ARCHITECTURE SPECIFICATION · CLOUD PLATFORM ENGINEERING

AI-Driven Trading Platform on AWS

Reference architecture for a systematic, AI-augmented trading platform built on Amazon Bedrock, Amazon SageMaker, Amazon OpenSearch Service, and AWS Lambda–based inference serving.

- Amazon Bedrock · GenAI & RAG
- Amazon SageMaker · ML lifecycle
- Amazon OpenSearch · Vector & search
- AWS Lambda · Inference serving

DOCUMENT	VERSION	STATUS	DATE
Architecture Spec	1.0 (Draft)	For Design Review	June 2026

CLASSIFICATION	AUDIENCE	OWNER	PATTERN
Confidential	Eng / Risk / Compliance	Platform Engineering	Serverless-first

Prepared by Platform Engineering · AWS reference architecture

Confidential — Internal Use Only

Contents

1 Executive Summary

2 Scope, Objectives & Assumptions

3 Architecture Principles

4 System Context

5 High-Level Architecture

6 Data Architecture

7 Machine Learning Architecture (SageMaker)

8 Generative AI & Research (Bedrock)

9 Retrieval & Search (OpenSearch)

10 Serving & Latency (Lambda)

11 Signals, Strategy & Execution

12 Real-Time Data Flows

13 Security Architecture

14 Risk, Compliance & Governance

15 Observability & Operations

16 Resilience, HA & DR

17 Deployment & DevOps

18 Capacity, Scaling & Cost

19 Risks, Trade-offs & Open Questions

20 Appendices

1 Executive Summary

This document specifies a reference architecture for an **AI-augmented systematic trading platform** on Amazon Web Services. The platform ingests real-time market and alternative data, derives trading signals from both classical/ML models and generative-AI research, enforces non-bypassable pre-trade risk controls, and routes orders to execution venues — with reproducibility, auditability, and least-privilege security treated as first-class architectural concerns rather than afterthoughts.

Four AWS services anchor the design. **Amazon SageMaker** owns the quantitative machine-learning lifecycle (feature engineering, training, governance, monitoring). **Amazon Bedrock** provides generative-AI research — retrieval-augmented analysis of filings, news, and transcripts, plus agentic workflows. **Amazon OpenSearch Service** serves as the vector store backing retrieval and as the operational search/log-analytics tier. **AWS Lambda** provides event-driven, low-latency inference serving and the orchestration glue that binds the event-driven core together.

1.1 Platform at a glance

Service	Role in the platform	Primary capabilities used
Amazon SageMaker CORE	Quant ML lifecycle: forecasting, classification, execution models; MLOps and model governance.	Feature Store, Training + Spot, Automatic Model Tuning, Pipelines, Model Registry, Clarify (explainability), Model Monitor (drift), real-time / serverless / async endpoints.
Amazon Bedrock CORE	Generative-AI research, summarization, sentiment, and multi-step agentic analysis — human-in-the-loop and risk-gated.	Foundation models (reasoning + embeddings), Knowledge Bases (managed RAG), Agents + action groups, Guardrails, Provisioned Throughput, model evaluation.
Amazon OpenSearch CORE	Vector retrieval for RAG; hybrid semantic + lexical search over the research corpus; centralized log/observability analytics.	Serverless vector engine (k-NN/ANN), hybrid search, ingestion pipelines, OpenSearch Dashboards, UltraWarm/cold tiering for logs.
AWS Lambda CORE	Low-latency inference serving for lightweight models; stream processors; API backends; Bedrock agent actions; orchestration glue.	Container images (≤10 GB), Provisioned Concurrency, Kinesis/DynamoDB stream triggers, EventBridge, Step Functions integration.

KEY ARCHITECTURAL DECISIONS

- **Serverless-first, event-driven core** — managed services and Lambda by default; provision dedicated compute (ECS/EC2) only where stateful, long-lived connections or strict latency demand it (feed handlers, FIX gateway, in-path risk).
- **Hard separation of research/training from live trading** — isolated AWS accounts, networks, and credentials. Capital is never at risk from an experiment.
- **Fail-closed risk** — every order passes through a non-bypassable pre-trade risk gate implementing SEC Rule 15c3-5 controls and a global kill switch; the safe default on any partial failure is to *stop* trading.
- **Train/serve parity** — SageMaker Feature Store and shared inference code paths eliminate train/serve skew and look-ahead leakage.
- **Reproducibility & lineage** — every order can be traced to the exact model version, feature snapshot, and code commit that produced it, persisted to immutable (WORM) storage.
- **Generative AI is advisory, never autonomous** — LLM output is a risk-gated, human-reviewable signal grounded by RAG and constrained by Guardrails; it does not place trades on its own authority.

LATENCY POSTURE — READ THIS FIRST

This architecture targets **systematic / quantitative strategies operating on second-to-sub-second decision horizons** (intraday signal-driven trading, scheduled rebalancing, research-driven positioning). The Lambda + Bedrock + managed-service path realizes end-to-end decision latency in the tens-to-low-hundreds of milliseconds. It is **explicitly not** an ultra-low-latency / high-frequency (microsecond) trading system — that regime requires colocated kernel-bypass networking, C++/FPGA, and exchange cross-connects, and is out of scope (§2.4, §10.1).

2 Scope, Objectives & Assumptions

2.1 Objectives

Functional

- Ingest real-time market data (quotes, trades, depth) and alternative data (news, filings, sentiment) with bounded latency.
- Train, validate, govern, and continuously monitor quantitative ML models for signal generation and execution.
- Perform retrieval-augmented generative research over filings, transcripts, and news, with source citations.
- Aggregate heterogeneous signals into target positions via a portfolio/strategy engine.
- Enforce pre-trade risk checks and route compliant orders to execution venues; capture fills and positions.
- Provide a paper-trading environment with production parity for validation before live capital.
- Produce complete, immutable audit trails and lineage for every decision and order.

Non-functional

Quality attribute	Target / intent
Decision latency	Market-data event → order decision within a defined budget (see §10.1); tens–low-hundreds of ms for the hot path.
Availability	Trading-critical path multi-AZ; safe-state (halt) on degradation. Target ≥ 99.9% during market hours for serving + execution.
Durability / integrity	Orders & positions: near-zero RPO. Audit records: immutable, tamper-evident (WORM / ledger).
Security	Least privilege, encryption everywhere, no public ingress to the trading core, full traceability.
Reproducibility	Forensic replay of any decision from pinned data + model + code versions.
Cost efficiency	Pay-per-use defaults; provisioned capacity reserved for hot paths; Spot for training/backtests.
Regulatory alignment	Architecture supports 15c3-5 market access, model-risk governance, and surveillance requirements (validated by Compliance — see §14).

2.2 In scope / Out of scope

In scope	Out of scope (this revision)
Data ingestion, lake, and feature platform	Ultra-low-latency / HFT colocation stack (§2.4)
ML lifecycle & generative-AI research	Strategy alpha logic / proprietary model internals
Signal aggregation, risk gate, OMS, execution gateway design	Vendor selection for market-data feeds and broker connectivity
Security, observability, DR, DevOps, cost model	Front-office discretionary UI / client portal
Compliance & governance *architecture*	Legal/regulatory sign-off (firm Legal & Compliance own this)

2.3 Assumptions

Asset class US cash equities as the primary asset class, with the data model and execution layer designed to extend to futures, FX, and listed options.

Trading frequency Medium-frequency / systematic: decisions on second-to-minute horizons; intraday and scheduled rebalancing. No microsecond-sensitive strategies.

Regulatory entity	The operating entity is (or trades through) a registered broker-dealer, making SEC Rule 15c3-5 market-access controls mandatory. Exact obligations confirmed with Compliance.
Build posture	Greenfield core on AWS. OMS/EMS may be built or sourced from a vendor; this spec assumes a built core with pluggable execution adapters.
Topology	Single primary AWS Region (lowest-latency to chosen venues) plus a warm-standby DR Region (§16).
Model authority	No model — ML or LLM — has autonomous authority to commit capital outside configured risk limits and (for discretionary strategies) human review.

2.4 Non-goals

EXPLICIT NON-GOALS

(1) **Microsecond/HFT execution.** Lambda cold-start floors, Bedrock inference latency, and Kinesis hops make this path unsuitable for latency-arbitrage strategies. (2) **Fully autonomous LLM trading.** Generative models inform; they do not decide capital allocation unilaterally. (3) **A trading "black box."** Every decision must be explainable and reproducible; opacity is a design failure, not a feature.

3 Architecture Principles

These tenets are the tie-breakers for design decisions throughout the platform. Where a section below makes a specific choice, it traces back to one or more of these.

#	Principle	Implication
P1	**Separate research from trading**	Distinct accounts, VPCs, and IAM. Experiments and training cannot reach production order flow or live credentials.
P2	**Fail closed**	Any ambiguity, breach, or partial failure on the trading path resolves to *stop trading*, not continue. Order generation is fail-closed by default.
P3	**Event-driven & loosely coupled**	Components communicate via streams/events (Kinesis, EventBridge, DynamoDB Streams), enabling independent scaling, replay, and isolation of failures.
P4	**Serverless-first, but right-sized**	Managed/serverless by default for elasticity and low ops; dedicated compute only where stateful sessions or hard latency require it.
P5	**Reproducibility is mandatory**	Data snapshots, feature versions, model versions, and code commits are pinned and recoverable for any historical decision.
P6	**Train/serve parity**	The same feature definitions and inference code serve backtest, paper, and live trading — no parallel implementations that can drift.
P7	**Least privilege & defense in depth**	Scoped IAM roles, permission boundaries, network isolation, encryption, and Guardrails layered so no single control is the only line of defense.
P8	**Everything is observable & audited**	Metrics, logs, traces, and an immutable decision/order ledger. If it isn't logged with lineage, it didn't happen.
P9	**AI is grounded and bounded**	Generative output is retrieval-grounded, citation-bearing, Guardrail-constrained, and always subordinate to risk controls.
P10	**Everything as code**	Infrastructure, pipelines, models, and policies are version-controlled and deployed through reviewed CI/CD with environment promotion.

4 System Context

The platform sits between external **data providers** (sources) and **execution venues** (sinks), while serving internal quantitative, risk, and compliance users. Everything inside the boundary runs within the firm's AWS organization; all external connectivity is authenticated and private — there is no open-internet ingress to the trading core.

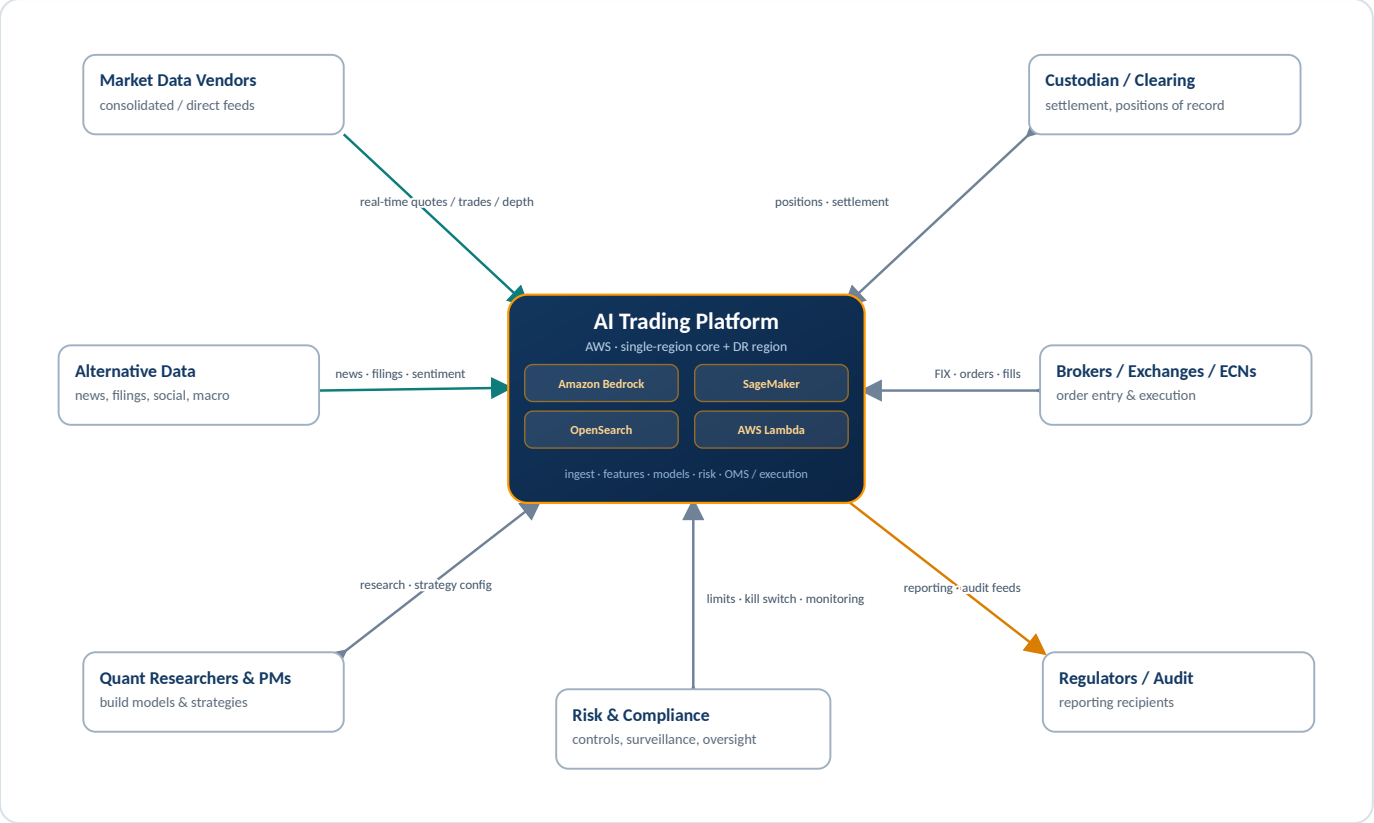


Figure 1. System context (C4 Level 1). Teal arrows are inbound data, the orange arrow is outbound reporting, and grey arrows are bidirectional interactions. The four core AWS services are highlighted inside the platform boundary.

4.1 External entities & interfaces

Entity	Direction	Interaction & transport
Market data vendors	Inbound	Real-time quotes/trades/depth and reference data via vendor feed handlers over persistent TCP/WebSocket or multicast; bulk history via S3 drops.
Alternative data providers	Inbound	News, regulatory filings (e.g., EDGAR), social/sentiment, and macro series via streaming APIs and batch object delivery.
Brokers / exchanges / ECNs	Bidirectional	Order entry and fills, predominantly FIX sessions (long-lived, sequenced) over Direct Connect / dedicated VPN; some venues via authenticated REST/WebSocket.
Custodian / clearing	Bidirectional	Start-of-day positions, corporate actions, and end-of-day settlement reconciliation via SFTP/API batch.
Quant researchers & PMs	Bidirectional	Author models/strategies and configuration in the research account; consume research output. Access via SSO to SageMaker Studio and internal tooling.
Risk & compliance officers	Bidirectional	Configure limits and kill switch; monitor exposure and surveillance dashboards; access via SSO with elevated, audited permissions.

Entity	Direction	Interaction & transport
Regulators / audit	Outbound	Periodic regulatory and audit reporting generated from the immutable trade/decision ledger.

TRUST BOUNDARY

The platform boundary is the firm's AWS organization. Inbound feeds terminate at hardened feed handlers; venue connectivity uses dedicated private circuits; AWS service traffic stays on the AWS network via VPC endpoints / PrivateLink. No component of the trading core is reachable from the public internet (§13).

5 High-Level Architecture

The platform is organized into nine functional layers, with security, observability, governance, and DevOps applied as cross-cutting concerns across all of them. Data flows top-to-bottom: external feeds are ingested and landed, transformed into features and model inputs, processed by the ML and generative-AI tiers, served through Lambda, aggregated into target positions, and finally risk-checked and executed. The four core services are highlighted.

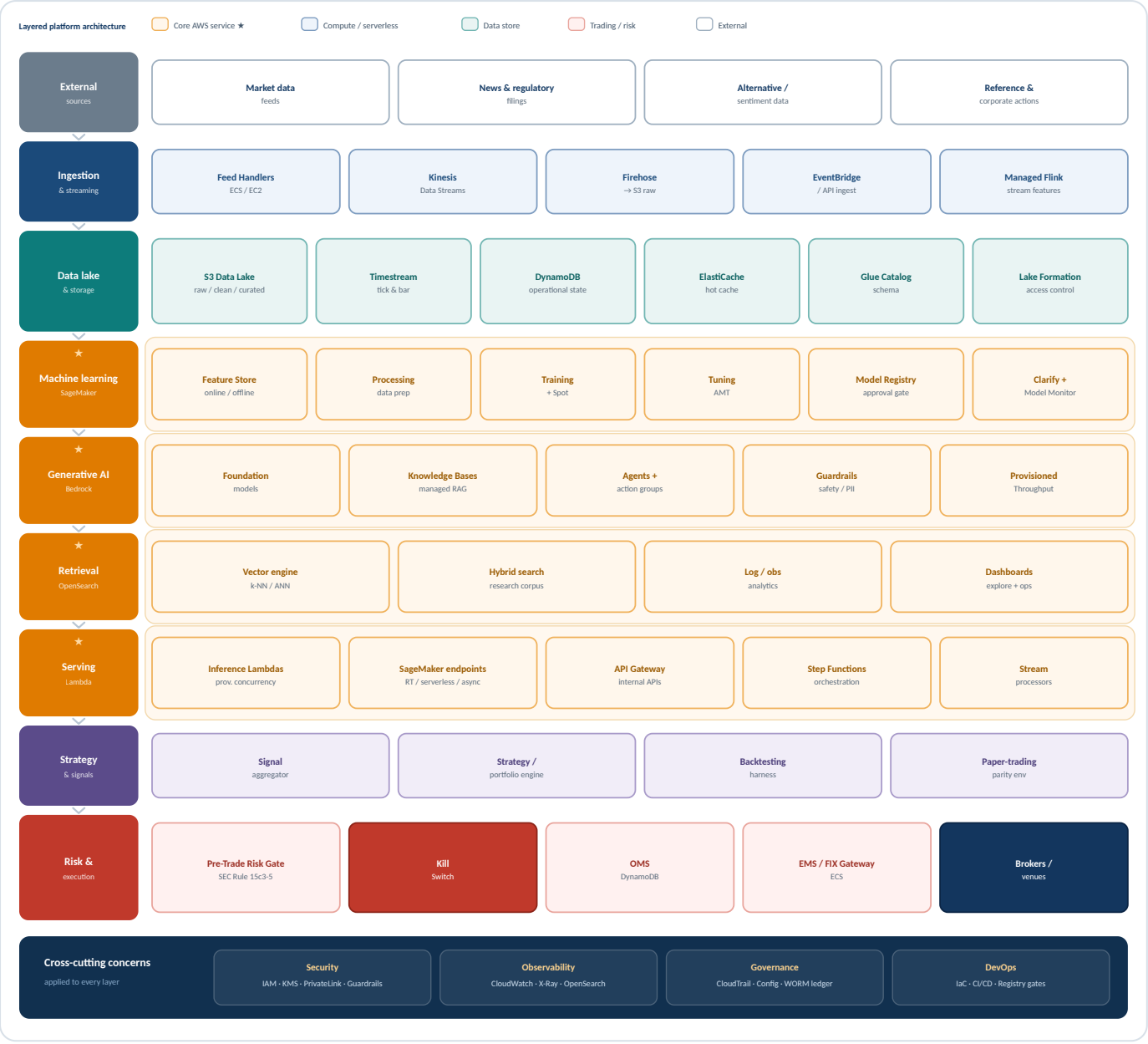


Figure 2. High-level layered architecture. Highlighted bands (★) are the four core AWS services. Boxes name the principal AWS service or component per layer; the cross-cutting band applies to every layer above it.

5.1 Layer responsibilities

Layer	Responsibility	Principal AWS services
1 External sources	Origin of market, alternative, and reference data; execution venues as sinks.	Vendor feeds, EDGAR, broker/exchange APIs
2 Ingestion & streaming	Terminate feeds, normalize, publish to streams, land raw data, compute streaming features.	ECS/EC2 feed handlers, Kinesis Data Streams, Firehose, EventBridge, Managed Service for Apache Flink

Layer	Responsibility	Principal AWS services
3 Data lake & storage	Durable analytical storage plus low-latency operational state and hot caches.	S3, Timestream, DynamoDB, ElastiCache, Glue Data Catalog, Lake Formation
4 Machine learning CORE	Feature management, training, tuning, governance, explainability, drift monitoring.	Amazon SageMaker (Feature Store, Training, AMT, Pipelines, Registry, Clarify, Model Monitor)
5 Generative AI CORE	Retrieval-augmented research, summarization, sentiment, agentic multi-step analysis, AI safety.	Amazon Bedrock (FMs, Knowledge Bases, Agents, Guardrails, Provisioned Throughput)
6 Retrieval & search CORE	Vector store for RAG; hybrid semantic/lexical search; centralized log and observability analytics.	Amazon OpenSearch Service (Serverless vector engine, managed domains, Dashboards)
7 Serving & orchestration CORE	Low-latency inference, internal APIs, stream processing, workflow orchestration.	AWS Lambda, SageMaker endpoints, API Gateway, Step Functions
8 Strategy & signals	Aggregate heterogeneous signals; construct portfolios; backtest; paper-trade with parity.	AWS Fargate, Lambda, Step Functions, SageMaker (backtests)
9 Risk, OMS & execution	Non-bypassable pre-trade risk, kill switch, order lifecycle, venue connectivity.	Fargate/Lambda (risk), DynamoDB (OMS), ECS/EC2 (FIX/EMS)
X Cross-cutting	Identity, encryption, audit, telemetry, and delivery applied uniformly.	IAM, KMS, CloudTrail, Config, CloudWatch, X-Ray, CDK/Terraform, CodePipeline

WHERE DEDICATED COMPUTE LIVES

Serverless is the default, but three components run on dedicated compute (ECS/Fargate or EC2) because they need persistent, sequenced connections or deterministic latency: **(1) market-data feed handlers** (long-lived TCP/multicast sessions Lambda cannot hold), **(2) the EMS/FIX gateway** (stateful FIX sessions with sequence numbers), and **(3) the in-path pre-trade risk service** (consistent single-digit-millisecond latency with no cold-start variance). Everything else — ingestion glue, feature jobs, inference serving, orchestration, reporting — is serverless.

6 Data Architecture

Data is the substrate for every model, signal, and audit record. The data architecture separates **analytical** storage (the lake, optimized for throughput and reproducibility) from **operational** storage (low-latency stores on the trading path), and treats point-in-time correctness as a first-class requirement rather than an afterthought.

6.1 Data domains

Domain	Examples	Characteristics & handling
Market data	L1/L2 quotes, trades, OHLCV bars, order-book depth	High-volume, latency-sensitive, append-only. Streamed via Kinesis; bars/ticks in Timestream; full history in S3 (Parquet).
Alternative data	News, filings (EDGAR), transcripts, sentiment, supply-chain	Semi-structured/text. Lands in S3; indexed for RAG (§8); embedded into OpenSearch vectors.
Reference data	Security master, corporate actions, calendars, sector maps	Slowly changing; bitemporal. Authoritative copies in DynamoDB + S3; corporate actions must be applied with effective dates.
Derived data	Features, signals, model outputs, backtest results	Versioned and lineage-tracked. Features in SageMaker Feature Store; signals/outputs persisted for audit and replay.
Operational data	Orders, executions, positions, P&L, risk state	Strong-consistency, near-zero RPO. DynamoDB (OMS), with change-data-capture to the lake for analytics.

6.2 The data lake — S3 zoned architecture

Amazon S3 is the durable backbone. Data flows through three zones, each a separate bucket (or prefix) with its own access policy, lifecycle, and KMS key:

- **Raw zone** — immutable landing area for source data exactly as received, written by Kinesis Data Firehose. Never mutated; the system of record for "what the vendor actually sent." Object Lock (WORM) for regulated source sets.
- **Clean zone** — validated, deduplicated, schema-normalized, type-corrected. Corporate actions and symbology mapped to the security master. Stored as columnar **Parquet**, partitioned by `dataset/date/symbol-bucket`, compressed (Snappy/ZSTD).
- **Curated zone** — analysis-ready datasets and feature source tables, joined and aggregated for training and backtesting. Often Apache Iceberg tables for snapshot isolation and time-travel.

The **Glue Data Catalog** is the single technical metadata store across all zones; **Glue** jobs (or EMR Serverless for heavy transforms) perform raw→clean→curated processing, triggered by EventBridge on object arrival. **Lake Formation** enforces fine-grained (table/column/row) permissions so that, for example, research roles can read curated features but never raw PII or entitled vendor data they are not licensed for. **Athena** provides ad-hoc SQL for exploration without standing infrastructure.

> **DECISION — COLUMNAR FORMAT & PARTITIONING**
>
> Parquet + Iceberg in the clean/curated zones, partitioned by date and a hashed symbol bucket. *Rationale:* training and backtests scan by date range and symbol universe; columnar layout plus partition pruning minimizes scanned bytes (directly lowering Athena/Glue cost and job runtime). Iceberg snapshots give reproducible, point-in-time reads of a dataset as it existed on any prior date — essential for honest backtests (P5).

6.3 Operational & low-latency stores

Store	Used for	Latency	Why this service
Amazon Timestream	Tick & bar time-series, intraday metrics	ms reads	Purpose-built time-series DB: automatic tiering of recent→historical data, time-series functions, scales with cardinality.
Amazon DynamoDB	OMS state, positions, risk limits, idempotency keys	single-digit ms	Predictable low latency at scale, strong consistency per item, Streams for CDC, Global Tables for DR (§16).
Amazon ElastiCache (Redis)	Hot quotes, latest features, rate counters	sub-ms	In-memory for the read-heavy hot path where even DynamoDB latency is too high; ephemeral, rebuildable from source.
SageMaker Feature Store (online)	Low-latency feature serving at inference	single-digit ms	Managed online store sharing definitions with the offline store, guaranteeing train/serve parity (P6).

6.4 Streaming backbone

Real-time data moves on **Amazon Kinesis Data Streams**, partitioned by symbol so all events for an instrument are ordered within a shard. Consumers fan out independently (enhanced fan-out where latency matters): feature processors, the inference path, persistence, and surveillance each read the same stream without contending. **Kinesis Data Firehose** batches the raw stream to the S3 raw zone. **Amazon Managed Service for Apache Flink** computes stateful streaming features (rolling windows, VWAP, imbalance) with event-time semantics and watermarking for out-of-order ticks. Stream retention (extendable to 365 days) enables **replay** — reprocessing a historical window through the same code to reproduce a past decision or test a new feature.

POINT-IN-TIME CORRECTNESS & LOOK-AHEAD BIAS

Every feature row is stamped with the event time at which its inputs were *knowable*, not when it was computed. Training joins use point-in-time (as-of) semantics so a label is never associated with features that incorporate future information. The same Feature Store definitions serve backtest, paper, and live, so a feature cannot “leak” in research and silently differ in production. Look-ahead bias is the most common way a backtest lies; the architecture makes the correct path the default path.

7 Machine-Learning Architecture & MLOps

Amazon SageMaker provides the end-to-end ML lifecycle: feature management, training, tuning, governance, deployment, and monitoring. The guiding constraints are **reproducibility** (any historical model and its inputs can be reconstructed), **governance** (no model reaches production without passing automated checks and human approval), and **train/serve parity** (one feature and inference implementation across all environments).

7.1 Model classes

The platform hosts several model families, all managed through the same lifecycle: **tabular signal models** (gradient-boosted trees, e.g. XGBoost, for cross-sectional return/score prediction), **sequence models** (for time-series and regime detection), **NLP/embedding models** (sentiment, event extraction — increasingly served via Bedrock foundation models, §8), and **execution models** (slippage and market-impact estimation feeding the execution layer). Strategy alpha logic and proprietary model internals are explicitly out of scope (§2.2); this section governs *how* models are built and operated, not *what* they predict.

7.2 The model lifecycle

The pipeline in Figure 3 is implemented as a **SageMaker Pipeline** — a versioned DAG in which every step is a tracked, reproducible unit. A run is triggered on a schedule, on new data thresholds, or by a drift signal from Model Monitor.

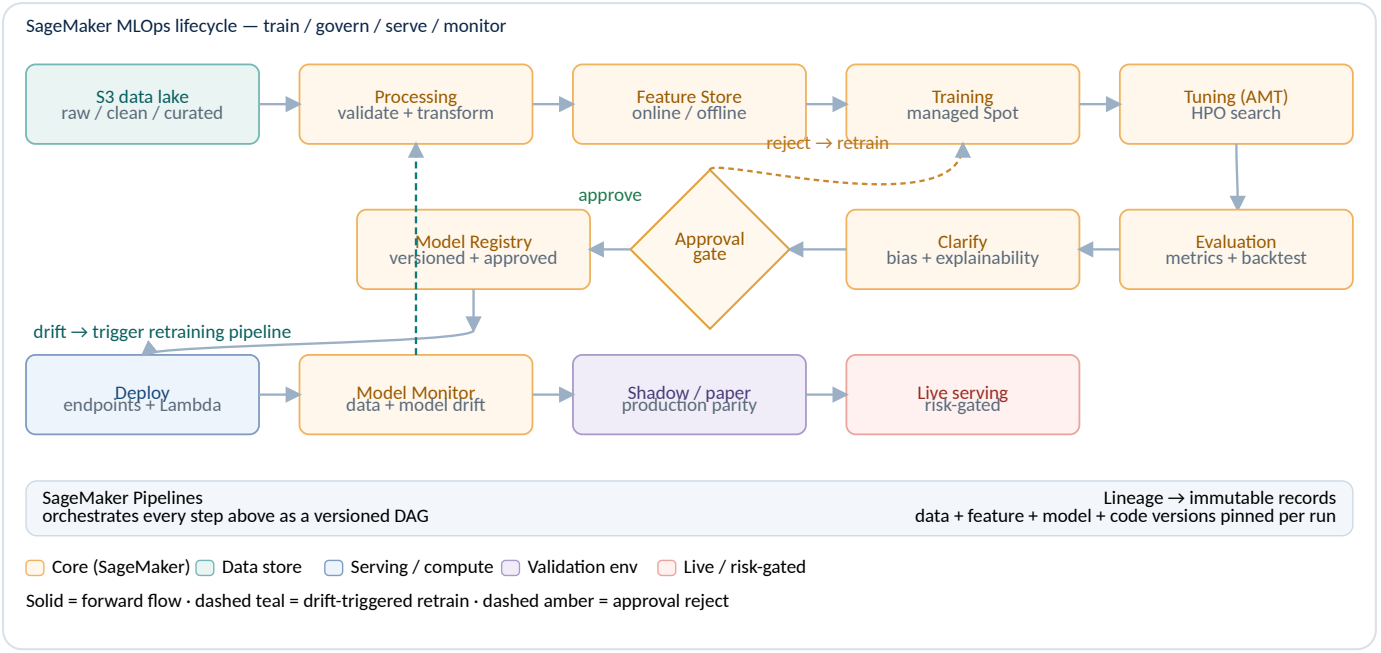


Figure 3. Model lifecycle on Amazon SageMaker. Training and tuning produce candidates that are evaluated (with Clarify for bias and explainability) and gated by a human approval step in the Model Registry before deployment. Deployed models are watched by Model Monitor; detected drift triggers an automated retraining run. SageMaker Pipelines orchestrates the chain and pins full lineage for every run.

Stage	Function & key choices
Processing	SageMaker Processing validates, cleans, and transforms curated data into training sets; computes/writes features to Feature Store. Containerized and versioned.
Training (+ Spot)	Training jobs run on managed **Spot** instances with checkpointing for cost efficiency; framework containers pinned by digest. Hyperparameters, code commit, and input data S3 URIs recorded per job.
Tuning (AMT)	Automatic Model Tuning searches hyperparameters (Bayesian/Hyperband) against a validation objective, with early stopping to bound cost.

Stage	Function & key choices
Evaluation	Held-out and walk-forward evaluation plus an event-driven **backtest** branch; metrics captured to the run and the Registry. Promotion thresholds are explicit and version-controlled.
Clarify	SageMaker Clarify computes feature attributions (SHAP) and bias/stability metrics, attached to the model package for the approver and for audit (§7.4).
Registry + approval gate	Candidates register as *PendingManualApproval*. A human (model risk / desk owner) reviews metrics, attributions, and backtests, then approves or rejects. Only *Approved* packages are deployable. control point
Deploy	Approved models deploy to SageMaker endpoints and/or are packaged for Lambda serving (§10) via blue/green with canary shadowing (§17).
Model Monitor	Monitors data-quality, feature drift, and prediction drift against a captured baseline; breaches raise CloudWatch alarms and can trigger an automated retraining run.

DECISION — MANAGED SPOT FOR TRAINING & BACKTESTS

Training and backtesting are interruption-tolerant, bursty, and cost-dominant, making them ideal for managed Spot with checkpointing (substantial savings vs on-demand). Inference serving, by contrast, uses on-demand/provisioned capacity because the hot path cannot absorb Spot reclamation. Compute posture is matched to each workload's tolerance, not applied uniformly (P4).

7.3 Reproducibility & lineage

Every model package in the Registry pins four coordinates: the **data snapshot** (Iceberg snapshot ID / S3 version), the **feature versions** (Feature Store), the **code commit** (training + inference container digests), and the **hyperparameters**. SageMaker ML Lineage Tracking records the graph linking datasets, processing jobs, training jobs, and models. Given any production decision, an engineer can reconstruct the exact model and inputs that produced it — a hard requirement for both debugging and regulatory inquiry (P5, §14).

7.4 Explainability & bias

Opacity is a design failure (§2.4). Clarify produces global and per-prediction SHAP attributions so signal contributions are inspectable; feature-attribution drift is monitored alongside data drift (a model whose *reasons* change is suspect even if its accuracy holds). For any LLM-assisted research output, explainability is delivered through retrieval citations (§8) rather than post-hoc attribution.

7.5 Model-risk governance

The lifecycle is designed to satisfy model-risk-management expectations (in the spirit of SR 11-7): independent **validation** before promotion, an enforced **approval gate**, ongoing **monitoring**, and full **documentation/lineage**. Champion/challenger is supported natively — a challenger runs in shadow against live traffic (scoring without trading) until it earns promotion. Material model changes are versioned, reviewed, and auditable. *The firm's Model Risk and Compliance functions own the validation standards; this architecture provides the mechanisms to enforce them (§14).*

RISK — OVERFITTING & DATA SNOOPING

Backtests are trivially gamed by repeated tuning against the same out-of-sample window. Mitigations baked into the pipeline: strict train/validation/test temporal separation, walk-forward evaluation, point-in-time feature joins (§6.4), a hold-out window untouched until final validation, and tracking of how many configurations were evaluated. A strong backtest is necessary but never sufficient for promotion; shadow/paper parity (§11.4) is the final gate before live capital.

8 Generative AI & Research Platform

Amazon Bedrock provides the generative-AI capability: retrieval-augmented research over filings, transcripts, and news; summarization and event extraction; and agentic multi-step analysis. The governing rule is unconditional — **generative models inform; they never decide capital allocation**. Every output is retrieval-grounded, citation-bearing, Guardrail-constrained, and subordinate to the risk controls (P9, §2.4).

8.1 Foundation models & access

Bedrock exposes multiple foundation models behind one API, letting the platform match model to task — a smaller, cheaper model for high-volume classification/sentiment, a larger model for complex synthesis — without managing inference infrastructure. Model invocation is brokered through an internal gateway so that model selection, prompt templates, token budgets, and cost attribution are centralized and versioned rather than scattered across callers.

DECISION — PROVISIONED THROUGHPUT FOR PRODUCTION RESEARCH PATHS

Interactive, on-demand invocation suits experimentation and low-volume use. For production workloads with predictable, sustained volume (e.g. scheduled morning summarization of overnight filings/news), **Provisioned Throughput** reserves capacity for consistent latency and predictable cost. On-demand remains the default for spiky/exploratory use; provisioned capacity is reserved where it earns its keep (P4).

8.2 Knowledge Bases — managed RAG

A **Bedrock Knowledge Base** implements retrieval-augmented generation as a managed service (Figure 4). On ingestion it chunks source documents, generates embeddings with a Bedrock embedding model, and writes vectors to the OpenSearch Serverless vector index (§9). At query time it embeds the query, retrieves the most relevant chunks, and supplies them to the foundation model as grounding context — returning the answer *with source citations*. Grounding answers in a controlled, current corpus is what keeps generative research useful in a domain where a fabricated figure or stale filing is actively dangerous.

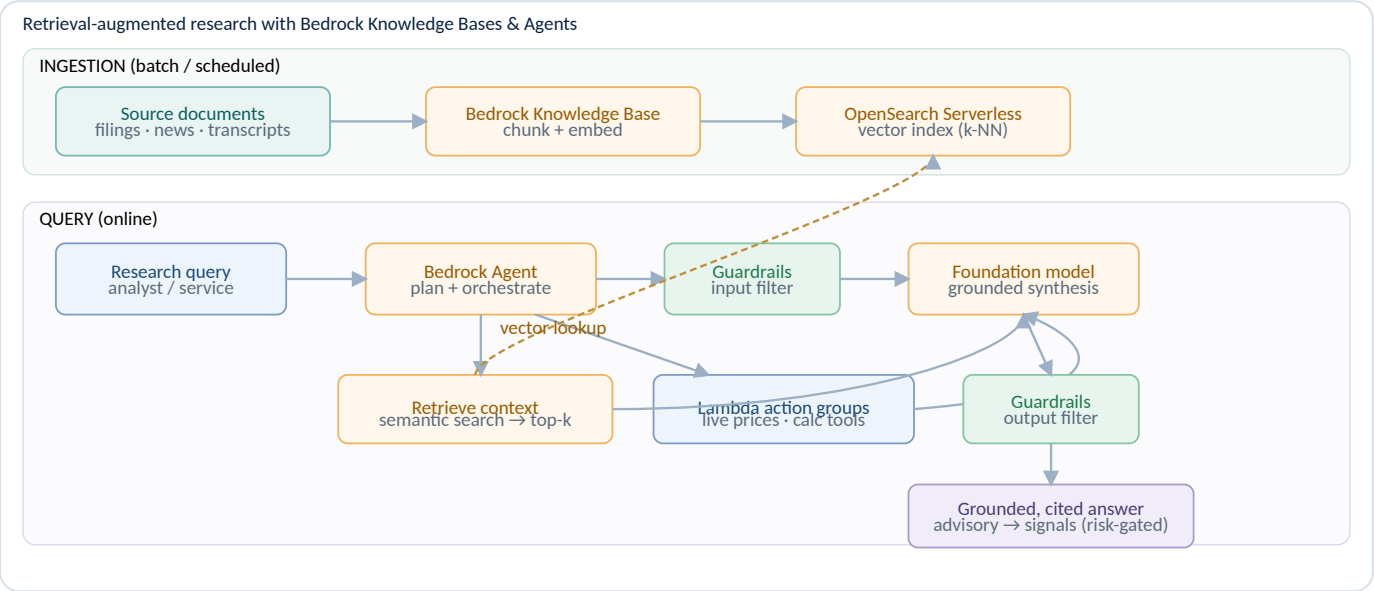


Figure 4. Generative research flow. Documents are ingested once by a Bedrock Knowledge Base — chunked, embedded, and written to an OpenSearch Serverless vector index. At query time a Bedrock Agent plans the task, retrieves relevant context from the same index, and invokes Lambda action groups for live data; the foundation model synthesizes a grounded answer with citations. Guardrails filter both input and output, and the result is advisory only — it feeds signal generation but never bypasses the risk controls.

8.3 Agents & action groups

Bedrock Agents decompose a multi-step request into a plan and execute it, invoking **action groups** — Lambda functions exposing typed tools — to fetch live data or run calculations the model cannot do reliably on its own (current prices, position lookups, deterministic financial math). This keeps the model responsible for language and reasoning while delegating facts and arithmetic to trusted code. Action-group Lambdas run with their own scoped IAM roles; an agent can therefore only reach the specific, read-oriented tools it has been granted, never arbitrary trading actions.

8.4 Guardrails

Bedrock Guardrails apply a consistent safety and compliance layer independent of the underlying model: blocking denied topics, redacting/PII-filtering sensitive entities, filtering harmful content, and applying contextual-grounding checks that flag responses insufficiently supported by retrieved sources. Guardrails are enforced on both input and output (Figure 4) and are configured centrally so policy does not drift between callers.

8.5 Representative use cases

- **Filing & transcript summarization** — condense 10-K/10-Q/earnings transcripts into structured, cited briefs.
- **News & event extraction** — detect and classify market-relevant events (guidance changes, M&A, management changes) as structured features.
- **Sentiment scoring** — entity-level sentiment feeding the signal layer as one input among many.
- **Research Q&A** — analyst queries answered from the governed corpus with citations and provenance.
- **Anomaly narration** — turn a detected market/risk anomaly into a human-readable, sourced explanation for operators.

HARD BOUNDARY — ADVISORY ONLY

No generative output is wired directly to order generation. LLM-derived signals enter the same aggregation and risk pipeline as any other input (Figure 5) and are subject to identical pre-trade risk controls. An LLM may *propose* or *summarize*; it cannot *commit*. This boundary is architectural, not merely procedural: there is no code path from a Bedrock response to the FIX gateway that bypasses the risk gate.

9 Retrieval & Search

Amazon OpenSearch Service plays two distinct roles in the platform, deliberately kept on separate deployments so a heavy analytical query can never degrade a retrieval call on the research path.

Deployment	Purpose	Configuration
Vector engine (Serverless)	Embedding store backing Bedrock Knowledge Bases (§8) and semantic search over research corpora.	OpenSearch Serverless vector collection; k-NN (HNSW) indexes; auto-scaled OCUs; isolated from the logging cluster.
Observability cluster	Centralized log, metric, and trace analytics for the whole platform (§15); ops dashboards.	Managed domain (or Serverless), ingest from CloudWatch/Firehose; retention tiers with UltraWarm for cost.

9.1 Vector store for RAG

The vector engine stores chunk embeddings and serves approximate-nearest-neighbour queries (HNSW) for low-latency semantic retrieval at scale. Bedrock Knowledge Bases manage the write path (chunk→embed→index); the read path is a vector similarity search returning the top-k chunks for grounding. Metadata filters (symbol, document type, date) constrain retrieval so a query about one issuer cannot surface another's filings.

9.2 Hybrid search

Pure vector search can miss exact identifiers (tickers, CUSIPs, defined terms). The platform combines **semantic** (vector) and **lexical** (BM25) retrieval and fuses the rankings, improving recall on both conceptual and exact-match queries — important when an analyst searches for a specific instrument or a precise regulatory phrase.

9.3 Observability & log analytics

The second deployment is the search backend for operations: structured logs, decision/audit events, and traces flow in for interactive querying and dashboards (latency percentiles, error rates, drift, fill quality). Keeping vectors and logs on separate collections/domains means a heavy historical investigation never contends with a live retrieval on the research path.

WHY OPENSEARCH SERVERLESS FOR VECTORS

It removes shard/node capacity management for a workload whose size tracks the document corpus, integrates natively as a Bedrock Knowledge Base vector store, and scales compute (OCUs) with indexing and query load. The trade-off — less low-level tuning control than a self-managed domain — is the right call for a retrieval store whose performance envelope is well within managed limits.

10 Serving & Latency Architecture

AWS Lambda is the primary inference-serving and glue-compute layer: event-driven, automatically scaled, and billed per invocation. The serving design is organized around an explicit **latency budget** and a clear rule for when to serve from Lambda versus a SageMaker endpoint.

10.1 Latency budget (hot path)

The decision path — market event to order decision — targets tens to low-hundreds of milliseconds. Indicative per-stage budget for a representative inference-driven decision (illustrative, not a guarantee; tuned per strategy):

Stage	Budget	Relative
Ingest → stream (Kinesis)	1–3 ms	
Feature computation (Lambda + online store)	3–8 ms	
Model inference (Lambda / endpoint)	10–40 ms	
Signal aggregation + strategy	2–6 ms	
Pre-trade risk checks	1–5 ms	
OMS → FIX → venue	3–10 ms	

Inference dominates the budget, which is why serving choice, provisioned concurrency, and model size are the primary latency levers. This is explicitly *not* a microsecond/HFT budget (§2.4).

10.2 Lambda inference serving

Inference Lambdas load a model artifact (or call a lightweight runtime) and return predictions for stream/API events. To eliminate cold-start variance on the trading path they run with **Provisioned Concurrency**, keeping a warm pool of initialized execution environments. Larger models or custom runtimes are packaged as **container images** (up to 10 GB); models too large or too latency-variable for Lambda are served from SageMaker endpoints instead (§10.3).

DECISION — PROVISIONED CONCURRENCY ON THE HOT PATH

On-demand Lambda is subject to cold starts (initialization latency on scale-out) that are unacceptable mid-trading. Trading-critical inference and risk Lambdas use Provisioned Concurrency sized to peak market-hours concurrency, with scheduled scaling around the session. Non-latency-critical functions (reporting, async ingestion, backfills) stay fully on-demand to retain pay-per-use economics (P4).

10.3 Lambda vs SageMaker endpoint

Serve from…	When
Lambda (provisioned concurrency)	Small/medium models, lightweight runtimes, spiky or event-driven traffic, tight per-call latency, no GPU. The default for the hot path.
SageMaker real-time endpoint	Large models, GPU inference, heavy frameworks, high sustained throughput, multi-model hosting, or A/B and shadow at the endpoint.
SageMaker Serverless Inference	Intermittent traffic that still needs SageMaker hosting without a persistently provisioned endpoint.

Serve from…	When
SageMaker Asynchronous Inference	Large payloads or long-running inference (e.g. batch research scoring) where seconds-to-minutes latency is acceptable.

10.4 APIs & orchestration

API Gateway fronts internal synchronous services (private endpoints only, §13) with auth, throttling, and request validation. **Step Functions** orchestrates multi-step workflows — research pipelines, end-of-day processing, retraining — with explicit state, retries, and error handling, so complex sequences are observable and resumable rather than buried in ad-hoc Lambda chains. Event distribution uses **EventBridge** (routing/fan-out) and **SQS** (buffering, backpressure, dead-letter queues) so a slow or failed consumer never blocks producers.

11 Signals, Strategy & Execution

This layer turns model outputs into target positions and target positions into executed orders. Its design priorities are determinism, auditability, and a hard separation between *deciding* and *executing* — with the non-bypassable risk gate (§14) sitting between them.

11.1 Signal aggregation

The **signal aggregator** consumes heterogeneous inputs — tabular model scores, time-series predictions, LLM-derived sentiment/event features, and rule-based signals — and combines them into a coherent view per instrument. It normalizes scales, applies configured weights and decay, reconciles conflicting signals, and stamps each contribution with its source model version for lineage. Inputs arrive asynchronously via streams; the aggregator maintains current state per symbol and emits an updated composite when material inputs change.

11.2 Strategy & portfolio engine

The **strategy/portfolio engine** maps composite signals to **target positions** subject to portfolio construction rules: position sizing, exposure and concentration limits, sector/factor neutrality where required, and turnover/cost awareness. It computes the delta between current and target positions and emits proposed orders. These are *proposals* — nothing reaches a venue until it clears the pre-trade risk gate. (Proprietary alpha and portfolio-optimization logic are out of scope, §2.2; this describes the surrounding mechanism.)

11.3 Order & execution management

The **Order Management System (OMS)** owns order lifecycle and state (new, pending, partial, filled, cancelled, rejected) on **DynamoDB**, chosen for low-latency strong-consistency reads/writes per order and Streams-based change capture. The **Execution Management System (EMS) / FIX gateway** maintains stateful FIX sessions to brokers/venues on dedicated ECS compute (sequence-number continuity Lambda cannot provide, §5.1), translating accepted orders into venue protocol, applying execution algorithms/routing, and capturing fills back into the OMS. Positions and P&L update on fills and propagate to risk state and the lake.

> **DECISION — PLUGGABLE EXECUTION ADAPTERS (BUILD VS BUY)**
>
> The spec assumes a built core with a thin, well-defined execution-adapter interface, so a vendor OMS/EMS or broker-provided connectivity can be substituted without disturbing the strategy and risk layers (§2.3). The risk gate sits *above* the adapter boundary, so it governs order flow regardless of whether execution is built or bought.

11.4 Backtesting & paper-trading parity

Two validation environments precede live capital. **Backtesting** replays historical data (point-in-time correct, §6.4) through the *same* feature and signal code to estimate strategy behaviour, with realistic cost/slippage/impact models. **Paper trading** runs the full live path — same features, models, aggregation, risk checks, and order generation — against live market data but routing to a simulated venue. Parity is the point: the only difference between paper and live is the execution endpoint, so behaviour observed in paper is a faithful predictor of production (P6).

ONE IMPLEMENTATION, THREE ENVIRONMENTS

Backtest, paper, and live share a single feature-definition set (Feature Store) and a single inference/aggregation/risk implementation. There is no parallel “research” codebase that can silently diverge from production — the most common source of the backtest-to-live performance gap. Differences are confined to data source (historical vs live) and execution sink (simulated vs real).

12 Real-Time Data Flows

Figure 5 traces the live path from a market event to an executed order and back to position state, with indicative per-hop latencies. It is the concrete realization of the layers in Figure 2 under hard timing and fail-closed constraints.

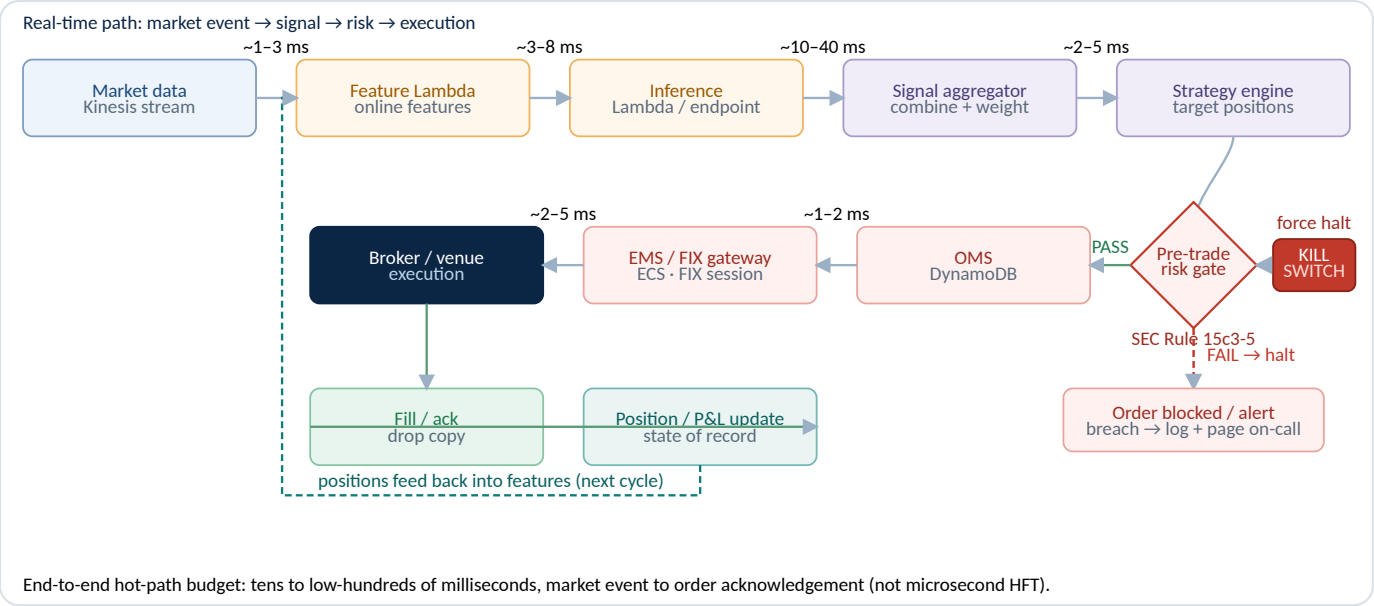


Figure 5. The live trading path with indicative per-hop latencies. A market event flows through feature computation, inference, signal aggregation, and the strategy engine to a proposed order, which must clear the non-bypassable pre-trade risk gate (SEC Rule 15c3-5). A pass routes to the OMS, FIX gateway, and venue; a breach — or a kill-switch activation — halts the order and alerts operators. Fills update the position/P&L state of record, which feeds back into feature computation.

12.1 The live path, step by step

1. A market event lands and is published to **Kinesis**, partitioned by symbol to preserve per-instrument ordering.
2. A **feature Lambda** updates online features (with state from the online store/cache) and emits the feature vector.
3. **Inference** (Lambda or endpoint) scores the event; outputs carry the model version for lineage.
4. The **signal aggregator** updates the instrument's composite; the **strategy engine** recomputes target positions and proposes orders.
5. Each proposed order enters the **pre-trade risk gate** (§14). **Pass** → OMS → FIX gateway → venue. **Breach** (or kill-switch activation) → the order is blocked, logged, and on-call is paged.
6. Fills return through the FIX gateway, update the **OMS** and **position/P&L** state of record, and feed back into features for the next cycle.

12.2 Correctness under concurrency

Orders carry **idempotency keys** so a retry or duplicate stream delivery cannot create a second live order; the OMS deduplicates on these keys. FIX session sequence numbers provide ordered, gap-detected delivery to venues. Where the platform must choose, it prefers *not sending* a duplicate to *sending* one (fail-closed, P2). Stream replay (§6.4) reconstructs any window deterministically for investigation, using recorded inputs rather than re-deriving them.

12.3 The kill switch

A **kill switch** can immediately halt order generation — globally, or scoped to a strategy or instrument. It is wired into the risk gate (Figure 5): when engaged, the gate rejects all new orders regardless of upstream state, and the platform moves

to a safe (halted) posture. Activation is available to operators and can also be triggered automatically by risk-limit or anomaly conditions (§14). Because the gate is the single mandatory chokepoint, the kill switch cannot be bypassed by any upstream component.

FAIL-CLOSED ON THE TRADING PATH

Any ambiguity, dependency failure, or breached limit on the order path resolves to *stop trading*, never to "continue and hope" (P2). A missing risk input, a stale price beyond tolerance, an unreachable dependency, or a tripped limit halts new order generation. The default state of the system in the presence of uncertainty is **not trading**.

13 Security Architecture

Security is layered so that no single control is the only line of defense (P7). Identity, network, encryption, secrets, and the AI surface are each independently hardened, and the production trading core is isolated from research, from the public internet, and from human standing access. The controls below assume a registered broker-dealer's posture: regulated data, credentialed venue connectivity, and auditable change.

13.1 Identity & access management

Every component assumes a **scoped IAM role** sized to exactly the actions and resources it needs — a feature Lambda can read its Feature Store groups and write its output stream and nothing else; the FIX gateway task role can read its broker secret and write the OMS table and nothing else. There are **no long-lived access keys** anywhere on the trading path: services use role credentials, and human access is federated through SSO with MFA and short-lived session credentials. **Permission boundaries** cap the maximum privilege any role in an account can hold, so even a mis-authored policy cannot exceed the account's ceiling, and **service control policies** at the organization level forbid whole categories of action (disabling logging, leaving the Region, creating public endpoints) regardless of local IAM. Production change is performed by pipelines (P10), not people; standing human write-access to production is the exception, time-bound and broken-glass, not the rule.

DECISION — PIPELINES HOLD PRIVILEGE, HUMANS DO NOT

The high-privilege actions that mutate production — deploying a model, changing a risk limit, rotating a venue credential — are exercised by audited CI/CD roles under change control, not by interactive operators. This shrinks the human attack surface, makes every production change reviewable and reversible, and means a compromised individual credential cannot by itself alter live trading. Emergency manual intervention (including the kill switch, §14) remains available through a separate, heavily-audited break-glass path.

13.2 Account topology & network isolation

The platform is partitioned into **separate AWS accounts** under one organization, giving hard blast-radius and identity boundaries between functions (P1, P7). Figure 6 shows the topology: a security/log-archive account that owns audit and is writable by no workload, a shared-services account for CI/CD and tooling, a data/research account where experimentation and training run, and the production trading account that holds live order flow. Research and production share no network and no credentials — an experiment cannot reach a venue session or a live limit.

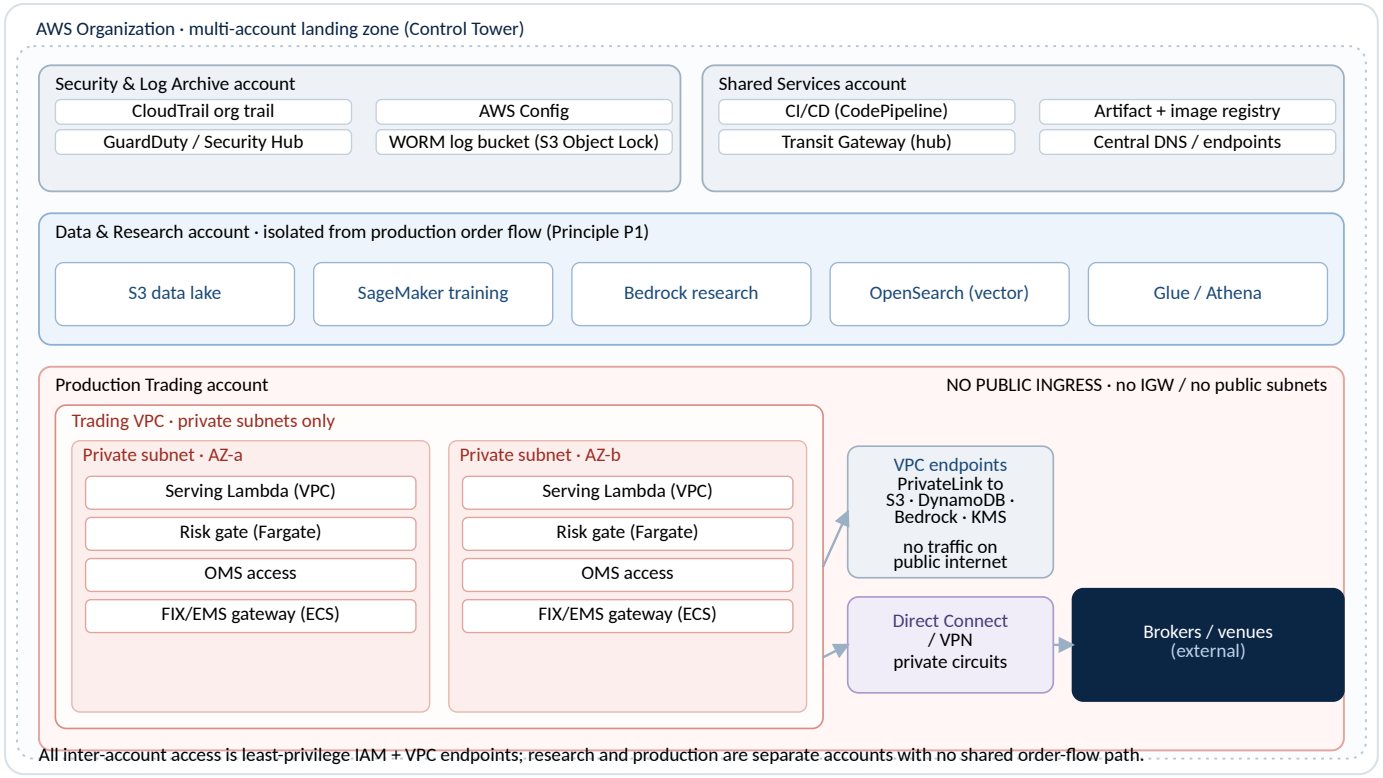


Figure 6. Multi-account network topology. Governance (security, logging) and shared services sit in dedicated accounts; research/training is fully separated from the production trading account. The Trading VPC has private subnets only across two AZs — serving, risk, OMS, and the FIX gateway — reaching AWS services exclusively through PrivateLink/VPC endpoints and brokers through Direct Connect. There is no internet gateway and no public ingress to the trading core.

The **production trading VPC has private subnets only**, spanning two Availability Zones. It has no internet gateway and no public ingress: serving, the strategy and risk engines, the OMS, and the FIX gateway reach AWS services (S3, DynamoDB, Bedrock, Secrets Manager, KMS) exclusively through **VPC endpoints / AWS PrivateLink**, keeping that traffic on the AWS network, and reach brokers and venues over **Direct Connect** (or dedicated private links) rather than the open internet. Outbound access, where unavoidable, is brokered through controlled egress with allow-lists. Security groups are default-deny and reference each other by identity rather than by CIDR, so the lateral path between tiers is explicitly enumerated rather than implied by subnet membership.

13.3 Encryption & key management

All data is **encrypted at rest and in transit by default**. At rest, S3, DynamoDB, OpenSearch, EBS, and the Feature Store use **AWS KMS** customer-managed keys, with separate keys per domain (market data, models, orders, audit) so that key policy is itself an access boundary — the order-ledger key, for instance, is usable by the audit pipeline and readable by no research role. Keys are rotated, and KMS key policies are reviewed alongside IAM. In transit, every hop is **TLS**; internal service-to-service traffic stays within the VPC over PrivateLink, and venue connectivity rides encrypted private links. Envelope encryption is used for large objects, and field-level protection is applied to the most sensitive attributes where defense in depth warrants it.

13.4 Secrets & credential handling

Venue and broker credentials, FIX session parameters, and any third-party API keys live in **AWS Secrets Manager** (or Parameter Store for non-secret config), encrypted with KMS, retrieved at runtime by narrowly-scoped roles, and never baked into images, environment text, or source. Secrets support **rotation**, and access to each secret is restricted to the single role that needs it and logged on every retrieval. This keeps the credentials that can move real money out of the reach of everything except the one component authorized to use them, and makes credential access an auditable event.

13.5 Securing the generative-AI surface

The GenAI subsystem (§8) introduces input- and data-driven risks that conventional controls do not fully cover, so it is bounded on three sides. **Amazon Bedrock Guardrails** enforce content, topic, and word policies and screen for prompt-injection patterns on both prompts and responses (P9). The **retrieval boundary** is an access boundary: the knowledge bases and indices a given assistant can query are scoped by role, so retrieval cannot become a data-exfiltration path into material the caller is not entitled to, and ingested documents are treated as untrusted input rather than trusted instructions. Finally, generative output is **advisory and never authoritative on the order path** — it informs research and surfaces context, but it cannot place, size, or release an order; the deterministic risk gate (§14) governs all execution regardless of what any model produces.

RISK — TREATING RETRIEVED OR MODEL-GENERATED TEXT AS INSTRUCTIONS

A standing hazard in RAG and agentic designs is that a retrieved document or a crafted prompt induces the model to take an unintended action or reveal restricted context. The platform mitigates this structurally, not just with prompt hygiene: scoped retrieval limits what can be reached, Guardrails filter inputs and outputs, agent tool-use is allow-listed and read-biased, and — decisively — no model output is trusted to move capital. The blast radius of a successful injection is therefore bounded to the research surface, never the execution path.

14 Risk, Compliance & Governance

This is the control plane that makes automated trading safe to operate. It has three pillars: a non-bypassable **pre-trade risk gate** that every order must clear, an **immutable audit trail** that records every decision with lineage, and a **model-governance** process that holds the ML and GenAI components to a documented standard. All three are mandatory, and all three default to stopping rather than proceeding when stressed (P2, P8, P9).

14.1 The pre-trade risk gate

Every proposed order — without exception — passes through a **pre-trade risk gate** before it can reach a venue (Figure 5). The gate is the single mandatory chokepoint on the order path, positioned *above* the execution-adapter boundary (§11.3) so it governs flow whether execution is built or bought. It enforces, at minimum: per-order and aggregate **notional and share limits**, **position and exposure limits** (per instrument, sector, and book), **price collars** and fat-finger checks against reference prices, **order-rate and message-rate throttles**, restricted-list and hard short-sale checks, and available-capital / buying-power validation. This control set is the architecture's realization of the financial-risk-management and supervisory obligations that fall on a broker-dealer providing market access (**SEC Rule 15c3-5**, the Market Access Rule), and it is designed so those checks are applied automatically, pre-trade, and cannot be disabled in the order flow.

The gate runs on **dedicated low-latency compute** (not best-effort serverless) so its checks add bounded, predictable latency to every order, and it is **fail-closed**: if a required input is missing, a dependency is unreachable, a reference price is stale beyond tolerance, or a limit cannot be evaluated, the order is rejected rather than released (P2). There is no "degraded mode" that lets orders through unchecked.

> **NON-NEGOTIABLE — THE GATE CANNOT BE BYPASSED**
>
> No strategy, model, operator convenience, or generative output may route an order around the pre-trade gate. It is enforced in the order path itself, not as advisory middleware a caller can skip. Disabling or weakening a control is a change-managed, audited action performed outside the live flow — never a runtime toggle. If the gate is unavailable, the platform does not trade.

14.2 The kill switch & automated halts

A **kill switch** can immediately halt order generation at three scopes — global, per-strategy, and per-instrument — and is wired directly into the gate: when engaged, the gate rejects all new orders in scope regardless of upstream state, and affected components move to a safe halted posture. Activation is available to operators through the break-glass path (§13.1) and is also triggered **automatically** by risk conditions: a breached aggregate loss limit, an anomalous order-rate spike, loss of a critical risk input, or a venue/connectivity fault. Because the gate is the single chokepoint, the halt is authoritative — nothing upstream can keep trading through it. Resumption is a deliberate, logged action after the triggering condition is understood and cleared.

14.3 Immutable audit & decision lineage

Every economically meaningful event is recorded to an **append-only, tamper-evident ledger**: order requests and their gate disposition (and the specific limit that blocked a rejection), fills and cancels, limit and configuration changes, model deployments, and kill-switch activations. The audit store is **write-once / immutable** — an Object-Lock (WORM) S3 tier for the durable record, backed by a cryptographically-verifiable ledger (a QLDB-style append-only journal) where chain-of-custody integrity matters — and it lives in the security/log-archive account (§13.2) that no workload role can overwrite. Crucially, each decision is **traceable end to end**: an executed order links back to the signals, model versions,

feature values, and input data snapshot that produced it (P5, P8). The standard the platform holds itself to is simple: *if it isn't logged with lineage, it didn't happen*, and any historical decision can be reconstructed and explained on demand.

14.4 Model risk governance

The ML and GenAI components are treated as **governed model risk**, not as opaque code that ships when it passes CI. Each model carries documented purpose, training data lineage, validation results, known limitations, and an accountable owner; promotion to live trading requires **independent validation** distinct from development, and a registry records what is approved for production and under what constraints. In production, models are **monitored for drift and degradation** (input distribution, prediction distribution, and realized performance, §7.5) with automatic alerting and a defined rollback path. **SageMaker Clarify** supports bias and explainability analysis, and the GenAI layer adds citation-grounded traceability (§8) so advisory output can be checked against sources. This lifecycle aligns the platform with established **model-risk-management** expectations (the SR 11-7 supervisory framework is the common reference point) — emphasizing validation, ongoing monitoring, and clear ownership across each model's life.

EXPLAINABILITY IS A CONTROL, NOT A NICETY

Because decisions must be defensible to risk officers and regulators, every automated action is reconstructable: the inputs, model versions, and signal state behind any order are retained and linkable (§14.3), Clarify provides attribution for model outputs, and RAG responses carry citations. "The model said so" is never a sufficient account; the platform can show *why*.

14.5 Trade surveillance

Beyond pre-trade limits, the platform supports **post-trade surveillance** over the immutable order and execution record to detect patterns that warrant review — potential manipulative or erroneous behaviour, limit-approach trends, and anomalous strategy activity. The OpenSearch log-analytics tier (§9.3, §15) and the audit ledger together give compliance the queryable, complete history this requires. Surveillance complements rather than replaces the pre-trade gate: the gate prevents, surveillance detects and evidences.

SCOPE — INFORMATIONAL, NOT LEGAL ADVICE

References here to SEC Rule 15c3-5, SR 11-7, short-sale and market-conduct rules, and surveillance obligations describe how the *architecture* is designed to support such controls. They are not legal or compliance advice and are not exhaustive. The applicable obligations depend on the firm's registrations, jurisdictions, venues, and instruments, and must be determined and validated by the firm's Compliance and Legal functions. The architecture provides the mechanisms — non-bypassable pre-trade checks, immutable auditability, model governance, and surveillance — that a compliance program can be built on; it does not by itself constitute that program.

15 Observability & Operations

A platform that trades autonomously must be observable in depth: operators need to see, at all times, what it is doing, how fast, and whether anything is drifting toward a limit (P8). Observability spans three signal types — metrics, traces, and logs — unified into dashboards and alarms, and backed by runbooks and an on-call rotation tuned to market hours.

15.1 Metrics & alarms

Amazon CloudWatch collects both **system metrics** (Lambda duration and concurrency, endpoint latency and errors, stream iterator age, DynamoDB throttles, queue depths) and **business metrics** (orders per second, fill rates, rejection counts by limit, end-to-end order latency, current exposure versus limits, realized and unrealized P&L). Service-level objectives are defined for the hot path — notably tail latency on the signal→risk→execution sequence (§10) — and CloudWatch alarms fire on both technical faults and **risk-relevant thresholds** (limit-approach, rejection spikes, latency-budget breaches, drift alerts from §7.5). Alarms route by severity to dashboards and paging (§15.5); risk-critical conditions can also trigger the automated halts described in §14.2.

15.2 Distributed tracing

AWS X-Ray traces requests across the serving path so a single order can be followed through feature computation, inference, aggregation, the strategy engine, the risk gate, and the FIX gateway, with per-hop timing. This makes latency-budget analysis concrete — when the hot path slows, the trace shows *which* hop — and ties directly to the latency model in §10 and the flow in Figure 5. Trace and metric context is correlated by common identifiers so a latency alarm leads straight to the offending span.

15.3 Centralized logging & analytics

Components emit **structured logs** to CloudWatch Logs, and operational and trade logs are indexed into the **Amazon OpenSearch** analytics tier (§9.3) for fast querying, correlation, and visualization. This is the same retrieval substrate used elsewhere in the platform, here turned to operations and to the surveillance queries of §14.5. Logs carry lineage identifiers (model versions, signal IDs, order IDs) so an operator can pivot from a symptom to the full decision context, and the immutable ledger (§14.3) remains the authoritative record for anything economically meaningful.

15.4 Dashboards & operational view

Real-time dashboards present platform health and trading state side by side — throughput and latency next to exposure, P&L, fill quality, and rejection reasons — so operators read system behaviour and trading behaviour as one picture. Dedicated views cover the risk posture (headroom to each major limit), model health (drift and performance), and connectivity (venue session status). The intent is that the state of the autonomous system is legible at a glance, not reconstructed after the fact.

15.5 On-call, runbooks & incident response

An **on-call rotation** covers trading hours with the urgency the domain demands; paging is wired to the severity routing in §15.1. **Runbooks** document the response to expected conditions — venue disconnect, drift breach, latency-budget violation, dependency outage, and the procedure for engaging and later releasing the kill switch. Incident response is explicitly **market-hours-aware**: the cost and urgency of a fault differ sharply inside versus outside the trading session, and the playbooks reflect that. Post-incident review feeds fixes back through the change-managed pipeline (P10).

OBSERVE THE TRADING, NOT JUST THE INFRASTRUCTURE

Conventional infra dashboards (CPU, errors, latency) are necessary but insufficient here. The signals that matter most are financial: exposure approaching a limit, an unusual rejection pattern, slipping fill quality, a model drifting. First-class business-metric observability — alongside the system metrics — is what lets operators catch a trading problem before it becomes a loss.

16 Resilience, High Availability & Disaster Recovery

The platform must survive component and infrastructure failure without losing track of orders or positions, and must default to a safe state when it cannot continue (P2). Availability is engineered at the AZ level for normal operation and at the Region level for disaster recovery, with the strictest recovery guarantees reserved for trading state — orders, positions, and the audit record — where data loss is unacceptable.

16.1 High availability within a Region

Core components run across **at least two Availability Zones**. Serverless services (Lambda, managed streaming, DynamoDB) are inherently multi-AZ; the dedicated compute that cannot be serverless — feed handlers, the FIX gateway, the risk gate — is deployed redundantly across AZs with health-checked failover, so the loss of one AZ degrades capacity but does not halt trading. Stateful venue sessions are designed to **re-establish** on failover with sequence-number continuity (§11.3), and in-path components are sized with headroom so a surviving AZ can carry the load.

16.2 Recovery objectives for trading state

Recovery targets are **tiered by data criticality**. Order and position state and the audit ledger are the most stringent: they are held in services with synchronous multi-AZ durability (DynamoDB, Object-Lock S3) and replicated for DR, targeting **near-zero RPO** — the system must not "forget" a live order or an executed fill. Reconstructable data (derived features, caches, model artifacts re-deployable from the registry) tolerates looser objectives because it can be regenerated from durable sources and the replay pipeline (§6.4). The guiding rule: *never lose authoritative trading state*; everything else can be rebuilt.

State class	Examples	Recovery posture
Authoritative trading state	Open orders, positions, P&L of record, audit ledger	Synchronous multi-AZ durability + cross-Region replication; near-zero RPO; reconciled on recovery
Operational state	OMS working data, config, limits	Multi-AZ, replicated; short RPO; restored before resuming order flow
Reconstructable data	Online features, caches, model artifacts, indices	Regenerated from durable sources / registry / replay; bounded by rebuild time, not data loss

16.3 Disaster recovery across Regions

DR uses a **warm-standby** secondary Region (§2.3): infrastructure defined as code (P10) is pre-provisioned at reduced scale, authoritative state is continuously replicated — **DynamoDB Global Tables** for order/position data and cross-Region replication for the audit and data tiers — and the standby scales up on failover. The choice of warm standby over a full active-active deployment trades a modest, bounded recovery time for materially lower cost and complexity, which suits a single-primary trading operation; the decision is revisited if availability requirements tighten.

DECISION — FAIL-CLOSED BEATS FAIL-OVER FOR THE ORDER PATH

When a fault makes correct trading uncertain, the platform's first move is to **stop trading safely** and protect state, not to race a partial failover that might double-send or trade on stale data (P2). A trading halt is recoverable; a duplicated or erroneous order may not be. DR therefore prioritizes preserving and reconciling authoritative state and resuming deliberately once integrity is confirmed — not minimizing downtime at the expense of correctness.

16.4 Market-hours-aware recovery

Recovery procedures account for the trading calendar. A failover **during market hours** prioritizes safely halting and protecting open positions, reconciling order and position state against venue records (drop-copy and venue-side position), and only then resuming — flat-or-known, never guessing. **Outside market hours**, recovery and validation can proceed without live exposure, and the system can be brought up and verified before the session opens. Reconciliation against the broker/venue source of truth is a required step before any resumption of order generation.

17 Deployment & DevOps

Everything is code and ships through reviewed, automated pipelines with environment promotion (P10). Infrastructure, application logic, models, and policies follow the same disciplined path from change to production, and the deployment mechanics for the trading path are deliberately conservative: graded rollout, the ability to shadow and roll back instantly, and a hard preference for changing the live system when markets are closed.

17.1 Infrastructure as code

All infrastructure is defined declaratively — **Terraform or AWS CDK** — and version-controlled, so accounts, networks, IAM, and services are reproducible and auditable, and the warm-standby Region (§16.3) is the same definition applied to a second target. There is no click-built production: a change to infrastructure is a reviewed commit that flows through the pipeline like any other, which is also what makes the environment boundaries of §13.2 enforceable and consistent.

17.2 Environments & promotion

Changes promote through **isolated environments mapped to separate accounts** — development, then paper/staging, then production — mirroring the account topology in Figure 6. The same artifact is promoted across stages (not rebuilt per environment), so what runs in production is what was validated upstream. Promotion to production is gated on tests, validation, and approvals, and crosses an account boundary that research and development credentials cannot (P1).

17.3 CI/CD pipelines

Pipelines build, test, and deploy on every change, running unit and integration tests, security and policy checks, and — for models — validation and approval steps (§14.4) before any production promotion. The pipeline roles are the privileged identities that mutate production (§13.1); operators do not. This makes every production change reviewable, attributable, and reversible, and keeps standing human write-access out of the live environment.

17.4 Safe deployment of serving components

Inference endpoints and serving Lambdas deploy with **graded rollout** — canary or blue/green — so a new version takes a small share of traffic under watch before full cutover, with automatic rollback on error- or latency-regression. New models can run in **shadow mode** first, scoring live traffic in parallel without acting, so their behaviour is observed against production before they influence any decision. Combined with the train/serve parity of P6, this lets a model change be de-risked on real data before it ever sizes an order.

17.5 Promoting models & strategies to capital

A strategy or model reaches live capital only after a defined progression: validated in backtest (point-in-time correct, §6.4), then run in **paper trading** on the full live path with simulated execution (§11.4), and only then promoted to production under the model-governance gate (§14.4). Because paper and live differ solely in the execution sink, paper behaviour is a faithful predictor of production (P6). Promotion is reversible by the same rollback machinery as any deployment.

DECISION — CHANGE THE LIVE TRADING PATH OUTSIDE MARKET HOURS

Deployments that touch the order path are scheduled, by default, for **outside trading hours**, when a regression carries no live-exposure cost and the system can be validated before the session opens. Canary, shadow, and instant rollback exist precisely so that the rare in-hours change is low-risk — but the standing preference is to avoid mutating a running, position-holding system mid-session unless there is a clear reason to.

18 Capacity, Scaling & Cost

The platform scales elastically where load is bursty and provisions deliberately where latency is non-negotiable (P4). Cost management follows the same split: pay-per-use for variable workloads, committed capacity where utilization is steady or latency demands it. The largest costs are not always compute — market-data licensing and provisioned low-latency capacity tend to dominate — so the cost model is built around those levers.

18.1 Scaling model

Ingestion and event processing scale with market activity: streaming shards/throughput and Lambda concurrency expand under load (e.g., the open, the close, and volatility spikes) and contract afterwards, so the platform pays for peak capacity only when it is needed (P3, P4). Research and training are inherently elastic and burst on demand. The exceptions are the components that must not scale-from-cold on the hot path — the risk gate, FIX gateway, and latency-critical inference — which run on **provisioned, warm capacity** sized for peak so that a traffic surge never pays a cold-start penalty in the order path.

18.2 Capacity for the latency-critical path

Where Lambda serves latency-sensitive inference, **Provisioned Concurrency** keeps execution environments warm, eliminating cold starts at a predictable cost (§10.2). Real-time SageMaker endpoints are sized for peak concurrent load with auto-scaling around a warm baseline. The dedicated risk and FIX compute is capacity-planned for peak order rates with headroom for AZ failover (§16.1). The principle is that the hot path is **provisioned for the worst-case moment**, not the average — latency budgets (§10) are honoured precisely when the market is busiest.

18.3 Cost levers

Each major service has a primary cost lever, summarized below. Steady-state usage moves to committed pricing; variable usage stays on-demand; the heaviest training runs exploit interruptible capacity.

Area	Primary cost driver	Lever
Bedrock	Per-token inference for GenAI/RAG; embeddings	On-demand for spiky research; **Provisioned Throughput** for steady, latency-sensitive load; right-size models per task
SageMaker	Training compute and real-time endpoint hours	**Managed Spot** for training; **Savings Plans** for steady endpoints; scale-to-baseline off-hours; serverless/async for spiky inference
OpenSearch	Provisioned cluster / OCU capacity for vector + logs	Right-size OCUs / node types; tier and retention-manage log data; separate search and logging concerns
Lambda	Invocations + duration; Provisioned Concurrency	Reserve PC only for hot-path functions; let non-latency-critical functions scale on-demand
Streaming / storage	Shard-hours / throughput; S3 volume + tiers	Scale shards to activity; S3 lifecycle to colder tiers; partition for query efficiency (§6)
Market data	Exchange/vendor **licensing & redistribution fees**	Often the largest single external cost; license to actual need; governs feed-handler design (§6)

THE BIGGEST LINE ITEM MAY NOT BE AWS

For a market-data-intensive platform, **data licensing** — exchange fees, vendor feeds, redistribution rights — is frequently the dominant recurring cost, independent of cloud spend. It also shapes architecture: which feeds are ingested, how data is stored and who may access it, and what can be retained. Treating market-data cost and entitlement as a first-class design input, not an afterthought, avoids both overspend and compliance exposure.

Specific pricing is intentionally omitted: it changes over time and depends on Region, commitment level, and negotiated terms. The architecture is designed so the levers above can be applied without structural change — committing capacity where utilization justifies it and staying on-demand elsewhere.

19 Risks, Trade-offs & Open Questions

This section makes the architecture's known risks, deliberate trade-offs, and unresolved decisions explicit, so the design review can interrogate them directly. None of these are hidden assumptions: each is a choice with a rationale and, where relevant, a mitigation or an open question for the firm to close.

19.1 Key architectural risks & mitigations

Risk	Why it matters	Mitigation in this design
Runaway / erroneous orders	Automated generation can emit bad orders at machine speed	Non-bypassable pre-trade gate, rate throttles, price collars, kill switch, fail-closed defaults (§14.1–14.2)
Backtest-to-live divergence	A strategy that works in research but not production loses money quietly	Single feature/inference implementation across backtest, paper, live; point-in-time data; paper-trading gate (P6, §11.4)
Model drift / decay	Markets change; a stale model degrades silently	Drift and performance monitoring with alerting and rollback; governed retraining (§7.5, §14.4)
Cold-start latency on hot path	A serverless cold start can blow the order latency budget	Provisioned Concurrency / warm endpoints; dedicated compute for in-path risk and FIX (§10.2, §18.2)
LLM hallucination / prompt injection	Generative output could mislead research or attempt unintended actions	RAG grounding with citations, Guardrails, scoped retrieval, advisory-only role — never on the order path (§8, §13.5)
Loss of authoritative state	Forgetting a live order or position is unacceptable	Synchronous multi-AZ durability, cross-Region replication, near-zero RPO, venue reconciliation (§16.2)
Credential / venue-access compromise	Venue credentials can move real money	Secrets Manager + KMS, per-role scoping, rotation, network isolation, pipelines-hold-privilege (§13.1, §13.4)
Audit / lineage gap	An unexplainable decision is a regulatory and risk failure	Immutable WORM/ledger record with end-to-end lineage; "if it isn't logged, it didn't happen" (P8, §14.3)

19.2 Principal trade-offs

The design makes several conscious trade-offs. It targets **seconds-to-sub-second** systematic trading rather than microsecond HFT (§2.1), accepting cloud-managed latency in exchange for elasticity, breadth of AI capability, and operational simplicity — an explicit non-goal, not a limitation overlooked. It chooses **warm-standby DR** over active-active (§16.3), accepting a bounded recovery time for materially lower cost and complexity. It favours **fail-closed** everywhere (P2), accepting halts and missed opportunities as the price of never trading on bad state. And it keeps generative AI **strictly advisory** (P9), forgoing fully-autonomous AI execution in exchange for deterministic, auditable control of capital. Each trade-off is defensible for the stated objectives; each would be revisited if those objectives changed.

19.3 Open questions to close with the firm

- **Instrument & venue scope:** exact asset classes, venues, and order types at launch — which fixes the specific 15c3-5 checks and FIX details.
- **Regulatory obligations:** the firm's registrations and jurisdictions, to be confirmed by Compliance/Legal (§14, informational only).
- **Build-vs-buy at the execution boundary:** built OMS/EMS versus a vendor or broker connectivity behind the adapter interface (§11.3).
- **Latency targets per strategy:** concrete budgets, to confirm where provisioned/dedicated capacity is required (§10, §18.2).
- **Risk-limit framework:** the specific limits, thresholds, and automated-halt triggers, set with the risk function (§14.1–14.2).
- **Market-data entitlements:** licensed feeds, redistribution rights, and retention — a primary cost and access constraint (§18.3).
- **Model-governance process:** the firm's validation, approval, and ownership standard for ML and GenAI models (§14.4).
- **DR objectives:** formal RTO/RPO targets per state class, to confirm the warm-standby posture is sufficient (§16.2–16.3).

20 Appendices

Reference material: the AWS service inventory mapped to platform roles, a glossary of recurring terms, an illustrative least-privilege IAM policy, and the standards referenced in the body.

20.1 AWS service inventory

Service	Role in the platform	Section
Amazon Bedrock (core)	GenAI inference, RAG, embeddings; Guardrails	§8, §13.5
Amazon SageMaker (core)	ML training, tuning, Feature Store, endpoints, Clarify	§7, §14.4
Amazon OpenSearch (core)	Vector search for RAG; log analytics & surveillance	§9, §15.3
AWS Lambda (core)	Event processing, feature computation, inference serving	§10
Amazon Kinesis	Real-time market-data and event streaming	§6, §12
Amazon DynamoDB	OMS / order & position state; Global Tables for DR	§11.3, §16.3
Amazon S3	Data lake; Object-Lock (WORM) audit tier	§6, §14.3
Amazon ECS	Dedicated compute: FIX gateway, in-path risk	§5.1, §11.3
Amazon EventBridge	Event routing / loose coupling	§3 (P3)
AWS KMS	Encryption keys; per-domain key isolation	§13.3
AWS Secrets Manager	Venue/broker credentials, rotation	§13.4
AWS IAM & Organizations	Scoped roles, permission boundaries, SCPs, multi-account	§13.1–13.2
CloudWatch & X-Ray	Metrics, alarms, distributed tracing	§15.1–15.2
PrivateLink & Direct Connect	Private AWS-service and venue connectivity	§13.2
Terraform / AWS CDK	Infrastructure as code; environment promotion	§17.1–17.2

20.2 Glossary

RAG	Retrieval-Augmented Generation — grounding model output in retrieved, citable source documents.
OMS / EMS	Order / Execution Management System — order lifecycle state and venue execution, respectively.
FIX	Financial Information eXchange — the standard session protocol for venue order connectivity.
Pre-trade gate	The mandatory risk-check chokepoint every order clears before reaching a venue (§14.1).
Kill switch	Operator/automated control that immediately halts order generation, globally or by scope (§14.2).
WORM	Write-Once-Read-Many — immutable storage for the tamper-evident audit record.
RPO / RTO	Recovery Point / Time Objective — tolerable data loss / downtime on recovery (§16.2).
Drift	Divergence of live input/prediction distributions or performance from training (§7.5).
Shadow mode	Running a new model on live traffic without acting, to observe before promotion (§17.4).

Point-in-time correct	Reconstructing historical data as it was known then, with no look-ahead (§6.4).
Provisioned Concurrency	Pre-warmed Lambda execution environments that remove cold starts (§10.2).
Warm standby	A pre-provisioned, reduced-scale DR Region that scales up on failover (§16.3).

20.3 Illustrative least-privilege IAM policy

A feature-computation Lambda's execution role, scoped to exactly the resources it needs — read its Feature Store group and input stream, write its output stream, decrypt with one key — and nothing more (P7, §13.1). Illustrative only; resource ARNs and conditions are placeholders.

```
// Execution role policy — feature Lambda (least privilege, illustrative)
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadMarketDataStream",
      "Effect": "Allow",
      "Action": ["kinesis:GetRecords", "kinesis:GetShardIterator", "kinesis:DescribeStream"],
      "Resource": "arn:aws:kinesis:us-east-1:ACCT:stream/market-data-equities"
    },
    {
      "Sid": "WriteFeatureStream",
      "Effect": "Allow",
      "Action": "kinesis:PutRecord",
      "Resource": "arn:aws:kinesis:us-east-1:ACCT:stream/online-features"
    },
    {
      "Sid": "ReadFeatureGroupOnly",
      "Effect": "Allow",
      "Action": "sagemaker:GetRecord",
      "Resource": "arn:aws:sagemaker:us-east-1:ACCT:feature-group/equity-online-v3"
    },
    {
      "Sid": "DecryptWithFeatureKeyOnly",
      "Effect": "Allow",
      "Action": ["kms:Decrypt", "kms:GenerateDataKey"],
      "Resource": "arn:aws:kms:us-east-1:ACCT:key/FEATURE-DOMAIN-KEY-ID"
    }
  ]
}
```

Note what is absent: no order-path actions, no venue-credential access, no broad `kinesis:*` or `kms:*`, no wildcard resources. The role can compute features and nothing else; the order-domain and audit-domain keys are unusable to it by key policy (§13.3). Permission boundaries (§13.1) cap this role's ceiling regardless of future edits.

20.4 Referenced standards & frameworks

- **SEC Rule 15c3-5** (Market Access Rule) — pre-trade financial-risk and supervisory controls for market access; the basis for the non-bypassable gate (§14.1).
- **SR 11-7** — supervisory guidance on model risk management (development, validation, ongoing monitoring, governance); reference point for §14.4.
- **FIX Protocol** — venue order-connectivity session standard (§11.3).
- **AWS Well-Architected Framework** — operational excellence, security, reliability, performance, cost, and sustainability pillars informing the design throughout.

Disclaimer. This document is a technical architecture specification. It is not financial, investment, legal, or compliance advice, and it does not constitute a recommendation to trade. Regulatory references are informational and non-exhaustive; applicable obligations depend on the firm's registrations, jurisdictions, venues, and instruments and must be validated by the firm's Compliance and Legal functions. AWS service capabilities, limits, and pricing change over time and should be confirmed against current AWS documentation. Proprietary trading logic, model details, and specific risk parameters are intentionally out of scope (§2.2).