game tech

ゲームのインフラをAWSで！
実戦Tips全て見せます！

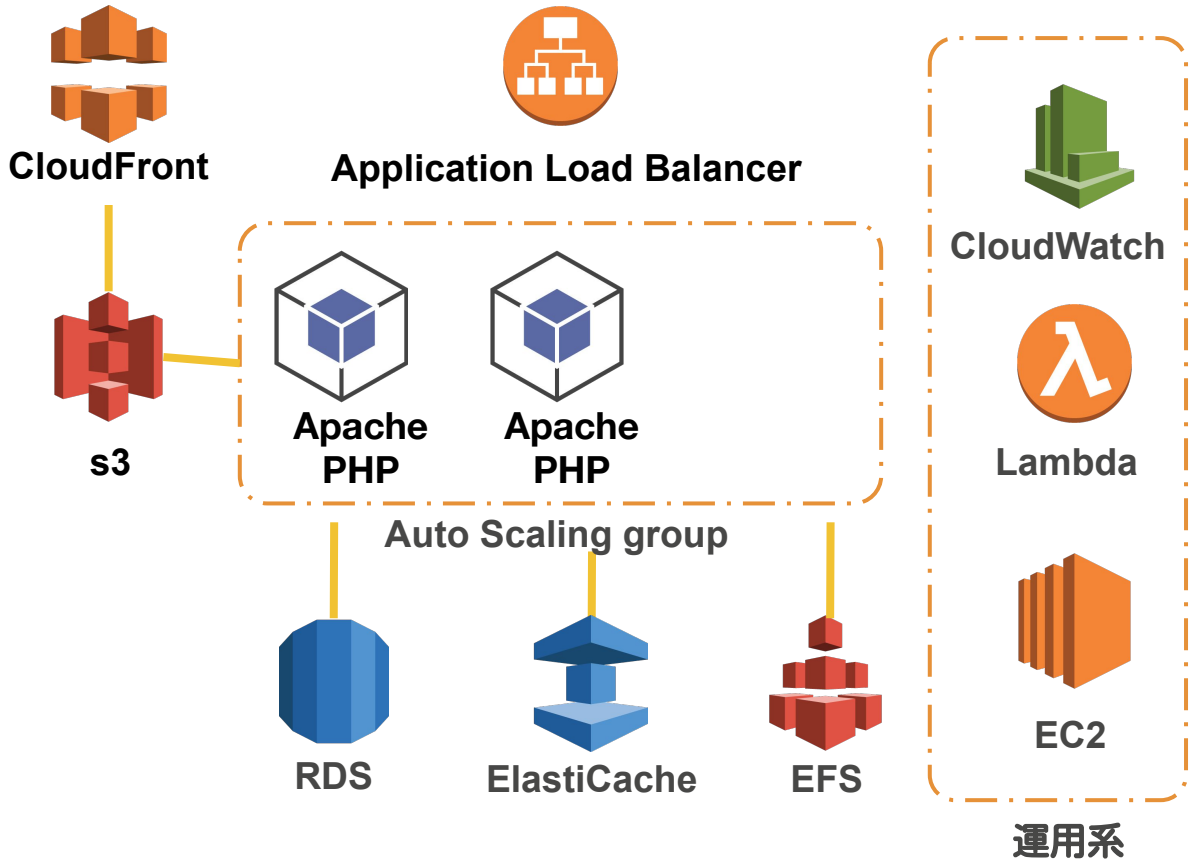
株式会社インフィニットループ
技術研究グループ テクニカルディレクター
波多野信広（twitter:@nobuhatano）

自己紹介

- 商用DBサーバーのサポートエンジニア
- 2012年～ Uターンで（株）インフィニットループ
  - インフラ（MySQL）担当
  - AWS はじめて使う
- 2016年～ 技術研究グループ
- 2018年～ （株）バーチャルキャスト インフラ兼務
- 趣味：SciFi、ゲーム、数理科学、犬の散歩
- 数学勉強会@札幌

HTTPなAPIのためのLAMPスタック編

よく使うアーキテクチャと設計

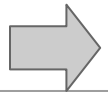
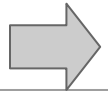
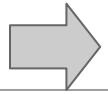
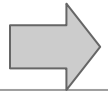


インフラ
運用系ツール

- bash
- awscli
- jq
- ansible
- rsync
- python
- Node-RED
- …
- Slack
- GitHub
- New Relic

ローカルファイル問題

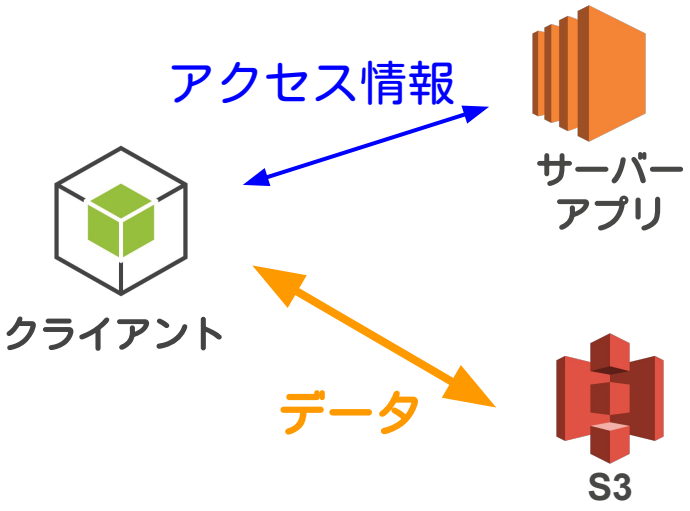
単体サーバー由来のアプリ（例：WordPress）
をクラウド化したい

アプリケーション・コード		サーバー群へデプロイ
セッション		ElastiCache (Redis)
MySQL		RDS
状態持つローカルファイル		**どうする？**

解① s3 を使うよう改修する

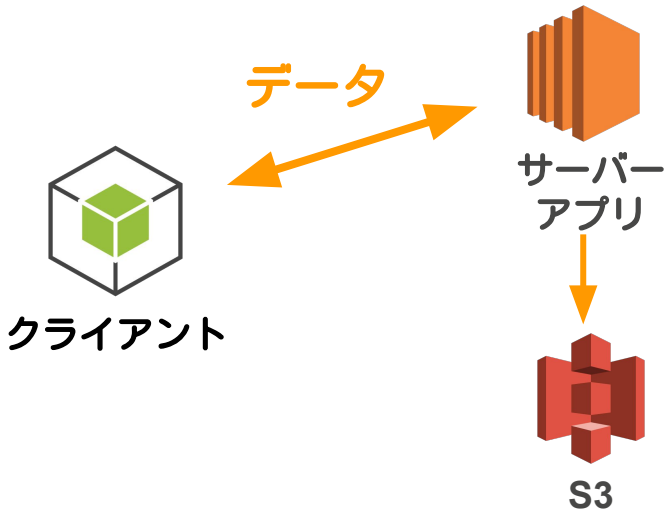
🌞 アクセス情報入手後、クライアントが s3 と直接通信

☂️ クライアントとサーバーアプリ双方の改修必要



🌞 アプリが s3 と通信する

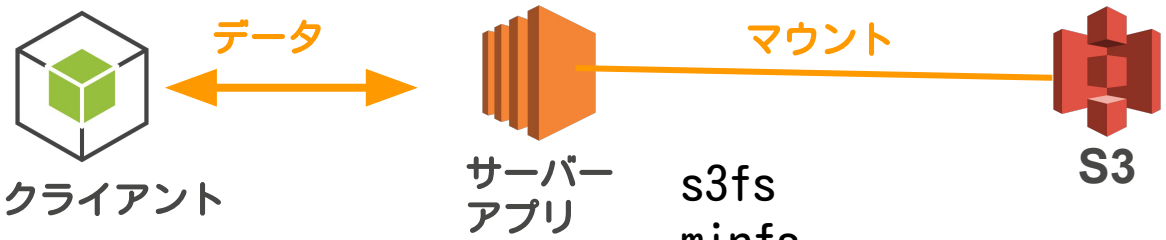
☂️ ファイルアクセスをs3に改修する小コスト



解② s3ファイルシステム

🌞 s3 を直接マウントし無改修

☔ 共有している容量が大きい、ファイル数が多い、更新が多い、接続サーバーが多い、などの場合に大きく遅延する



s3fs
minfs
goofys

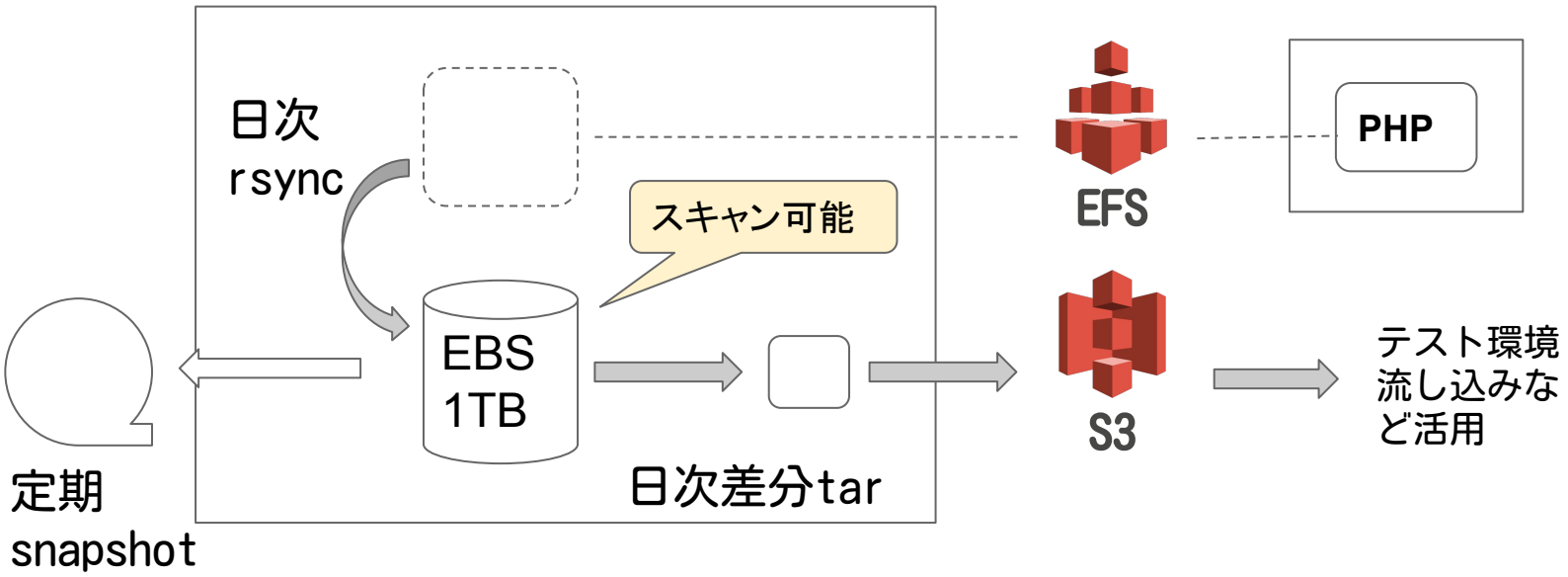
...いいのあれば是非教えてください！

解③ NFS

- 🌞 中央のサーバーを複数サーバーでマウントして利用する王道
- 🌞 2010年代前半まで Kernel のメモリがリークして定期的な再起動が必要だったが近年は安定
- ☔ 冗長化や復旧用にバックアップの自営必要

EFSのバックアップ

- ~~EFSにバックアップ機能が無い~~ （7/8訂正:6/27に AWS Backup がリリースされてましたw）
- バージョニング的に差分を記録したかったのでローテクに自作




```
find /mnt/backupdisc -type f -mtime -${before}
-print0 | tar -czf ${tarfile} -T - --null
```

EC2/オートスケール/ロードバランサー

- 1個のタグでグループや日時バージョンなど属性を入手したい
- AWS 外のリソースで、タグが使えないものにも属性システムを
- ネーミングルールによる自前の属性システム

AMI イメージ名
オートスケーリンググループ　起動コンフィグ名
アプリのデプロイした識別名

app-group-YYYYMMDD-hhmm

アプリ種別	グループ	年月日	時分
api, web	production, staging		

EC2 インスタンス名
オートスケーリング　グループ名
ロードバランサー ターゲットグループ名
Ansible のターゲット名
アプリのコンフィグ名／デプロイ単位

EC2/オートスケール/ロードバランサー

🌞 自分作成の全ての AMI 名と ID を作成時刻の降順で取得する例

```
aws ec2 describe-images --owner self |  jq -r '.[][] | .Name,
.ImageId' |awk -F '-' '{print $1, $2, $3, $4, $5}' | awk
'{if(NR%2){ORS=" "}else{ORS="\n"};print;}' | sort -n -k 3r,4
```

```
api production 20190411 1459  ami 048c37dc4axxxxxx
api staging 20190403 1602  ami 088d809ef52xxxxxx
web staging 20190228 1149  ami 0483b6860xxxxxx
api production 20190226 1350  ami 0b7058826e8xxxxxx
api staging 20190226 1246  ami 0dadb7d8674xxxxxx
          :
          :
```

grep と head で
あるグループの
最新イメージを簡
単に特定可能

EC2/オートスケール/ロードバランサー

🌞 EC2インスタンスとオートスケールグループ、IP を1回でリストする

```
echo "NAME              AUTOSCALE        PUBLIC           PRIVATE"
echo "-----------------------------------------------------------------"
aws ec2 describe-instances | jq -r '.Reservations[].Instances[] |  .Tags[0].Value,
.Tags[1].Value, .PublicIpAddress, .PrivateIpAddress ' | awk '{if (NR % 4) ORS ="
"; else ORS="\n"; print}' | sort -k 1 | awk '{printf "%-16s %-16s %-16s %-16s\n",
$1, $2, $3, $4}'
```

```
NAME              AUTOSCALE        PUBLIC           PRIVATE
-----------------------------------------------------------------
app-dev           null             18.182.xxx.xxx   172.31.xx.xxx
app-production    app-production   13.112.xxx.xxx   172.31.xx.xxx
app-production    app-production   13.231.xxx.xxx   172.31.xx.xxx
                       :
                       :
                       :
```

EC2/オートスケール/ロードバランサー

ec2.py を使った ansible の /etc/ansible/hosts/hosts 設定例

```
[tag_Name_api_production]

[tag_Name_api_staging]

[api-production:children]
tag_Name_api_production

[api-staging:children]
tag_Name_api_staging

[api:children]
api-production
api-staging
```

ec2 で Name タグが “api-production”になっているもの
（ - が _ になるところに注意）

ansible 上でもホスト “api-production” で↑のインスタンス群にアクセス出来るようになる

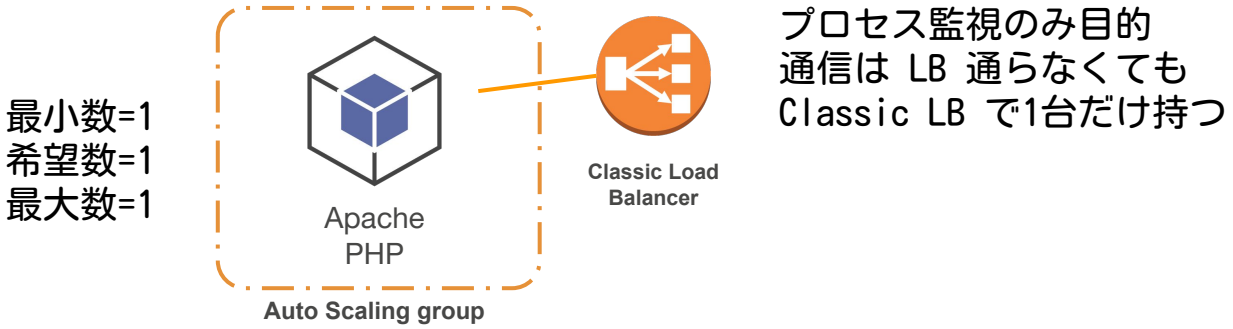
ansible のホスト “api” で production と staging 両方アクセス可能に

ansible でネーミングルールに沿った運用が可能に

EC2/オートスケール/ロードバランサー

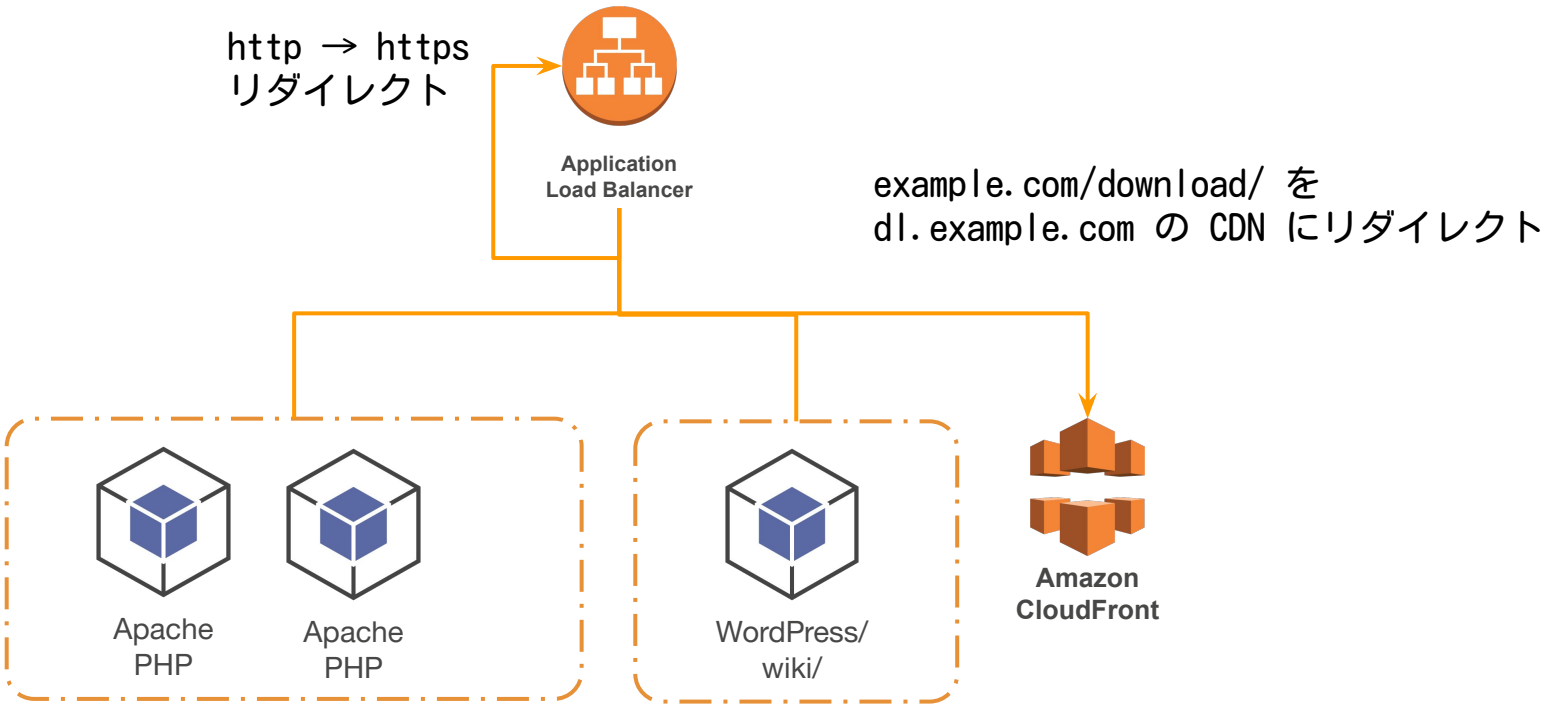
🌞 全ての EC2 インスタンスをどこかのオートスケーリングループに所属させる

1台だけの単目的のサーバーも 1台だけのオートスケーリンググループを作成し、その中に入れて、監視や再起動、自動再追加の保護下に入れます



EC2/オートスケール/ロードバランサー

- Application Load Balancer で複数のオートスケーリンググループや異なるサービスを一つに束ねる



EC2/オートスケール/ロードバランサー

Application Load Balancer でリダイレクトのルールを作成する際
301 リダイレクトか 302 リダイレクト にするか選択できます

ブラウザの 301 リダイレクトキャッシュはデベロッパーツールなど
を使わないと消せないのもあります

リダイレクトは全て 302 で！

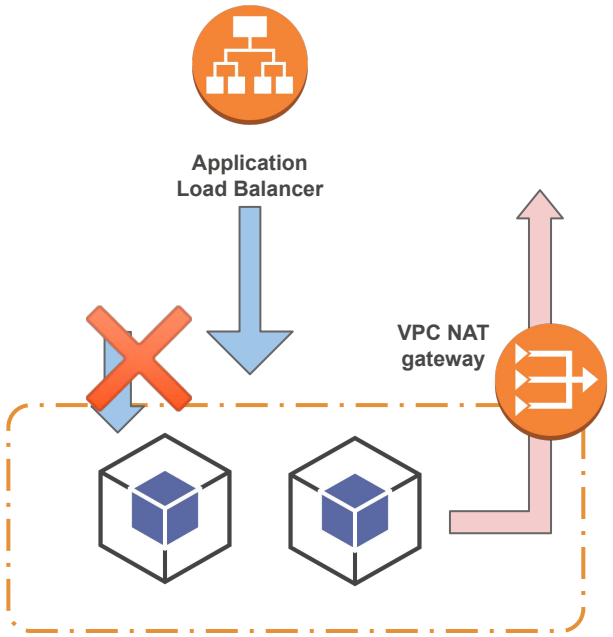
NAT Gateway

Out: NAT Gateway を経由してのアクセスになり IP が固定出来る

In: サーバー直接不可、**LB 経由のみ**

☔ 時間課金。セキュリティ理由だけで使う場合は本当に Security Group で性能が足りてないか確認しましょう

🌞 他サービス API の仕様で動的な app サーバー群も IP アドレス固定化が必要なとき NAT Gateway 有用



RDS / RDS Aurora

🌞 DB 性能・運用に優れた AWS の看板サービス

☔ Aurora は max connections が 16000 という制約があり、軽量なクエリを多数の APP(PHP) サーバーから受ける構成で1万コネクションを超えるケースは普通にあるので注意

RDS / RDS Aurora

🌞 query_cache_size

MySQL 8.0 からはクエリキャッシュは無効

高速 OLTP を支えるゲーム API には
query_cache_type =0, query_cache_size =0 にして完全に停止 or
query_cache_size 10MB など極小の構成

🌞 innodb_flush_log_at_trx_commit = 2

=1 がコミット毎にログとバッファの両方をフラッシュ
=2 のコミット毎ログのフラッシュ & 毎秒バッファフラッシュ
高速 OLTP のゲーム API には =2 で

その他ベストプラクティスは AWS Database Blog の 2019/01/23 の記事参照
https://aws.amazon.com/jp/blogs/database/best-practices-for-configuring-parameters-for-amazon-rds-for-mysql-part-1-parameters-related-to-performance/

ElastiCache（Redis）

- コマンドとデータ型が豊富な揮発性キャッシュとして使います
- 更新がなくて永続性が必要な場合はクエリキャッシュを効かせた単独MySQL(RDS）が適している
- Redis を使いたいケースはレイテンシが命なので、**Web/APPサーバーと Redis の Availability Zone は揃えて使います**
- ネットワーク性能やCPU能力を求めてメモリは余っていてもスケールアップを選択することもある
- シャーディングの Redis クラスタはオーバーヘッドで性能が低下する場合もあるので使う場合は慎重に（例：単純にクラスタで pubsub を使うと1台より性能が低下します）

https://redislabs.com/wp-content/uploads/2018/04/Redis-Day-TLV-2018-Scaling-Redis-PubSub.pdf

ElastiCache（Redis）

- 「クラスタ」 = シャーディングのクラスタ
- 「MultiAZ」= フェールオーバーを可能にするマスタースレーブ
- 「クラスタのみ」「MultiAZのみ」「クラスタかつ MultiAZ 」3通り
- 開発で使うT2ノードタイプは、**MultiAZ のみの構成は取れない**
- 本番の m や r タイプも MultiAZ のみ**不可と勘違い**
- **実は MultiAZ のみ可能！**

- **非クラスタの ElastiCache で MultiAZ にしてプライマリエンドポイントだけをシンプルに使う**

PHP (Apache) / nscd

- 🌞 apache は prefork で良い

- 特定 API だけメモリを喰う処理がある
  - 暗号化等で大きなデータでたまに数百MB使うが、他は数MB
  - 1 httpd 数百MBの見積りだとプロセス数少なすぎ
  - 🌞 MaxRequestPerChild 1000 あえてリスタート頻度を上げる

- PHPサーバーの TCP TIME_WAIT でソケット枯渇
  - さくらのナレッジ「高負荷環境でDBが直面する問題とは？PHPとMySQLの TCP TIME-WAIT チューニング(前編) / (後編)」
    https://knowledge.sakura.ad.jp/author/nobuhiro_hatano/
  - **tcp_fin_timeout は FIN-WAIT-2 用 TIME-WAIT は変わらない**
  - TIME-WAIT を再利用する **tcp_tw_reuse** はあくまで LB 配下 LAMP 前提、万能ではない

nscd

- EC2 上で DNS ルックアップするレートリミットがある
- ルックアップ多すぎ良くないので、自分側でキャッシュする
- DNSの変化に鈍くなるので設定は慎重に

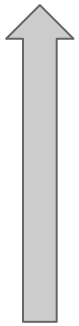
nscd.conf 内の変更箇所例

```
restart-interval        60
enable-cache            passwd          no
persistent              passwd          no
enable-cache            group           no
persistent              group           no
enable-cache            hosts           yes
positive-time-to-live   hosts           15
negative-time-to-live   hosts           5
enable-cache            services        no
persistent              services        no
enable-cache            netgroup        no
persistent              netgroup        no
```

rsync デプロイ


PHPのrsyncの差分同期は稼働中可能で高速


PHP opcacheがrealpathでの変化を検知出来ないため、symlink を使うツール類を使わず自作



対象	デプロイ方法	デプロイタイミング
アプリのソース	ビルド済みディレクトリを rsync で対象サーバーへ同期	コマンド 管理画面からの実行時
アプリのソース	管理サーバー上でビルド tar で固めて s3 に格納	サーバー起動時に必ず s3 から取得して展開する
OS, Apache, PHP 設定	デタッチした稼働サーバーを元にイメージ生成	オートスケールによる自動追加時

- デプロイ毎：管理サーバーでビルド→ s3 にアップ → rsync 同期
- サーバー起動毎：s3 から DL & 展開 → NG なら httpd 停止

CloudFront

CloudFront で実際に配信しているロケーションを調べる方法

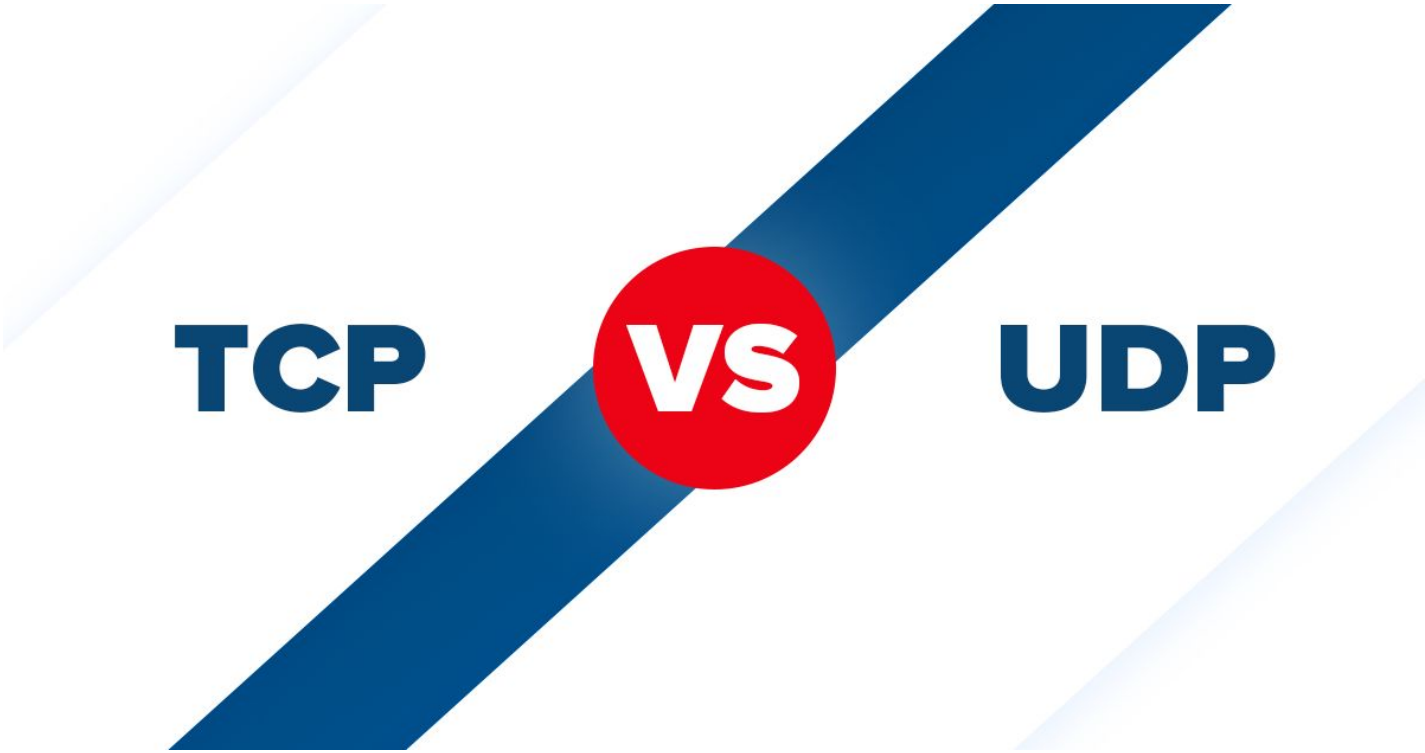
- dig などで今使われている IP アドレスを得る（例 13.33.0.250）
- dig -x 13.33.0.250 などで DNS 逆引きする

```
250.0.33.13.in-addr.arpa. 21112 IN      PTR
server-13-33-0-250.nrt12.r.cloudfront.net.
```

今配信に使われているリージョンに近い空港の3文字コード

（例 NRT 成田）

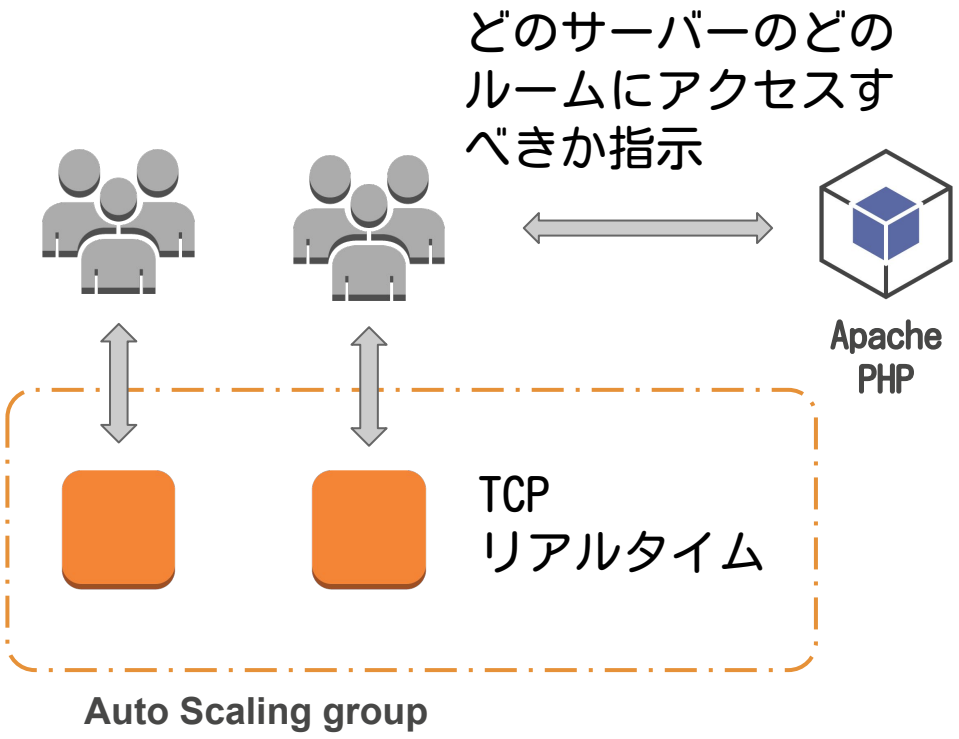
TCP/UDPソケット通信なリアルタイムのインフラ編



https://www.vpnmentor.com/blog/tcp-vs-udp/ より

アーキテクチャ

リアルタイムなサーバーは直接クライアントと1対1で通信する



サービス・ディスカバリ

マッチングサーバー
API 的に接続割り振り

ルックアサイド・ロードバランサー
ラウンドロビンやハッシュなど機械的な割り振り

Amazon Linux 以外の OS

LAMP では Amazon Linux が最適ですが、リアルタイムでは C/C++ で作成されたミドルウェアの要件で Amazon Linux では導入が難しいケースもあります。そんなときおススメするのが

CentOS 7 (x86_64) - with Updates HVM

https://aws.amazon.com/marketplace/pp/Centosorg-CentOS-7-x8664-with-Updates-HVM/B00O7WM7QW

拡張ネットワークを使うためのカーネルモジュールも準備されています。
cloud-init も使えるので使用感は Amazon Linux と似ています。

Amazon Linux と CentOS 7 with update HVM と両方揃えておくと盤石です

EC2 拡張ネットワーク

パケットのレイテンシも問われるリアルタイムの通信では EC2 の仮想インターフェース（vif）ではなく拡張ネットワークを必ず使いましょう

Intel 82599 VF

C3,C4,D4,I2,M4(16xlarge以外),R3 インスタンス

Elastic Network Adapter

C5,M5,R4 以降のインスタンスタイプ

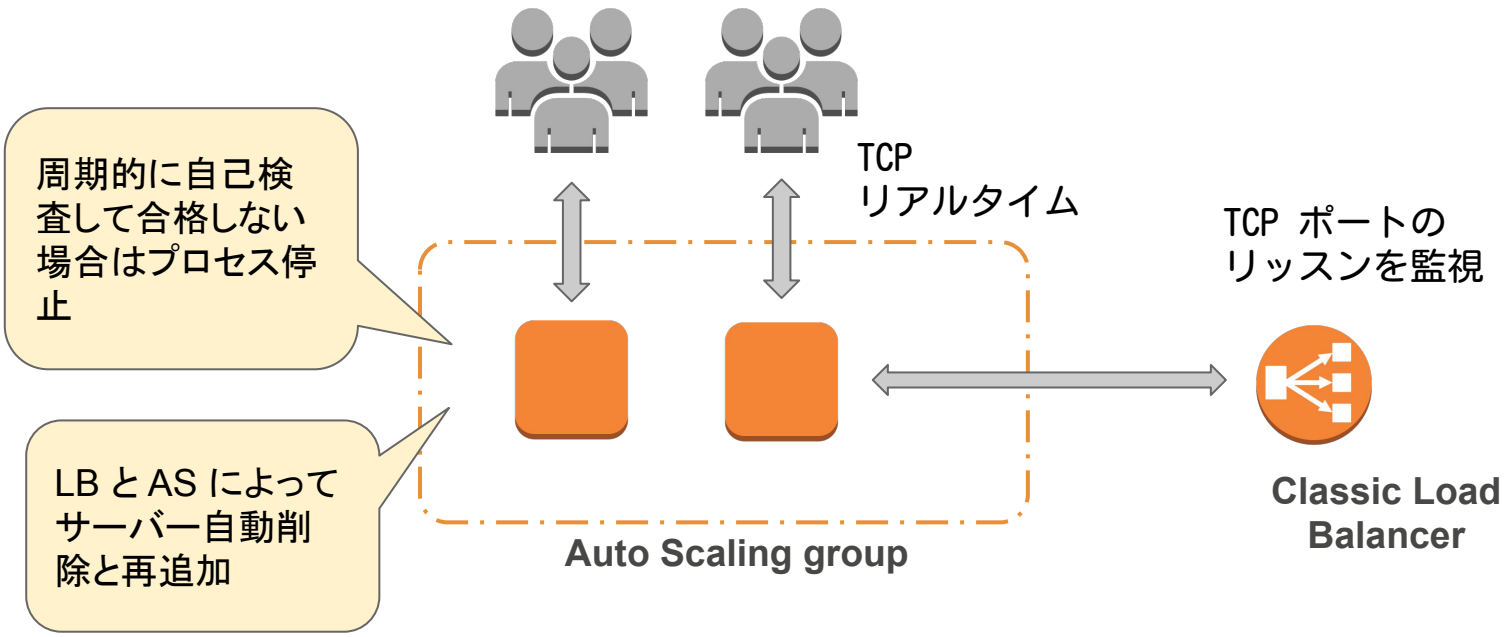
ixdbevf または ena カーネルモジュールを使い、ec2 でも設定

（ena例）

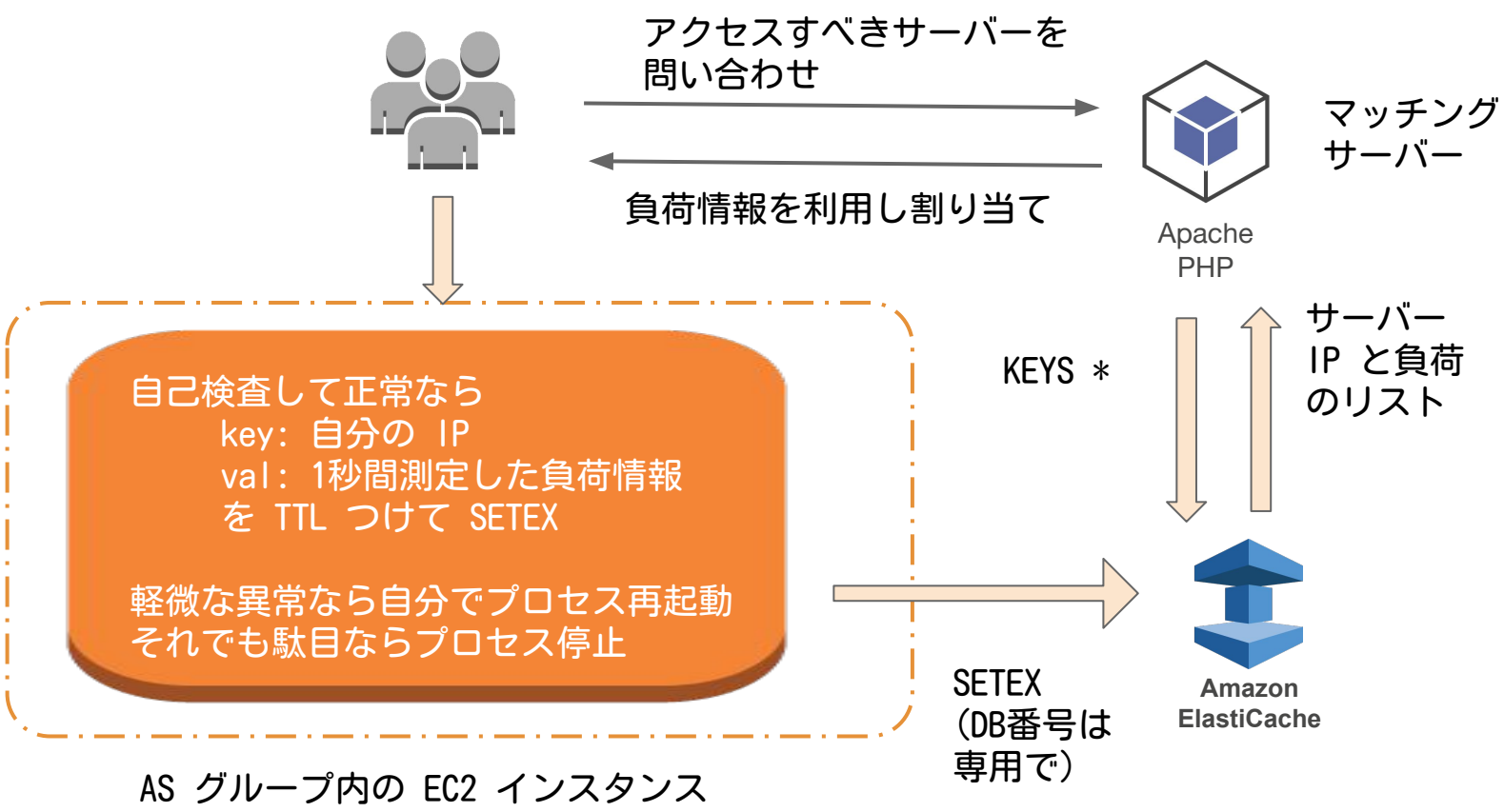
```
aws ec2 modify-instance-attribute --instance-id $id --ena-support
```

LB とオートスケールを可用性で使う

ユーザーとの通信は LB を通りませんが、リッスンしているプロセスのポートを Classic LB で監視させることで障害サーバーの自動削除と再追加を行います



デッドマンスイッチによるスケーリング



まとめ

- AWS は豊富なサービスやツールがありますが好きなサービスやツールを好きな方法で使っています、そうした特徴的なところを今回紹介しました
- CodeDeploy, CloudFormation, CloudWatch, ElasticSearch 等々幅広く利用していますがマニュアルどおりの正しい使い方で今回割愛
- サーバーやインフラのチューニングなどお困りでしたら、お気兼ねなく弊社サイトのお問い合わせページ https://www.infiniteloop.co.jp/mail/ でご連絡ください _(._.)_