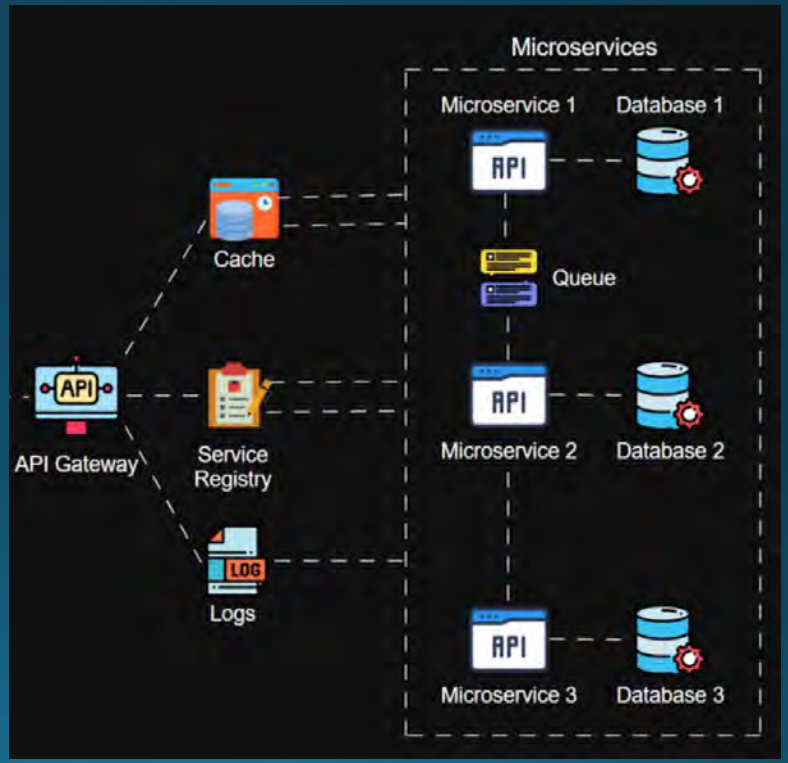
Chaos Engineering for Resilient Microservices

Muhammad Ahmad Saeed

Typical Microservices Architecture

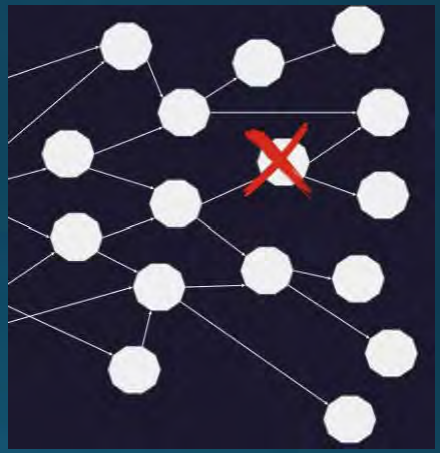
1. Services communicate with their databases

2. Services communicate with each other



Goals

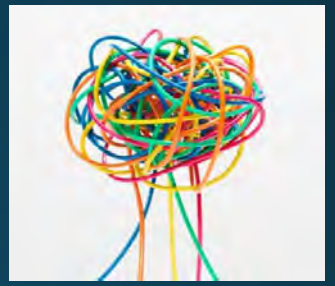
1. Uncover weaknesses
2. Improve system robustness
3. Handle unexpected failures gracefully



Why is Chaos Engineering Critical for Microservices?

1. **Distributed Complexity**: Multiple moving parts, failures in one service can ripple through the system.

2. **Network Dependencies**: Network communication is prone to latency, packet loss, and outages.



3. **Dynamic Scaling**: Often run in containerized environments (e.g., Kubernetes) that dynamically scale, making it harder to predict failure modes.

4. **Frequent Deployments**: Continuous deployment in microservices increases the risk of introducing bugs or regressions.

Key Areas to Test in Microservices

1. Service Failures
2. Network Issues
3. Dependency Failures
4. Load and Scalability
5. Data Consistency
6. Configuration Changes
7. Cascading Failures

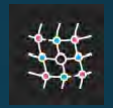
Principles of Chaos Engineering

1. Define a Steady State: Identify normal system behaviour using metrics.
2. Hypothesize About Failure: Predict how the system should behave on failure.
3. Introduce Controlled Chaos: Simulate failures in a controlled manner.
4. Observe and Measure: Monitor how the system responds.
5. Automate and Integrate into CI/CD: Continuously run chaos experiments.

Popular Chaos Engineering Tools

1. Chaos Monkey: Developed by Netflix, this tool randomly terminates instances in production to ensure systems can handle failures.

2. Chaos Mesh: A Kubernetes-native tool for chaos testing in cloud environments.



3. Gremlin: A platform for running chaos experiments across infrastructure, network, and application layers.

4. Toxiproxy: Simulates network conditions like latency and packet loss.

Real-World Case Studies of Chaos Engineering

Slack relies on a distributed system to serve **millions** of users.

Failure Testing: Slack conducts regular chaos experiments to test the resilience of its infrastructure and services.

Incident Response Drills: Slack uses chaos engineering to simulate incidents and practice its incident response processes.

Impact: Minimized downtime and maintained a reliable platform for its users.

Real-World Case Studies of Chaos Engineering

Microsoft – Azure Chaos Studio

- Managed service that uses chaos engineering to help measure, understand, and improve service resilience.

- Setup experiments, describe the faults to run and the resources to run against.

- organize faults to run in parallel or sequence, depending on your needs.

The Benefits of Adopting Chaos Engineering for Microservices

1. Financial benefit: stop significant financial losses by preventing prolonged outages
2. Technical benefit: reduce incidents, less firefighting
3. Customer benefit: Improved end-user experience

Best Practices for Chaos Engineering

1. Start Small: Begin with non-critical services before expanding to production.
2. Run Experiments in a Controlled Manner: Use feature flags.
3. Monitor Everything: Utilize observability tools
4. Automate and Integrate: Integrate into the DevOps pipeline.
5. Have a Recovery Plan: Ensure rollback and self-healing mechanisms exist.

Thank You