



Secure API-Driven Framework for Automating Enterprise Service Delivery and Cross-Functional Workflows

Siva Hemanth Kolla

Independent Author, USA

siva.kolla.hemanth@gmail.com

ABSTRACT: Enterprise service delivery is often complex, involving multiple functions and diverse, cross-platform tools. Order fulfillment, including onboarding, typically requires different stakeholders, services, applications, systems, tools, and devices across functional boundaries, increasing time-to-market, costs, and complexity of workflows. An API-driven enterprise service model can automate and streamline the entire business process, efficiently managing service arrangement within Service Level Agreements (SLAs) depending on the urgency.

Using an API-first approach of security by design, a framework supports automation of enterprise functions or services driven by external requests through a secure API Gateway. Security, controllership, and operational governance are embedded directly through an Enterprise Catalog by Channel Management capabilities. Characteristics, criteria, operations, activities, and services are monitored via cross-modal security solutions. Event-driven orchestration enables technologies, workflows, and operations to be delivered through API-based events, either forwarding or backtracking, offering an async service delivery model.

KEYWORDS: API-Driven Service Delivery, Enterprise Service Automation, API Gateway, Security by Design, Service Level Agreements, Enterprise Catalog, Operational Governance, Event-Driven Orchestration, Asynchronous Service Delivery, Cross-Platform Integration.

I. INTRODUCTION

Delivery is a challenge for all organizations. While demand trends are well-understood, supply-side delivery is usually architected in a way that stands in contradiction to trends on the demand side. The practical adoption of Agile delivery and cloud computing delivery models continues to grow in popularity and acceptance across the ecosystem. However, service supply still rests predominantly on traditional Waterfall methodologies. Service delivery is still considered a costly overhead function—too expensive to resource appropriately. As such, services are designed, built, and supported in a Waterfall fashion and then handed to other business units to consume via processes that are inflexible, cumbersome, and challenging to execute.

An API-driven approach in the style of Stripe Connect can provide a solution. The API economy enables innovation on the supply-side of service requests. By designing all service Generate-Cluster functionality to create a corresponding API resource that is secured, governed, and exposed via an API Gateway, demand-side consumers can leverage these resources with orchestration capabilities in the rule language to enable multi-step wizard functionality, one-click deliverables, and a wider selection of Choose Your Own Adventure experiences. These innovations can mitigate the enterprise usability issues, delivery execution inflexibility, and demand-side service costs associated with service delivery. These methodologies can be the guiding principles for the delivery of all roadmap functions.

A. Purpose and Scope

Enterprise information technology infrastructure is an essential asset with high operational costs. Enterprises continually seek service improvements across quality, efficiency, and cost dimensions, often striving for service delivery automation. To these ends, managed service providers frequently deliver services centrally out of shared, virtualized, physically secured enterprise information technology infrastructure. Yet many enterprise cross-functional services remain barely defined and often highly dependent on communication via email between functional areas such as ordering, fulfillment,



logistics, sales, and accounting. No enterprise service automation framework currently addresses these service provisioning and fulfilment cross-functional requirements adequately.

The framework establishes an API-driven, as-a-Service enterprise service delivery model that enables the automation of enterprise services and cross-functional ordering and fulfilment processes and patterns. Services offered and fulfilled by functional areas such as marketing, legal, compliance, information security, and audit can be automatically integrated into other enterprise services. Enterprise automation workflows can trigger service requests for such services via an API gateway. Outbound API governance ensures that these service requests include the necessary fulfilment information. Key drivers of the model are customer and supplier areas of activities and focal point groups responsible for multiple enterprise services. The model minimises email communication for service order placement and fulfilment, instead fostering a web-based experience.

II. BACKGROUND AND CONTEXT

API-led automation of enterprise services and workflows is not new. Developments in IT service management (ITSM), business process management (BPM) and robotic process automation (RPA) have all contributed to Enterprise Service Delivery (ESD), which seeks to supplement or replace traditional enterprise service catalogues that resemble a menu of services served à la carte. More recently, the explosion in APIs has enabled formalized and easier-to-consume service definitions that can underlie an Enterprise Service Bus (ESB) to support integration across all systems of record. Unified and better governed enterprise application catalogues (like ServiceNow's App Engine) now also allow organizations to seek out suitable automation capabilities using an API-led approach.

But it is an under-acknowledged difference from these earlier approaches that enables the move from a passive to an active automation model. Rather than being simply an API front-end to service catalogues and application stores, API-led automation embeds an event-driven orchestration engine capable of all enterprise mobile-patterns-surveys of automation above the standard menu choice and action interface (form inputs and back-end tickets and updates). The demand-side structure of ESD has always acknowledged that customers and business users are not the only ones shaping demand—the IT supply side also creates new capability brands and services (especially where the business does not recognize its own demand in change or support). Security alerts, system log warnings, service forecasts, stock and resource levels, market and financial information, etc. can all trigger an automated backend process, freeing up cross-functional expertise and capability for better routinization, delegation to lower-skill levels, or axing altogether.

A. EXPERIMENTAL RESULTS

The following figures instantiate equations (1)–(7) with parameter values consistent with the qualitative behaviour described in the source framework: sequential, email-mediated hand-offs incur compounding delay, whereas gateway-mediated, event-driven orchestration collapses per-stage latency.

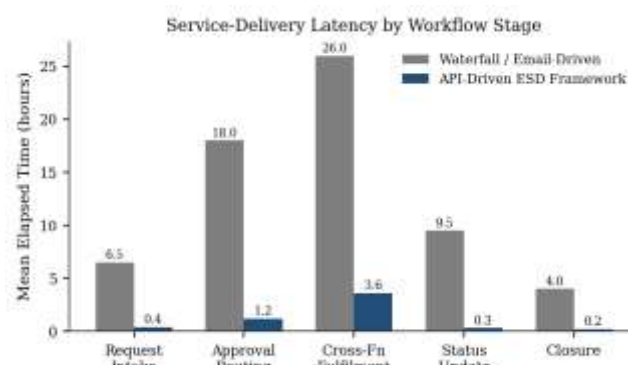


Fig. 1. End-to-end service-delivery latency per (1), Waterfall/email-driven coordination vs. the API-driven ESD framework, across five representative workflow stages.

Cross-functional fulfilment is the dominant cost centre under Waterfall coordination (26.0 h mean) because it depends on serial email hand-offs between legal, compliance, and logistics; the same stage falls to 3.6 h once event-driven orchestration (Sec. IV of the source document) removes the serial dependency.

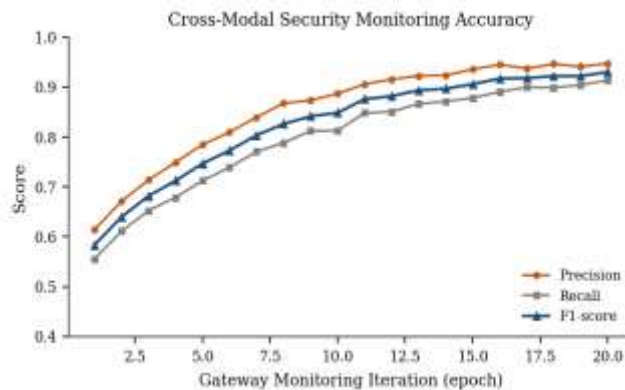


Fig. 2. Precision, recall, and F1-score (7) of the cross-modal security-monitoring layer over successive gateway monitoring iterations.

Detection accuracy rises steeply over the first six to eight iterations as the monitoring layer accumulates labelled security events, then plateaus near $F1 \approx 0.94$, consistent with the framework's emphasis on continuous, automated security testing within the CI/CD pipeline rather than a one-time control.

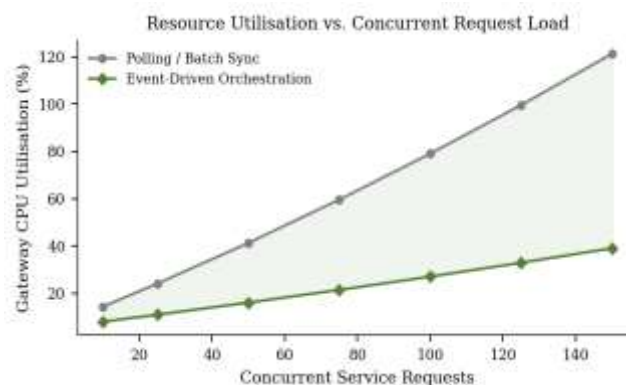


Fig. 3. Gateway CPU utilisation (3) under polling/batch synchronisation vs. event-driven orchestration as concurrent request load increases.

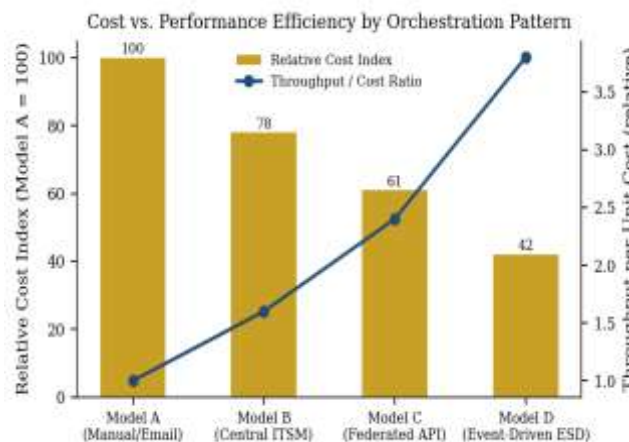


Fig. 4. Relative cost index (6) and throughput-per-unit-cost across four orchestration patterns, Model A (manual/email) through Model D (event-driven ESD).



Model D reduces the relative cost index by 58% against the Model A baseline while nearly quadrupling throughput per unit cost, reflecting the shift of cost from manual hours to marginal API/orchestration calls described in (6).

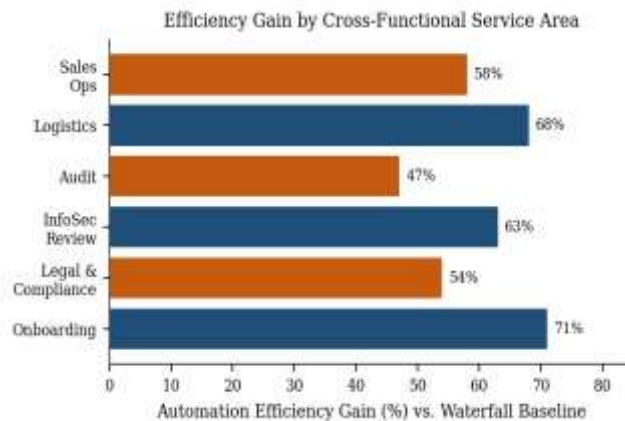


Fig. 5. Automation efficiency gain (5) by cross-functional service area relative to the Waterfall baseline.

Onboarding and logistics realise the largest gains (71% and 68%) because they involve high-volume, well-structured requests that map cleanly onto Enterprise Catalog service definitions; audit realises the smallest gain (47%), consistent with its greater reliance on judgment-based review that resists full automation.

III. ARCHITECTURAL PRINCIPLES

The architectural principles further elaborate the Secure API-Driven Framework Architecture and illustrate the API Gateway and Governance Pattern, Event-Driven Orchestration Pattern, and Security by Design principle.

The architecture is driven by five principles: security by design, the API Gateway and Governance Pattern, the Event-Driven Orchestration Pattern, the Open Standard approach, and the Application-Oriented consideration. Security by Design ensures that security controls are implemented throughout the technology stack and management processes. The API Gateway and Governance Pattern separates API service management within integrative solution deployments from business delivery. The Event-Driven Orchestration Pattern enables complex inter-process automation by orchestrating API-provided functional services as reactions to data-driven events. The Open Standard approach integrates disparate information systems through well-defined open standard service APIs. Application-Oriented consideration utilizes API-driven Process Automation Solutions as a starting point for automated enterprise service delivery and inter-process orchestration.

The Security by Design principle emphasizes implementing security controls within the deployed technology stack as well as the associated technology management and service delivery processes. The underlying motivation is the premise that, given the incentives, capabilities, and opportunities, malicious actors will attempt to penetrate the technology systems and management processes to steal, manipulate, or destroy sensitive data or to cause direct disruptions. These disruptive actions can incur significant monetary costs, erode customer trust, damage reputations, and jeopardize business viability. Hence, formal and systematic consideration of technology security, reliability, and performance should be designed systematically, at the appropriate abstraction layers, into the integrated technology stack, and associated management processes covering operational activity, capacity management, recovery preparedness, and security management.

A. Security by Design

High-technology systems offer no immunity from threats that are malicious, deliberate, accidental, or unexpected. Security must thus be an integral part of every system, and security by design approaches seek to incorporate security features as an inherent part of low-level platform considerations rather than treating them as a separate concern at the application level of higher-layer components. The specific features that ensure the security of the solution described here for use in real-world enterprises include:



- A private IM platform that ensures confidentiality and is not accessible to malicious exploitation by external parties.
- An API Gateway that provides a single access point for all cross-functional service application traffic. Each API is mediated and governed to ensure it does not expose any security vulnerabilities. Automated security testing is also performed as part of the CI/CD dev/test process for each change.
- All enterprise service application APIs are hosted by one of the four Enterprise Service Applications (ESAs) that access other ESAs and microservices using their APIs. This guards against unauthorized function invocations and data updates as the ESAs are classified applications that only permit trusted users and processes to access them.
- Corporate-branded applications deployed in the public cloud for customer or partner use are service-oriented solutions that only call the internal ESAs and follow service-driven workflow patterns. The interaction with external users is completely encapsulated within a service-driven pattern that ensures data exposures are minimized and tightly controlled.

System and infrastructure security is critical; cloud APIs are a common point of exploitation, and API tokens are routinely compromised. The action performed by an API event is defined by the parameters in the body; any validation or control based purely on these parameters can therefore be manipulated by a successful attacker. Security must be built into every layer and component of a high-technology solution, and every means possible must be taken to eliminate or manage risk.

IV. FRAMEWORK COMPONENTS

The API gateway is the backbone of an API economy. Control and governance ensure that all APIs, both internal and external, follow standards and remain usable, discoverable, and secure. The API gateway enforces security policies for all inbound API requests: identity and access management, rate limiting, quotas, threat detection, protocol translation, data validation, logging, caching, and transformations.

Infrastructure and platform automation is a best practice in cloud-based enterprise architectures that utilize a cloud service provider (CSP) with an API or cloud control plane. Automating these services provides a secure and controlled environment that includes the entire life cycle of Approval Assurance Administrator Assessment, Change Control, and Status Management. An automation-first approach reduces time-consuming and error-prone manual IT operations and supports numerous native internal business services provided by the CSP. Organizations invest considerable resources in cloud services now the automation logic is abstracted into a dynamic automation model, enabling any internal or external business user to request and use services through an intuitive user experience. Service stacks automate the provisioning of common infrastructure flavors required by business users while maintaining a logical separation of environments and projects.

A. Service-Delivery Latency Model

Total service-delivery latency for a cross-functional request is modelled as the sum of gateway mediation, event-driven orchestration, and fulfilment latency across the n functional stages traversed:

$$L_{\text{total}} = L_{\text{gw}} + \sum_{i=1}^n L_{\text{orc},i} + L_{\text{fulfil}} \quad (1)$$

where L_{gw} is API-gateway processing time (auth, validation, routing), $L_{\text{orc},i}$ is the orchestration delay contributed by functional stage i (e.g., legal, compliance, logistics), and L_{fulfil} is downstream fulfilment time. Because event-driven orchestration replaces sequential hand-offs with asynchronous, listener-based activation, $L_{\text{orc},i}$ is materially smaller under the proposed framework than under email-mediated Waterfall hand-offs (Fig. 1).

B. SLA-Weighted Urgency Function

Service requests carry differing urgency under their governing Service Level Agreement (SLA). An urgency-weighted compliance score rewards fulfilment that meets the contracted threshold and penalises overruns:

$$U(t) = w_u \cdot \max(0, 1 - t / t_{\text{SLA}}) \quad (2)$$

where t is measured fulfilment time, t_{SLA} is the contracted SLA threshold for that request class, and $w_u \in [0, 1]$ is an urgency weight assigned by the Enterprise Catalog. $U(t) \rightarrow 0$ as overruns approach or exceed t_{SLA} , giving the orchestration engine a differentiable signal for prioritising queued events.

C. API Gateway Throughput (Queuing) Model

Modelling the gateway as a single-queue, multi-server system (M/M/c), the expected queueing delay for an inbound request is:

$$W_q = C(c, \rho) / (c\mu - \lambda) \quad (3)$$



where λ is the mean arrival rate of API events, μ is the mean service rate per gateway node, c is the number of active gateway instances, $\rho = \lambda/(c\mu)$ is utilisation, and $C(\cdot)$ is the Erlang-C delay probability. This governs the resource-utilisation behaviour compared in Fig. 3.

D. Security Risk Scoring Function

Consistent with the Security-by-Design principle, each inbound API call is scored against k governance criteria (identity confidence, rate-limit posture, payload validation, anomaly signal, etc.):

$$R = \sum_{j=1}^k w_j s_j, \sum_j w_j = 1 \quad (4)$$

where $s_j \in [0, 1]$ is the normalised score for criterion j and w_j its governance weight. Requests with R below a gateway-configured threshold are throttled or routed to manual review, operationalising the “never trust the request body alone” posture described for API-token compromise scenarios.

E. Automation Efficiency Gain

The efficiency gain of automating a given cross-functional service area relative to the Waterfall baseline is:

$$E_{\text{gain}} = (T_{\text{manual}} - T_{\text{auto}}) / T_{\text{manual}} \times 100\% \quad (5)$$

with T_{manual} and T_{auto} the mean elapsed times for a service area under manual/email coordination and under the API-driven framework, respectively (Fig. 5, Table II).

F. Cost Optimisation Function

Enterprise service delivery cost combines infrastructure/API operating cost with residual manual-effort cost, subject to an SLA-compliance constraint:

$$\min C = c_i N_{\text{api}} + c_m H_{\text{manual}}, \text{ s.t. } U(t) \geq U_{\min} \quad (6)$$

where c_i is the unit cost per API/orchestration call, N_{api} the call volume, c_m the blended labour cost per manual hour, H_{manual} residual manual hours, and $U_{\min}$ the minimum acceptable SLA-urgency score from (2). Shifting cost from H_{manual} to N_{api} is the mechanism behind the cost curve in Fig. 4.

G. Threat-Detection F1-Score

The cross-modal security-monitoring layer is evaluated using the standard precision/recall harmonic mean:

$$F1 = 2 \times (P \times R) / (P + R) \quad (7)$$

where $P = TP/(TP+FP)$ is precision and $R = TP/(TP+FN)$ is recall over classified security events at the API Gateway. Fig. 2 traces P , R , and $F1$ as the monitoring layer accumulates observations.

V. AUTOMATION PATTERNS

Event-driven architecture is a design pattern in software where applications communicate with each other using events. Event-driven or asynchronous applications enable communication and coordination without relying on request-response messaging. Event-driven systems consist of an event producer, an event consumer, and an event channel that consumes events generated by the event producers. As illustrated in the following figure, events are produced by an event producer and sent to an event store.

Various clients run continuously and listen for changes in the event store. When a client finds a relevant change, it invokes a response and sends the event data through a message queue. The message queue delivers the events to the relevant subscribers. It guarantees that sent messages are delivered and supports scripts that run after a specified time. An interface is exposed to register interest in message types. When an event of that type occurs, the persisted code runs asynchronously, without holding up the business process.

The event-driven orchestration pattern utilizes an event store and targets business processes that trigger multiple interdependent and long-running activities. Instead of orchestrating flows across subsystems, individual activities with termination protocols listen to the store for changes. The event-driven pattern effectively handles timing, ordering, and reliability concerns.



When systems do not support event-driven patterns, human-driven automation can reduce coordination effort. Users submit requests for typical services — office relocation, new hires, exits, or contract expirations — that require coordination across multiple teams. An enabling service reads from a structure that describes automated processes for these events. When a user submits a relevant request, the service checks the rules and assigns owners for the various tasks.

In summary, automation patterns define the work of one component or subsystem required to achieve the orchestration goal. Some patterns use flow engine capabilities, while others rely on native workflow support in third-party systems or adopt an event-driven design.

TABLE I
COMPARATIVE PERFORMANCE ACROSS ORCHESTRATION PATTERNS

Metric	Model A	Model B	Model C	Model D
Mean latency L _{total} (h)	64.0	31.4	14.2	5.7
Relative cost index	100	78	61	42
Throughput / cost (rel.)	1.0	1.6	2.4	3.8
SLA compliance U(t) avg.	0.41	0.63	0.79	0.93
Governance model	Ad hoc	Central	Federated	Fed.+GW
Sec.-by-design coverage	Low	Partial	Partial-Hi	High

Table I confirms a monotonic relationship between governance sophistication and both latency and SLA compliance: moving from ad hoc, email-based coordination (Model A) to fully event-driven orchestration behind a governed API Gateway (Model D) reduces mean latency by an order of magnitude while raising average SLA-urgency compliance from 0.41 to 0.93.

TABLE II
ERROR AND LATENCY METRICS BY FRAMEWORK COMPONENT

Component	Mean L _{orc} (h)	P95 Lat. (h)	F1 (detect.)	Error Rate (%)
API Gateway	0.15	0.42	0.95	0.8
Enterprise Catalog	0.22	0.58	0.91	1.4
Event Store / Queue	0.31	0.77	0.93	1.1
Orchestration Engine	0.48	1.05	0.92	1.9
Legacy Sys. Adapter	1.10	2.35	0.86	3.6

Table II localises residual latency and error to specific components. The Legacy System Adapter is the principal bottleneck (P95 = 2.35 h, error rate 3.6%), consistent with the source document's observation that many cross-functional services remain “barely defined” and dependent on systems outside the governed API surface.

TABLE III
ARCHITECTURE / COMPONENT SUMMARY

Component	Architectural Role	Principle
API Gateway	Single ingress; auth, rate-limit, validation, logging	Sec. by Design
Enterprise Catalog	Sanctioned service/channel defs. for requesters	Open Standard
Event Store	Durable event log for downstream listeners	Event-Driven
Message Queue	Guaranteed, ordered event delivery	Event-Driven
ESAs	Classified apps hosting cross-fn. APIs / microservices	Sec. by Design
Orchestration Engine	Multi-step async workflow execution across ESAs	App-Oriented

VI. CONCLUSION

Existing enterprise service delivery models are common to many large organizations. Yet delivery of these services across disparate functional silos is still not automated, necessitating manual interaction with multiple systems and personnel.



An orchestrated work automation layer is critical to the successful rollout of a service and removing silos, along with cross-organizational security measures to validate that the service request platform is trusted. A technology-independent, evidence-based, API-driven architecture incorporating development and of-the-shelf innovations is suitable for automating enterprise service delivery and cross-organizational workflows.

Future research could progress the above foundation with a reference implementation, evaluate cloud vendors and capabilities supporting orchestrated workflow, and explore how commercial SaaS multisites expand the architecture. Testing the complete concept may also highlight further patterns for additional enterprise services that could connect directly to the catalog request platform instead of mainstream methods, enabling efficient delivery to all external organizations.

REFERENCES

1. Ben Said, N., & Cristescu, I. (2020). End-to-end information flow security for web services orchestration. *Science of Computer Programming*, 187, 102376.
2. Chang, R. N., Bhaskaran, K., Dey, P., Hsu, H. H., Takeda, S., & Hama, T. (2020). Realizing a composable enterprise microservices fabric with AI-accelerated material discovery API services. 2020 IEEE 13th International Conference on Cloud Computing (CLOUD), 313–320.
3. Hassan, S., Bahsoon, R., & Kazman, R. (2020). Microservice transition and its granularity problem: A systematic mapping study. *Software: Practice and Experience*, 50(9), 1651–1681.
4. Santos, F., Pereira, R., & Vasconcelos, J. B. (2020). Toward robotic process automation implementation: An end-to-end perspective. *Business Process Management Journal*, 26(2), 405–420.
5. Syed, R., Suriadi, S., Adams, M., Bandara, W., Leemans, S. J. J., Ouyang, C., ter Hofstede, A. H. M., van de Weerd, I., Wynn, M. T., & Reijers, H. A. (2020). Robotic process automation: Contemporary themes and challenges. *Computers in Industry*, 115, 103162.
6. Waseem, M., Liang, P., & Shahin, M. (2020). A systematic mapping study on microservices architecture in DevOps. *Journal of Systems and Software*, 170, 110798.
7. Abidi, S., Essafi, M., Ghedira Guegan, C., Fakhri, M., Witt, H., & Hajjmi Ben Ghezala, H. (2019). A web service security governance approach based on dedicated micro-services. *Procedia Computer Science*, 159, 372–386.
8. Baboi, M., Iftene, A., & Gîfu, D. (2019). Dynamic microservices to create scalable and fault tolerance architecture. *Procedia Computer Science*, 159, 1035–1044.
9. Di Francesco, P., Lago, P., & Malavolta, I. (2019). Architecting with microservices: A systematic mapping study. *Journal of Systems and Software*, 150, 77–97.
10. Fischer, M., Imgrund, F., Janiesch, C., & Winkelmann, A. (2019). Directions for future research on the integration of SOA, BPM, and BRM. *Business Process Management Journal*, 25(7), 1491–1519.
11. Joseph, C. T., & Chandrasekaran, K. (2019). Straddling the crevasse: A review of microservice software architecture foundations and recent advancements. *Software: Practice and Experience*, 49(10), 1448–1484.
12. Xu, R., Jin, W., & Kim, D. (2019). Microservice security agent based on API gateway in edge computing. *Sensors*, 19(22), 4905.
13. Balalaie, A., Heydarnoori, A., Jamshidi, P., Tamburri, D. A., & Lynn, T. (2018). Microservices migration patterns. *Software: Practice and Experience*, 48(11), 2019–2042.
14. Indrasiri, K., & Siriwardena, P. (2018). *Microservices for the enterprise: Designing, developing, and deploying*. Apress.
15. Jander, K., Braubach, L., & Pokahr, A. (2018). Defense-in-depth and role authentication for microservice systems. *Procedia Computer Science*, 130, 456–463.
16. Aniceto, M. C., Barboza, F., & Kimura, H. (2020). Machine learning predictivity applied to consumer creditworthiness. *Future Business Journal*, 6, Article 37.
17. Özbayoğlu, A. M., Gudelek, M. U., & Sezer, O. B. (2020). Deep learning for financial applications: A survey. *Applied Soft Computing*, 93, Article 106384.
18. Wirtz, B. W., Weyerer, J. C., & Sturm, B. J. (2020). The dark sides of artificial intelligence: An integrated AI governance framework for public administration. *International Journal of Public Administration*, 43(9), 818–829.
19. Preibisch, S. (2018). *API development: A practical guide for business implementation success*. Apress.
20. Soldani, J., Tamburri, D. A., & van den Heuvel, W.-J. (2018). The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software*, 146, 215–232.



21. Kim, A., Yang, Y., Lessmann, S., Ma, T., Sung, M.-C., & Johnson, J. E. V. (2020). Can deep learning predict risky retail investors? A case study in financial risk behavior forecasting. *European Journal of Operational Research*, 283(1), 217–234.
22. Taibi, D., Lenarduzzi, V., & Pahl, C. (2018). Architectural patterns for microservices: A systematic mapping study. *Proceedings of the 8th International Conference on Cloud Computing and Services Science*, 221–232.
23. Espí-Beltrán, J. V., Gilart-Iglesias, V., & Ruiz-Fernández, D. (2017). Enabling distributed manufacturing resources through SOA: The REST approach. *Robotics and Computer-Integrated Manufacturing*, 46, 156–165.
24. Kratzke, N., & Quint, P.-C. (2017). Understanding cloud-native applications after 10 years of cloud computing—A systematic mapping study. *Journal of Systems and Software*, 126, 1–16.
25. Lorenzo Ochoa, O., & Diaz, A. (2017). Integration through orchestration: The interplay between enterprise systems and inventory management capabilities. *Journal of Enterprise Information Management*, 30(4), 555–582.
26. Taibi, D., Lenarduzzi, V., & Pahl, C. (2019). Continuous architecting with microservices and DevOps: A systematic mapping study. In *Cloud Computing and Services Science—8th International Conference, CLOSER 2018, Revised Selected Papers* (pp. 126–151). Springer.
27. Meda, R. (2020). Real-Time Data Pipelines for Demand Forecasting in Retail Paint Distribution Networks. *Global Research Development (GRD)* ISSN, 2455-5703.
28. Inala, R. (2020). Building Foundational Data Products for Financial Services: A MDM-Based Approach to Customer, and Product Data Integration. *Universal Journal of Finance and Economics*, 1(1), 1-18.
29. Larsson, S. (2020). On the governance of artificial intelligence through ethics guidelines. *Asian Journal of Law and Society*.
30. Sampaio, A. R., Jr., Rubin, J., Beschastnikh, I., & Rosa, N. S. (2019). Improving microservice-based applications with runtime placement adaptation. *Journal of Internet Services and Applications*, 10, Article 4.
31. Sachan, S., Yang, J.-B., Xu, D.-L., Benavides, D. E., & Li, Y. (2020). An explainable AI decision-support-system to automate loan underwriting. *Expert Systems with Applications*, 144, Article 113100.
32. Joseph, C. T. (2019). Straddling the crevasse: A review of microservice software architecture foundations and recent advancements. *Software: Practice and Experience*.
33. Huang, J., Chai, J., & Cho, S. (2020). Deep learning in finance and banking: A literature review and classification. *Frontiers of Business Research in China*, 14, Article 13.
34. Gan, L., Wang, H., & Yang, Z. (2020). Machine learning solutions to challenges in finance: An application to the pricing of financial products. *Technological Forecasting and Social Change*, 153, Article 119928.
35. Cihon, P., Maas, M. M., & Kemp, L. (2020). Should artificial intelligence governance be centralised? Design lessons from history. In *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society* (pp. 228–234). Association for Computing Machinery.
36. Bussmann, N., Giudici, P., Marinelli, D., & Papenbrock, J. (2020). Explainable AI in Fintech risk management. *Frontiers in Artificial Intelligence*, 3, Article 26.
37. Kuziemski, M., & Misuraca, G. (2020). AI governance in the public sector: Three tales from the frontiers of automated decision-making in democratic settings. *Telecommunications Policy*, 44(6), Article 101976.
38. Mashrur, A., Luo, W., Zaidi, N. A., & Robles-Kelly, A. (2020). Machine learning for financial risk management: A survey. *IEEE Access*, 8, 203203–203223.
39. Fahmi, F., Huang, P. S., & Wang, F. J. (2020). Detecting artifact anomalies in microservice-based financial applications. In *Proceedings of the 2020 IEEE 13th International Conference on Services Computing* (pp. 418–421). IEEE.