VANTIQ Developers Guide Series – Introduction to VANTIQ Development

Version 2.0

30/03/2026

Table of Contents

List of Figures

1. Introduction to Vantiq Applications

Vantiq is a platform for building and executing Real-Time, Event-Driven, Situational-Aware Intelligent Systems. The Vantiq platform provides both a development and an execution run-time that has been specifically designed to support this style of application.

1.1 What are Real-Time, Event-Driven, Situationally Aware Intelligent Systems?

These are advanced, adaptive software systems designed to respond quickly and intelligently to changes in their environment. They combine four key characteristics—real-time responsiveness, event-driven architecture, situational awareness, and intelligence—to enable autonomous or semi-autonomous decision-making. They:

- React instantly (**real-time**) to events as they happen. This is crucial in scenarios where delay can lead to missed opportunities or risks, such as autonomous vehicles avoiding collisions, fraud detection in financial transactions, or smart grid systems balancing power demand
- Are triggered by specific events in the environment (**event-driven**). Events are discrete changes in state, like a sensor detecting motion, a user clicking a button, or a server receiving a request.
- Understand the context or situation in which those events occur (**situational awareness**). These systems don't just react blindly to events—they interpret them within the broader context. Situational awareness includes factors such as:
  - **Spatial context**: Where is the event occurring?
  - **Temporal context**: When is it happening (time of day, season)?
  - **Historical context**: Has this happened before? What patterns exist?
  - **Environmental context**: What's the weather, traffic, or surrounding conditions
- Use that understanding to make smart, adaptive decisions (**intelligent**). At the heart of these systems is **intelligence**—the ability to make decisions, solve problems, and adapt over time. The more advanced the intelligence, the more capable the system is of handling complex, uncertain, or novel situations.
  - Rules and Logic (***Deterministic Rules***). These systems follow predefined rules and decision trees: "If X happens, do Y." This approach is simple and fast, making it ideal for well-understood, repeatable scenarios.
  - Machine Learning (***Predictive AI***). These systems learn from historical data to make predictions about future outcomes. They identify patterns and correlations that would be too complex for static rules.
  - Generative AI (***Creative + Adaptive AI***). Generative AI takes things further by enabling systems to *generate* new content, solutions, or interactions on the fly. It brings a new level of flexibility, personalisation, and usability to intelligent systems.

    Key capabilities of generative AI include:
    - **Dynamic Decision-Making**: Generates responses or solutions rather than selecting from a fixed list.
    - **Natural Language Understanding and Generation**: Enables more intuitive and conversational interfaces with users.
    - **Adaptive Personalisation**: Tailors experiences, content, or workflows for each user or context.

    - **Creative Problem Solving**: Synthesizes insights, explanations, or strategies in complex or ambiguous situations.
    - **Cognitive Load Reduction**: Perhaps one of its most powerful benefits, generative AI simplifies complex interfaces and tasks by understanding intent and generating the next best action or response, allowing users to focus on outcomes rather than process:
        - Instead of navigating multiple dashboards, users can ask in plain language, *"What do I need to do to prepare for the next customer visit?"* and receive a detailed, personalised summary, suggestions, and even action items—all generated in real-time.
        - This shift from user-driven exploration to system-assisted guidance significantly reduces mental effort, friction, and decision fatigue.
  - Agentic AI (**Autonomous, Goal-Driven Intelligence**) Agentic AI represents a distinct form of intelligence focused on goal-driven action, not just event response or content generation. While rules, predictive AI, and generative AI determine what should be done, agentic AI determines how to do it and carries it through. Agentic systems are designed to operate with varying levels of autonomy, from human-in-the-loop assistance to fully autonomous execution within defined guardrails. They are characterised by:
    - **Goal Orientation**
      Agentic AI works toward explicit objectives rather than responding to single events or prompts. A goal may span multiple steps, systems, and timeframes.
    - **Planning and Execution**
      The system decomposes goals into actions, selects appropriate rules, models, workflows, or tools, and executes them dynamically as conditions change.
    - **Continuous Feedback and Adaptation**
      Agentic AI observes the results of its actions, evaluates progress, and adjusts its behaviour in real time—forming a closed decision loop rather than a one-off response.
    - **Orchestration Across Systems**
      Agentic AI coordinates events, services, data, and intelligence across distributed environments, enabling end-to-end outcomes rather than isolated decisions.

    In this model, agentic AI acts as the execution layer of intelligence, orchestrating real-time responsiveness, situational awareness, rules, machine learning, and generative AI into purposeful, outcome-driven behaviour.

1.2 Event-Driven Architecture and Events

Vantiq applications are fundamentally built around an **event-driven architecture (EDA)**—a design pattern where the occurrence and handling of events determine the application's flow. This architecture enables systems to be **reactive, loosely coupled, scalable, and resilient**, making it ideal for real-time, intelligent, and situationally aware applications.

1.2.1 What Is an Event?

An **event** is a discrete signal that something has occurred or the state of something has changed in the system or its environment. Events are often expressed as messages or data structures representing a **change in state**, such as:

- A temperature reading from a sensor.
- An object detection event from a camera or CV system.
- An update to inventory levels.
- A new order.

In Vantiq, events can originate from various sources, including IoT devices, external message brokers (e.g., Kafka, MQTT), user interactions, APIs, or internal service logic. Events are **asynchronous**, enabling reactive workflows that respond to stimuli without blocking or waiting for the completion of other workflows.

1.2.2 What Is Event-Driven Architecture?

Event-Driven Architecture is a paradigm in which system components communicate and operate based on specific events. Instead of relying solely on synchronous request/response patterns (e.g., REST APIs), an EDA enables services to **publish and subscribe to events**, forming a decoupled network of producers and consumers.

Key characteristics of EDA include:

- **Loose Coupling**: Event producers and consumers operate independently of each other. Changes to one component rarely require changes to others, improving maintainability and scalability.
- **Asynchronous Communication**: Events can be processed as they occur, enabling real-time responsiveness and efficient resource usage.
- **Resilience and Fault Tolerance**: By decoupling services, the system can isolate failures and retry or reroute events as needed.
- **Scalability**: EDA systems naturally support horizontal scaling, as event consumption can be distributed across multiple instances or services.

1.2.3 Systems and Applications and Relationship to Domain vs Application Events

An **application** is a cohesive piece of software built to deliver a specific capability or solve a well-defined problem. It has a clear purpose, encapsulates its own behaviour and rules, and exposes functionality and state through **domain events** and API's. An application can usually be understood, developed, and evolved in relative isolation because it owns its logic and responsibilities. In domain-driven terms, an application often aligns with a bounded context and speaks a clear, intentional language about the problem it exists to solve.

Characteristics of an **application**

- **Clear business purpose**: Focuses on a single, well-defined capability or problem space
- **Encapsulated logic and rules**: Owns its workflows, invariants, and decision-making internally
- **Explicit Interfaces**: Exposes functionality via well-defined domain events and API's
- **Independent lifecycle**: Can often be developed and deployed independently, and developed by a single team

A **system**, by contrast, is the broader environment in which applications operate together. It is composed of multiple applications. A system is concerned less with individual responsibilities and more with how parts collaborate to deliver end-to-end outcomes. Its behaviour emerges from the interaction of its components rather than being fully defined in a single codebase.

Characteristics of a System

- **End-to-end scope:** Delivers complete business outcomes by combining multiple applications.
- **Multi-application composition:** Consists of several independently owned applications working together.
- **Cross-team ownership:** Evolves through collaboration among multiple teams and domains.

Within an application, it's useful to distinguish **domain events** from **application events**. A domain event is a business fact stated in the past tense—something meaningful happened in the domain (e.g., OrderPlaced, CustomerRegistered). These come from the domain model and represent outcomes the business cares about. An application event is more about coordinating work at the application layer—signals that trigger technical actions or workflows (e.g., SendWelcomeEmailRequested, RebuildSearchIndexRequested). Domain events describe "what happened"; application events often describe "what the system should do next."

To form a system from multiple applications is to have each application publish its domain events to a shared **catalog**. The **catalog** becomes the system's contract for event-driven integration: it lists the events an application emits, their schemas, meanings. Other applications subscribe to those published domain events to react and coordinate across boundaries—turning independent applications into a connected system without tight coupling.

Publishing Domain Events to Form a System

- **Event publication**: Each application publishes its domain events when business facts occur.
- **Shared event catalog**: Events are registered with schema, meaning, ownership, and versioning.
- **Loose coupling between applications**: Consumers react to events without direct knowledge of producers.
- **System-level behaviour through composition**: Coordinated outcomes emerge from independent reactions to shared events.
- **Scalable evolution**: New applications can join the system by subscribing to existing events without changing producers.

1.2.4 Event Catalog and Distributed Systems

Vantiq Event Catalog provides a centralized place to define and govern events, while still enabling a fully distributed system at runtime. The catalog is not the system's message broker; instead, it is the system's shared source of truth for event meaning. This separation allows teams to align on business facts without forcing a centralized execution model.

In this approach, domain events remain owned and published by individual applications. Each application registers domain events in the centralized catalog, describing their schemas and meaning/intent. This ensures that every event in the system has a clear definition and a stable contract, even though the applications that produce and consume the events are independently deployed and operated.

At runtime, publishers and consumers are distributed and communicate using **brokerless** event distribution mechanisms. Events flow directly between applications—peer to peer, without a centralized broker becoming a bottleneck or single point of failure. The centralized catalog defines what events exist, but it does not dictate how they are transported. This decouples governance from execution, enabling low latency, resilience, and local autonomy.

Applications evolve independently yet remain interoperable because they speak a common event language defined in the catalog. New applications can join the system by consulting the catalog and subscribing to existing domain events, without requiring changes to publishers or the introduction of new centralized components. The event catalog and domain events together form the backbone of a distributed system coordinated by semantics, not by tightly coupled technology.

1.3 Collaborative Aspects of Applications

In the evolving landscape of intelligent software systems, **collaborative applications** represent a shift from automation-centric design to **human-machine teamwork**. These **collaborative applications** could also involve **intelligent machine-machine collaboration** and automation. These systems are engineered to enable **real-time interaction, shared decision-making**, and **knowledge-driven workflows** between automated services and human users. Instead of automating away the human role, they amplify human capabilities by integrating contextual data, collaboration models, and AI-powered knowledge access.

1.3.1 Human-in-the-Loop

Collaborative applications assume from the outset that not all decisions can or should be made by machines alone.

They:

- Treat users as active participants in the system's logic.
- Include mechanisms for decision input, overrides, and feedback.
- Support partial automation, where the system handles repetitive or analytical tasks but humans guide critical steps, especially in high-risk, novel, or sensitive scenarios.

Examples:

- In maintenance workflows, a system detects a fault and proposes a plan, but a technician validates the action before proceeding.
- In fraud detection, flagged transactions are reviewed and approved by a compliance officer.

1.3.2 Situational Awareness

All applications require some level of situational awareness —the system's ability to understand the context in which tasks are being performed, whether the application involves automated responses or human involvement. Vantiq applications continuously ingest and process:

- Sensor and system data (e.g., machine telemetry, weather, GPS).
- Temporal signals (e.g., time since event occurred, shift schedules).
- Historical patterns (e.g., prior failure cases).

- Enhance data streams with static data (e.g., add location information based on a GPS property).

This real-time context enables applications to:

- Prioritise and personalise task assignments.
- Suggest context-specific recommendations.
- Modify workflows on the fly as new information emerges.

1.3.3 Integrating AI: Knowledge at Your Fingertips

Modern collaborative applications extend beyond workflow by embedding intelligent agents that tap into large-scale knowledge bases.

Traditionally, applications have presented users with details about an issue and assumed they had the required skills to solve it, or with a huge amount of information to help them solve it. With the advent of Generative AI, it has become much easier to move beyond this to providing some of the following capabilities;

- Using the existing knowledge base to provide the user with the required steps to solve an issue
- Allow the user to query in a natural language interface (chat, voice, etc.) the knowledge bases
- Provide the user instance access to up-to-date knowledge without the requirement to read a vast amount of new material as it is available

These capabilities can be implemented on top of some of the following capabilities.

1.3.4 AI Agents as Role Participants

AI agents can be assigned to collaborator roles, just like human users. For example:

- An AI support analyst that reviews incoming customer issues and suggests initial responses.
- An AI triage bot that assigns priority and routes tasks based on context and severity.
- An AI field advisor that offers diagnostic steps to technicians based on historical fixes.

These agents can:

- Receive and respond to notifications.
- Execute tasks triggered by events.
- Escalate or delegate as needed.
- Interact with users via chat, forms, or voice.

1.3.5 Built-in Escalation

Collaborative systems are built to handle complexity and exceptions:

- If a user is unavailable, the system escalates to a supervisor.
- If progress is delayed, new users are invited, or AI agents step in to assist.
- If the situation worsens, priority and visibility are increased.

Escalation logic is configurable and often triggered automatically based on timeouts, conditions, or user feedback.

1.4 Knowledge-based Rules, Predictive, Generative, and Agentic AI in Vantiq

Intelligent, real-time applications rarely rely on a single form of intelligence. Instead, they combine multiple complementary approaches, each suited to different kinds of decisions, uncertainty, and operational constraints. In Vantiq applications, intelligence spans a spectrum—from knowledge-based rules that encode what is already known, through predictive models that infer likely outcomes, to generative and agentic capabilities that reason, plan, and act independently.

At the foundation are **knowledge-based rules**, which encode explicit domain knowledge, policies, and constraints. These rules provide deterministic, predictable, and explainable behaviour. They are essential where correctness, safety, compliance, and clear cause-and-effect logic are required. Because they are explicitly defined by developers, they deliver speed, reliability, and auditability.

Predictive AI extends beyond predefined logic by leveraging machine learning models trained on historical data. Rather than encoding known relationships, predictive systems learn statistical patterns, correlations, and nonlinear dependencies directly from data. This enables detection of subtle anomalies, grey-area conditions, and emerging behaviours that static rules cannot easily express. Predictive models can be retrained over time, allowing systems to adapt as environments evolve.

Generative AI moves from detection to synthesis. Instead of selecting from predefined outputs, generative systems dynamically construct explanations, summaries, recommendations, and plans based on context, retrieved knowledge, and intent. This capability is particularly valuable in complex environments where human interaction, interpretation, and situational understanding are critical.

Agentic AI introduces goal-driven behaviour. Agentic systems do not merely produce outputs; they plan, make decisions, use tools, coordinate actions, and adapt dynamically to achieve defined objectives. Often implemented as multi-agent systems, specialized agents collaborate—such as planners, executors, or evaluators—to improve robustness and scalability.

Each intelligence type has trade-offs in determinism, explainability, performance, flexibility, and cost. Most real-world Vantiq applications combine them deliberately to create scalable, resilient, and situationally aware systems. The document positions these capabilities not as alternatives, but as composable elements within a unified real-time architecture.

1.4.1 Knowledge-Based Rules

Knowledge-based rules represent the most explicit and deterministic form of intelligence in an event-driven system. They encode **domain expertise, policies, regulatory constraints, and well-understood cause-and-effect relationships** into clear conditional logic. Given identical inputs, they always produce identical outputs, making them essential for safety, governance, and trust.

Rules function as immediate perception mechanisms in real time. When a defined condition is met—such as a threshold breach, policy violation, or required state transition—the system responds instantly and deterministically. This makes them ideal for enforcing compliance, triggering mandatory actions, detecting known failure conditions, maintaining hard safety boundaries and encoding domain knowledge and expertise.

Statistical functions such as moving averages or standard deviation calculations may be incorporated into rules, but these are used to detect known signal changes rather than infer unknown behaviour. The purpose remains deterministic enforcement of explicitly defined conditions.

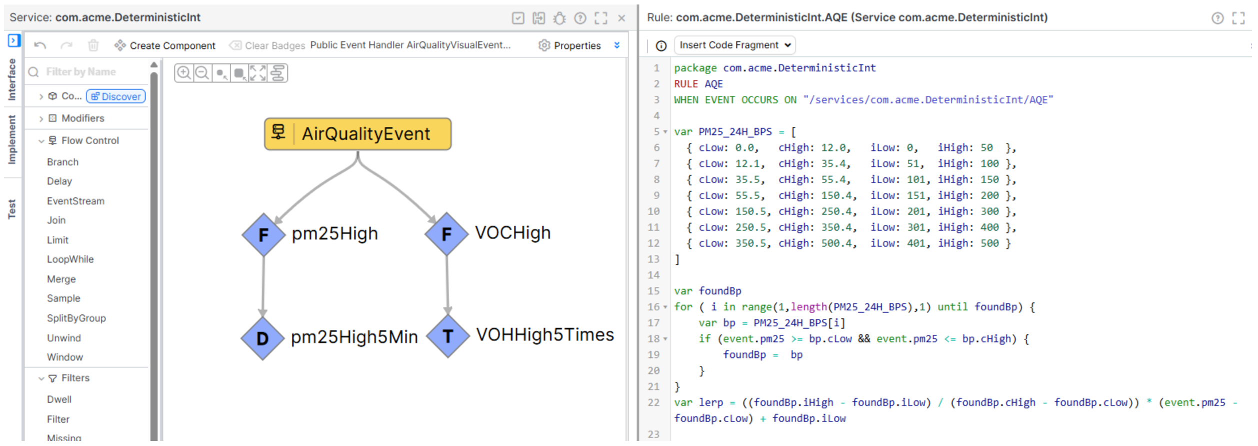
The strengths of rules include precision, explainability, low latency, auditability, and governance control. They are transparent and easy to reason about because the logic is explicitly defined and inspectable.

However, rules are inherently limited by prior knowledge. They cannot discover unknown patterns, adapt automatically to new behaviours, or efficiently model complex multi-signal interactions. As systems grow, rule sets can expand dramatically, leading to nested logic, interdependencies, and maintenance overhead. Attempting to encode all possible combinations of interacting signals can result in rule explosion and structural complexity.

To address scaling challenges, complex knowledge can be structured using decision models, constraint solvers, knowledge graphs, or state machines, reducing brittleness while preserving determinism.

1.4.1.1 Knowledge-Based Rules and Vantiq

Knowledge-Based Rules can be implemented in Vantiq either visually or programmatically. Vantiq's Visual Event Handler provides a rich suite of predefined building blocks to apply deterministic rules to your event flows. These can either be supplemented with programmatic logic or implemented as programmatic rules using the Vantiq VAIL language.



```
package com.acme.DeterministicInt
RULE AQE
WHEN EVENT OCCURS ON "/services/com.acme.DeterministicInt/AQE"

var PM25_24H_BPS = [
  { cLow: 0.0,   cHigh: 12.0,   iLow: 0,   iHigh: 50  },
  { cLow: 12.1,  cHigh: 35.4,   iLow: 51,  iHigh: 100 },
  { cLow: 35.5,  cHigh: 55.4,   iLow: 101, iHigh: 150 },
  { cLow: 55.5,  cHigh: 150.4,  iLow: 151, iHigh: 200 },
  { cLow: 150.5, cHigh: 250.4,  iLow: 201, iHigh: 300 },
  { cLow: 250.5, cHigh: 350.4,  iLow: 301, iHigh: 400 },
  { cLow: 350.5, cHigh: 500.4,  iLow: 401, iHigh: 500 }
]

var foundBp
for ( i in range(1,length(PM25_24H_BPS),1) until foundBp) {
    var bp = PM25_24H_BPS[i]
    if (event.pm25 >= bp.cLow && event.pm25 <= bp.cHigh) {
        foundBp =  bp
    }
}
var lerp = ((foundBp.iHigh - foundBp.iLow) / (foundBp.cHigh - foundBp.cLow)) * (event.pm25 -
foundBp.cLow) + foundBp.iLow

```

1.4.2 Predictive AI

Predictive AI introduces machine learning models **trained** on historical data to identify **hidden patterns, correlations, and statistical relationships**. Unlike rules, which react to explicitly defined conditions, predictive models learn what normal behaviour looks like and detect subtle deviations or emerging risks.

These models are particularly valuable in real-time systems where early warning and foresight improve safety, responsiveness, and efficiency. Predictive AI can detect grey-area anomalies—situations where no single threshold is violated but patterns across multiple signals suggest elevated risk. It can also forecast future states, estimate probabilities, classify complex situations, and identify weak signals of failure.

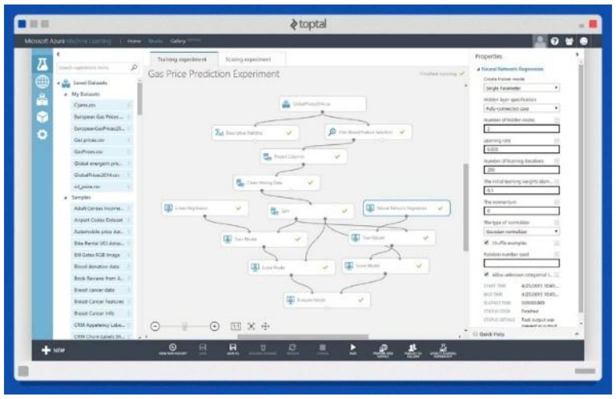
A key advantage is the ability to detect what rules cannot. Machine learning models recognize subtle behavioural drift, rare edge cases, and multidimensional anomalies that would be impractical to encode manually. Outputs are typically emitted as enriched events containing confidence scores or severity estimates, enabling prioritization and human-in-the-loop decision-making.

Computer vision (CV) is a powerful application of machine learning within real-time systems because it transforms raw visual input into structured, semantic information such as object detection, classification, tracking, and activity recognition. Downstream processing of the results of CV event data can be used to detect situations of interest, detect temporal patterns, and infer behaviours.

Predictive AI provides adaptability through retraining as environments change. It supports probabilistic reasoning and improved anomaly detection in dynamic environments. However, trade-offs include dependence on training data quality, model drift, lifecycle management overhead, computational cost, and reduced explainability compared to deterministic rules.

1.4.2.1 Predictive AI and Vantiq

Predictive AI is implemented externally to Vantiq and can be integrated via either built-in sources, such as Azure ML Studio or Apache Camel's Deep Java Library component, or, more commonly, through REST APIs to a variety of ML platforms. These connections are then very easy to integrate into the event flow via synchronous procedure calls.


toptal
Microsoft Azure Machine Learning
Gas Price Prediction Experiment
Properties

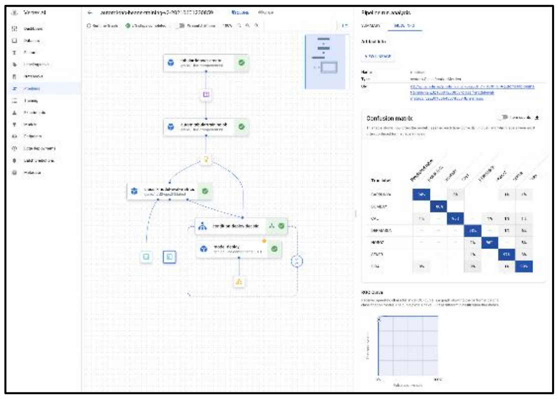


Figure 1 - ML Studio & Google Vertex AI

1.4.3 Generative AI

Generative AI differs fundamentally from rules and predictive models by **producing new content or responses** rather than selecting predefined responses. It produces text, summaries, explanations, structured outputs, recommendations, and plans based on context, intent, retrieved knowledge, and learned representations.

This capability is particularly valuable in environments characterized by ambiguity, unstructured data, and heavy human interaction. Generative AI synthesizes multiple inputs—current events, historical context, domain knowledge, and user intent—into coherent responses tailored to the situation. Two similar inputs may produce different outputs due to contextual nuance and probabilistic generation.

A defining feature is natural language **understanding and generation**. Generative AI allows conversational interaction, interprets unstructured queries, extracts intent, and translates technical system outputs into human-readable explanations. This reduces barriers to accessing complex systems.

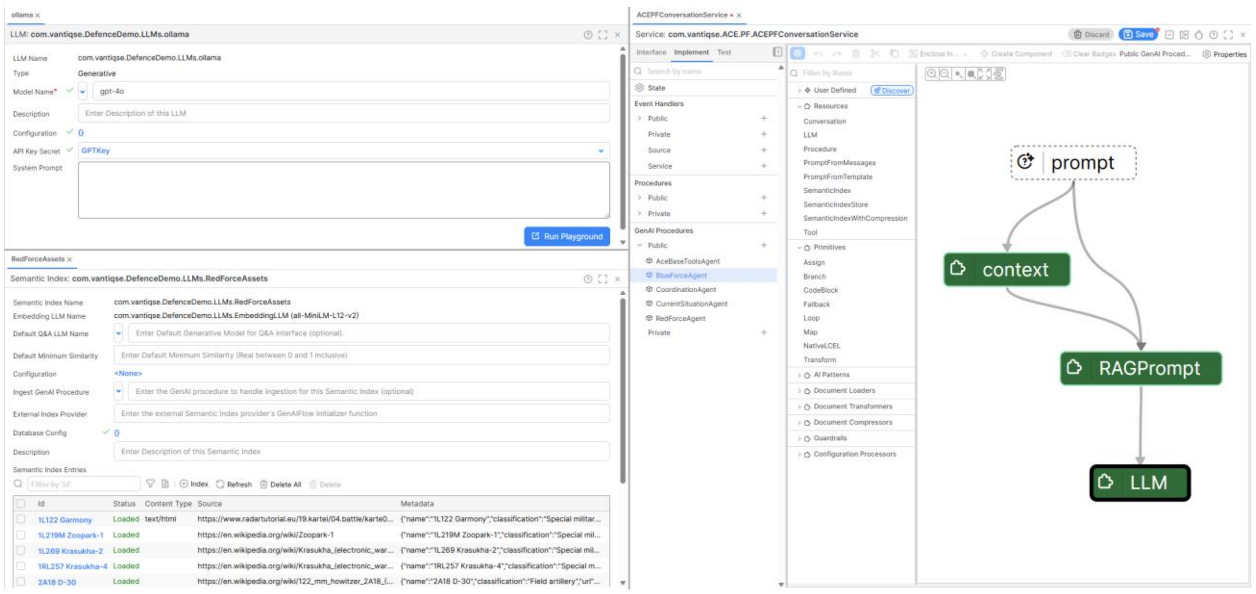
Modern systems are increasingly multimodal, interpreting and generating content across text, images, video, and audio. This enables contextual integration of visual and textual information.

Retrieval-Augmented Generation (RAG) strengthens reliability by grounding responses in authoritative, domain-specific sources retrieved at runtime. By indexing documents into embeddings stored in vector databases, systems can retrieve relevant knowledge and supply it to the model during generation. RAG improves accuracy, reduces hallucinations, provides access to domain knowledge, maintains up-to-date knowledge without retraining, and enhances transparency.

Generative AI reduces cognitive load by synthesizing large volumes of data into concise explanations, connecting signals across systems, and clarifying why events matter. However, it introduces trade-offs including probabilistic variability, hallucination risk, sensitivity to prompt design, limited intrinsic understanding, reduced internal explainability, computational cost, and governance concerns.

1.4.3.1 Generative AI and Vantiq

Vantiq provides a rich set of Generative AI resources and capabilities, including integration with a large set of Generative and Embedding models, support for Semantic Index and RAG through the integration with a vector database, unstructured content ingestion services, and built-in patterns such as submitPrompt and answerQuestion, allowing easy integration into the visual event flow. Vantiq also provides a rich low-code tool for constructing more advanced generative AI logic using the GenAI Builder, which can be integrated into the visual event flow or accessed programmatically. In addition to backend support for Generative AI, the integrated Client Builder provides built-in widgets that enable developers to include chat interfaces in their web and mobile applications.



LLM: com.vantiqse.DefenceDemo.LLMs.ollama

LLM Name	com.vantiqse.DefenceDemo.LLMs.ollama
Type	Generative
Model Name*	gpt-4o
Description	Enter Description of this LLM
Configuration	{}
API Key Secret	GPTKey
System Prompt	

Semantic Index: com.vantiqse.DefenceDemo.LLMs.RedForceAssets

Semantic Index Name	com.vantiqse.DefenceDemo.LLMs.RedForceAssets
Embedding LLM Name	com.vantiqse.DefenceDemo.LLMs.EmbeddingLLM (all-MiniLM-L12-v2)
Default Q&A LLM Name	Enter Default Generative Model for Q&A interface (optional).
Default Minimum Similarity	Enter Default Minimum Similarity (Real between 0 and 1 inclusive)
Configuration	<None>
Ingest GenAI Procedure	Enter the GenAI procedure to handle ingestion for this Semantic Index (optional)
External Index Provider	Enter the external Semantic Index provider's GenAIFlow initializer function
Database Config	{}
Description	Enter Description of this Semantic Index

Semantic Index Entries

Id	Status	Content Type	Source	Metadata
1L122 Garmony	Loaded	text/html	https://www.radartutorial.eu/19.kartei/04.battle/karte0...	{"name":"1L122 Garmony","classification":"Special militar...
1L219M Zoopark-1	Loaded		https://en.wikipedia.org/wiki/Zoopark-1	{"name":"1L219M Zoopark-1","classification":"Special mil...
1L269 Krasukha-2	Loaded		https://en.wikipedia.org/wiki/Krasukha_(electronic_war...	{"name":"1L269 Krasukha-2","classification":"Special mil...
1RL257 Krasukha-4	Loaded		https://en.wikipedia.org/wiki/Krasukha_(electronic_war...	{"name":"1RL257 Krasukha-4","classification":"Special m...
2A18 D-30	Loaded		https://en.wikipedia.org/wiki/122_mm_howitzer_2A18_(...	{"name":"2A18 D-30","classification":"Field artillery","url"...

Service: com.vantiqse.ACE.PF.ACEPFConversationService

Figure 2 – Generative AI & Vantiq

1.4.4 Agentic AI

Agentic AI builds on reactive or assistive intelligence to **be goal-driven and outcome-oriented**. Rather than treating AI as something that produces answers, predictions, or generated content on demand, agentic AI focuses on carrying intent through to outcomes, especially in scenarios where the right response cannot be specified in advance and must be generated through reasoning and action.

An agentic system does not simply determine what should be done. It interprets a **goal**, constructs a **plan** to achieve it, and dynamically selects and sequences actions based on evolving conditions. Rather than following a fixed workflow or predefined script, the system uses **generative AI** capabilities to reason about possible strategies, synthesize options, evaluate trade-offs, and decide among alternative courses of action. Planning is not a one-time activity: as actions are executed and new information becomes available, the system reassesses its assumptions, updates its understanding of the environment, and revises its plan accordingly.

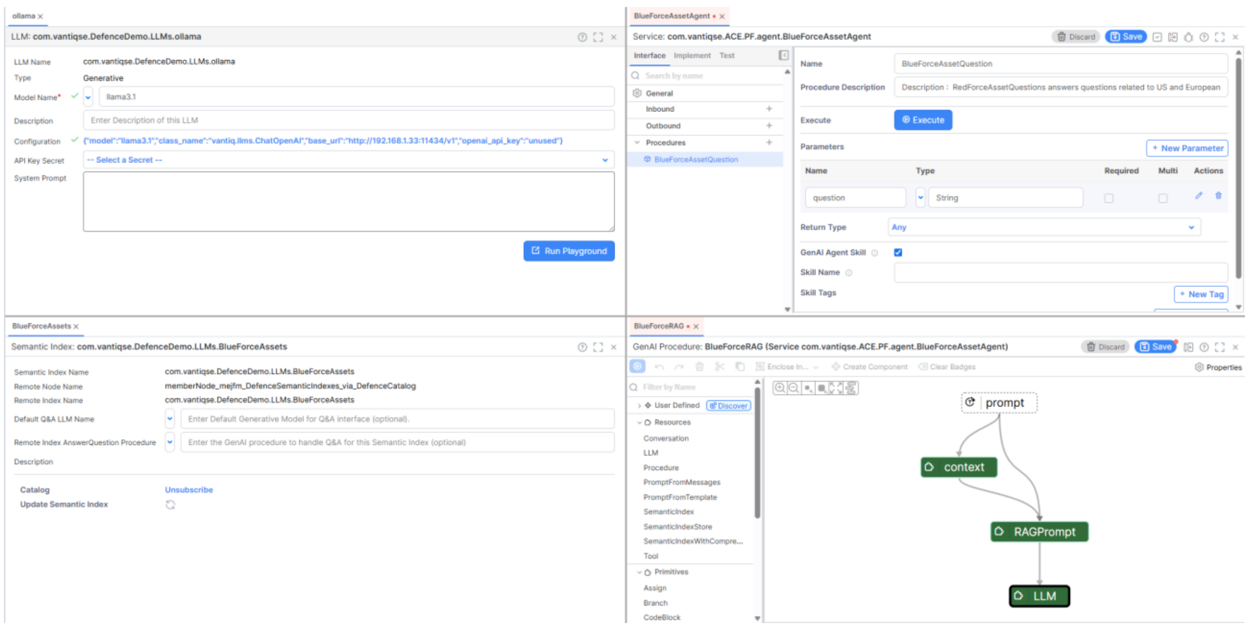
Planning is dynamic and iterative. As actions produce results, the agent reassesses assumptions and modifies its strategy. Generative AI typically supports reasoning and plan construction within these systems.

Agentic implementations may be single-agent—where one agent owns an end-to-end objective—or multi-agent, where specialized agents collaborate around a shared goal. Multi-agent systems improve scalability and robustness by distributing responsibilities across planners, executors, evaluators, or domain specialists.

Strengths of agentic AI include sustained goal ownership, adaptive planning, decomposition of complex tasks, and autonomous decision-making within defined constraints. However, trade-offs include increased architectural complexity, risk of excessive autonomy without guardrails, uncertainty in generated plans, debugging challenges, resource overhead, and governance requirements.

1.4.4.1 Agentic AI and Vantiq

Vantiq Agentic AI implementation builds on the Generative AI capabilities and enables developers to define an AI Agent as a service-based resource with a set of associated Skills (representing the Agent's capabilities). The system also provides a set of routing and planning algorithms, including ReWoo (Reasoning Without Observation), Plan-Execute, and Reflection, to enable developers to build and orchestrate a group of agents into an advanced multi-agent application. These capabilities can be integrated into the event flows and situation awareness, as well as support human-in-the-loop and advanced web and mobile clients using the Client Builder.


LLM: com.vantiqse.DefenceDemo.LLMs.ollama

LLM Name	com.vantiqse.DefenceDemo.LLMs.ollama
Type	Generative
Model Name*	llama3.1
Description	Enter Description of this LLM
Configuration	{"model":"llama3.1","class_name":"vantiq.llms.ChatOpenAI","base_url":"http://192.168.1.33:11434/v1","openai_api_key":"unused"}
API Key Secret	-- Select a Secret --
System Prompt	

Service: com.vantiqse.ACE.PF.agent.BlueForceAssetAgent

Name	BlueForceAssetQuestion
Procedure Description	Description : RedForceAssetQuestions answers questions related to US and European

Name	Type	Required	Multi	Actions
question	String	☐	☐	

Return Type	Any
GenAI Agent Skill	✓
Skill Name	
Skill Tags	

Semantic Index: com.vantiqse.DefenceDemo.LLMs.BlueForceAssets

Semantic Index Name	com.vantiqse.DefenceDemo.LLMs.BlueForceAssets
Remote Node Name	memberNode_mejfm_DefenceSemanticIndexes_via_DefenceCatalog
Remote Index Name	com.vantiqse.DefenceDemo.LLMs.BlueForceAssets
Default Q&A LLM Name	Enter Default Generative Model for Q&A interface (optional).
Remote Index AnswerQuestion Procedure	Enter the GenAI procedure to handle Q&A for this Semantic Index (optional)
Description	
Catalog	Unsubscribe
Update Semantic Index	

GenAI Procedure: BlueForceRAG (Service com.vantiqse.ACE.PF.agent.BlueForceAssetAgent)

Figure 3 - Agentic AI & Vantiq

1.4.5 Composing and Orchestrating Intelligence

Intelligent systems are most effective when they deliberately combine multiple forms of intelligence rather than relying on a single approach. Different types of decisions demand different properties—certainty, adaptability, contextual understanding, or sustained execution. In Vantiq, intelligence is treated as a coordinated architecture of complementary capabilities, each applied where it delivers the greatest value.

- **Knowledge-based rules**
  Knowledge-based rules provide certainty and control. They enforce explicit policies, safety limits, regulatory requirements, and operational invariants. Their role is to guarantee what must always hold true. They deliver deterministic, auditable, and low-latency responses where correctness and compliance are essential.
- **Predictive AI**
  Predictive AI expands perception beyond what has been explicitly encoded. By learning statistical relationships from historical data, predictive models detect complex patterns, emerging behaviours, weak signals, and probabilistic risk. Rather than asserting certainty, they estimate likelihood—enabling early detection of anomalies or trends that would be impractical to define manually.

- **Generative AI**
  Generative AI introduces interpretation and contextual understanding. It synthesizes structured and unstructured information, historical context, and domain knowledge into explanations, summaries, and recommendations. Its purpose is not simply to detect or estimate, but to clarify meaning—transforming signals into coherent narratives that reduce cognitive load and support informed decision-making.
- **Agentic AI**
  Agentic AI adds persistence and outcome ownership. Instead of responding to isolated inputs, agentic systems adopt explicit goals and remain responsible for achieving them. They plan, coordinate actions across services and people, monitor intermediate results, and adapt strategies as conditions evolve. This capability becomes critical when execution paths are conditional, dynamic, or multi-step.

A common composition pattern emerges across intelligent Vantiq applications:

- **Perception**: Rules detect known conditions immediately; predictive models identify complex anomalies or future risk.
- **Understanding**: Generative AI synthesizes events, context, and knowledge into explanations and candidate actions.
- **Execution**: Agentic AI orchestrates services, coordinates stakeholders, and adapts behaviour to achieve defined goals.

Examples such as safety and security video monitoring or hospital discharge workflows demonstrate this layered approach. Perception provides structured awareness, rules enforce constraints, predictive models identify risk patterns, generative reasoning explains the situation, and agentic coordination ensures that objectives are carried through to completion.

Scalable, resilient, and situationally aware systems do not emerge from any single form of intelligence. They result from intentional architectural composition—balancing deterministic enforcement, statistical insight, contextual reasoning, and adaptive execution to meet real-world business outcomes.

2. Development Methodology

Vantiq applications are constructed from two foundational building blocks: Services and Clients. These components interact through synchronous (procedures) and asynchronous (events) interfaces to create loosely coupled, highly responsive systems. The methodology used to develop these applications focuses on event-driven, stateful, and resilient design patterns optimized for real-time, situationally aware intelligent systems. Vantiq is a full-stack development environment. Vantiq development teams can be made up of full-stack developers who have experience in both traditional backend development techniques and client-side development, or more specialized developers who work on the front end and the backend of an application.

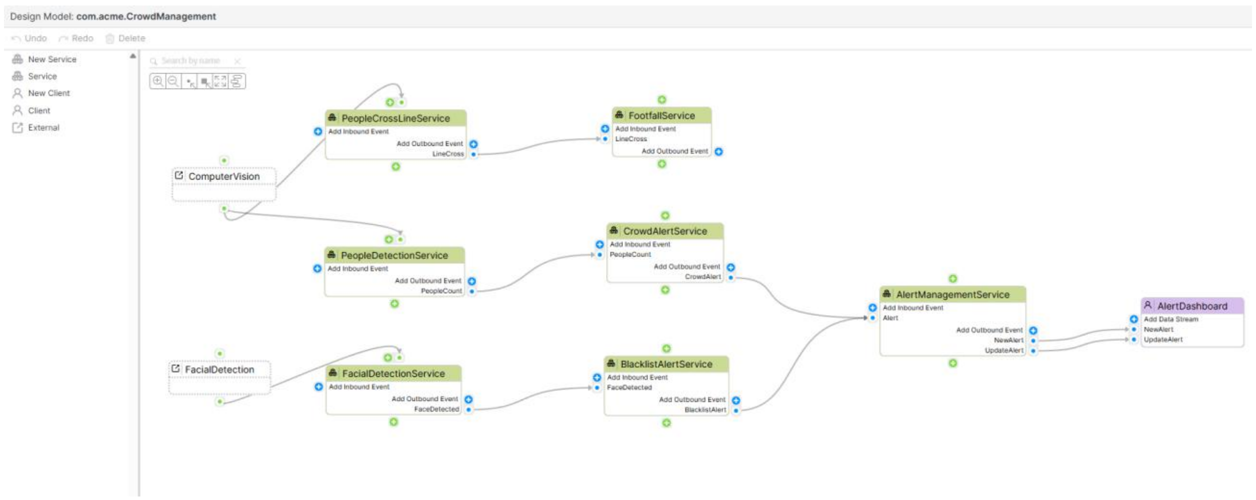


Figure 4 - Model of a Service-Based Application

2.1 Service-Based Architecture

2.1.1 Traditional Service-Based Approaches

Historically, service-based architectures such as SOA, Microservices, and Serverless/lambda-style services have centered on request-response interactions, exposing defined APIs and encapsulating the logic behind those interfaces. These services are often:

- Stateless (using external databases to store state).
- Synchronous (tight request/response coupling).
- API-centric (focused on describing and invoking service contracts).

Challenges in these approaches include:

- Tight Coupling: Changing an API can have a ripple effect across consumers.
- Latency from State Management: Using databases for transient state introduces delays.
- Complex Middleware: Event-driven behavior requires additional infrastructure, such as brokers or lambda dispatchers.

2.1.2 Vantiq Service Architecture

Vantiq extends and modernizes the service-based model by introducing the following core innovations:

Asynchronous Interfaces (Event-Driven)

Services can consume and publish events via visual or code-based handlers:

- Visual Event Handlers: Low-code modelling for business logic.
- Rules: JavaScript-like logic for flexible event processing.

This model supports both tight coupling and loose coupling.

In-Memory Stateful Services

To reduce latency, Vantiq supports resilient in-memory state:

- Caching transient and update/access-intensive state in-memory improves performance, responsiveness, and scalability in real-time applications. This type of state typically includes data that changes frequently (e.g., sensor readings, session data, or intermediate computation results) or is accessed very often (e.g., configuration settings, user context, or lookup tables). Storing such data in-memory avoids the overhead of constantly reading from or writing to a database, significantly reducing latency and freeing up database resources for other operations.
- State can also be partitioned – a partition key in conjunction with a hashing algorithm to allocate a subset of the state to a single cluster member - to improve performance. If this is within a cluster, then the various partitions will be managed by different cluster members, and therefore the load will be spread across the cluster as a whole.
- Clustering enables replicated state across nodes, improving fault tolerance. Two copies of the state will be maintained on separate cluster members. The loss of a single cluster member can be tolerated without any degradation of the state. So, in essence, replicated in-memory state across cluster nodes behaves much like RAID 1 for volatile (non-persistent) data, ensuring continuity and resilience in real-time applications
- Developers control state replication through simple configuration, achieving high availability with minimal effort.

Loose Coupling and Event Routing

Using event routes defined in the Design Model (see below for more details), developers can rewire how services interact without modifying service code:

- Promotes service reuse.
- Enables external service integration (from catalog or assemblies, see below for more details).
- Enhances maintainability and flexibility in dynamic environments.

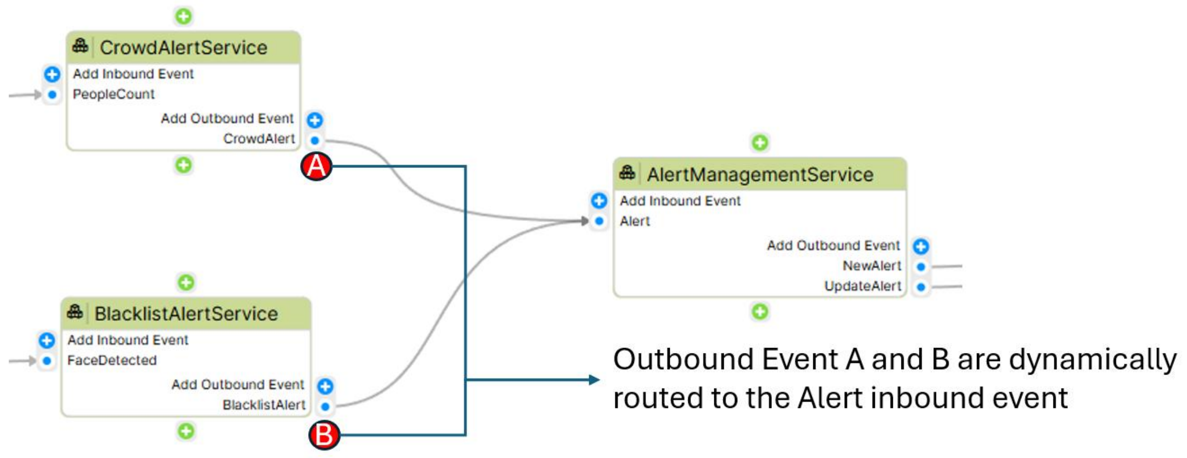


Figure 5 - Event Routing Example

2.2 Clients

Vantiq applications interact with users and external systems through Clients, which serve as the front-end interfaces or integration points for user interaction. Clients can be created natively within Vantiq or built using external platforms and SDKs. They consume services by invoking procedures and publishing and subscribing to events, enabling real-time, interactive applications.

2.2.1 Vantiq-Built Clients

Browser Clients (Web Applications)

Developed Using: Vantiq Client Builder.

- Developed Using: Vantiq Client Builder (a low-code, HTML-based tool built on AngularJS).
- Execution: Accessed through standard web browsers.
- Purpose: Provide rich, responsive interfaces for desktop or browser-based use cases.
- Features:
  - Drag-and-drop UI creation.
  - Integration with services via procedures and events.
  - Support for dashboards, forms, maps, and charts.

Mobile Clients

Developed Using: Vantiq Client Builder (same tool as browser clients).

- Execution: Runs inside the Vantiq Mobile App, available on the Apple App Store and Google Play Store.
- Purpose: Native-like mobile applications with real-time interaction.
- Features:
  - Access to many device capabilities (camera, GPS, etc.).
  - Offline mode support.
  - Seamless deployment and updates via the Vantiq Mobile container app.
  - Consistent experience with desktop counterparts optimized for mobile screens.

Note: The same Client Builder can be used to design responsive interfaces that adapt to both browser and mobile environments.

2.2.2 Externally-Built Clients

Vantiq offers SDKs and RESTful APIs to support client development in external environments, providing developers with the flexibility to build fully customized user interfaces or integrate with existing applications.

React Native Clients

- Use Case: Cross-platform mobile apps with full control over the user experience.
- Integration: Via the Vantiq React Native SDK.
- Best For: Developers needing custom mobile apps that use native UI components or advanced functionality beyond what the Vantiq Mobile App offers.

iOS or Android Clients (Native)

- Use Case: Applications explicitly built for Apple or Android devices.
- Integration: Via the Vantiq iOS/Android SDK.
- Best For: Enterprises standardizing on Apple or Android devices or requiring platform specific integrations.

3. Service Types

In Vantiq applications, services are the core building blocks used to encapsulate logic, integrate with external systems, manage data, and interact with AI models. While the Vantiq IDE does not explicitly enforce service classifications, adopting a consistent categorization of service types is extremely valuable when developing event-driven, real-time applications on the Vantiq platform. Vantiq applications often involve a wide mix of reactive logic, integrations, data access, and AI workflows—each with distinct responsibilities. Defining clear service types helps organize these capabilities in a way that aligns with Vantiq's event-driven, loosely coupled architecture.

Categorizing services provides several key benefits:

- **Clarity and Consistency:** A precise classification helps developers understand the purpose and scope of a service at a glance, making the application easier to reason about.
- **Separation of Concerns:** By distinguishing service responsibilities—such as integration, business logic, utility operations, data handling, and AI interactions—teams can design and evolve services independently with less risk of cross-impact.
- **Reusability and Modularity:** Defined service types encourage reuse of common patterns (e.g., integration connectors or utility functions), reducing duplication and accelerating development.
- **Scalability and Collaboration:** A well-structured service model enables larger teams to collaborate more effectively, aligning efforts around clear responsibilities and shared architectural guidelines.

The following sections outline a practical and flexible categorization of service types used within Vantiq applications. This model encompasses Integration Services (event- and API-based), Event-Driven Services, Utility Services, Data Services, and AI Agent Services—each addressing a distinct set of concerns commonly encountered in modern, event-driven application architectures.

3.1 Integration Services

Integration with external sources of events, external sources of data, or business functionality is important in almost every application. In Vantiq, these capabilities are encapsulated by services.

3.1.1 Event-Based Integration Services

Event-based integration Services enable seamless interaction between a Vantiq application and external event-driven systems. These services encapsulate the ingestion and publishing of events from external event brokers, asynchronous interfaces, or other systems that generate real-time data streams. By abstracting the event source details and standardizing event data schemas, they allow applications to remain loosely coupled while ensuring reliable and scalable event flow across systems.

Event-based integration Services typically:

- Encapsulate a single external source of events, such as a message broker, device hub, or event gateway.
- Standardize the handling of incoming or outgoing events, including transformation, enrichment, or routing.

- Support event protocols and technologies such as MQTT, AMQP, Kafka, and Google Pub/Sub directly or through an intermediary such as the Apache Camel Enterprise Connector.
- Enable secure and efficient integration with cloud-native event streams, IoT platforms, or enterprise messaging systems.
- Serve as reusable components across multiple workflows or domains requiring the same event source.

These services are generally implemented using one of the following:

- **Vantiq Event Source**: A direct connection to a supported event broker (e.g., MQTT, AMQP, Kafka, Google Pub/Sub).
- **Camel Source**: A Camel-based integration that provides connectivity to a wide range of event platforms, such as AWS Kinesis, Azure Event Hub, and more.

Examples:

- Subscribing to a Kafka topic to receive telemetry from industrial sensors.
- Ingesting events from an MQTT broker connected to IoT devices in a smart building.
- Publishing enriched events to Google Pub/Sub for downstream analytics or alerting.
- Wrapping a Camel Source to integrate with Azure Event Hub for streaming event ingestion from enterprise applications.

By encapsulating the complexity of real-time event integration, Event-Based Integration Services ensure that Vantiq applications can respond to external stimuli quickly and consistently, forming the backbone of any event-driven architecture.

3.1.2 API-Based Integration Services

API-based integration Services enable Vantiq applications to communicate with external systems through synchronous API calls. Often, these external systems provide access to information, help in external systems/databases, or expose external functions. Unlike event-driven integrations, these services are typically initiated by application logic or user actions rather than incoming events. They are typically synchronous and request/response in nature. They encapsulate the details of interacting with external APIs, including request construction, response handling, authentication, and data transformation.

API-based integration Services typically:

- Provide outbound access to external systems and services through standardized APIs, most commonly REST-based, but also supporting other protocols via integration frameworks like Apache Camel.
- Handle synchronous request/response patterns, often used for retrieving data on demand or triggering actions in third-party systems.
- Perform data formatting, validation, and transformation to ensure compatibility between Vantiq and the external service.
- Support interaction with a wide range of systems, such as ERP platforms, cloud services, SaaS applications, and internal enterprise APIs.
- Abstract external dependencies so that application workflows can call these services without managing the low-level integration logic.

These services are commonly implemented in the following ways:

- **Direct REST Integrations**: Calling external APIs over HTTP(S), often secured via OAuth, API keys, or other authentication methods.
- **Camel-Based API Connectors**: Leveraging Apache Camel components to integrate with non-RESTful APIs or legacy protocols such as SOAP, FTP, or JDBC.

Examples:

- Querying a CRM system to retrieve customer data during a user session.
- Sending an update to a warehouse management system when inventory levels change due to an internal transaction.
- Requesting real-time weather or traffic information from a public API to enrich a business process.
- Performing a synchronous validation of a payment method through an external financial service before approving a transaction.

By encapsulating API interactions, API-based integration Services ensure that applications remain cleanly decoupled from external systems, improving maintainability and enabling consistent integration practices across the platform.

3.2 Event-Driven Business Services

Event-driven business services encapsulate the core domain logic and operational rules that define how an event-driven application responds to and acts upon business-relevant events. These services are responsible for interpreting events, enforcing business rules, coordinating workflows, and initiating appropriate actions based on the event context.

Event-driven Services typically:

- Consume and analyse events from one or more sources to drive business decisions and actions.
- Orchestrate interactions across Integration Services, Data Services, and AI Agent Services in response to business events.
- Implement business logic such as conditional branching, time-based rules, thresholds, or complex event patterns.
- Act as the bridge between raw event data and meaningful business outcomes.

Event-Driven Examples:

- In a logistics application, a Business Service might monitor incoming location events from delivery trucks and trigger a customer notification or route adjustment when a delivery is delayed.
- In a manufacturing setting, machine sensor events indicating overheating may trigger a Business Service that evaluates severity, checks maintenance history, and initiates a shutdown or alert protocol.
- In a financial application, a Business Service may detect a series of high-value transactions within a short period and escalate the pattern as a potential fraud case.

3.3 Utility Services

Utility Services provide reusable application-agnostic functionality that supports the operation of other services within a Vantiq application. These services encapsulate common logic, helper functions, or background tasks that are not tied to a specific domain but are essential for enabling consistent and efficient behavior across the system.

Utility Services typically:

- Perform shared operations such as time/date calculations, data formatting, unit conversions, or ID generation.
- Encapsulate low-level functionality that multiple services or workflows rely upon, reducing code duplication and simplifying maintenance.
- Operate independently of specific business workflows, but may be invoked by Integration, Event-Driven, Data, or AI Agent Services as needed.
- Can be implemented using procedures for synchronous utility calls or event handlers for reactive tasks.

Examples:

- A time utility service that computes time zone-adjusted timestamps or schedules periodic events based on configurable intervals.
- A data formatting service that standardizes numeric precision, date display formats, or string casing across multiple UIs and reports.
- A unit conversion service that translates measurements (e.g., Celsius to Fahrenheit, kilometers to miles) for location-specific use cases.
- A unique ID generation service that issues globally unique or domain-scoped identifiers for use across distributed services.

By providing standardized and reusable functionality, Utility Services help ensure consistency, reduce redundancy, and promote cleaner separation of concerns across the application architecture.

3.4 Data Services

Data Services are responsible for managing the storage, retrieval, and manipulation of data that an application depends on. These services provide a clean abstraction over internal or external data stores, ensuring that applications can access and manage data efficiently, securely, and consistently.

Data Services typically:

- Support full **CRUD operations** (Create, Read, Update, Delete) for structured or semi-structured data.
- Interface with **internal Vantiq storage**, **external databases**, or **cloud-based data platforms**, either directly or through integration layers such as Apache Camel.
- Handle **data transformation, normalization, and validation** to ensure consistency and usability across different parts of the application.
- Manage **reference data** (e.g., configuration settings, metadata, lookup tables) and provide caching mechanisms for frequently accessed but infrequently changing data.
- Enable **query and filtering** capabilities for dashboards, analytics, and reporting needs.

Examples:

- Persisting customer profiles, order history, or transaction logs.
- Retrieving inventory levels or configuration data needed by other services.
- Providing read-optimized data for dashboards or reports.
- Synchronizing with external systems of record, such as ERP or CRM databases.

By encapsulating data access logic, Data Services ensure that applications remain modular, maintainable, and adaptable to changes in underlying data technologies or business requirements.

3.5 AI Agent Services

AI Agent Services provide a modular interface between your application and advanced AI capabilities, particularly Generative AI and Machine Learning (ML) models. These services abstract model access, allowing AI-powered functionality to be cleanly integrated into broader application logic without being tightly coupled to specific model implementations or providers.

AI Agent Services focus on two primary types of capabilities:

- **Agentic/Generative AI Services**: These services interact with large language models (LLMs) or multimodal models to generate natural language responses, summarize documents, classify content, extract structured data from unstructured inputs, or support conversational interfaces.
- **Machine Learning Services**: These services access traditional ML models for tasks such as prediction, classification, scoring, anomaly detection, or trend forecasting. Models may be pre-trained and hosted externally (e.g., in cloud ML platforms) or deployed internally.

AI Agent Services typically:

- Provide a **standardized interface** for invoking AI models, handling input formatting, output interpretation, and error handling.
- Allow agents to operate **asynchronously** and in an event-driven approach
- Allow **reuse of AI logic** across multiple business workflows or services without replicating implementation details.
- Manage **conversation memory,** either transient conversations or persistent conversations via integration with **collaborations.**
- **Expose Tools** that can be invoked by LLMs, allowing LLMs to access the other Vantiq and integrate with the other services.

Examples

- A new customer support ticket creates an event that triggers an AI Agent Service to summarize the issue and suggest potential resolutions.
- A transaction anomaly event triggers an ML-based fraud detection service to assess risk and recommend an action.
- A user uploads a document, triggering a generative AI service to extract structured fields and populate a data form for review.
- Periodic events can trigger batch scoring or forecasting models to produce updated insights.

4. Application Development Workflow

Vantiq supports both bottom-up and top-down approaches to development. Developers can either start by creating their services and clients (bottom-up) or by defining requirements and designing their application, and having these tools drive the creation of the services and clients (top-down). The following will show both models. As with any development methodology, this is not a fixed approach; developers could start top-down and then switch to bottom-up.

Often, the bottom-up approach is used when requirements are not known up front and when prototyping. If the requirements for an application are well-known then the top-down approach is generally preferred workflow.

4.1 Top-Down Approach

In the Top-down approach to development, the developer will start by either defining some requirements and then mapping these requirements onto a design or by jumping straight into the application design before developing the implementation logic.

4.1.1 Define Requirements – System Modeler (optional start point)

The System Modeler is a visual drag-and-drop tool for modeling requirements within Vantiq using a simplified Event Storming technique using Events, Reactions, and Commands.

Event Storming is a collaborative modeling technique used to understand and design complex business domains. In a simplified form focusing only on events, reactions, and commands, it works like this:

- ***Events*** represent things that happened in the domain (e.g., Order Placed, Payment Received).
- ***Reactions*** describe how the system or domain responds to events (e.g., triggering business rules or processes).
- ***Commands*** are explicit requests to perform an action (e.g., Ship Order, Cancel Subscription), typically issued in response to events or user interactions.

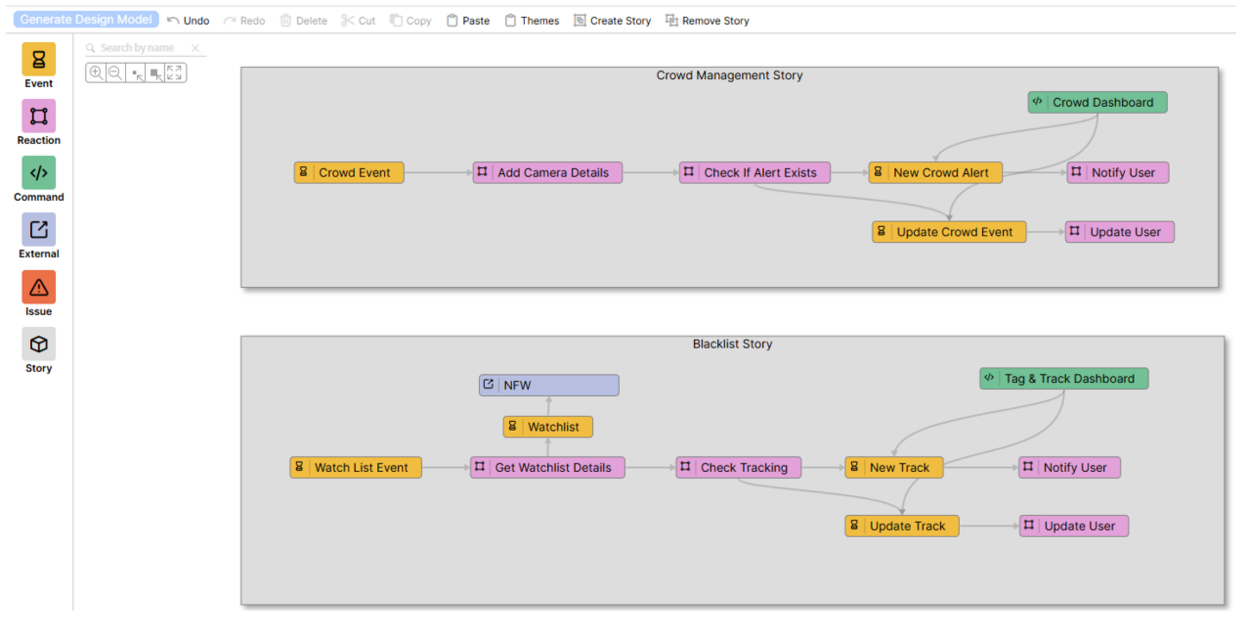


Figure 6 - Requirements Model

4.1.2 Define or Generate the Design – Design Modeler - *(optional start point)*

The Design Modeler is a visual drag-and-drop tool for modeling a Vantiq application using Services, Clients, and External Systems, and modeling the flow of events between these resources.

- **Services**: Represent real or placeholder Vantiq Services.
- **Clients**: Represent real or placeholder Vantiq Clients.
- **External Systems**: Represent external sources of events such as event brokers, etc.

Event Routes can also be defined within the Design Model. These can either flow between existing event interfaces or define new event interfaces on the Services and Clients. These Event Routes represent loosely coupled event flows in that the individual services are not publishing or subscribing to these events, but rather the internal event bus is routing these events between the Services.

If the developer does not wish to start with the requirements model but at the design model, then they are able to provide a textual description of their application that can help bootstrap the process or use the drag and drop interface to define the various Services and Clients.

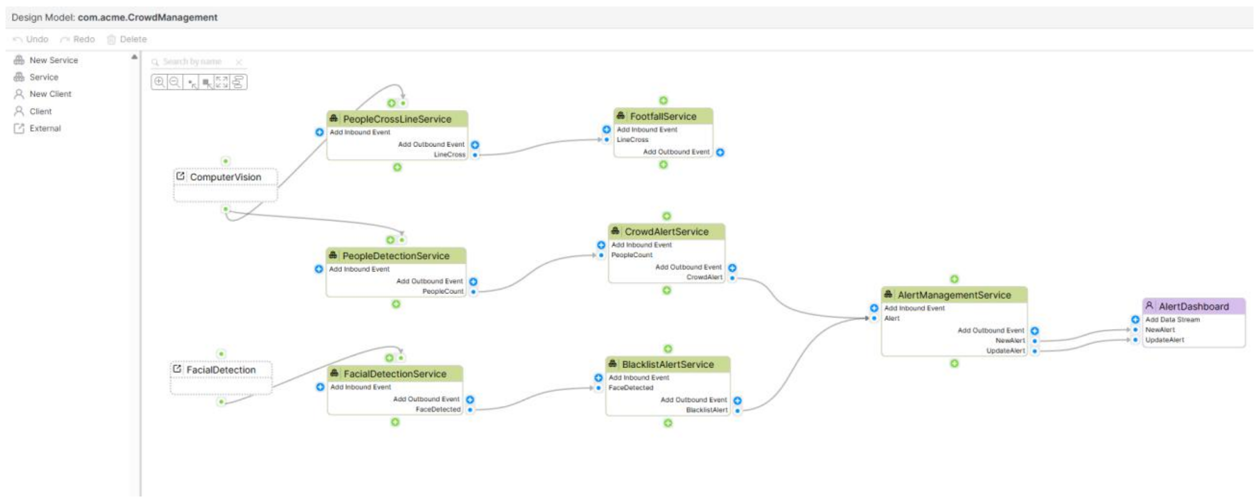


Figure 7 - Design Model

4.1.3 Define or Update Services – Service Builder

While designing your application, you will have identified several services and represented them on the Design Model. The next step in the process would be to begin implementing each of these services using the Service Builder. In addition to defining the Service using the Service Builder, the developer may also need to specify some additional resources, depending on the type of service.

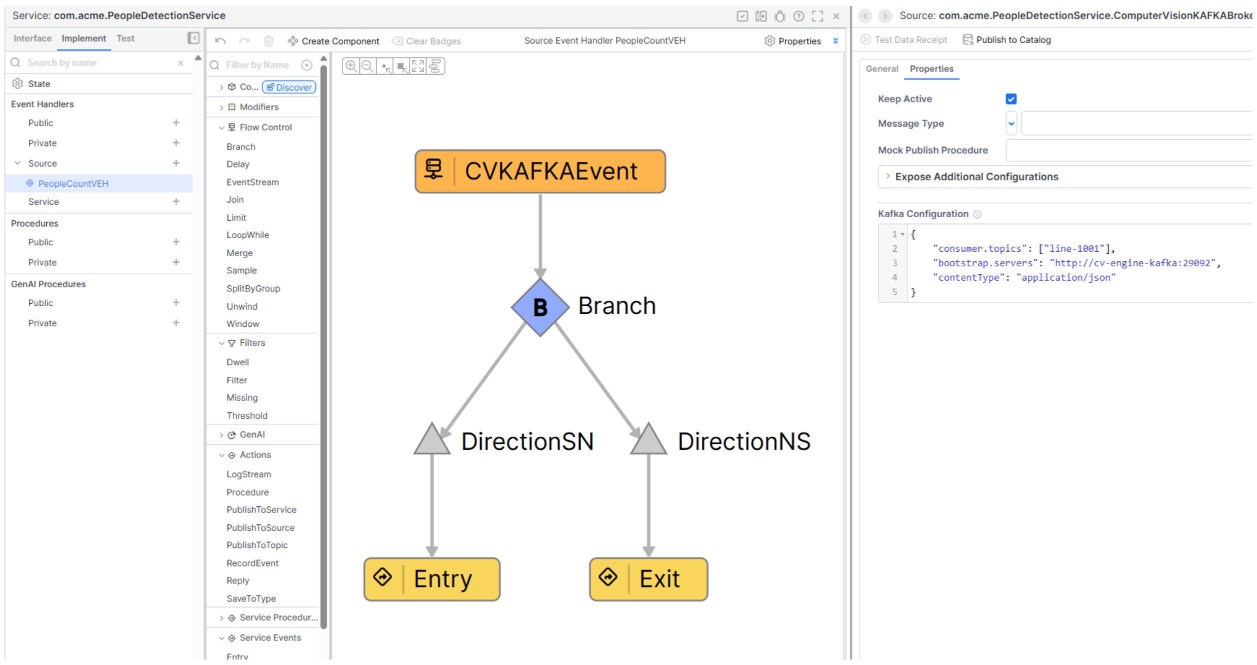
The Service Builder allows developers to define, implement, and test Services, which encapsulate functional behavior in an application. Services expose their capabilities through Procedures (synchronous operations) and Event Types (asynchronous messaging).

- **Interface**
  - Inbound Event Types (from consumers to the service)
  - Outbound Event Types (from service to consumers)
  - Procedures (synchronous function calls)

  This interface can be designed up front or evolve dynamically as implementation proceeds.

- **Implementation**
  - State Management – definition of the in-memory state that this service manages
  - Event Handlers - implemented as Visual Event Handlers (Visual Editor) or VAIL rules (JavaScript-like scripting language).
  - Procedures
    - VAIL Procedures: General-purpose procedures written in VAIL.
    - GenAI Procedures: Implemented using the GenAI Builder (Visual Editor) to integrate LLMs and semantic processing.
    - Automatic Interface Syncing

  Interface elements, such as event types and procedures, are auto-generated or updated based on visual handler definitions.



Keep Active ✓

Message Type

Mock Publish Procedure

Kafka Configuration

```
{
    "consumer.topics": ["line-1001"],
    "bootstrap.servers": "http://cv-engine-kafka:29092",
    "contentType": "application/json"
}
```

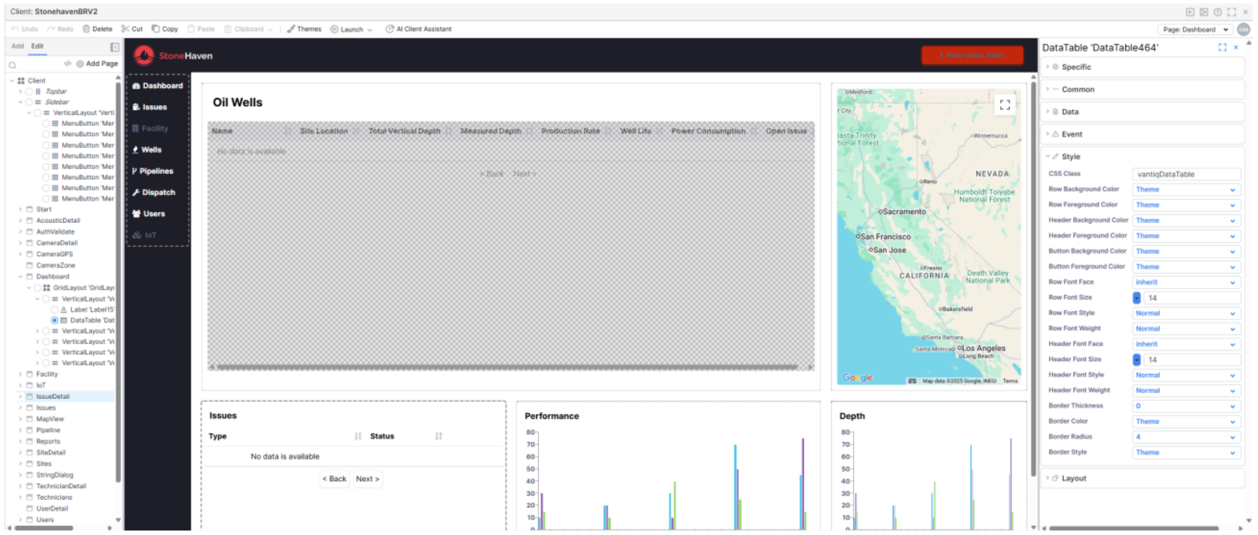
Figure 8 - Service Builder

4.1.4 Define or Update Clients – Client Builder

Vantiq Clients are developed using the Client Builder, which is a visual drag-and-drop tools that create HTML5/AngularJS Client applications that are hosted within the Vantiq platform.

- **Visual Development**
  - Drag-and-drop interface to build Pages with interactive Widgets.
  - Canvas view for real-time editing and layout.
  - Slide out property editor for configuring Widgets and code.
- **MVC Architecture**
  - Models: Use Data Objects for managing structured data and Data Streams for real-time event data.
  - Views: Build UI with Widgets like buttons, input fields, charts, maps, etc.
  - Controllers: Implement logic with JavaScript event handlers for user actions and lifecycle events.
- **Page Management**
  - Organize Clients into multiple Pages (each with its own UI and logic).
  - Built-in navigation with methods like goToPage and returnToCallingPage.
  - Optional persistent UI elements via Topbars and Sidebars.
- **Event-Driven Programming**
  - Attach event handlers to Client, Page, and Widget events (e.g., OnClick, OnStart).
  - Manage startup logic, cleanup tasks, and user interactions with JavaScript.
- **Data Integration**
  - Bind Widgets to Data Objects or Data Streams for dynamic updates.
  - Define Data Streams driven by server-side events, queries, or custom logic.
  - Support for real-time visualizations via charts, gauges, tables, and maps.
- **Customization and Extensibility**
  - Add reusable JavaScript fragments (Client or Namespace level).

- Include third-party JavaScript and CSS via Custom Assets.
- Customize appearance using Themes, Colors, and CSS.



CSS Class	vantiqDataTable
Row Background Color	Theme
Row Foreground Color	Theme
Header Background Color	Theme
Header Foreground Color	Theme
Button Background Color	Theme
Button Foreground Color	Theme
Row Font Face	inherit
Row Font Size	14
Row Font Style	Normal
Row Font Weight	Normal
Header Font Face	inherit
Header Font Size	14
Header Font Style	Normal
Header Font Weight	Normal
Border Thickness	0
Border Color	Theme
Border Radius	4
Border Style	Theme

Figure 9 - Client Builder

A non-Vantiq client would be developed externally to Vantiq and utilize one of the SDKs or vanilla REST calls.

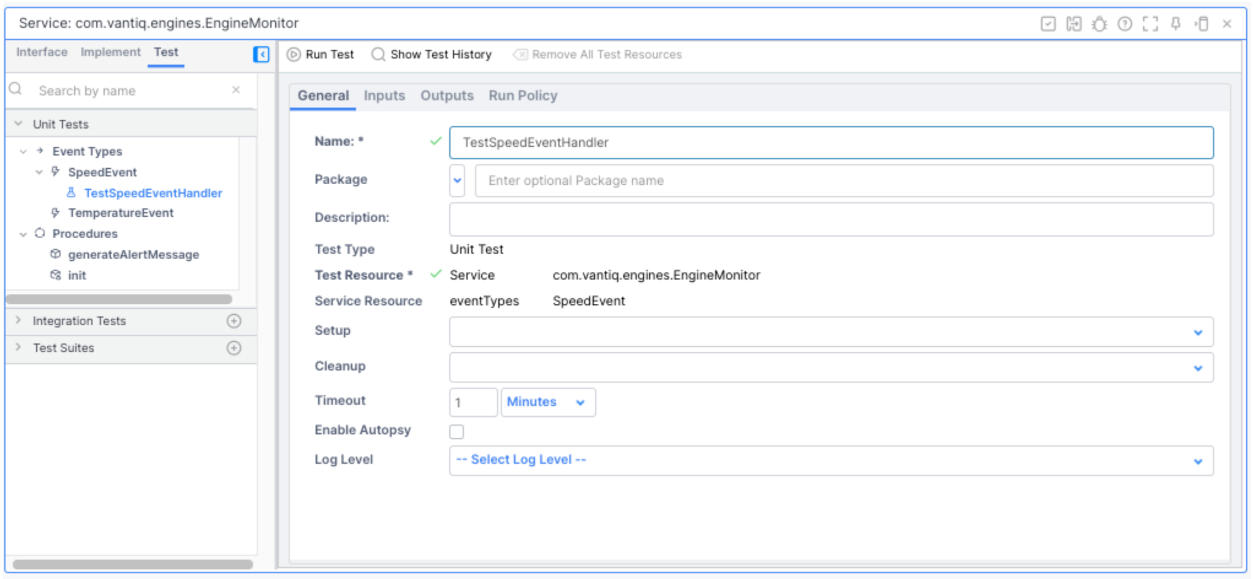
4.1.5 Define and Execute Tests - Service Builder

Vantiq provides robust testing tools to ensure the correctness of event-driven applications. Testing is integrated directly into the platform and supports both fine-grained and system-level validation.

- **Unit Tests**
  - Target individual components: Procedures and Event Handlers.
  - Run within the same namespace—fast and ideal for testing logic in isolation.
  - Validate inputs, outputs, and expected errors.
  - In the Service Builder, Unit Tests can test both public and private service elements, making them essential for testing private procedures or internal event handlers.
- **Integration Tests**
  - Test entire Projects or individual Services in isolation.
  - Deploys the resource to a clean testing namespace for realistic interaction.
  - Simulate external events and validate real-world behaviour.
  - Integration Tests for services are especially useful to validate service interfaces in production-like conditions.
- **Test Suites**
  - Organize and run multiple tests together.
  - Runs each test sequentially and generates individual and composite reports.
  - Supports scheduling through run policies.
- **Event Generators**
  - Used to generate a simulated stream of events
  - Generate synthetic event data to test application behavior without needing live external input.

- Define the structure and content of each event, allowing precise control over test scenarios.
- Emit events at defined intervals (e.g., every second or minute) for realistic pacing

- **Scheduled Testing (Run Policies)**
    - Automate test execution at fixed intervals (e.g., hourly, daily). Minimum interval: 1 hour.
    - Ensures regression tests run regularly without manual intervention.



Service: com.vantiq.engines.EngineMonitor

Name: *	TestSpeedEventHandler	
Package	Enter optional Package name	
Description:		
Test Type	Unit Test	
Test Resource *	Service	com.vantiq.engines.EngineMonitor
Service Resource	eventTypes	SpeedEvent
Setup		
Cleanup		
Timeout	1	Minutes
Enable Autopsy	☐	
Log Level	-- Select Log Level --	

Figure 10 - Testing

4.1.6 Deployment – Partitioning and Deploying Application

The Deployment Tool enables developers to manage the deployment of Vantiq projects across various environments (e.g., development, testing, production). It provides a graphical interface to organize, configure, and track deployments across distributed systems.

- **Nodes and Environments**
    - Nodes represent access points to Vantiq namespaces.
    - Environments group nodes logically for targeted deployment.
    - Nodes can be added to environments manually or dynamically via constraints.
- **Project Partitions**
    - Projects can be divided into partitions, which define what resources go to which nodes.
    - Auto-partitioning is based on PROCESSED BY statements.
    - Manual partitioning allows fine-grained control over deployment scope.
- **Deployments**
    - A Deployment links a project to an environment.
    - Graph and tree views visualize how resources are distributed.
    - Includes options for deploying as assemblies, configuring parameters, managing documents, and setting up secure secrets.
- **Deployment Management**

- Supports initial setup, updates, and rollbacks.
- Test and verify deployments with logs, error filtering, and resource tracking.
- Automatically creates projects on target nodes for deployed resources.

- **Advanced Deployment Features**
  - Reliable Deployment retries deployment on failed or newly added nodes until successful.
  - Selective Retry allows redeploying only to nodes that failed.
  - Secrets Management enables secure configuration via hidden credentials.

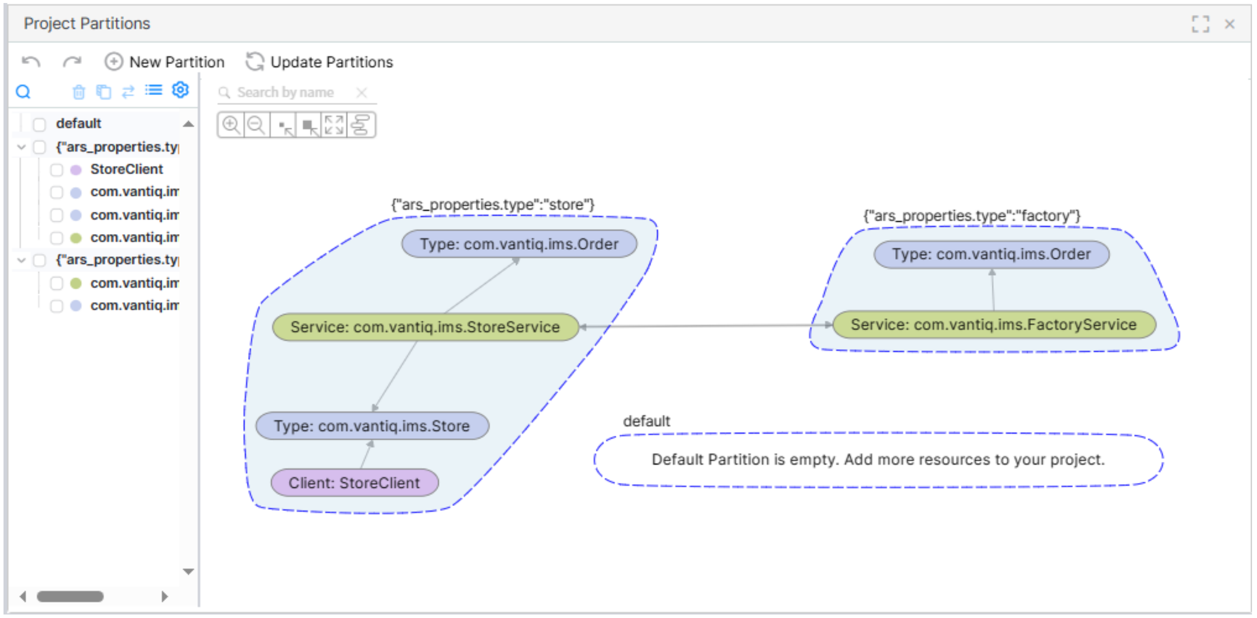

Project Partitions
New Partition
Update Partitions
Search by name
default
{"ars_properties.ty
StoreClient
com.vantiq.im
com.vantiq.im
com.vantiq.im
{"ars_properties.ty
com.vantiq.im
com.vantiq.im
{"ars_properties.type":"store"}
Type: com.vantiq.ims.Order
Service: com.vantiq.ims.StoreService
Type: com.vantiq.ims.Store
Client: StoreClient
{"ars_properties.type":"factory"}
Type: com.vantiq.ims.Order
Service: com.vantiq.ims.FactoryService
default
Default Partition is empty. Add more resources to your project.

Figure 11 - Partition & Deployment

4.1.7 Monitoring and Management of your Application – Grafana Dashboards

Vantiq integrates Grafana to provide real-time monitoring and performance insights for deployed applications. It allows developers, organization admins, and system admins to visualize system health, application behavior, and resource usage at different scopes—Namespace, Organization, and System-wide.

- Namespace-Level Monitoring examples.
  Visualize event flows, service and rule executions, source activity, and resource usage. Identify performance bottlenecks like:
  - High database activity.
  - Slow procedures or services.
  - Event drop rates and latency issues.
  - Dashboards include:
    - Event Processing.
    - Service Handler & Execution.
    - Procedure & Rule Execution.
    - Source Activity.
    - Resource Usage & Profiling.
    - Reliable Events.
    - Type Storage.
    - Storage Manager.

- Real-Time and Historical Analysis
    - View metrics over custom time windows.
    - Inspect specific time points or zoom into detailed trends
- Profiling & Debugging
    - Enable detailed profiling for services, procedures, rules, and apps.
    - Drill down into execution times, dependency performance, and latency via specialized dashboards.
    - Profiling data is temporary and configurable via debug settings.

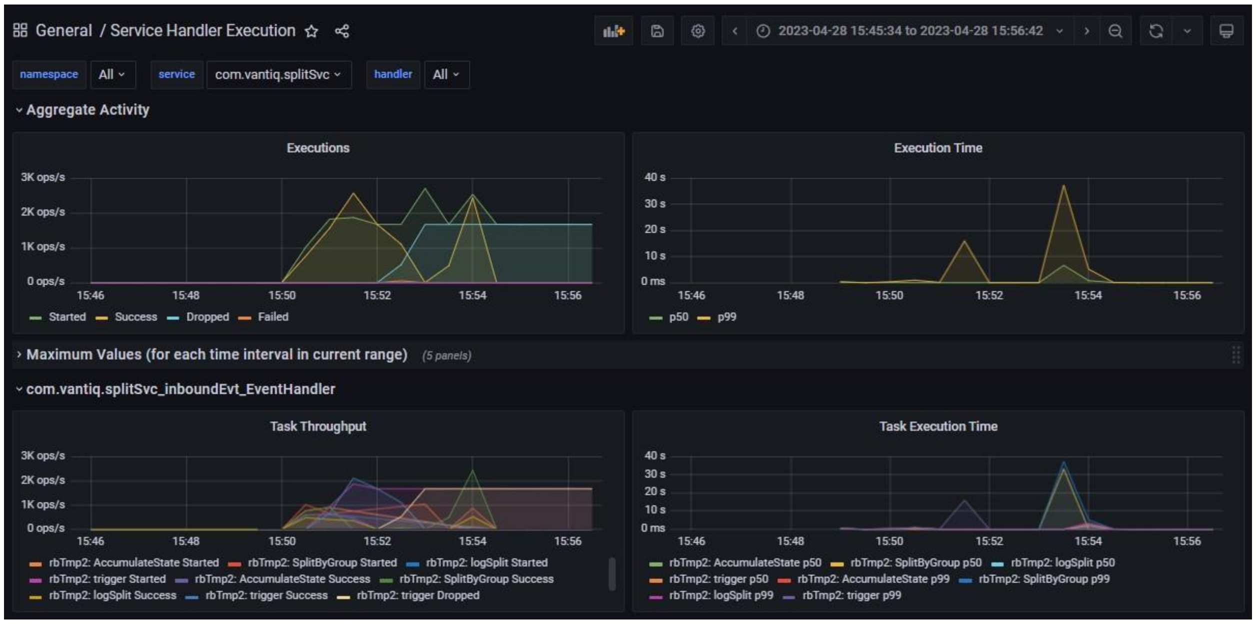

General / Service Handler Execution
2023-04-28 15:45:34 to 2023-04-28 15:56:42
namespace All
service com.vantiq.splitSvc
handler All
Aggregate Activity
Executions
Execution Time
Started
Success
Dropped
Failed
p50
p99
Maximum Values (for each time interval in current range) (5 panels)
com.vantiq.splitSvc_inboundEvt_EventHandler
Task Throughput
Task Execution Time
rbTmp2: AccumulateState Started
rbTmp2: SplitByGroup Started
rbTmp2: logSplit Started
rbTmp2: trigger Started
rbTmp2: AccumulateState Success
rbTmp2: SplitByGroup Success
rbTmp2: logSplit Success
rbTmp2: trigger Success
rbTmp2: trigger Dropped
rbTmp2: AccumulateState p50
rbTmp2: SplitByGroup p50
rbTmp2: logSplit p50
rbTmp2: trigger p50
rbTmp2: AccumulateState p99
rbTmp2: SplitByGroup p99
rbTmp2: logSplit p99
rbTmp2: trigger p99

Figure 12 - Grafana Monitoring

4.2 Bottom-Up Approach

In contrast to the top-down approach, where application requirements and design drive the implementation, the bottom-up approach starts with the construction of low-level components — primarily services and clients — before integrating them into a larger design model. This method is particularly effective for developers who already understand the problem domain, have existing backend integrations, or prefer prototyping key capabilities before modeling the entire system.

4.2.1 Build Core Services First

Begin by identifying and implementing reusable services that encapsulate data, logic, or external integration.

- **Integration Services**: Start by creating API or event-based integration services for existing systems (e.g., ERP, IoT sensors, RESTful APIs).
- **Event-Driven Services:** Implement core business logic that responds to and processes events. These services orchestrate workflows, apply decision logic, and trigger appropriate downstream actions in response to system or external events.
- **Data Services**: Define the internal data models and create services to manage CRUD operations and business reference data.

- **Utility Services**: Build helper functions (e.g., time conversion, ID generation) used across multiple parts of the application.
- **AI Agent Services**: Where applicable, wrap generative AI or ML models for classification, summarization, or prediction.

Use the Service Builder to implement procedures, event handlers, and stateful logic. Develop these components incrementally, testing each in isolation as they are built.

4.2.2 Create Clients and Interfaces

Once the foundational services are functional, proceed to define the user-facing clients that will consume them.

- Use the Client Builder to visually construct web or mobile interfaces.
- Bind UI components to service procedures and event streams.
- Create client-side logic using JavaScript for user interactions and real-time feedback.
- Focus on delivering core user tasks such as data entry, dashboard views, or notifications.

4.2.3 Iterative Approach

Development of your services and clients is often iterative in a bottom-up approach, and both resources will evolve as the developers implement new and additional functionality, allowing rapid prototyping and real-world testing early in the lifecycle.

4.2.4 Define Application Design Model

Although the Application Design is essentially defined iteratively by the definition of the service and clients, it is often helpful to be able to visualize your design through the definition of a Design Model. Also, if the intent is to utilize loose coupling, then this can only be achieved within the Design Model.

- Drag and drop existing Services, Clients, and External Systems into the visual design model.
- Define Event Routes to link components in a loosely coupled manner, capturing real-time event flows.
- Annotate the design with documentation, naming conventions, and relationships for clarity and future maintainability.

This reverse mapping from implementation to design helps validate architectural cohesion and enables stakeholder visibility into the system structure.

4.2.5 Testing and Iteration

Testing in the bottom-up approach is continuous and embedded from the start:

- Use Unit Tests to verify each service’s event handlers and procedures.
- Implement Integration Tests once services begin interacting with each other or external systems.
- Validate client behavior through simulated inputs or test namespaces.
- Employ Event Generators to simulate system behavior under load or over time.

Test results inform incremental refinements to both logic and architecture.

4.2.6 Deployment and Monitoring the Application

After building, integrating, and testing your services and clients, the final step in the bottom-up approach is to deploy and monitor the application in your target environment.

Deployment

Use the Deployment Tool to organize your application into partitions and deploy them across environments (e.g., development, staging, production).

- Define Projects and group related resources for structured deployment.
- Use Auto-Partitioning or manually specify which services and clients deploy to each node.
- Configure Environments and Nodes to align with organizational infrastructure.
- Use Assemblies for modular deployment of reusable components across projects.

Monitoring

Leverage Grafana Dashboards integrated into Vantiq to track system behavior and health in real-time.

- Monitor event throughput, service execution times, error rates, and source activity.
- Use profiling features to analyze performance bottlenecks in procedures, event handlers, and services.
- Set up Alerts and Logs to track anomalies and system degradation.
- Enable Reliable Events and retry policies for fault tolerance and resilient event delivery.
- This deployment and monitoring step ensures that the system is not only functional but also robust, observable, and scalable in real-world conditions.

5. Conclusion

The Vantiq Development Methodology offers a comprehensive and adaptable framework for developing real-time, event-driven, and situationally aware intelligent systems. By leveraging a service-oriented architecture combined with a low-code development environment, Vantiq empowers developers to rapidly design, implement, and deploy scalable applications that integrate tightly with both human and machine collaborators.

The methodology supports both top-down and bottom-up development approaches, ensuring flexibility for a wide range of use cases and development team preferences. Developers can begin with high-level requirement modeling and application design or start directly with building core services and user interfaces. Regardless of the starting point, Vantiq's tools—such as the System Modeler, Design Modeler, Service Builder, and Client Builder—provide seamless transitions between design and implementation.

Vantiq’s modular service types, including integration, event-driven, utility, data, and AI agent services, enable robust encapsulation of logic and external interactions. This modularity promotes reuse, agility, and maintainability across complex systems. Integrated testing, deployment management, and real-time monitoring through Grafana dashboards ensure that applications are not only performant and reliable but also easy to manage and evolve.

Ultimately, this development methodology reflects Vantiq’s vision of building intelligent, collaborative, and adaptive systems that respond dynamically to their environment. By combining powerful modeling tools, event-driven execution, and integrated AI capabilities, Vantiq equips developers to deliver the next generation of responsive, context-aware applications.