AWS Summit Tokyo 2019
13:00-13:40　H2-04　技術セッション　ゲーム　移行

ネクソンの多彩なゲームコンテンツポートフォリオを支える基盤と運用の最適化について

サービス基盤室

コンテンツ

† 10年以上続くPCオンラインゲーム(＝レガシーシステム)を
リホスト戦略によりオンプレミスからクラウドへマイグレーション！

† クラウドオリエンテッドに設計したモバイルゲームタイトルの
構築フェーズからリリース／運用で押さえるべきポイント！

ネクソン概要・代表作等

2003

2019

メイプルストーリー

アラド戦記

FAITH

メイプルストーリー2

マビノギ

テイルズウィーバー

配信タイトル数

60

タイトル以上

配信地域数

190

カ国以上

OVERHIT

メイプルストーリーM

サドンアタック

Tree of Savior

ドミネーションズ

DarkAvenger X

サービス基盤のミッション

- **高効率なサービスの安定運用**
- **柔軟かつスピーディーなプロビジョニング**

ビジネスの持続的成長をドライブ

仲間を集い

技術を研き

基盤と運用を常に最適化していく

第一部へ

† 10年以上続くPCオンラインゲーム(＝レガシーシステム)を
リホスト戦略によりオンプレミスからクラウドへマイグレーション！

全体概要

移行背景

- クラウド移行のきっかけ
  - ユーザ数の変化に合わせたインフラコストの最適化。
  - インフラ構築時間の短縮。

- なぜAWSなのか
  - 既存ナレッジの活用。
  - 情報量が圧倒的に多い。

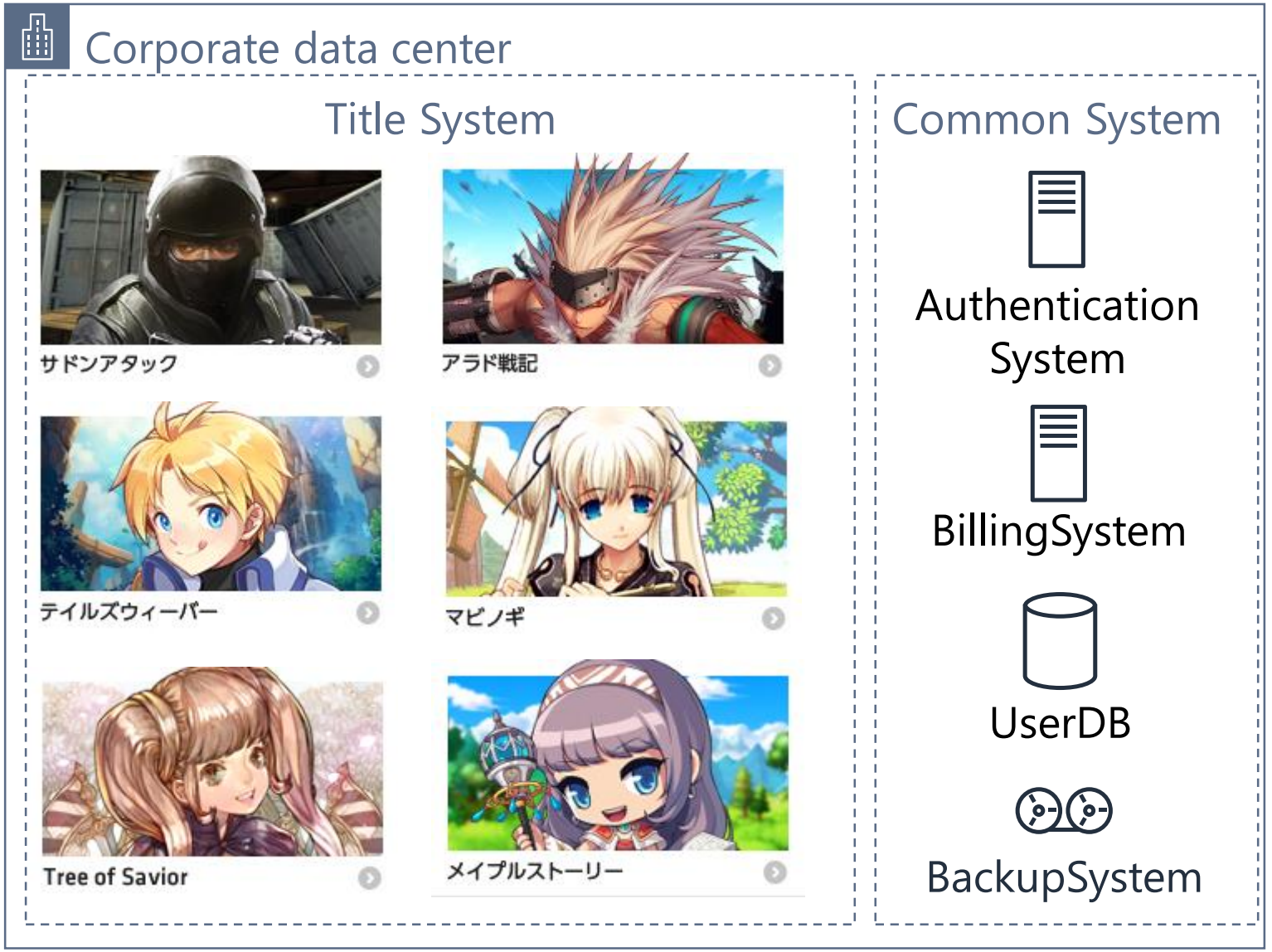
オンプレミス環境の全体構成図

タイトル毎のシステム

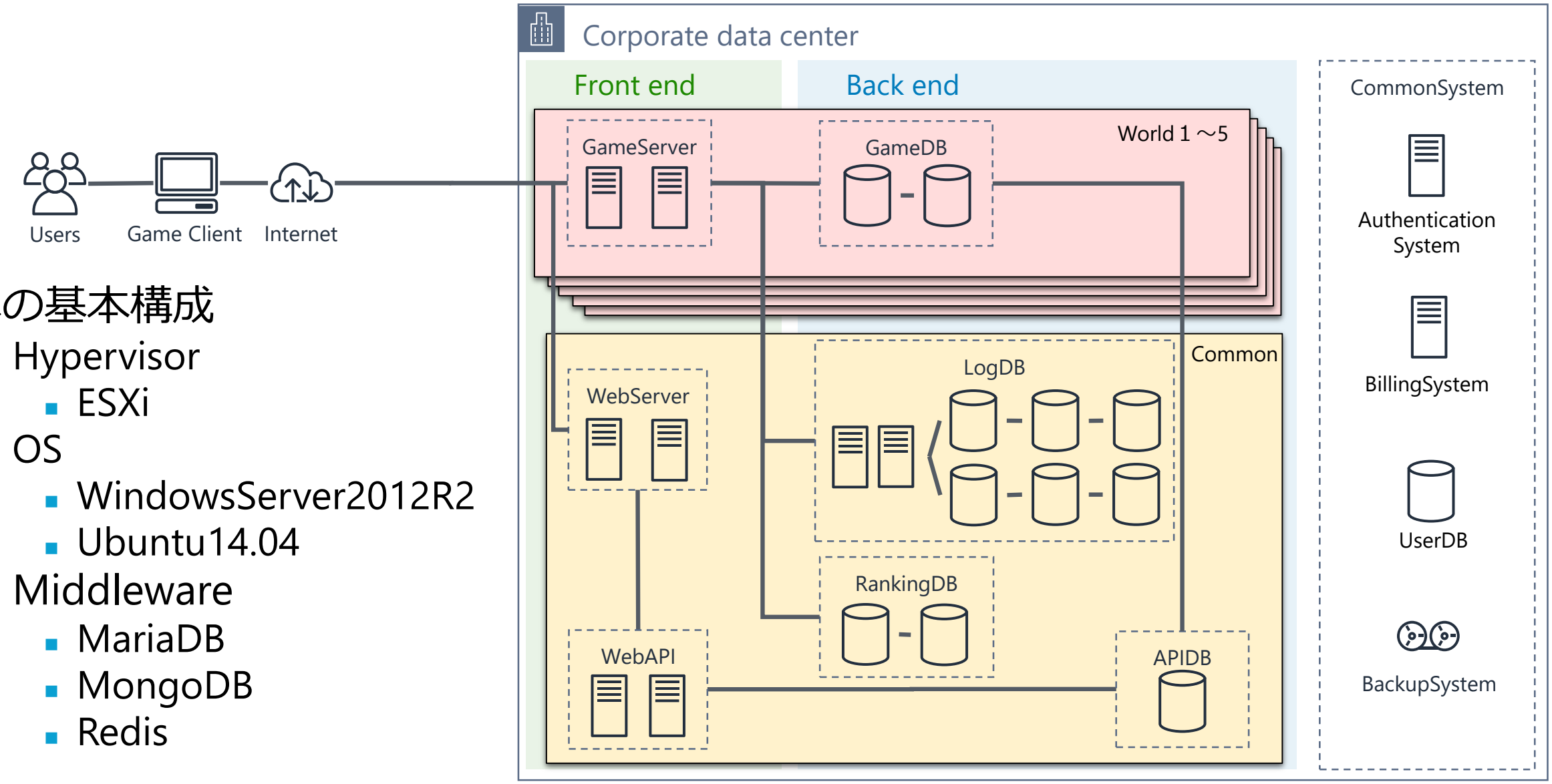
- 10年前後IDCで稼働している
- 各タイトルは複数ワールド構成
- 約6タイトルほどがIDCで稼働
- **今回の移行対象**

共通システム

- 各タイトルが連携する共通の各システム
  - 認証システム
  - 決済システム
  - 会員データベース
  - バックアップシステム
  - その他
- 今後移行予定



オンプレミス環境のタイトルシステム構成図

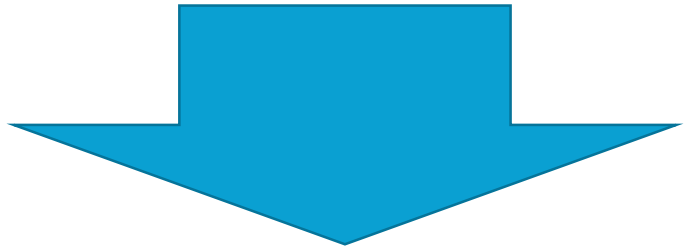


全体の基本構成

- Hypervisor
  - ESXi
- OS
  - WindowsServer2012R2
  - Ubuntu14.04
- Middleware
  - MariaDB
  - MongoDB
  - Redis

移行方針と構成のポイント

サービスへの影響を出さない



レガシーシステムの様々な制約を考慮する必要が有る。

- ブラックボックス化されたプログラム
- 冗長構成が取れない(AWS Auto Scalingが使用不可)。
- サーバ間通信が弱い(Multi-AZ構成が使用不可)。

クラウドへの移行パターン

Retire	移行せずシステムを撤去/廃棄
Retain	移行せずオンプレミスに保持
Rehost	アプリに改修を加えずそのまま移行
Refactor	アプリ仕様は維持し、一部マネージドサービスを利用
Revise	既存のアプリ仕様をベースに機能追加/改修
Rebuild	アプリを書き換え、マネージドサービスを活用
Replace	既存のアプリをマネージドアプリ・サービスに置き換え

https://image.slidesharecdn.com/20150915-awsmigration-150915091024-lva1-app6891/95/awswebinar-aws-25-638.jpg?cb=1442308346

移行方法の確定

タイトル毎にRehost ➡ Refactorを実施

フェーズ１ ：Rehost

まずは最適化等は考慮せずにクラウド化のみを行う。

フェーズ２ ：Refactor

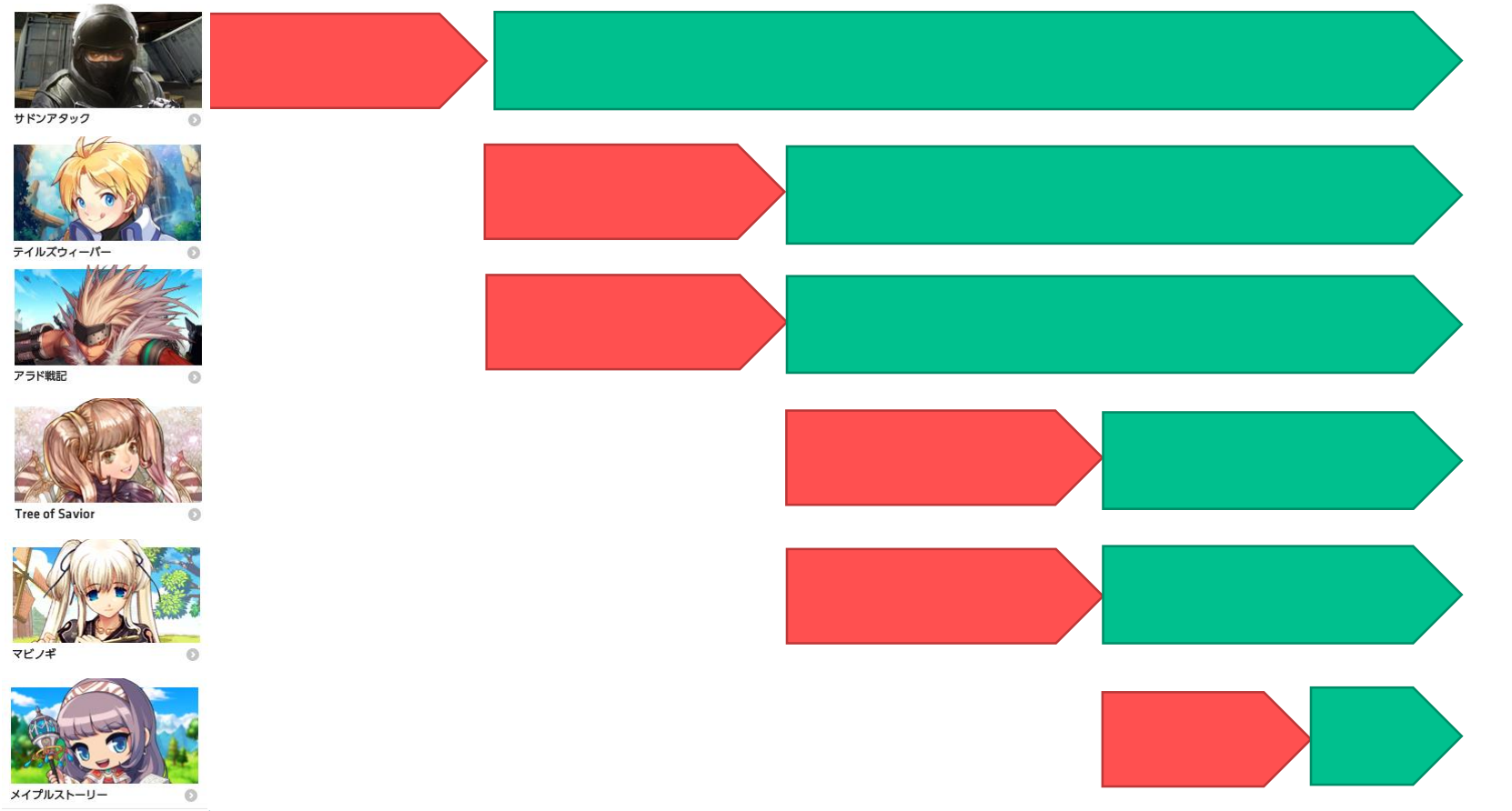
安定稼動後にクラウドへ最適化させていく。

移行ノウハウを蓄積しながら、安定的に移行を実施。

AWSへの移行フェーズ

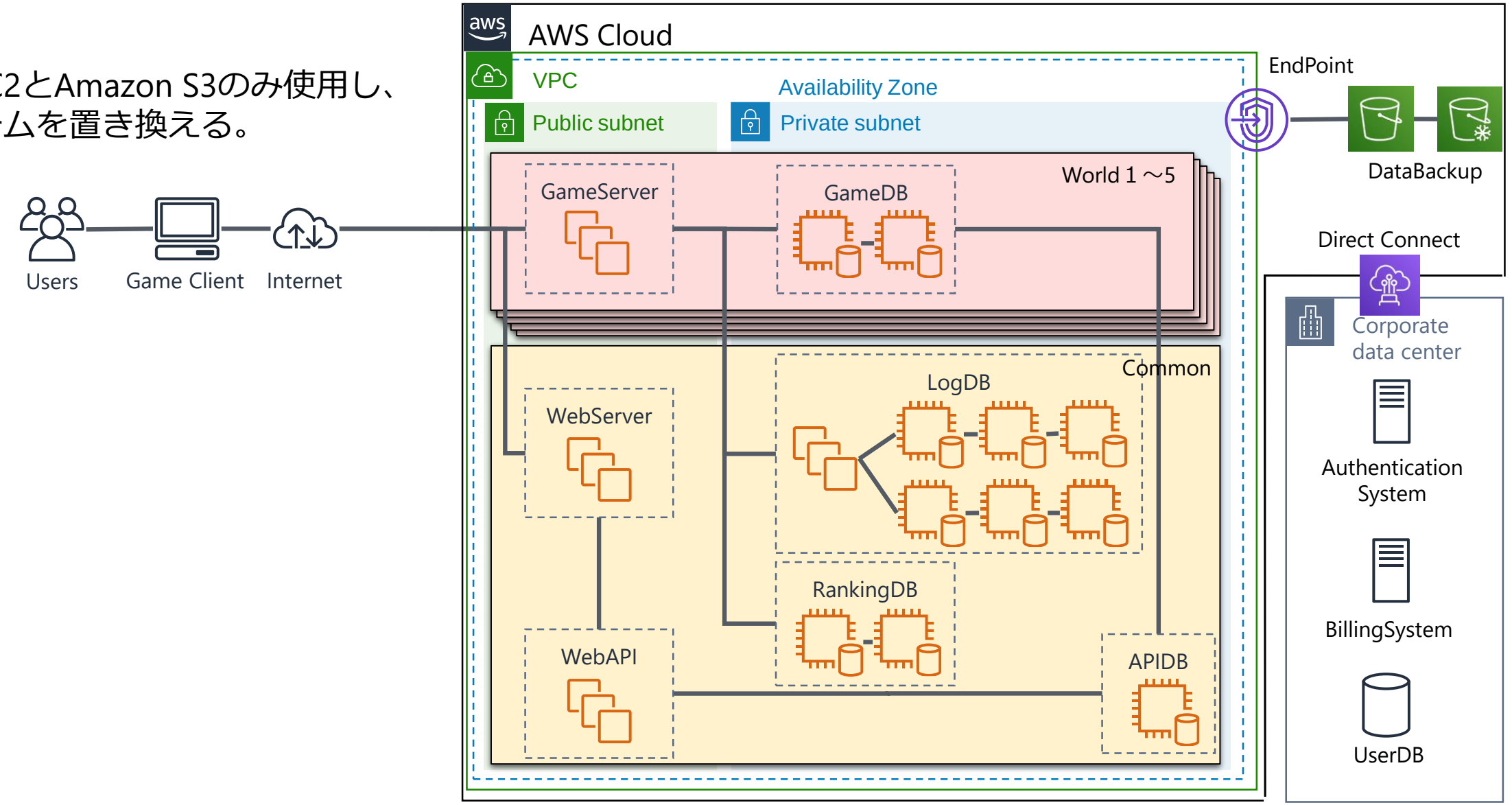
フェーズ１：Rehost
- 既存構成でリフト&シフト
- 短期間で移行

フェーズ２：Refactor
- クラウド最適化



AWS環境のタイトルシステム構成図

Rehost
Amazon EC2とAmazon S3のみ使用し、
既存システムを置き換える。



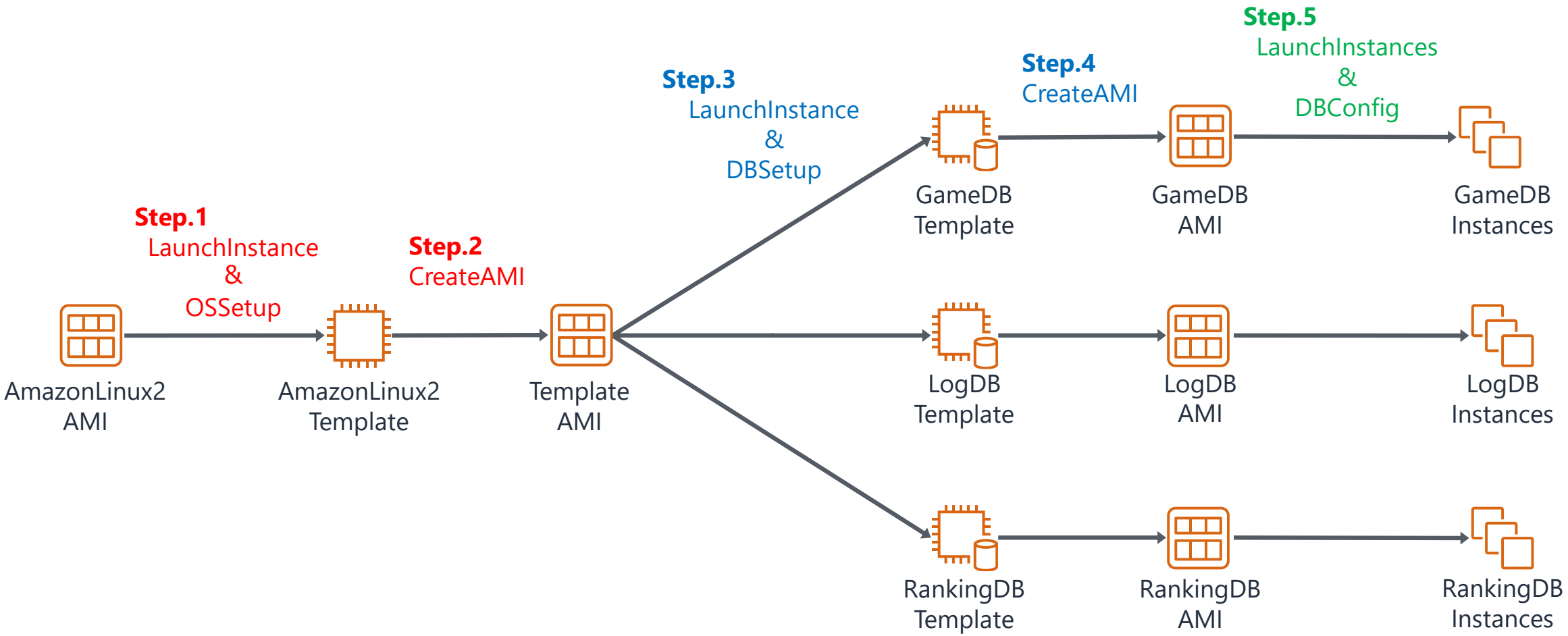
オンプレミスとAWSでの変更点

項目	オンプレミス	AWS
Frontend(Server)	Physical/ESXi	Amazon Elastic Compute Cloud (Amazon EC2)
Frontend(OS)	WindowsServer2012R2	Not Changed
Frontend(Middleware)	IIS	Not Changed
Backend(Server)	Physical/ESXi	Amazon EC2 on Database
Backend(OS)	Ubuntu14.04	AmazonLinux2　※EOLへの対応
Backend(Middleware)	MariaDB Redis MongoDB	Not Changed
DataBackup	SMBServer TapeStorage	Amazon Simple Storage Service (Amazon S3) Amazon Glacier

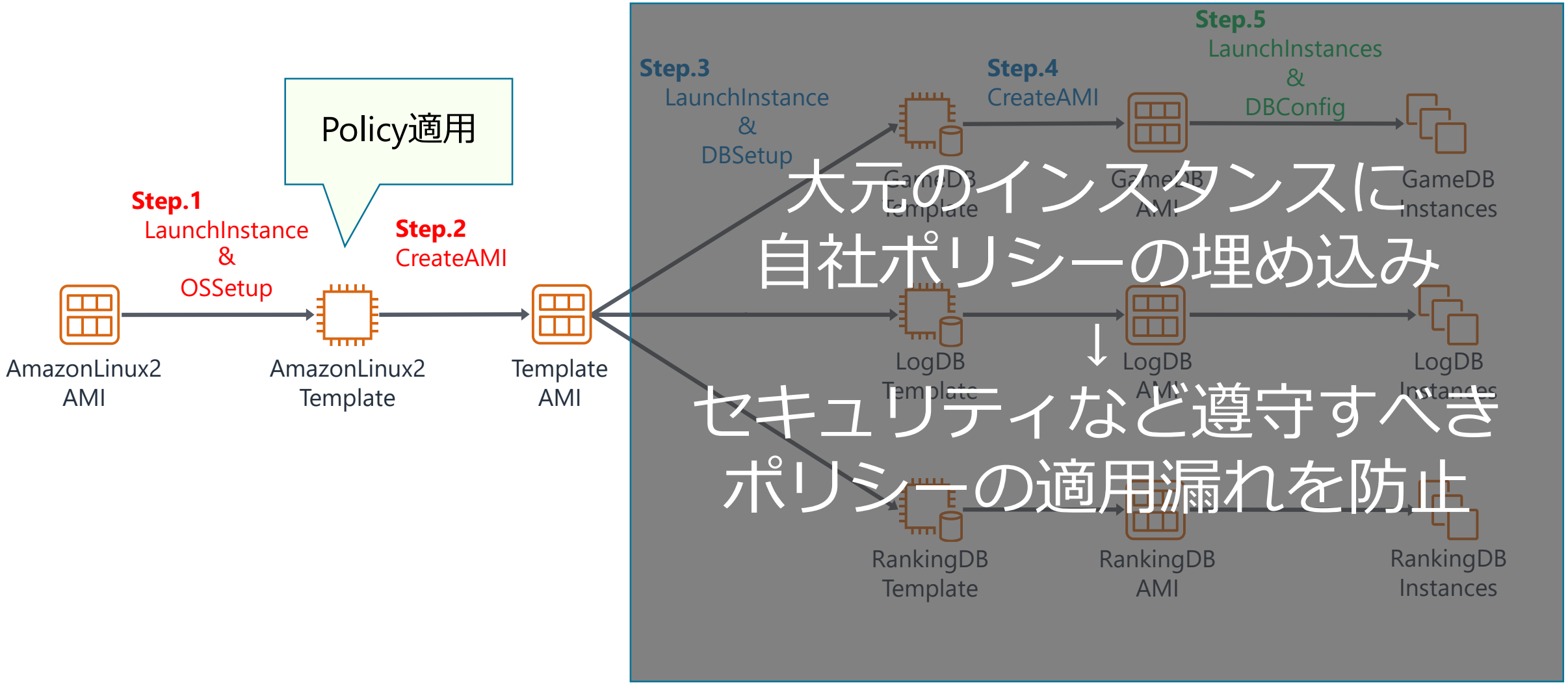
変更範囲を最小化し、難易度とメリットを考慮し一部変更

移行の流れ

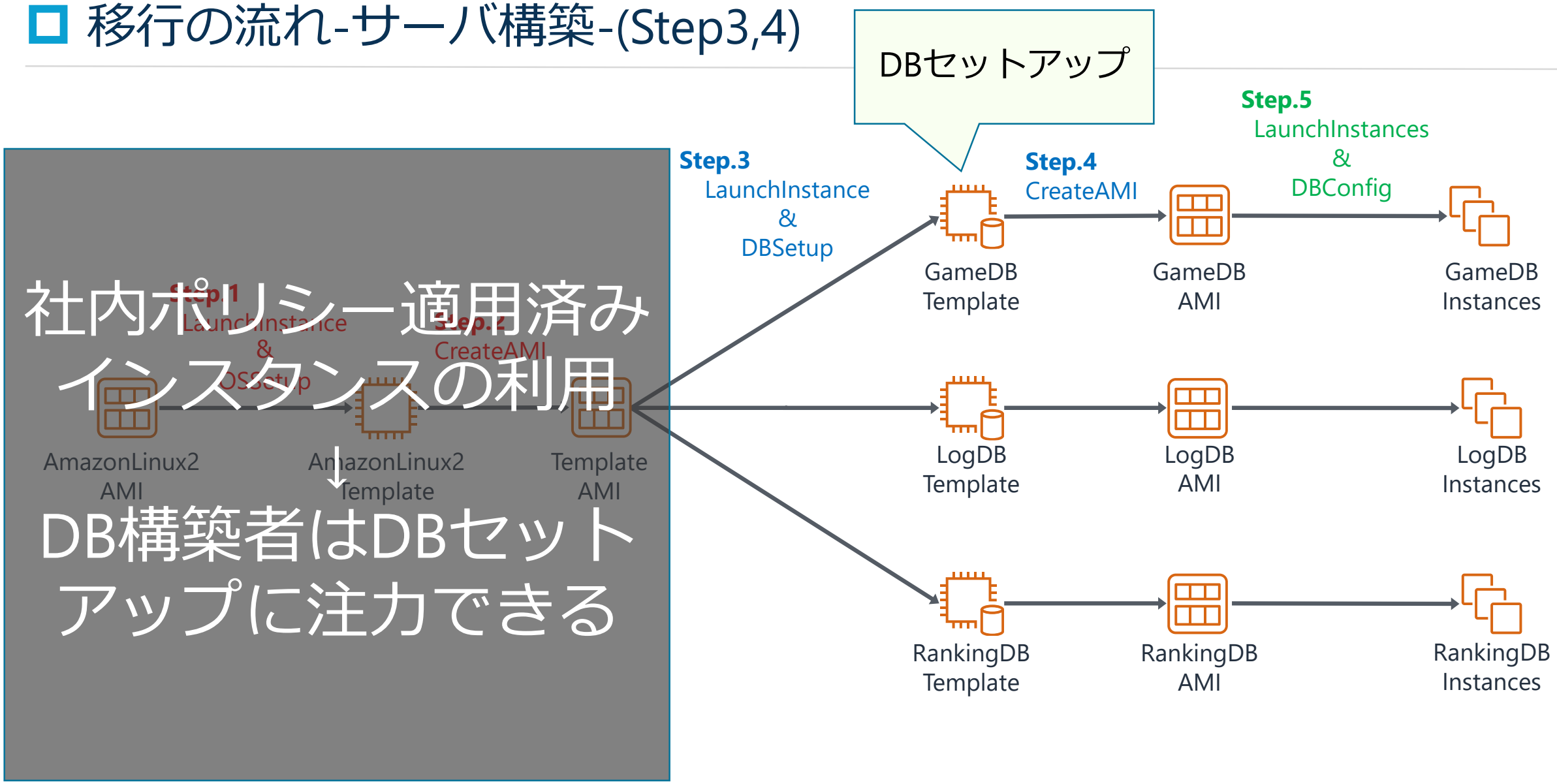
移行の流れ-サーバ構築-



移行の流れ-サーバ構築-(Step1,2)



移行の流れ-サーバ構築-(Step3,4)



移行の流れ-サーバ構築-(Step5)

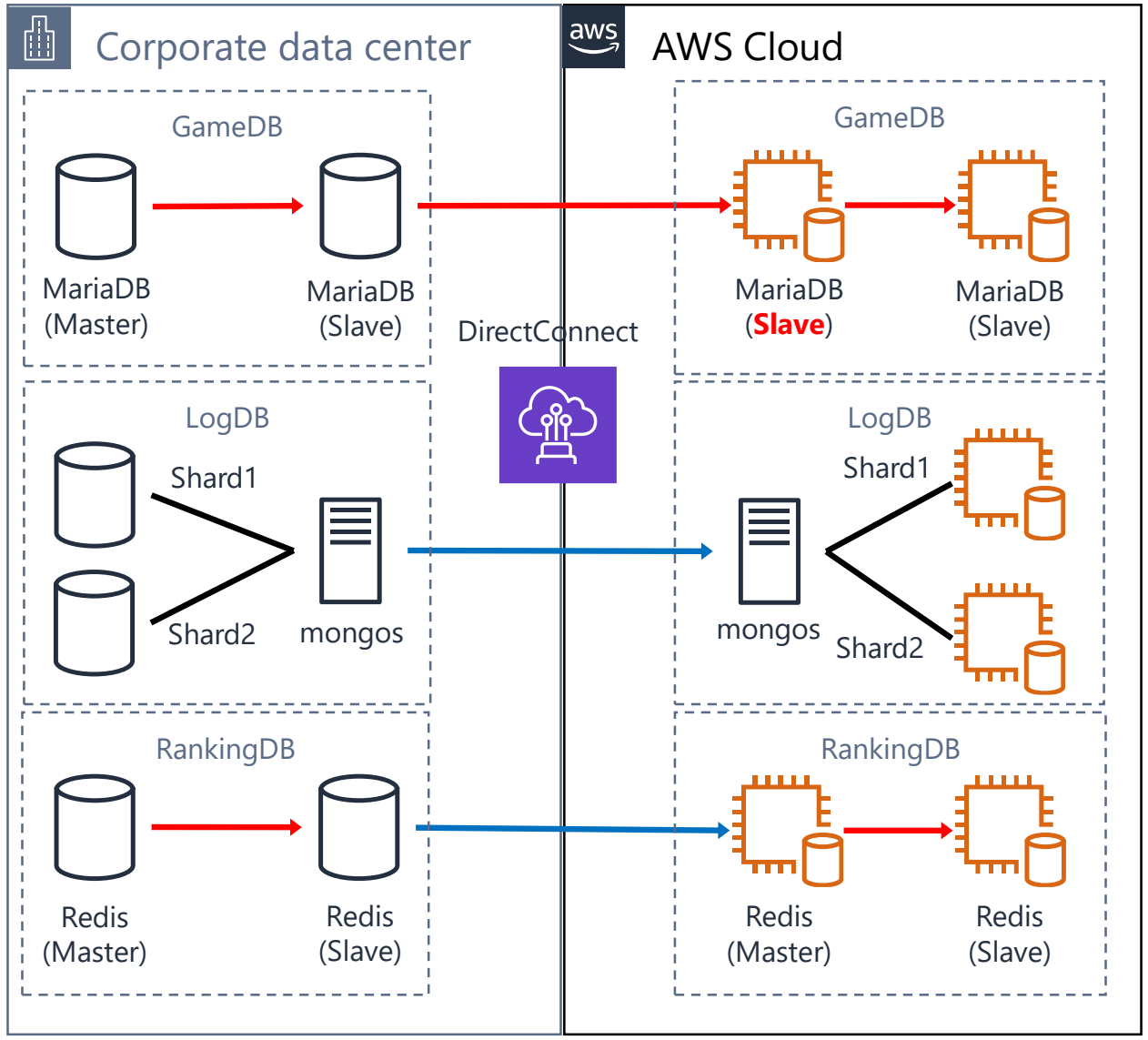
DBセットアップ済みインスタンスの利用

↓

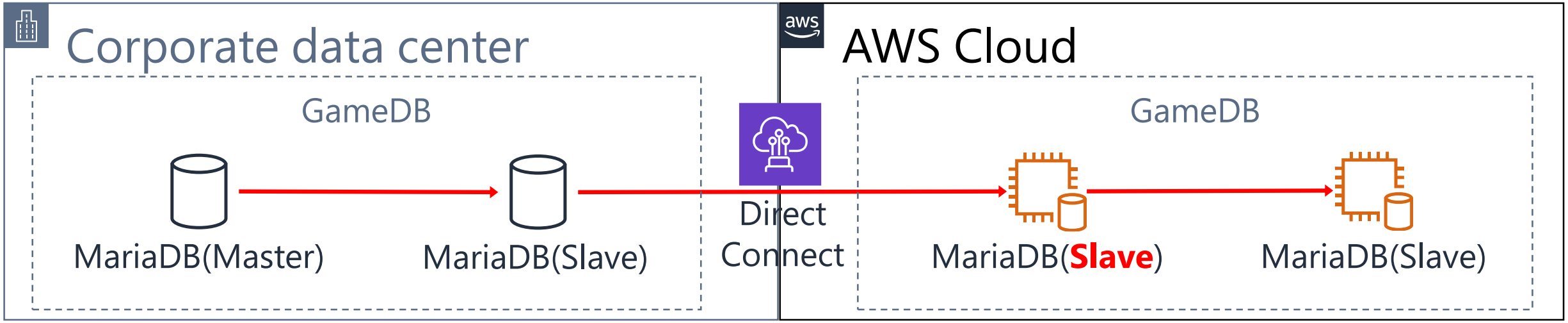
Config以外が同一設定のインスタンスを効率よく作れる

移行の流れ-3つのデータソースに対する移行方法-

- 3つのデータストアを利用。
- データソースの特性に合わせて、移行方式を決定。
- Rehost戦略に基づいて、移行先はEC2で構築する。



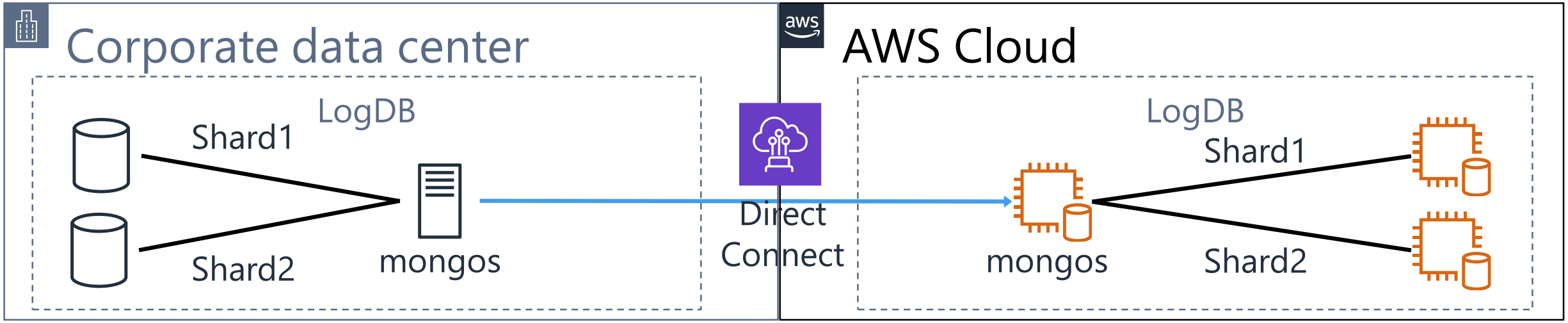
移行の流れ-3つのデータソースに対する移行方法- MariaDB



MariaDB(GameDB)

- FullBackupのデータサイズが大きい。
- オンプレミス環境と多段構成でレプリケーションを行いデータ同期。
- 切り替え当日はレプリケーションの切り離しのみでメンテナンス時間を短縮。

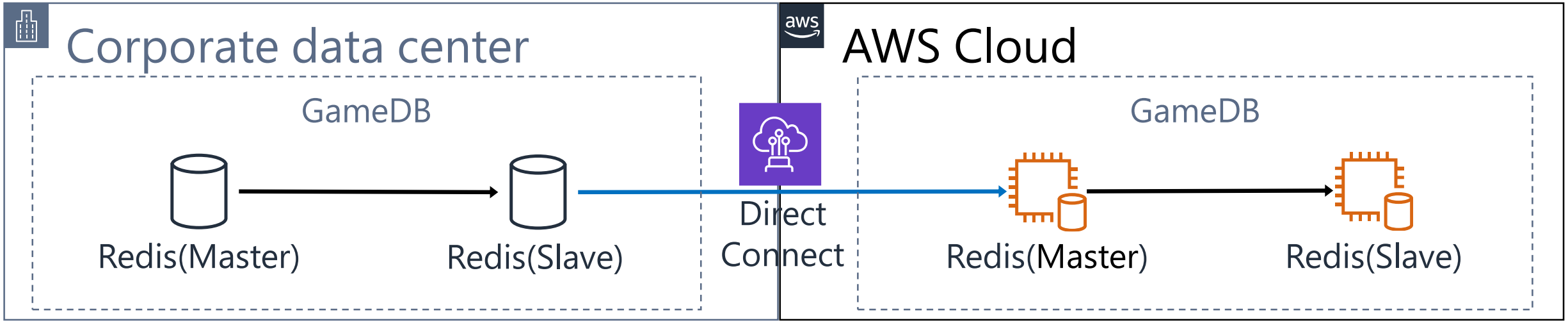
移行の流れ-3つのデータソースに対する移行方法- MongoDB



MongoDB(LogDB)

- ログデータは日別データベースで格納(LOG_20190612、LOG_20190613)
- 毎日AWS側のmongosへDumpDataをファイル転送＆リストア。

移行の流れ-3つのデータソースに対する移行方法- Redis

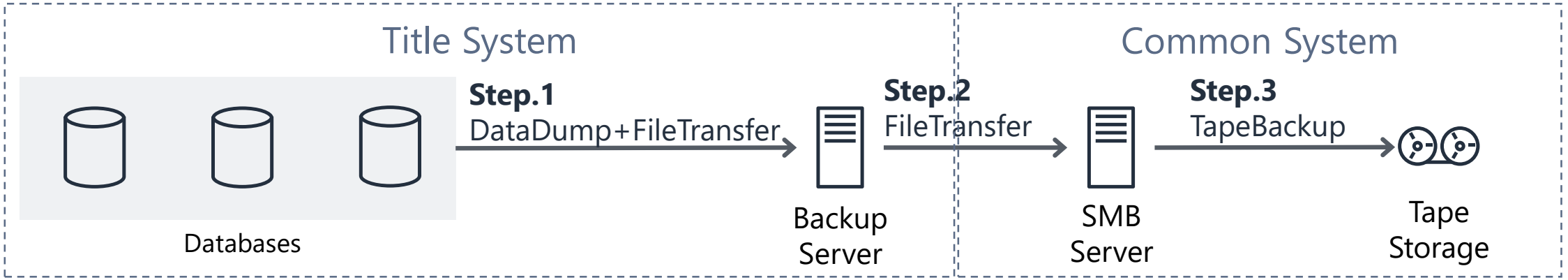


Redis(RankingDB)

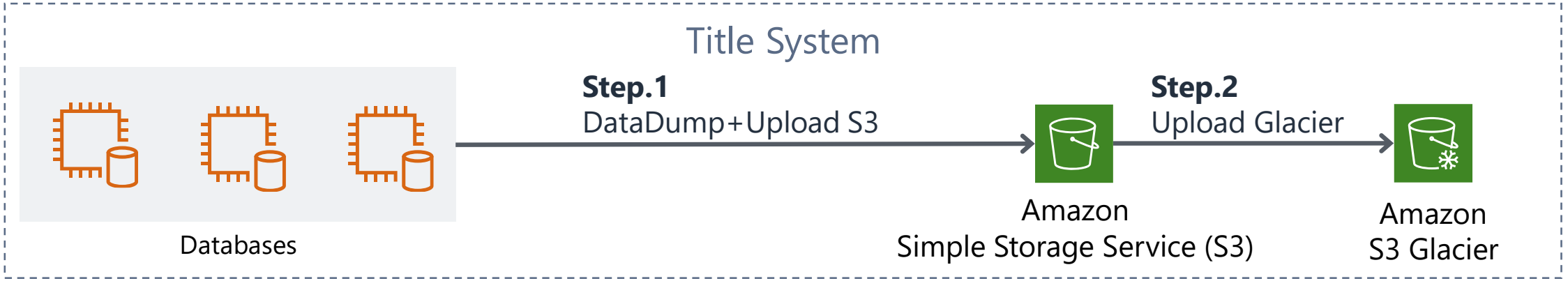
- DumpDataのサイズが小さい。
- 切り替え当日にFullDumpをファイル転送＆リストア。

移行前後でのバックアップ運用の比較

Corporate data center

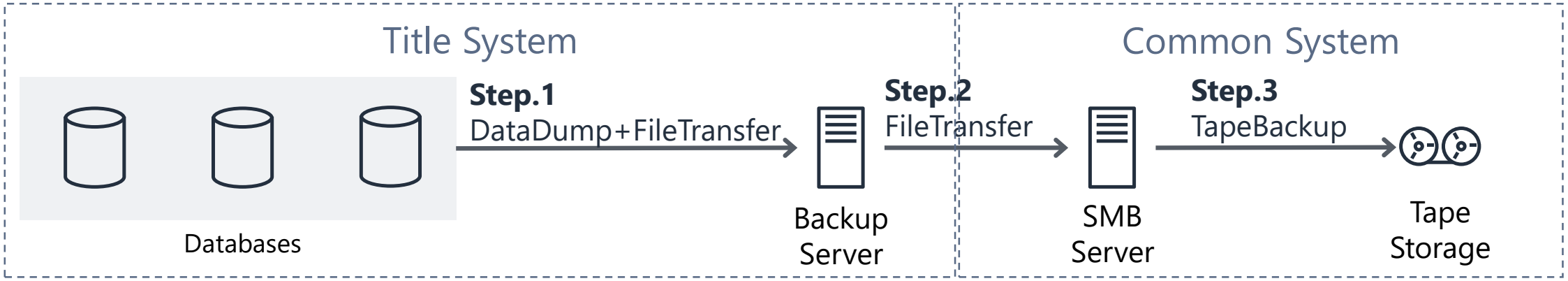


AWS Cloud



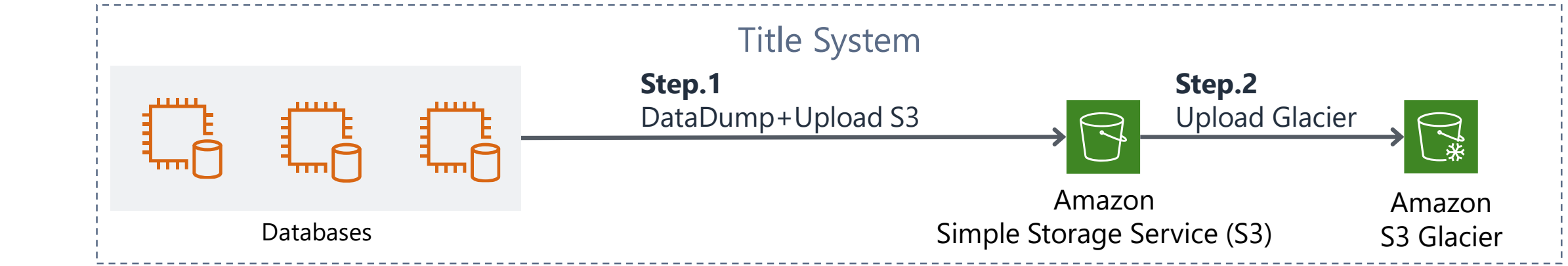
移行前のバックアップ運用の課題

Corporate data center



- テープの管理、テープ劣化への対応。
- BackupサーバやSMBサーバの運用と管理。
- オンプレミス環境全体でのバックアップスケジュール等の調整が必要。

移行後のバックアップ運用



- テープ管理からの開放。
- BackupサーバやSMBサーバが無くなることによるコストダウン。
- 一連の処理が一つのAWSアカウント内で完結し、構成がシンプルに。

移行の流れ-AWS環境への切り替え-

ユーザーアクセスを遮断してメンテナンス開始

WebServerアクセスの切り替え（ホスト名ベース）

- DNSのAレコード書き換え

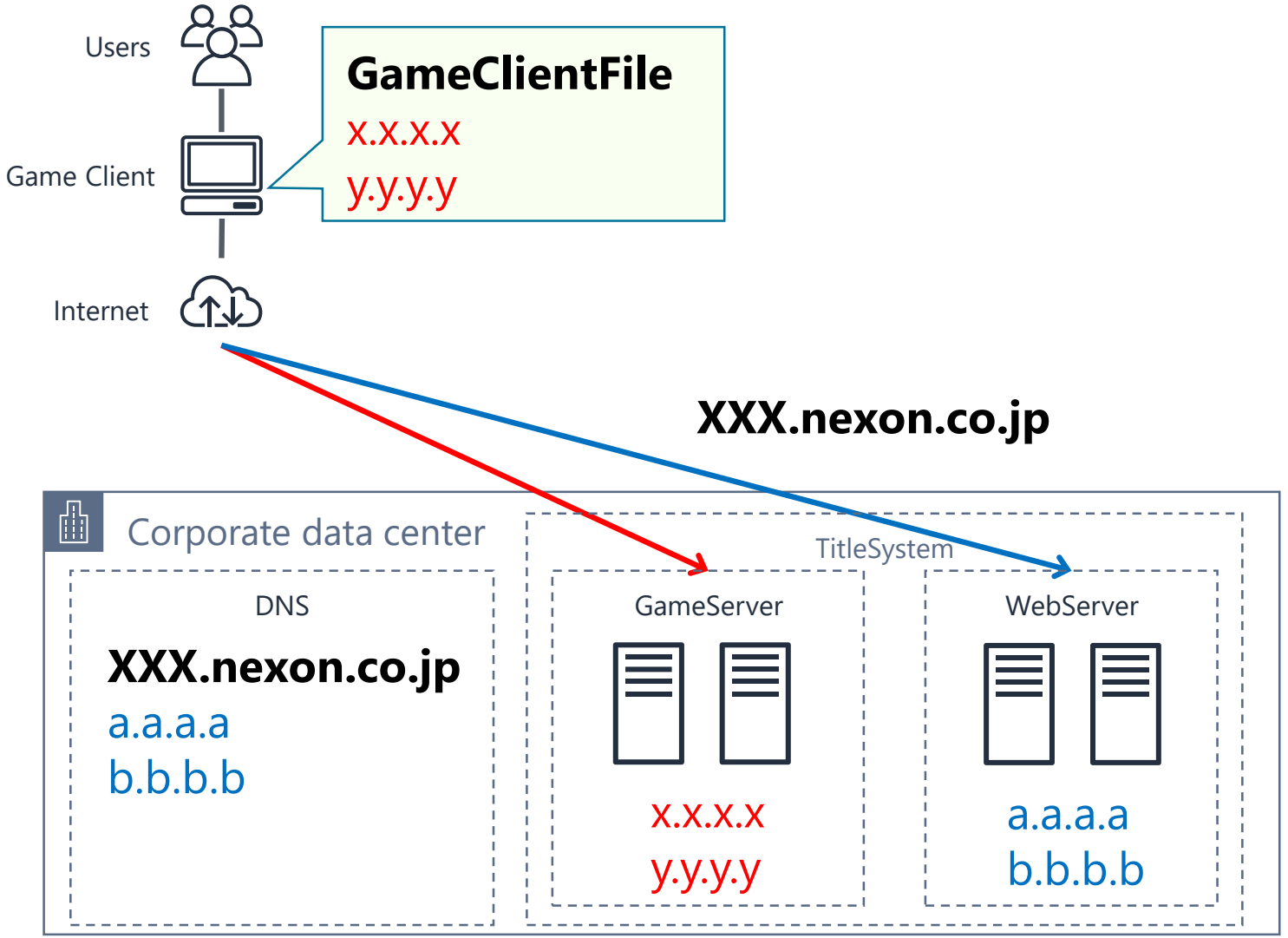
GameServerアクセスの切り替え（IPアドレスベース）

- クライアントパッチの作成＆配布による書き換え

ユーザーアクセスの遮断を解除してメンテナンス終了

既存システムへのアクセスは、ホスト名ベースアクセス以外に、IPアドレスを直接埋め込んでアクセスしていた。

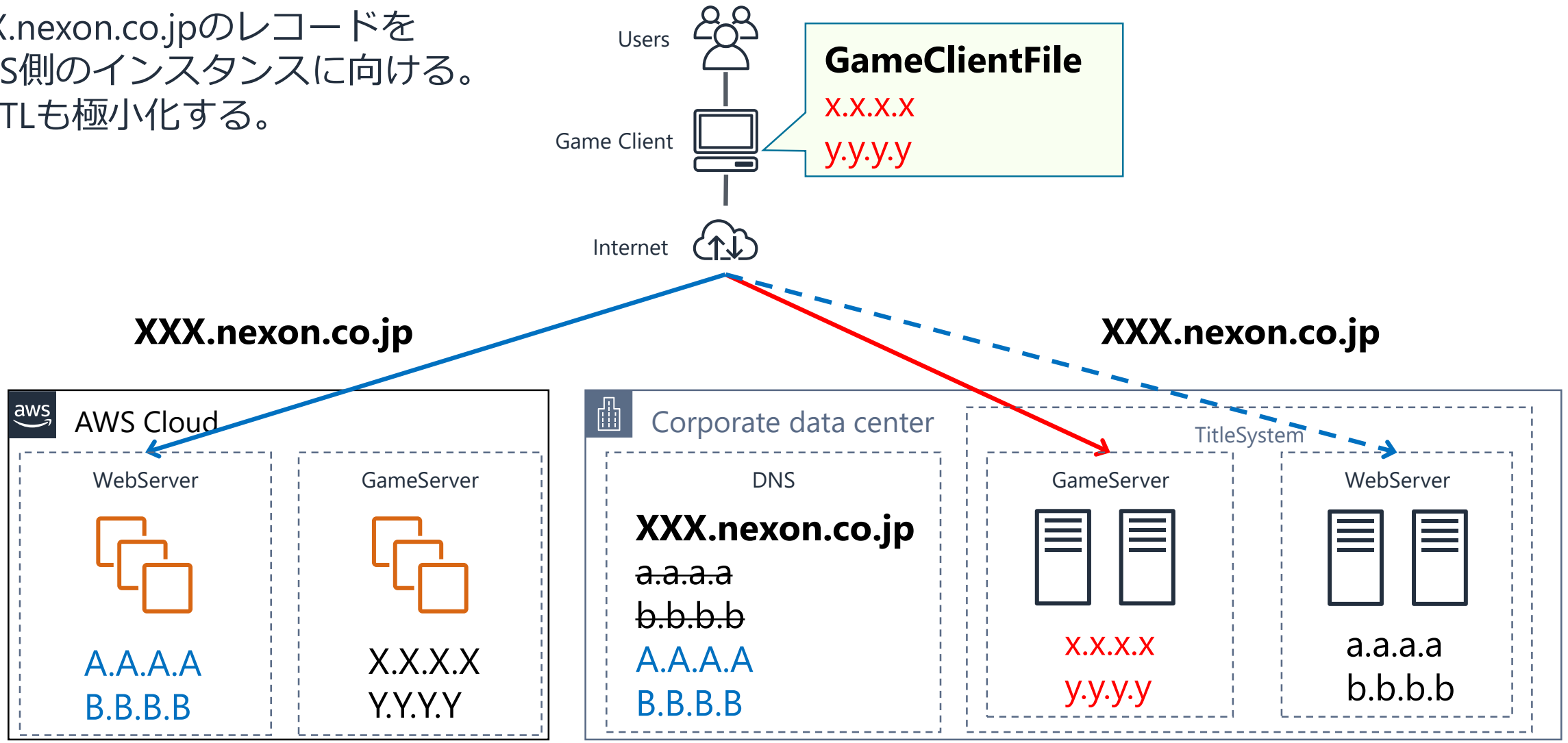
移行の流れ-AWS環境への切り替え-



移行の流れ-AWS環境への切り替え(WebServer)-

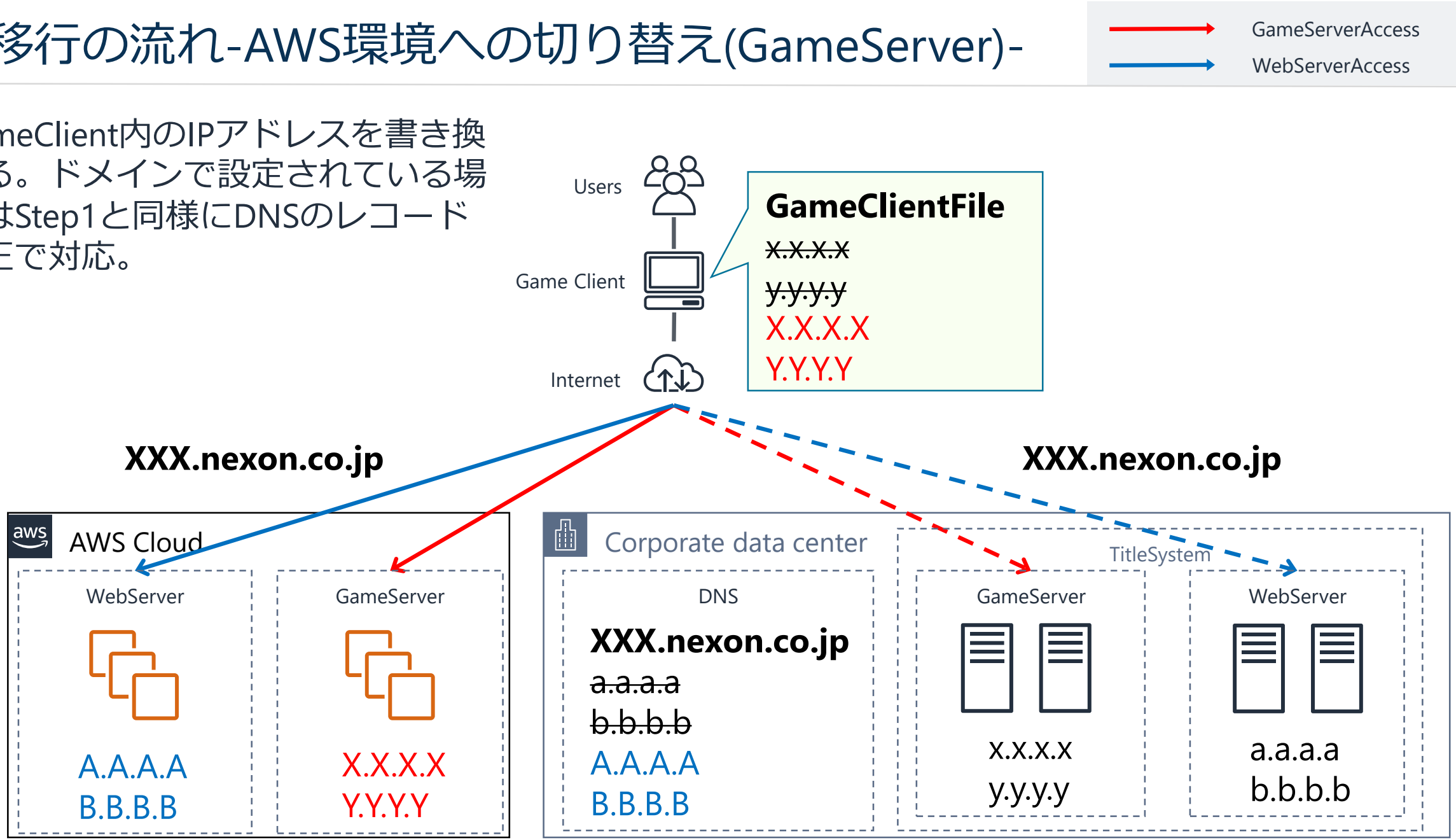
GameServerAccess
WebServerAccess

外向けDNSに登録されている
XXX.nexon.co.jpのレコードを
AWS側のインスタンスに向ける。
※TTLも極小化する。



移行の流れ-AWS環境への切り替え(GameServer)-

GameClient内のIPアドレスを書き換える。ドメインで設定されている場合はStep1と同様にDNSのレコード修正で対応。



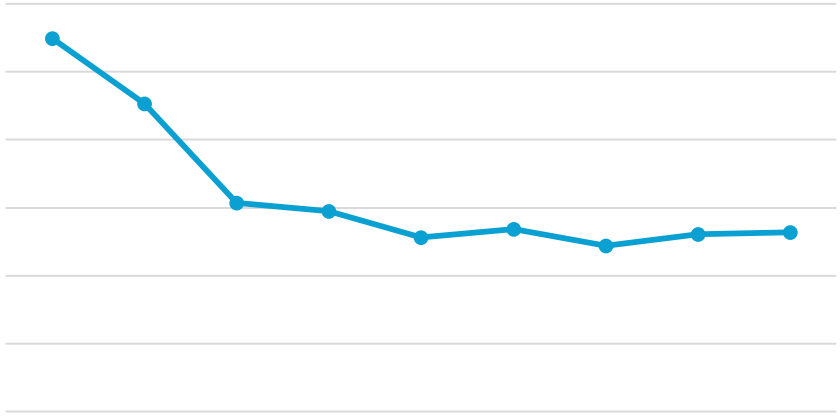
移行の結果と感想

移行の感想

- 思っていたより問題が起きなかった。
  - インフラが起因となる問題は殆ど無し。
- ハードウェア障害、リタイヤの発生頻度は決して少なくはない。
  - 単一障害点の排除が急務。
- ハードウェア障害が発生時の復旧時間が短くなった。
- サーバ設置の検討～構築完了までの期間が圧倒的に早くなった。
  - 物理機器で1ヶ月程度掛かるところを1時間程度で可能。

移行後のインフラコストについて(移行後の推移)

- 移行後1週間程度でリソース使用状況が見えてくるので段階的に最適化を実施。
- 一番効果が高かったタイトルで、移行直後と比較して半額近くまでコストを削減。
- リソースを細かく監視出来る仕組みの準備がコストに直結してくる。



移行後のインフラコストについて(オンプレミスとの比較)

オンプレミス環境ではタイトル毎のコスト算出は不可。

- 単純なインフラコストの比較はできない。
- AWSへ移行したことでタイトル毎のコストが視覚化され、担当者がよりコストを意識した対応を行えるようになった。

移行のまとめ

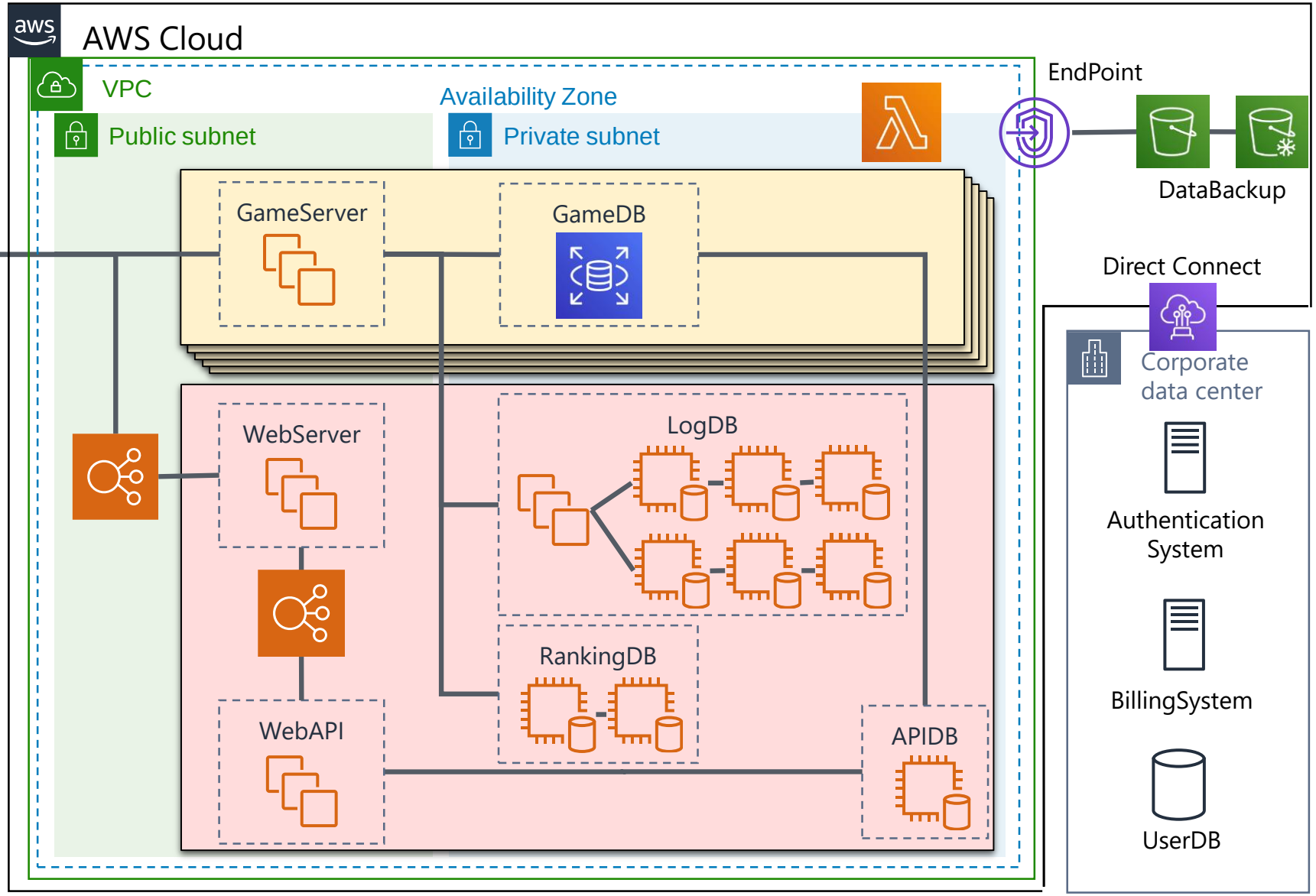
今後の課題

- リソース監視の強化
  - CloudWatchと連携させながら、Zabbix＋Grafanaで監視システムを構築。
  - ミドルウェア(特にDB系)のパフォーマンス監視を充実させる(コスト最適化)
- EC2インスタンスの細分化
  - 役割を複数持たせているサーバを1機能＝1インスタンス構成に変更(コスト最適化)。
- マネージドサービスの活用
  - Lambda、StepFunction(バッチ処理のサーバレス化)
  - DirectoryService(AD環境のサーバレス化)
  - Amazon RDS(Databaseの可用性向上)

マネージドサービスの活用

Refactor
各種マネージドサービスの活用を検討していく。

- Amazon RDS
  - GameDBの可用性向上
- Elastic Load Balancing (ELB)
  - 共通システムのAWS移行に合わせた負荷分散の最適化
- AWS Lambda
  - 各種バッチ処理の置き換え



まとめ

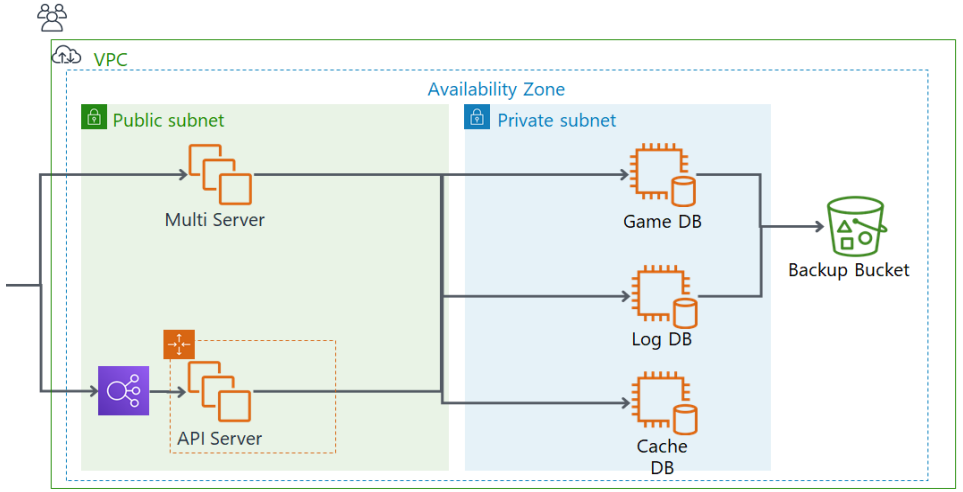
- 無事に全PCタイトルのAWS移行が完了。
- オンプレミス環境からの移行完了がゴールではない。

†クラウドオリエンテッドに設計したモバイルゲームタイトルの構築フェーズからリリース／運用で押さえるべきポイント！

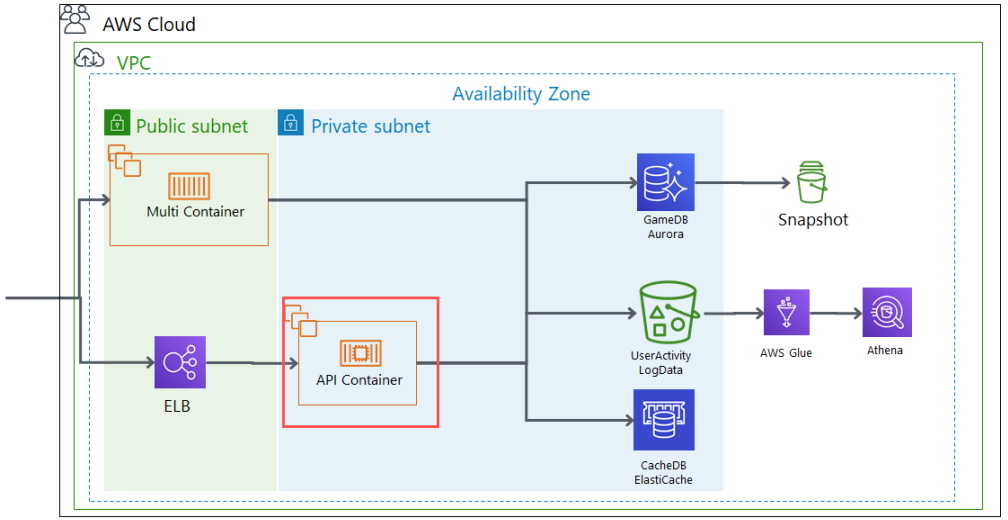
モバイルゲームサービスでのAWS活用について

ネクソンジャパンのモバイルゲームにおけるAWS活用

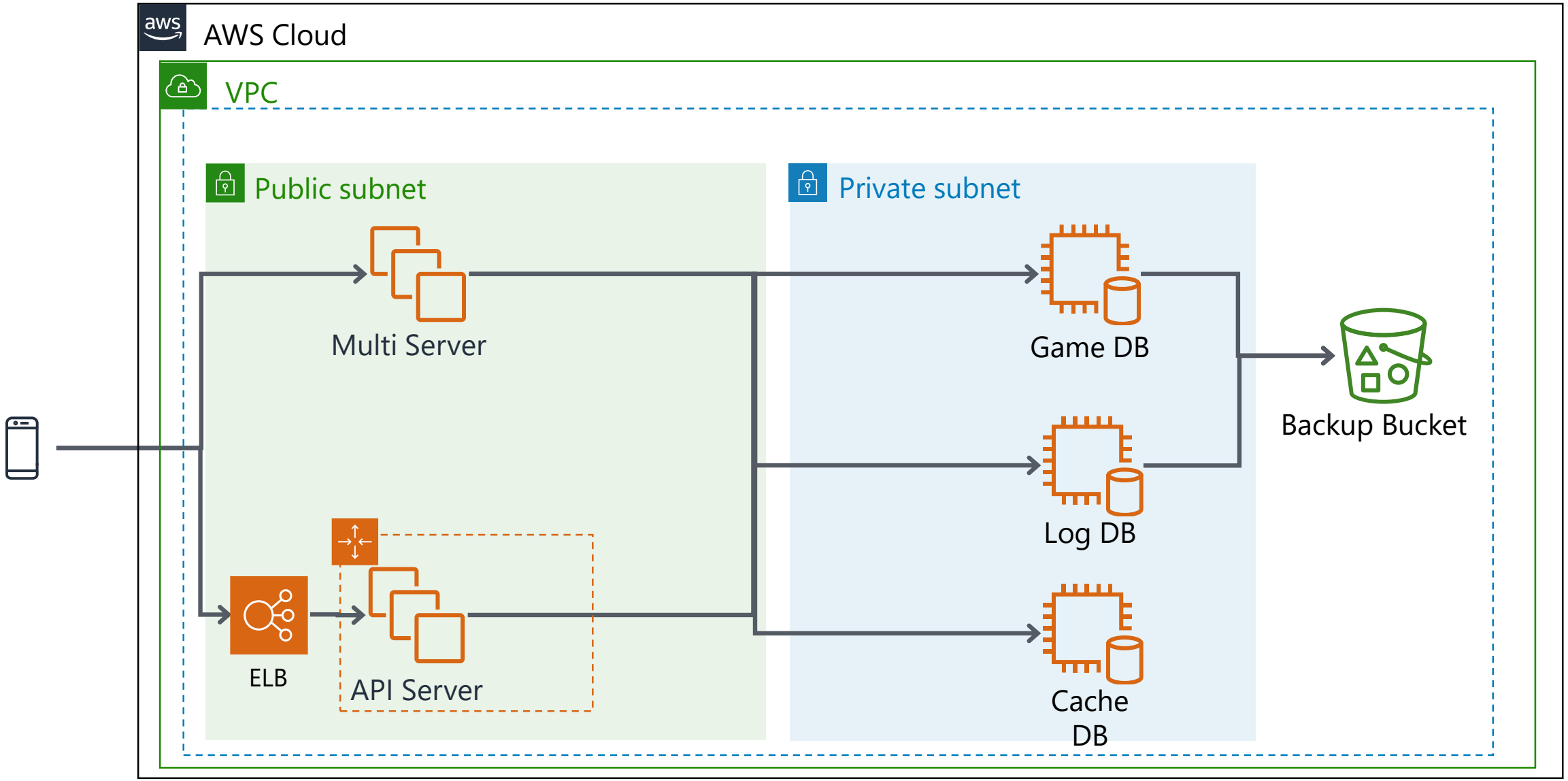
- 安定は何より大事
  - 最初のAWS利用は2014年から
  - EC2インスタンスをIaaS的に使用
  - オンプレミスと同レベルの管理が目標
  - マネージドサービスは、
    サービス影響がないシステムで運用



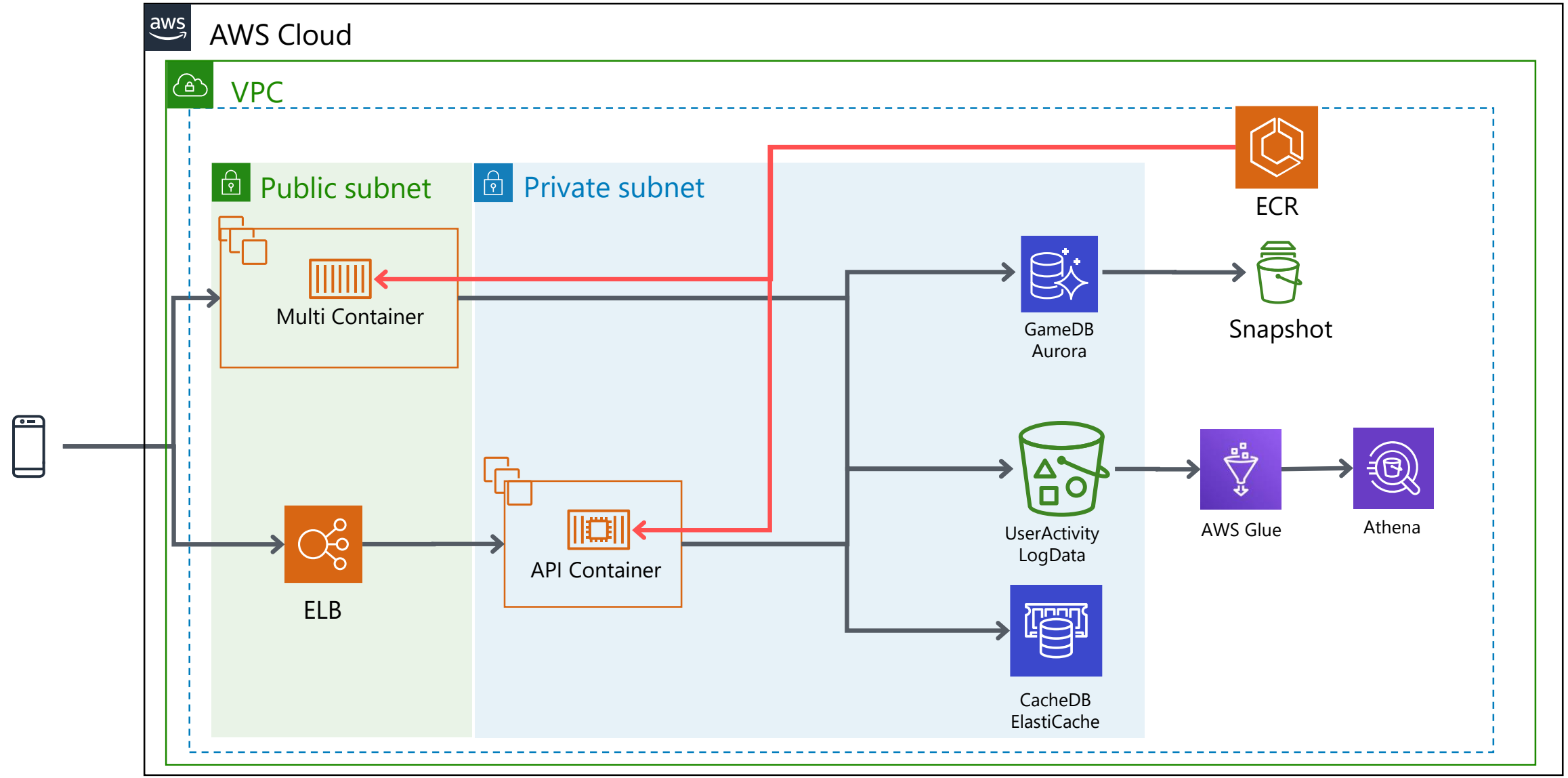
- 安定後は効率的な運用・費用の最適化
  - 2016年以降はマネージドサービスを活用
  - グローバルサービスを含むナレッジ共有の促進
  - CloudTrail などの証跡ツールの拡充
  - データレイク要件の需要への対応（S3＋Athena）



AWS運用を主軸にして変化したこと（2014年～）



AWS運用を主軸にして変化したこと（最近）



クラウドネイティブなアーキテクチャ

Amazon CloudWatchを利用した、システム運用状態のモニタリング

CloudTrail等によるロギング + Athenaでのログ分析

IAMによるアクセスコントロール

AWS Shield 等のDDoS対策

AutoScalingやRDS Clusterによる冗長性確保

EC2 Eventによる自動復旧

CloudWatch・Zabbix・Grafanaを活用したリソース監視と適正化

TrustedAdvisorを参考にした環境の最適化 / CostExplorerを活用しリソースの把握・適正化

RI / Spotインスタンスの活用

https://aws.amazon.com/jp/architecture/well-architected/

安定運用の為の工夫

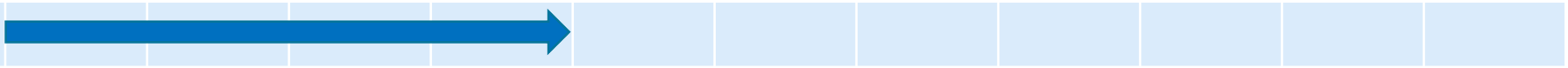
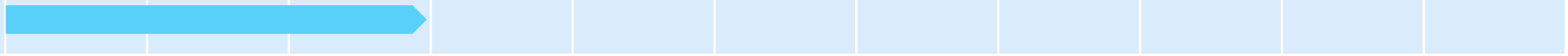
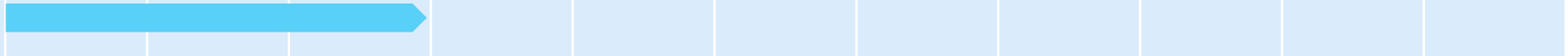
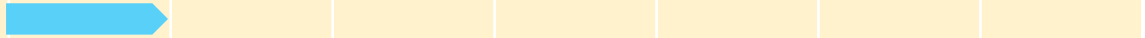
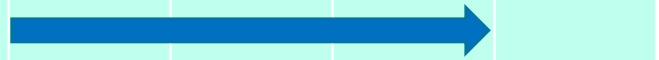
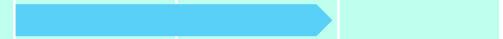
ゲームリリースまでの流れ

構成検討	• 必要な仕様の把握 • アーキテクチャの検討・協議
動作検証	• 検証環境の構築 • 動作検証
構成修正	• 問題点の変更・修正 • 基本動作の確認
性能テスト	• 負荷テスト • 冗長性テスト
リリース	• リリーススペック確定

性能テストを重視

- 初めて導入するマネージドサービスは、想定通りの挙動を示すか確認
- デプロイやフェイルオーバーなどの運用時を意識したテストも本フェーズで実施
- 本番時のコスト試算および拡張計画を立てる

性能テスト(概要)

作業項目	スケジュール感
1. 単体テスト	
API開発	
BOT開発	
単体APIテスト/再テスト	
2. シナリオテスト	
シナリオ作成	
シナリオテスト/再テスト	
3.冗長性テスト	
テスト項目作成	
冗長性テスト/再テスト	
4. 結果の反映	
レポート作成	

性能テスト(Step1：単体テスト)

高負荷下での単体機能確認

ボトルネックの把握

インスタンス系統選定

性能テスト(Step2：シナリオテスト)

スケールアウト動作確認

スケーリング箇所の限界確認

高負荷下のプレイアビリティ

性能テスト(Step3：冗長性テスト)

フェイルオーバー動作の確認

障害発生時のプレイアビリティ

性能テスト(Step4：テスト結果のインフラ反映)

リリーススペックの確定

概算コストの算出

対応計画の策定

エンタープライズサポートの活用

エンタープライズサポートの活用について

- 構築/運用フェーズにおけるTAM との連携
- Trusted Advisor を活用した運用改善

	ベーシック	開発者	ビジネス	エンタープライズ
AWS Trusted Advisor	4 項目	4 項目	すべての項目	すべての項目
技術サポートへのアクセス	ヘルスチェックのサポート	メール（平日9:00 - 18:00)	電話、チャット、メール（24時間年中無休）	電話、チャット、メール（24時間年中無休）、**TAM**
AWS サポート API			○	○

https://aws.amazon.com/jp/premiumsupport/compare-plans/

テクニカルアカウントマネージャー（TAM）

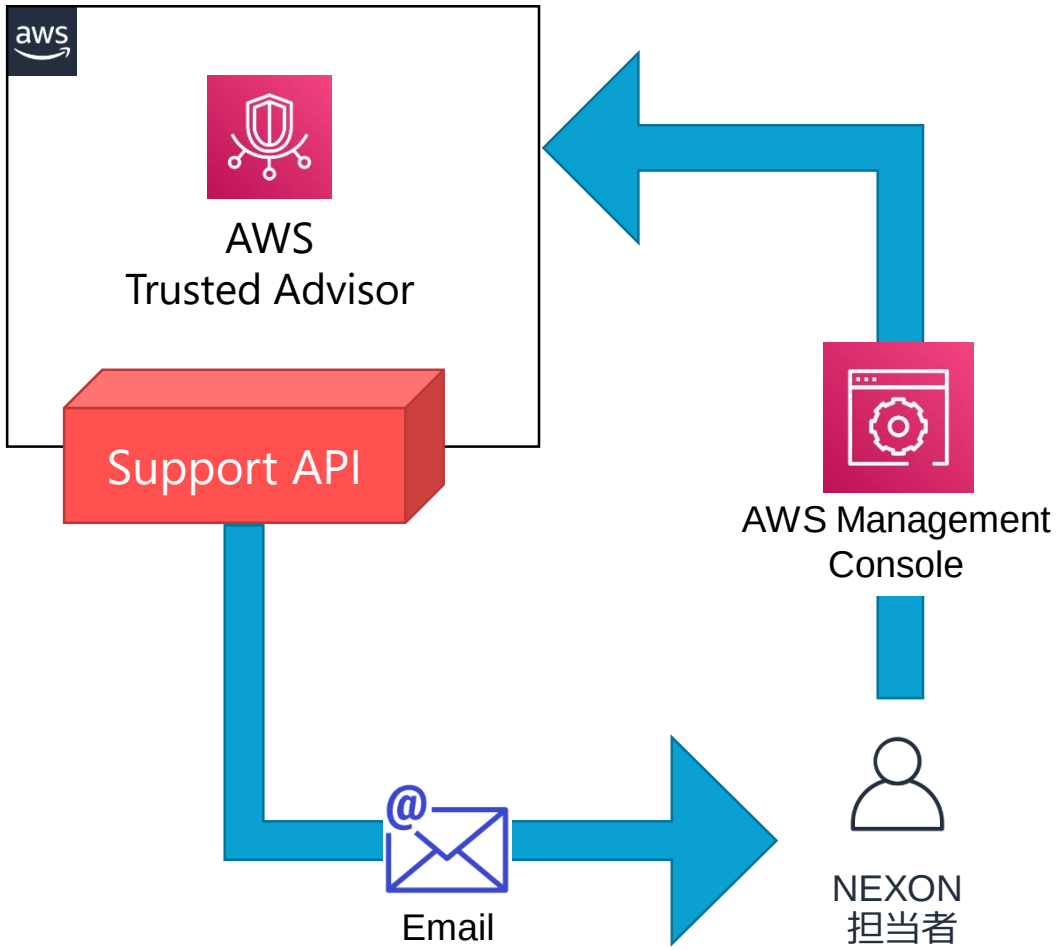
構築フェーズ

- ユースケースやベストプラクティスの案内
- TAM＆SAとの要件に沿ったソリューション計画の相談

運用フェーズ

- トラブルケースへのフォロー
- 運用課題に対する解決策の相談

Trusted AdvisorとサポートAPIの活用



- Trusted Advisor
  - 月に一度の定期チェックをチーム内で運用ルール化
  - 検出された項目を元に今後の改善内容を検討
- サポートAPI
  - 不要リソースの通知
    (EC2、EBS、Etc...)
  - セキュリティ警告の通知
    (IAM利用状況、SGのAnyPermit等)
  - サービス上限緩和要否の通知

まとめ

- 最初から全てをマネージドサービスで最適化するのは必須ではない。
  - 徐々に使用して慣れていくのも大事。
- 負荷テストは思いつく限りしっかり行う。
  - ユーザビリティ、スケーラビリティ、コストマネジメント…色々なものが見えてくる。
  - 初めて使用するマネージドサービスは、性能テストを通じて注意するポイントを確認。
- エンタープライズサポートは障害対応など「事後」のサポートではない。
  - 色々な協力を受けて「事前」に障害を起こさない仕組みを模索していく。

ご清聴ありがとうございました。