Real-Time Risk Watcher — Architecture V3 (Implementation-Ready, Expanded)

Purpose: An FSD-style multi-agent framework for real-time portfolio risk monitoring and action planning: *Perception → Risk Computation → Event Detection → MCDA Prioritization → Reasoning (RAG) → Debate → Execution Planning/Execution/Monitoring → Learning*.

> **Key preferences incorporated:** LangChain/LangGraph | Qdrant | Alpha Vantage | SEC EDGAR | PostgreSQL | AWS + Kubernetes | Agile sprints

---

1) What is missing to start implementation (Gap List)

Area	Missing / Under-specified	Concrete additions in this document
Messaging semantics	NATS is mentioned, but reliability semantics (at-least-once vs at-most-once), dedupe, ordering, and persistence boundaries are not defined.	Defined **NATS JetStream** streams, consumer acks, replay strategy, idempotency keys, and Postgres "source of truth" storage.
Event schemas	Subjects listed, but schema versions, required fields, and correlation IDs are not formalized.	Provided JSON schema outlines for `portfolio.state`, `risk.snapshot`, `risk.event`, `risk.prioritized`, debate, and execution objects.
Agent orchestration	Agents are described, but the runtime orchestration (graph, triggers, dependencies, time budgets) is not specified.	Added **LangGraph orchestration** design (nodes, state, routing) plus trigger rules and backpressure strategy.
MCDA details	MCDA exists conceptually; scoring method, normalization, weight management, and audit trail are missing.	Defined criteria set, normalization strategy, scoring formula, governance of weights, and "regime label" rules.
RAG pipeline	Vector DB and LLM named, but ingestion, chunking, embedding model, and citation linking are not defined.	Added RAG ingestion pipeline, chunking strategy, embedding options, and reference IDs for UI traceability.
Execution safeguards	Planner/executor described, but guardrails and approvals are not formalized.	Added kill switch, policy checks, simulated mode, "advisory vs auto" modes, and broker adapter boundaries.
Data vendor constraints	Alpha Vantage rate limits and latency implications not accounted for.	Added caching, sampling strategy, and migration path to premium/alternate feeds without rewrites.

---

2) Conflicts & Fixes (Design Corrections)

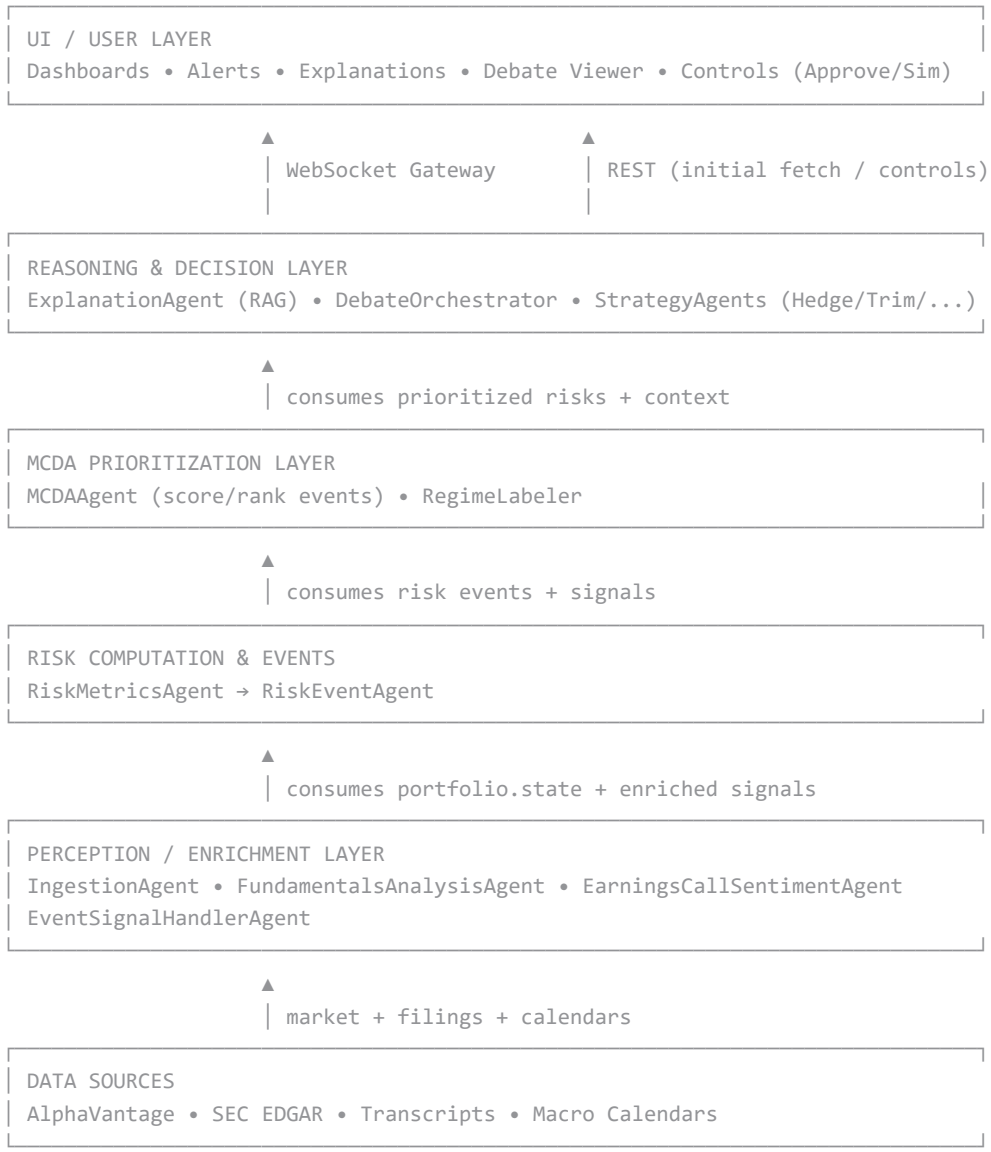
> **Conflict A:** "All messages persisted in PostgreSQL" vs NATS pub/sub (ephemeral) semantics.
> **Fix:** Use **NATS JetStream** for durable streams + consumer acks; persist canonical "facts" into Postgres with idempotent writes. Postgres remains the audit store; JetStream is the delivery + replay backbone.

> **Conflict B:** "Push/Pull/Subscribe" language can be ambiguous (NATS is push to subscribers; pull-consumers are an implementation detail).
> **Fix:** Define two delivery patterns:
>
> - **Streaming:** Subscribers receive events continuously (WebSocket gateway subscribes to subjects).
> - **Request/Response over bus:** `req.*` subject with correlation ID; responder publishes to `resp.*.<correlation_id>`.

> **Conflict C:** Model choice "Llama 5" is not an established public release at the time of writing (Meta's latest widely documented release is Llama 4).
> **Fix:** Treat "Llama5" as a *pluggable* LLM interface. Implement with a model adapter that can run **Llama 4** now, and switch to "Llama 5" later without architectural change.

---

3) Layered Architecture (Expanded)




```
Execution Sub-Layer (sidecar):
MCDA/Debate → ExecutionPlannerAgent → ExecutionEngineAgent → Broker API
                                    ↓
                          ExecutionMonitorAgent
```

---

4) Communication & Storage (Data Bus + Thought Bus)

4.1 Recommended baseline

- **Message Bus:** NATS + JetStream for durable subjects and replay.
- **Data Bus topics:** `data.*` (structured JSON, high frequency)
- **Thought Bus topics:** `thought.*` (natural language, debate transcripts, explainability)
- **Audit store:** PostgreSQL (append-only event tables + materialized views)

4.2 Streams and durability (JetStream)

Stream	Subjects	Retention	Notes
RT_RW_DATA	`data.>`	Hours–Days	High-frequency risk snapshots; keep short in JetStream, long-term in Postgres.
RT_RW_THOUGHT	`thought.>`	Days–Weeks	Debate transcripts and explanations useful for user review and learning loops.

Stream	Subjects	Retention	Notes
RT_RW_REQRESP	`req.> resp.>`	Minutes–Hours	Temporary request-response; UI can subscribe to response subject by correlation ID.

4.3 Idempotency and replay (required for correctness)

> **Rule:** Every message MUST include:
>
> - `msg_id` (UUIDv7 recommended)
> - `correlation_id` (ties a chain of actions: snapshot → event → MCDA → explanation → plan)
> - `schema_version`
> - `produced_at` (UTC ISO8601)
>
> **Consumers:** write with UPSERT semantics to Postgres using `msg_id` unique constraint to ensure exactly-once persistence even under at-least-once delivery.

---

5) Canonical Message Schemas (Implementation Starter)

> **Note:** Keep schemas in a shared repo (e.g., `/schemas`) and enforce validation in every agent (Pydantic in Python).

5.1 portfolio.state (data.portfolio.state)


```
{
  "schema_version": "1.0",
  "msg_id": "01J...UUIDV7",
  "correlation_id": "01J...UUIDV7",
  "produced_at": "2025-12-14T20:00:00Z",
  "account_id": "acct_123",
  "positions": [
    {"symbol":"NVDA","qty":100,"avg_cost":120.5,"mark":145.2,"sector":"Semis","asset_class":"Equity"},
    {"symbol":"SPY","qty":-2,"avg_cost":3.10,"mark":4.20,"asset_class":"Option","expiry":"2026-01-16","strike":50(
  ],
  "cash": 25000.0,
  "margin": {"used": 35000.0, "available": 15000.0},
  "broker_ts": "2025-12-14T19:59:58Z"
}
```


5.2 risk.snapshot (data.risk.snapshot)


```
{
  "schema_version":"1.0",
  "msg_id":"01J...",
  "correlation_id":"01J...",
  "produced_at":"2025-12-14T20:00:02Z",
  "account_id":"acct_123",
  "nav": 125000.0,
  "pnl": {"intraday": -2100.0, "intraday_pct": -0.0165},
  "drawdown": {"intraday_pct": -0.0180, "peak_nav": 127300.0},
  "exposure": {
    "gross": 1.35,
    "net": 0.92,
    "by_symbol":[{"symbol":"NVDA","net_pct":0.28},{"symbol":"SPY","net_pct":0.35}],
    "by_sector":[{"sector":"Semis","net_pct":0.28}]
  },
  "var": {"horizon_days":1,"confidence":0.99,"pct":0.045},
  "realized_vol": {"lookback_days":5,"pct":0.52},
```

```
  "limits": {"max_var_pct":0.04, "max_concentration_pct":0.25}
}
```

5.3 risk.event (data.risk.event)

```
{
  "schema_version":"1.0",
  "msg_id":"01J...",
  "correlation_id":"01J...",
  "produced_at":"2025-12-14T20:00:03Z",
  "account_id":"acct_123",
  "event_type":"VAR_BREACH",
  "severity": {"raw":0.045, "normalized":0.85},
  "context": {"symbols":["NVDA","SPY"], "window":"intraday"},
  "links": {
    "fundamentals_signal_id":"fund_01J...",
    "earnings_signal_id":"earn_01J...",
    "event_signal_id":"evt_01J..."
  }
}
```

5.4 risk.prioritized (data.risk.prioritized)

```
{
  "schema_version":"1.0",
  "msg_id":"01J...",
  "correlation_id":"01J...",
  "produced_at":"2025-12-14T20:00:04Z",
  "account_id":"acct_123",
  "regime":"ELEVATED",
  "items":[
    {"event_type":"VAR_BREACH","mcda_score":0.88,"rank":1},
    {"event_type":"CONCENTRATION","mcda_score":0.73,"rank":2}
  ],
  "mcda": {"method":"weighted_sum","weights_version":"w_2025_12_01"}
}
```

---

6) Agent Definitions (Inputs/Outputs/Tools)

Implementation principle: each agent is a containerized service with: (1) subscriptions, (2) deterministic business logic, (3) schema validation, (4) idempotent persistence, (5) observability.

Agent	Consumes (Subjects)	Publishes (Subjects)	Open-source tools / libraries	Missing details added
IngestionAgent	Broker API + Market API (polling) + optional watchlist changes	`data.portfolio.state`, `data.market.quote`	Python, requests/httpx, pydantic; caching (redis optional)	AlphaVantage throttling + caching; broker adapter interface; market-hours scheduler
RiskMetricsAgent	`data.portfolio.state` + `data.market.quote`	`data.risk.snapshot`	numpy/pandas; statsmodels (optional); pydantic	VaR method selection (hist/param/MC); volatility windows; factor exposure hooks

Agent	Consumes (Subjects)	Publishes (Subjects)	Open-source tools / libraries	Missing details added
RiskEventAgent	`data.risk.snapshot` + `data.fundamentals.signal` + `data.earnings.sentiment` + `data.event.signal`	`data.risk.event`	rules engine (simple); pydantic	Event taxonomy; normalization functions; dedupe rules; cooldown windows
MCDAAgent	`data.risk.event` (batch over window)	`data.risk.prioritized`	numpy; pydantic	Criteria definitions, weight governance, audit logs, regime mapping
FundamentalsAnalysisAgent	EDGAR filings (scheduled) + optional fundamentals API	`data.fundamentals.raw`, `data.fundamentals.signal`	SEC EDGAR JSON + XBRL parsing; bs4/lxml; pydantic	CIK mapping, XBRL fact selection, ratio computation, "trend score" rules
EarningsCallSentimentAgent	`data.earnings.transcript.raw`	`data.earnings.sentiment`	LLM (adapter); text extraction; pydantic	Transcript ingestion pipeline; sentiment schema; uncertainty score extraction
EventSignalHandlerAgent	Macro/earnings calendars	`data.event.signal`	ics parsers; cron; pydantic	Event windows, importance scoring, scope mapping (index/sector/symbol)
ExplanationAgent (RAG)	`data.risk.prioritized` + Qdrant retrieval	`thought.risk.explanation`	LangChain, Qdrant, embeddings model	Chunking, embeddings, citations/reference IDs, confidence + "unknowns" list
DebateOrchestratorAgent	`data.risk.prioritized` + `thought.risk.explanation`	`thought.debate.start`, `thought.debate.room.<id>`, `thought.debate.summary`	LangGraph (multi-agent), LLM adapter	Debate rounds, critique rules, convergence criteria, structured proposals
StrategyAgents (Hedge/Trim/Hold/Oppo)	`thought.debate.room.<id>`	`thought.action.proposal`	LangChain tools + constraints checker	Proposal schema, risk impact estimate hooks, pros/cons, cost estimation
ExecutionPlannerAgent	`thought.debate.summary` or `thought.risk.explanation`	`data.execution.plan`	Rules + optimizer (optional)	Constraints, objectives, ADV limits, slippage budget, approval state machine
ExecutionEngineAgent	`data.execution.plan`	`data.execution.order`, `data.execution.status`	Broker SDK (adapter), idempotent order keys	Dry-run/sim mode, pre-trade checks, error taxonomy, retries, kill switch
ExecutionMonitorAgent	`data.execution.status` + market quotes	`data.risk.event` (execution risk), `thought.execution.summary`	Rules + metrics	Slippage computation, partial fill policies, escalation events

7) Orchestration: LangGraph "Risk Loop" (Data Flow + Dependencies)

Why LangGraph: You want explicit, auditable orchestration: deterministic graph with state, retries, and routing across agents.

7.1 Graph State

```
{
  "account_id": "acct_123",
  "tick": "2025-12-14T20:00:00Z",
  "latest": {
    "portfolio_state_id": "01J...",
    "risk_snapshot_id": "01J...",
    "risk_events": ["01J...","01J..."],
    "prioritized_id": "01J...",
    "explanation_id": "01J...",
    "debate_id": "deb_01J...",
    "execution_plan_id": "plan_01J..."
  },
  "regime": "ELEVATED",
  "config": {"mode":"advisory","weights_version":"w_2025_12_01"}
}
```

7.2 Graph Nodes (recommended)

1. **FetchPortfolioAndQuotes** → publish `data.portfolio.state`, `data.market.quote`
2. **ComputeRiskSnapshot** → publish `data.risk.snapshot`
3. **DetectRiskEvents** → publish `data.risk.event`
4. **PrioritizeWithMCDA** → publish `data.risk.prioritized`
5. **ExplainWithRAG** → publish `thought.risk.explanation`
6. **RouteDebate** (if regime ELEVATED/CRITICAL or low confidence) → start debate
7. **PlanExecution** → publish `data.execution.plan` (with `approved=false`)
8. **AwaitApproval** (UI command) → on approval publish updated plan
9. **ExecuteAndMonitor** → orders + statuses; feed back into risk loop

> **Data flow rule:** The "fast loop" is `portfolio.state → risk.snapshot → risk.event → risk.prioritized` on a cadence you can sustain. RAG/Debate/Execution are conditional and time-budgeted to avoid stalling the fast loop.

---

8) MCDA Prioritization (Detailed)

8.1 Criteria (starter set)

Criterion	Description	Example normalization
Severity	How large is the breach (drawdown, VaR exceedance, vol spike)	min-max or sigmoid: `norm = sigmoid((x - limit)/scale)`
Immediacy	How soon can damage compound (e.g., margin stress, earnings in 24h)	time-to-event mapping (shorter = higher)
Concentration	Exposure size and single-name/sector concentration	cap at 1.0 above concentration limit
Correlation/Contagion	Risk that spreads across positions (high beta, factor loading)	factor sensitivity score (0..1)
Explainability Confidence	Confidence of signals (data quality, missingness)	penalty term (lower confidence increases priority or triggers debate)
Cost to Mitigate	Estimated hedge cost / liquidity constraints	used in ranking tie-breaker or plan selection

8.2 Scoring (weighted sum baseline)

```
mcda_score(event) = Σ_i w_i * norm_i(event)

Where:
- w_i are versioned weights (stored in Postgres, deployed via config)
- norm_i are documented normalization transforms
- every MCDA output stores weights_version + inputs for audit
```

8.3 Regime mapping (starter rules)

- **NORMAL:** top score < 0.40
- **WATCH:** 0.40–0.60
- **ELEVATED:** 0.60–0.80
- **CRITICAL:** >= 0.80 OR margin stress flagged OR multiple severe events

---

9) Reasoning & RAG with Qdrant (Detailed)

9.1 Knowledge sources to index

- Internal playbooks (hedging rules, drawdown actions, governance)
- Historical incident summaries (what happened, what worked)
- EDGAR 10-K / 10-Q risk factors and MD&A for large positions
- Macro event notes (CPI/FOMC interpretation templates)
- System configuration docs (limits, thresholds, “advisory vs auto” policy)

9.2 Ingestion and chunking

- Chunk size: 500–1,200 tokens (with overlap 10–15%)
- Metadata required: `doc_id`, `source`, `symbol`, `filing_type`, `filing_date`, `section`
- Each retrieved chunk returned with a `reference_id` so UI can show “why” and provide audit trail.

9.3 Embeddings (open-source options)

> Use any strong open embedding model (pluggable). Keep embeddings separate from the LLM choice. Store model name + version with every vector for rebuildability.

9.4 Explanation output (thought.risk.explanation)

```
{
  "schema_version":"1.0",
  "msg_id":"01J...",
  "correlation_id":"01J...",
  "produced_at":"2025-12-14T20:00:05Z",
  "agent":"ExplanationAgent",
  "regime":"ELEVATED",
  "summary":"VaR exceeded limit driven by NVDA concentration and vol spike ahead of CPI window.",
  "why_now":[
    "VaR 4.5% > limit 4.0%",
    "NVDA concentration 28% > limit 25%",
    "Macro event window: CPI in 36h"
  ],
  "uncertainties":[
    "Earnings transcript sentiment not updated (missing transcript)",
    "AlphaVantage quote stale beyond 60s due to rate limiting"
  ],
  "retrieval":[
    {"reference_id":"ref_001","title":"NVDA 10-K Risk Factors — Supply constraints","score":0.78},
    {"reference_id":"ref_002","title":"Internal Playbook — VaR breach actions","score":0.74}
```

```
  ],
  "recommended_next_steps":[
    "Trigger debate (hedge vs trim) given concentration+macro window",
    "Simulate SPX/QQQ hedge impact on VaR"
  ],
  "confidence":0.72
}
```

---

10) Debate & Strategy Layer (Structured and Safe)

10.1 Trigger logic

- Trigger debate if regime is **ELEVATED** or **CRITICAL**.
- Trigger debate if ExplanationAgent confidence < threshold (e.g., 0.65) or signals conflict (e.g., fundamentals strong but market shock severe).

10.2 StrategyAgent proposal schema

```
{
  "schema_version":"1.0",
  "msg_id":"01J...",
  "correlation_id":"deb_01J...",
  "produced_at":"2025-12-14T20:00:10Z",
  "agent":"HedgeAgent",
  "proposal":{
    "actions":[
      {"type":"HEDGE","instrument":"QQQ_PUT_SPREAD","notional":250000,"expiry_days":30}
    ],
    "expected_impact":{
      "var_reduction_pct_points":0.6,
      "net_exposure_reduction":0.10
    },
    "estimated_cost":{"premium_usd":4200,"fees_usd":35},
    "pros":["Fast VaR reduction","Keeps core positions intact"],
    "cons":["Premium cost","Gap risk if vol expands"]
  }
}
```

10.3 DebateOrchestrator rounds (recommended)

- Round 1: each StrategyAgent posts proposal
- Round 2: each agent critiques the top-2 proposals (cost, feasibility, unintended risk)
- Convergence: Orchestrator outputs 1–3 candidate plans with structured tradeoffs

---

11) Execution Engine (Planner/Executor/Monitor) — Guardrails Added

> **Safety requirement:** default to **Advisory Mode** in POC and MVP. Execution requires explicit approval in UI.

11.1 ExecutionPlannerAgent

- Produces `data.execution.plan` with constraints: max slippage, window, max %ADV, margin checks, “do-not-trade” list
- Includes `approved=false` and a required `approval_token` in advisory mode

11.2 ExecutionEngineAgent

- Broker adapter boundary: **one interface**, multiple brokers
- Implements idempotent order keys to avoid duplicate orders on retries

- Supports **SIMULATE** mode: generates hypothetical fills and slippage for testing

11.3 ExecutionMonitorAgent

- Computes realized slippage and compares to budget
- Raises new `data.risk.event` for execution risk (liquidity dried up, partial fills, margin pressure)

---

12) Data Sources & Practical Constraints

12.1 Alpha Vantage (market data)

- Implement a **rate-limit aware cache** (per-symbol TTL) to prevent exceeding limits.
- Use a "fast/slow lane" approach:
  - Fast lane: watchlist symbols, lower TTL (e.g., 30–60s)
  - Slow lane: non-core symbols, higher TTL (e.g., 5–15m)
- Abstract market provider behind `MarketDataProvider` so you can swap to premium feeds later without refactoring agents.

12.2 SEC EDGAR (fundamentals)

- Build a CIK mapping service for tickers to CIK (cached in Postgres).
- Ingest:
  - Structured facts (XBRL/JSON) for ratios
  - Risk-factor text sections for RAG

---

13) Local Development → AWS Kubernetes (Concrete Plan)

13.1 Local-first stack (docker compose)

- Services: NATS (JetStream), Postgres, Qdrant, Agent services
- Run agents as separate containers or as a single "monolith" service in POC (then split by sprint)

```
services:
  nats:
    image: nats:latest
    command: ["-js"]
  postgres:
    image: postgres:16
    environment: { POSTGRES_PASSWORD: "dev" }
  qdrant:
    image: qdrant/qdrant:latest
  risk-metrics:
    build: ./services/risk_metrics
    depends_on: [nats, postgres]
  explanation:
    build: ./services/explanation
    depends_on: [nats, qdrant, postgres]
```

13.2 AWS production (EKS + RDS + EBS)

- EKS for agent services and WebSocket gateway
- RDS Postgres for durability
- Qdrant as StatefulSet with EBS (or managed alternative later)
- Observability: Prometheus/Grafana + logs to CloudWatch

---

14) Milestones: POC → Production (Agile Sprints)

Sprint	Goal	Deliverables	Definition of Done
S1	POC risk loop (no RAG/debate/execution)	IngestionAgent + RiskMetricsAgent + RiskEventAgent	Live `risk.snapshot` and `risk.event` published + stored; unit tests for schema + idempotency
S2	MCDA prioritization + regime	MCDAAgent, weight config, regime mapping	`risk.prioritized` stable, audited (weights_version stored), replay works
S3	RAG explanations (Qdrant)	RAG ingestion pipeline, ExplanationAgent	Explanations cite retrieved refs, include uncertainties, confidence score
S4	Debate system (LangGraph)	DebateOrchestrator + StrategyAgents	Debate rooms stream to UI; proposals structured; convergence summary produced
S5	Execution planning (advisory mode)	ExecutionPlannerAgent + constraints + simulation	Plan generated with objectives/constraints; UI can approve/simulate/dismiss
S6	Broker execution adapter + monitoring	ExecutionEngineAgent + ExecutionMonitorAgent	Sim mode passes; real mode behind kill switch; execution events generate risk feedback
S7	Production readiness on AWS	EKS deploy + RDS + secrets + scaling	SLOs defined; alerts on agent health; replay + audit proven end-to-end

---

15) Implementation Starter: Repository Layout

```
risk-watcher/
  schemas/
    portfolio_state.json
    risk_snapshot.json
    risk_event.json
    risk_prioritized.json
    explanation.json
    debate_proposal.json
    execution_plan.json
  services/
    ingestion/
    risk_metrics/
    risk_event/
    mcda/
    fundamentals/
    earnings_sentiment/
    event_signal/
    explanation_rag/
    debate/
    execution_planner/
    execution_engine/
    execution_monitor/
    ws_gateway/
  infra/
    docker-compose.yml
    k8s/
      base/
      overlays/dev
      overlays/prod
  db/
    migrations/
    seed/
  docs/
    architecture.html
```

16) Final Notes

Model adapter approach (for "Llama5"): Build `LLMProvider` with a standard interface (generate, embed optional, safety settings). Run a currently available open model now; swap later without touching agent logic.

Outcome: This document is now implementation-ready: explicit schemas, durable dataflow, orchestration graph, MCDA details, RAG ingestion, execution guardrails, and an agile sprint roadmap from POC to production.