

CloudFront + S3環境から Cloudflare R2 + Workers環境に 移行した話

2023-03-16 吉祥寺.pm 32

@teckl

自己紹介

- @teckl (てっくる)
- シーサー株式会社でPerlのサービスを中心にインフラ的なお守りをしたり、データ移行したり、いろんな雑用をしたりしています
- 最近の趣味：糖尿病, 糖質制限
 - 糖質の取りすぎには注意しましょう…

糖尿病の検査結果 # _____
検査日: 2023年 2月20日

血糖値	優	良	可
空腹時	80~109	110~129	130~160
食後2時間	80~139	140~179	180~220

血糖値は検査した時点での測定値です。
食事や運動、ストレスなどによって変動します。

HbA1c (NGSP) 9.1 %
HbA1cは過去1~2か月の血糖値の平均を反映します。
良好な血糖コントロールのためには6.0%未満を、
合併症予防のためには7.0%未満を目指しましょう。

尿糖 - . ± . + .
尿糖は血糖値が高くなると陽性になります。

祝 ✨ YAPC::Kyoto 2023

<https://yapc-japan.org/2023kyoto/>

今週末はいよいよYAPC::Kyoto 2023が開催されるということで、
自分も何かアウトプットをせねばという事で初めてのLTさせていただきます。

ちなみにYAPC::Kyoto 2023はYouTube配信もあるので現地に行けない方でも是非…！



今回のテーマ

テーマは、YAPC::Kyoto 2023の「try/catch」にインスパイアされての、「finally」です。

色々な取り組みの果てに到達した成果、境地、慢心、環境の違い等について

ということで、

最近取り組んでいた

「CloudFront + S3環境を Cloudflare R2 + Workers環境に移行した話」
について紹介しようと思います。

目次1.

- Cloudflare Workersとは
- Cloudflare R2とは
- 移行する事になった経緯
- 移行前の構成
- 移行の流れ

目次2.

移行の流れ

1. Workers機能実装
2. アプリケーションからS3とR2のダブルライト実装
3. S3 -> R2へのフルコピー
4. キャッシュページの实装
5. ロギング
6. 整合性チェック
7. 切り替え (NSの委任)
8. 残作業

Cloudflare Workersとは

- <https://www.cloudflare.com/ja-jp/products/workers/>
- サーバレス(FaaS)
- Lambda@Edge, CloudfFront Functions みたいなやつ (雑)
- JavaScript(V8)などが動作
- ServiceWorkerが動く
- 簡単なKV (key value) のキャッシュ機能
- 簡単なアプリケーションならサーバレスで可能

Cloudflare Workersとは

- …と、にわかな自分が説明するよりも、GPT-4先生に聞いてみたものを貼ったほうが正確かもしれません。。



Cloudflare Workersの特徴を教えてください。



Cloudflare Workersは、Cloudflareのエッジネットワーク上で実行されるサーバーレスプラットフォームです。JavaScript、Rust、またはC言語で書かれたコードを実行できます。以下に、Cloudflare Workersの主な特徴をいくつか挙げます。

1. エッジコンピューティング: Cloudflareのグローバルネットワークを活用して、ユーザーに近い位置でコードを実行することができます。これにより、レイテンシーが低減され、パフォーマンスが向上します。
2. サーバーレス: インフラストラクチャの管理やサーバーのプロビジョニングを心配することなく、コードをデプロイできます。スケーラビリティが自動的に提供されるため、トラフィックの増減に対応できます。
3. 柔軟性: リクエストのルーティング、ABテスト、APIゲートウェイ、カスタムレスポンスの生成など、さまざまなユースケースに対応できます。
4. クイックデプロイメント: コードを素早くデプロイでき、変更もリアルタイムで適用されます。開発者はすぐにフィードバックを得ることができます。

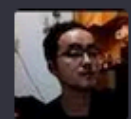
5. V8 JavaScriptエンジン: Cloudflare WorkersはGoogleのV8 JavaScriptエンジンを利用しており、高速な実行性能を提供します。
6. WebAssemblyサポート: RustやC言語などのWebAssembly (Wasm) 対応言語もサポートしており、パフォーマンスの向上や他のプログラミング言語を使用した開発が可能です。
7. セキュリティとプライバシー: Cloudflareのセキュリティ機能 (DDoS対策、SSL、Web Application Firewallなど) と連携して、アプリケーションのセキュリティを向上させることができます。
8. インテグレーション: Cloudflare Workersは、Cache APIやKVストレージなどのCloudflareのサービスと簡単に統合できます。また、外部のAPIやサービスとの連携も容易です。

これらの特徴により、Cloudflare WorkersはウェブアプリケーションやAPIの開発、デプロイメント、運用において、高速でスケーラブルでセキュアなプラットフォームを提供します。

Cloudflare R2とは

- <https://blog.cloudflare.com/r2-ga/>
- Amazon S3互換のオブジェクトストレージ
- 2022-09にGAされた
- S3と比べてさらに低コスト
- S3と違い、リージョンの概念は無い

Cloudflare R2とは



Cloudflare R2の特徴を教えてください。

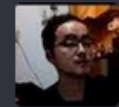


Cloudflare R2は、2021年に発表されたCloudflareのストレージサービスです。このサービスは、高速で安全なオブジェクトストレージを提供し、エッジネットワーク上で動作します。R2の主な特徴は以下の通りです。

1. エッジネットワーク: Cloudflareのグローバルエッジネットワークを活用して、データを世界中のサーバーに分散させ、低遅延で高速なアクセスを提供します。
2. 低コスト: R2は、競合他社と比較して非常に低いコストでストレージ容量を提供します。これにより、開発者や企業がコストを抑えながら、大量のデータを保存できます。
3. セキュリティ: Cloudflareは、DDoS攻撃などのサイバー攻撃からユーザーを保護することで知られています。R2も同様に、データ保護とプライバシーに重点を置いています。
4. 容量のスケラビリティ: R2は、必要に応じてストレージ容量を簡単に増減できる柔軟性を提供します。これにより、ユーザーは事前に大量のストレージを購入することなく、必要に応じて容量を追加できます。
5. 統合性: R2は、他のCloudflare製品やサービスとシームレスに統合されており、開発者は簡単にストレージ機能を追加できます。
6. パフォーマンス: Cloudflare R2は、高速なデータ転送速度と低遅延を実現するように設計されており、ユーザーはデータへのアクセス速度が向上します。

これらの特徴により、Cloudflare R2は、開発者や企業にとって魅力的なオブジェクトストレージソリューションとなっています。

R2とS3の違い



R2はS3とはどういった違いがあるのでしょうか？

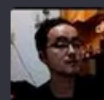


Cloudflare R2とAmazon S3は、どちらもオブジェクトストレージサービスですが、いくつかの重要な違いがあります。以下に主な違いを示します。

1. エッジネットワーク: Cloudflare R2は、Cloudflareのグローバルエッジネットワークを活用して、データを世界中のサーバーに分散させて低遅延で高速なアクセスを提供します。一方、Amazon S3はAWSのリージョンベースのインフラストラクチャに依存しています。これにより、R2は一般的にS3よりも低遅延でデータにアクセスできます。
2. コスト: Cloudflare R2は、ストレージ容量やデータ転送に関して競合他社と比較して非常に低いコストを提供します。特にデータ転送に関しては、R2は他のサービスと比較して大幅な節約が可能です。一方、Amazon S3は、ストレージやデータ転送のコストが高くなることがあります。
3. セキュリティとプライバシー: 両者ともにデータセキュリティとプライバシーに重点を置いていますが、CloudflareはDDoS攻撃対策などのサイバーセキュリティにおいて独自の強みを持っています。
4. 統合と相互運用性: Amazon S3はAWSのエコシステムに統合されており、他のAWSサービスとの相互運用性が強化されています。一方、Cloudflare R2はCloudflareの製品やサービスとシームレスに統合されています。ユーザーの既存のインフラストラクチャとの統合が重要な場合、どちらを選ぶかは使用しているサービスによって異なります。
5. パフォーマンス: Cloudflare R2はエッジネットワークのおかげで高速なデータ転送速度と低遅延を実現していますが、Amazon S3も高いパフォーマンスを提供しています。選択肢はユーザーのパフォーマンス要件と予算によって異なります。

R2とS3の違い

これらの違いにより、Cloudflare R2とAmazon S3は異なるニーズと予算のユーザーに向けたオブジェクトストレージソリューションとなっています。使用状況や既存のインフラストラク



続きをお願いします。



続きです。

状況や既存のインフラストラクチャに応じて、Cloudflare R2とAmazon S3は異なるニーズに対応するオブジェクトストレージソリューションを提供しています。以下に、どちらのサービスが適しているかを示す一般的なシナリオを示します。

1. Cloudflare R2が適している場合:

- ・ エッジネットワークを活用した低遅延が重要な場合
- ・ データ転送コストを最小限に抑えたい場合
- ・ Cloudflareの他のサービスや製品とのシームレスな統合が重要な場合

2. Amazon S3が適している場合:

- ・ AWSエコシステムとの相互運用性が重要な場合
- ・ より詳細なストレージクラスやデータライフサイクル管理が必要な場合
- ・ AWSの豊富な機能セットやサービスを活用したい場合

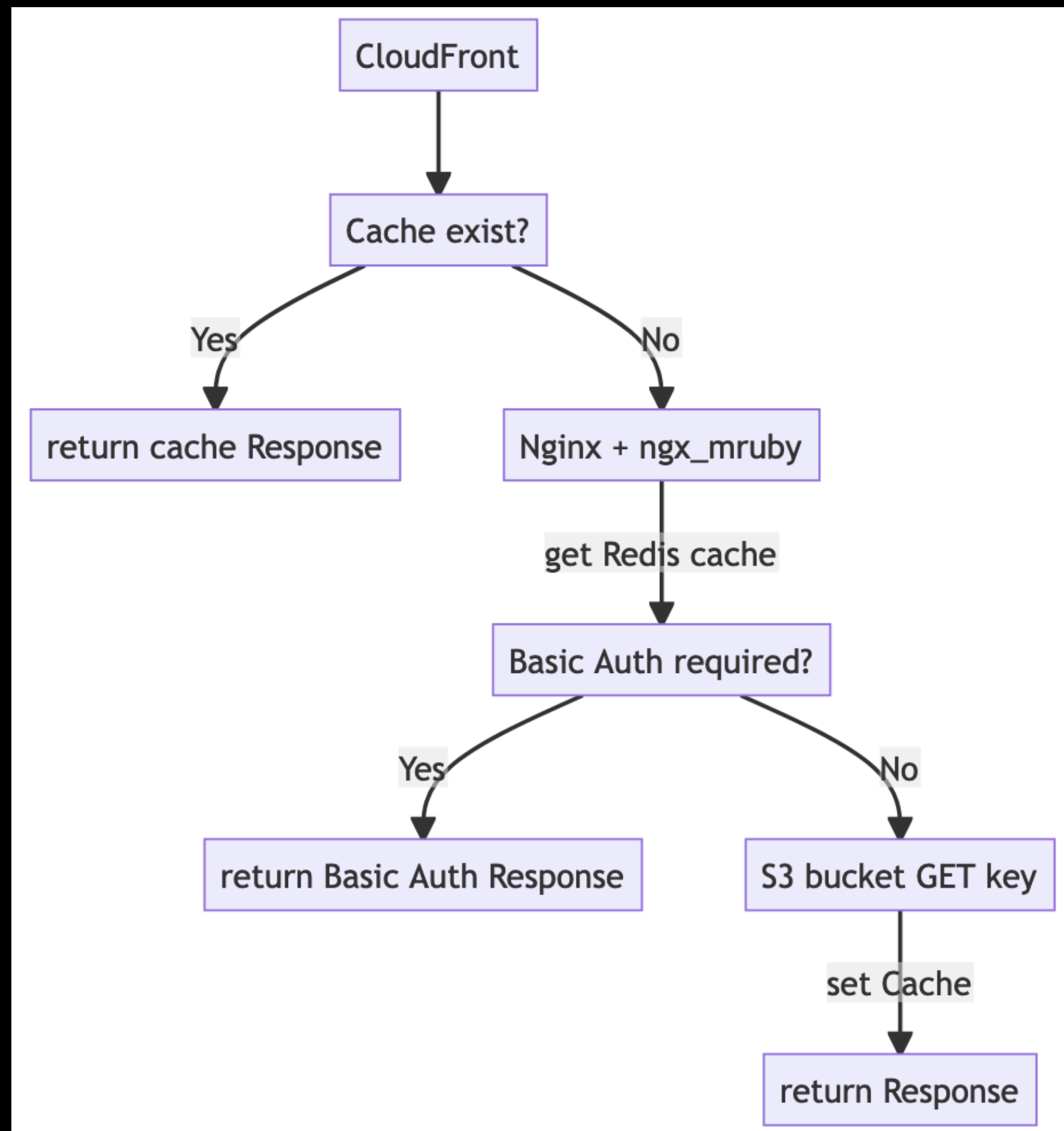
最終的には、どちらのサービスを選ぶかは、ユーザーのニーズ、予算、既存のインフラストラクチャ、およびパフォーマンス要件によって異なります。両方のサービスを評価し、自分の状況に最適なオブジェクトストレージソリューションを選ぶことが重要です。

…しつこいのでこの辺でGPT-4先生は終了にしますw

Cloudflare に移行する事になった理由

- 昨今の急激な円安の影響によるコスト増加への対策
- トラフィックとストレージコストを少しでも安く抑えたい
- まだまだGAされたばかりで実際の事例も少なく、初物は少々怖いところはあるが、チャレンジする価値があると考えた
- 実際にどれだけ安くなるかどうかはユースケース次第なので注意

移行前の構成



- Basic認証の機能のおかげで結構裏側が複雑な事になっていた
- 要件的に全キャッシュ出来ず、Nginx + ngx_mrubbyでURL毎に判定させていた
- ここも地味にコストがかかっていたため、Cloudflare移行を機にBasic認証機能は廃止してシンプルな構成にする事に

移行の流れ

1. Workers機能実装

- R2のGAのタイミングで、公開バケットという機能が提供されるようになったため、シンプルなバケットの公開だけであればWorkersを通さなくても配信できるように
- しかし、現在利用しているS3のバケットは、ちょっと厄介なパス構成になっており、公開バケット機能そのままでは使えないことが判明

1. Workers機能実装

現在のURLとバケットのイメージ

- URL

- `https://hogehoge.example.jp/image/hello.jpg`

- `https://test-subdomain.example.jp/image/abc.png`

- バケット内パス

- `h/ho/hogehoge/image/hello.jpg`

- `t/te/test-subdomain/image/abc.png`

こういう時はWorkersを使うと好きなようにR2バケット内のパスのマッピング等を柔軟にカスタマイズ可能

1. Workers機能実装

- 今回は移行のお話がメインなのでコード等は省略
 - 以下のページなどを参照していただけると…
 - <https://dev.classmethod.jp/articles/workers-r2/>
- なお、もう少し凝ったことをやりたい方は ゆーすけべーさんによる超高速な Cloudflare Workers フレームワーク 「Hono」 などを使うと良さそうです。
 - <https://hono.dev/>

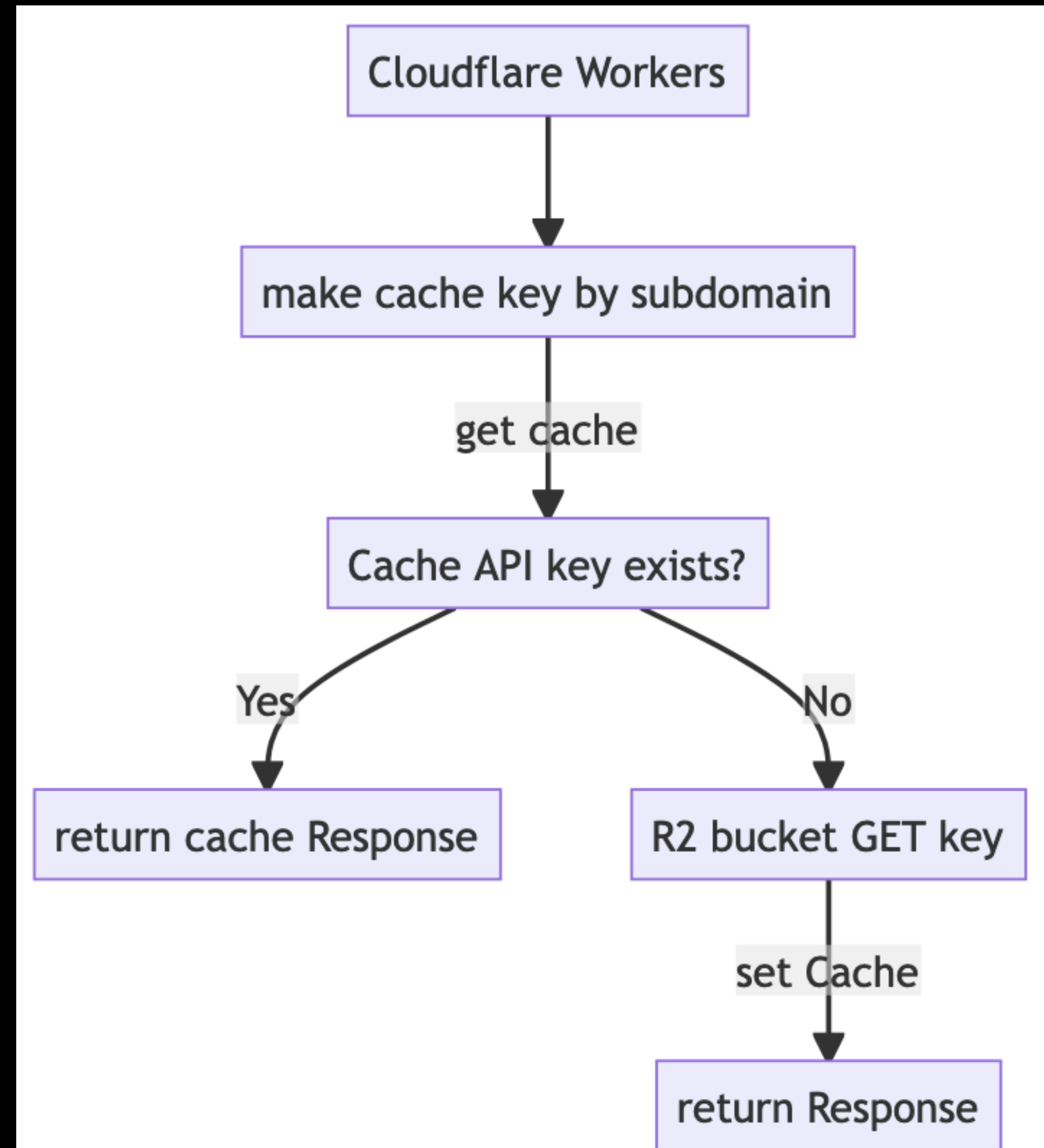
1. Workers機能実装

Cacheについて

- 画像配信が中心になるため、これまでのように出来る限りキャッシュしたいと考えました。
- Cloudflare Workersのキャッシュは複数の方法があります
 - fetch cf
 - Cache API
 - KV
 - Durable Objects

1. Workers機能実装 Cacheについて

- キャッシュヒット率などを考えるとKVが理想的に見えたのですが、KVはコストが別途かかってしまう…
- このため今回はCache APIを使ってキャッシュするように
- <https://developers.cloudflare.com/r2/examples/cache-api/>

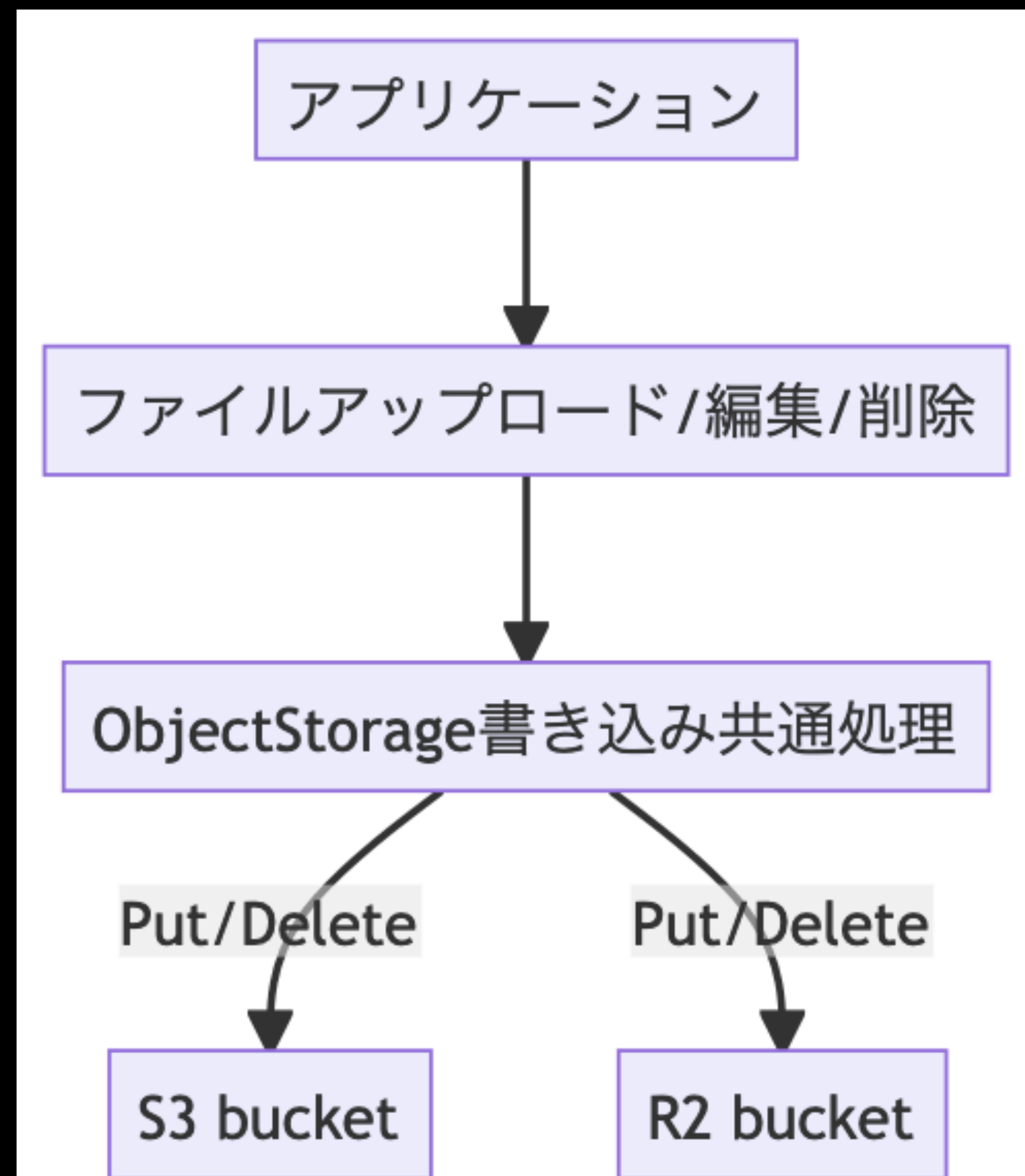


2. アプリケーションからS3とR2のダブルライト実装

- 以前オンプレからS3に移行した時は転送量をあまり意識しなくても良かったので素朴にデイリーで差分シンクをしていた
- しかしS3からR2への移行ではクラウド間の移行となるためトラフィックの料金にシビアになる必要がある
- バケットのコピーは出来る限り1回で成功させる必要がある
- 切り替え時のコピーによるサービスの停止時間（メンテナンス）も極力無くしたい

2. アプリケーションからS3とR2のダブルライト実装

- そのための準備としてダブルライト方式を採用することに
- アプリケーション側でS3とR2へ両方のストレージに書き込むように改修
- これにより、移行準備中も両方のストレージが最新の状態を保てるように



2. アプリケーションからS3とR2のダブルライト実装

- R2はS3互換のため、アプリケーション側はさほど大きな改修にならずに済んだ
- S3との主な違いとしては、R2には`region`の概念が存在しないので`auto`を指定する点、SSLが必須な点、エンドポイントを指定する点などくらい

```
# Amazon S3
$attr{client} = Amazon::S3::Thin->new({
  aws_access_key_id    => $aws_access_key_id,
  aws_secret_access_key => $aws_secret_access_key,
  region               => 'ap-northeast-1',
});

# Cloudflare R2
$attr{second_client} = Amazon::S3::Thin->new({
  aws_access_key_id    => $cloudflare_access_key_id,
  aws_secret_access_key => $cloudflare_secret_access_key,
  region               => 'auto',
  secure               => 1,
  endpoint_url         => $r2_endpoint_url,
});
```

3. S3からR2への全データコピー

- ダブルライトが本番環境で正常に動作する事を確認後、いよいよS3からR2へのバケットコピーを行っていきます。
- それなりのサイズのバケットの場合、今のところ大きく分けて2つの方法があります。
 1. R2 Migrator(R2 Super Slurper) で移行
 2. rclone で移行

3. S3からR2への全データコピー

方法1. R2 Migrator(R2 Super Slurper) で移行

- 公式によるR2への一括移行ツール
- R2 Migratorは本日時点ではまだBeta版
- オブジェクトのサイズやオブジェクト数などにいくつか制限がある
 - <https://blog.cloudflare.com/ja-jp/cloudflare-r2-super-slurper-ja-jp/>
 - <https://developers.cloudflare.com/r2/data-migration/r2-migrator/>

3. S3からR2への全データコピー

方法2. rclone で移行

- Rcloneとはオープンソースのコマンドラインベースのファイル同期および転送ツール
 - <https://rclone.org/>
- 公式でもR2でのrcloneの使い方を紹介している
 - <https://developers.cloudflare.com/r2/examples/rclone/>
- R2 Migratorはまだベータ版であり、バケットのサイズやコピーのタイミング制御などに不安があったため、今回はこちらの rclone を使用することに

3. S3からR2への全データコピー

rclone動作環境について

- コピー元のS3バケットが東京リージョンのため、東京リージョンに`c5n.4xlarge`の移行用EC2インスタンスを用意
 - (※c5nはネットワーク帯域が最大100Gbps)

3. S3からR2への全データコピー

rcloneのコマンドオプション

- rcloneには数多くのオプションがありますが、以下のサイトなどを参考に、最適となりそうなオプションを調整しました。

- <https://forum.rclone.org/t/how-to-maximize-single-file-multipart-upload-speed-to-cloudflare-r2-s3-compatible/34311>

- <https://rclone.org/flags/> より

- ```
--transfers int Number of file transfers to run in parallel (default 4)
--s3-upload-concurrency int Concurrency for multipart uploads (default 4)
--s3-chunk-size SizeSuffix Chunk size to use for uploading (default 5Mi)
--s3-disable-checksum Don't store MD5 checksum with object metadata
--no-check-dest Don't check the destination, copy regardless
--ignore-checksum Skip post copy check of checksums
```

# 3. S3からR2への全データコピー

## rcloneの罨

- ちなみに rclone は `copy` と `sync` コマンドがあるが、
- rsync と違って `sync` コマンドを使うと rsync —delete の挙動となるので要注意
- 一回限りのフルコピーでは `copy` を使うのが安全
- <https://dexlab.net/pukiwiki/index.php?Memo/Linux/rclone#se233937>

# 3. S3からR2への全データコピー

## rclone実行

- 簡単なrcloneのラッパー的なPerlスクリプトを用意してrcloneコマンドを動的に生成して時間・ログ・コストを計測しながら順番にコピー
- パスの階層がそれなりに多いため、アルファベット順の第2階層までをコピーするパスの1単位として 100並列 x 6 = 600並列くらいで徐々に増やししながら実行

# 動作イメージ

```
[ec2-user@xxx]$ time carton exec -- ./rclone_s3_to_r2_sync.pl
2023-02-06T17:05:24 [INFO] start - rclone copy --transfers 100 --s3-upload-concurrency 100 \
--no-check-dest --ignore-checksum --s3-disable-checksum -P \
s3:xxxx-production/0/0-/ r2:xxxx-production/0/0-/
```

# 3. S3からR2への全データコピー

## コピーにかかるR2操作関連のコストチェック

- 巨大なバケットをコピーする時はどれだけのオブジェクト操作でコストがかかるかをS3 Storage Lens等で予め予測しておくが良い
- <https://developers.cloudflare.com/r2/platform/pricing/>
- Class A Operations (ListObjects, PutObjectなど)
- Class B Operations (HeadObject, GetObjectなど) がどれだけかかるかを把握する

# 3. S3からR2への全データコピー

## コピーにかかるR2操作関連のコストチェック

- CloudflareではGraphQL APIが用意されているため、
- 以下の記事で紹介されているように、R2の各種メトリクスをGrafanaでちよくちよく確認出来るようにすると良さそう
- Cloudflare R2 の GraphQL Analytics を Grafana で可視化する
  - <https://qiita.com/khayama/items/305bbf005b99b8ab4d39>



# 3. S3からR2への全データコピー

## NAT Gateway課金の罠

- バケットコピー作業開始当初、素朴にrcloneを開始したものの、思ったよりもAWS側の料金が跳ね上がってしまう問題が発生
- 調査したところ、作業用のEC2がプライベートサブネットに属していて、そこからS3をR2にコピーしたため外部Cloudflareへのアウトバウンド通信がNAT Gateway経由になってしまっていた
- これを回避するにはEC2をパブリックサブネットに置くか、VPC Endpointを利用するとNAT Gateway経由にならなくて済むので切り替えを行った

# 3. S3からR2への全データコピー

## NAT Gateway課金の罠

- 以下の記事などが参考になりました。。
  - そのトラフィック、NATゲートウェイを通す必要ありますか？適切な経路で不要なデータ処理料金は削減しましょう
    - <https://dev.classmethod.jp/articles/reduce-unnecessary-costs-for-nat-gateway/>
  - EC2からS3へアクセスする4つのルートとコスト
    - [https://tech.nri-net.com/entry/access\\_routes\\_from\\_EC2\\_to\\_S3](https://tech.nri-net.com/entry/access_routes_from_EC2_to_S3)
- やはりこういう作業をする際は、短期間で一気にコピーを行わず、Cost Explorerを細かくチェックしながらそれなりに時間をかけてやっていくのが良さそう

## 4. キャッシュスーパーページの実装

- Cloudflare Cache APIを使ってキャッシュしている場合はキャッシュのページをCloudflare APIを使って行う必要がある
  - <https://developers.cloudflare.com/cache/how-to/purge-cache#purge-by-single-file-by-url>
- CloudFrontのInvalidationsとは違い、Cloudflareのキャッシュスーパーページリクエスト自体は無料のため、コストは気にせずに素朴に削除されたファイルのURLを適宜キューに入れて投げるように組み込み

# 5. ロギングの設定

- 移行前のCloudFrontでは、特定の拡張子のリクエスト件数などを定期的に集計してDBに入れるといった事を行っていた
- Cloudflareでは Logpush という機能があるため、これを利用 (Logpush はEnterpriseプランのみ使用可能)
- Cloudflare R2 に HTTP アクセスログをプッシュして API で取得するまで
  - <https://dev.classmethod.jp/articles/cloudflare-r2-logpush/>

# 5. ロギングの設定

## Logpush

- Logpushはロギングに必要なルールを正規表現, AND, ORなどで細かく指定可能
  - <https://developers.cloudflare.com/logs/reference/filters/>
- 出力したログはR2に溜め込んでAPIで取得可能
  - 時間を指定して特定のログを取り出す場合
    - <https://developers.cloudflare.com/logs/r2-log-retrieval/#retrieve-logs-by-time-range>

## 6. 整合性チェック

- ここまでの手順でS3からR2へのコピーが完了しており、
- アプリケーションからはS3とR2の両方に書き込みがされているため、基本的にはバケットの中身は一緒になっている「はず」
- が、実際に一緒のレスポンスが返ってくるのか、コピー漏れやファイルの欠損などは無いのかの整合性の確認を行うようにしてみた

# 6. 整合性チェック

- Workersは複数のカスタムドメインをマッピングする事が可能
- 本番とは仮のドメインを用意して、事前に見比べ出来るようにテスト
  - production.example.net (本番用ドメイン)
  - production.example.com (整合性チェック用ドメイン)
- バケット内のURLをランダムに生成して実際のレスポンスコードや`Content-Type`、`Content-Length`などの比較を行うスクリプトを用意
- この整合性チェックにより一部バケットのコピー漏れを検知出来ました…

# 7. 切り替え (NSの委任)

- ・ 停止メンテナンスをしないで済むように、Partial (CNAME) setupを利用して切り替えを行った

## 1. Partial (CNAME) setupで本番用ドメインを認証させる

- 元々Route53を利用していたので、Route53側で `cloudflare-verify.example.net` のようなTXTレコードを作成

## 2. Cloudflare側のDNSに予め既存のRoute53と同じレコードをすべて設定しておく

- Route53側のCloudFrontのAliasレコードと同じレコードをCloudflare側にもCNAMEで登録しておく。この際、Proxiedではなく「DNS only」とする

## 3. Workersの Custom Domainを設定



# 7. 切り替え (NSの委任)

4. Workersの `wrangler.toml` 内の `custom_domain` も設定し、`wrangler publish` でデプロイする

5. Workersの本番用Custom Domainが `Initializing` から `Active` に変わるのを待つ

- Activeになったら、curl等で確認

- `curl -I -X GET 'https://hoge.example.net/test.png' --resolve 'hoge.example.net:443:${cloudflare-IP}'`

6. ここまで問題無かったら、NSの委任作業を行う

- `CNAME Setup` から `Full Setup` に変更する

- Route53側にNSレコードを作成して、Cloudflareに委任

- 事前に入れておいた cloudfrontのレコードを workers に変更する。この際、`DNS only` から `Proxied` に変更

7. `wrangler tail` でログを監視する

# 7. 切り替え (NSの委任)

万が一の場合の切り戻し

- ・ もし切り戻しを行いたい場合は、Cloudflare側DNSの Workers の CNAMEを元の cloudfrontのALIASレコードに変更 (かつ `Proxied` を `DNS only` に) することで切り戻しが可能

# 8. 残作業

- Cloudflare R2 + Workers環境に切り替え後、しばらくの期間並行稼動で様子を見つつ、切り戻しの必要が無くなったと判断出来たタイミングになったらアプリケーション側のS3とR2へのダブルライトの仕組みを廃止し、R2のみ書き込みのシングル構成に変更する
- 必要があればライフサイクルルール同等処理の組み込みなどを検討
- S3バケットやCloudFrontのディストリビューションなど、関連するリソースのアーカイブや掃除を行う
- 以上で移行作業としては終了になります

小ネタ

## 小ネタ1. LWP (libwww-perl) の UserAgentがデフォルトで拒否されている問題

- Perl使いとしては最初知らずにちょっとハマった点を紹介します。
- Cloudflare全般のデフォルト設定として、Perlの標準HTTPクライアントであるLWPが拒否されてしまっていますw

```
$ curl -X GET -I https://blog.cloudflare.com/
HTTP/2 200

$ curl -X GET -I https://blog.cloudflare.com/ -A libwww-perl
HTTP/2 403

$ curl -X GET -I https://blog.cloudflare.com/ -A libwww-per
HTTP/2 200
```

## 小ネタ1. LWP (libwww-perl) の UserAgentがデフォルトで拒否されている問題

- この問題は Browser Integrity Check という仕組みのようで、その昔スパマーがこのLWPのUAを悪用したケースが多かったんだと推測される…
  - <https://developers.cloudflare.com/support/firewall/settings/understanding-the-cloudflare-browser-integrity-check/>
- Browser Integrity Checkは、Websites - Security - Settings から機能のON/OFFが可能
- もしくは、OFFにするのではなく、特定のUAを許可したい場合はWAFのCustom Ruleから任意のUser-Agentを追加する方法も

## 小ネタ1. LWP (libwww-perl) の UserAgentがデフォルトで拒否されている問題

- が、なるべくCloudflare標準に合わせておきたかったので今回は以下で対応w

```
my $ua = LWP::UserAgent->new;
$ua->agent('curl/7.29.0');
```

なお、Furlなら気にしなくて良いので Furl を使うとシンプルで良いと思います🐪

```
$ perl -MFurl -E 'say Furl->new->get("https://blog.cloudflare.com/")->status_line'
200 OK
```

# 小ネタ2. WebPの対応が簡単に設定可能

- こちらはハマった問題ではないのですが、素晴らしいと思ったので紹介します。
- Cloudflare Polishという機能があり、Pro以上のプランであれば、ポチポチするだけでWebP対応が可能に…！
- <https://developers.cloudflare.com/images/polish/>
- アプリケーション側でWebP対応をしようとする結構大変だったりするので、これは素晴らしい





まとめ

# まとめ

- CloudFront + S3環境から Cloudflare R2 + Workersに移行する方法の一例を紹介しました。
- まだR2はGAされてからさほど経過していないため、予期しない問題などもあるかもしれませんが、コスト削減の観点からすると結構なインパクトがあると思います。
- 今回は一つの事例として紹介しましたが、間違っている点やもっと良いやり方などありましたら教えていただけるとありがたいです。

最後に一句 (人間)

初物も 地道にやれば なんとかなる

# 最後に一句 (AI)



「Cloudflare R2 + Workers環境に移行してみた」というテーマと、  
「色々な取り組みの果てに到達した成果、境地、慢心、環境の違い等について」に沿った俳句を複数作ってください。



1. 移行の果て R2とWorkers 新境地へ
2. 試練乗り越え Cloudflare R2 澄む空気
3. 慢心捨てし R2の世界 新たな道
4. 経験重ね Workers舞い上がる 高みを求む
5. 環境変化 R2の風に 成果の種