Babeș-Bolyai University, Faculty of Mathematics and Computer Science

AN EMPIRICAL VALIDATION OF SCALABILITY AND FAULT TOLERANCE PATTERNS IN A .NET MICROSERVICES ARCHITECTURE ON KUBERNETES

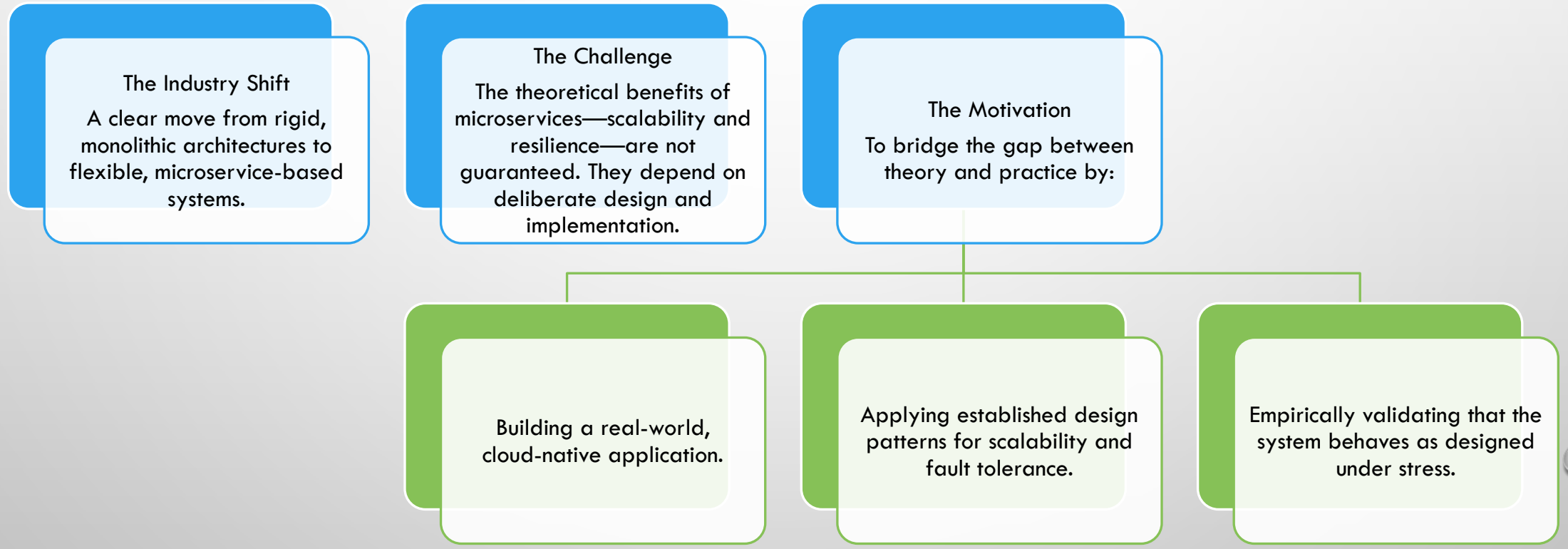
Supervisor
Assoc. Prof. Rares Florin Boian Ph.D.

Author
Daniel Cujba

CONTENT

- Motivation for the Topic
- Current State of the Art
- Methods, Architecture & Implementation
- Testing, Problems & Results
- Live Application Demonstration
- Conclusions & Future Directions

MOTIVATION FOR THE TOPIC



STATE OF THE ART

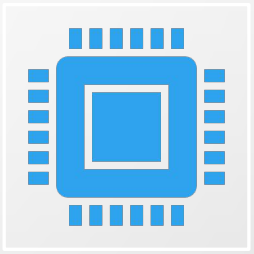
Architectural Paradigm:

Microservices are the de-facto standard for large-scale, distributed systems (e.g., Netflix, Amazon).

Enabling Technologies:

Containerization (Docker): Standardizes application packaging.

Orchestration (Kubernetes): The industry leader for managing, scaling, and healing containerized applications.



Key Design Patterns

Scalability: Horizontal Pod Autoscaling (HPA) is a core Kubernetes feature.

Fault Tolerance: Patterns like **Retry, Circuit Breaker,** and **Bulkhead** are essential for building resilient inter-service communication.

Asynchronous Communication: Message brokers like **RabbitMQ** are widely used to decouple services and absorb load spikes.

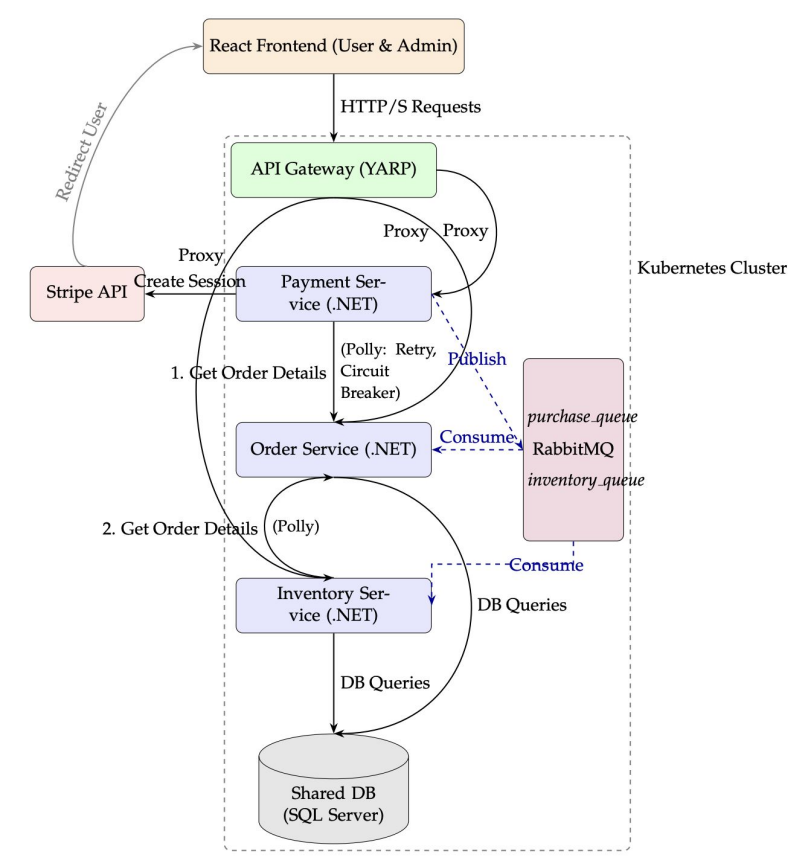
METHODS, ARCHITECTURE & IMPLEMENTATION

•TECHNOLOGY STACK:

- **BACKEND:** .NET 9 MICROSERVICES (INVENTORY, ORDER, PAYMENT).
- **FRONTEND:** REACT (USER STOREFRONT & ADMIN DASHBOARD).
- **GATEWAY:** YARP (YET ANOTHER REVERSE PROXY).
- **DATABASE:** SHARED SQL SERVER.

•IMPLEMENTATION METHODS:

- **SCALABILITY:** KUBERNETES HORIZONTALPODAUTOSCALER CONFIGURED TO TRIGGER ON >80% CPU UTILIZATION.
- **FAULT TOLERANCE:** THE .NET RESILIENCE LIBRARY, **POLLY**, WAS USED TO IMPLEMENT:
  - **RETRY POLICY:** WITH EXPONENTIAL BACKOFF FOR TRANSIENT ERRORS.
  - **CIRCUIT BREAKER POLICY:** TO ISOLATE FAILING SERVICES AFTER 10 CONSECUTIVE FAILURES.
- **DECOUPLING: RABBITMQ** USED FOR ASYNCHRONOUS COMMUNICATION BETWEEN SERVICES AFTER A PAYMENT IS PROCESSED, WITH DURABLE QUEUES TO PREVENT MESSAGE LOSS.



TESTING, PROBLEMS & RESULTS

- **PROBLEM:** HOW TO EMPIRICALLY PROVE THE SYSTEM'S STABILITY AND VALIDATE THE AUTO-SCALING CONFIGURATION?
- **SOLUTION:** A 30-MINUTE SUSTAINED LOAD TEST USING **LOCUST**.
  - **TARGET:** INVENTORYSERVICE GET ENDPOINT.
  - **LOAD:** 10 CONCURRENT USERS.

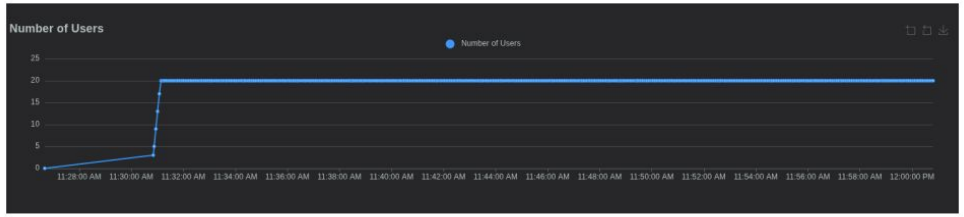


Figure 6.3: Number of users during the 30-minute load test, showing a stable count of 10 concurrent users.

TESTING, PROBLEMS & RESULTS

- **RESULTS:**
  - **PERFORMANCE:** STABLE THROUGHPUT OF **~124 RPS** AT A LOW **35MS MEDIAN LATENCY**.
  - **RELIABILITY: ZERO FAILURES** RECORDED.
  - **KEY FINDING:** THE SERVICE POD'S CPU USAGE WAS SUSTAINED AT ITS MAXIMUM LIMIT. THIS IS THE EXACT CONDITION REQUIRED TO TRIGGER THE HPA, THUS **VALIDATING THE SCALING MECHANISM.**

Figure 6.1: Requests per Second (RPS) during the 30-minute load test.

Figure 6.4: CPU usage of the InventoryService pod during the 30-minute load test, showing sustained maximum utilization.

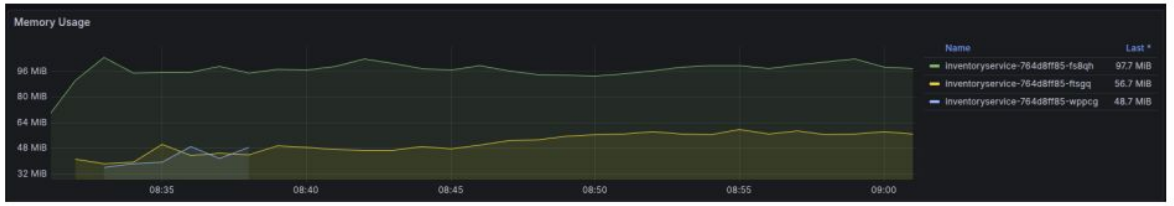


Figure 6.5: Memory usage of the InventoryService pod during the 30-minute load test, showing stable memory consumption.

DEMO

CONCLUSIONS

- **ORIGINAL ASPECTS & CONTRIBUTIONS:**
  - THIS WORK PROVIDES AN **END-TO-END IMPLEMENTATION** OF A MODERN, CLOUD-NATIVE APPLICATION.
  - THE PRIMARY CONTRIBUTION IS THE **EMPIRICAL VALIDATION** OF THE INTERPLAY BETWEEN APPLICATION-LEVEL RESILIENCE PATTERNS (POLLY) AND INFRASTRUCTURE-LEVEL AUTO-SCALING (KUBERNETES HPA).
- **ADVANTAGES OF THE SOLUTION:**
  - **SCALABLE:** DESIGNED TO AUTOMATICALLY HANDLE TRAFFIC SPIKES.
  - **RESILIENT:** TOLERANT TO TRANSIENT ERRORS AND SERVICE FAILURES.
  - **DECOUPLED & MAINTAINABLE:** ADHERES TO MICROSERVICE BEST PRACTICES.
- **LIMITATIONS:**
  - THE TEST WAS CONDUCTED ON A SINGLE SERVICE IN A CONTROLLED ENVIRONMENT.
  - DID NOT EXPLORE COMPLEX DATA CONSISTENCY PATTERNS OR SERVICE MESH TECHNOLOGIES.
  - THE LIMITATION OF THE HARDWARE TO 16 GB OF RAM AND 8 CPU CORES MADE TESTING MORE COMPLEX IMPLEMENTATIONS IMPOSSIBLE
- **FUTURE DEVELOPMENT DIRECTIONS:**
  - IMPLEMENT **CHAOS ENGINEERING** TO PROACTIVELY TEST RESILIENCE BY INJECTING FAILURES.
  - INVESTIGATE AI/ML FOR PREDICTIVE AUTO-SCALING.