信頼性の確保

耐障害性向上

フェイルオーバー（Route53）

別リージョンにバックアップを設ける（S3）

BCPの整備を進めて復元やフェイルオーバー手順を作って検証する

ドキュメント作成もしていくこと

高可用性の確保

自動でシステムのダウンタイムを０にしていくこと

高可用なサービスを利用していく（S3など）

高可用なアーキテクチャ設計をユーザでする（EC2）

高可用性の非機能要件

目標復旧時間

目標復旧時点

耐障害性（組み込まれた冗長性）

復元可能性

拡張性（AutoScalingをいかにうまく使うか）

可用性があがるとコストと複雑さが上がる

AWSのサービス配置により高可用性を高める

リージョンの選択

AZの選択

VPCの設計

↓

AWSサービスの利用

単一障害点の排除

EC2　DirectConnect　RDS

にはELBなどのリソースをしっかり使っていく

ELBなどを使ってマルチAZ構成にして単一障害点を排除する

データベース

マスタースレーブ構成

常にマスターのDBを使う

スレーブを別AZにして危機管理

リードレプリカで負荷を軽減する（スケーリング）

ElasticIP

Route53のIPフローティングを使うと別サーバにトラフィックを移せる

ELBの概要

AWSサービスが提供する負荷を分散する

インスタンス間の負荷を分散する

異常なインスタンスを認識して対応する

パブリック・プライベートどちらでも使用可能

ELB自体もキャパシティを自動増減するスケーリングを実施

従量課金

マネージドサービスのため管理が不要（単一障害点にならないAWS側が保障）

AutoScaling　Route53などと連携

基本機能

ヘルスチェック

負荷分散

SSＬサポート(セキュアな通信を可能にする　暗号化される)

スティッキーセッション(同じユーザからのリクエストは同じインスタンスに送信)

Connection Draining(異常なインスタンスへの通信をぎりぎりまで完了させる)

S3でログ保管

ELBの主要機能

スケーラビリティの確保

複数EC2インスタンス、ECSserviceへのトラフィック分散

トラフィック量を分散

EC2インスタンスA：30　EC2インスタンスB：70　みたいな

高可用性

EC2インスタンス。ECSserviceの正常ターゲットのみ振り分ける（集中）

ELBのタイプ

CLB（Classic Load Balancer）

初期のELB

複雑な設定が出来ない

HTTP/HTTPSとTCP/SSLのL4/L7に対応

ALB（Application）

L7に特化したELB

Dockerなどのコンテナにも対応可能

複雑なトラフィック制御が可能

アクセスログの取得

自動的にキャパシティ増減

パスルーティングで複数機能のバランシングが可能

パスルーティングイメージ

従来のELB（CLB）

Order.japan.com

Procure.japan.com

2つのURLを使って２つのELBをセットしていた

Jpan.com/order

Japan.com/procure

１つのELBで対応可能に

NLB（Network）

高スループットが必要

めちゃくちゃリクエストが多い

時に使う

L4レイヤー特化

VPC内のIPや静的IPをターゲット

大量のアクセスに対応可能

フォールトトレランス機能を内蔵したコネクション処理で数年のオープンな

コネクションを処理できる

コンテナ化されたアプリケーションのサポート（Dockerなど）

AutoScaling

スケーリングタイプ・・・水平スケーリング

スケールアウト：サーバの台数を増加する

スケールイン　：サーバの台数を低減する

要素

Auto-Scalingグループ：基本情報設定

起動インスタンス数の最大数と最低数を設定する

起動台数をAZ間でバランシングする

AZ自体の障害も見れる

最適なインスタンス数(Desired Capacity)になるように調整

Auto-Scaling　Configration：起動設定

設定項目　 AMI インスタンスグループ

セキュリティグループ　キーペア　IAMロール

Auto-ScalingPlan

How To Scaling？

Auto-ScalingGroupサイズの維持

現在のインスタンス最小台数を維持する

手動スケーリング

スケジュールベース

需要増を見込んでおく

動的スケーリング

予測不能な事態を考えて設定しておく

複数組み合わせ出来る

ヘルスチェック

EC2ステータス

状態がRunning以外の時にスケーリング

ELB（推奨）

ELBのヘルスチェック機能を使う

ターミネーションポリシー

①起動が一番古い奴から

②一番古い起動設定で起動している奴から

③課金されそうなやつから

④デフォルト　②③ランダムの順

連携

ELBとAuto-Scaling

Cloud WatchとAuto-Scaling

設計ポイント

最大値最小値の設定は慎重にすること

ステートフルなアプリケーションの設定にはAuto-Scaling Groupでの設定をする

ライフサイクルフック（終了するときのアクションを設定　例：ログ送信）

Auto-Scalingスラッシング（急なスケールインによる影響）を避ける

RDSの概要

フルマネージド型

標準ソフトウェアを利用したデータベースを構築できる

AWSでのDB構築

・RDSを使う

構築・管理が楽

AWS提供の範囲内での利用制限

・EC2内に構築する

自由にDB構成や機能を利用

構築、運用が手間

RDSの制約

バージョンが限定される：最新バージョンはすぐには使えない

キャパシティに制限がある

OSへのログインが出来ない

ファイルシステムへのアクセスが出来ない

IPアドレスが固定できない

一部の機能が使えない

個別パッチは適用できない

特徴

RDS自体がマネージド型の高可用

マルチAZによるますーたースレーブ構成を容易にできる

非同期でレプリケーション（リードレプリカを介する）

読み込み速度をスケーリングできる

Auroraは15台　RDSは5台

S3にスナップショットやトランザクションログを保存することで

耐障害性向上　スケーリング

スケーリング

マネージメントコンソールやAPIからスケールアップ可能

拡張はできるけど縮小はできない

シャーディングソフトウェア

EC2にいれる

RDSマスターをダブル構成にしてユーザIDによってアクセス先を変更可能

書き込み処理をスケーリングする

DBインスタンスの暗号化

暗号化対象

DBインスタンス

自動バックアップ

リードレプリカ

スナップショット

暗号化方式

AES-256暗号化

KMSによる鍵管理

インスタンス作成時のみ可能

スナップショットのコピー暗号化・リストア可能

RDSのスケーリング

運用時にパフォーマンス低下

書き込みが止まる・読み取りが遅いなど

スケーリングタイプ

垂直スケーリング

インスタンスのメモリやCPUの増減

水平スケーリング

サーバの台数を増減する

スケールアップ方法

インスタンスサイズの変更

インスタンスタイプの変更（汎用→？？？）

ストレージタイプの変更（IO処理が多いときはIOPSにするなど）

インスタンス

Db.t2.micro

ｔ２ ・・・インスタンスタイプ：得意とする性能領域の決定

Micro ・・・インスタンスサイズ：パフォーマンスの決定

インスタンスタイプ

汎用：Ｔ(汎用バーストパフォーマンスインスタンス)

バランスの取れたもの

多様なワークロードに対応

Ｔ(汎用バーストパフォーマンスインスタンス)

Ｍ(エンタープライズアプリケーションに適した：企業用)

メモリ最適化：

メモリ内の大きいデータセットに対応

高速ＤＢやインメモリＤＢなどに使う

Ｒ(メモリ負荷の高いDBワークロード向けに最適化、一番安い)

Ｘ(大規模アプリケーション　エンタープライズ向け)

Z(最強　だが高い)

ストレージタイプ

汎用

プロビジョンドIOPS（大量IOが必要、汎用だと処理できない）

マグネティック（HDDタイプ、そんなに使われない）

ストレージの容量変更

ストレージのAutoScaling可能

スケールアップのみ可能

RDSのスケールアウト

リードレプリカの増設：

読み込み処理の負荷分散

キャッシュの利用：

Elastic Cache ・ MYSQL Memcachedを利用

SQLクエリをキャッシュで保持して高速に対応可能

Auroraへの移行：

DBの変更することでリードレプリカの上限突破などが可能

リードレプリカの増設

マルチAZやクロスリージョン構成が可能

冗長性とスケーリングを同時に可能

インスタンスタイプを独自にもてる

障害対応などでリードレプリカからスタンドアロンのDBインスタンス

に手動で切り替え可能（Aurora以外）

Auroraの場合はプライマリーインスタンスに昇格可能

RDSの構成

マルチAZ

クロスリージョン

リードレプリカ

キャッシュの利用

ElasticCache：

AWSが提供するインメモリDBサービス

クエリ処理をインメモリDB内に保持して高速処理を可能

Memcached：

MYSQLのオプション機能

Auroraへの移行

RDSより高パフォーマンスを期待できる