900万ダウンロードアプリ『Gunosy』を支える大規模モバイルプッシュ通知基盤

2015.6.3 AWS Summit Tokyo 2015 DevCon

自己紹介

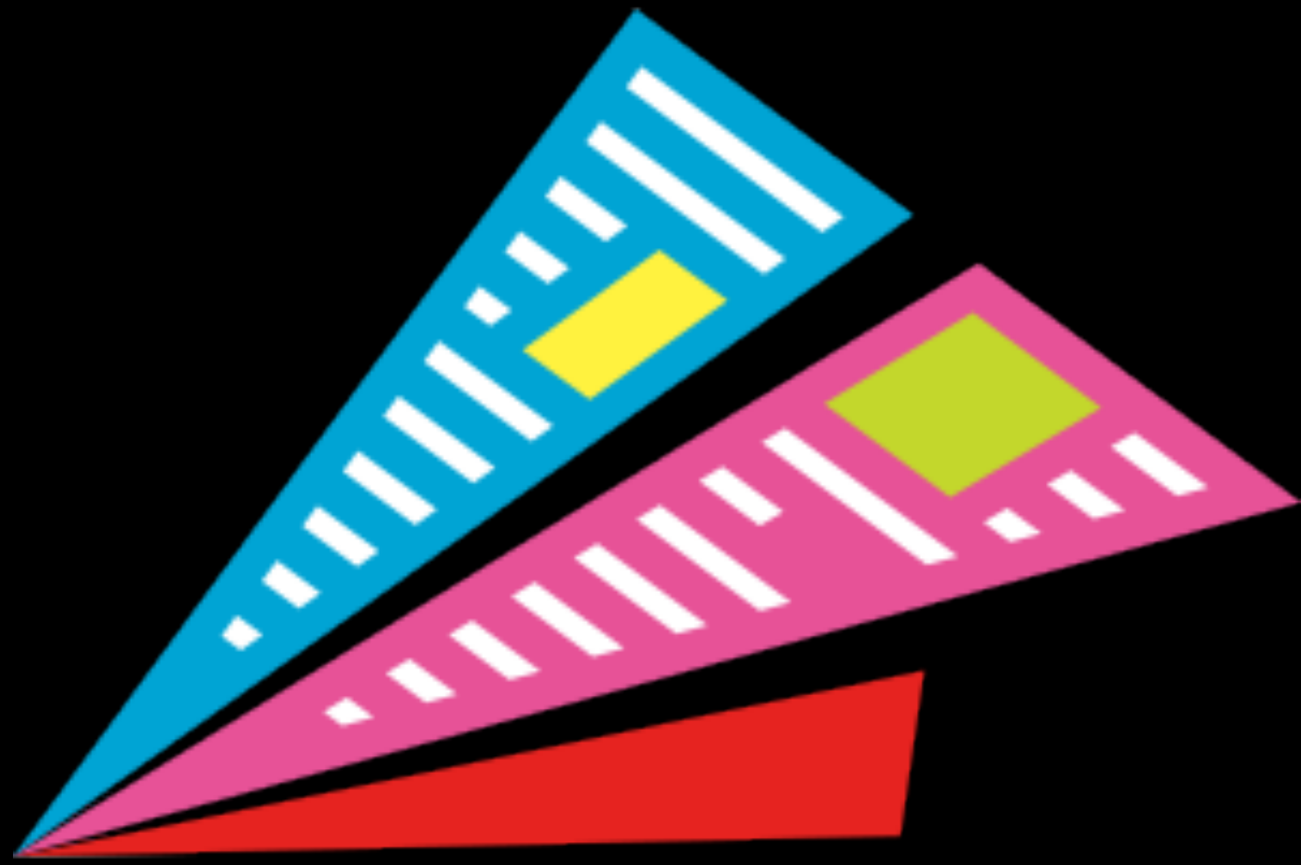
- 榎本 敏丸 (@toshimaru_e)
- WEBエンジニア@Gunosy
- GunosyではRails書いてます
- 好きなAWS Service: OpsWorks

アジェンダ

- 第一部 Gunosyのプッシュ基盤
- 第二部 GunosyでのAmazon SNS利用事例

Gunosy

- ニュースアプリ（iPhone/Andoid）
- 900万ダウンロード
- **「情報を世界中の人に最適に届ける」**
- 【宣伝】先日 gunosy.com を大型リニューアル



Gunosyとプッシュ通知 🚀

- モバイルプッシュ通知はDAUを支える重要要素
- プッシュ通知はGunosyアプリを開くきっかけを与える → ユーザー・リテンション
- プッシュ通知結果がDAUを左右する

２つのプッシュ通知

1. 定時プッシュ
    - 朝・昼・夕・夜の決まった時間のプッシュ通知
2. 速報プッシュ
    - 速報性のある即時送りたいプッシュ通知
    - 例: 大きな地震、大きな事件

定時プッシュ 🕒

- 【**いつ❓**】 ユーザーが設定した時間に（朝・昼・夕・夜の４スロット）、
- 【**誰に❓**】（プッシュ通知設定済みの）ユーザーに、
- 【**どれくらいで❓**】 （APIサーバーに負荷をかけない程度で）できるだけ短時間で、
- 【**何を❓**】 適切なタイトル・内容でプッシュ通知を送る

速報プッシュ 💨

- 【**いつ❓**】 大きな事件・事故が起こった瞬間に、
- 【**誰に❓**】（プッシュ通知設定済みの）ユーザーに、
- 【**どれくらいで❓**】 できるだけ短時間で、
- 【**何を❓**】 適切なタイトル・内容でプッシュ通知を送る

Gunosyプッシュ通知の歴史と変遷

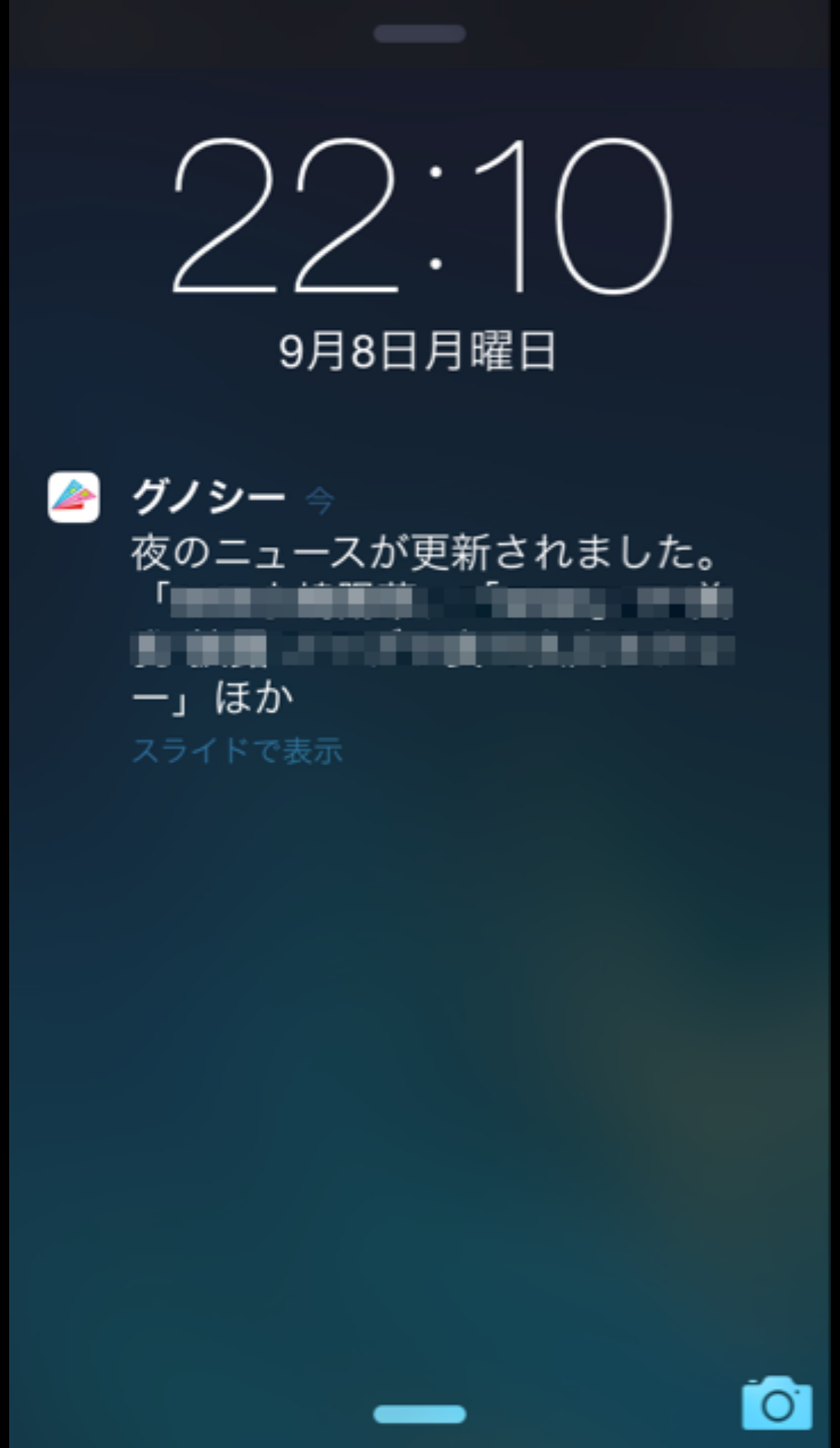
- 第一期 ~ローカルプッシュ時代~
- 第二期 ~リモートプッシュ移行期~
- 第三期 ~本格リモートプッシュ時代~
- 第四期 ~パーソナライズドプッシュ時代~

プッシュ第一期
~ローカルプッシュ時代~

- ローカルプッシュ時代
- 設定された時間に「ニュースが届きました」という定型文での通知
- クライアント側（iOS/Android）で実装

結果

- ローカルプッシュ通知をきっかけにアプリを立ち上げるユーザーが増加
- まずまずの開封率



プッシュ第二期
~リモートプッシュ移行期~

- Amazon SNS（モバイルプッシュメッセージング機能）の導入
- 『ニュースが更新されました「ニュースタイトル」ほか』という通知
- ローカルプッシュ ➡ リモートプッシュへの移行

結果

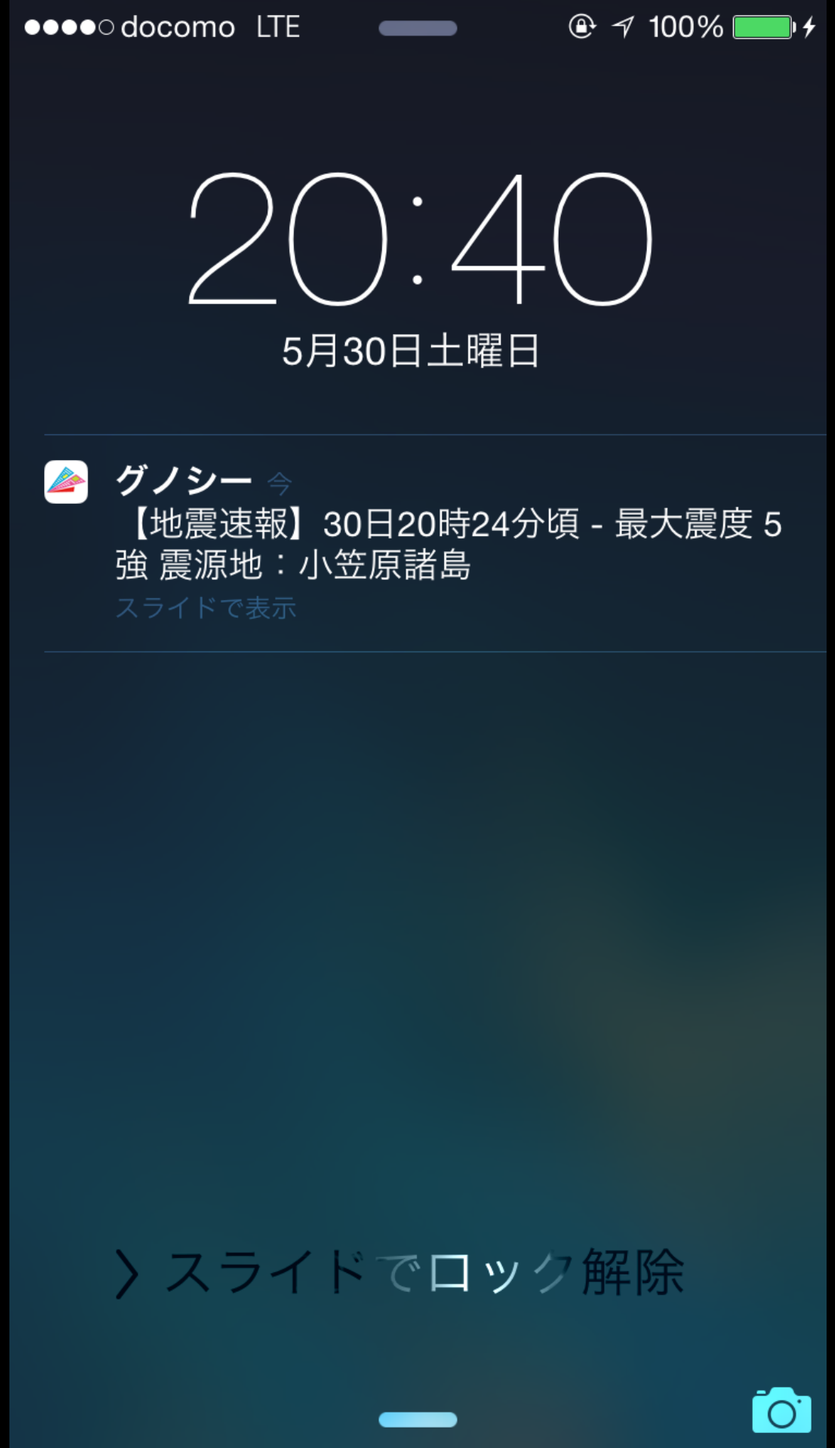
- プッシュ開封率アップ
- プッシュ通知 ➡ DAU向上

問題 ⚠

雑なプッシュ通知運用

問題

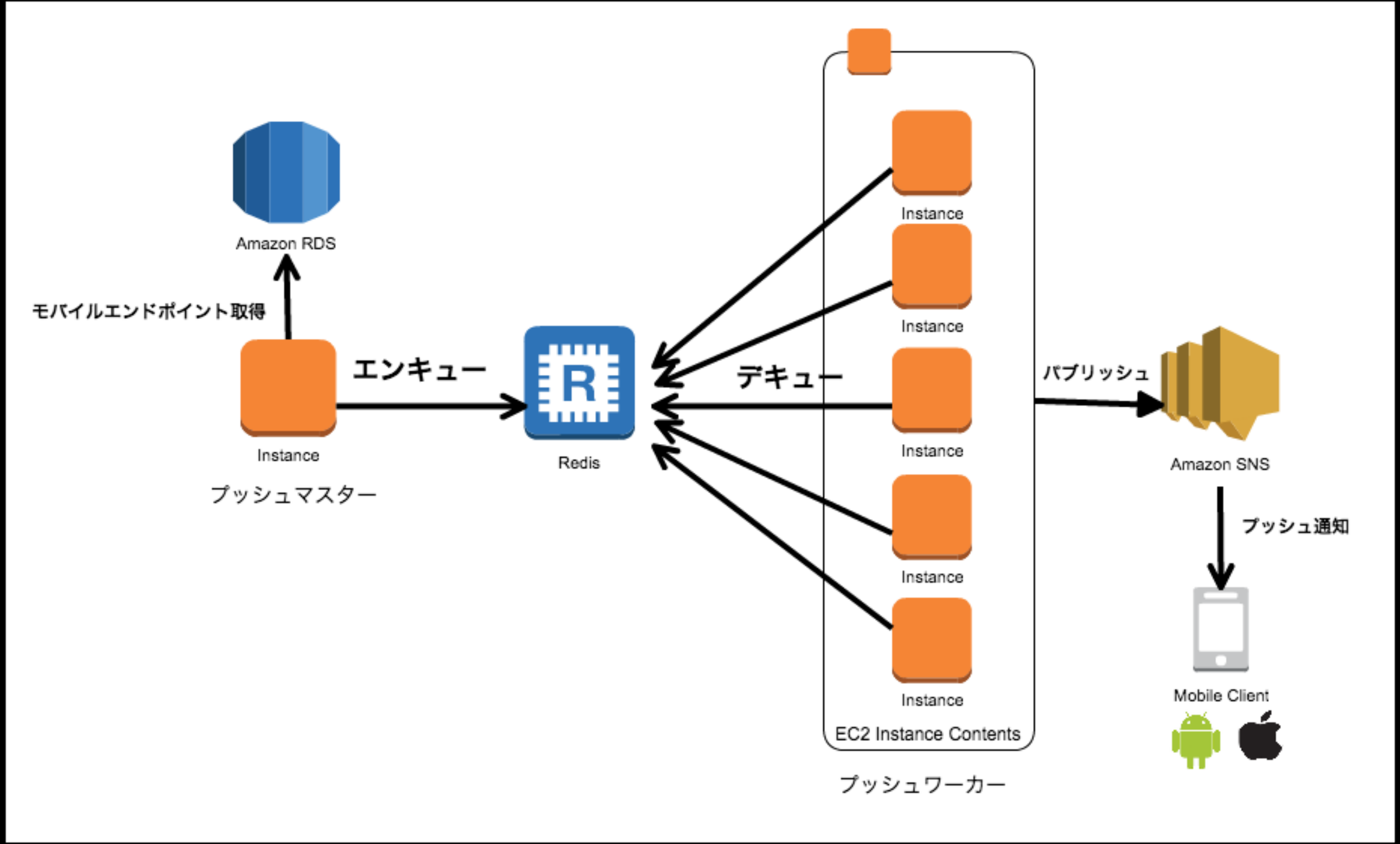
- 開発者がサーバーにSSHログインしてプッシュのためのコマンドを打ってプッシュ通知... ⤵
- プッシュ通知管理画面が不安定... 😵



プッシュ第三期
~本格リモートプッシュ時代~

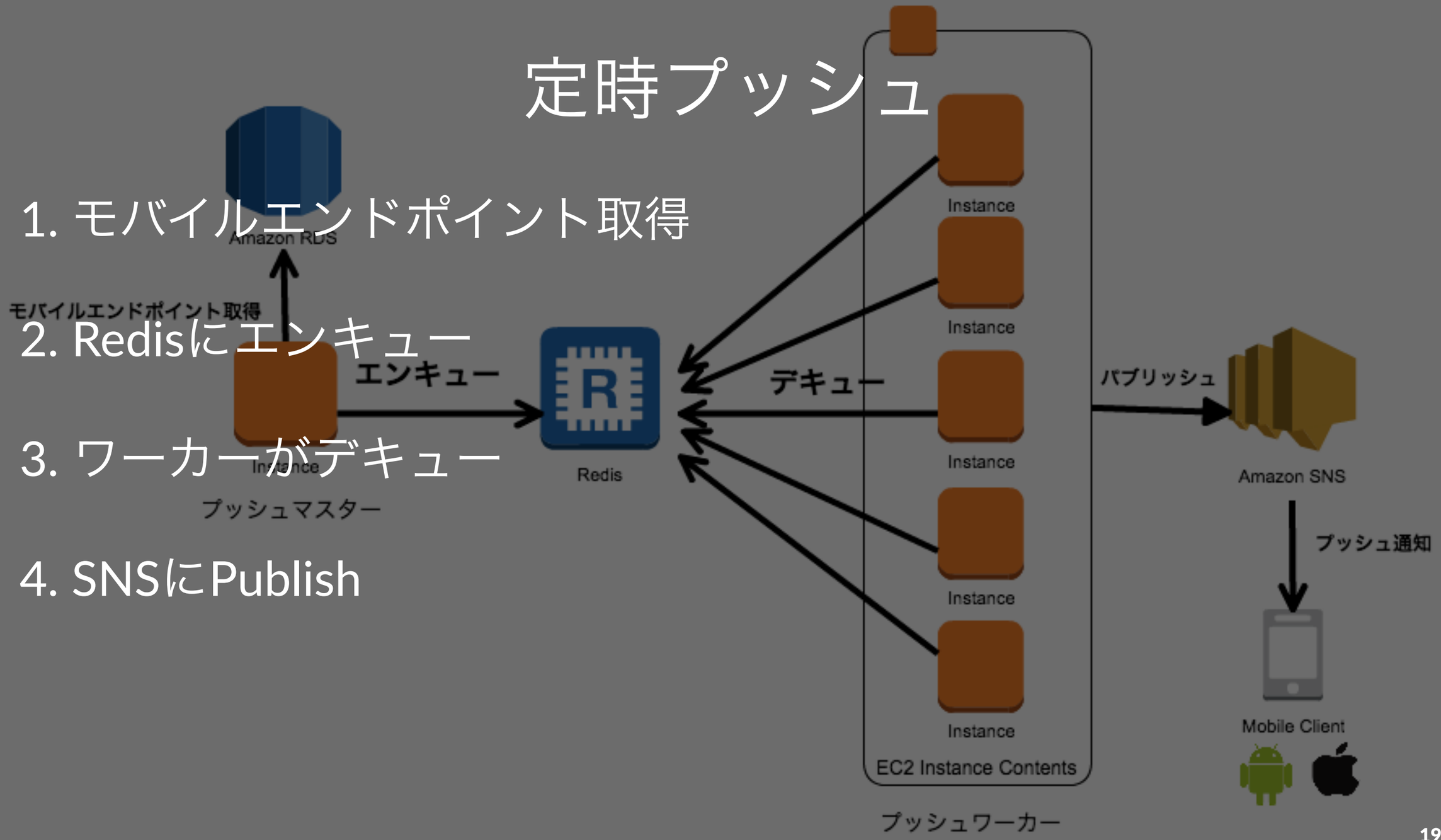
- Amazon SNSを用いたリモートプッシュ通知の本格運用時代
- 大規模プッシュに耐えうるアーキテクチャをきちんとゼロから考えて再設計

プッシュアーキテクチャ
（定時プッシュの場合）

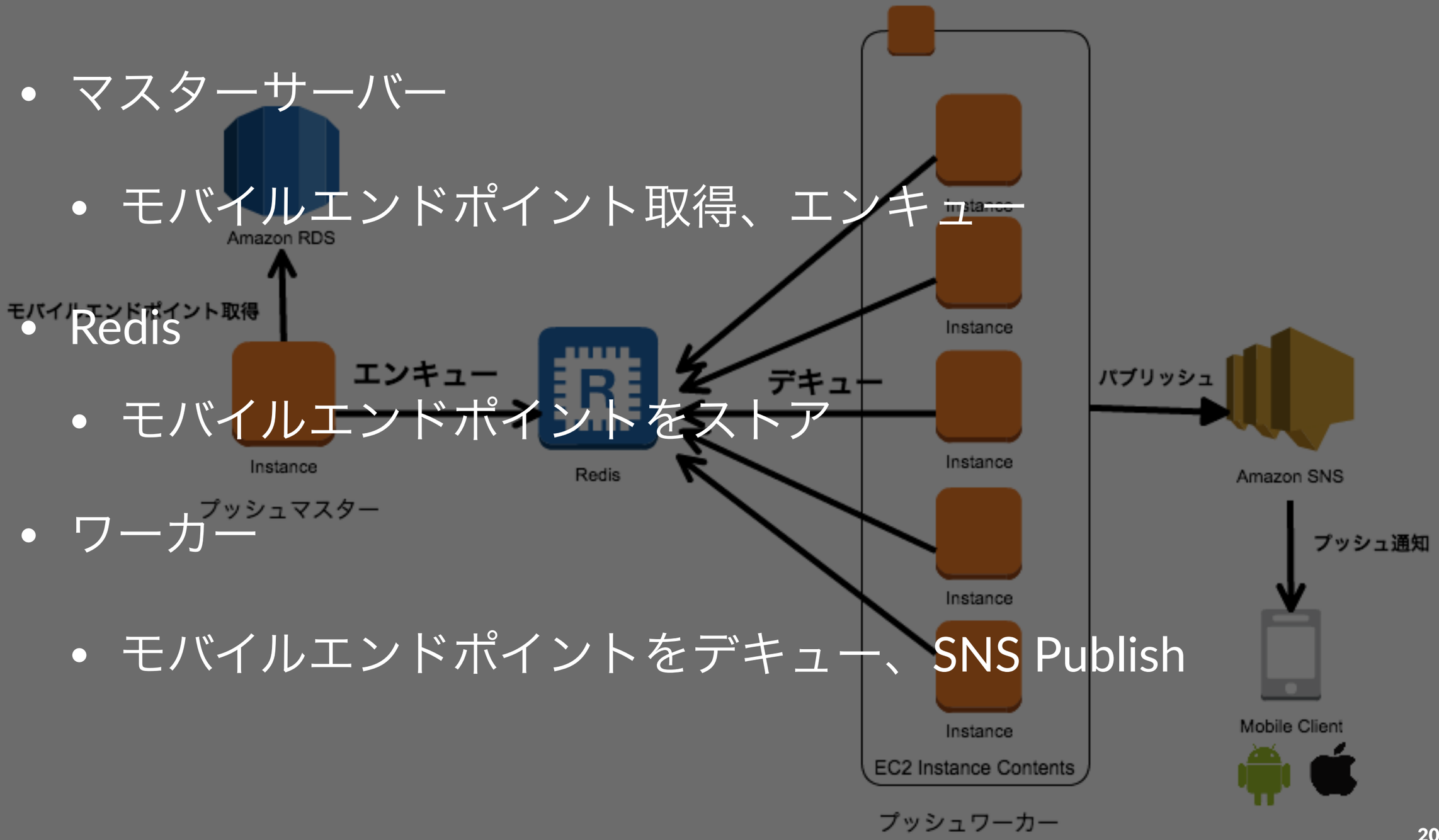

Amazon RDS
モバイルエンドポイント取得
Instance
プッシュマスター
エンキュー
Redis
デキュー
Instance
Instance
Instance
Instance
Instance
EC2 Instance Contents
プッシュワーカー
パブリッシュ
Amazon SNS
プッシュ通知
Mobile Client

定時プッシュ

1. モバイルエンドポイント取得
2. Redisにエンキュー
3. ワーカーがデキュー
4. SNSにPublish



- マスターサーバー
  - モバイルエンドポイント取得、エンキュー
- Redis
  - モバイルエンドポイントをストア
- ワーカー
  - モバイルエンドポイントをデキュー、SNS Publish



1インスタンスあたりの
プッシュ性能

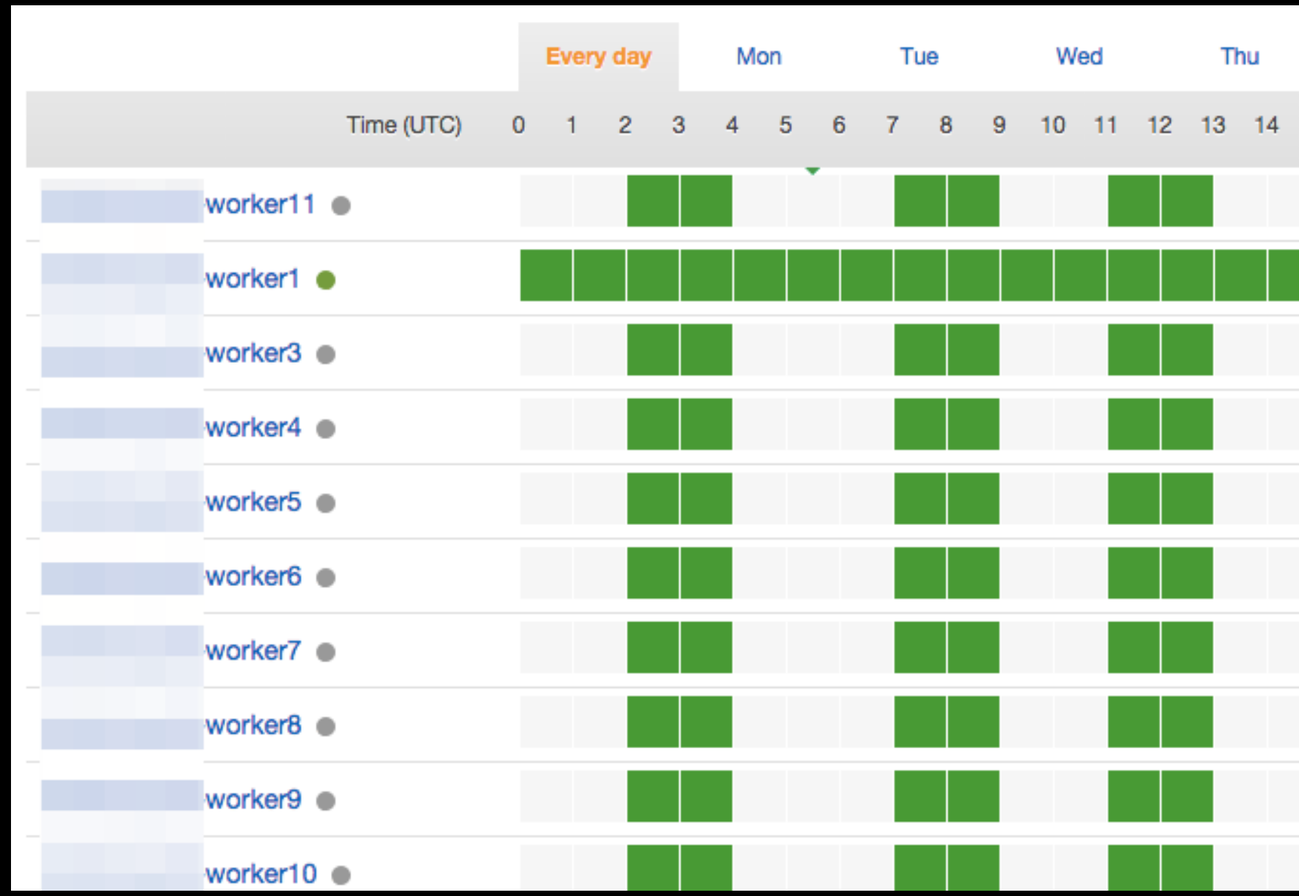
400 Pushes / Sec

1インスタンスあたりの価格

$0.02 / hour (t2micro)

時間帯で増減するインスタンス

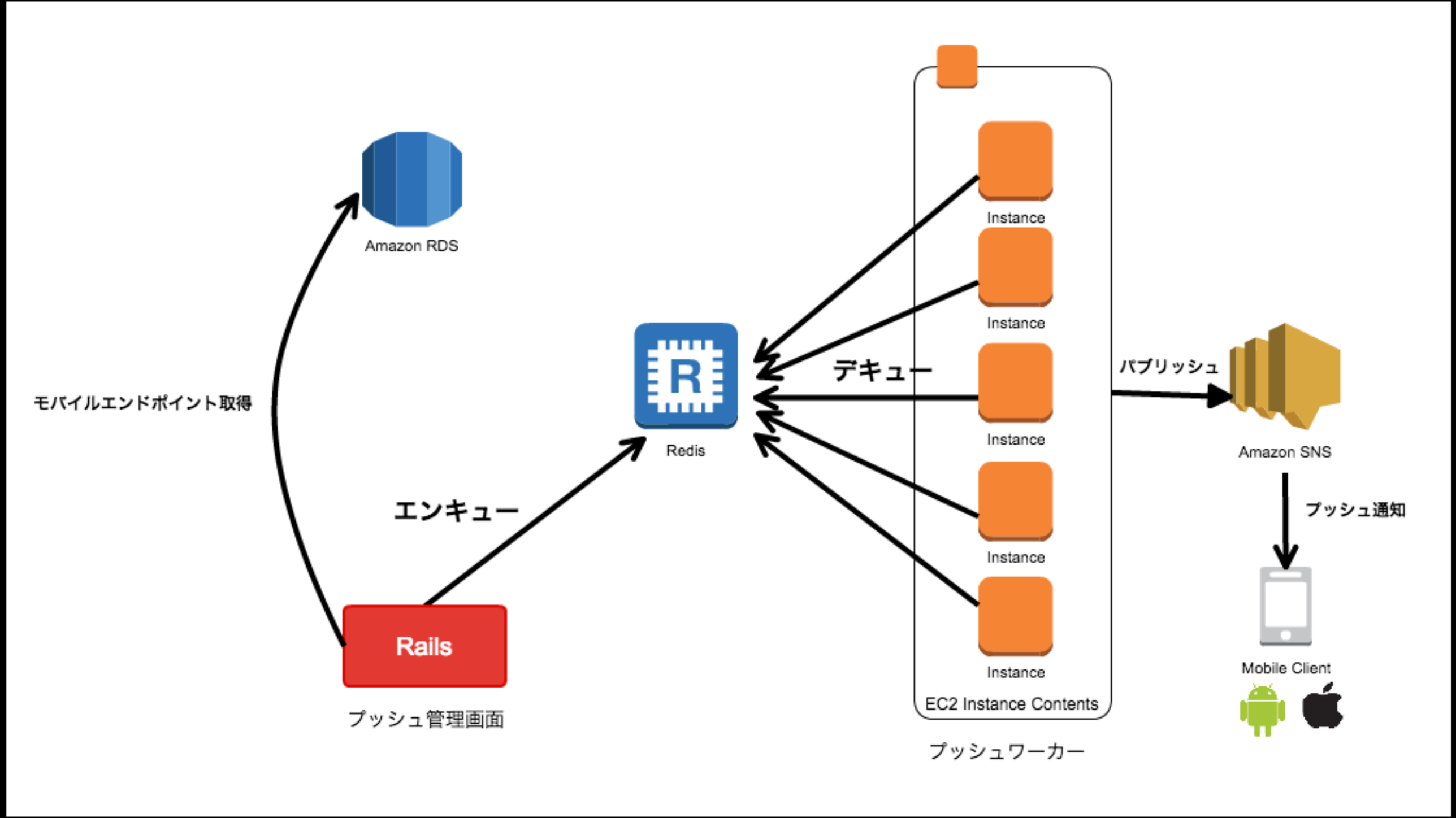
(powered by OpsWorks)



プッシュアーキテクチャ

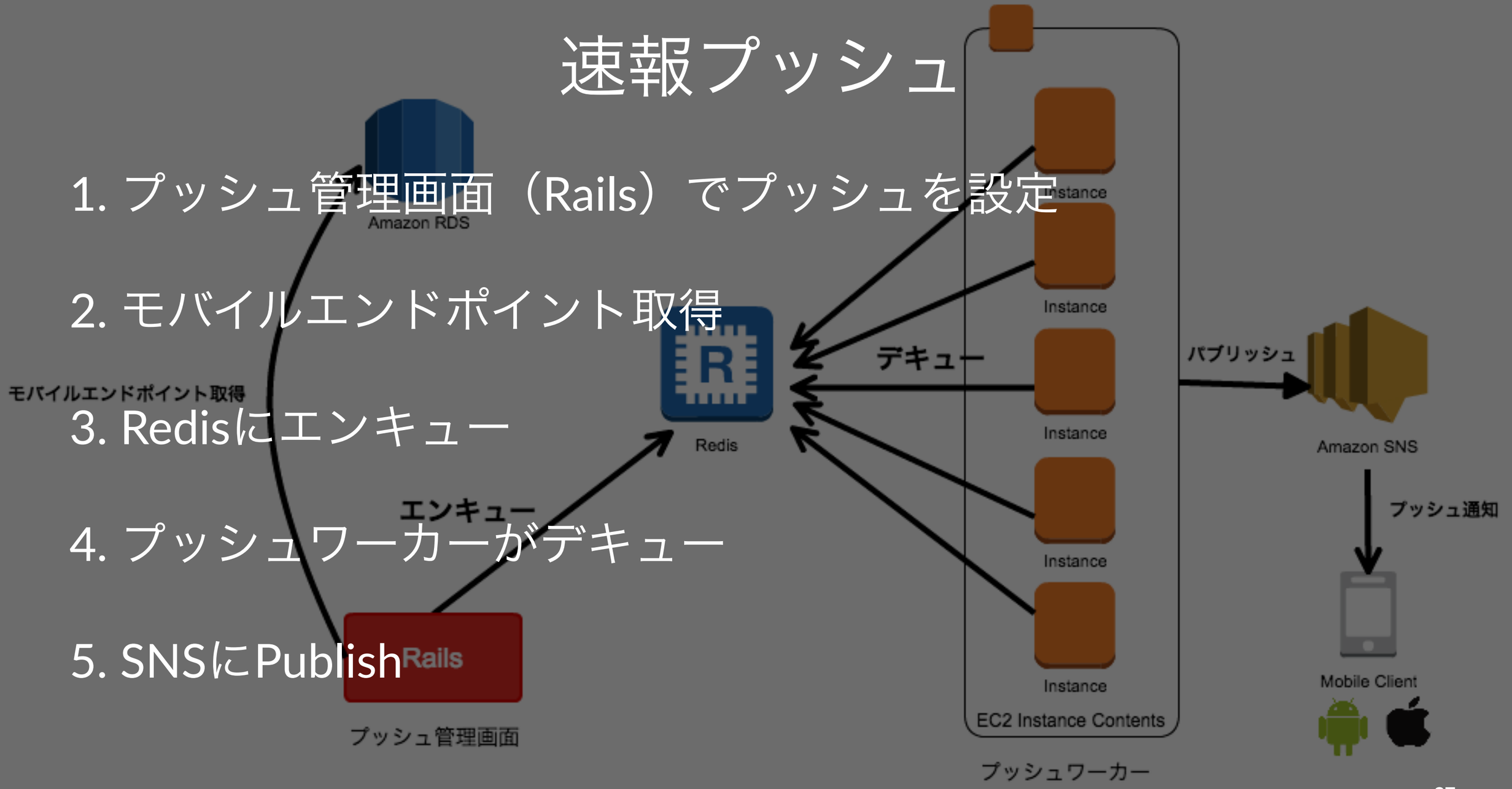
- Redisを用いたシンプルなPub/Sub
- スケーラブルなプッシュワーカー
- タイムベースなインスタンスで低コスト運用

プッシュアーキテクチャ
（速報プッシュの場合）

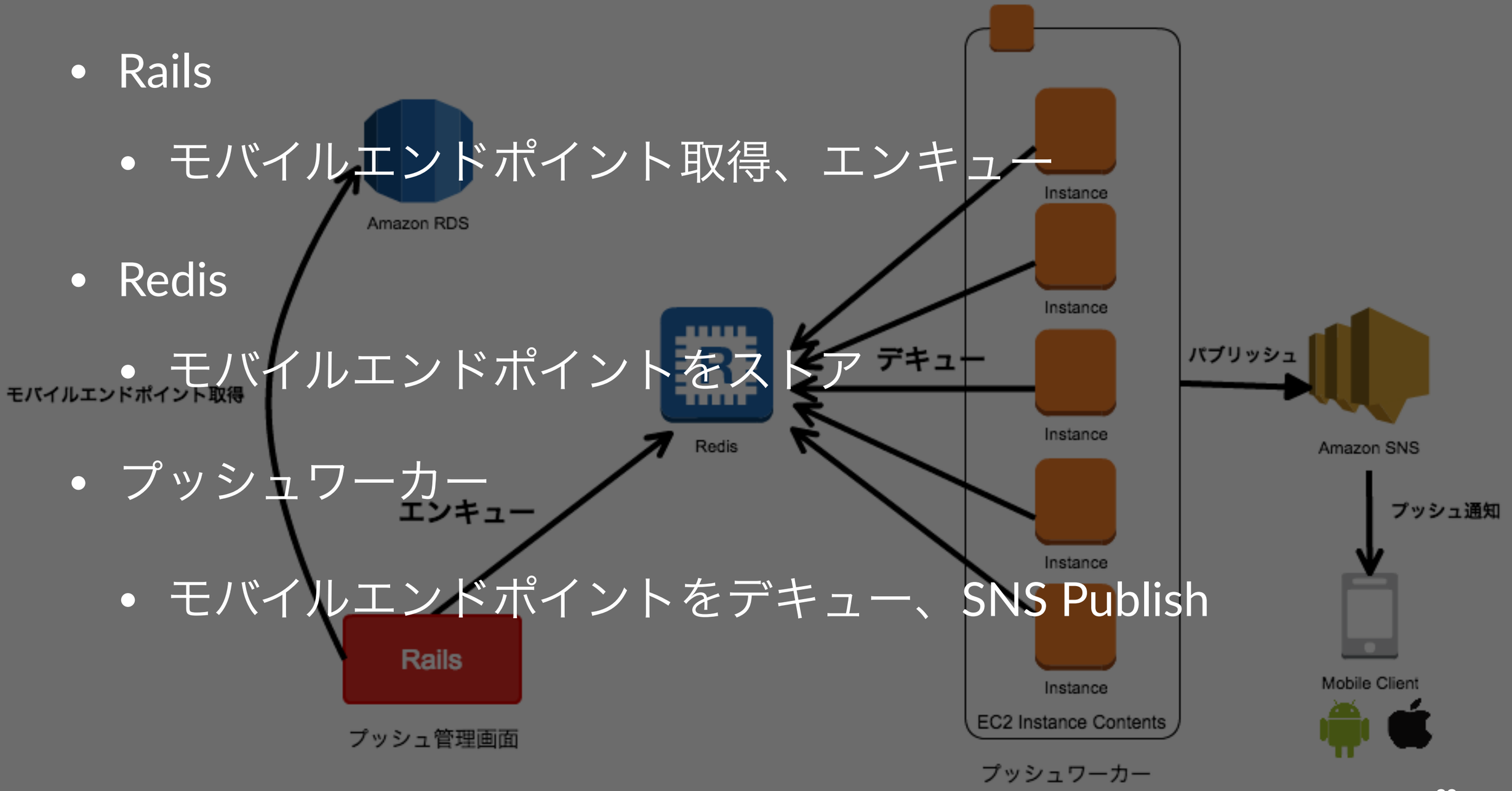

Amazon RDS
モバイルエンドポイント取得
Redis
エンキュー
デキュー
Rails
プッシュ管理画面
Instance
Instance
Instance
Instance
Instance
EC2 Instance Contents
プッシュワーカー
パブリッシュ
Amazon SNS
プッシュ通知
Mobile Client

速報プッシュ

1. プッシュ管理画面（Rails）でプッシュを設定
2. モバイルエンドポイント取得
3. Redisにエンキュー
4. プッシュワーカーがデキュー
5. SNSにPublish



- Rails
  - モバイルエンドポイント取得、エンキュー
- Redis
  - モバイルエンドポイントをストア
- プッシュワーカー
  - モバイルエンドポイントをデキュー、SNS Publish



プッシュ管理画面

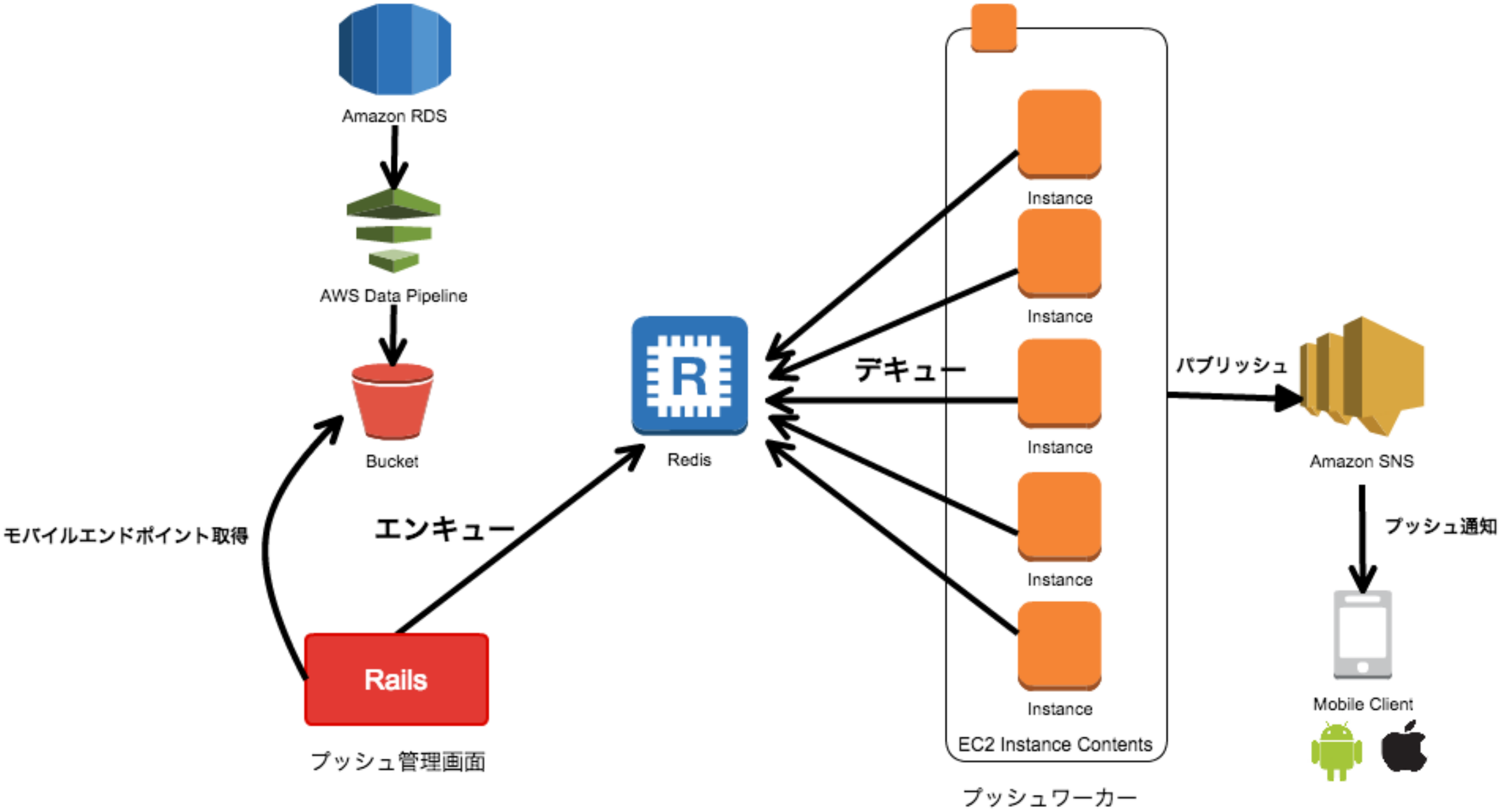
- プッシュ内容・タイトルを設定する管理画面
- Rails ＋ Sidekiq
- Sidekiqワーカーがエンドポイント取得、Redisにエンキュー

問題 ⚠

モバイルエンドポイント取得SQL

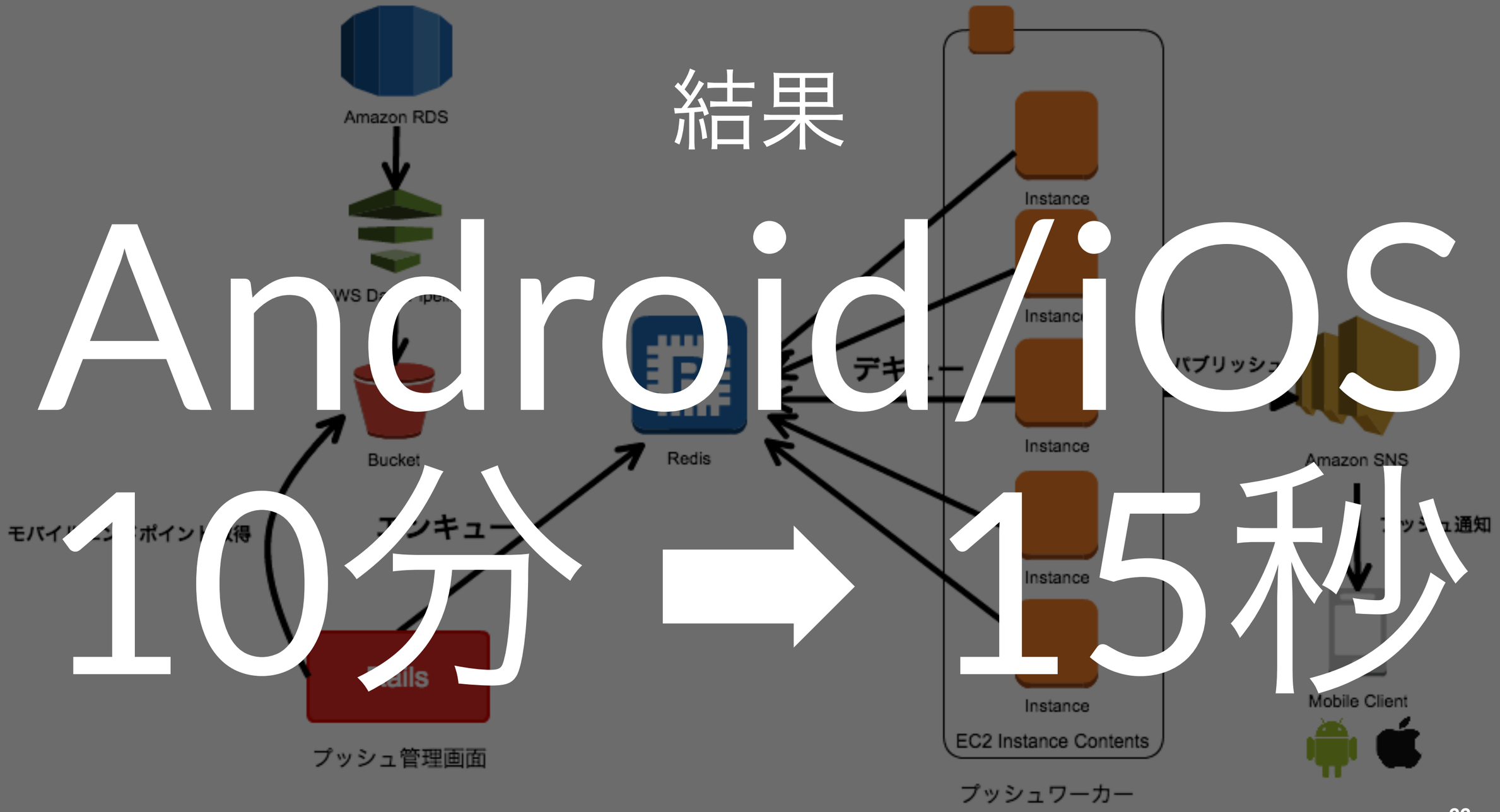
➡ えっ、私のクエリ遅すぎ...

高速化対策 ⚡
endpoint arnをS3ロード化


Amazon RDS
AWS Data Pipeline
Bucket
モバイルエンドポイント取得
エンキュー
Redis
Rails
プッシュ管理画面
デキュー
Instance
Instance
Instance
Instance
Instance
EC2 Instance Contents
プッシュワーカー
パブリッシュ
Amazon SNS
プッシュ通知
Mobile Client

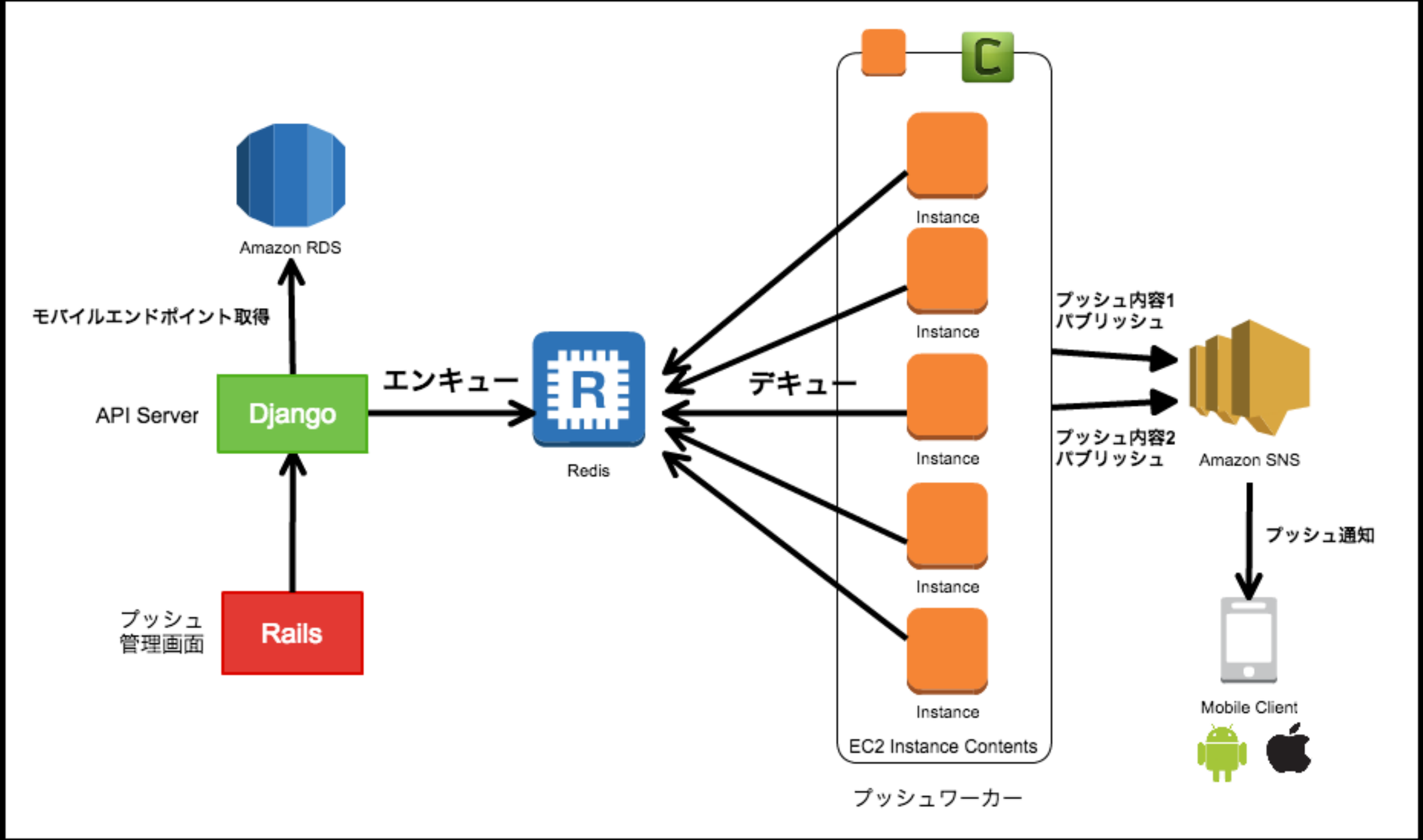
結果

Android/iOS
10分 ➡ 15秒



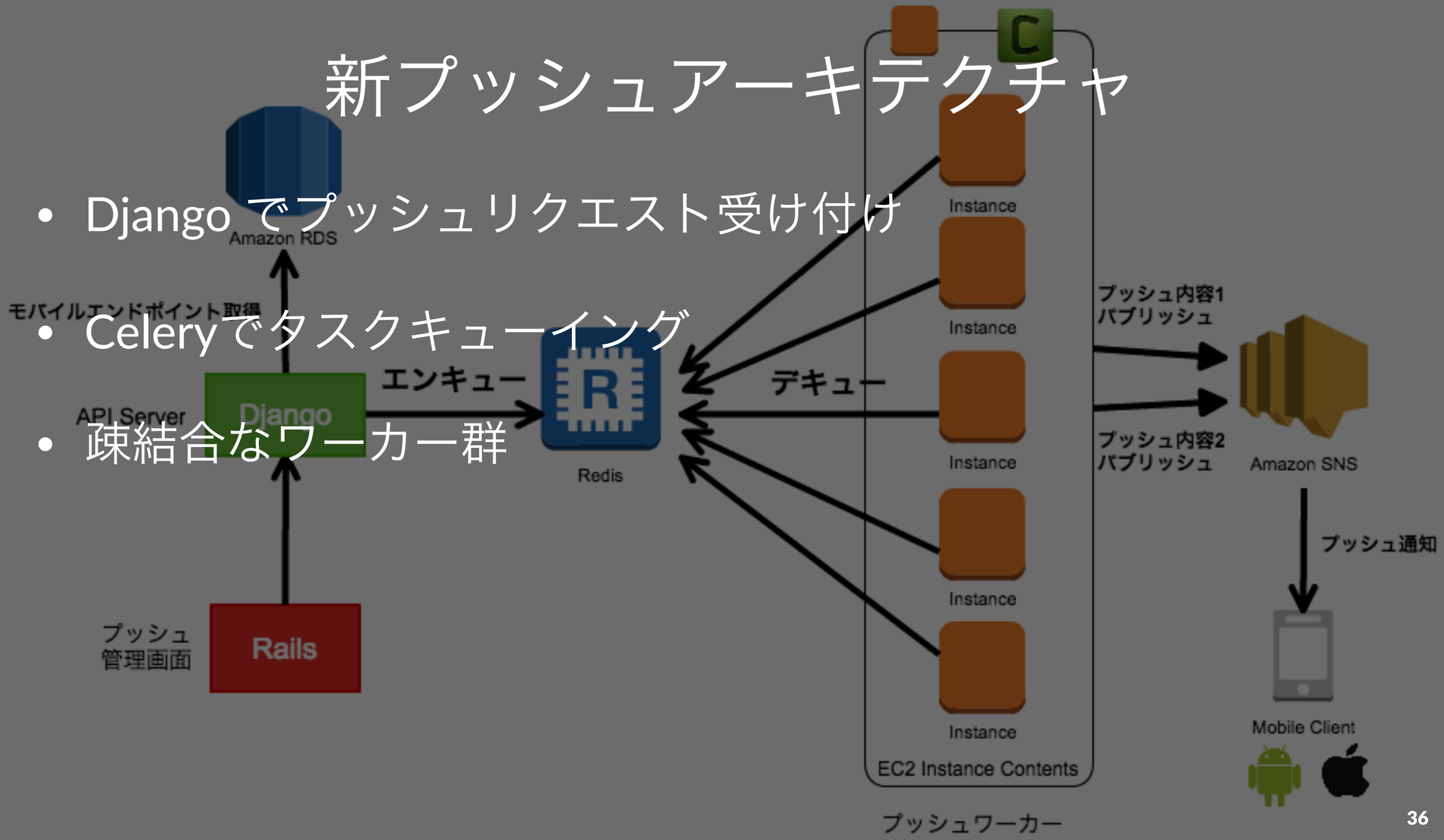
プッシュ第四期
~パーソナライズド・プッシュ時代~

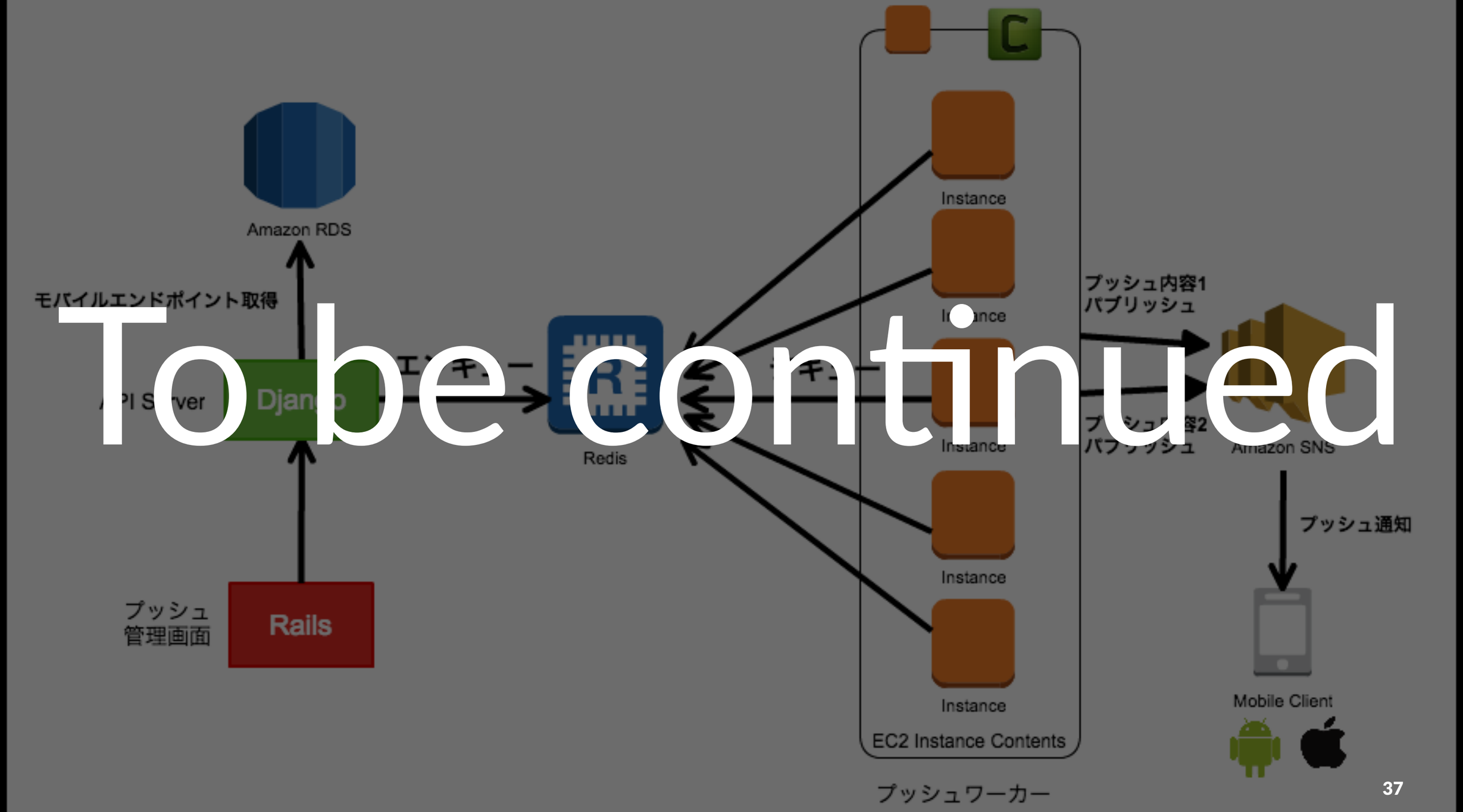
- ユーザーの嗜好に応じたプッシュ通知
- 絶賛開発中 👷


Amazon RDS
モバイルエンドポイント取得
API Server
Django
エンキュー
Redis
デキュー
Rails
プッシュ
管理画面
Instance
Instance
Instance
Instance
Instance
EC2 Instance Contents
プッシュワーカー
プッシュ内容1
パブリッシュ
プッシュ内容2
パブリッシュ
Amazon SNS
プッシュ通知
Mobile Client

新プッシュアーキテクチャ

- Django でプッシュリクエスト受け付け
- Celeryでタスクキューイング
- 疎結合なワーカー群

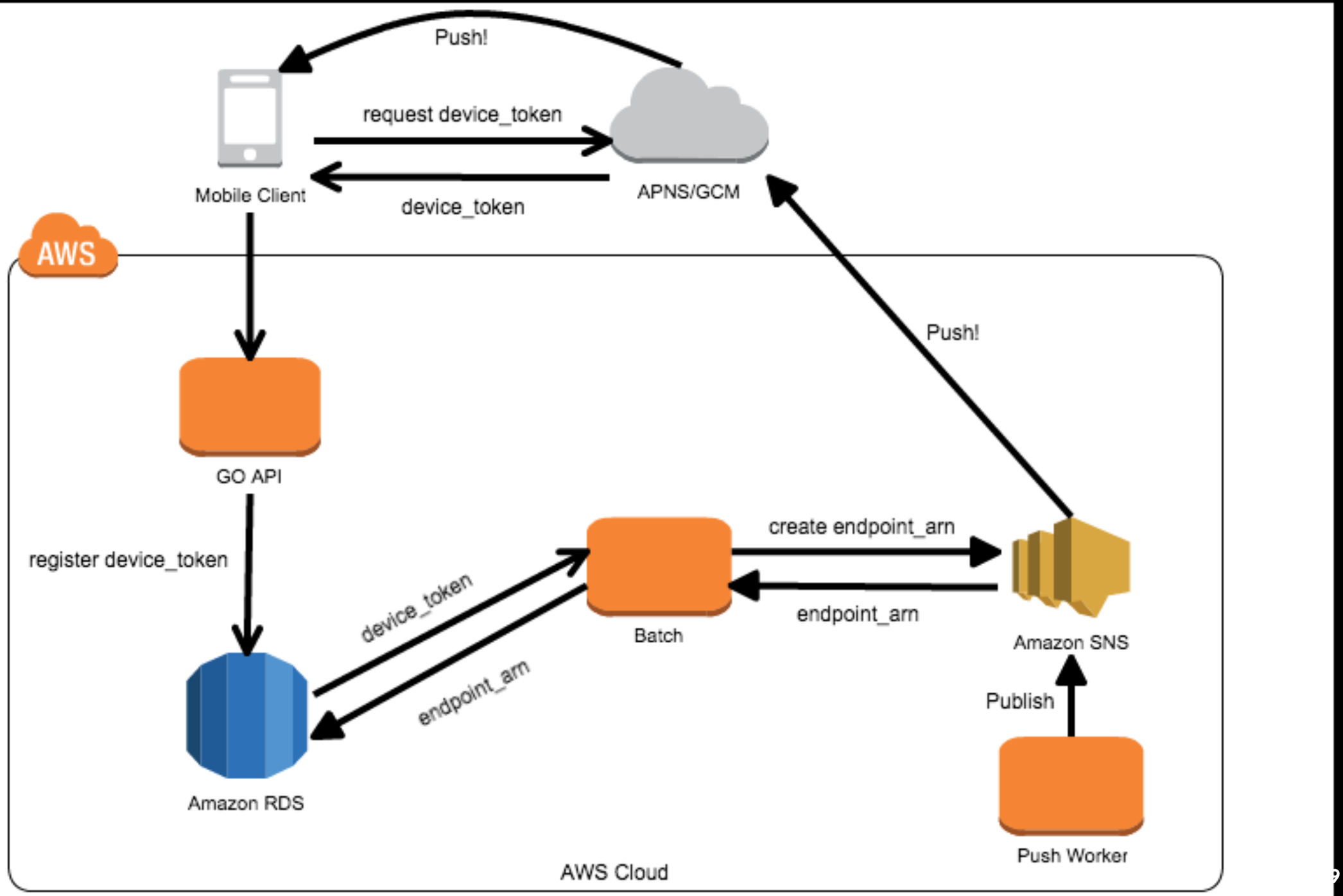




To be continued

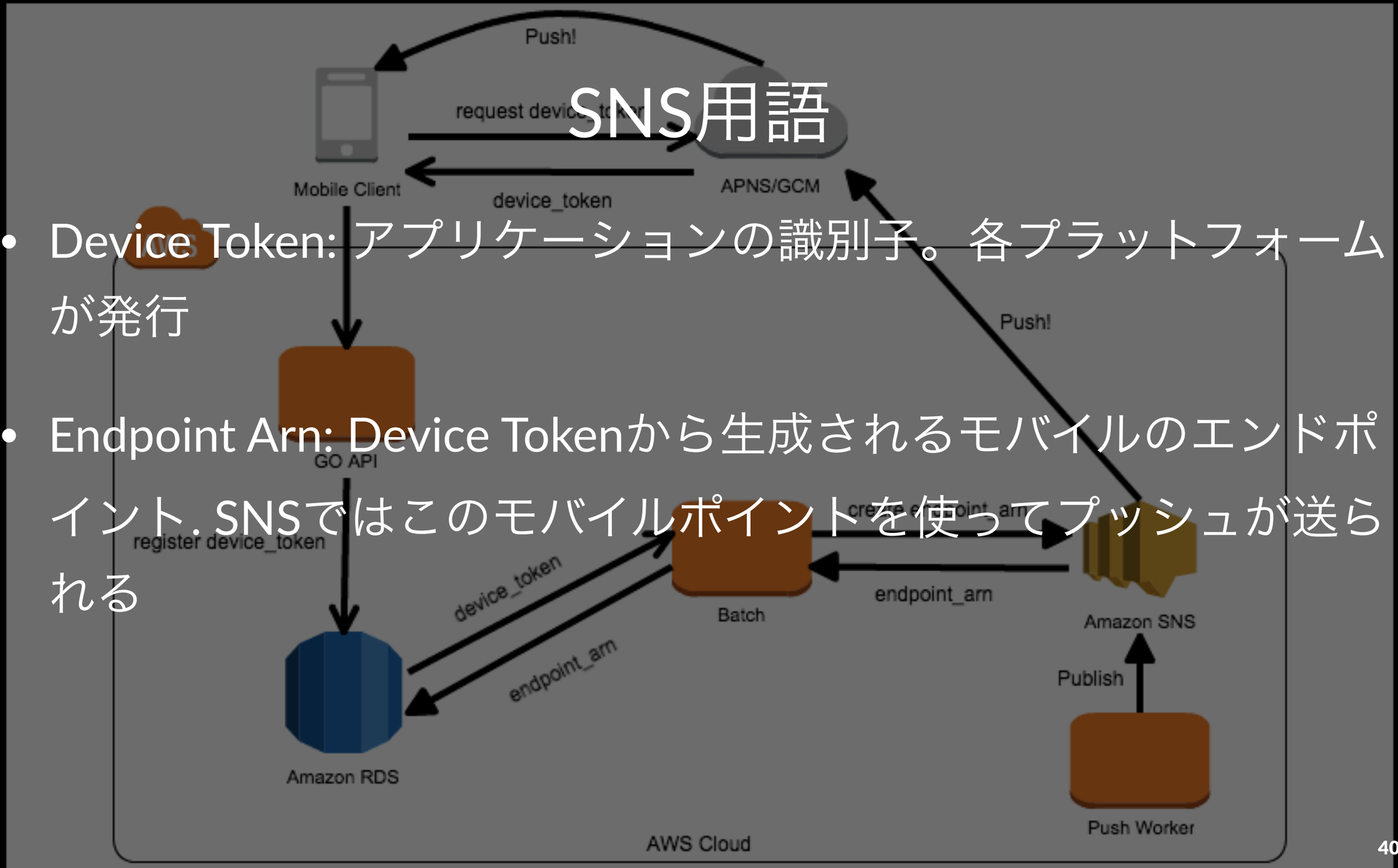
第二部

GunosyでのAmazon SNS利用事例


Push!
request device_token
Mobile Client
APNS/GCM
device_token
AWS
Push!
GO API
register device_token
create endpoint_arn
device_token
Batch
endpoint_arn
Amazon SNS
endpoint_arn
Publish
Amazon RDS
Push Worker
AWS Cloud

SNS用語

- Device Token: アプリケーションの識別子。各プラットフォームが発行
- Endpoint Arn: Device Tokenから生成されるモバイルのエンドポイント. SNSではこのモバイルポイントを使ってプッシュが送られる



Amazon SNSのメリット

- APNS/GCMの各プラットフォームのAPIを抽象化、プラットフォーム毎の違いを意識する必要がない

価格

- 100万リクエスト/月 が無料
- 超過する場合 100万リクエスト毎に $0.5

デバイストークンの管理の難しさ

1. デバイストークンが変わる

2. Enable/Disable制御

3. DisableなEndpoint Arnの削除

デバイストークンが変わる(Android)

Q.デバイストークンってどんなときに変わるの？

1. アプリを再インストールしたとき
2. 「データを削除」されたとき
3. アプリをバージョンアップしたとき[1]

---

[1] 引用元: 【Android】GCMのregistrationIdの一意性 | Monstar Lab, Inc.

デバイストークンが変わる

Q.いつ、なんでリフレッシュするの？

A. google様の気分次第。。。[2]

[2] 引用元: 【Android】GCMのregistrationIdの一意性 | Monstar Lab, Inc.

InstanceID

Instance ID is stable but may become invalid, if:

- App deletes Instance ID
- Device is factory reset
- User uninstalls the app
- User clears app data[0]

---

[0] InstanceID | Google APIs for Android | Google Developers

ポイント ✔

- デバイストークンを定期的に更新
- 更新されたデバイストークンを元にバッチ側で再度モバイルエンドポイント生成

Enable/Disable制御

CreatePlatformEndpoint は PlatformEndpoint が既に存在する場合、成功しますが Enabled にしません。そのため、結果を返す前に PlatformEndpoint が Enabled になっているかどうかチェックする必要があります。[3]

[3] 引用元: Amazon SNS のモバイルトークン管理についてのベストプラクティス | Developers.IO

SNSでは、何度か通知に失敗した場合、endpointがdisableになる

※一度disableになってしまったendpointは、以降どれだけpublishしてもAPNSが叩かれることはない[4]

[4] 引用元: iOS8アップデートでアプリへのプッシュ通知でハマる点 - Qiita

寝てるEndpointArnを起こす(Re-enable処理)

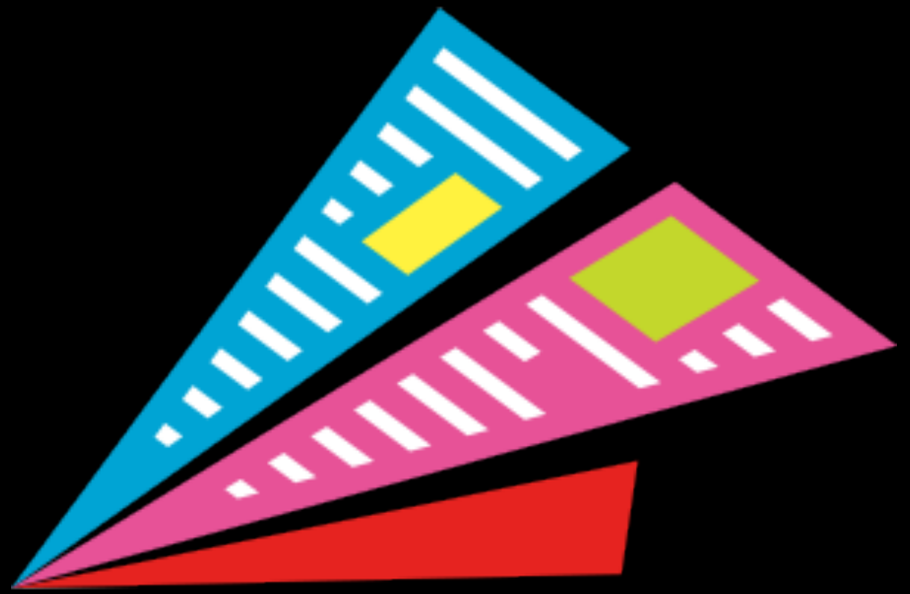
```ruby
def reenable_endpoints(endpoint_arn)
  # client = AWS::SNS.new ###省略###
  attributes = client.get_endpoint_attributes(endpoint_arn: endpoint_arn)[:attributes]
  if attributes['Enabled'] == 'false'
    # re-enable endpoint
    client.set_endpoint_attributes(endpoint_arn: endpoint_arn, attributes: {'Enabled' => 'true'})
  end
end
```

DisableなEndpoint Arnの削除

- モバイルエンドポイントのムダなPublishはしたくない
- SNSへのPublish時にdisableなEndpointArnは削除

🏆 SNS ベスト・プラクティス 🏆

- デバイス毎(iPhone/Android)にデバイストークンを管理
- 定期的にデバイストークンをチェックして更新
- モバイルエンドポイントのRe-enable処理
- disableなEndpointArnは削除してムダなPublishは控える



ありがとうございました。

We're hiring!!

参考情報

- 【Android】GCMのregistrationIdの一意性 | Monstar Lab, Inc.
- Amazon SNS のモバイルトークン管理についてのベストプラクティス | Developers.IO
- 気軽なSNS Mobile Push の話
- iOS8アップデートでアプリへのプッシュ通知でハマる点 - Qiita