サーバレス

疎結合化の追求

なぜ疎結合化の追求をする？

どう実現する？

コンポーネント間の相互依存を減らした構成にする

障害時の影響を抑える

密結合

全体に影響が起きやすい

システムの変更・追加が難しいなど

疎結合

耐障害性が高まる

負荷対応やスケーリングが容易

疎結合化向けサービス

ELB　サーバ間の結合間を減らせる

SQS　キューイングによる通信でインスタンス連携を結ぶことで疎結合化を実現

SNS SNSのアプリ間通信でインスタンス連携を結ぶことで疎結合化を実現

Lambda　サーバインスタンスでなくLambdaによるトリガー処理で疎結合化

疎結合化設計

サーバレス・キューイング通信・マネージド型サービスの利用などを駆使する

SQSの概要

ポーリング型キューイングサービス

複数のプログラム間通信に対して一定のタイミングの問い合わせがあると

送受信処理を行う

①通信内容を送信（送信処理→SQS）：リクエストキュー

②キューとして保持（SQS）

③通信内容を問い合わせ（SQS←受信処理）：レスポンスキュー

④キューがあれば受信　　　　　　　＜メッセージありますか！！

特徴

フルマネージド型

高可用性・高スケーラビリティ・高スループット・低コストを実現

無料枠がある　従量課金による低コスト

メッセージサイズ256KBまでの軽いデータ：メッセージゆえに

60秒から14日間保持する

In Fightメッセージ

キューのタイプ

標準キューとFIFOキュー

標準キュー

なるべく順番通りに処理する

同じメッセージを複数回受信する可能性あり

少なくとも1回以上の配信：１＝＜

FIFOキュー

絶対順番通り

ShortPoling　 キューが空なら即時リターン

LongPoling　 キューが空の場合はタイムアウトまで待つ

デッドレターキュー　ずっと残ったメッセージを別キューに移動する

正常に処理できなかったメッセージを隔離できる

Visibility Timeout　 新しいメッセージを見えなくする

重複処理をやめさせたいとき

優先的に処理してほしいとき

Spotインスタンス使うときに効果的に利用できる

遅延キュー 0～１５秒キューが誰にも見えなくなる

優先度付きキュー キューの処理順序に優先度付けが可能

一時キュー 高スループットでコスト効率に優れた

アプリケーション向けに利用されるキュー

キューの重複解除ID　 FIFOのみで利用する

暗号化 KMSで暗号化する

メッセージタイマ 個々のメッセージで表示させない遅延時間を変更する

アクセス管理

IAMポリシー

SQSポリシー

バッチアクション

AmazomSDKを使用してバッチアクション活用

SQSの基本構成

フロントエンドとバックエンドを疎結合化

フロントエンドからキューがトリガーされる

バックエンドサーバが並列処理

SQSとAutoScalingの構成は相性がよい

KinesisとSQS使い分け

SQS コマンド内容を処理する

Kinesis IoTデータを投入するなどのデータフローに使う

SNSの概要

SimpleNotificationService

受信側から送られたものを送信側へプッシュするだけ

特徴

単一発行メッセージ

メッセージ通信順番は保障されない

配信ポリシーによる再試行を実施

最大256KB

連携

Cloud Watch ：

SES ：

S3 ：

Elastic Transcoder ：

疎結合アーキテクトが可能

SNS

メッセージは永続ではない

プッシュ型配信方式

SQS

メッセージは永続性あり

ポーリング側配信方式

SESの概要

メールサーバとして利用するだけでなくアプリから自動でメール送信や連携可能

送信方法

HTTP REST API

SMTPエンドポイント

利用準備

ドメインを登録

メールの事前申請

をする必要がある

サーバレス

主要サービス

Lambda（中核）

SNS・SQS・SES

ELB

DynamoDB

API Gateway

Cognito

概念

EC2インスタンス　→　　Lambda

高コスト 低コスト

密結合 疎結合化

サービス指向アーキテクチャ　→　マイクロサービス

大きなシステムをひとつのサービスとして定義していたものからさらに

小さくしていく

疎結合化とマイクロサービスは表裏一体

SOA マイクロサービス

年金給付サービス　　→　給付対象者決定業務

支払額算定業務

支払業務

API活用

マイクロサービスはAPI通信によりコンポーネント間を疎結合する

サーバレス化のポイント

シンプルな機能や処理を実行するだけなのにEC２インスタンス構築している

Lambda

インフラを一切気にすることなくアプリケーションコードを実行できる

従来：PC　→　EC2インスタンス　→　DynamoDB

　PC　→　　　Lambda　　　 →　DynamoDB

Node.js/Javaで書かれたコードを実行

Lambdaの仕組み

Lambdaファンクションを用意する：コーディング

アプリからLambdaを呼び出す

Lambdaの起動

Pushモデル

リソースからLambdaを呼び出す

Pullモデル

LambdaからデータをPull（取りに行く）する

Lambdaのパーミッション

Execution

Lambda側が何を動作できるのか

IAMロールをLambdaにアタッチする

Invocation

外部のリソースからLambdaへのアクションを設定する

Lambdaの連携

Amazom　S3

Amazom　Kinesis

Amazom　DynamoDB　Streams

Amazom　Cognito（認証機能）

Amazom　SNS

Alexa　Skills　Kit

Amazom　SWF

Lambdaの設定

コードをアップロードする

関数を設定

必要なメモリ量を指定

タイムアウト時間を設定

VPCアクセス用にVPCを指定する

関数を起動する

ブループリント

サンプルコード集を利用することが可能

Lambdaを利用するユースケースを設計

ブループリントでサンプルコードを探す

サンプルコードを修正して実行

→効率的！！！

VPCアクセス

インターネットを経由せずにVPC内のAWSリソースにアクセスできる

LambdaLayer

Lambdaファンクションに共通機能があったとき集約できる→効率化

ロードバランサー機能

ALBのバックエンドにLambdaを呼び出せる

Lambdaユースケース

・Alexaスキルを呼び出す

・異常検知ツール

CloudTrailのログをS3に保存

S3内にイベント発生することでLambdaがプッシュ

・KinesisAnalyticsでのスケーリング

・モバイルアプリ

Cognitoで認証

S3にクライアントが写真を登録←これをトリガーにLambda起動

LambdaによりメタデータをDynamoDBに登録

クライアントはDynamoDBからメタデータを取得

Lambdaエッジ

CloudFront　＋　Lambda

エッジロケーションCDN機能が利用できる

API Gatewayの概要

API

システムをつなぐ連結器

様々なリソースはAPIを通じて繋がる

APIエコノミー

サービスの機能をAPI化して他社に使ってもらう

API利用の必須事項

APIの作成

利用状況の監視

アクセス管理

バージョン管理

→　AWSではこの構築から保守までAPIGatewayとして提供

APIGateway

DDoS攻撃対応やスロットリングによるバックエンド保護

WebSocketを利用したリアルタイム且つ双方向通信のAPIも処理可能

Lambdaと密接に統合されている

EC2/Lambda/任意のウェブアプリケーションのワークロード処理を実行

連携方法が豊富

ユースケース

APIGatewayを連携口として外部アプリケーションとの連携を実現

Webアプリ

APIGateway

Lambda・APIGatewayキャッシュ

AWSサービス

LambdaやAPIGatewayはリージョンに配置される