Technical Operations Excellence

A comprehensive guide to Site Reliability Engineering, Observability, and Platform Operations

34	10	35+
ONE-PAGERS	CORE THEMES	RESEARCH SOURCES

VISION & OVERVIEW

- → **Reliability Unleashed** — From Chaos to Confidence

SRE FOUNDATIONS

- → **SRE Foundations** — SLIs, SLOs, Error Budgets
- → **DORA 24 Capabilities** — DevOps Research Framework
- → **SRE Maturity Assessment** — Measuring Capabilities
- → **SLO Design Framework** — Effective Objectives

OBSERVABILITY

- → **Observability Mastery** — Three Pillars & OTel
- → **Multi-Window Alerting** — Burn Rate Strategy
- → **USE Method** — Utilization, Saturation, Errors
- → **Observability 2.0** — High Cardinality Events
- → **Alert Tuning Playbook** — Reducing Noise

RESILIENCE PATTERNS

- → **Resilience Patterns** — Circuit Breakers, Bulkheads
- → **Defense in Depth** — Layered Security
- → **HRO Patterns** — High-Reliability Orgs
- → **Release It! Patterns** — Stability Patterns
- → **Chaos Engineering** — GameDay Practices

INCIDENT MANAGEMENT

- → **Incident Excellence** — Response & Postmortems
- → **Learning from Catastrophe** — Case Studies
- → **Runbook Quick Reference** — Templates & Practices

RELEASE & CAPACITY

- → **Capacity & Release** — DORA, Progressive Delivery
- → **NALSD Framework** — Large System Design
- → **Designing for Recovery** — Breakglass Access

INFRASTRUCTURE

- → **Infrastructure Reliability** — K8s, TSDB, Backends
- → **Kubernetes Patterns** — K8s Operational Patterns
- → **Platform Engineering** — Golden Paths, Self-Service

AI/ML & AGENTIC

- → **AI/ML Operations** — MLOps, Non-Determinism
- → **Agentic Operations** — Bot Operations, AI Agents

PEOPLE & CULTURE

- → **People & Culture** — Westrum, Team Topologies
- → **On-Call Excellence** — Sustainable Rotations
- → **Three Ways of DevOps** — Flow, Feedback, Learning
- → **Team Topologies** — Organizing Teams

INDUSTRY & IMPLEMENTATION

- → **Industry Leaders** — Google, Netflix, NASA
- → **Implementation Roadmap** — Getting Started
- → **Automation Paradoxes** — When Automation Hurts
- → **SRE Evolution Timeline** — History & Future

Reliability Unleashed

From Chaos to Confidence

Vision & Overview | Technical Operations Excellence

182x	2,293x	70%	35+
MORE DEPLOYS[1]	FASTER RECOVERY[1]	AUTO-RESOLUTION[2]	RESEARCH SOURCES

WHAT IS SRE?

> *SRE is what happens when you ask a software engineer to design an operations team.*
>
> - Google SRE Book

- → **DevOps** is the philosophy; **SRE** is the implementation
- → 50% engineering / 50% operations cap (max toil)
- → Error budgets govern release velocity

10 CORE THEMES

#	THEME	FOCUS
1	**Foundations**	SLOs, error budgets, toil
2	**Observability**	Three pillars, OTel, alerting
3	**Resilience**	Patterns, blast radius, defense
4	**Incidents**	Response, postmortems, HRO
5	**Release**	CI/CD, progressive delivery
6	**Infrastructure**	K8s, IaC, platform engineering
7	**AI/ML Ops**	Non-determinism, drift, MLOps
8	**Agentic Ops**	Bot operations, autonomy
9	**Culture**	Teams, on-call, sustainability
10	**Industry**	Case studies, benchmarks

DORA ELITE BENCHMARKS

METRIC	ELITE	LOW
Deploy Frequency	On-demand	> 6 months
Lead Time	< 1 day	> 6 months
Change Failure	0-15%	> 30%
MTTR	< 1 hour	> 6 months

Source: DORA State of DevOps 2024 - 36,000+ professionals

FOUR GOLDEN SIGNALS

Latency	How fast?
Traffic	How much?
Errors	Failing?
Saturation	How full?

Reliability is a Feature

Users don't distinguish between "the app is slow" and "the app is broken"

From Alert Fatigue to Autonomous Operations

70% auto-resolution | 30-second MTTD | <2 pages per on-call shift

THREE PILLARS OF OPERATIONS

Reactive
Respond to incidents, triage alerts, execute runbooks

Proactive
Trend analysis, capacity planning, SLO monitoring

Predictive
Anomaly detection, AIOps, chaos engineering

GUIDING PHILOSOPHY

> *Learn from industries where failure means lives lost.*
>
> - HRO Research

- → **Blameless culture:** Focus on systems, not individuals
- → **Embrace complexity:** Simple explanations often miss root cause
- → **Authority to expertise:** Knowledge trumps hierarchy in crisis

SRE MATURITY JOURNEY

LEVEL	STATE	CHARACTERISTICS
1	**Ad-Hoc**	Reactive, firefighting
2	**Foundational**	Basic monitoring, SLOs
3	**Standardized**	IaC, CI/CD, postmortems
4	**Advanced**	Predictive, chaos, AIOps
5	**Optimized**	Autonomous operations

KEY ACRONYMS

SLI/SLO/SLA	Indicator / Objective / Agreement
MTTR/MTTD	Mean Time to Recover / Detect
DORA	DevOps Research & Assessment
HRO	High-Reliability Organization

[1] DORA State of DevOps 2023 (elite vs low performers) [2] Target based on industry AIOps benchmarks

SRE Foundations

SLIs, SLOs, Error Budgets & The Philosophy of Reliability

SRE Foundations | Technical Operations Excellence

50%	99.9%	43m	4
MAX TOIL CAP	TYPICAL SLO	ERROR BUDGET/MO	GOLDEN SIGNALS

THE CORE PHILOSOPHY

> *Hope is not a strategy.*
>
> \- Google SRE Book

- → **class SRE implements interface DevOps**
- → Apply engineering discipline to operations
- → Balance reliability with feature velocity
- → Measure everything; improve continuously

SLI / SLO / SLA HIERARCHY

TERM	DEFINITION	EXAMPLE
SLI	Service Level Indicator	Request latency P99
SLO	Service Level Objective	P99 < 200ms
SLA	Service Level Agreement	99.9% or credits

SLOs should be stricter than SLAs for early warning

ERROR BUDGET MATH

SLO	BUDGET	MONTHLY
99%	1%	7.2 hours
99.9%	0.1%	43.2 minutes
99.95%	0.05%	21.6 minutes
99.99%	0.01%	4.32 minutes
99.999%	0.001%	26.3 seconds

Each 9 costs 10x more - choose wisely

ERROR BUDGET POLICY

Healthy (>50%)
Ship features freely, accept calculated risks

Warning (25-50%)
Prioritize reliability, increase review rigor

Critical (<25%)
Feature freeze, focus exclusively on stability

THE FOUR GOLDEN SIGNALS

SIGNAL	MEASURES	QUESTION
Latency	Request time	How fast?
Traffic	System demand	How much?
Errors	Failed requests	Failing?
Saturation	Utilization	How full?

If you can only measure four things, measure these

TOIL: THE ENEMY OF SRE

Toil = manual, repetitive, automatable work that scales linearly with service growth

TOIL	NOT TOIL
Manually restarting services	Writing automation
Copy-paste deployments	Designing CI/CD
Manual scaling	Auto-scaling policies
Repetitive tickets	Self-service tools

<50% MAX TOIL (GOOGLE RULE)

WHAT IS TOIL?

CHARACTERISTIC	EXAMPLE
Manual	Human runs script
Repetitive	Done frequently
Automatable	No judgment needed
Tactical	Interrupt-driven
No lasting value	Doesn't improve system

Google SRE: Cap toil at 50% of time; invest the rest in engineering

SLO CATEGORIES

Availability	% successful requests
Latency	% under threshold
Throughput	Requests processed
Freshness	Data staleness

Error Budgets Enable Innovation

When healthy, take risks. When depleted, stabilize.
It's data for decisions, not punishment.

DORA 24 Capabilities

The Science of Software Delivery Performance

Performance Metrics | Technical Operations Excellence

5 CAPABILITY CATEGORIES

24 CORE CAPABILITIES

182x ELITE DEPLOY FREQ

36K+ SURVEY RESPONDENTS

THE 4 KEY METRICS

METRIC	ELITE	LOW
Deployment Frequency	On-demand	>6 months
Lead Time for Changes	<1 day	>6 months
Change Failure Rate	0-15%	>64%
MTTR	<1 hour	>6 months

Elite performers: 182x more deploys, 2,293x faster recovery

MEASUREMENT CAPABILITIES (4)

#	CAPABILITY
17	Monitoring & observability
18	Proactive failure notification
19	WIP limits
20	Visualizing work

TECHNICAL CAPABILITIES (8)

#	CAPABILITY
1	Version control
2	Deployment automation
3	Continuous integration
4	Trunk-based development
5	Test automation
6	Database change management
7	Shift left on security
8	Loosely coupled architecture

PRODUCT CAPABILITIES (4)

#	CAPABILITY
21	Customer feedback
22	Value stream visibility
23	Working in flow state
24	User research integration

IMPROVEMENT PATHWAYS

Start: Automation
CI/CD, version control, test automation

Then: Architecture
Loosely coupled, trunk-based, small batches

Finally: Culture
Generative culture, learning, leadership

CULTURAL CAPABILITIES (5)

#	CAPABILITY
9	Generative culture (Westrum)
10	Job satisfaction
11	Learning culture
12	Transformational leadership
13	Work-life balance

KEY INSIGHT

> *You can't buy your way to high performance. Culture and practices matter more than tools.*
>
> - DORA Research

PROCESS CAPABILITIES (3)

#	CAPABILITY
14	Work visibility
15	Working in small batches
16	Team experimentation

Continuous Improvement

The journey to elite performance is incremental.

SRE Maturity Assessment

Measuring and Improving SRE Capabilities

Strategic Roadmap | Technical Operations Excellence

15 ASSESSMENT DOMAINS

450 MAX POINTS

5 MATURITY LEVELS

Q QUARTERLY REVIEW

15 ASSESSMENT DOMAINS

#	DOMAIN	MAX POINTS
1	**SLOs & Error Budgets**	30
2	**Observability**	30
3	**Incident Management**	30
4	**Postmortems**	30
5	**Toil Reduction**	30
6	**Capacity Planning**	30
7	**Change Management**	30
8	**CI/CD Pipeline**	30
9	**Disaster Recovery**	30
10	**Security**	30
11	**Documentation**	30
12	**On-Call**	30
13	**Chaos Engineering**	30
14	**Culture**	30
15	**Platform**	30

5 MATURITY LEVELS

LEVEL	NAME	SCORE
1	**Ad-hoc**	0-90
2	**Foundational**	91-180
3	**Standardized**	181-270
4	**Advanced**	271-360
5	**Optimized**	361-450

SCORING GUIDE (PER DOMAIN)

SCORE	CRITERIA
0-6	No formal practice
7-12	Basic/reactive approach
13-18	Documented processes
19-24	Proactive, measured
25-30	Optimized, automated

PRIORITY ACTION PLANNING

High Impact, Low Effort
Quick wins: implement first

High Impact, High Effort
Strategic: plan carefully

Low Impact
Deprioritize or defer

RADAR CHART DOMAINS

→ **Core SRE:** SLOs, observability, incidents, postmortems
→ **Automation:** CI/CD, toil reduction, platform
→ **Resilience:** Capacity, DR, chaos, security
→ **Culture:** On-call, documentation, culture

ASSESSMENT CADENCE

ACTIVITY	FREQUENCY
Full assessment	Quarterly
Progress review	Monthly
Action items	Weekly tracking
Stakeholder report	Quarterly

COMMON GAPS

DOMAIN	TYPICAL ISSUE
SLOs	No error budgets enforced
Postmortems	Blame-focused reviews
On-Call	Alert fatigue, burnout
Chaos	No regular practice

Measure to Improve
What gets measured gets managed.

SLO Design Framework

From SLI to SLA: Building Meaningful Reliability Targets

SRE Foundations | Technical Operations Excellence

3	99.9%	8.76hr	30d
SLI/SLO/SLA TIERS	COMMON TARGET	ANNUAL BUDGET (99.9%)	ROLLING WINDOW

SLI → SLO → SLA HIERARCHY

TERM	DEFINITION	OWNER
SLI	Metric that measures service	Engineers
SLO	Target value for the SLI	SRE/Product
SLA	Contract with consequences	Business/Legal

Rule: SLO should be stricter than SLA (buffer for internal response)

COMMON SLI TYPES

TYPE	SLI FORMULA
Availability	Successful requests / Total requests
Latency	Requests < threshold / Total requests
Throughput	Requests served / Time period
Correctness	Correct responses / Total responses
Freshness	Data age < threshold / Total reads

TARGET SELECTION GUIDE

TARGET	MONTHLY DOWNTIME	USE CASE
99%	7.3 hours	Internal tools
99.5%	3.6 hours	Non-critical services
99.9%	43.8 min	Standard production
99.95%	21.9 min	Business-critical
99.99%	4.4 min	Mission-critical

ERROR BUDGET POLICY

BUDGET STATUS	ACTION
>50% remaining	Ship features freely
25-50%	Ship with caution
<25%	Reliability focus only
Exhausted	Feature freeze

SERVICE-TYPE PATTERNS

SERVICE TYPE	PRIMARY SLIS
User-facing API	Availability, latency
Background job	Completion rate, freshness
Data pipeline	Freshness, correctness
Storage	Durability, availability

MULTI-WINDOW SLO

- → **30-day:** Long-term reliability view
- → **7-day:** Recent trend indicator
- → **1-day:** Acute issue detection
- → **1-hour:** Real-time burn rate

Alert on short windows; report on long windows

IMPLEMENTATION CHECKLIST

STEP	ACTION
1	Identify critical user journeys
2	Define SLIs for each journey
3	Set initial targets (start conservative)
4	Implement measurement & dashboards
5	Create error budget alerts
6	Establish review cadence

KEY INSIGHT

> *100% is the wrong target. Choose the reliability that balances user happiness with development velocity.*

SLOs Enable Decisions

Error budgets are the currency of reliability.

Observability Mastery

Three Pillars, Observability 2.0 & Modern Instrumentation

Observability | Technical Operations Excellence

3	30s	OTel	2.0
CLASSIC PILLARS	TARGET MTTD	STANDARD	NEW PARADIGM

THREE CLASSIC PILLARS

Metrics
Aggregated counts, gauges, histograms. Tools: Prometheus, InfluxDB

Logs
Discrete events, timestamps, stack traces. Tools: Loki, Elasticsearch

Traces
Request flow, spans, latency breakdown. Tools: Tempo, Jaeger

OPENTELEMETRY STANDARD

SIGNAL	STATUS	FEATURE
Traces	Stable	W3C context
Metrics	Stable	Temporality
Logs	Stable	Trace correlation
Profiling	Exp	Continuous

`traceparent: {version}-{trace-id}-{parent-id}-{flags}`

OBSERVABILITY 2.0

> *Observability is about asking arbitrary questions without shipping new code.*
>
> **- Charity Majors, Honeycomb**

CLASSIC (1.0)	MODERN (2.0)
Pre-defined metrics	Arbitrary queries
Three separate pillars	Wide structured events
Dashboard-driven	Exploration-driven
Known unknowns	Unknown unknowns

CARDINALITY & COST

> *You can't run observability infra the same size as production.*
>
> **- Liz Fong-Jones**

ISSUE	IMPACT	FIX
High cardinality	Query slowness	Aggregation
Unbounded labels	OOM, index boom	Label policies

Danger: user_id, request_id in labels

USE METHOD

For every resource: **Utilization, Saturation, Errors**

RESOURCE	U	S	E
CPU	%busy	Run queue	Errors
Memory	%used	Swap I/O	OOM
Disk	%util	Queue	SMART
Network	BW	Retrans	Errors

RED METHOD

For every service: **Rate, Errors, Duration**

SIGNAL	METRIC	QUESTION
Rate	req/sec	How busy?
Errors	fail/sec	Breaking?
Duration	latency	How slow?

SAMPLING STRATEGIES

TYPE	WHEN	TRADE-OFF
Head	At request start	Fast, may miss errors
Tail	After completion	Smart, more overhead
Adaptive	Dynamic rate	Best of both

Always sample 100% of errors and slow requests

INSTRUMENTATION BEST PRACTICES

- → **Standardize naming:** Consistent metric/span names
- → **Add context:** Include service, env, version labels
- → **Correlate signals:** Trace IDs across logs/metrics
- → **Sample wisely:** 100% errors, sample successes

Instrument at code level, not just infrastructure

Debug Unknown Unknowns

Instrument first, decide what to alert later.

Multi-Window Alerting

Burn Rates, SLO-Based Alerts & Alert Attributes

Observability Deep Dive | Technical Operations Excellence

14.4x	4	<2	<5%
CRITICAL BURN RATE	ALERT ATTRIBUTES	PAGES/WEEK TARGET	FALSE POSITIVES

MULTI-WINDOW BURN RATE ALERTS

TYPE	BUDGET	WINDOW	BURN
Page (Critical)	2% / 1h	1h + 5m	14.4x
Page (High)	5% / 6h	6h + 30m	6x
Ticket	10% / 3d	72h + 6h	1x

Dual windows prevent alert flapping while catching fast burns

SYMPTOM VS CAUSE HIERARCHY

User-Facing Symptoms
Error rate, latency, availability - PAGE these

System Symptoms
Queue depth, connection pool - NOTIFY these

Underlying Causes
CPU, memory, disk - TICKET or LOG these

FOUR ALERT ATTRIBUTES

ATTRIBUTE	DEFINITION	GOAL
Precision	% genuine alerts	Minimize FPs
Recall	% incidents caught	Catch all issues
Detection	Time to notify	Alert quickly
Reset	Time to resolve	Auto-clear

HEALTHY ON-CALL METRICS

METRIC	TARGET
Pages per week	<2
False positive rate	<5%
Off-hours pages	<1
Actionable %	>95%

BURN RATE FORMULA

> *Burn Rate = (1 - SLO) / Time Window*
> *14.4x = consume 30-day budget in ~2 days*

BURN RATE	BUDGET EXHAUSTION
>5%/day	Immediate incident
2-5%/day	Investigation needed
<2%/day	Normal ops

NOISE REDUCTION TECHNIQUES

- → **Aggregation:** Group related alerts
- → **Suppression:** Mute during maintenance
- → **Deduplication:** Same incident once
- → **Auto-escalation:** After X minutes
- → **Alert correlation:** Link to root cause

ALERT CATEGORIES

CATEGORY	RESPONSE	WHEN
Page	Immediate	User impact
Notify	Hours	Degradation
Ticket	Next day	Slow drift
Log	Review	Informational

GOLDEN RULES

- → **Actionable:** Can I do something?
- → **Urgent:** Does it need attention now?
- → **Real:** Is this actually happening?
- → **Human judgment:** Does this need a person?

Alert on Symptoms
Page for user pain, ticket for slow burns.

USE Method Performance

Utilization, Saturation, and Errors

Observability | Technical Operations Excellence

3	7	<80%	0
CORE METRICS	RESOURCE TYPES	UTILIZATION TARGET	IDEAL SATURATION

USE METHOD DEFINED

METRIC	DEFINITION	TARGET
Utilization	% time resource is busy	<80%
Saturation	Queued work beyond capacity	0
Errors	Error count/rate	0

Created by Brendan Gregg for systematic resource analysis

RESOURCE TYPES

RESOURCE	U METRIC	S METRIC
CPU	% busy	Run queue length
Memory	% used	OOM events, swap
Disk I/O	% busy	Queue depth
Network	% bandwidth	Drop/retransmit
Storage	% capacity	Out of space
Threads	Pool usage	Blocked threads
File Handles	Open FDs	FD exhaustion

USE VS RED VS GOLDEN SIGNALS

METHOD	FOCUS	BEST FOR
USE	Resources	Infrastructure, VMs
RED	Requests	Services, APIs
Golden Signals	User experience	Customer-facing

KEY INSIGHT

> *For every resource, check utilization, saturation, and errors. Start here for performance issues.*
>
> \- Brendan Gregg

COMMON PROMETHEUS QUERIES

METRIC	QUERY PATTERN
CPU Util	`rate(cpu_seconds[5m])`
Mem Util	`used / total * 100`
Disk Util	`rate(io_time[5m])`
Net Util	`rate(bytes[5m]) / bw`

PROFILING & FLAME GRAPHS

- → **CPU flame graph:** Where time is spent
- → **Off-CPU flame:** What code is waiting
- → **Memory flame:** Allocation patterns
- → **Differential flame:** Before/after comparison

Tools: perf, bcc, bpftrace, async-profiler, pprof

PERFORMANCE ANTI-PATTERNS

ISSUE	SYMPTOM
Resource leak	Gradual degradation
Lock contention	High CPU, low throughput
Thundering herd	Bursty overload
N+1 queries	Linear database calls

SATURATION INDICATORS

- → **Run queue > cores:** CPU saturation
- → **Swap active:** Memory saturation
- → **Disk queue > 1:** I/O saturation
- → **TCP retransmits:** Network saturation

Measure First

Never guess; always profile before optimizing.

Observability 2.0

Wide Events, High Cardinality, and Beyond the Three Pillars

Observability | Technical Operations Excellence

100s	10^6	1	<10s
FIELDS PER EVENT	HIGH CARDINALITY	UNIFIED FORMAT	QUERY RESPONSE

OBSERVABILITY 1.0 VS 2.0

ASPECT	1.0	2.0
Data	3 pillars (siloed)	Wide structured events
Cardinality	Low (pre-aggregated)	High (millions)
Questions	Known unknowns	Unknown unknowns
Debug	Correlate across tools	Single pane of glass

CORE PRACTICES

Instrument Everything
Every service emits structured events on every request

Query Interactively
Ad-hoc questions, slice and dice by any field

SLO Integration
Events feed SLI calculations directly

WIDE STRUCTURED EVENTS

> *Emit one wide event per unit of work, with all relevant context attached.*
>
> - Charity Majors

- → **Request context:** user_id, tenant_id, request_id
- → **Timing:** duration, queue_time, db_time
- → **Result:** status, error_type, cache_hit
- → **Environment:** version, host, region, pod

EVENT SCHEMA EXAMPLE

FIELD	EXAMPLE VALUE
service	api-gateway
endpoint	/v2/users/:id
duration_ms	47.3
status_code	200
user_id	u_abc123
cache_hit	true
db_queries	3

HIGH CARDINALITY FIELDS

FIELD	CARDINALITY
user_id	Millions
trace_id	Billions
request_id	Billions
build_id	Thousands
endpoint	Hundreds

Traditional metrics explode with high cardinality

TOOLS FOR OBSERVABILITY 2.0

TOOL	STRENGTH
Honeycomb	Query-first, high cardinality
Grafana + Loki	Open-source ecosystem
OpenTelemetry	Vendor-neutral instrumentation

KEY QUESTION

> *Can you debug problems you've never seen before, without adding new instrumentation?*

CHARITY MAJORS PRINCIPLES

- → Observability is about understanding *new* problems
- → Debug from production, not staging
- → Instrument at the code level, not infrastructure
- → Exploratory investigation over dashboards

Ask New Questions

True observability answers questions you haven't thought to ask yet.

Alert Tuning Playbook

Reducing Noise, Improving Signal

Observability | Technical Operations Excellence

<2	<5%	80%	30s
PAGES PER SHIFT	FALSE POSITIVE RATE	ACTIONABLE TARGET	MTTD GOAL

ALERT QUALITY FRAMEWORK

QUALITY	CRITERIA
Actionable	Requires immediate human action
Symptom-based	Alerts on user impact, not causes
Timely	Detects issues within SLO window
Prioritized	Clear severity levels
Documented	Linked to runbooks

NOISE REDUCTION STRATEGIES

Aggregate Related
Group alerts by service or component

Adjust Thresholds
Based on historical data and SLOs

Add Hysteresis
Require sustained violations to fire

SYMPTOM VS CAUSE ALERTS

TYPE	EXAMPLE	PAGE?
Symptom	Error rate >1%	Yes
Symptom	Latency p99 >500ms	Yes
Cause	CPU >80%	Notify only
Cause	Disk >90%	Ticket

Page on symptoms; ticket causes for investigation

SEVERITY LEVELS

SEVERITY	RESPONSE	EXAMPLE
P1	Page, escalate	Service down
P2	Page, working hours	Degraded service
P3	Ticket, next day	Non-critical issue
P4	Ticket, backlog	Improvement

BURN RATE ALERTING

WINDOW	BURN RATE	ACTION
1 hour	14.4x	Page immediately
6 hours	6x	Page
24 hours	3x	Ticket
72 hours	1x	Review weekly

Burn rate = (1 - SLI) / (1 - SLO target)

ALERT REVIEW CADENCE

ACTIVITY	FREQUENCY
Alert review	Weekly
Threshold tuning	Monthly
Alert inventory	Quarterly
Delete unused	Quarterly

GOLDEN RULE

> *Every alert should either require immediate action or be deleted.*

ALERT FATIGUE INDICATORS

- → >2 pages per on-call shift
- → >5% false positive rate
- → Same alert firing repeatedly
- → Engineers ignoring alerts
- → No runbook links

Signal Over Noise

The best alert is one that never fires unnecessarily.

Resilience Patterns

Circuit Breakers, Bulkheads & Graceful Degradation

Resilience Patterns | Technical Operations Excellence

N+2 REDUNDANCY

3 CIRCUIT STATES

<1s TIMEOUT TARGET

5 DEFENSE LAYERS

CIRCUIT BREAKER PATTERN

Prevents cascading failures by stopping requests to failing services.

Closed
Normal operation, requests pass through

Open
Requests blocked, return fallback immediately

Half-Open
Limited test requests to check recovery

BULKHEAD PATTERN

> *Like watertight compartments in ships - isolate failures to prevent sinking.*

TYPE	MECHANISM	USE CASE
Thread Pool	Dedicated pool	Isolate slow deps
Semaphore	Concurrency limit	Lightweight isolation

GRACEFUL DEGRADATION

Reduce work or quality to maintain availability during failures.

STRATEGY	EXAMPLE
Quality Reduction	Lower image resolution
Feature Shedding	Disable recommendations
Subset Query	Search cache only
Default Response	Return static content

RETRY WITH BACKOFF

```
delay = min(maxBackoff, base * 2^attempt + jitter)
```

→ **Do:** Add jitter, cap max delay, limit attempts
→ **Don't:** Retry non-idempotent ops, nest retries

LOAD BALANCING: L4 VS L7

ASPECT	L4 (TRANSPORT)	L7 (APPLICATION)
Routing	IP + Port	HTTP headers, URLs
Latency	10-100 μs	0.5-3 ms
CPU	Low	High (TLS)
Best For	DDoS, non-HTTP	Smart routing

Production: Layer both (L4 edge → L7 internal)

TIMEOUT STRATEGY

TYPE	TYPICAL VALUE
Connect	250ms - 1s
Header	5 - 30s
Idle	30 - 300s

Critical: Timeouts DECREASE deeper in call chain

DEFENSE IN DEPTH

Multiple independent layers - no single layer is exclusively relied upon.

1. → Prevention of abnormal operation
2. → Control of abnormal operation
3. → Control within design basis
4. → Control of severe conditions
5. → Mitigation of consequences

CASCADING PREVENTION

PATTERN	PURPOSE
Timeouts	Bound waiting time
Bulkheads	Isolate resources
Load Shedding	Reject before instability
Deadlines	Propagate time limits

Fail Fast, Recover Faster
Every pattern protects downstream dependencies.

Defense in Depth

5-Layer Model, Compartmentalization & Blast Radius Control

Resilience & Infrastructure | Technical Operations Excellence

5	3	N+2	0
DEFENSE LAYERS	COMPARTMENTS	REDUNDANCY	SINGLE POINTS

5-LAYER DEFENSE MODEL

LAYER	FUNCTION	EXAMPLE
1. Perimeter	Edge protection	WAF, firewall
2. Network	Segmentation	VLANs, VPCs
3. Host	Hardening	Patches, config
4. Application	Code security	Input validation
5. Data	Encryption	At rest, in transit

Multiple layers must fail for a breach to succeed

REDUNDANCY PATTERNS

PATTERN	DESCRIPTION
N+1	One spare for failover
N+2	Two spares (for critical systems)
Active-Active	All replicas serve traffic
Active-Passive	Standby on failover

N+2 for tier-0 critical systems

ADVANCED AUTHORIZATION

- → **MPA:** Multi-party approval for sensitive ops
- → **Temporary Access:** Time-bound permissions
- → **Business Justification:** Tie to tickets/incidents
- → **Breakglass:** Emergency override with audit

COMPARTMENTALIZATION STRATEGIES

STRATEGY	DESCRIPTION
Role Separation	Different jobs run as distinct accounts
Location Separation	Geographic isolation (multi-region)
Time Separation	Key rotation forces continuous presence

DESIGNING FOR RECOVERY

PRINCIPLE	APPLICATION
Go fast, guarded	Speed with policy guardrails
Minimize time deps	Don't wait for wall-clock
Know intended state	Encode complete config
Emergency access	Works when systems fail

BLAST RADIUS CONTROL

Failure Domains
Partition into independent copies

Circuit Breakers
Stop cascading failures at boundaries

Bulkheads
Isolate resource pools per tenant/service

ZERO SINGLE POINTS OF FAILURE

- → Every component has a backup
- → Every process has redundancy
- → Every region has failover
- → Every credential has rotation

ACCESS CLASSIFICATION

TIER	DATA TYPE	CONTROLS
Public	Company-wide	Low-risk
Sensitive	Authorized only	Medium-high
Highly Sensitive	No permanent access	MPA required

Assume Breach

Design so attackers must breach ALL layers.

HRO Pattern Recognition

Learning from High-Reliability Organizations

Resilience Patterns | Technical Operations Excellence

5 HRO PRINCIPLES

10 ROOT CAUSE CATEGORIES

4 SWISS CHEESE LAYERS

10^{-6} AVIATION ERROR RATE

5 HRO PRINCIPLES DEEP DIVE

PRINCIPLE	APPLICATION
Preoccupation with Failure	Treat near-misses as failures; never assume safety
Reluctance to Simplify	Resist simple explanations; embrace complexity
Sensitivity to Operations	Maintain situational awareness at all times
Commitment to Resilience	Focus on recovery, not just prevention
Deference to Expertise	Authority migrates to knowledge in crisis

PATTERN RECOGNITION TABLE

SIGNAL	PATTERN	ACTION
Latency spike	Capacity/Dependency	Scale or isolate
Error burst	Deploy/Config	Rollback
Gradual degrade	Resource leak	Restart/investigate
Cascading fail	Missing circuit breaker	Shed load
Partial outage	Network partition	Failover

HRO VS TRADITIONAL ORGS

ASPECT	TRADITIONAL	HRO
Failures	Hide/blame	Learn/share
Complexity	Simplify away	Embrace
Authority	Hierarchy	Expertise
Focus	Efficiency	Reliability

BIG 10 ROOT CAUSES

#	CATEGORY	EXAMPLE
1	**Config Change**	Bad deploy, wrong flag
2	**Capacity**	Resource exhaustion
3	**Dependency**	Upstream/downstream fail
4	**Hardware**	Disk, network, memory
5	**Security**	Attack, credential leak
6	**Human Error**	Typo, wrong command
7	**Software Bug**	Race condition, logic error
8	**Data**	Corruption, schema drift
9	**Network**	Partition, DNS, latency
10	**External**	Cloud provider, 3rd party

INDUSTRIES WE LEARN FROM

INDUSTRY	KEY PRACTICE
Aviation	Checklists, crew resource mgmt
Nuclear	Defense in depth, safety culture
Healthcare	Root cause analysis, just culture
Military	After-action reviews, command

FAILURE TAXONOMY

→ **Active failures:** Immediate triggers (human error)
→ **Latent conditions:** Dormant system weaknesses
→ **Error-provoking:** Conditions that invite mistakes

SWISS CHEESE MODEL

> *Accidents occur when holes in multiple defense layers momentarily align.*
>
> \- James Reason

→ **Layer 1:** Organizational controls
→ **Layer 2:** Technical safeguards
→ **Layer 3:** Monitoring & detection
→ **Layer 4:** Human operators

Failures Are Teachers

Every incident is a window into system weaknesses.

Sources: Weick & Sutcliffe, James Reason, NASA, NTSB

Release It! Patterns

Stability Patterns for Production Systems

Resilience Patterns | Technical Operations Excellence

15	12	2007	5s
STABILITY PATTERNS	ANTI-PATTERNS	FIRST EDITION	TIMEOUT DEFAULT

KEY STABILITY PATTERNS

PATTERN	PURPOSE
Circuit Breaker	Stop cascading failures
Bulkhead	Isolate failures to partitions
Timeout	Prevent indefinite waits
Retry	Handle transient failures
Fallback	Graceful degradation
Shed Load	Reject excess traffic
Handshaking	Verify capacity before work

STABILITY ANTI-PATTERNS

ANTI-PATTERN	RISK
Integration Points	Every call is a risk
Chain Reactions	One failure cascades
Cascading Failures	Avalanche effect
Users	Unpredictable traffic
Blocked Threads	Thread pool exhaustion
Unbounded Queues	Memory exhaustion

MORE ANTI-PATTERNS

ANTI-PATTERN	RISK
Self-Denial	Marketing DDos
Unbalanced Capacity	Bottleneck fails first
Slow Responses	Worse than no response
SLA Inversion	Depend on weaker SLA

CIRCUIT BREAKER STATES

STATE	BEHAVIOR
Closed	Normal operation, count failures
Open	Fast fail, don't call downstream
Half-Open	Test with limited traffic

Thresholds: 5 failures, 30s timeout, 1 test request

TIMEOUT GUIDELINES

TYPE	RECOMMENDATION
Connect	1-3 seconds
Read	5-30 seconds
Total	Max acceptable latency

Always set timeouts! Never use language defaults.

BULKHEAD STRATEGIES

- → **Thread pool isolation:** Separate pools per dependency
- → **Semaphore isolation:** Limit concurrent requests
- → **Process isolation:** Separate containers/pods
- → **Network isolation:** Separate subnets

KEY QUOTE

> *Every integration point will eventually fail in some way.*
>
> \- Michael Nygard, Release It!

MORE STABILITY PATTERNS

PATTERN	USE CASE
Steady State	Self-cleaning logs/data
Test Harness	Simulate bad behaviors
Decoupling	Async via queues
Fail Fast	Check prereqs early

Expect Failure

Design for failure; plan for success.

Chaos Engineering

Principles, Experiments & GameDay Practices

Resilience Patterns | Technical Operations Excellence

5	4	2011	0
MATURITY LEVELS	EXPERIMENT PHASES	CHAOS MONKEY BORN	INCIDENTS DURING

CHAOS ENGINEERING PRINCIPLES

PRINCIPLE	DESCRIPTION
Hypothesis	Define steady state & expected behavior
Vary Real-World	Simulate production conditions
Run in Prod	Staging doesn't catch all issues
Automate	Continuous experimentation
Minimize Blast	Start small, abort on harm

SAFETY REQUIREMENTS

Abort Conditions
Define clear stop criteria before starting

Blast Radius
Limit scope; start with 1% of traffic

Rollback Plan
Instant recovery must be ready

EXPERIMENT DESIGN

PHASE	ACTIONS
1. Hypothesis	Define steady state metrics
2. Design	Choose failure injection type
3. Execute	Run with monitoring active
4. Analyze	Compare results to hypothesis

GAMEDAY FORMAT

TIME	ACTIVITY
0:00	Brief team, review hypothesis
0:15	Start observability baseline
0:30	Inject failure
1:00	Observe, document behaviors
1:30	Stop injection, verify recovery
2:00	Debrief, document findings

MATURITY MODEL

LEVEL	CAPABILITY
1. Ad-hoc	Manual, sporadic testing
2. Basic	Simple failure injection
3. Repeatable	Documented experiments
4. Automated	CI/CD integrated chaos
5. Optimized	Continuous chaos in prod

GAMEDAY ROLES

ROLE	RESPONSIBILITY
Facilitator	Run experiment, track time
Observer	Monitor dashboards
Scribe	Document findings
Safety Officer	Call abort if needed

SAFETY CHECKLIST

- → ☐ Abort conditions defined
- → ☐ Rollback plan documented
- → ☐ Blast radius limited (<10% traffic)
- → ☐ Monitoring dashboards open
- → ☐ Stakeholders notified

10 CORE EXPERIMENTS

EXPERIMENT	TESTS
Instance Kill	Auto-recovery, failover
Zone Failure	Multi-AZ resilience
Network Latency	Timeout handling
Packet Loss	Retry logic
Dependency Down	Circuit breakers

Also: CPU stress, memory pressure, disk fill, DNS failure, clock skew

Fail Safely

Better to find weaknesses before your customers do.

Incident Excellence

ITIL Lifecycle, Blameless Postmortems & On-Call Sustainability

Incident Management | Technical Operations Excellence

5	≤2	3Cs	48h
ITIL PHASES	PAGES/SHIFT	IMAG FRAMEWORK	POSTMORTEM SLA

ITIL INCIDENT LIFECYCLE

1. Identify
Detection via monitoring, alerts, or reports

2. Categorize
Classify by type, service, impact area

3. Prioritize
Assign SEV level based on impact + urgency

4. Respond
Diagnose, mitigate, resolve, communicate

5. Close
Verify, document, postmortem, action items

SEVERITY LEVELS

LEVEL	IMPACT	RESPONSE
SEV1	Critical outage	<15 min
SEV2	Major degradation	<30 min
SEV3	Minor impact	<4 hours
SEV4	Low/cosmetic	Next business day

IMAG FRAMEWORK (3CS)

PRINCIPLE	ACTIONS
Coordinate	IC assigns roles, manages workstreams
Communicate	Status updates, stakeholder briefs
Control	Authorize changes, manage scope

Crisis triage: data criticality, trust relationships, compensating controls

CRISIS TRIAGE QUESTIONS

- → **Data criticality:** What can be accessed from compromised systems?
- → **Trust relationships:** What other systems trust the affected one?
- → **Compensating controls:** Are there mitigations in place?
- → **Blast radius:** How many users/services affected?

INCIDENT ROLES

ROLE	RESPONSIBILITY
Incident Commander	Owns resolution, delegates
Ops Lead	Technical investigation
Comms Lead	Stakeholder updates
Scribe	Documents timeline

SEV1/2: Add Remediation Lead, Legal (if needed)

BLAMELESS POSTMORTEMS

> *Ask "what" and "how" questions, never "why" - it forces justification and blame.*
>
> **- John Allspaw, Etsy**

- → **Timeline:** What happened, when?
- → **Contributing factors:** What conditions existed?
- → **Action items:** Preventative, detective, mitigating

COMMUNICATION CADENCE

SEVERITY	UPDATE FREQUENCY
SEV1	Every 15 minutes
SEV2	Every 30 minutes
SEV3/4	Hourly or as needed

Playbooks improve MTTR by 3x on average

TRAINING: WHEEL OF MISFORTUNE

Role-play exercise for IC practice. Spin wheel to select historic incident, responders handle in real-time simulation.

- → **Do:** Practice handoffs, escalation
- → **Don't:** Use for evaluation/blame

Learn from Every Incident

Blameless culture enables honest retrospectives.

Learning from Catastrophe

Swiss Cheese Model, Big 10 Root Causes & Pattern Recognition

Historic Incidents | Technical Operations Excellence

50+ INCIDENTS ANALYZED | **40%** CONFIG/DEPLOY ERRORS | **$10B+** CROWDSTRIKE DAMAGE | **4** DEFENSE LAYERS

BIG 10 ROOT CAUSES

#	ROOT CAUSE	FREQ
1	**Config/Deploy Errors**	~40%
2	**Ignored Warnings**	High
3	**Single Point of Failure**	High
4	**Inadequate Testing**	High
5	**Simple Bugs at Scale**	High
6	**Monitoring Gaps**	Med
7	**Complex Interdependencies**	Med
8	**Human Error Under Pressure**	Med
9	**Vendor/3rd Party Failures**	Med
10	**Legacy System Fragility**	Med

SWISS CHEESE MODEL

`Hazard → [Prevention] → [Detection] → [Containment] → [Recovery] → Accident`

LAYER	IF HOLE
Prevention	Near miss
Detection	Degradation
Containment	Incident
Recovery	Catastrophe

Key: Catastrophic failures require ALL layers to fail simultaneously

CROWDSTRIKE CASE STUDY (2024)

- → **Impact:** $10B+ damages, 8.5M Windows systems
- → **Root Cause:** Content update bypassed validation
- → **Kernel driver:** Single point of failure

Lesson: Staged rollouts essential for security updates

NOTABLE INCIDENTS

INCIDENT	ROOT CAUSE	LESSON
GitLab	Config error	Staged rollouts
737 MAX	Single PoF	Redundancy
Knight Capital	Bug at scale	Code review
Therac-25	Bad testing	Integration tests

MITIGATIONS BY ROOT CAUSE

CAUSE	MITIGATION
Config errors	Canaries, staged rollouts
Ignored warnings	Safety culture, incentives
Single PoF	Redundancy, chaos testing
Testing gaps	Comprehensive coverage
Dependencies	Dependency mapping

CROSS-INDUSTRY LESSONS

- → **Aviation:** Crew resource management
- → **Nuclear:** Defense in depth
- → **Healthcare:** Checklists, near-miss reporting
- → **Finance:** Circuit breakers, kill switches

PATTERN RECOGNITION

> *Every catastrophe is a near-miss that was ignored.*

Defense in Depth

Build redundant, independent defenses at every layer.

Runbook Quick Reference

Templates, Decision Trees, and MTTR Targets

Incident Management | Technical Operations Excellence

10	<5min	<1hr	80%
RUNBOOK TEMPLATES	TRIAGE TARGET	MTTR TARGET	RUNBOOK COVERAGE

10 ESSENTIAL RUNBOOK TYPES

#	RUNBOOK	MTTR TARGET
1	**Service Restart**	5 min
2	**Deployment Rollback**	10 min
3	**Database Failover**	15 min
4	**Cache Clear**	5 min
5	**Traffic Shift**	10 min
6	**Scale Out**	5 min
7	**Certificate Rotation**	15 min
8	**DNS Update**	10 min
9	**Feature Flag Toggle**	2 min
10	**Emergency Access**	5 min

VERIFICATION CHECKLIST

CHECK	HOW
Service healthy	Health endpoint returns 200
Metrics normal	Grafana dashboards green
Errors stopped	Error rate below threshold
Latency normal	p99 within SLO
Logs clean	No error spikes in logs

DECISION TREE: ERRORS SPIKE

→ **Check:** Error type?
- → 5xx → Server-side issue
- → 4xx → Client or config issue

→ **Check:** Pattern?
- → Sudden spike → Deployment or config
- → Gradual → Resource exhaustion

→ **Check:** Scope?
- → One endpoint → Check that handler
- → All endpoints → Check infrastructure

RUNBOOK STRUCTURE

SECTION	CONTENT
Overview	What this runbook addresses
Symptoms	How to recognize the issue
Prerequisites	Required access & tools
Steps	Numbered procedure
Verification	How to confirm success
Rollback	If things go wrong
Escalation	Who to contact next

RUNBOOK QUALITY CRITERIA

Testable
Can be verified in staging/DR drills

Automatable
Steps are scriptable for future automation

Measurable
Includes timing targets and success criteria

DECISION TREE: HIGH LATENCY

→ **Check:** Is it a single service or all?
- → Single → Check that service's resources
- → All → Check shared dependencies (DB, cache)

→ **Check:** Recent deployment?
- → Yes → Consider rollback
- → No → Check traffic levels

→ **Check:** Resource exhaustion?
- → Yes → Scale or restart
- → No → Check network, dependencies

QUICK COMMANDS

ACTION	EXAMPLE
Pod restart	`kubectl rollout restart`
Rollback	`kubectl rollout undo`
Scale	`kubectl scale --replicas`

Document to Automate

Today's runbook is tomorrow's automation.

Capacity & Release Engineering

DORA Metrics, Progressive Delivery & Safe Changes

Capacity & Release | Technical Operations Excellence

4	<5%	1-5%	<1h
DEPLOY STRATEGIES	ELITE CFR	CANARY SIZE	ELITE LEAD TIME

RELEASE PERFORMANCE TARGETS

METRIC	ELITE TARGET
Deploy Frequency	On-demand (multiple/day)
Lead Time	<1 hour commit to prod
Change Fail Rate	<5% of deploys cause issues
Time to Restore	<1 hour to recover

Based on DORA research: elite performers achieve 182x higher deploy frequency

DEPLOYMENT STRATEGIES

Canary
Route 1-5% traffic to new version, monitor, expand gradually

Blue-Green
Two identical envs, instant switchover, easy rollback

Feature Flags
Decouple deploy from release, targeted rollouts

CANARY BEST PRACTICES

- → **One at a time:** Avoid signal contamination
- → **5-12 metrics:** Monitor error rate, latency, saturation
- → **Absolute thresholds:** Define rollback criteria upfront
- → **Bake time:** Allow sufficient observation window

40%+ of incidents stem from config/deployment errors

PROGRESSIVE DELIVERY

`Commit → CI/CD → Canary (1-5%) → Rollout → Full Deploy`

STAGE	GATE
Build	Tests pass, security scan
Canary	Error budget not exceeded
Rollout	Metrics within thresholds

NALSD FRAMEWORK

Non-Abstract Large System Design - 4 essential questions:

QUESTION	FOCUS
Is it possible?	Can we build it at all?
Can we do better?	Optimize design choices
Is it feasible?	Cost, time, resources
Is it resilient?	Graceful degradation

CAPACITY PLANNING

COMPONENT	APPROACH
Demand Forecast	Historical trends + growth models
Headroom	N+1 minimum, N+2 for critical
Load Testing	Regular stress tests at 2x expected
Auto-scaling	HPA/VPA with proper limits

CHANGE RISK CATEGORIES

TIER	EXAMPLES	PROCESS
Low	Config, docs	Auto-deploy
Medium	App code	Canary + review
High	Infra, DB schema	Change board

LAUNCH CHECKLIST

- → ✓ SLOs defined and dashboards ready
- → ✓ Runbooks documented
- → ✓ Rollback procedure tested
- → ✓ On-call coverage confirmed
- → ✓ Load test completed

Ship Fast, Ship Safe

Elite teams deploy frequently with low failure rates.

NALSD Framework

Non-Abstract Large System Design

Capacity & Release | Technical Operations Excellence

4	N+2	2x	30d
ESSENTIAL QUESTIONS	HEADROOM TARGET	LOAD TEST TARGET	FORECAST WINDOW

THE 4 ESSENTIAL QUESTIONS

1. Is it possible?
Can we build it at all?

2. Can we do better?
Optimize design choices

3. Is it feasible?
Cost, time, resources

4. Is it resilient?
Graceful degradation

CAPACITY METRICS

METRIC	TARGET
CPU Utilization	<70% avg, <90% peak
Memory	<80% avg, <95% peak
Disk I/O	<70% queue depth
Network	<60% bandwidth

Leave headroom for traffic spikes and incidents

CAPACITY PLANNING PROCESS

STEP	ACTIVITY
1. Demand Forecast	Historical trends + growth models
2. Supply Analysis	Current capacity, bottlenecks
3. Gap Assessment	Where will we run out?
4. Headroom Planning	N+1 min, N+2 for critical

SCALING STRATEGIES

TYPE	WHEN TO USE
Vertical	Simple, single-instance
Horizontal	Stateless, distributed
Auto-scaling	Variable traffic patterns
Predictive	Known events (launches)

FORECASTING INPUTS

→ **Historical trends:** Past 90+ days growth
→ **Seasonality:** Day/week/month patterns
→ **Business events:** Launches, campaigns
→ **External factors:** Market trends

LOAD TESTING STRATEGY

TEST TYPE	PURPOSE	TARGET
Baseline	Normal load	Current traffic
Stress	Find limits	2x expected
Spike	Sudden surge	10x for 30s
Soak	Leaks, drift	24-48 hours

WARNING SIGNS

→ Utilization >80% sustained
→ P99 latency creeping up
→ Queue depths growing
→ Error rates increasing

DESIGN TRADE-OFFS

DIMENSION	TRADE-OFF
Consistency	vs. Availability (CAP)
Latency	vs. Throughput
Cost	vs. Resilience
Complexity	vs. Maintainability

Plan for 2x
Capacity planning is cheaper than outages.

Designing for Recovery

Recovery Principles, Breakglass & Emergency Access

Infrastructure Reliability | Technical Operations Excellence

3-2-1 BACKUP RULE | **<15m** TIER-0 RTO | **0** TIER-0 RPO | **MPA** MULTI-PARTY AUTH

RECOVERY DESIGN PRINCIPLES

PRINCIPLE	APPLICATION
Go fast, guarded	Speed with policy guardrails
Minimize time deps	Don't wait for wall-clock
Know intended state	Encode complete configuration
Test restores	Untested backups = no backups

BREAKGLASS PROCEDURES

MECHANISM	PURPOSE
Breakglass	Override normal access controls
MPA	Multi-party authorization
Offline creds	Independent of primary systems
Temp access	Time-bounded elevation

Document business justification for all elevated access

3-2-1 BACKUP STRATEGY

3 Copies
Original + 2 backups minimum

2 Media Types
Different storage technologies

1 Offsite
Geographic separation

EMERGENCY ACCESS MUST-HAVES

- → **Work when systems fail:** Independent channel
- → **Pre-staged credentials:** Not just-in-time during crisis
- → **Tested regularly:** Part of disaster drills
- → **Audit trail:** All access logged

DISASTER VALIDATION

EXERCISE	FREQUENCY
Tabletop	Monthly
Failover drill	Quarterly
Full DR test	Annually
Chaos experiments	Continuous

RTO & RPO TARGETS

TIER	SYSTEMS	RTO	RPO
0	Critical APIs	<15m	0
1	Core services	<4h	<1h
2	Internal tools	<24h	<4h
3	Dev/test	<72h	<24h

RECOVERY CHECKLIST

- → ✓ Runbooks documented and tested
- → ✓ Contact list current
- → ✓ Backup restore verified
- → ✓ Failover procedure practiced

RECOVERY TESTING

- → **Quarterly:** Full restore drill for Tier-0
- → **Monthly:** Point-in-time recovery test
- → **Weekly:** Backup integrity verification
- → **Daily:** Automated backup monitoring

Plan to Fail

The best recovery is the one you've practiced.

Infrastructure Reliability

Kubernetes, Databases, TSDB & Observability Backends

Infrastructure | Technical Operations Excellence

3	N+2	IaC	mTLS
K8S PROBES	REDUNDANCY	GITOPS PATTERN	SERVICE MESH

KUBERNETES RELIABILITY

COMPONENT	PURPOSE	KEY CONFIG
HPA	Scale pods out	CPU/memory targets
VPA	Right-size pods	updateMode: Off
PDB	Protect availability	minAvailable: 2

HPA + VPA conflict on same metrics - use VPA in recommend-only mode

KUBERNETES PROBES

PROBE	PURPOSE	WHEN
Startup	Container started	First (slow apps)
Liveness	Container running	Catch deadlocks
Readiness	Ready for traffic	Load balancer

Liveness: lightweight checks. Let fatal errors crash, don't restart.

DATABASE RELIABILITY

PATTERN	USE CASE
Read replicas	Scale read traffic
Multi-AZ	HA failover
Sharding	Horizontal scale
Connection pooling	Limit connections

Replication ≠ Backup - corrupt data replicates everywhere

RESOURCE MANAGEMENT

RESOURCE	LIMIT STRATEGY
CPU	Requests = P50, Limits = P99
Memory	Request = Limit (no OOM)
Ephemeral	Limit to prevent node evict

Profile in production to set accurate requests

SECRETS MANAGEMENT

HashiCorp Vault core capabilities:

FEATURE	BENEFIT
Dynamic secrets	Short-lived, on-demand
Encryption as a service	Transit secrets engine
Identity-based access	RBAC, namespaces
Audit logging	SIEM integration

SERVICE MESH

FEATURE	BENEFIT
mTLS	Encrypted service-to-service
Traffic mgmt	Canary, A/B, retries
Observability	Distributed tracing
Circuit breaking	Prevent cascade failures

Start simple; add mesh when complexity justifies overhead

OBSERVABILITY BACKENDS

SIGNAL	OSS STACK	KEY FEATURE
Metrics	Prometheus, Mimir	PromQL, federation
Logs	Loki, OpenSearch	LogQL, labels
Traces	Tempo, Jaeger	Trace correlation

Grafana unifies all three signals in one UI

TIME SERIES DATABASES

TSDB	BEST FOR
InfluxDB	IoT, high cardinality
Prometheus	K8s metrics, alerts
kdb+	Finance, ultra-low latency
VictoriaMetrics	Long-term retention

Cattle, Not Pets

Infrastructure should be reproducible and replaceable.

Kubernetes Patterns

Foundational, Behavioral, and Structural Patterns

Infrastructure | Technical Operations Excellence

4 PATTERN CATEGORIES

25+ DESIGN PATTERNS

2014 K8S RELEASED

92% ENTERPRISE ADOPTION

FOUNDATIONAL PATTERNS

PATTERN	PURPOSE
Health Probe	Liveness, readiness, startup checks
Predictable Demands	Resource requests/limits
Automated Placement	Node selectors, affinity rules
Declarative Deployment	Desired state via manifests

OPERATOR PATTERN

> *Operators encode operational knowledge as software, automating day-2 operations.*

- → **Custom Resource:** Domain-specific API
- → **Controller:** Reconciliation logic
- → **Levels:** Basic install → Full lifecycle

BEHAVIORAL PATTERNS

PATTERN	USE CASE
Batch Job	Run-to-completion workloads
Periodic Job	CronJobs for scheduled tasks
Daemon Service	Per-node agents (logging, monitoring)
Singleton Service	Leader election, exactly one instance
Stateful Service	Ordered, sticky identity (StatefulSet)

RESILIENCE PATTERNS

PATTERN	K8S IMPLEMENTATION
Self-Healing	Restart policy, pod disruption budget
Scaling	HPA, VPA, cluster autoscaler
Rolling Updates	Deployment strategy
Blue-Green	Service selector switch
Canary	Weighted traffic split

STRUCTURAL PATTERNS

PATTERN	DESCRIPTION
Init Container	Setup tasks before main container
Sidecar	Extend without modifying main app
Ambassador	Proxy for external communication
Adapter	Normalize heterogeneous output

SECURITY PATTERNS

PATTERN	IMPLEMENTATION
Least Privilege	RBAC, SecurityContext
Network Isolation	NetworkPolicy
Secret Management	External Secrets Operator
Pod Security	PSS/PSA, read-only root

OBSERVABILITY PATTERNS

- → **Sidecar logging:** Fluentbit, Fluent-bit
- → **Service mesh:** Istio, Linkerd for tracing
- → **Metrics:** Prometheus ServiceMonitor
- → **Events:** K8s event exporter

CONFIGURATION PATTERNS

PATTERN	USE FOR
EnvVar Config	Simple key-value settings
ConfigMap	Non-sensitive config files
Secret	Sensitive data (encrypted)
Immutable Config	Version-pinned configurations

Declarative Operations

Define desired state; let Kubernetes reconcile.

Sources: K8s Patterns (Ibryam & Huss), CNCF

Platform Engineering

Internal Developer Platforms, Golden Paths & Self-Service

Infrastructure | Technical Operations Excellence

80% GOLDEN PATH USE

<10m ENV PROVISION

0 TICKETS TO DEPLOY

IDP INTERNAL PLATFORM

PLATFORM ENGINEERING GOALS

GOAL	OUTCOME
Reduce cognitive load	Devs focus on features
Standardize tooling	Consistency at scale
Self-service	No ticket queues
Paved roads	Easy path for 80% cases

SELF-SERVICE CAPABILITIES

CAPABILITY	NO TICKET
Environment	✓ API/CLI provision
Database	✓ Catalog request
Secrets	✓ Vault self-serve
Monitoring	✓ Auto-instrumented
Domains/TLS	✓ Cert-manager

INTERNAL DEVELOPER PLATFORM

Developer Portal
Backstage, Port, Cortex - service catalog & docs

CI/CD Pipeline
Standardized builds, tests, deployments

Infrastructure Abstraction
Crossplane, Terraform modules, GitOps

PLATFORM MATURITY

LEVEL	CHARACTERISTICS
1. Provisional	Tribal knowledge, manual
2. Managed	Documented, some automation
3. Defined	Self-service, golden paths
4. Optimized	Metrics-driven, evolving

GOLDEN PATHS

Pre-built, tested, supported paths for common tasks:

→ **New service:** Template → CI/CD → observability
→ **Database:** Request → provision → connect
→ **Secrets:** Vault integration → auto-rotation
→ **Deployment:** Git push → canary → production

Optional, not mandatory. Compelling, not mandated.

PLATFORM SUCCESS METRICS

→ **Time to first deploy:** New dev productivity
→ **Golden path adoption:** >80% target
→ **Ticket reduction:** Fewer ops requests
→ **Developer NPS:** Platform satisfaction

ANTI-PATTERNS TO AVOID

→ **Mandated use:** Force kills adoption
→ **No feedback loop:** Building in isolation
→ **Feature bloat:** Too much, too complex
→ **Shadow IT:** Teams route around you

PLATFORM TEAM MODEL

ASPECT	APPROACH
Mindset	Treat devs as customers
Feedback	Regular user research
Roadmap	Based on dev pain points
Success	Adoption rate, not features

Paved Roads, Not Walled Gardens
Make the right way the easy way.

AI/ML Operations

Model Serving, LLM Observability & Drift Detection

AI/ML Operations | Technical Operations Excellence

<1%	>90%	>98%	<5%
HALLUCINATION TARGET	TASK COMPLETION	TOOL ACCURACY	HUMAN ESCALATION

LLM OBSERVABILITY

Traditional observability measures infrastructure. LLM observability measures:

DIMENSION	QUESTION
Behavior	Is the model doing what we expect?
Quality	Are outputs accurate, helpful, safe?
Reasoning	Is the chain-of-thought sound?

BOT PERFORMANCE METRICS

METRIC	TARGET
Task Completion Rate	>90%
Tool Call Accuracy	>98%
Context Utilization	>70%
Hallucination Rate	<1%
Human Escalation	<5%

HALLUCINATION DETECTION

SelfCheckGPT
Sample multiple completions, check consistency. Inconsistent facts = hallucination.

LLM-as-Judge
Use another LLM to evaluate groundedness against retrieved context.

CLAP
Cross-Layer Attention Probing - classifier on model activations (open-source only).

DRIFT DETECTION

TYPE	WHAT TO WATCH
Data Drift	Input distribution shifts
Concept Drift	Relationship changes
Model Drift	Prediction quality decay

Monitor production predictions vs training distribution continuously

CHAIN-OF-THOUGHT MONITORING

ASPECT	QUESTION
Faithfulness	Does CoT reflect actual reasoning?
Verbosity	Is reasoning externalized?
Readability	Can humans understand it?
Necessity	Is CoT required for complexity?

CoT most relevant when task is difficult enough to externalize reasoning

LLM OBSERVABILITY PLATFORMS

PLATFORM	STRENGTH	OSS?
Langfuse	Tracing, evals	Yes
Arize Phoenix	RAG analysis	Yes
LangSmith	LangChain native	No

MODEL SERVING

- → **Canary deploys:** A/B test model versions
- → **Shadow mode:** Compare new vs old without impact
- → **Circuit breakers:** Fallback to cached/simpler model
- → **GPU monitoring:** Utilization, memory, thermals

TRAINING PIPELINES

STAGE	RELIABILITY PRACTICE
Data Ingest	Schema validation, drift checks
Feature Store	Versioning, consistency
Training	Checkpointing, resource limits
Eval	Automated benchmarks, holdouts

Observe the Reasoning

AI reliability requires new observability primitives.

Sources: Langfuse, Arize AI, Datadog LLM Observability

Agentic Operations

AI Agents, Self-Healing Systems & Human-Bot Collaboration

Agentic Operations | Technical Operations Excellence

70%	60%	16%	$32B
AUTO-RESOLUTION	SELF-HEALING BY '26	TRUE AGENTS TODAY	AIOPS MARKET '28

AIOPS EVOLUTION

LEVEL	CAPABILITY	ACTION
Reactive	Respond to incidents	Alert triage, runbooks
Proactive	Prevent incidents	Trend analysis, SLO watch
Predictive	Anticipate issues	Anomaly detection, ML
Autonomous	Self-heal	Auto-remediate, adapt

40% fewer outages + 45% faster MTTR with SRE + AIOps (Gartner)

SELF-HEALING SYSTEMS

> *By 2026, over 60% of large enterprises will have self-healing systems powered by AIOps.*
>
> \- Gartner

→ Restart containers automatically
→ Cycle unhealthy nodes
→ Shift traffic from degraded services
→ Recreate failed pods

AGENT MATURITY REALITY

DEPLOYMENT	TRUE AGENTS
Enterprise	16%
Startups	27%

True agent = LLM that plans, executes, observes feedback, and adapts

40% of agentic initiatives may fail by 2027 due to unclear ROI

PRODUCTIVITY IMPACT

ACTIVITY	AI IMPROVEMENT
Documentation	45-50% faster
Code Generation	35-45% faster
Refactoring	20-30% faster

HUMAN-BOT COLLABORATION

M-Shaped Supervisors
Humans oversee multiple specialized bots, intervening strategically

Tiered Autonomy
Low-risk: full autonomy. High-risk: human approval required

MULTI-AGENT ORCHESTRATION

PATTERN	DESCRIPTION
Puppeteer	Manager bot coordinates specialists
Specialist	Deep expertise in narrow domain
Reviewer	Quality gate before actions
Escalation	Bot-to-bot, then bot-to-human

AUTO-RESOLUTION TARGETS

PHASE	TARGET	SCOPE
Phase 3	70%	Known issues
Phase 5	90%	All incidents

Zero manual escalations for known issues

AI AS AMPLIFIER

"AI magnifies existing organizational strengths and weaknesses. AI adoption improves throughput but also increases delivery instability."

\- DORA Report

Bots as Teammates

Autonomy within guardrails, escalation as exception.

Sources: Gartner AIOps, DORA Report, O'Reilly Signals

People & Culture

Westrum Culture, Team Topologies & Sustainable Operations

People & Culture | Technical Operations Excellence

30%	4	3	<2
GENERATIVE BOOST	TEAM TYPES	INTERACTION MODES	PAGES/SHIFT TARGET

WESTRUM CULTURE TYPES

TYPE	CHARACTERISTICS	PERFORMANCE
Pathological	Power-oriented, fear	Low
Bureaucratic	Rule-oriented, silos	Medium
Generative	Performance-oriented	+30%

Generative cultures: high cooperation, messengers welcomed, failures lead to inquiry

BLAMELESS CULTURE

> *Blameless postmortems focus on systems, not individuals. The goal is learning, not punishment.*
>
> - Google SRE

→ **Psychological safety:** Speak up without fear
→ **Just culture:** Distinguish error from recklessness
→ **Learning reviews:** Focus on "how" not "who"

ON-CALL SUSTAINABILITY

METRIC	TARGET
Pages per shift	<2
Interrupt ratio	<25%
Rotation size	6-8 engineers
Max consecutive days	3-4 days

Burnout risk: >2 pages/night or >25% interrupt work

KEY FRAMEWORKS

FRAMEWORK	CORE CONCEPT
Three Ways	Flow, Feedback, Learning
Team Topologies	4 team types, 3 modes
Five Ideals	Locality, Flow, Improvement
Conway's Law	Teams mirror architecture

See dedicated pages for deep dives on each framework

SRE CORE COMPETENCIES

TECHNICAL	NON-TECHNICAL
Distributed systems	Communication
Observability	Incident command
Automation	Documentation
Networking	Collaboration

TEAM HEALTH SIGNALS

→ **Healthy:** Proactive improvements, low burnout
→ **Warning:** Increasing toil, delayed projects
→ **Unhealthy:** High turnover, reactive only

SRE TEAM MODELS

MODEL	BEST FOR
Centralized	Shared expertise, standards
Embedded	Deep product context
Hybrid	Balance of both approaches

Culture Eats Strategy

Generative culture is the foundation of elite performance.

On-Call Excellence

Sustainable Rotations, Escalation Paths & Alert Quality

People & Culture | Technical Operations Excellence

<2	6-8	<25%	24/7
PAGES/SHIFT	ROTATION SIZE	INTERRUPT RATIO	COVERAGE

HEALTHY ON-CALL METRICS

METRIC	TARGET	WARNING
Pages/shift	<2	>5
Interrupt ratio	<25%	>50%
Night pages	0	>1
False positives	<10%	>30%

High alert volume = burnout risk. Fix alerts, not engineers.

ALERT QUALITY GATES

GATE	REQUIREMENT
Actionable	Clear remediation steps
Urgent	Needs human intervention now
Documented	Runbook link in alert
Tuned	<10% false positive rate

If it doesn't page, make it a ticket. If it's noise, delete it.

ROTATION DESIGN

PARAMETER	RECOMMENDATION
Team size	6-8 engineers minimum
Shift length	Max 3-4 consecutive days
Handoff	Overlapping 30-min window
Shadow period	2 weeks for new members

HANDOFF CHECKLIST

- → ✓ Active incidents briefed
- → ✓ Recent deployments noted
- → ✓ Pending changes flagged
- → ✓ Known issues documented
- → ✓ Contact info verified

BURNOUT PREVENTION

SIGN	INTERVENTION
Dreading shifts	Review alert load
Constant fatigue	Extend rotation gaps
Cynicism	Pair with supportive peer
Avoidance	Temporary rotation break

ESCALATION TIERS

L1: Primary On-Call
First responder, initial triage, known fixes

L2: Secondary/SME
Domain expert, complex issues, escalation

L3: Management
SEV1 coordination, customer comms, exec updates

CONTINUOUS IMPROVEMENT

- → **Weekly:** Review noisy alerts, tune or delete
- → **Monthly:** On-call retrospective
- → **Quarterly:** Rotation structure review

COMPENSATION & FAIRNESS

- → **Comp time:** Time off after heavy shifts
- → **Pay differential:** Extra pay for on-call hours
- → **Equitable rotation:** Fair holiday distribution
- → **Opt-out option:** Accommodations for burnout

Sustainable On-Call

Great on-call is boring on-call. Fix the system, not the people.

Sources: Google SRE, PagerDuty State of On-Call, Honeycomb

Three Ways of DevOps

Flow, Feedback, and Continuous Learning

People & Culture | Technical Operations Excellence

3 CORE PRINCIPLES

4 TYPES OF WORK

5 FIVE IDEALS

2009 DEVOPS MOVEMENT

THE FIRST WAY: FLOW

> *Optimize for fast left-to-right flow from Development to Operations to the customer.*

- → Make work visible
- → Reduce batch sizes
- → Reduce handoffs
- → Identify and elevate constraints
- → Eliminate waste and hardships

THE SECOND WAY: FEEDBACK

> *Enable fast and constant right-to-left feedback at every stage.*

- → Create quality at source
- → Amplify feedback loops
- → Swarm and solve problems
- → Push quality closer to source
- → Stop the line for defects

THE THIRD WAY: LEARNING

> *Create a culture of continual experimentation, learning from success and failure.*

- → Enable organizational learning
- → Institutionalize improvement
- → Transform local discoveries into global
- → Reserve time for improvement
- → Create a safe environment to fail

FOUR TYPES OF WORK

TYPE	PRIORITY
Business Projects	Strategic value
Internal IT Projects	Infrastructure
Changes	Maintenance
Unplanned Work	Minimize!

FIVE IDEALS (UNICORN PROJECT)

IDEAL	MEANING
Locality	Teams own end-to-end
Focus & Flow	Minimize interruptions
Improvement	Daily practice, not events
Safety	Safe to experiment and fail
Customer Focus	Outcomes over output

ANTI-PATTERNS TO AVOID

ANTI-PATTERN	SYMPTOM
Hero Culture	Single person knows system
Wall of Confusion	Dev throws over to Ops
Ticket Queue	Long waits for changes
Change Freeze	Fear of deployments
Blamestorming	Punishing failures

KEY METRICS ALIGNMENT

WAY	KEY METRICS
Flow	Lead time, deploy freq
Feedback	CFR, MTTR, test coverage
Learning	Experiments, postmortems

DEVOPS DEFINITION

> *DevOps is the outcome of applying the most trusted principles from physical manufacturing to IT.*
>
> - The DevOps Handbook

DevOps is a Philosophy

SRE implements DevOps with engineering rigor.

Team Topologies

Organizing Business and Technology Teams

People & Culture | Technical Operations Excellence

4	3	5-9	1968
TEAM TYPES	INTERACTION MODES	IDEAL TEAM SIZE	CONWAY'S LAW

4 FUNDAMENTAL TEAM TYPES

Stream-Aligned Team
Aligned to a single stream of work (product, feature, service)

Platform Team
Provides internal services to reduce cognitive load

Enabling Team
Helps stream-aligned teams adopt new capabilities

Complicated-Subsystem Team
Deep expertise for complex components

TEAM TYPE DISTRIBUTION

TYPE	TYPICAL RATIO
Stream-Aligned	60-80%
Platform	10-15%
Enabling	5-10%
Complicated-Subsystem	0-5%

Stream-aligned should always be the majority

PLATFORM TEAM PRINCIPLES

→ **Self-service:** Teams can provision without tickets
→ **Paved roads:** Easy path for 80% use cases
→ **Optional:** Not mandated, but compelling
→ **Thin interface:** Hide complexity behind APIs
→ **Product mindset:** Treat teams as customers

3 INTERACTION MODES

MODE	WHEN TO USE
Collaboration	Discovery, rapid innovation
X-as-a-Service	Clear API, reduce cognitive load
Facilitating	Coaching, capability building

Warning: Collaboration is expensive; use sparingly

SRE TEAM MODELS

MODEL	TOPOLOGY
Centralized SRE	Enabling + Platform hybrid
Embedded SRE	Part of Stream-Aligned
Hybrid	Core platform + consulting

CONWAY'S LAW

> *Organizations design systems that mirror their own communication structure.*
>
> - Melvin Conway, 1968

Inverse Conway Maneuver: Design teams to get the architecture you want

DUNBAR'S NUMBER

GROUP	SIZE
Close team	5-9 people
Trust group	15 people
Clan/tribe	50 people
Max relationships	150 people

COGNITIVE LOAD TYPES

TYPE	DEFINITION
Intrinsic	Inherent problem complexity
Extraneous	Environmental/tooling noise
Germane	Valuable learning investment

Teams Over Individuals
Minimize cognitive load, maximize flow.

Industry Leaders

Lessons from Google, Netflix, NASA & Beyond

Industry Leaders | Technical Operations Excellence

50%	100s	5	6
GOOGLE ENG CAP	NETFLIX DEPLOYS/DAY	HRO PRINCIPLES	AWS PILLARS

GOOGLE SRE PRINCIPLES

PRINCIPLE	APPLICATION
50% Rule	Max 50% time on ops/toil
Error Budgets	Balance reliability vs velocity
SLO-based	Objective reliability targets
Blameless	Focus on systems, not people

> *class SRE implements interface DevOps*

INDUSTRY BEST PRACTICES

COMPANY	KEY CONTRIBUTION
Amazon	Well-Architected (6 pillars)
Meta	Production Eng, SEV culture
Spotify	Squads/Tribes, golden paths
Toyota	Kaizen, Jidoka, JIT

MISSION-CRITICAL LESSONS

INDUSTRY	LESSON
NASA	Checklists, redundancy, simulation
Aviation	Crew resource mgmt, near-miss analysis
Nuclear	Defense in depth, safety culture
Finance	Ultra-low latency, compliance

NETFLIX CHAOS ENGINEERING

"Avoid failure by failing constantly"

TOOL	WHAT IT DOES
Chaos Monkey	Randomly kills instances
Latency Monkey	Injects network delays
Chaos Gorilla	Simulates AZ failure

2014 AWS outage: 10% of servers affected; Netflix ran uninterrupted

HIGH-RELIABILITY ORGS

5 principles from aviation, nuclear, healthcare:

- → Preoccupation with Failure
- → Reluctance to Simplify
- → Sensitivity to Operations
- → Commitment to Resilience
- → Deference to Expertise

See HRO Pattern Recognition for deep dive

SRE EVOLUTION

ERA	PERIOD	FOCUS
Chaos Years	1990-2005	Cowboy ops
DevOps	2005-2015	Automation
SRE	2014-2018	Reliability
Platform	2018-Now	Developer UX

KEY TAKEAWAYS

- → **Automate everything:** Eliminate manual toil
- → **Embrace failure:** Practice makes resilient
- → **Measure what matters:** SLOs drive decisions
- → **Culture first:** Blameless enables learning

TOOL EVOLUTION

- → **2000s:** Nagios, Puppet, Chef
- → **2010s:** Docker, K8s, Prometheus, Terraform
- → **2020s:** OpenTelemetry, GitOps, AI/ML Ops

Learn from the Best

Adopt practices, not just tools.

Implementation Roadmap

5-Phase Journey to AI-Native Operations

Strategic Roadmap | Technical Operations Excellence

5	12	90%	99.9%
PHASES	MONTHS	AUTO-RESOLUTION GOAL	AVAILABILITY TARGET

PHASE 1: FOUNDATION (MONTH 1-2)

Objective: Establish core operational capabilities

- → Deploy Grafana Alerting
- → Implement PagerDuty integration
- → Create incident response playbooks
- → Build runbook automation framework
- → Establish on-call rotation

Metrics: Alerting live, <15m MTTA, top 10 runbooks

PHASE 2: RELIABILITY (MONTH 3-4)

Objective: Achieve target SLOs and error budget governance

- → Error budget dashboard & automation
- → Post-mortem workflow automation
- → Feature flags infrastructure
- → First chaos engineering GameDay
- → Canary deployment pipeline

Metrics: 99.0% availability, 95% success rate

PHASE 3: AUTOMATION (MONTH 5-6)

Objective: Reduce toil below 50%, increase auto-resolution

- → Automated incident triage
- → Self-healing runbooks (top 5 alerts)
- → Capacity auto-scaling
- → Compliance automation

Metrics: 70% auto-resolution, toil <50%

PHASE 4: INTELLIGENCE (MONTH 7-8)

Objective: Predictive operations and AIOps

- → Anomaly detection ML models
- → Predictive capacity alerting
- → Automated root cause analysis
- → AI-powered post-mortem generation

Metrics: 80% 48hr prediction accuracy, 50% MTTR reduction

PHASE 5: EXCELLENCE (MONTH 9-12)

Objective: World-class operations, continuous improvement

- → Cloud migration enablement (AWS/GCP)
- → Multi-region resilience
- → Full OpenTelemetry instrumentation
- → Autonomous operations (zero-touch)

Metrics: 99.9% availability, <30s MTTA, 90% auto-resolution

SUCCESS METRICS JOURNEY

METRIC	START	END
Availability	95%	99.9%
MTTA	Hours	<30s
Auto-Resolution	0%	90%
Toil	>80%	<30%

BOT ARMY OWNERS

BOT	PRIMARY RESPONSIBILITY
Ops Bot	Incident response, runbooks
SRE Bot	Resilience, deployments
Observability Bot	Metrics, alerting, dashboards
Security Bot	Compliance, secrets

KEY MILESTONES

- → **Month 2:** First PagerDuty alert fired
- → **Month 4:** First GameDay completed
- → **Month 6:** Self-healing runbooks active
- → **Month 8:** AI-powered RCA deployed
- → **Month 12:** Autonomous operations

From Reactive to Autonomous

12 months to world-class AI-native operations.

Automation Paradoxes

Bainbridge's Ironies & Human-Agent Balance

Agentic Operations | Technical Operations Excellence

1983	40%	5-10%	M-Shaped
BAINBRIDGE PAPER	AGENTIC AI MAY FAIL	EDGE CASES	NEW SUPERVISOR ROLE

IRONIES OF AUTOMATION

> *The more advanced automation becomes, the more crucial human intervention becomes when it fails.*
>
> - Lisanne Bainbridge (1983)

Automation doesn't eliminate human involvement - it changes it.

HUMAN-AGENT BALANCE

Low Risk: Full Autonomy
Routine tasks, easily reversible, well-understood

Medium Risk: Supervised
Complex tasks, human approval required

High Risk: Human-in-Loop
Critical systems, irreversible actions

THE FOUR IRONIES

IRONY	DESCRIPTION
Skill Degradation	Operators lose skills they don't practice
Harder Failures	Automation handles easy cases, leaves hard ones
Lost Situational Awareness	Out-of-the-loop syndrome
Increased Criticality	When intervention needed, stakes are highest

EDGE CASE PROBLEM

SCENARIO	CHALLENGE
Novelty	Bot hasn't seen this before
Ambiguity	Multiple valid actions
Conflict	Competing objectives
Context	Missing business knowledge

5-10% of cases need human judgment - but they're the hardest

M-SHAPED SUPERVISORS

The new human role: oversee multiple specialized bots

→ Broad awareness across domains
→ Deep expertise for intervention
→ Strategic decision-making
→ Exception handling

SRE AUTOMATION SPECTRUM

LEVEL	HUMAN ROLE	BOT ROLE
Manual	All decisions	None
Assisted	Decides	Suggests
Supervised	Approves	Executes
Monitored	Watches	Autonomous
Autonomous	Reviews post-hoc	Full control

WARNING SIGNS

→ **Complacency:** "The bot handles it"
→ **Skill atrophy:** "I forgot how to do that"
→ **Blind trust:** "The bot must be right"

MAINTAINING HUMAN EXPERTISE

→ **Wheel of Misfortune:** Practice manual interventions
→ **GameDays:** Disable automation, respond manually
→ **Shadow mode:** Watch bot decisions before approval
→ **Runbook reviews:** Understand what bots do

Augment, Don't Replace

The best automation makes humans more capable, not irrelevant.

Sources: Bainbridge (1983), Gartner, Human Factors Research

SRE Evolution Timeline

From Sysadmin to Platform Engineering

Historical Perspective | Technical Operations Excellence

60+	2003	182x	2018
YEARS OF EVOLUTION	SRE BORN AT GOOGLE	ELITE DEPLOY FREQ	PLATFORM ENG ERA

THE ERAS OF OPERATIONS

ERA	PERIOD	CHARACTERISTICS
Pre-History	1960-1990	Mainframes, UNIX
Chaos Years	1990-2005	Cowboy ops, silos
DevOps	2005-2015	Automation, CI/CD
SRE	2014-2018	Error budgets, SLOs
Platform	2018-Now	Golden paths, DX

DORA METRICS EVOLUTION

LEVEL	DEPLOY FREQ	LEAD TIME
Low	Monthly-6mo	6+ months
Medium	Weekly-Monthly	1-6 months
High	Daily-Weekly	1 week-1mo
Elite	Multiple/day	<1 day

Elite: 182x more deploys, 2,293x faster MTTR

ROLE EVOLUTION

`Sysadmin → DevOps → SRE → Platform Engineer`

ROLE	FOCUS
Sysadmin	Manual operations
DevOps Engineer	Automation, culture
SRE	Reliability engineering
Platform Engineer	Developer experience

PLATFORM ENGINEERING

Golden Paths
Paved roads for common workflows

Internal Developer Platforms
Self-service infrastructure

Developer Experience
Reduce cognitive load

TOOL EVOLUTION

DECADE	TOOLS
2000s	Nagios, Puppet, Chef
2010s	Docker, K8s, Prometheus, Terraform
2020s	OpenTelemetry, GitOps, AI/ML Ops

SRE CAREER PATHS

IC TRACK	MANAGEMENT TRACK
SRE	-
Senior SRE	SRE Manager
Staff SRE	SRE Director
Principal SRE	VP Engineering

THE DEFINITION

> *"class SRE implements interface DevOps"*
>
> **- Ben Treynor, Google**

KEY MILESTONES

- → **2003:** Ben Treynor coins "SRE" at Google
- → **2009:** Flickr "10+ deploys/day" talk
- → **2013:** Docker released
- → **2016:** Google SRE Book published
- → **2018:** DORA "Accelerate" published

What's Next?

AI-native operations and autonomous systems.