The Eastasouth Journal of Information System and Computer Science
Vol. 1, No. 02, December, pp. 349-358

Event-Driven Decomposition Strategies for Migrating Monolithic Enterprise Platforms to Resilient Microservice Architectures

Shubh Prabhat
Architect, GlobalLogic Inc., Cary, North Carolina, USA

Article Info

Article history:

Received December, 2023
Revised December, 2023
Accepted December, 2023

Keywords:

Event-Driven Architecture;
Eventual Consistency;
Microservices;
Monolith Decomposition;
Resilience;
Strangler Pattern

ABSTRACT

Enterprise platforms are often operated in large monoliths. They are moved to microservices by teams to achieve scale, quicker releases and isolation of failures. Most of these moves do not provide resilience since services make long synchronous chains among themselves. The paper suggests an event-based decomposition plan to such migrations. There are five phases of the strategy. They consist of domain mapping, cut scoring, event contract design, strangler extraction and fault injection testing. The work is directed by two scores. A cut score is measured to assess the effectiveness of a service in retaining its calls and data within the boundary of the service. An event conversion ratio calculates the number of cross service interactions that consume events as compared to direct call. Design decisions are related to resilience through a path availability equation and an event lag equation. The test is carried out on a virtual order management system with 1.4 million code lines and 236 database tables. The site is divided into two services into eight parts. One of them is synchronous calls. The remaining uses the suggested strategy. The foundation of a monolith is the basis. The event driven version supports 1,000 orders/minute with a p95 latency of 360 milliseconds compared to 655 milliseconds in the synchronous version. The one service outage completes 91 percent against 47 percent of orders. The recovery time decreases from 22 to 9 minutes. The variations also decrease from 3.9 to 1.6 per 10,000 orders. After release, one of eight service boundaries had to change. The event driven version takes 38 person months compared to 31. The cuts should be influenced by event design.

This is an open access article under the CC BY-SA license.

Corresponding Author:

Name: Shubh Prabhat
Institution: Architect, GlobalLogic Inc., Cary, North Carolina, USA
Email: sprabhat.research@gmail.com

1. INTRODUCTION

Order, billing and inventory logic is executed within single deployable systems of large enterprises. The chapter in the survey of microservices enumerates six common monolith problems. They suffer hard maintenance, dependencies issues, lengthy restart times, deployment issues, limited scaling and technology lock in [1]. It is mentioned in the same chapter that the name microservices emerged in 2011 at a workshop on architecture.

Migration is not that easy. In interviews with 27 practitioners, a taxonomy of 20 anti-patterns was created. The 56 percent of the practitioners mentioned wrong service cuts, which were among the most detrimental

[2]. A faulty cut causes numerous transfers between services. The cost was measured in an industrial study of 16 engineers in 12 companies. The time spent on debugging increased to 48 hours to debug a fault that spanned over three services when the time spent on debugging a fault spanned a single service (approximately 9.5 hours) was used [3].

The gap is indicated by these findings. The cut methods look at structure, defining code coupling and data tables. They do not have the event flow of the business as cut rule. An event causes a service to publish a fact and proceed. A failed consumer will not fail the request, but will delay work.

There are three objectives of this paper. The first is to specify an event driven decomposition strategy. The second one is to test it with a synchronous version and a monolith. The third one would be to provide engineering managers with definite course of action by the results.

The paper attempts to validate three propositions. Cuts based on events increase the quality-of-service limits. Request routes are shortened between services through events. An outage is maintained within the failed service by events.

2. LITERATURE REVIEW

2.1 Migration Practice

Migration has been reported to be a gradual task. A single report took the move step by step to microservices of a commercial mobile back-end service. It discovered that the move was made smoother by the DevOps practice [4]. The other study suggests a process to divide the legacy applications. It contends that smaller code bases and clearer boundaries of components enhance maintainability [5]. An online retailer report indicates a functional division of business, with each team having a vertical cut of the platform [6]. Table 1 compares six other studies which measure the migration practice and cut methods.

Table 1. Side by side analysis of 6 credible published works on practice of migration and service cutting, with evidence provided and the gap left by each source to this paper

Study focus	Main contribution	Reported evidence	Gap for this paper	Source
Migration practice	Company survey of migration	Practitioner survey responses	No association between design of event and cut choice	[7]
Migration motives	Interviews on processes, motives and issues	Maintainability and scalability cited by 15 each with 21 practitioners	No resilience measures	[8]
Migration strategies	Intentions, strategies and challenges interview	16 interviews, 14 systems, can be broken down in 8	No direction in the event designing	[9]
Extraction by coupling	Logical coupling, semantic coupling and contributor coupling graph clustering	Tested on open-source projects	Ignores runtime events	[10]
Service cutting	List of 16 clustering of coupling criteria	Reductions in less than five seconds across 600 entities	Works with fixed artifacts	[11]
Trace based candidates	Execution traces with multi objective search	Six web applications, 28,770 traces cut to 3,139	Needs complete test coverage, no messaging	[12]

2.2 *Cut Methods*

Table 1 indicates that well structure and motives are measured in studies. The two interview studies concur on the motives. First in the lists are maintainability and scalability. The greatest hassle is caused by decomposition and information dissection. The coupling, catalogue and trace methods provide methods to suggest cuts to the teams but each one ends at the cut. Five out of six studies do not measure resilience, post-cut. This is the same with other methods. One develops a data flow model and combines activities into candidate services [13]. The other reads interface specification and clusters operations based on semantic similarity [14]. Both use a static information. Neither enquires on what a service should do in the event of a failing neighbour.

3. EVENT DRIVEN DECOMPOSITION STRATEGY

The plan consists of five stages. Domain mapping enumerates capabilities, calls and tables. Cut scoring ranks the candidate services. Event contract design substitutes instant calls with events where it is possible to the business. Strangler extraction relocates a capability sequentially, behind a facade. It checks recovery and fault injection. The process of flow in figure 1 is between domain mapping and verification. An unsuccessful verification will get the team back to cut scoring.

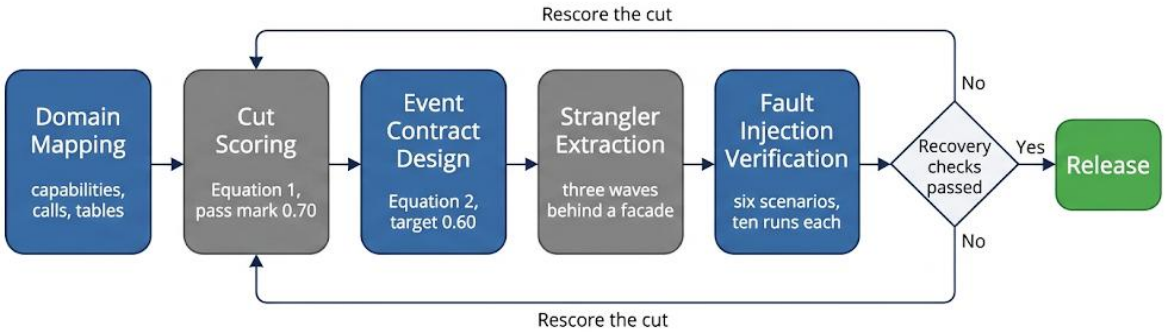


Figure 1. Five phase event-based deposition strategy, domain mapping to verification, with one way fed back to cut scoring

3.1 *Domain Mapping and Cut Scoring*

Business capabilities, calls and data tables are listed by the team. An effective cut brings related things. Equation (1) scores a candidate service k.

$$Q(k) = \frac{w_1 \cdot C_i + w_2 \cdot D_i}{w_1 \cdot (C_i + C_e) + w_2 \cdot (D_i + D_e)} \#(1)$$

In Equation (1), C_i is the number of calls that remain within service k, and C_e is number of calls that exit it. D_i counts accesses to table that remain in, and D_e counts accesses that exit. The two weights w1 and w2 are 0.6 and 0.4 due to the fact that call links are broken easily compared to data links. To pass, a service has to make 0.70. The order service has 148 internal calls, 22 external calls, 31 internal table accesses and 4 external ones. Its score is 0.87.

3.2 *Event Contracts and Conversion*

There are two types of cross service interactions. A command requires an answer currently, like an authorization of payment. A fact may be published such as an order that has been fulfilled. The plan converts all the facts into an occurrence on a broker and maintains commands in direct calls. This division guards the business regulations that require rigid timeline and permits all the rest to proceed without holding. The four rules used are listed in Table 2 and the pattern behind each one is the event.

Table 2. Four decomposition rules are employed in the strategy, whereby the basis of each rule, event pattern to which the rule will apply and the predominant risk to be handled.

Rule	Decomposition basis	Event pattern	Main risk
Business event cut	Splits all over events like order placed	Publish and subscribe on issues	So many trivial matters
Strangler extraction	Going through a facade one route at a time	The new service is fed by change capture	Dual writers
Outbox publishing	The events are written in their respective transactions in each service	Outbox table sent to the broker	Relay delay
Saga coordination	Long flows are broken down to local steps	Choreographed compensating events	Hard failure tracing

An example of banking demonstrates the trend. This does not have a direct interface in the sense that the messages that can be spoken on a broker are only the service of it, meaning there is no actual interface that can be broken by a neighbour [15]. The extent to which a design adheres to this concept is measured in equation (2).

$$A = \frac{N_e}{N_e + N_r} \#(2)$$

N_e in Equation (2) is the number of cross service interactions per order utilizing events. N_r is the number that are direct calls. A value close to 1 signifies the loose time coupling. The desired value in this experiment is 0.60.

3.3 *Target Architecture*

The target architecture is presented in figure 2. A facade routes requests, each service is an owner of data and a three-node broker is a carrier of events. Table 3 enumerates the eight services and events they have published and consumed.

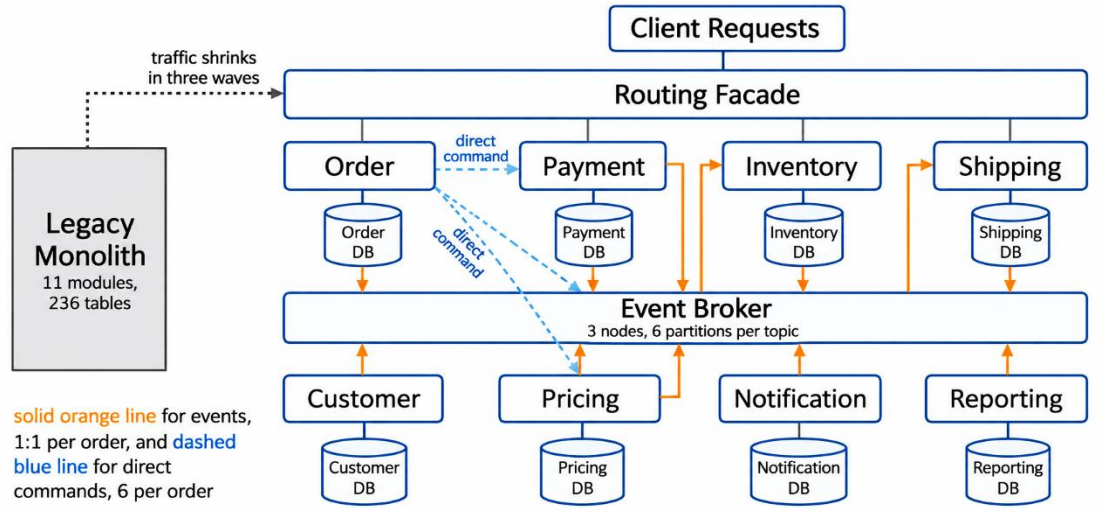


Figure 2. A routing facade, eight service owning their data store, and three node brokers are used as target event driven architecture

Table 3. Eight services generated by the cut, and each service has data it owns and events that it publishes or receives in the order flow

Service	Owned data	Events published	Events consumed
Order	Orders, order lines	Order Placed, Order Cancelled	Payment Settled, Stock Reserved
Payment	Payments	Payment Settled, Payment Failed	Order Placed

Service	Owned data	Events published	Events consumed
Inventory	Stock levels	Stock Reserved, Stock Low	Order Placed, Order Cancelled
Shipping	Shipments	Shipment Created	Payment Settled, Stock Reserved
Customer	Profiles	Customer Updated	None
Pricing	Price lists	Price Changed	None
Notification	Message log	None	Shipment Created, Payment Failed
Reporting	Read models	None	All order events

Events remain in the log and subsequent alterations to the structure of events require scheduled conversion of data. This cost is explained in a study of event sourcing [16]. The team thus versifies any event schema.

3.4 Consistency and Recovery

Isolated data warehouses pose a recovery issue. A backup, availability and consistency theorem describes how making backup services at various times leaves some broken links, orphan state or missing state following a restore. The plan uses snapshots of databases along with broker offsets in order to prevent this.

4. EXPERIMENTAL SETUP

The simulated order management platform was used in the experiment. The monolith contained 1.4 million Java code, 236 tables and 11 modules in the database. Four loads of 500, 1,000, 1,500 and 2,000 orders in one minute were driven by replayed order traces. The various loads were done in three consecutive times of 30 minutes each. There were 12 virtual machines in the test cluster. Each topic had three nodes and six partitions with the broker.

There were three versions that were constructed. The foundation was the monolith. The synchronous version divided the platform into the pair of identical eight services but called upon the direct calls exclusively. The strategy was followed by event driven version. Services were extracted in three waves by the team. The availability of each service was measured in a 21 day soak run at 1,000 orders/minute. Each service reached 0.995.

Faults were added between services on the message layer. The concept is based on a tool that tries the recovery patterns by swapping traffic between microservices [17]. Figure 3 shows the test pipeline. There are six fault scenarios listed in Table 4. All scenarios were performed 10 times in all versions of the scenario.

13 measures were noted in the study. They include quality of design, speed of request, capacity, availability, recovery, data consistency, boundary variations, cost and effort. Health checks every ten seconds served as the source of availability. Inconsistency incidents were the orders whose payment, stock or shipping records were conflicting with each other following a run. Effort came from the team time sheets.

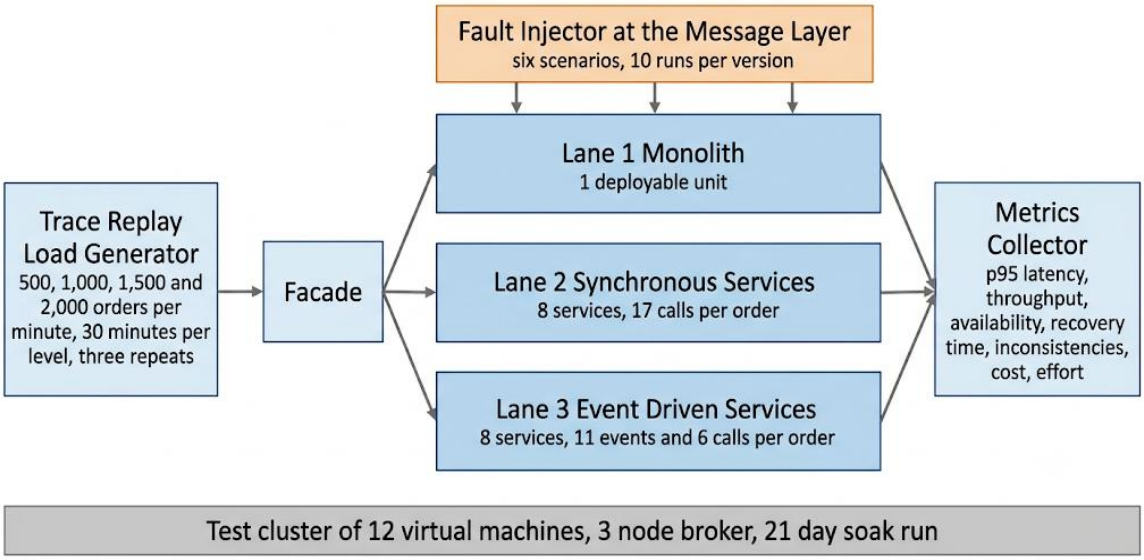


Figure 3. Load trace replay pipeline at load point, the tested three versions, the fault injector, and the metrics collector of the twelve-machine cluster.

Table 4. Each version was injected with six fault scenarios with the injection method, the resilience pattern being tested and the expected behavior of the event driven version.

Scenario	Injection method	Pattern under test	Expected behavior
Payment service down	Take a break every 5 minutes	Retrying and buffering of brokers	Accepting orders, to be paid later
Slow inventory	Include delay of 800 ms to responses	Stop and circuit breaker	Order path remains fast
Broker node loss	Shut down one among three nodes	Replication	No event lost
Duplicate delivery	Replay 5 percent of events	Idempotent consumers	No double charge
Network partition	Reducing a single link down by 3 minutes	Outbox and retry	Repair is followed by events
Consumer crash loop	Start (consumer) shipping after 30 sec	Offset commit and dead letter queue	No message lost

5. RESULTS

Table 5 reports the main measures. Most of them were scored best by the event driven version.

Table 5. Comparative results of the monolith, the synchronous version and the event driven version in terms of latency, resilience, consistency, cost and effort

Measure	Monolith	Synchronous services	Event driven services
Mean cut score Q	Not applicable	0.69	0.81
Synchronous calls per order	0	17	6
Events per order	0	0	11
Median latency at 1,000 orders per minute, ms	182	268	151
p95 latency at 1,000 orders per minute, ms	410	655	360
Peak throughput, orders per minute	2,400	3,100	4,300

Measure	Monolith	Synchronous services	Event driven services
Measured availability, percent	98.5	96.8	98.9
Orders completed in one service outage, percent	62	47	91
Mean recovery time, minutes	38	22	9
Inconsistency incidents per 10,000 orders	0.3	3.9	1.6
Boundaries changed after release	Not applicable	3 of 8	1 of 8
Infrastructure cost index	100	138	126
Migration effort, person months	Not applicable	31	38

The event driven version kept 11 events and 6 calls per order. Equation (2) then gives A of 0.65, above the 0.60 target. The plot of p95 latency versus load is shown in figure 4. At 500 orders per minute the three versions gave 290, 380 and 250 ms. At 2,000 orders per minute, they gave 1,650, 2,100 and 790 ms. Less than half of the peak capacity at that load was used by the event driven version.

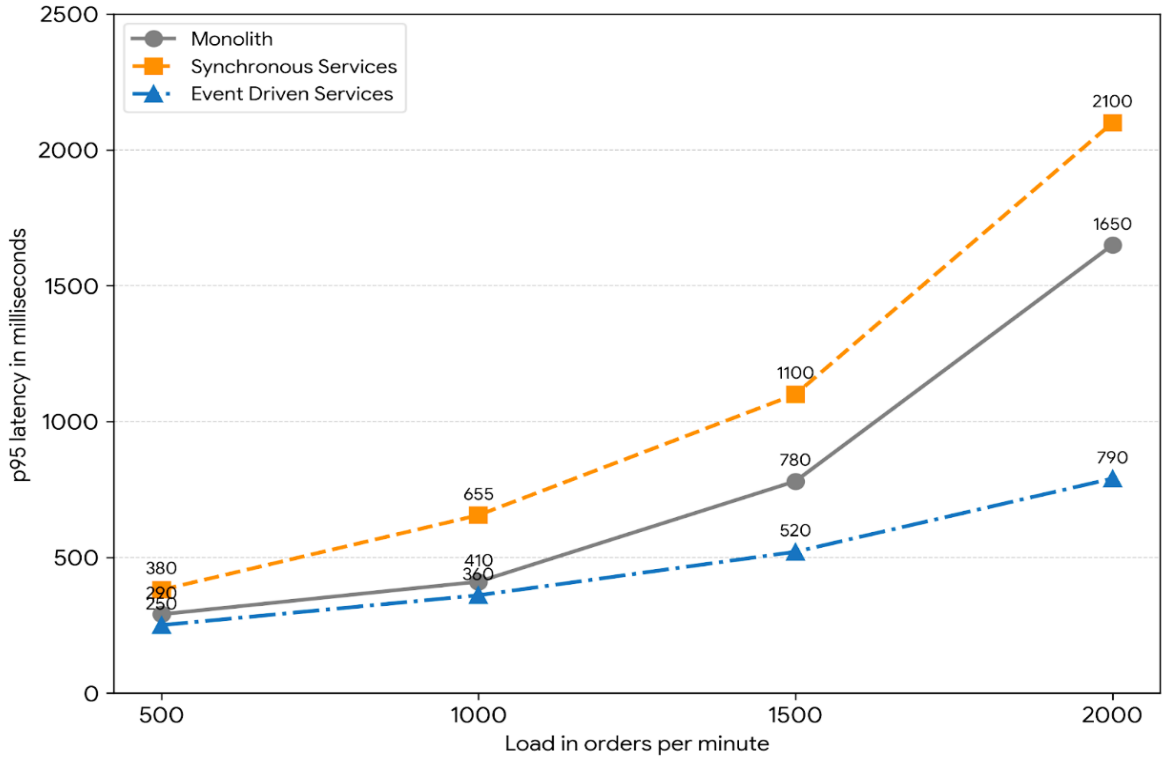


Figure 4. Monolith, synchronous and event driven version latency at 500, 1,000, 1,500 and 2,000 order per minute simulated by creating a p95 curve that depicts the p95 position as a function of time

The availability of a request path is modeled by equation (3).

$$A_{path} = a_1 \times a_2 \times \ldots \times a_n \#(3)$$

The availability of the request requirements in the form of the n components is represented by a_n in Equation (3). The synchronous line requires six 0.995 services. The prediction of the model is 0.970 and the soak run is 0.968. The facade, order service and the broker require the event driven path to value 0.995, 0.995 and 0.9995. The model forecasts 0.9895 and the run was 0.989.

The outage runs are based on the logic of chaos engineering that provokes a live system containing real faults to develop confidence in its behaviour [18]. In the event driven version, 91 percent of orders were completed in the payment outage. The rest were retained by the broker, who after repair was discharged. The synchronous version was 47 per cent complete. The recovery time

was 9 mins versus 22. Table 4 provided the same results with the other scenarios. In the loss runs of the broker node, there were no lost events since each topic had three replicas. When delivering a duplicate there was a charged order that was delivered twice. The circuit breaker was opened by slow inventory runs, thus keeping the order path straight.
New state after the event lag is observed in Equation (4) by consumers.

$$L = T_{pub} + T_{brk} + T_{con} \#(4)$$

T_{pub} is the time of publication, T_{brk} is the time of the broker and T_{con} is the time of consume in Equation (4). The p95 values were 14, 27 and 40 ms. This delay of 81 ms can be tolerated by reporting and notification.

6. DISCUSSION

Three things are backed by the results. First, an increased cut score facilitated. The average score increased to 0.81 as compared to 0.69 and wrong boundaries decreased to one of eight as compared to three of eight. This is in line with the practitioner opinion that wrong cuts are the most prevalent migration issue. Second, events reduced the request paths. The event driven version employed 6 calls per order with the synchronous version employing 17. Third, there were events which restricted the spreading of faults. Outages remained local due to the buffering of the work done by a broker.

Events do not suit all engagements. The remaining six calls that came through the event driven version were commands e.g. payment authorization and stock checks. They require a solution immediately. A design that would compel them to happen would conceal failures and disorient users. The conversion ratio of 0.65 is thus reasonable and not an aspiration to push to 1.

The cost of event driven version is there. It required 38 person months as compared to 31, an increase of 23 percent. Event contracts and schema versions took the extra time. Infrastructure cost remained 9 per cent less than the synchronous version since the broker eliminated the necessity to have additional capacity on each call path. In a review of the microservice journey, the evolution of the style over time and the challenges that lie ahead can be described [19]. Table 6 transforms the results into actions.

Table 6. Designs that have been informed by the experiment, including what supports an action and the impact that is likely to occur in a migration program

Finding	Engineering action	Expected effect
Q rose from 0.69 to 0.81	Categorize a score that is less than 0.70 as rejected	Fewer wrong cuts
A reached 0.65	Transform every piece of published fact into an incidence	Shorter request paths
The completion of outages became 91 percent, as opposed to 47	Buffering within the broker	Local outages
The number of inconsistencies dropped to 1.6 per 10,000	Utilize outbox and idempotent consumers	Safer recovery
The increase in effort was 31-38 person months	Budget event contract time	Realistic plans

There are limitations of the study. It put one artificial platform and one broker. The weightings of the equation (1) were through the judgment and calibration of weights on field data. The lag metric was within a single cluster.

7. CONCLUSION

This paper has suggested an event-based approach to the process of migrating enterprise monoliths to micro services. The strategy scores cuts, transforms facts to events, and extracts services by wave and

validates resilience with injected failures. It performed better in the experiment with lower latency, less time to recover, fewer inconsistencies and fewer false boundaries than a synchronous split. It required additional efforts as well. Managers are supposed to score charges prematurely; plan events preceding code and maintain commands as immediate invocations. They should also make arrangements of the additional effort that event contracts require. The strategy should be experimented on a live program in future work and the weights calibrated with field data.

REFERENCES

[1] N. Dragoni *et al.*, "Microservices: Yesterday, Today, and Tomorrow," in *Present and Ulterior Software Engineering*, M. Mazzara and B. Meyer, Eds., Cham: Springer International Publishing, 2017, pp. 195–216. doi: 10.1007/978-3-319-67425-4_12.

[2] D. Taibi, V. Lenarduzzi, and C. Pahl, "Microservices Anti-patterns: A Taxonomy," in *Microservices: Science and Engineering*, A. Bucchiarone, N. Dragoni, S. Dustdar, P. Lago, M. Mazzara, V. Rivera, and A. Sadovykh, Eds., Cham: Springer International Publishing, 2020, pp. 111–128. doi: 10.1007/978-3-030-31646-4_5.

[3] X. Zhou *et al.*, "Fault Analysis and Debugging of Microservice Systems: Industrial Survey, Benchmark System, and Empirical Study," *IEEE Trans. Softw. Eng.*, vol. 47, no. 2, pp. 243–260, 2021, doi: 10.1109/TSE.2018.2887384.

[4] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture," *IEEE Softw.*, vol. 33, no. 3, pp. 42–52, 2016, doi: 10.1109/MS.2016.64.

[5] H. Knoche and W. Hasselbring, "Using Microservices for Legacy Software Modernization," *IEEE Softw.*, vol. 35, no. 3, pp. 44–49, 2018, doi: 10.1109/MS.2018.2141035.

[6] W. Hasselbring and G. Steinacker, "Microservice Architectures for Scalability, Agility and Reliability in E-Commerce," in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, 2017, pp. 243–246. doi: 10.1109/ICSAW.2017.11.

[7] P. Di Francesco, P. Lago, and I. Malavolta, "Migrating Towards Microservice Architectures: An Industrial Survey," in *2018 IEEE International Conference on Software Architecture (ICSA)*, 2018, pp. 29–2909. doi: 10.1109/ICSA.2018.00012.

[8] D. Taibi, V. Lenarduzzi, and C. Pahl, "Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation," *IEEE Cloud Comput.*, vol. 4, no. 5, pp. 22–32, 2017, doi: 10.1109/MCC.2017.4250931.

[9] J. Fritzsch, J. Bogner, S. Wagner, and A. Zimmermann, "Microservices migration in industry: Intentions, strategies, and challenges," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, 2019, pp. 481–490. doi: 10.1109/ICSME.2019.00081.

[10] G. Mazlami, J. Cito, and P. Leitner, "Extraction of Microservices from Monolithic Software Architectures," in *2017 IEEE International Conference on Web Services (ICWS)*, 2017, pp. 524–531. doi: 10.1109/ICWS.2017.61.

[11] M. Gysel, L. Kölbener, W. Giersche, and O. Zimmermann, "Service Cutter: A Systematic Approach to Service Decomposition," in *Service-Oriented and Cloud Computing*, M. Aiello, E. B. Johnsen, S. Dustdar, and I. Georgievski, Eds., Cham: Springer International Publishing, 2016, pp. 185–200. doi: https://doi.org/10.1007/978-3-319-44482-6_12.

[12] W. Jin, T. Liu, Y. Cai, R. Kazman, R. Mo, and Q. Zheng, "Service Candidate Identification from Monolithic Systems Based on Execution Traces," *IEEE Trans. Softw. Eng.*, vol. 47, no. 5, pp. 987–1007, 2021, doi: 10.1109/TSE.2019.2910531.

[13] R. Chen, S. Li, and Z. Li, "From Monolith to Microservices: A Dataflow-Driven Approach," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, 2017, pp. 466–475. doi: 10.1109/APSEC.2017.53.

[14] L. Baresi, M. Garriga, and A. De Renzis, "Microservices Identification Through Interface Analysis," in *Service-Oriented and Cloud Computing*, F. De Paoli, S. Schulte, and E. Broch Johnsen, Eds., Cham: Springer International Publishing, 2017, pp. 19–33. doi: https://doi.org/10.1007/978-3-319-67262-5_2.

[15] A. Bucchiarone, N. Dragoni, S. Dustdar, S. T. Larsen, and M. Mazzara, "From Monolithic to Microservices: An Experience Report from the Banking Domain," *IEEE Softw.*, vol. 35, no. 3, pp. 50–55, 2018, doi: 10.1109/MS.2018.2141026.

[16] M. Overeem, M. Spoor, and S. Jansen, "The dark side of event sourcing: Managing data conversion," in *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2017, pp. 193–204. doi: 10.1109/SANER.2017.7884621.

[17] V. Heorhiadi, S. Rajagopalan, H. Jamjoom, M. K. Reiter, and V. Sekar, "Gremlin: Systematic Resilience Testing of Microservices," in *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, 2016, pp. 57–66. doi: 10.1109/ICDCS.2016.11.

[18] A. Basiri *et al.*, "Chaos Engineering," *IEEE Softw.*, vol. 33, no. 3, pp. 35–41, 2016, doi: 10.1109/MS.2016.60.

[19] P. Jamshidi, C. Pahl, N. C. Mendonça, J. Lewis, and S. Tilkov, "Microservices: The Journey So Far and Challenges Ahead," *IEEE Softw.*, vol. 35, no. 3, pp. 24–35, 2018, doi: 10.1109/MS.2018.2141039.

BIOGRAPHIES OF AUTHORS



Shubh Prabhat — Researcher Biography

Shubh Prabhat is a Software Engineering Leader, Solution Architect, and Independent Researcher with over 18 years of experience in distributed systems, cloud computing, cybersecurity, and AI-driven enterprise solutions.

He earned his **Bachelor of Technology (B.Tech.) in Electronics and Communication Engineering in 2008**.

He is the Creator and Maintainer of **AeglysAI**, an open-source research initiative focused on AI-driven security automation, intelligent anomaly detection, Zero Trust architectures, and resilient distributed systems.

His research interests include Artificial Intelligence, Machine Learning, Cloud Security, Distributed Systems, Identity and Access Management, and Cybersecurity.

Research Profiles:

- **ORCID:** https://orcid.org/0009-0008-1716-8789
- **Google Scholar:** https://scholar.google.com/citations?user=HrEzOfIAAAAJ
- **Email:** sprabhat.research@gmail.com