Date _____ Day _____

CLOUDFLARE OUTAGE - 18th, Nov 2025

• Root Cause

The outage was triggered by a change to database permission. That change caused a metadata query to return duplicate column entries. This resulted in faulty auto-generated feature file that ~~match~~ was much larger than expected

Normally the feature file is about around 60 features but due to duplicate database entries it grew beyond 200 features, exceeding the hardcoded run time limit of in the system

This faulty autogenerated file was used by Cloudflare Bot Mitigation service. When the oversized file was loaded, it exceeded the allowed limit and triggered an unhandled run time error, causing server to crash.

• Design Problems

① Tight Coupling of Service

↳ Many cloudflare services depended directly on Bot Mitigation data. When the service failed dependent services had no fallback behaviour and crashed.

Date → Cloudflare has massive redundancy Every redundant sys running same buggy code Day

(2) Homogenous Global Deployment

↳ The Cloudflare has the same software and configs deployed accross its overall network, so the bug affected the entire global network at once

(3) Lack of Configuration Validation

↳ The configuration file had no restrict size validation or safety checks before being loaded into production system

(4) Synchronous Dependencies

↳ Services expected bot mitigation data to be always available, instead of handling its absence gracefully

- **How to Overcome**

→ Reduce tight coupling by allowing services to operate with default or cached behaviour

→ Use staged deployments instead of global rollouts

→ Add config validations, limits & sanity checks

→ Design services to be eventually consistent rather than strictly synchronous

→ ~~Design~~

Date _____ Day _____

- **Steps Through which we can avoid these Fcilures**

① Treating Configuration as Code
↳ Use vession control, automated testing, size validations and rollback mechanisms

② Design For Graceful degradation
↳ Services should continue operating with reduced functionality when dependencies fail

③ ~~Fndr~~ Introducing Operation diversity
↳ Avoid running identical code & configs everywhere at sametime

④ Improve Observability
↳ Use per-service and per-datacenter metrics to detect issues easly and reduce diagnosis time

⑤ Progressive Rollouts – region by region deployment

- **Lessons Learnt**

↳ Configuration is as dangerous as code, must be tested
↳ Redundancy alone doesnot gurantee resilience if all sys fail in the same way
↳ Real word env expose edge cases that testing cannot always predict

Date _____ Day _____

↳ Critical sys with large blast radius require extra monitoring.

↳ Sys should be design to degrade gracefully, not collapse completely.

CLOUDFLARE OUTAGE – 5th December 2025

- **Root Cause**

↳ The root cause was an internal configuration change made to the Cloudflare's Web Application Firewall (WAF) logic while rolling out a security fix for a critical React Server Component vulnerability. Engineers increased the HTTP request body buffer size from 128 KB → 1 MB [done to analyze malicious payloads] and then disabled an internal "WAF rule testing tool" using Cloudflare's global configuration sys. This global change propogated instantly to all servers. On Older FL1 Proxy, this caused a runtime error "attempt to index field 'execute' (a nil value)" because the code assumed a rule field always existed. As a result many requests failed with HTTP 500 errors.

→ tool was disabled bcz that tool doesn't support the increased payload change. This tool was an internal testing tool and didnot affect the original traffic so they disabled it.

Date _____ Day _____

- **System Design Problems**
  - ↳ Global Configuration rollout without gradual validation — no phased staging or health monitoring before affecting all traffic
  - ↳ Legacy code assumption — the code assumed certain rule objects (like "execute") would always exists. When disabled by the kill switch, assumption voilated and triggered error. Tight coupling b/w code logic and configuration state.
  - ↳ Lack of Fail-Open Logic — The system should have continue to serve traffic in a degraded mode rather than closing completely

- **Lessons Learnt**
  - ↳ Configuration changes can be as risky as software bugs
  - ↳ Global rollouts without safeguards increase blast radius
  - ↳ Fail open design incr. resilience
  - ↳ Legacy assumption creates hidden risks

Date _____ Day _____

- **Client-side Faults**

↳ Single CDN / Edge Dependency
↳ Many clients relied solely on Cloudflare CDN, WAF and traffic routing. • When cloudflare failed they had no alternative path

↳ Lack of Multi-CDN or Fallback Strategy
↳ Clients did not implement multi-CDN failover, which could have routed to other provider on failure

↳ Insufficent Client-Side Resilience
↳ Some applications lacked
↳ Proper caching ↳ Retry logic ↳ Graceful error handling

- **Post Incident Response**

↳ Immediate Rde back of faulty configurations that disabled the WAF test rule was reverted
↳ As soon as faulty configs were removed, affected services recovered and HTTP 500 error rates return to normal
↳ Cloudflare added additional safeguards

Date _____ Day _____

- **Post Incident Response (18th Nov)**

↳ Engineers halted the spread of the oversized config file accross global network

↳ They replaced the faulty file with prior known-good version and redistributed it globally

↳ Forced restarts where necessary