Inside the Vendor Management Platform's Architecture

Nataraj Agaram Sundar[1], Tejas Morabia[1], Hari Rangarajan, Ph.D.[1], Piyush Shrivastava[1] and Jeganathan Srinivasan[1]

[1]eBay Inc.

October 30, 2024

Abstract

This technical report adapts the LinkedIn article "Inside the Vendor Management Platform's Architecture" into a Scholar-ready format. It describes how eBay approached vendor integration at scale by introducing a Vendor Management Platform (VMP) to standardize onboarding, vendor metadata, connector patterns, and lifecycle operations across more than 50 payment and risk vendors. The report contrasts a code-heavy, iPaaS-like current state with a bottom-up VMP architecture centered on connector standardization, controlled execution, guard rails, and reusable platform services.

Keywords: vendor management platform, enterprise integration, digital commerce, API integration, iPaaS, observability, vendor onboarding, dependency injection

Author note. This technical report adapts the LinkedIn article *Inside the Vendor Management Platform's Architecture* into a Scholar-ready archival format. Section structure and language closely follow the original article, with light normalization for technical-report presentation. Figures in this version are formatted for print and web clarity.

1 The Challenge in Digital Commerce

Marketplaces rely on essential vendor integrations, varying from a few to dozens, depending on size and scope. A small domestic marketplace might suffice with one payment gateway and a couple of risk tools, while a larger international one may require multiple gateways, along with risk management, compliance, AML, and sanctions screening tools. At eBay, the source article notes, the platform integrates with over 50 vendors to manage payments and risks [1].

Consolidation trends favor platforms offering integrated services to simplify operations, though the cost escalates exponentially, impacting payment processing expenses.

In digital commerce, particularly for large enterprises, managing vendor relationships presents challenges. Historically, dispersed management across platforms or departments created inefficiencies, including complex infrastructure, redundancy, subpar experiences, and unnecessary costs.

2 Introducing the Vendor Management Platform (VMP)

To tackle these inefficiencies, the authors introduced the Vendor Management Platform (VMP) at eBay. The project is designed to refine vendor assessment, onboarding, and integration. Initially

focusing on API-based integrations, VMP aims to consolidate vendor information and simplify integration procedures in ways that reduce operational cost and create a more cost-effective environment for both eBay and its vendors.

Given the scale of active integrations and the substantial budget dedicated to this area, the need for a streamlined system is clear. VMP serves as a centralized hub for operations, offering a straightforward interface for vendor onboarding and managing vendor data throughout its lifecycle. It is designed to elevate the developer experience by accelerating onboarding, improving issue detection and resolution, and driving down costs.

3 Integration Types

API integrations involve extensive coding efforts for data enrichment and interface development, along with the need for robust observability and security measures. Integrating cloud-hosted vendor solutions presents a different challenge set, including interoperability inside private data centers and maintaining data security and compliance.

VMP specifically addresses the former by simplifying and standardizing the API integration process in a way that supports cost savings, improved operational efficiency, and a more agile response to the needs of a dynamic digital marketplace.

4 Understanding Vendor Integrations

Before diving into VMP, it is useful to evaluate the typical changes required to integrate a new vendor. Each integration usually requires API integration between eBay and the vendor, data enrichment through cache, DB, or domain service, persistence design to save vendor results, observability for vendor performance, network security setup, and capacity evaluation for downstream components.

The current-state integration pattern has several major drawbacks, including code-heavy implementation, redundant audit and observability logic, no clear distinction between vendor and domain business metrics, increased maintenance cost, and limited standardization when multiple applications integrate with the same vendor. Figure 1 summarizes that burden by showing how each application ends up owning custom code plus repeated operational concerns around every vendor call.

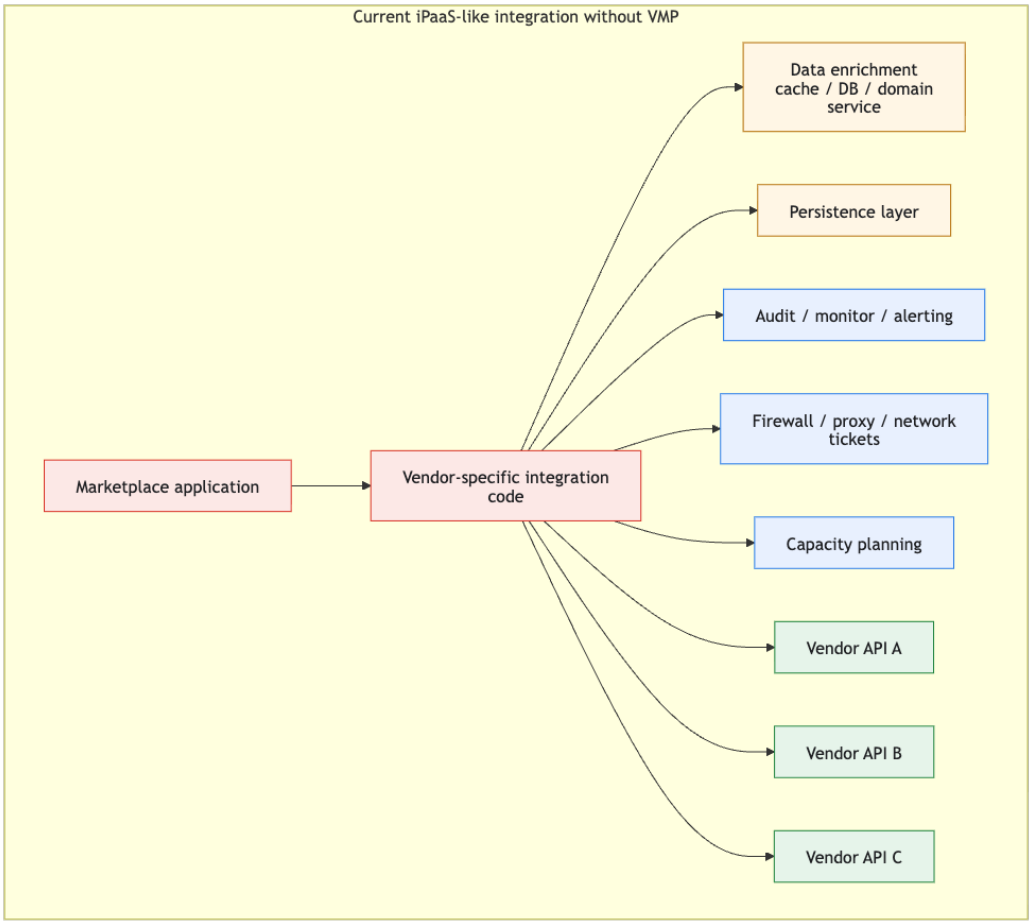

Figure 1: Current-state vendor integration pattern without VMP. A marketplace application carries vendor-specific integration code and repeatedly wires enrichment, persistence, observability, security, capacity planning, and direct vendor API connectivity.

5 iPaaS vs VMP

Integration Platform as a Service (iPaaS) typically follows a top-down approach, focusing on integrating disparate systems at a high level without enforcing standardization at the registry and data sink levels. VMP adopts a bottom-up approach, emphasizing standardization at the connector level and introducing a new paradigm in vendor management and integration.

iPaaS: Top-Down Approach

The article explains that iPaaS platforms focus on integrating cloud-based and on-premises applications and services for data exchange and process orchestration. However, their top-down approach often leads to a lack of standardized structures at the registry and sink levels, resulting in a messy integration landscape in which each integration evolves with its own rules and inconsistencies. While this can accelerate early development, it often increases long-term complexity.

VMP: Bottom-Up Approach

VMP is designed to streamline and standardize the management of vendor relationships, data, and processes. It acts as a centralized platform for vendor onboarding, data management, and communication, ensuring a cohesive approach to vendor interaction.

In this bottom-up approach, the platform standardizes connectors by dynamically assembling and injecting the required connectors and configurations based on vendor-specific needs and predefined standards. This yields a modular and flexible model in which new integrations can be added or modified without extensive reconfiguration of the broader system.

By enforcing a standardized connector framework, each vendor integration adheres to a set of predefined rules and structures, ensuring consistency, reducing complexity, and facilitating maintenance and scalability. The result is a more organized integration landscape with less clutter and lower operational overhead.

6 Comparative Analysis

VMP lets vendor-specific requirements drive the dynamic assembly and injection of dependencies, fostering a modular and scalable integration ecosystem. By standardizing connectors and adapting dynamically to vendor needs, VMP maintains a cleaner and more manageable integration landscape than the inconsistent scenarios often encountered in iPaaS solutions.

VMP also decouples execution logic from execution runtimes and leverages microservices to improve reliability, observability, and scalability. Figure 2 captures the target-state architecture: client logic remains outside the platform while VMP centralizes onboarding records, controlled execution, logging, guard rails, and standardized vendor connectors.

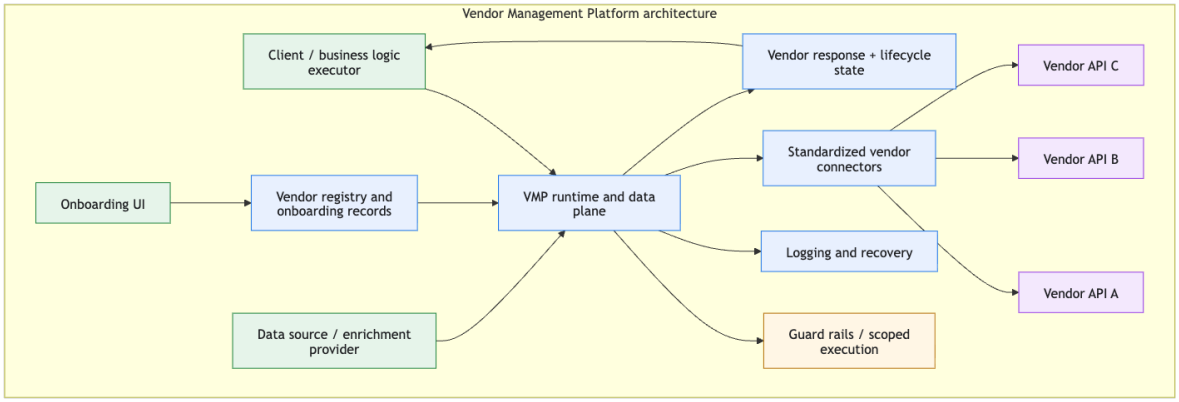

Figure 2: Vendor Management Platform target-state architecture. VMP centralizes onboarding records, controlled execution, guard rails, logging, standardized connectors, and lifecycle state while business logic remains in the client layer.

7 High-Level Design

VMP aims to solve the drawbacks described above by transforming the current-state architecture into a governed target-state platform. At its core, VMP serves as the operational hub, responsible for onboarding vendors via a user-friendly interface and managing the data plane across the full lifecycle of vendor data. It is engineered to initiate vendor calls and is equipped with logging and recovery capabilities that improve resilience and operational integrity.

The client component executes core business logic around vendor calls. It interfaces with VMP to request and receive vendor data, acting as the business logic executor and data consumer. The data-source component provides the data required by the data plan and vendor-specific needs, simplifying onboarding and provisioning.

Guard rails are integrated to ensure VMP does not extend beyond its intended scope, such as executing business logic outside vendor onboarding and vendor call execution. This preserves system integrity while focusing platform capabilities on vendor lifecycle management. Figure 3 shows the operational flow from onboarding request through registry updates, configuration planning, data preparation, controlled execution, vendor invocation, and downstream logging and consumption.

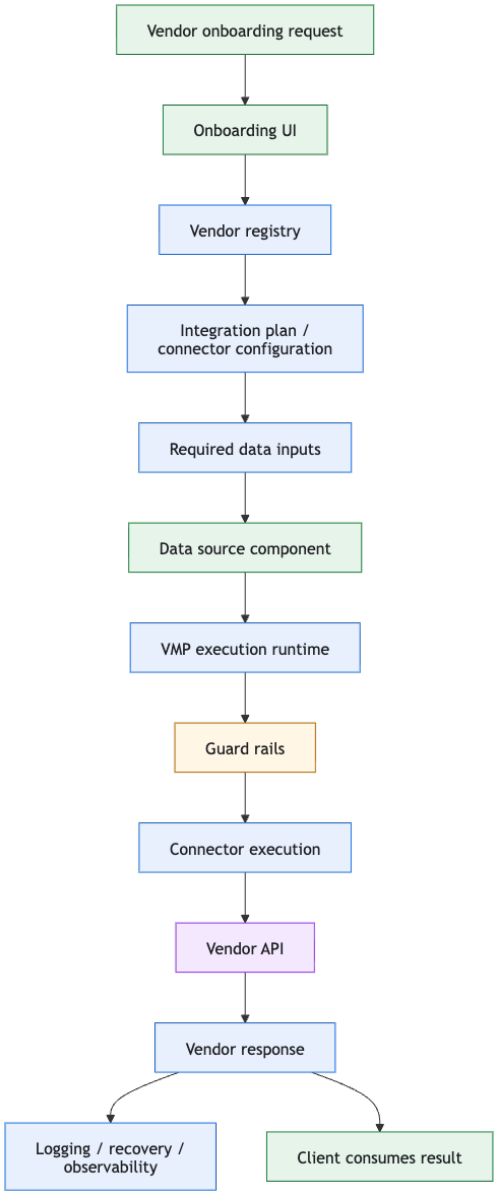

Figure 3: Vendor onboarding and execution workflow. The sequence moves from onboarding request and registry updates through configuration planning, data sourcing, controlled VMP execution, vendor invocation, response handling, logging, and client consumption.

8 Conclusion

As described in the original article, the architecture of VMP is intended to make vendor integration more standardized, scalable, and observable by centralizing onboarding, normalizing connectors, and separating business logic from vendor lifecycle execution. The broader claim is that organizations should stop treating each vendor integration as a bespoke engineering project and instead build a governed platform beneath the business workflow.

References

[1] Sundar, N., Morabia, T., Rangarajan, H., Shrivastava, P., & Srinivasan, J. (2024, October 30). *Inside the Vendor Management Platform's Architecture*. LinkedIn.

[2] G. Hohpe and B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley, 2003.

[3] M. P. Papazoglou and W.-J. van den Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415, 2007.

[4] S. Newman. *Building Microservices*. O'Reilly Media, 2015.

[5] C. Richardson. *Microservices Patterns: With Examples in Java*. Manning, 2018.

[6] M. Kleppmann. *Designing Data-Intensive Applications*. O'Reilly Media, 2017.

[7] J. Humble and D. Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley, 2010.

[8] J. Turnbull. *Effective Observability: 10 Specific Ways to Improve Observability in Distributed Systems*. O'Reilly Media, 2023.

[9] OWASP Foundation. *OWASP API Security Top 10 – 2023*. OWASP, 2023.

[10] M. T. Nygard. *Release It!: Design and Deploy Production-Ready Software*. 2nd edition, Pragmatic Bookshelf, 2018.

[11] M. Fowler. Inversion of Control Containers and the Dependency Injection pattern. Martin-Fowler.com, 2004.