ChatGPT

Agentic Infrastructure & Hyper-Personalisation: Building an AI-Ready Stack for Investors

Transforming Savers into Investors demands an AI-powered stack that spans mid-tier APIs to customer-facing chat. This global strategy emphasizes technology as a trusted enabler across all channels and partners – not just within one country. Over the next 3–5 years, the firm will modernize its API fabric and CX infrastructure to be "agent-ready" and hyper-personalised, while ensuring security, compliance and scalability. The vision: an always-on *financial companion* that proactively helps each investor, powered by autonomous agents and real-time data.

A. Agentic-Ready API Foundation

APIs as Agent Interfaces

LLM-based agents will interact with our systems by calling APIs like programmable tools. To enable this, our APIs must carry **richer semantics** and schema metadata. For example, embedding detailed "action schemas" in OpenAPI definitions (unique action names, purpose, examples, schemas) helps agents map user intents to API calls [1] [2]. AWS already supports this pattern: Bedrock's OpenAPI *Action Groups* let an LLM agent discover and invoke endpoints safely [3] [4]. In practice, we will publish APIs with enhanced OpenAPI/JSON-Schema (or GraphQL contracts) and descriptive annotations (e.g. JSON-LD links to business ontology) so that an agent can plan multi-step workflows. Toolkits like LangChain demonstrate this approach: its OpenAPI Agent automatically queries any service conforming to Swagger/JSON specs [5]. We will adopt similar frameworks (LangChain, Semantic Kernel) to orchestrate calls, handle tokens and parse structured JSON results. Designing "agent-native" APIs also means supporting conversational patterns (ReAct-style prompts and function-calling models) so agents can verify actions or ask clarifying questions before execution.

Security for Autonomous Access

Autonomous agents require **finely-grained, context-aware authorization** – far beyond static user logins. Instead of blanket privileges, each agent call should carry a narrowly-scoped token and pass a risk check. We will extend OAuth2/OIDC with agent-specific credentials and delegation flows, as suggested by recent research [6] [7]. For example, a user could register an AI agent as a delegated client (via an OAuth consent screen) for "portfolio-read" or "transaction-initiate" rights [8] [7]. User-Managed Access (UMA) profiles can let clients dynamically request tokens for multiple services under a single consent flow [9]. We will also use policy engines (OPA/XACML) that evaluate agent requests in context – checking device, time, location or transaction amount. Recent surveys of "Internet of Agents" security emphasize **continuous authentication and task-based policies** [10] [11]. For instance, if an agent shifts from low-risk queries to executing trades, the system can prompt for re-authentication or risk scoring. We will pilot context-aware risk models (similar to bank fraud scoring) and enforce just-in-time MFA or ephemeral scopes. Long-term, decentralized identity (DIDs, verifiable credentials or digital wallets) could bind agent identity to users without central passwords

[12]. These measures ensure agents have the *minimal necessary* rights for each call, avoiding over-permissioning.

Composability via Event-Driven Orchestration

To compose complex workflows, our API/messaging layer will be **event-driven** and modular. Streaming platforms (e.g. Apache Kafka or AWS Kinesis) will feed live data into agent pipelines, ensuring no stale information. In practice, we'll deploy a real-time backbone: use Confluent Cloud (Kafka + ksqlDB/Flink) to ingest transactions, market feeds, and user events. As Kai Waehner notes, *"Agentic AI requires an event-driven architecture"* because agents act on continuous data streams [13] [14]. Indeed, an event-driven system **decouples** producers and consumers, so an agent can subscribe to relevant topics and react instantly.

Figure: An event-driven streaming architecture (Kafka + Flink) decouples data producers and agent consumers, ensuring agents see the latest state [13].

For example, a "new account" event can automatically trigger an agent workflow (welcome email, risk profile survey, chatbot outreach) using serverless functions (AWS Lambda or K8s functions) or a managed framework (AWS Step Functions or Temporal) as the orchestrator. We will implement intent-routing middleware so that when an agent or user intent arises (e.g. "rebalance portfolio"), it triggers the correct microservices in sequence. This might leverage FinOps patterns or custom orchestrators built on event bridges. Containerized services (on AWS EKS, ECS or Lambda) will host domain-specific actions (e.g. compliance check service, quote fetcher, reporting). By aligning our API design with event-driven "publish-subscribe" patterns, each tool or function can be invoked dynamically, letting agents compose workflows on the fly.

New Protocols & Decentralized Interfaces

Emerging multi-agent standards will shape our interface design. We're monitoring protocols like Anthropic's **MCP** and Google's **Agent2Agent (A2A)**, which define how agents discover tools and talk to each other securely [15] [16]. For instance, MCP provides an HTTP+OAuth interface for LLMs to retrieve tool descriptions and JSON-RPC for execution [15]. Google's A2A protocol uses JSON "Agent Cards" and long-lived tasks, enabling agents from different vendors to interoperate over HTTP/SSE with enterprise-grade auth [16] [17]. We will design our mid-tier to be *MCP/A2A-compatible*, so third-party assistants (Alexa, Siri, etc.) or future agent platforms can invoke our services natively.

Architecturally, we will avoid single points of failure by distributing the agent interface. For example, multiple API gateways (Kubernetes Ingress, API Management, and service meshes) can serve different consumer domains. A **service mesh** (Envoy/Istio) can provide mTLS, traffic shaping and policy enforcement across our internal microservices as well as partner networks. In fact, proxy technology is already evolving for AI: Tetrate's new Envoy AI Gateway uses Envoy Proxy to unify calls to AWS Bedrock, OpenAI, etc., with per-client rate limiting and fine-grained auth [18]. We will leverage similar proxies or API gateways to centralize LLM service access. In short, our design will be microservices-based and mesh-enabled, so agents can route to any tool or partner service via resilient, programmable proxies.

AI-Native Middleware Vision

Looking ahead, we envision an **agent-augmented API gateway** – the "nervous system" of our AI stack. Such a gateway would automatically integrate with LLM orchestration layers, handling tasks like query routing, prompt assembly and result validation. For example, it could identify on-the-fly which services (internal or external) are needed for a given user intent and invoke them in the proper context. This gateway might host plug-in modules (WASM extensions or serverless functions) to preprocess prompts (sanitizing user data, inserting API keys) and postprocess outputs (redacting PII, signing responses). By embedding AI workflows into the gateway, we ensure that all interactions – internal, partner, or customer-facing – flow through a controlled, consistent layer. This future-proof stack would allow new agents to plug in easily; think of it as an "API control center" that logs every agent decision, enforces global policies, and seamlessly connects on-prem and cloud services. In practice we'll start by augmenting our API gateway (e.g. AWS API Gateway or Kong) with LLM integration (as seen in Envoy AI Gateway [18]) and gradually expose meta-data endpoints for capability discovery.

B. Hyper-Personalised, Agentic CX

Real-Time Hyper-Personalisation

Investors expect the experience to feel **tailored and immediate**. We will build a data architecture that fuses streaming behavior with historical profiles. Confluent and Databricks jointly illustrate this "stream-fed lakehouse" pattern: Kafka streams can be automatically materialized into Apache Iceberg tables for analytics [19] [20]. In practice, customer clicks, market data and CRM events will flow through Kafka (and be cleaned/governed via Tableflow), landing into our Iceberg/Delta lakehouse on S3. Query engines (Trino, Databricks, Athena) will access these tables for both batch and real-time querying. This unified store lets us feed real-time features into ML models that score recommendations or risk profiles.

The personalization engine itself will combine **business rules, ML models and RAG**. Rules enforce compliance and branding (e.g. "never recommend product X to VIP customers without manager approval"). ML models (built with AWS SageMaker or Databricks) will segment investors and predict needs (propensity to invest, churn risk). LLM-driven RAG agents will then generate the *tailored content* (advice, product suggestions, email copy) by retrieving relevant documents (portfolio data, investment research, marketing copy) into the prompt. For example, a RAG pipeline might pull the user's financial goals (from their profile table) plus the latest ESG report, and generate a natural-language recommendation. This composite approach (rules+ML+RAG) is becoming common practice: hyper-personalization means *"collecting and analyzing swaths of data"* and using AI to forecast needs in real time [21] [22].

[23] [24] – as Confluent notes, customers demand *"instant access to tailor-made, meaningful content"* delivered at the right moment [23] [24]. By streaming behavioral signals (web clicks, app usage, emails) and combining them with analytics, we can adapt journey content on the fly. For instance, if a user browses fixed-income funds, the next chatbot interaction or email can immediately mention bond strategies. And as the bank channel and global markets move (even seconds), our models adjust recommendations without waiting for nightly batch. Over time, this adaptive personalization will increase loyalty and ROI – just as studies show personalized engagement boosts satisfaction and lifetime value [25] [23].

Omni-Channel Conversational Layer

Investors interact through many channels (web chat, mobile app, voice assistants, email, messaging). We will establish a unified conversational platform that preserves context across them. In practice, this means a central "dialogue memory" store (e.g. DynamoDB or vector DB) keyed by user ID and session. Agent orchestration frameworks (LangChain, Microsoft Semantic Kernel) will manage multi-turn flows and can inject channel-specific formatters (text for chat, voice prompts for Alexa, rich cards for apps). For example, Semantic Kernel's chat skills or LangChain's ConversationChain can maintain a persona and history state across API calls.

This unified layer will incorporate LLMs for natural language understanding and generation. When a user says "Tell me about my portfolio," the orchestrator routes that intent to a finance agent, which may call our portfolio API (via OpenAPI), fetch data, and feed an LLM to craft a reply. Libraries like LangChain allow us to plug in memory buffers, chain logic, and tool invocation. We'll also leverage specialized bot platforms where needed (e.g. Amazon Connect or Twilio for phone bots, web SDKs for site chat) but funnel them into a common backend. Crucially, if the user switches channels (say, moves from SMS to phone), our system will resume the same conversation state seamlessly (similar to how modern contact centers integrate channels). Tone will be managed by templating – for example, email responses use a formal style while a chat can be more casual.

Agentic Customer Interactions

Looking further ahead, investors may use *their own* AI assistants to interface with us. Imagine a user saying to their phone's AI: "Check if my investments meet my green portfolio goal," and our systems responding through the user's personal agent. To support this, we'll expose secure capabilities via standards that external agents understand. Google's A2A and similar protocols can allow third-party assistants to discover our "agent cards" (services we offer) and invoke tasks on behalf of an authenticated user [16] [17]. We will also build skills or Actions for major platforms (e.g. an "Advisor Copilot" for Alexa/AWS, Google Assistant, Apple's future intelligence). Technically, this means implementing OAuth2/OIDC flows so the external agent gets a short-lived token for our API on the user's behalf.

Additionally, our APIs must be accessible via voice and IoT endpoints. For example, connecting to an existing voice bot (using AWS Lambda or Azure Functions) that consumes our Chat API. We may need to support streaming/gRPC to handle continuous voice dialogs. Over time, standards like **OpenVoice** for banking might emerge (e.g. secure banking voice skills). We will align with emerging specs and ensure our OpenAPI endpoints can be called by any authorized digital assistant. In short, we treat external personal agents as clients: they authenticate through our gateway, specify the user's intent (with provenance), and receive signed data in response.

Trust, Transparency & Compliance

Financial conversations demand high trust. Our agentic CX will be fully auditable and explainable. Every output (especially LLM-generated advice) will cite its sources or rationale wherever possible. For instance, when giving portfolio recommendations, the system may attach a bullet list of market data facts or policy citations used in the decision. We will log entire chat transcripts, prompt history, and model parameters to satisfy regulators' record-keeping (e.g. investment advice rules). Sensitive user data will never be sent to the model in plain text; instead, agents will work on abstracted profiles or encrypted queries. This avoids

prompt-injection risks – a known LLM security concern [26] – by sanitizing inputs and storing private info securely.

On regulation: we comply globally by design. In the EU, GDPR grants the user "right to explanation," so we will ensure any profiling (even via AI) is transparent and opt-in. We will obtain consent before any personalized analysis, and allow users to review/delete their data. We'll also follow EU Digital Markets Act guidelines: if platforms require fair access, we'll be ready to interoperate (e.g. allow regulated aggregators to query user-permitted investment data). Digital identity wallets (under eIDAS2 or emerging frameworks) may be integrated so customers can authenticate without sharing raw credentials. For example, a user could sign an "intent to invest" payload with their European eID wallet, which we verify against user identity before acting. Such signed intents and verifiable credentials will be pivotal for secure autonomous transactions.

In summary, our end-state is a **"trusted, intelligent companion"**: always-on, context-aware, and transparent. It proactively notifies clients of opportunities (e.g. "You've reached your savings goal – consider these investment options"), keeps investment literacy high with plain-language tips, and escalates to human advisors when needed. We remain agent-ready (MCP/A2A standards), customer-centric (personal data in service of them), and platform-friendly (open APIs for partners) simultaneously – all while upholding the highest regulatory standards.

Implementation Roadmap & Architecture

1. Modernize Data & Messaging (Year 1–2): Deploy a cloud streaming platform (Confluent Cloud or Amazon MSK with Kafka Connect). Ingest real-time customer events and market feeds. Use Tableflow or Kafka Connect to materialize streams into a governed Iceberg/Delta lakehouse (on AWS S3 or Snowflake). This creates the "single source" of streaming truth [19] [20].

2. Build Secure API Layer (Year 1–2): Introduce an API gateway (e.g. AWS API Gateway or Kong) in front of microservices. Integrate OAuth2.0/OIDC (Cognito/Auth0) with support for agent clients. Tag APIs with OpenAPI metadata; begin supplying JSON Schema definitions for LLM use [1] [4]. Enforce mTLS and zero-trust policies via a service mesh (Istio/Envoy) for east-west traffic.

3. Develop Base Agentic Services (Year 2–3): Create reusable *tools* as microservices (e.g. Portfolio Query, Market Data, Compliance Check). Each tool is exposed via the API layer. Assemble them using AWS Lambda/ Step Functions or Kubernetes (for long-running tasks). Build simple AI agents using LangChain or AWS Strands: e.g. a "Research Assistant" agent that chains two or three tools. Use Amazon Bedrock and OpenAI APIs as LLM backends. Tetrate's Envoy AI Gateway shows how to unify LLM access – we'll adopt similar proxies for multi-provider support [18].

4. Launch Pilot Hyper-Personalized Channels (Year 2–4): Roll out a next-gen chat/voice assistant for existing customers. For instance, an in-app chat that uses a Retrieval-Augmented Generation (RAG) chain: it loads the user's profile (from lakehouse), retrieves relevant documents (investment guides), and answers queries. Integrate this with our CRM and email system. In parallel, start small with voice (e.g. an Alexa skill linked to our services, using OAuth). Iterate using data science teams to refine ML models for personalization.

5. Expand to Agent Ecosystem (Year 3–5): Expose public APIs for partner agents and third-party assistants (using MCP/A2A compatibility). Participate in industry working groups on agent interoperability. Build or integrate digital identity (verifiable credential) capabilities for secure user-agent handoff. Ensure all new features (AI-driven advice, proactive notifications) have audit trails and human-in-the-loop override.

Tech Stack & Vendors:
- **Cloud & Compute:** AWS (EC2/ECS, Lambda, Step Functions, API Gateway), or hybrid with Azure/AWS multi-cloud options. Use AWS Bedrock/OpenAI for LLM, plus LangChain/Semantic Kernel as frameworks.
- **Data & Streaming:** Confluent Cloud (Kafka, ksqlDB/Flink) or AWS MSK + Kinesis + KSQL for ingestion; Confluent Tableflow/Snowflake/Databricks for lakehouse tables [27] [20]. Vector DB (Pinecone or Weaviate) for embeddings.
- **Observability:** Service Mesh (Istio/Tetrate) and Envoy proxies for traffic management and LLM gateway needs [18]. OpenTelemetry for logging agent actions.
- **Conversational UI:** Twilio or Amazon Connect for multi-channel, with a central conversation backplane.
- **Security:** OAuth2/OIDC (Auth0/Cognito), fine-grained roles via OPA or AWS IAM policies. Ongoing compliance audits with data catalogs (e.g. AWS Glue/Data Catalog).

Example Architecture:
1. **Event Bus:** Kafka topics ingest trades, interactions, market updates.
2. **Stream Processing:** Flink/KSQL materializes features (e.g. transaction summaries) into Iceberg tables via Tableflow [19] [20]. Real-time ML scoring (fraud, churn) also on streams.
3. **API & Agents:** An Envoy gateway fronts microservices. Agent controllers (in LangChain) orchestrate API calls. For instance, a "rebalancing agent" on trigger (end-of-month) fetches portfolio via a REST API, runs an LLM to propose adjustments, then calls a Trading API to execute with user consent.
4. **Conversational Layer:** A chat session is managed by an orchestrator service that holds session memory. On user utterance, the orchestrator invokes an intent recognizer (LLM) and routes to an agent or scripted flow. Responses may call back-end APIs or RAG templates and are rendered as chat messages, voice prompts or emails.

Conclusion

By investing in a composable, event-driven mid-tier and a hyper-personalized front-end, our firm will set a **new standard** in investor experience. We'll be "agent-ready" – with OpenAPI-augmented services, secure agent protocols, and AI-native gateways – while treating each customer as a unique individual. Through real-time data streaming (Kafka + Iceberg) and orchestration frameworks (LangChain, Semantic Kernel), we enable AI agents to operate at enterprise scale. This future-ready stack not only differentiates us technologically, but also deepens trust: every recommendation will be tailored, explainable and compliant. As agents become the "new apps," our firm will evolve into a platform where autonomous assistants and financial advisors collaborate seamlessly, empowering millions of savers globally to invest smarter [14] [23].

Sources: Extensive research on AI agents and architectures [1] [4] [15] [10] [13] [18] [16] [23] [21] [20] supports these recommendations. (All details above are derived from expert blogs, whitepapers and standards documents.)

---

[1] [2] [4] [26] Exposing OpenAPI as MCP Tools - Semantics Matter – ceposta Technology Blog
https://blog.christianposta.com/semantics-matter-exposing-openapi-as-mcp-tools/

[3] Define OpenAPI schemas for your agent's action groups in Amazon Bedrock - Amazon Bedrock
https://docs.aws.amazon.com/bedrock/latest/userguide/agents-api-schema.html

[5] OpenAPI Toolkit | LangChain
https://python.langchain.com/docs/integrations/tools/openapi/

[6] [7] [8] [9] Authenticated Delegation and Authorized AI Agents
https://arxiv.org/html/2501.09674v1

[10] [11] [12] [15] Security of Internet of Agents: Attacks and Countermeasures
https://arxiv.org/html/2505.08807v1

[13] How Apache Kafka and Flink Power Event-Driven Agentic AI in Real Time - Kai Waehner
https://www.kai-waehner.de/blog/2025/04/14/how-apache-kafka-and-flink-power-event-driven-agentic-ai-in-real-time/

[14] The Future of AI Agents is Event-Driven | by Sean Falconer | Medium
https://seanfalconer.medium.com/the-future-of-ai-agents-is-event-driven-9e25124060d6

[16] [17] Announcing the Agent2Agent Protocol (A2A) - Google Developers Blog
https://developers.googleblog.com/en/a2a-a-new-era-of-agent-interoperability/

[18] The Beginning of a Scalable GenAI Future: The First Release of Envoy AI Gateway
https://tetrate.io/blog/the-beginning-of-a-scalable-genai-future-the-first-release-of-envoy-ai-gateway

[19] [27] A Real-time Open Lakehouse with Redpanda and Databricks | Databricks Blog
https://www.databricks.com/blog/real-time-open-lakehouse-redpanda-and-databricks

[20] Introducing Tableflow: Unifying Streaming and Analytics
https://www.confluent.io/blog/introducing-tableflow/

[21] [22] [25] Hyper-personalization in banking: The new imperative | Luxoft
https://www.luxoft.com/blog/luxoft-hyper-personalization-future-of-banking

[23] [24] Real-Time Customer Personalization at Scale
https://www.confluent.io/use-case/real-time-customer-personalization/