The Art of SLOs

ファシリテーター用ハンドブック

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

SLO の技術 (Art) とは

Art (名詞): 特定のことを行う能力、
通常は練習を行うことで獲得する

SLOの技術 (Art) は、開発・運用・製品およびビジネスの領域全体の多様なオーディエンスにサービス レベル目標 (SLO) の開発に不可欠な要素を教えるために設計されたワークショップです。

理論について説明するワークショップでは、提供するサービスに求められる信頼性を示す目標を設定することで、開発チームと運用チームの間でしばしば起こる組織間の緊張の解決方法を学ぶことができます。SLO とエラー バジェットを活用して、データドリブンで客観的なユーザーを中心とした方法でサービスの信頼性の測定と管理が可能であることを紹介します。ワークショップの技術的な部分では、参加者に対して良い SLI に必要な特性の簡単な紹介をします。セッションの最後では、架空のモバイル ゲームのサーバー側インフラストラクチャとユーザーとの間の簡単なインタラクションに SLI を開発する 4 ステップのプロセスを当てはめてワークショップは終了します。

ワークショップの実践的な説明のパートでは、参加者にゲームのサービスを提供するインフラストラクチャとユーザーとの間のより複雑なインタラクションに対して、学習したことを適用してもらいます。それぞれのインタラクションではひねった課題があり、サービスがどれだけうまくユーザーの期待に応えているかについて、適切な代替手段の見つけ方を参加者に真剣に考えてもらいます。 最後に解答例が示され、参加者が出した答えとその理由付けを解答例と比較していきます。

参加対象者

このワークショップは比較的技術的な内容で、主に開発エンジニアと運用エンジニア及びその直属の管理者を対象としていますが、技術志向の製品担当者や経営管理職の皆様も参加いただければ、更に効果が発揮されます。SLO の目標はユーザーを念頭に置いて設定することが求められます。エラー バジェットはその目標を超えた場合の結果に対して経営陣のサポートがあってこそ、組織の部門間の緊張も解決されるからです。

ワークショップの構成

このワークショップは終日のイベントとして行うことが推奨されますが、デッキの編集を慎重に行うことで、2時間半に短縮することもできます。下記のセッションの時間割は、あくまでも提案です。ワークショップ全体を通して参加者からの質問にいくらか時間をかけても、以下の時間割で実施できることがわかっています。

09:30–09:45	はじめに
09:45–10:45	SLI、SLO とエラー バジェット
10:45–11:00	休憩
11:00–12:00	SLO と SLI の作成
12:00–13:00	昼食
13:00–14:30	実践演習
14:30–14:45	休憩
14:45–15:00	解答例
15:00–16:00	Q&A セッション

「はじめに」の部分のスライドはありません。ここではプレゼンターを紹介し、主催者の会社の事情に合わせてワークショップを説明し、トイレの場所や火災の場合の避難経路などの事務的な注意事項の説明を行います　(必要な場合)。どのような人たちが参加をしているのか、少し時間を費やすとワークショップで参加者にどのようなメッセージを伝えればよいのかの判断に役立ちます。自己紹介を行うときは、ソフトウエア エンジニアや運用エンジニアの方から挙手をして自己紹介をしていただく、または経営管理職の役職の方から挙手をして自己紹介をしていただくなどのやり方で行うのもよいでしょう。

Q&A セッションは特定のニーズによっては関係ない内容も含まれるかもしれませんが、SLO を使用してサービスの信頼性を管理するための組織的な取り組みの一環としてこのワークショップが実施される場合、Q&A セッションは参加者の懸念に対応する絶好の機会になります。

ワークショップの進行

人々

十分な参加者がいないと、ワークショップを実施する意味がありません。提供するサービスに対する SLI と SLO の作成方法を学ぶことによって、具体的な価値が得られるような合理的な人数に参加者数が満たない状態で、Art of SLO のワークショップを実施することにはあまり意味がありません。実践的な演習は、6 ～ 8 人のグループで一緒に作業するように設計されています。1 グループだけでの実施も可能ですが、組織としてのオーバーヘッドは同じなので、可能な限り大人数で実施することが理にかなっています。我々はこれまでに、80人以上の参加者が集ったワークショップを成功させたこともあります。

ワークショップを円滑に進めるために、コンセプトをよく理解し自発的に協力してくれるボランティアを見つけることが重要です。そうしたボランティアはサービスで実際に SLO を使用した経験のある SRE や運用エンジニアなどの場合もあります。各テーブルやグループごとに 1 人のボランティアを用意する必要はありません。内容を事前に理解している人がいると自分たちで問題を解決しようとせず、そうしたボランティアにリードしてもらうことを期待しがちになるので、ボランティアがいることがかえって逆効果になることもあります。また一方では、うまくスタートできないグループもあったり、SLO を作成するために必要な最小限の前提条件にフォーカスするのではなく、システム設計の詳細に入ってしまうグループもいたりします。細部にとらわれずに話し合いを軌道にのせてくれる専門家がいることで、すべての参加者にとってスムーズな学習体験ができるようになります。

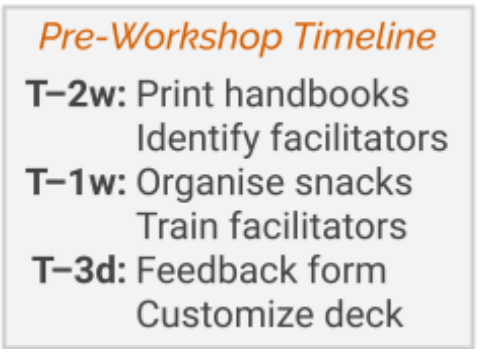
こうしたボランティアは、ワークショップ後の Q&A セッションのパネル メンバーとしても活躍してくれます。

場所

ワークショップの会場としては、プレゼンテーション機器を備えた大きな部屋が必要です。会場の設定で最も重要なことは、参加者と一緒にグループで実践的な演習ができるように、「カフェテリア スタイル」 (会場全体に正方形または円形のテーブルをあちこちに配置する) 会場設定をすることです。一般の会議場ではこのような配置が難しい場合もあるので、設営のリクエストは早めに会場に伝えるようにします。 同じ場所に休憩とランチタイムのスペースがあることも非常に役立ちます。休憩のたびに参加者がワークショップの会場に戻ってこなければならない場合は、貴重な時間が無駄になる可

能性があり、休憩場所がワークショップ会場から遠ければ遠いほど時間のロスが発生します。

準備



イベントの数週間前に、参加者全員に当日配布するハンドブックを印刷します。 オフィスのプリンターに二つ折りや中とじホチキス留め機能などがある場合は、少人数のグループであれば資料の印刷製本を自分で行うことは可能ですが、 7 枚の用紙の二つ折りをマニュアルで行い、ホチキスで留めて 28 ページの小冊子にするのは大変な作業です。それなりの人数の参加者に資料を準備する場合には、1 ～ 200 冊ぐらいまで手頃な価格で小冊子の印刷を代行してくれるオンライン印刷サービスは数多くあります。 更に、SLO ワークシートも十分な枚数を印刷して、各テーブルに少なくとも 5 つのワークシートを準備してください。

この時点で、ボランティアと Q&A のパネル メンバーも特定しているはずです。 たとえこのワークショップの進行に比較的慣れていたとしてもイベントの 1 週間ほど前に、「トレーナーのトレーニング」セッション (このハンドブックの後半を参照) を行うとよいでしょう。これにより、ユーザー ジャーニー (ひとつの目的を達成するための、架空のモバイル ゲームのサービスを提供するインフラストラクチャとユーザーとの間の一連のインタラクション) を振り返り、質問に答えたりテーブル ディスカッションをすすめるための準備が整います。

終日のイベントになる場合は、昼食のケータリングと休憩用の軽食が必要です。参加者がワークショップを通じて頭を回転させるためには食事が必要です。ケータリングの準備には時間がかかる場合もあります。

ワークショップの数日前に、元となる公開デッキのコピーを作成してカスタマイズする必要があります。独自のロゴやプレゼンターの名前をスライドに追加するなど単純なことですが重要です。フィードバック フォームを作成し、参加者に聞きたい質問を追加します。Google フォームの「コース評価」テンプレートをまずは使用することをお勧めします。多くの無料のオンライン ジェネレーターを使用してフォームにリンクする QR コードを生成し、この QR コードを「ご参加いただきありがとうございました」のスライドの指定の位置に添付します。ほとんどの人は、このリンクをスキャンできる携帯電話を持っていると思いますので、Q&A セッション中に携帯からフィードバック フォームに入力

できます。その場で評価をしていただくことで、得られるフィードバックの量が大幅に増加します。

当日

当日は参加者よりも少し前に会場に行きます。会場設営と AV 機器のテストに加えて、印刷されたハンドブックとワークシートを各テーブルに準備します。静かなクラシック音楽をかけるなどして、参加者が会場に到着したときに参加者を歓迎する雰囲気をつくることもできます。参加者用の筆記用具を用意するのも忘れないようにしてください。

ワークショップを開始する前に、参加者全員にハンドブックが行き渡っているかどうかを確認してください。2　番目のスライドの「ようこそ」のアクティビティをとばさないようにしてください。参加者がグループの一員であると感じれば、率先して質問をしたり、自発的に回答してくれるようになります。

デッキは、ほぼ同じ量の 4 つのパートに分かれていて、30 分のセクションごとに参加者の質問をうけるようになっています。質問はセクションごとに約　5 分に制限するようにしてください。そうでないと、時間が足りなくなります。スピーカー　ノートには、箇条書きと文章の両方が含まれます。アドリブで話をするのが苦手な場合は、スピーカーノートに記載の文章をそのまま読むこともできます。

デッキには、参加者に考えてもらったり、話しをしてもらうためのきっかけとしての双方向のアクティビティが多くあります。質問が多く出て時間が足りなくなった場合は、アクティビティを手短に行うか、完全にとばしてしまうこともできます。参加者が静かだったり、早く進みすぎている場合は、内容を掘り下げて参加者の意見の違いを引き出すこともできます。

「ゲーム内通貨の購入」はデッキで回答例が出されているユーザー　ジャーニーなので、午後はすべてのテーブルで最初の問題に取り組むことをお勧めします。参加者は、4　ページの時間割の例から　90　分のセッションでこのジャーニーともう1つのジャーニーをカバーできるはずです。どの問題に挑戦するかを各テーブルで決定できるようにすることをお勧めします。

こうしたワークショップを複数回行う計画がある場合は、ワークショップ後のふりかえりの用としてノートをとり、質問やスライドのタイミングを記録する人を決めておくと、次回のワークショップのスムーズな進行に役立ちます。

宿題

このワークショップの目標は、参加者全員がサービスの SLO を定義することの必要性を認識し、実際にその定義を行えるように自信をつけてもらいワークショップを終了することです。そのため、会社の人たちを対象にこのワーク

ショップを実施している場合は、会議の出席者に宿題を出すことは憚られますが、参加者の責任範囲となっているサービスに対して最初の SLO を設定してもらうようにお願いすることは有益な場合があります。

よくある質問

このワークショップを行うと必ず聞かれる傾向のある質問が一定で発生します。そのような質問の答えをスライドですべてとりあげると、あっという間に 2 倍の時間がかかり、一冊の（短い）本ができあがるくらいになります。ここでは、そうしたよくある質問をリストにまとめましたので、事前に考えて準備するようにしてください。

SLO に関する質問

- ユーザーの満足度はどのように測定するのですか?
- SLO を始めるにはどのような手順をふむのですか？
- SLO を始めるためのソフトやツールはありますか？
    - 監視用の SLO も必要ですか?
- SLO を文書化するにはどうしたらよいですか？
    - SLO を再考し変更した場合は、昔のSLOも記録しておくべきでしょうか？
- 「カバレージ」と「正確性」の違いは何ですか?
    - (このテーマに出てくる他の用語も同様に)
- 「修正」が短期間でより多くのエラーを意味する場合はどうなりますか?
- 「計画的」ダウンタイムと「偶発的」ダウンタイムの違いは何ですか?
- 競合他社のパフォーマンスに基づいて SLO 目標を設定する必要がありますか?
    - 99.999 で十分ですか？
- サードパーティ製のシステムの SLO を作成するにはどうすればよいですか?
    - サードパーティ製のサービスに障害が発生した場合、私達のサービスに対して何をすべきかを定義する必要はありますか？
    - ユーザーと私達のサービスの間の CDN についてはどうですか？
- 100 以上のマイクロサービスがありますが、すべてに SLO が必要ですか?
    - SLO はボトムアップで定義する方が良いですか? それともトップダウンですか？
    - マイクロサービスごとのきめ細かなレベルの SLO をシステム全体として集計するにはどうすればよいでしょうか?
- 時間帯や季節で異なる SLO の目標を設定することはできますか?
    - 違うタイプのリクエストを不均等に重み付けすることはどうですか?

- SLO でサービスの低下をどのように表現しますか?

SRE に関する質問

- SRE　チームを立ち上げる際におかしやすい最大の間違いは何ですか?
- SRE を採用する際に何を重視しますか?
- 組織全体でこれらの概念が一貫したレベルで理解されていることをどのように確認しますか?
- 締め切りが迫っているプロジェクト　マネージャーに対して、どのように抵抗しますか?

ワークショップのボランティア

ワークショップが開催される約 1 週間前に、すべてのボランティアを集めて、「トレーナーのトレーニング」セッションを行います。このセッションは、ボランティアの皆さんにゲームのサービスを提供するインフラストラクチャと、参加者が SLO を作成できる 5 つのユーザー ジャーニーに慣れてもらうためのものです。事前に、各ボランティアに両方のハンドブックのコピーを渡し、ハンドブックにある「ゲーム内通貨の購入」のジャーニーの詳細を読んでおいてもらいます。トレーニング セッションの所要時間は約 1 時間で、次のような時間割になります。

XX:00–XX:10	段取りと基本ルール説明
XX:10–XX:15	ゲームのサービスを提供するインフラストラクチャ
XX:15–XX:30	ユーザージャーニーの概要
XX:30–XX:35	4 ステップ プロセスの説明
XX:35–XX:55	「ゲーム内通貨の購入」の詳細

段取りと基本ルール

ランチタイムの終わり頃 (ワークショップ セッションが始まる前) にボランティアの人たちに来てもらい、午後のほとんどは一緒にワークショップを手伝ってもらいます。 Q&A セッションを行う場合は、この時に Q&A セッションに参加するの人は誰になるのかを決めておく良い機会です。

ボランティアは、議論が本筋から逸れないように維持し、各テーブルで意見をまとめられるように準備します。よくある失敗は、参加者がサービスを提供するインフラストラクチャの詳細な設計に入り込みすぎてしまうことです。合理的な SLO を定義できる最小限の前提条件を設定さえすれば、順調に進みます。ボランティアは時間に注意を払い、参加者が脇道にそれている場合や 1 つのジャーニーの SLI に 30 分以上の時間を費やしている場合は、SLO ワークシートと 4 ステップ プロセスに戻すようにします。

逆に、すべてのジャーニーを早々とやり終えてしまっている場合、おそらくそのグループではいずれのジャーニーも十分に深く掘り下げていないと思われます。そのグループで導き出した SLI では、サービスを提供するインフラストラクチャの障害モードの少なくとも一部を捉えることができないでしょう。

したがってボランティアは、そのようなグループには、検出できない障害を考え出すようにうながします。

ボランティアがサポートをする時に心に留めておくとよいことをリストにまとめました。

- 参加者は楽しく学習するためにワークショップに参加しているので、優しくする ;-)
- テーブルの端ではなく、中央に座る。
- 全員に声が聞こえるように、また平等に発言できるようにする。
- 答えを教えるのではなく、質問をして考えてもらう。
- アーキテクチャの定義が不十分な部分については、正当化できる仮説を立てて先に進めるように、参加者をうながす。
- 特に測定の戦略を選ぶときには、トレードオフについて議論するようにうながす。完璧な唯一のアプローチはない。

サービスを提供するインフラストラクチャとユーザージャーニーの提供

ワークショップ ハンドブックには、基本的なインフラストラクチャの ブロック図と、参加者にやっていただく 5 つのユーザー ジャーニーの説明があります。ワークショップが始まる前にボランティアの皆さん全員がサービスを提供するインフラストラクチャに期待される機能を理解し、各ジャーニーの SLI についてアイデアを持っていることが重要です。この小冊子の次のセクションでは、各ジャーニーについていくつかの考慮事項と、「ゲーム内通貨の購入」と「アプリの起動」の両方のジャーニーに関する詳細な説明があります。

こうした説明文は、45分　　で演習を終えるためだけではなく、もっと詳細にジャーニーについて説明をしており、測定の戦略または優先順位付けの選択に関する、工学的決定の根底にある思考プロセスに洞察を与えることを目的としています。ボランティアのみなさんにワークショップの前に、残りのジャーニーの詳細な説明文を作成することを、宿題として取り組んでもらうとよいでしょう。

4 ステップ プロセス

ワークショップの参加者向けには、ハンドブック p.14 から 4 ステップ プロセスについて、非常に簡単な手順を載せています。実践的なセッションの後、参加者には詳細な説明文の要約版を見てもらいます。トレーニング セッションの後半では、ワークショップ プレゼンテーションのスライドの p.76 ～ 92 を説明し、ボランティアの皆さんがジャーニーと解答例を理解しているかどうか

を確認します。さて、ボランティアの皆さんは、ワークショップの参加者が SLI を導き出せるように十分なサポートをする自信がついたでしょうか？

ユーザー ジャーニーの概要

ゲーム内通貨の購入

考慮しなければならない重要事項は次のとおりです。

- 2つの重要なリクエスト/レスポンスのインタラクションは、ゲームのサーバー側インフラストラクチャではなく、Play ストアとのやり取りです。
- ユーザーは通常、SKU リストを開いた後に商品を購入しません。購入率は 1 ～ 2% と想定しても良いでしょう。
- Play ストアが応答する、OK ではないステータス コードの多くは、ビジネスまたはユーザーの観点からは、エラーではない可能性があります。

内容を少し難しくしたい場合は、購入の正確性についての質問をしてください。ただし、多くのサポートも必要ですのでその準備もしてください。 Play ストアはトランザクション リストを CSV 形式で日次で提供するため、システムが正しいゲーム内アイテムをユーザーに提供したかどうかを判断できるはずです。

考えられる SLI:

- ロード バランサの指標と合成クライアントの組み合わせで測定された、「SKU の取得」の可用性とレイテンシ。
- クライアント側の計測で測定された「SKU の購入」の可用性とレイテンシ。
- Play ストアの購入ログに対して日次で実行される突合パイプラインで測定された「SKU の購入」の正確性。

アプリの起動

考慮が必要な重要事項は次のとおりです。

- (登録 + 認証) と (同期) の間に大きな QPS の相違があるということは、これらを単純に集約できないことを意味します。
- ユーザー名の検証はクライアントで対話的に行われるため、残りのフローには当てはまらない、厳密なレイテンシを保証することが求められます。サーバー側の検証が必要になりますが、それには SLI は不要でしょう。
- 登録の失敗によるビジネスへの影響は、認証の失敗による影響よりも大きくなります。この場合は、ユーザーは二度と戻ってこない可能性があります。

- サーバーからゲームの状態をダウンロードするのは比較的速くできるはずですが、CDN のアセットは新規インストールではかなりの量になる可能性があるため、クライアント側全体の同期のレイテンシの測定を行うと問題が生じます。

内容を難しくしたい場合は、最初と 2 番目のステップで Facebook や Google などのサードパーティ認証プロバイダーへの OAuth の呼び出しの追加を検討してください。

考えられる SLI:

- 合成クライアントで測定されたジャーニー全体の可用性。
  - 新しいアカウントを作成し、新しいユーザーとして認証し、ゲームの状態を同期してから、空のアカウント データを検証します。
  - 既知の「ゴールデン」ユーザーとして認証し、ゲームの状態を同期してから、応答が既知のデータと一致することを検証します。
- (作成、認証) および (同期) フェーズの可用性とレイテンシは、ロード バランサ の指標で測定されます。
  - ロード バランサは、通常どおり可用性において「500 番台のレスポンス コードは失敗」とみなします。400 番台のレスポンスコードについて質問することを忘れないでください。例えば、「パスワードが間違っている」「ユーザーが存在しない」などは SLI に含めるべきではありません。

陣地の管理

考慮が必要な重要事項は次のとおりです。

- ユーザーの観点から見た非同期的な成功。ユーザーは構築が完了することを気にしますが、これは 300 秒後にわかります。
- ここには 3 つの リクエスト/レスポンスのペアがありますので、集約が重要になります。

内容を難しくしたい場合は、参加者にこれらのアクションに対する正確性の SLI の作成を行ってもらいます。 ユーザーが構築アクションを開始したもののうち、構築が完了しなかった回数は何回でしょう？ このようなことは、例えばプレーヤーの陣地をホストしているゲーム サーバーがトランザクションの途中で突然停止するような場合に発生することがあります。

考えられる SLI

- ロード バランサで測定されて集約されたすべてのエンドポイントの可用性とレイテンシの複合メトリクス。

- 合成クライアントによって測定された可用性の複合 SLI。
  - クライアントは、構築が完了したことを DB へのサイドチャネルを介して検出します。
- サーバー側で測定されるゲーム サーバーのスループットの SLI。
  - ゲームにおいてXミリ秒以下で実行されたティックレート（更新処理の割合）として表されます。
- ゲームアクションの正確性の SLI。API サーバーからゲーム アクションの Pub/Sub フィードをパブリッシュし、期待される構築時間が経過した後に、それらのアクションがゲームの状態に反映されたことを、リスナーが検証することによって測定します。

別のプレーヤーとの対戦

この対戦のしくみまわりのゲームデザインは非常に不完全であるため、このジャーニーに取り組むときは、参加者に仮説を立ててもらうことが必要になります。ここでは次のようなことを検討します。

防御側は対戦に参加するかしないかを選べるでしょうか？ 攻撃側には選択肢があるので、防御側もおそらく選択肢があるはずですが、これによって、対戦の準備段階における対戦相手のマッチング アルゴリズムやレイテンシーに対する期待は、どのように変わってくるでしょうか？

ゲームの状態を対戦の進捗とともにリアルタイムで更新する必要があるでしょうか？ リアルタイムに更新することで、対戦中に発生した「負け」が、ネットワークが (場合によっては故意に) 切断された後も存在し続けることになります。しかしこれがシステム設計と SLI にとって何を意味するのでしょうか？

それぞれの対戦の時間はどのくらいでしょう？ 我々は、対戦時間は 3 ～ 5 分、攻撃側は 30 秒ごとに次の部隊を投入できることとし、防御側はいつでもタワーを設置できる、と想定しました。

考慮が必要な重要事項は次のとおりです。

- 対戦のトラフィックがゲーム サーバーを経由するかどうか。我々は、次の理由から、おそらく実際の実装ではサーバーを経由することになると想定しました。
  - キャリア ネットワーク内での NAT の普及を考えると、デバイス同士の直接通信は、まさに悪夢™ になります。
  - 対戦のトラフィックをプロキシすることで、ゲーム内の分析が可能になります。それはビジネスの観点からは望ましいことと思われます。
  - サーバーをゲームの状態の真実のソースにすることで、悪意のあるクライアントから対戦が「いたずら」されることを防ぎます。

- プレイヤーの行動。負けているプレイヤーは、良くない結果を避けるために、意図的にネットワークを切断する可能性があります。単純な SLI は、このような意図的なネットワークの切断やその他の接続の問題を考慮していない場合があります。一般的には、「成功」の定義は、人々が最初に想像するほど明確に切り分けられるわけではありません。
- ユーザー ジャーニーの HTTPの リクエスト/レスポンスと UDP フェーズをどのように扱うべきか。SLI を検討するにあたって、各 UDP パケットは別の「イベント」と見なすべきでしょうか？

この内容を難しくする必要はあまりないと思いますが、早く終わってしまったグループには、ゲーム サーバーのスループット SLI の検討を行ってもらいます。これが重要である理由は、ゲーム　サーバーはこうしたバトルの多くを同時に処理することになるからです。対戦中にユーザーがリアルタイムだと認識していても、実際には何らかの形で時間の量子化が行われるはずです。もし、たとえば過負荷によって、この量子化がスリップすると、ユーザーはゲームが遅いことに気付くでしょう。

考えられる SLI

- ロードバランサで測定された　launchAttack　のエンドポイントの可用性。
- API サーバーで測定されたプレーヤー マッチングのレイテンシ。
- API サーバーで測定された対戦準備のレイテンシ。
    - これは、防御側に選択肢があることを前提としているため、全体的な攻撃開始のレイテンシを測定することはできません。
- ゲーム クライアントで測定された対戦中のアクションのレイテンシ。
- ゲームサーバーで測定された対戦アクションのスループット。

リーダーボードの生成

このジャーニーは、Apache Beam のモバイル ゲームの例[1] にゆるく基づいています。参加者は、スコアリング　データがどのように保存され処理されるかについて、仮説を立てる必要があります。簡単な方法は、リーダーボードストアは　BigQuery　のようなものを想定し、また、提供されるスナップショットはそのクエリの実行結果を　BigTable　に書き込むことによって生成すると想定することです。

考慮が必要な重要事項は次のとおりです。

- 可変部分は多くありますが、そのなかでもプレイヤーが一番に気にする部分はどこでしょう？

[1] https://beam.apache.org/get-started/mobile-gaming-example/

- SLI　の測定を実現するには、データ処理を行うアーキテクチャを介して、追加で何かしらのデータを伝えることが必要になります。

考えられる SLI

- 提供するスナップショットの鮮度。スナップショットに反映されるまでにかかる時間、つまりデータ処理を行うアーキテクチャにおける直近のゲーム完了の時刻と現在の時刻の差。
- リーダーボード ストアとアーカイブにおけるゲーム完了のカバレッジ。
- エリアごとのトップ スコア テーブルの正確性。アーカイブされたスコアデータから再構築することによって検証する。

ゲーム内通貨の購入

「購入フロー」には、5 つの HTTP リクエスト/レスポンス のペアが含まれます。サーバー側から見えるのはこれらのうち 3 つだけですが、フローが成功するには、すべてが成功しなければなりません。その中で最も重要なのは、Play ストアへのリクエスト[2]です。サーバー側またはフロント エンドの指標のみに基づく SLO は、ジャーニー全体の信頼性について適切に検討する上では十分なカバレッジを提供しません。

Play Billing テストの API[3]を使用して、Play ストアでの購入用の合成クライアントを構築することができます。 これはそこまで実用的なものではありませんが、「Play Billing が機能しているか」を判別するための簡易な信号として使用することができます。 getSKU、Play ストア SKU 詳細リクエスト、completePurchase を合成クライアントでテストすると、より扱いやすくなります。また、Play ストアの請求フローが成功したことにして、合成クライアントが既知の偽物の購入トークンを送信すれば、API サーバーは必要に応じて特別なケースとして扱うことができます。

これは収益を生み出す重要なジャーニーであるため、クライアント側の埋め込み機能を構築することは妥当な投資であり、合成クライアントよりもユーザー体験の全体的なカバレッジが向上します。ユーザーが彼らのデバイスからこの種の統計情報を送ることに反対する可能性を考慮することは大切です。ユーザーが統計情報の送信に同意するかしないかを選べるように、ゲームの中に設定機能を準備します。

購入の可用性

SKU リストを読み込む人の多くは何も購入しないため、可用性に対してジャーニーを 2 つの部分（「利用可能な SKU の表示」と「SKU の購入 」）に分割することが重要です。成功したトランザクションごとに測定可能な収益が発生するため、会社における真の価値は「SKU の購入」にあります。したがって、ジャーニーのこの部分で可用性を測定することが最も重要です。Play ストアが返すエラーには正当な理由[4]があるものも含まれるので、「成功」とは何かを具体的にすることが重要です。単に OK だから成功、というわけではありません。

可用性の **SLI**: Play ストアへのリクエストが次の成功コードのいずれかにに当てはまる、同意を示したユーザーが開始した購入フローの割合:

- `OK`
  `Hopefully self-explanatory`

[2] https://developer.android.com/google/play/billing/billing_library_overview
[3] https://developer.android.com/google/play/billing/billing_testing
[4] https://developer.android.com/reference/com/android/billingclient/api/BillingClient.BillingResponseCode

- FEATURE_NOT_SUPPORTED
  User is using an unsupported android / play store version, we can't fix this.
- ITEM_UNAVAILABLE
  Either a race condition (we disabled the item since they listed the available SKUs) or someone faking SKUs.
- USER_CANCELED
  User gave up trying to buy the SKU. This conveniently includes all forms of payment declined errors, because the buy flow prompts users to try other forms of payment and forces them to cancel if none work.

そして /api/completePurchase へのリクエストは、次の HTTP ステータスコードのいずれかになります。

- 200 OK
  Hopefully self-explanatory, again
- Any 4xx code
  Client errors. Specifically we use 402 Payment Required to signify to the requestor that the validation of their purchase token with the Play Store failed.

そして、ゲームクライアントによって測定され非同期に報告される JSON のレスポンスが、パース可能で正当な内容か否かで検証します。

購入のレイテンシ

デバイス側の請求フローには、リクエスト以外の「ユーザーがデバイスを操作する」時間が含まれていて、これらの時間は不確定です。そのため、購入レイテンシの SLO では completePurchase API 呼び出しのレイテンシを測定します。これには、Play ストアへの比較的安価な呼び出しとデータベースへの書き込みが含まれます。これをロード バランサで測定し、非常に変動が大きく制御不能なデバイスの数字を捕捉しないようにします。

レイテンシの **SLI:** ロード バランサがリクエストを受信してから完全なレスポンスがサーバーから返されるまでの時間が 1000 ミリ秒未満の completePurchase API リクエストの割合。

getSKUs と SKUDetails

これらの SLI のいずれも、getSKU や SKU の詳細の側面はカバーしていません。1 つのジャーニーごとに 3 ～ 5 個の SLI を設けるというガイドラインを守るには、ここでは少しズルをします。多くの API エンドポイントを集約した、一般的な「読み取り専用 API の可用性」および「読み取り専用 API レイテンシー」の SLI がすでに存在していると想定します。getSKUs API 呼び出しは、このような集約された SLI に含める良い候補です。これらすべての API リクエストは、合成クライアントのリクエストとロード バランサの指標の組み合わせでカバーされます。

「SKUの詳細」の可用性を捕捉するためには、合成クライアントに Play ストアへリクエストも送信させる必要がありますが、これを API の可用性 SLI に含めたくないかもしれません。代わりに、このようなリクエストが失敗した際、「Play ストアに接続できませんでした」などのエラー メッセージがゲームに表示されると、ユーザーの不満が私たちのサービスに向けられることはありません。このようなメッセージは根底にある問題を正確に反映しているので、ユーザーに嘘をついていることにはなりませんし、Play ストアの問題なので、いずれにしても我々にできる手立てはありません。

購入の正確性

次の理由から、正確性はこのジャーニーに適しています。(1) ビジネスにとって重要であり、収益を生んでいる。(2) インタラクションには、ユーザーが非常に気にする重要な副作用 (ゲーム内通貨の増加) があり、そうしたやりとりの流れが複雑でユーザーが途中で抜けてしまうポイントがいくつもある。(3) 検証用の外部データ ソース (Play ストア) がある。

クロスチェックには多くの値があるため、すべてが適切に行われたことが確認できます。ユーザーは、 Play ストアの請求フローを正常に完了しても、しばらくの間、/api/completePurchase にリクエストを送信できなくなることがあります。このような場合、Play ストアのガイドラインではアプリの起動時に購入を再試行することを推奨しています。

Play ストアは、指定した月のゲームにおける全てのトランザクションの記録が記載された財務レポート[5]を生成します。財務レポートは日次で更新されます。これを使えば、実際の口座残高とこのデータから計算された残高を比較する突合パイプラインを構築できます。日初に各ユーザーが持っているゲーム内通貨の残高のスナップショットを取得し、その日に費やしたり稼いだりしたゲーム内通貨の量を追跡します。その日の終わりには、これを財務レポートにあるその日に購入したゲーム内通貨の量と組み合わせて、その日の終わりに持っているはずの残高を導き出します。

正確性の **SLI**: 実際の残高と、前日の実際の残高、Play ストアの注文情報およびその日の支出履歴から計算された想定残高が一致するユーザーの割合。

SLO の目標

すべての SLO は、長期と短期の優先順位付けのニーズの適切なバランスを保つため、28 日間の移動期間を使用します。このような SLO を使ってインシデント対応を促す場合には、1 時間、12 時間、および 1 週間の移動期間でも測定します。

[5] https://support.google.com/googleplay/android-developer/answer/6135870?hl=en-GB

購入の可用性: 購入の 99.95% が、過去 28 日間の移動期間において、成功すること。

Play ストアの請求フローが 99.99% の可用性があることを想定しているため、ここで 99.99% を目標にしてしまうと、Play ストア以外の障害を許容できません。 99.95% の目標では、重要な収益に関わるユーザー ジャーニーに対して適切な厳格さを保ちつつも、処理中のクライアント情報取得のバッファが大きくなるなどの、他の障害のための余裕があります。

購入のレイテンシ: completePurchase API リクエストの 99% が、過去 28 日間の移動期間において、1000 ミリ秒未満で処理されること。

この SLO ではロングテールをターゲットにしています。 アプリ内購入は、ユーザーがより多くのお金を使うように、スピード感が必要になりますので、この検証ステップが購入フロー全体を滞らせてはなりません。

購入の正確性: ユーザーの 99.999% は、過去 28 日間の移動期間において、実際の残高と人工的な残高とを検算した結果が一致すること。

究極の到達点、99.999%！ 30 日間のアクティブ ユーザーは 5,000 万人います。1 日あたりそのうちの 1% が購入すると仮定すると、この目標の場合は 1 か月あたり約 140 件の残高に一貫性がなくなるようなエラー バジェットになります。これは、サポートチームが個々のケースに対処できる十分な数です。

アプリ起動

ジャーニーを分解

このジャーニーの SLI を検討する際に最初に行うことは、アプリ起動のプロセスの各フェーズの相対的な重要性について、いくつかの仮説を立てることです。ユーザーがゲームをプレイできるようになるには、3 つのフェーズすべてが正常に完了されなければなりませんが、アプリを起動するたびに発生するのは最後のフェーズである syncData だけです。この API 呼び出しに影響する障害は、他の 2 つのいずれよりも多くのユーザーに影響を与えるため、ジャーニーのこの部分が最も重要であると考えます。

すでにアカウントを持っていながら新しいデバイスに移行したばかりのユーザーは、エラーが発生しても、おそらく最初からもう一度試すでしょう。それまでの間、syncData が機能している限り、古いデバイスで引き続きプレイすることもできます。したがって、これはジャーニーの中でおそらく最も重要でないフェーズです。アカウント作成のフェーズでエラーを受け取ったユーザーは、友達と一緒に遊びたいなどの社会的なプレッシャーがない限り、再試行することはほぼないと言えます。

実際、リクエスト率の違いから同期フェーズの可用性は他の 2 つのフェーズの可用性とは別に測定する必要があるでしょう。ユーザー名の検証はユーザーが好みの名前を入力してバックグラウンドで行われるため、この部分は全て無視します。クライアントは非同期的に検証を行い、テール レイテンシを抑えるために短いリクエスト タイムアウトを設定し、検証要件が失敗した場合には何も表示しません。いずれにせよ、競合する状況を避けるためにもアカウント作成の検証をサーバー側で行います。

測定戦略の選択

ユーザーのやりとりに関するデータがゲーム開発にとってどれほど重要であるか考えると、ゲーム クライアントにはすでに何らかの情報取得機能があると想定するのが妥当です。この取得されたデータを受信するサーバー側のインフラストラクチャは、信頼性を念頭に置いて構築されており、その上に高可用性の SLO を構築するのに適した損失が少ないデータ取り込み機能を提供していると想定します。こうした想定から、SLI の指標の情報源としてのクライアント埋め込みの主な欠点 (実装のコスト) のうちの 1 つを取り除けるようにみえます。このジャーニーをカバーするために開発するすべての SLI の測定戦略として、これをデフォルトにするのはどうでしょう？

Art of SLO では、通常の運用中に SLI が持つ値の範囲が、システムに障害が発生したときの値の範囲と明らかに異なる場合にのみに SLI が役立つことを紹介しています。クライアント側で取得したデータ、特にモバイル クライアントからのデータには、本質的にノイズが含まれています。モバイル ネット

ワークの品質は変動が非常に大きく、ネットワークは予期せず切断され、また能力が大きく異なるさまざまなデバイスがあります。この変動の要因の多くは、我々の制御の範疇にはなく、その要因を分離することは困難です。 高い変動要因を SLI に含めると、障害が発生しているときとそうでないときの SLI の範囲が重複する可能性が高くなり、S/N 比が低下します。

実際には、クライアント側の埋め込みから実用的な SLI を作成するには、取得したデータを前処理して、我々が気にする必要がない変動を削除し、我々が重視する変動部分を明確にする必要があります。この種のデータ処理は複雑で、開発と微調整に時間がかかります。このユーザー ジャーニーでは、基礎データから高品質の SLI を導き出すために必要なフィルタリングの構築および維持と処理方法の試行錯誤にかかるエンジニアリング コスト (何人月もかかる可能性がある) のデメリットは、ユーザーの近くで SLI を測定できるわずなメリットを上回ります。

それよりも、ロード バランサで測定された HTTP ステータス コードに基づく SLI を使用して、少ないエンジニアリング工数で妥当なカバレッジを提供するほうが、現実的です。この測定戦略には、主に不正な応答を検出できないという既知の欠点があります。ただし、ロード バランサの指標が既に取得できているとすれば、その欠点を補う合成クライアントを構築するために必要な工数は、おそらく 1 人月未満の仕事量になります。ひとつの合成クライアントの SLI は、すべてのジャーニーを一度に簡単にカバーでき、全体的な処理の流れが正しく機能することを検証し、ロード バランサの SLI に内在する測定の穴を埋めることができます。エンドポイントごとのロード バランサの SLI を、ひとつの全ジャーニーの合成クライアントの SLI と結合することは一般的であり、多くのシナリオで推奨できるパターンです。

SyncData ロード バランサの可用性

上述のとおり、このジャーニーの最も重要なフェーズは、ゲームの状態をデバイスに同期することですので、まずはそれに取り組むことにします。ユーザー側の誤操作や偽物の (または悪意のある) クライアントが引き起こす 400 番台のレスポンスの量は、サーバーが正当なリクエストに対して 403 や 404 などのレスポンスを返してしまう確率をはるかに上回ると予想されるため、我々は 400 番台のレスポンスを成功として扱います。もし SLI にそれをエラーとして含めてしまうと、このバックグラウンドノイズのために、500 番台のレスポンスの割合がわずかに増加した場合の信号が埋もれてしまいます。

ここでのリスクは、コードまたは設定のバグにより、「全ユーザーの認証が不具合を起こす」または「ロード バランサが /api/syncData にトラフィックを送信しない」といった大規模な問題が発生する可能性です。幸いなことに、こうした問題は合成クライアントが捉えます。

SyncData ロード バランサの可用性: ロード バランサで測定された、200 番台、300 番台、または 400 番台のレスポンス コードが返答される /api/syncData へのリクエストの割合。

SyncData 可用性の **SLO:** 過去 28 日間において、/api/syncData へのリクエストの 99.95% 以上が成功すること。

認証ログベースの可用性

ジャーニー全体を検証する合成クライアントの詳細を掘り下げる前に、ジャーニーの他のフェーズを個別に検討してみましょう。ハンドブックによると、実際のアカウント作成とトークン交換のレートは非常に低いということです。これは、1 つの「ユーザーの作成」と 1 つの「新しいデバイス」がおよそ 10 秒ごとに発生することを意味する、と仮定しましょう。アカウントの作成ごとに、さらに後続の認証トークンのリクエストが必要になるため、トークン交換とアカウント作成は 2 対 1 の割合で、あわせて 10 秒ごとに 3 つのリクエストが発生することになります。

ここで問題となるのは、こうしたエンドポイントに対して合成クライアントが本物のユーザーとほぼ同じ量のトラフィックを作り出し、ロードバランサの指標がこの 2 つのトラフィックの送信元を識別しなくなることです。

この問題の対処方法の 1 つとして、認証の可用性の SLI をロードバランサの指標ではなく、ログ分析を基礎とすることです。外部監視装置に「wooden-stake」(木の杭; これは吸血鬼ゲームですからね) のような既知の一意的な UserAgent を使用させ、ログを処理して「良いイベント」と「すべてのイベント」をカウントするときに、これらのリクエストを無効として無視することができます。合成クライアントの SLI は、短い時間枠の SLO で使用して、短時間で障害対応を促すことができます。ログベースの SLI (取り込みと処理のレイテンシで困るかもしれません) は、より長い時間枠のSLOで使用して、チームの優先順位の決定を促進できます。

認証ログベースの可用性の **SLI:** 成功した有効なリクエストログの割合

- 有効なリクエストは、UserAgent に `/^wooden-stake v[\d.]+$/` が、リクエスト パスには /api/createAccount または /api/getAuthToken が含まれている。
- 成功したリクエストの HTTP ステータス コードは 200 番台、300 番台または 400 番台とする。

認証ログベースの可用性の **SLO:** 過去 28 日間において、99.95% 以上の本物のユーザーの認証リクエストが成功すること。

アプリ起動の合成クライアントの可用性

ジャーニー全体を実行しようとする合成クライアントは、その個々の部分をカバーするステータス コード ベースの SLI の補完的な役割を提供します。次のような合成クライアントを使用することで、ジャーニーが正しく機能していることを検証できます。

1. 新規アカウントを作成
2. 作った新規アカウントの認証トークンを要求
    a. /api/syncData を要求してこのトークンが使えるかを検証
    b. レスポンスが空っぽのアカウントであることを検証
3. 既知の「prober_user」アカウントの認証トークンを要求
    a. /api/syncData を要求してこのトークンが使えるかを検証
    b. レスポンスが既知のゲーム状態と一致するかを検証
4. 新しく作成したアカウントを削除

既知のアカウントの正常な状態が古くならないように、ビルドプロセスの一部として「prober_user」の正常な状態を疑似乱数的に生成し、ビルドを本番環境にリリースするときに、既知のアカウントと合成クライアントを更新します。クライアントはその新しい正常な状態を使って、APIサーバーから受信した syncData 応答を検証します。このアカウントはリリース間で変更すべきではありません。

アプリ起動の合成クライアントの可用性の **SLI:** すべての HTTP のステータス コードが 200 OK かつすべての検証ステップが正常に完了する、合成クライアントのアプリ起動フローの割合。

アプリ起動の合成クライアントの可用性の **SLO:** 過去12時間において、合成アプリの起動の 99.8% 以上が成功すること。

ここでは低い可用性の目標を選択します。なぜならば、測定枠がはるかに短く、10 秒ごとに 1 回の合成アプリ起動のジャーニーしか完了しないからです。12 時間では 4320 回アプリが起動することになりますが、99.95% の可用性の目標を設定した場合、起動が 3 回以上失敗すると、その目標が達成されません。こうした失敗は、ユーザーがほとんど気付かないような、本筋ではないネットワーク接続の問題である可能性があります。

99.8% を目標にした場合のエラー バジェットは 12 時間で 8 回の起動に相当し、SLO が影響を受ける前の 12 時間で 90 秒近くのダウンタイムに相当します。ここでは、感度を下げるという明確なトレードオフを行っているため、本当に必要な場合にのみ障害対応がトリガーされます。12 時間でエラー バジェットの約 7 %を消費した場合にのみアラートを発生させることを選択しているので、このようなことが定常的に発生することは許されませんが、他の SLO でその側面に対応します。

アプリ起動の *API* 複合レイテンシ

レイテンシについては、ユーザーが最も気にすることは、アプリの起動時間全体、つまり「ホーム画面のアイコンを押した」時から「自分の陣地が表示される」までの時間です。このレイテンシはユーザー体験の理想的な尺度ですが、ユーザーのデバイスの速度、同期する必要のあるコンテンツの量、接続速度などによって大きく変動します。 他のクライアントの埋め込みと同様に、意味のある、実用的な「遅すぎる」信号をこのノイズから分離するには、複雑なデータ処理が必要になる場合があります。

代わりに、妥協案として、ロード　バランサで測定されたすべてのアプリ起動関連の API 呼び出しのレイテンシを捕捉します。理論的には、これはアプリの起動時間の　SLI　ほど理想的なものではありませんが、実際には、この測定方法は一貫性と信頼性があるので、最初の　SLI　としては適切です。この代替案の　SLI　が十分に代わるものではないことが判明した場合、つまりユーザーが起動時間に不満を持っているにもかかわらずそれが　SLO　に反映されていない場合は、いつでも SLI の実装を再検討し、変動性の高いアプリ起動時間のデータから信頼できる信号を引き出すためのさらなる努力をはかることができます。

ロード バランサでレイテンシを追跡する場合に、 API の呼び出しはレイテンシの特徴とリクエストレートが比較的類似していることが前提になっています。現実的には、getAuthToken は syncData よりも速いことが考えられ、しかも　createAccount　よりも速いのですが、その差は十数ミリ秒程度の違いです。我々が懸念するのは秒単位のレイテンシです。

ここで問題となるのは、 syncData と他の 2 つのハンドラーのリクエストレートの差です。複合 SLI/SLO では、それらがほぼ同等の重みになることを期待します。この目的を達成する最も簡単な方法は、API の呼び出しごとの レイテンシの SLI と SLO を作成し、「悪い時間」のアプローチを利用して、SLI ではなく SLO として集約します。これにより SLI ごとにレイテンシのしきい値が設定でき、仮定の正確性を懸念する必要がなくなります。

3つの SLI すべてがそれぞれの SLO の目標値を上回っている、連続した時間　(分)　を「良い」時間とし、いずれかが目標を下回っている連続する時間 (分) は「悪い」時間とします。そして、「悪い」時間を 1 か月に何分まで許容するかという目標値を設定します。

LB createAccount レイテンシの **SLI:** ロードバランサで測定された、完全な応答が 2000 ミリ秒未満で送信される /api/createAccount リクエストの割合。

LB getAuthToken レイテンシの **SLI:** ロードバランサで測定された、完全な応答が 400 ミリ秒未満で送信される /api/getAuthToken リクエストの割合。

LB syncData レイテンシの **SLI:** ロードバランサで測定された、完全な応答が 2000 ミリ秒未満で送信される /api/syncDataリクエストの割合。

App 起動 **API** 複合レイテンシの **SLI:** 以下を満たす時間 (分) の割合

- /api/createAccount リクエストの 99% 以上が 2000 ミリ秒未満かつ
- /api/getAuthToken リクエストの 99% 以上が 400 ミリ秒未満かつ
- /api/syncDataリクエストの 99 % 以上が 2000 ミリ秒未満

App 起動 **API** 複合レイテンシの **SLO:** 過去 28 日間における、API の リクエストが、それぞれの SLO を 99.9% 以上の時間で達成すること。

ハンドブックの PDF の作成

ハンドブックの PDF を作成することは易しいと思っていませんか？ ほとんどの場合はそうです。問題なのは、A4 サイズのハンドブックを A5 サイズやレターサイズ、ハーフレターサイズでつくるときです。Google Docs のページサイズを変更する場合、最初からレイアウトをやり直さなくてはなりませんが、GhostScript[6] を使ってGoogle Doc で生成した PDF のサイズ変更をすることができます。

克服すべき問題は、 PDF をダウンロードするときにフォントが埋め込まれていないので、ドキュメントを小さいサイズで再度正しくレンダリングすることができない点です。ここでは、Google Fonts Library[7] で入手可能な Raleway と Robotoを使います (訳注: 日本語の場合はこの限りではありません)。 同じディレクトリに PDF とフォントの両方を配置すると、下記の Magic Incantation でフォントが正しく埋め込まれた変更後のハンドブックのドキュメントのサイズが変更されたバージョンで生成されます。

```
gs \
  -dSAFER \
  -sDEVICE=pdfwrite \
  -dPDFSETTINGS=/prepress \
  -dColorConversionStrategy=/LeaveColorUnchanged \
  -dSubsetFonts=true \
  -dEmbedAllFonts=true \
  -sFONTPATH=. \
  -dFIXEDMEDIA \
  -dPDFFitPage \
  -sPAPERSIZE="a5" \
  -o "OUTPUT.pdf" \
  -f "INPUT.pdf"
```

PAPERSIZE パラメータを “a5” ではなく “letter” または “halfletter” にするとそのサイズが生成されます。

4 の倍数の中綴じ印刷でない小冊子の印刷に不満をもらす印刷ショップもあります。その場合には、上記のコマンドに -c showpage を追加して出力して PDF に単一のパディング ページを追加できます。パディングページが複数ある場合はこの方法を複数回繰り返します。

[6] https://ghostscript.com/download/gsdnld.html or via your friendly local package manager.
[7] https://fonts.google.com/specimen/Raleway and https://fonts.google.com/specimen/Roboto