WE ARE E3UDRES2
EUROPEAN UNIVERSITY

Informatik & Security

fh/// st.pölten
UNIVERSITY OF APPLIED SCIENCES

Assessing the Impact of Asynchronous Communication on Resilience and Robustness

A Comparative Study of Microservice and Monolithic Architectures

Nathanael Bosilia, Gerald Weinberger and Philipp Haindl

Introduction

- **The Problem:** One configuration error in a monolithic application
→ entire system down due to tight coupling
- **Research Question:** Does async communication in microservices improve resilience/robustness vs. monolithic systems?
- **Method:** Chaos Engineering on two equivalent e-commerce systems

Related Work

- **Performance & scalability comparisons**: Microservices excel in distributed environments but degrade after certain instance count
- **Communication patterns:** Event-driven architectures show 19% faster response time and 34% lower error rate (Rahmatulloh et al.)
- **Quality attributes & design patterns:** Focus on availability, monitorability, security, testability
- **Gap:** No empirical fault tolerance comparisons under controlled failure conditions

Chaos Engineering

- **Definition:** Controlled fault injection under load to prove resilience
- **Key Questions:** Fail fast? Degrade gracefully? Cascade?
- **Our Approach:** Chaos Mesh for Kubernetes pod termination during high-load traffic

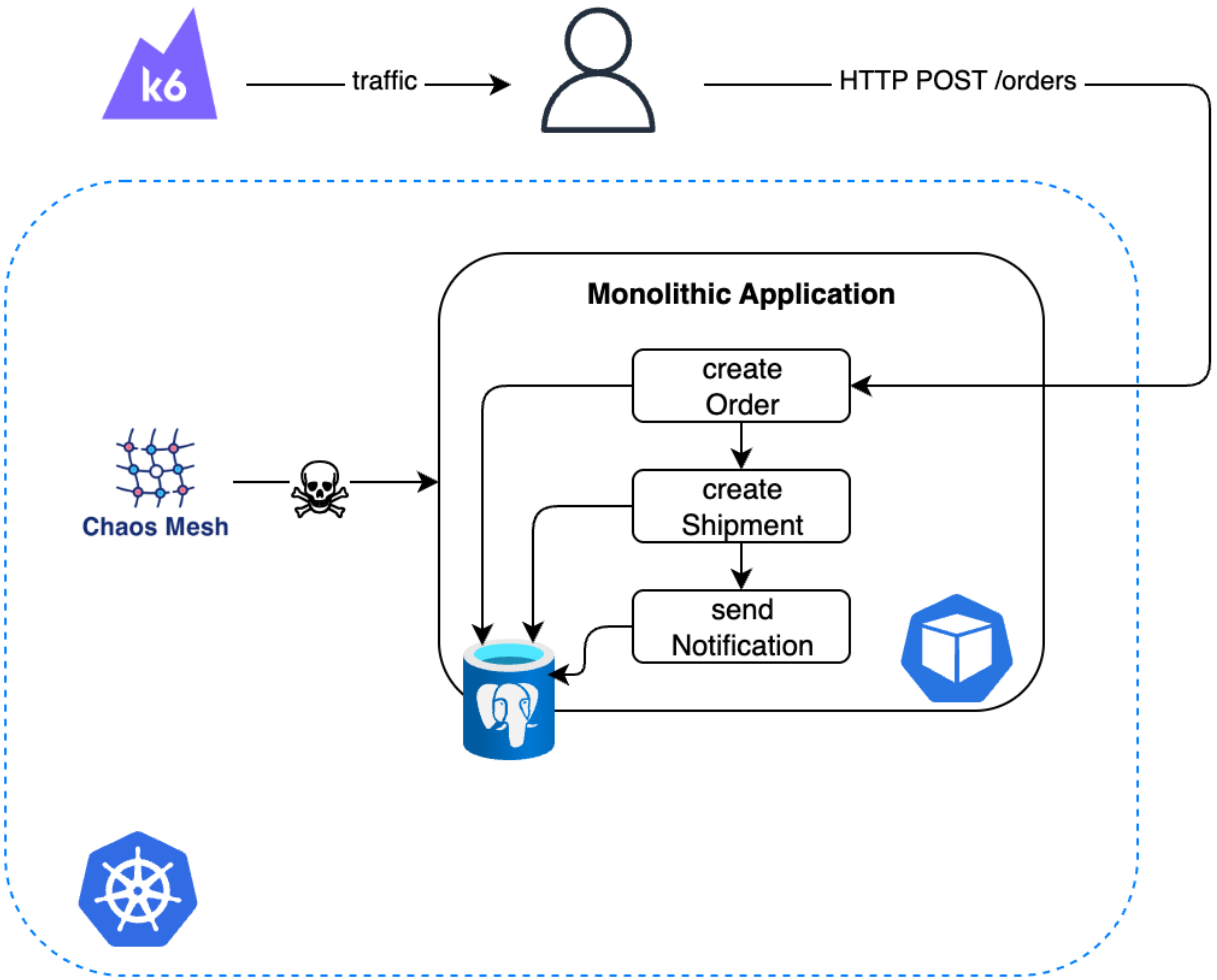
Methods

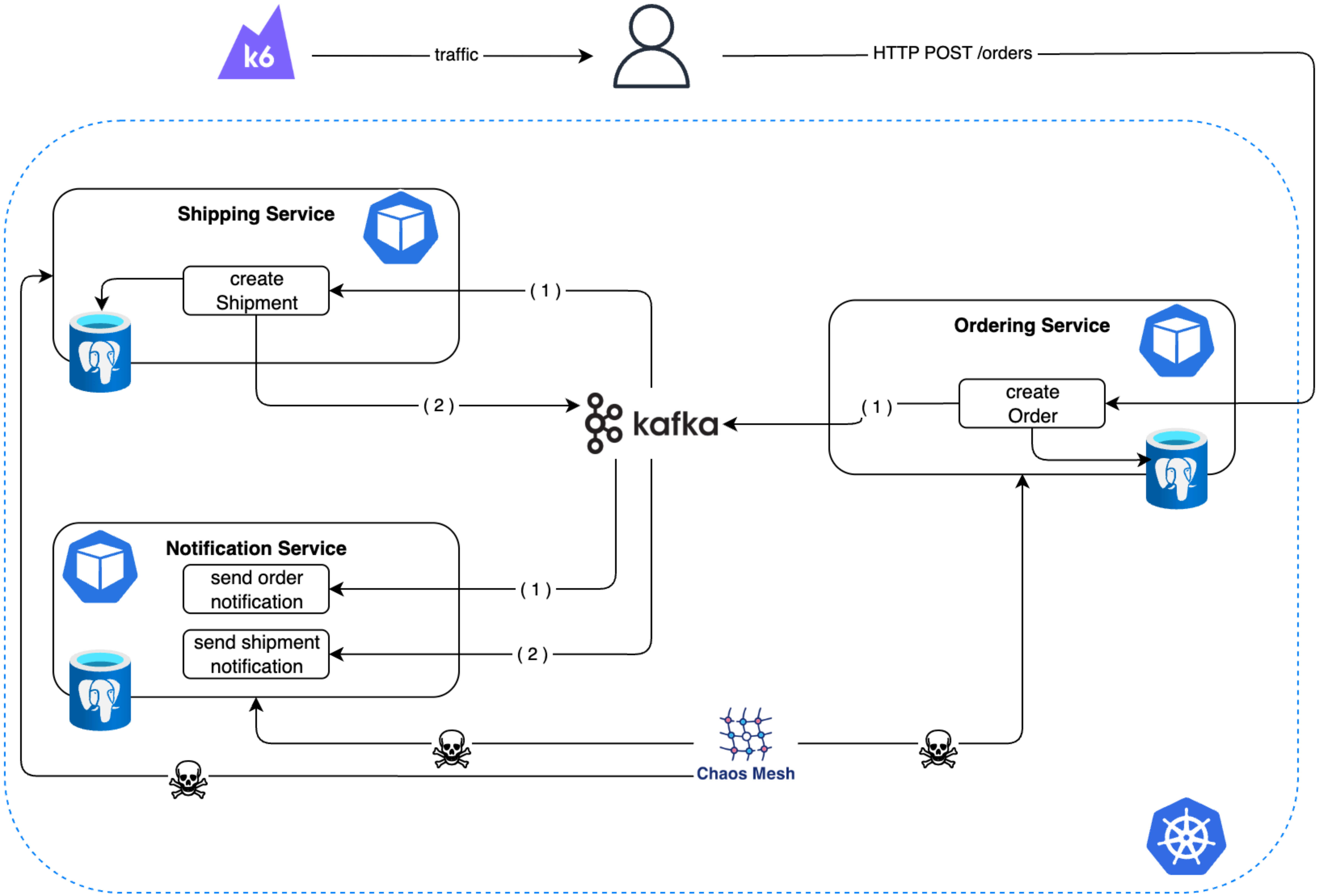
- **Systems:**
  - **Monolith:** Single Spring Boot app + 1 shared PostgreSQL database
  - **Microservices:** 3 independent services + Kafka + 3 dedicated databases
- **Same Business Logic:** Order → Shipment → Notification
- **3 Experiments:**

  #1: Baseline - 20 virtual users, 2 minutes, no failures

  #2: Entry-point failure - constant 10 req/sec, up to 40 virtual users, 2 minutes

  #3: Random internal failure - same load pattern as Experiment 2
- **Metrics:** Orders processed, failure rate, response time


k6
traffic
HTTP POST /orders
Monolithic Application
create Order
create Shipment
send Notification
Chaos Mesh


k6
traffic
HTTP POST /orders
Shipping Service
create Shipment
(1)
(2)
kafka
Ordering Service
create Order
(1)
Notification Service
send order notification
(1)
send shipment notification
(2)
Chaos Mesh

Results

- **Baseline Performance (No Failures):**
  - **Throughput:** 509.19 req/sec (microservices) vs 178.41 req/sec (monolith) = ~3× higher
  - **Response** Time: 39.18ms (microservices) vs 112.11ms (monolith) = ~3× faster
- **Entry-Point Failure Results:**
  - **Orders Processed:** 984 (microservices) vs 956 (monolith)
  - **Failure Rate:** 18.04% vs 20.35% = 2.3 percentage points lower
  - **Response Time:** 33.58ms vs 59.57ms = 43% faster
- **Random Internal Failure Results:**
  - **Failure Rate:** 7.88% (microservices) vs 20.35% (monolith) = 12.47 pp lower
  - **Response Time:** 24.87ms vs 59.57ms = 58% faster
- **Key Insight:** Kafka buffers work, services fail independently

Discussion

- **Proven Benefits:**
  - **Fault isolation:** Failures stay local, don't cascade
  - **Lower failure rates:** Up to 12 percentage points improvement
  - **Buffered resilience:** Kafka absorbs turbulence vs. synchronous propagation
- **Real Trade-offs:**
  - **Resource footprint:** ~3× higher operational overhead
  - **Complexity:** Multiple services + databases + Kafka cluster
  - **Skills required:** Distributed tracing becomes mandatory, not optional
  - **Operational costs:** Higher due to distributed infrastructure
- **Decision Framework:**
  - Critical systems (downtime = thousands/minute) → worth it
  - Internal tools/MVPs → stick with monolith

Conclusion

- **Anecdote Callback:**
  Event-driven microservices would likely have prevented the canteen outage

- **Key Numbers to Remember:**
  - ~3× throughput improvement (509 vs 178 req/sec)
  - ~3× faster baseline latency (39ms vs 112ms)
  - ~12 percentage points lower failure rate under random failures

WE ARE EU3DRES2
EUROPEAN UNIVERSITY

/Informatik & Security

/fh/// st.pölten
UNIVERSITY OF APPLIED SCIENCES

Requirements, not trends, should guide your architecture.