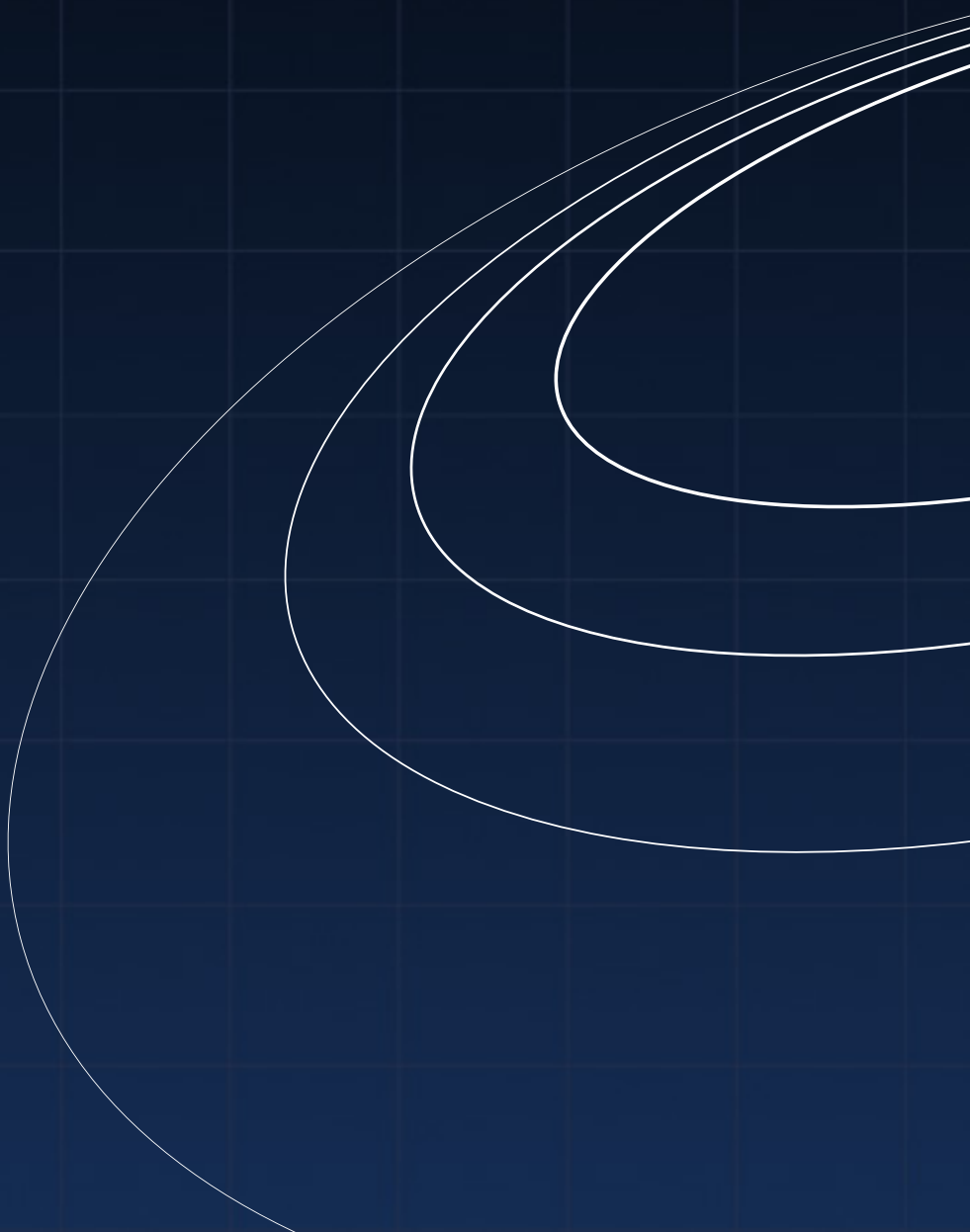
WHITE PAPER

# A blueprint for real-time recommendation systems



Aerospike

# Contents

# Introduction: Managing the complexity of today's recommendation engines

Recommendation systems are used widely across domains, including e-commerce, gaming, entertainment, and financial services, to engage users in the brief moments when decisions are made. Product recommendations, page personalization, and surfacing relevant search results are now essential to earning customer engagement in a world of crowded markets and companies fighting for consumer attention. In each case, the expectation is that systems respond in real time based on a user's known characteristics and actions.

Early well-known use cases include recommending a movie for later viewing, where minor latency may be acceptable. Increasingly, though, recommendation engines must deliver in real time, for instance, selecting which item to promote during checkout, which content module to load on a mobile homepage, or which power-up to surface in an online game. Even movie and TV apps now display personalized options the moment a user enters the app.

What makes this hard is the nature of the input. It is often partial, noisy, or missing altogether. Sometimes users make their intent clear through a specific request. Other times, they land on a page and scroll silently. To make relevant decisions in the moment, the system must quickly capture session signals, such as page context, recent history, engagement patterns, device, or location, then apply them instantly during inference. Even in the absence of a direct request, it must infer intent, evaluate options, and return something relevant.

This must be done quickly. User behavior, context, inventory, pricing, history, and other signals and features can be extensive, but the time available is short. Inference pipelines include multiple steps, such as feature lookup, scoring, and ranking. If early stages don't yield useful results, the system may add steps or try alternate logic, if the time window allows. Any of these, though, can be a bottleneck.

Modern recommendation systems must be able to handle multiple levels of data richness. They should support both anonymous and known users, limited context and detailed history, direct queries, and passive behavior. Across billions of user interactions, the system must capture what matters in the moment and use it immediately during inference. It must decide even when inputs are missing, delayed, or even contradictory.

**This paper presents:**

- A framework for how recommendation systems operate
- The architecture of modern, real-time recommendations
- The critical role of the feature store
- How Aerospike, a real-time database, powers these workloads

## The new context: more data, less certainty, real-time pressure

Recommendation systems today are data-rich and signal-poor. They ingest vast amounts of input such as clickstreams, impressions, product metadata, price updates, and more. But that volume does not guarantee clarity. The system often lacks a clear indication of user intent. Worse, some architectures skip valuable signals because accessing them in real time is expensive.

User signals are frequently partial. Someone may browse anonymously, block cookies, use multiple devices, or start a session with no historical context. Still, the system is expected to act. It must populate a homepage, select a next-best offer, or decide what not to show, often with limited or conflicting input.

Most modern recommendation engines are triggered by user actions such as visiting a page, opening an app, scrolling, or tapping. These real-time interactions dominate how relevance is delivered today. Some slower workflows still exist, including computing email recommendations or training models in batch, but they are exceptions. Architectures that prioritize these slower paths risk falling behind. Systems must be built to respond in the moment, using the freshest signals to guide every decision.

## Recommendations and the three-lens framework

Systems recommend content in many forms: a personalized email, a homepage carousel, a product detail page, or a search results page. Each represents a moment when the system selects what to show, based on signals from the session, the user's history, and current business priorities.

The specificity of that input can vary widely, from highly refined sessions where user intent is clear, to early-session visits or passive browsing with little context. What ties them together is the need to act quickly using session-level signals, even when that intent is only partially revealed.

We can frame these moments across three broad inputs:

- **User request:** What the user reveals about their intent in the current session, through explicit or implicit signals
- **User context:** What's historically known or inferred about the individual
- **Business context:** What the business wants to promote, avoid, or optimize for right now

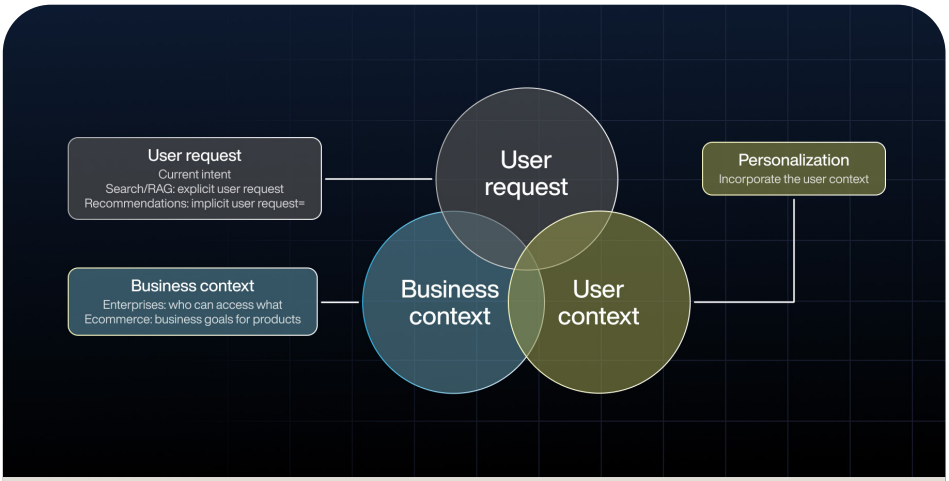



**Figure 1: Three key inputs for various types of recommendations.**

This lens-based framework gives us a way to understand how customer-facing systems behave and how they adapt to varying levels of input. The ideal personalized experience uses all three inputs: responding to a specific request, enriched by user context, and balanced by business goals.

But reality doesn't make it that easy. Systems often face incomplete data: a vague session with no clear request, an anonymous user with little history, or a business constraint that overrides personalization. That's why modern recommendation engines must be able to sequence their logic, falling back from request to context to business goals as needed, and combining signals when possible.

A user may arrive by clicking on a search result or an ad with no prior interaction, while another may have refined their search several times, revealing detailed intent. In each case, the system must adapt. These shifting inputs make it even harder to meet the tight latency windows required for real-time decisions.

The same three-lens visual can also help us map different types of recommendations and how they vary in complexity or maturity.

With these inputs in mind, we can now look at how recommendation systems are structured to operate in real time, even when signals are incomplete or delayed.

# Architectures

Modern recommendation systems combine offline and real-time components. They gather signals, generate features, train models, and deliver results within tight time windows. This section outlines the main stages and how they work together to support fast, relevant decisions.

## Recommendation architecture: Stages and flow

Recommendation engines are designed to make fast, intelligent decisions. To do that, they process data from multiple sources, including the user requests, user context, and business context of the lens framework we discussed.

While the architecture behind these engines varies, many share a similar structure made up of key components or stages. Some parts must operate in real time. Others can move, or at least tolerate, slower-moving batch updates. What's increasingly essential is the ability to connect long-term data with real-time signals in a way that keeps the system responsive and relevant.

Most of the decision comes from machine learning (ML), where models are trained, scored, and refined. But model quality depends just as much on the surrounding data pipeline. The value of real-time recommendations hinges on how inputs are processed, features are generated, and outputs are delivered.

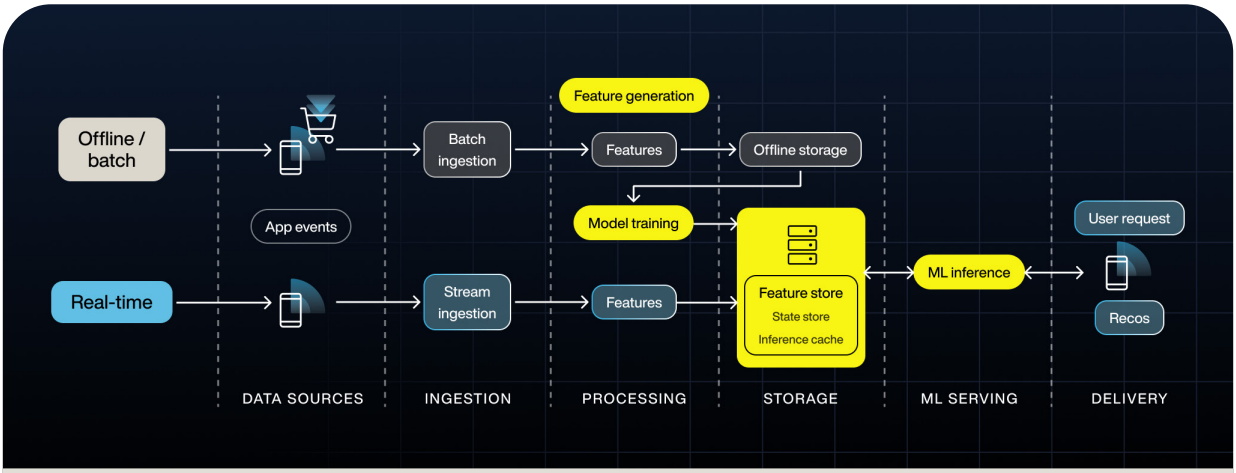



**Figure 2: Offline and real-time elements.**

Here's a breakdown of the main stages.

## Data sources

Every interaction a user has with the business, whether through an app, website, or other channel, can be captured and used as input. These include actions like entering a search query, clicking a product, adding to cart, or completing a transaction. Systems also rely on internal data such as content metadata, pricing, inventory, and promotional logic to shape results. Together, these interactions and assets form the core data corpus that powers recommendations. Some of this data arrives and evolves in real time, while other parts are collected and enriched over time, such as user history or catalog changes to available items, content, or services.

## Ingestion

Data enters the system through both batch and real-time streams. Batch ingestion is used to periodically retrain global models based on evolving behavior across individuals, segments, and the broader population. Real-time ingestion captures a subset of user events as they happen, such as clicks, page views, product interactions, video plays, or in-app behaviors, and feeds them directly into the inference pipeline. This enables the system to incorporate session-level signals and adjust recommendations in the moment, without waiting for the next batch update.

## Processing and feature generation

Once data enters the system, it's cleaned, joined, and transformed into features, often using complex logic. These features are the core inputs for model training and inference. They may reflect short-term user activity, product trends, or calculated metrics like conversion rates. Some are pre-computed in batch, others are generated in real time.

## Storage

Feature stores play a central role in modern ML systems. They are designed to manage, serve, and reuse features consistently across both training and inference workflows. Raw inputs may land in object storage, where they can be processed and transformed into features. Those features are then stored in systems tailored to their access patterns. Some may be used in batch training jobs. Others are needed for real-time inference and must reside in low-latency databases or caches.

However, not all feature stores can serve this real-time need. Many were originally built for training workflows and were not

designed to support fast, online access. As real-time inference grows increasingly common, the need for production-grade feature stores that support both training and inference is critical.

Some systems include adjacent infrastructure such as state stores or inference caches. A state store may provide raw or derived features similar to those in a feature store, while an inference cache typically stores the outputs of models rather than their inputs. While these components serve different roles, they may appear side by side in production environments. What matters most is that the feature store delivers consistent, low-latency access to model-ready data during inference.

### Model serving

This is where models are run in production. Trained models receive feature inputs and return predictions, rankings, or scores. These can be simple heuristic models or more complex architectures served through ML platforms or custom APIs. The goal is to keep inference fast and reliable, even when features arrive in real time and change constantly. As we'll detail ahead, model inference pipelines are often further broken into stages, depending on the complexity and latency requirements.

### Delivery

This is the final step, where results are pushed back to the application or service that requested them. It might be a product page, search result, notification engine, or in-app message. This layer is where speed and accuracy become most visible. Even a well-scored recommendation is useless if it can't be delivered in time or the right format.

## Recommendation architecture: Edge and core design considerations

Real-time delivery depends on more than just fast inference pipelines within a data center or availability zone. For global applications, minimizing network latency across countries and continents is equally important. To meet these demands, many architectures replicate both the feature store and model-serving infrastructure. This happens within clusters to support high request volume and availability, and across regions to bring data and models closer to users.


**Figure 3: Geographic distribution for real-time global inference.**

This diagram shows how core infrastructure supports large-scale batch processing and model training, while edge environments handle real-time inference and delivery. Designing for both is key to delivering fast, consistent customer moments at scale.

## Multi-stage model serving under real-time constraints

Serving in real-time systems is often more than a single scoring step. For many recommendation engines, the serving stage involves multiple steps that build on each other. This structure helps the system handle large input sets or ambiguous signals by narrowing options before applying more detailed logic. But there's a tradeoff. Each additional step adds potential latency, so the entire flow must be tightly optimized to keep response times within acceptable bounds.

A common pattern in recommendation systems is the two-stage approach: retrieval followed by re-ranking. The first stage quickly selects a subset of candidates from a large product or content pool, often containing millions of items. The second stage applies a more compute-intensive model to score and sort those candidates using user and business context. This approach balances scalability with precision, enabling fast decisions that remain highly relevant.

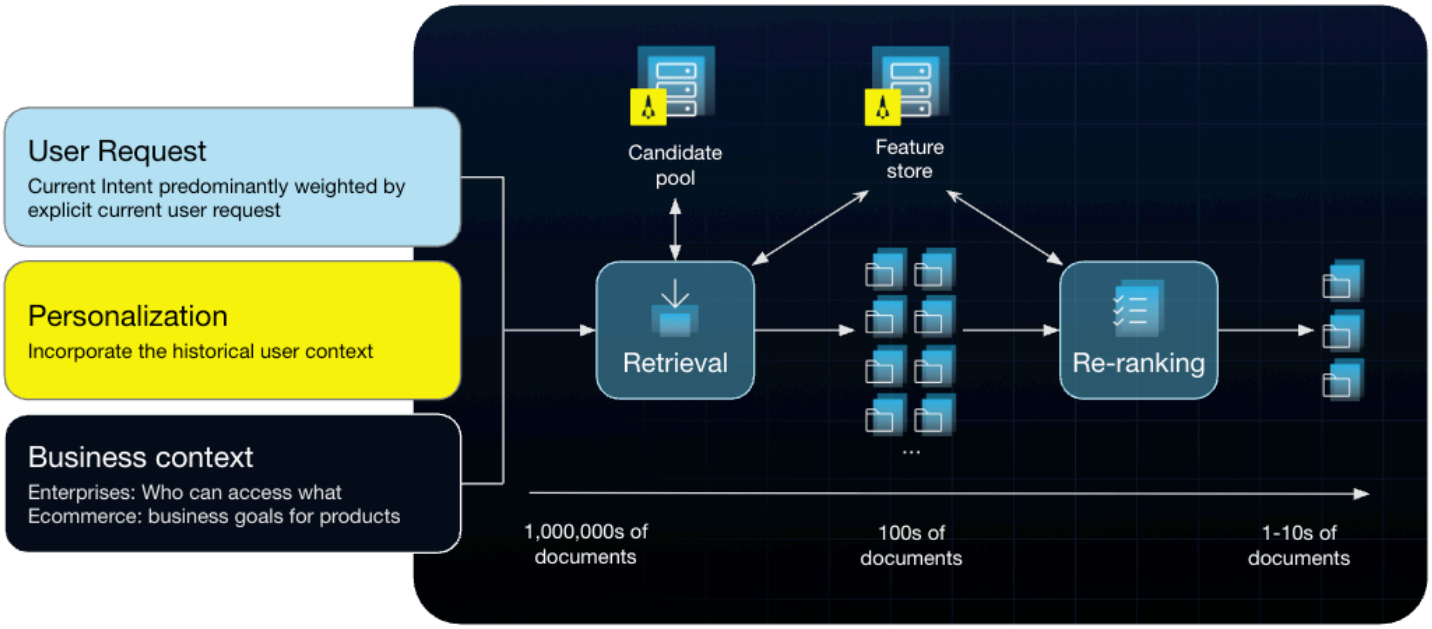



**Figure 4: Two-stage recommendation: Retrieval followed by re-ranking**

### Retrieval

The goal of the first step is to reduce the size of the search space. Rather than apply a complex scoring function to every item in the catalog, the system retrieves a smaller set of plausible candidates.

This can be done using simple rules, keyword filters, vector-based similarity search over learned embeddings, or a combination of these techniques. Depending on the use case, systems may rely on structured databases for fast lookups, keyword search engines like Elasticsearch, or hybrid approaches that combine keyword filtering with vector search. The retrieval step is designed for speed and high recall. It prioritizes getting the right items into the pool, even if they are not yet ranked accurately. Because this step often runs across millions of items, it depends on highly optimized data structures and fast access to indexes, embeddings, or precomputed features.

### Re-ranking

Once a candidate set has been retrieved, the system applies more expensive models to score and sort those items. This is where the ranking becomes personalized and context-aware. These models may use a wide range of features: user session history, item attributes, recency signals, and more. Some systems use gradient-boosted trees or neural networks, while others rely on transformer-based models that compute fine-grained similarity between query and item. Re-ranking is typically limited to hundreds or fewer items and must be completed within tight latency bounds. It puts significant pressure on the storage stage of the pipeline, since fine-grained features for each user-item pair must be fetched in real time.

### Additional modeling stages when time allows

Adding a few more steps beyond the standard two-stage architecture can significantly improve recommendation quality, especially to handle sparse information on user identity, history, or clear intent.

The system might first check an inference cache to see if a recommendation already exists for a similar request. If not, it fetches known user features from a feature store and supplements them with short-term session context, such as recent clicks or page views.

After these three steps, retrieval and re-ranking use the combined signals to select and sort a candidate set. Each step adds value, but also consumes time. The ability to run all five stages within tight latency budgets depends on fast access to features and session data, and efficient infrastructure throughout the pipeline.

# Feature stores: Where real-time pressure shows up first

The feature store is sometimes treated as behind-the-scenes infrastructure, useful for batch training, model reproducibility, or sharing features across teams. But in real-time recommendation systems, it is often where performance issues can emerge if the design is not built for production.

## Beyond batch: Why inference raises the stakes

Feature stores support training pipelines by managing feature definitions, enabling governance, and ensuring consistency across experiments. However, in real-time recommendation systems, they also serve inference workloads that are highly sensitive to latency and freshness. Models depend on a combination of signals from the current session (user request) and historical behavior (user context). Some of this is generated in batch, while some must be computed and retrieved within milliseconds.

As real-time inference grows more complex, the upstream pipelines described earlier become even more critical. They must use robust stream processing to handle changing inputs and enable diverse real-time access patterns. The feature store depends on them to compute and deliver model-ready data with the necessary latency and consistency.

A feature store must support all of these patterns with consistency and speed. It needs to provide model-ready inputs in real time, even during peak traffic, and irrespective of where the user is located geographically. It must also handle a wide range of freshness windows, including historical aggregates, rolling updates, and current user session signals.

As shown in Figure 5, slow feature lookups shrink the time available for inference. That limits how many features can be considered, how many models can run, and ultimately how strong the recommendation can be. A faster, high-throughput feature store leaves more room for modeling, enabling additional steps and larger feature sets within the same time window. As the figure highlights, a high-throughput feature store provides more time for modeling, supports more modeling steps, and allows more data to be considered per decision.


**Figure 5: Feature store latency vs. inference capacity**

Understanding these pressures is key to defining what truly makes a feature store production-grade for real-time systems.

## What makes a feature store production-grade

A real-time feature store must do more than store data. It must handle diverse data formats such as structured fields, lists, nested objects, and embeddings. It must maintain consistent definitions between training and serving. And it must deliver features with predictable latency, not just in the average case but at the tail end, where delays can compromise end-to-end performance and user experience.

Many issues that appear to come from model complexity or slow serving infrastructure actually originate in the feature store. Stale data or slow lookups can degrade the quality of recommendations, even when the underlying models are sound. That is why we treat the feature store as a core part of the architecture. It is not a supporting tool but a central enabler of real-time, production-grade recommendation systems.

# Infrastructure requirements and how Aerospike delivers

Modern recommendation systems push infrastructure to its limits. Data is massive, signals vary in quality, and expectations for relevance and responsiveness continue to rise. To support real-time inference at scale, the system must meet several key requirements.

Most feature stores and databases fall into one of two extremes. Some achieve speed by relying heavily on memory, but struggle with cost and data volume. Others lean on SSDs and caching, which can help with cost but often introduce latency and complexity.

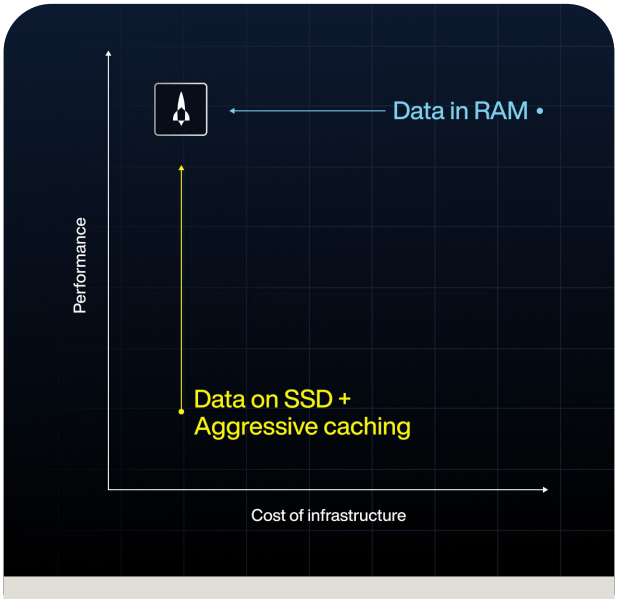



**Figure 6: Performance vs. infrastructure cost tradeoff**

Aerospike takes a different approach. Its architecture combines the speed of memory with the cost profile of flash storage. This unique design allows Aerospike to deliver performance typically associated with in-memory systems while handling large data volumes cost-effectively. As shown in Figure 7, this positions it in a quadrant that most systems cannot reach.

Here's how Aerospike addresses each of the key infrastructure requirements:

### Low and predictable latency

Real-time pipelines require consistency, not just average speed. Aerospike delivers sub-millisecond reads at scale using memory-resident indexes, efficient data structures, and optimized read paths that maintain tail latency even under peak load.

### High throughput and concurrency

Aerospike supports millions of reads and writes per second. It handles simultaneous user sessions and diverse surfaces without bottlenecks, making it a strong fit for high-traffic, latency-sensitive applications.

### Support for mixed workloads

Real-time inference workloads are rarely clean or isolated. Aerospike supports concurrent reads, stream ingestion, time-to-live (TTL)-driven freshness, batch writes, and backfills without locking or performance degradation.

### Graceful handling of sparse or rich sessions

Some users arrive with detailed histories. Others are new or anonymous. Aerospike supports flexible, conditional lookups that adapt to the available context, enabling strong results even when inputs are partial.

### Replication and global availability

Aerospike supports both intra-cluster replication and asynchronous Cross Datacenter Replication (XDR), allowing data and models to be located close to users around the world, with strong consistency and availability guarantees.

### Flexible data modeling

Evolving schemas, nested structures, and complex feature types are all supported. Aerospike's flexible data model allows teams to ship changes faster and adapt their pipelines without disruptive refactors.

### Efficiency at scale

Aerospike's architecture was built for scale. Hybrid Memory Architecture (HMA) reduces memory requirements while maintaining speed. This lowers infrastructure costs, supports experimentation, and removes scaling as a limiting factor for the business.

Meeting these requirements is what separates experimental systems from those that generate real revenue. Aerospike provides that foundation with an architecture designed for production and proven at scale.

## Real-world examples and design insights

Aerospike powers recommendation systems across industries and regions. While use cases differ, these organizations share the need for real-time inference, consistent low-latency lookups, and high-performance infrastructure. Many choose Aerospike for its ability to deliver sub-millisecond reads at scale, simplify pipeline complexity, and reduce total cost of ownership.


Sony Interactive Entertainment uses Aerospike to personalize the PlayStation experience across its global user base. The system supports over 100 million monthly active users, serving game and content recommendations in under 10 milliseconds. Aerospike enables this scale with high-performance lookups over 5TB of player data while ensuring fast personalization even in regions with lower bandwidth.

Myntra

Myntra, a major fashion e-commerce platform in India, re-architected its real-time recommendation pipeline to reduce latency and improve consistency. They built a feature store with Aerospike at the core, supporting homepage recommendations and personalization modules. Their architecture emphasizes session-level context, model chaining, and infrastructure simplification through stateless design and Kubernetes orchestration.

Quantcast

Quantcast uuses Aerospike to power real-time content recommendations as part of its personalization and audience targeting engine. A feature store backed by Aerospike delivers user and model-level features with sub-millisecond latency, enabling rapid predictions and decisioning. This architecture supports high-throughput inference workloads that drive personalized content delivery across Quantcast's platform.

wayfair

Wayfair uses Aerospike to support real-time product recommendations, including experiences on the homepage and throughout the shopping journey. With over 30TB of data and heavy user concurrency, they rely on lookup speeds under 10 milliseconds to power these personalized experiences at scale.

Rakuten

Rakuten's media division uses Aerospike to support targeted ad delivery and content personalization. The system supports real-time inference and feature storage for user segmentation, helping serve relevant content and promotions across web and mobile.

Flipkart

Flipkart, one of India's largest e-commerce companies, operates at enormous scale, especially during peak sale events. Its recommendation systems support product discovery, user experience, and promotional effectiveness across key surfaces like search, browse, and homepages. Real-time inference is essential, as user paths trigger personalized product listings and next-best offers. To support these demands, Flipkart built a centralized database-as-a-service platform to standardize infrastructure and adopted Aerospike Enterprise to handle the growing complexity of its workloads. Their recommendation workloads span memory-first clusters for high-throughput use cases and hybrid SSD clusters for broader personalization. These systems are supported by Kubernetes-based automation and a custom monitoring engine.

# Blueprint summary and next steps

Companies that fail to deliver relevant, timely experiences risk losing customers to those that do. Building high-performing recommendation systems requires not only better models but also the right infrastructure to support them.

This paper outlined the architectural realities of modern recommendation systems: large data volumes, partial or conflicting input, and narrow time windows for inference. In this environment, model quality matters, but so does system performance. Fast feature access, predictable latency, and support for high-concurrency inference pipelines are all critical.

Aerospike helps enable these systems by acting as a real-time feature store. It supports structured and unstructured data, sub-millisecond reads, and a scalable architecture proven in production across e-commerce, AdTech, FinTech, and media.

If you are building or modernizing a real-time recommendation system, consider these questions:

- Can your system serve features with millisecond latency at high concurrency?
- Do you support recency logic, versioning, and reuse across workflows?
- Can your infrastructure scale with user growth and usage spikes without driving up cost?
- Are you confident in your system's ability to act without clear user input?
- Is slow feature retrieval eating into your inference window?

For teams facing these challenges, Aerospike offers a proven foundation for real-time recommendation systems.

If you're looking to modernize or scale your architecture, we can help. Contact Aerospike to:

- Schedule a technical architecture session with our experts
- Explore real-world benchmarks of the Aerospike database
- Review Aerospike customer stories solving similar challenges in productionBlueprint summary and next steps

# About Aerospike

Aerospike is the real-time database for mission-critical use cases and workloads, including machine learning, generative, and agentic AI. Aerospike powers millions of transactions per second with millisecond latency, at a fraction of the cost of other databases.

Global leaders, including Adobe, Airtel, Barclays, Criteo, DBS Bank, Experian, Grab, HDFC Bank, PayPal, Sony Interactive Entertainment, The Trade Desk, and Wayfair, rely on Aerospike for customer 360, fraud detection, real-time bidding, profile stores, recommendation engines, and other use cases.

Try Aerospike for free: aerospike.com/try-now



**2440 W. El Camino Real, Suite 100, Mountain View, CA 94040 | (408) 462-2376 |** aerospike.com