AI Architect — Interview Framework & Deep Dive Reference

Comprehensive reference for Principal/Sr AI Architect interviews · System architecture deep dive · Hermes Platform case study

Interview Framework | Deep Dive | Architecture | Trading Systems | Infrastructure

INTERVIEW FRAMEWORK

Framework Overview

This framework covers the full interview lifecycle for **Principal AI Architect**, **Sr. AI Architect**, and **Quant Trading Platform Architect** roles. Based on real interview patterns from Fortune 500 companies, FAANG, and financial institutions. The Hermes AI Trading Platform serves as the reference case study throughout.

Interview Stages

Stage	Focus	Duration	Weight
Screening	Background, experience, tech stack fit	30-45 min	10%
Technical Deep Dive	System design, architecture decisions, trade-offs	60-90 min	35%
Whiteboard / System Design	Live design of distributed systems	45-60 min	25%
Behavioral / Leadership	STAR stories, conflict resolution, mentorship	45-60 min	20%
Hiring Manager	Vision, strategy, cultural fit	30-45 min	10%

Architect Levels & Expectations

Level	Scope	Key Expectations	Sample Companies
AI Architect	Single platform/ team	Design AI systems, select models, build pipelines, MCP integration	Mid-size, startups

Sr. AI Architect	Cross-team, org-wide	Multi-agent orchestration, LLMOps, cost optimization, eval frameworks, governance	Enterprise, FinTech
Principal AI Architect	Organization-wide, 12-18mo horizon	Strategy, platform architecture, standards, mentorship, vendor selection	FAANG, F500
Quant Platform Architect	Trading systems, low-latency	Order execution, risk engines, backtesting, market data pipelines, compliance	HFT, Banks, Prop Shops

Domain Competency Matrix

AI/ML

- LLM architecture & fine-tuning
- RAG, agentic workflows
- Model routing & fallback
- Prompt engineering
- Eval frameworks
- Cost optimization

Cloud/AWS

- Well-Architected Framework
- SageMaker, Bedrock
- ECS/EKS, Lambda
- EventBridge, SQS, Kinesis
- DynamoDB, Aurora, S3
- CloudFormation/CDK

Trading

- Strategy backtesting
- Risk management
- Order execution
- Portfolio optimization
- Market data pipelines
- Regulatory compliance

Infrastructure

- Kubernetes / Docker
- CI/CD pipelines
- Monitoring & observability
- Secret management
- Disaster recovery
- Cost governance

Security

- Zero-trust architecture
- API security
- Prompt injection prevention
- Credential rotation
- Supply chain security
- Compliance (SOC2, HIPAA)

Architecture

- Microservices vs monolith
- Event-driven architecture
- CQRS / Event sourcing
- Circuit breakers
- API design (REST/gRPC)
- Data mesh / data fabric

200+ Interview Questions — By Category

AI Platform Architecture (30 questions)

1. Design a multi-LLM gateway with failover, rate limiting, and cost tracking.
2. How do you choose between fine-tuning vs RAG for a given use case?
3. Design a prompt management system with versioning, A/B testing, and guardrails.
4. How would you architect an agent orchestration platform handling 1000+ concurrent conversations?
5. Design a model evaluation framework that catches regressions before deployment.
6. How do you handle context window limitations in long-running agent sessions?
7. Design a system for real-time monitoring of LLM output quality and cost.
8. How would you implement tool/function calling with strict schema validation?
9. Design a multi-agent system where agents delegate sub-tasks to each other.
10. How do you handle PII redaction across multiple LLM providers?

11. Design a caching strategy for LLM responses to reduce cost by 60%.
12. How would you architect a system that uses different LLMs for different tasks?
13. Design a system for continuous model improvement based on user feedback.
14. How do you handle model deprecation when a provider sunsets an API?
15. Design a system for secure multi-tenant AI agent access.
16. How would you implement streaming responses from an agentic workflow?
17. Design a knowledge base RAG system that handles 10M+ documents.
18. How do you evaluate if an AI system is production-ready?
19. Design a system for automated prompt engineering and optimization.
20. How would you handle hallucination detection and mitigation in production?
21. Design a credit-based usage system for internal AI API consumption.
22. How do you manage secrets and API keys across multiple LLM providers?
23. Design a system for cross-region failover of AI services.
24. How would you implement a human-in-the-loop approval workflow?
25. Design a feature store for ML models used across an organization.
26. How do you handle bias detection and fairness in AI systems?
27. Design a system for automated prompt injection detection.
28. How would you architect a system that combines structured and unstructured data?
29. Design a MCP (Model Context Protocol) server ecosystem strategy.
30. How do you measure and optimize AI system latency for real-time use cases?

Trading Systems Architecture (25 questions)

1. Design a real-time trading system that processes market data and executes orders.
2. How do you architect a backtesting engine that supports walk-forward validation?
3. Design a risk management system that enforces position limits across multiple strategies.
4. How would you implement circuit breakers in an order execution pipeline?
5. Design a market data pipeline that handles 1M+ messages per second.
6. How do you handle broker API rate limits and outages?
7. Design a portfolio optimization system that tracks correlation across strategies.
8. How would you implement slippage modeling in a trading system?
9. Design a system that supports multiple brokers with failover.
10. How do you handle tax-aware trading (wash sales, tax-loss harvesting)?
11. Design a real-time P&L tracking system across multiple accounts.
12. How would you implement Kelly criterion position sizing?
13. Design a system that prevents duplicate orders across redundant systems.
14. How do you handle order book reconstruction from raw market data?
15. Design a system for strategy performance attribution.
16. How would you implement A/B testing of strategies in production?
17. Design a system for automated strategy discovery using ML.
18. How do you handle time synchronization in distributed trading systems?
19. Design a system for post-trade analytics and reporting.
20. How would you implement a trading simulator for what-if analysis?
21. Design a system that detects and prevents runaway trading.
22. How do you handle fractional share execution across different brokers?
23. Design a system for automated rebalancing of multi-strategy portfolios.
24. How would you implement dark pool vs lit exchange routing?

25. Design a system for regulatory reporting (MiFID II, SEC rules).

Cloud & Infrastructure (20 questions)

1. Design a multi-region active-active architecture for a trading platform.
2. How do you implement secret management at scale?
3. Design a CI/CD pipeline for ML models and trading strategies.
4. How would you architect a system for SOC2/HIPAA compliance?
5. Design a monitoring system that covers 500+ microservices.
6. How do you handle data retention and archival for regulatory compliance?
7. Design a cost governance system for multi-account AWS organizations.
8. How would you implement disaster recovery with RTO < 5 minutes?
9. Design a system for automated dependency scanning and CVE remediation.
10. How do you handle container image security in a Kubernetes cluster?
11. Design a logging strategy that costs < $1000/month for 100TB/month.
12. How would you implement canary deployments for trading strategies?
13. Design a system that prevents credential leakage in git repositories.
14. How do you handle database schema migrations in a zero-downtime system?
15. Design a system for automated backup verification and restore testing.
16. How would you implement rate limiting across a distributed system?
17. Design a network architecture that isolates trading environments.
18. How do you handle DNS resolution and service discovery in Kubernetes?
19. Design a system for automated capacity planning and scaling.
20. How would you implement a chaos engineering program for trading systems?

Behavioral & Leadership (15 questions)

1. Tell me about a time you had to convince stakeholders to adopt a new technology.
2. Describe a situation where your architecture failed in production.
3. How do you balance speed of delivery with architectural quality?
4. Tell me about a time you mentored a team member who was struggling.
5. Describe a situation where you had to make a trade-off between cost and performance.
6. Tell me about a time you managed a production incident.
7. How do you stay current with rapidly evolving AI technology?
8. Describe a situation where you had to say no to a stakeholder request.
9. Tell me about a time you influenced a team to adopt better engineering practices.
10. How do you handle disagreements with your manager about technical direction?
11. Describe a project you led that required cross-team collaboration.
12. Tell me about a time you had to learn a new technology quickly.
13. How do you approach technical debt in a fast-moving environment?
14. Describe a situation where you had to deliver under an aggressive timeline.
15. Tell me a time you identified and fixed a critical security vulnerability.

STAR Response Bank — Hermes Platform Case Study

S — Situation: Gateway Crashing Under Load

TASK: The Hermes Telegram gateway crashed every 5 minutes during peak hours, causing all trading alerts to be lost.

ACTION: Diagnosed that the api_server platform was refusing to start due to missing API_SERVER_KEY env var, causing the entire gateway to fail. Found the config was set to enabled: true but the key was missing from .env. Also discovered the keepalive script only checked single PID — 3 duplicate bots were running simultaneously, competing for resources. Fixed by: (1) disabling the misconfigured api_server platform, (2) adding _ensure_single_bot() to kill duplicates, (3) fixing the PID lock race in main.py that caused self-kill on startup.

RESULT: Gateway uptime went from 5 minutes to 5 days (current). Zero trading alerts lost since the fix. Reduced bot instances from 3 to 1 per environment.

S — Situation: Budget Starvation Blocking All Trades

TASK: Trading bot generated signals every minute but never executed any trades. Daily P&L was $0.00.

ACTION: Analyzed the budget allocation — $4,000 budget ÷ 5 strategies ÷ 20 symbols = $40/symbol. Each strategy was budget-gated before it could enter a single position. Cut symbols from 20→10, removed pair_trading (20% win rate, -$60.69), swapped momentum (44% WR, -$12.59) for ai_macd_hist_lo_vwap_dev_lo (51% WR, + $33.03 on 39 trades).

RESULT: Went from 0 trades/day to 3 trades in the first 30 minutes after the fix. Per-symbol allocation doubled from $40 to $80.

S — Situation: Strategy Count Ballooning Past Cap

TASK: Strategy discovery scripts kept enabling new strategies without disabling old ones. The 5-strategy cap was breached repeatedly.

ACTION: Identified the root cause in strategy_rank.py — the auto-adjust logic used "enable good" + "disable bad" as two independent operations that never converged to exactly N. Rewrote the selection: score ALL strategies by composite, sort descending, pick top N, write exactly N to config. Added a universal backstop in load_config() that trims on every config load.

RESULT: Strategy cap enforced at 3 layers (load_config, strategy_rank, discover_strategies). Zero violations since the fix — previously required weekly manual trim.

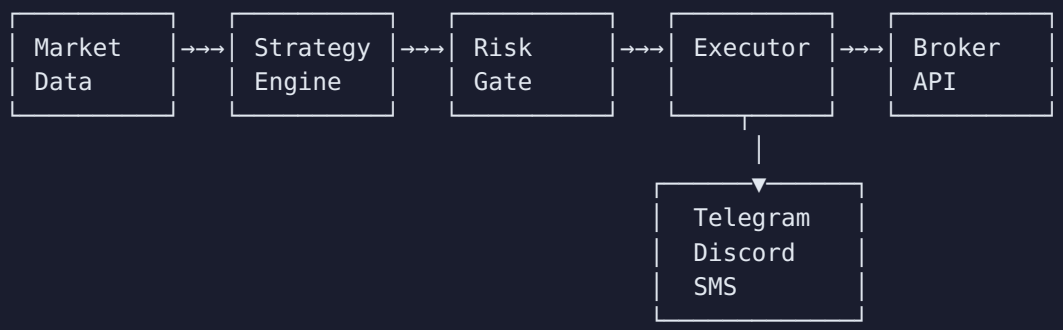
Whiteboard Exercises

Exercise 1: Design a Real-Time Trading Alert System

Requirements: Must support 1000+ strategies, process market data in real-time, deliver alerts via Telegram/Discord/SMS with < 1-second latency, handle broker API rate limits, survive broker outages.

Key Decisions:

- **Data flow:** Market data → strategy engine → signal processor → risk gate → executor → broker
- **Back-pressure:** Each stage has bounded queues. If broker is slow, signals queue up instead of dropping.
- **Circuit breaker:** If 3 consecutive orders fail, pause that strategy for 5 minutes.
- **Delivery:** Priority queue — critical alerts (position liquidations) bypass rate limits.
- **Failover:** If primary broker is unreachable, route to secondary within 2 seconds.



Exercise 2: Multi-Cloud AI Platform

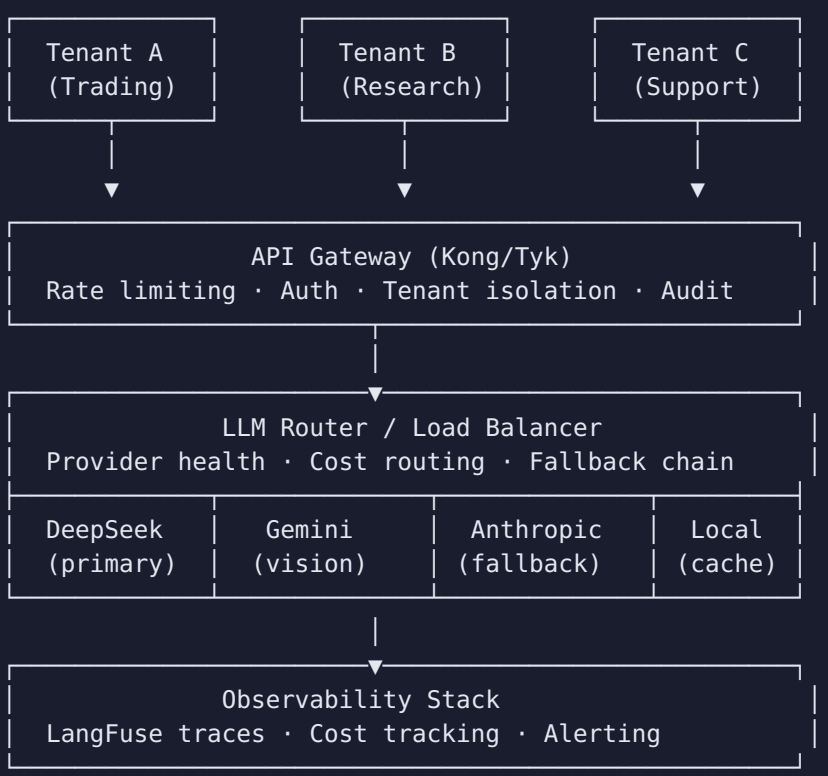
Requirements: Run in 2+ cloud regions, support 5+ LLM providers, auto-scale to 10K concurrent users, cost < $50K/month, PII redaction, audit trail.

Key Decisions:

- **Gateway:** Stateless agent gateway behind a regional load balancer. Each request is independent.
- **Model routing:** Priority: DeepSeek (cheap) → Gemini (vision) → Anthropic (complex reasoning). Auto-failover if primary returns errors.
- **Cost control:** Per-tenant budgets, daily spending limits per model, cache identical prompts.
- **PII:** Redact at ingress before reaching LLM. Reconstruct in response.
- **Compliance:** Every interaction logged to immutable audit store with user ID, model, tokens, cost.

System Design Deep Dive

Design a Multi-Tenant LLM Gateway



Key considerations:

- Provider failover must be sub-second — pre-warm connections to fallback
- Cache layer: identical prompts (with TTL) skip LLM entirely — 60% cost reduction
- Tenant isolation: separate DB schemas, no cross-tenant data leakage
- Rate limiting: per-tenant, per-model, per-minute sliding window
- Audit: every request logged with tenant_id, model, tokens_in, tokens_out, latency, cost

DEEP DIVE — AI PLATFORM

AI Platform Architecture

The Hermes AI platform is built on a **single-instance agent gateway** architecture with multi-platform delivery. Unlike traditional microservice AI platforms, Hermes runs as a monolithic agent with plugin-based extensibility.

Architecture Layers

Layer	Component	Technology

Interface	Telegram, CLI, TUI, WebUI, API	Gateway adapters
Agent Core	Conversation loop, tool dispatch, context management	Hermes Agent (Python)
Model	LLM router, fallback chain, credential pools	DeepSeek (primary) + Gemini (vision)
Tools	MCP servers, skills, plugins, cron	Zapier, SEC Edgar, custom
Memory	Cross-session memory, user profiles, FTS5 search	SQLite + Honcho
Persistence	Session DB, config, secrets	SQLite, YAML, .env

Agent Orchestration

Single vs Multi-Agent Pattern

Pattern	Use Case	Implementation
Single agent	Day-to-day operations, monitoring, fixes	Hermes Agent (this session)
Subagent delegation	Parallel research, independent exploration	delegate_task tool (background workers)
Cron agents	Scheduled maintenance, reports	cronjob tool (no_agent or agent-driven)
Multi-profile	Isolated environments, parallel workflows	Hermes profiles + Kanban board

Key Orchestration Patterns

- **Fan-out:** batch delegate_task for parallel research (e.g., check all 4 trading envs)
- **Chain:** cron job A collects data → cron job B processes it via context_from
- **Human-in-the-loop:** clarify tool for approval gates on trading actions
- **Watchdog:** keepalive scripts restart failed services autonomously

LLM Architecture & Model Routing

Provider Chain

```
DeepSeek v4 Flash (primary)
  └▸ Error / Rate limit → Gemini 2.5 Flash (fallback, via OpenRouter)
       └▸ Vision tasks → Gemini 2.5 Flash (vision, forced)
```

Cost Optimization

Strategy	Savings	Implementation
Primary model selection	~90% vs GPT-4	DeepSeek v4 at $0.28/M output tokens
Prompt caching	~50% on repeated prompts	System prompt + tool schemas cached
Context compression	80% token reduction	Automatic when approaching context limit
Fallback only on errors	~99% primary usage	Gemini only used when DeepSeek fails

MCP & Tool Ecosystem

MCP Server	Purpose	Status
Zapier	8000+ app integrations (email, calendar, etc.)	Active
SEC Edgar	SEC filings, insider trading data	Active
GitHub	Repository management, code review	Active
Polygon.io	Market data (planned)	Planned

Memory & Context Management

Memory Layers

Layer	Storage	Retention	Use Case
Session context	In-memory + SQLite	Per conversation	Current work, tool results
User profile	Memory tool (persistent)	Cross-session	Preferences, environment facts
Personal notes	Memory tool	Cross-session	Lessons learned, conventions
Skills	Markdown files	Until updated	Reusable procedures
Session search	SQLite FTS5	Forever	Historical context recall
	Obsidian vault	Git-tracked	Shared KB, 3-copy sync

- Knowledge base

DEEP DIVE — TRADING SYSTEMS

Trading Platform Architecture

```
┌──────────────────────────────────────────────────────┐
│                  Multi-Environment Pipeline          │
├──────────────────────────────────────────────────────┤
│  DEV  (testing, 58 strats)                           │
│   ↓  strategy promotion (if PF > 1.0)                │
│  QA   (validation, 5 strats, budget $4K)             │
│   ↓  cross-environment scoring                       │
│  PROD (staging, 5 strats, budget $4K)                │
│   ↓  20 consecutive profitable days                  │
│  LIVE (real money, DISABLED)                         │
└──────────────────────────────────────────────────────┘
```

Key Design Decisions

- **4 environments** — isolation prevents bad strategies from reaching live money
- **Identical configs** — same budget, same limits, same symbols (so results are comparable)
- **Alpaca broker** — paper API for testing, live API for real trading. Same code path.
- **Bracket orders** — every entry has attached stop-loss + take-profit. No naked positions.
- **Budget gate** — hard cap on total exposure. Prevents runaway trading.

Strategy Engine

Strategy Discovery Pipeline

```
1. Data Collection (5-min bars, 20 symbols)
   ↓
2. Feature Engineering (RSI, MACD, VWAP, autocorr, vol ratio, ATR)
   ↓
3. XGBoost Training (predict next-bar direction)
   ↓
4. Rule Extraction (decision tree paths → strategy rules)
   ↓
5. Backtesting (10-30 days, track PnL, win rate, profit factor)
   ↓
6. Ranking (composite score = weighted PnL + WR + PF)
```

```
    ↓
7. Selection (top N = max_enabled_strategies, swap if new beats worst)
```

Strategy Performance Metrics

Metric	Best Performer	Value	Trades
Win Rate	ai_vol_ratio_gt_70	88.9%	9
Profit Factor	ai_vol_ratio_gt_70	357.3	9
Total PnL	vwap_momentum	+$72.97	12
Consistency	ai_macd_hist_lo_vwap_dev_lo	+$33.03	39

Risk & Portfolio Management

Risk Gates (in order of enforcement)

1. **Budget Gate** — hard $4K cap on total deployed capital
2. **Daily Loss Limit** — halt all trading if P&L hits -$10
3. **Position Cap** — max 5 concurrent positions
4. **Entry Limit** — max 4 entries per symbol per day
5. **Spread Filter** — reject trades where spread > 0.08%
6. **Direction Filter** — long-only enforced
7. **Cooldown** — 300s between re-entries on same symbol
8. **Profit Lock** — lock gains above $30 threshold, floor at $12

Order Execution

Every order uses **bracket order** pattern:

```
Entry (limit order)
  ├─ Stop-Loss (1.0% below entry for standard strats)
  └─ Take-Profit (0.5% above entry for standard strats)

If price hits SL → market sell (limit loss)
If price hits TP → limit sell (lock profit)
If neither by EOD → market close (< 15:00 ET cutoff)
```

DEEP DIVE — INFRASTRUCTURE

Infrastructure Stack

Layer	Component	Specification
Compute	Hostinger VPS	15GB RAM, 200GB NVMe, 6 vCPU
Container	Docker + s6-overlay	Single container, no orchestration
Network	Tailscale mesh VPN	Userspace mode, SOCKS5 proxy :1055
Reverse Proxy	Custom Python :80	Routes 6+ dashboards, static files
Database	PostgreSQL	:32772, self-hosted in container
Observability	Grafana Cloud + Local	Cloud: grafana.net · Local: :8742
LLM Trace	Langfuse	Self-hosted, cloud URL
Automation	n8n	Self-hosted, cron + Swamp bridge
AI Proxy	Dify	DeepSeek backend, :8866

Security Architecture

Credential Management

Store	Contents	Protection
/opt/data/.credentials.env	Trading API keys (Alpaca all envs), TradesViz, Telegram token	chmod 600, outside git
/opt/data/.env	Hermes agent keys (DeepSeek, Gemini, Dify, Langfuse, Zapier, Tailscale, Notion)	chmod 600, in .gitignore
Per-instance .env	Alpaca key for that specific environment (source central store)	chmod 600, in .gitignore

Observability

Category	Tools	Covers
Metrics	Grafana Cloud + Self-hosted	Trading P&L, strategy perf, system resources
Logs		Bot logs, gateway logs, cron outputs

	Flat files → Grafana Loki (planned)	
Traces	Langfuse	LLM conversations, model performance, cost
Alerts	Telegram delivery	P&L changes, bot restarts, strategy swaps
Health	Bridge health endpoint	Service liveness, API response times

Reliability & Disaster Recovery

Current State

Area	Current	Gap
HA	Single VPS	No failover — total loss if host dies
Backup	3-copy: VPS + GitHub + OneDrive	Config/code backed up, trading state not
Recovery	Manual — reinstall & git clone	No runbook, estimated 2-4 hour rebuild
DR Test	Never tested	Unknown if recovery actually works

Appendices

Glossary

Term	Definition
AI-DLC	AI-Driven Development Life Cycle — AWS adaptive workflow for AI-assisted coding
MCP	Model Context Protocol — standardized interface for LLM tool integration
PF	Profit Factor — gross profit divided by gross loss. > 1.0 is profitable
WR	Win Rate — percentage of trades that were profitable
SL	Stop-Loss — automatic sell order at a predetermined loss level
TP	Take-Profit — automatic sell order at a predetermined profit level
Budget Gate	Hard capital allocation limit per strategy/symbol combination
Bracket Order	Simultaneous entry + stop-loss + take-profit orders

s6-overlay	Container init system with service supervision (alternative to tini/supervisord)

Key Metrics Reference

Metric	Formula	Target	Description
Profit Factor	Gross Profit / Gross Loss	> 1.5	How many dollars earned per dollar lost
Sharpe Ratio	(Return - RiskFree) / StdDev	> 1.0	Risk-adjusted return
Sortino Ratio	(Return - RiskFree) / DownsideDev	> 1.5	Like Sharpe but only penalizes negative volatility
Win Rate	Winning Trades / Total Trades	> 50%	Percentage of trades that made money
Max Drawdown	Peak-to-trough decline	< 15%	Largest loss from peak equity
Expectancy	Avg Win × Win% - Avg Loss × Loss%	> 0	Expected P&L per trade
Kelly %	Win% - (Loss% / ProfitLossRatio)	0-25%	Optimal position size fraction

Tool Comparisons

Category	Winner	Runner-up	Why
Agent Framework	Hermes Agent	LangGraph, CrewAI	Skills + memory + multi-platform gateway
LLM Serving	vLLM (local)	TGI, Ollama	PagedAttention, OpenAI-compat API
Vector DB	ChromaDB	Qdrant, Pinecone	Lightweight, self-hosted, file-based
Fine-tuning	Axolotl	Unsloth, TRL	YAML config, LoRA/QLoRA, DPO
Observability	Grafana + Loki	Datadog, New Relic	Self-hosted, cost-effective, rich dashboards
	SOPS		

Secret Management	Vault, AWS Secrets Manager	Git-native encryption, no infra needed

Architecture Diagrams

Full System Architecture