

JAPAN | 2024

aws SUMMIT

# Nintendo Switch™ 向け プッシュ通知システムのリプレイス事例

**林 愛美**

ニンテンドーシステムズ株式会社

システム開発部

**坂東 聖博**

ニンテンドーシステムズ株式会社

システム開発部

# Nintendo Switch™ 向け プッシュ通知システムの リプレイス事例

ニンテンドーシステムズ株式会社  
システム開発部

林 愛美

坂東 聖博

# 自己紹介

# 林 愛美

- ▶ 任天堂株式会社 キャリア入社3年目
- ▶ 所属
  - ▶ ニンテンドーシステムズ システム開発部
- ▶ これまで
  - ▶ Nintendo Switch 向けネットワークシステムの開発/運用を担当
  - ▶ プッシュ通知システムの SRE 主担当



# 坂東 聖博

- ▶ 任天堂株式会社 キャリア入社5年目
- ▶ 所属
  - ▶ ニンテンドーシステムズ システム開発部
- ▶ これまで
  - ▶ Nintendo Switch 向けネットワークシステムの開発/運用を担当
  - ▶ 社内 GitHub Enterprise Server, GitHub Actions self-hosted Runner の運用を担当
  - ▶ プッシュ通知システムの SRE として、構築・検証を担当



# 今日の内容

- ▶ サービス紹介
- ▶ なぜリプレイスするのか
- ▶ システム構成
- ▶ Amazon ECS on Fargate で大規模常時接続システムを実現する
- ▶ リプレイスを終えて
- ▶ まとめと展望

# サービス紹介

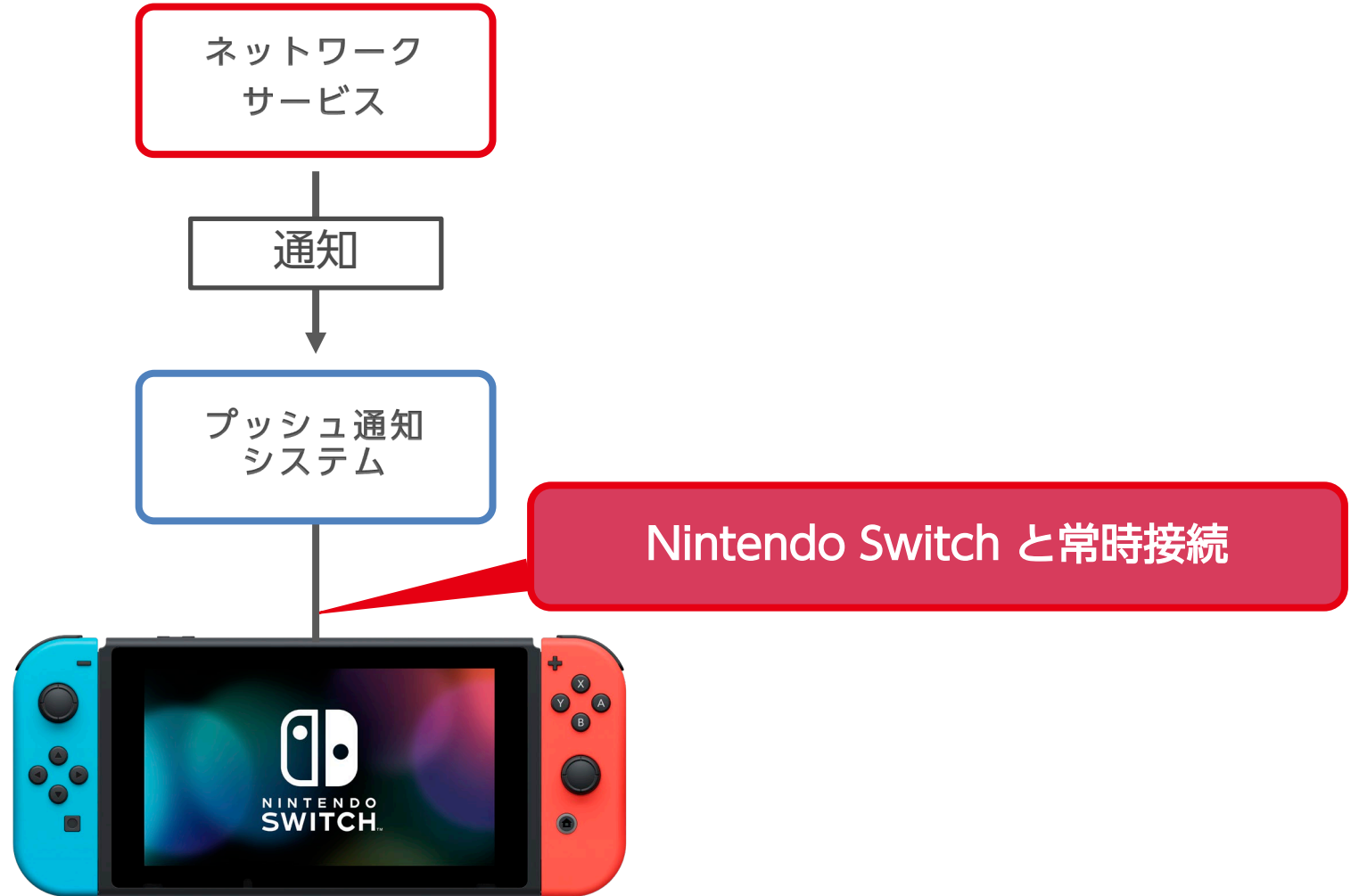


# Nintendo Switch

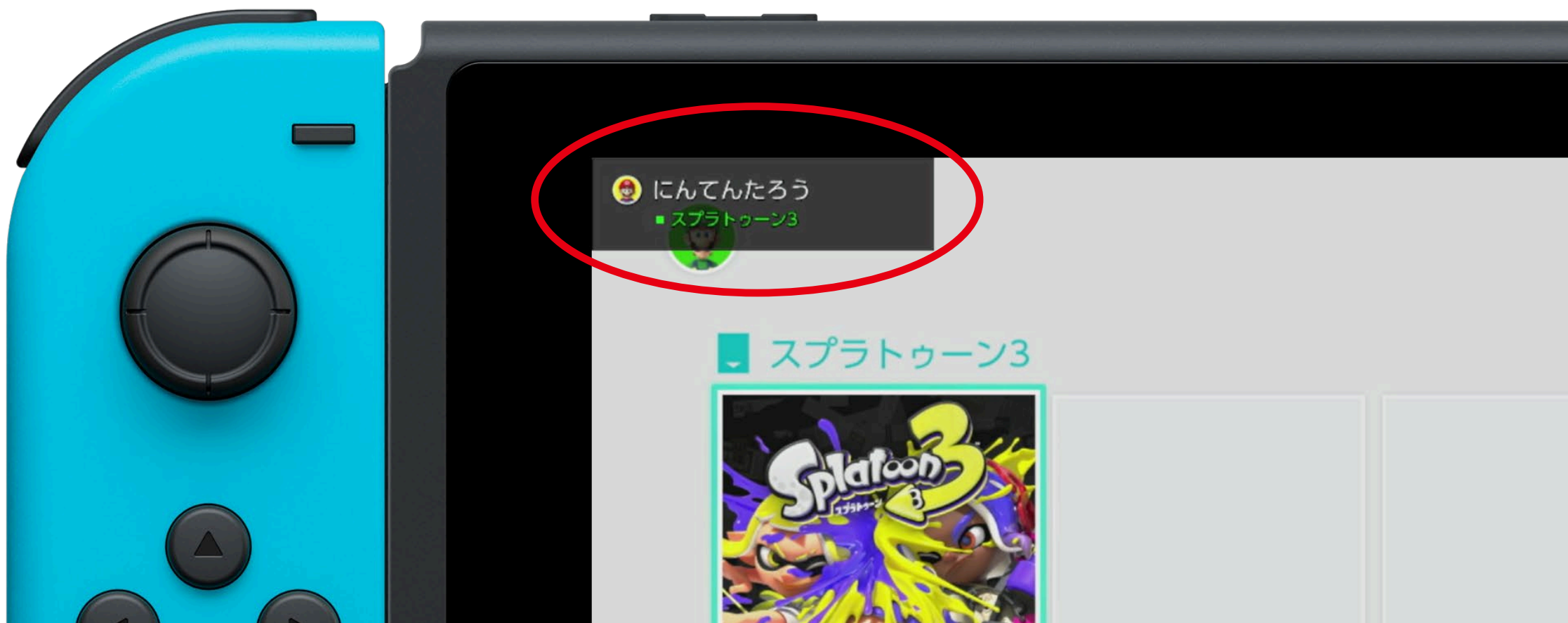
全世界累計販売台数  
1億4,132万台 (2024年03月末時点)



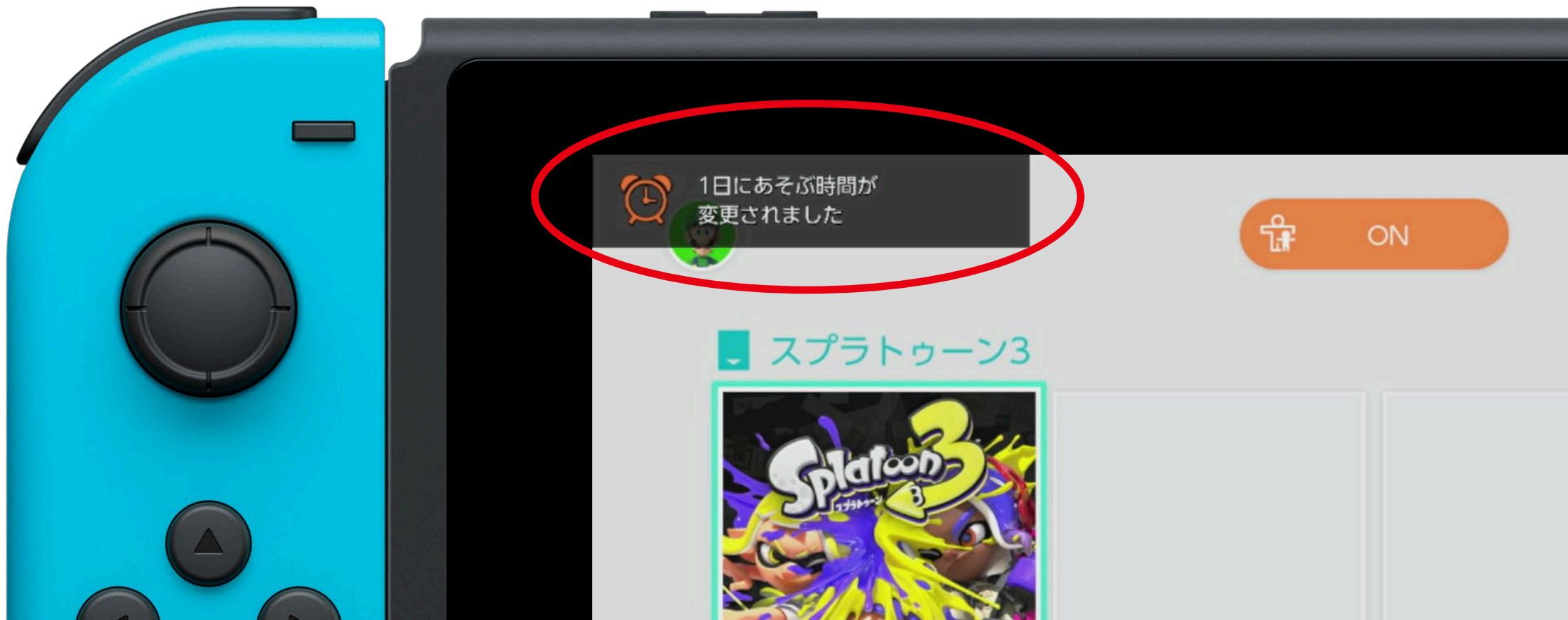
# 任意のタイミングで通知を送信



# 利用例: フレンドのオンライン通知



# 利用例: Nintendo みまもり Switch での設定変更通知

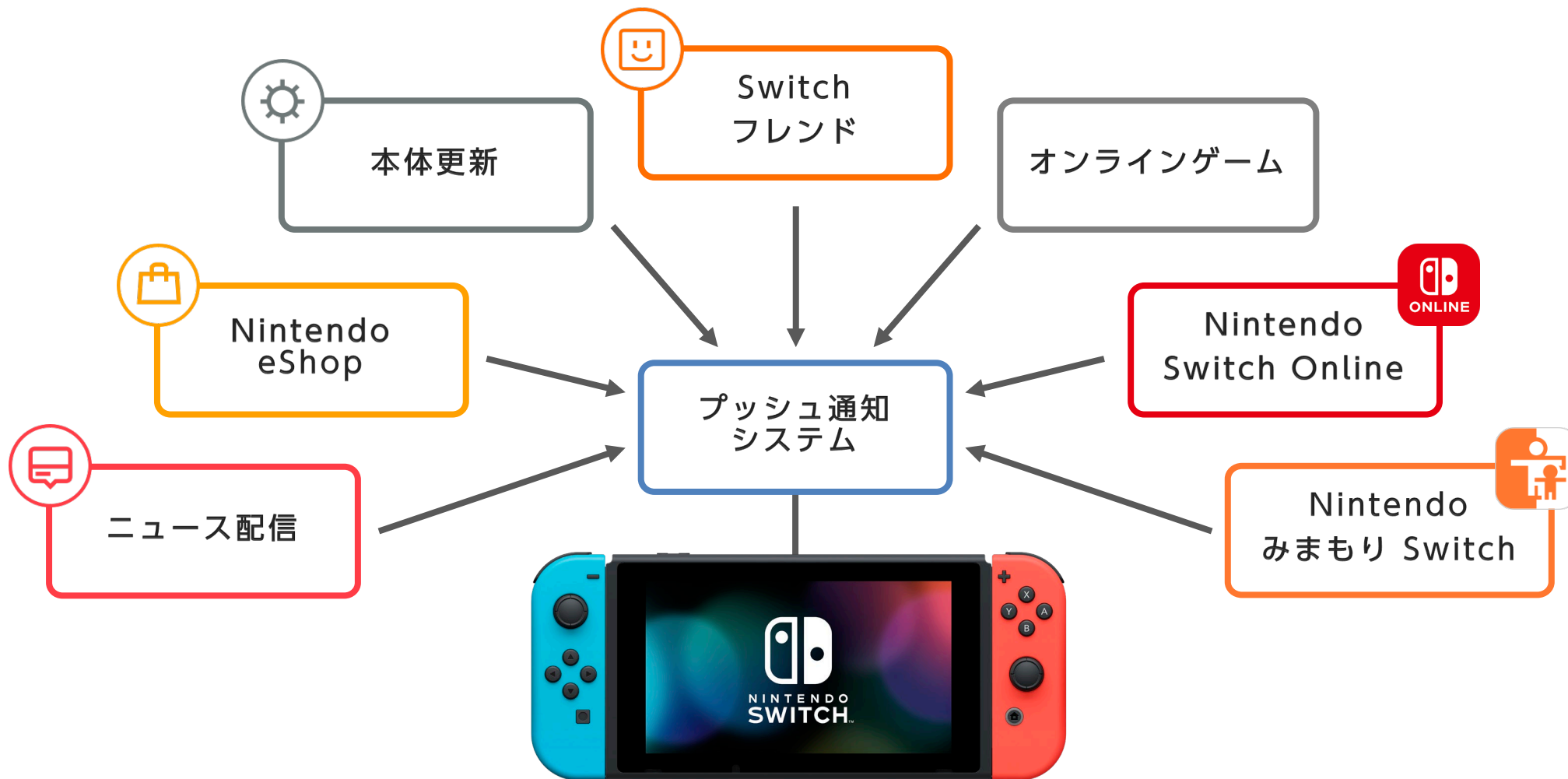


# 利用例: スマートフォンや PC でゲームを 購入した時の、ダウンロード開始指示

スマホで購入、  
そのまま自動で  
ダウンロード

家に帰ったらすぐに遊べる!

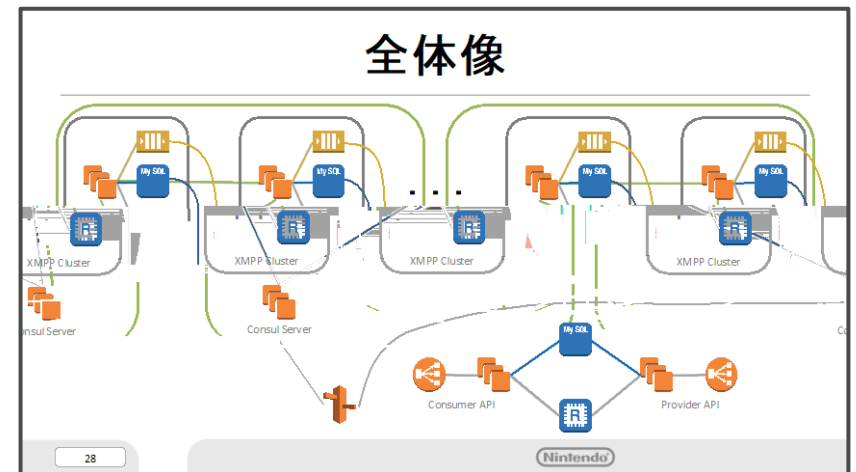
# プッシュ通知システム利用サービス





# 既存のプッシュ通知システム

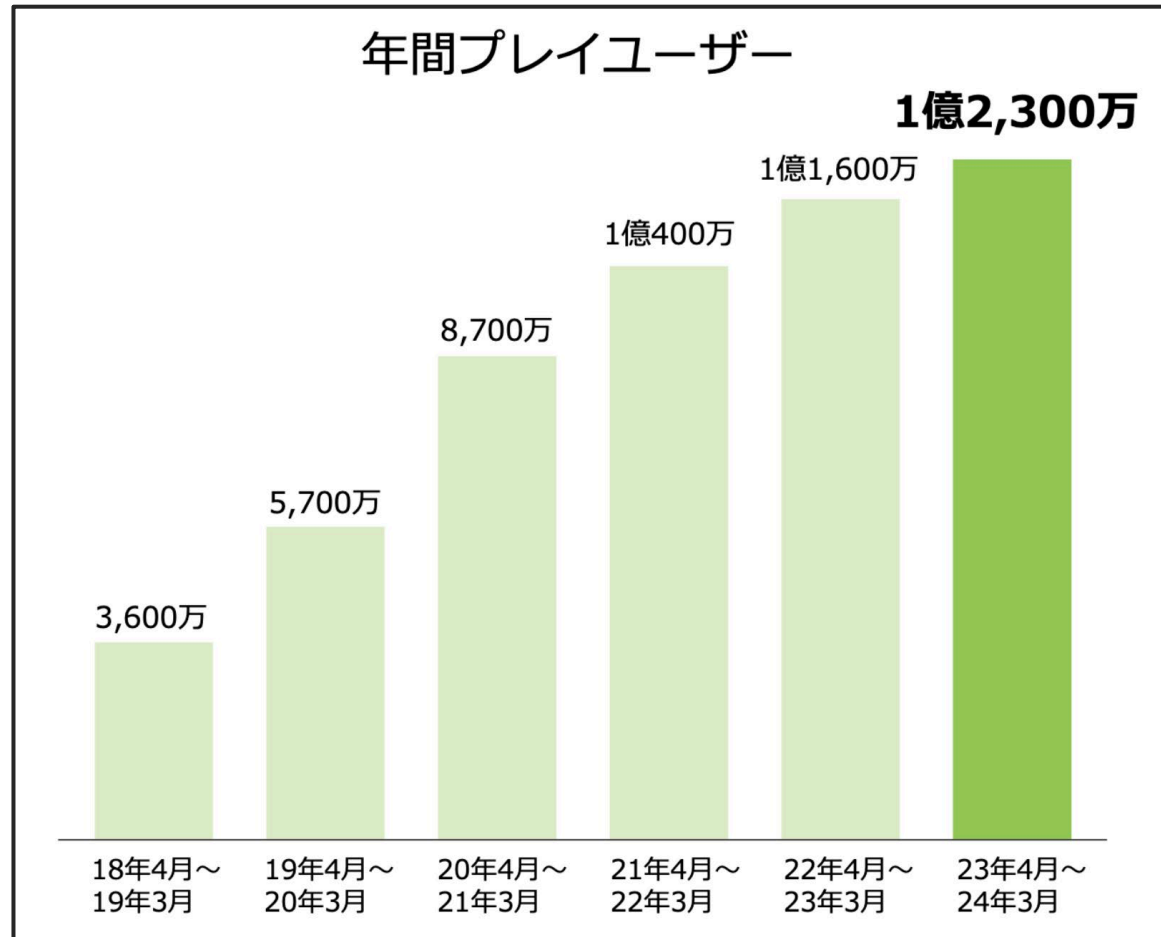
- ▶ 2017 年の Nintendo Switch 発売  
同時にサービスイン
- ▶ システム構成
  - ▶ 常時接続には ejabberd を利用し、  
複数のクラスタで構成される
  - ▶ Amazon Elastic Compute Cloud  
(Amazon EC2) や Amazon Relational  
Database Service (Amazon RDS) など  
実績のある安定したサービスを採用
- ▶ AWS Summit Tokyo 2018で公表 ※



なぜリプレイスするのか



# 年間プレイユーザーは増え続けている



2024年3月期第3四半期 決算説明資料 より

# 長期運用に向けて解決したい課題

- ▶ 柔軟な機能追加をしたい
  - ▶ OSS の改造から、独自アプリケーション開発に
- ▶ 開発者の流動性を確保したい
  - ▶ 他システムで採用実績のある Go で開発
- ▶ 開発/運用の効率を高めたい
  - ▶ クラスタ構成をやめ、シンプルなアーキテクチャーに
- ▶ 運用工数を削減したい
  - ▶ サーバーレスサービスを積極的に採用

# 長期運用に向けて解決したい課題

- ▶ 柔軟な機能追加をしたい
  - ▶ OSS の改造から、独自アプリケーション開発に
- ▶ 開発者の流動性を確保したい
  - ▶ 他システムで採用実績のある Go で開発
- ▶ 開発/運用の効率を高めたい
  - ▶ クラスタ構成をやめ、シンプルなアーキテクチャに
- ▶ 運用工数を削減したい
  - ▶ サーバーレスサービスを積極的に採用

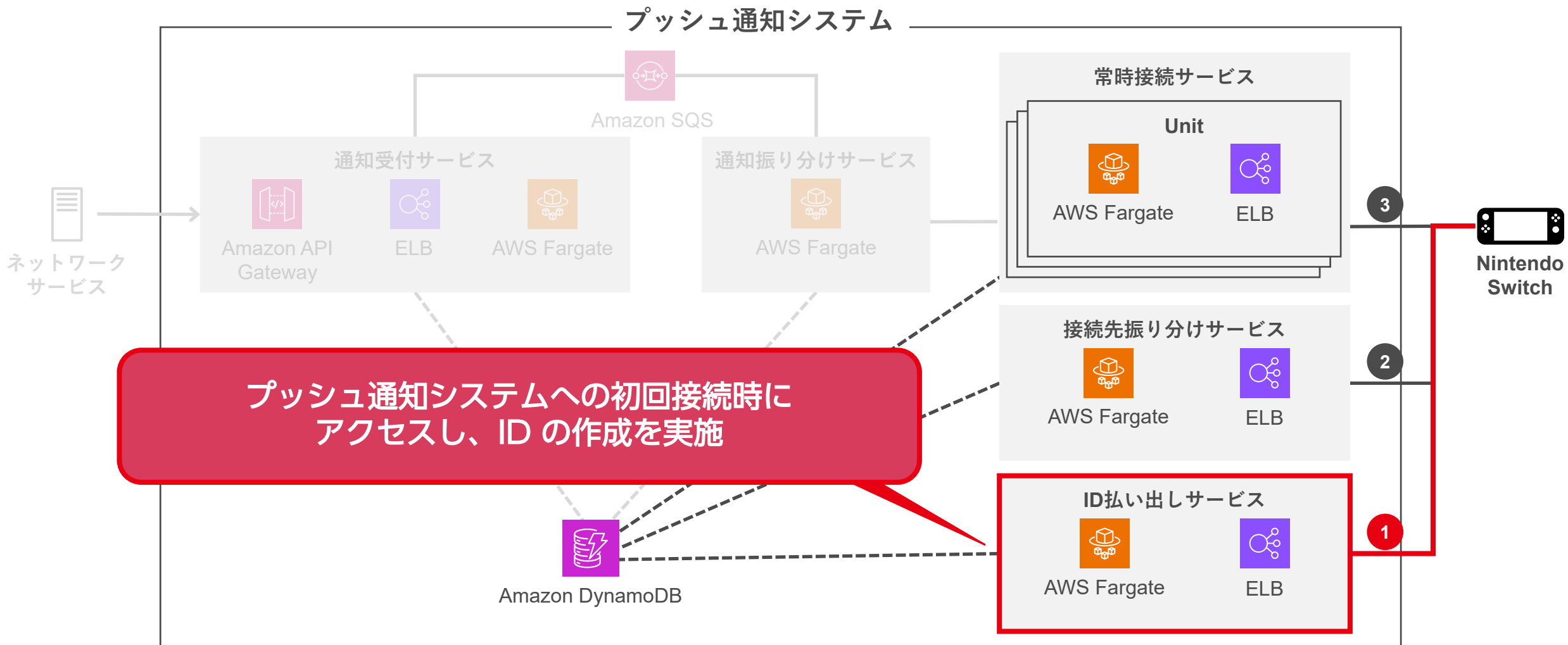
システム構成をご紹介します

Amazon ECS on AWS Fargate  
(Amazon ECS on Fargate) で大規模常時接続を  
実現するための工夫をご紹介します

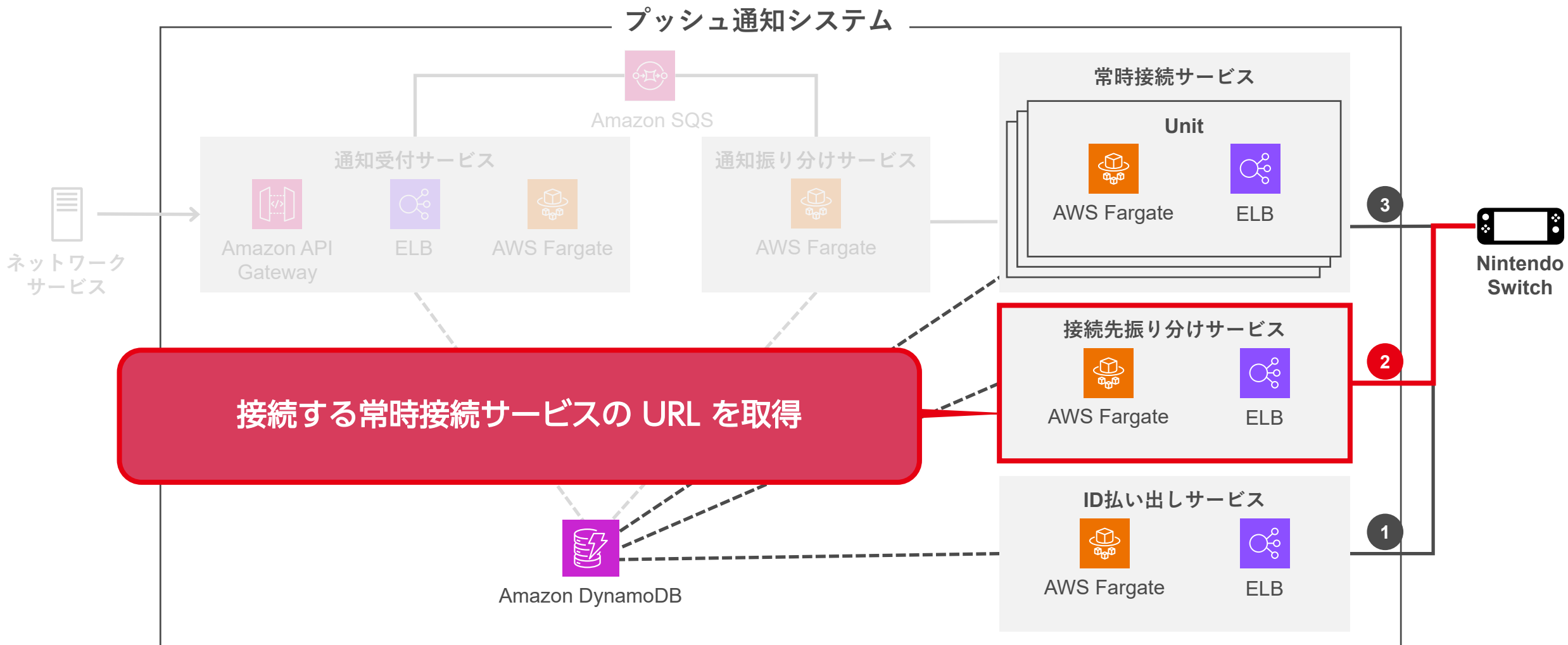
# システム構成



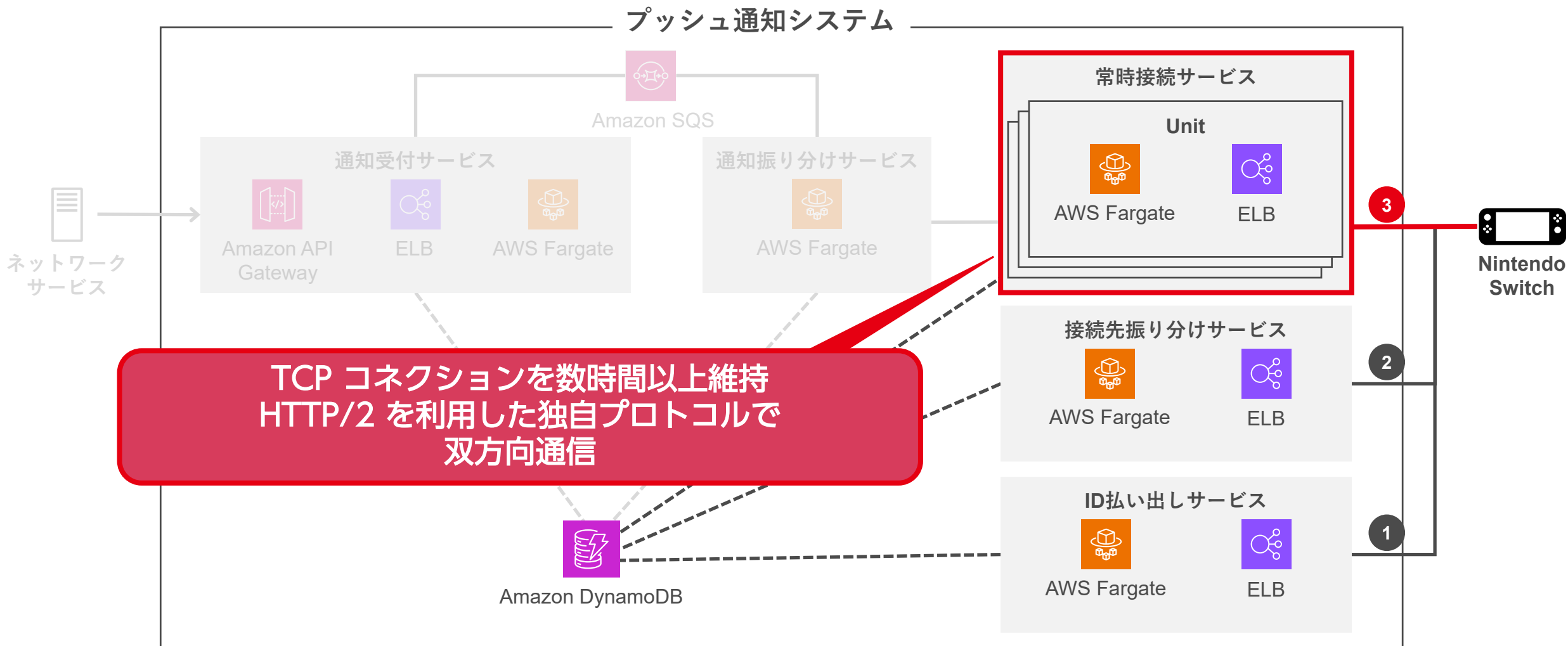
# ユースケース: ログイン



# ユースケース: ログイン

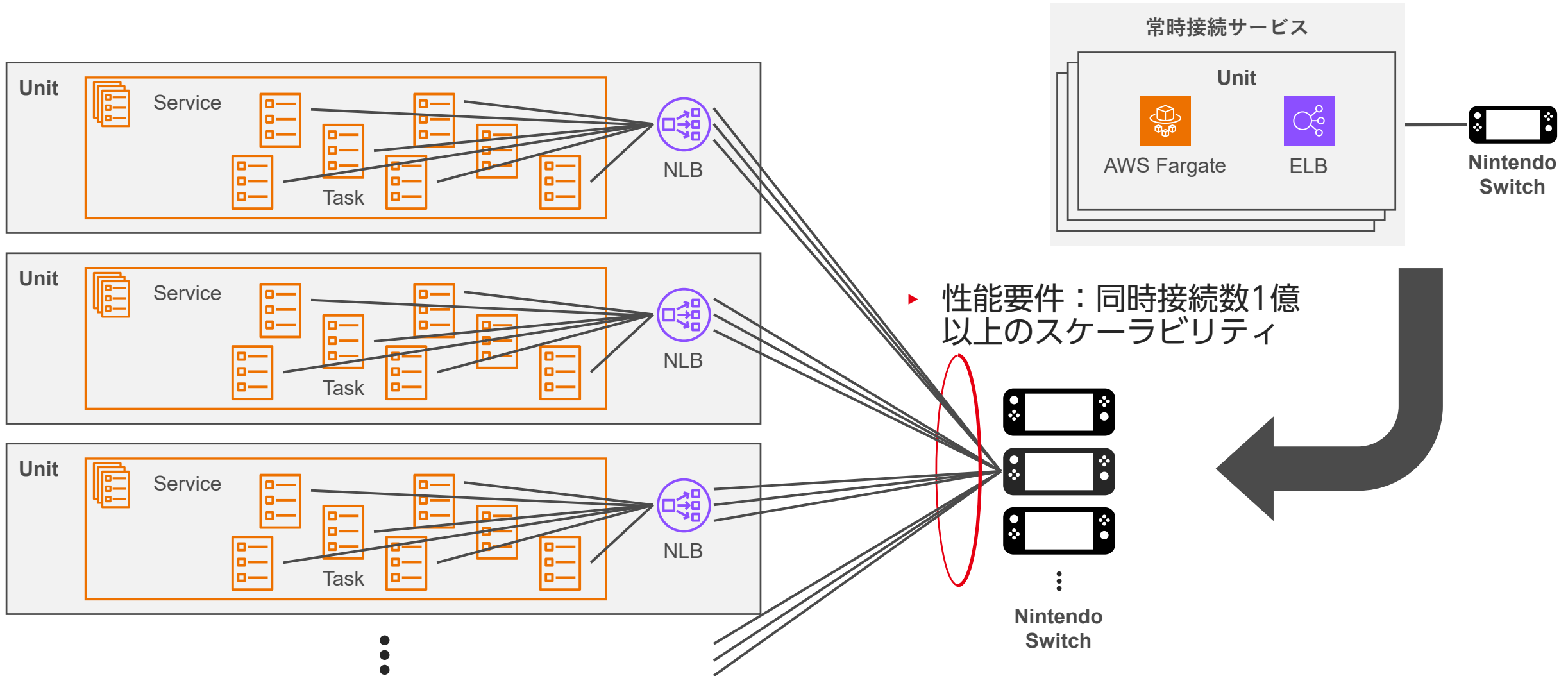


# ユースケース: ログイン

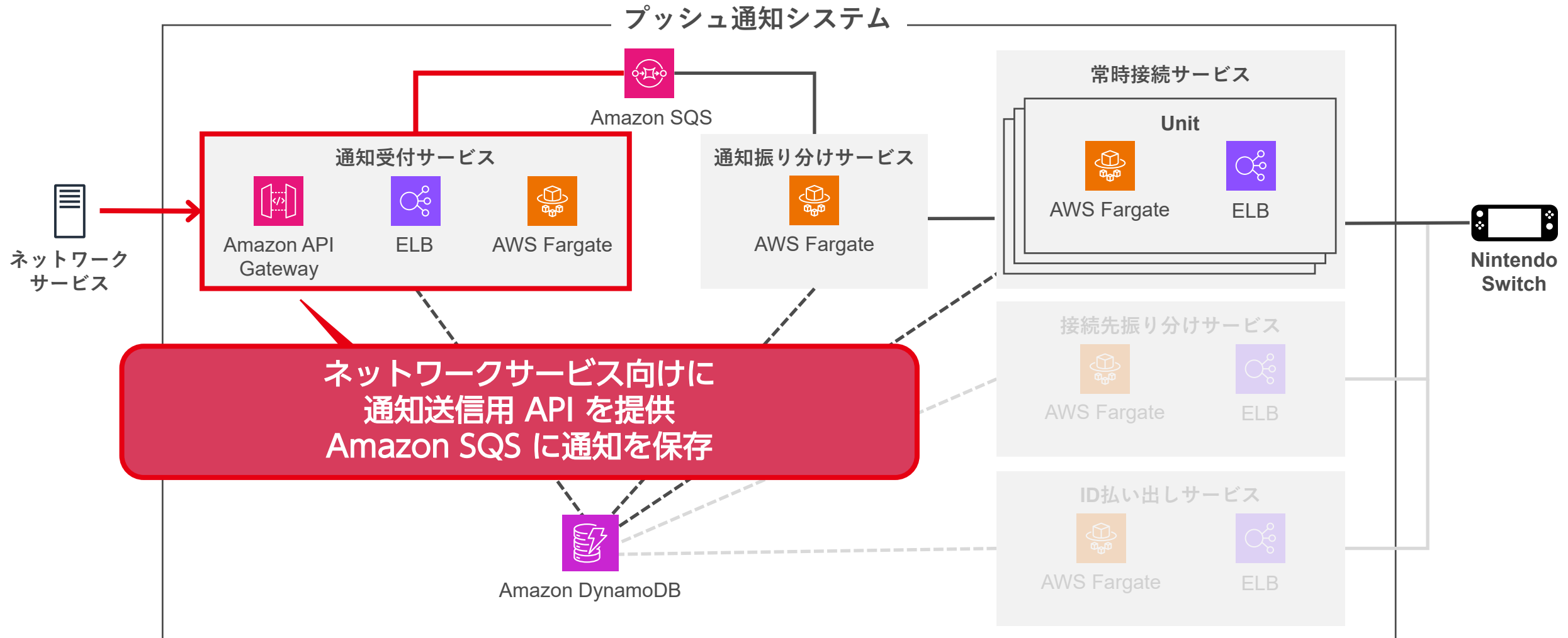




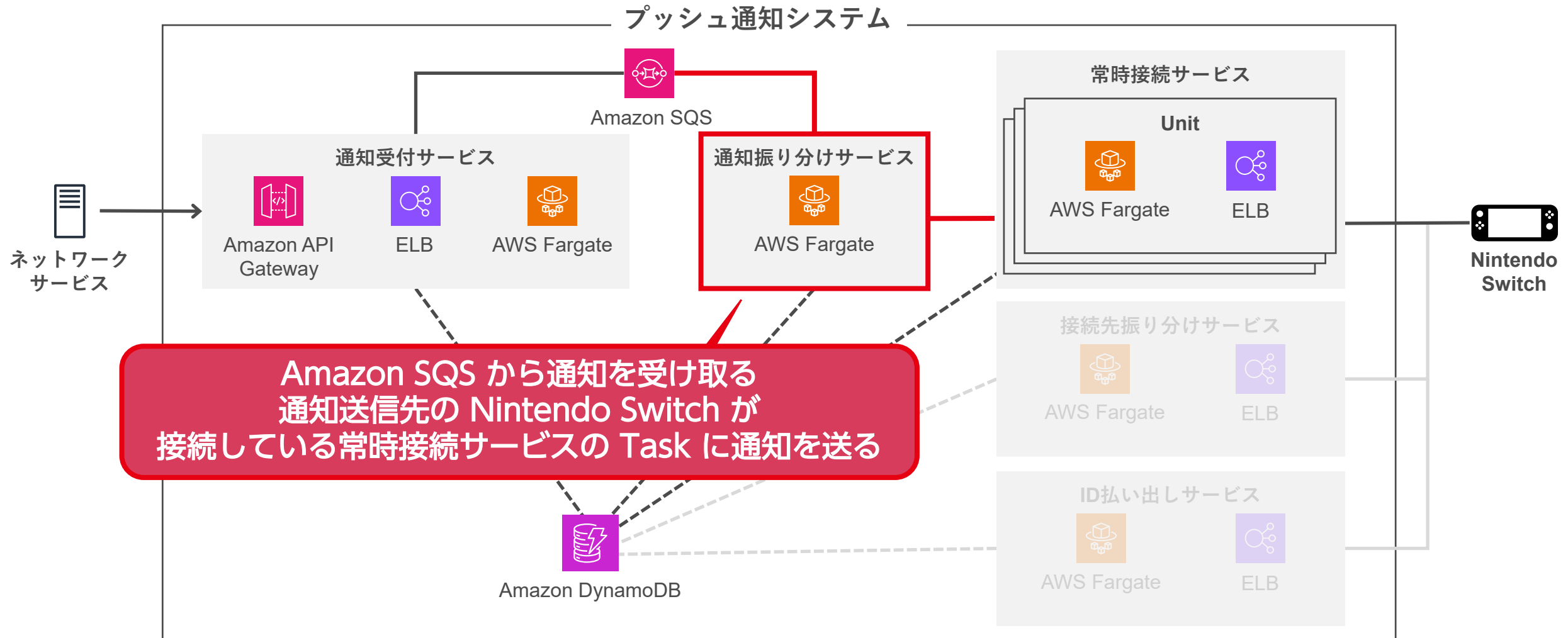
# 常時接続サービス



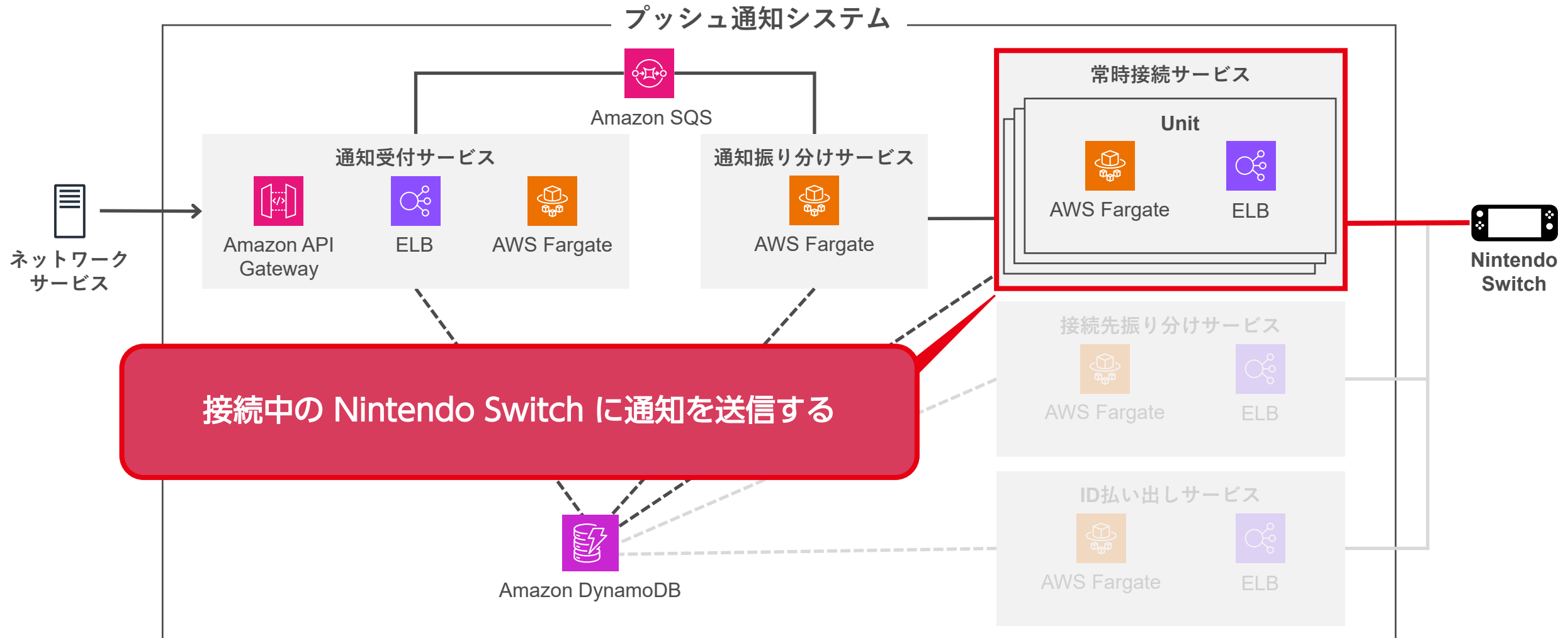
# ユースケース: 通知送信



# ユースケース: 通知送信

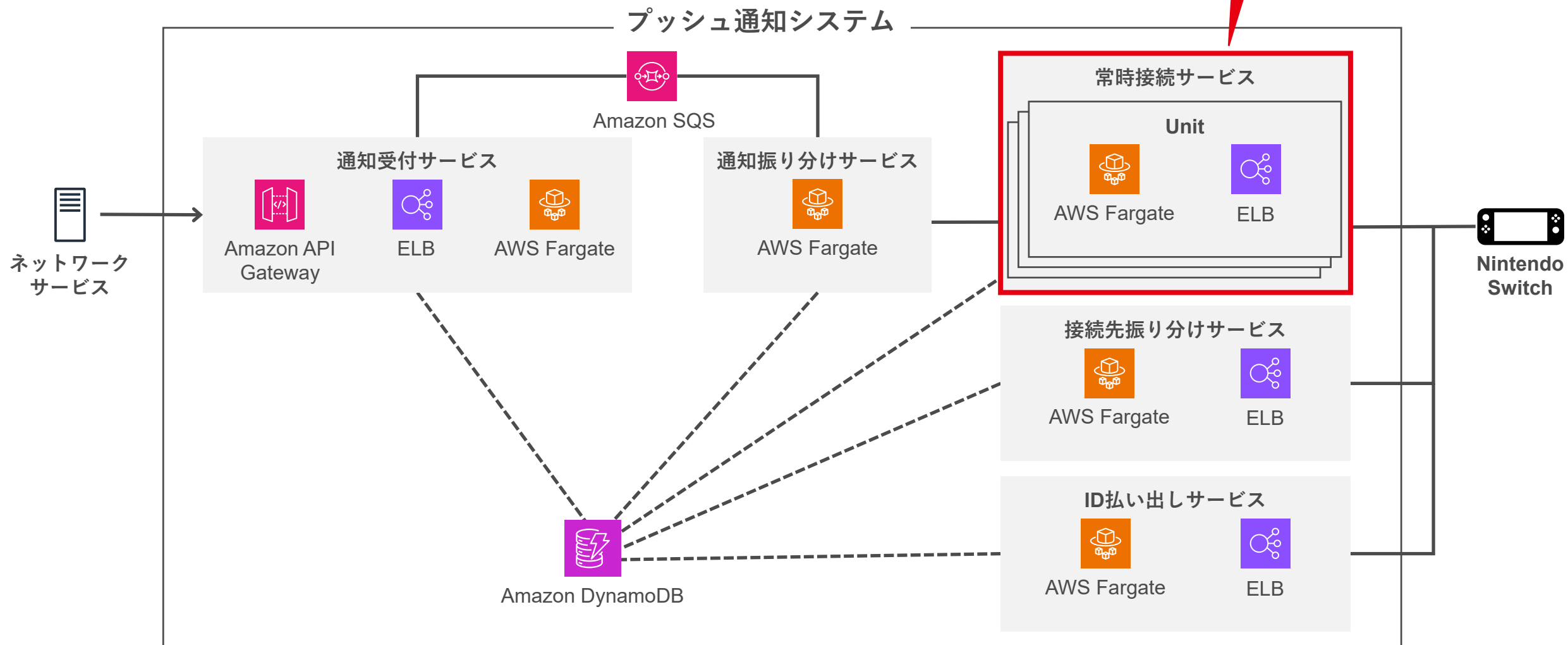


# ユースケース: 通知送信



# システム構成図

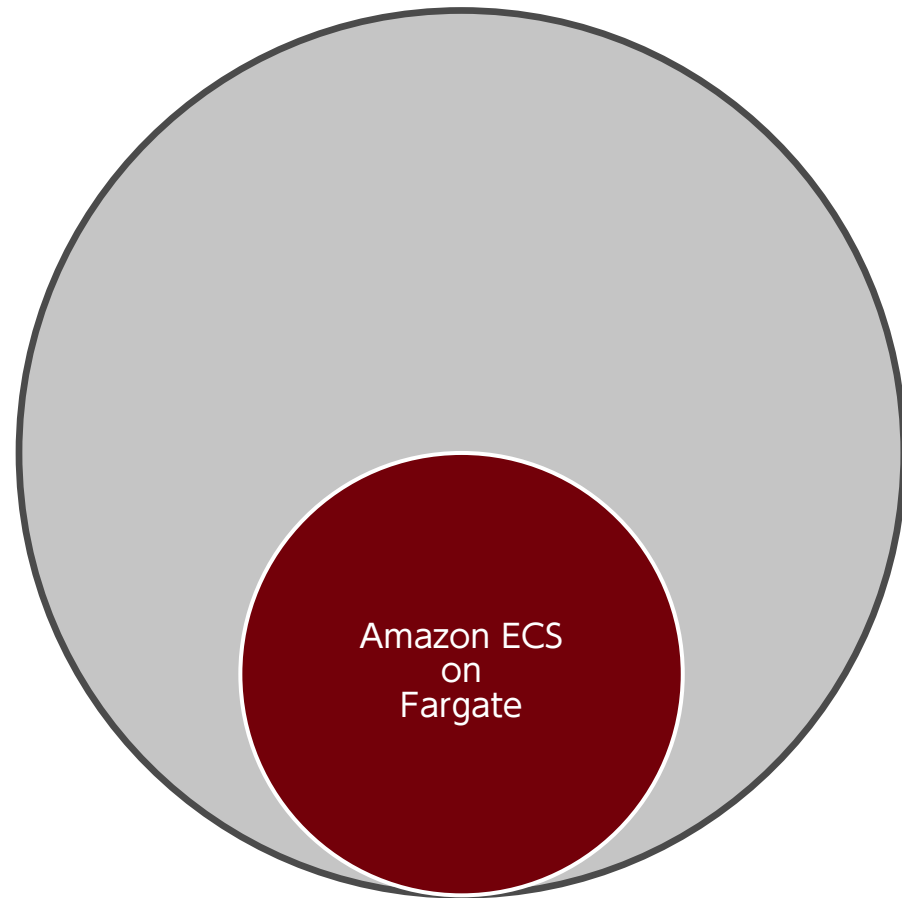
Amazon ECS on Fargate で大規模常時接続を実現するための工夫をご紹介



# Amazon ECS on Fargate で 大規模常時接続システムを 実現する

# 常時接続と Amazon ECS on Fargate

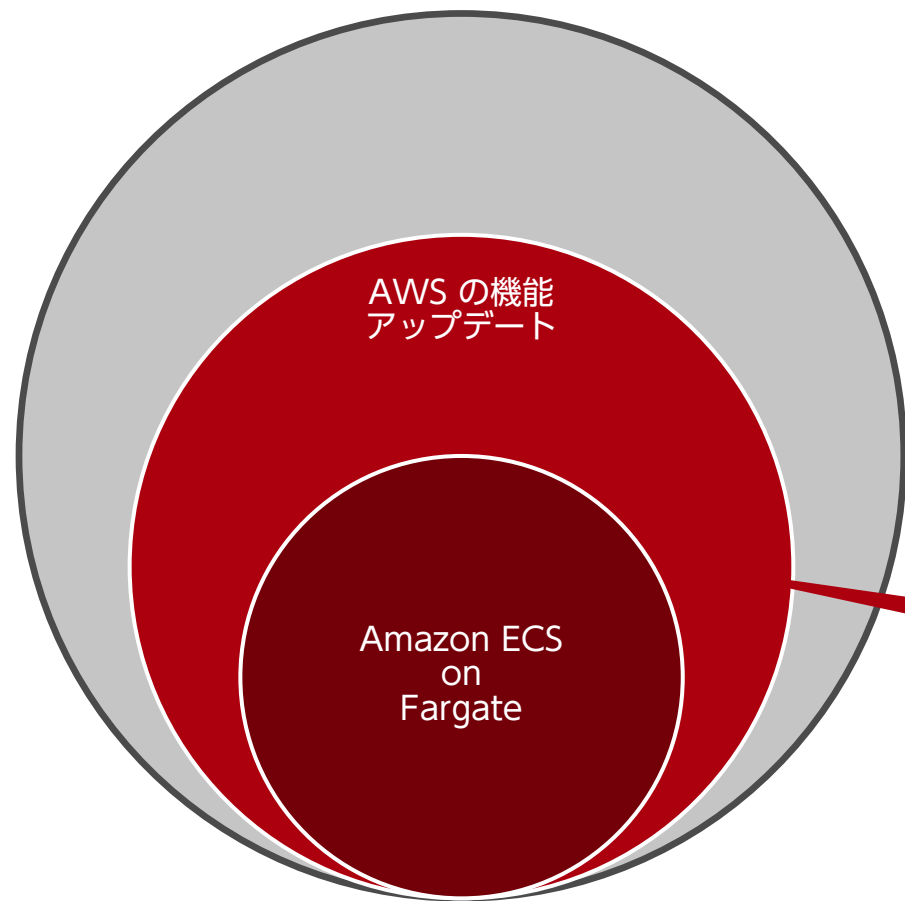
大規模常時接続に必要な要件



- ▶ 旧システムのリリース当初は、Amazon ECS on Fargate の機能では要件を満たすことができなかった

# 常時接続と Amazon ECS on Fargate

大規模常時接続に必要な要件



- ▶ 旧システムのリリース当初は、Amazon ECS on Fargate の機能では要件を満たすことができなかった
- ▶ 機能アップデートにより、要件の一部が満たされた

常時接続システムにとって  
重要だったアップデートの例をご紹介します



# TCP 接続を維持するためのチューニング

- ▶ スリープ中の本体とも接続を維持している
- ▶ バッテリーの消費を防ぎつつ、きちんと通知を届けるために
  - ▶ TCP keepalive を利用して接続を維持している
  - ▶ TCP 再送関連の挙動を意識する必要がある
- ▶ カーネルパラメータを変更する必要がある

# TCP 接続を維持するためのチューニング

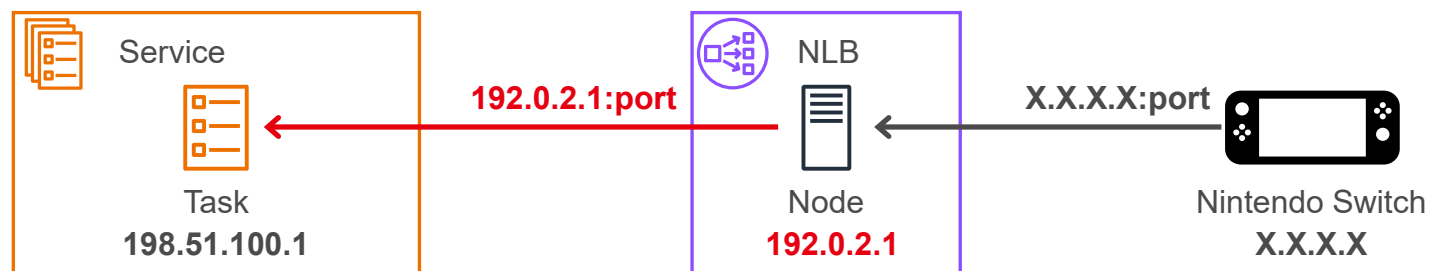
- ▶ 2023/08 に、Amazon ECS on Fargate で一部のカーネルパラメータの変更ができるようになった
  - ▶ 要望していた net.\* 系のパラメータは対象に含まれた
- ▶ Amazon ECS on Amazon EC2 で検討していた常時接続サーバーを、Amazon ECS on Fargate に変更することができた

# 少ないリソースで、たくさんの接続を担う

- ▶ 1 Task あたり数十万同時接続を達成したい
- ▶ NLB の クライアント IP の保存 を有効にする必要がある
  - ▶ 2021年に有効/無効を切り替えられるようになった

# 少ないリソースで、たくさんの接続を担う

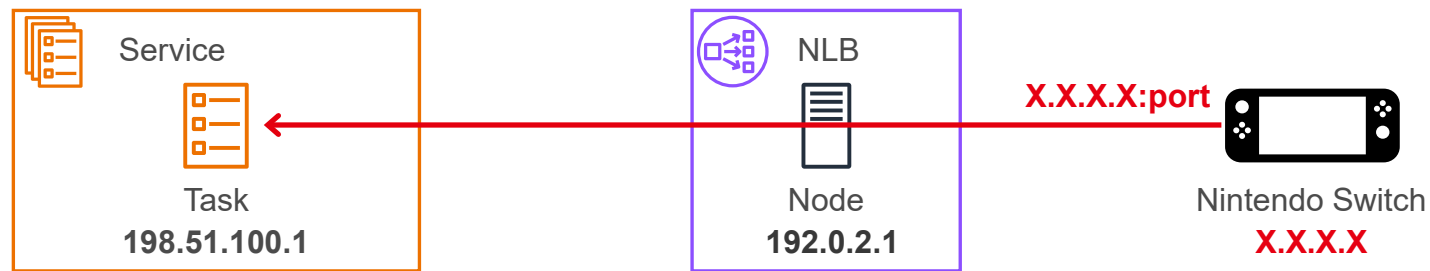
## ▶ クライアント IP の保存が**無効**だと



Source: 192.0.2.1:port  
Destination: 198:51.100.1:3000

- ▶ 55,000 接続程度で NLB でポートアロケーションエラーが発生する

## ▶ クライアント IP の保存を**有効**にすると



Source: X.X.X.X:port  
Destination: 198:51.100.1:3000

- ▶ 同時接続数が NLB のポート数がボトルネックにならなくなる

# 少ないリソースで、たくさんの接続を担う

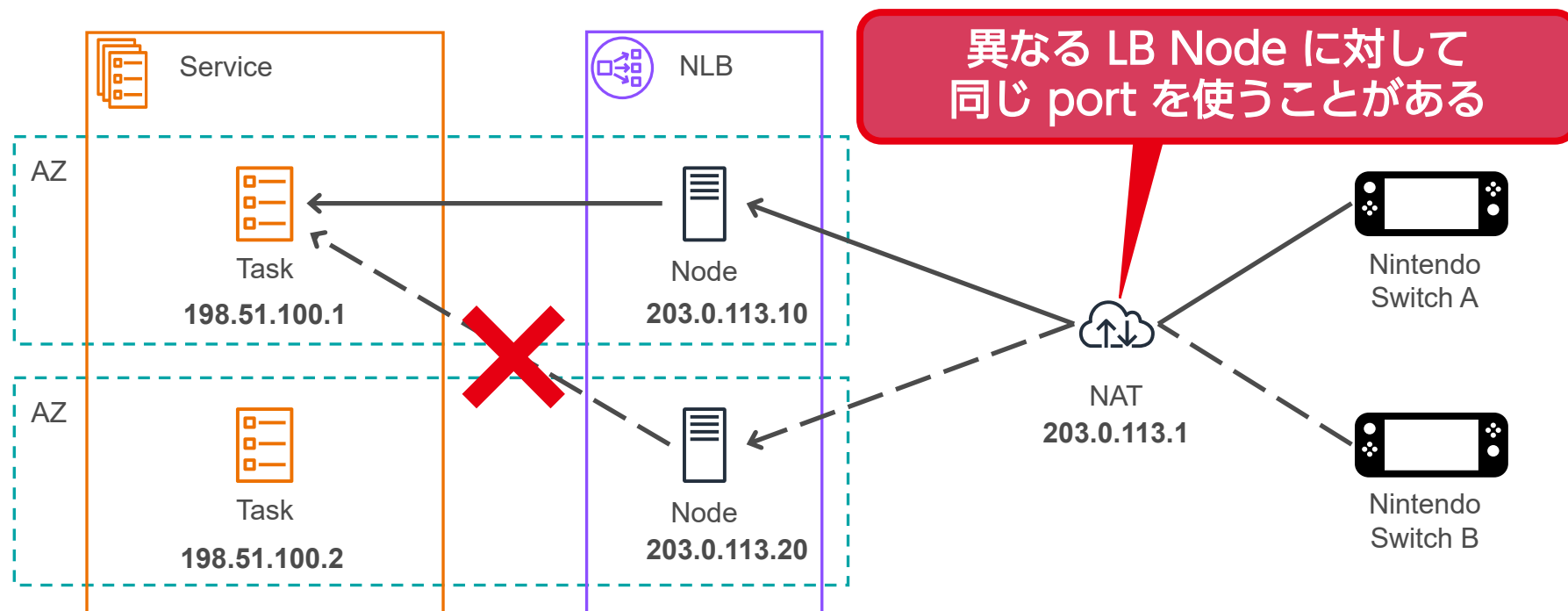
- ▶ クライアント IP の保存 を有効にすると、ポートの共有問題が発生する

Source: 203.0.113.1:12345  
Destination: 198.51.100.1:3000



Source: 203.0.113.1:12345  
Destination: 198.51.100.1:3000

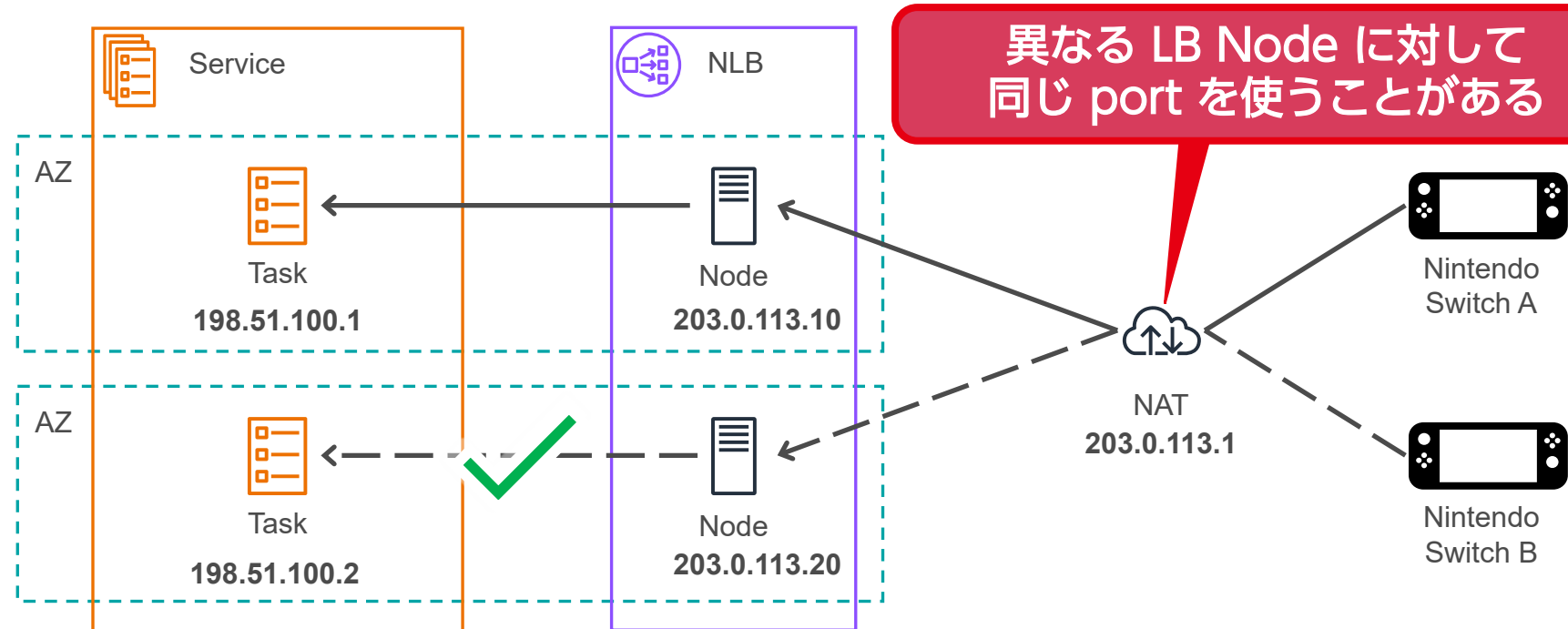
- ▶ 同じデバイスからの通信に見えてしまう



# 少ないリソースで、たくさんの接続を担う

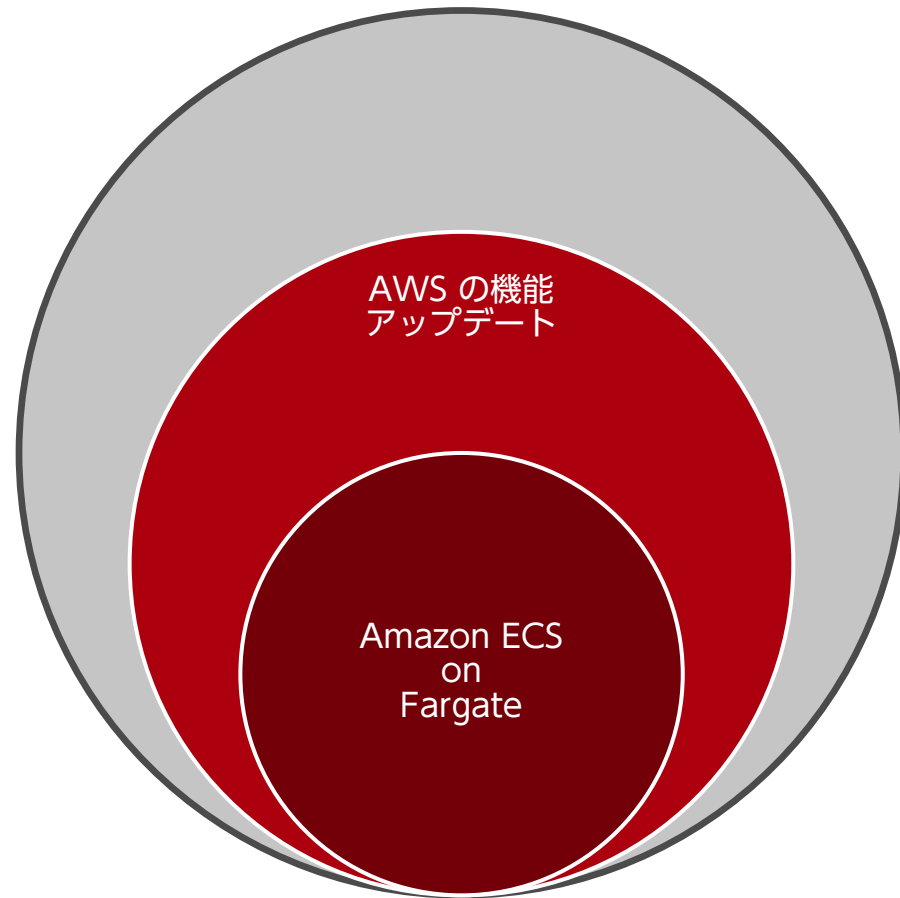
- ▶ クライアント IP の保存 を有効にすると、ポートの共有問題が発生する
- ▶ クロスゾーン負荷分散 を無効にすると回避可能

- ▶ AZ と跨いだ通信をしなければ、異なる Node への接続は異なる Task にフォワードされる



# 常時接続と Amazon ECS on Fargate

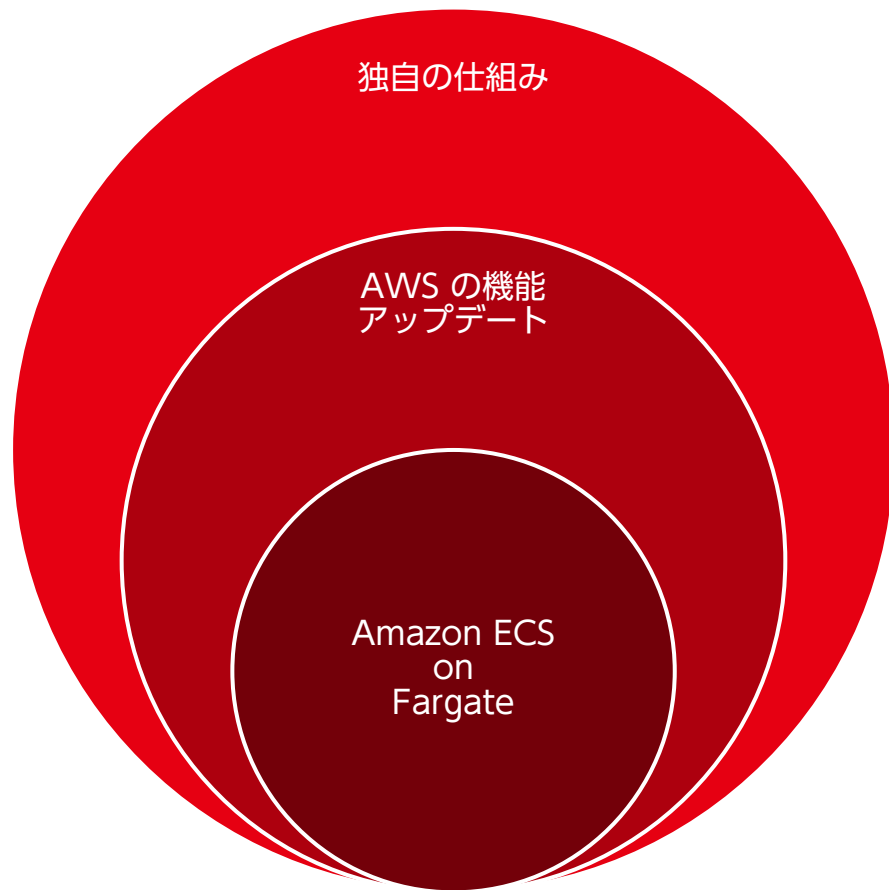
大規模常時接続に必要な要件



- ▶ 旧システムのリリース当初は、Amazon ECS on Fargate の機能では要件を満たすことができなかった
- ▶ 機能アップデートにより、要件の一部が満たされた

# 常時接続と Amazon ECS on Fargate

## 大規模常時接続に必要な要件



- ▶ 旧システムのリリース当初は、Amazon ECS on Fargate の機能では要件を満たすことができなかった
- ▶ 機能アップデートにより、要件の一部が満たされた
- ▶ 補完する独自の仕組みを実装することにより、大規模常時接続システムを Amazon ECS on Fargate で実現することができた

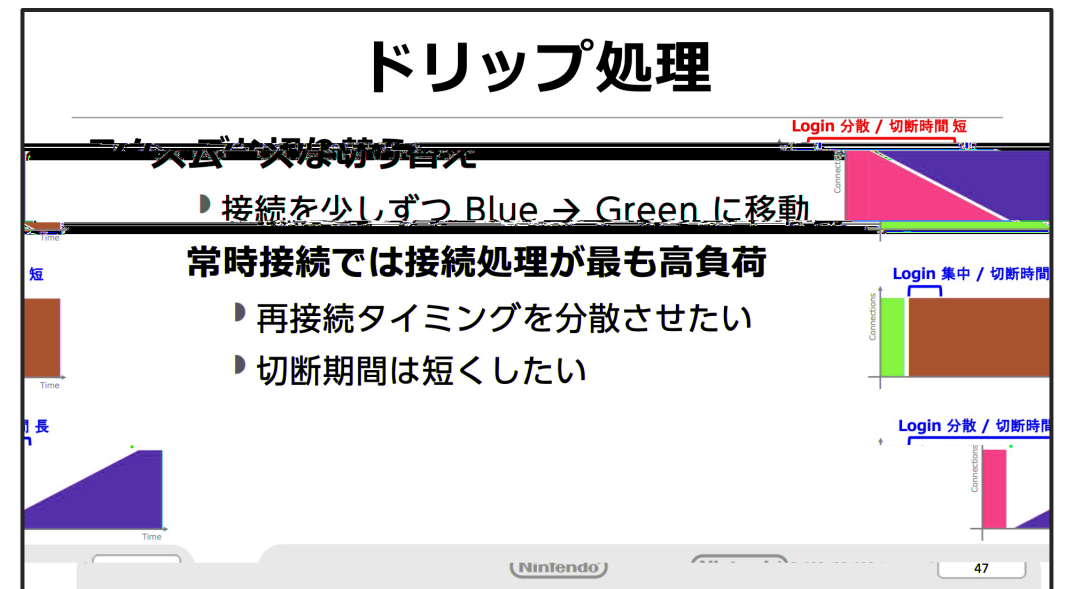


# デプロイツールの目的は負荷分散

- ▶ 最も負荷の高い処理は常時接続開始処理
- ▶ デプロイ時は全ての Task が入れ替わり、再接続処理が行われるので、特に気をつける必要がある
  - ▶ 理想的には全ての Task に対して均等に、かつ接続処理が重なり高負荷にならないように分散させたい

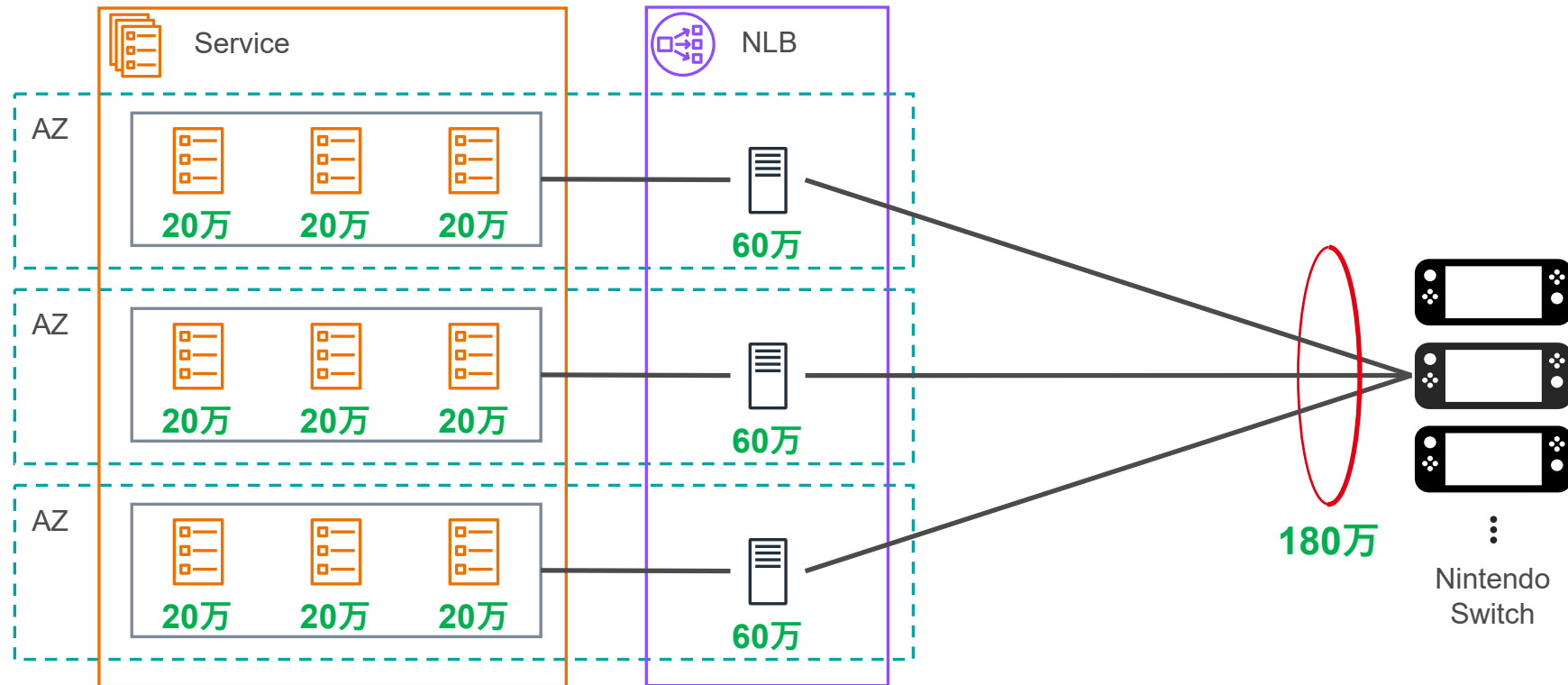
# 接続のタイミングを分散させる

- ▶ リプレイス前のシステムの方法を踏襲
- ▶ 常時接続アプリは一定の速度で再接続要求を出すことができる



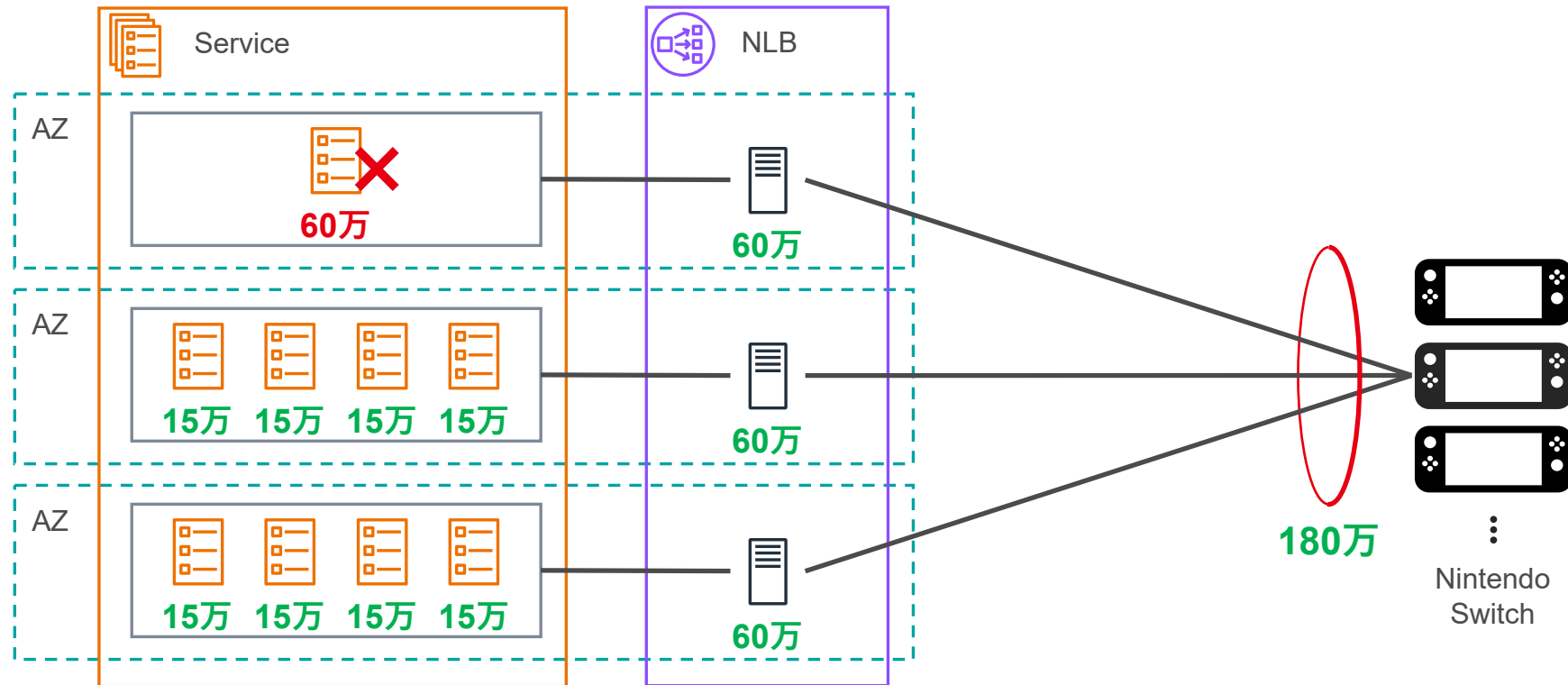
# 接続先の Task を分散させる

- ▶ 理想的には全ての Task に均等に接続してほしい
  - ▶ もし180万接続を 3AZ / 9Task で処理するなら...



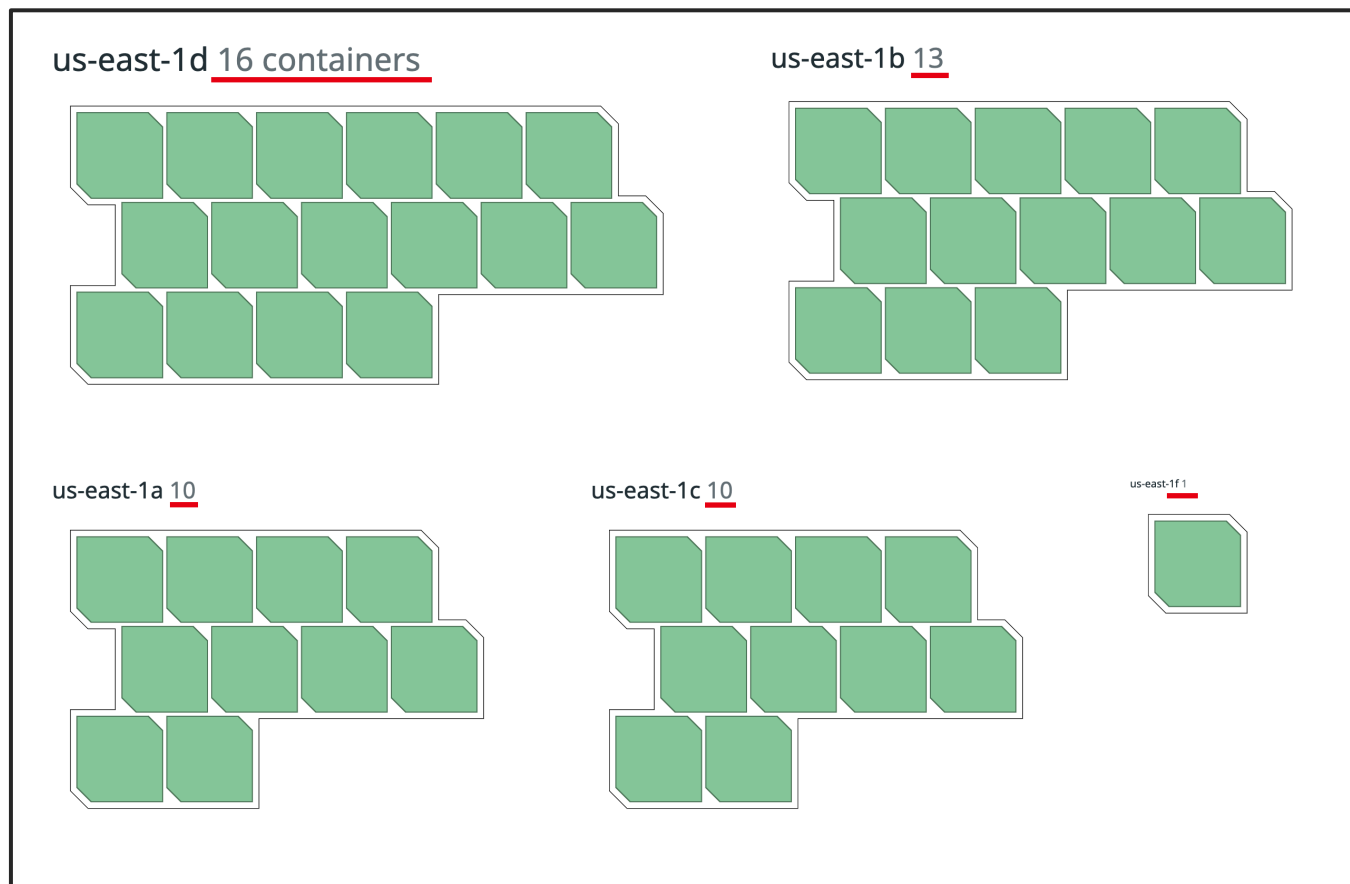
# 不均等な Task 配置

- ▶ Task が AZ に均等に配置される保証はない
  - ▶ Task 数の偏りによって、限界を超える負荷が集中してしまう可能性がある



# 不均等な Task 配置

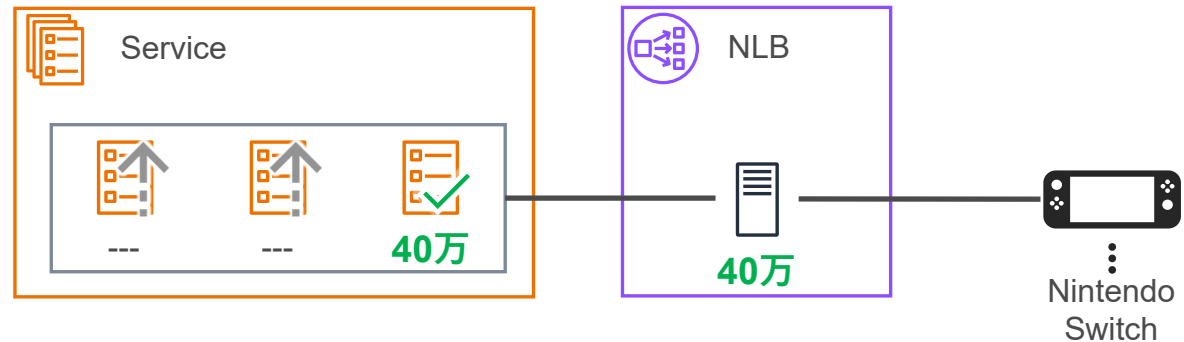
- ▶ AZ に Task が一つしか無い状況も実際に起こる



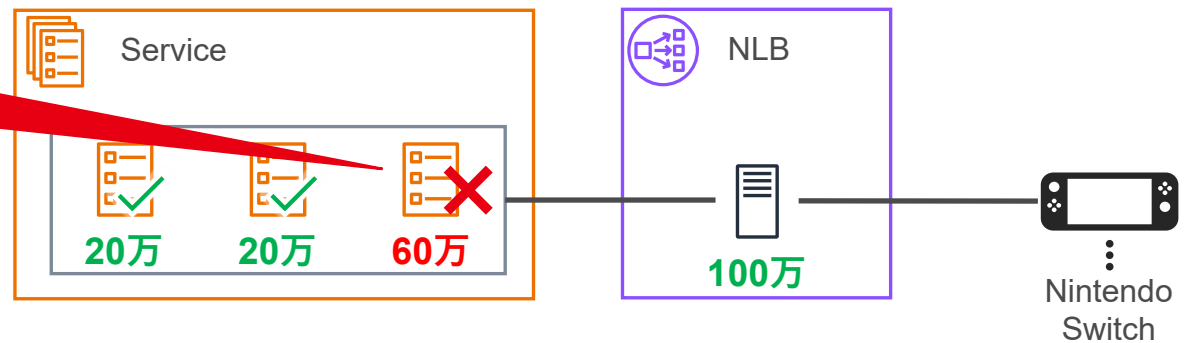
# 空間的な負荷分散

- ▶ NLB は least connection をサポートしていない
- ▶ 一度接続が開始されると、再接続までリバランスの機会がない

一部の Task が  
遅れて起動すると...



早く起動した Task に  
接続が偏ってしまう

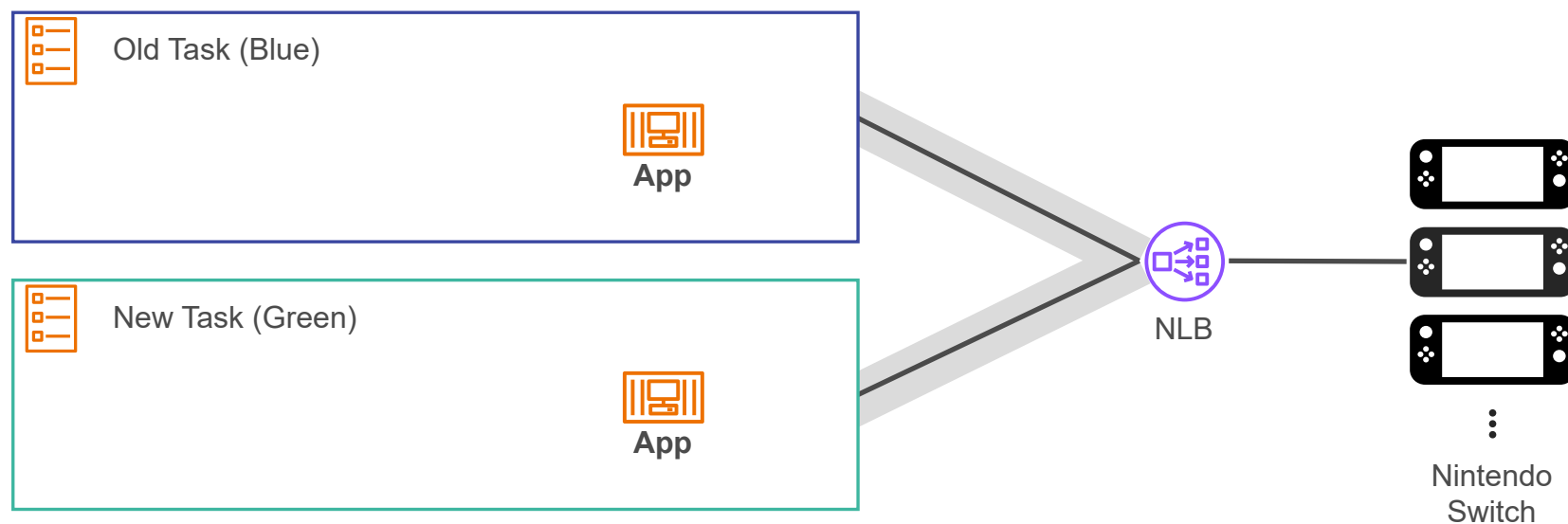


# 常時接続サービスのデプロイの要求

- ▶ 再接続処理はゆっくりとしてほしい
- ▶ Task は AZ に均等であってほしい
- ▶ Task はなるべく同時にサービスインしてほしい

# 常時接続サーバのデプロイ [1/16]

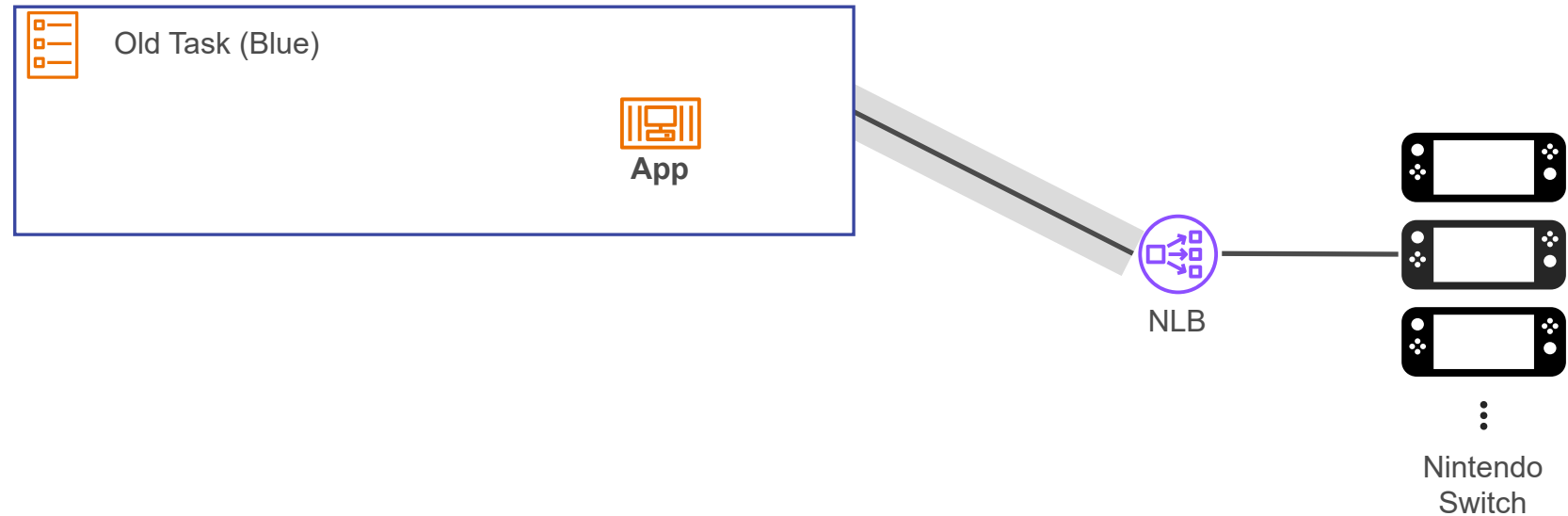
- ▶ Old Task から New Task に接続を切り替えることを考える





# 常時接続サーバのデプロイ [2/16]

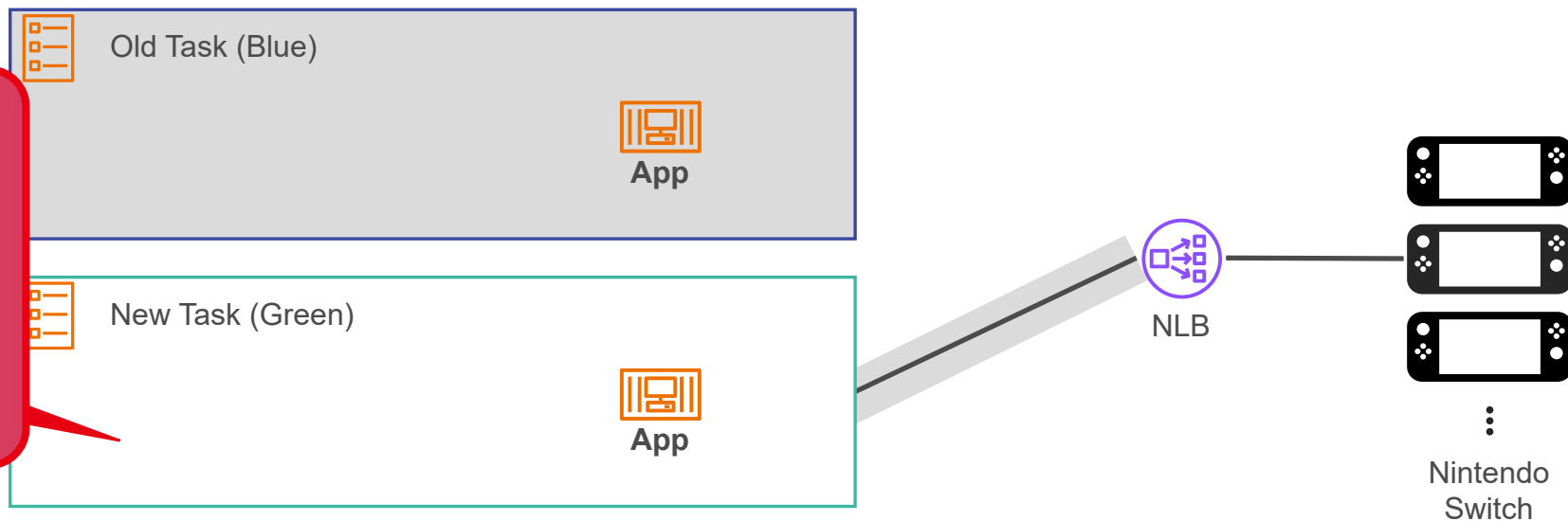
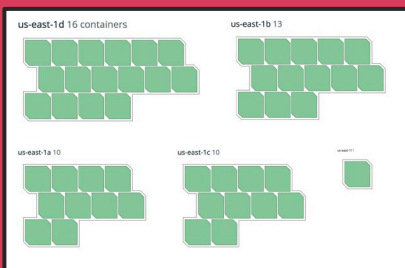
## ▶ デプロイ開始前



# 常時接続サーバのデプロイ [3/16]

✖ ▶ 新しい Task を立ち上げると即座に新規接続が流れてしまう

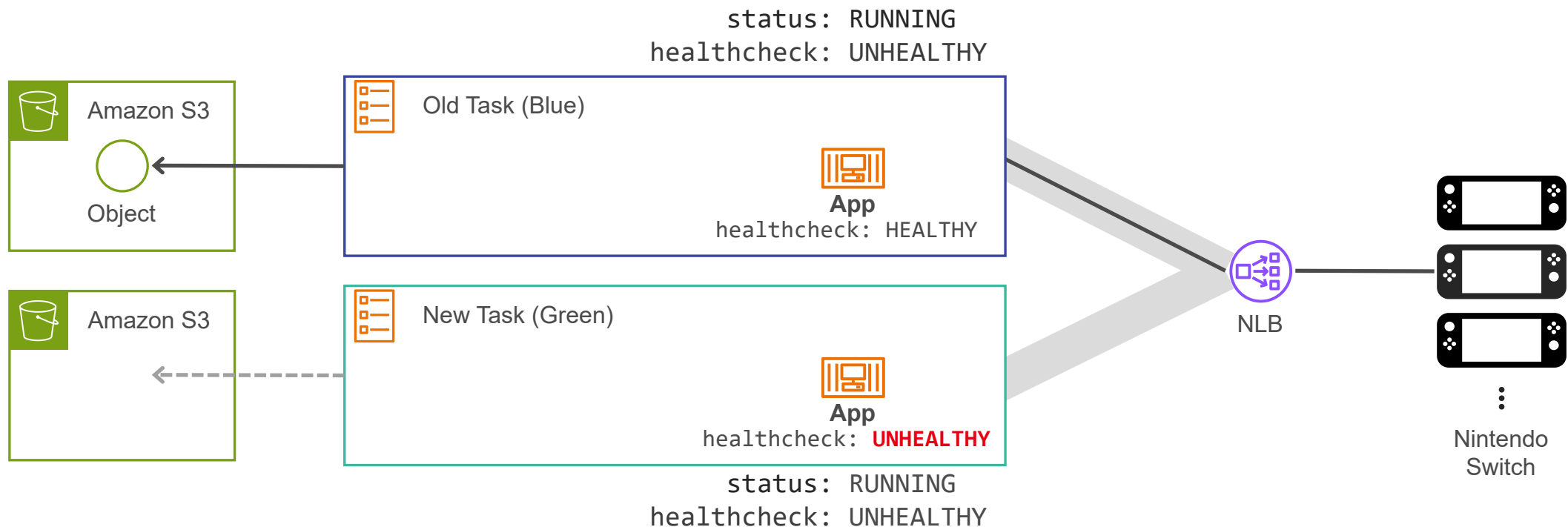
もしかしたら  
AZ 不均等な状態かも...?



# ヘルスチェックの状態を制御する仕組み

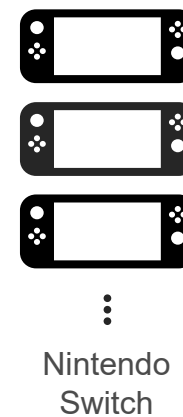
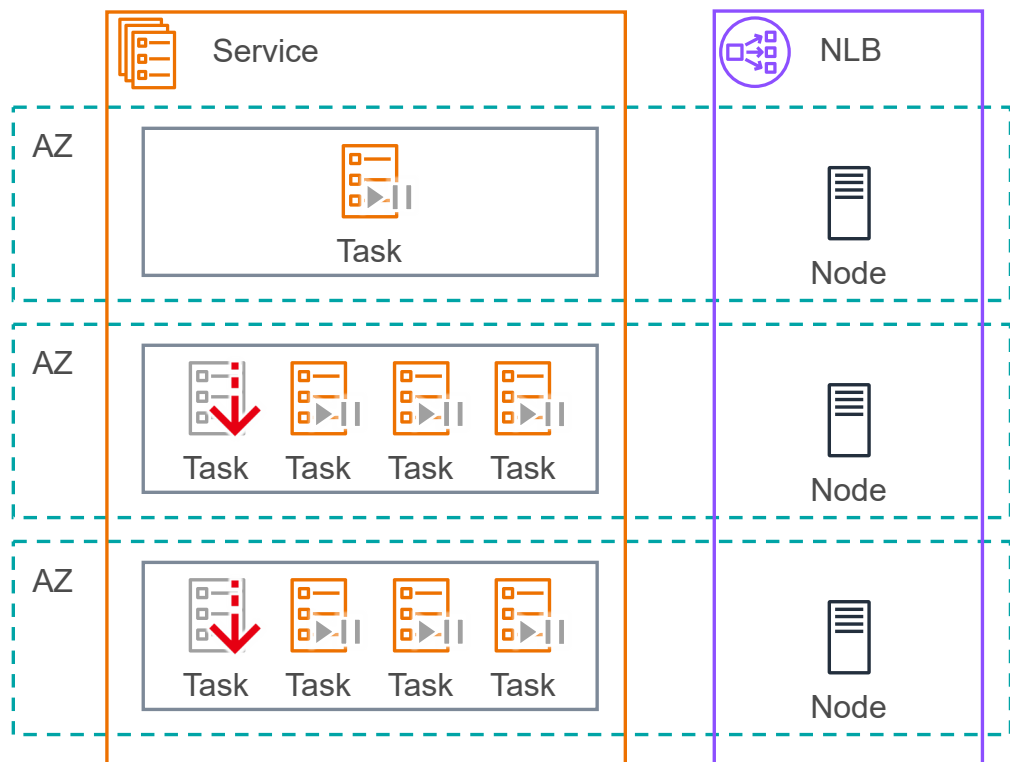


- ▶ AZ 均等な状態になるまで NLB から UNHEALTHY と判定させる
  - ▶ App の healthcheck を Amazon Simple Storage Service (Amazon S3) 上に特定のオブジェクトが存在しない限り失敗させる



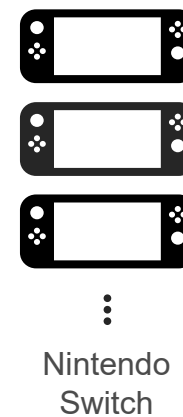
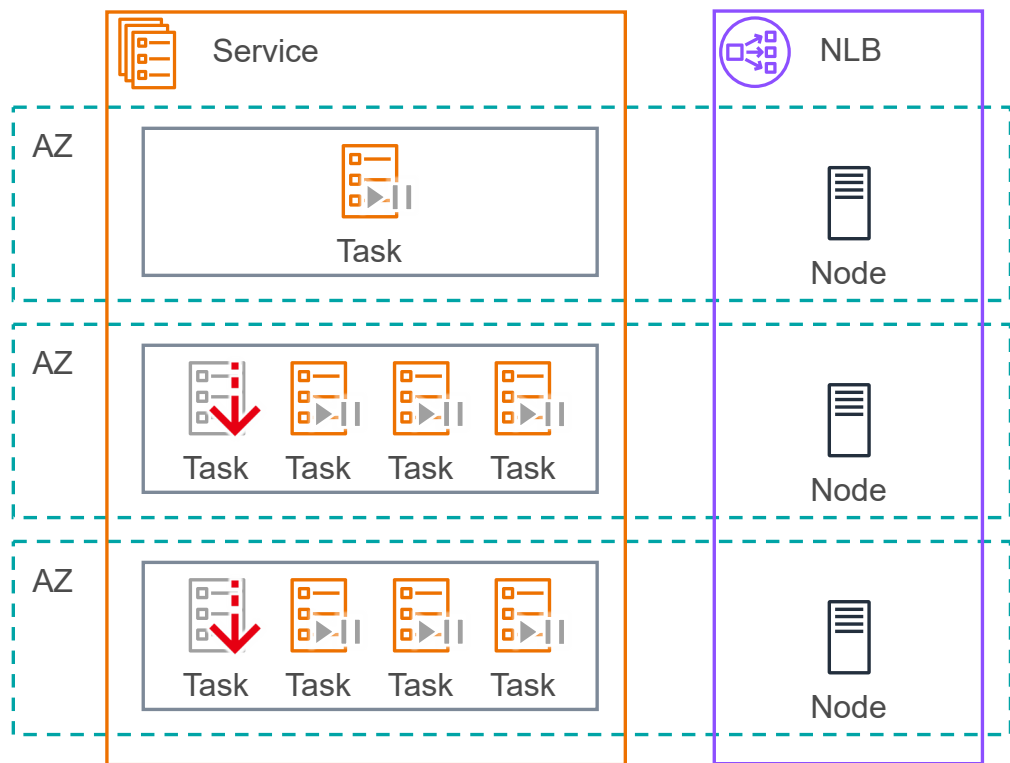
# 常時接続サーバのデプロイ [4/16]

- ▶ この間に Task が多い AZ の Task を stopTask する



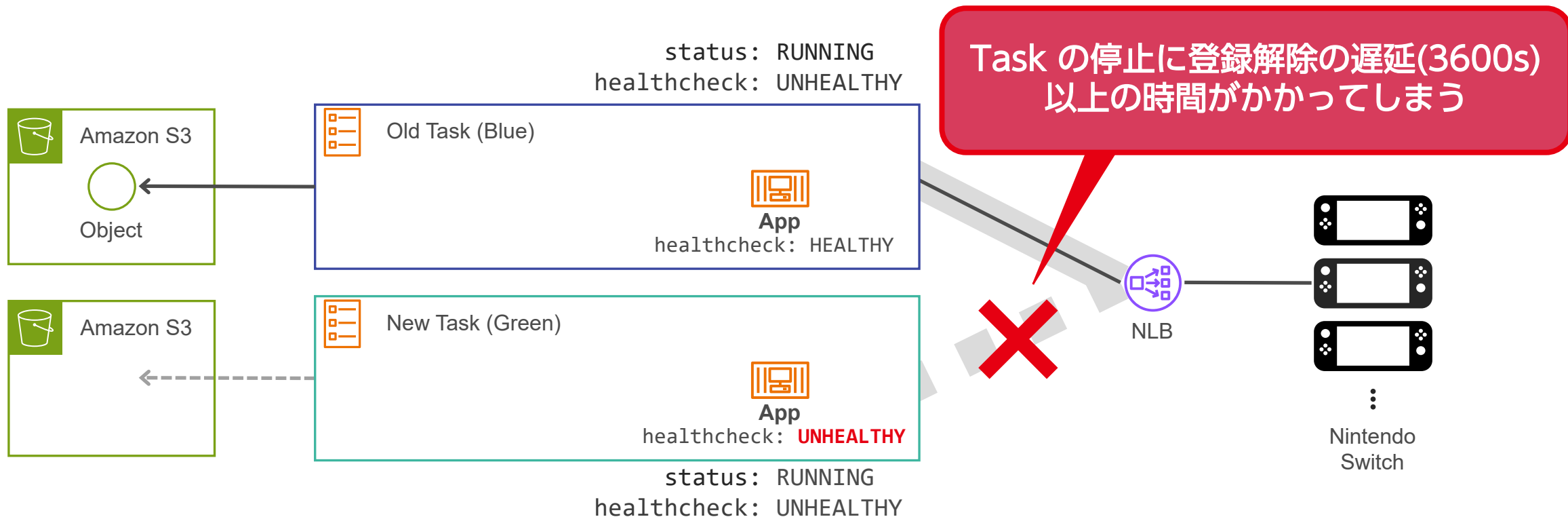
# 常時接続サーバのデプロイ [5/16]

- ✖ この間に Task が多い AZ の Task を stopTask する
- ▶ 今の状態だと、リバランスに数時間かかる



# ヘルスチェックの状態を制御する仕組み

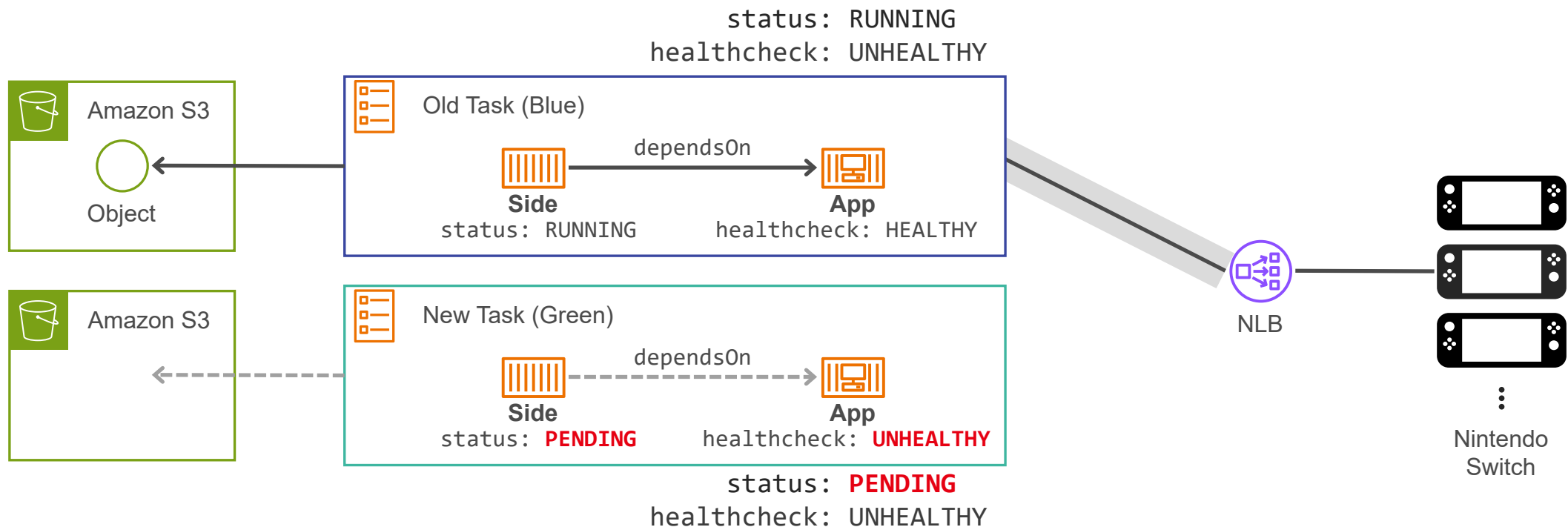
- ✖ AZ 均等な状態になるまで NLB から UNHEALTHY と判定される
  - App の healthcheck を Amazon S3 上に特定のオブジェクトが存在しない限り失敗させる



# NLB への登録を阻止する仕組み

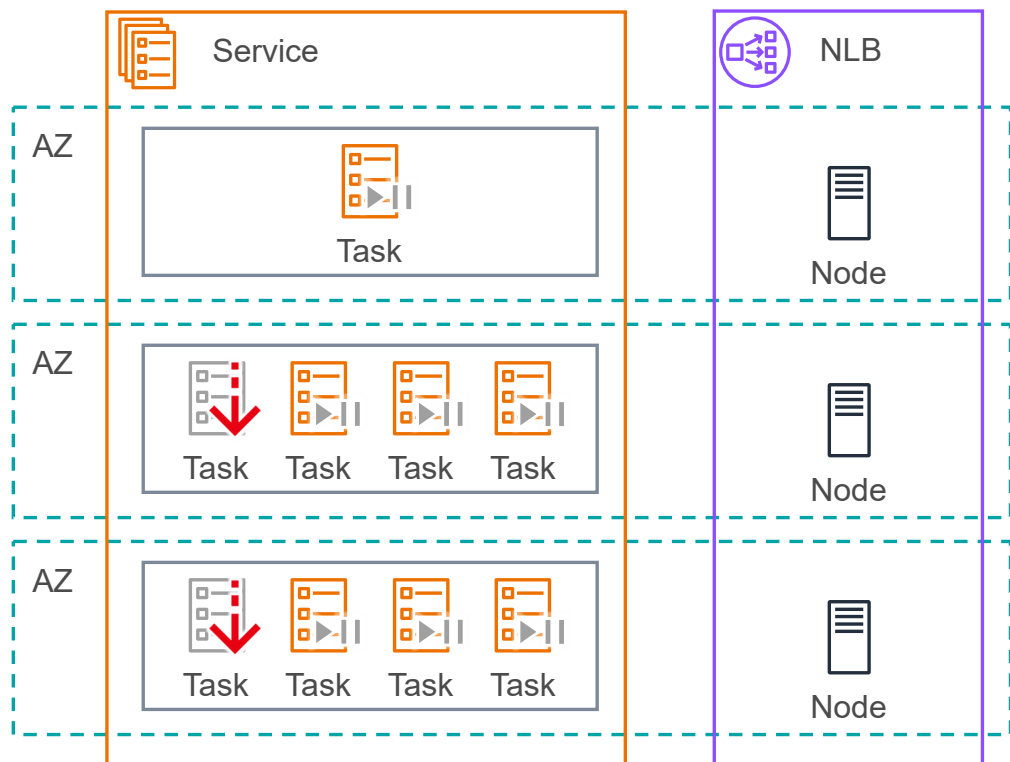
## に登録させない

- ▶ AZ 均等な状態になるまで NLB から ~~UNHEALTHY~~ と判定される
  - ▶ App の healthcheck を Amazon S3 上に特定のオブジェクトが存在しない限り失敗させる
  - ▶ App に dependsOn する Side コンテナを追加する

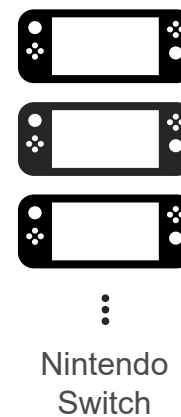


# 常時接続サーバのデプロイ [6/16]

- ✓ ▶ この間に Task が多い AZ の Task を stopTask する
  - ▶ NLB に登録されなければ、Task の再作成は数十秒で完了



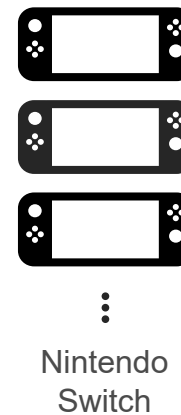
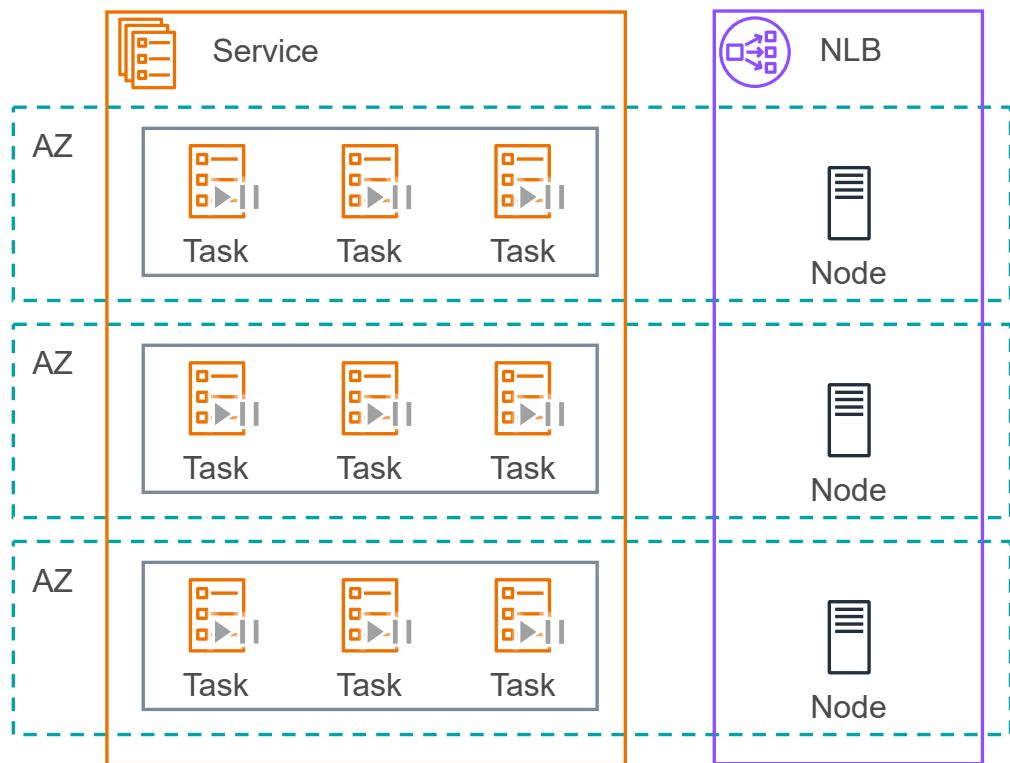
この仕組みを実現するためには Task 内の全てのイメージを Tag ではなく Digest で指定する必要があります※  
(2024/8/1 補足追記しました)





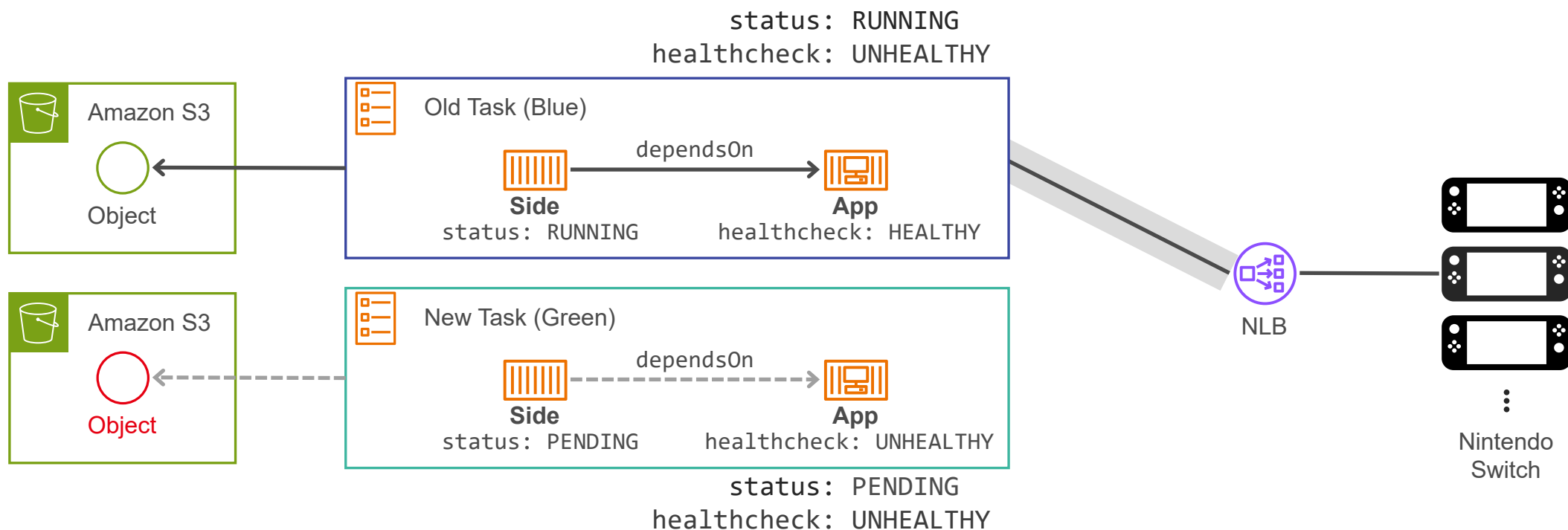
# 常時接続サーバのデプロイ [7/16]

- ▶ AZ 毎の Task 数が同じになるまで繰り返す



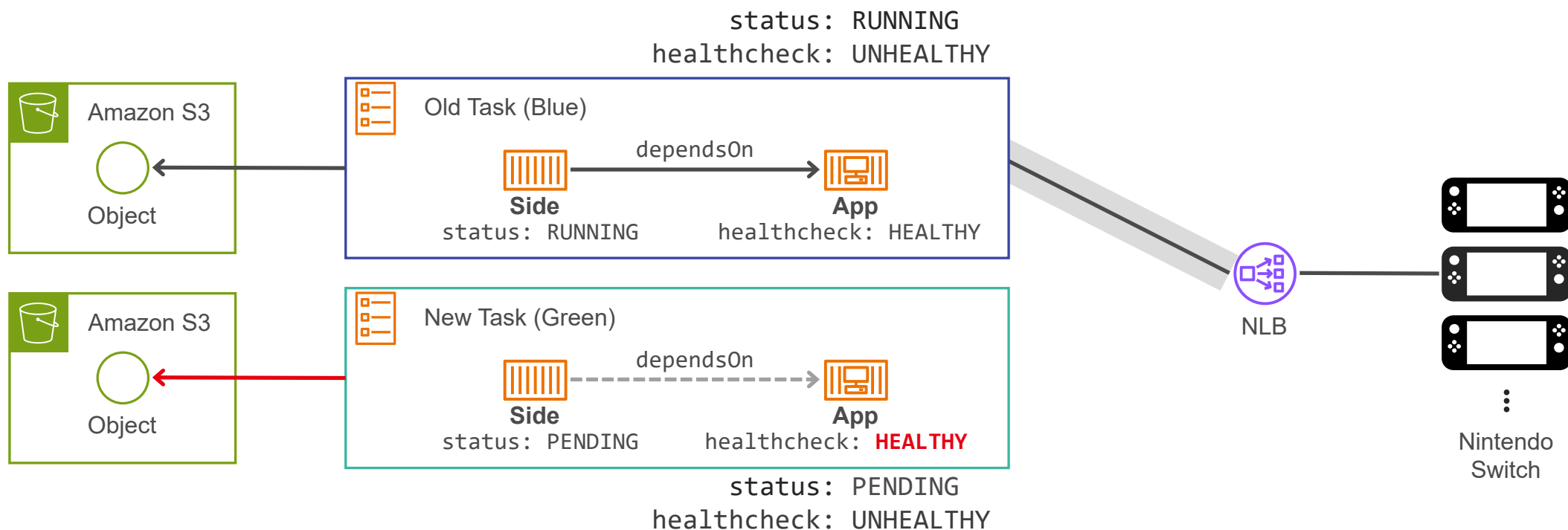
# 常時接続サーバのデプロイ [8/16]

- ▶ Amazon S3 上にオブジェクトを置く



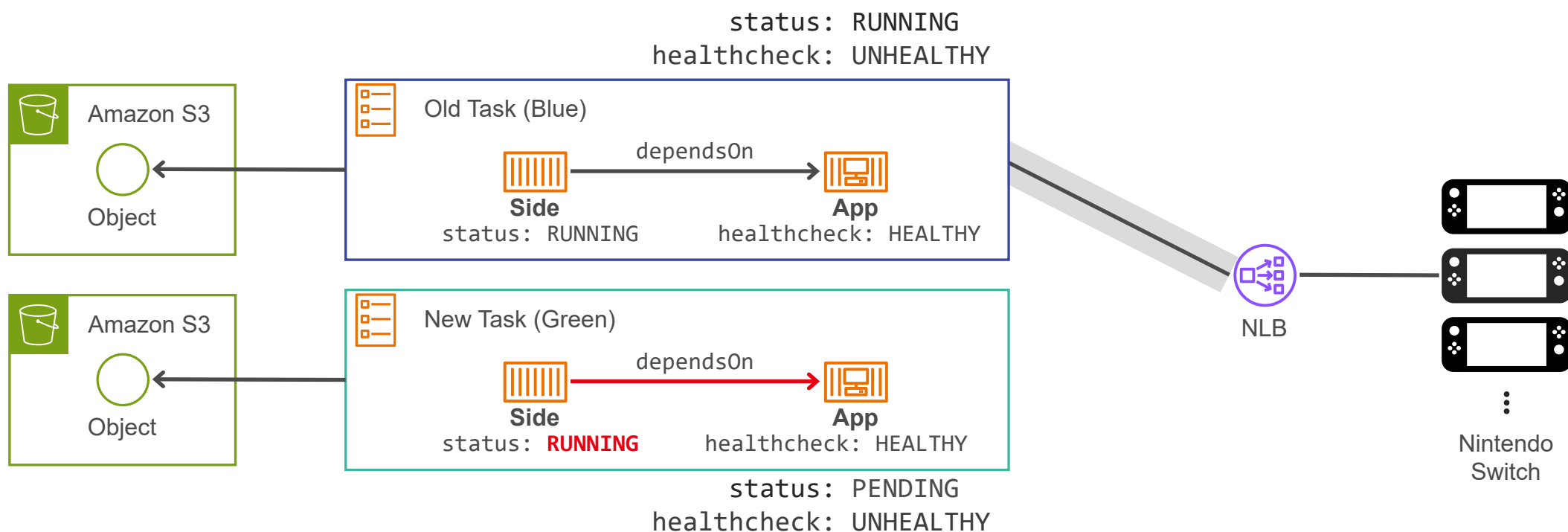
# 常時接続サーバのデプロイ [9/16]

- ▶ App コンテナの healthcheck が成功するようになる



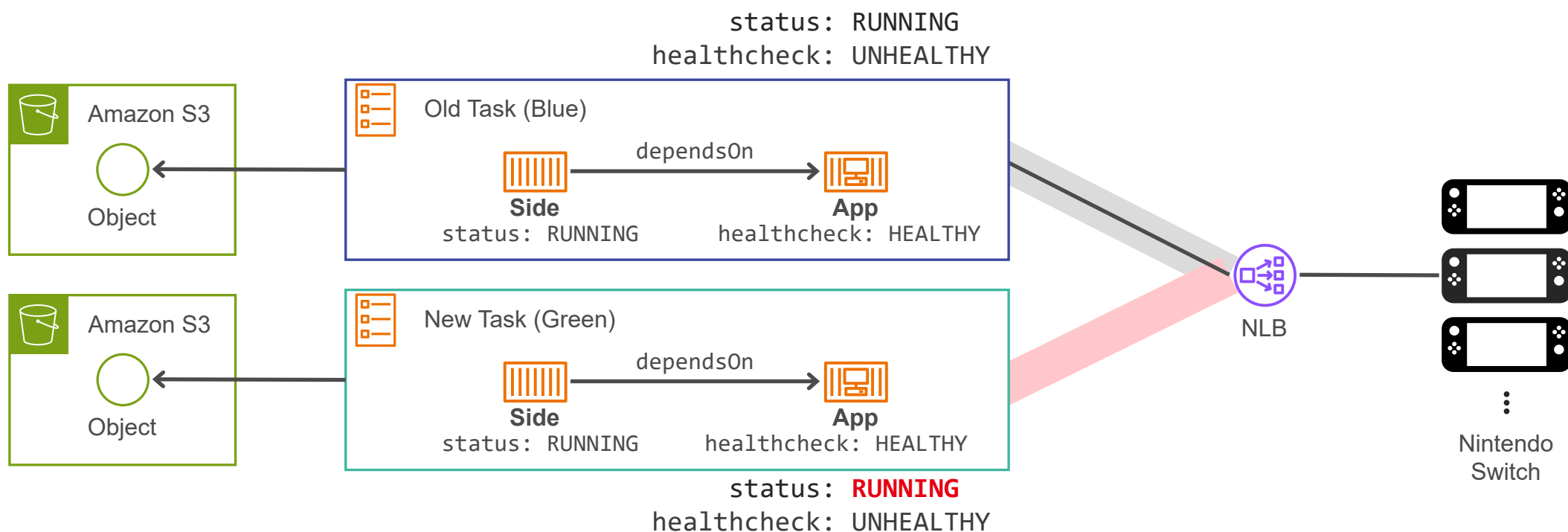
# 常時接続サーバのデプロイ [10/16]

- ▶ Side コンテナが RUNNING になる



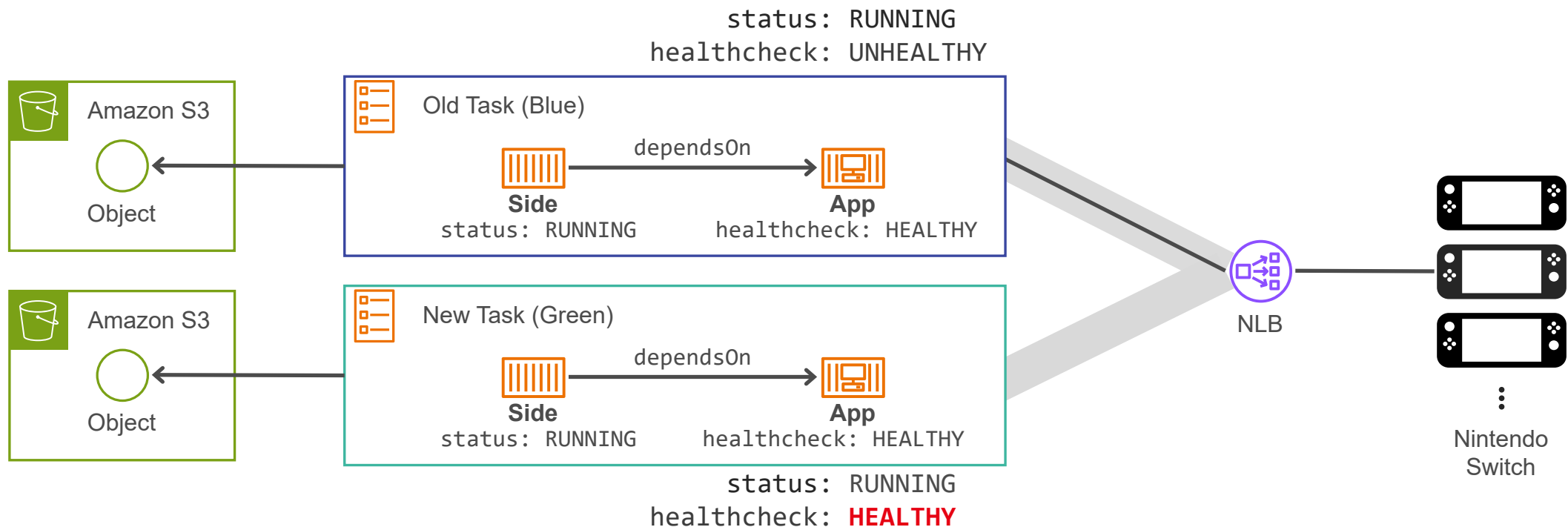
# 常時接続サーバのデプロイ [11/16]

- ▶ Task の status が RUNNING になり NLB に登録される



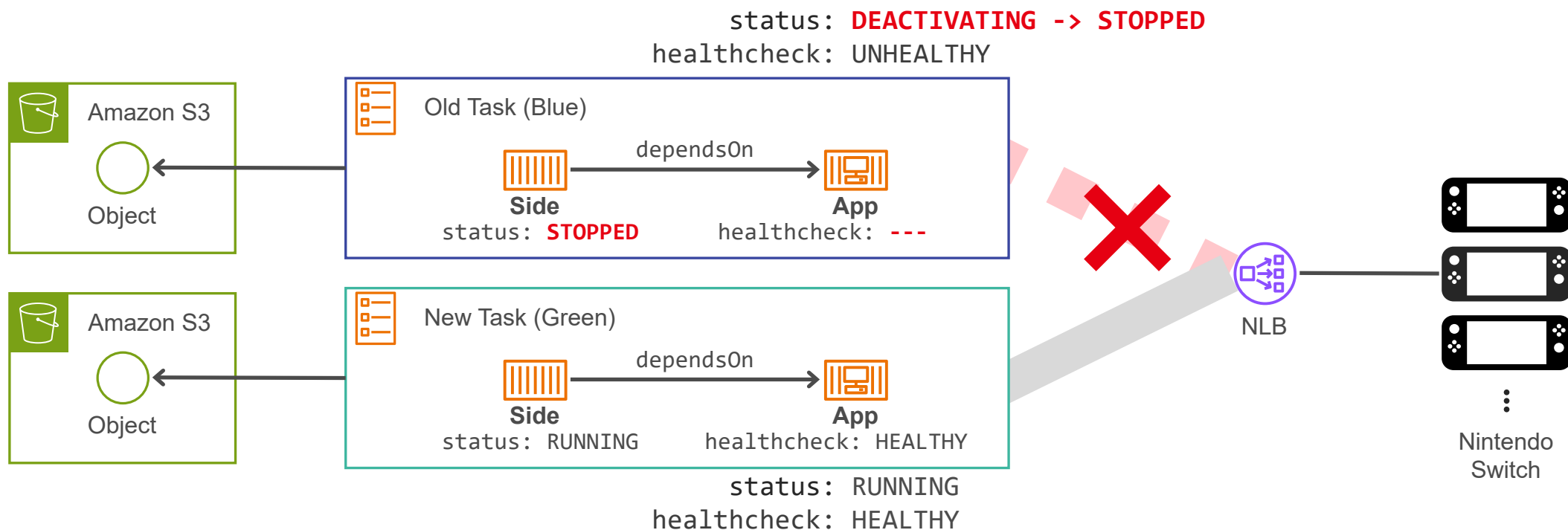
# 常時接続サーバのデプロイ [12/16]

- ▶ healthcheck に成功し、新規接続が New Task に流れるようになる



# 常時接続サーバのデプロイ [13/16]

✖ Old Task は stopTask により全接続が強制切断されてしまう



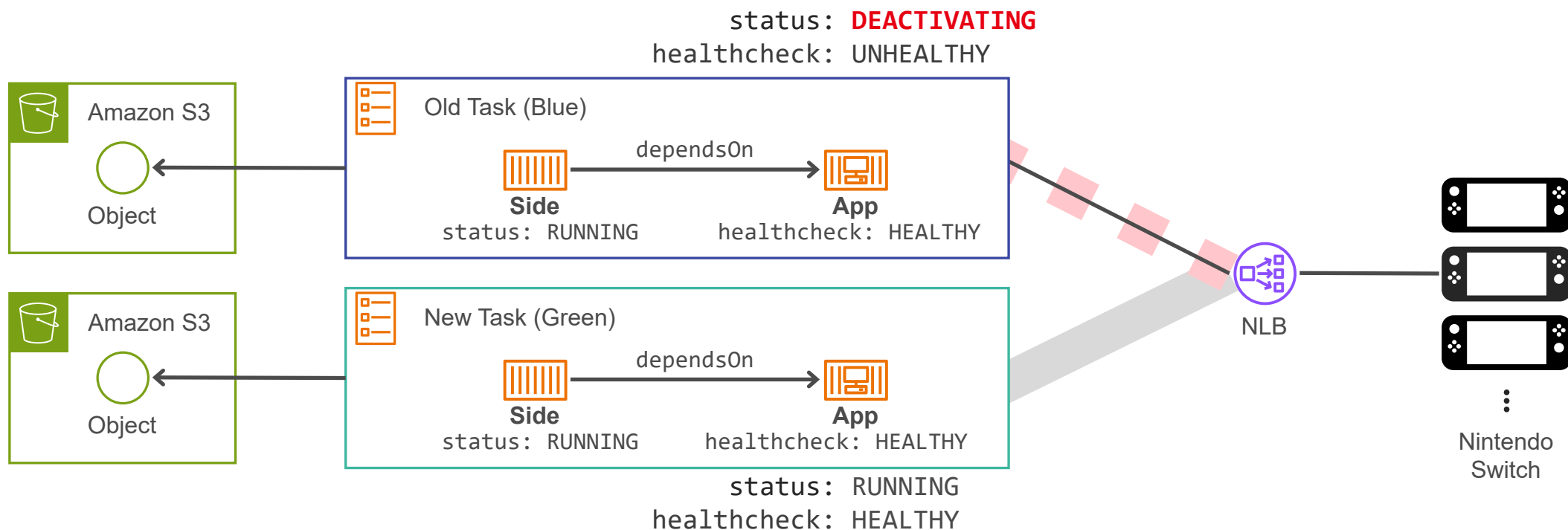
# デプロイ時に再接続する仕組み

- ✓ ▶ App が NLB からの登録解除を検知出来るロジック
- ▶ Nintendo Switch に一定の速度で再接続を促す機能 (ドリップ処理)
  
- ▶ デプロイを実現するための設定
  - ▶ 登録解除の遅延 : 3600s
    - ▶ NLB から登録解除されてから 3600s 待機後に Amazon ECS が Task のコンテナに SIGTERM を送信する



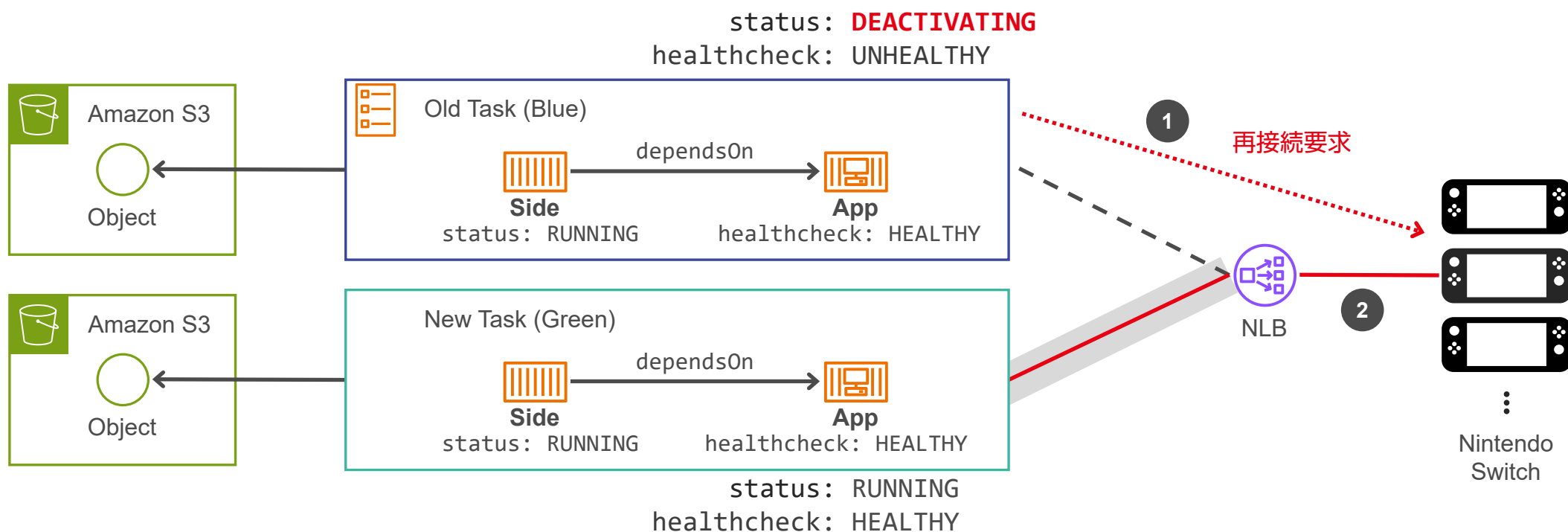
# 常時接続サーバのデプロイ [14/16]

- ▶ stopTask を受けると status は DEACTIVATING になり、NLBから登録解除される



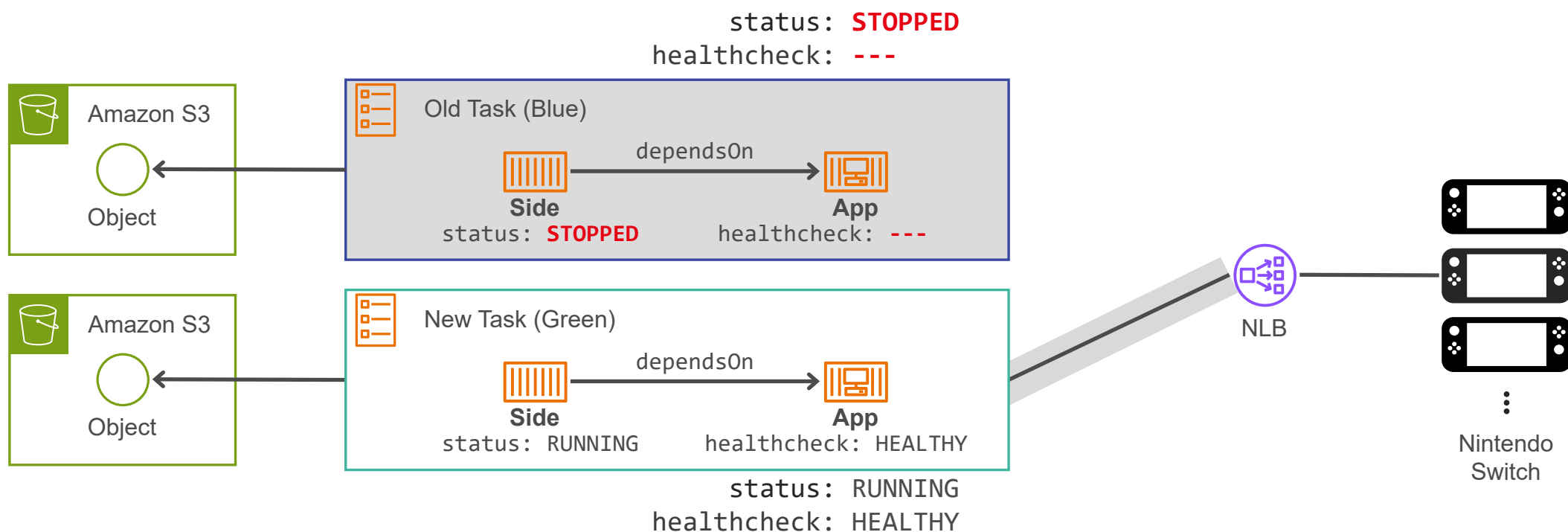
# 常時接続サーバのデプロイ [15/16]

- ▶ App は自身の Task が登録解除されたことを検知して、一定の速度で Nintendo Switch に再接続要求を出していく



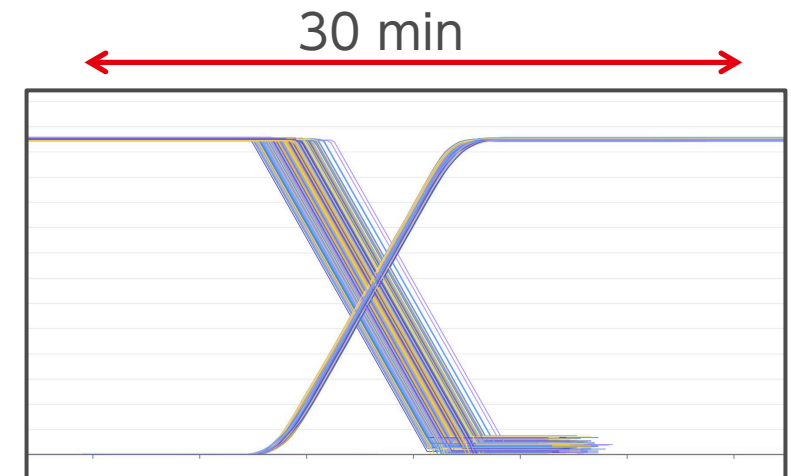
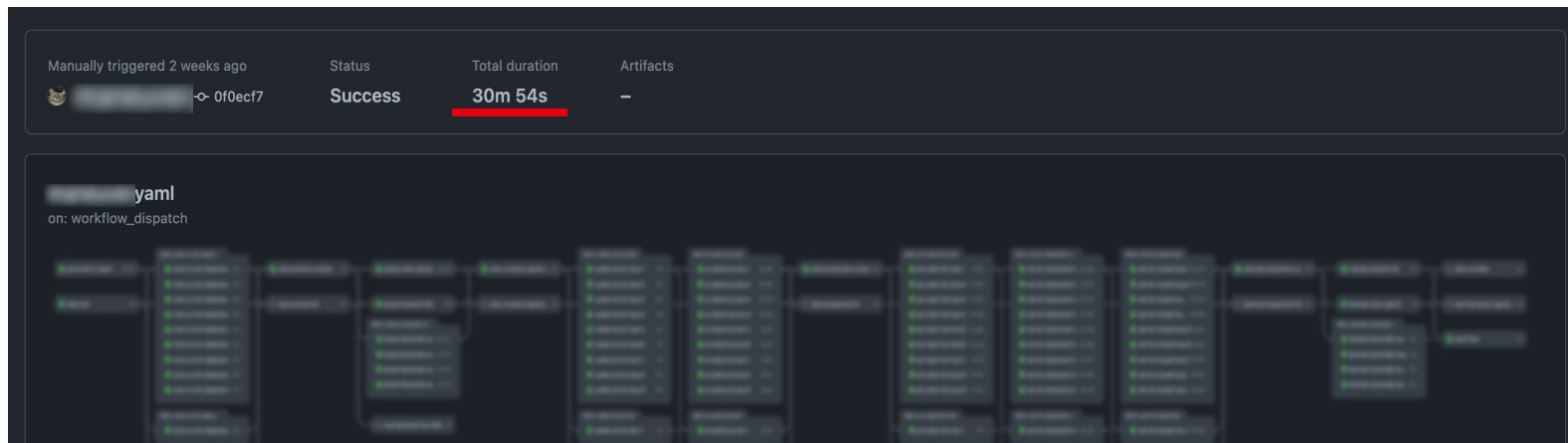
# 常時接続サーバのデプロイ [16/16]

- ▶ 全ての Nintendo Switch に再接続要求を出し終わると、App プロセスが終了し、Task も STOPPED 状態となる



# デプロイの自動化

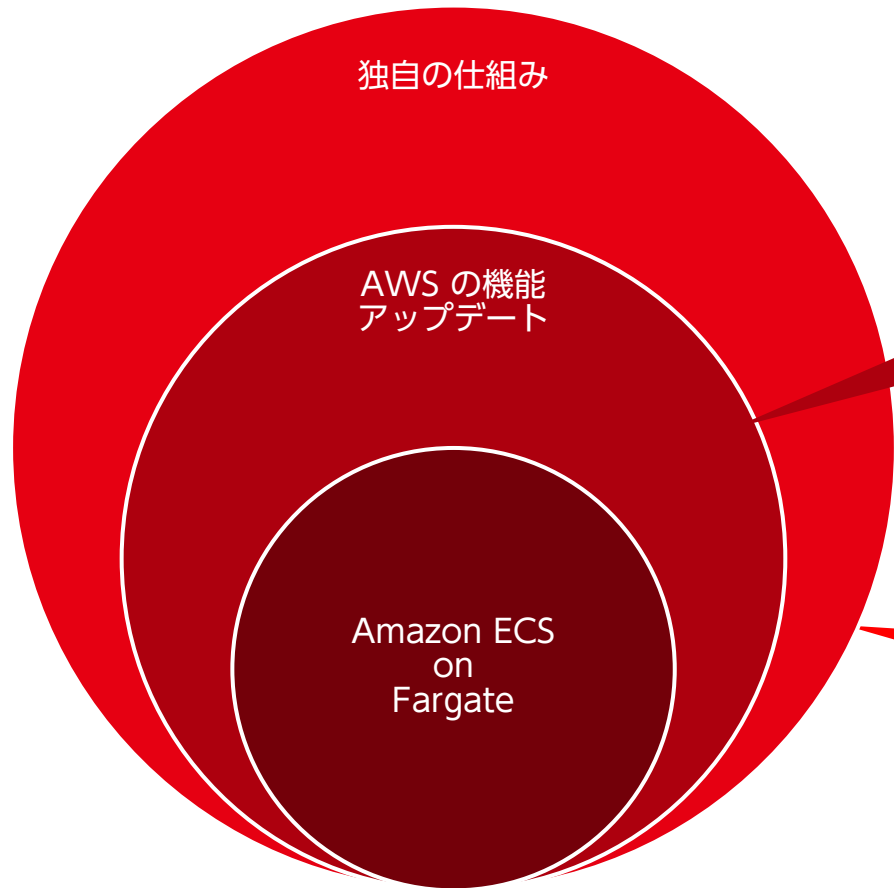
- ▶ GitHub Actions のワークフローとして実装
  - ▶ Slack 通知やデプロイ前後の作業も含め、全て自動で実施
  - ▶ 負荷試験にて1億台の接続を維持した状態で挙動が問題ないことを確認
- ▶ 約30分でデプロイが完了する
  - ▶ 切断中に来た通知は再接続完了後に受け取れるため、ユーザー影響なし



Task ごとの接続台数

# Amazon ECS on Fargate で 大規模常時接続システムを実現する

## 大規模常時接続に必要な要件



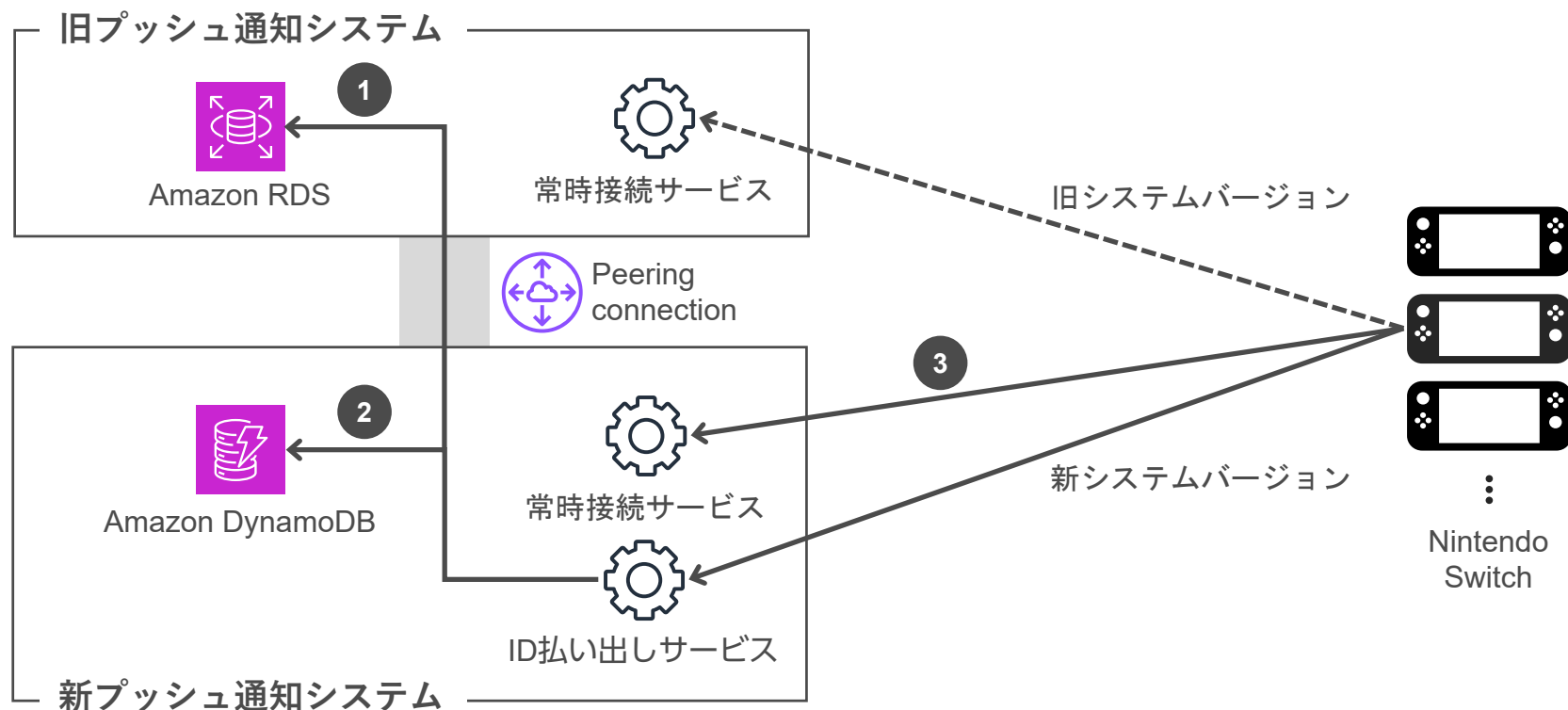
- ▶ TCP カーネルパラメータをチューニングして、低負荷で安定した通知サービスを引き続き提供
- ▶ クライアントIPの保存 を有効にすることで、NLB の port 枯渇を回避可能に

- ▶ デプロイツールを自作することで、大量の接続を安定かつ高速に再接続させる

# リリース方法と振り返り

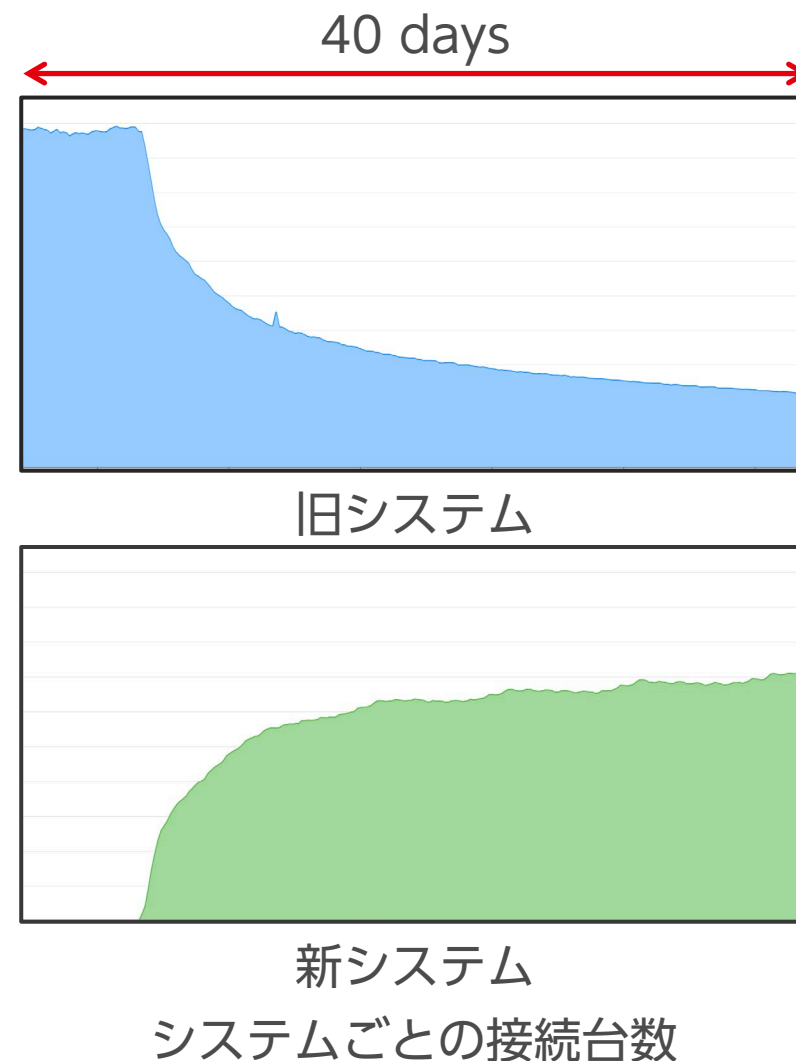
# リリース方法

- ▶ システムバージョン更新により接続先システムを切り替える
  - ▶ 初回アクセス時に旧システムからのデータ移行を行う



# リリース後の振り返り

- ▶ 2024年前半にリリース済み
  - ▶ 無停止で実施
  - ▶ 着実に移行が進んでいる (右のグラフ)
- ▶ リリース後も安定した運用を達成
  - ▶ Amazon Elastic Kubernetes Service を利用した k6 による本番規模負荷試験
  - ▶ AWS Fault Injection Service を利用した障害試験





# まとめと展望

# まとめ

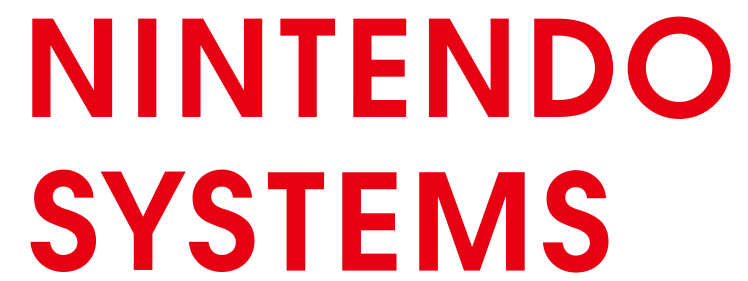
- ▶ より長期での運用を見据えて、Nintendo Switch 向けプッシュ通知システムのリプレースを実施
- ▶ Amazon ECS on Fargate を利用して、同時接続数1億以上のスケーラビリティをもつ大規模常時接続システムの構築を達成
- ▶ サービス無停止でのリリースに成功

# 今後の展望

- ▶ Graviton ベースの Fargate の活用
  - ▶ 常時接続サーバーが、インフラ費用の大部分を占める
- ▶ 機能拡張
  - ▶ フレンドのオンライン状態管理機能の巻取り
    - ▶ 常時接続によって、デバイスのオンライン状態を効率的に把握できる
    - ▶ システム間トラフィックが削減される
  - ▶ デバイス間通知のサポート
    - ▶ サーバー to デバイスだけでなく、デバイス to デバイスの通知送信をできるようにしたい

The Nintendo logo, featuring the word "Nintendo" in white, sans-serif font inside a white, rounded rectangular border, set against a solid red background.

**Nintendo®**

The text "NINTENDO SYSTEMS" in a bold, red, sans-serif font, arranged in two lines. The text is centered within a white rectangular area that is framed by a thick red border.

**NINTENDO  
SYSTEMS**