転ばぬ先の AWS Well-Architected

～ AWSサポートからみた「信頼性の柱」～

アマゾンウェブサービスジャパン 合同会社
Sr. Cloud Support Engineer
立野広樹

© 2022, Amazon Web Services, Inc. or its affiliates.

アジェンダ

- AWS Well-Architected Framework「信頼性の柱」
- RDS/Auroraにおける実際

© 2022, Amazon Web Services, Inc. or its affiliates.

はじめに

© 2022, Amazon Web Services, Inc. or its affiliates.

AWS Well-Architected フレームワークとは

- AWSにおける「アーキテクチャを評価し、スケーラブルな設計を実装するための一貫したアプローチ」に関するベストプラクティス
- 6 つの柱に基づく

優れた運用効率	セキュリティ	信頼性
パフォーマンス効率	コストの最適化	持続可能性

- 今回は「信頼性の柱」を、RDS/Auroraの観点からご紹介します

© 2022, Amazon Web Services, Inc. or its affiliates.

よくあるお問い合わせ

データベースでフェイルオーバーが起きました

アプリケーションが正常に動作しませんでした

原因と対処を教えてください

© 2022, Amazon Web Services, Inc. or its affiliates.

フェイルオーバーが起きると困る点

- アプリケーションが正常に動作しない状況が継続してしまった
- フェイルオーバーのせいでアプリケーションの問題が発生した
- フェイルオーバーを避けられるなら避けたい
- フェイルオーバーが起きた原因が分からないから避けられない
- フェイルオーバーの原因を知らないといけない

本当にそうでしょうか？

© 2022, Amazon Web Services, Inc. or its affiliates.

フェイルオーバーとは

アプリケーションによる
データベースオペレーションは、
管理者による手動での介入なしで
再開できます

(参照: [RDS よくある質問 : DB インスタンスをマルチ AZ 配置として実行することにはどのような意味がありますか?])

© 2022, Amazon Web Services, Inc. or its affiliates.

あるべき姿

データベースでフェイルオーバーが起きました

アプリケーションは一時的にエラーになりました

その後、十分短時間で自動的に復旧しました

© 2022, Amazon Web Services, Inc. or its affiliates.

実はこれが

AWS Well-Architected フレームワーク
「信頼性の柱」

- しかし、RDBMS観点から理解しにくい部分も多い
- クラウドならではのベストプラクティス「AWS Well-Architected」
- 古くから使われてきた「RDBMS」
- 今回は、この話題を（改めて）考えます

© 2022, Amazon Web Services, Inc. or its affiliates.

AWS Well-Architected
「信頼性の柱」

© 2022, Amazon Web Services, Inc. or its affiliates.

「信頼性の柱」とは

- クラウドにおける信頼性向上のため一般原則
- 設計原則は5つ
  - **障害から自動的に復旧する**
  - **復旧手順をテストする**
  - **水平方向にスケールしてワークロード全体の可用性を高める**
  - **キャパシティーを推測することをやめる**
  - **オートメーションで変更を管理する**
- 今回は障害とその復旧にフォーカスしてご案内します

(参照: [AWS Well-Architected Framework 信頼性の柱 -> 信頼性 -> 設計原則])

© 2022, Amazon Web Services, Inc. or its affiliates.

失敗する可能性があるものは、失敗する

可用性 = 使用可能時間 / 合計時間

- ならば、「高信頼性＝低故障率」か？
- 残念ながら、ダウンタイム 0 のコンポーネントは存在しません
- ありとあらゆるものに、問題が起きる可能性があります
- 分散システムでは更に、複数の依存性が生じます
- ならば、何を考えるべきか？

(参照: [AWS Well-Architected Framework 信頼性の柱 -> 定義 -> 可用性])

© 2022, Amazon Web Services, Inc. or its affiliates.

「可用性」におけるトレードオフ

- **可用性には、トレードオフが存在します**
- 可用性を高めると、必要な設計・運用上のコストが増大します
  - 網羅的な検証の必要性によるコストの増大
  - 依存するシステムの選択の幅の低下
    - 「高可用性に特化した＝目的に特化したサービス」問題
  - コスト増大による弾力性の低下＝イノベーションの困難性増大
    - 「頻繁にテストできないので変更できません」問題

(参照: [AWS Well-Architected Framework 信頼性の柱 -> 定義 -> 可用性])

© 2022, Amazon Web Services, Inc. or its affiliates.

「信頼性」の基本は「回復力」

***回復力** はインフラストラクチャやサービスの中断から復旧し、需要に適したコンピューティングリソースを動的に獲得し、誤設定や一時的なネットワークの問題といった中断の影響を緩和するワークロードの能力です。*

- 「障害が絶対に起きない」ということは「絶対にない」
- 可用性とは（的を絞った）「回復力の目標」でもある
- 可用性要件はそれぞれのシステムによって異なる
- 用途に応じた可用性要件を考えることが重要

(参照: [AWS Well-Architected Framework 信頼性の柱 -> 定義 -> 回復力、および信頼性のコンポーネント])

© 2022, Amazon Web Services, Inc. or its affiliates.

RDS/Auroraの「可用性」低下要因

- 基盤側の避けられない障害事象
  - アベイラビリティゾーン／ネットワーク／インスタンス／ストレージ等
- 可用性に影響する種類のメンテナンス
  - 基盤・DBエンジンのメンテナンス（障害およびセキュリティ要因）
- ワークロードの予期せぬ負荷増大
- RDSは、これらの影響を最低限にするために、自動的に回復する（自動的にフェイルオーバーする）アーキテクチャ
- RDSの前提「回復力を持つことで、可用性を維持する」

(参照: [Amazon RDS によるスタンバイレプリカへのフェイルオーバーは、どのようなイベントのときに発生しますか?])

© 2022, Amazon Web Services, Inc. or its affiliates.

RDS/Auroraの「回復性」

- 可用性を守る仕組みは2つ
  - Single-AZでは、自動バックアップから「特定時点への復元」
  - Multi-AZでは、フェイルオーバーでスタンバイにいつでも切り替え
  - RDSの設計目標は99.95%、MultiAZで99.99%
- 「フェイルオーバー」は、「例外」ではなく「起きうる」こと
- なら「システム全体」でも、それを織り込む必要がある

これが「**Design for Failure**」

(参照: [AWS Well-Architected Framework 信頼性の柱 -> 付録A: 一部のAWSサービスの可用性設計])

© 2022, Amazon Web Services, Inc. or its affiliates.

「信頼性の柱」
RDS/Auroraにおける実際

© 2022, Amazon Web Services, Inc. or its affiliates.

RDS/Auroraにおけるシステムの「回復性」

- RDS/Auroraは障害発生時にフェイルオーバーが実施される
- 「システム全体」とは
  - データベースインスタンス（基盤、DBエンジン、ワークロード）
  - データベースクライアント
- この「全体」が回復しなければ、システムは回復しない

© 2022, Amazon Web Services, Inc. or its affiliates.

RDBMSにおけるワークロードの性質

- REL04-BP02 疎結合の依存関係を実装する
- REL04-BP04 すべての応答に冪等性を持たせる
- しかし、RDBMSにおけるクライアントとサーバの間は密結合
- TCP接続上でステートフルに処理を実施
- そもそも「データ」がステート
- 「サーバ」の「問題（障害・メンテナンス）」は「ワークロード」を「ロールバック」　（リカバリ）で対処する
- するとクライアントは「リトライ」できる

(参照: [AWS Well-Architected Framework 信頼性の柱 -> ワークロードアーキテクチャ])

© 2022, Amazon Web Services, Inc. or its affiliates.

フェイルオーバーで何がおきるか

- 「データベースがエラーを返す」わけではない
- 「エラーとなる」は「エラーという正常な応答を返している」
- 「フェイルオーバー」は「応答がない」パターン
  - 接続リクエスト送信
  - 実行リクエスト送信
  - 実行レスポンス受信
- 「応答がない」に対応して「リトライする」必要がある

© 2022, Amazon Web Services, Inc. or its affiliates.

RDBMSにおける障害検出の難しさ

- 「障害のイベント通知を受け取って、対応すればよい」か？

AWS では、通常、サービスを「データプレーン」と「コントロールプレーン」に分けて考えます。 ..... 一般的にデータプレーンの可用性設計の目標は、コントロールプレーンより高くなります。したがって、高い可用性要件を持つワークロードでは、コントロールプレーンの操作に対するランタイム依存を避ける必要があります。

- 「コントロールプレーン」への依存性は避けたい
- 「データプレーン」でいかに障害に対処するか

(参照: [AWS Well-Architected Framework 信頼性の柱 -> 信頼性-> 可用性ニーズの理解])

© 2022, Amazon Web Services, Inc. or its affiliates.

織り込むべきFailure(1) 接続確立しない

- SYNパケット送信
- ACKが帰らず再送を繰り返し、タイムアウト
- OSレベルのタイムアウト設定があります
  - net.ipv4.tcp_syn_retries
  - 所要時間は再送メカニズムに依存します
- データベースクライアントにも、接続ないしログインのタイムアウトが存在します
  - 例) PostgreSQL JDBC: loginTimeout

(参照: [AWS Well-Architected Framework 信頼性の柱 -> REL05-BP05 クライアントタイムアウトを設定する])

© 2022, Amazon Web Services, Inc. or its affiliates.

織り込むべきFailure(2) リクエスト送信に応答しない

- 既に存在する接続上でリクエストを送信
- ACKが帰らず再送を繰り返し、タイムアウト
- OSレベルの再送タイムアウトがあります
  - net.ipv4.tcp_retries2
  - 所要時間は再送メカニズムに依存します
- しかし次の問題によりこれだけでは対応できません

(参照: [AWS Well-Architected Framework 信頼性の柱 -> REL05-BP05 クライアントタイムアウトを設定する])

© 2022, Amazon Web Services, Inc. or its affiliates.

織り込むべきFailure(3) レスポンスが返らない (1)

- リクエストが完了するまでの間、通信は生じない
- アプリケーションレベルでのタイムアウトも通常、存在します
  - 例) PostgreSQL JDBC: socketTimeout
- ただし、「正常で時間を要しているか」「応答がないか」の区別なくタイムアウトします
- 「短時間で終わるもの」と「長時間かかりうるもの」で別の接続設定を使うといった工夫が必要となります

(参照: [AWS Well-Architected Framework 信頼性の柱 -> REL05-BP05 クライアントタイムアウトを設定する])

© 2022, Amazon Web Services, Inc. or its affiliates.

織り込むべきFailure(3) レスポンスが返らない (2)

- 正常か異常かの判断は可能か
- TCP上にtcp_keepaliveというメカニズムが存在します
- Linuxのデフォルトでは、2時間＋リトライ
- Aurora PostgreSQLでは、より短いタイムアウトを推奨
  - tcp_keepalive_time = 1
  - tcp_keepalive_intvl = 1
  - tcp_keepalive_proves = 5
- 確認のために追加処理が必要であり、トレードオフがある

(参照: [Amazon Aurora PostgreSQLによる高速フェイルオーバー])

© 2022, Amazon Web Services, Inc. or its affiliates.

織り込むべきFailure(4) 名前解決に失敗する

- RDSのフェイルオーバーは名前解決で実施される
- 名前解決に失敗するとフェイルオーバーが完了しない
  - 結果がアプリケーションレベルでキャッシュされている等
  - JavaVMで問題になる場合があるが、他の可能性も
- フェイルオーバー時に名前解決が改めて実行されるかテスト

(参照: [Amazon Aurora PostgreSQLによる高速フェイルオーバー])

© 2022, Amazon Web Services, Inc. or its affiliates.

より高速なフェイルオーバーに向けて(1)

- サーバとクライアントが協調するアーキテクチャも考えられる
  - Amazon Web Services (AWS) JDBC Driver for MySQL
  - Amazon Web Services (AWS) JDBC Driver for PostgreSQL
- Auroraは、フェイルオーバー元(writer)とフェイルオーバー先(reader) が、いずれも常に動作している
- フェイルオーバーは「役割の変更」
- 従って、裏で複数のDBインスタンスに接続し、高速なフェイルオーバーを実現可能
  - Failover Plugin

© 2022, Amazon Web Services, Inc. or its affiliates.

より高速なフェイルオーバーに向けて(2) タイムアウト改善

- 単純なタイムアウト : リクエストの性質に依存する
- TCP KeepAlive : OSに依存している
- 接続ドライバのレベルで実装すれば、より柔軟に対応可能
- 設定例) Aurora PostgreSQL向け

```
// 接続文字列は「jdbc:aws-wrapper:postgresql:〜」
final Properties properties = new Properties();
properties.setProperty("wrapperPlugins", "failover,efm");
properties.setProperty("connectTimeout", "30");
properties.setProperty("socketTimeout", "30");
properties.setProperty("monitoring-connectTimeout", "10");
properties.setProperty("monitoring-socketTimeout", "10");
conn = DriverManager.getConnection(url,properties);
```

© 2022, Amazon Web Services, Inc. or its affiliates.

より高速なフェイルオーバーに向けて(3)

- 高速フェイルオーバーの実施例

```
$ java PgAvail5
software.amazon.jdbc.wrapper.ConnectionWrapper@7f03894c - org.postgresql.jdbc.PgConnection@1b9a901e
....
Mon Oct 31 10:14:06 UTC 2022 2022-10-31 10:14:05.707066+00
Mon Oct 31 10:14:07 UTC 2022 2022-10-31 10:14:06.861876+00
10月 31, 2022 10:14:17 午前 software.amazon.jdbc.plugin.failover.FailoverConnectionPlugin failover
重大: The active SQL connection has changed due to a connection failure. Please re-configure session state if required.
software.amazon.jdbc.plugin.failover.FailoverSuccessSQLException: The active SQL connection has changed due to a connection failure.
Please re-configure session state if required.
    at software.amazon.jdbc.plugin.failover.FailoverConnectionPlugin.failover(FailoverConnectionPlugin.java:654)
    at software.amazon.jdbc.plugin.failover.FailoverConnectionPlugin.pickNewConnection(FailoverConnectionPlugin.java:774)
    at software.amazon.jdbc.plugin.failover.FailoverConnectionPlugin.dealWithOriginalException(FailoverConnectionPlugin.java:590)
    at software.amazon.jdbc.plugin.failover.FailoverConnectionPlugin.execute(FailoverConnectionPlugin.java:196)
    at software.amazon.jdbc.ConnectionPluginManager.lambda$execute$3(ConnectionPluginManager.java:298)
    at software.amazon.jdbc.ConnectionPluginManager.lambda$makePluginChainFunc$2(ConnectionPluginManager.java:250)
    at software.amazon.jdbc.ConnectionPluginManager.executeWithSubscribedPlugins(ConnectionPluginManager.java:228)
    at software.amazon.jdbc.ConnectionPluginManager.execute(ConnectionPluginManager.java:295)
    at software.amazon.jdbc.util.WrapperUtils.executeWithPlugins(WrapperUtils.java:224)
    at software.amazon.jdbc.wrapper.StatementWrapper.executeQuery(StatementWrapper.java:41)
    at PgAvail5$1.run(PgAvail5.java:39)
    at java.base/java.lang.Thread.run(Thread.java:829)
Mon Oct 31 10:14:25 UTC 2022 2022-10-31 10:14:17.133971+00
Mon Oct 31 10:14:26 UTC 2022 2022-10-31 10:14:25.331213+00
```

- ログから詳細な挙動も確認可能です

© 2022, Amazon Web Services, Inc. or its affiliates.

より高速なフェイルオーバーに向けて(4)

- 注意点
  - フェイルオーバー時の対処が異なるため、コードの見直しも必要です
    - PostgreSQLの場合、例外がスローされた後に接続を再作成する必要がなくなります
  - また、例外に適切に対応する必要があります
    - 接続が再利用できますが、トランザクションの再実行が必要です
  - 設定は簡単ですが、十分なテストが必要です

© 2022, Amazon Web Services, Inc. or its affiliates.

データベースクライアントの「回復力」

- これで全てのFailureに手当できているか
- できていない可能性もある
- どういう理由でDBクライアントにエラーがおきても
- 最後にアプリが遭遇するのは
  - 接続できない・ないし、接続が中断になってタイムアウトエラー
  - 応答がなくアプリケーション側でタイムアウトエラー
- データベースに応答ができない、は最終的にアプリの動作になる

© 2022, Amazon Web Services, Inc. or its affiliates.

「回復力」について、別の側面から考える

- なにが「問題」でなにが「回復」なのか？
  - DB / EC2 / OS / Network / ....
  - ユーザにとって問題なら、それが問題
- フォーカスすべきは、個々のサービスの技術的側面か

この場合の KPI は、サービスの運用の技術的側面ではなく、ビジネス価値に関する指標である必要があります。これによって障害発生時の自動通知と追跡が可能になり、障害に対処する、または障害を修正するための復旧プロセスを自動化できます。

(参照: [AWS Well-Architected Framework 信頼性の柱 -> 信頼性 -> 設計原則])

© 2022, Amazon Web Services, Inc. or its affiliates.

「信頼性の柱」における「復旧戦略」

再起動自体は、
障害から復旧するための簡単で信頼できる方法です。

さまざまなタイプの障害をそれぞれ捕捉、特定、修正するための新しいメカニズムを構築するのではなく、さまざまなカテゴリの障害を同じ復旧戦略にマッピングします。

状況ごとにカスタム修復を構築するのではなく、
そのいずれかをインスタンスの障害として扱います。

(参照: [AWS Well-Architected Framework 信頼性の柱 -> 障害管理 -> REL11-BP03 すべてのレイヤーの修復を自動化する])

© 2022, Amazon Web Services, Inc. or its affiliates.

他にも「復旧戦略」はいろいろありうる

- Elastic Load Balancing ＋ EC2 Auto Scaling
  - 「ヘルスチェック」して、「失敗」したら「交換（＝再起動）」
  - 「DB処理がうまくいかなかった」ら「ヘルスチェック失敗」にする
- 毎回接続を作成する
  - 接続を使い回すから、フェイルオーバーが起きたら問題になる
  - 毎回接続を作成すると、問題がおきてもすぐ回復できる
- などなど……

期待通りに動くか？

© 2022, Amazon Web Services, Inc. or its affiliates.

「回復力」に対する、科学的アプローチ

- 仮説：「きっとこれなら数分で復旧できるはず」
- 実験：「実験したところ、所要時間が5分」
- 検証：「障害検出まで想定より時間を要した」
- 改善：「ヘルスチェック設定？ or 別のアプローチ？」
- どのアプローチが良いかは、場合による

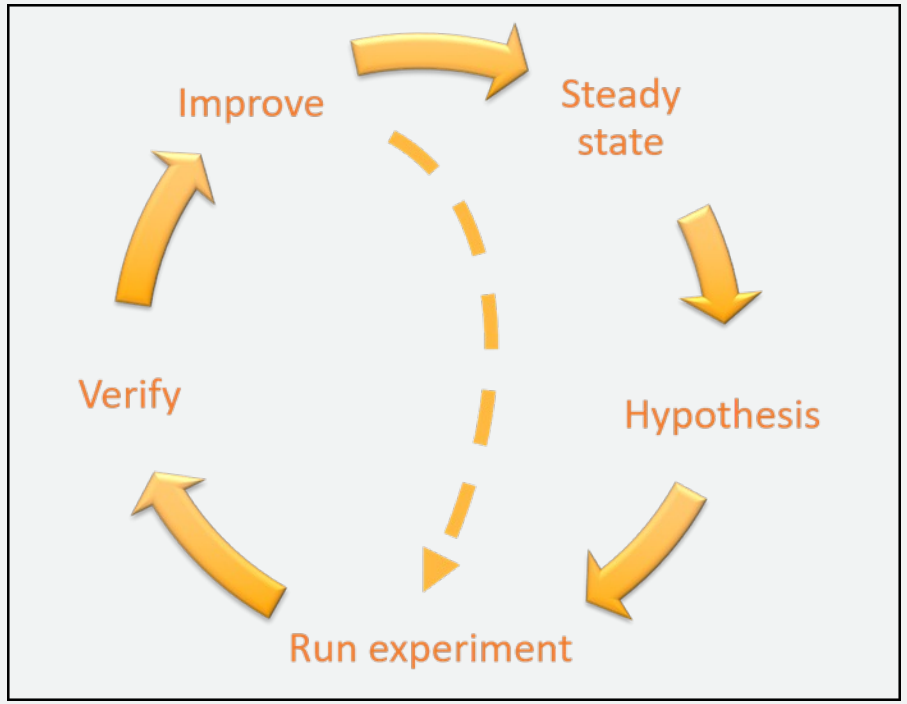
必ずテストすることが大事

(参照: [REL12-BP05 カオスエンジニアリングを使用して回復力をテストする])

© 2022, Amazon Web Services, Inc. or its affiliates.

AWS Well-Architected Labs 「信頼性の柱」

- 「RDS Failure Injection」ラボで、科学的なアプローチの例をご紹介しています



© 2022, Amazon Web Services, Inc. or its affiliates.

それでもまだ残るFailure

- 本当に処理が重いパターン
  - クエリ変更起因
  - 不適切な統計情報
  - 基盤側のボトルネック
- チューニングや、基盤側のスケーリング（スケールアウト・スケールアップ）といった対応が必要
- ワークロードにより対処が異なる

AWS Well-Architected「パフォーマンス効率の柱」へつづく……

© 2022, Amazon Web Services, Inc. or its affiliates.

まとめ

© 2022, Amazon Web Services, Inc. or its affiliates.

Amazon RDS には回復力があります

- システム全体に回復力がある時のみ、効果を発揮します
- 障害からの自動回復ができれば、速やかな回復もできます
- 様々なメンテナンスの適用リスクも低くなります
- システムを最新の状態に保ちやすくなります

「フェイルオーバー」も「メンテナンス」も
「想定外の特別なこと」ではなく「想定内」へ

© 2022, Amazon Web Services, Inc. or its affiliates.

One more thing…

- 尚、フェイルオーバー原因はRDS イベントにてご案内しています
  - *データベースインスタンスクラスへの変更の適用*
  - *ユーザーが DB インスタンスのフェイルオーバーをリクエストしました*
  - *The primary host of the RDS Multi-AZ instance is unhealthy (RDS マルチ AZ インスタンスのプライマリホストが異常です)*
  - *The primary host of the RDS Multi-AZ instance is unreachable due to loss of network connectivity (RDS マルチ AZ インスタンスのプライマリホストで、ネットワーク接続が切断されたため、到達できません)*
  - *The RDS Multi-AZ primary instance is busy and unresponsive (RDS マルチ AZ プライマリインスタンスはビジーで応答しません)*
  - *Database instance patch*
    - *等々*
- サポートにお問い合わせいただくことなくフェイルオーバー原因をご確認いただけますので、ご活用ください！

(参照: [AWSナレッジセンター : Amazon RDS DB インスタンスが再起動、回復、またはフェイルオーバーしたのはなぜですか？])

© 2022, Amazon Web Services, Inc. or its affiliates.

Thank you!

© 2022, Amazon Web Services, Inc. or its affiliates.