

GetawayGo

# Design Oriented Research

Fontys University of Applied Sciences

Anna Kadurina  
07/12/2024

Version	Date	Author(s)	Changes	State
1.0	03/10/2024	Anna Kadurina	Start of the document	First draft
2.0	01/11/2024	Anna Kadurina	Start of the research	First version
3.0	10/11/2024	Anna Kadurina	Research continuation	In progress
3.1	05/12/2024	Anna Kadurina	Research continuation	In progress
4.0	07/12/2024	Anna Kadurina	Finish research	Final version

## Table of Contents

1. Abstract.....	3
2. Introduction .....	4
3. Methodology.....	5
4. Results .....	7
4.1. Objective 1 .....	7
4.2. Objective 2 .....	9
4.3. Objective 3 .....	11
4.4. Objective 4 .....	14
4.5. Objective 5 .....	16
4.6. Objective 6 .....	18
5. Discussion.....	22
6. Conclusion .....	23
7. References .....	24
8. Appendix .....	26
8.1. Appendix A – Project Plan .....	26
8.2. Appendix B – Expert Interview on Scaling Strategies .....	26
8.3. Appendix C – Expert Interview on Fault Tolerance .....	26
8.4. Appendix D – GDPR .....	27
8.5. Appendix E – OWASP Top 10.....	27
8.6. Appendix F – Load Testing .....	27

# 1. Abstract

This research investigates the ways in which a scalable microservices-based platform can efficiently handle bookings for short-term accommodation while upholding high levels of security, user friendliness, and performance. With an emphasis on the quickly expanding market for short-term accommodation, the study tackles issues including peak-time scalability, high availability, and user data security. A number of cloud-native technologies and architectural patterns are looked into in order to suggest the best option for the GetawayGo platform.

## 2. Introduction

The market for short-term accommodation is expanding quickly, which has increased the demand for scalable, reliable platforms that can manage a high volume of customers and reservations. This raises the difficulty of keeping security and performance levels high while preserving user-friendliness.

This study's main objective is to provide a thorough analysis of scalability possibilities, architectural patterns suitable for such applications, needed security measures, monitoring strategies, and cloud native tools.

In this research the main question to be answered is:

*“How can a scalable microservices-based platform effectively support short-term accommodation bookings while ensuring high performance, user-friendliness, and security for global travellers?”*

### 3. Methodology

This research will incorporate the ICT Research Methods (ICT Research Methods, 2024). Specifically, several key strategies will be employed including Library Research, Field Research, Lab Research, Workshop Research and Showroom. Each of these strategies will be tailored to incorporate essential methods necessary for rigorous exploration and analysis within my research domains. By integrating the ICT Research Methods from the DOT Framework, I aim to achieve my research objectives in a structured and methodical matter. This approach not only ensures comprehensive exploration across the application, available work, and innovation domains as defined by the DOT framework but also enhances the robustness and reliability of my research findings.

	Question	Strategies	Method(s)
<b>Objective 1</b>	How can the platform dynamically scale to handle varying loads of users and bookings during peak times?	Library Research	Available product analysis, Best good and bad practices, Literature study, Expert Interview
<b>Objective 2</b>	What architectural patterns can ensure high availability and fault tolerance in a microservices-based platform?	Library Research, Workshop	Literature Study, IT architecture sketching, Expert Interview
<b>Objective 3</b>	What security measures are necessary to protect sensitive user data on a global platform?	Library Research, Lab Research, Showroom	Literature study, Security test, Guideline conformity analysis
<b>Objective 4</b>	How can communication between different microservices be efficiently managed to avoid bottlenecks and ensure real-time data flow?	Library Research, Lab Research	Literature study, System test
<b>Objective 5</b>	What strategies can be employed to effectively monitor and	Library Research	Available product analysis, Literature Study

	manage microservices to prevent service disruptions?		
<b>Objective 6</b>	How can cloud-native tools be used to automate the scaling, deployment, and management of services to improve efficiency?	Library research, Field Research	Literature study, Available product analysis, Document analysis

**Reference from:** [Appendix A – Project Plan](#)

## 4. Results

### 4.1. Objective 1

**How can the platform dynamically scale to handle varying loads of users and bookings during peak times?**

**Methods used: Available Product Analysis, Best good and bad practices, Literature Study and Expert Interview**

In today's fast-paced digital landscape, the ability of a platform to dynamically scale in response to fluctuating user demands is critical for ensuring optimal performance and user satisfaction. This objective investigates how the platform can efficiently manage varying loads of users and bookings, particularly during peak times.

Booking.com operates with a dynamic system architecture specifically crafted to meet ever-changing customer expectations while integrating technological innovations (Medium, Jan 10, 2024.). With over 1,500,000 experiences booked every 24 hours as of 2019, the platform handles immense traffic, highlighting the necessity for a high-capacity, resilient server infrastructure capable of efficiently managing large volumes of data and user interactions during peak usage periods (Medium, Jan 10, 2024). This significant level of demand necessitates a robust and scalable architecture to accommodate a user base that included 100 million mobile users in 2022, further complicating the challenges of concurrent requests and data processing (Medium, Jan 10, 2024).

The high-level architecture of Booking.com employs a microservices architecture pattern, which allows individual services to scale up or down based on demand without impacting other areas of the system (Medium, Jan 10, 2024). This design choice reflects the platform's commitment to providing an efficient and responsive user experience. Moreover, the integration of relational databases such as MySQL ensures that the system can handle high read operations, essential for a platform where hotel searches significantly outnumber actual reservations (Medium, Jan 10, 2024).



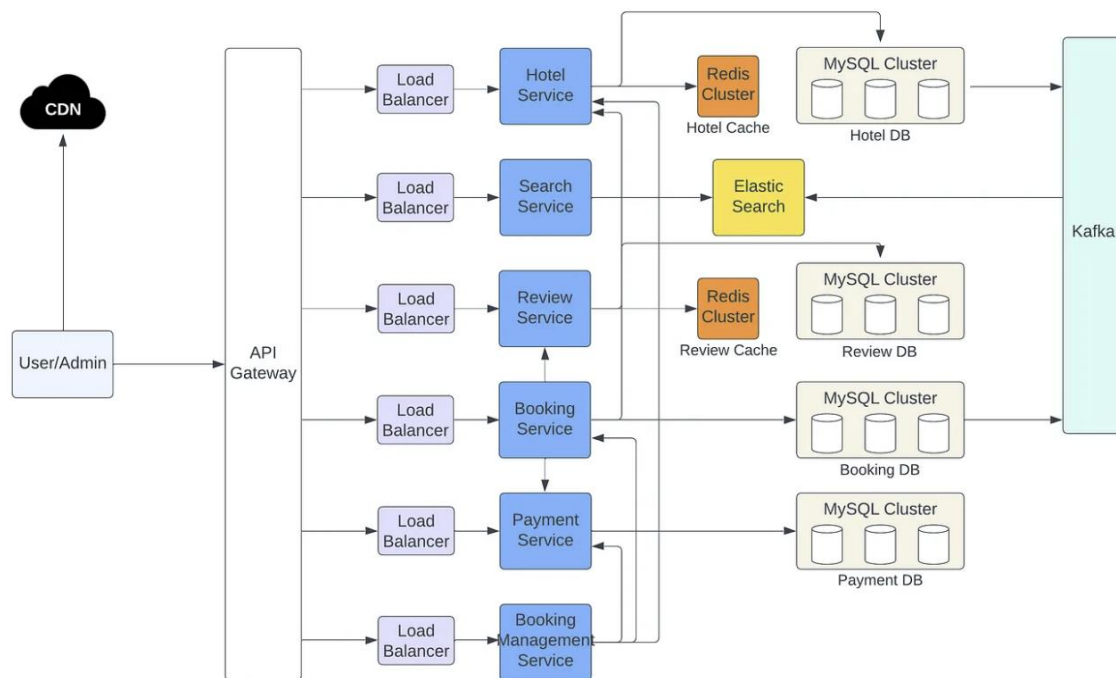


Figure 1 – Possible high-level architecture of Booking.com – Reference from Medium (Jan 10, 2024)

To further explore the issue, the best good and bad practices were explored and taken into consideration. According to Medium (Sep 10, 2023) and Ramotion (Nov 13, 2023), these are some of the key components of developing a scalable application.

Good practices	Bad practices
Stateless Design	Over-engineering
Database sharding	Ignoring database scalability
Auto-scaling	Lack of monitoring
Opt for microservices architecture	
Use load balancers	

Effective scaling strategies are crucial for ensuring platform performance and reliability. According to an interview with Mihail Stoyanov, a Senior Backend Developer (see [Appendix B](#)), the key factors in designing a scalable application are scalability, performance, and cost-efficiency. He emphasized on the importance of horizontal scalability. Additionally, load balancers are essential for distributing traffic evenly across servers. To manage sudden traffic spikes, he recommends auto-scaling with predefined rules. For handling high-volumes of database operations, caching is effective for reads, while splitting the data across multiple

instances is better for writes. Tools like Azure Monitor are highlighted for providing real-time monitoring. These insights underline the importance of combining robust architecture with proactive monitoring to handle dynamic loads efficiently.

## 4.2. Objective 2

**What architectural patterns can ensure high availability and fault tolerance in a microservices-based platform?**

**Methods: Literature Study, IT Architecture Sketching and Expert Interview**

Architectural patterns play a crucial role in ensuring high availability and fault tolerance in microservices-based platforms. Fault tolerance is particularly important in distributed systems like microservices, where the failure of one service could cascade to the entire system. To address this, several strategies are recommended by Meherban Singh (Jun 12, 2023):

- **Design for Failure:** Anticipate and plan for potential failures. For instance, separating user authentication from payment processing ensures that issues in one service do not affect the other.
- **Decentralization:** Avoid single points of failure by replicating services and distributing them across multiple data centers. Service discovery and load balancing are essential.
- **Isolation:** Maintain service independence to contain failures within specific components.
- **Fail-Fast Principles:** Detect and address issues early using robust monitoring systems, automated testing, and health checks. CI/CD pipelines also help catch bugs before they reach Production.
- **Testing Fault Tolerance:** Employ strategies like integration testing, and load testing to simulate and address potential failures, enhancing system resilience.

By combining these patterns and principles, microservices architectures can effectively achieve fault tolerance and high availability.

According to Imperva, in the context of web application delivery, fault tolerance relates to the use of load balancing and failover solutions to ensure availability via redundancy and rapid disaster recovery.

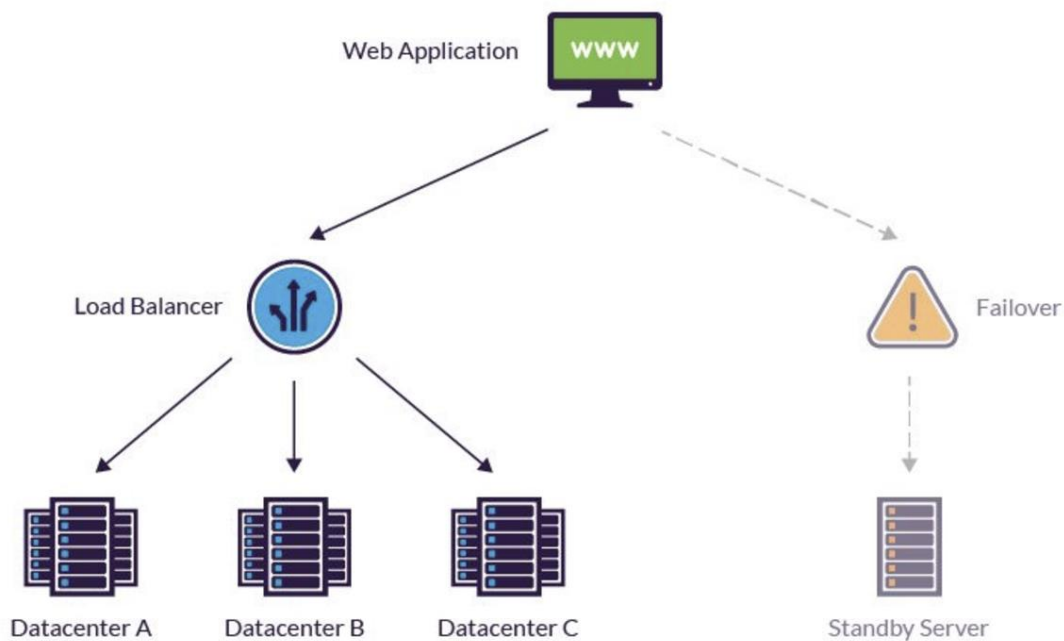


Figure 2 – Fault tolerance for web applications– Reference from Imperva

To learn more about fault tolerance, an interview was conducted with Maria Petkova, a software developer. According to Maria, fault tolerance is paramount for making sure the system can be available 24/7, mentioning that a failure without such mechanisms is going to cause prolonged downtime, loss of revenues or even permanent data. The most typical ways to become more fault-tolerant according to Sahin are redundancy, load balancing, failover mechanisms and data replication. Nonetheless, she pointed out the difficulty of designing such systems as it can cost a lot to have redundant and failover mechanisms. Maria also emphasized the need of monitoring in fault tolerance, outlining how failures can be avoided by proactively identifying possible problems. See [Appendix C](#) for the complete interview transcript.

After the Literature Study and Expert Interview, an architecture sketch for GetawayGo was made. The goal is to handle various loads of users while remaining cost-effective. The application had multiple resource groups to separate concerns, React frontend in a static web app, API Gateway, multiple microservices in Azure App Services and distributed data through various databases. All those resources are able to scale automatically under various loads of data. A typical load balancer is not needed with these types of resources.

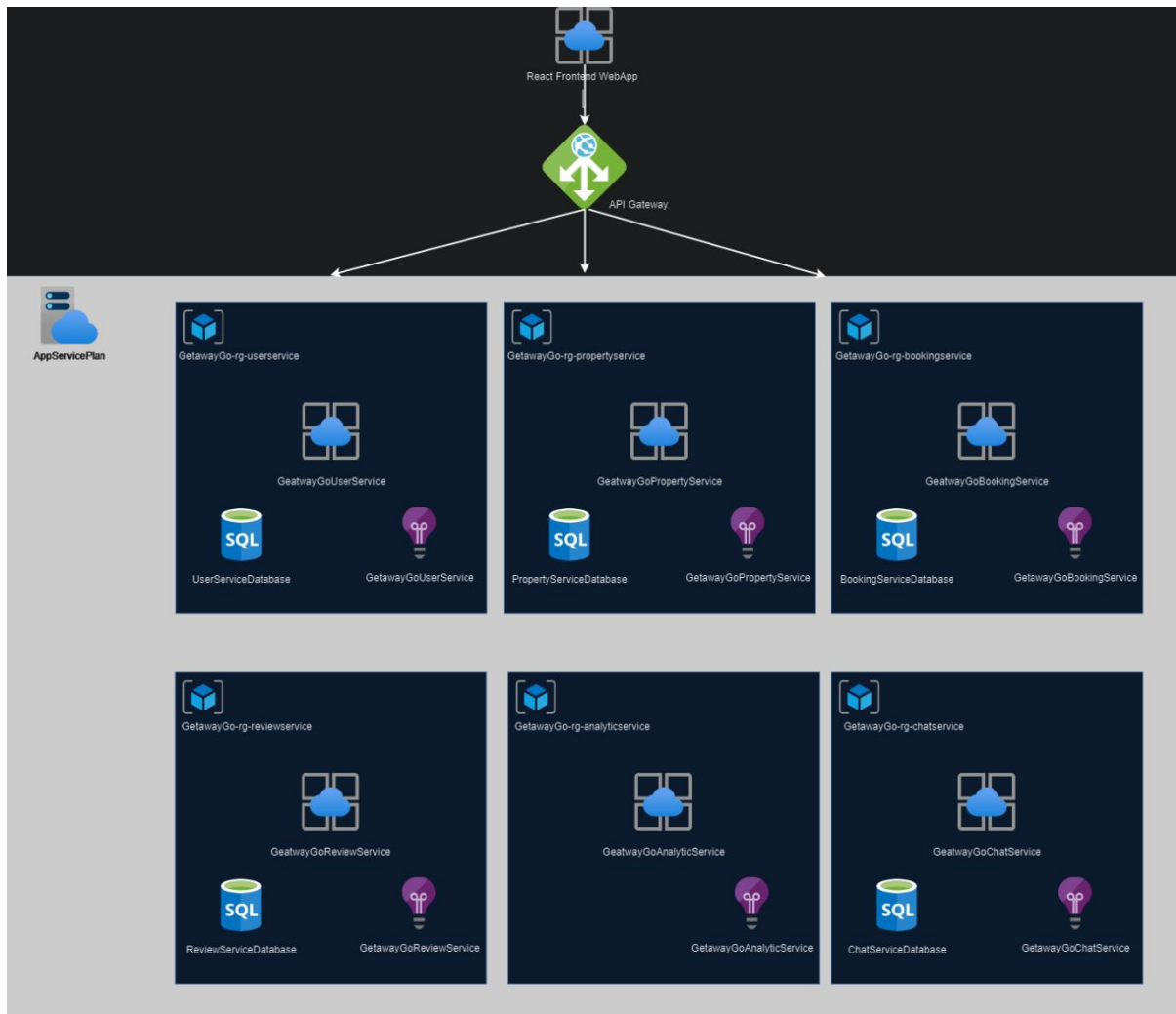


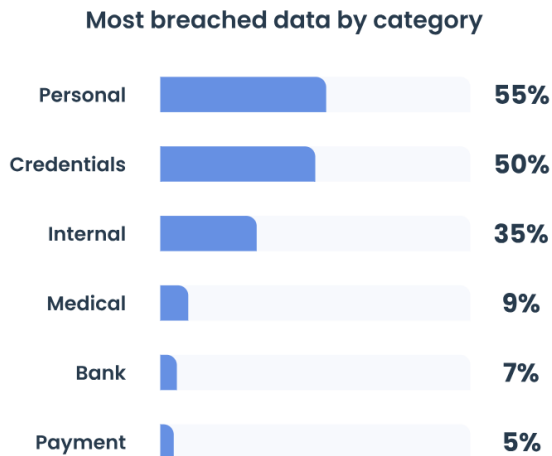
Figure 3 – Cloud Architecture Diagram for GetawayGo

### 4.3. Objective 3

**What security measures are necessary to protect sensitive user data on a global platform?**

**Methods: Literature Study, Security Test and Guideline Conformity Analysis**

According to Yevhen Zhurer (Apr 09, 2024), data security is a combination of processes and tools that aim to protect an organization's sensitive assets. Valuable data must be protected both at rest and in transit. It is important to understand which data is at risk and most commonly stolen.



*Figure 4 – Data Breach Investigations Report by Verizon. Retrieved from Syteca (Apr 09, 2024)*

In order to secure the data, there are some main ways to do that. According to Gartner, the four key methods are:

- Encryption – prevents unauthorized parties from reading your data
- Masking – suppresses or anonymizes high-value data by replacing sensitive information with random characters
- Data erasures – involves cleaning your repositories of data that is no longer used or active
- Data resilience – involves full, backups of your critical data. Storing your valuable data in different locations helps to make it recoverable and resilient to different threats.

To ensure that the applications are secured, security testing needs to be performed. One of the ways is to use OWASP ZAP to detect common vulnerabilities. The OWASP ZAP is integrated into the GetawayGo CI/CD pipelines to perform automated vulnerability scanning. By incorporating ZAP into the pipelines, it is ensured that security testing is continuously integrated into the software development cycle. ZAP performs automated scans on each application being deployed. ZAP is configured to run dynamic scans, where it interacts with the application in real-time to find vulnerabilities in the live web interface. It is being triggered on each commit on the main branch of the microservices. After running, ZAP provides reports on detected vulnerabilities, categorized by risk level (high, medium, low).

←

Jobs in run #20241103.1

UserManagementService

Build

▼

Attempt #2

▼

✓

Build

1m 30s

✓

Initialize job

<1s

✓

Checkout UserMa...

2s

✓

NuGetToolInstaller

<1s

✓

NuGetCommand

4s

✓

VSBUILD

5s

✓

VSTest

7s

✓

SnykSecurityScan

44s

✓

Run OWASP ZAP...

16s

✓

PublishPipelineArt...

5s

✓

PublishBuildArtifa...

4s

✓

Post-job: Checko...

<1s

✓

Finalize Job

<1s

✓

Run OWASP ZAP Scan

🔍


View raw log

⋮

```

1 Starting: Run OWASP ZAP Scan
2 =====
3 Task : Command line
4 Description : Run a command line script using Bash on Linux and macOS and cmd.exe on Windows
5 Version : 2.246.1
6 Author : Microsoft Corporation
7 Help : https://docs.microsoft.com/azure/devops/pipelines/tasks/utility/command-line
8 =====
9 Generating script.
10 Script contents: shell
11 docker run --rm -v C:\Users\anika\agent\vsts-agent-win-x64-3.243.1\work\1\:/zap/wrk:rw -t zapoxy/zap-stable zap.sh
12 ===== Starting Command Output =====
13 "C:\Windows\system32\cmd.exe" /D /E:ON /V:OFF /S /C "CALL "C:\Users\anika\agent\vsts-agent-win-x64-3.243.1\work\_temp\
14 Found Java version 11.0.24
15 Available memory: 7768 MB
16 Using JVM args: -Xmx1942m
17 784 [main] INFO org.parosproxy.paros.Constant - Copying default configuration to /home/zap/.ZAP/config.xml
18 854 [main] INFO org.parosproxy.paros.Constant - Creating directory /home/zap/.ZAP/session
19 854 [main] INFO org.parosproxy.paros.Constant - Creating directory /home/zap/.ZAP/dirbuster
20 855 [main] INFO org.parosproxy.paros.Constant - Creating directory /home/zap/.ZAP/fuzzers
21 855 [main] INFO org.parosproxy.paros.Constant - Creating directory /home/zap/.ZAP/plugin
22 ##[warning]Free memory is lower than 5%; currently used: 96.84%
23 Nov 03, 2024 7:39:23 PM java.util.prefs.FileSystemPreferences$1 run
24 INFO: Created user preferences directory.
25 Writing results to /zap/wrk/zap_report.html
26 Finishing: Run OWASP ZAP Scan

```



ZAP

by

Checkmarx

ZAP Report

Generated on Sun, 3 Nov 2024 19:39:27

ZAP Version: 2.15.0

ZAP by [Checkmarx](#)

Summary of Alerts

Risk Level	Number of Alerts
High	0
Medium	0
Low	0
Informational	0
False Positives:	0

Alerts

Name	Risk Level	Number of Instances
------	------------	---------------------

Alert Detail

To ensure the protection of user data and comply with relevant legal frameworks, GetawayGo adheres to the General Data Protection Regulation (GDPR). GDPR is a comprehensive data privacy regulation set forth by the European Union to protect individuals' personal data and privacy. GetawayGo's approach to data protection is outlined in the accompanying GDPR

Compliance Document (see [Appendix D](#)), which details the methods and measures taken to ensure that user data is handled securely.

The key elements of the GDPR compliance document include:

- **Data Collection and Usage:** Only essential personal data is being collected.
- **Legal Basis for Processing Data:** Users are informed of their rights and the purpose of data collection, ensuring transparency.
- **Data Security:** To protect user data, GetawayGo employs encryption, security assessments, role-based access control, and secure cloud storage.
- **User Rights:** Users can exercise their GDPR rights, including the right to access, update, or delete their personal data.
- **Data Retention Policy:** Personal data is retained only for as long as necessary.

By following these guidelines, GetawayGo aligns with GDPR requirements, ensuring the highest standards of data privacy and security.

To safeguard the GetawayGo platform from the most critical web application security risks, the application aligns with the OWASP Top 10 security guidelines. These risks are some of the most common vulnerabilities faced by web applications, and addressing them is crucial for ensuring the platform's security and integrity. The OWASP Top 10 Document (see [Appendix E](#)) outlines how each identified risk is mitigated, including steps such as implementing role-based access control (RBAC) to prevent unauthorized access, encrypting sensitive data both in transit and at rest, and using parameterized queries to avoid injection flaws. Through these measures, GetawayGo takes a proactive approach to managing and mitigating the risks outlined in the OWASP Top 10. For further details, please refer to the OWASP Top 10 Document in [Appendix E](#).

## 4.4. Objective 4

**How can communication between different microservices be efficiently managed to avoid bottlenecks and ensure real-time data flow?**

**Methods: Literature Study and System Test**

Communication between microservices is a fundamental aspect of a microservices architecture. Since microservices are distributed and independently deployable, they need to communicate with each other efficiently and reliably to function as a cohesive system (Design

Gurus). There are different strategies to handle communication between microservices. According to Design Gurus, some of them are:

1. Synchronous Communication – It involves direct, immediate interaction between services, where one service sends a request to another and waits for a response. Common protocols for this communication include HTTP/HTTPS and gRPC. Tools include RESTful APIs, gRPC, Thrift. Synchronous communication is simple to implement and understand.
2. Asynchronous Communication – Asynchronous communication allows services to communicate without waiting for an immediate response. This is typically done using message queues or event streams, where messages are sent to a broker and processed by the recipient service at a later time. Tools include RabbitMQ, Apache Kafka, AWS SQS, Google Pub/Sub. Asynchronous communication improves decoupling between service, enabling better scalability and resilience by allowing services to operate independently and handle different workloads at their own pace.
3. Message Queues – Use message queues to enable asynchronous communication between microservices. Message queues store and forward messages, ensuring they are reliably delivered even if the recipient service is temporarily unavailable. Tools include RabbitMQ, Apache Kafka, Amazon SQS, Google Cloud Pub/Sub.
4. Event-Driven Architecture – Services communicate by publishing and subscribing to events. Tools include Apache Kafka, AWS SNS, Google Cloud Pub/Sub, NATS.
5. Service Discovery – It is used to enable dynamic location and communication between services. Tools include Consul, Eureka, Kubernetes DNS, Zookeeper.
6. API Gateway – It manages and routes client requests to the appropriate microservices. The API Gateway can handle cross-cutting concerns such as authentication.

To further address this question, a system test was conducted using load testing. Load balancing is a critical mechanism for managing microservice communication as it ensures that incoming requests are evenly distributed across services, preventing bottlenecks and enhancing the system's ability to handle high traffic. By testing the application under simulated traffic using Azure Load Testing, it was possible to evaluate the system's response times, throughput, and error rates under varying loads.



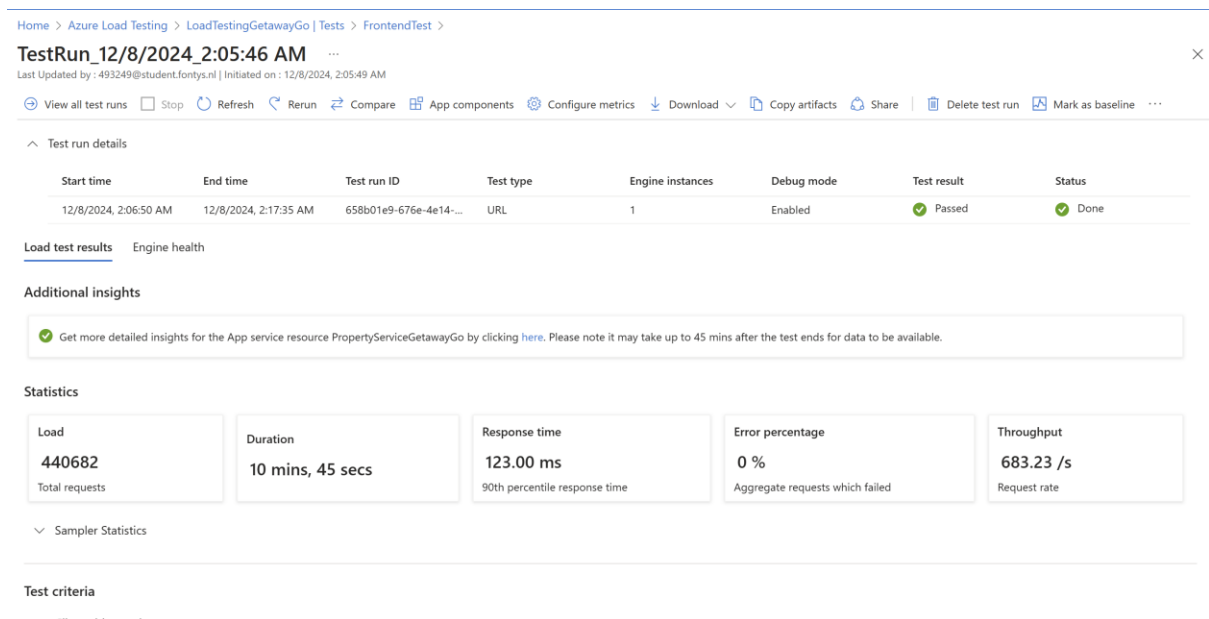


Figure 7 – Overview of the load balancing test – Reference from [Appendix F](#)

The system performed extremely well. It handled a lot of requests and it never compromised the performance. The load balancing tests are executed via a script generated by Azure for Apache Jmeter. More information on the load testing can be found in [Appendix F](#).

## 4.5. Objective 5

**What strategies can be employed to effectively monitor and manage microservices to prevent service disruptions?**

### Methods: Available Product Analysis and Literature Study

According to Charles Mahler (Mar 06, 2024), users have higher expectations than ever when it comes to performance and reliability in apps they use every day. A critical part of meeting those expectations is having a robust monitoring system in place. Some key benefits of having a solid monitoring system are:

- Improved End User Experience
- Improved Availability
- Cost Savings
- Enhanced Observability

There are a variety of metrics used to measure the performance of microservice applications. Common metrics are:

- Latency and Response Time – how quickly the services respond to requests
- Error Rate – frequency of errors within your services

- Resource Utilization – monitoring CPU, memory, and other resources
- SLO/SLI – Service Level Objectives (SLOs) and Service Level Indicators (SLIs) are critical for measuring the performance of the services against set benchmarks

The above metrics are derived from the three following types of data collected when monitoring applications:

- Logs – They are detailed textual records generated by software applications and infrastructure components. Logs contain events, transactions, and other activities that occur within the system.
- Metrics – They are a quantitative data points that measure various aspects of system performance and health. Common metrics include CPU usage, memory consumption, response times, etc.
- Traces – They provide a detailed step-by-step account of a single transaction or request as it travels through the various components of a distributed system.

All systems require monitoring, especially the ones with most traffic. Spotify, a worldwide used application for streaming music, utilized Grafana and Prometheus and ELK Stack, according to Design Gurus. As it is stated in the Uber Blog, Uber’s software architecture consists of thousands of microservices. To maintain the growth, Uber’s Observability team built a robust, scalable metrics and alerting pipeline responsible for detecting, mitigating and notifying engineers of issues with their services as soon as they occur. There are 2 in-data center alerting systems built, called uMonitor and Neris, that flow into the same notification and alerting pipeline. uMonitor is the metrics-based alerting system that runs checks against the metrics database M3, while Neris primarily looks for alerts in host-level infrastructure. In addition, there is also a black box alerting system which detects high-level outages from outside the data center in cases where the internal systems fail or there are full data center outages.

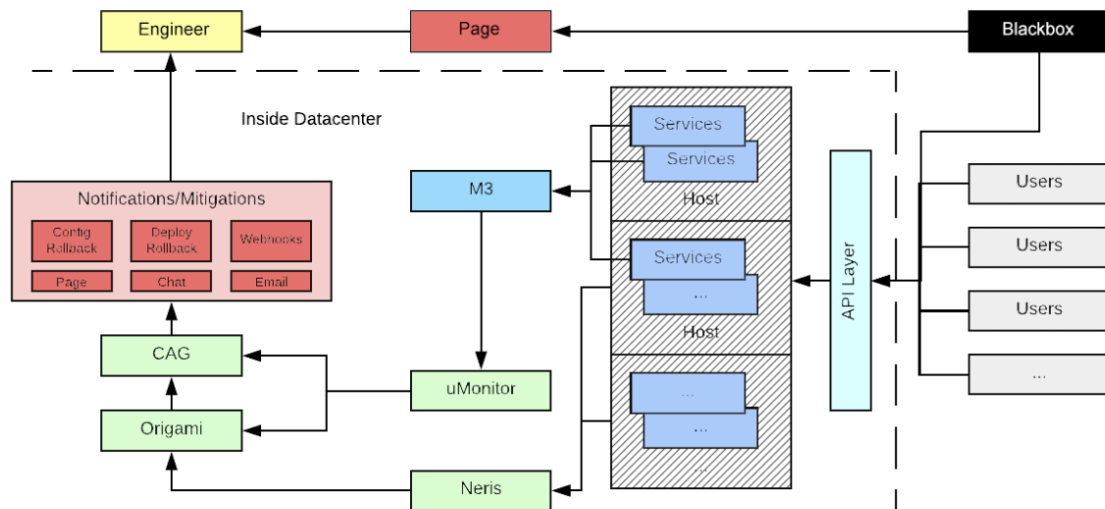


Figure 8 – Alerting at Uber

## 4.6. Objective 6

**How can cloud native tools be used to automate the scaling, deployment, and management of services to improve efficiency?**

**Methods: Literature Study, Available Product Analysis and Document Analysis**

A cloud-native application employs cloud technology and computing models to create and run scalable applications in a flexible environment while minimizing infrastructure costs (Joydip Kanjilal, Jun 18, 2024). Some benefits of Cloud-Native Applications are:

- Improved Agility – Cloud-native applications enable faster application development and delivery.
- Improved Resilience and Fault Tolerance - Cloud-native applications have built in support for resilience and fault tolerance. Kubernetes is used to enable applications to recover from failures automatically, without manual intervention.
- Enhanced Resource Utilization – By scaling resources as needed, cloud native application help to optimize resource utilization.
- Flexible Deployment – A cloud-native approach to application deployment and management provides support for automation. Using CI/CD pipelines with infrastructure as code (IaC) and configuration management tools simplifies and streamlines the deployment process.

To build a resilient and scalable application using cloud-native tools, key strategies should be followed, according to Joydip Kanjilal. First is to embrace microservices. By doing this, you can build applications that comprise a suite of small, independent microservices that can be independently developed, deployed and scaled. Second is to leverage CI/CD. Automate the testing process to detect problems in your application quickly. To release the software in a short time and with reduced effort, automate the deployment process. Cloud-native apps need DevOps processes such as CI/CD and IaC and monitoring and feedback continuously. Lastly, utilize Observability and Monitoring. Monitor the real-time health and performance of your applications with comprehensive logging, distributed tracing and metrics collection.

There are a lot of tools that can help with scalability, deployment and management. Here are some of the available products.

Tool	Primary Focus	Key Features	Use Case	Efficiency Impact
Kubernetes	Orchestration	Autoscaling, rolling updates, self-healing	Scalable microservices applications	Automates scaling and resource usage
AWS Lambda	Serverless Computing	Event-driven, auto-scaling, cost-efficient	Event-driven tasks	Reduces server management overhead
Terraform	IaC	Cross-platform, declarative syntax, automation	Multi-cloud infrastructure deployment	Ensures consistency, saves time
Azure DevOps	CI/CD pipelines	Automated deployment, advanced strategies	Application lifecycle automation	Faster, reliable updates
Istio	Service Mesh	Traffic management, service discovery, security	Secure, efficient service communication	Streamlines interservice communication

Cloud-native tools like Kubernetes, AWS Lambda, Terraform, Azure DevOps, and Istio collectively enhance automation, scaling, and management of services.

To answer this sub-question properly, a thorough document analysis was conducted. Below are key findings from the analysis:

### 1. Kubernetes Documentation

Key features:

- Cluster Autoscaler – Automatically adjusts the size of the Kubernetes cluster based on the workload demand.
- Horizontal Pod Autoscaler (HPA) – Scales the number of pods based on real-time CPU or memory usage metrics
- Rolling Updates and Canary Deployments – Allow seamless updates to applications without downtime

Kubernetes prioritizes resources efficiency through dynamic scaling and automated recovery from failures. These features align with the needs of applications like GetawayGo.

### 2. AWS Lambda Documentation

Key features:

- Event-Driven Scaling – Automatically triggers functions in response to specific events
- Resource Efficiency – Pay-per-use pricing model ensures that resources are only allocated and billed during function execution
- Integration – Easily integrates with other AWS services

AWS Lambda supports cost-effective scaling by eliminating the need for pre-provisioned infrastructure. It is useful for handling varying workloads.

### 3. Terraform Documentation

Key features:

- Infrastructure as Code (IaC) – Enables repeatable and consistent deployment of cloud resources

Terraform's IaC approach ensures consistent infrastructure setups, making it ideal for automating deployment pipelines.

#### 4. Azure DevOps Documentation

Key features:

- CI/CD Pipelines – Automates the build, test, and release process.
- Integration with Kubernetes
- Monitoring and Feedback

Azure DevOps simplifies the deployment process with seamless integration into cloud-native ecosystems.

#### 5. Istio Documentation

Key features:

- Traffic Routing – Allows control of service traffic, including load balancing and fault injection
- Security – Ensures secure communication between microservices using mutual TLS

Istio provides a robust framework for managing service-to-service communication, enabling real-time monitoring.

The document analysis highlights the strength of these tools in automated scaling, deployment, and management tasks.

## 5. Discussion

The findings from this research have highlighted the importance of integrating advanced technological solutions and best practices to ensure scalability, performance, security, and reliability in GetawayGo. Each objective provided valuable insights:

1. **Scalability:** Leveraging auto-scaling and load balancers ensures the platform can handle varying loads during peak times.
2. **Fault tolerance:** The principles of redundancy, isolation, and fail-fast strategies, combined with proactive monitoring are critical for ensuring system availability.
3. **Security:** OWASP ZAP integration and adherence to GDPR and OWASP Top 10 guidelines reinforce user data protection.
4. **Inter-Microservice Communication:** Efficient communication using asynchronous methods like message queues and event-driven architectures minimized bottlenecks.
5. **Monitoring and Management:** Observability through logs, metrics, and traces, along with tools like Azure Monitor, ensures system health.
6. **Cloud-Native Tools:** Cloud-native solutions streamline scalability, deployment and management. Such tools are Kubernetes, AWS Lambda, and Terraform.

## 6. Conclusion

The research demonstrates that a microservices-based platform like GetawayGo can effectively support global short-term accommodation bookings by employing scalable, secure and efficient architectural patterns. By integrating dynamic scaling, fault-tolerant mechanisms, robust security measures, and efficient communication protocols, the platform can ensure high performance and reliability.

Future research could explore emerging technologies like AI for predictive scaling and enhanced user experience.



## 7. References

*ICT Research Methods* (2024). Retrieved from:

<https://ictresearchmethods.nl/>

Talha, S. (2024, January 10). *High-level system architecture of Booking.com*. Medium. Retrieved from:

<https://medium.com/@sahintalha1/high-level-system-architecture-of-booking-com-06c199003d94>

AppsChisel. (2023, Sep 10). *Building Scalable Web Applications: Best Practices and Pitfalls to Avoid*. Medium. Retrieved from:

[https://medium.com/@resilient\\_indigo\\_turtle\\_10/building-scalable-web-applications-best-practices-and-pitfalls-to-avoid-2c8c7af7afae](https://medium.com/@resilient_indigo_turtle_10/building-scalable-web-applications-best-practices-and-pitfalls-to-avoid-2c8c7af7afae)

Ramotion. (2023, Nov 13). *Best Practices to Develop Scalable Web Applications*. Retrieved from:

<https://www.ramotion.com/blog/scalable-web-applications/>

Meherban Singh (2023, Jun 12). *Fault Tolerance in Microservices Architectures*. Medium. Retrieved from:

<https://medium.com/cloud-native-daily/fault-tolerance-in-microservices-architecture-patterns-principles-and-techniques-explained-20cfa3d7f98f>

Yevhen Zhurer (2024, Apr 09). *10 Data Security Best Practices: Simple Methods to Protect Your Data*. Syteca. Retrieved from:

<https://www.syteca.com/en/blog/data-security-best-practices>

Gartner (N/A). *Data Security*. Retrieved from:

<https://www.gartner.com/en/marketing/glossary/data-security>

Design Gurus (N/A). *How do you handle communication between microservices?* Retrieved from:

<https://www.designgurus.io/answers/detail/how-do-you-handle-communication-between-microservices>

Charles Mahler (2024, Mar 06). *An Introduction to Microservices Monitoring – Strategies, Tools, and Key Concepts*. Influxdata. Retrieved from:

<https://www.influxdata.com/blog/microservices-monitoring-strategies-tools-key-concepts/>

Design Gurus (N/A). *Discuss Spotify System Architecture*. Retrieved from:

<https://www.designgurus.io/answers/detail/discuss-spotify-system-architecture>

Uber Blog (2018, Nov 20). *Observability at Scale: Building Uber's Alerting Ecosystem*. Retrieved from:

<https://www.uber.com/en-NL/blog/observability-at-scale/>

Joydip Kanjilal (2024, Jun 18). *Strategies for Building Resilient, Scalable Cloud-Native Applications*. Cloud Native Now. Retrieved from:

<https://cloudnativenow.com/promo/cloud-native/strategies-for-building-resilient-scalable-cloud-native-applications/>

Kubernetes Documentation. Retrieved from:

<https://kubernetes.io/docs/home/>

AWS Lambda Documentation. Retrieved from:

<https://docs.aws.amazon.com/lambda/>

Terraform Documentation. Retrieved from:

<https://developer.hashicorp.com/terraform/docs>

Azure DevOps Documentation. Retrieved from:

<https://learn.microsoft.com/en-us/azure/devops/?view=azure-devops>

Istio Documentation. Retrieved from:

<https://istio.io/latest/docs/>

## 8. Appendix

### 8.1. Appendix A – Project Plan

[ProjectPlan.pdf](#)

### 8.2. Appendix B – Expert Interview on Scaling Strategies

This interview was conducted to gather expert insights on scaling strategies for the platform.

**Name and role of the expert:** Mihail Stoyanov, Senior Backend Developer

**Date:** 05/11/2024

**Q1:** What are the most important factors to consider when designing a platform to handle variable traffic loads?

**A1:** In my opinion, the most factors are scalability, performance, and cost-efficiency. You need to design for horizontal scalability to make sure that your system can add or remove instances without affecting performance. Using load balancers is also key.

**Q2:** How would you recommend managing sudden traffic spikes without over-provisioning resources?

**A2:** I'd recommend implementing auto-scaling with predefined rules and thresholds. For example, if CPU usage exceeds 70%, new instances can spin up automatically. A well configured load balancer can also distribute the load effectively.

**Q3:** How can databases handle a high volume of read and write operations during peak times?

**A3:** For reads, caching is a good idea. For writes, split your data across multiple instances.

**Q4:** Are there any tools or technologies you recommend for monitoring?

**A4:** For monitoring, tool like Azure Monitor is great for real-time insights.

**Q5:** Can you share a real-world example of successfully scaling a platform?

**A5:** Sure! At a previous job, we scaled a platform during a flash sale. We set up auto-scaling with a load balancer. This ensured the system handles a 5x traffic spike without downtime.

### 8.3. Appendix C – Expert Interview on Fault Tolerance

The interview was conducted to gather insights on fault tolerance.

**Name and role of the expert:** Maria Petkova, Software Developer

**Date:** 07/11/2024

**Q1:** Do you think fault tolerance is important in IT systems?

**A1:** In my opinion it is crucial because it ensures uninterrupted service. Without fault tolerance, a single failure could lead to downtime, revenue loss, or even data loss.

**Q2:** What are some common strategies for implementing fault strategies?

**A2:** Some common strategies I know are redundancy, load balancing, failover mechanisms, and data replication.

**Q3:** What are the biggest challenges in designing a fault-tolerant system?

**A3:** Implementing redundant systems and failover mechanisms can be expensive. Also, ensuring that failover mechanisms work as intended can be difficult, especially in real-world scenarios.

**Q4:** What role does monitoring play in fault tolerance?

**A4:** Monitoring is vital for fault tolerance. It helps identify potential issues before they cause failures.

#### 8.4. Appendix D – GDPR

[GDPR.pdf](#)

#### 8.5. Appendix E – OWASP Top 10

[OWASPTop10.pdf](#)

#### 8.6. Appendix F – Load Testing

[LoadTesting.pdf](#)