ISTANBUL SOFTWARE TESTING CONFERENCE

Exploring Chaos Engineering in Microservice Architectures

Burak Tuzlutaş
Ethem Utku Aktaş, PhD

#ISTC2026

Content

- Introduction
- Motivation
- Methodology
- Architecture
- Experiments
- Telemetry Analysis
- Lessons Learnt
- Conclusion and Feature Work

Introduction

Softtech is one of Turkey's largest software companies with domestic capital.

Founded in 2006 as a subsidiary of İş Bankası, Turkey's largest private bank.

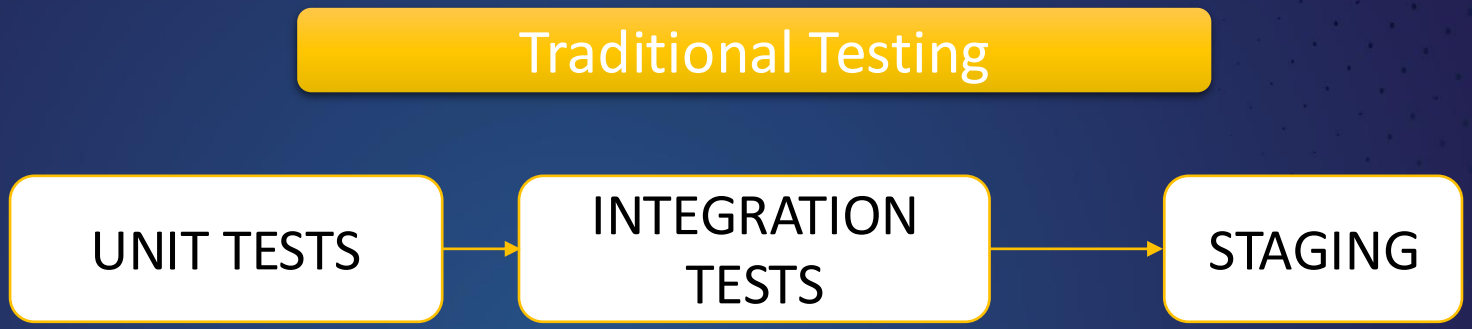
Headquartered in Istanbul, Softtech operates with more than 50 software teams.

Its primary client, İş Bankası, serves around 16 million digital customers, and Softtech maintains a portfolio of over 500 software products.

At Softtech, we aimed to integrate chaos engineering into our testing processes to proactively strengthen the resilience of our 500+ software products, ensuring that the services we deliver to 16 million digital customers through İş Bankası remain uninterrupted and reliable.

Introduction

From Reactive to Proactive: Chaos Engineering

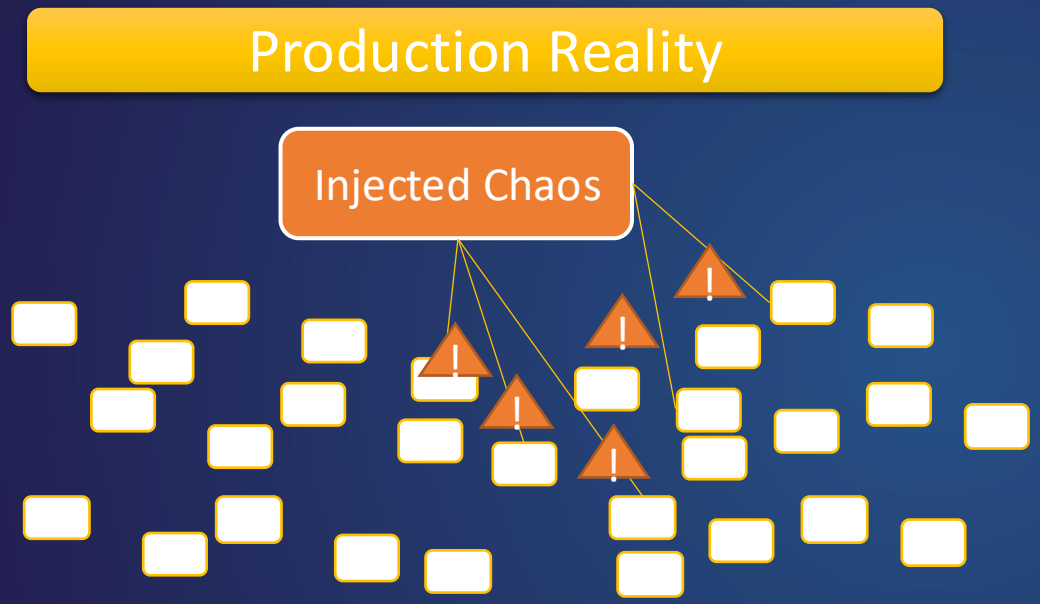


- **QA Gap:**

Traditional testing relies on isolated components and predefined scenarios, making it blind to real-world interactions, sudden traffic spikes, and cascading failures in production.

Introduction

From Reactive to Proactive: Chaos Engineering



- **Challenge: Illusion of Predictability**

Microservices complexity makes cascading failures unpredictable through conventional testing. We need to go beyond “Does it work?” and ask “How does it break?”

- **Approach: Controlled Failure Injection**

Proactively introduce real-world disruptions -like pod crashes or network loss- to test and strengthen self-healing capabilities.

- **Goal: Resilience by Design**

Expose weaknesses before disasters strike, ensuring zero downtime for critical user transactions.

Introduction

From Reactive Monitoring/Testing to Proactive Chaos

Dimension	Traditional QA	Chaos Engineering
Scope	Component-level functionality	Systemic socio-technical resilience
Paradigm	Reactive (Testing known, predefined paths)	Proactive (Uncovering unknowns)
Environment	Isolated, synthetic staging environments	Production or high-fidelity production-like environments
Metric Focus	MTBF (Mean time between failures – avoiding outages).	MTTR (Mean time to recovery – surviving and rapidly healing from outages)

Motivation

Chaos Engineering at Softtech

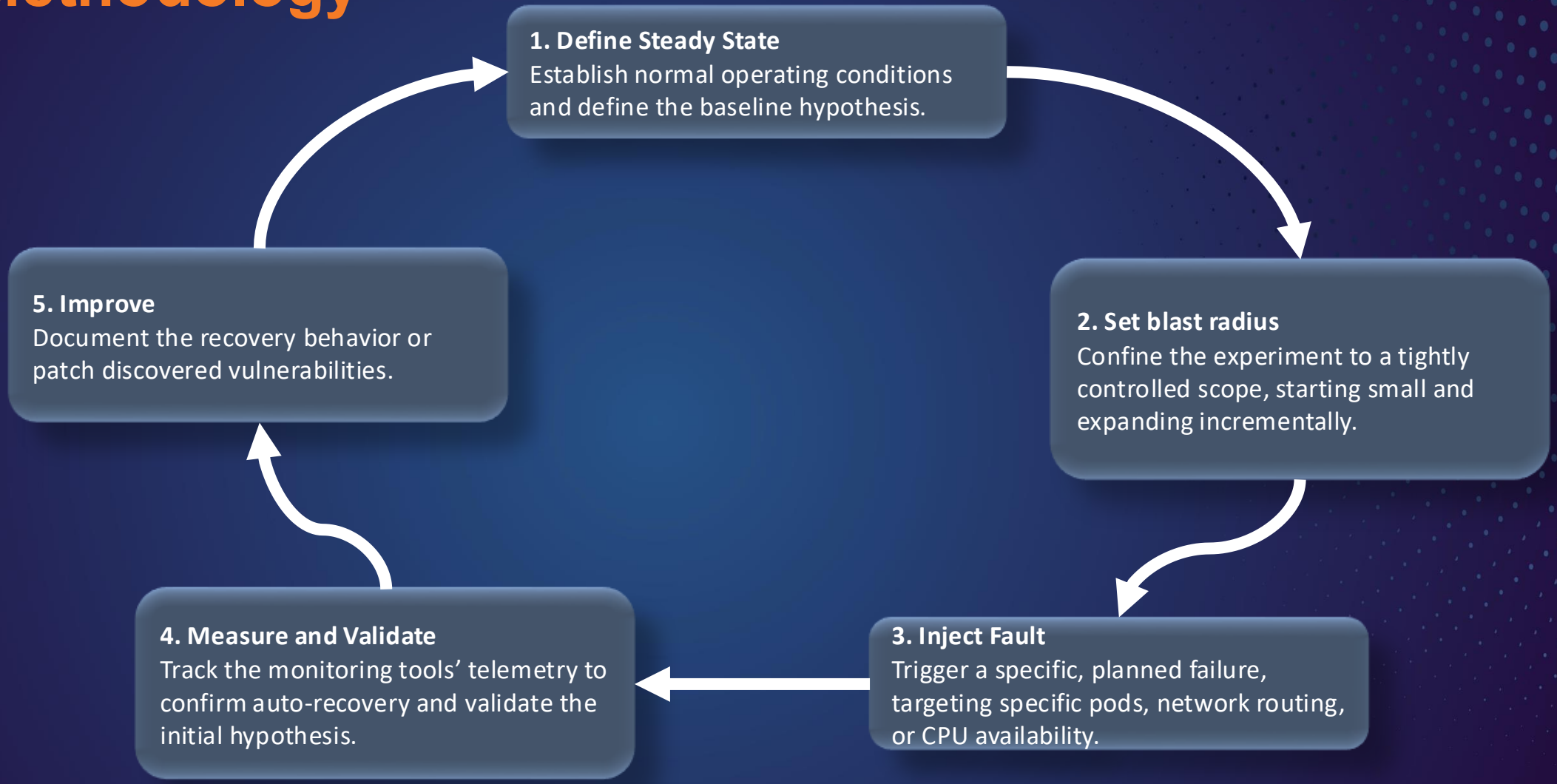
Plateau Framework

- A low-code microservice framework on Kubernetes
- Developed by Softtech to meet IsBank's needs and serve other customers
- Plateau platform is Kubernetes-based and central to Softtech's future architecture

Motivation

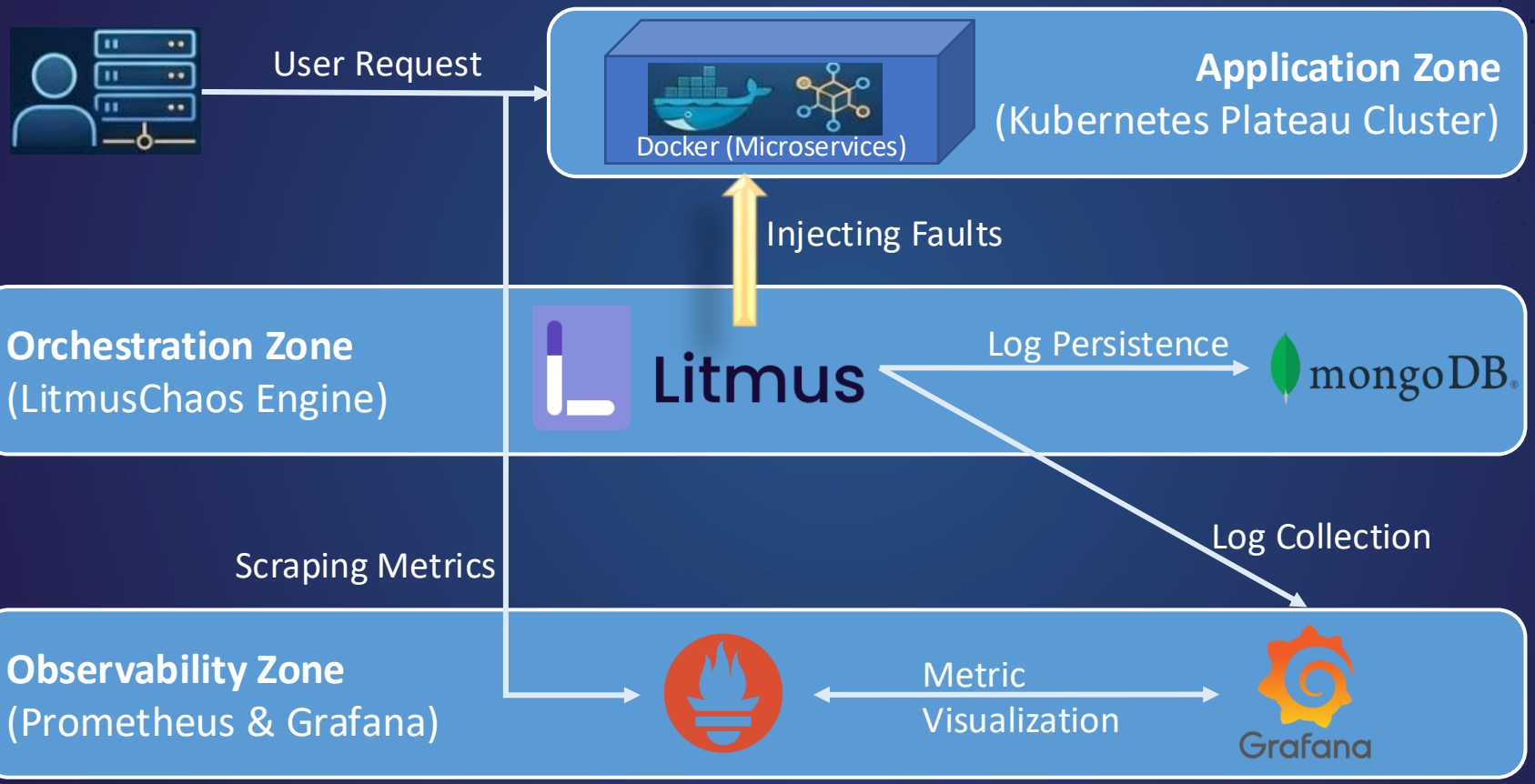
- Detecting unexpected errors in microservice architectures before they go live.
- To reduce the risk of outages and downtime.
- Making self-healing behavior measurable.

Methodology



Architecture

The Plateau Ecosystem

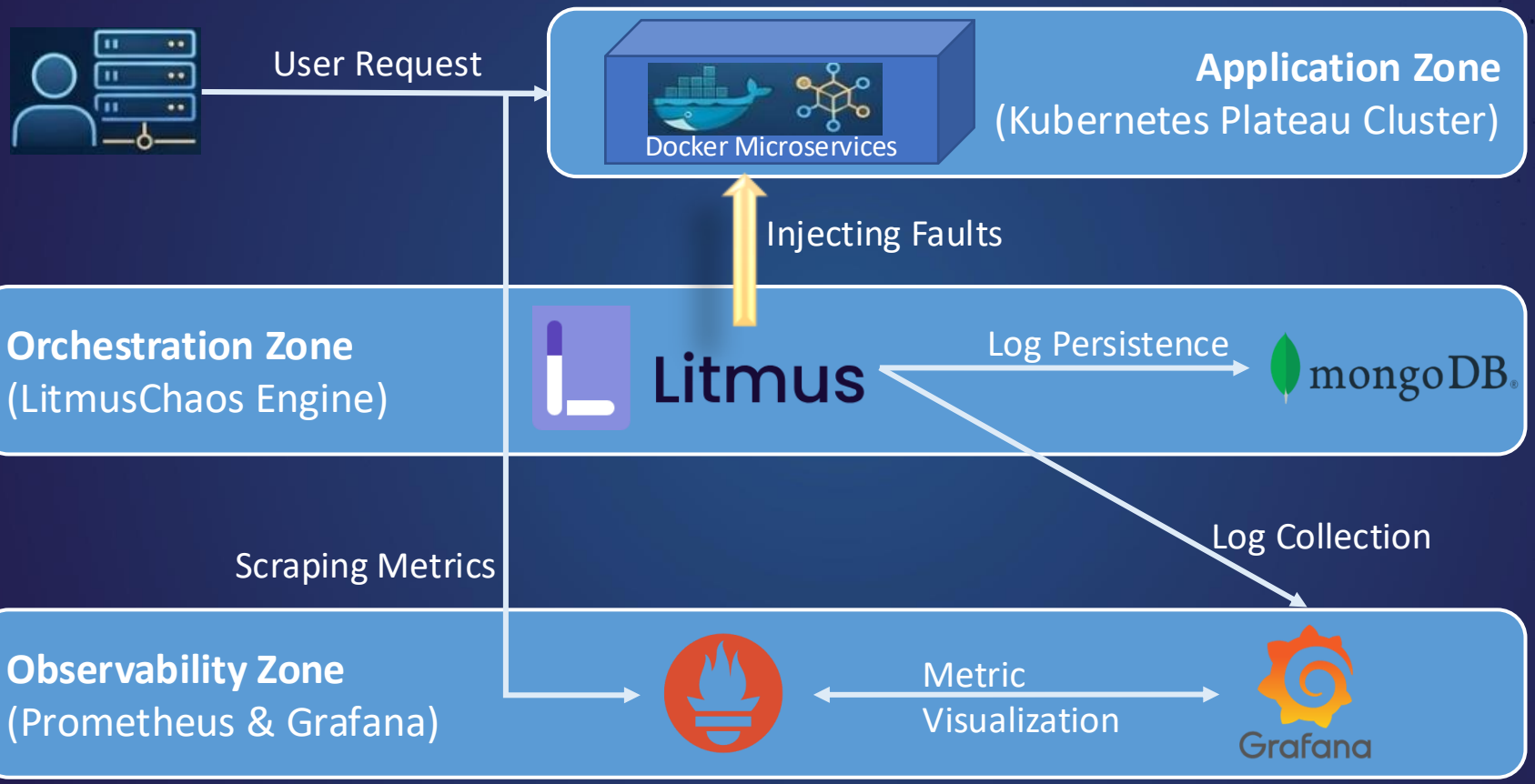


Orchestration and Deployment

LitmusChaos: Selected as the primary orchestration tool due to its Kubernetes-native capabilities, making it ideal for planning and executing experiments on the Plateau platform.

Helm Charts: Deployment was executed using Helm chart architecture rather than manual commands. This decision, made in coordination with the DevOps team, ensures easier operational management and standardization.

Architecture
The Plateau Ecosystem



Observability and Monitoring
To ensure real-time anomaly detection and metric tracking during experiments:

Prometheus & Grafana: Used for system-wide monitoring and real-time visualization of pod status and recovery.

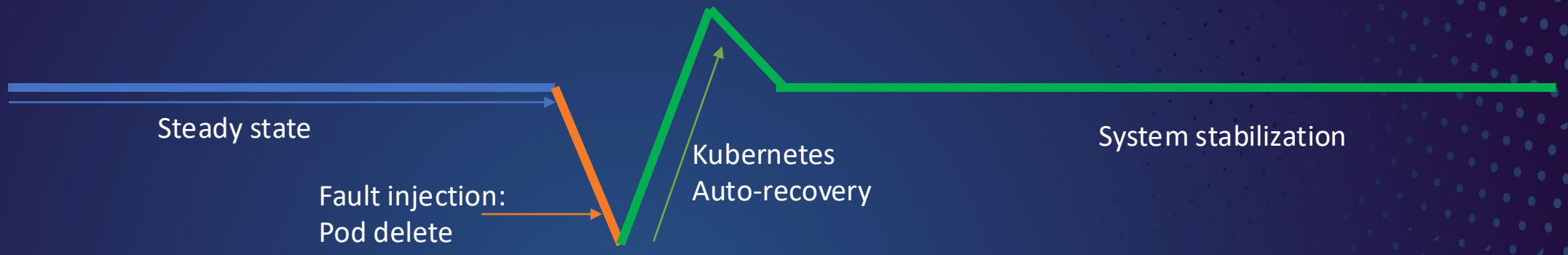
Kubernetes Dashboard: Utilized during tests to provide live evidence of pod termination and subsequent recreation.

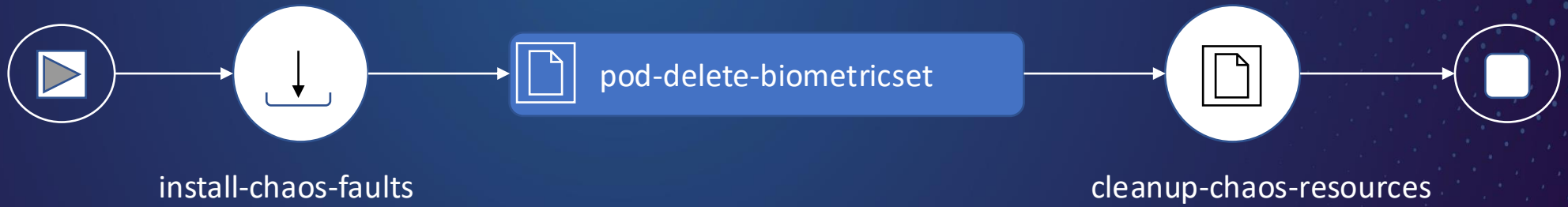
Experiments

Experiment / Service	Risk Tested	Injected Fault	Blast Radius	Observed Signal	Result	Action
Security Module	Authorization continuity	Pod Deletion + Network Corruption	Authorization & User Pods	Auth success rate, 5xx errors, pod readiness status	Successful	No action required
Operational Support Service	Agent component recovery	Pod Deletion	Agent Components	Ready replicas, restart count, Litmus score	Successful / Litmus score: 100%	Can be standardized
Financial Record / Onboarding	Tolerance to latency and CPU pressure	Network Latency + CPU Throttling	Isolated app pods	p95 latency, CPU usage, error rate	Tolerated (absorbed)	Monitor threshold
Biometric Verification	Service health after pod restart	Pod Deletion	User Pods	Probe, TLS, readiness events	Recovered	Adjust TLS / Probe settings

Telemetry Analysis

Scenario: Biometric Verification

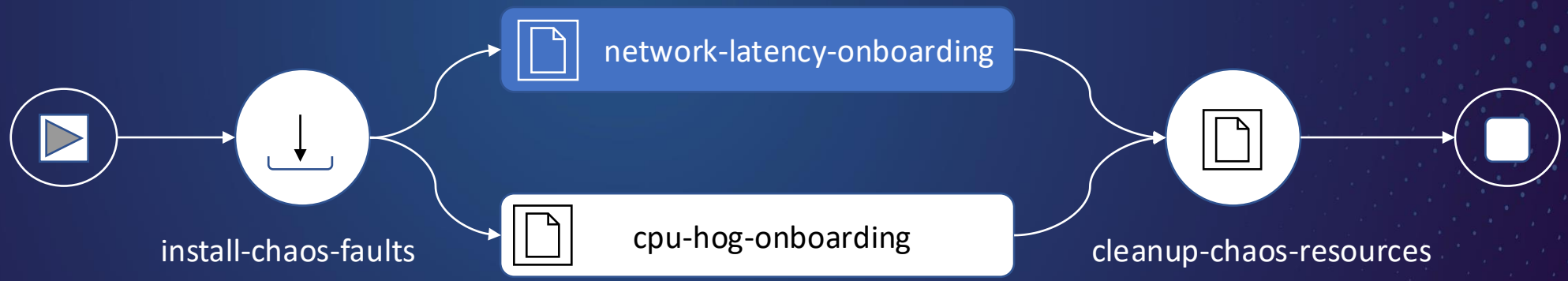




Service recovered after pod recreation. Telemetry indicated TLS / probe configuration adjustments were required.

Telemetry Analysis

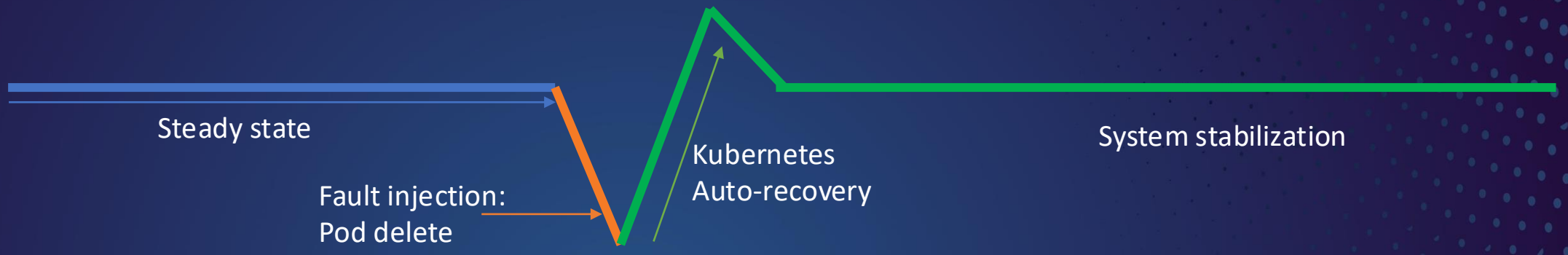
Scenario: Financial Record / Onboarding

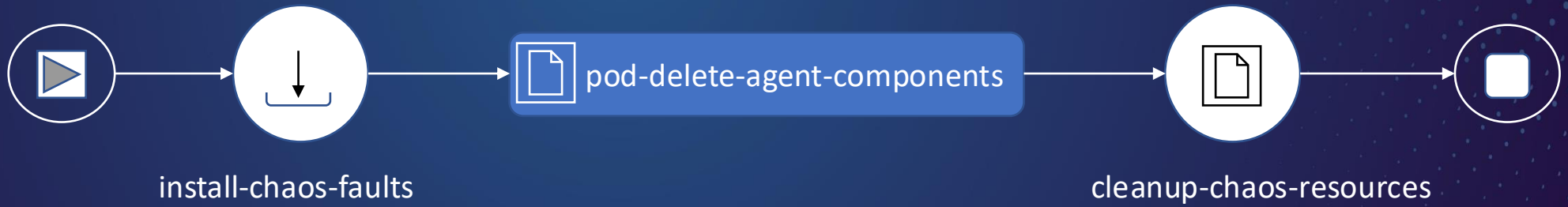


Metrics showed temporary stress under latency and CPU pressure, but the system absorbed the load and remained within expected thresholds.

Telemetry Analysis

Scenario: Operational Support

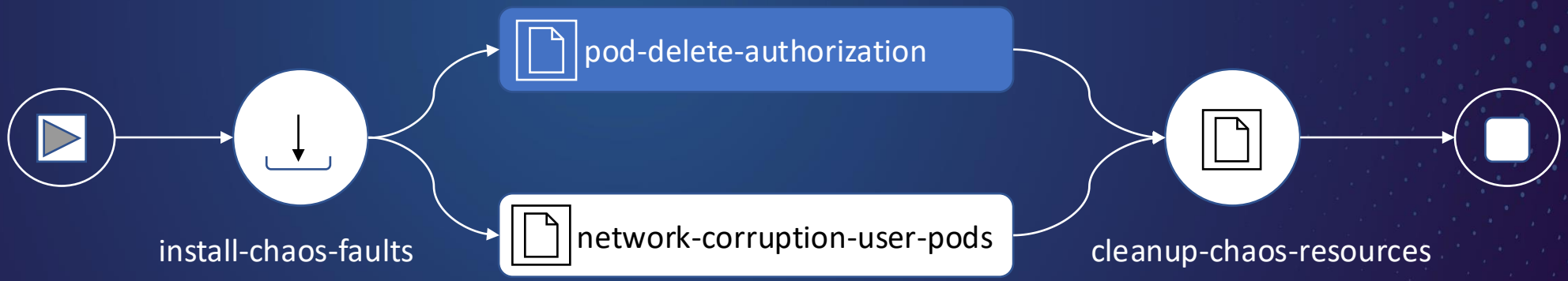




Live monitoring confirmed that agent components were recreated successfully and the resilience score remained at 100%.

Telemetry Analysis

Scenario: Security Module



Live monitoring confirmed continuous authorization and successful recovery under controlled pod delete and network corruption.

Lessons Learnt

- **The Illusion of Predictability:**
We demonstrated that systems must be proactively tested not only for how they function under normal conditions, but also for how they fail through cascading errors.

- **From MTBF to MTTR:**
Through chaos testing, we learned that success is not just about preventing outages (MTBF), but more critically about how quickly a system can recover and remain resilient during a crisis (MTTR).

- **Controlled Blast Radius:**
We experienced that chaos experiments are not random destruction, but a scientific process—starting small and gradually increasing scope in a controlled manner.

- **The Necessity of Live Observability:**
We realized that the effectiveness of self-healing systems cannot be proven without real-time metric monitoring (e.g., Prometheus/Grafana).

- **DevOps and Operational Alignment:**
We moved away from manual processes, standardized infrastructure deployments using Helm Charts, and significantly improved operational manageability.

Conclusion and Future Work

- **Successful “Self-Healing”:**

Chaos experiments involving network corruption, pod deletion, and CPU hog in live components demonstrated that Kubernetes' auto-recovery mechanisms function as intended.

- **Paradigm Shift:**

It has become clear that testing strategies must evolve from being solely focused on “avoiding failures” (MTBF) to a proactive model centered on “fast recovery and resilience” (MTTR).

- **Full CI/CD Automation:**

The goal is to integrate chaos injections -currently triggered manually- into the DevOps pipeline, enabling continuous and automated execution as part of CI/CD workflows.

- **Role-Based Access Controls (RBAC):**

Our objective is to enable product teams to run their own (self-service) chaos tests securely and with pre-approved permissions within their isolated namespaces using RBAC.

ISTANBUL
SOFTWARE
TESTING
CONFERENCE

Thank you!