Educational Administration: Theory and Practice
2021, 27(3) 1178-1183
ISSN: 2148-2403
https://kuey.net/

Research Article

Scalability Patterns for Microservices Architecture

Dileep Kumar Pandiya*

*Software Engineer, Wayfair Inc

Citation: Dileep Kumar Pandiya (2021), Scalability Patterns for Microservices Architecture ,*Educational Administration: Theory and Practice*, *27(3)* 1178-1183, Doi: 10.53555/kuey.v27i3.6897

ARTICLE INFO

ABSTRACT

In software engineering, scalability is a key pillar when it comes to microservice architecture, which is a big goal in large-scale complex systems. An important part to know is the scalability patterns of microservices, which are well-rounded and optimized for performance under different demand peaks, creating the significance of the e-learning platform for both architects and developers alike. With technology evolving fast, systems grow in complexity, and scalable architectures are crucial. Microservices architecture, which is made of modular and distributed parts, has become the preferred choice for building large-scale programs today. Nevertheless, scaling up within the barrier of this methodology has prerequisites of special models and tactics to be used along the process. This article helps uncover the scalability patterns, starting with horizontal scaling as the first pattern. Horizontal scaling, as seen at the bottom, of duties in many instances, is still mentioned as one of the main bases of scalability in microservices. Using comprehensive explanations on load balancing, service instance autoscaling and database sharding in this essay, the practicality, real-world examples and intricacies of horizontal scaling are elaborated, giving an in-depth understanding of what it entails. Another mechanism that is critically analyzed is vertical scaling, which is also related to scalability. Also, this might seem like 100% contradiction to the advantages of horizontal scaling but different in nature, vertical scaling may offer a viable solution to the scalability challenges. Temporal database scaling and service scaling will be explained through details around when and how they are applicable, as well as when they are limited in a microservice environment. Another important issue in the essay deals with elasticity: the blog discusses on-the-fly provisioning as a possible tool for tackling demand peaks smoothly and spotlights serverless computing too. By the application of the theoretical concepts and the practical applications, readers are able to consolidate all the information, helping them to perceive the issue from a comprehensive point of view, besides which case studies, contextual cases, add a practical dimension to the discussed patterns.

Keywords: Microservices architecture, Scalability patterns, Horizontal scaling, Vertical scaling, Elasticity patterns

INTRODUCTION

The term scalability in the microservices architecture context refers to the system's capability of handling more and more workloads with increasing demands without imparting any performance or stability degradation. In Contrary to monolithic architectures, where scaling is usually achieved by merely replicating the entire application stack, microservices present a more flexible approach by allowing individual services to be scaled up or down as required. The fact is that it doesn't mean that implementing scalability in microservices isn't also without obstacles. The scalability of microservices is impeded by the fact that they are distributed architectures that necessitate some unique technologies. All microservices run by themselves while connecting to other services through APIs and storing their own information in their private databases. Consequently, whereas a strong central authority is holding the scale of pervasively connected services while preserving the system, coherence appears to be a challenge. The scalability equation is further complicated by elements like traffic patterns, inter-service dependencies, and different resource requirements [2]. Although scalability is noted as a merely technical aspect, the business side of scalability should not be overlooked. The digital world has today become the fastest and most fleeting environment; therefore, companies need to respond quickly to the

dynamic market conditions and consumers' needs. Scalability makes it possible for an organization to manage firespecific and rapid increase in traffic users by expanding data sets and introducing live features and services without interruption. Insufficient scaling based on a certain threshold in production content delivery may end up leading to impaired performance, downtime, revenue loss, and a decline in customer happiness and loyalty in the end. While scalability is one of the most prominent requirements in microservices architecture, patterns of scalability are the strategic frameworks for addressing the challenges attached to the scaling of microservices. Such patterns incorporate efficient organizational models, design concepts, and architecture principles for scalable microservices based system evolution. Organizations may confidently manage the challenges of scaling by implementing scalability patterns. These patterns leverage best practices and established principles to maximize resource utilization, minimize bottlenecks, and guarantee the durability and robustness of systems [3]. Scalability basics are the foundation that scalability in microservices architecture could eventually be built upon, leading to overall business success and sustainable growth. Generally, organizations may fully realize the benefits of microservices architecture and become competitive players in the ever-changing field of modern software engineering, by comprehending the difficulties associated with scaling microservices and using scalability patterns as guiding principles.

HORIZONTAL SCALING PATTERNS

Horizontal scaling, termed scale-out scaling, is one of the main tendencies that can be used to increase the scalability of the microservices architecture. Unlike vertical scaling, which is about upgrading the capability of individual components within a system, horizontal scaling is all about the multiplication of more instances of nodes to distribute the workload among multiple resources. This way, horizontal scaling suggests some benefits, including improved automation from faults, better resource utilization and increased load handling with better efficiency. Nonetheless, horizontal scaling is a crucial element in microservices architecture, and is differently affected by the factors that affect the detached services being distributed [3]. All microservices are responsible for Maintaining business logic individually and can be distributed to particular instances. This leads to an improvement in information density ancreases the scalability without the need for every single resource scaling strategy. Therefore, in this case, vertical scaling is often emphasized to ensure the distribution of the traffic is uniform across all the instances of microservices. At the same time, load balancers address this issue by playing the role of a mediator that selectively directs client requests onto the least loaded node in the cluster so that overall performance and resource utilization are efficiently optimized [3].

By way of illustration, given a popular e-commerce platform that experienced an increase in user transactions during a Christmas sale, the next example is as follows. The checkout microservice can be handed over to several instances for deployment using a load balancer. This provides the application with the capability to make the highest use of this load, defend the application performance and, at the same time, deliver a smooth online shopping experience for users. Furthermore, service instance autoscaling is another critical horizontal scaling technique that allows organizations to flexibly adjust the number of service instances depending on workload variation [4]. The autoscaling capabilities constantly evaluate critical metrics, including CPU usage, memory utilization, and request latency, and hence can extend or decrease instances to satisfy the set performance targets without any user manual intervention [5]. For instance, microservices dealing with tasks that run in the background will have high demands virtually throughout the day. The effectiveness of an organization can be transformed by using auto-scaling policies that lead to scaling additional instances during peak hours and scaling down during the off-peak period; this helps to reduce the operation costs without affecting the job timeliness. Also, database sharding is a pattern of scaling that targets, precisely, the bottlenecks of data storage, in a microservices architecture [6].

Modern monolithic databases find it difficult to cope with the rise in volume and frequency of the data produced by microservices-based applications as a result. Sharding is a type of partitioning that is done horizontally on multiple instances (databases) or shards according to a defined criterion such as user ID or geographical locale [7]. Celery architecture provides a mechanism of multiplexing where every shard acts independently, which then allows to finish queries in parallel or to increase read/write throughput. We can take an example of, say, a social media platform that may fragment its user data about different geographical regions, so localized access can be realized and latency can be overcome for users across different locations. Consequently, horizontal patterns of scaling are the significant evolving factors that promote the usability and performance of microservice architecture [3]. Load balancing, service instance autoscaling and database sharding can be strategically adopted by businesses to conduct workload distribution, improve the process of resource utilization and enable the resilience of their systems that are microservices based. As the demand for scalable and reliable software solutions increases, there's an urgent need in this domain to develop horizontally scalable solutions.

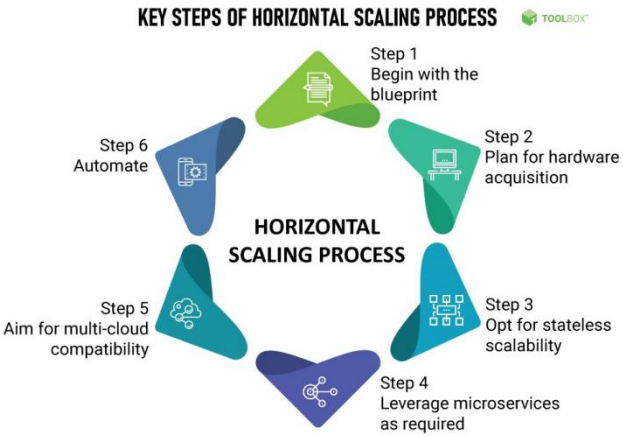

Figure 1. Horizontal Scaling Process [3]

As the picture portrays, planning carefully is essential for efficiently increasing cloud infrastructure, which includes assessing historical trends and projecting needs with stakeholders. In partnership with IT, purchase hardware while keeping compatibility and cost in mind. Use microservices to improve resource allocation, go for stateless apps for smooth server scaling, and make sure your application is compatible with several clouds for flexibility. In order to effectively manage demand surges, automate scaling procedures. Additionally, regularly monitor and enhance infrastructure performance over time.

VERTICAL SCALLING PATTERNS

Vertical scaling, commonly known as scale-out scaling, can be an alternative scaling method to address microservice architecture scalability and high performance problems. As mentioned earlier, vertical scaling, improves the capacity of the individual components within the system. This introduces enhancements to the hardware resources (e.g., CPU and memory units) of a single server or the microservice instance to process high loads [3]. One predominant scaling pattern is vertical database, which consists of adding the resources (CPU, RAM, and storage) of a single database instance in order to improve its performance and capacity. The most striking aspect of using this pattern is that a system's organization has a single, centralized database for different microservices. Using vertical scalability, organizations can simply increase the database instance dimensions to serve growing data volumes and query traffic volumes in complex distributed databases that do not require sharding [3]. In order to manage the growing volume of transactions and product data, an e-commerce platform that is expanding quickly may vertically scale its database server, guaranteeing smooth performance and scalability. Besides, vertical service scaling, which entails enhancing each microservice's hardware or capability to handle growing workload needs [8],. This technique of running microservices may apply to tasks that deplete resources or perform very intensive work requiring computational power. Through a vertical scaling of specific microservice reporting issues, organizations will find technology not vulnerable to possible performance bottlenecks and will continually stay responsive regardless of the number of instances to manage. For example, the machine learning microservice that is responsible for image processing may need additional CPU cores and GPU resources just to manage the image uploading during the busy hours. This approach, however, allows for low latency even when processing images and means that the user will obtain real-time results. Organizations should have a number of considerations to make when deciding whether to use vertical scaling or horizontal scaling, including cost, complexity, and performance needs. The main factors to consider while deciding between vertical and horizontal scaling are listed in Comparison Table 1 below:

Table 1. Comparison between Vertical and Horizontal Scaling.

Criteria	Vertical Scaling	Horizontal Scaling
Scalability	Limited scalability due to hardware constraints	Highly scalable, can handle unlimited growth
Complexity	Relatively simple, involves upgrading hardware	More complex, requires managing multiple instances
Cost	Can be expensive, as hardware upgrades are costly	Generally more cost-effective, utilizes commodity hardware

Performance	Suitable for resource-intensive applications	Optimal for distributed workloads and parallel processing
Flexibility	Limited flexibility, scaling is constrained by hardware limitations	Offers greater flexibility, can scale individual components independently
Fault Tolerance	Weak fault tolerance means that the system as a whole may be affected by a single point of failure.	Enhanced fault tolerance, failures in individual instances have minimal impact

In cases where speed and fault tolerance are critical to some applications, large amounts of horizontal scaling are often chosen over vertical scaling. With the help of horizontal scaling (adding more computing instances), for instance, organizations can distribute the work load and, thereby, ensure overall resilience and elastic performance even under a varying load. Apart from that, horizontal scaling provides additional flexibility, which means that an organization can scale some elements of its infrastructure independently, each at its own pace based on the resource requirements for every unique component. Horizontal scaling is a cost-effective strategy in situations where individual applications need limited additional power, while vertical scaling may be preferred in circumstances where the stringent performance requirements of a system or the cost of hardware upgrades call for simplicity of management. Vertical scaling is something that will come in very handy for those applications that have been built with monolithic architectures or those that need centralized resources. The vertical and horizontal deployment approaches are considered key patterns mobilized to achieve scalability and enhance the performance of the new distributed architecture model of microservices. This knowledge of the pros and cons of each method, coupled with a judicious assessment of their appropriateness to particular use situations, offers the necessary inputs for the design of solid systems that are scalable, cost-effective, and meet the evolving demands of modern software environments.

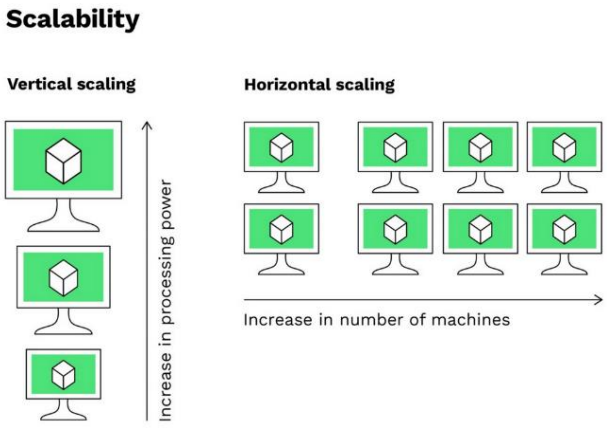

Figure 2. Vertical and Horizontal scaling [3].

From the figure above, Adding more nodes constitutes horizontal scaling, while boosting the power of the current machines is referred to as vertical scaling. If the server required more processing power, for example, vertical scaling would entail upgrading the CPUs. Additionally, a company can increase memory, storage, or network speed while utilizing vertical scaling.

ELASTICITY PATTERNS FOR MICROSERVICE ARCHITECTURE

Elasticity in microservice architecture plays a crucial role in scalability, ensuring that modern software systems are resilient, scalable and cost-effective. Elasticity is the system's capability to automatically adjust the number of allocated resources in relation to changing demand [9]. As a result of this, the system can scale up or down when the workload needs it while still providing high performance. Dynamic provisioning, which is one of the major elasticity patterns, is capable of automatically increasing or decreasing the machine instances according to load variation in runtime [10]. This method suits so well as it is based on the cloud computing platform as the main area where the infrastructure is provided and infrastructure-as-code tools that can essentially create and remove instances of virtual machines, containers or other resources at the same time. Take, for instance, a case where an e-commerce application experiences a boom in user traffic during a flash sale event that is characterized by limited time offer. Dynamic provisioning takes place when the application is able to establish the increasing activity of the microservices by itself. Moreover, this approach ensures that an appropriate

number of instances of the microservices are running to handle the increasing load. On the contrary, when the flow of traffic and excess resources are freed up, the resources are automatically decommissioned to keep costs down. In addition to containers, serverless computing is another paradigm of cloud elasticity that has met with a lot of success among cloud consumers [9]. Serverless platforms, like AWS Lambda and Azure Functions, enable organizations to execute event-driven functions on-demand, responding to certain triggers or events, without the need to manage the underlying infrastructure.

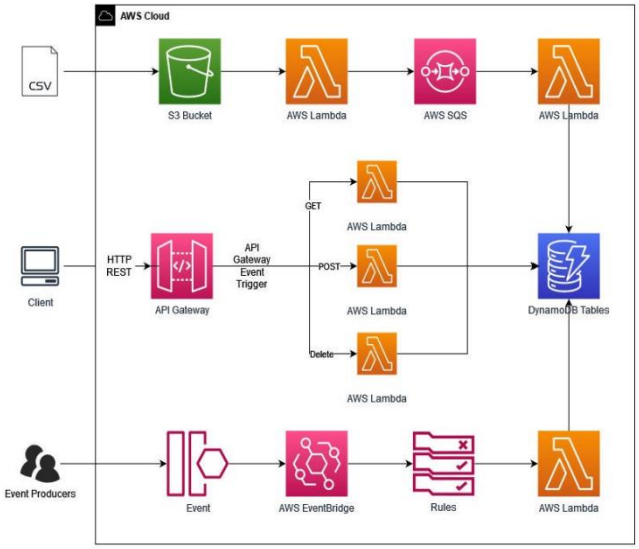

Figure 3. AWS Serverless Event-driven architecture

As portrayed in the Figure 3, AWS Lambda functions can be used as the event-driven computing service in a serverless event-driven architecture. Code that executes in response to an event, like an HTTP request, a database change, or a message posted to an Amazon Simple Queue Service (SQS) queue, is known as a lambda function. Lambda functions can be used for a wide range of tasks, including real-time data processing, data transformation and validation, and processing of images and videos.

In contrast to the usual static and scalable model, where extra resources are added to process flows, under a serverless architecture, the right resources are to be created and billed according to actual workload This gives an unparalleled high level of scalability. Let's take the example of a social media platform that utilizes serverless functions to facilitate uploads by its users. When the platform is under heavy user loads, the serverless functions system comes into play by scaling up automatically with inbound requests, thus ensuring smooth growth that does not require any human interference. Even though serverless computing and dynamic provisioning have many advantages, each elasticity pattern has unique difficulties that businesses must overcome. These obstacles include:

Dynamic Provisioning:

Complexity: An automated and orchestrated system that takes workload patterns and resource requirements into account is a must for the implementation of agile resource provision. Managing the operational complexity of continuously changing and large infrastructure could result in additional complexity and operational hurdles.

Cost Management: Although dynamic provisioning offers flexibility and responsiveness, the fact is that resource usage has to be monitored carefully in order to avoid unnecessary surplus spending. Balance and governance are crucial for optimized resource provisioning, as otherwise, budget overruns and inefficiencies can occur.

Performance Overhead: The act of automatic scaling and de-scaling resources will involve delays and expenses, so systems may suffer performance issues, particularly during scaling up. Organizations need to find a compromise between dynamic provision workflows and low latency in order to provide a responsive environment.

Serverless Computing:

Cold Start Latency: Every serverless function incurs a temperature latency to boot up when it is invoked for the first time or after a period of idleness because the substrate must allocate resources for the request [13]. This latency is something that may affect the speed of response of the application, particularly for those sensitive apps that have a time deadline.

Vendor Lock-in: A migration to a serverless platform introduces this risk of vendor lock-in since organisations may have to stay with their preferred cloud provider to ensure flexibility and portability [14].

Vendor lock-in risks have to be duly considered, and strategies to avoid dependency on proprietary serverless frameworks have to be developed.
Resource Limits: Serverless platforms have resource restrictions planned into them in terms of run time and memory provisioning, which might restrict the expansion of applications [15]. Organizations should take into account criteria such as memory limitations and efficient resource management in the design of their serverless functions and utilize these practices to improve their performance and cost.

CONCLUSION

In this article, several scaling patterns specifically for microservices architecture have been presented: horizontal and vertical scaling, as well as elasticity models. It had dwelt on scalability as an issue that contributes a lot to the reliability, performance and cost savings of modern software systems. Pick up and customize scalability patterns suitably because scalability modes directly effect system capacity to handle many workloads and dynamic requirements. The ability to grasp the subtleties of both horizontal and vertical scaling, as well as detailed elasticity patterns, means that firms can create a robust and scalable architecture that is designed to handle dynamic needs better. Therefore, this work is a good foundation for practitioners to expand their study to analyze specific scalability patterns, which are contextual and thus vary according to use cases, and advanced techniques for scalability, including microservice orchestration, containerization, and cloud-native architectures. Also, continuous monitoring and evaluation of scalability strategies remain on the priority list to overcome the pitfalls and help with improved resource allocation throughout the period. In summary, the adoption of developer-friendly design principles and the application of applicable scalability patterns allow businesses to construct robust and ever-developing microservice architectures that supply the latest technologies and business development.

REFERENCES

1. D. Lu, D. Huang, A. Walenstein, and D. Medhi, “A Secure Microservice Framework for IoT,” 2017 IEEE Symposium on Service-Oriented System Engineering (SOSE), Apr. 2017, doi: https://doi.org/10.1109/sose.2017.27.
2. Chander Dhall, Scalability Patterns. 2018. doi: https://doi.org/10.1007/978-1-4842-1073-4.
3. C. Qu, R. N. Calheiros, and R. Buyya, “Auto-Scaling Web Applications in Clouds,” ACM Computing Surveys, vol. 51, no. 4, pp. 1–33, Jul. 2018, doi: https://doi.org/10.1145/3148149.
4. J. H. Novak, S. K. Kasera, and R. Stutsman, “Cloud Functions for Fast and Robust Resource Auto-Scaling,” 2019 11th International Conference on Communication Systems & Networks (COMSNETS), Jan. 2019, doi: https://doi.org/10.1109/comsnets.2019.8711058.
5. Binildas Christudas, “Advanced High Availability and Scalability,” Apress eBooks, Jan. 2019, doi: https://doi.org/10.1007/978-1-4842-4501-9_16.
6. D. Taibi and K. Systä, “From Monolithic Systems to Microservices: A Decomposition Framework based on Process Mining,” Proceedings of the 9th International Conference on Cloud Computing and Services Science, 2019, doi: https://doi.org/10.5220/0007755901530164.
7. U. Gias, G. Casale, and M. Woodside, “ATOM: Model-Driven Autoscaling for Microservices,” Spiral (Imperial College London), Jul. 2019, doi: https://doi.org/10.1109/icdcs.2019.00197.
8. F. Klinaku, M. Frank, and S. Becker, “CAUS: An Elasticity Controller for a Containerized Microservice,” Apr. 2018, doi: https://doi.org/10.1145/3185768.3186296.
9. Rodrigo, Vinicius Facco Rodrigues, G. Rostirolla, Cristiano, E. Roloff, and Philippe, “A lightweight plug-and-play elasticity service for self-organizing resource provisioning on parallel applications,” Future Generation Computer Systems, vol. 78, pp. 176–190, Jan. 2018, doi: https://doi.org/10.1016/j.future.2017.02.023.
10. Pérez, S. Risco, D. M. Naranjo, M. Caballer, and G. Moltó, “On-Premises Serverless Computing for Event-Driven Data Processing Applications,” IEEE Xplore, Jul. 01, 2019. https://ieeexplore.ieee.org/abstract/document/881451
11. S. Kolb, On the Portability of Applications in Platform as a Service. University of Bamberg Press, 2019.
12. E. Jonas et al., “Cloud Programming Simplified: A Berkeley View on Serverless Computing,” arXiv (Cornell University), Jan. 2019, doi: https://doi.org/10.48550/arxiv.1902.03383.