aws SUMMIT

PayPay 100億円キャンペーンのシステム構築

PayPay

Zero Hardware, Three Months, Millions of Users

PayPay 100億円キャンペーンのシステム構築

PayPay株式会社
プロダクト本部テクノロジー1部
山本 啓介

PayPay株式会社
プロダクト本部テクノロジー2部
Shilei Long

PayPay

Agenda

- Section 1 : PayPay 100億円キャンペーン
- Section 2 : PayPay Secret Sauce

PayPay

Keisuke Yamamoto

- KSK(ニックネーム）
- PayPay株式会社プロダクト本部

略歴:

2007年 Yahoo! Japan入社

主に決済関連のプロダクトを担当

PayPay

Section 1 Agenda

- PayPayとは
- Timeline
- First Mission ローンチ
- Second Mission
  - 「100億円あげちゃうキャンペーン」（第1弾）
- Third Mission
  - 「第2弾100億円キャンペーン」

PayPay

PayPayとは

PayPay - QR・バーコード決済サービス

PayPay

PayPayが使えるお店順次拡大中！

コンビニ

総合スーパー、ディスカウントストア

ドラッグストア

飲食

居酒屋

家電量販

書籍

ファッション

百貨店、モール

交通

宿泊、旅

その他

店舗オーナー様向け

PayPay

PayPay for Business

PayPay

沿革

- 2018年6月15日
  - PayPay株式会社設立
- 2018年10月5日
  - サービス開始
- 2018年12月4日～12月13日
  - 100億円あげちゃうキャンペーン （第1弾）
- 2019年2月12日～5月13日
  - 第2弾100億円キャンペーン
- 2019年6月1日～
  - PayPayとドラッグストアの強力タッグ！おトクなスマホ決済キャンペーン

ローンチから９ヶ月

- 登録者数 700万超
- 加盟店数 60万超

2019年6月現在

PayPay

職場環境

- 多様性
  - 日本、インド、中国、ケニア、カナダから集まった多種多様なメンバー
- 自由さ
  - フラット
  - スーパーフレックス制
- 最新の開発環境

PayPay


Join us!
Be a Rockstar
Job Openings
scroll

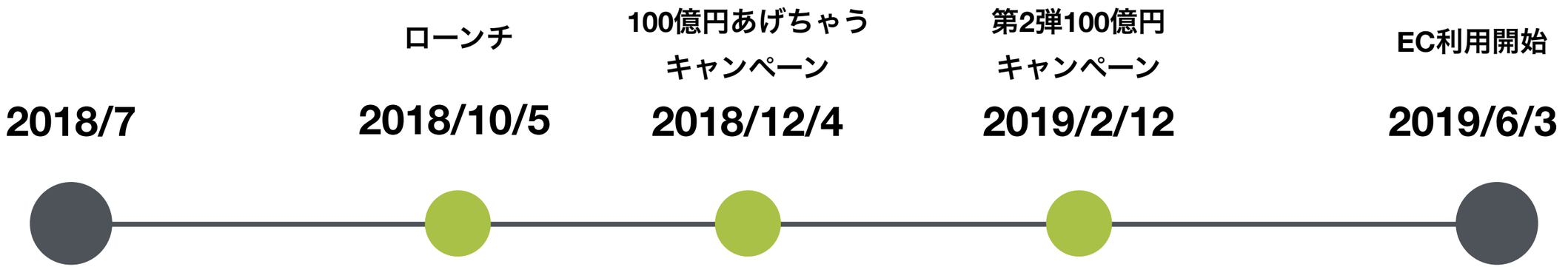
一緒に働きましょう！

ご連絡はこちら

- momoko.hayakawa@paypay-corp.co.jp

PayPay

Timeline


2018/7
ローンチ
2018/10/5
100億円あげちゃう
キャンペーン
2018/12/4
第2弾100億円
キャンペーン
2019/2/12
EC利用開始
2019/6/3

First Mission

- **3ヶ月**でQRコード決済サービスをローンチすること

PayPay

混沌の中で開発を進めるため

- 開発サイクルを上げる
  - → Microserviceアーキテクチャ
- 多数のサービスを効率的に管理
  - → Kubernetes
- インフラ構築の短縮
  - → AWS

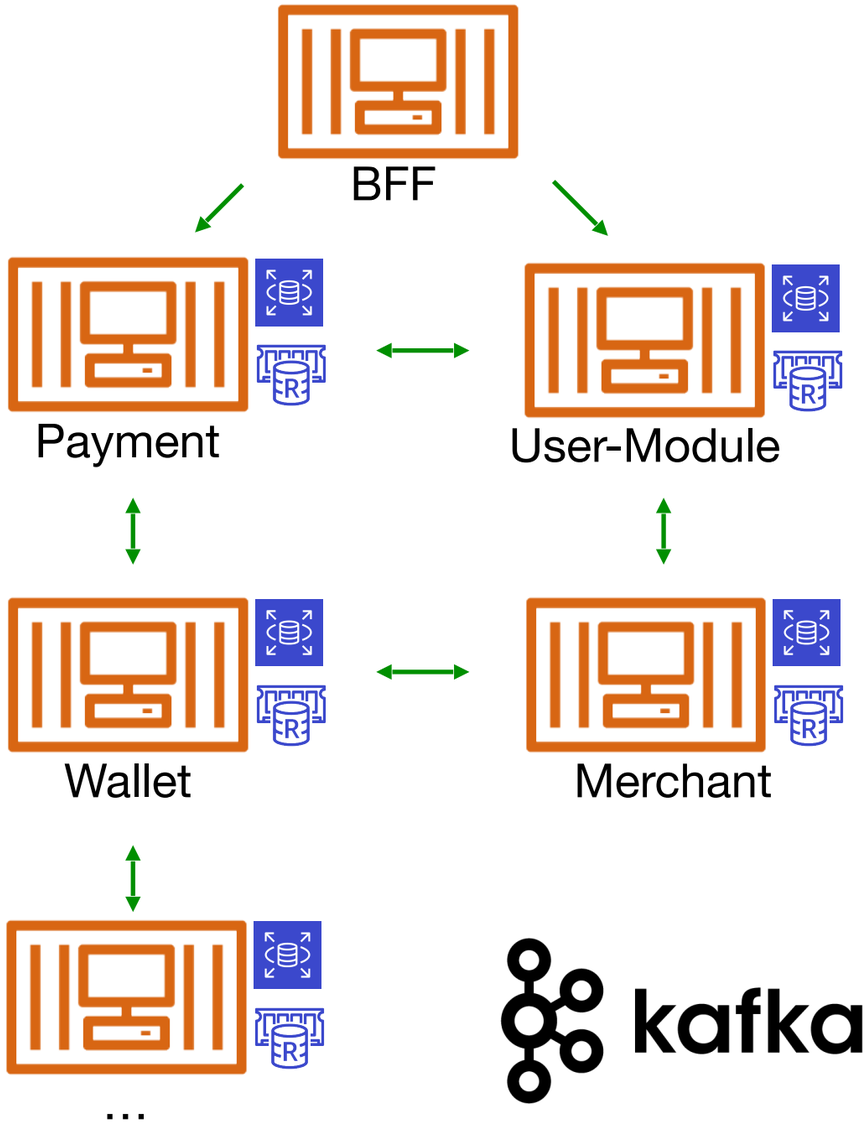
Microservice

kubernetes

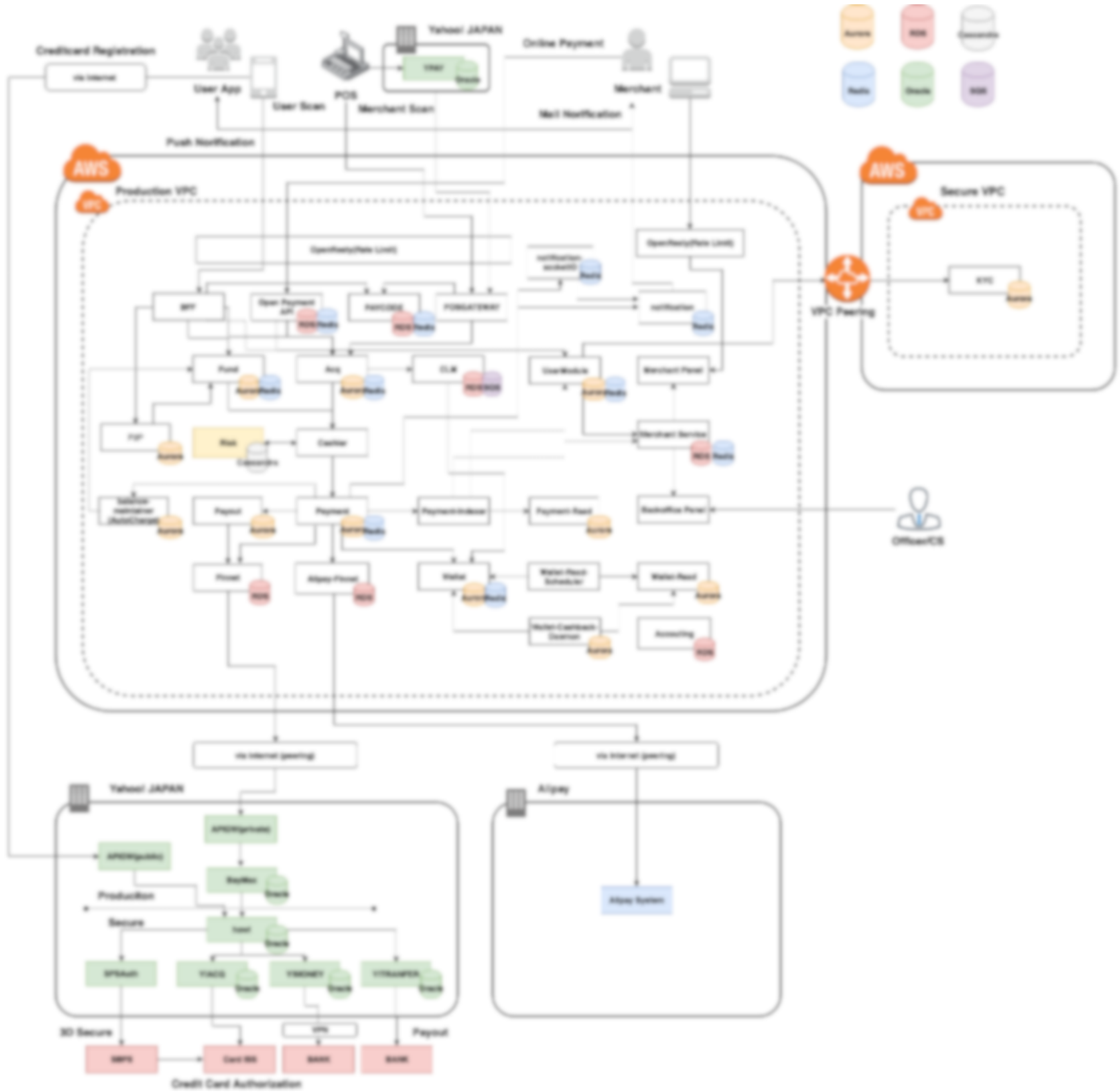
PayPay

PayPay Microservices

- 60+のサービス
- 7+チームで開発
- サービス毎に独立したリソース
- サービス間はAPIやKafkaを通した
  非同期メッセージでやりとり



PayPay



Creditcard Registration
User App
User Scan
POS
Merchant Scan
Yahoo! JAPAN
Online Payment
Merchant
Mail Notification
Push Notification
AWS
Production VPC
AWS
Secure VPC
VPC Peering
Officer/CS
Yahoo! JAPAN
Alipay
Production
Secure
3D Secure
Payout
Credit Card Authorization

PayPay Kubernetes

- EC2上に独自にKubernetesクラスタを構築
  - kopsを使ってAWS上に構築
- 構成
  - 40 nodes
  - 1000+ pods
  - 60+ services
  - 4クラスタ: dev, stg, perf, prod
- EKSでないのはリリース当初にTokyo regionで利用できなかったため

PayPay

PayPay AWS

- AWS
  - EC2 *
  - RDS (Aurora, MySQL)
  - ElastiCache (Redis)
  - Elasticsearch
  - CloudFront
  - S3
  - SES
  - WAF
  - EMR
  - GuardDuty
  - …

PayPay

AWSのリソースはコード管理

- 各リソースはTerraformでコード管理
  - レビュー、ロールバックなど管理コストの削減


```
resource "aws_rds_cluster_instance" "rds_aurora_user_db1_ro_uma" {
  count                       = 1
  identifier                  = "user-db1-ro-uma-${count.index}-prod"
  cluster_identifier          = "${aws_rds_cluster.rds_aurora_user_db1.id}"
  instance_class              = "db.r4.8xlarge"
  db_subnet_group_name        = "${aws_db_subnet_group.prod.id}"
  db_parameter_group_name     = "${aws_db_parameter_group.rds_aurora_user_db1.name}"
  monitoring_role_arn         = "${data.aws_iam_role.rds_monitoring_role.arn}"
  monitoring_interval         = 60
  engine                      = "aurora-mysql"
  auto_minor_version_upgrade = "false"
  apply_immediately           = "true"
  promotion_tier              = 5
  depends_on                  = ["aws_rds_cluster_instance.rds_aurora_user_db1_rw"]

  tags {
    dbservice   = "user-db1"
    Environment = "prod"
  }
}
```

PayPay

その他技術スタック

- アプリケーション
  - Java / Kotlin / Scala
  - Spring Boot
- CI/CD
  - Jenkins
- Repository
  - Artifactory
  - GitHub
- 監視
  - DataDog
  - PagerDuty

- ミドルウェア
  - Kafka
  - Cassandra
- Other
  - Slack
  - JIRA
  - Confluence
  - Kibana

PayPay

慣れない技術スタックであったが、無事ローンチ

- 10月5日にサービスイン

PayPay

First Mission

- **3ヶ月**でQRコード決済サービスをローンチすること

PayPay

Second Mission

- 100億円キャンペーンを安定稼働させること

PayPay

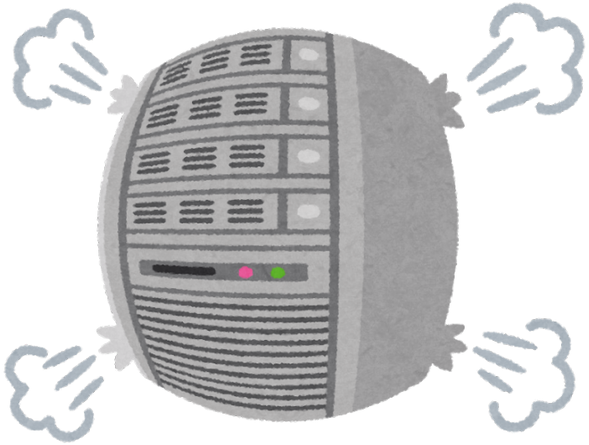
Second Mission

- 100億円キャンペーンを安定稼働させること

キャンペーン開始後サービスが不安定に...

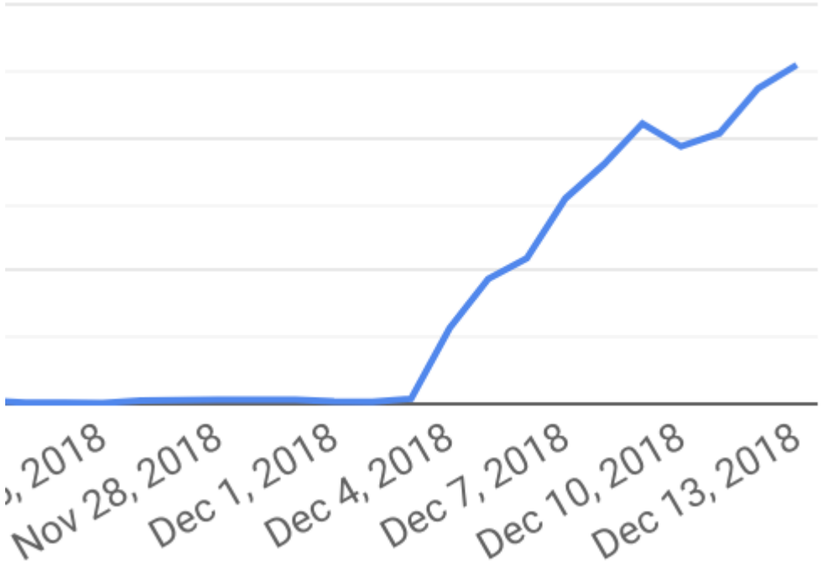
- アクセスが急増し、システムが不安定に
- 数度の緊急メンテナンスを実施

結果的には10日間でキャンペーン終了

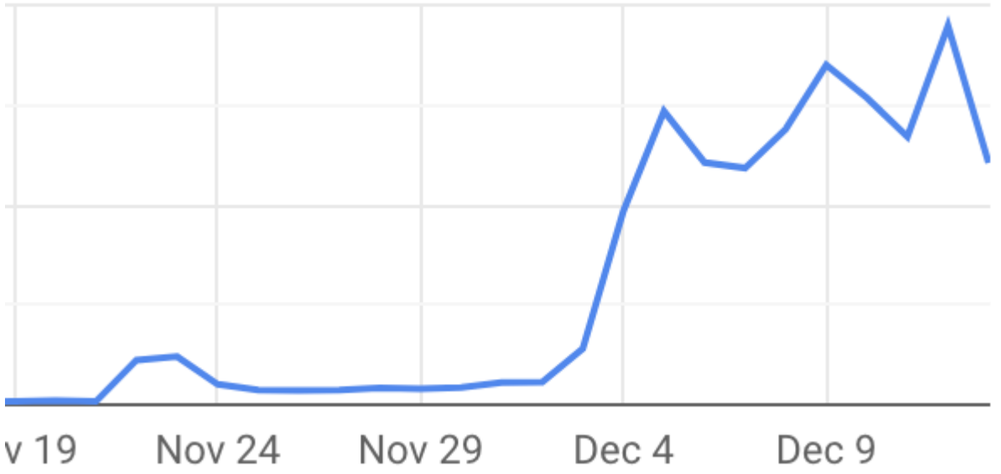


アクセスの状況

決済トランザクション



ユーザサインアップ



PayPay

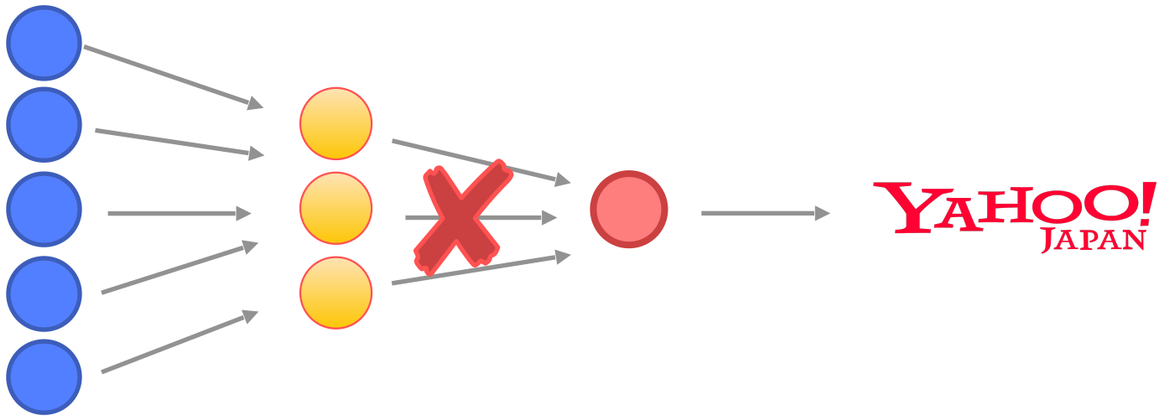
数々の問題が発生

- 外部連携サービスでトランザクションが滞留し、全断
- 加盟店向けツールの負荷の高いクエリが想定以上にコールされ不安定に
- RedisのBigKeys
- SESのLimitに到達し、メール送信が停止
- Singupが多すぎて500円付与が遅延

PayPay

外部連携サービスでトランザクションが滞留し、付随サービスが全断

- 外部決済システムと連携する一部サービスのリソース(Pod数)が不十分だった
- リクエストが滞留して各サービスのスレッド数が上限に到達



PayPay

各リソースの大幅な増強

- Pod・ノード数の拡張
- RDSのスケールアップ
  - db.m4.2xlarge → db.m4.16xlarge
  - IOPSの増強
- コストは5倍以上に…

PayPay

負荷の高いクエリの想定以上の呼び出し

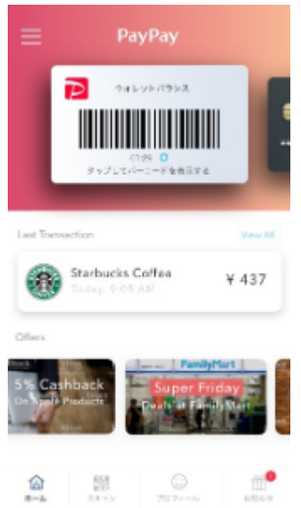
- PayPay for Business(加盟店向けツール)が想定以上に使用され、負荷の高いクエリが高頻度でコール
- 取引データを扱うPaymentサービスはコンシューマサイドでも使用しており、決済にも影響

PayPay

メンテナンス時に負荷対策を実施

負荷の分散

- 加盟店情報をRedisにキャッシュするように変更
- サービスを複製し、片方はSlaveのみにアクセス



機能の削減

- 加盟店ツールのオートリロードの停止
- ダウンロード機能の制限
- アプリホーム画面のデザインを変更
  - 全支払方法をスライドできるデザインだった
  - 外部決済システムも参照していたため、負荷が高い
  - PayPay残高のみのデザインに急遽変更

PayPay

RedisのBigKeys

- 認証サービスで使っていたRedisのCPU使用率が増加
- Spring SecurityがList型キーにトークンを保存
- 認証のたびにレコードが追加され、特定のシャードの負荷が高くなっていた
  - 要素数375万

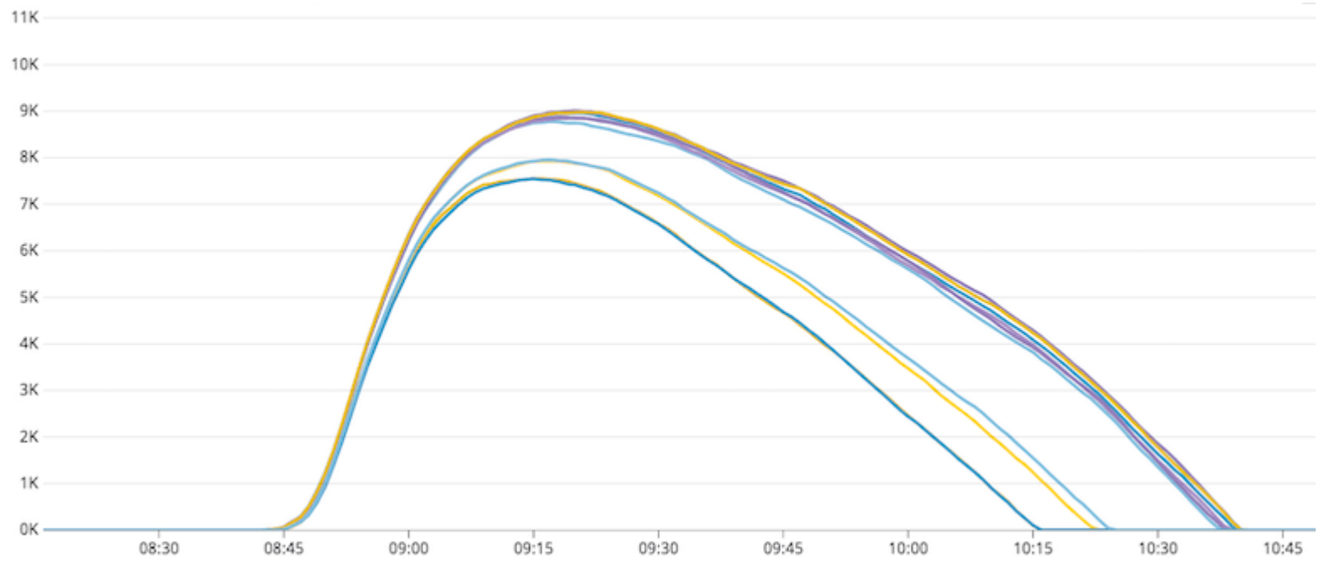
↓

- 一旦シャードの再配置である程度分散
- List型のキー設定を使用しないように変更

PayPay

Signupが多すぎて500円付与が遅延

- Signupでの500円付与は非同期処理
- TVで紹介されたことで一気にSignup数が増加
- SMS送信で利用していた外部サービスのlatency増加により大幅遅延
- これについてはアクセスが落ち着くのを見守った



PayPay

SMS送信の遅延

- SMS送信で利用していた外部サービスのlatency増加により大幅遅延
  - SMSが届く頃には有効期限が切れる・・・
- サインアップ用のSMS送信とその他のSMS送信を分割

PayPay

Third Mission

- 第2弾キャンペーンに備えて、システムをチューニング

PayPay

課題

- 一部機能の停止で重要機能が停止してしまう
- 負荷試験の考慮漏れ
  - 事前に負荷試験は実施していたが、問題となった部分のデータパターンが不足していた
- チューニングの甘さ
  - ウィークポイントの可視化ができていない

PayPay

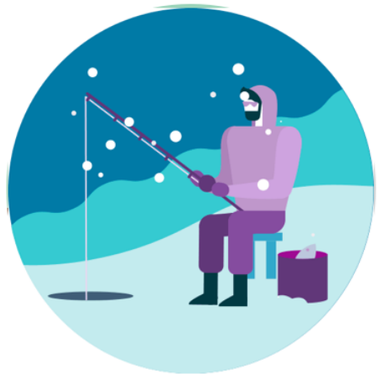
Graceful Handling

- Rate limiting
- Circuit Breaker
- 可視化
  - New Relic
  - VividCortex

PayPay

Rate Limiting

- システムの健全性を守るため独自のレートリミットのシステムを構築
- 詳細な負荷試験を実施し、システムの限界を超えないように制限
- 詳細は次の章で説明します



ご利用ありがとうございます

ただいま一部のユーザーがご利用できないようです。
ご不便おかけして申し訳ございませんが、
しばらく経ってから再度お試しください。

PayPay

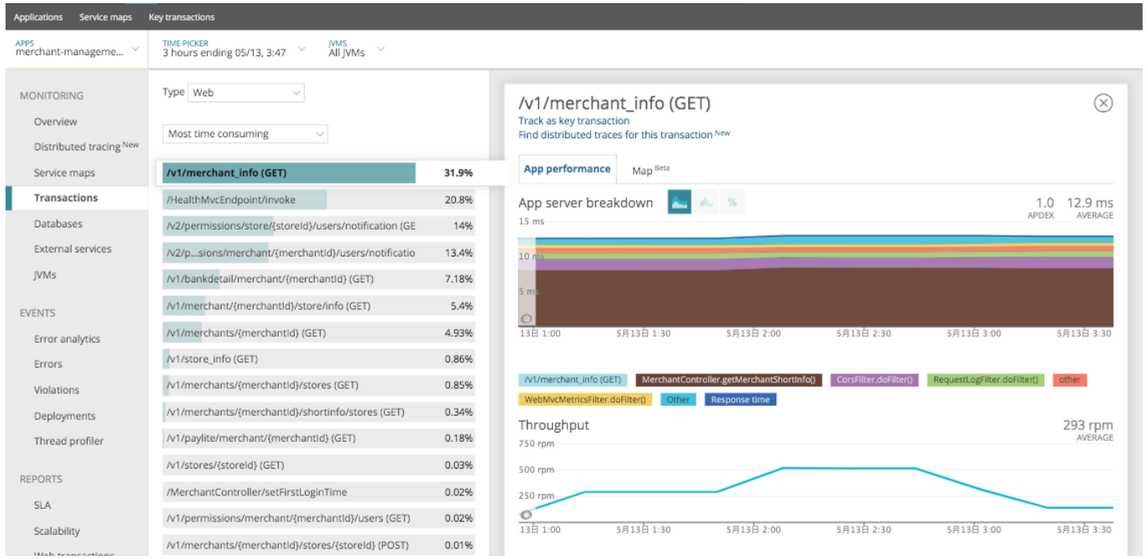
Circuit Breaker

- 下流のサービスがダウンした場合に一定時間該当の呼び出し処理をスキップする
  - 正常性が確認できれば再開
- 下流サービスがリクエストが滞留し、死に続けることがなくなる
- ライブラリはHystrixの後継のResilience4jを使用

PayPay

遅い処理の可視化

- APMとしてNewRelicを導入
- アプリケーションのどの部分で時間が掛かっているか一目瞭然
- インシデントの調査やチューニングの効率が格段に向上



セキュリティの強化

- 3D Secureの導入

Third Mission

- 第2弾キャンペーンに備えてシステムの問題点を解決する

PayPay Secret Sauce

PayPay

Shilei Long

- PayPay株式会社
  - Product Division

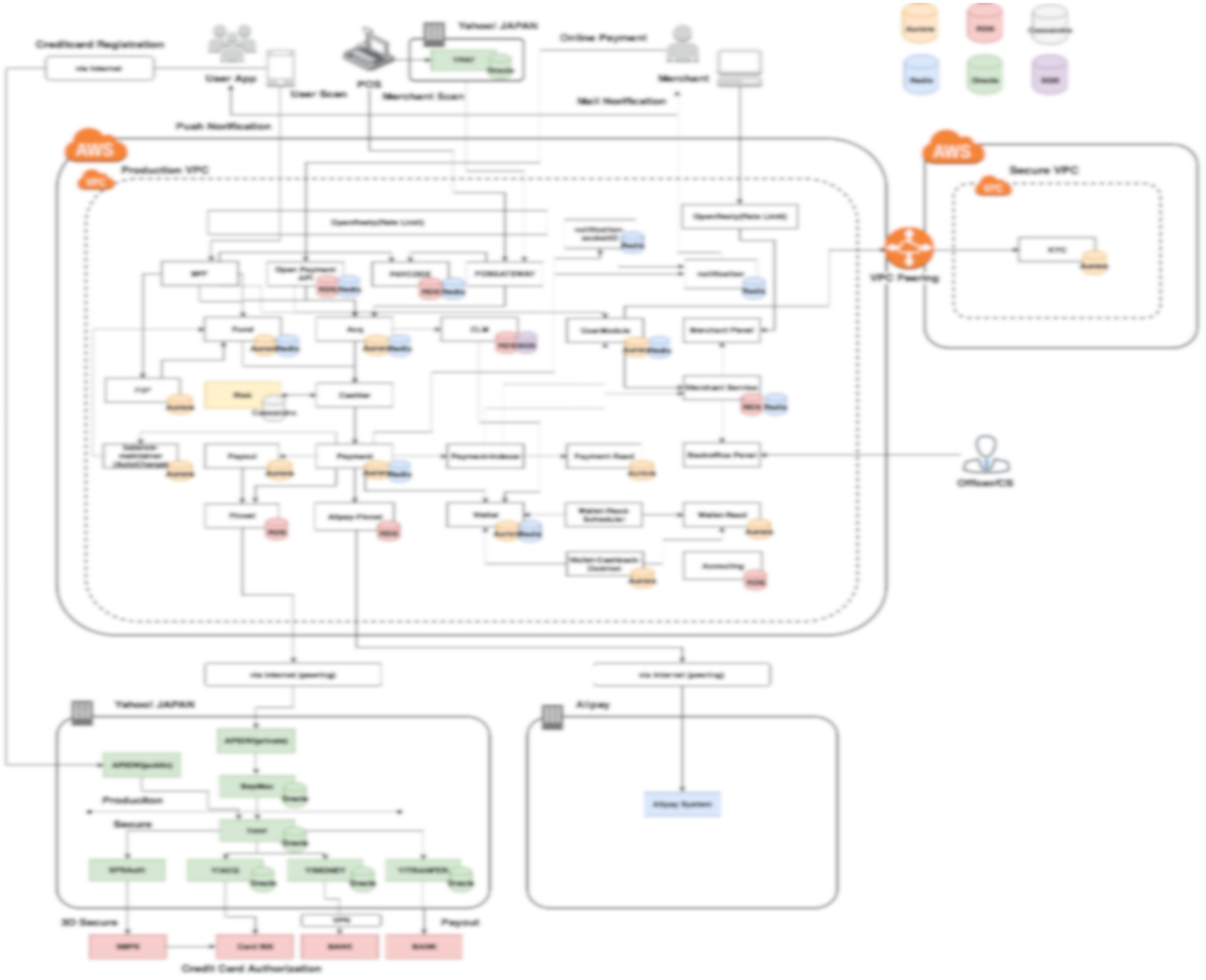
- Born and brought up in China, lives in Canada
- Secondment from Paytm Canada
- Previously SDE@Amazon Toronto

PayPay

Section 2 - Agenda

- A few things about microservices..
- Storage
- CQRS in Payment System
- Kafka
- The good and bad points of AWS for us

PayPay


Creditcard Registration
via Internet
User App
User Scan
POS
Merchant Scan
Yahoo! JAPAN
Online Payment
Merchant
Mail Notification
Push Notification
AWS
Production VPC
VPC
VPC Peering
Secure VPC
via Internet (peering)
Yahoo! JAPAN
Production
Secure
3D Secure
Payout
Credit Card Authorization
Alipay
Alipay System
Offline/ICS

A few things about microservices..

1. They will fail. I promise. But you can prevent cascading failures.

    - Example: User Module (authentication, user info, asset/payment method info)

    - Timeout! Timeout! Timeout!
    - Separation of concerns.
    - Circuit Breaker.

PayPay

A few things about microservices..

2. Test it

- How?
  - Staging Environment? Performance Environment? These may help, but they are not enough.. You don't know what you don't know.
  - **Read Traffic**
  - **Write Traffic**

PayPay

A few things about microservices..

3. Standardise it - only way to scale

- How?
  - Java + Spring Boot
  - Logging (log4j/fluentd -> Kafka -> Kafka Connect -> ES -> Kibana)
  - Monitoring (Datadog + NewRelic)
  - Distributed Tracing (Zipkin)
  - Kubernetes (service discovery, load balancing, routing ...)
  - CI/CD
- Put as much as you can in the base docker image
- Have a team of super star SREs/DevOps. Ask them to setup rules for every other team.

PayPay

How to add a service easily

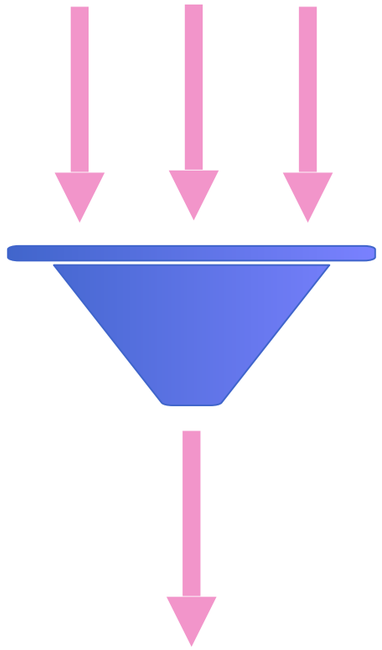
- 共通のDocker Base Image
  - NewRelicやDataDogなど設定済み
  - 各アプリケーションは環境変数を設定するだけ
- CI/CD
  - JenkinsのPipeline Scriptの追加
  - Build部分のみ個別作成で、あとはService名を登録するだけ
- KubernetesのManifestは Kustomize で省力化
- Monitoring設定
  - DataDogのTemplateを用意
  - コピーしてService名を入れるだけ

自動生成できるように対応中

PayPay

A few things about microservices..

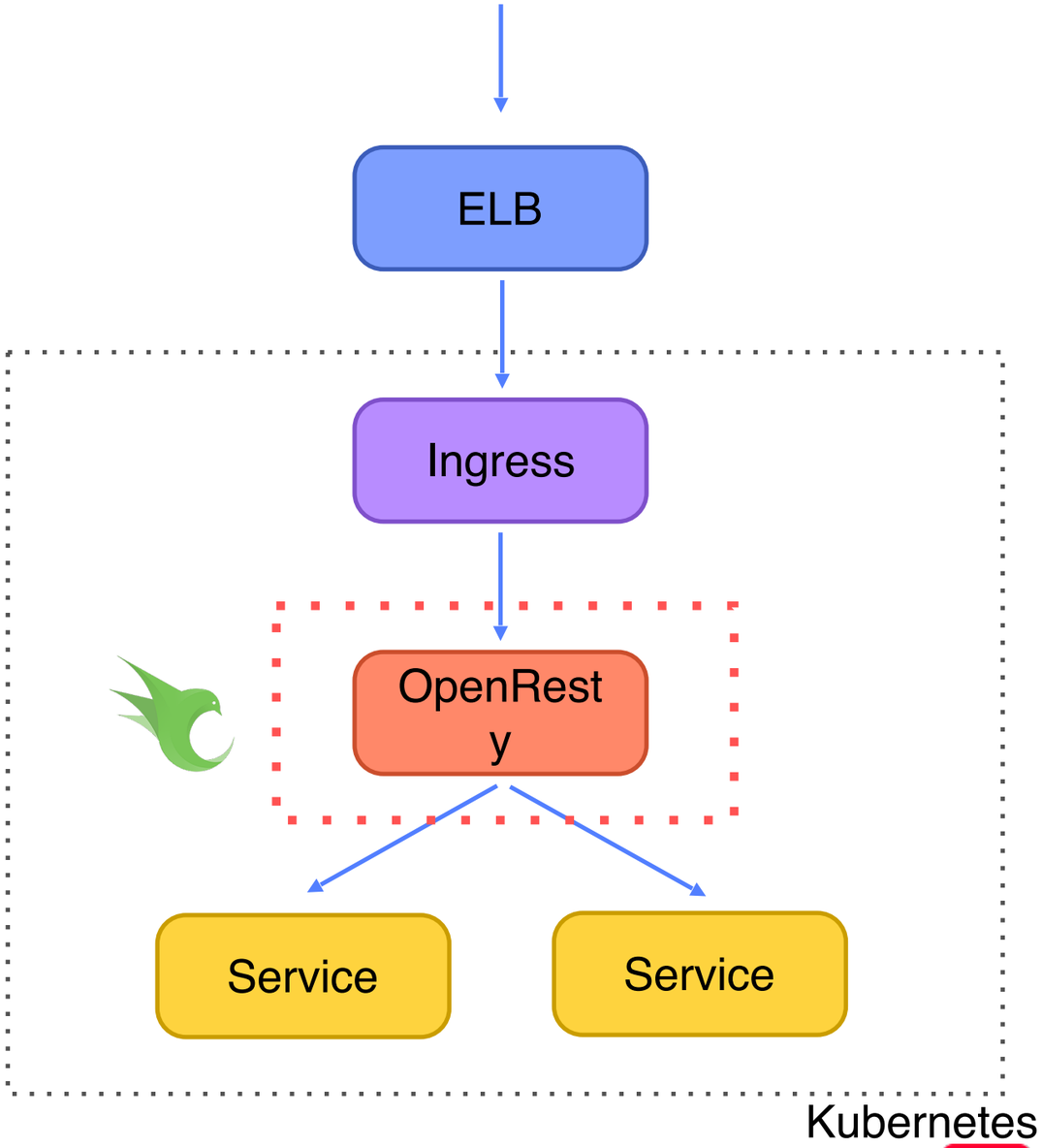
4. Rate Limiting



PayPay

Rate Limiting Continued..

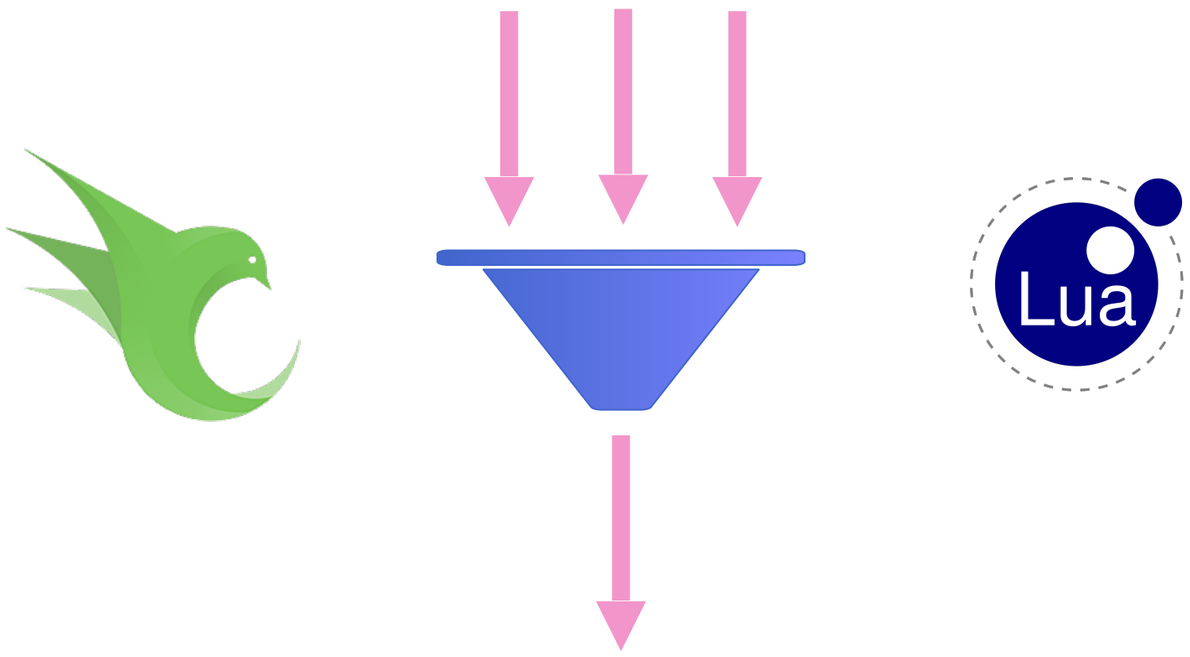
- Kubernetes Ingress not very customizable
- Add Nginx/OpenResty as a service behind Ingress



PayPay

Rate Limiting Continued..

- Using OpenResty with Lua script, you get very creative rate limiting mechanism.



PayPay

Rate Limiting Continued..

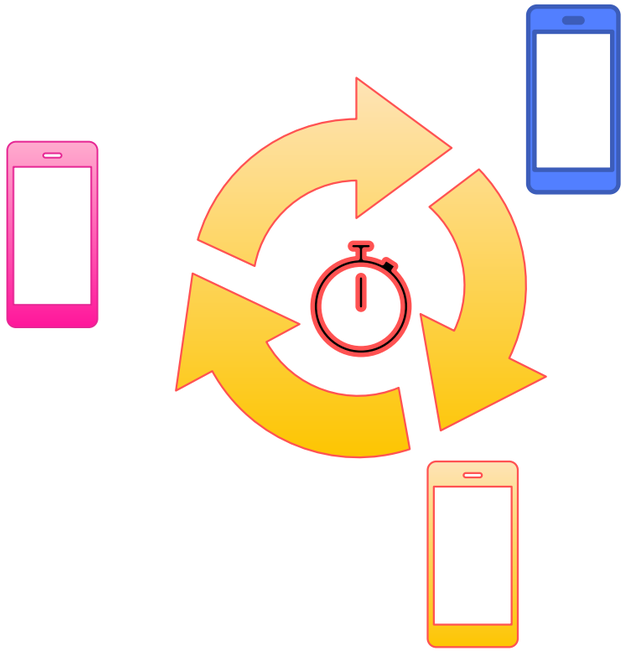
- Basic rate based, IP whitelisting..
- Allow only 10% of customers to use our service for 2 min without interruption; After 2 min, allow next 10% of customers to use. Most of our customers will complete a transaction within 2 min.
- Lua code:

```lua

--uid is unique id which can be any string
--allowed allowed percentage of request
function _M.IsDeviceValid(uid, allowed_pct, ttl)
  local current_ts_sec = ngx.time()

  local time_bucket = math.floor(current_ts_sec / math.max(ttl*60, 1))

  local uid = tostring(uid)
  local device_hash = hashlib.hashcode(uid)
  local remainder = math.abs(device_hash + time_bucket*allowed_pct) % 100

  return remainder < allowed_pct
end

```

PayPay

Device IDを基にしたGroup単位でのAccess制限

- 単純な流量制限だと決済の一連の流れの途中で制限が掛かってしまう
- Device IDをHash化し、グループ単位で制限を掛けるようにした
  - 制限が掛かるグループは一定期間ごとにローテーションする



ex)
20%のUserを制限
1分毎にローテーション

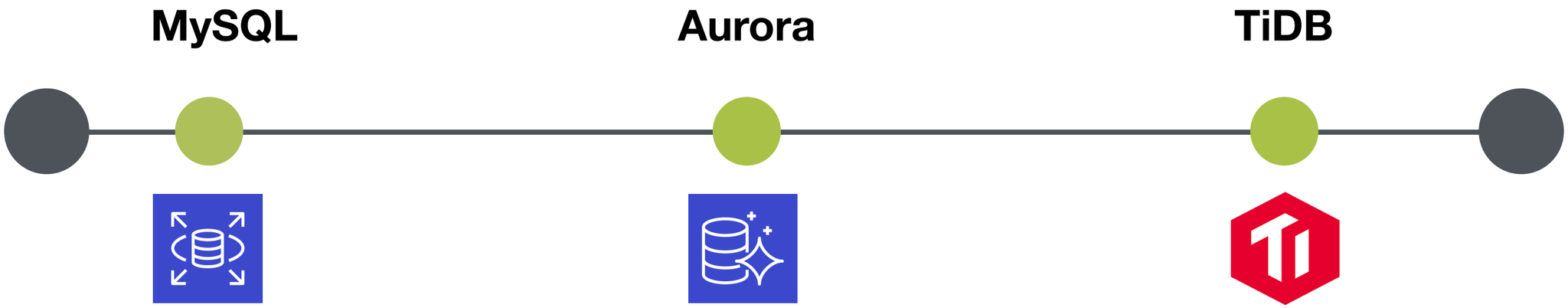
PayPay

Storage Design

PayPay

Database: RDS MySQL

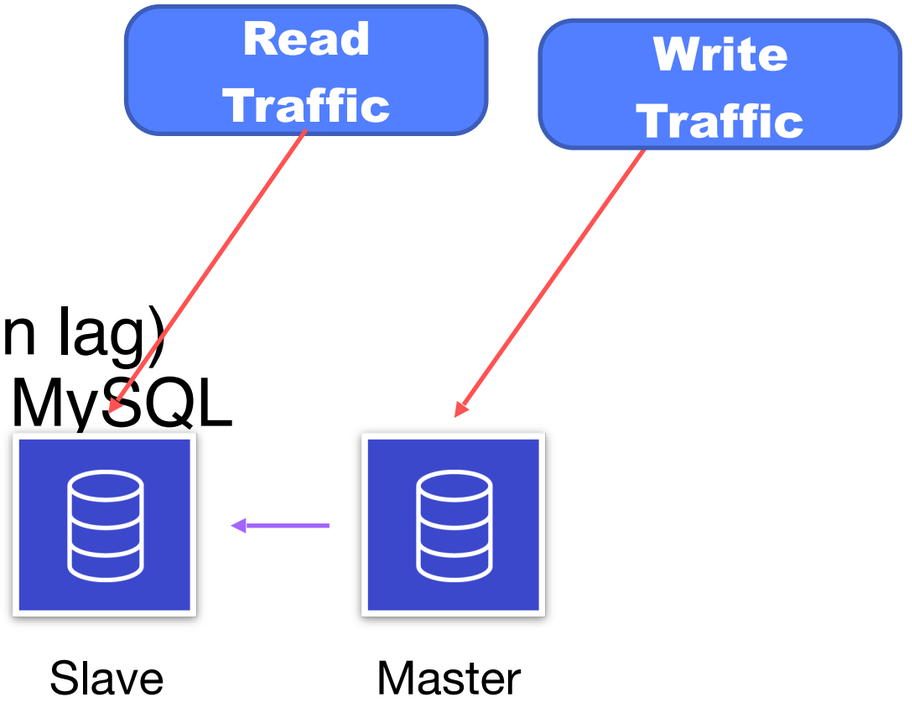
- MySQL should satisfy 90% of all the use cases without issue.
- AWS RDS with multi-AZ automatic failover is super reliable.
- You need to have good index design and some basic tuning
  - VividCortex or Performance Insights will help you identify issues.
  - Load test your system in production!

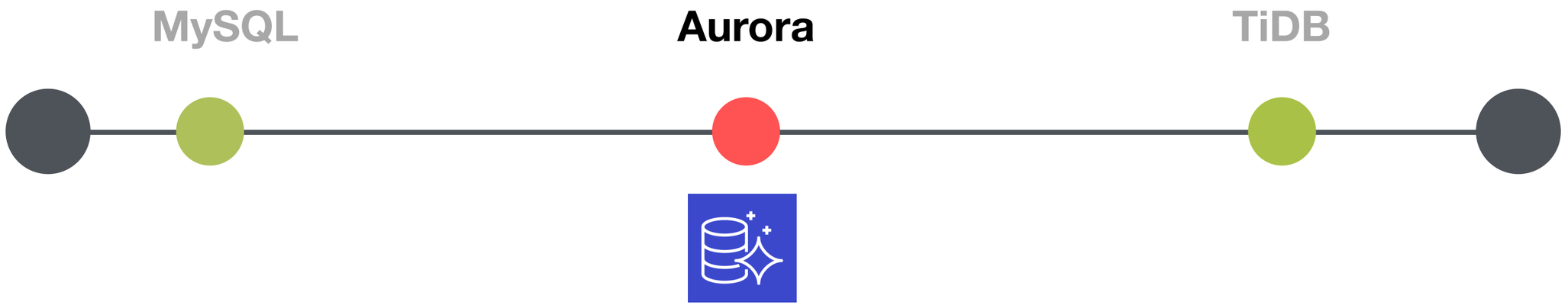


PayPay

Database: AWS Aurora MySQL

- Why?
  - Read-Write Split made possible (low replication lag)
  - Much better overall performance compared to MySQL
  - MySQL Compliant



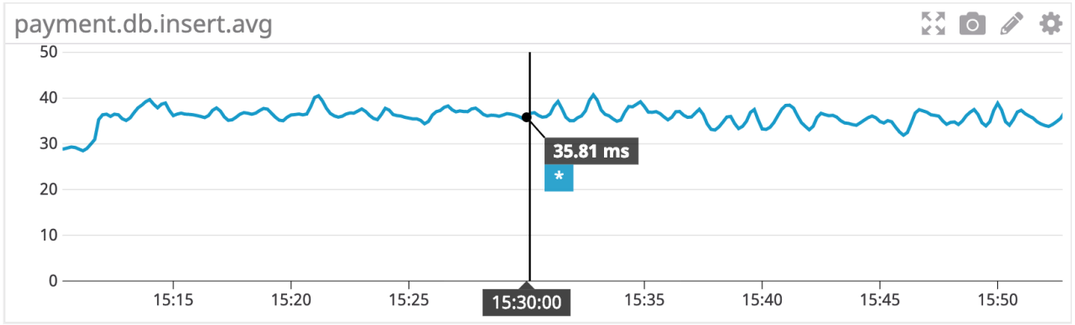


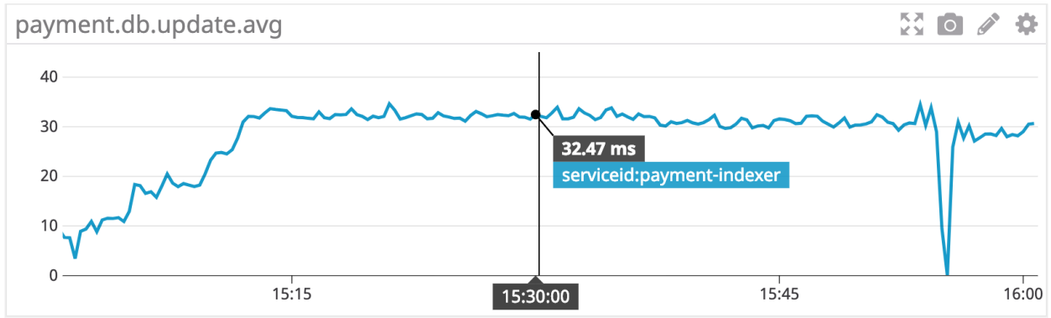
PayPay

Writeに関してもMySQLの2.5倍速く

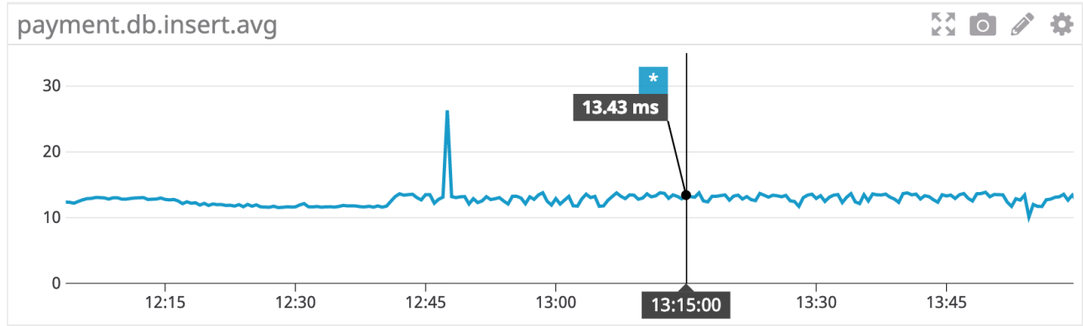
性能試験をしたところ、更新系も現状の使い方ではMySQLの2.5倍程度の性能となった

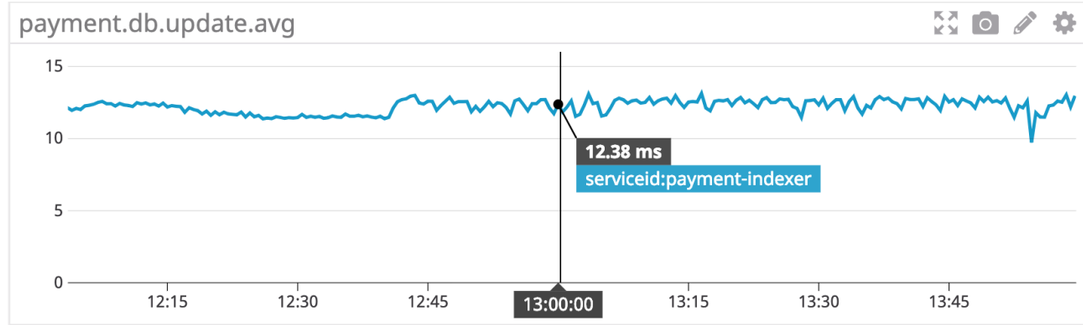
MySQL





Aurora



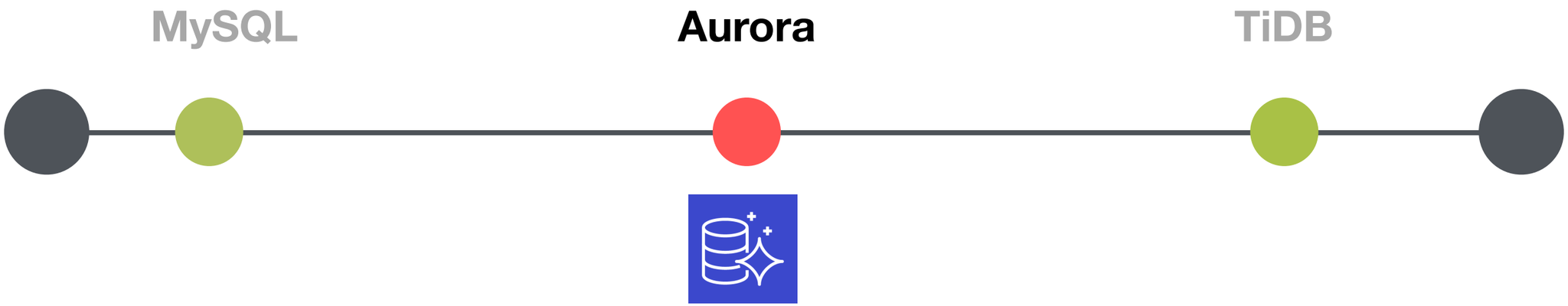


PayPay

現在は主要DBはAuroraに移行済み

- 想定以上の性能となったためAuroraで運用
- 負荷試験においても、まだ性能限界を確認できていない
- Slaveが最適化されCostについても削減できた

- My suggestion? Go with Aurora from the outset.



PayPay

Database: Future?

- > 500 million rows (currently, some of our database already have > 200 million rows)
- > 20,000 RPS, > 2000 TPS
- Horizontal Scalability is needed

PayPay

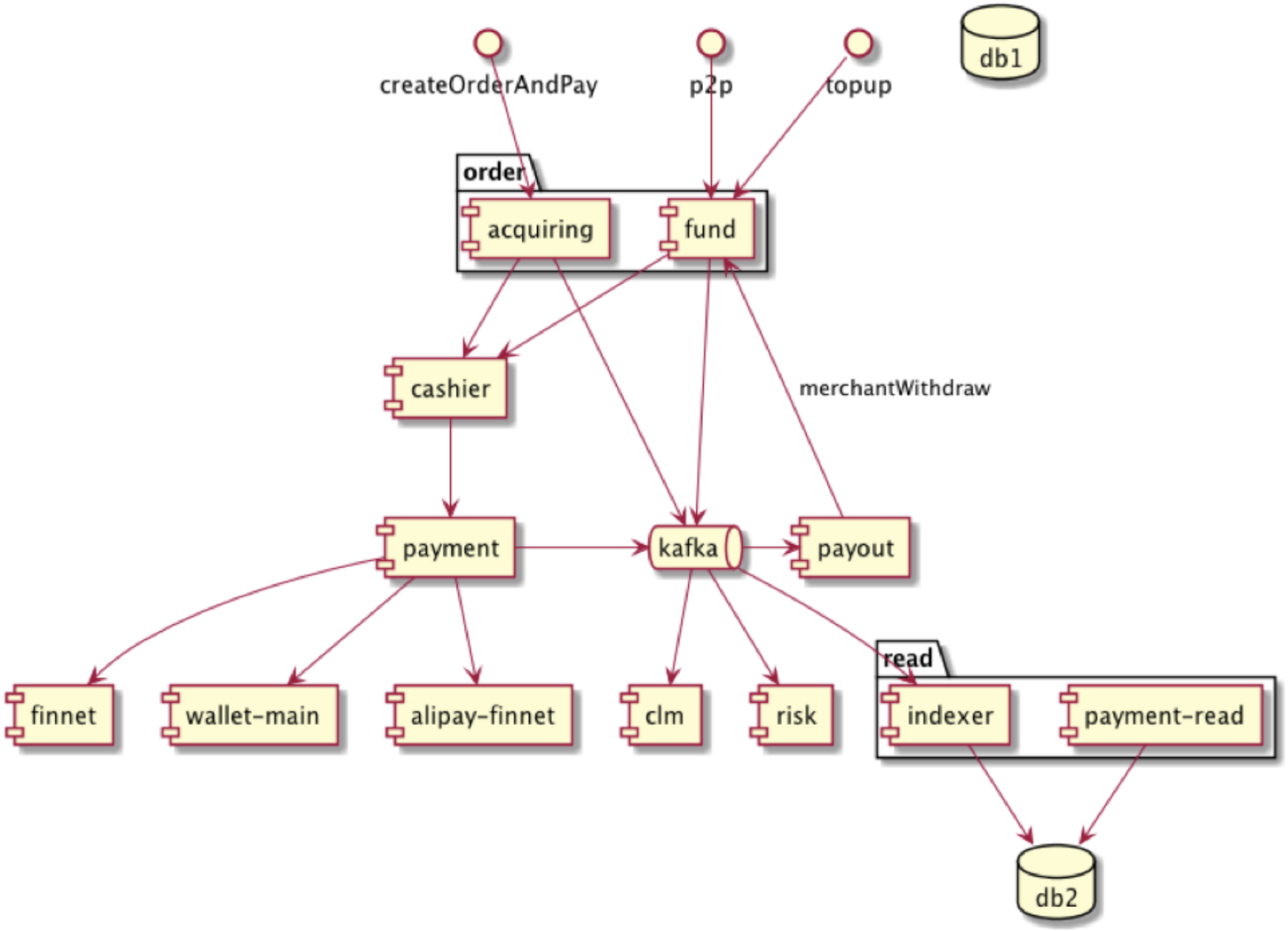
Database: Future?

- For certain use cases, we are considering Sharding vs NewSQL vs Aurora Multi-Master
- Aurora Multi-Master

- TiDB: Open Source Distributed Database
- Characters:
  - Horizontally scalable
  - MySQL Compliant
  - Distributed Transactions with Strong Consistency
  - Cloud Native (AWS EKS)
  - HA

PayPay

Payment System: CQRS (Command Query Responsibility Segregation)



PayPay

1. Write DB has almost 0 index. No query served. Append only.

2. Fully async communication from write system to read system. Write system publish events to Kafka, which are consumed by consumers writing into Read DB.

3. Read DB has lots of indices and are de-normalized. Best for query performance.

3. Complete read write segregation.

PayPay

Problem?

Missing events..

Reliable publishing + Reconciliation Job

PayPay

Kafka

PayPay

PayPay Kafka

- 用途
  - アプリケーション間の非同期メッセージ
  - アプリケーションログの転送
- 構成
  - EC2上に構築
  - 7 nodes * 2 cluster

ElasticSearch

PayPay

現行Kafkaの問題点

- 1つのClusterで管理している
  - 影響範囲が大きい
  - Service毎の最適化がやりにくい
- 保守性の低さ
  - EC2上で管理しており、ほとんどManual Operation
  - Scalingが柔軟にできない

PayPay

Kafka on Kubernetes

- Moving towards Kafka on Kubernetes
- StatefulSet, PersistenVolume, etc..
- Evaluating Strimzi

×

PayPay

Strimzi

- Kubernetes Operatorとして動作し、ClusterとTopicを宣言的に管理

PayPay

ClusterやTopicを宣言的に管理

Cluster

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: Kafka
metadata:
  name: kafka-log
spec:
  kafka:
    version: 2.1.0
    replicas: 5
    config:
      offsets.topic.replication.factor: 3
      transaction.state.log.replication.factor: 3
      # …
    resources:
      requests:
        memory: "24Gi"
        cpu: "3"
      limits:
        #…
    storage:
      type: persistent-claim
      size: 2000Gi
```

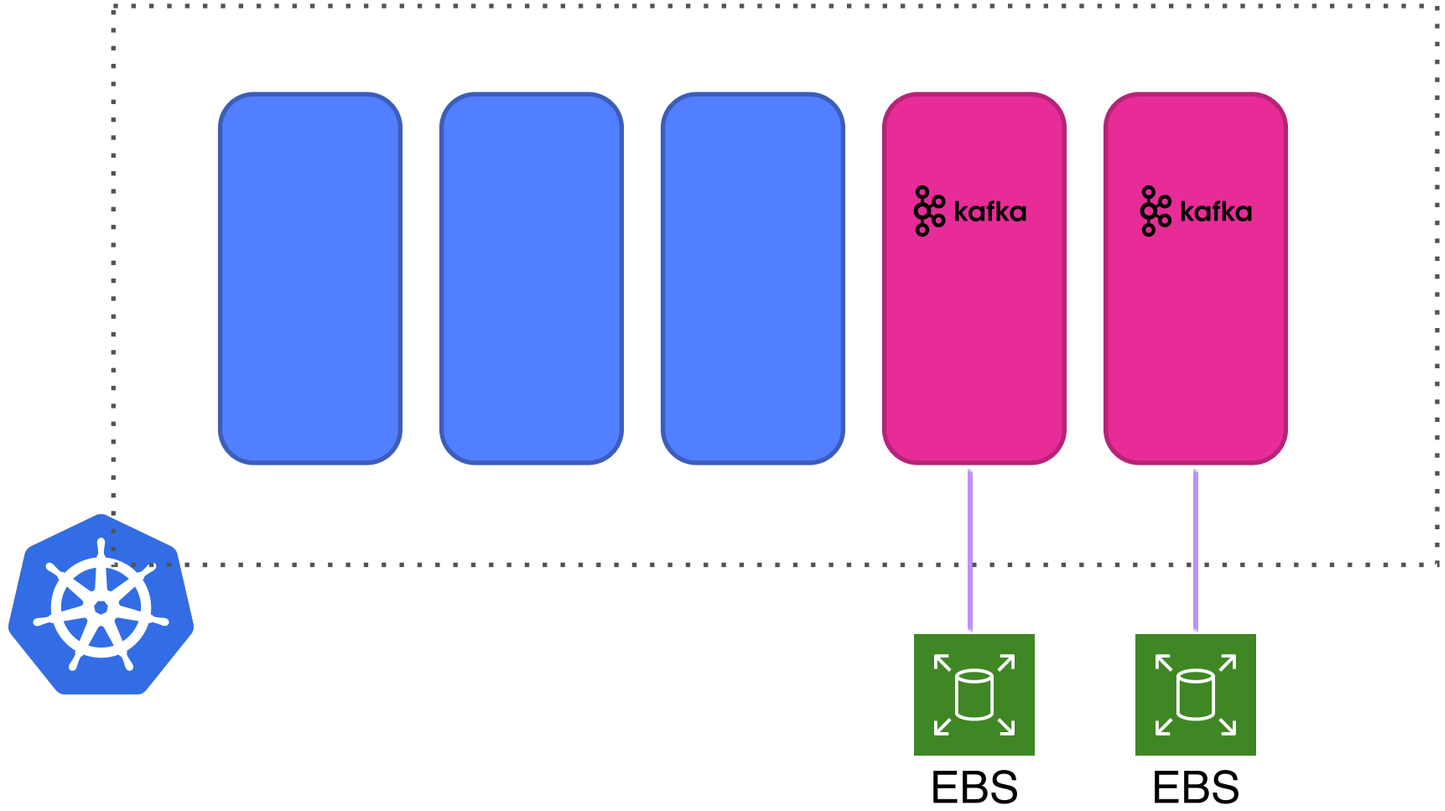
Topic

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaTopic
metadata:
  name: payout
  labels:
    strimzi.io/cluster: payment
spec:
  partitions: 10
  replicas: 3
```

PayPay

システム構成

- Dedicated node上に配置
  - Noisy neighborの回避



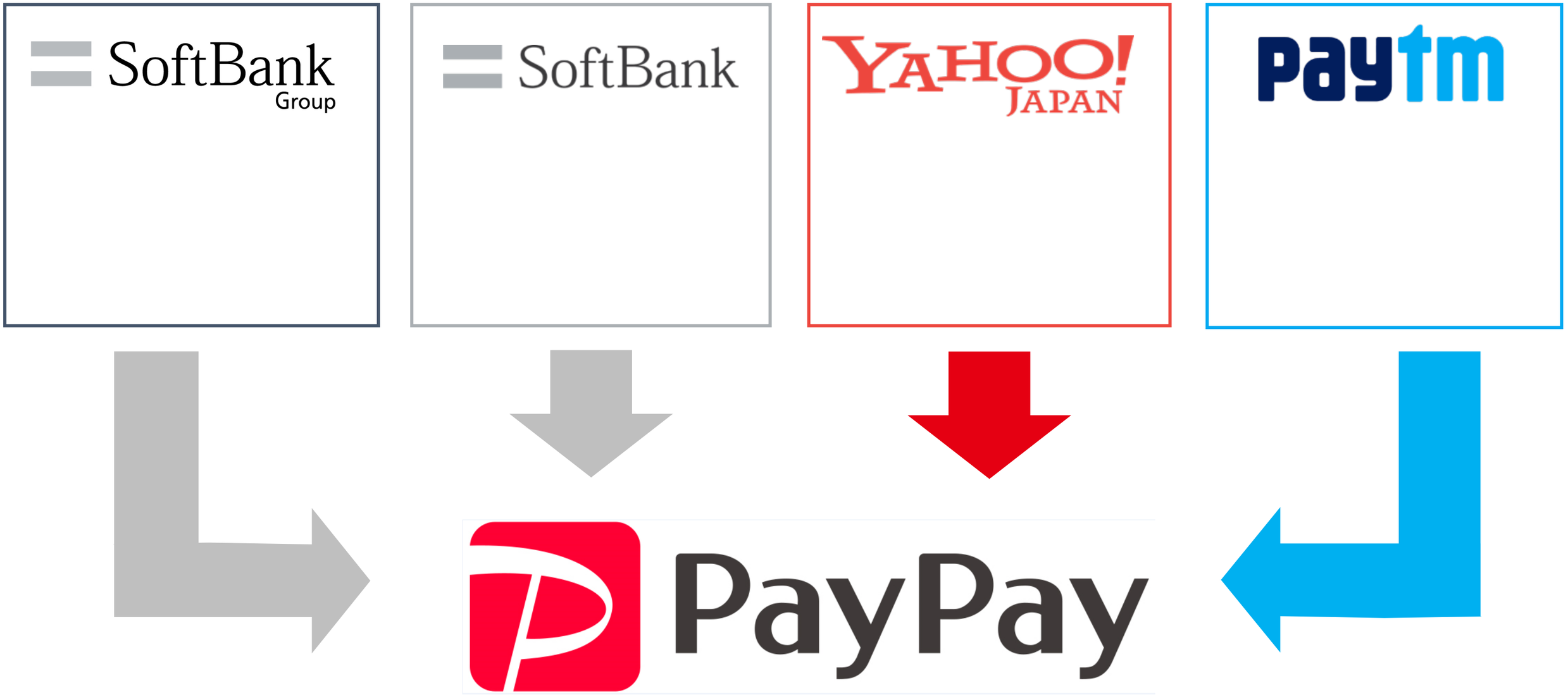
The good and bad points of AWS for us

- Good
  - Easy and fast to start
  - Super reliable: RDS, EC2, ElasticCache, VPC...
  - Powerful yet maintainable
  - Scalability

- Not Good
  - Osaka Region
  - Support of newest versions
  - Costs (it will become expensive:-))

PayPay

SoftBank Group Synergy



一緒に働きましょう！

ご連絡はこちら

- momoko.hayakawa@paypay-corp.co.jp
- https://paypay.ne.jp/career

PayPay