Jhorlin De Armas

Principal AI Engineer, Agent Infrastructure

Orlando, FL · Remote · jhorlin@gmail.com · jhorlin@skillfaber.com · jhorlin.com · skillfaber.com · linkedin.com/in/jhorlin · AWS ML Blog (PDIQ)

HIGHLIGHTS

- I build the harness around LLMs: the agentic runtime that turns a model into a working system (tool-use loops, streaming, context management, capability detection, metering, and evals), on AWS in TypeScript and Go.
- Founder of Skillfaber, a production role-based AI agent platform on Amazon Bedrock, built solo: ~270 Lambdas, 128 MCP tool integrations, and a checkpoint-and-continue runtime with automatic context compaction that carries a single agent turn across hours and dozens of Lambda generations. The assistant on this page runs on it.
- Built the whole suite solo in 8 months: the agent platform, plus Sessio (a meeting platform whose transcripts feed agents) and Notitia (a serverless RAG service in Go).
- Co-author of the AWS Machine Learning Blog post on PDIQ, PDI's enterprise RAG system, where answer approval measured 60% to 79%.
- Impact in numbers: cut a PDI support team's resolution time from four days to one, and a payroll customer's agent run from 25 million tokens and 45 minutes to about 100k tokens and two minutes.
- Eval-driven: benchmarked a hybrid retriever at 86% qrels-hit@k on SciFact with regression floors in CI; every claim here is measured, not asserted.
- Independently converged on the tool-calling pattern MCP would standardize, shipping it within days of MCP's announcement, and applied browser-era feature-detection-and-polyfill to LLM capabilities, a pattern I first shipped in 2010.
- Validated by others: Waypoints, a training platform I architected, was sold to General Dynamics; PDIQ was published by AWS; Skillfaber runs in production for real users.
- 20 years across the stack, from real-time C++ and CDN internals to micro-frontends and agent runtimes, hiring and leading teams throughout.

EXPERIENCE

AI Platform Architect · PDI Technologies Mar 2024 – Present

- Architected PDIQ, PDI's multi-tenant AI assistant platform on AWS: composable assistants built from knowledge bases, models, guardrails, and agent tools, streaming chat via CloudFront and Lambda Function URLs. Co-authored the AWS Machine Learning Blog post on its RAG architecture.
- Designed the 'one pipeline, many crawlers' ingestion architecture for a company grown through 35+ acquisitions: rather than asking the organization to centralize its data, containerized crawlers (Confluence, SharePoint, Jira, ServiceNow, Azure DevOps, web, and more) go to it, writing raw content to S3 while a single processing service owns chunking, summary-prepended embeddings (answer approval 60% to 79%), image captioning, and video indexing.
- Made images first-class knowledge: most support documentation lived in screenshots, so the pipeline captions each image and embeds the caption as a markdown document alongside it. Image content became semantically searchable, and answers surface both the relevant text and links to the source images, so support sees the fix in words and pictures.
- Built PDIQ's agent-tool framework, independently converging on the pattern MCP would standardize (it shipped within days of MCP's announcement, long before the spec had traction): each tool is a tagged, versioned Lambda with a two-verb contract, where describe returns the same JSON Schema that is injected into the model and execute runs it — one Lambda per tool, where MCP bundles many tools per server. A discovery process caches every tool's name, description, and schema in DynamoDB; assistants bind tools plus per-tool credentials; the runtime loads that registry and invokes execute. Extended cross-account so customers can expose their own Lambdas as agent tools.
- Mounted the PDIQ assistant directly onto Salesforce case pages for support agents by reusing the platform wholesale (a thin Salesforce adapter over the unchanged app), bridged by an asymmetric-key SSO: Salesforce mints an RS256 JWT signed with its private key, Cognito verifies it against the public certificate via a custom-auth challenge and JIT-provisions the user. Selected over a third-party vendor and Salesforce's native AI, and adopted by the support org. With past case resolutions ingested as knowledge, solutions found once by any team became findable by every agent, and average resolution time fell from four days to one.

- Hired to explore what AI could do for PDI; as the proof of concept showed viability it became PDIQ, the support org's daily tool, embedded in Salesforce as a Lightning component. To sustain what it had grown into, stood up and trained an offshore team of 8 new-graduate engineers: fundamentals first, then AI-augmented development with Claude Code.
- Architect on MyPDI, PDI's convergence answer to growth by acquisition, owning AI integration and authentication. PDI's brands each arrived with their own identity provider and their own data; MyPDI is the aggregation layer that gives a customer one login and, through a converged Trino layer, one view across every PDI property they own. Built on a Single-SPA micro frontend shell where remote apps, their routes, and their navigation register at runtime from data, so tenant admins add applications without a shell redeploy.
- Made the dashboards conversational, so a customer can interrogate the numbers instead of just reading them. A Superset dashboard publishes what the user is looking at (dashboard, title, and the underlying dataset ids) up to the shell, and the assistant opens already scoped to it. The agent then queries the same Trino layer the dashboard was built from through a read-only SQL toolkit, carrying the user's own platform token into Trino's row-level-security authenticator, so it can only reach rows that user could already see. A schema crawler supplies the tables, sample values, statistics, and curated question-to-SQL pairs it needs to write those queries well.
- Built MyPDI's identity federation: a provider-agnostic RFC 8693 token-exchange broker (APISIX edge plus a Go service) with a per-issuer allow-list that federates Okta, Salesforce, and partner identity providers into KMS-signed, tenant-scoped platform tokens, plus a voucher-based Okta on-behalf-of flow for embedded multi-app SSO over a multi-tenant RBAC model.
- Built an AI legal workbench over PDI's contract corpus, deliberately without RAG: an LLM compiles natural language into inspectable search queries and long-context models read whole contracts, with parallel fan-out Q&A streaming across selected agreements, live prompt administration, and durable audit trails.
- Introduced 'virtual contracts' to the legal workbench: contracts are immutable, so amendments stack, and after dozens of addendums the effective agreement is opaque. The tool replays each amendment (sections added, removed, or updated) over the master to synthesize the current effective contract, with a timeline of when each change took effect.

Founder & Principal Engineer · Skillfaber (Independent) Jan 2026 – Present

- Founded Skillfaber on the thesis that an organization's roles should define the guardrails an agent runs under, and that a role's skills are the vendor services a person in that role already uses. Solo-built into a production, multi-tenant AI agent platform on Amazon Bedrock (~270 Lambdas, fully serverless on SST/Pulumi).
- Built the agent runtime: a checkpoint-and-continue loop that chains 15-minute Lambdas into turns that run for hours, with full loop state serialized to S3 and resumed across self-invocations under a configured generation ceiling (~50 generations, about 11–12 hours), retry orchestration with primary-to-backup model failover, and sequence numbers that stay continuous across Lambda generations.
- Owned context management and session state for those turns — what the model sees, what stays cached, and what happens when it no longer fits. When a conversation overflows the model's context window, the harness compacts it in place: the history moves to a linked archive conversation while the live one keeps its id, so its S3 files, task state, MQTT topic, and public API ids never change. Overflow is detected four different ways because providers fail differently — two never reject at all (one answers past its own declared window, one silently drops the stream) — classified against a live-probed catalog of 57 provider overflow wordings that pins the classifier's test suite. Tool calls already issued mid-loop are re-seeded with their original ids so the agent never redoes finished work, and the compaction itself survives a Lambda handoff: one generation clones the history and starts the summary, and the next resumes against the same mailbox instead of re-sending the array that overflowed.
- Engineered the client to the same standard: an RxJS pipeline re-sequences out-of-order MQTT delivery (AWS IoT does not guarantee ordering), every in-flight event persists to IndexedDB so a mid-turn page refresh replays and rejoins the live stream, heartbeats drive idle detection without disturbing the reorder buffer, and a closed tab's broker session is reclaimed so the message backlog survives.
- Made agents composable: an orchestrator calls other agents as tools, each delegate running the same long-turn checkpointing runtime, with a durable outcome mailbox so results survive handoffs on both sides and depth and cycle guards against runaway recursion. Custom tools run as Lambdas or stateful AgentCore shell sessions, and checkpoints treat them differently: stateless Lambda tools abort near the 15-minute wall and re-invoke fresh on the far side, while shell sessions keep running and the next generation reconnects to continue or collect the finished result.

- Designed 'skill books', a progressive-disclosure context system: pinned knowledge becomes cached system-prompt stanzas, while on-demand books are indexed in a hidden tool's description and loaded only as tool results, keeping per-turn context cost flat. Applied the same pattern to the codebase itself, instrumenting it with 32 agent skills for AI-assisted development.
- Invented task-based guardrails with progressively disclosed steps: agents see a task index, then a roadmap, then only the active step's instructions. Steps name files and tools inline, and while a step is active the harness narrows an agent with hundreds of tools to just that step's tool surface, restoring the full set when a step names none; file and tool scoping work independently. Long tool loops also push the system prompt thousands of blocks out of the model's attention, and models start winding down mid-step (a partial result, a question asking permission) long before any platform limit is hit. The harness counters with an ephemeral step reminder re-asserted near the end of the context on every iteration: placed after the cache point so it costs nothing, and never written into the checkpoint, the persisted turn, or the archive. It names only the observed failure mode, so a genuinely blocked or ask-the-user stop stays legal and the model still owns the end of its turn.
- Solved multi-model support with the HTML5 playbook, feature detection plus polyfill: a probe harness tests every model across Amazon Bedrock, Mantle, OpenRouter, and Meta, running its own experiments where provider capability APIs fall short (max-token rules that vary by endpoint, whether tool use and structured output can coexist, per-provider schema quirks). A model lacking vision or document input is told its limitation in-prompt and handed reader tools (OCR, spreadsheet, document) to work around the gap knowingly; where emulation would fabricate, as with structured output, the harness fails loud.
- Created 'recipes' after watching an agent burn tokens calling a code interpreter for the same work every run: TypeScript tools authored in the product, stored in DynamoDB, and executed in a QuickJS sandbox inside Lambda with brokered, allow-listed access to the team's tools, MCPs, other agents, HTTP, and KMS-held secrets. Codifying one payroll customer's repeated agent work cut a 25-million-token, 45-minute run to about 100k tokens and two minutes.
- Made spend a control surface rather than a report: per-tenant budget ceilings gate turn admission on the live request path, evaluated before the turn's first model call, warning at 80% and hard-denying at the ceiling, and every turn lands with eleven attribution dimensions (org, team, user, role, conversation, prompt, allocation, and a customer-versus-platform flag that keeps support turns out of customer billing). The gate fails open on unknown subscription status, because losing a few tokens beats blocking a paying customer. Underneath: a single-compute-site billing invariant enforced by a blocking CI guard, allocation accounting as atomic, idempotent DynamoDB transactions made conflict-free by EventBridge Pipe FIFO re-keying, and Athena partition-projection reporting.
- Abstracted delivery into a channels layer of paired request and response channels: every medium plugs into one of two shared ingress paths, and a FIFO queue keyed by conversation serializes each thread so user and assistant turns always stay in order. Email, Slack (streaming edit-loop), Discord, Teams, Messenger, and Instagram are live; the channel catalog (auth kind, delivery mode, availability) is where WhatsApp, Telegram, and SMS slot in next. Scheduled prompts ride the same rails: timezone-aware recurrence, and pipelines where one agent's reply becomes the next role's prompt. Each link either starts a fresh context, inherits the chain's knowledge, or pins a standing conversation, and a guaranteed terminal channel means every chain ends.
- Integrated 128 third-party services (Salesforce, Workday, NetSuite, SAP, and more) as MCP tools behind an AWS Bedrock AgentCore Gateway, with KMS-encrypted credentials resolved most-granular-first across three tiers (user, role, team). Also shipped an issue-to-PR coding agent in CI: mentioning the bot on a GitHub issue runs a credential-free container that codes the issue, adversarially self-reviews, and opens a human-gated PR.
- Built a live-inference test lane for agent behavior whose oracle asserts side effects over the API rather than model output: an agent that narrates progress without actually calling the update fails, which is the part most agent tests get wrong. Real model calls on a single worker made it too slow and too variable to gate a promotion, so it runs beside the release train rather than eroding trust in it. Production signal comes from a separate correction analyzer that classifies user corrections of agent output by category, confidence, and scope. Both are quality signal; neither is a prompt-regression harness.
- Instrumented the platform for production operation: X-Ray active tracing across every AWS client including the Bedrock Converse call, Lambda Insights scoped to the production stage so branch stages never pay for it, a layered dashboard, 33 metric alarms, a dead-letter queue and alarm on every channel queue, and a liveness alarm on the schedule sweeper so silence pages. Tracing stops where most tracing stops, at the infrastructure boundary: spans answer which AWS call was slow, logs answer what the agent did, and joining them into a replayable turn is the next thing to build.

- Synced metering to Stripe so a token is never lost and never billed twice: each turn's TokensConsumed event fans out over EventBridge to the allocation ledger, the Athena time series, and an SQS queue drained in batches into Stripe's Billing Meter Events API. The event id doubles as Stripe's meter-event identifier for server-side dedup, and because Stripe rejects a repeated identifier rather than ignoring it, the reporter treats that rejection as proof the event was already metered — a rule written after a dead-letter redrive bounced 38 already-billed events straight back. Partial-batch failure reporting means a failed record retries and dead-letters instead of being silently deleted, since lost usage is lost revenue, not late revenue. Billable tokens are computed once at the source (consumed × rate, rounded up so a customer can recompute the charge by hand) and never downstream; webhooks dedup through a conditional write with a 72-hour TTL matching Stripe's retry window and process one at a time for ordering; and every server-initiated Stripe mutation carries a purpose-prefixed idempotency key a human can read in the Stripe dashboard.

Co-founder & Director of Software Development · Kazzcade Jan 2017 – Feb 2024

- Owned architecture, delivery, and cost for Kazzcade's appointment-setting and lead-distribution platform on a fully serverless AWS stack (AppSync, Lambda, Aurora PostgreSQL, DynamoDB, SQS, EventBridge, CloudFront, Cognito, QuickSight, Redshift, Fargate, Athena).
- Conceived, architected, and built 'Suggested Campaigns', a tool surfaced inside Salesforce on the prospect record. It cross-referenced the campaigns a prospect's company had run against who funded each one, so a person already met under one OEM- or reseller-funded program was never re-pitched to the same funder under a different campaign banner, cutting appointment rejections driven by 'we've already met.' Built the first version in native Salesforce JavaScript, then directed the team to rebuild the front end in React behind a reusable hook.
- Built sfSync in-house after Salesforce quoted ~$250k a year for five-minute-at-best sync: one extracted metadata file fans out into seven generated artifacts (the Liquibase changelog, a Go ORM, TypeORM entities, a GraphQL layer, an ERD, 107 per-object Apex triggers, and an integration-test suite that boots a real Postgres and replays the real migrations). When Salesforce's schema changes, regenerate, diff against the committed migrations with Liquibase in Docker, commit the changeset, and CI applies it.
- Made the sync path nearly free to run: triggers publish platform events whose handlers re-query fresh records and push batches to S3 through a direct API Gateway service proxy (no compute in the ingest path), and a Go Lambda upserts them guarded by SystemModstamp so replays and out-of-order deliveries are no-ops. Multi-select picklists land as Postgres text arrays; sub-second, push-driven replication bypassed API limits and per-seat licensing.
- Turned that synced data into a multi-tenant customer portal (React + AppSync) where sales reps view, claim, and reassign their opportunities and clients watch campaign progress near-real-time off the Postgres replica, with per-user embedded Amazon QuickSight analytics, white-label branding across 27 partner domains, and Cognito auth carrying Salesforce identity into every query.
- Designed a buyer-rewards ledger on Amazon QLDB with USDC (Circle) escrow and Plaid payouts, cryptographically auditable end to end.
- Hired and mentored the engineering team; ran a paid internship program teaching serverless best practices on AWS.
- Implemented granular observability with X-Ray and CloudWatch alarms on uptime, latency, error rate, and queue depth.

Senior Software Engineer · Under Armour Jan 2016 – Nov 2016

- Joined Endless Aisle as a full-stack engineer (back end, front end, and the authentication layer). It let any Under Armour store find a product across every inventory source — warehouses, other stores, even stock earmarked for Amazon — and ship it to the customer free and expedited, capturing sales that would otherwise walk out the door. Rolled out to every Under Armour retail store in the United States.
- When the project started slipping a month in, mapped the system with sequence diagrams to expose every cross-system dependency, then built mock interfaces to each so five engineers could build in parallel instead of blocking on one another, unblocking delivery and earning a promotion to Senior Software Engineer.
- Worked across the platform, roughly 11 microservices (Node.js; gRPC internally, REST at the aggregation layer) on Docker and Kubernetes with Kafka messaging, and designed JWT-based authentication for the external-facing service, eliminating stateful session verification.

Sr. Architect · Riptide Software Mar 2013 – Jan 2016

- Hired to lead, architect, and staff the team building Riptide's Elements platform on xAPI, a then-new standard for capturing how a learner moves through material: not just start, completion, and mastery, but time per topic, choices, and selections, as a live event stream. Hired and led a seven-person team across front-end, DevOps, full-stack, and courseware design.

- Built Waypoints, an in-application training overlay: React components injected onto the client's own software that locate on-screen elements, explain what each does, capture every interaction as xAPI statements, and test the learner for mastery inline, right where the feature lives. Riptide later sold the product to General Dynamics.
- Architected it as Node.js microservices on EC2 and Elastic Beanstalk over Postgres, with an early-Angular single-page front end and RxJS (then brand new) processing the xAPI streams to drive real-time feedback.
- Built the Learning Record Store (Node.js + MongoDB) with live WebSocket dashboards, multi-tenant OAuth2, and a scaffolding tool that spins up a courseware project in minutes.

Sr. Software Engineer · NCR Corporation Sep 2010 – Mar 2013

- Built Delta's airline check-in for web and cross-platform mobile (Knockout MVVM, jQuery Mobile, PhoneGap; Spring Web Flow/MVC), deployed across Delta's airport network and piloted at Orlando (MCO). A client-side state machine kept the check-in business rules in one place, and a lazy-loading binding framework kept views fully interchangeable from their view models.
- Shipped it cross-browser with feature detection and polyfills (Modernizr plus shims), the same detect-capabilities-then-fill-the-gaps pattern I would apply to LLMs in Skillfaber fifteen years later.
- Built Marriott hotel kiosk applications (Silverlight; WPF with Unity DI) and an n-tier room-notification system (WCF), with CI and unit-tested MSI packaging.

Software Developer · Toptech Systems Sep 2007 – Sep 2010

- Modernized real-time control software for fuel-terminal automation, a safety-critical domain: ported QNX C systems to object-oriented Linux C++, wrote cross-platform C++ plugins for both, and built a client/server bill-of-lading reconciliation tool (C++ server, C# client, SOAP).
- Built the customer-facing transaction portal by hand: a dynamic, data-driven results table written in JavaScript and jQuery before front-end frameworks took off, over an n-tier ASP.NET MVC2 and MySQL back end.
- Halved deployment time and recovered 200 MB by converting static libraries to shared objects.

Software Developer · Highwinds Software Sep 2006 – Sep 2007

- Optimized the CDN cache index, a 130 GB on-disk data structure in multithreaded C++: the tuning doubled storage capacity and cut drive-lookup time by three seconds without slowing responses.

PROJECTS

PDIQ — Enterprise RAG at PDI — AI assistant that turns PDI's scattered enterprise knowledge into one searchable chat, published as an AWS Machine Learning Blog case study. · AWS ML Blog post

Skillfaber — role-based agent platform — A production, multi-tenant AI agent platform on Amazon Bedrock, built solo: compose a role (model + guardrails + knowledge + tools) and reach it from web, an embeddable widget, email, Slack, Discord, Teams, Messenger, Instagram, schedules, or agent-to-agent delegation. The assistant on this page is a Skillfaber agent. · skillfaber.com

Notitia — serverless RAG — A serverless-first, multi-tenant RAG service in Go for clients with large corpora. Designed as a drop-in knowledge layer for Skillfaber. Retrieval quality is measured, not assumed.

Sessio — meeting platform for agents — A self-built video-meeting platform on the AWS Chime SDK whose recordings and transcripts feed Skillfaber agents, so they learn from real meetings while being tuned for customers.

MyPDI — unified portal — PDI's aggregation layer, and the delivery vehicle for a company-wide strategy of converging brands acquired over decades: one login, one data layer, one place a customer sees everything they own. MyPDI aggregates; PDIQ is the intelligence brand that sits across it. A micro frontend platform where remote apps, their routes, and navigation register at runtime from data (no shell redeploys), with the AI assistant available everywhere as 'chat as a service.'

AI legal workbench — Search-and-interrogate workbench over PDI's contract corpus: natural language compiled into inspectable search queries, and fan-out Q&A that streams parallel answers across every selected contract.

This site — A live demo of my own platform: the AI assistant answering questions here is a Skillfaber agent. One typed TypeScript module renders this page, the downloadable PDF and Word resumes, and the interactive architecture view, so nothing can drift.

sfSync — Salesforce to Postgres in near real time — Salesforce quoted roughly $250k a year for a sync whose floor was five minutes. I built the alternative: one extracted metadata file drives code generation for the entire pipeline, and changes stream out of Salesforce as they happen.

Ocean — lead marketplace and customer portal — The customer-facing product built on top of sfSync: partners sign in to see, claim, and reassign the opportunities they are buying, and watch campaign performance without anyone exporting a spreadsheet.

Suggested Campaigns — funder-aware pitch targeting — Appointment setting lives or dies on not wasting a prospect's time. This tool ran inside Salesforce on the prospect record and worked out which campaigns could honestly be pitched.

SKILLS

Agent Infrastructure: Agent runtimes / tool-use loops, Agentic systems & AI agents, Large Language Models (LLM) & generative AI, Multi-agent orchestration & sub-agents, MCP / tool calling, Bedrock AgentCore Gateway, Prompt & context engineering, Model capability detection & polyfill, Guardrails & responsible AI, Evals & benchmarking (LLM-as-judge), Long-horizon agents (checkpointing, auto-compaction), Durable execution & agent state machines, Context-window management, Streaming (SSE / MQTT), Inference cost & latency optimization, Usage metering, per-tenant cost attribution & enforced budget ceilings, Live-inference agent testing (effect-based oracles), Sandboxed code-mode tools

Models & Providers: Amazon Bedrock, OpenRouter, Anthropic Claude, Amazon Nova, Meta Llama, Cohere (embed + rerank), Multi-model routing & failover, Prompt caching

Data & Retrieval: Retrieval-Augmented Generation (RAG), Hybrid retrieval & reranking, Chunking & semantic search, Embeddings, Knowledge bases, Conversational analytics / text-to-SQL (Trino, Superset), pgvector, OpenSearch, Aurora PostgreSQL, DynamoDB, Athena / Redshift, Data pipelines & governance

Salesforce Platform: Apex, Lightning Web Components, Aura, Platform Events, Metadata API, SOQL, Screen Flows, JWT-bearer SSO, Embedded AI

Identity & Security: Enterprise SSO (Okta OIDC, SAML, OAuth 2.0), Cognito custom auth, RFC 8693 token exchange / OBO, JWT (RS256) / JWT-bearer, Multi-tenant isolation & RBAC, Least-privilege IAM, KMS encryption & audit logging, JIT provisioning

Cloud, Infra & Delivery: AWS Well-Architected Framework, Infrastructure as Code (SST/Pulumi, CDK, CloudFormation), Serverless (Lambda, Step Functions), Event-driven (EventBridge + Pipes, SQS/SNS, Kafka), Microservices & APIs (AppSync, gRPC, REST), IoT Core (MQTT), Containers (Docker, Kubernetes, Fargate/ECS), CI/CD (CodePipeline, GitHub Actions), Observability (X-Ray, CloudWatch, RUM, Datadog), Cost optimization / FinOps, High availability & multi-region, Chime SDK, CloudFront

Frontend: React, Micro-frontends (Single-SPA + import maps), RxJS, Angular, MUI, Tailwind, Vite

Leadership & Delivery: Technical leadership, Team building, hiring & mentoring, Founder / zero-to-one, Executive & stakeholder communication, Technology roadmap & build-vs-buy, Cost & budget ownership, Published (AWS ML Blog co-author), AI-assisted development (Claude Code)

Languages: TypeScript, Go, JavaScript / Node.js, C#, C / C++, Java, SQL, Apex

EDUCATION

BS in Computer Science, University of Central Florida (2006)