



Building Resilient Systems: A Guide to High Availability Engineering

Designing systems that stay up when things go down

Our Journey Today

1

Understanding High Availability

What makes systems reliable and why it matters

2

Core Techniques

Redundancy, load balancing, and failover strategies

3

Measuring Resilience

Key metrics like MTTR and how to track them

4

Planning for Growth

Capacity planning and scaling strategies

5

Chaos Engineering

Breaking things on purpose to build stronger systems

6

Resilience by Design

Building systems that withstand the unexpected

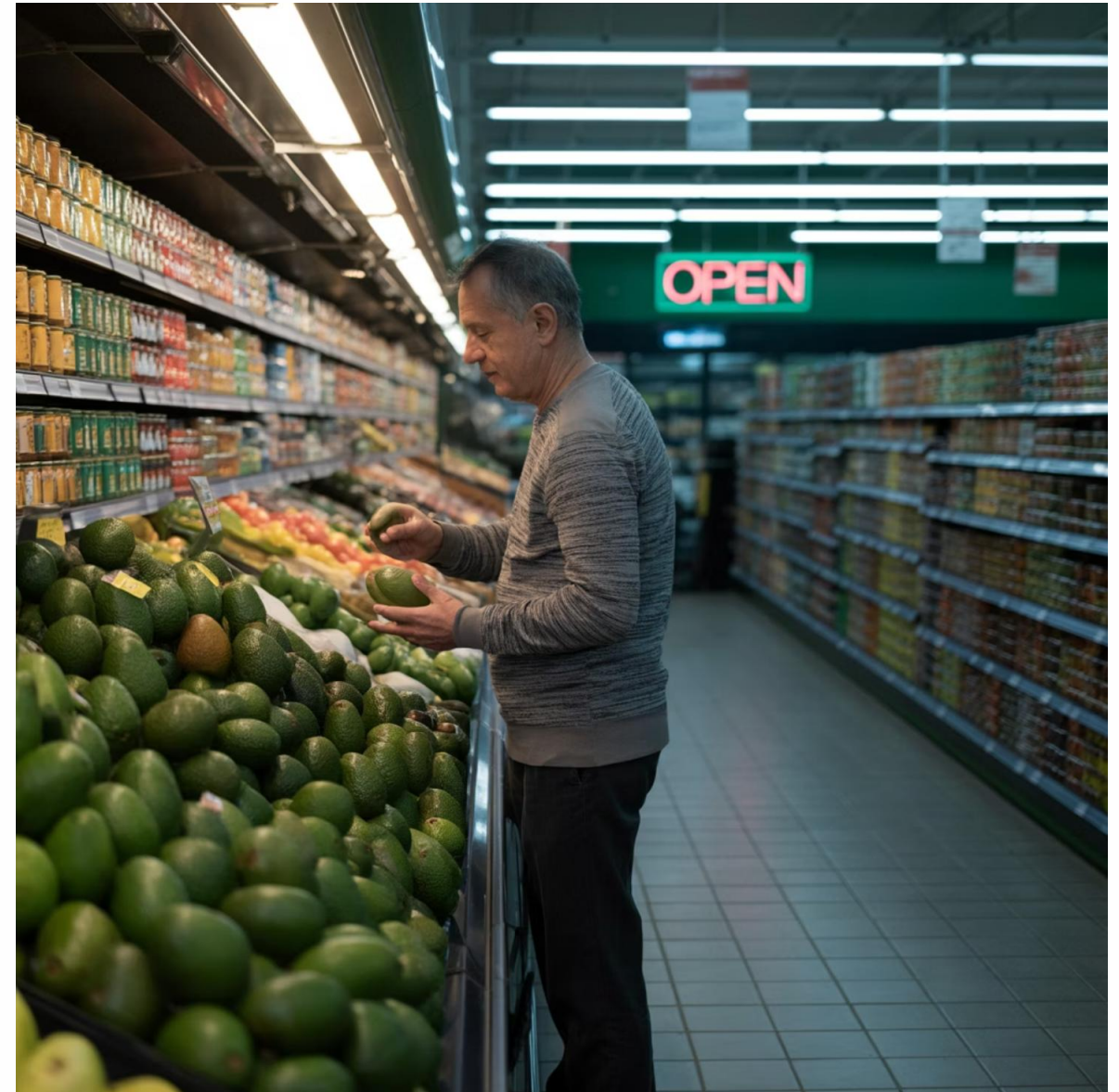
What is High Availability?

High availability is a system's ability to operate continuously without failing for a designated period of time.

Think of it like your local grocery store:

- You expect it to be open when you need it
- You need access to essential items
- You don't want to drive across town when something's unavailable

Just like customers walking away when a store is unexpectedly closed, users abandon services that aren't reliable.



The Cost of Downtime

\$5,600

Per Minute

Average cost of IT downtime

\$84K

Per Hour

For small businesses

\$540K

Per Hour

For enterprise organizations

Beyond direct financial impact, downtime damages reputation, customer trust, and employee productivity. For Amazon, a one-hour outage can cost over \$13 million in sales.

Measuring Availability: The 9s

System availability is typically measured in "nines" - the percentage of time a system is operational.

99% ("Two Nines")

3.65 days of downtime per year

99.9% ("Three Nines")

8.76 hours of downtime per year

99.99% ("Four Nines")

52.56 minutes of downtime per year

99.999% ("Five Nines")

5.26 minutes of downtime per year



Each additional "nine" is exponentially more difficult and expensive to achieve!



Real-World Example: Netflix

Netflix serves over 221 million subscribers worldwide and cannot afford significant downtime. Their high availability strategy includes:

- Global content delivery network with redundant servers
- Microservices architecture where failures are isolated
- Chaos engineering practice (created Chaos Monkey)
- Regional failover capabilities

Result: Even when parts of their system fail, users can still watch their favorite shows.

Core Techniques for High Availability

The building blocks of resilient systems

Redundancy: Eliminating Single Points of Failure

Redundancy means having backup components ready to take over when primary components fail.

Think of it like:

- Having a spare tire in your car
- Carrying both a credit card and cash
- A restaurant with multiple chefs who know all recipes



In systems design, we implement redundancy at multiple levels:

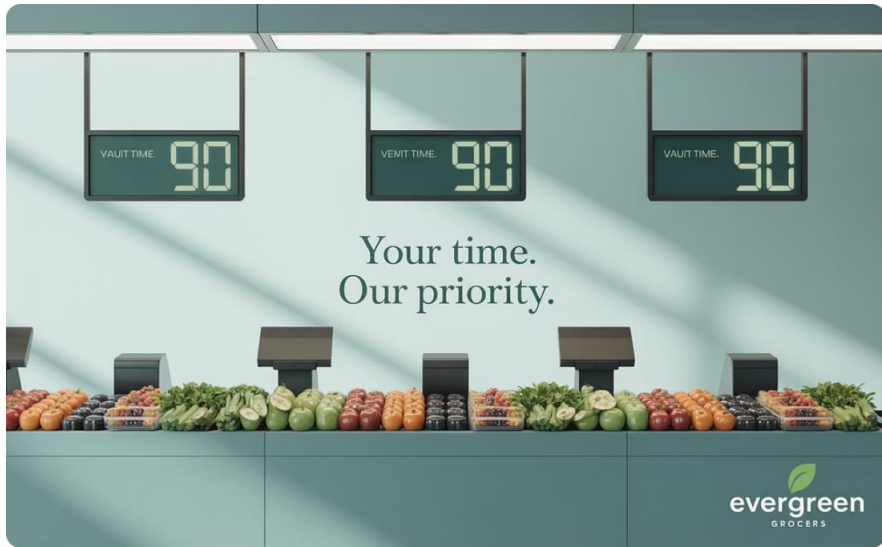
- Hardware (servers, disks, power supplies)

Load Balancing: Distributing the Work



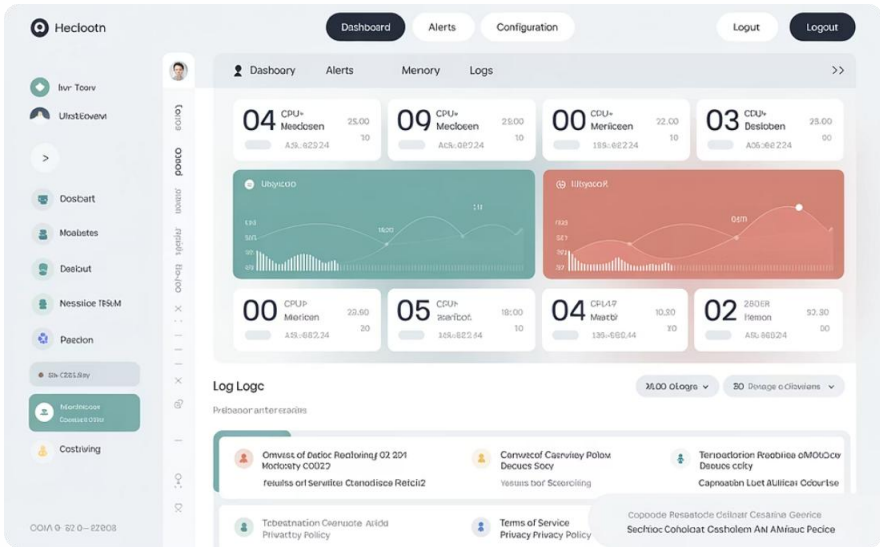
Traffic Distribution

Directs requests to different servers based on availability and current load, just like a traffic officer directing cars to less congested roads.



Even Workload

Prevents any single server from becoming overwhelmed, similar to a store opening more checkout lanes during busy periods.



Health Monitoring

Automatically detects when servers fail and stops sending traffic to them, like a hospital routing patients away from full departments.

Failover Techniques: The Backup Plan



Example: When your GPS loses satellite signal, it might switch to using cell towers for location data until the signal returns.



Real-World Example: Banking ATM Networks

Banks implement high availability for their ATM networks:

Redundancy

- Multiple ATMs in high-traffic areas
- Backup power systems
- Redundant network connections

Load Balancing

- Transaction routing across multiple backend servers
- Geographic distribution of processing centers

Failover

- Automatic switching to backup processing centers
- Offline transaction capabilities

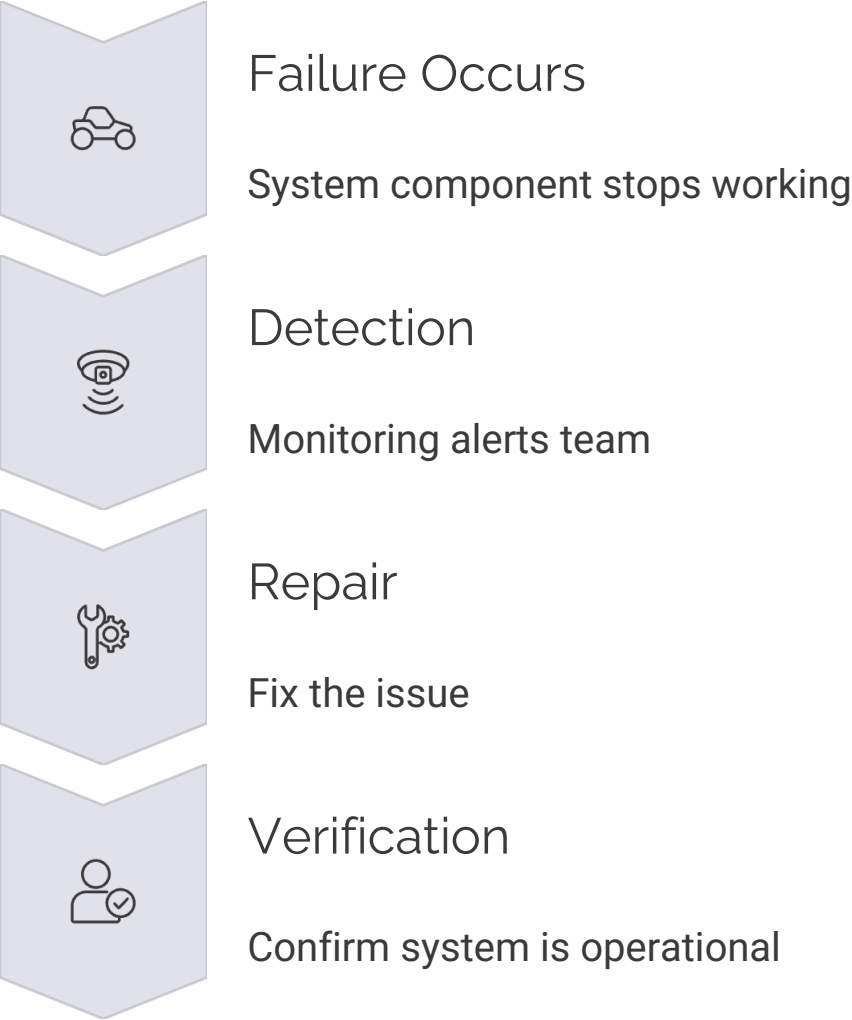
Result: You can still withdraw cash even during partial system outages.

Measuring Resilience

You can't improve what you don't measure

Key Metrics: Mean Time To Recovery (MTTR)

MTTR measures how quickly your system can recover from failure.



Lower MTTR = Better customer experience



Real-world example: When a power outage affects a hospital, their backup generators should kick in within seconds (very low MTTR).

Other Important Metrics

Mean Time Between Failures (MTBF)

The average time between system failures. Like a car that runs for 100,000 miles before needing major repairs.

Mean Time To Detect (MTTD)

How quickly you notice something is wrong. Like a smoke detector alerting you to a fire before it spreads.

Recovery Point Objective (RPO)

Maximum acceptable amount of data loss. Like how much work you'd lose if your computer crashed before you saved.

Recovery Time Objective (RTO)

Maximum acceptable downtime. Like how long a hospital can operate without its electronic records system.

Real-World Example: E-commerce Checkout

An online retailer's checkout system experiences intermittent failures:

Before Improvements

- MTTD: 15 minutes
- MTTR: 45 minutes
- Customer impact: High (lost sales)

After Improvements

- MTTD: 30 seconds
- MTTR: 5 minutes
- Customer impact: Minimal

How They Did It

- Added real-time monitoring dashboards
- Implemented automated recovery procedures
- Created redundant payment processing pathways
- Designed graceful degradation (fallback options)



Planning for Growth

Building systems that scale with demand

Capacity Planning: Predicting Future Needs

Capacity planning ensures your system can handle growing demands without failing.

Think of it like planning a wedding:

- How many guests will attend?
- How much food and drink to prepare?
- What size venue is needed?
- How many staff to hire?



Good capacity planning looks at:

- Current usage patterns

Scaling Strategies: Vertical vs. Horizontal

Vertical Scaling

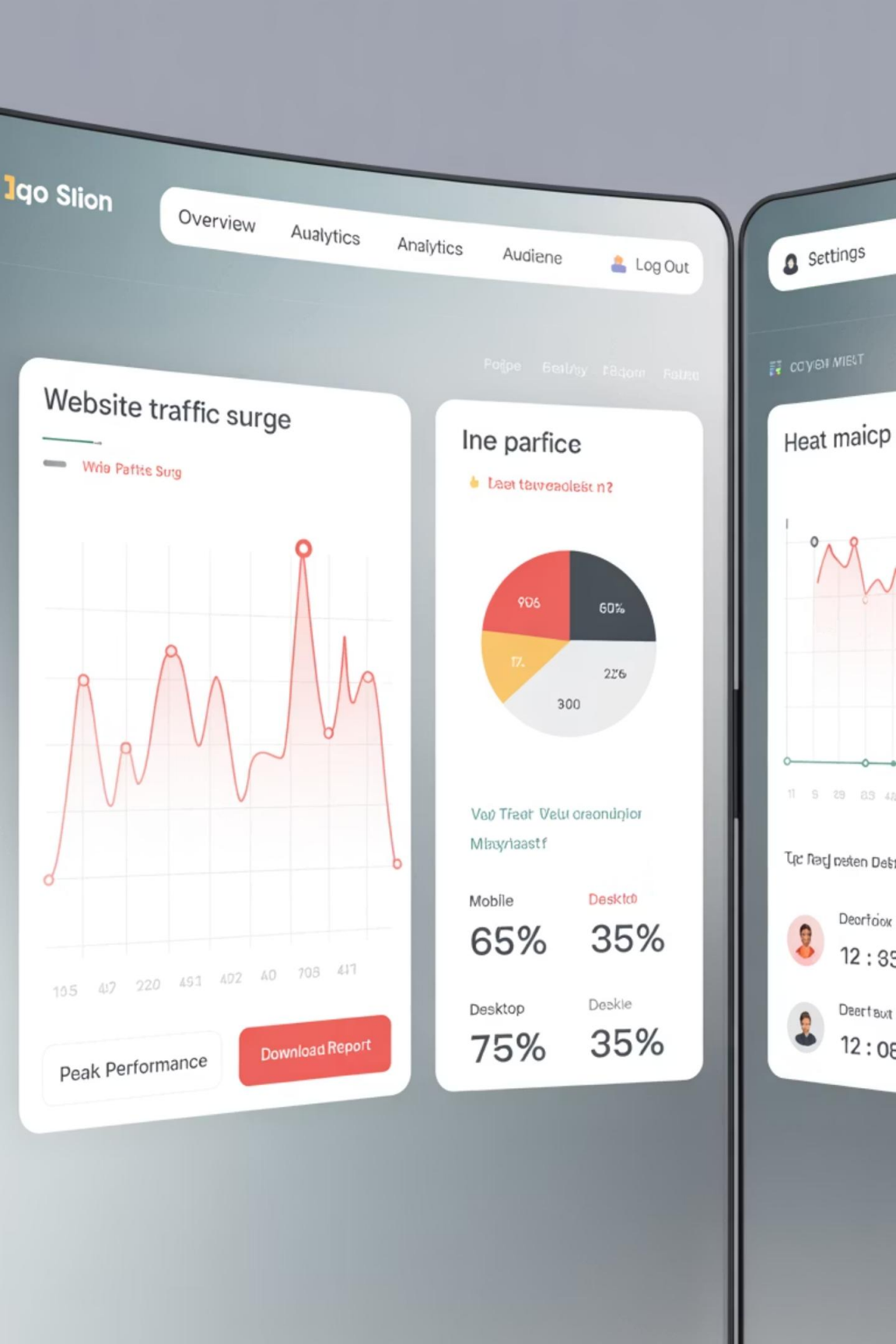


Adding more resources (CPU, RAM, storage) to existing machines

Horizontal Scaling



Adding more machines to share the workload



Real-World Example: Black Friday Shopping

E-commerce sites prepare for Black Friday traffic spikes:

Analyze Historical Data

Review past years' traffic patterns and sales volumes

Pre-Scale Infrastructure

Increase server capacity 3-5x normal levels before the event

Implement Auto-Scaling

Configure systems to add more resources automatically as traffic increases

Test at Scale

Run load tests simulating peak traffic conditions

Chaos Engineering

Breaking things on purpose to build stronger systems

What is Chaos Engineering?

Chaos Engineering is the practice of intentionally injecting failures into your system to test its resilience.

It's like:

- A fire drill to practice emergency procedures
- Stress testing a bridge before opening it to traffic
- Training soldiers with live-fire exercises

"Chaos Engineering is the discipline of experimenting on a system in order to build confidence in the system's capability to withstand turbulent conditions in production."

— Principles of Chaos Engineering



Principles of Chaos Engineering

Start in Test Environment

Begin experiments in non-production environments to minimize customer impact.

Contain the Blast Radius

Limit the potential damage of experiments by carefully controlling their scope.

Build a Hypothesis

Clearly define what you expect to happen when you introduce a failure.

Measure Everything

Collect detailed data about system behavior during experiments.

Automate Experiments

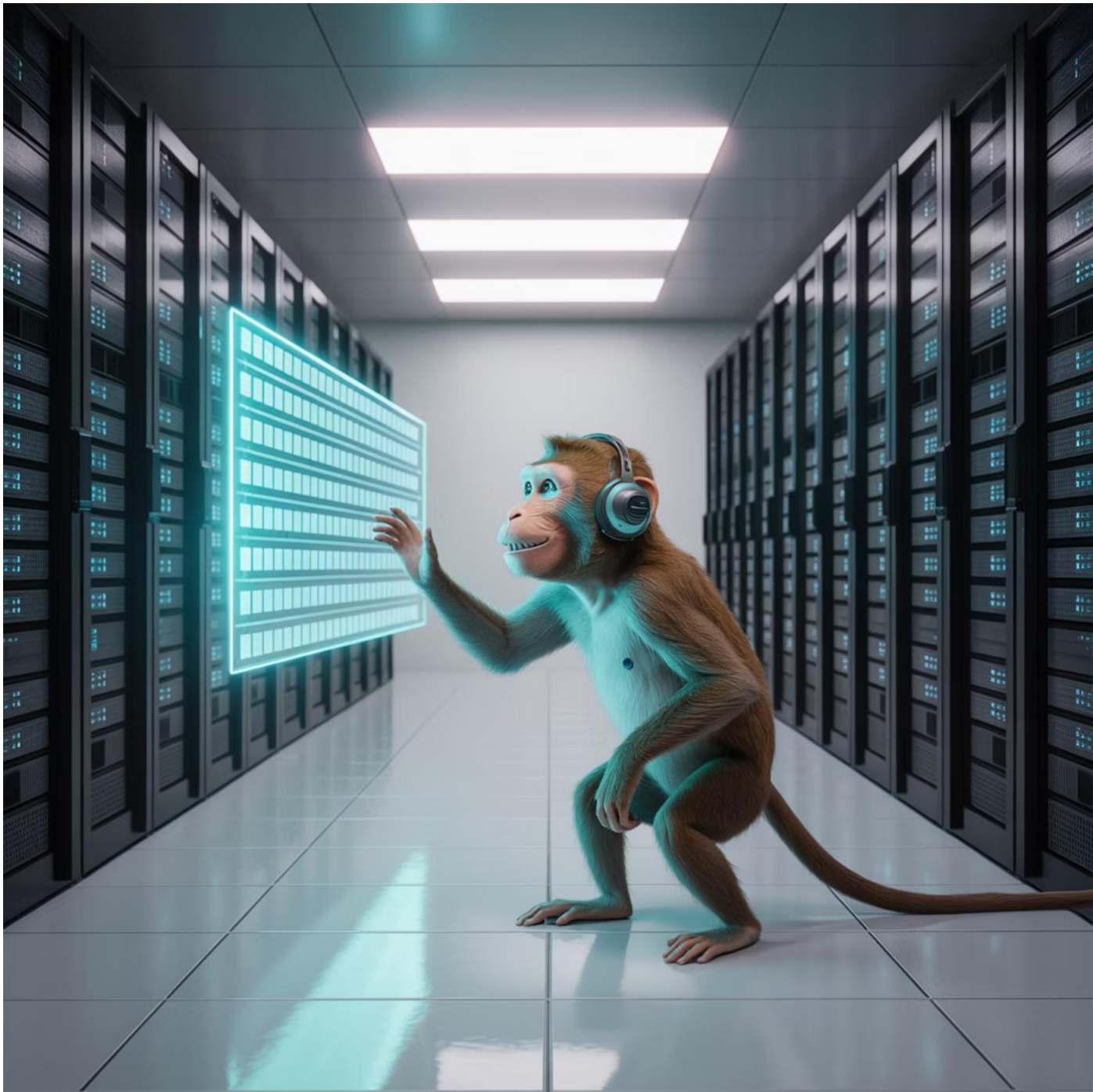
Create repeatable, consistent tests that can run regularly.

Run in Production

Eventually, test in real environments to ensure true resilience.

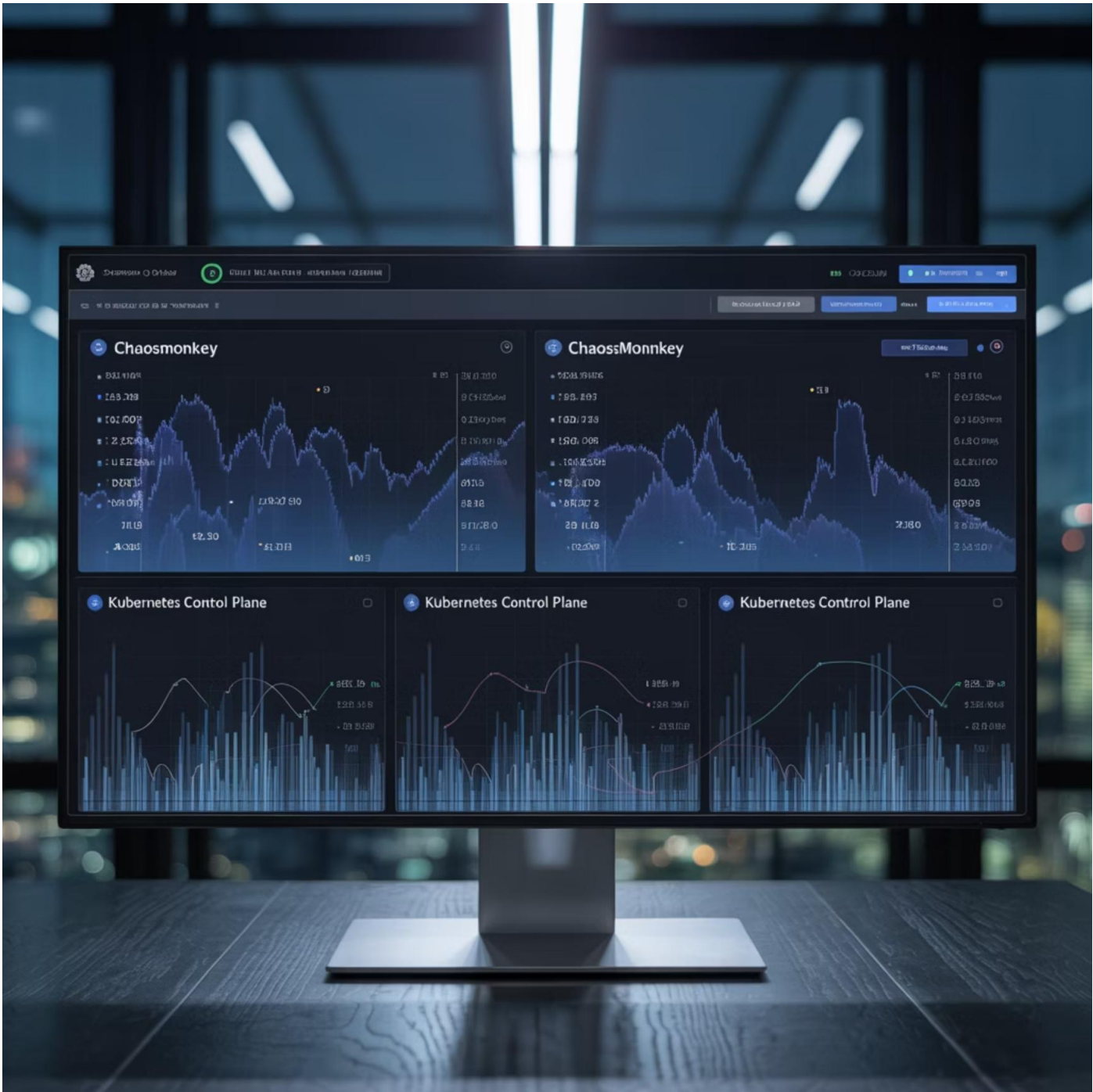
Chaos Engineering Tools

Chaos Monkey



Developed by Netflix, used to test system resilience.

LitmusChaos



Kubernetes-native chaos engineering tool.

Planning Chaos Experiments Safely

1 Define Steady State

Establish what "normal" looks like for your system before introducing chaos.

2 Form a Hypothesis

"If we terminate database connections, the application will automatically reconnect within 5 seconds."

3 Plan the Experiment

Define the scope, timeline, metrics to monitor, and abort conditions.

4 Notify Stakeholders

Make sure everyone knows an experiment is happening to avoid confusion.

5 Run the Experiment

Introduce the planned failure while closely monitoring the system.

6 Analyze Results

Did the system behave as expected? What weaknesses were discovered?

7 Fix Weaknesses

Strengthen your system based on what you learned.



Real-World Example: GameDay at Amazon

Amazon regularly conducts "GameDay" exercises where they deliberately cause failures in their production systems.

Example Scenario	Expected Behavior	Results & Learnings
Simulate the complete failure of a data center right before Prime Day (their busiest shopping event).	<ul style="list-style-type: none">• Automatic traffic routing to other data centers• No customer-facing disruption• Degraded but functional internal tools	<ul style="list-style-type: none">• Identified bottlenecks in failover processes• Improved monitoring for regional failures• Enhanced documentation for emergency procedures

Resilience by Design

Building systems that withstand the unexpected

Design Principles for Resilient Systems



Isolation

Failures in one component shouldn't cascade to others. Like firebreaks in a forest that prevent a small fire from spreading.



Modularity

Build systems from independent components that can be replaced or upgraded. Like modular furniture that can be reconfigured.



Graceful Degradation

When parts fail, maintain core functionality. Like a car that can still drive safely with a flat tire at reduced speed.



Loose Coupling

Components interact through well-defined interfaces without depending on each other's internal details. Like standardized electrical outlets.



Simplicity

Simpler systems have fewer points of failure. Like a bicycle being more reliable than a complex sports car.



Retry with Backoff



Automatically retry failed operations with increasing delays. Like waiting longer between attempts to call someone who isn't answering.

Circuit Breakers: Preventing Cascade Failures

Circuit breakers detect when a component is failing and stop sending requests to it.

Just like electrical circuit breakers in your home that cut power when there's a dangerous overload.



-  Closed
Normal operation: requests flow through
-  Open
After failures: blocks requests



- Benefits:
- Prevents overwhelmed services from completely crashing

Real-World Example: Airline Booking Systems

Modern airline reservation systems handle millions of bookings daily and use multiple resilience strategies:

Regional Isolation

Booking systems for different regions operate independently, so an outage in Asia doesn't affect flights in North America.

Multi-Channel Access

Customers can book through website, mobile app, phone, or in-person, providing multiple pathways when one fails.

Cached Data

Flight schedules and pricing are cached locally, allowing basic information to be displayed even when backend systems are unavailable.

Queue-Based Processing

Booking requests are queued, ensuring they're not lost during system disruptions and can be processed when systems recover.



Key Takeaways



Design with Redundancy

Eliminate single points of failure at every level



Measure What Matters

Track MTTR and other key metrics to identify improvements



Plan for Growth

Build systems that can scale with increasing demand



Test Through Chaos

Break things intentionally to find weaknesses

Building resilient systems isn't about preventing all failures—it's about designing systems that continue working despite inevitable failures. Start small, focus on the most critical components first, and gradually build a culture of resilience throughout your organization.