# クラウド・SaaS型
# 統合基幹業務システム
# 「CAM MACS」を支える
# PostgreSQL
# ~雲に乗ったゾウ~

第7回PostgreSQLエンタープライズ・コンソーシアムセミナー in 大阪

2014/02/07

# 自己紹介

- 名前： 松崎 学
- 所属： 株式会社キャム
- 職業：ソフトウェアエンジニア
  (プログラマ、インフラ設計・構築・運用管理)
- データベース歴：Oracle 16年
  PostgreSQL 3年
  MySQLも少しだけ

# 株式会社キャムについて

- 事業内容

  SaaS型統合基幹業務システム「CAM MACS」

  の開発および、サービス提供

- AWSテクノロジーパートナー

- 沿革

| | |
|---|---|
| 1993年 6月 1日 | 設立 |
| 1999年 4月 1日 | C/S型 CAM MACS リリース |
| 2007年 4月 1日 | SaaS型統合基幹業務システム CAM MACS 開発着手 |
| 2007年11月 1日 | SaaS型統合基幹業務システム CAM MACS ローンチ |

# 弊社サービス「CAM MACS」について

- SaaS型 CAM MACSの導入実績 34社
  （2014年2月現在）

- 導入実績業種
  製造業、卸売業、小売業、飲食業、
  施工・工事・メンテナンス業、サービス業、
  EC・通販 など

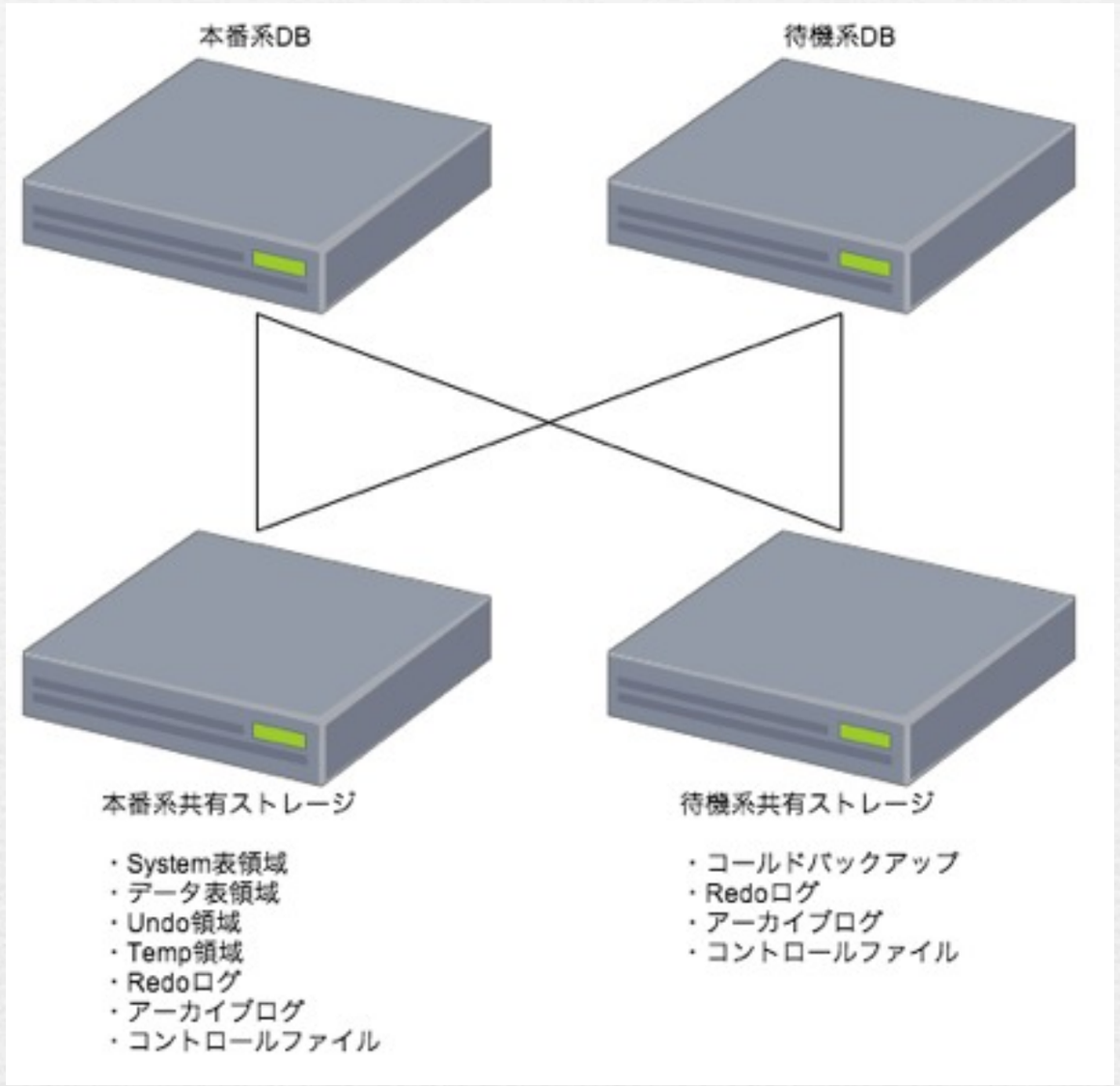
- サービス内容
  経費精算、購買債務、販売債権、製造、
  物流・在庫、原価計算、財務会計、管理会計、
  工事・保守管理、CRM、店舗管理、EC受注管理

# 弊社サービス基盤の歴史

# 第1世代 (2007) ※まもなく移行完了

| | |
|---|---|
| インフラ | 国内データセンター |
| OS | RHEL5 + CentOS5 |
| Webサーバ | Apache |
| APサーバ | Tomcat |
| DB | Oracle 10g |
| 言語 | Java |
| フレームワーク | Seasar2 (Teeda + S2Dao) |
| その他 | DBサーバ用共有ストレージ×2 |

# 第1世代DB構成図



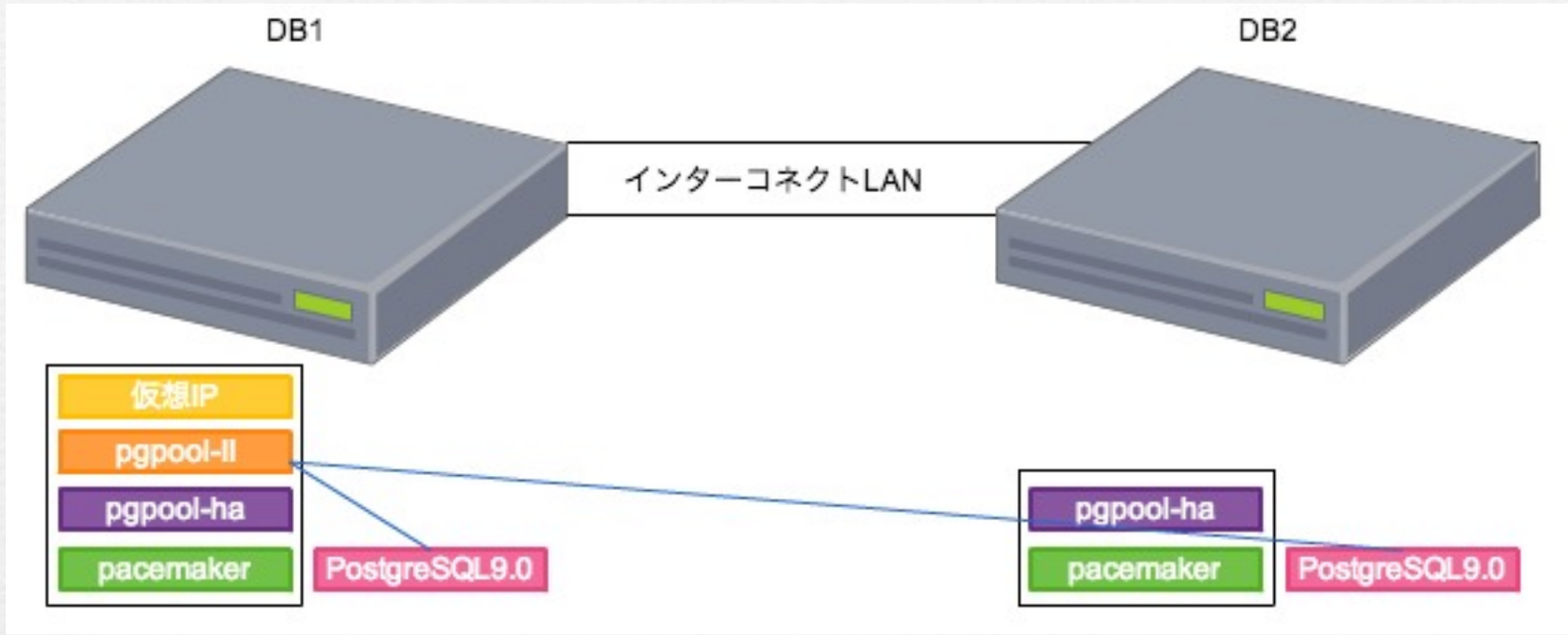

- 定期的に待機系ストレージにコールドバックアップ
- 障害発生時は手動リカバリ
  コールドバックアップ + Redoログ + アーカイブログでロールフォワード

# 第2世代 (2010) ※現在は使っていない

| | |
|---|---|
| インフラ | 国内データセンター |
| OS | Ubuntu10.10 |
| Webサーバ | Apache |
| APサーバ | Tomcat |
| DB | PostgreSQL 9.0 |
| 言語 | Java |
| フレームワーク | Seasar2 (Teeda + S2Dao) |
| その他 | Pacamaker(Ubuntu標準パッケージ) +<br>pgpool-ha + pgpool-II + Orafce |

※黄色は前世代からの変更点

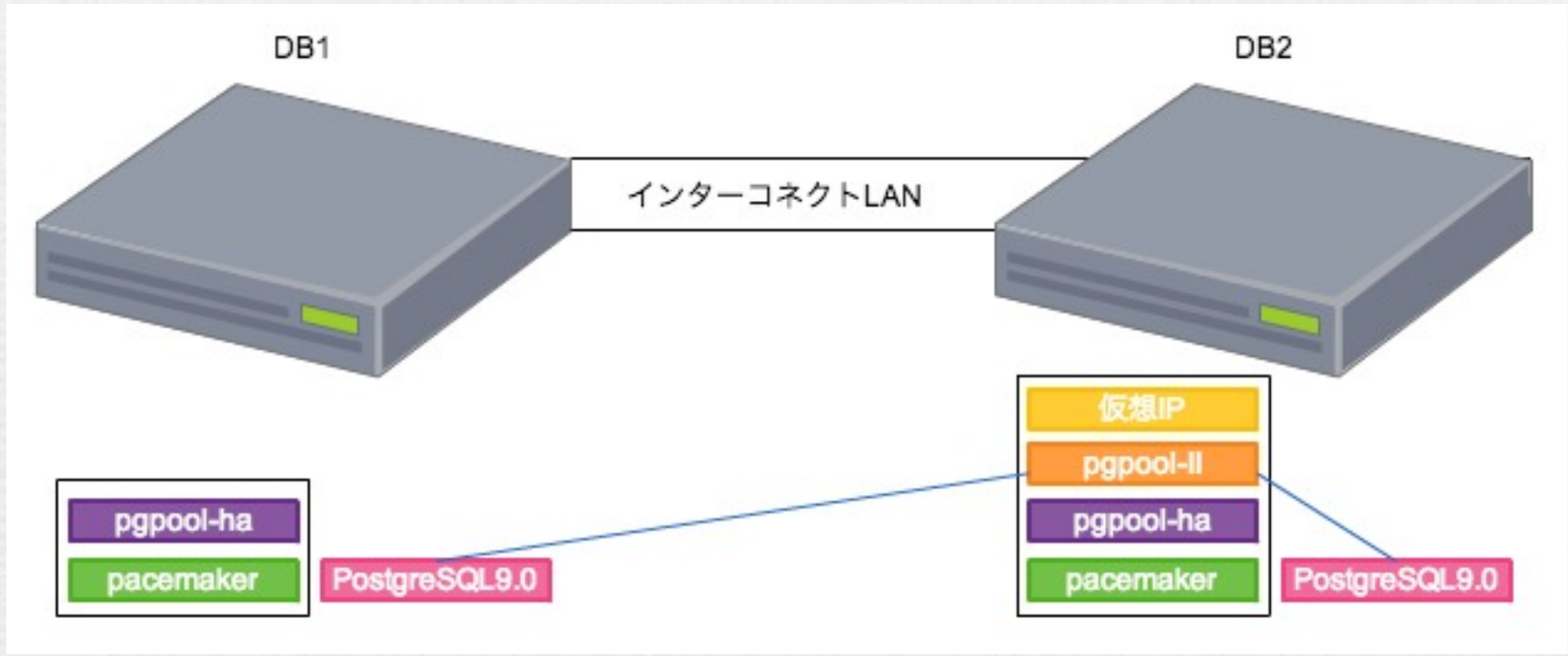
# 第2世代DB構成図




- PostgreSQL9.0には非同期レプリケーションしかなかった
  同期レプリケーションが必要だったのでpgpool-IIを導入
  レプリケーションモードで運用

- PostgreSQLはpacemakerの管理外とした
  pgpool-IIと仮想IPだけpacemakerで管理

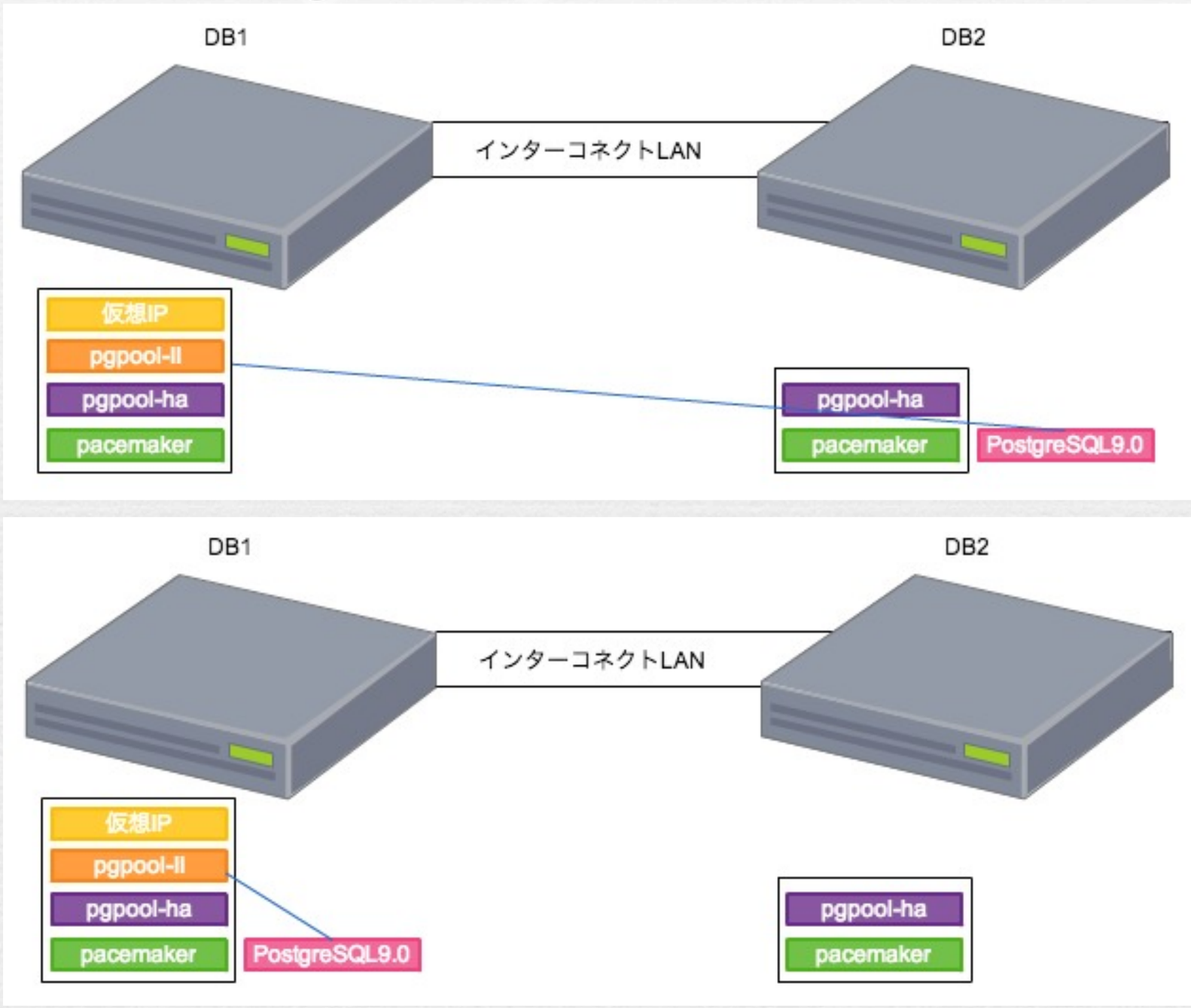
# 第2世代DB構成図
# （pgpool-II障害発生時）




- 仮想IPとpgpool-IIがフェイルオーバー

# 第2世代DB構成図

# （PostgreSQL障害発生時）



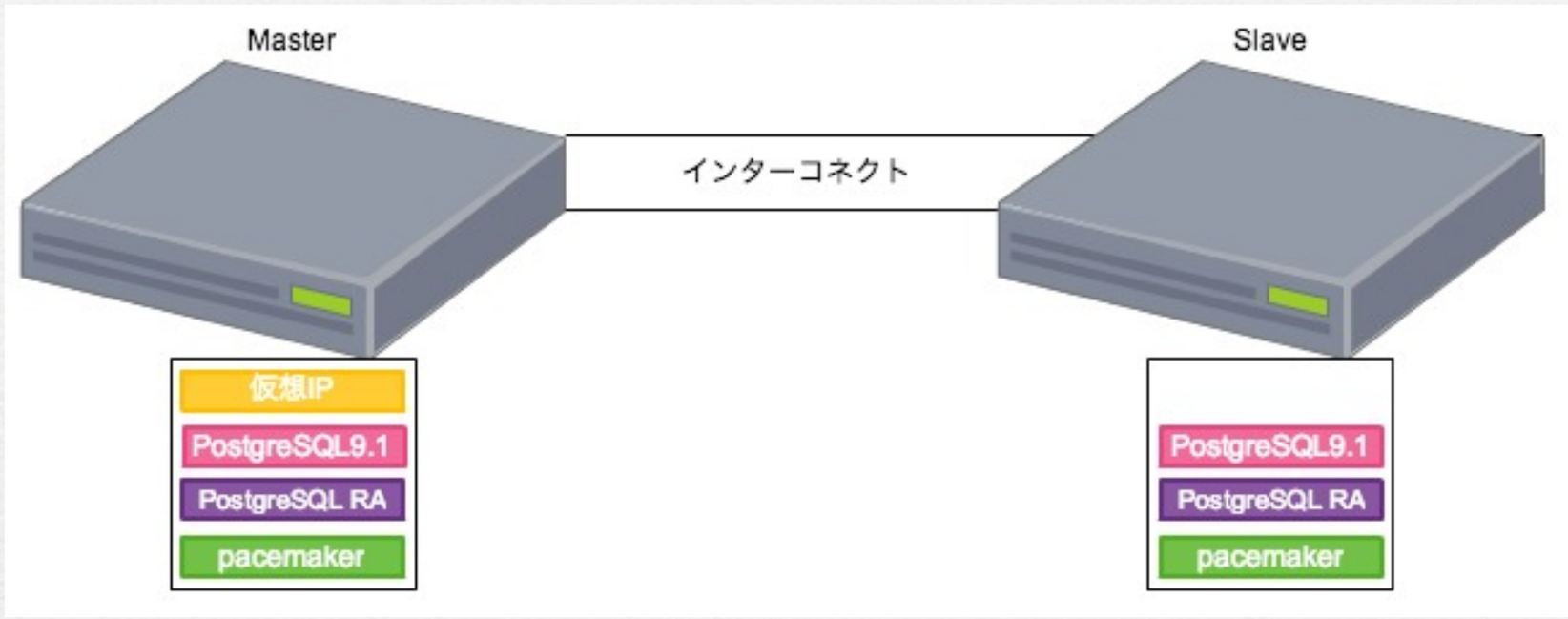

- 障害が発生したPostgreSQLがpgpool-IIから切り離されて縮退運転となる

# 第3世代 (2011) ※現在は使っていない

| | |
|---|---|
| インフラ | 国内データセンター |
| OS | Scientific Linux 6 |
| Webサーバ | Apache |
| APサーバ | Tomcat |
| DB | PostgreSQL 9.1 |
| 言語 | Java |
| フレームワーク | Seasar2 (Teeda + S2Dao) |
| その他 | Pacamaker(Linux-HA Japan提供パッケージ) + Orafce |

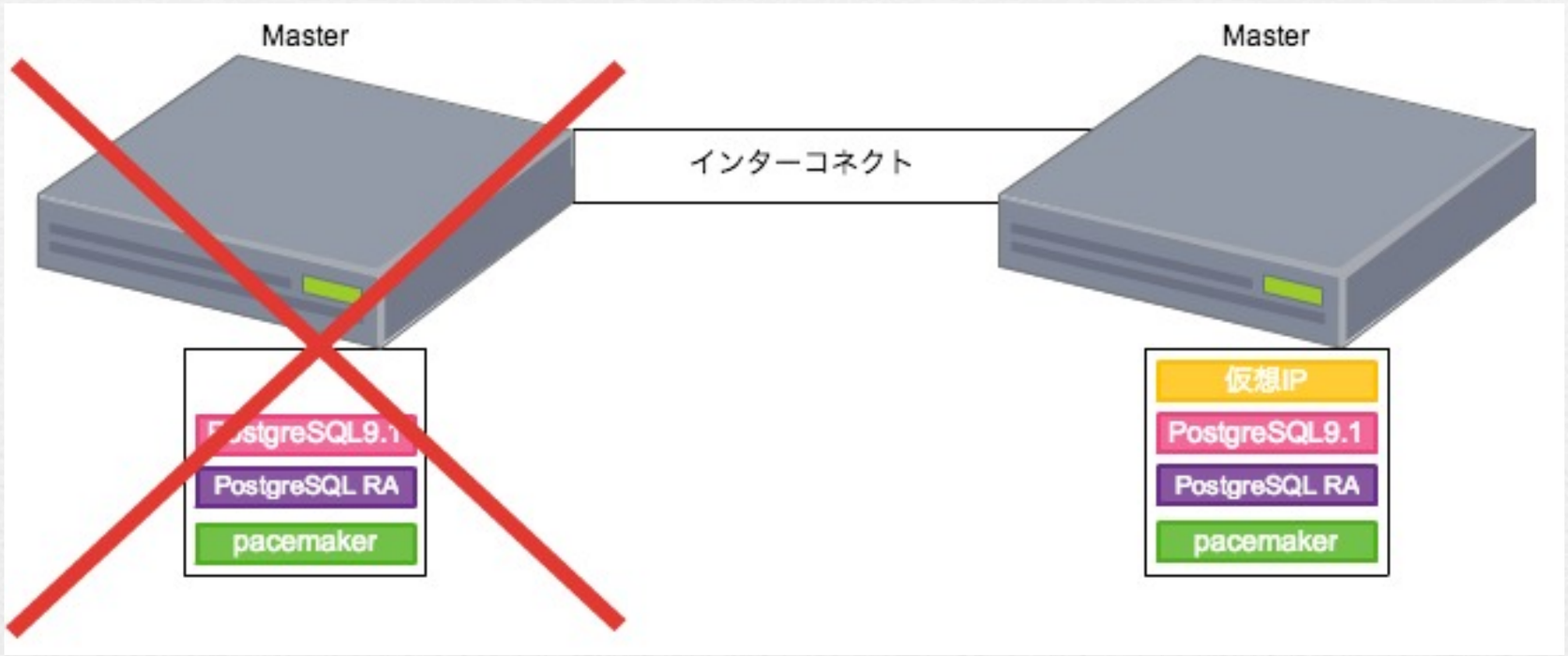
※黄色は前世代からの変更点

# 第3世代DB構成図



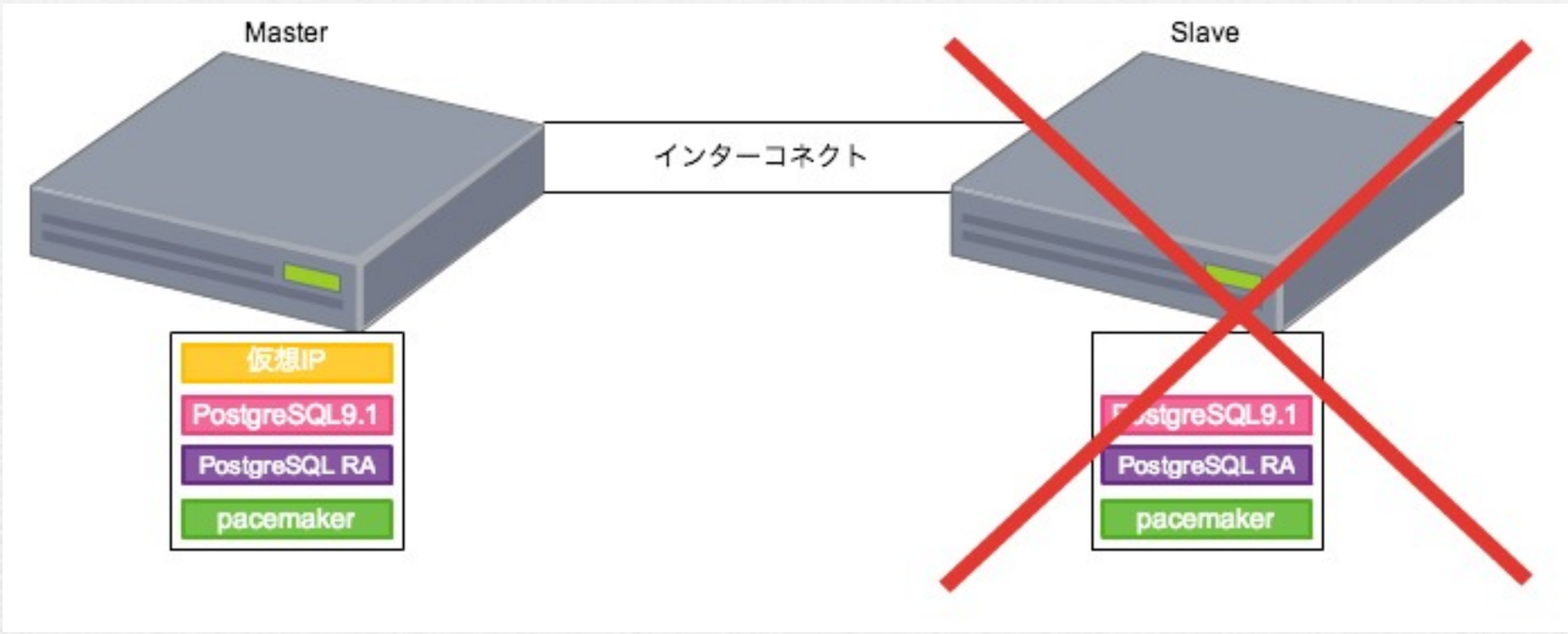

- PostgreSQL9.1で同期レプリケーションが
  追加になったので、pgpool-IIを使わない構成に変更した
  今後PostgreSQLをバージョンアップしていく事を
  考えると、出来るだけ標準機能のみで構成したかった

# 第3世代DB構成図
# (マスタ障害発生時)




- フェイルオーバー

# 第3世代DB構成図
# （スレーブ障害発生時）




- スレーブが切り離されて、
  非同期レプリケーションモードになる

# 第4世代 (2013)

| | |
|---|---|
| インフラ | AWS |
| OS | Amazon Linux |
| Webサーバ | Apache |
| APサーバ | Tomcat |
| DB | PostgreSQL 9.1 |
| 言語 | Java |
| フレームワーク | Seasar2 (Teeda + S2Dao) |
| その他 | Orafce |

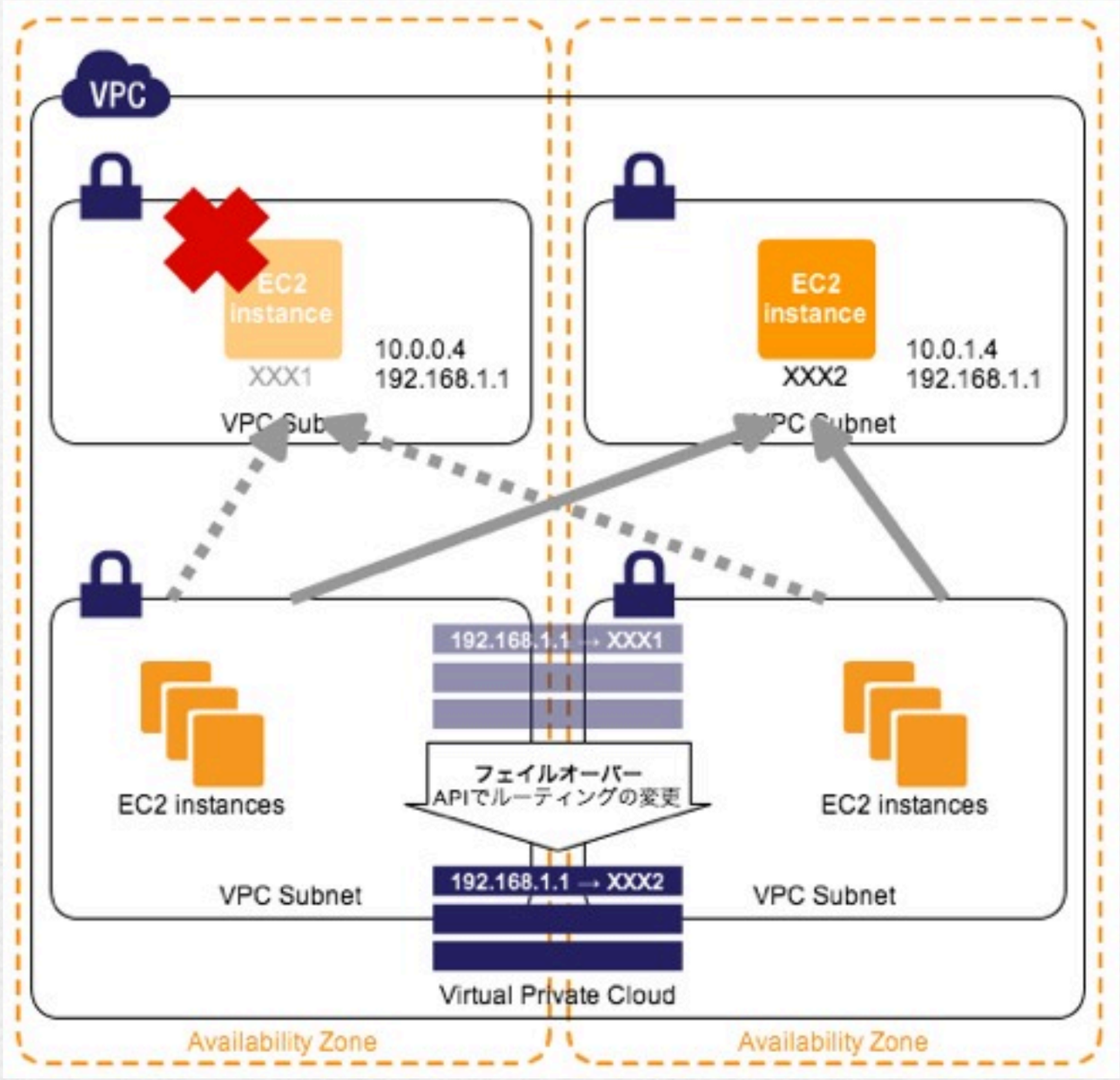
※黄色は前世代からの変更点

# 第5世代 (2013)

| | |
|---|---|
| インフラ | AWS |
| OS | Amazon Linux |
| Webサーバ | Apache |
| APサーバ | GlassFish 3.1.2.2 |
| DB | PostgreSQL 9.2 |
| 言語 | Java |
| フレームワーク | JavaEE6 |
| その他 | なし |

※黄色は前世代からの変更点

# 第4世代、第5世代DB構成図




- CDP: Routing-Based HAパターン

  http://aws.clouddesignpattern.org/index.php/CDP:Routing-Based_HA%E3%83%91%E3%82%BF%E3%83%BC%E3%83%B3

- 弊社が使用しているインスタンスタイプ

  マスタ: C3.2xlarge　スレーブ: C3.large

# PostgreSQLに
# 移行した理由

# コスト面の理由

- 弊社サービスの利用料金は、コストの積み上げでは決めていない

  ソフトウェア費用、ハードウェア費用、保守費用などのインフラに関わる費用は全て弊社が負担している

  コストを抑えないと利益が出せない

# 機能面の理由 (1)

- レプリケーション機能の存在

  Oracle10g Standard Edition Oneを使用していたので、
  レプリケーションがない
  コールドバックアップに掛かる時間が伸びていた
  障害発生時のリカバリにも数時間かかる見込みだった

出来るだけ落ちないサーバ構成（例えばホットスワップHDD、共有ストレージ、FTサーバなど）にするとコストが高い上に、100%無停止は難しい
サーバは壊れる事を大前提とし、
出来るだけ速く自動フェールオーバーする構成に変更

# 機能面の理由 (2)

- パーティショニング機能の存在

  Oracle10g Standard Edition Oneを使用していたので、
  パーティショニング機能がない

  増え続けるデータに対して対応が必要だった

PostgreSQLではなく

MySQLという

選択肢もあったが

# MySQLにしなかった理由

- PostgreSQLの方がOracleとのSQL互換性が高く、
  ソースの改修量が少ない

- 当時のMySQLはNested Loop JOINしか無かった
  ※Nested Loop JOINが遅いという訳ではありません
  （5.6からBKA JOINが追加された）

- 当時のMySQLは4バイトUTF-8を
  サポートしていなかった
  （5.5からサポートされた）

# PostgreSQL移行提案に対する社内メンバーの反応

- 安定性
  - 導入実績は？
  - バグでデータ消えたりしない？
  - 落ちたりしない？
- 性能
  - Oracleと比べてパフォーマンスは？

# 安定性に対する不安を解決するために行った事

- Let's Postgresの事例ページを紹介
  http://lets.postgresql.jp/documents/case
- 「日本PostgreSQLユーザ会 九州支部」の勉強会に参加して情報収集

# 性能に対する不安を解決するために行った事

- アプリケーションの一部の機能をPostgreSQLで動くように改修してOracleと比較

どちらが早いかは一概には言えないと思います

弊社のアプリケーションの場合、
Oracleの方が早いパターン、PostgreSQLの方が早いパターンの両方がありました

・Oracleは細かい事を気にせずSQLを書いてもある程度早い
・PostgreSQLはきちんと書けば早い
という印象です

# 実際に行った
# SQL改修作業について

# SQL改修の方針

- 完全に移行が完了するまではアプリケーションは
  OracleでもPostgreSQLでも動く状態を維持する

- 同じSQLで両方のデータベースで動くのが望ましい

- Orafce(Oracle互換関数拡張)を導入して改修量を減らす
  ※2011/05を最後にバージョンアップしていないので
  今後は出来れば使わない方がよいかも知れません

- 必要に応じてデータベースの差異を吸収する関数を作る

- どうしても同じSQLにする事が出来ない場合は
  それぞれのデータベース毎にSQLを分けて作る

# 改修の流れ

- SQLを改修する前にDaoのユニットテストを全て書く
  - ・仕様化テスト（Characterization Test）
    ※コードの振る舞いを知るためのテスト

    http://devtesting.jp/pekema/?0002%2FTddNextStep

- SQLの改修

  OracleとPostgreSQLの両方でユニットテストがGreenな状態になるように改修する

# SQL改修のパターン

- 基本的な型エラー

（table01.flgがvarcharである場合の例）

```
select col1
 from table01
where flg = 0
```

↓

```
select col1
 from table01
where flg = '0'
```

OracleでもPostgreSQLでも動きます

- rownum → row_number() over()

```
select rownum
        , col1
 from table01
order by col1
```

↓

```
select row_number() over (order by col1)
        , col1
 from table01
```

OracleでもPostgreSQLでも動きます

- 副問合せに対して別名が必要

```
select col1
     , col2
 from (select col1, col2 from table01) t01;
```

OracleでもPostgreSQLでも動きます

- DECODE関数 → CASE式

  Orafceを導入したので、改修不要な気がしたが、

  CASE式に書き換える事にした

  ※decode()に関する注意

  http://orafce.projects.pgfoundry.org/index-ja.html#decode

```
select case
            when col1 = '0' then 'A'
            when col1 = '1' then 'B'
            else 'Z'
        end col1
  from table01
```

OracleでもPostgreSQLでも動きます

- プロシージャはJavaで再実装

  2007年の開発開始当初からOracle以外への移行は
  考えていたので、
    (その時点ではPostgreSQLに移行する予定は
      全くありませんでしたが)
  プロシージャは一切作らない予定だったが、
  スケジュールの関係で一部だけは
  プロシージャを作成してしまっていた

  Javaで再実装するとパフォーマンスが落ちてしまうが、
  サーバのスケールアップで対応

# 改修作業中にわかった事

- PostgreSQLではNullと空文字列が別物
- OracleのDATE型は秒までしか保持できないが、
  PostgreSQLのtimestampはマイクロ秒まで保持
  （ちなみにJavaのjava.util.Dateはミリ秒まで保持）
- ||でnullを結合すると結果がnullになる
  弊社では関数を作って対応したが、
  PostgreSQL9.1以降でしか動かさないのであれば、
  CONCAT関数の利用がおすすめ

- unknown型の型変換エラーが発生
  CASTを追加して対応
  （JDBC固有の問題？psqlではエラーにならない）

```
create cast (unknown as varchar)                        with inout as implicit;
create cast (unknown as text)                           with inout as implicit;
create cast (unknown as timestamp without time zone) with inout as implicit;
```

# データ移行

- まず、PostgreSQLに全てのオブジェクトを
  CREATEしておく
- OracleからCSVファイルを出力し、PostgreSQLに取込む
- シーケンスのcurrval値をデータディクショナリから取得
  して、PostgreSQL側に反映

```
select 'select setval(''' || sequence_name || ''', ' || last_number || ', false);'
  from user_sequences
where last_number > 1
```

↓

```
select setval('SEQXXX', 9999, false);
```

# 運用管理

- ZabbixでOS自体のリソース監視と、
  PostgreSQLの死活監視
- 稼働統計情報を取得
  http://lets.postgresql.jp/documents/technical/statistics/1
- 今後、pg_monzを導入する予定
  PostgreSQLのリソースの可視化

# PostgreSQLに移行して良かったと思う事を弊社開発メンバーに聞きました

# 便利な機能や関数が色々ある

- テーブル継承

  共通的なカラムの定義

- ILIKE

  大文字と小文字を同一視するLIKE検索

- GENERATE_SERIES関数

  UNION不要でレコード生成

- 配列データ型

- 他にも色々あるので、使いこなすと生産性や

  パフォーマンス向上が期待出来る

# まとめ

- Oracleから移行して3年経ちましたが、
  PostgreSQLは安定性も性能も問題ありません
  バージョンアップの度に、性能面も機能面も
  より良くなっていっています

# お薦め書籍

- 個人的に、今はコレがお薦めです
  PostgreSQLの内部構造まで説明されています

  公式マニュアルも非常に充実していますので
  あわせて読みましょう

弊社の事例が

何かのお役に立てれば幸いです

ご清聴ありがとう

ございました