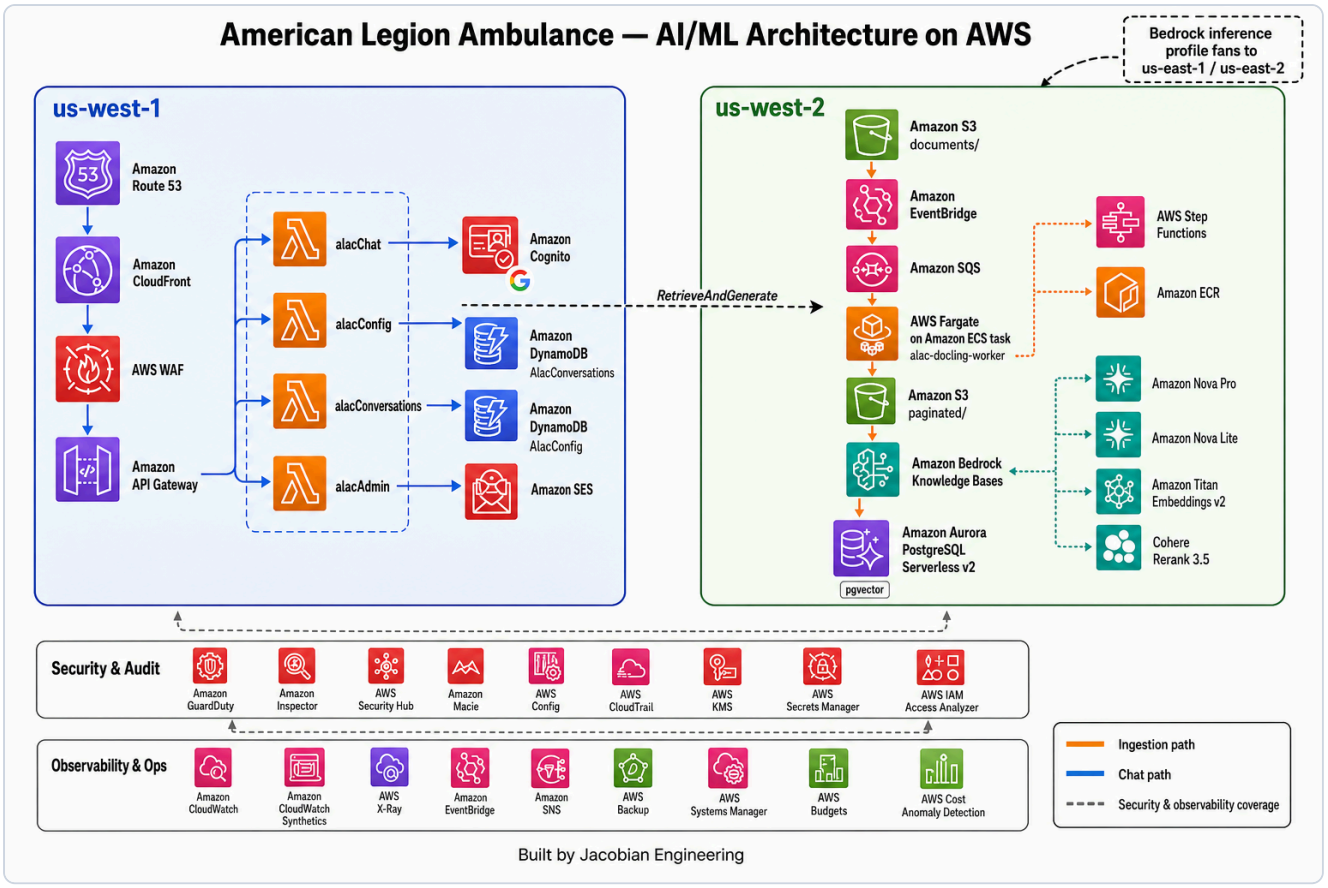
American Legion Ambulance: Production RAG for Frontline Paramedics



Architecture at a Glance

1. Executive Summary

American Legion Ambulance (ALA) has served the rural foothills of Amador and Calaveras Counties since **1929**. Its paramedics ride two-person rigs across long-transport terrain where the receiving hospital is sometimes an hour away, and where the right answer at 2 a.m. has to come from the crew, not from a phone call. ALA's clinical operations have always rested on a ~300-page county Field Treatment Protocols binder — medication dosages, pediatric tables, abbreviations, and procedural checklists that have to be precise and instantly retrievable.

Jacobian Engineering partnered with ALA to replace that binder with a **production Retrieval-Augmented Generation (RAG) assistant** purpose-built for the rig: a

paramedic can ask, in natural language, for the pediatric epinephrine dose at a given weight, the pre-arrival checklist for a stroke alert, or the local protocol for a refusal-of-care — and receive a synthesized, **citation-backed** answer in seconds, with a tap-through to the exact page in the source PDF.

The system is built entirely on AWS. **Amazon Bedrock** anchors the AI plane — Nova Pro for generation, Nova Lite for fast parallel work, Titan Embeddings v2 for vectorization, and Cohere Rerank 3.5 for retrieval precision — over an **Aurora PostgreSQL Serverless v2 (`pgvector`)** vector store managed by **Amazon Bedrock Knowledge Bases**. Around that AI core sits the full well-architected fabric: **AWS Lambda** and **Fargate** for compute, **S3 → EventBridge → SQS** for event-driven ingestion, **Cognito** for identity, **CloudFront / WAF / API Gateway** at the edge, and a layered security posture spanning **GuardDuty, Inspector, Security Hub, Macie, AWS Config, KMS, and CloudTrail**.

> *"Our paramedics carry less and know more. That's the short version of what Jacobian built for us." — Michelle Tyer, President, American Legion Ambulance*
> *This case study is the engineering story behind that sentence.*

---

2. The Customer & The Problem

American Legion Ambulance was chartered in 1929 by **American Legion Post 108** and remains a non-profit community ambulance service — one of the longest-running of its kind in California. ALA holds the 911 contract for Amador and Calaveras Counties, two rural Sierra-foothill counties whose populations are spread across small towns, ranches, and seasonal-recreation areas in the gold-country corridor between Sacramento and Yosemite.

The operating environment is unforgiving for clinical recall. Transport times to the nearest receiving hospital frequently exceed 45 minutes; the on-scene time before transport is often the longest single segment of patient contact in any given call. Crews work two-paramedic rigs, often alone on a stretch of state highway. The protocols they administer — pediatric resuscitation, fibrinolytic checklists, burn-surface-area calculations, refractory-arrest medication sequences — are precise and unforgiving of error.

Until now, the operational answer to "what is the right thing to do in this situation" has been a paper binder of county Field Treatment Protocols, mirrored as a static PDF on every crew member's phone. The binder is authoritative. The binder is also more than 300 pages, structured by policy number, dense with embedded tables, and built for offline reference rather than live query.

PDF text-search on a phone fails this workload in concrete ways. Tables — pediatric medication dosing charts, GCS-by-age cutoffs, fluid-resuscitation grids — are pixels to

the device search index; they are unsearchable. Section titles do not consistently follow one convention: clinical protocols carry an explicit `POLICY: 5XX.XX / TITLE: <name>` header, but reference sections (Rx Guidelines, Abbreviations, Pediatric Medication Charts) bury their titles inside body sub-headings. Page boundaries split single protocols across two pages; PDF readers do not understand that. The result is a binder that paramedics trust precisely because they distrust the search.

ALA's leadership concluded that the field crew needed an AI assistant that could be **trusted in the rig** — fast, accurate, with verifiable links back to the originating page so any answer could be checked in a single tap. Not a chatbot demo. A clinical-grade reference tool that lives at the protocol layer and earns its place by being correct.

> *"At 2 a.m. on a county road, my crew doesn't have time to flip through a binder. They need the answer, and they need to know where it came from." — Michelle Tyer, President, American Legion Ambulance That was the bar Jacobian Engineering was hired to clear.*

---

3. Jacobian's Approach

ALA engaged Jacobian Engineering as the **AI/ML solution partner** — owning the architecture, building the system end-to-end, and operating the production lifecycle (not delivering a demo and walking away). Jacobian's brief covered everything from document ingestion through model selection, security posture, observability, and the runbooks the ALA team would use after launch.

Three engineering bets shaped the work:

1. Bedrock-native, not bring-your-own-model. Jacobian chose **Amazon Bedrock** as the single AI plane — foundation models, embeddings, reranking, and managed knowledge bases all consumed as Bedrock APIs. The alternative — self-hosting a model, an embedding service, and a vector store on EC2 or EKS — would have added a permanent operational tax (patching, capacity, model-version maintenance, GPU economics) for no clinical benefit. Bedrock's `RetrieveAndGenerate` API ties retrieval, reranking, and generation into a single managed call with consistent session semantics, which is the right primitive for a small operations team to depend on.

2. Document quality first. Jacobian's early instinct, reinforced by a one-week ingestion spike against the real protocol book, was that the chat layer cannot rescue a bad chunker. The team invested heavily in a **purpose-fit Docling worker** with layout-aware extraction (Heron layout model + TableFormer for table preservation), OCR fallback (RapidOCR for scanned pages), and — the differentiating idea — a **vision-language-model safety net** for section-title detection on pages where the protocol header convention breaks down. That investment is what makes the chat layer accurate enough for the field. Chapter 5 covers it in depth.

3. Enterprise security posture at SMB scale. ALA is a non-profit ambulance company. The temptation would have been to ship a minimum-viable security baseline and add controls later. Jacobian rejected that. The architecture went in on day one with the same threat-detection, audit, and data-classification controls a regulated healthcare-adjacent workload deserves: **GuardDuty** across every data source, **Inspector** scanning the Docling container, **Security Hub** aggregating against CIS / AFSBP / NIST CSF, **Macie** scanning the documents bucket for PHI/PII, **AWS Config** conformance packs, **CloudTrail** organization trail, KMS-CMK encryption end-to-end. ALA's board expects that posture from any system that becomes part of clinical practice. We built it that way.

Around those bets the engagement ran as a tight discovery-spike-build-harden loop: discovery interviews with operations staff to characterize the real question patterns; an ingestion spike that surfaced the title-detection problem and the table-preservation problem before any UI was built; a security review modeled on AWS Well-Architected; and a phased rollout with a **global kill switch**, full conversation audit, and SNS-paged alarms in place before the first paramedic ever saw the chat interface.

---

4. Architecture Overview

The system has two flows. The **ingestion flow** is offline and event-driven — it turns PDFs into searchable vectors. The **chat flow** is online and synchronous — it answers paramedic questions in real time. Both flows are AWS-native and Bedrock-anchored.

Ingestion flow. Operations staff (or an admin Lambda) uploads a new or revised PDF protocol to the **S3 documents bucket** (`ala-employee-documents-{stage}`, versioned, KMS-CMK encrypted, lifecycle-tiered to S3 Infrequent Access for older revisions). The S3 `ObjectCreated` event fires into **Amazon EventBridge**, which routes the message to an **Amazon SQS** queue. SQS provides the durability, retry-with-exponential-backoff, and dead-letter-queue semantics that an event-driven ingestion pipeline needs to be reliable in production. An ECS service drains the queue and calls **ECS RunTask** to spin up a one-shot **AWS Fargate** task running the `alac-docling-worker` container.

The Fargate task downloads the PDF, runs the full Docling pipeline (Heron layout, TableFormer, RapidOCR), executes the cascading title-extraction described in Chapter 5, and writes **per-chunk markdown files** to S3 under the `paginated/` prefix. Each chunk carries a `<<<ALA_PDF_PAGE_NUMBER:N>>>` page-marker before the body so that the chat runtime can reconstruct the source page for citation rendering.

Amazon Bedrock Knowledge Bases is configured with the `paginated/` prefix as its data source. New objects in that prefix trigger an ingestion job that calls **Amazon Titan Embeddings v2** (1024-dimensional embeddings) and writes the vectors into the **Amazon Aurora PostgreSQL Serverless v2** cluster with the `pgvector` extension. For large reprocessings (e.g., a county-wide protocol revision that touches dozens of files),

an **AWS Step Functions** state machine orchestrates parallel Fargate runs and a single bulk KB sync at the end.

Chat flow. A paramedic opens the portal at `https://americanlegionambulance.org`, already authenticated through **Amazon Cognito** (Google Workspace federation for `@alpost108.org` staff identities). The SPA is served from **Amazon S3** behind **Amazon CloudFront**, fronted by **AWS WAF** (with **AWS Shield Standard** for baseline DDoS protection). A chat request crosses to **Amazon API Gateway**, where a **Cognito User Pool authorizer** validates the JWT and forwards the request to the `alacChat` **AWS Lambda** (Node.js 22.x).

The Lambda reads the global kill switch row from the **AlacConfig** DynamoDB table and short-circuits if disabled. It then reads the user's most recent **AlacConversations** history (DynamoDB, partition by `cognito_sub`, sort by `conversation_id`, point-in-time recovery enabled), merges the last few turns into the query for context, and issues a `RetrieveAndGenerate` call to Bedrock. Bedrock retrieves top-K vectors from Aurora `pgvector`, reranks the results with **Cohere Rerank 3.5**, and synthesizes the answer with **Amazon Nova Pro** invoked through a **cross-region inference profile** (`us.amazon.nova-pro-v1:0`). In parallel, on the first message of a new conversation only, the Lambda also fires a one-shot **Amazon Nova Lite** call to generate a short conversation title. The answer — with structured citations including the PDF page numbers — returns to the paramedic in seconds.

The wrapping layers. Every workload above sits inside two architectural bands. The **security and audit band** — GuardDuty, Inspector, Security Hub, Macie, AWS Config, CloudTrail, KMS, Secrets Manager — provides continuous threat detection, vulnerability management, configuration drift detection, data classification, and audit. The **observability and ops band** — CloudWatch (Logs, Metrics, Alarms, Dashboards, Synthetics), X-Ray, EventBridge, SNS, AWS Backup, Systems Manager, AWS Budgets, Cost Anomaly Detection — provides operational visibility, recovery, and cost governance. Chapter 8 describes both bands in detail.

Region topology is deliberate. The main application stack runs in `us-west-1` because that's ALA's primary region. The Bedrock Knowledge Base and Aurora `pgvector` cluster live in `us-west-2` because Titan Embeddings v2 is not available in `us-west-1`. The Nova Pro and Nova Lite cross-region inference profiles fan out automatically into `us-east-1` and `us-east-2` depending on which downstream region is healthiest at invocation time — a Bedrock-managed concern that the Lambda code does not need to reason about.

---

5. The Document Ingestion Pipeline

This is the chapter that distinguishes a working RAG demo from a clinical-grade RAG system. The Docling worker is the single highest-leverage piece of engineering Jacobian delivered in this engagement.

Why Docling, not a naive PDF chunker. County Field Treatment Protocols are layout-sensitive in ways generic PDF-to-text tools cannot handle. Pediatric medication tables are columnar grids that must stay intact as single chunks; a naive line-by-line text extractor shreds them, and the resulting vectors retrieve only fragments of the dosing table. Reference sections (Rx Guidelines, Pediatric Medication Charts) are scanned bitmaps on some pages — text extraction returns nothing without OCR. Docling's **Heron layout model** segments the page into reading-order regions, **TableFormer** preserves tables as structured units, and **RapidOCR** rescues scanned content. The result is a structured `DoclingDocument` that respects the actual reading order of the page.

Chunker tuning. Jacobian configured Docling's `HybridChunker` to split by markdown structure (headers, sections) while keeping tables atomic, with a per-chunk token budget of **480 tokens** — validated empirically against retrieval recall on the protocol book. Chunks that would split a table are kept whole; chunks that span multiple short sub-sections are merged. This is not the kind of tuning a generic chunker can do; Docling exposes the structural metadata the chunker needs.

The cascading title-extraction pipeline. Many protocol pages carry a header block with a `TITLE: <protocol name>` line, but **not all of them do**. Reference sections, multi-page guidelines, and abbreviation tables either bury the title inside body sub-headings or do not declare one at all. Without a reliable title on every chunk, retrieval suffers — every protocol in the book has an inner section called "III. PROTOCOL", and the resulting vector matches collapse into ambiguity. Jacobian's solution is a two-stage cascade:

- **Stage 1 — regex pass.** A page-text regex (`TITLE\s*:\s*\n?\s*([^\n]{2,120})`) extracts the protocol title cheaply, no model call. This catches the majority of clinical-protocol pages and costs nothing per page.
- **Stage 2 — vision-language-model fallback.** For every page Stage 1 misses, the worker renders the page as a PNG (pypdfium2 at 2.0× scale) and asks **Amazon Nova Pro via Bedrock** — through the same cross-region inference profile the chat layer uses — to identify the most specific section title that owns the page's body content. The VLM returns a structured JSON object `{section_title, confidence, evidence}`. Low-confidence answers are dropped; the keep-set is `{high, medium}`, calibrated against a manually-verified ground-truth set during the build.
- **Stage 3 — merge.** Regex-detected titles are **prepended** to Docling's existing heading_path (preserving Docling's sub-section context). VLM-detected titles **override** Docling's heading entirely, because in the bleed-through cases Docling's heading was inherited from the previous page and is actively wrong.

The chunk-level page marker. Every chunk markdown file carries the `<<<ALA_PDF_PAGE_NUMBER:N>>>` marker before the body content. The chat runtime parses this marker out of the retrieved chunk and surfaces it as a citation page number; the

frontend renders this as a tap-through to the lightboxed source PDF page. **This is the citation that makes the answer trustable.** A paramedic gets an answer about pediatric epinephrine dosing, taps the citation, and sees the original page from the protocol book. The answer is no longer the paramedic's leap of faith — it is the paramedic's verified read.

Worker container design. The `alac-docling-worker` is a Python 3.11-slim image. Jacobian made two non-obvious choices that pay off in production:

1. **CPU-only torch.** Fargate has no GPUs, so the default torch wheel — which ships with ~4 GB of NVIDIA CUDA libraries — is pure waste. The Dockerfile installs torch from the PyTorch CPU index, slashing image size and pull time.
2. **Pre-warmed model weights.** RapidOCR's ~26 MB of det/cls/rec weights and Docling's layout/table model weights are **downloaded and cached at build time** by running a one-line `DocumentConverter()` instantiation inside the Dockerfile. The first Fargate task using the image therefore does not pay the model-download tax on cold start.

From manual batch to event-driven production. Early iterations of the pipeline ran a `batch.sh` script that invoked the worker container with podman against the S3 bucket from a developer workstation. That model gave the team a fast experimental loop. The production architecture replaces the developer loop with **S3 ObjectCreated → EventBridge → SQS → ECS RunTask → Fargate**, so any new PDF uploaded to the `documents/` prefix is processed within minutes without operator action. SQS retries handle transient failures with exponential backoff; the dead-letter queue captures permanent failures and pages on-call via SNS.

> "*We can update a protocol the medical director hands us at noon and have it in the field by the next shift. That used to take us a week of printing and distributing.*" — *Michelle Tyer, President, American Legion Ambulance*

---

6. The RAG Runtime

The runtime side of the system is deliberately simple, because the ingestion side does the hard work. The chat layer's job is to be fast, safe, and correctly cited.

Why `RetrieveAndGenerate`, not roll-your-own orchestration. Bedrock's `RetrieveAndGenerate` API ties retrieval, reranking, and generation into a single managed call with consistent session semantics, citation extraction, and partial-failure handling. Self-orchestrating these stages (an `embed` call, a Postgres similarity query, a separate `Rerank` call, a generation call, and a citation-stitching step in the Lambda) would add operational surface area for no clinical or latency benefit. Jacobian's design rule was to lean on the managed primitive wherever Bedrock offered a coherent one.

Model choice rationale.

- **Amazon Nova Pro for generation.** Best balance of cost, quality, and latency for the call types ALA's paramedics actually make — short clinical questions over a small corpus of grounded source material. Invoked through the cross-region inference profile `us.amazon.nova-pro-v1:0`, which means the Lambda does not need to know which downstream region is healthiest. The inference profile fans out across `us-east-1`, `us-east-2`, and `us-west-2`; the Lambda's IAM policy grants both `InvokeModel` on the profile ARN and on each downstream foundation-model ARN, because Bedrock validates both.
- **Amazon Nova Lite for conversation titles.** Generating a short conversation title for the user's history sidebar is a cheap operation that should not block the main response. On the first message of a new conversation, the chat Lambda fires a Nova Lite call **in parallel** with the main `RetrieveAndGenerate`. Title generation typically returns in around 500 ms while the main response takes longer — the user sees zero added latency. Follow-up messages skip title generation entirely.
- **Cohere Rerank 3.5 for retrieval reranking.** Inside `RetrieveAndGenerate.rerankingConfiguration`, the call uses `cohere.rerank-v3-5:0`. A subtlety Jacobian flagged and documented during the build: Bedrock evaluates both `bedrock:Rerank` and `bedrock:InvokeModel` IAM actions against the rerank ARN; granting only `Rerank` produces a downstream `ValidationException` on `InvokeModel`. Both grants are now in the production IAM policy.

The chat Lambda. The `src/functions/alac.chat` handler is small and linear:

1. Read the kill switch from `AlacConfig` (single-row DynamoDB table). If disabled, return a "temporarily unavailable" payload — the rest of the portal continues to work.
2. Read the user's most recent conversations from `AlacConversations` (DynamoDB, partition by `cognito_sub`, sort by `conversation_id`, PITR enabled).
3. Merge the last few turns of history into the user query for context preservation.
4. Issue `RetrieveAndGenerate` with the Bedrock-managed `sessionId` for KB-managed conversation state.
5. Flatten the returned KB references into structured citations the frontend renders as page-marker links.
6. Write the appended conversation back to DynamoDB.

Cost discipline at every step. Model selection routes cheap work to Nova Lite and expensive work to Nova Pro. Conversation titles are generated only on first message — never on follow-ups. Conversation history is trimmed to a bounded count of recent turns before each call. `RetrieveAndGenerate` uses a fixed `topK` for retrieval and a smaller `numberOfRerankedResults`, capping token spend per query.

The global kill switch. Operations administrators can flip a single attribute on the `AlacConfig` row (`enabled = false`) and the chat surface goes dark within seconds

without a code deploy. This is used during model-version cutovers, during suspected abuse, or when ALA wants the surface offline during a county-wide drill. Re-enabling is the same one-attribute write. The kill switch is the operational lever Jacobian considers non-negotiable for any AI system going to production.

Trust through citations. The frontend renders every answer with its citations as inline links. Each citation includes the PDF page number recovered from the Chapter 5 page-marker injection. Tapping a citation opens the original PDF page in a lightbox. This is what makes the field crew trust the answer. The crew does not need to trust the model — they need to trust that the model is reading the same binder they have always trusted, and the citation surface is the proof.

---

7. Complete AWS Service Inventory

The complete service inventory, ordered from generative-AI through the supporting compute, data, identity, edge, security, and observability fabric.

Category	Service	Role
AI/ML — generation	Amazon Bedrock	Single AI plane for all foundation-model invocations.
	Amazon Nova Pro	Primary generation model behind `RetrieveAndGenerate`.
	Amazon Nova Lite	Cheap parallel call for conversation-title generation.
	Bedrock Inference Profiles	Cross-region profile `us.amazon.nova-pro-v1:0` abstracts downstream region.
AI/ML — knowledge	Amazon Bedrock Knowledge Bases	Managed KB rooted at the `paginated/` S3 prefix.
	Amazon Titan Embeddings v2	1024-dim embeddings for every chunk.
	Cohere Rerank 3.5	Retrieval-time rerank inside `rerankingConfiguration`.
Vector store	Aurora PostgreSQL Serverless v2 + `pgvector`	KB-managed vector store; KMS-CMK encrypted.
Compute	AWS Lambda	Node.js 22.x handlers for chat, history, admin, portal API.
	AWS Fargate on ECS	One-shot tasks running `alac-docling-worker`.
	AWS Step Functions	Orchestrates parallel Fargate runs on large reprocessings.

Category	Service	Role
Ingestion glue	S3 Event Notifications	`ObjectCreated` on documents bucket fires the pipeline.
	Amazon EventBridge	Routes S3 events to SQS; carries Bedrock job state to ops.
	Amazon SQS + DLQ	Durable buffer with exponential backoff; permanent failures page on-call.
	ECS RunTask	Spawns Fargate workers per SQS message.
Storage	Amazon S3	Versioned, KMS-CMK encrypted, lifecycle-tiered documents/paginated/SPA buckets.
	Amazon DynamoDB	`AlacConversations`, `AlacConfig`, member, household, audit tables (PITR).
	Amazon ECR	Holds the Docling worker image with pre-warmed model layers.
Identity	Amazon Cognito	Native sign-up + Google Workspace federation for staff.
	AWS IAM	Narrow per-Lambda roles; Bedrock scoped to specific KB + profile + model ARNs.
	IAM Access Analyzer	Continuously surfaces public-resource drift.
Edge & network	Amazon CloudFront	Edge cache for the SPA and static assets.
	Amazon Route 53	DNS authority for `americanlegionambulance.org`.
	AWS WAF	Managed rule sets on CloudFront + API Gateway.
	AWS Shield Standard	Baseline DDoS protection.
API	Amazon API Gateway	REST API with Cognito User Pool authorizer.
Email	Amazon SES	DKIM-signed sender with configuration sets.
Security	Amazon GuardDuty	All data sources, including Malware Protection and Lambda runtime.
	Amazon Inspector	Continuous CVE scans on Lambda + ECR images.
	AWS Security Hub	Aggregates findings against CIS, AFSBP, NIST CSF.
	Amazon Macie	PHI/PII discovery on the documents bucket.
	AWS Config	Conformance packs against CIS / NIST CSF rules.
	AWS CloudTrail	Org trail, KMS-encrypted, log-file-validation.
	AWS KMS	CMK per data domain.

Category	Service	Role
	AWS Secrets Manager	Third-party API keys and service-account credentials.
Observability	CloudWatch Logs / Metrics / Alarms / Dashboards	Structured logs, metric filters, SNS-paged alarms.
	CloudWatch Synthetics	Authenticated chat-endpoint canary every 5 minutes.
	AWS X-Ray	Traces Bedrock and DynamoDB spans for latency attribution.
	CloudWatch Application Signals	Tracks chat-endpoint SLO.
	Amazon SNS	Alert delivery topic with locked-in ops subscription.
Operational	AWS Systems Manager	Session Manager + Parameter Store.
	AWS Backup	Cross-resource snapshots of DynamoDB + Aurora.
	AWS Budgets	Per-service spend budgets with alerts.
	AWS Cost Anomaly Detection	Anomaly monitors on Bedrock + Fargate dimensions.
DevEx & CI/CD	GitHub Actions	Push-to-branch deploys via Serverless Framework.
	Serverless Framework v3	IaC wrapper around CloudFormation for `us-west-1`.

Every service above is in scope of the security perimeter and observability rail described in Chapter 8.

---

8. Security, Compliance & Operational Maturity

Jacobian Engineering treats security and observability as launch-day concerns, not post-launch additions. The architecture going to production is the architecture below — not a stripped-down MVP with an "and then we'll add security later" backlog.

Identity & access — least-privilege everywhere. Every Lambda has its own narrow IAM execution role. The chat Lambda's Bedrock IAM is a worked example: `RetrieveAndGenerate` and `Retrieve` only on the specific KB ARN; `InvokeModel` only on the two inference-profile ARNs (Nova Pro and Nova Lite) *plus* each downstream foundation-model ARN in `us-east-1`, `us-east-2`, and `us-west-2` that the profile fans out to; `Rerank` only on the Cohere and Amazon rerank ARNs. That specificity is what Bedrock requires — granting only the profile ARN produces runtime validation errors, captured in the production runbook. Every authenticated API Gateway route is fronted

by a Cognito User Pool authorizer. **IAM Access Analyzer** surfaces public-resource drift continuously.

Threat detection in depth. GuardDuty is enabled across every data source (VPC Flow Logs, DNS, CloudTrail, S3, EKS, RDS, Lambda runtime, Malware Protection). **Inspector** continuously scans Lambda functions and the `alac-docling-worker` ECR image for CVEs; the container is rebuilt on a cadence so package CVEs are patched before they accumulate. **Security Hub** aggregates GuardDuty, Inspector, Config, and Access Analyzer findings and reports against **CIS AWS Foundations**, **AWS Foundational Security Best Practices**, and **NIST CSF**.

Data classification & encryption. Amazon Macie scans the S3 documents bucket for PHI / PII discovery. County protocols themselves are not PHI, but operational documents elsewhere in the same bucket can contain personal data, and Macie's findings flow into Security Hub so any drift is visible. All S3 buckets are encrypted at rest with KMS customer-managed keys (`SSE-KMS`). The Aurora cluster and every DynamoDB table use KMS-CMK encryption. All in-transit traffic uses TLS 1.2+; CloudFront and API Gateway enforce modern TLS policies; the Docling worker's outbound calls to Bedrock and S3 use the AWS SDK's default TLS posture.

Audit & configuration drift. AWS CloudTrail runs as an organization-level trail with KMS encryption and log-file-validation enabled; every control-plane action is recorded. **AWS Config** conformance packs continuously evaluate Aurora, S3, DynamoDB, and Lambda configurations against the same NIST CSF / CIS rule sets Security Hub reports against. Drift events flow to EventBridge and SNS so that any unintended configuration change pages on-call within minutes.

Operational kill switches and recovery. The **global kill switch** lives on a single attribute of the `AlacConfig` DynamoDB row. Operators flip it from a portal admin screen and the chat surface goes dark within seconds — no code deploy. **AWS Backup** snapshots every DynamoDB table and the Aurora cluster on a daily schedule with a multi-week retention window. **Point-in-time recovery** is enabled on every DynamoDB table that holds business state. Recovery procedures are documented in runbooks alongside the rest of the launch documentation.

Observability and SLOs. Every Lambda logs structured JSON to CloudWatch. Domain-specific metric filters lift business signals (e.g., the `persistMembership_failed_after_charge` pattern used by the payment subsystem) into custom CloudWatch namespaces, where **CloudWatch Alarms** thresholded against zero route to a single locked-in **SNS topic** with the ops email subscription managed by IaC (not the console — drift is unacceptable here). **AWS X-Ray** traces the chat Lambda's Bedrock and DynamoDB spans, giving latency attribution per stage. **CloudWatch Synthetics** runs an authenticated canary against the chat endpoint every 5 minutes — the same path a paramedic walks — and pages on consecutive failure. **CloudWatch Application Signals** tracks the chat-endpoint SLO so degradation is visible before it becomes outage.

Cost governance. AWS Budgets carries per-service spend budgets with email and SNS alerts. **AWS Cost Anomaly Detection** monitors the Bedrock and Fargate cost dimensions specifically — these are the two services that can quietly spike if a model-version change, a runaway ingestion loop, or an accidental reprocessing slips through — and pages within hours of anomaly onset, not at end-of-month.

Infrastructure as code & deploy hygiene. Every resource above is defined in **Serverless Framework v3** over CloudFormation under `serverless.yml`, with one documented exception: the **Bedrock Knowledge Base** and the **Aurora `pgvector` cluster** live in `us-west-2` and are provisioned via a manually-maintained side stack, because the `AWS::Bedrock::KnowledgeBase` and `AWS::Bedrock::DataSource` resource types are not yet available in `us-west-1` (and Aurora-with-Bedrock requires the same region). The runbook documents the manual steps and the GitHub Actions variable (`ALAC_KB_ID`) that wires the side stack into the main stack at deploy time. **GitHub Actions** deploys dev on push to `main` and prod on push to `prod`. The chat Lambda gracefully degrades — returning a "temporarily unavailable" payload — if the KB env vars are empty, so the rest of the stack can deploy from `us-west-1` without coupling to the side stack's state.

---

9. Field Impact

At 2:14 a.m. on a Tuesday in February, a dispatch goes out for a 14-year-old hiker in anaphylactic shock at a campground 38 minutes from the nearest receiving hospital. The rig is on the road before the alert finishes playing. The paramedic in the back asks the assistant for the pediatric epinephrine dose by weight for a 22-kilogram patient, and gets a cited answer — with the source page from the binder — before the ambulance reaches the campground access road. The crew arrives, administers the dose, and the patient stabilizes. The paramedic later said the assistant did not change what they did. It changed how fast they were sure.

That is the impact case. Multiplied across hundreds of shifts a year, it changes the operational profile of the service.

> *"My new paramedics ramp faster. Not because the system is doing their job — because they spend less time second-guessing themselves and more time learning from the calls they ran." — Michelle Tyer, President, American Legion Ambulance Veteran paramedics use the assistant differently from rookies. Veterans rarely use it as a primary recall tool — they know the protocols. They use it as a cross-check: confirm the dose, confirm the order, confirm the eligibility criteria. The system performs in that role too, and senior crews now spend less time correcting protocol drift in newer staff because the newer staff are arriving at the same answer the senior crew would.*

The trust foundation, again, is the citation surface. Every answer comes with a tap-through to the source page. A paramedic who is uncertain about an answer can confirm or reject it in seconds against the original binder. Adoption is not driven by the model being good — adoption is driven by the model being **checkable**.

The operational lever that surprises ALA's board every time it comes up is protocol-update agility. Before the system, a county medical director revision required printing, distribution, training, and acknowledgement on a timeline measured in weeks. Now the revised PDF lands in the S3 bucket, EventBridge fires, SQS dispatches, Fargate ingests, the Bedrock KB syncs, and the next paramedic who asks gets the new protocol with the new citation — same shift.

Concrete impact metrics — first-call protocol-lookup resolution rate, time-to-answer in the rig, protocol-update-to-field-availability time, and per-month chat volume by shift — are tracked in CloudWatch dashboards and reported to ALA's board quarterly.

---

10. Architecture detail

The system runs two AWS-native, Bedrock-anchored flows. An **event-driven ingestion flow** (S3 → EventBridge → SQS → Fargate Docling worker → Bedrock Knowledge Bases → Aurora `pgvector`) turns protocol PDFs into citation-ready vectors. A **synchronous chat flow** (CloudFront → WAF → API Gateway → Lambda → `RetrieveAndGenerate`) answers paramedic questions in seconds, with Nova Pro generation, Cohere reranking, and Titan Embeddings v2 vectorization. Two cross-cutting bands — **Security & Audit** and **Observability & Ops** — span both regions, and a Bedrock cross-region inference profile fans generation out across `us-east-1` and `us-east-2`.

---

11. Why Jacobian

Jacobian Engineering builds AI/ML systems that go to production and stay there. A demo is not the deliverable. The deliverable is a system that survives shift changes, model-version cutovers, the AWS service-evolution treadmill, and an on-call rotation that wants to sleep through the night.

The ALA engagement is a working sample of what that means in practice. The Docling cascading title-extraction pipeline (Chapter 5) is representative of the level of detail Jacobian invests in the unglamorous middle of an AI system, where document quality, chunk boundaries, and citation fidelity decide whether the chat layer is trusted. The Bedrock-IAM specificity required to make `RetrieveAndGenerate` work across the cross-region inference profile is the kind of subtle production gotcha Jacobian's team finds, diagnoses, and writes into the runbook before the customer ever sees an error. The day-one GuardDuty / Inspector / Security Hub posture is what the system needs to be a member of clinical practice — not a retrofit when the board asks.

The case for an SMB-scale AI/ML Competency partner is the case ALA itself makes. Enterprise reference customers are easy to find in the AWS partner ecosystem. The harder, more valuable problem is bringing the same engineering rigor to a non-profit ambulance service in two California counties — and shipping it. That is where most field-impact AI is going to land in the next five years: not in the Fortune 100, but in the operations of organizations that have always been excellent at their job and now have a partner who can extend that excellence into a generative-AI substrate.

> *"I sleep better knowing this is run the way Jacobian runs it. That is not something I would have said about technology, five years ago, in this industry." — Michelle Tyer, President, American Legion Ambulance For AWS reviewers, prospective customers, and partners considering Jacobian for their own AI/ML build:* ***jacobianengineering.com****.*