

Managing SLAs Across Distributed Applications

A lifecycle-based methodology for setting credible service commitments, calculating end-to-end availability, and managing production dependencies.

19 SEPT 2026 · 10 MIN READ



Darren Sim

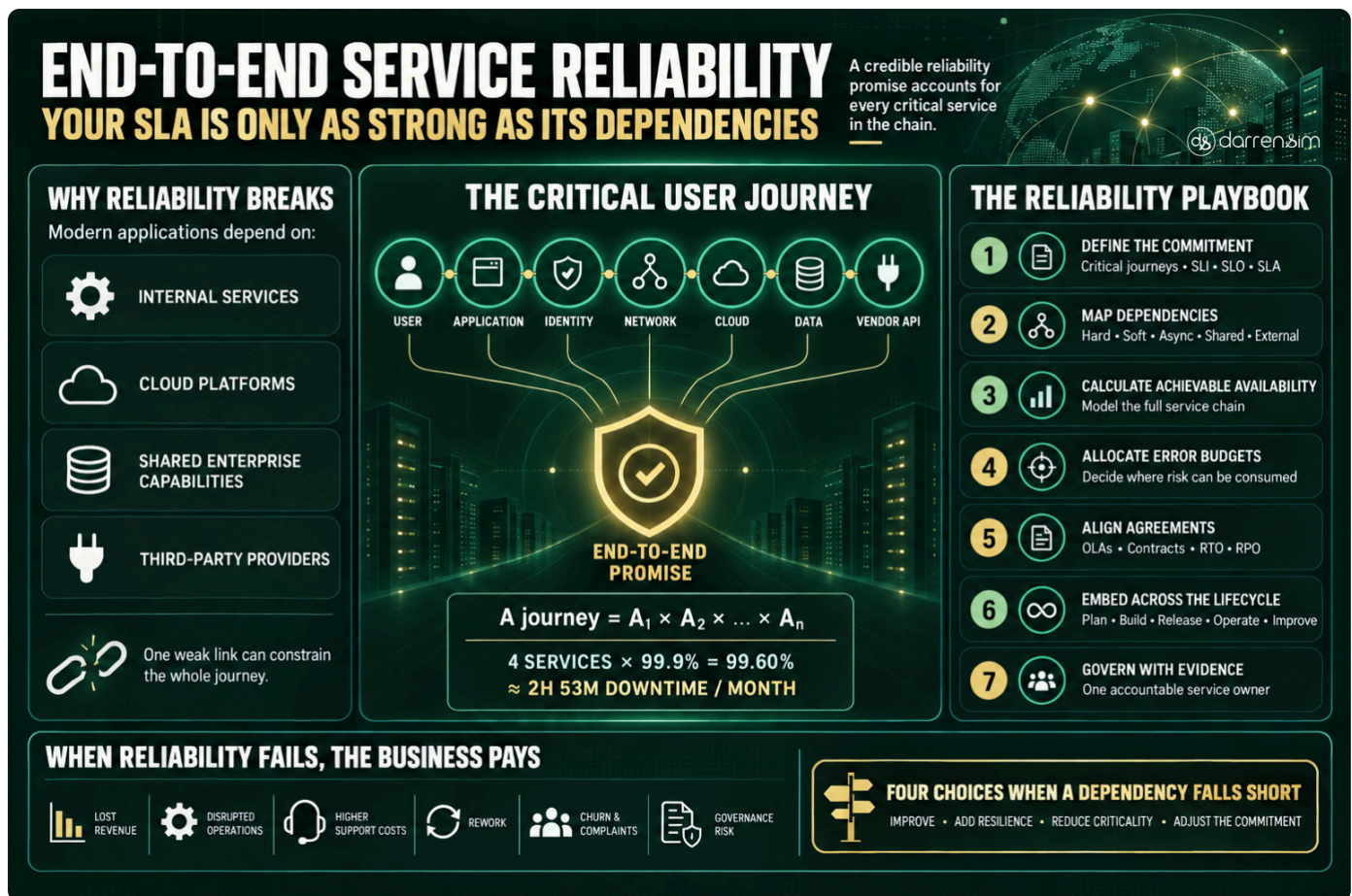
Senior technology & product leader

All content published here is shared in a personal capacity. Nothing on this site should be interpreted as an official statement, endorsement, or communication from my employer.

Most modern applications are distributed systems, relying on a complex network of internal services, cloud platforms, third-party providers, and shared enterprise capabilities. This dependency chain makes service reliability far more difficult to define, measure, and guarantee.

An application's reliability promise is credible only when it accounts for every critical system on which the service depends.

An application may carry a demanding Service Level Agreement (SLA), while one or more of its critical dependencies operate under weaker—or simply different—service commitments. As a result, the application's reliability cannot be assessed in isolation. Its achievable performance is constrained by the combined reliability of the entire service chain.



When Reliability Fails, the Business Pays

This is not just an engineering concern—it is a leadership and governance issue that directly impacts business continuity, customer trust, and the bottom line.

Service Level Indicators (SLIs), Service Level Objectives (SLOs), and SLAs must be designed with end-to-end dependencies in mind. They help leaders answer three questions: what matters most (critical journeys and operating hours), how we will measure it (simple, agreed indicators), and what we will commit to (the service promise).

When these commitments are unclear, reliability is managed reactively. The organization experiences surprise outages, inconsistent expectations (“always on” vs. practical limits), and repeated fire drills.

The **bottom-line impact** shows up quickly: **lost sales** or **delayed revenue**, **disrupted operations** and **staff productivity**, higher **support** and **vendor escalation costs**, **rework**, and **reputational damage** that increases **churn** and **complaint volume**. It also creates **governance risk**—difficulty demonstrating performance against **contractual** or **regulatory** expectations.

Clear SLIs/SLOs/SLAs improve decision-making and accountability. They make tradeoffs explicit (speed vs. stability), highlight where dependencies weaken the end-to-end promise, and provide a fact base to prioritize investments, align internal teams, and hold suppliers to measurable outcomes. In short, they turn reliability from a recurring surprise into a capability leaders can plan for, govern, and improve.

A Practical Playbook for End-to-End Service Reliability

A practical approach is to manage reliability across the full **application lifecycle**—from planning and service design through build, release, production operations, continuous improvement, and retirement. The objective is not merely to calculate an SLA, but to ensure the commitment is supported by the architecture, dependencies, operating model, and investment decisions.

1. Define the service commitment during planning

Begin with the business service rather than the underlying technology. Identify:

- The application's critical users and business processes
- Its **critical user journeys**, such as login, search, transaction submission, or payment
- Business-critical operating periods
- Regulatory, contractual, and customer commitments
- The business impact of downtime, degraded performance, or data loss

For each critical journey, define:

- **Service Level Indicator (SLI)**: What will be measured, such as successful transaction rate, latency, availability, or data freshness.

- **Service Level Objective (SLO):** The internal reliability target used to operate and improve the service.
- **Service Level Agreement (SLA):** The formal commitment made to customers or business stakeholders.

The internal SLO should normally be more demanding than the external SLA. This creates an **operational safety margin** for detection delays, reporting differences, unexpected failures, and recovery activities.

The following table provides an illustrative example:

MEASURE	External SLA
ILLUSTRATIVE TARGET	99.90% monthly availability

MEASURE	Internal SLO
ILLUSTRATIVE TARGET	99.95% monthly availability

MEASURE	Engineering design target
ILLUSTRATIVE TARGET	99.97% availability



What the availability percentages mean

Availability is the proportion of an agreed measurement period during which a service is usable. For example, **99.9% monthly availability** means the service may be unavailable for up to **0.1% of that month** while still meeting its target. It does not mean that outages cannot occur.

AVAILABILITY TARGET	99.90%
MAXIMUM DOWNTIME IN A 30-DAY MONTH	43 minutes 12 seconds
APPROXIMATE MAXIMUM DOWNTIME PER YEAR	8 hours 46 minutes

AVAILABILITY TARGET	99.95%
MAXIMUM DOWNTIME IN A 30-DAY MONTH	21 minutes 36 seconds
APPROXIMATE MAXIMUM DOWNTIME PER YEAR	4 hours 23 minutes

AVAILABILITY TARGET	99.97%
MAXIMUM DOWNTIME IN A 30-DAY MONTH	About 13 minutes
APPROXIMATE MAXIMUM DOWNTIME PER YEAR	About 2 hours 38 minutes

These figures assume **24x7 service coverage** and continuous measurement. The actual allowance depends on the SLA definition, including the measurement window, business hours, planned-maintenance exclusions, what qualifies as downtime, and whether partial degradation counts as unavailability. Each additional “nine” materially reduces the permitted downtime and usually requires stronger architecture, operations, monitoring, and recovery capabilities.

2. Model the service dependency chain

During architecture and service design, create an end-to-end **service dependency map** for every critical user journey. Classify dependencies as:

DEPENDENCY TYPE	Hard dependency
WHAT IT MEANS	The journey fails when the dependency is unavailable.
DEPENDENCY TYPE	Soft dependency
WHAT IT MEANS	The application continues with reduced functionality.
DEPENDENCY TYPE	Asynchronous dependency
WHAT IT MEANS	Failure delays processing but does not immediately interrupt the user.
DEPENDENCY TYPE	Redundant dependency
WHAT IT MEANS	Another service can provide the same capability.
DEPENDENCY TYPE	Shared platform dependency
WHAT IT MEANS	Identity, network, cloud, database, integration, observability, or security services.
DEPENDENCY TYPE	External dependency
WHAT IT MEANS	A vendor, SaaS provider, partner API, or managed service.

Maintain this map as part of the application's architecture baseline or service configuration—not as a one-time design document. It should be reviewed whenever the architecture, vendor landscape, or critical user journeys change.

3. Calculate achievable end-to-end availability

For a critical user journey that requires several independent components to be available at the same time, approximate end-to-end availability as:

$$A_{\text{journey}} = A_1 \times A_2 \times \dots \times A_n$$



If a journey depends on four services, each delivering 99.9% availability:

$$0.999^4 = 0.996006$$

The resulting theoretical availability is approximately **99.60%**, not 99.9%. In a 30-day month, 99.9% permits about 43 minutes of unavailability, while 99.6% permits about 2 hours and 53 minutes.

For genuinely redundant components where either component can support the service:

$$A_{\text{redundant}} = 1 - (1 - A_1)(1 - A_2)$$

These formulas are a starting point, not a substitute for production evidence. They assume independent failures. In practice, services may share a cloud region, identity platform, network, deployment pipeline, data store, or operational team. These create **correlated failure risks** that simple multiplication will not capture. Architecture reviews, resilience tests, failure-mode analysis, and historical production data must supplement the theoretical calculation.

4. Allocate the reliability and error budgets

Once the application SLO is established, allocate a **reliability budget** across the components and teams supporting each critical journey. The corresponding **error budget** is:

$$\text{Error budget} = 1 - SLO$$

For a 99.9% monthly availability SLO, the error budget is 0.1%, or approximately 43 minutes in a 30-day month. The application team should decide how much of this budget can be consumed by:

- Application defects
- Planned maintenance included in the service-level definition
- Platform or infrastructure outages
- Vendor failures
- Deployment failures
- Capacity constraints
- Operational or configuration errors

Where a dependency cannot support the required service level, the application team has four choices:

1. **Improve the dependency** through a stronger operational agreement, supplier commitment, or engineering investment.
2. **Introduce resilience** through redundancy, caching, queues, retries, circuit breakers, graceful degradation, or offline processing.
3. **Reduce criticality** by redesigning the journey so that the dependency is no longer synchronous or mandatory.
4. **Adjust the application commitment** so that the SLA reflects what the production architecture can realistically deliver.

5. Align contracts and operational agreements

The application SLA must be supported throughout the delivery and support model. Relevant controls may include:

- **Operational Level Agreements (OLAs)** among application, platform, infrastructure, security, and support teams
- **Underpinning Contracts (UCs)** with external vendors and providers
- Incident response and escalation commitments
- Recovery Time Objectives (RTOs)
- Recovery Point Objectives (RPOs)
- Support coverage and on-call requirements
- Maintenance-window arrangements
- Capacity and performance commitments

A dependency's commitment should not merely equal the application SLA. It generally needs to be stronger because the application must absorb failures from multiple sources while still meeting its end-to-end commitment.

6. Embed reliability across the application lifecycle

Plan and design

- Define critical user journeys and business-impact tolerances.
- Establish SLIs, SLOs, and the external SLA.
- Identify single points of failure and correlated dependencies.
- Document resilience, continuity, and disaster-recovery patterns.
- Confirm that the proposed architecture can support the target service level.

Build and test

- Include reliability requirements in acceptance criteria and the definition of done.
- Implement health checks, timeouts, retries, circuit breakers, and graceful degradation.
- Test dependency failure and recovery behavior.
- Conduct performance, load, failover, and disaster-recovery testing.

- Ensure observability is production-ready before release.

Release and deploy

- Include SLO and dependency risk in release-readiness reviews.
- Use progressive delivery, canary releases, or blue-green deployment where appropriate.
- Define rollback and roll-forward procedures.
- Track deployment failures against the error budget.
- Restrict high-risk releases when the error budget has been substantially consumed.

Operate in production

- Measure SLIs from the customer and business-service perspective.
- Monitor critical journeys rather than infrastructure components alone.
- Maintain production dashboards for SLO attainment and error-budget consumption.
- Establish dependency-specific alerts, runbooks, and escalation paths.
- Link incidents to affected journeys, dependencies, and service-level impact.
- Review vendor and internal platform performance against their commitments.

Improve continuously

- Use post-incident reviews to identify dependency and lifecycle-control failures.
- Prioritize reliability work based on business impact and error-budget consumption.
- Update dependency maps and architecture records after material changes.
- Recalibrate SLOs using actual production performance and changing business requirements.
- Feed recurring failure patterns into the application roadmap and technical-debt backlog.

Retire or transition

- Maintain service commitments throughout migration or decommissioning.

- Identify consumers before removing interfaces or dependencies.
- Define data-retention, archival, and continuity requirements.
- Confirm that replacement services meet or exceed the required SLOs.

7. Establish production governance and accountability

Each production application should have a clearly accountable **service owner** responsible for the end-to-end service level, even when multiple teams operate its dependencies.

A regular service review should cover:

- SLO and SLA attainment
- Error-budget consumption
- Availability and performance trends
- Major incidents and recurring problems
- Dependency performance and concentration risks
- Capacity and scalability risks
- Recovery-test results
- Upcoming releases and architectural changes
- Reliability investments and accepted risks

This creates a direct connection between production evidence and **application portfolio management**. Applications that consistently miss their objectives may require modernization, additional resilience investment, renegotiated dependency agreements, or revised business commitments.

Questions senior leaders should ask

1. Which critical user journeys underpin our SLA?
2. What are their hard dependencies, and who owns them?

3. Can those dependencies collectively support the promised service level?
4. Where do shared platforms create correlated failure risk?
5. How much error budget remains, and what is consuming it?
6. Are supplier contracts and internal OLAs aligned with our end-to-end commitment?
7. Which lifecycle investments would most improve customer and business outcomes?

Conclusion

The application team may not operate every dependency, but the application owner remains accountable for understanding how those dependencies affect the business service.

A credible SLA is therefore more than a contractual percentage. It is the outcome of disciplined **service design, dependency management, release governance, production operations, and continuous lifecycle improvement**. Senior leaders should treat reliability as a managed business capability—supported by architecture, operating agreements, production evidence, and explicit investment decisions.



— THE AUTHOR

Darren Sim

Darren is a senior technology and product leader based in Singapore. He writes about the decisions, systems, and people behind meaningful transformation across Asia-Pacific.

[More about Darren ↗](#)

[Connect on LinkedIn ↗](#)