓
Bedrock Foundation Model (Nova Micro/Pro)
    ↓
Generated Answer
    ↓
API Gateway Response
```

Application Components

Lambda Handler (`handler.py`):

- Validates POST requests and JSON payload
- Uses Pydantic for input validation
- Calls Bedrock client for RAG processing
- Returns structured JSON responses with proper error handling

Bedrock Client (`bedrock_client.py`):

- Retrieves up to 5 relevant context chunks from Knowledge Base
- Invokes Bedrock foundation model (Nova Micro/Pro) for text generation
- Handles errors and structured logging

Cloud Architecture

Infrastructure Components

Compute Layer:

- AWS Lambda (Python 3.11)
- Memory: 128MB
- Timeout: 12 seconds
- Handler: `handler.lambda_handler`

API Layer:

- API Gateway HTTP API v2
- Protocol: HTTP
- Route: POST `/query`
- Integration: AWS_PROXY (Lambda)
- CORS: Enabled for all origins
- Auto-deploy: Enabled

AI/ML Services:

- Amazon Bedrock Knowledge Base
- Type: VECTOR
- Embedding Model: Amazon Titan Embeddings G1 (`amazon.titan-embed-text-v1`)
- Text Generation Model: Nova Micro (`amazon.nova-micro-v1:0`) or Nova Pro (configurable)
- Data Source: S3 bucket with PDF documents

Vector Database:

- Amazon S3 Vectors (native AWS solution)
- Storage Type: S3 bucket for vector embeddings
- Fully managed by Bedrock Knowledge Base

- No manual index configuration required
- True serverless vector storage

Storage:

- Amazon S3
- Encryption: AES256 (server-side)
- Public Access: Blocked
- Contains: AWS Serverless Application Lens PDF

Security Architecture

IAM Policies:

- Least-privilege principles (no AdministratorAccess)
- Resource-level permissions (specific ARNs)
- Separate roles for Lambda and Bedrock Knowledge Base

Data Protection:

- S3 encryption at rest (AES256) for both document and vector buckets
- No public S3 access
- All data encrypted in transit and at rest

Infrastructure as Code

Terraform Configuration:

- Modular structure with separate files per service
- Single `terraform apply` deploys entire stack
- Automatic Lambda package building (requires Docker)
- Bedrock Knowledge Base ingestion starts automatically

Terraform Modules:

- `api_gateway.tf` - API Gateway HTTP API configuration
- `bedrock.tf` - Bedrock Knowledge Base configuration
- `lambda.tf` - Lambda function definition

- `s3.tf` - S3 buckets (documents and vectors) and object configuration
- `ui.tf` - S3 static website hosting for UI
- `iam.tf` - IAM roles and policies
- `variables.tf` - Input variables
- `outputs.tf` - Deployment outputs

Technology Stack

COMPONENT	TECHNOLOGY
Compute	AWS Lambda (Python 3.11)
API	API Gateway HTTP API v2
AI/ML	Amazon Bedrock (Nova Micro/Pro, Titan Embeddings)
Vector DB	Amazon S3 Vectors (native AWS)
Storage	Amazon S3
UI Hosting	S3 Static Website Hosting
IaC	Terraform
Validation	Pydantic

PoC Findings

Success Criteria

All project success criteria have been met:

Deployment:

- Infrastructure deployable via Terraform without manual steps
- Single command deployment (`terraform apply`)
- Automated Lambda package building

Functionality:

- Functional REST API endpoint
- Web UI for user interaction
- Accepts natural language questions
- Returns valid answers derived from the whitepaper

Security:

- IAM policies strictly scoped with least-privilege principles
- No AdministratorAccess permissions
- Resource-level access controls

Documentation:

- Comprehensive README with deployment instructions
- Architecture documentation
- Testing procedures documented

Performance Metrics

Response Times:

- Average end-to-end: 2-4 seconds per query
- Lambda execution: 1-2.5 seconds
- Bedrock Knowledge Base retrieval: ~500ms-1s
- Bedrock model inference: ~500ms-1.5s

Scalability:

- Auto-scaling Lambda function
- API Gateway HTTP API supports high throughput

- S3 Vectors scales automatically with usage
- No idle costs (fully serverless architecture)

Cost Analysis

Monthly Cost Estimate (1M requests/month):

SERVICE	COST RANGE
Bedrock Nova Micro	$45-210
Lambda	$2-6
API Gateway	$1
S3 Vectors	<$1
Other Services	$1-2
Total	**$49-217**

Note: S3 Vectors provides cost-effective vector storage (approximately <$1/month for vector storage).

Cost Optimization Features:

- Pay-per-use model (no idle costs)
- Minimal Lambda memory configuration (128MB)
- S3 Vectors provides native AWS solution with minimal cost
- No provisioned infrastructure costs
- Cost-effective vector storage solution

Note: Bedrock costs vary significantly based on query complexity, context length, and response length. Token usage depends on Knowledge Base retrieval results (typically 500-2000 input tokens including context).

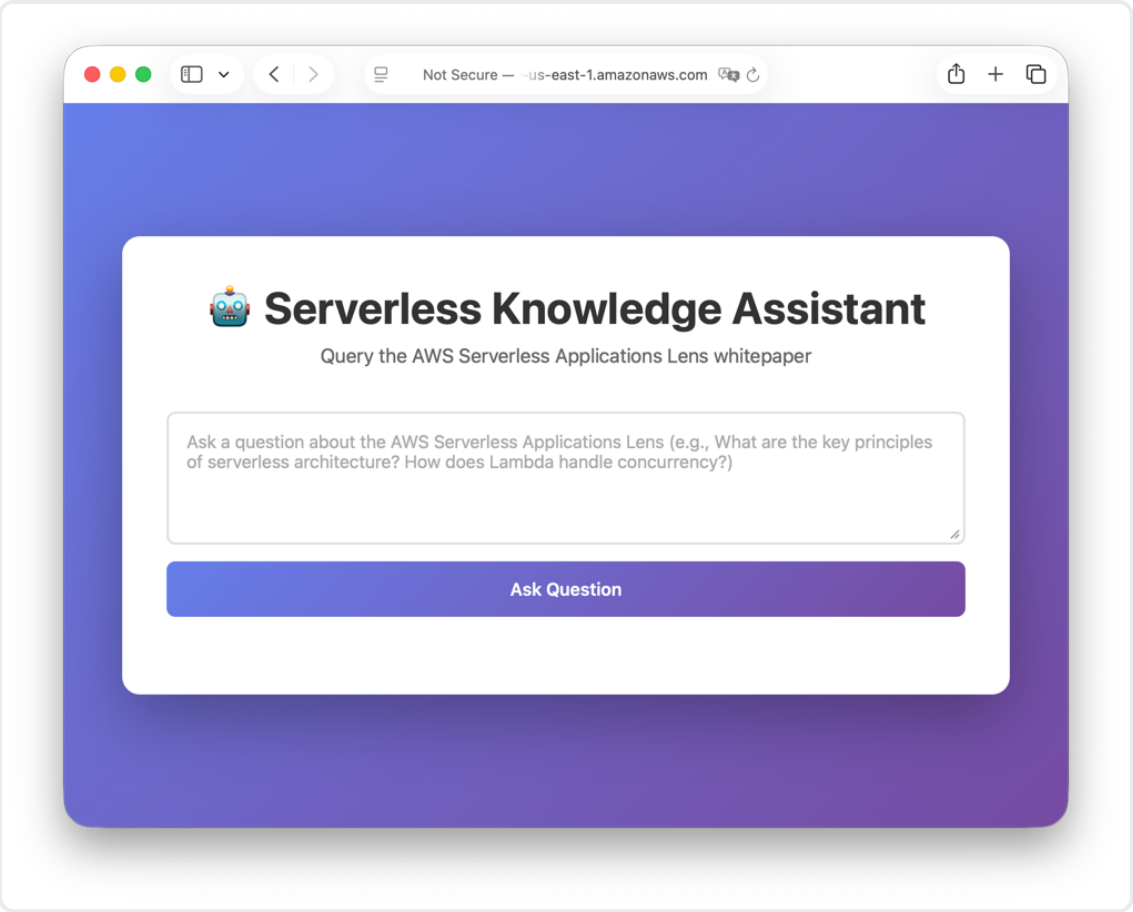
User Interface

Features:

- Static HTML interface hosted on AWS S3 static website hosting
- Auto-configured API Gateway endpoint (injected during deployment)
- Copy-to-clipboard functionality
- Query history (last 5 queries, stored in browser localStorage)
- Loading indicators
- Mobile-responsive design

Access: UI Website URL is automatically displayed after `make deploy` (format: `http://<bucket-name>.s3-website-<region>.amazonaws.com`)

UI Screenshot:



Testing Coverage

Unit Tests:

- Lambda handler tests with mocked Bedrock calls
- Bedrock client tests
- Schema validation tests
- Code coverage: 80%+ requirement met

Infrastructure Tests:

- Terraform resource validation
- IAM policy verification
- API Gateway configuration tests
- S3 bucket configuration tests

Gap Analysis

Out-of-Scope Items

The following items were explicitly excluded from this PoC phase as per project requirements:

Production-Grade Vector Database:

- Current: S3 Vectors (native AWS, production-ready)
- Gap: None - S3 Vectors is a fully managed, production-ready solution
- Impact: Suitable for both PoC and production workloads with minimal cost

Complex Frontend Framework:

- Current: Static HTML UI hosted on S3
- Gap: No React/Angular framework
- Impact: Sufficient for PoC demonstration, production may need modern framework

Authentication & Authorization:

- Current: Public API endpoint (CORS enabled)
- Gap: No Cognito integration or API keys
- Impact: Not suitable for production without additional security layers

CI/CD Pipeline:

- Current: Manual deployment via Terraform
- Gap: No automated testing, building, or deployment pipeline
- Impact: Manual process requires developer intervention for updates

Technical Limitations

Lambda Configuration:

- Memory: Default 128MB (not explicitly optimized)
- Timeout: 12 seconds (may be insufficient for complex queries)
- Cold starts: First invocation may experience latency

Bedrock Model Selection:

- Default: Nova Micro (cost-effective but less capable than Nova Pro)
- Model change: Requires Terraform variable update and redeployment

Knowledge Base:

- Document support: Single PDF document (AWS Serverless Application Lens)
- Ingestion time: 5-15 minutes per document (one-time process)
- Re-ingestion: No automatic re-ingestion on document updates

Monitoring & Observability:

- Current: CloudWatch Logs only
- Missing: Metrics dashboards, alerting, distributed tracing (X-Ray)
- Impact: Limited visibility into system performance and issues

Error Handling:

- Current: Basic error messages returned to client

- Missing: Retry logic for transient Bedrock API failures
- Impact: User may experience failures without automatic recovery

---

Future Recommendations

Production Readiness

Security Enhancements:

- Implement API Gateway API keys or Cognito authentication
- Add rate limiting to prevent abuse
- Enable AWS WAF for additional protection
- Review and tighten IAM policies based on production usage patterns

Performance Optimization:

- Configure Lambda memory based on actual usage patterns (monitor CloudWatch metrics)
- Implement caching layer (ElastiCache) for frequently asked questions
- Consider Lambda provisioned concurrency to reduce cold starts
- Evaluate Nova Pro for better quality (if budget allows)

Monitoring & Observability:

- Set up CloudWatch dashboards for key metrics
- Configure CloudWatch alarms for error rates and latency
- Enable AWS X-Ray for distributed tracing
- Implement structured logging with correlation IDs
- Set up cost anomaly detection

Scalability Improvements

Vector Database:

- S3 Vectors is already optimized for production workloads
- Monitor S3 storage costs and implement lifecycle policies if needed

- Consider S3 Intelligent-Tiering for cost optimization at scale

API Layer:

- Configure API Gateway throttling
- Implement request queuing for high-traffic scenarios
- Add API versioning strategy

Lambda:

- Consider async processing for long-running queries (SQS + separate Lambda)
- Use Lambda layers for shared dependencies
- Optimize package size to reduce cold start time

Feature Enhancements

Multi-Document Support:

- Extend to support multiple PDF documents
- Implement document metadata and filtering
- Add document source attribution in responses

Query Improvements:

- Support follow-up questions (conversation context)
- Implement query suggestions/autocomplete

Response Enhancements:

- Include source citations in responses
- Add confidence scores
- Support streaming responses for better UX

User Interface:

- Migrate to modern framework (React/Vue) if needed
- Add user authentication

DevOps & Automation

CI/CD Pipeline:

- Set up GitHub Actions or AWS CodePipeline
- Automated testing on pull requests
- Automated Terraform validation and deployment
- Infrastructure change approvals

Documentation:

- API documentation (OpenAPI/Swagger)
- Architecture decision records (ADRs)
- Runbooks for common operations
- Disaster recovery procedures

Testing:

- Integration tests with real AWS services (test environment)
- Load testing to identify bottlenecks

Cost Optimization

Right-Sizing:

- Monitor actual usage patterns and adjust resources
- Optimize Bedrock model selection based on quality vs. cost trade-offs
- Review S3 storage usage and implement lifecycle policies if needed

Reserved Capacity:

- Evaluate Lambda provisioned concurrency if cold starts are an issue
- Consider Savings Plans for predictable workloads

Cost Monitoring:

- Set up AWS Cost Explorer dashboards
- Implement cost alerts
- Regular cost reviews and optimization