スマポでのPostgreSQL導入事例

株式会社スポットライト
CTO 高橋三徳

株式会社スポットライトについて

会社概要

会社名	株式会社スポットライト (Spotlight Inc.)
設立	2011年5月
事業内容	スマポの運営、ハードウェア開発・製造
資本金	5億円（資本準備金含む）
所在地	東京都渋谷区宇田川町（開発拠点）
代表取締役	柴田 陽
主要取引先 （順不同）	株式会社 大丸松坂屋百貨店 株式会社 ビックカメラ

沿革

2011年5月　　創業
2011年9月　　日本初の来店ポイントサービス「スマポ」ローンチ
2012年5月　　伊藤忠テクノロジーベンチャーズから1億5000万円調達
2013年10月　楽天株式会社にJOIN（１００％子会社）
2013年11月　グッドデザイン賞受賞

“スマポについて”

はじめにスマポについてご紹介します。

来店ポイント スマポ

ユーザー様から

来店するだけでポイントを獲得
貯めたポイントはお得な得点に交換！

新しいお店を発見！

来店するだけで
ポイントが貯まる

お店のポイントカードや各種商品券等に等に交換が可能
例えばビックカメラならば1スマポ-1ビックポイントに。

来店ポイント スマポ

お店側から

お客様に来店動機を与える新しい店舗集客サービス

1.店舗を認知

自分におすすめされている
店舗を探す

2.実際に来店

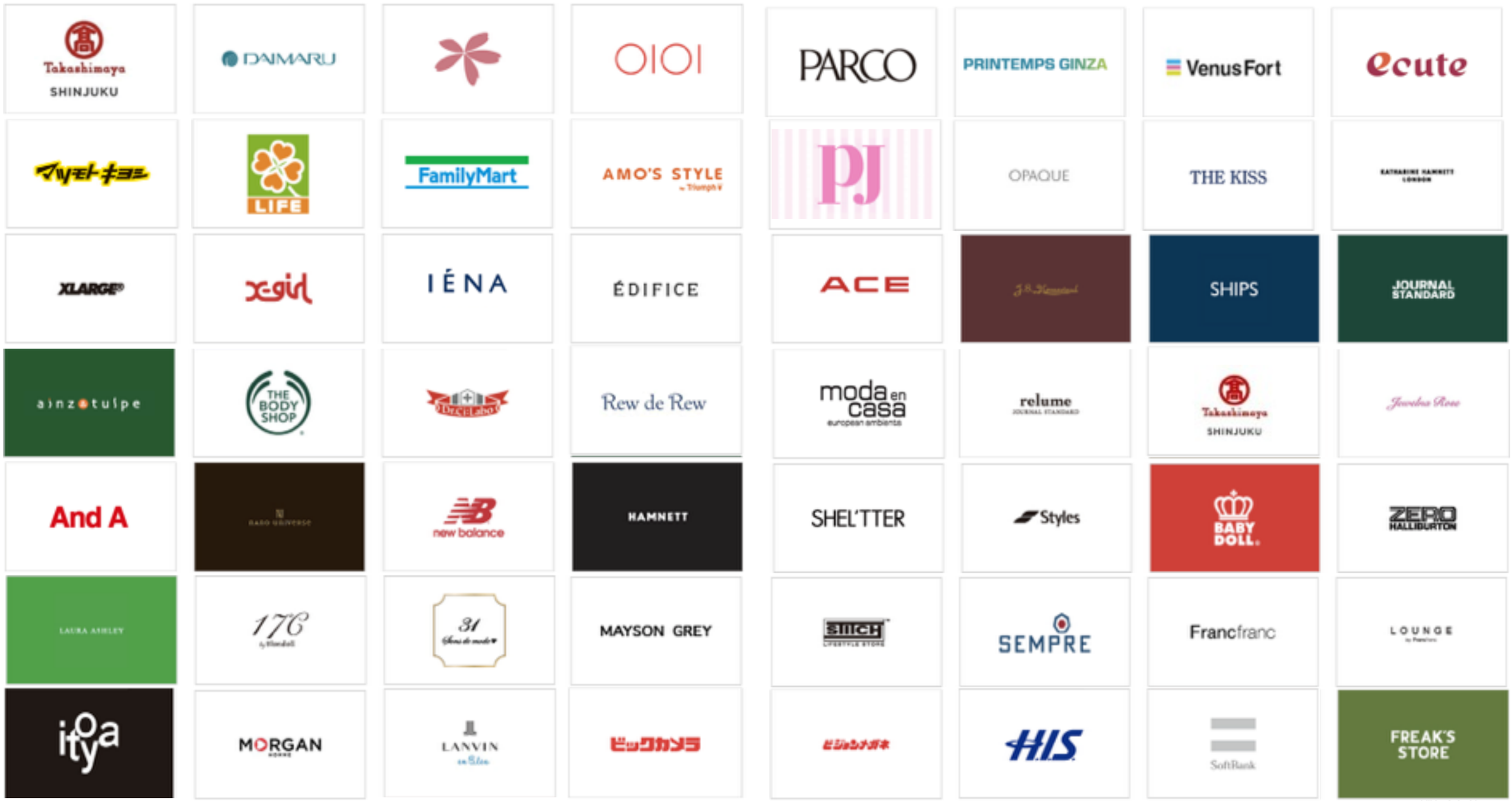
超音波ビーコン

来店するだけで
共通ポイント
が貯まる

来店ポイント スマポ

約100ブランド全国700店舗で利用可能

2013/11現在



（順不同）

来店ポイント スマポ

スマポの独自開発の技術

従来のGPSやWi-Fiでは実現できない、
ユーザーの来店を正確に検知できる技術を開発

ＧＰＳやＷｉＦｉではユーザーの位置情報を正確に把握する精度が低く、
また同じ建物内の店舗を識別することは困難だとされてきました。

しかしながら弊社は人間の耳には聞こえない超音波を発信する
独自のデバイス「スマポシグナル発信機」を開発することで、
正確にユーザーの来店を検知することを実現させています。

こちらの詳細については下記をご参照ください
http://www.slideshare.net/mistakah/gpsgnss

来店ポイント
スマポ

Location Base サービス

＋

独自開発の超音波ビーコン

(”確実”に来店したという事が分かる)

＋

共通ポイント

＝

スマポ

来店ポイント
スマポ

“スマポでのPostgreSQLの利用”

spotlight

スマポ x PostgreSQL 選定から導入まで

2011/8　本格開発開始

当初はMySQLやMongoDBで開発を進めていたが

リリース直前でPostgreSQLに変更

→GIS機能が充実、信頼性採用の決め手

2011/9　サービスLaunch

PostgreSQL 9.0/PostGIS1.5 採用

2012/1　9.1にupgrade

Hstore型を利用開始

2013/9　9.2/PostGIS 2.0 にupgrade

JSON型を一部で利用開始

現在　9.3を検証中

PostGIS　現在地周辺の店舗を検索

あらかじめ登録されたユーザー属性とGPSで取得した位置情報を元に近い順に店舗をソートして表示
付近の店舗取得し地図上にプロット

- 現在位置から近い順にソート(ST_Distance)
- 地図範囲にある店舗を取得(ST_Within)
- その他条件を保持したテーブルとJOINしてFilter

spotlight

PostGIS　現在位置から住所を逆引き

国土交通省の地図データをインポート。
スマートフォンから送られてくる位置情報を元に
Postgisを使って逆GEOコーディング

※2009年のPGCons発表資料を参考にさせていただきました
　http://www.postgresql.jp/events/pgcon09j/doc/b1-2b.pdf/view

※現在のスマポアプリからなくなってしまった機能ですが、内部的には現在も利用中（代わりに地図が表示されているため）

spotlight

PostGIS　分析での応用

収集したデータとGoogleMapを連携ヒートマップにして分析

店舗の位置や
ユーザーの行動などをプロット

アプリに見える所から、バックエンドの解析基盤まで幅広くPostGISを利用しています。

他DBとのGIS機能比較

MariaDB: The New M in LAMP?

Products that implement GIS

	PostgreSQL	MySQL & MariaDB	MongoDB	Solr	SQLite
Standard feature	PostGIS + Extension	+	+	+	Spatialite
Type: Point	+	+	+	+	+
Type: Geometry (x,y)	+	+	*	-	+
Type: Geography (lat, lon)	+	-	*	-	-
Type: 3D (ish)	+	-	-	-	-
SRID projections	+	-	*	-	+
Query by radius	+	~	+	+	~
Precise decimal math	-	MariaDB	-	-	-
Query by bounding box	+	+	*	-	+
Notes:	Most functions don't support Geography	MyISAM only	WGS84 only Limited function set.		Indexes have to be explicitly JOINed

* Since MongoDB 2.4. This evaluation was done on v 2.0.

~ No, but you can query with bounding box (uses index) AND sort that result set by radius.

http://www.slideshare.net/henrikingo/spatial-functions-in-mysql-56-mariadb-55-postgis-20-and-others

Hstore“りれき”機能の事例１

“りれき”はチェックインやポイント交換履歴などが閲覧できる機能、
リストとして扱いたいが、行単位で保持したい内容が異なる、頻繁な機能拡張
スキーマを決めにくく、データ量が多くALTERも時間がかかるため

→　HstoreにKey/Valueで保存

spotlight

Hstore“りれき”機能の事例２

スマポではExtensionが追加された9.1から導入
前述”りれき”ほか多数のテーブルで採用

Hstoreの利点

- RDBなのにKVSのようにも使える
- INDEX(gist)が効くのでhstore内の検索も高速
- Hstoreの操作関数も多数用意されている

Hstoreのはまりポイント

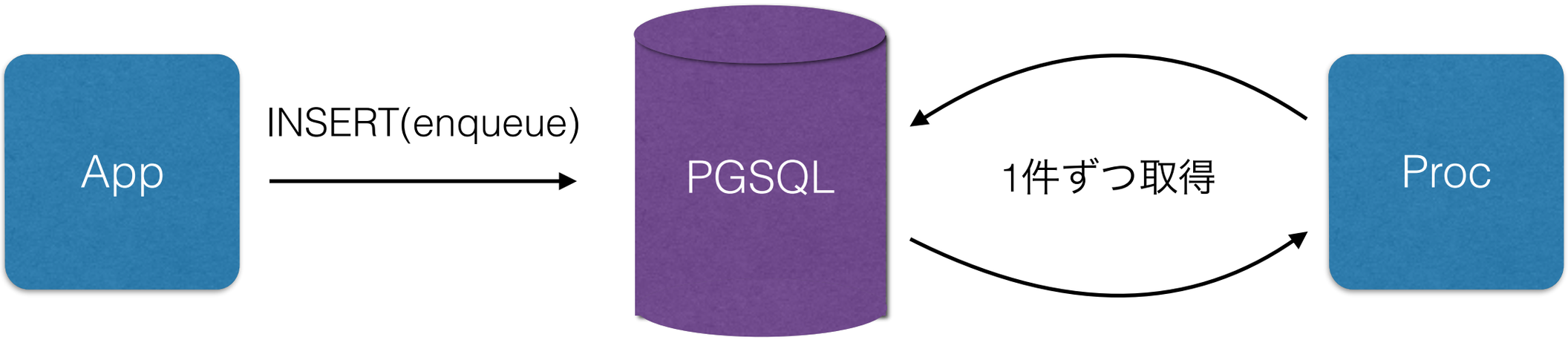
- TEXTベースなので型を保持できない
- 9.2で演算子が替わり一部対応が必要だった

※今後はJSON型も検証して使っていく予定(PostgreSQL 9.3以降のタイミング)

MessageQueue

```
select * from mq
where case
when (schedule is null or schedule <= current_timestamp))
then pg_try_advisory_lock(tableoid::int, id)
else false
end
order by id
limit 1;
```

SELECTクエリ（dequeue)



PostgreSQLベースでQueueを実装

- 信頼性が高い(バックエンドがPostgreSQLなので）
- 同一トランザクションでキューイングと他のクエリを処理できる（２層コミット不要）
- Scheduled Queueも実現

Pythonライブラリを開発、公開しています

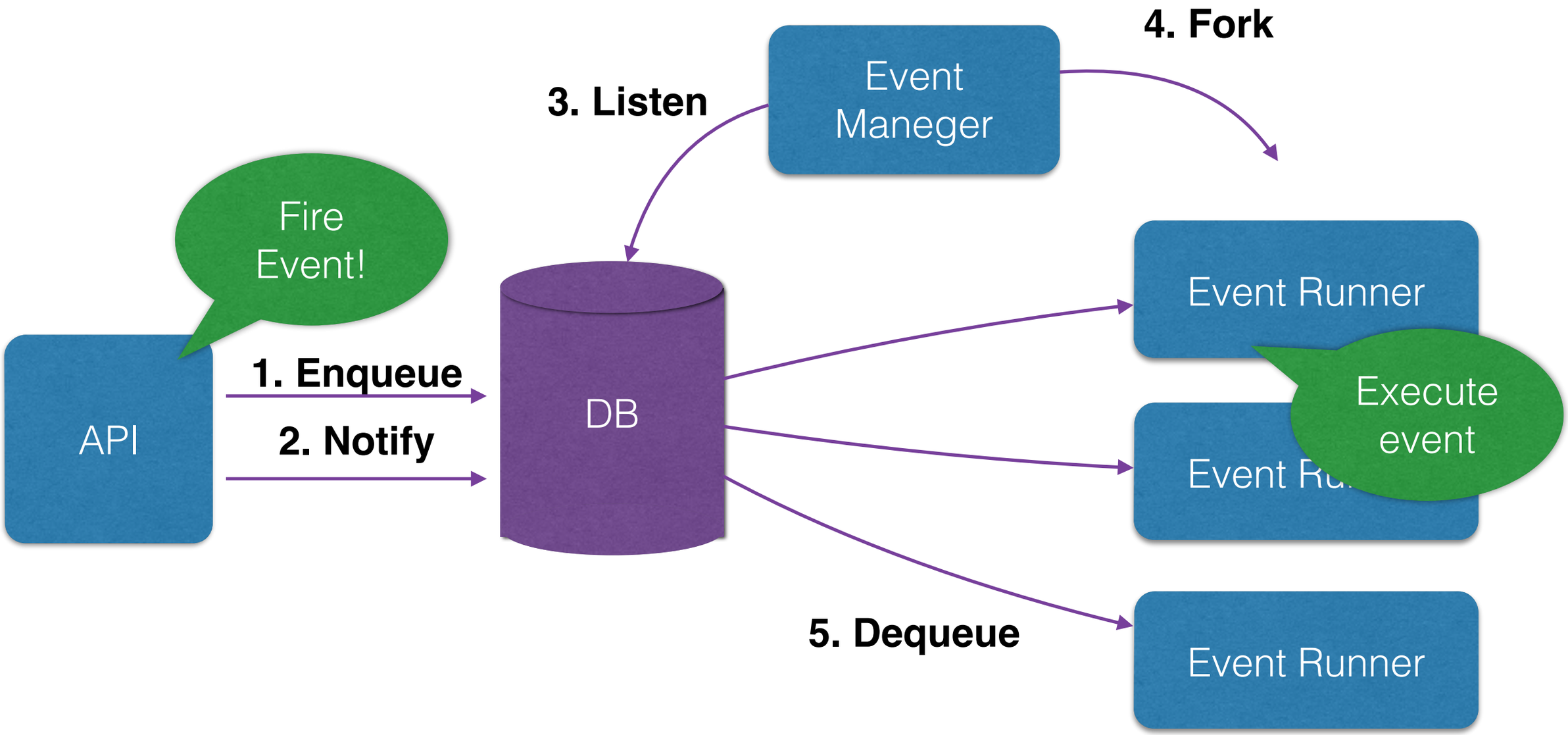
https://github.com/smihica/py-q4pg

※下記を参考に開発

http://d.hatena.ne.jp/n_shuyo/20090415/mq

来店ポイント スマポ

Message QueueとListen/Notifyを組み合わせ



Listen/Notifyと組み合わせてJobQueueとして利用

同期処理の必要の無いタスクは非同期で実行、処理の負荷分散にも利用

spotlight

来店ポイント
スマポ

“開発／運用について”

spotlight

運用：レプリケーションとバックアップ

Streaming Replication

PostgreSQLの標準機能を使ってhot standbyのSlaveDBを運用

WAL-e

HerokuがAWS上でpostgresqlのバックアップを取るために開発していた

Pythonで書かれたスクリプト。OpenSouce化されているの誰でも利用可能

https://github.com/wal-e/wal-e

スマポではWal-eを使ってS3にWALデータをバックアップ

AWS S3／Glacier

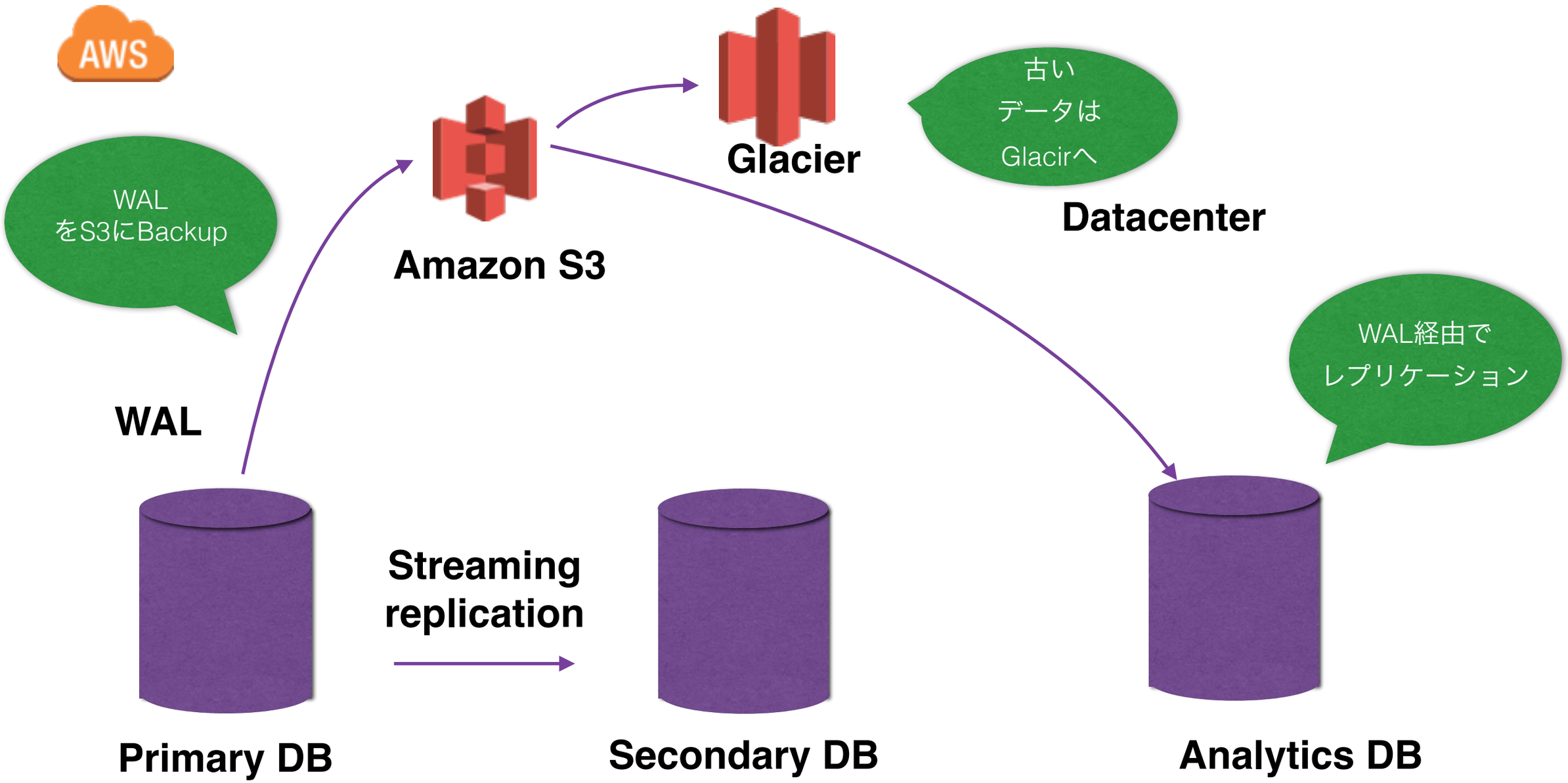
Glacierはアーカイブ向けのクラウドストレージ。0.01ドル/GBと安価、

ただし、アーカイブから取り出すのに数時間かかる

S3と連携することで、一定条件を満たしたファイルを自動的にGlacierに

アーカイブされる事が可能（３ヶ月以上たった古いバックアップなど）

運用：レプリケーションとバックアップ１

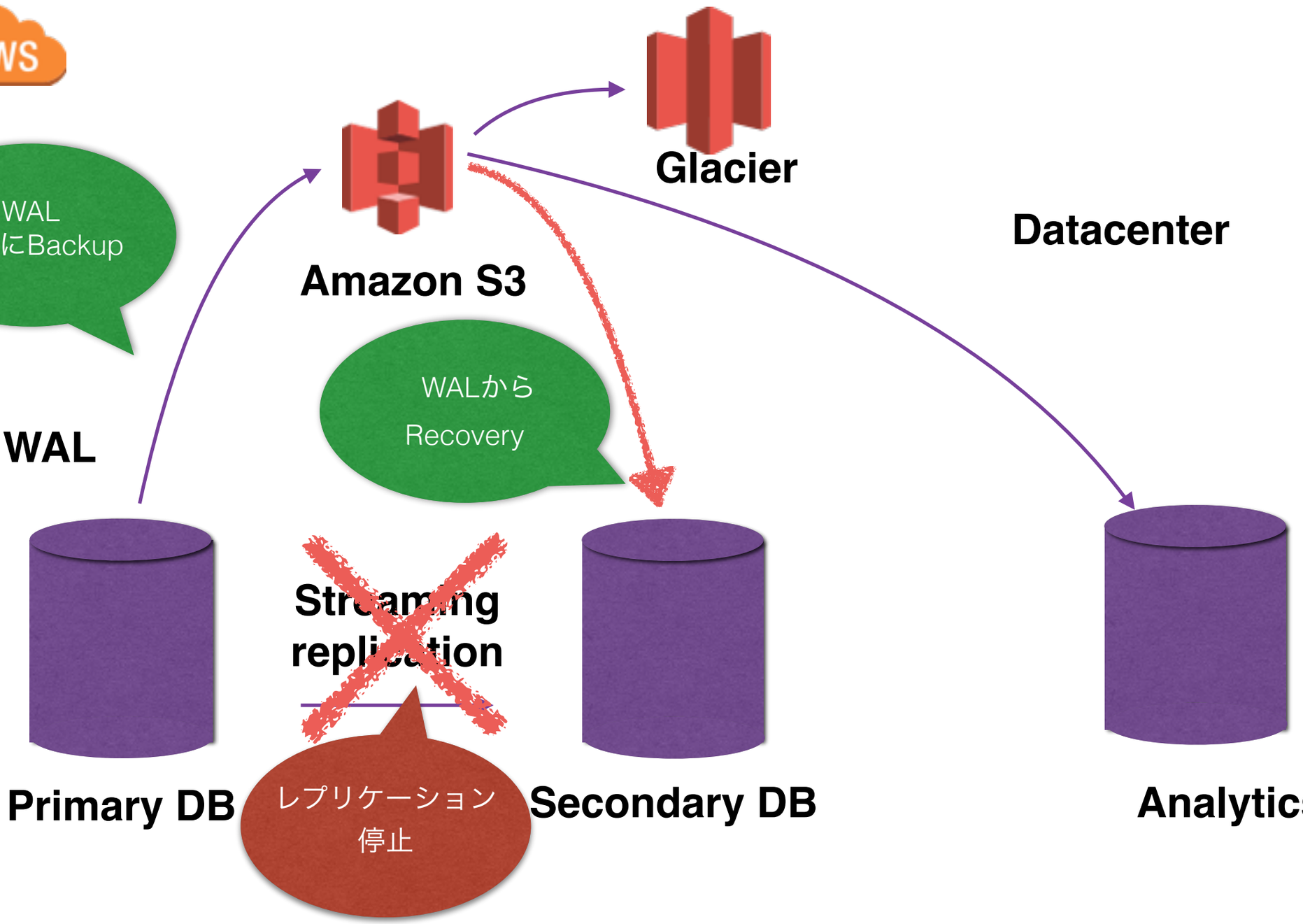


StreamingReplicationで同期、Wal-eを利用してS3にバックアップ

Analytics用のDBは離れたデータセンターにありWAL経由で非同期で更新

古いデータはGlacierに送り、コスト低減

運用：レプリケーションとバックアップ２



StreamingReplicationが途切れても自動的にリカバリ、レプリケーションを継続

Primary DBが落ちても直前のWALからリカバリ

任意のタイミングのデータに戻す事も可能

構成管理ツールのAnsibleで環境整備

Puppet や Chef のようなツール、Python製

あらかじめPlaybookを定義しておくと、それにそって環境を自動構築

同じPlaybookを使い開発者のローカルのVMに本番サーバーと同等の環境を簡単に用意できる

JenkinsによるCI環境

アプリだけでなく、Alembicのmigrationファイルも継続的にテスト

さらにサーバー構成までをテストできるように構築中

※Alembicについては後述

開発：スキーマのバージョン管理

Python製のツールAlembic によりDBスキーマを管理

スマポの開発ではDBに変更を行う場合、原則として
Alembicを利用して更新するようにしています。

☆スキーマのバージョン管理のメリット

1. いつ、どんな変更を行ったのか把握できる
2. DBにどこまで更新が反映されたか分かる
3. DB変更が競合しても
2. スキーマ更新漏れの防止

開発：スキーマのバージョン管理

マイグレーションファイルを作成

```
$ alembic revision -m "Add user table"
$ ls alembic/versions/
1b4b2e259907_add_user_table.py
```

ファイルが生成される
このファイルをgitで管理

DSLかSQLで変更操作を記述

```
def upgrade():
    op.create_table(
        table_name,
        Column('id', Integer, primary_key = True),
        Column('name', String),
    )

def downgrade():
    op.drop_table(table_name)
```

DBを更新するときの操作

変更を戻すときの操作

DBへの反映

```
$ alembic upgrade head
```

AppのDeployと同時に実行
DBを更新する

開発：スキーマのバージョン管理

スキーマがどこまで反映されたか確認

```
$ alembic
$ ls alembic/versions/
INFO [alembic.migration] Context impl PostgresqlImpl.
INFO [alembic.migration] Will assume transactional DDL. Current
revision for postgres:/
37a38bba9dfd (h
```

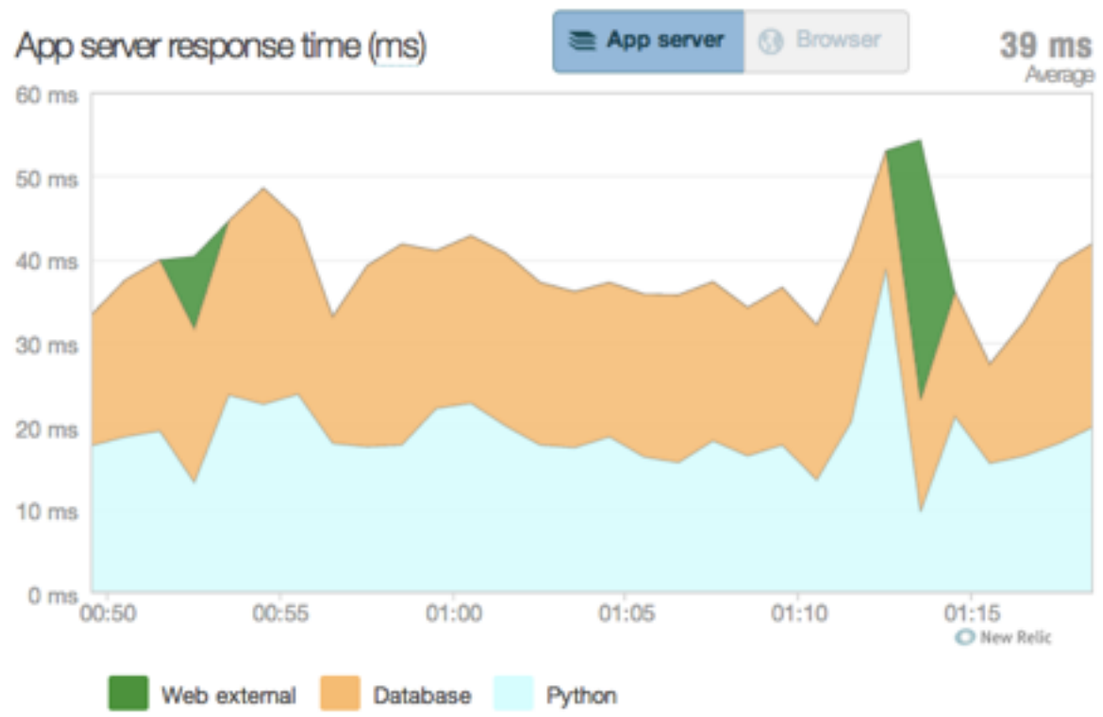
現在のリビジョン

変更の履歴を確認（変更が競合している場合）

```
$ alembic history
4feb86442f82 -> 37a38bba9dfd (head), add u
4feb86442f82 -> 57afagasduw4 (head), add s
39624ab9474b -> 4feb86442f82(branchpoint),
None -> 39624ab9474b, initial create tables
```

DBスキーマへの更新が
競合しているのが分かる

モニタリング



NewRelicでモニタリング

→ クラウドベースのパフォーマンス監視ツール

- クラウドベースなので、監視サーバー不要
- エージェントのインストールが簡単
- 各言語対応(PHP,Perl,Python etc…)

- DB監視

  EnterpriseDB公開しているNewRelicPlugin

  を利用(これには監視サーバーが必要)

Slowクエリに現れにくい性能低下などに

気がつける

“テレビ放映での大量アクセス”

来店ポイント スマポ

チューニング／パフォーマンス

ゴールデンタイムでの全国テレビ放映　大量アクセス事例

9/9 9:00-21:00 NTV 有吉ゼミ

事前の対策

- SQLチューニング

  NewRelicでボトルネックになりそうなクエリを探しチューニング

  適切なIndexを張るなど

- DBパラメーターチューン

  Bufferサイズなど、コネクションプールの調整

- サーバースケールアップ

  深夜にDBを一旦停止、スケールアップして再スタート、Warming

  -Master　:　m3.2xlarge standard 1000 IOPS
  -Slave　:　m1.large

- APPサーバーの増強

- ELBのPreWarming

spotlight

実際のアクセス（ピーク時）

- 約3,000req/sec　(APIアクセス)
- 約3,000 TPS
- 約16,000query/sec

※単純なSELECTだけでなく、INSERTやUPDATEを含んだ数字

結果としてはAppServer側が負荷に耐えられず、数分ダウンしてしまった
DBサーバー側はまだ余力があるように見えた。

PostgreSQLでも十分なパフォーマンスが出せる

2013/11 RDSがPostgreSQL対応をリリース

http://aws.typepad.com/aws_japan/2013/11/amazon-rds-for-postgresql-now-available.html

- 最新のPostgreSQL 9.3のみ
- PostGIS対応
- Hstore/JSON等の
-自動バックアップ／リカバリ機能
- 高性能なIO (最大30,000 IOPS)
- モニタリング

AWSを利用しているならRDSを使わない手は無い！

来店ポイント スマポ

最後に

- 想像以上にパフォーマンスは良い
  7〜８系をイメージしていると圧倒的

- 信頼性が高い
  PostgreSQLに起因するデータ破損は今まで皆無

- 位置情報を扱うならPostGIS
  圧倒的な機能数、ドキュメント、パフォーマンスも十分

- データ型が豊富
  Hstore, JSON型、配列型など便利

- AWSで簡単に構築できるようなった(RDS)

spotlight

※もっと深い話は先月のdb tech showcaseの資料をご覧下さい

http://www.slideshare.net/yoshiyukiasaba/dbtechshowcase2013-smapo