# Resilient SADAD Payment Network — EMAM Framework Report

**Author**: Fuad Al Alawi
**Course**: Design Resilient National System
**Date**: November 26, 2025
**GitHub**: https://github.com/FuadAlawi/Resilient-SADAD-System

---

## Executive Summary

This report presents a comprehensive resilient system design for Saudi Arabia's SADAD payment network, implementing the EMAM framework (Understand, Practice, Master, Excellence). The design addresses critical national infrastructure requirements through multi-layered resilience combining preventive security, fault tolerance, rapid recovery, and continuous operation under adverse conditions.

The SADAD network processes millions of daily transactions for utilities, government services, and e-commerce, making it a cornerstone of Saudi Arabia's digital economy and Vision 2030 objectives. Any disruption directly impacts citizens' ability to fulfill financial obligations and access essential services.

**Key Deliverables**: - Multi-Availability Zone architecture with automatic failover across 3 regions - Comprehensive chaos engineering suite testing 5 critical failure scenarios - Sub-15-minute RTO (Recovery Time Objective) for critical services - Infrastructure as Code using Terraform for rapid disaster recovery - Kubernetes-orchestrated services with horizontal auto-scaling - Prometheus-based observability with SLO-driven alerting

This design is grounded in Islamic principles of Amanah - stewardship), Adl - fairness), and No harm).

---

## افهم (Understand) — Why Resilience > Security, Threat Analysis

### Saudi Context and Critical Infrastructure

The Kingdom of Saudi Arabia's digital payment infrastructure represents a critical national asset that underpins both economic activity and Vision 2030's digital transformation objectives. The SADAD network, along with mada and SARIE systems, processes over 50 million transactions monthly, supporting:

- **Utility Payments**: Electricity, water, telecommunications bills for millions of households
- **Government Services**: Tax payments, visa fees, municipal services, traffic violations
- **E-Commerce**: Online retail, food delivery, digital services representing SAR billions in annual volume
- **Interbank Settlements**: Real-time payment settlements between financial institutions

Any extended outage creates cascading societal impacts: - Citizens unable to pay critical bills, risking service disconnection - Government revenue collection delays affecting public services - E-commerce merchants losing sales during peak periods (Ramadan, Eid) - Erosion of public trust in digital payment systems

**Historical Context**: The 2012 Shamoon malware attack on Saudi Aramco demonstrated the destructive potential of cyber threats targeting Saudi infrastructure. This attack erased data on over 30,000 computers, highlighting the need for resilience beyond traditional security measures.

**Regulatory Framework**: The Saudi Arabian Monetary Authority (SAMA) mandates: - Business Continuity Planning (BCP) with tested disaster recovery procedures - Regular resilience testing including chaos engineering scenarios - Incident reporting within 4 hours of service degradation - Annual third-party audit of critical systems - Minimum 99.9% availability for payment processing services

**Why Resilience Exceeds Security**

While security focuses on **preventing** incidents, resilience acknowledges that failures are inevitable and focuses on:

**1. Continuing Operations Under Stress** - Security: “How do we prevent this attack?” - Resilience: “How do we maintain 80% service capacity if attacked?”

Example: During a DDoS attack, security measures (rate limiting, WAF) prevent some traffic, but resilience (auto-scaling, CDN, graceful degradation) ensures legitimate users can still transact.

**2. Rapid Recovery Over Perfect Prevention** - RTO (Recovery Time Objective): 15 minutes for payment API - RPO (Recovery Point Objective): 1 minute for transaction ledger - MTTR (Mean Time To Recovery): Target < 30 minutes

A resilient system recovers from a compromise in minutes using Infrastructure as Code, whereas a security-only approach might take hours to investigate and rebuild.

**3. Graceful Degradation** - Payment processing continues even if reporting dashboards fail - Read-only mode for ledger queries if write capacity is impaired - Queue transactions during network partition, replay when connectivity restored

**Islamic Principles in Resilient Design**

**Amanah - Trust and Stewardship)**: - Citizens entrust their financial data and critical transactions to SADAD - Operators have a sacred duty to protect this trust through reliable service - Implementation: End-to-end encryption, audit trails, transparent incident reporting

**Adl - Justice and Fairness)**: - All citizens deserve equal access to payment services regardless of geography - No group should be disproportionately affected by outages - Implementation: Multi-region deployment ensures service continuity if one region fails

**No harm ضرار ولا (No Harm)**: - Minimize societal harm through rapid containment and recovery - Clear communication during incidents to prevent panic or financial hardship - Implementation: Automated failover, pre-approved runbooks, public status page

**Comprehensive Threat Model**

Using the STRIDE methodology enhanced with operational failure scenarios:

**Threat Actors**: 1. **State-Sponsored APT Groups**: Sophisticated, well-resourced, targeting national infra 2. **Cybercriminal Organizations**: Financially motivated ransomware, data exfiltration 3. **Insider Threats**: Disgruntled employees, social engineering victims 4. **Supply Chain Compromises**: Malicious dependencies, compromised CI/CD pipelines 5. **Environmental/Operational**: Natural disasters, misconfigurations, human error

**Critical Assets**:

| Asset Category | Examples | Resilience Measures |
|---|---|---|
| Payment API | REST endpoints, authentication service | Multi-AZ, circuit breakers, rate limiting |
| Transaction Ledger | Database, audit logs | Replicated across 3 AZs, PITR backups |
| Settlement Services | Bank connectors, clearing house links | Redundant network paths, message queuing |
| Cryptographic Keys | KMS, HSM, TLS certificates | Key rotation, backup KMS in DR region |
| Customer PII | Names, IBANs, transaction history | Encryption at rest/transit, access logging |

**Attack Paths and Mitigations**: 1. **Credential Theft → Privilege Escalation** - Threat: Stolen admin credentials used to access production systems - Resilience: Break-glass access with MFA, session recording, automated deprovisioning

2. **Lateral Movement**
    - Thre at: Compromised POD spreads to other services
    - Resilience: Network policies, microsegmentation, namespace isolation
3. **Container Breakout**
    - Threat: Escape from container to underlying node
    - Resilience: Pod Security Standards, read-only file systems, AppArmor/SELinux
4. **Supply Chain Attack**
    - Threat: Malicious dependency in application code
    - Resilience: SBOM generation, signed images, admission controllers

**Environmental Failures**: - **AZ Failure**: AWS Availability Zone outage affecting 33% of infrastructure - **Regional Failure**: Entire AWS region (me-central-1) becomes unavailable - **Network Partition**: Split-brain scenario where services can't communicate - **DNS Outage**: Route53 or internal DNS resolution failures - **Data Center Fire**: Physical destruction of infrastructure (DR scenario)

---

## مراس (Practice) — 4 Rs and Swiss Cheese Model

**The Four Rs of Resilience**

Our implementation operationalizes the 4 Rs framework across all system layers:

**1. Robustness: Building Strength Into the System** **Application-Level Robustness**: - **Idempotent Operations**: All payment processing endpoints use idempotency keys to prevent duplicate charges if requests are retried `java @PostMapping("/api/v1/payments") public PaymentResponse processPayment( @RequestHeader("Idempotency-Key") String idempotencyKey, @RequestBody PaymentRequest request) { // Check if this exact request was already processed Optional<Payment> existing = paymentRepository .findByIdempotencyKey(idempotencyKey); if (existing.isPresent()) { return PaymentResponse.fromPayment(existing.get()); } // Process new payment atomically return paymentService.processWithKey(request, idempotencyKey); }`

- **Circuit Breakers**: Prevent cascading failures when dependent services degrade
    - Open circuit after 5 consecutive failures
    - Half-open state after 30 seconds to test recovery
    - Fallback to cached data or degraded functionality
- **Rate Limiting**: Protect against abuse and ensure fair resource allocation
    - Per-user limits: 100 requests/minute for standard tier
    - Per-IP limits: 1000 requests/minute to prevent DDoS
    - Exponential backoff with jitter for retries
- **Input Validation**: Strong typing and schema validation at API boundaries

**Infrastructure Robustness**: - Kubernetes resource requests and limits prevent resource starvation - Pod Disruption Budgets (PDB) ensure minimum availability during updates - Topology spread constraints distribute pods across failure domains

**2. Redundancy: Eliminating Single Points of Failure** **Geographic Redundancy**: - **Multi-AZ Deployment**: Services deployed across 3 Availability Zones in me-central-1 - Each AZ is a physically separate data center with independent power/cooling - Minimum 2 pod replicas per AZ for critical services - Traffic automatically routes around failed AZsthrough Kubernetes Service

- **Cross-Region DR**: Warm standby in me-south-1 (UAE)
  - Database replica with 5-minute replication lag
  - Pre-deployed application infrastructure
  - DNS failover capability through Route53

**Component Redundancy**:

| Layer | Component | Redundancy Strategy |
|---|---|---|
| Load Balancing | AWS ALB | Multi-AZ with health checks |
| Application | Payment Pods | Min 6 replicas (2 per AZ), HPA up to 12 |
| Database | RDS PostgreSQL | Multi-AZ with automated failover |
| Cache | Redis Cluster | 3-node cluster with replication |
| Message Queue | Amazon SQS | Regionally redundant by design |
| KMS | AWS KMS | Regional service with cross-region backup keys |

**Network Redundancy**: - Dual NAT Gateways per AZ for outbound connectivity - Multiple VPN tunnels for on-premise connectivity - BGP-based routing with automatic failover

**3. Resourcefulness: Adapting to Changing Conditions** **Automated Response Playbooks**:

1. **High Error Rate Detected**:

```
- Alert: ErrorRate > 1% for 5 minutes
- Action 1: Auto-scale pods +50%
- Action 2: Enable circuit breakers if not already active
- Action 3: Page on-call engineer
- Action 4: If error rate > 5%, initiate gradual rollback
```

2. **Database Performance Degradation**:

```
- Alert: DB latency p95 > 500ms
- Action 1: Scale read replicas
- Action 2: Enable query result caching
- Action 3: Throttle non-critical background jobs
- Action 4: If latency > 2s, enable read-only mode
```

**Feature Flags for Graceful Degradation**: - Real-time fraud checks can be disabled to reduce latency - Analytics/reporting features disabled during peak load - Transaction limits lowered during capacity constraints

**Break-Glass Procedures**: - Pre-approved emergency changes (no CAB review required) - Escalation matrix with mobile contacts for 24/7 response - Automated provisioning of emergency capacity

**4. Rapidity: Speed of Detection and Recovery** **Detection Speed** (Target: < 2 minutes): - Prometheus scrapes metrics every 15 seconds - AlertManager evaluates rules every 30 seconds - PagerDuty notification within 30 seconds of alert firing - Health check failures trigger immediate pod replacement

**Response Speed** (Target RTO: 15 minutes): - Automated horizontal scaling responds in < 60 seconds - Pod replacement (failed health checks) completes in < 90 seconds - AZ failover (Kubernetes native) completes in < 3 minutes - Region failover (manual DNS update) completes in < 15 minutes

**Recovery Automation**:

```
# Infrastructure recovery via Terraform
terraform apply -auto-approve -target=module.eks
```

```
# Application redeployment via Kubernetes
kubectl rollout restart deployment/payment-tier1
kubectl rollout status deployment/payment-tier1 --timeout=5m

# Database recovery from snapshot
aws rds restore-db-instance-from-db-snapshot \
  --db-instance-identifier sadad-prod-recovered \
  --db-snapshot-identifier sadad-backup-$(date +%Y%m%d)
```

#### Swiss Cheese Model: Defense in Depth

Each layer provides independent protection; an attack must penetrate ALL layers to cause harm:

**Layer 1: Prevent** (Stop attacks before they reach the system) - CI/CD pipeline scans for vulnerabilities (Snyk, Trivy) - Software Bill of Materials (SBOM) generation for all dependencies - Container image signing and verification (Cosign/Notary) - Least privilege IAM roles and service accounts - Network firewalls and security groups

**Layer 2: Detect** (Identify anomalies quickly) - Prometheus metrics: latency, error rates, saturation, traffic (RED/USE) - Distributed tracing (OpenTelemetry) for request flow visualization - Structured application logs aggregated to CloudWatch - SLO-based alerting (availability, latency, errors) - Blackbox probes from external vantage points

**Layer 3: Contain** (Limit blast radius) - Kubernetes Network Policies restrict pod-to-pod communication - Namespace isolation separates production from staging - KMS key scoping limits which services can decrypt sensitive data - Pod Security Standards prevent privilege escalation - Resource quotas prevent resource exhaustion

**Layer 4: Recover** (Restore service rapidly) - Multi-AZ deployment survives AZ failures automatically - Automated backups with Point-In-Time Recovery (PITR) - DR automation scripts with runbooks - Transaction replay from event log if ledger corrupted - Idempotency keys prevent duplicate processing during recovery

**Layer 5: Learn** (Continuous improvement) - Blameless postmortems after every incident - Chaos engineering drills quarterly - Automated runbook updates based on incident learnings - Security audits and penetration testing annually - Change Advisory Board (CAB) reviews high-risk changes

### Practical Chaos Engineering Scenarios

We implement 5 chaos scenarios aligned with Saudi-specific threat landscape:

**Scenario 1: Pod Kill (Medium Severity)** **Objective**: Verify Kubernetes self-healing and service continuity

```
name: pod-kill-scenario
severity: medium
steps:
  - description: Kill one payment pod
    action: pod-kill
    parameters:
      selector: app=resilient-sadad-network
      mode: one
  - description: Verify service remains available
    action: http-check
    parameters:
      url: http://payment/api/v1/healthz
      expect: 200
```

**Expected Outcome**: Pod recreated in < 90s, no dropped requests

**Scenario 2: Network Partition (High Severity)** **Objective**: Test split-brain handling and consensus protocols

```
name: network-partition-scenario
severity: high
steps:
  - description: Partition AZ-1 from AZ-2/3
    action: network-partition
    parameters:
      source_zone: me-central-1a
      target_zones: [me-central-1b, me-central-1c]
      duration: 5m
  - description: Verify leader election
    action: verify-leader
    parameters:
      expect_new_leader: true
      timeout: 60s
```

**Expected Outcome**: New leader elected in minority partition, no data loss

**Scenario 3: AZ Outage (Critical Severity)** **Objective**: Validate multi-AZ resilience under complete AZ failure

```
name: az-outage-simulation
severity: critical
steps:
  - description: Simulate complete AZ failure
    action: drain-zone
    parameters:
      zone: me-central-1a
  - description: Monitor service availability
    action: availability-check
    parameters:
      min_success_rate: 0.95
      duration: 10m
```

**Expected Outcome**: Traffic shifts to AZ-2/3, 95%+ availability maintained

**Scenario 4: DDoS During Eid (High Severity)** **Objective**: Test auto-scaling and rate limiting under extreme load

```
name: ddos-eid-scenario
severity: high
steps:
  - description: Ramp traffic to 10x normal
    action: traffic-generator
    parameters:
      target_rps: 50000
      duration: 15m
  - description: Verify auto-scaling response
    action: verify-hpa
    parameters:
      expect_min_replicas: 12
      expect_max_cpu: 70%
```

**Expected Outcome**: HPA scales to max replicas, latency < 1s p95

**Scenario 5: Shamoon-Class Mal ware (Critical Severity)** **Objective**: Test containment and recovery from destructive malware

```
name: shamoon-simulation
severity: critical
steps:
  - description: Simulate disk wiper on node
    action: destroy-node-disks
    parameters:
      node: worker-node-3
      dry_run: true
  - description: Verify pod evacuation
    action: verify-pod-migration
    parameters:
      expect_all_pods_rescheduled: true
      timeout: 5m
  - description: Verify terraform can rebuild node
    action: terraform-plan
    parameters:
      target: aws_eks_node_group.general
```

**Expected Outcome**: Pods migrate off compromised node, node rebuilt from IaC

---

## اتقن (Master) — Resilient Architecture & Critical Services

### System Architecture Overview

The SADAD resilient payment network is built on a cloud-native architecture leveraging AWS managed services and Kubernetes for container orchestration. The design prioritizes:

1. **Horizontal scalability**: Add capacity by deploying more pods, not bigger instances
2. **Geographic distribution**: Multi-AZ within region + cross-region DR
3. **Automated recovery**: Self-healing infrastructure with minimal human intervention
4. **Observability**: Comprehensive metrics, logs, and traces for rapid troubleshooting

### Application Layer

**Payment Processing Service (Spring Boot)** The core payment service is a stateless REST API built with Spring Boot 3.3.5 and Java 17:

**Key Features**: - **Idempotent Processing**: Prevents duplicate charges during retries - **Circuit Breakers**: Resilience4j library provides automatic circuit breaking - **Rate Limiting**: Bucket4j for token bucket algorithm implementation - **Health Checks**: Actuator endpoints for Kubernetes readiness/liveness probes - **Metrics Export**: Micrometer exports metrics in Prometheus format

**Configuration** (`src/main/resources/ application.yml`):

```yaml
spring:
  application:
    name: resilient-sadad-network

management:
  endpoints:
    web:
      exposure:
        include: health,prometheus,metrics
  metrics:
    export:
      prometheus:
        enabled: true
  health:
    livenessState:
      enabled: true
    readinessState:
      enabled: true

resilience4j:
  circuitbreaker:
    instances:
      paymentProcessing:
        registerHealthIndicator: true
        slidingWindowSize: 10
        permittedNumberOfCallsInHalfOpenState: 3
        failureRateThreshold: 50
        waitDurationInOpenState: 30s
```

**Chaos Monkey Integration** (`application-chaos.yml`):

```yaml
spring:
  profiles: chaos

chaos:
  monkey:
    enabled: true
    assaults:
      level: 5
      latencyActive: true
      latencyRangeStart: 500
      latencyRangeEnd: 2000
      exceptionsActive: true
      killApplicationActive: false
```

**Ledger Writer Service** Asynchronous service for writing payment records to the database: - Uses Spring's `@Async` for non-blocking writes - Implements outbox pattern for reliable message delivery - Guarantees at-least-once delivery semantics - Supports replay from transaction log if data corrupted

**Settlement Connector** Integrates with SARIE (Saudi RTGS) and bank APIs: - Maintains persistent connections with retry logic - Queues settlement requests during network outages - Reconciles settlements daily with automated reports

### Infrastructure Layer (AWS + Terraform)

**VPC and Networking** Our Terraform configuration provisions a production-grade network: 70% for 30 seconds - Scale-down delay: 5 minutes to avoid flapping

### Observability and SLOs

**Prometheus Monitoring** **ServiceMonitor for Metrics Collection** (`kubernetes/service-monitor.yaml`):

```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: payment-metrics
  labels:
    app: resilient-sadad-network
spec:
  selector:
    matchLabels:
      app: resilient-sadad-network
  endpoints:
    - port: http
      path: /actuator/prometheus
      interval: 15s
```

**PrometheusRule for SLO Alerts** (`kubernetes/prometheus-rules.yaml`):

```yaml
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  name: payment-slo-alerts
spec:
  groups:
    - name: slo-alerts
      interval: 30s
      rules:

        - alert: HighErrorRate
          expr: |
            (
              sum(rate(http_server_requests_seconds_count{status=~"5.."}[5m]))
              /
              sum(rate(http_server_requests_seconds_count[5m]))
            ) > 0.001
          for: 5m
          labels:
            severity: critical
          annotations:
            summary: "Error rate above SLO threshold"
            description: "Error rate is {{ $value | humanizePercentage }}"


        - alert: HighLatency
          expr: |
            histogram_quantile(0.95,
              sum(rate(http_server_requests_seconds_bucket[5m])) by (le)
            ) > 0.5
          for: 5m
          labels:
            severity: warning
          annotations:
            summary: "p95 latency above SLO threshold"
            description: "p95 latency is {{ $value }}s"


        - alert: HighCPUUsage
          expr: |
            avg(
              rate(container_cpu_usage_seconds_total{pod=~"payment-.*"}[5m])
            ) > 0.8
          for: 10m
          labels:
            severity: warning
          annotations:
            summary: "High CPU usage detected"
            description: "Average CPU usage is {{ $value | humanizePercentage }}"
```

**SLO Targets**:

| Metric | Target | Measurement Window |
|---|---|---|
| Availability | 99.9% | 30-day rolling |
| Latency (p95) | < 500ms | 5-minute window |
| Error Rate | < 0.1% | 5-minute window |
| CPU Saturation | < 80% | 10-minute average |

**Data Persistence and Backup**

**Database (RDS PostgreSQL)**

- **Multi-AZ Deployment**: Automatic failover to standby in different AZ
- **Automated Backups**: Daily snapshots with 7-day retention
- **Point-In-Time Recovery**: Restore to any second within backup window

- **Read Replicas**: Offload analytics queries from primary

**Backup Strategy**

| Data Type | Backup Method | Frequency | Retention | RPO |
|---|---|---|---|---|
| Transaction Ledger | RDS Automated Backup | Daily | 7 days | 1 minute |
| Configuration | Git + S3 Versioning | On commit | Indefinite | 0 (immutable) |
| Application Logs | CloudWatch Logs | Real-time | 90 days | 0 |
| Metrics | Prometheus TSDB | 15s scrape | 15 days | 15 seconds |

---

## ميز (Excellence) — Innovation & Vision 2030

- **Innovation**
  - Self-healing with policy-as-code to auto-quarantine compromised pods.
  - Proactive capacity predictions using telemetry.
  - Immutable infra with rapid rehydration from IaC.
- **Vision 2030 alignment**
  - Enable fintech ecosystem and cashless society goals with high uptime and trust.
  - Data residency & compliance by design; transparency and public confidence.
  - Ethical stewardship grounded in Islamic principles of fairness and harm minimization.

---

## Recovery Procedures (Runbooks)

See `docs/runbooks/recovery-procedures.md` for malware containment, DR failover, and key rotation playbooks, including RTO/RPO targets and verification steps.

---

## Chaos, Monitoring, and SLOs

- Run app with Chaos Monkey profile (local):
  - `mvn spring-boot:run -Dspring-boot.run.profiles=chaos`
  - `scripts/chaos-monkey-demo.sh`
- Kubernetes monitoring:
  - Apply `kubernetes/service-monitor.yaml` and `kubernetes/prometheus-rules.yaml` with kube-prometheus-stack.
- Scenarios:
  - See `chaos-tests/` YAMLs for pod kill, partition, zone outage, DDoS, and Shamoon simulation.

---

## Verification Results

**Application Build & Tests**

| Test Phase | Status | Details |
|---|---|---|
| **Maven Build** | □ SUCCESS | Compiled 3 source files |
| **Resources** | □ SUCCESS | Copied 2 resources to target/classes |
| **Unit Tests** | □ SUCCESS | All tests passed |
| **Build Time** | □ 1.045s | Fast build cycle |
| **Date** | □ Verified | 2025-11-26T20:03:15+03:00 |

**Command**: `mvn clean test`

---

**Chaos Test Scenarios**

| Scenario | File | Severity | Status | Description |
|---|---|---|---|---|
| **AZ Outage** | `az-outage-simulation.yml` | □ Critical | □ Ready | Simulates complete Availability Zone failure |
| **DDoS (Eid)** | `ddos-eid-scenario.yml` | □ High | □ Ready | High-traffic scenario during Eid period |
| **Network Partition** | `network-partition-scenario.yml` | □ High | □ Ready | Simulates network split between zones |
| **Pod Kill** | `pod-kill-scenario.yml` | □ Medium | □ Ready | Kills payment pod, verifies availability |
| **Shamoon Malware** | `shamoon-simulation.yml` | □ Critical | □ Ready | Disk-wipe simulation with recovery |

**Command**: `bash chaos-tests/run-all-tests.sh`

**Total Scenarios**: 5 (2 Critical, 2 High, 1 Medium)

---

**Infrastructure Components**

| Component | Technology | Configuration | Resilience Features |
|---|---|---|---|
| **VPC** | AWS VPC | 10.0.0.0/16 | 3 Availability Zones |
| **Network** | Public/Private Subnets | 3 public + 3 private | Multi-AZ isolation |
| **NAT** | NAT Gateway | One per AZ | High availability |
| **Compute** | EKS 1.27 | Auto Scaling | Min: 3, Max: 6 nodes |
| **Nodes** | t3.medium | ON_DEMAND | Spread across AZs |
| **Monitoring** | Prometheus | ServiceMonitor | SLO-based alerts |
| **Orchestration** | Kubernetes | HPA + PDB | Self-healing |

**Terraform Files**: `main.tf`, `variables.tf`, `outputs.tf`, `versions.tf`

---

## Submission

- Repository: include this EMAM report, runbooks, chaos scenarios, Terraform, and Kubernetes manifests.
- PDF: export this document via `scripts/build-pdf.sh` (requires pandoc) or Print-to-PDF.
- Checklist:
  - EMAM report covers Saudi context and Islamic principles.
  - 4 Rs and Swiss Cheese applied with practical tests.
  - Architecture, SLOs, and recovery procedures included.
  - Chaos tests runnable (stubs or integrated) and monitored.