API型ゲームサーバーのサーバーレス化と安定運用

Shuhei Kimura
アマゾン ウェブ サービス ジャパン株式会社
Solutions Architect

aws SUMMIT ONLINE

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

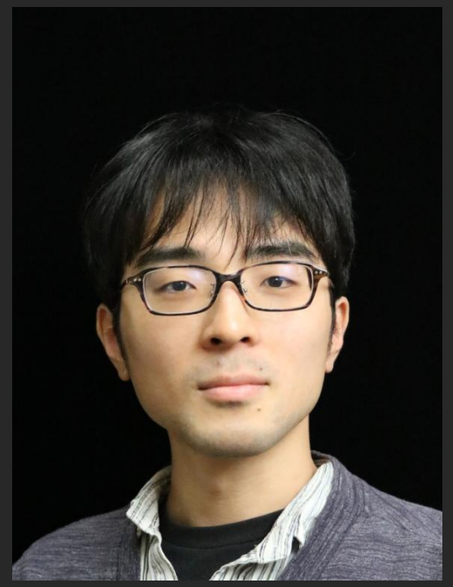
自己紹介

氏名: 木村 秀平 (Shuhei Kimura)

役職: ソリューションアーキテクト（SA）

経歴:

学生 → ゲーム開発 → Web開発 → 学生 → システムトレーダー → 現職



好きなAWSサービス:

AWS Lambda

Amazon S3

本セッションの内容

想定視聴者

- 現在のゲームサーバーがサーバーレス化できるか知りたい
- ゲームサーバーをサーバーレスで構築している（したい）

前提知識

- AWS Lambdaを使ったことがある

ゴール

- AWS Lambdaの**アップデートを把握**し、現在のシステムに適用可能か**検討**できるようになる
- 安定稼働のため注意すべきことを認識する

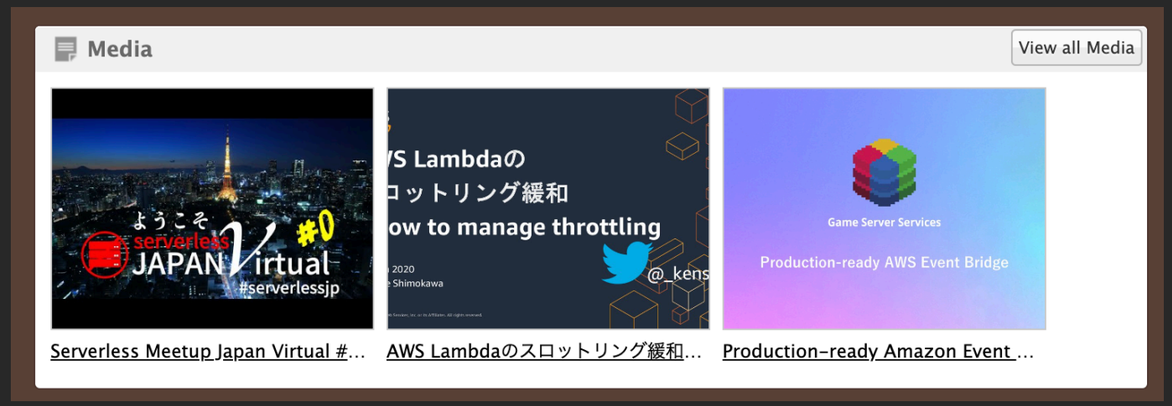
本セッションで扱わない内容

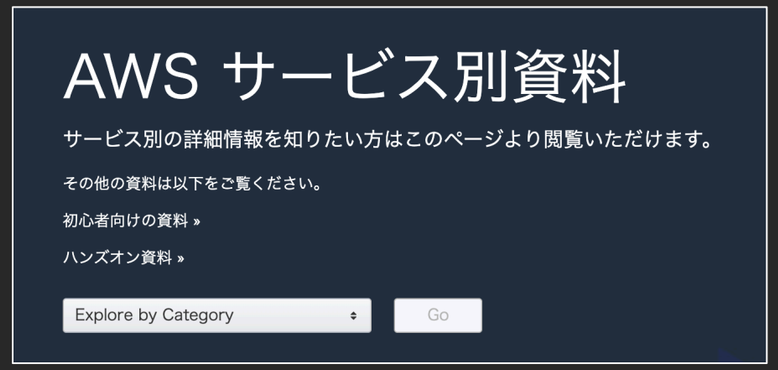
各サービスの基礎・詳細

実装方法の詳細

関連セッション・資料

- AWS Summit Online 2020: Let's dive deep into AWS Lambda Error Handling
- Serverless Meetup Japan Virtual
- AWS サービス別資料





このセッションで説明すること

API型のゲームサーバーではサーバーレスサービスを検討する価値が高い

サーバーレスサービスは機能追加で利用しやすくなっている

- 特にここ1-2年のアップデートはゲームサーバー向き

アクセスの増加（特に急増）で規模が大きくなると対策を検討

- スロットリング: 上限緩和、Provisioned Concurrencyなどで対処
- コスト最適化: Compute Savings Plansも利用可能

リトライ処理や負荷試験はサーバーレスでも必要

上手に使って良いサーバーレスライフを！

Agenda

サーバーレスサービスの簡単な紹介、そのメリット

サーバーレスゲームサーバーの事例

ゲームサーバーの特徴

サーバーレス化の課題と対処

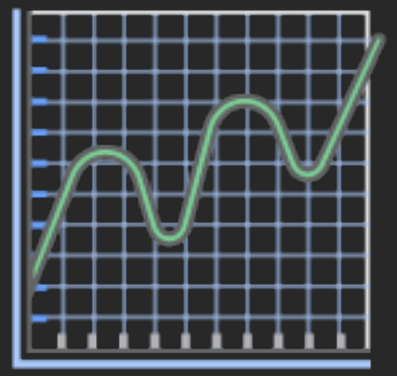
サーバーレスサービスの簡単な紹介、そのメリット

aws SUMMIT ONLINE

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

サーバーレスアプリケーションの特長

サーバー管理が不要



柔軟なスケーリング

アイドル時のリソース確保不要

高可用性

開発者とAWSの役割分担

オンプレミス	EC2	サーバーレスアーキテクチャ
アプリケーション作成	アプリケーション作成	アプリケーション作成
スケールアウト設計	スケールアウト設計	スケールアウト設計
定形運用設計	定形運用設計	定形運用設計
ミドルウェアパッチ	ミドルウェアパッチ	ミドルウェアパッチ
ミドルウェア導入	ミドルウェア導入	ミドルウェア導入
OSパッチ	OSパッチ	OSパッチ
OS導入	OS導入	OS導入
HWメンテナンス	HWメンテナンス	HWメンテナンス
ラッキング	ラッキング	ラッキング
電源・ネットワーク	電源・ネットワーク	電源・ネットワーク
開発者が担当	AWSが担当	

ゲームサーバー開発で考えられるメリット

設計検証から本番導入までの期間短縮

検証時の構成から、そのままスケール

キャパシティ管理、運用管理の負荷低減

アイドル時のコスト削減

構成変更なしに、ヒットしていない時期を低コストで運用可能

サーバーレスなアプリケーションモデル

イベントソース　　ファンクション　　サービス呼び出し

例：
- HTTPSによるリクエスト
- S3にオブジェクトが作られる
- Kinesisにストリームデータが保存される
- etc…

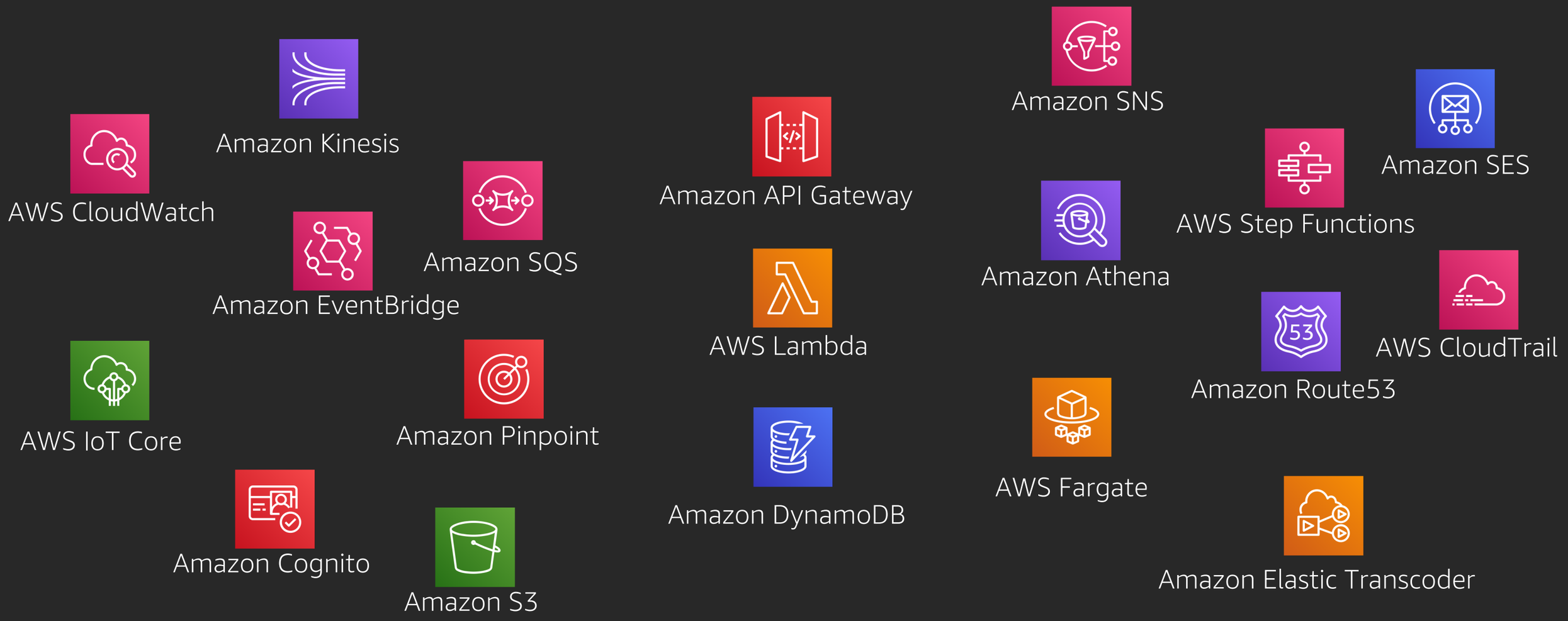
標準ランタイム
- Python
- Node.js
- Java
- Ruby
- Go
- .NET Core

+カスタムランタイム

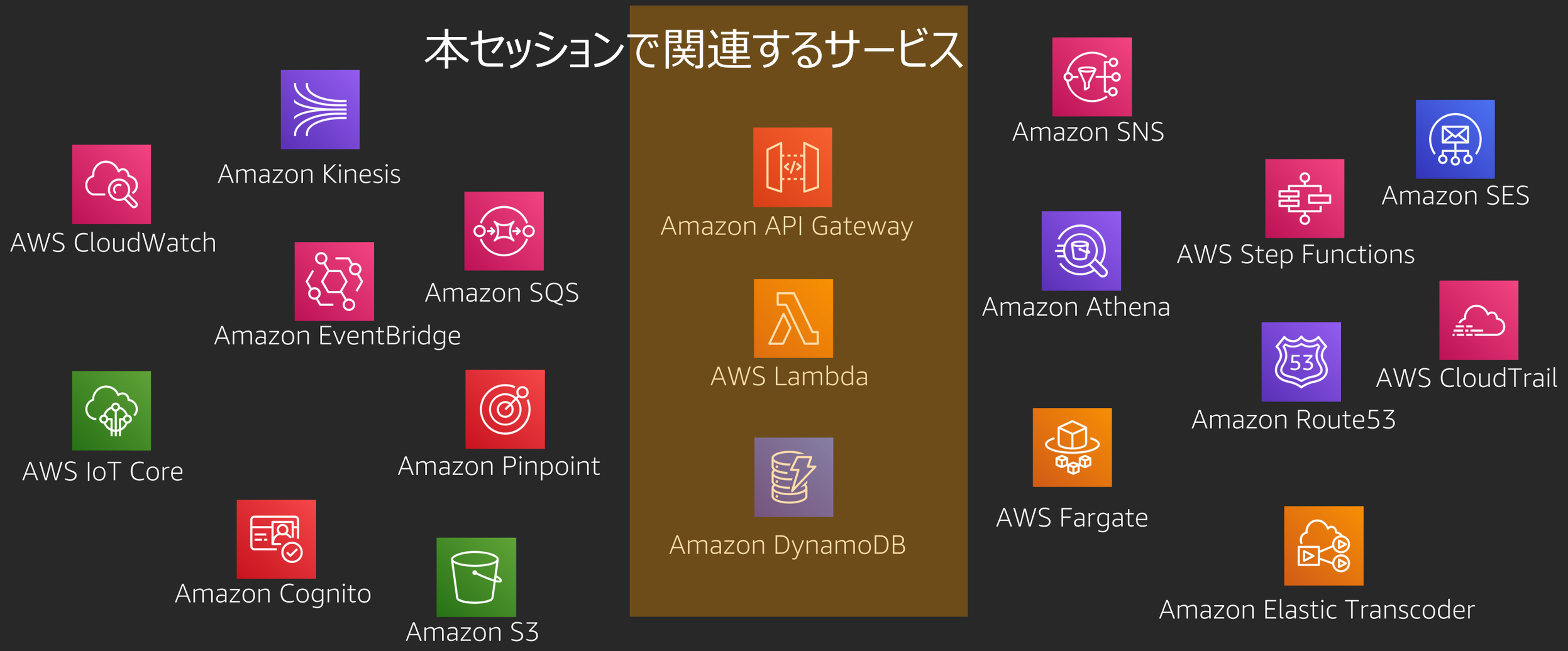
例：
- データベース読み書き
- ファイル操作
- 通知

https://www.slideshare.net/AmazonWebServicesJapan/20200610-aws-black-belt-online-seminar

AWSが提供するサーバーレスサービス（一部）



AWSが提供するサーバーレスサービス（一部）



事例紹介

aws SUMMIT ONLINE

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Joyient

スタートアップで人的リソースがあまり無かった

保守・監視の専門家を採用する人件費が割合として高かった

2018年の初め、サーバーレスアーキテクチャに完全に切り替えることを決定

サーバーレス化後は安定・柔軟

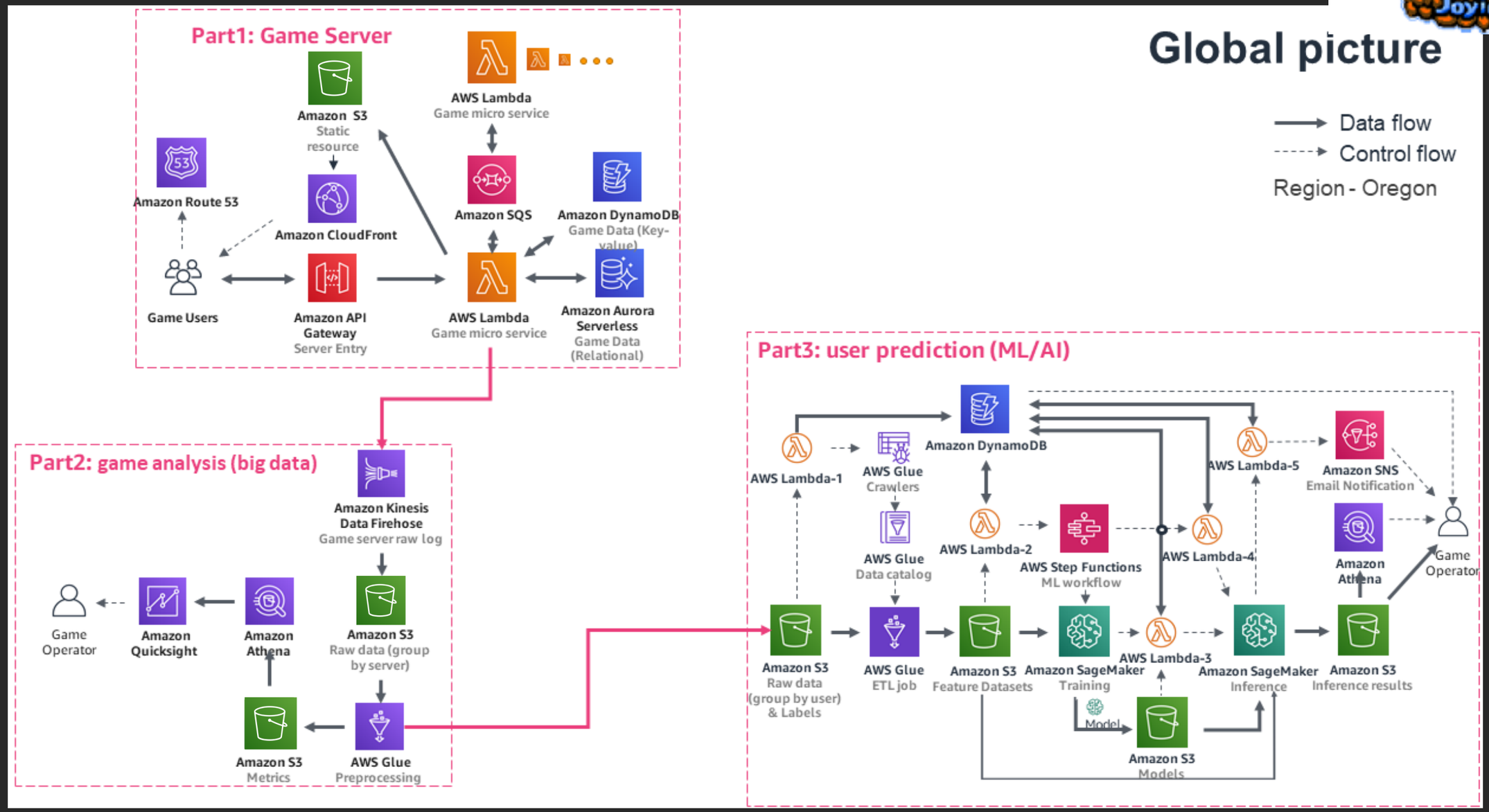
ゲームの中断はほとんどない。可用性は99.999%を超える

より柔軟なキャパシティ管理（EC2インスタンスの数 → 消費ユニットの数）

「AWSのホスティングサービスと高度なサービスを使用して、Joyientは製品開発とテストのサイクルを5～6か月から3～4か月に短縮し、リリース前のサーバー構成テストを30分から数秒に改善しました。」

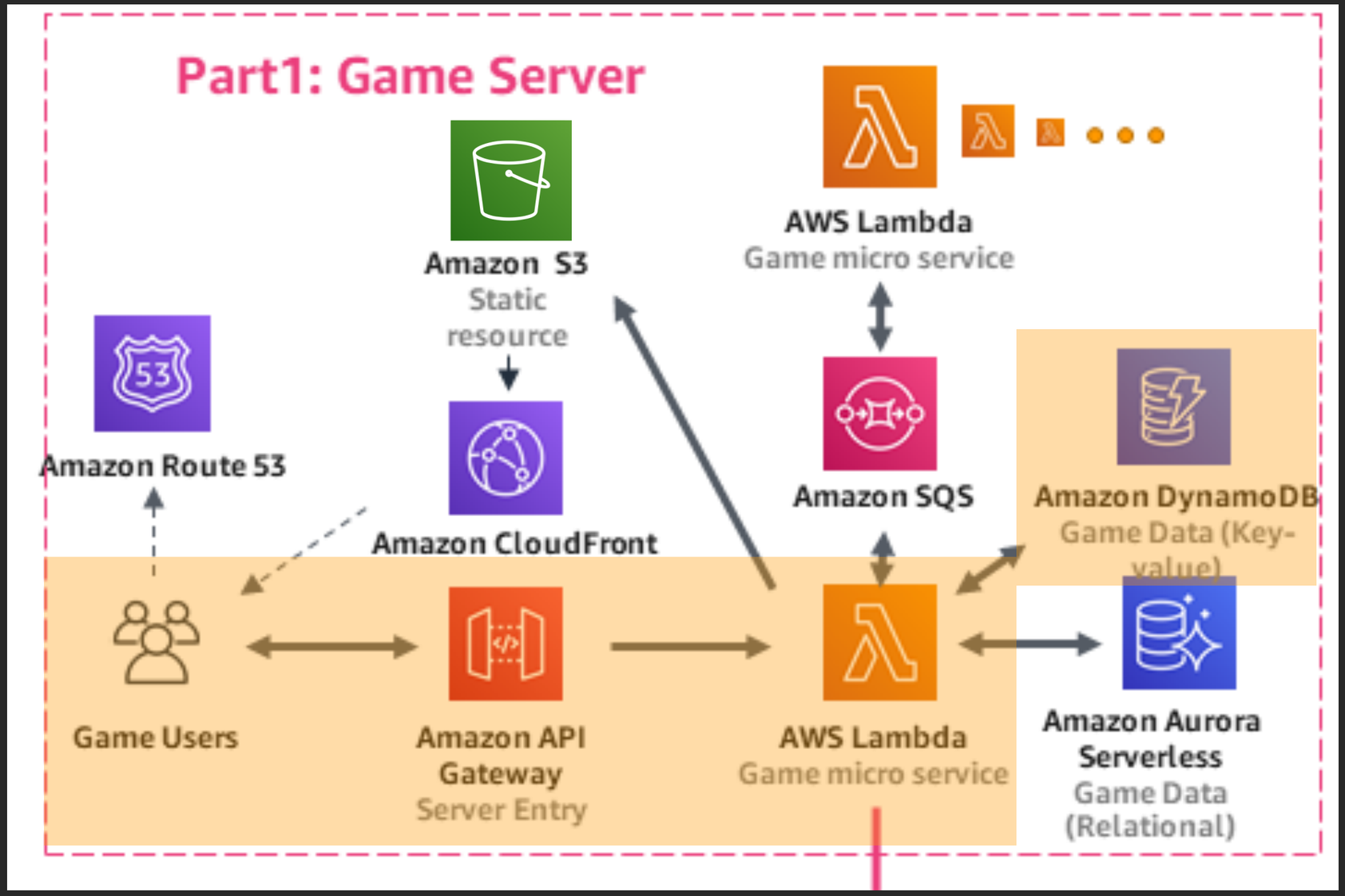
https://aws.amazon.com/cn/solutions/case-studies/joyient/

Joyient



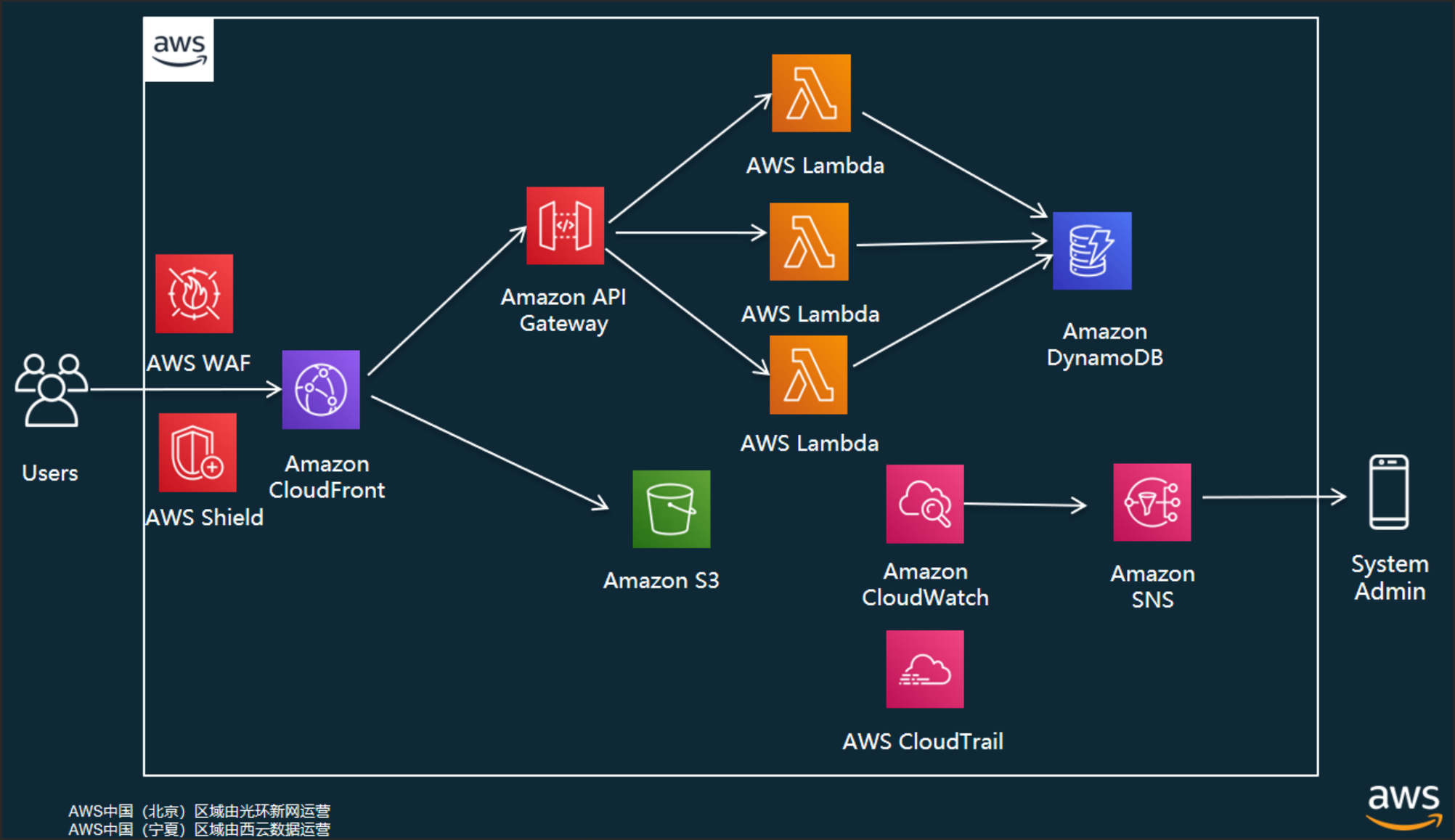
https://aws.amazon.com/cn/solutions/case-studies/joyient/

Joyient



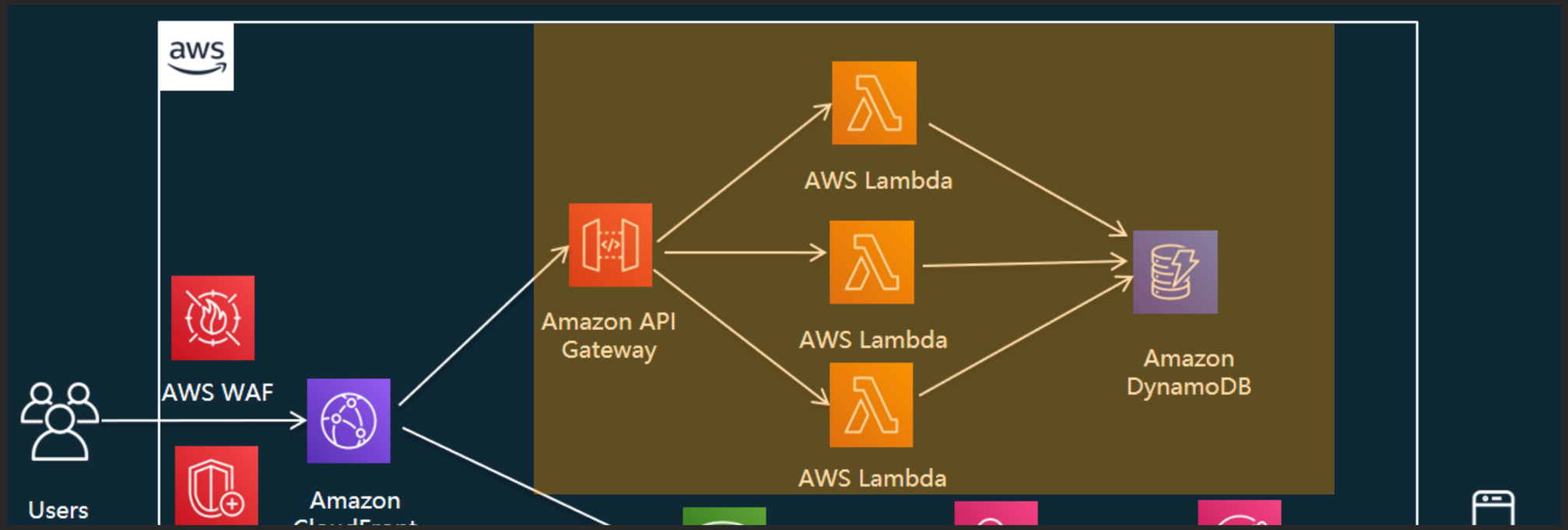
https://aws.amazon.com/cn/solutions/case-studies/joyient/

HABBY



https://aws.amazon.com/cn/solutions/case-studies/habby/

HABBY



「サーバーレスアーキテクチャは、サーバーのプロビジョニングや管理なしでコードを実行できます。これはゲームビジネスに非常に適しており、新しいゲームの展開を大幅に短縮できます。」

https://aws.amazon.com/cn/solutions/case-studies/habby/

Game Server Services (GS2)

ゲームサーバーに必要な機能をAPIサービスとして提供

フルサーバーレスの構成

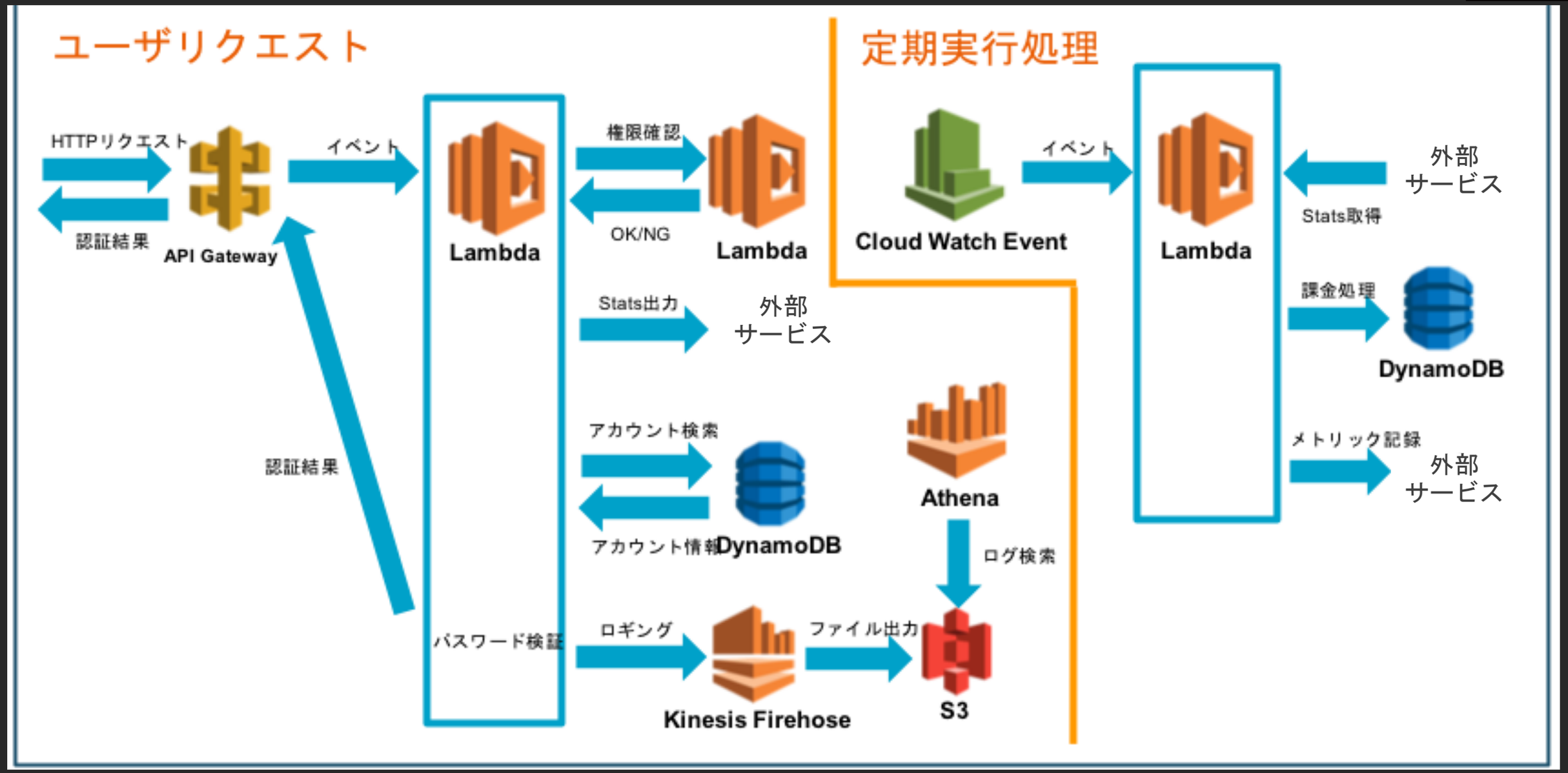
- アイドルコストの削減 + 高いスケーラビリティ・可用性
- Startup Architecture of the year 2018 オーディエンス賞を受賞
- re:Inventに社員全員で行ける（監視コストが低い）

「サーバレスアーキテクチャの採用によって、スケーラビリティ・耐障害性を得つつ、実際にトラフィックが発生しなければインフラコストが発生しないという事業継続に直接関係するメリットを得られました。」（GS2 CEO 丹羽様）

https://gs2.io/

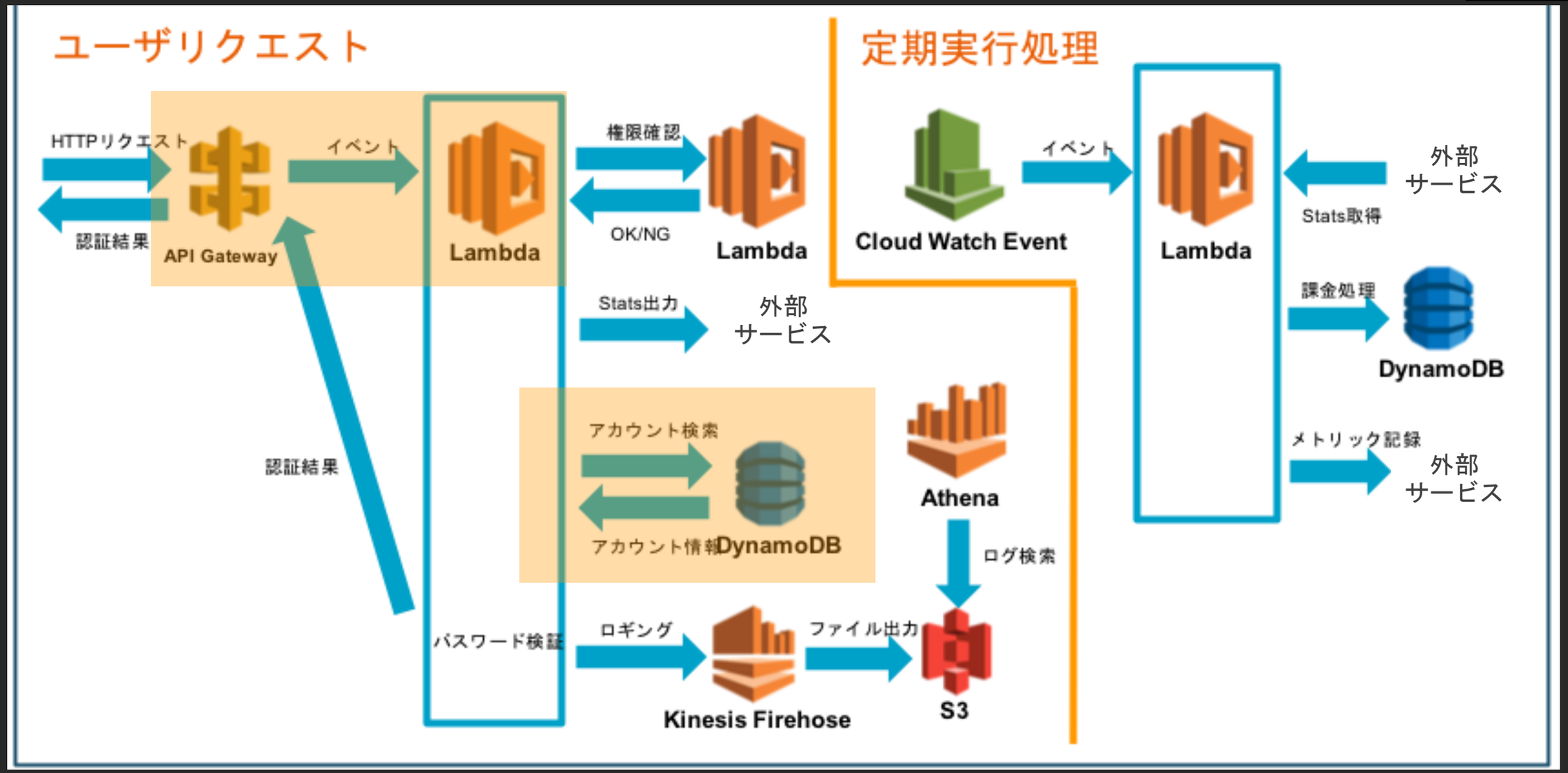
https://aws.amazon.com/jp/blogs/startup/saoy2018_gs2/

GS2のアーキテクチャ（アカウント認証の例）



https://aws.amazon.com/jp/blogs/startup/saoy2018_gs2/

GS2のアーキテクチャ（アカウント認証の例）



https://aws.amazon.com/jp/blogs/startup/saoy2018_gs2/

ゲームサーバーの特徴

aws SUMMIT ONLINE

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

ゲームサーバーの定義

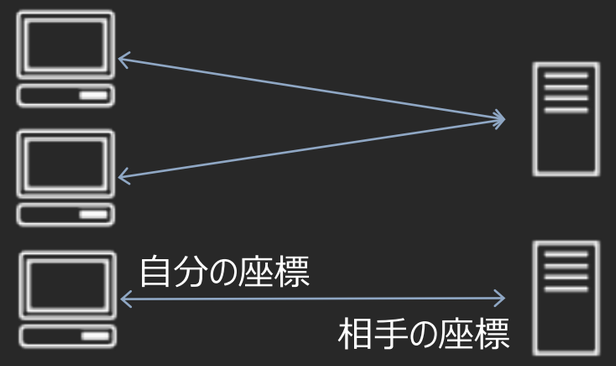
ゲームロジックを処理するサーバーのこと。また、その周辺システム

分類1: セッション型（常時接続）

多人数シューターなど頻繁な状態同期が必要なゲーム向き

高頻度な双方向通信（毎秒1～）

事例: dead by daylight, For honor, ...

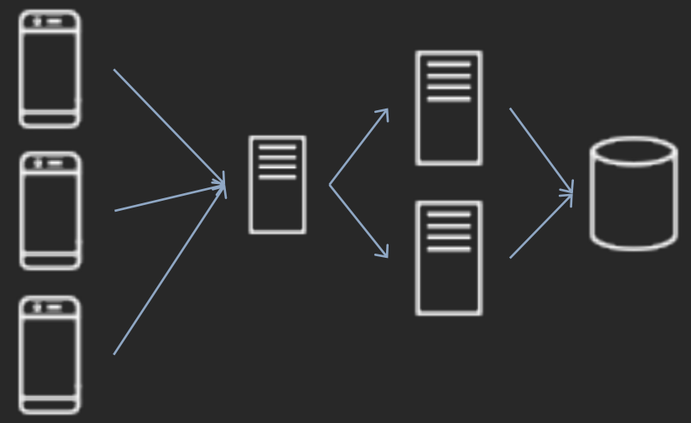


分類2: API型（都度接続）

1人用ゲーム、モバイルゲームなど状態同期が少ないゲーム向き

マッチングサーバー、決済サーバーなど周辺システムもこちら

低頻度なサーバーへのリクエスト（毎分1～10）



ゲームサーバーの定義

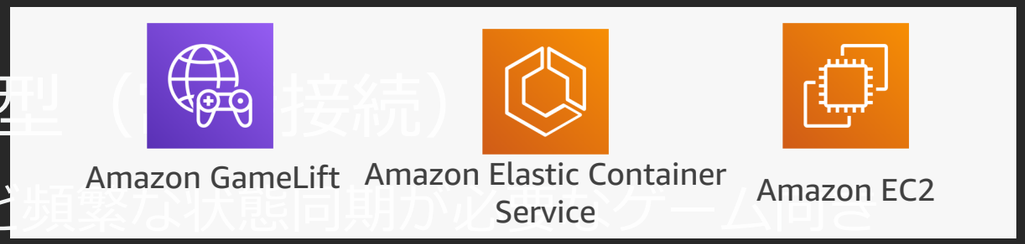
ゲームロジックを処理するサーバーのこと。また、その周辺システム

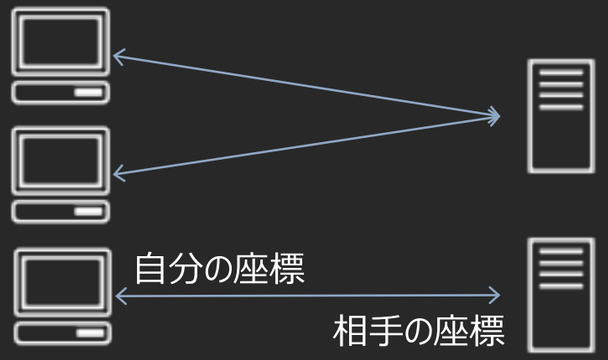
分類1: セッション型（接続）

多人数シューターなど頻繁な状態同期が必要なゲーム向き

高頻度な双方向通信（毎秒1～）

事例: dead by daylight, For honor, …





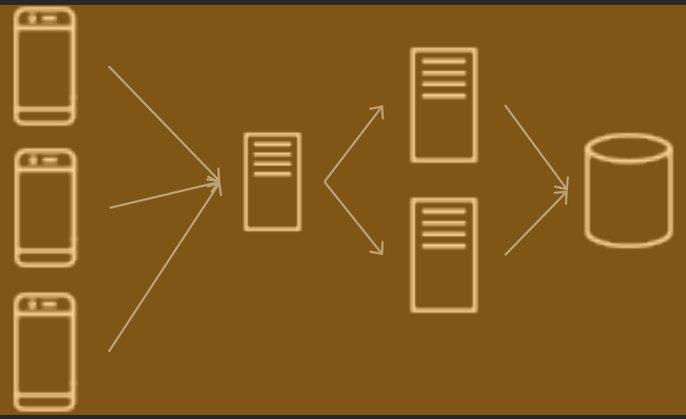
本セッションで扱うゲームサーバー

分類2: API型（都度接続）

1人用ゲーム、モバイルゲームなど状態同期が少ないゲーム向き

マッチングサーバー、決済サーバーなど周辺システムもこちら

低頻度なサーバーへのリクエスト（毎分1～10）



https://www.slideshare.net/AmazonWebServicesJapan/20191002-aws-black-belt-online-seminar-amazon-gamelift

ゲームサーバーの特徴

リリース後、メンテ明け、イベントなど、アクセスの急増が頻発

一日の中でも変動（夕方から徐々に増加するなど。海外展開していると時差でずれる）

RDBMS（MySQL、PostgreSQL）が多い

ロジックが複雑、ストアドプロシージャも多用

様々なプログラミング言語で書かれている場合がある

サーバー1台における集約度、パフォーマンスが重要なケース

ゲームクライアントと同じ言語で書いている / 自社ライブラリがC++しかないケース

24時間アクセスがあり、安定稼働期は最低限の利用量が掴みやすい

Reserved Instance (RI) でコスト削減をしているケースが多い

ゲームサーバーの特徴

リリース後、メンテ明け、イベントなど、アクセスの急増が頻発

一日の中でも変動[illegible]から徐々に増加するなど。海外展開していると時差でずれる）

RDBMS（My[illegible]

ロジックが複雑、ストア[illegible]

様々なプログラミング言語で書かれている[illegible]

サーバー1台における集約度、パフォーマンスが重要なケース

ゲームクライアントと同じ言語で書いている / 自社ライブラリがC++しかないケース

24時間アクセスがあり、安定稼働期は最低限の利用量が掴みやすい

Reserved Instance (RI) でコスト削減をしているケースが多い

（かつての）
サーバーレス化を困難にする要因たち

サーバーレス化と安定運用

aws SUMMIT ONLINE

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

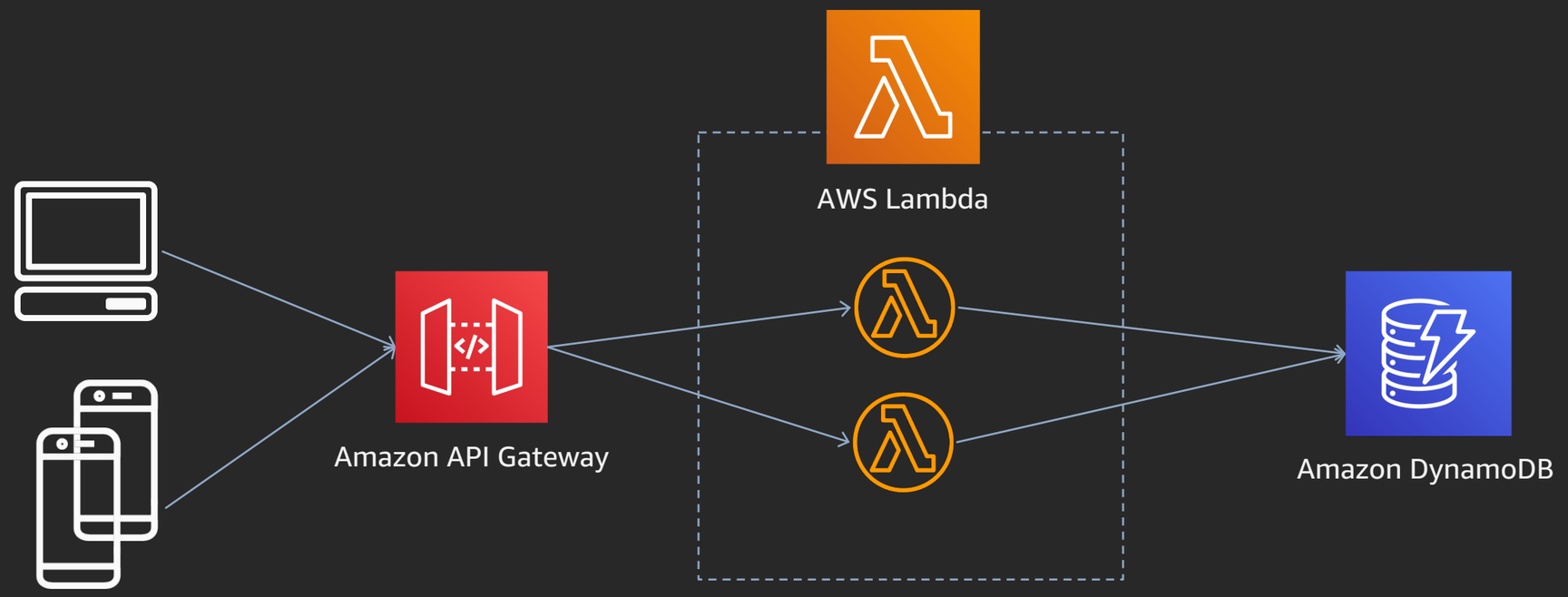
サーバーレスゲームサーバーの構築と運用

ゲームサーバーのサーバーレス化

安定運用

事例で出てきた"定番"の構成

API Gateway -> Lambda -> DynamoDB

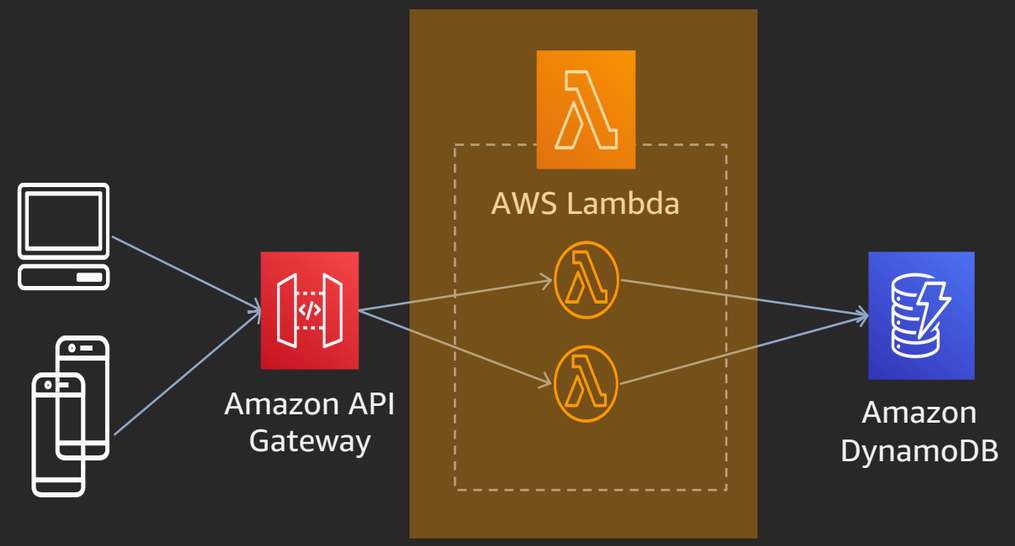


課題1: 言語ランタイムの対応

Lambdaはコードを実行するランタイムが予め決められていた

- Node.js (10.x, 12.x)
- Python (2.7, 3.6-3.8)
- Ruby (2.5, 2.7)
- Java (8, 11)
- Go (1.x)
- .NET Core (2.1, 3.1)

（2020/06時点）



未対応の言語・バージョンではLambdaを利用できなかった

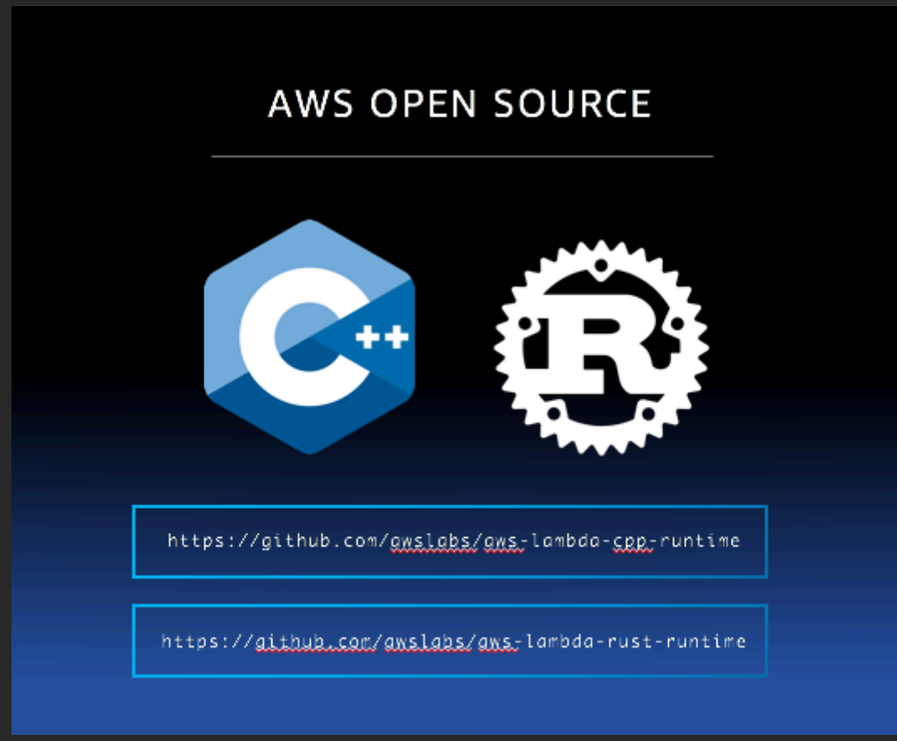
AWS Lambda カスタムランタイム – 2018/12

Update!

Amazon Linuxで動くランタイムを持ち込み、任意の言語でLambda関数を記述できる

AWSからはC++/RustをOSSでリリース

- https://github.com/awslabs/aws-lambda-cpp-runtime
- https://github.com/awslabs/aws-lambda-rust-runtime



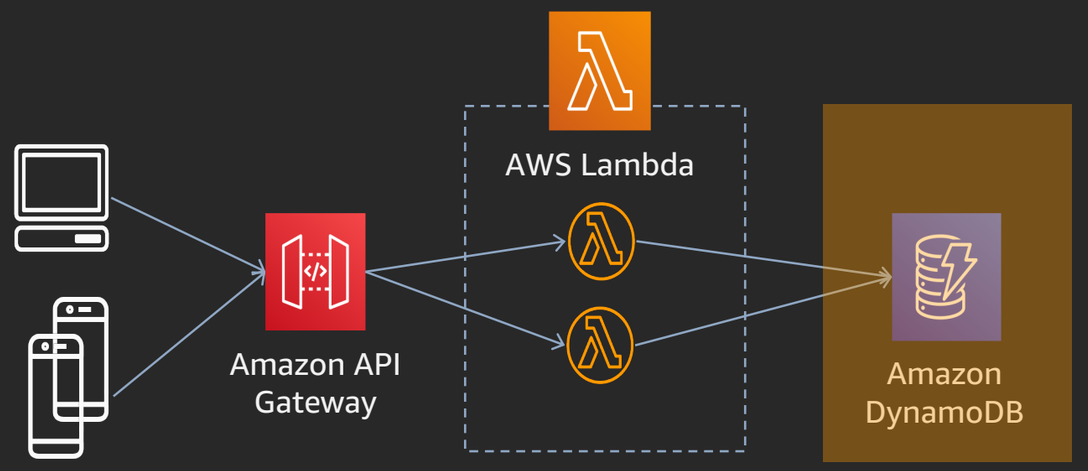
課題2: DBの変更が困難

DynamoDB（KVS: key-value store）へ変更するのが困難

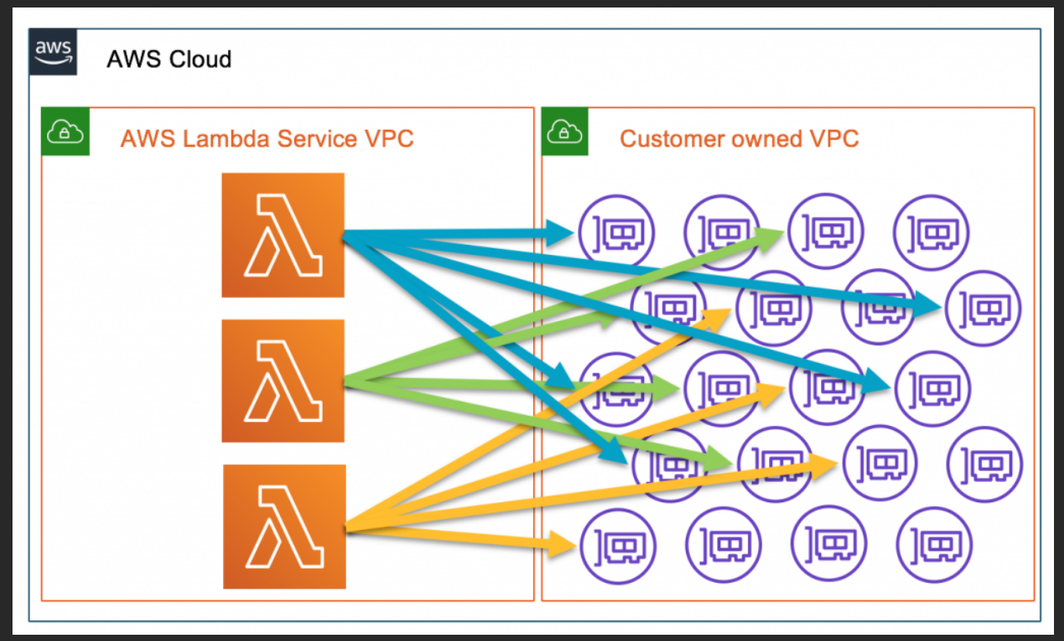
- RDBMSに強く依存したロジックがある
- 特に既存タイトルの移行は難しい

なぜ”定番”構成はDynamoDBなのか？

- Amazon RDSでは次ページのような課題がある



課題2: DBの変更が困難



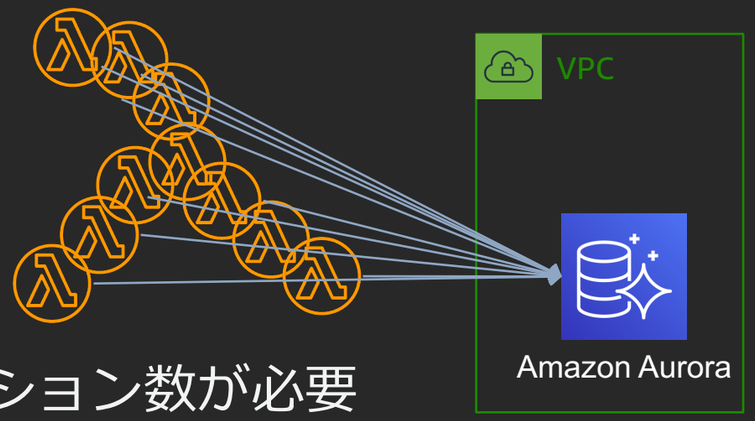
課題2-1: VPC内LambdaのCold Start

RDSはVPC内サービスなので、Lambdaも同じVPCにプロビジョン

実際は右図のように、ユーザーのVPCにENIを作成

ENI作成には1分程度かかり、長いCold Startが発生

→ Cold Startを受け入れるしかなかった



課題2-2: DBコネクション数の増加

1リクエストにつき1Lambda関数の実行に小分けされる

それぞれでDBに接続をすると、EC2/ECSに比べ遥かに多いコネクション数が必要

DBのCPU負荷が増える / 最大接続制限にかかる

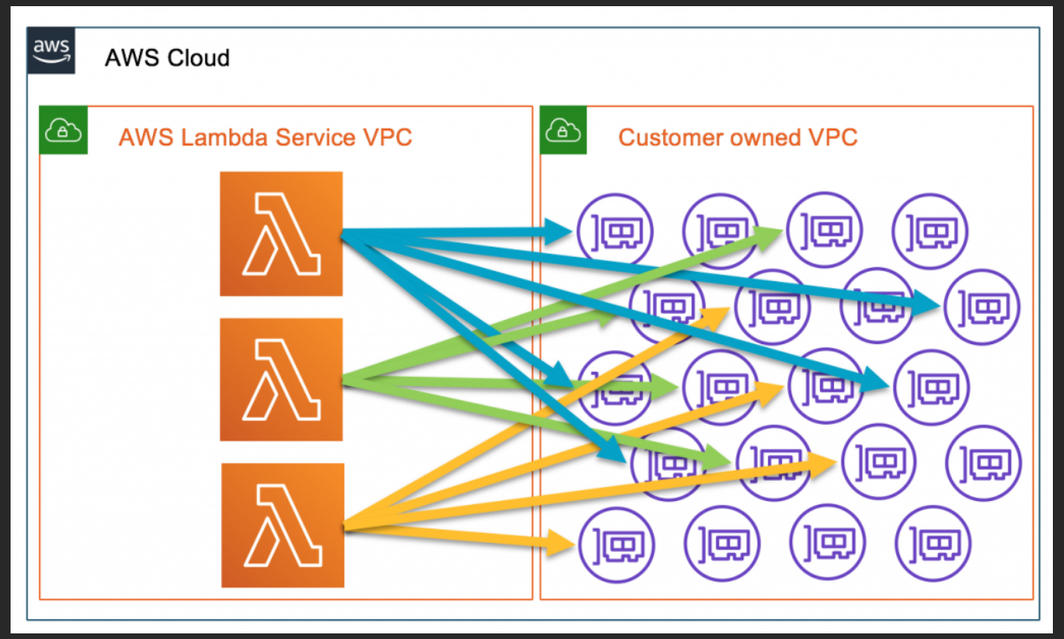
→ DBの増強・DB接続の共通化が必要

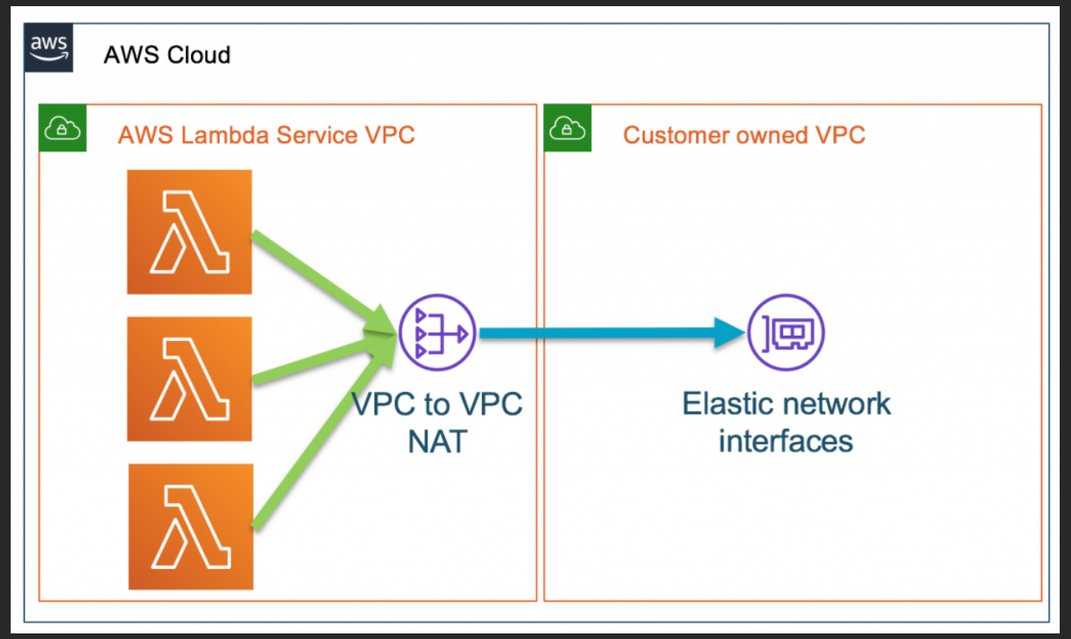
ENIの共通化 – 2019/09

Update!

アップデートにより、デプロイ時に共通のENIを作成するようになった

Cold Startが解消された





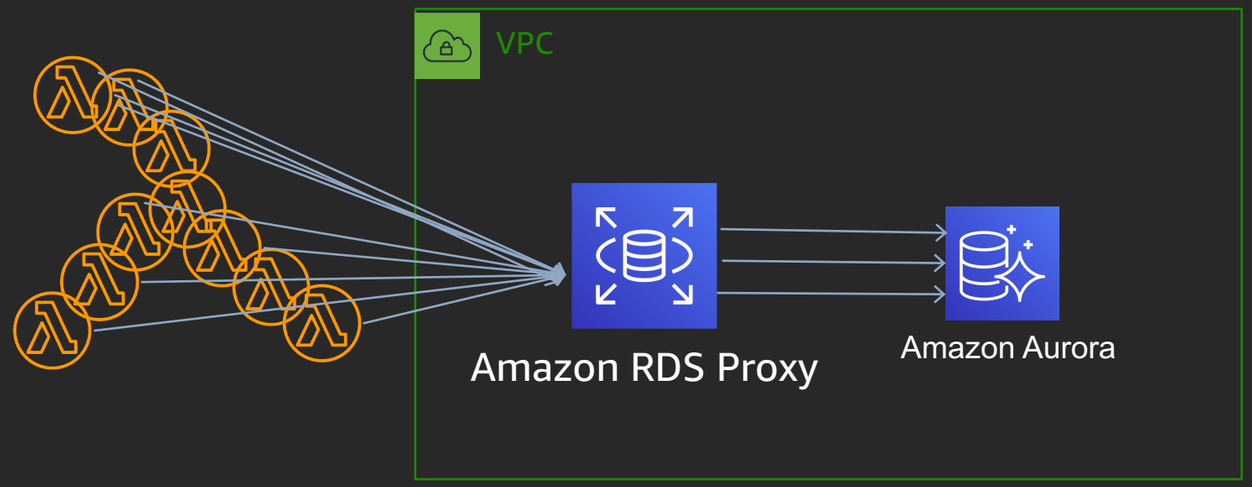
https://aws.amazon.com/jp/blogs/news/announcing-improved-vpc-networking-for-aws-lambda-functions/

Amazon RDS Proxy – 2020/06

Update!

マネージドなプロキシサーバー

- コネクションを使い回す（プーリング）が可能
- 各種制限はドキュメントを参照



※ あくまでプールするだけで、DBの接続許容数が増えるわけではない

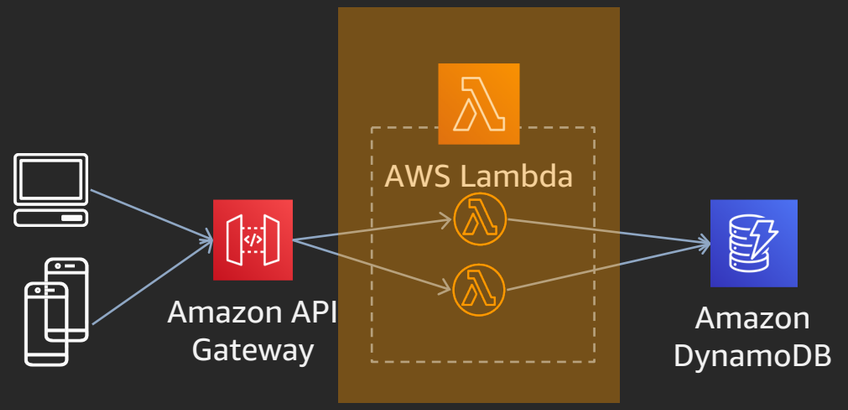
- スロットリングや実行時間の増加・必要同時実行数の増加、など考慮が必要

https://aws.amazon.com/jp/rds/proxy/

課題3: 既存のRIとの兼ね合い

EC2をReserved Instance (RI) でコスト削減していた場合

- Lambdaに移行すると、RIの適用率が下がる
- 他のプロジェクトで使えれば良いが、できない場合も多々ある



運用しているタイトルが少ないと特に影響大

- Lambdaを使ってコスト削減をするつもりだが、RIの買い替えまで動けない

Savings Plans for Lambda – 2020/02

Update!

Compute Savings Plans (SPs) = 最大66%の値引率で柔軟性の高い購入形式

- 1時間毎の支払い額を1年/3年で設定
- サイズ不問、リージョン不問など柔軟

Compute SPs はEC2に加え、Fargate/Lambdaにも適用される

- EC2、Fargateに適用された後、余ったものがLambdaに適用
- 最大17%の値引率

可能なら RI から SPs にすると無駄が少なくなる

- 毎時間での判定になることに注意
  - 夜だけ増加し昼は少ない、という場合は少ない方に合わせる

https://aws.amazon.com/blogs/aws/savings-plan-update-save-up-to-17-on-your-lambda-workloads/

サーバーレスゲームサーバーの構築と運用

ゲームサーバーのサーバーレス化

安定運用

不安定化要因1: 同時アクセスの増加

ゲームの人気が出て同時アクセスが増加していった結果、スロットリング（HTTPステータスコード: 429）が出るようになった

1. 同時実行数の上限を確認

   アカウント保護のため、東京リージョンでは各アカウントのリージョンにつき1,000に制限

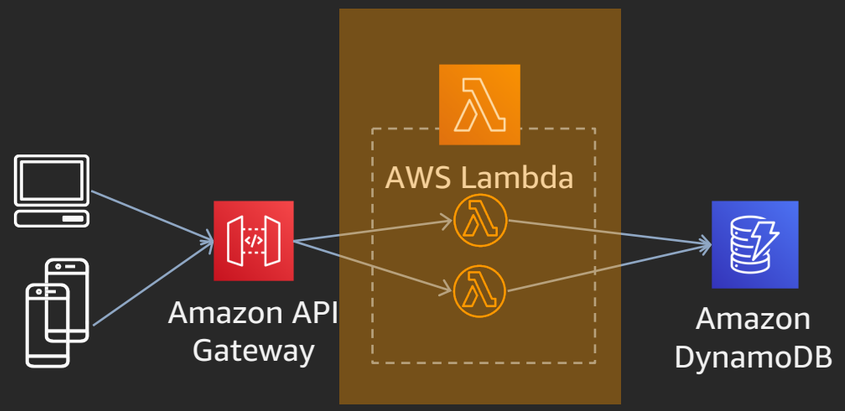
   → AWS Service Quotasで緩和申請可能

2. 同時実行可能数の増加速度を確認

   同時実行可能な数は 500/分ずつ増える

   これを超えるとスロットリング

   → 事前の対処が必要



https://docs.aws.amazon.com/ja_jp/lambda/latest/dg/gettingstarted-limits.html

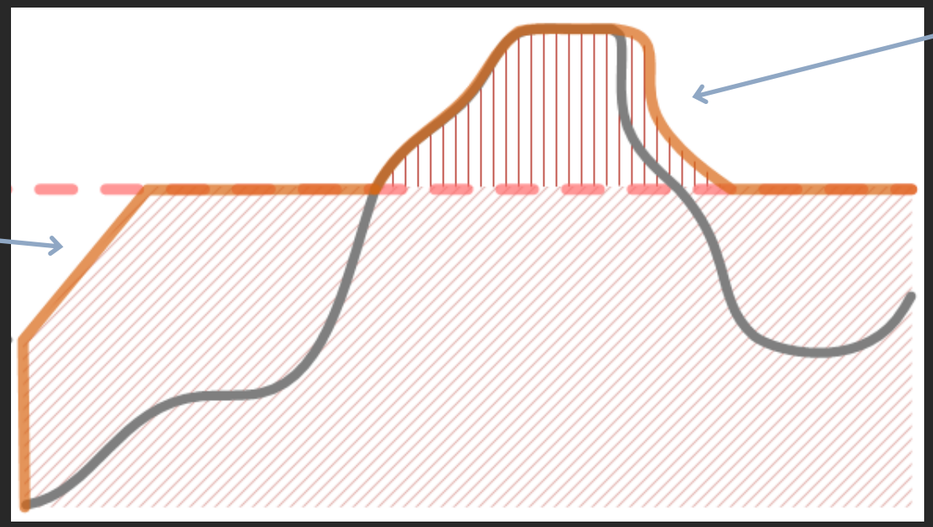
Provisioned Concurrency – 2019/12

Update!

アクセスの急増に対処するための機能

設定した数の同時実行をCold Start無しで行える

アカウントの同時実行上限の中から、1関数の1バージョン（エイリアスも可）に割り当てる



詳細はドキュメント、「AWS Lambdaのスロットリング緩和 」の資料へ

https://docs.aws.amazon.com/ja_jp/lambda/latest/dg/invocation-scaling.html
https://speakerdeck.com/_kensh/how-to-manage-throttling-for-aws-lambda

不安定化要因2: 不十分なリトライ処理

どのような場合でも、クライアントでのリトライ処理は必要

リトライは、exponential backoffを使って影響を減らす

加えて、一定回数試して通らない場合、期間を空けてリトライしてもらう

例:

1秒後にリトライ → 2秒後にリトライ → 4秒後にリトライ →
8秒後にリトライ → 「しばらく待ってリトライしてください画面」

詳細はセッション「AWS Summit Online 2020: Let’s dive deep into AWS Lambda Error Handling」へ

不安定化要因3: 不十分な負荷試験 & 上限緩和

誤って大量に利用してしまうことを防ぐため、各AWSサービスには上限が設定されている

Lambdaで特に重要なのは、紹介した最大同時実行数

API Gateway – 10,000req/秒

DynamoDB – キャパシティ制限、などなど

負荷試験を行うことで、制限を表出させる

Tips: 関連するサービスのアップデート

ゲームサーバーを構築する上で役立つアップデートを紹介

API Gateway HTTP API – 2019/12

REST APIに比べ、低レイテンシー・高コスト効率

一部機能制限があるので、用途に合うかはドキュメントで確認

AWS Step Functions Express Workflows – 2019/12

Lambdaなど、各サービスの処理を組み合わせてステートマシンを作るサービス

より低レイテンシー・高コスト効率で大量のリクエストを処理できるようになった

https://docs.aws.amazon.com/ja_jp/apigateway/latest/developerguide/http-api-vs-rest.html
https://aws.amazon.com/jp/about-aws/whats-new/2019/12/introducing-aws-step-functions-express-workflows/

まとめ

API型のゲームサーバーではサーバーレスサービスを検討する価値が高い

サーバーレスサービスは機能追加で利用しやすくなっている

- 特にここ1-2年のアップデートはゲームサーバー向き

アクセスの増加（特に急増）で規模が大きくなると対策を検討

- スロットリング: 上限緩和、Provisioned Concurrencyなどで対処
- コスト最適化: Compute Savings Plansも利用可能

リトライ処理や負荷試験はサーバーレスでも必要

上手に使って良いサーバーレスライフを！

Thank you!

aws SUMMIT ONLINE

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.