Designing for Failure

Strategies to Build Resilient, Always-On Services

Systems today are

Massively distributed, spanning multiple data centers, clouds, or even continents

Highly integrated, relying on third-party APIs, cloud providers, and external services

Under constant demand, with users expecting near-instant responses, even during outages.

Redundancy: Building Backup Systems

Infrastructure Redundancy

Data Redundancy

Service Redundancy

Failover Mechanisms

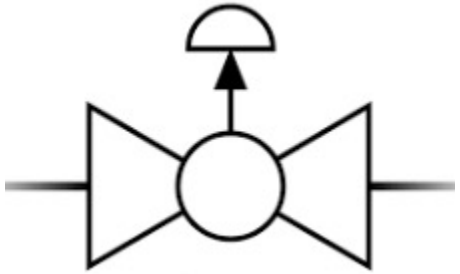
Automatic Failover

Manual Failover

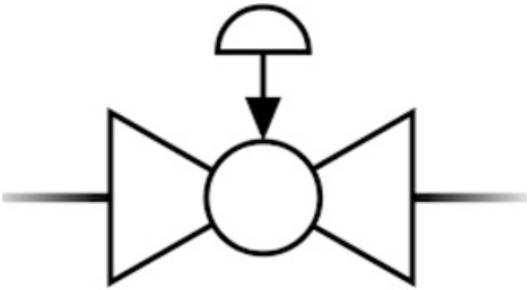
Challenges and best practices

- Latency during Failover
- Data consistency
- Testing failover scenarios
- Human error in Manual failover
- False positives
- Observability and monitoring

Graceful Degradation



Fail-Open



Fail-Close

Challenges and best practices

- Dependency Mapping
- Performance Optimization
- User Communication
- Testing degraded states
- Plan degradation early
- Leverage observability
- Iterate and improve

Graceful Shutdown

- Connection Draining
- Health Check Signaling
- State Persistence
- Timeouts and Grace Periods

Chaos Engineering

Failure Injection

Controlled Experiments

Resilience Metrics

Challenges and how to overcome them

- Managing risk
- Lack of observability
- Scaling chaos engineering

Circuit Breakers

Closed state

Open state

Half-open state

Benefits of Circuit breakers

Failure solation

Improved recovery

Beter user experience

Enhanced observability

Challenges and how to overcome them

- **Setting the right threshold**
- **Balancing failures and recovery**
- **False positives**

Automated Recovery Mechanisms

Auto-Scaling: Adjust resources dynamically to handle traffic spikes.

Self-Healing: Restart failed components automatically.

Rollback Strategies: Revert to stable versions after failed updates.

Conclusion

- Redundancy
- Failover Mechanisms
- Graceful Degradation
- Graceful Shutdown
- Chaos Engineering
- Circuit Breakers
- Automated Recovery

Thank you!

https://www.linkedin.com/in/abhishekvajarekar/