2026 Mar 12nd

Ren Goto

SLI/SLO 入門

サービスの信頼性を「数値」で管理し、組織の共通言語にする

本日のゴール

1. SLI / SLO / SLA の違いと関係がわかる
2. Error Budget の考え方がわかる
3. なぜ SLO が 開発チームに取って重要 なのかがわかる
4. チームで SLI/SLO を 導入する最初の一歩 のイメージができる

INDEX

1. なぜ「信頼性」を数値にするのか？

信頼性の課題

- 「最近サービスが不安定じゃない？」→ “不安定”ってどれぐらい？
- 「可用性をあげよう」→ 今いくつで、どこまで上げる？
- 「この障害、大丈夫？」→ 何を基準に “大丈夫” と判断する？

数値が無い状況では　主観・定性的な判断　に委ねられがち

数値がないと何が起きるか

開発チーム

新機能を早く
リリースしたい

双方の ”正しさ” が衝突

運用チーム

安定稼働を
最優先にしたい

- リリースのたびに **綱引き** が起きる
- 障害のたびに **犯人探し** が始まる
- 「完璧」を目指して **リリース速度** が落ちる
- 誰も「どこまでやれば十分か」を **説明できない**

Google SRE ― “The Art of SLOs” ASSERTION #1

「あらゆるシステムにおいて
最も重要な機能 は **信頼性** である」

― *The most important feature of any system is its reliability* ―

どんなに優れた機能でもサービスが動かなければ使えない

出典：https://cre.page.link/art-of-slos-howto-pdf-a4

Google SRE ― “The Art of SLOs” ASSERTION #2

「100% はほぼすべてのサービスにとって
間違った信頼性の目標 である」

― *100% is the wrong reliability target for basically everything* ―

ほぼすべてのサービスにおいて 100% は
不可能であり目指すべきでもない

出典：https://cre.page.link/art-of-slos-howto-pdf-a4

100% を目指すべきでない理由

可用性（目標値）	年間ダウンタイム	月間ダウンタイム	実現難易度
99%（Two Nines）	3日15時間	7時間18分	容易
99.9%（Three Nines）	8時間46分	43分50秒	標準的
99.99%（Four Nines）	**52分34秒**	**4分23秒**	**困難**
99.999%（Five Nines）	5分15秒	26秒	非常に困難

- 99.9% から 99.99% に上げる **コストは桁違いに高い**
- ユーザ自身のネットワークや端末も 100% ではない
- **「ちょうどよい信頼性」**を見つけるのが SLO（後述）の目的

2. SLI / SLO / SLA ー 3つの用語を整理する

全体像

用語	定義	性質	対象者
SLI	計測値	事実・データ	社内（開発・運用）
SLO	内部目標	合意事項	社内（＋PM等）
SLA	外部契約	ビジネス契約	顧客向け

SLI（Service Level Indicator）

ユーザ体験を反映する 定量的な計測値

基本の形：

$$\mathrm{SLI} = \frac{\text{良いイベントの数}}{\text{全イベントの数}} \times 100\%$$

指標の例	良いイベント	全イベント
可用性	HTTP ステータス < 500 のリクエスト数	全リクエスト数
レイテンシ	200ms 以内に応答したリクエスト数	全リクエスト数
データ鮮度	1 分以内に処理されたジョブ数	全ジョブ数

💡 ポイント：**常に 0〜100% の割合** で表現する

SLO（Service Level Objective）

SLI に対する 内部的な目標値 + 期間

基本の形：

$$\text{SLO} = \text{SLI の目標値} + \text{計測ウィンドウ（期間）}$$

> 例：「過去28日間のレスポンスの **99.9%** が **HTTP ステータス 2xx/3xx/4xx** を返す」

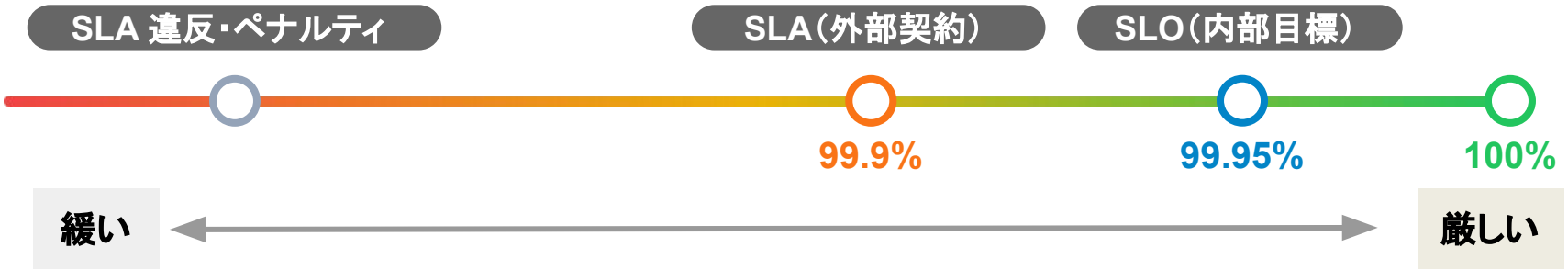
重要なポイント：

- SLO は **ユーザの期待** に基づいて設定する
- 技術的な限界ではなく **ビジネス要件から逆算** する
- 最初から完璧である必要はない ― イテレーティブに改善する

SLA（Service Level Agreement）

SLO を下回った場合の ペナルティを含む顧客契約

SLO と SLA の関係：



- SLA は SLO より **緩く設定** する（バッファを持たせる）
- SLO を守れていれば SLA 違反は起きない仕組みにする
- （内部向けサービスなどで）SLA がなくても **SLO は必ず設定すべき**

3. SLI ー 何を測るか？

SLI の 4 つのカテゴリ

カテゴリ	意味（どんな体験を測るか）	対象システムの例
可用性（Availability）	リクエストが成功したか？	API、Web アプリケーション
レイテンシ（Latency）	どれだけ速く応答したか？	API、Web アプリケーション
品質（Quality）	（正しい結果で）応答したか？	動画配信、検索システム
データ鮮度（Freshness）	取得したデータは最新か？	バッチ処理、データパイプライン

すべてのカテゴリが全サービスに必要なわけではない。
→ **ユーザの気にすること（関心事）** に絞るのが重要

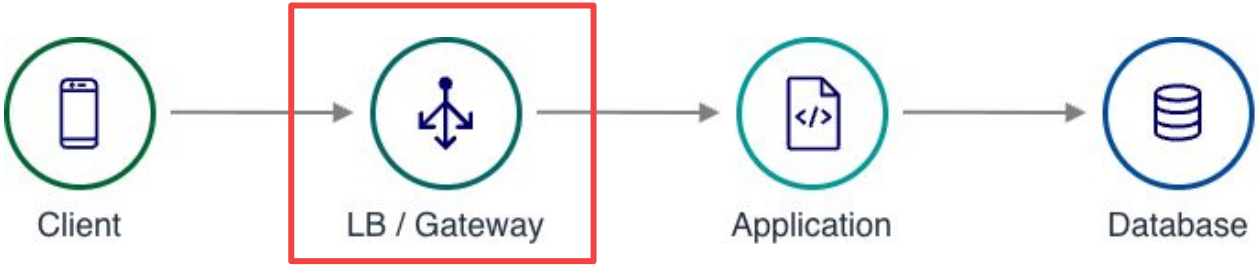
良い SLI / 悪い SLI

良い SLI の条件：

1. ユーザ体験と **直接相関** する
2. **計測可能** である
3. 0〜100% の **比率** で表せる

やりがちな指標	なぜダメか？	代わりに何を見るか？
CPU / メモリ使用率	高くてもユーザに影響がない場合がある。 （ユーザ体験と直接関係しない）	**レイテンシ（p99）**
障害件数 / アラート数	1件の巨大な障害と、10件の軽微な障害の 影響度の違いを表現できない。	**エラー率（%）**
平均レイテンシ	少数の極端に遅いリクエスト（外れ値）が 平均に埋もれる。	**パーセンタイル （p95 / p99）**

どこで計測するか？



計測ポイント	メリット	デメリット
LB / API Gateway ★推奨	インフラ障害を含めたユーザ体験に近い。既存ログで計りやすい。	クライアント～LB間のNW障害は捕捉できない。
Application（内部）	ビジネスロジックに特化した詳細な計測が可能。	アプリケーション到達前の障害（LBやプロキシのエラー）が漏れる。
Client（モバイル / Web）	真のユーザ体験（NW遅延含む）を測れる。	実装難易度が高くノイズ（ユーザ環境由来）が多い。

> 原則：ユーザに近い場所で測る

具体例: API サービスの SLI 定義

```
### Avaivality SLI ###
avaivality:
  good_eventL: "HTTP status < 500"
  total_events: "全 HTTP リクエスト"
  # = 非 5xx リクエスト数 / 全リクエスト数

### Latency SLI ###
latency:
  good_eventL: "レイテンシ < 300ms のリクエスト"
  total_events: "全 HTTP リクエスト"
  # = 300ms 以内のリクエスト数 / 全リクエスト数
```

> この定義を **SLO ドキュメント** として文書化しておく（後述）

4. SLO ー どこまで守るか？

SLO を決める 4 ステップ

1

現状を知る

まずは過去数週間のデータから現在の SLI の実績値を算出する。

2

ユーザの期待を理解する

ユーザがどこまでなら許容できるか、競合サービスの水準はどの程度かを検討する。

3

SLO を仮設定する

現状の実績より「少し厳しめ」に設定する。最初から完璧である必要はなく、仮置きでよい。

4

運用しながら調整する

厳しすぎてバジェットが枯渇し続けるなら緩める。緩すぎて誰も気づかないなら締める。

SLO ドキュメントの例

サービス名	Payment API
SLI	HTTP 5xx 系以外のレスポンスの割合
SLO	**28日間で 99.95%**
計測方法	ロードバランサーのアクセスログ
対象パス	/api/v1/payments/*
除外条件	計画メンテナンス時のダウン、クライアント起因の 4xx エラー
ウィンドウ	28日ローリング
レビュー頻度	四半期ごと

> SLO は **文章化してチームで合意する** ことが重要。
> 口頭の約束ではなくレビュー可能なドキュメントにする。

計測ウィンドウの種類

ローリングウィンドウ

★推奨

常に「現在から遡って直近の N 日間（例：28日）」を評価期間とする。

メリット

常に最新の状態を反映できる

↓

リセット日がないため、月末月初で行動が変わらない。

※Google SRE では 28日ローリング を推奨

カレンダーウィンドウ

「月初から月末まで」のように、固定の暦期間で評価する。

注意点

月末に Budget がリセットされる

↓

「月末はリリース凍結、月初にまとめてリリース（そして障害を起こす）」という歪んだインセンティブを生みやすい。

5. Error Budget ―「失敗してよい量」を管理する

Error Budget とは？

信頼性を 犠牲にできる余裕を数値化 したもの

計算例：

- SLO **99.9%**（28日間）→ 許容エラー = **0.1%**
- 28日間の総リクエスト数：1,000,000 件

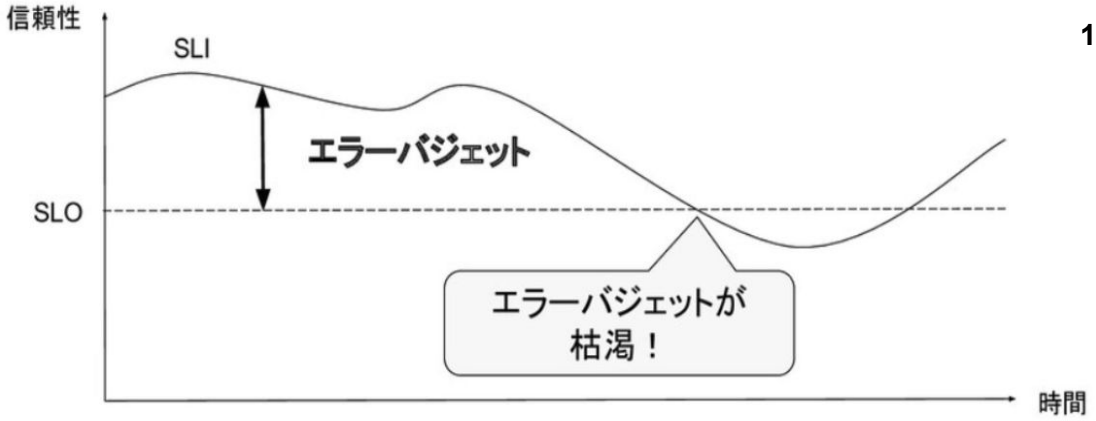
$$\text{Error Budget} = (1 - 0.999) \times 1{,}000{,}000 = \boxed{1{,}000 \text{ 件}}$$

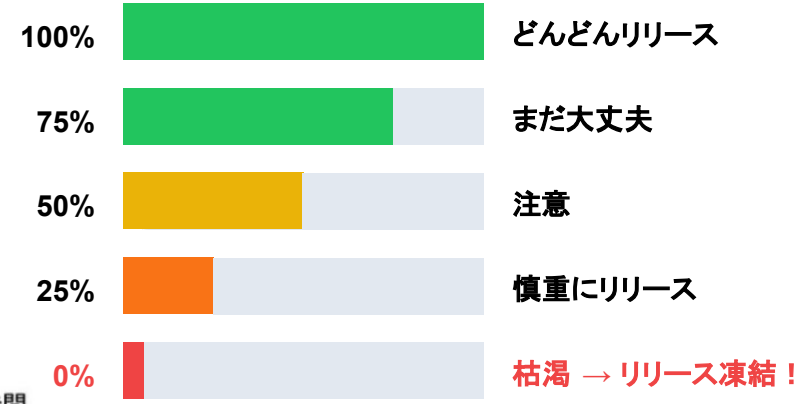
> この 1,000 リクエスト分の「失敗できる権利」を新機能リリースやインフラ（運用基盤）の変更に **戦略的に利用** する。

Error Budget の使い方

消費速度を確認して将来のリスクをコントロールする

- Error Budget が残っている間は機能開発・リリースを優先
- 使い切ったら新規リリースを止めて信頼性改善に集中する





出典：https://speakerdeck.com/shonansurvivors/error-budget-practice?slide=7

Burn Rate アラートの考え方

Error Budget の 消費速度 に応じてアラートを発報する

Fast Burn：急速な消費

- 例：このままだと数時間で Budget が枯渇する猛烈なペース
- 対応：緊急アラート、深夜でも即時対応が必要（On-Call など）

Slow Burn：じわじわ消費

- 例：このままだと数日で Budget が枯渇するペース
- 対応：チケットの起票、翌営業日に対応

Burn Rate アラートの導入

従来のアラート

CPU > 80%
→ ユーザ影響が無くても鳴る（ノイズが多い）

エラー > 10 件/分
→ 全体トラフィックが多い時は誤差かもしれない

アラートが形骸化しやすい

Burn Rate アラート

Fast Burn の発生
→ 緊急アラート、深夜でも即時対応が必要（On-Call など）

Slow Burn の発生
→ チケットの起票、翌営業日に対応

本当に必要な時のみ通知

Budget 消費速度が異常
→ **実際に ユーザ体験が損なわれる可能性がある場合のみ に通知**

Error Budget ポリシ

Budget が枯渇した時のルールを事前に合意しておく

ポリシの例:

- **新機能のリリース** を凍結する(バグ修正は許可)
- チームの工数の少なくとも **50% を信頼性改善** (負債解消)に充てる
- ポストモーテム(障害の振り返り)を徹底して再発防止策を立てる
- ローリングウィンドウで **Budget が回復するまで凍結を継続** する

> 開発者と PM(経営層)の双方が合意する必要がある
> → **これがなければ単に 組織に軋轢が生じる だけになる**

Google SRE ― “The Art of SLOs” ASSERTION #3

「Error Budget はそれを使い果たした場合に　経営陣の後ろ盾 があって初めて組織の緊張を解くことができる。」

― *SLO targets need to be set with your users in mind and error budgets can only resolve organizational tensions if the consequences for exceeding them have executive backing.* ―

「超過時にどうするか」を経営層がコミットしてこそ
組織的な意思決定ツールとして機能する

出典：https://cre.page.link/art-of-slos-howto-pdf-a4

6. SLO が組織にもたらすもの

SLO は「共通言語」になる

『定性的な主観』に基づいた “議論”

・開発:「早くリリースしたい。施策が最優先だ！」
・運用:「障害が起きたらどうするの？安定性も意識して！」
・PM:「で、どっちが正しいの？」

【結果】声の大きい人が勝つ（属人性を孕む）

『定量的な指標』に基づいた “判断”

・開発:「Error Budget が残り 70% だからリリースします。」
・運用:「同意。ただし残り 30% を切ったらリリースは控えよう。」
・PM:「了解。次のリリース計画はこれで進めよう。」

【結果】全員が同じ数字を見て判断する

SLO がもたらす 3 つの効果

1. **客観的な判断基盤**
   - 「リリースしてよいか？」が 数値で判断 できる
   - 障害時の対応判断も Budget 消費量 で決まる
2. **開発速度と信頼性のバランス**
   - Budget に余裕 → 速度重視（どんどんリリース）
   - Budget が逼迫 → 信頼性優先（改善に集中）
3. **組織横断のコミュニケーション**
   - 開発・運用・PM・経営が 同じ指標 で議論できる
   - 「何となく不安」ではなく「SLI が X% で SLO 未達」と言える

7. 導入のロードマップ

4 つのフェーズで段階的に導入する（ 1/2）

Phase 1：計測を始める（ 1〜2 週間）

- 1 つのサービス で SLI を定義する ― 最初から全部やらない
- 可用性 SLO だけでも OK ― 完璧を目指さない
- 既存のモニタリング基盤（Prometheus, Datadog など）を活用する

Phase 2：SLO を仮設定する（ 2〜4 週間）

- 過去のデータから SLI の実績値を確認する
- 実績値をもとに SLO を仮決め → SLO ドキュメント を作成する
- ダッシュボードで Error Budget を可視化する

4 つのフェーズで段階的に導入する（ 1/2）

Phase 3：Error Budget の運用を開始する（ 1〜3 ヶ月）

- Error Budget ポリシ を策定し合意する
- アラートを SLO ベース（Burn Rate）に移行する
- 月次で SLO の達成状況をレビューする

Phase 4：定着・拡大・組織浸透（継続）

- 四半期レビューで SLO を見直し・調整
- 他サービスへ横展開
- SLO レビューを 定例ミーティング（モニタリング定例）に組み込む

明日からできる SRE

やること	所要時間	誰が
自チームのサービスで **最も重要な API** を 1 つ選ぶ	10 min	チーム全体
そのエラー率を **過去 28 日分** 集計してみる	30 min	SRE
「SLO をいくつに設定するか？」を **議論** する	30 min	チーム全体
SLO ドキュメントの **ドラフト** を書く	30 min	SRE

> 最初の一歩は「1 つのサービスの 1 つの SLI」から。
> → **小さく開始して価値を実感してから拡大する**

まとめ

本日のまとめ

概念	一言で言うと
SLI（Service Level Indicator）	ユーザ体験を **数値化** する指標
SLO（Service Level Objective）	その指標の **内部目標**（期限付き）
SLA（Service Level Agreement）	SLO に基づく **外部契約**
Error Budget	信頼性を犠牲にできる **余裕度（予算）**

覚えて帰ってほしい 4 つのこと:

1. **100% の信頼性は目指さない** — ちょうどよい信頼性を見つける
2. **SLI はユーザ視点で選ぶ** — CPU 使用率ではなくレスポンスタイム
3. **Error Budget で開発速度と信頼性のバランスを取る**
4. **小さくはじめる** — 1 サービス 1 SLI からスタート

Q&A

ご質問があればどうぞ

参考資料

- Google SRE WorkBook
  https://sre.google/sre-book/table-of-contents/
- Google SRE ― “The Art of SLOs”
  https://cre.page.link/art-of-slos-howto-pdf-a4
- SLO は何を実現するのか（ @ymotongpoo）
  https://speakerdeck.com/ymotongpoo/what-does-slo-achieve