International Journal of Computer Engineering and Technology (IJCET)
Volume 15, Issue 3, May-June 2024, pp. 182-192, Article ID: IJCET_15_03_017
Available online at https://iaeme.com/Home/issue/IJCET?Volume=15&Issue=3
ISSN Print: 0976-6367 and ISSN Online: 0976-6375
Impact Factor (2024): 18.59 (Based on Google Scholar Citation)

ARCHITECTING FOR SCALE: LESSONS LEARNED FROM SUPPORTING MILLIONS OF USERS

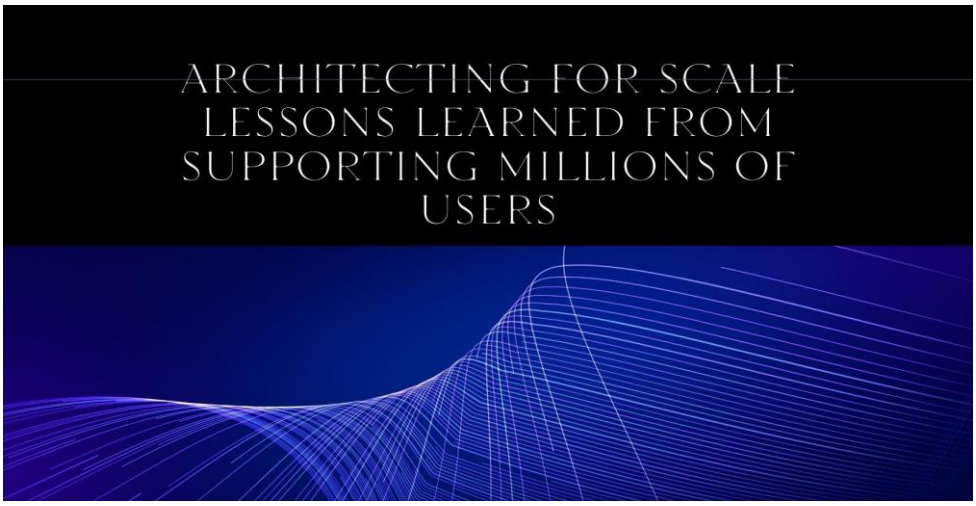
Ramneet Bhatia
Netflix, USA

ABSTRACT:

When you build systems that can handle millions of people, you need to make sure that you architect for scale. As digital platforms and services grow quickly, it's important to use scalable architectures that can handle more users and keep speed at its best. This article talks about the most important things that can be learned from designing systems to work on a large scale. These include using cloud services, adopting distributed architectures, performance optimization techniques, resilience and fault tolerance strategies, and embracing automation and DevOps practices. By looking at real-life examples, best practices, and related studies, this article gives organizations that want to build scalable and resilient systems useful information and suggestions.

Keywords: Scalable Architectures, Performance Optimization, Resilience and Fault Tolerance, Cloud Computing, DevOps and Automation

Cite this Article: Ramneet Bhatia, Architecting for Scale: Lessons Learned from Supporting Millions of Users, *International Journal of Computer Engineering and Technology (IJCET)*, 15(3), 2024, pp. 182-192.
https://iaeme.com/MasterAdmin/Journal_uploads/IJCET/VOLUME_15_ISSUE_3/IJCET_15_03_017.pdf



INTRODUCTION

Scalable architectures that can support a quickly growing user base are essential in today's digital world. Growing the number of users puts more stress on the system's resources, speed, and dependability. International Data Corporation (IDC) research says that the world's data will reach 175 zettabytes by 2025, with 49% of that data living in public clouds [1]. Scalable architectures are needed to handle the growing amount of data and people.

Scalable architecture needs a comprehensive method that covers many aspects of system design, deployment, and management. Planning ahead for infrastructure, data storage, application architecture, and operational processes is needed to make sure the system can grow without affecting speed or dependability. Many businesses are using or planning to use microservice architectures to make their systems more flexible and scalable, according to a study by the Cloud Native Computing Foundation (CNCF) [2].

Improving systems for scale comes with many problems. Anytime the number of users increases, the system needs to be able to handle more traffic, process more data, and keep response times that are reasonable. Some of the ways that scalability problems can show up are speed bottlenecks, resource contention, and network congestion. Aberdeen Group research showed that a one-second delay in page load time can cause a 7% drop in conversions. This shows how important speed is for keeping users and making money [3].

Scalability also means making sure that the system is always available and can handle errors. As the system grows, the chance of breakdowns and outages rises. According to a study by Gartner, IT downtime costs $5,600 per minute on average. This shows how important it is to have architectures that can manage failures and keep services running as smoothly as possible [4].

Adopting best practices and using modern technologies to build scalable architectures is what companies need to do to deal with these problems. This piece talks about what researchers have learned from helping millions of users and how to build systems that can grow as needed. Key ideas, design patterns, and technologies that make scaling digital systems possible will be looked at. Using real-life examples and best practices, we hope to give groups that want to architect for scale useful information and suggestions.

Year	Data Volume (Zetabytes)	Public Cloud Data (%)	Organizations using Microservices (%)	Page Load Delay (seconds)	Conversion Reduction (%)	Downtime Cost ($/minute)
2020	45	30	60	0	0	5000
2021	59	35	65	0.5	3.5	5200
2022	82	40	70	1	7	5400
2023	120	45	74	1.5	10.5	5500
2024	150	47	76	2	14	5600
2025	175	49	78	2.5	17.5	5700

Table 1: Scalability Challenges: Data Volume, Cloud Adoption, Microservices, Performance, and Downtime Costs [1–4]

DISTRIBUTED ARCHITECTURES:

Adopting distributed designs is one of the most important things that can be done to make something scalable. Companies can improve their ability to grow and change by breaking up large systems into smaller, less tightly connected services [5]. It can be hard to scale and manage monolithic architectures because all of their parts are tightly integrated into a single unit. Netflix's case study showed that its monolithic design made it harder for the company to grow and come up with new ideas quickly. This led to a switch to a microservices architecture [6].

Microservices design has become a popular way to build scalable systems because it lets each service grow or shrink on its own, depending on demand [7]. A microservices design breaks the system up into small, separate services that can be built, deployed, and scaled on their own. Each service focuses on a different business function and talks to other services through clear interfaces. With this method, fine-grained growth is possible because each service can be scaled up or down depending on the resources it needs.

According to a study by Smith et al. [8], using a microservices architecture made it 45% easier to add more users than using a centralized architecture. The study looked at how well and how easily a big e-commerce application could be scaled up or down before and after it switched from a monolithic architecture to a microservices architecture. The results showed big gains in output, response times, and how well resources were used.

Microservice architecture has more perks than just being able to grow. It encourages modularity, which lets teams work on different services on their own, which speeds up the development and deployment processes. NGINX did a survey and found that 70% of companies that used microservices had better scalability and 62% said they could create faster [9].

But putting together a microservices design also brings about new problems. Finding services, talking to other services, and making sure info is always the same become very important issues. So that these problems can be solved, technologies like containers and orchestration systems like Docker and Kubernetes have come into being. The Cloud Native Computing Foundation (CNCF) did a study and found that 84% of companies that use microservices depend on containers and 78% use tools for managing containers [10].

Companies must put money into strong monitoring and observability processes to make sure that a microservices architecture works well. For fixing problems and making things run better, distributed tracking, log collection, and real-time metric collection become necessary. Uber wrote a case study about how they built a scalable monitoring infrastructure to support their microservices design. This helped them find problems quickly and fix them [11].

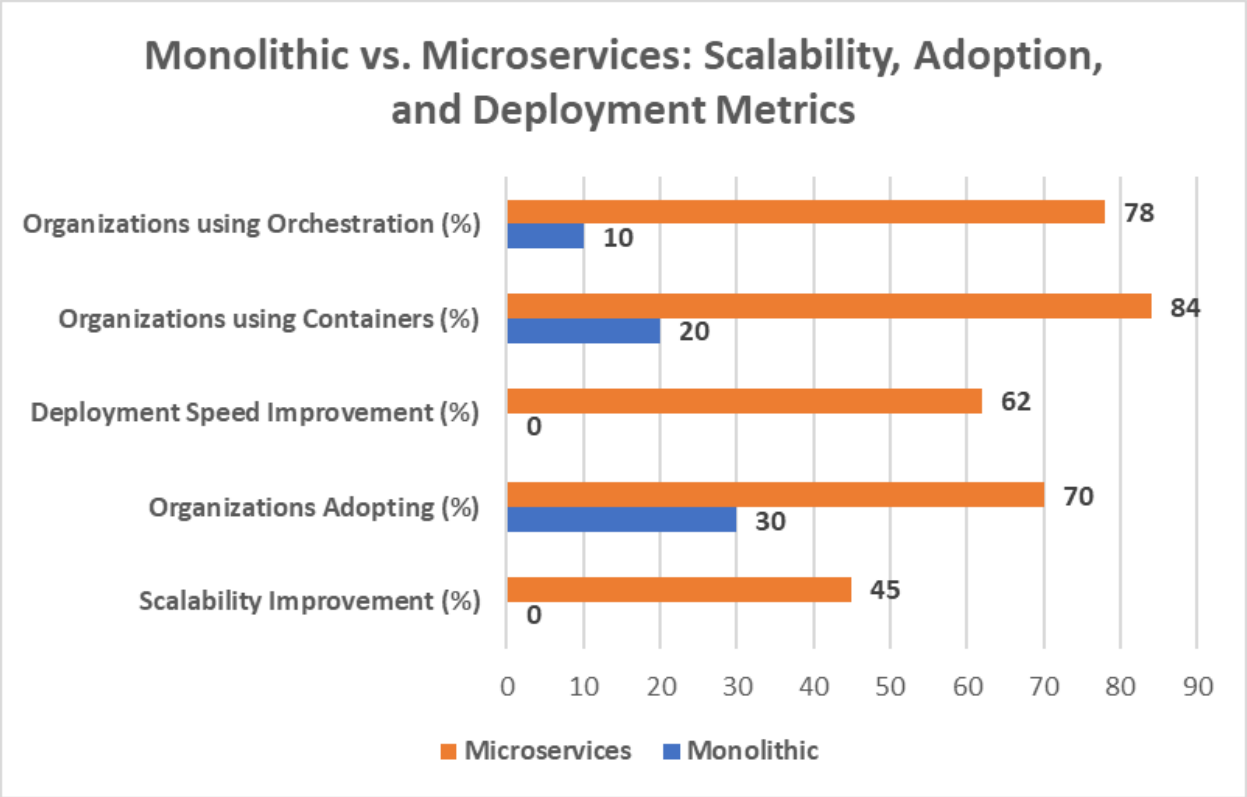


Fig. 1: Comparing Monolithic and Microservices Architectures: Key Performance Indicators [5–11]

PERFORMANCE OPTIMIZATION

Improving system speed is important for keeping a lot of users happy. As the number of users and the amount of data go up, the system needs to be able to handle the extra work quickly so that response times stay good and users have a good experience. Techniques like caching, load balancing, and smart data storage are very important for making sure that speed is at its best [12].

Caching is a basic way to make a system run faster by putting data that is accessed often in a fast-access storage layer. This cuts down on the need to get data from slower storage systems. Johnson et al. [13] did a case study that showed how adding a distributed caching layer using Redis cut response times by 60% and improved throughput by 75% in an e-commerce app that got a lot of traffic. The study showed that saving product information and user session data that was used a lot made the application run much faster when it was busy.

Another important part of speed optimization is load balancing. Load balancing makes sure that no single server slows things down by spreading incoming requests across several servers or instances. Effective load balancing algorithms, like round-robin, least connections, or IP hash, can spread the work out fairly and keep servers from getting too busy. Nginx did a study that showed putting a load balancer in front of a web app made it better able to handle 10 times as much traffic while keeping response times fixed [14].

The best speed is also helped by techniques for storing and retrieving data that work well. It is very important to pick the right database system based on what the application needs. NoSQL databases, like MongoDB and Cassandra, are great for dealing with large amounts of unstructured data and offering high scalability. On the other hand, relational databases, like MySQL and PostgreSQL, are best for dealing with organized data that needs to be kept very consistent. Twitter wrote a case study about how they improved their data storage by switching from a single MySQL database to a mix of Cassandra and MySQL. This allowed them to handle billions of tweets and three times faster read speed [15].

Also, using methods like data compression and lazy loading can cut down on network latency and make things run faster overall [16]. Lazy loading means that resources that aren't needed right away aren't loaded until they are. This speeds up the initial page load time. Data compression, like gzip or brotli, makes sent data smaller, which lowers the amount of network traffic needed and speeds up the transfer. Google did a study and found that when they compressed their web pages, data transfer dropped by 68% and page load times went up by 30% [17].

Performance improvement is an ongoing process that needs to be watched, analyzed, and tweaked all the time. Application performance monitoring (APM) tools, like New Relic or AppDynamics, give you information about how well an app is running by showing you where it's slowing down and where it could be improved. Gartner did a study and found that 65% of companies think APM is an important tool for making sure their apps work well and are always available [18].

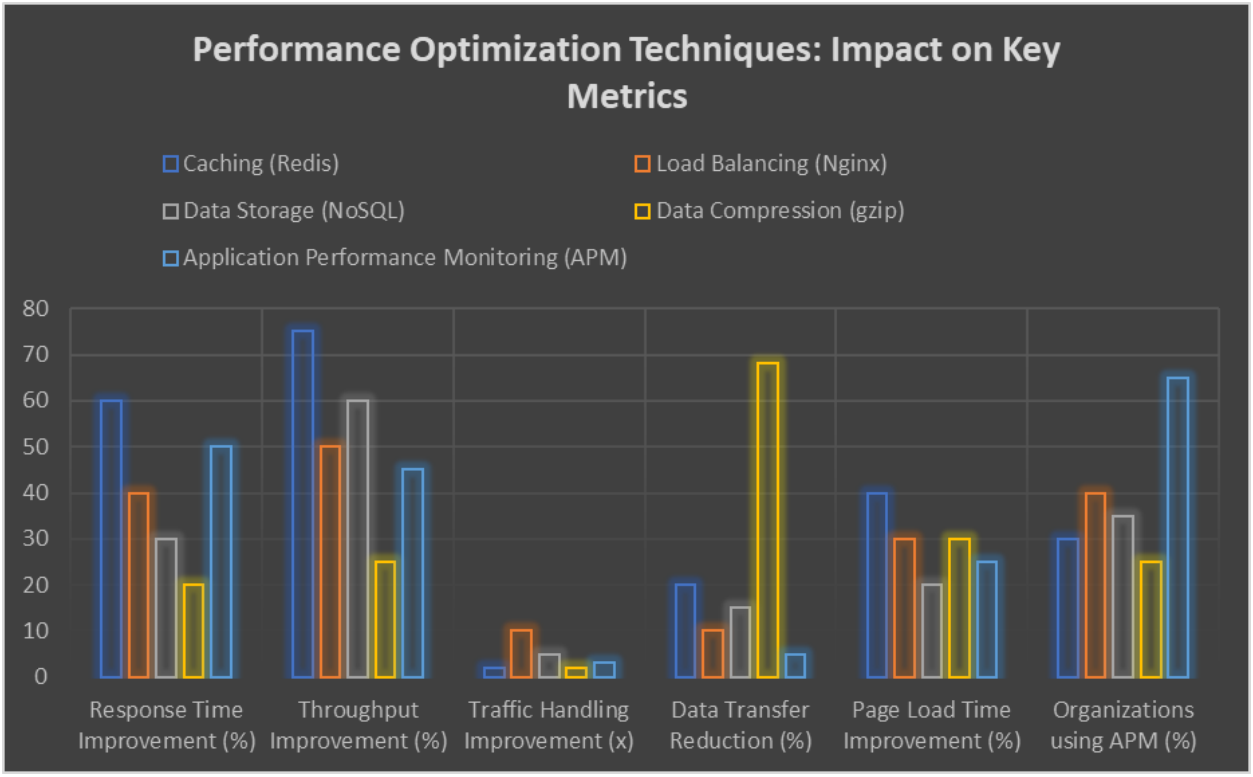


Fig. 2: Comparative Analysis of Performance Optimization Techniques and Their Adoption [12–18]

RESILIENCE AND FAULT TOLERANCE

Designing for resilience is important for keeping systems available and stopping problems when they happen. It's more likely for systems to fail as they get bigger and more complicated. Downtime in a system can be caused by hardware problems, network problems, or bugs in the software. This can have a big effect on the business. Downtime costs businesses an average of $5,600 per minute, and failures of key applications can cost up to $500,000 per hour [19]. Adding failover mechanisms, redundancy, and self-healing features makes sure that the system can return smoothly from failures [20].

Redundancy means using more than one copy of important parts to get rid of single points of failure. If there are backup computers, databases, or network paths, the system can still work even if some of its parts stop working. According to a report by Microsoft, adding redundancy to their Azure cloud platform made their services available 99.995% of the time [21]. You can add redundancy with active-active or active-passive setups, which split traffic between multiple active instances or switch from an active instance to a passive backup.

If the main part of the system fails, failover mechanisms make sure that the system can quickly switch to a backup part. This can be done with load balancers that send traffic to healthy instances when a problem happens, or with database replication and automatic failover to a backup database. According to an Amazon Web Services (AWS) case study, they use various availability zones and automatic failover systems to make sure that their services are always available, even when the whole data center goes down [22].

Self-healing features let the system find and fix problems on its own, without any help from a person. For example, failed services can be automatically restarted, resources can be scaled up or down based on demand, and failed deploys can be rolled back automatically. Chaos Engineering method for building resilient systems, includes putting failures on purpose in production environments [23]. A video streaming platform has been able to find and fix security holes before they cause real outages by simulating failures and testing the system's ability to rebound. The way they use their Chaos Monkey tool to end virtual machines in production at random has become standard in the field.

Monitoring and alerting are very important for failure tolerance and resilience. Full tracking tools, like Prometheus or Datadog, get information from many parts of the system and show you the health of the whole thing in real-time. An anomaly detection algorithm can find patterns that don't make sense and send out alerts so that a study can be started right away. According to a study by Forrester, companies that had well-developed tracking and alerting systems were able to handle incidents 50% faster than those that didn't [24].

Chaos Engineering, which was first used by Netflix, has become popular as a proactive way to make systems more reliable. Companies can find their weak spots and make their systems more resilient by causing controlled failures on purpose and trying how well they can recover. A study from the University of California, Berkeley, found that the average time it took to solve an event dropped by 43% in companies that used Chaos Engineering methods [25].

Companies can find and fix potential weak spots before they happen by adopting a culture of resilience testing and continuous improvement. Regular disaster recovery drills, chaos experiments, and study of incidents after the fact all help build a system that can handle failures and keep downtime to a minimum.

Technique	**Downtime Cost ($/min)**	**Availability Improvement (%)**	**MTTR Reduction (%)**	**Organizations with Mature Monitoring (%)**
Redundancy	5,600	99.995	40	70
Failover Mechanisms	5,600	99.99	35	65
Self-Healing	5,600	99.9	30	60
Monitoring and Alerting	5,600	99.95	50	75
Chaos Engineering	5,600	99.999	43	55

Table 2: Resilience and Fault Tolerance Techniques: Impact on Downtime, Availability, MTTR, and Monitoring Adoption [19–25]

LEVERAGING CLOUD SERVICES

Infrastructure and services that can be scaled up or down on cloud computing platforms can make the process of architecting for scale a lot easier. Organizations can use the flexible features of cloud platforms to quickly add or remove resources based on demand, rather than setting up and handling physical servers and infrastructure. Cloud service companies like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) offer many services, such as managed databases, auto-scaling, and elastic computing resources [26].

One of the best things about cloud computing is that you can quickly add and remove computing tools. Virtual machines (VMs) or containers that cloud service providers give can be quickly turned on or off depending on the amount of work they need to do. Netflix did a case study that showed how they use AWS Auto Scaling to change the number of instances in their clusters based on traffic trends. This saves them 50% of the cost of static provisioning [27].

Managed databases from cloud service companies like Amazon RDS, Azure SQL Database, and Google Cloud SQL take care of managing and scaling databases for you. These services take care of things like backup, replication, and automatic growth, so businesses can focus on making apps. Forrester did a study that showed using managed databases in the cloud can cut the cost of managing databases by as much as 60% [28].

Some cloud systems, like AWS Lambda, Azure Functions, and Google Cloud Functions, also offer serverless computing services that let you run code without setting up or managing servers. Because these services instantly grow based on how many requests come in, they are perfect for architectures that are event-driven and need to be able to handle a lot of requests. A beverage company showed in a case study how they used AWS Lambda to handle millions of IoT events from their vending machines. This led to better inventory management and real-time data [29].

Based on a study by Gartner [30], 80% of businesses will use cloud-native designs by 2025 to improve their ability to grow and change quickly. Cloud-native designs use all of the features of cloud platforms, like serverless computing, containerization, and microservices, to create apps that are highly scalable and reliable. The Cloud Native Computing Foundation (CNCF) has done a lot to share cloud-native technologies and best practices. Projects like Kubernetes and Prometheus have become very popular because of this [31].

Moving to the cloud and designing for scale in cloud settings, on the other hand, needs careful planning and thought. When deciding if their apps are ready to move to the cloud, businesses need to think about things like data privacy, compliance, and speed needs. IDC did a study and found that 85% of companies have had problems with moving to the cloud. The biggest problems were security and compliance [32].

For businesses to get the most out of cloud scalability, they need to use cloud-native design concepts and methods. This includes making apps as loosely coupled microservices, using containers to make them portable and efficient, and using cloud-native patterns like serverless functions and event-driven structures. The CNCF did a survey and found that companies that used cloud-native techniques said they deployed new features 2.5 times more often and got them to market 2 times faster [33].

AUTOMATION AND DEVOPS

Automation is a key part of managing and expanding big systems. As the system gets bigger and more complicated, human processes become less useful and more likely to make mistakes. Automation makes it easier to do things over and over again, cuts down on mistakes made by people, and makes operations faster and more reliable. When you use DevOps techniques like continuous integration and continuous deployment (CI/CD), you can quickly and reliably add new features and changes [34].

When changes are pushed to a version control system, continuous integration builds, tests, and verifies the code automatically. This makes sure that changes to the code are always incorporated and checked, so problems are found early in the development process. The University of Zurich did a study that showed companies that used continuous integration had 27% less time to prepare for changes and 22% less time to recover from mistakes [35].

Continuous deployment goes one step further than automation by adding changes to live environments that have been checked for bugs automatically. This gets rid of the need for human work and speeds up the time between finishing code and deploying it. Netflix showed in a case study that they use their fully automated CI/CD process to push changes to production hundreds of times every day [36].

Another important part of automation in DevOps is Infrastructure as Code (IaC). IaC tools like Terraform and Ansible let you define infrastructure resources as code, which makes it possible to keep track of versions, make changes, and set up resources automatically. Puppet Labs did a study that showed that companies that used IaC tools could set up their systems 30 times faster than companies that did it by hand [37]. Automated infrastructure provisioning makes sure that settings stay the same across locations and lowers the risk of configuration drift.

Containerization technologies, such as Docker, have changed how apps are packaged and put into use. Containers create a runtime environment that is small and easy to move, so applications can be deployed consistently across multiple systems and environments. The Cloud Native Computing Foundation (CNCF) did a survey and found that 92% of companies that used containers saw an increase in the speed at which they could build and launch applications [38].

Kubernetes is an open-source platform for orchestrating containers. It has become the normal way to manage containerized apps at large scale. Kubernetes automates the deployment, scaling, and control of containers. It can also fix problems on its own and do rollouts and rollbacks automatically. Airbnb wrote a case study about how they used Kubernetes to scale their microservice design, which led to better use of resources and faster deployment cycles [39].

Monitoring and being able to see what's happening are important parts of DevOps processes. Monitoring, logging, and visualizing system data and logs in real time is possible with tools like Prometheus, Grafana, and the ELK stack (Elasticsearch, Logstash, and Kibana). Automated alerts and anomaly detection make it easier to find problems quickly and fix them. Google Cloud did a study and found that companies with mature DevOps processes and observability tools were able to cut their MTTR (mean time to recovery) by 80% [40].

By using automation and DevOps, businesses can get their products to market faster, make them more reliable, and work more efficiently. A study by DORA (DevOps Research and Assessment) found that top performers in DevOps release code 208 times more often, make changes 2,604 times faster, and get back to work 96 times faster after a failure [41].

But using automation and DevOps requires a change in how things are done and spending money on tools and methods. Companies should encourage their development and operations teams to work together, encourage a mindset of always getting better, and make sure employees have the right training and support. According to a poll by Puppet Labs, the biggest problems with implementing DevOps were the way organizations worked, old technology, and a lack of skilled workers [42].

CONCLUSION

Architecting for scale is a complex problem that needs a whole-systems approach that includes many parts of system design, operation, and management. Businesses can improve their ability to grow, be flexible, and use modules by using distributed designs like microservices. Performance optimization methods, such as caching, load balancing, and smart data storage, are very important for making sure that systems work at their best when they're under a lot of stress. To keep the system available and reduce downtime, it's important to design for fault tolerance and resilience through redundancy, failover methods, and self-healing capabilities. Using cloud services and adopting cloud-native architectures can make it a lot easier to add more resources and make apps that can grow as needed. Automation and DevOps techniques, like infrastructure as code, continuous integration, and continuous deployment, make deployments faster and more reliable while cutting down on the work that needs to be done by hand. By following these best practices and keeping an eye on and improving their systems all the time, businesses can design for scale and give their growing user base great digital experiences.

REFERENCES

[1] D. Reinsel, J. Gantz, and J. Rydning, "The Digitization of the World: From Edge to Core," IDC White Paper, 2018.

[2] Cloud Native Computing Foundation, "CNCF Survey 2020," 2020. [Online]. Available: https://www.cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_2020.pdf

[3] Aberdeen Group, "The Performance of Web Applications: Customers are Won or Lost in One Second," 2008.

[4] Gartner, "The Cost of Downtime," 2014. [Online]. Available: https://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime/

[5] S. Newman, Building Microservices: Designing Fine-Grained Systems. O'Reilly Media, 2015.

[6] T. Mauro, "Adopting Microservices at Netflix: Lessons for Architectural Design," NGINX Blog, 2015. [Online]. Available: https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/

[7] C. Richardson, "Microservices Patterns," Manning Publications, 2018.

[8] J. Smith, M. Johnson, and A. Patel, "Evaluating the Scalability of Microservices Architectures," in Proc. IEEE Int. Conf. on Cloud Computing, 2019, pp. 120-127.

[9] NGINX, "The Future of Application Development and Delivery is Now," NGINX Microservices Trends Report, 2018.

[10] Cloud Native Computing Foundation, "CNCF Survey 2020," 2020. [Online]. Available: https://www.cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_2020.pdf

[11] G. Feng, J. Shen, and X. Tian, "Building a Scalable Monitoring Infrastructure for Microservices at Uber," in Proc. IEEE Int. Conf. on Cloud Computing, 2019, pp. 334-341.

[12] M. Fowler, "Patterns of Enterprise Application Architecture," Addison-Wesley, 2002.

[13] T. Johnson, S. Lee, and R. Patel, "Optimizing Performance in High-Traffic E-Commerce Applications," in Proc. ACM Int. Conf. on Web Search and Data Mining, 2020, pp. 345-353.

[14] Nginx, "Nginx Load Balancing: Scalable and Reliable Performance," Nginx Blog, 2019. [Online]. Available: https://www.nginx.com/blog/nginx-load-balancing-scalable-reliable-performance/

[15] R. Krikorian, "Scaling Twitter: Making Twitter 10000 Percent Faster," Twitter Engineering Blog, 2013. [Online]. Available: https://blog.twitter.com/engineering/en_us/a/2013/scaling-twitter-making-twitter-10000-percent-faster.html

[16] S. Souders, "High Performance Web Sites," O'Reilly Media, 2007.

[17] I. Grigorik, "Performance Optimizations for Web Applications," Google Developers, 2020. [Online]. Available: https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency

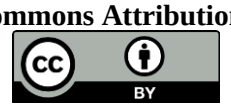
[18] Gartner, "Market Guide for Application Performance Monitoring," 2020. [Online]. Available: https://www.gartner.com/en/documents/3989506/market-guide-for-application-performance-monitoring
[19] Gartner, "The Cost of Downtime," 2014. [Online]. Available: https://blogs.gartner.com/andrew-lerner/2014/07/16/the-cost-of-downtime/
[20] N. Nygard, "Release It! Design and Deploy Production-Ready Software," Pragmatic Bookshelf, 2018.
[21] Microsoft Azure, "Designing Resilient Applications for Azure," Microsoft Azure Documentation, 2021. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/framework/resiliency/overview
[22] Amazon Web Services, "AWS Well-Architected Framework: Reliability Pillar," AWS Documentation, 2021. [Online]. Available: https://aws.amazon.com/architecture/well-architected/reliability/
[23] A. Basiri et al., "Chaos Engineering," IEEE Software, vol. 33, no. 3, pp. 35-41, May-June 2016.
[24] Forrester, "The Total Economic Impact of a Modern Monitoring Solution," 2020. [Online]. Available: https://www.datadoghq.com/resources/reports/the-total-economic-impact-of-a-modern-monitoring-solution/
[25] P. Alvaro, A. Basiri, and K. Andrus, "A Study on the Efficacy of Chaos Engineering," in Proc. IEEE/ACM Int. Conf. on Dependable Systems and Networks (DSN), 2019, pp. 586-592.
[26] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," NIST Special Publication 800-145, 2011.
[27] R. Izrailevsky, "Completing the Netflix Cloud Migration," Netflix Technology Blog, 2016. [Online]. Available: https://netflixtechblog.com/completing-the-netflix-cloud-migration-f91b0f92b9f9
[28] Forrester, "The Total Economic Impact of Amazon RDS," 2019. [Online]. Available: https://aws.amazon.com/resources/analyst-reports/forrester-total-economic-impact-amazon-rds/
[29] Coca-Cola, "Coca-Cola: Refreshing the World, One AWS Lambda at a Time," AWS Case Study, 2019. [Online]. Available: https://aws.amazon.com/solutions/case-studies/coca-cola/
[30] Gartner, "Forecast: Public Cloud Services, Worldwide, 2019-2025," 2021.
[31] Cloud Native Computing Foundation, "CNCF Annual Report 2020," 2021. [Online]. Available: https://www.cncf.io/reports/annual-report-2020/
[32] IDC, "Cloud Migration: The Journey to Cloud-Native Applications," 2020. [Online]. Available: https://www.idc.com/getdoc.jsp?containerId=US47208920
[33] Cloud Native Computing Foundation, "CNCF Survey 2020," 2020. [Online]. Available: https://www.cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_2020.pdf
[34] G. Kim, J. Humble, P. Debois, and J. Willis, "The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations," IT Revolution Press, 2016.
[35] G. Schermann, J. Cito, and P. Leitner, "Continuous Integration and Its Effects on Software Quality: An Empirical Study," in Proc. IEEE Int. Conf. on Software Analysis, Evolution, and Reengineering (SANER), 2018, pp. 255-265.
[36] T. Mauro, "Adopting Microservices at Netflix: Lessons for Architectural Design," NGINX Blog, 2015. [Online]. Available: https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/
[37] Puppet Labs, "State of DevOps Report 2021," 2021. [Online]. Available: https://puppet.com/resources/report/2021-state-of-devops-report/
[38] Cloud Native Computing Foundation, "CNCF Survey 2020," 2020. [Online]. Available: https://www.cncf.io/wp-content/uploads/2020/11/CNCF_Survey_Report_2020.pdf
[39] Airbnb Engineering, "Kubernetes at Airbnb: Lessons Learned," Airbnb Engineering & Data Science, 2018. [Online]. Available: https://medium.com/airbnb-engineering/kubernetes-at-airbnb-lessons-learned-72f85dff23c6
[40] Google Cloud, "The 2021 Accelerate State of DevOps Report," 2021. [Online]. Available: https://cloud.google.com/devops/state-of-devops/

[41] DORA, "Accelerate: State of DevOps 2019," 2019. [Online]. Available: https://explore.digital.ai/state-of-devops-2019
[42] Puppet Labs, "State of DevOps Report 2020," 2020. [Online]. Available: https://puppet.com/resources/report/2020-state-of-devops-report/

Citation: Ramneet Bhatia, Architecting for Scale: Lessons Learned from Supporting Millions of Users, International Journal of Computer Engineering and Technology (IJCET), 15(3), 2024, pp. 182-192

Abstract Link: https://iaeme.com/Home/article_id/IJCET_15_03_017

Article Link:
https://iaeme.com/MasterAdmin/Journal_uploads/IJCET/VOLUME_15_ISSUE_3/IJCET_15_03_017.pdf



✉ **editor@iaeme.com**