サーバレスアーキテクチャによる外部API連携の実現

時武 佑太
取締役CTO
株式会社LegalForce

aws SUMMIT ONLINE

発表者について

時武 佑太

株式会社LegalForce 取締役CTO　@tokichieto

- 大学時代、インターンとして複数のWebサービス開発に携わる
- 2016年、東京大学大学院 情報理工学系研究科を卒業
- 同年、株式会社ディー・エヌ・エーにエンジニアとして新卒入社
- 2017年10月からLegalForceに参画し、現職
- 2019年11月にCTO of the year 2019を受賞

会社概要

LegalForce

社名	株式会社LegalForce（カブシキガイシャ リーガルフォース）
代表	代表取締役CEO 角田望
資本金	153,000千円（2020年4月15日時点）
設立	2017年4月21日
従業員数	55名（2020年7月10日時点）
事業内容	法律業務に関するソフトウェアの研究・開発・運営・保守
ウェブサイト	https://www.legalforce.co.jp/

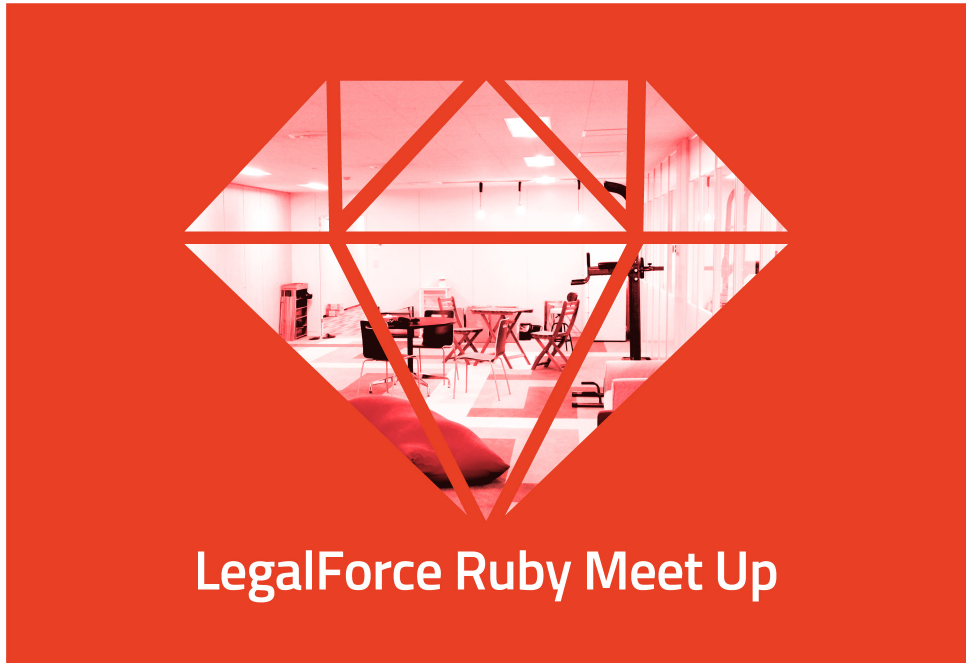
我々のミッション

全ての契約リスクを
制御可能にする

私たちの経済活動は消費者や企業が取り結ぶ、
大小さまざまな契約で成り立っています。
それは同時に「不利な条件や法令違反などの契約リスクが潜んでいる」ということ。
LegalForceは、最新のテクノロジーと法務の知見を組み合わせて、
契約リスクを可視化し、コントロール可能な状態へ導きます。
あるべき権利が適切に守られ、不測の事態が防がれるために。
そして、より豊かで信頼に満ちた経済社会を目指します。

LegalForceはRubyの会社です

- まつもとゆきひろさんが技術顧問に就任
- LegalForce Ruby Meetupを開催
- RubyKaigiや勉強会・学会等でのスポンサー活動



契約書レビューサービスLegalForce

- 契約書レビューを自然言語処理技術等を用いてサポートするプロダクト

LegalForce導入実績

- 2019年4月の正式版リリース以来、1年あまりで400を超える企業・法律事務所に導入されている

導入数	400 社・事務所以上
ユーザー数	1,000 名以上
AIによる 自動レビュー件数	40,000 件以上

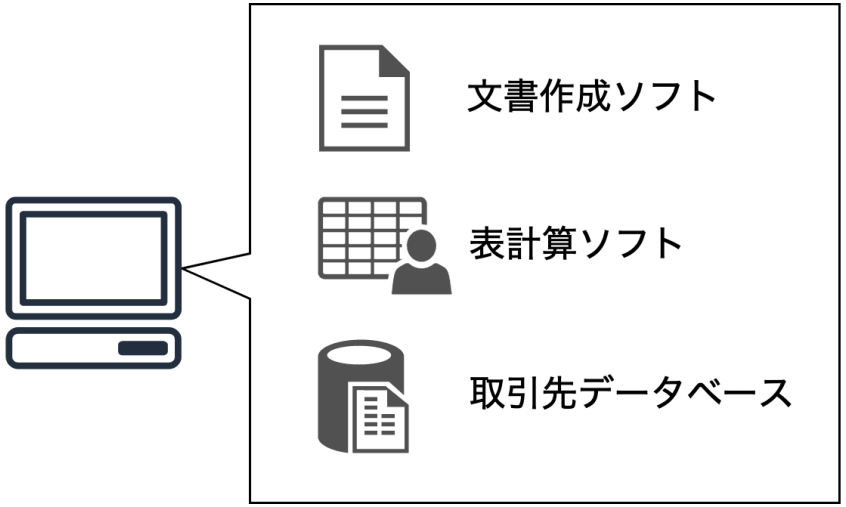
契約書管理サービスMarshall

- 締結した契約書を一元的に管理するためのプロダクト

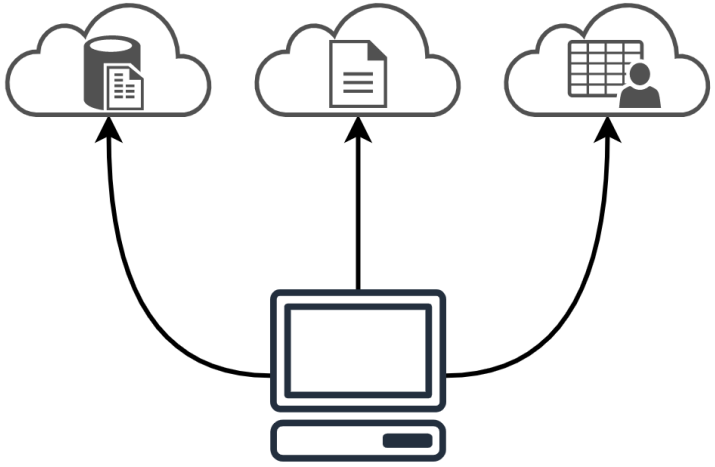
LegalForceにおけるAPI連携戦略

クラウドネイティブ時代の社内システム

- かつて内製・受託開発がメインだった社内システムはSaaSの利用がメインに
- オフィススイートは買い切りのアプリケーションではなくSaaS型が普及しつつある
- 社内のコミュニケーションツールとしてビジネスチャットの導入が進んでいる



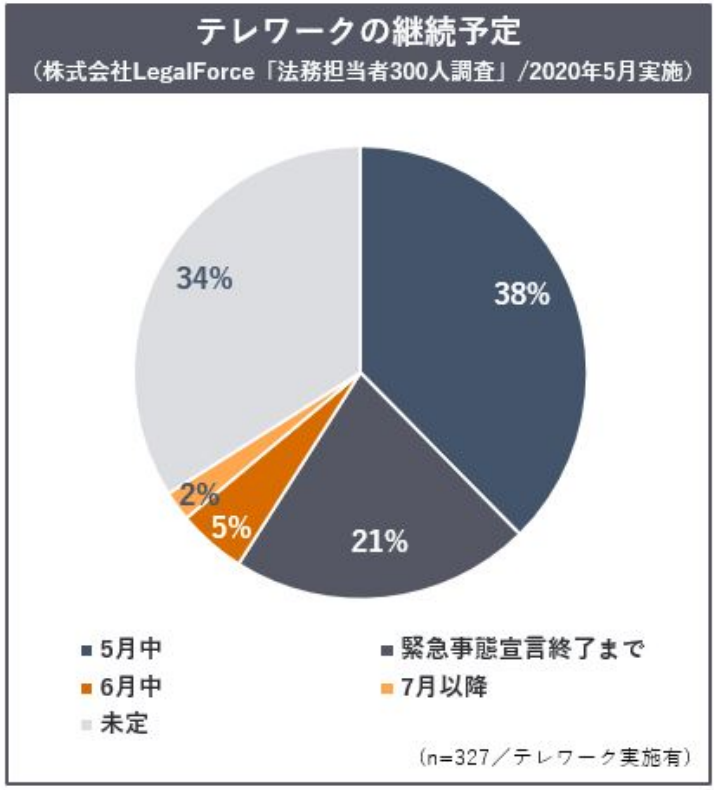
従来

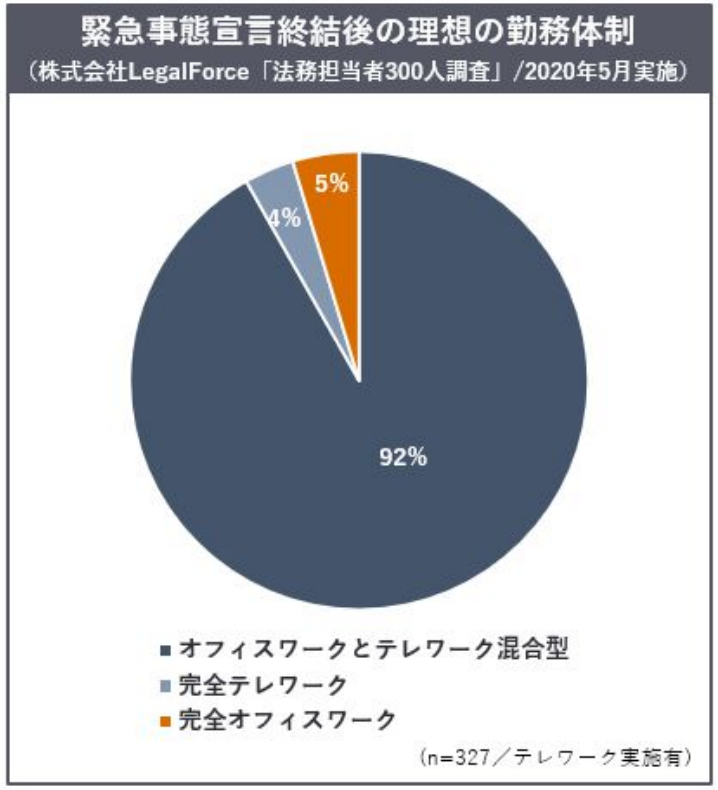


近年

新型ウイルスによる急激なDX (Digital Transformation) 熱の高まり

- テレワークの普及により、アナログな作業をデジタルで置き換える流れができつつある
- 弊社が5月に実施した調査によると、テレワークの継続を望む法務従事者は92%

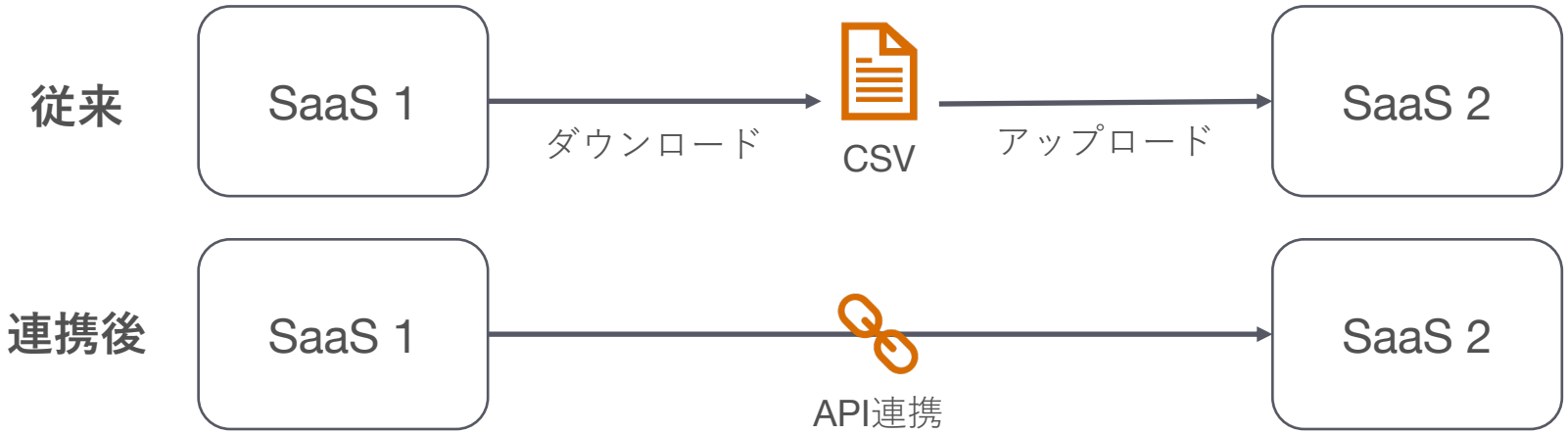




ユーザー目線での利便性の追求

- ほとんどのユーザーは複数のSaaSを使い分けながら仕事をしている
- 複数のSaaSによるワークフローをいかにシームレスに連携させるかが仕事効率化のカギになる
- SaaSのハブとなるiPaaSが近年注目を集めている

- B2B SaaSにおいては連携機能がユーザーへの提供価値を向上させる上で非常に重要な戦略になる



契約書のライフサイクル

LegalForce・Marshallの位置づけ

電子契約締結サービスとの連携

- LegalForceのプロダクト群では契約書の締結が網羅できていない
- 市場には複数の電子契約締結サービスが存在しており、利用者も拡大している

- 連携第一弾として「**GMO電子印鑑Agree**」との連携を発表

連携後の契約書ライフサイクル

連携のアーキテクチャ

LegalForceの技術スタック

- バックエンドの主な利用言語とフレームワーク

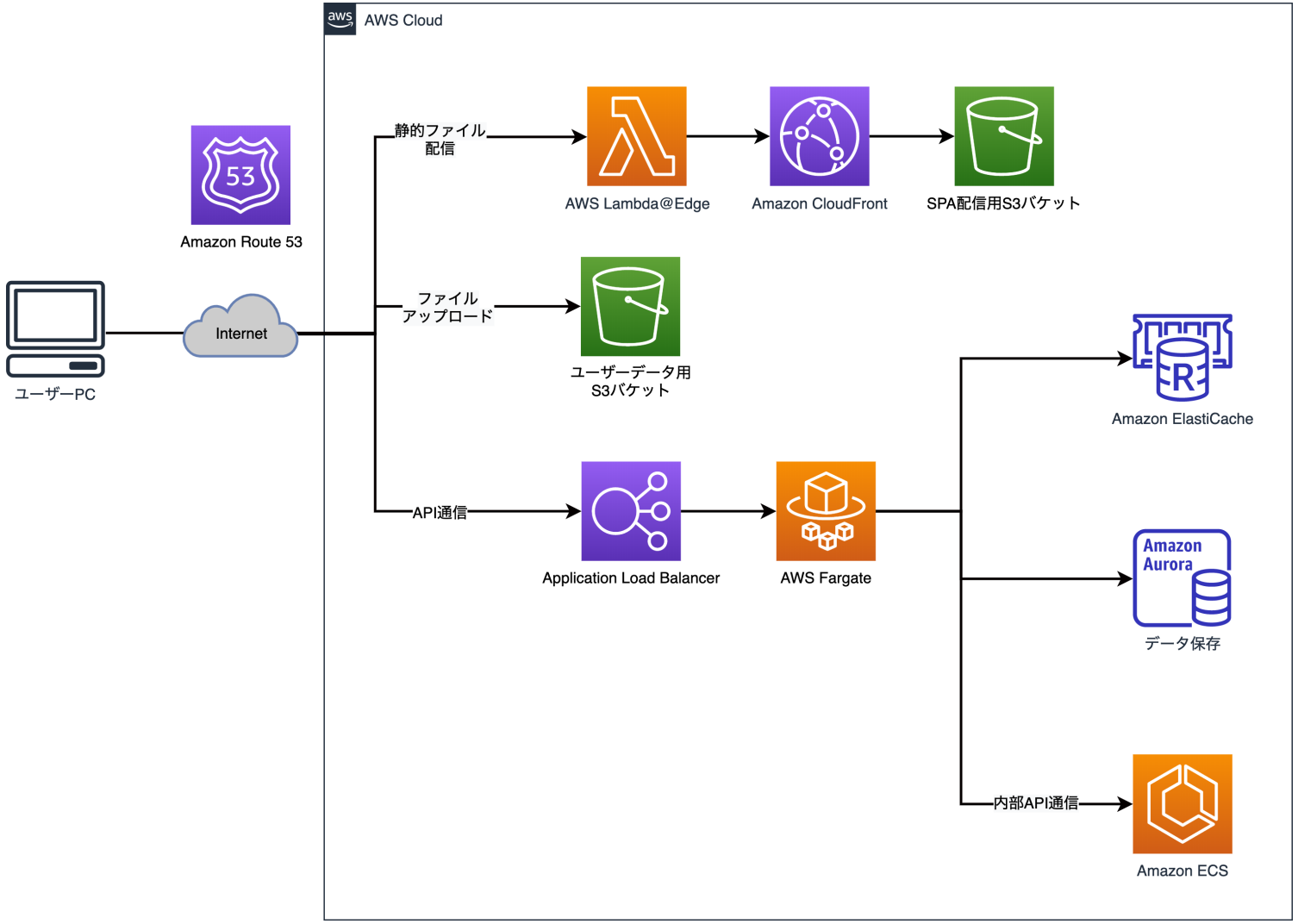
Ruby

python™

Hanami

既存のアーキテクチャ

- SPA + JSON API構成
- 内部APIをマイクロサービスとして構築
- データベースにはAmazon Aurora MySQLを利用



連携のスケジュール

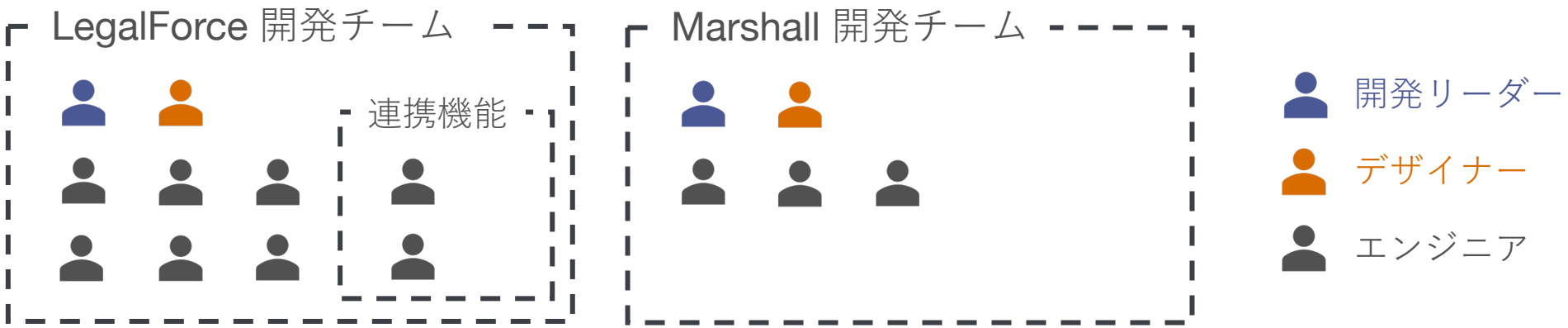
- 2020年5月　… GMO電子締結Agreeとの連携を発表
- 2020年6月　… 仕様策定やデザイン含めて開発を開始
- 2020年7月末 … 開発完了
- 2020年8月　… 品質検証・内部での試験運用・リリース

- 開発に使える期間は**約2ヶ月**

開発チームの構成

- LegalForce開発チームはエンジニアが9人＋デザイナー1人
  - フロント 2人、サーバー＋フロント 3人、サーバー 1人、インフラ 2人
- Marshall開発チームはエンジニア4人＋デザイナー1人

- LegalForce開発チームのエンジニア2人を連携機能にアサイン



サーバレスアーキテクチャの採用

- 2人のバックエンドエンジニアで2ヶ月以内に作らなければいけない
  - コード、インフラともになるべく少ない手数で実装したい
  - 拡張性や柔軟性がある程度確保できる構成で実現したい

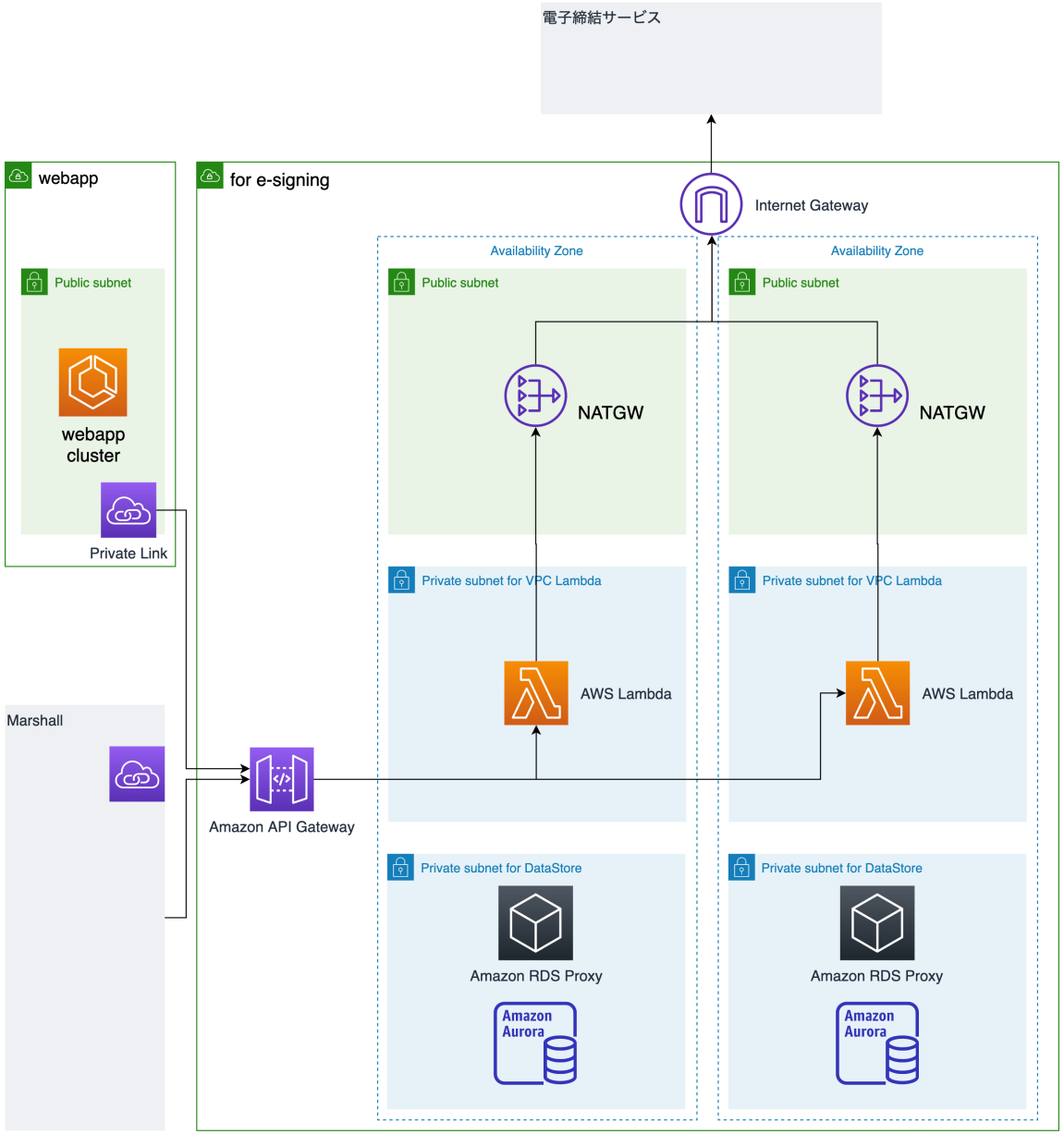
- AWS LambdaとAmazon API Gatewayによるサーバレスアーキテクチャを採用

AWS Lambda

Amazon API Gateway

サーバレスアーキテクチャの採用

- 最小限のコードで動かせる
- 最小限のインフラ構築で動かせる
- スケーラビリティを簡単に高められる



Serverless Frameworkの導入

- 複数ベンダーに対応したサーバレスアプリケーションの構成管理ツール
- YAMLファイルで簡単に構成管理ができる
- テンプレートからのfunction作成、デプロイ、実行と操作も覚えやすい

- ツールの導入による開発方法・運用の画一化

画像は https://www.serverless.com/ より引用

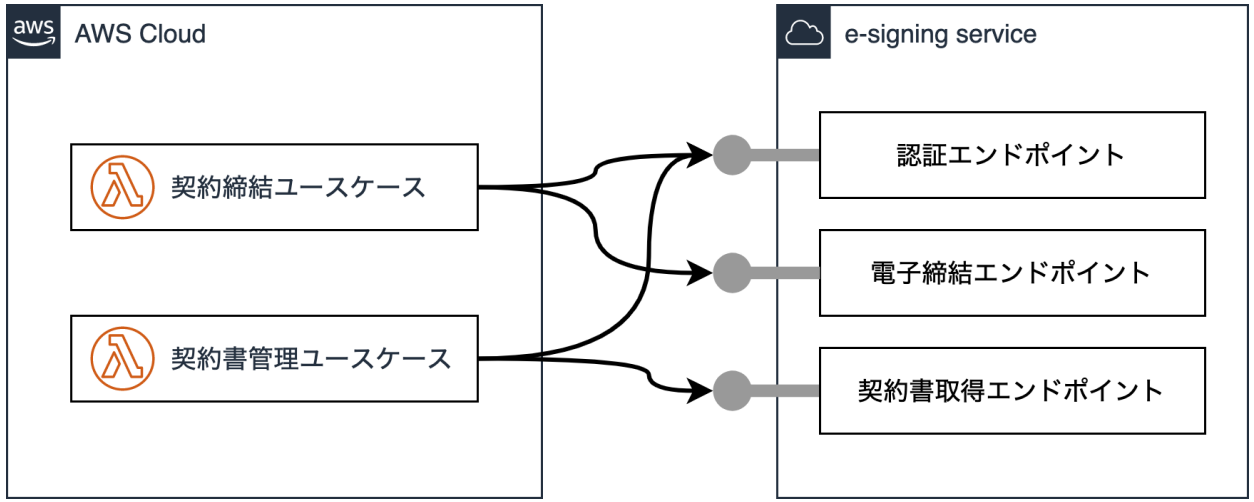
AWS Lambdaの柔軟性の高さ

- ここ数年の大規模アップデートにより、さらに使いやすくなった
- re:Invent 2018
  - AWS Lambda Layers
  - Rubyランタイム、カスタムランタイム
  - Application Load BalancerによるLambdaサポート
- re:Invent 2019
  - Provisioned Concurrency for AWS Lambda
  - Amazon RDS Proxy with AWS Lambda （プレビュー版）

サーバレスアーキテクチャにおけるFunctionの分割方針

- 1つの処理ごとにfunctionを分けるのがベストプラクティスとされる
- AWS Step Functionsと連携することで複数のfunctionをまとめることもできる
- 分けすぎるとfunctionが大量にでき煩雑になるためトレードオフ

- 今回はLegalForce側APIのユースケースごとにfunctionを分ける形で設計



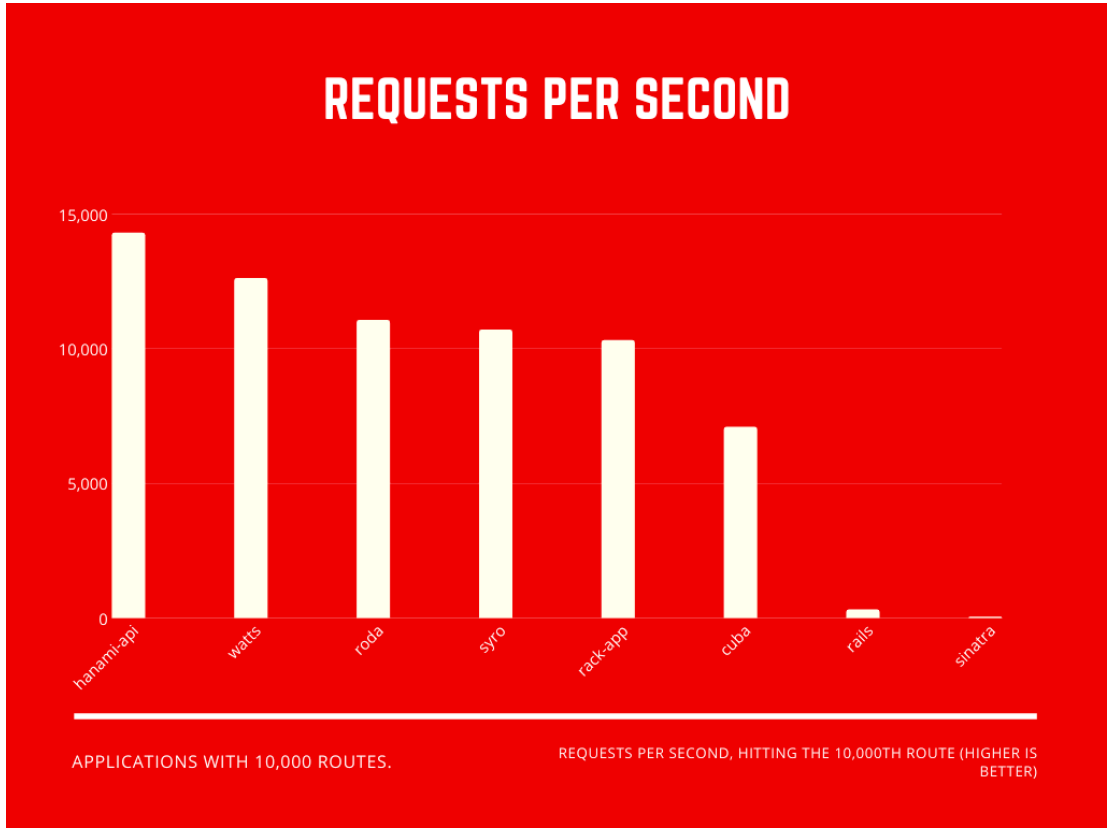
Ruby製サーバレス向けフレームワークの採用

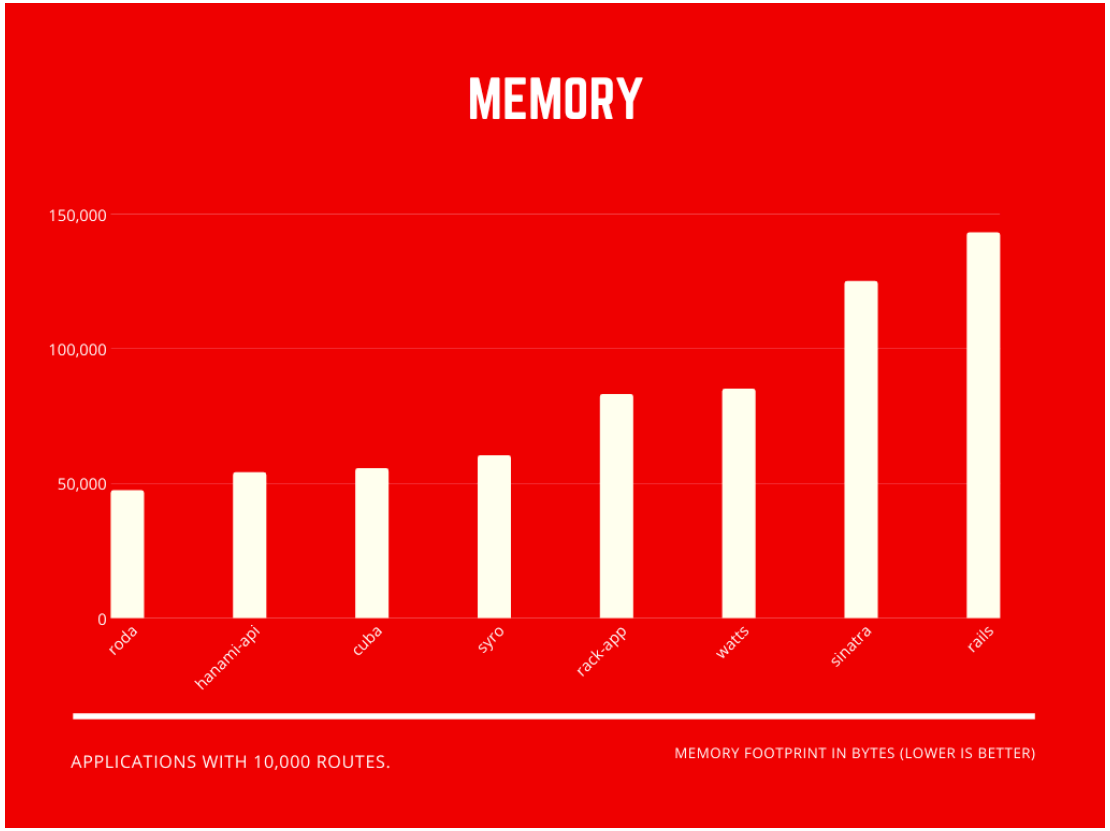
- Amazon API Gatewayからのリクエスト、レスポンス部分をフレームワークに任せたい
- 以下の2つのフレームワークを検討
  - Jets
  - Hanami API

- 社内での利用実績のあったHanami APIを用いて開発を進めることに

Hanami APIとは

- Hanamiフレームワークから APIに必要な要素以外を除いた軽量バージョン
- rack単体で使うよりもメモリ性能、リクエスト処理能力に優れる





比較画像は http://hanamirb.org/blog/2020/02/26/introducing-hanami-api/ より引用

Hanami APIによるLambda Function

endpoint.rb

```
class Endpoint
  def call(env)
    connection = Faraday.new(url: 'https://api.e-sign.example')
    res = connection.post(...)

    [200, {}, res.body['result']]
  end
end
```

app.rb

```
class App < Hanami::API
  get '/endpoint', to: Endpoint.new
end
```

handler.rb

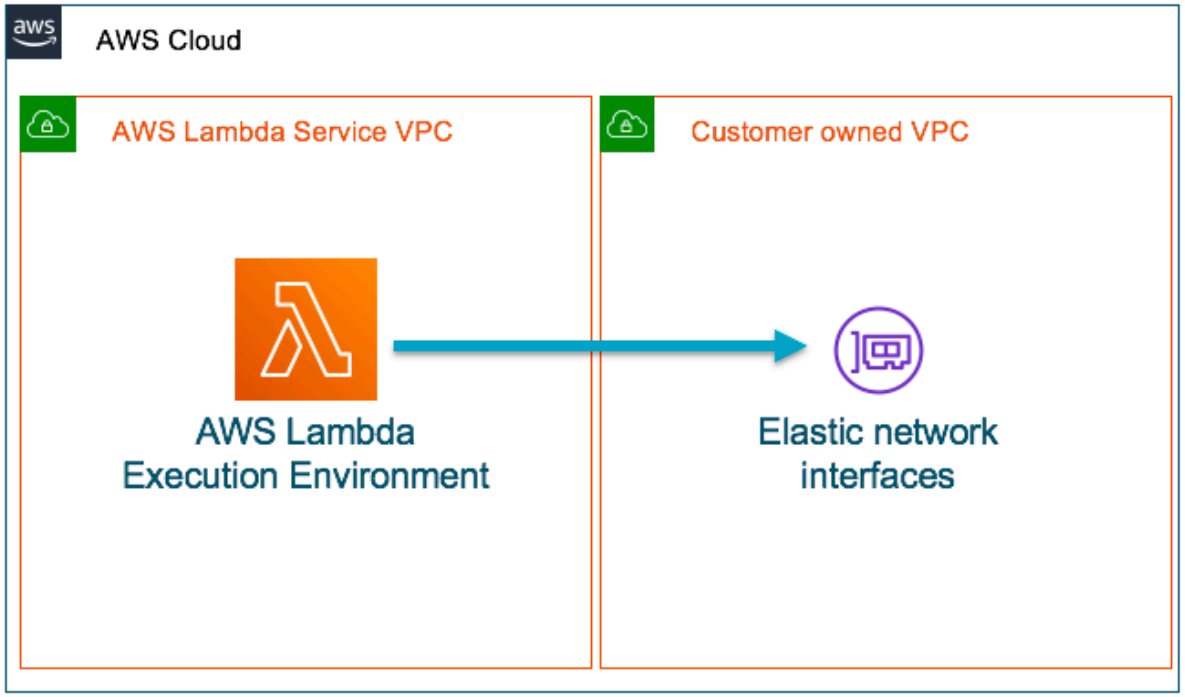
```
$app ||= App.new

def call(event:, context)
  AWS::Lambda.call($app, event, context)
end
```

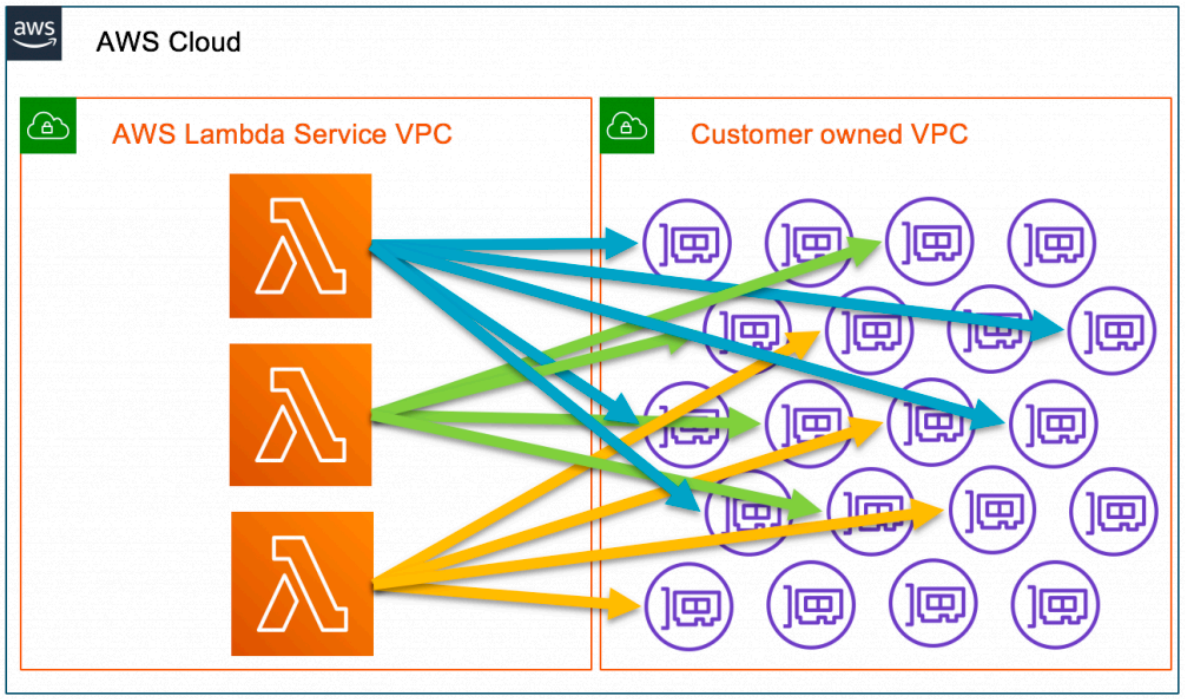
AWS LambdaとAmazon RDSとのつなぎ込み

- LambdaとRDSを接続するのは一般的にはアンチパターンと言われてきた
  - Amazon VPCに作成できるENIの制限
  - RDSインスタンスの同時接続数の制限
- 昨今の機能アップデートにより、これらのアンチパターンは過去のものとなりつつある

VPC Lambdaにおけるネットワーク環境が大幅に改善

- 従来Lambda FunctionがVPCと接続する方法はENI経由だった
- 関数がスケールすると大量のENIが生成され、VPCのIP枯渇やパフォーマンス低下の原因に

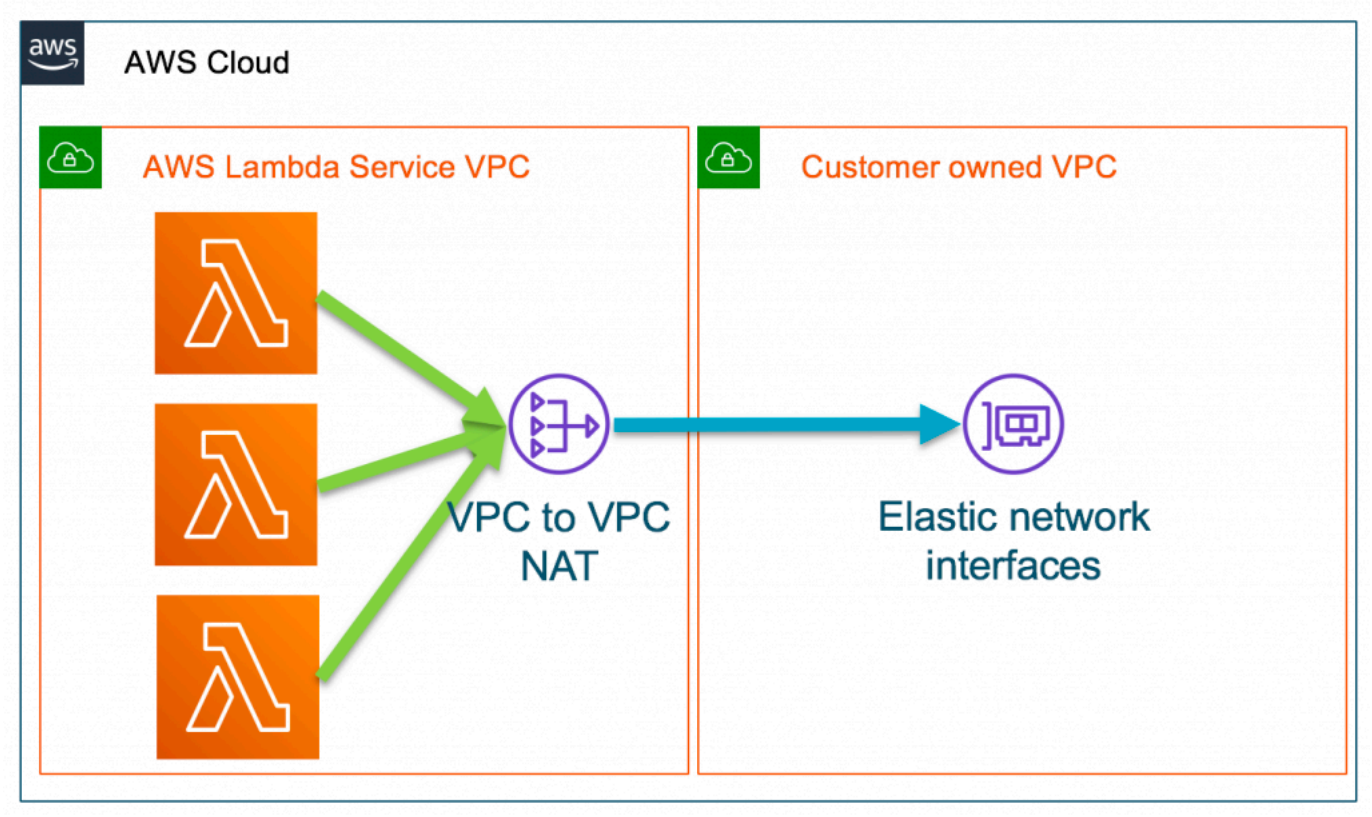




画像は https://aws.amazon.com/jp/blogs/news/announcing-improved-vpc-networking-for-aws-lambda-functions/より引用

Amazon VPC – AWS Lambdaでのネットワーク環境が大幅に改善

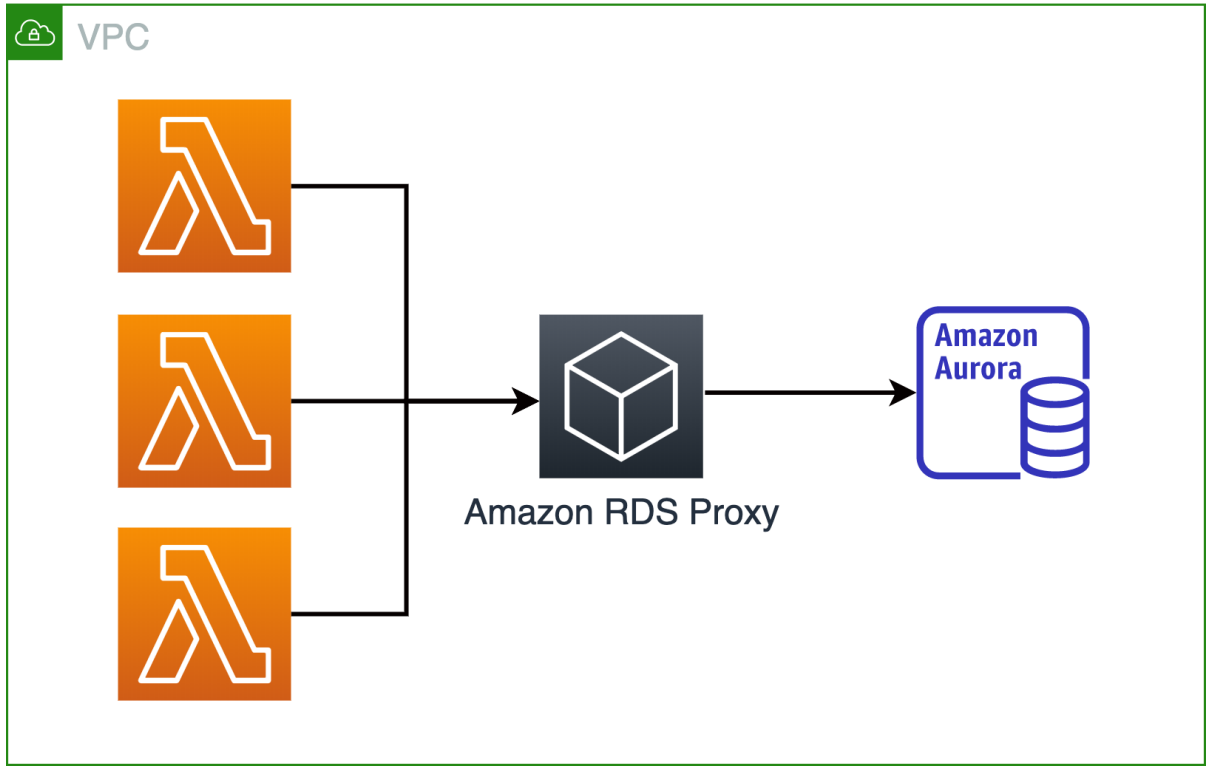
- 2019年9月からAWS Hyperplaneを通したVPC接続に切り替わった
- これによってLambda FunctionとVPCの接続環境が大きく改善された



画像は https://aws.amazon.com/jp/blogs/news/announcing-improved-vpc-networking-for-aws-lambda-functions/より引用

Amazon RDS Proxyが東京リージョンでも一般公開に

- RDS Proxyを用いることで、RDSインスタンスへの接続をプールできる
- LambdaやECSからデータベースに高速・大量にアクセスする際に接続を効率化



Amazon Auroraのインスタンスサイズと同時接続数について

- t2, t3系のsmall, mediumインスタンスはデフォルトの同時接続数上限が少ない
- 本番環境用途ではlarge以上を使うようにするのが無難

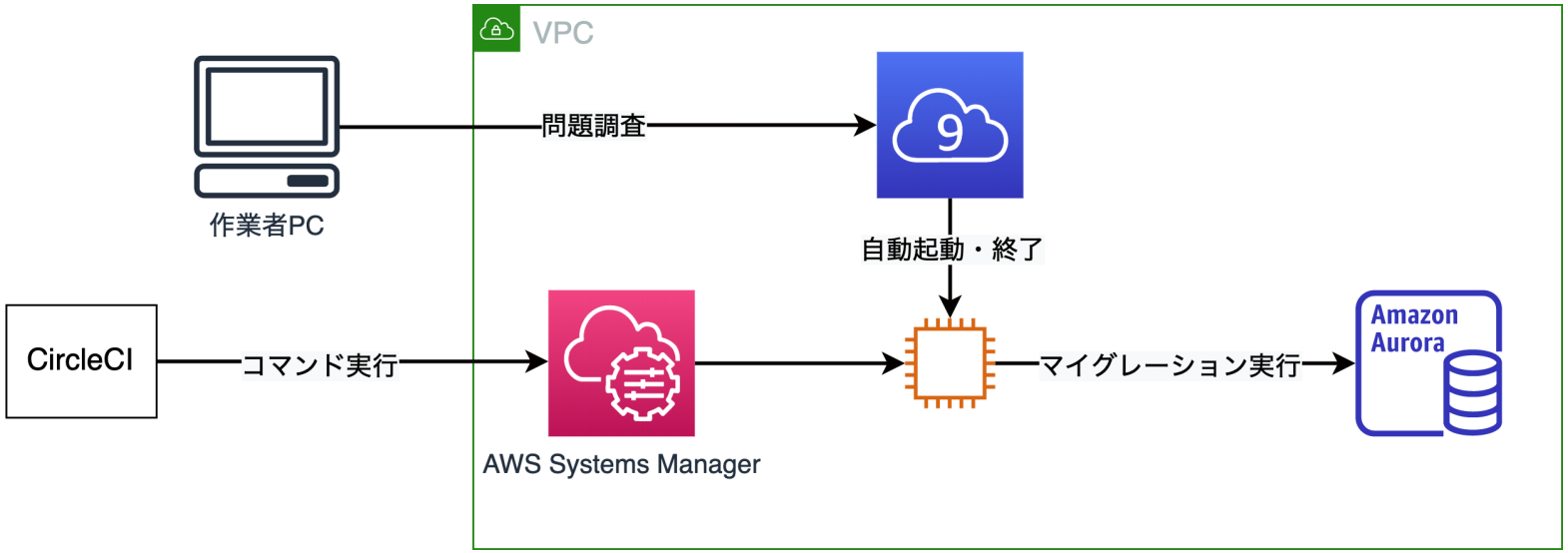
インスタンスクラス	max_connections デフォルト値
db.t2.small	45
db.t2.medium	90
db.t3.small	45
db.t3.medium	90
db.r3.large	1,000
db.r3.xlarge	2000
db.r3.2xlarge	3000
db.r3.4xlarge	4000

表は https://docs.aws.amazon.com/ja_jp/AmazonRDS/latest/AuroraUserGuide/AuroraMySQL.Managing.Performance.html より引用・一部抜粋

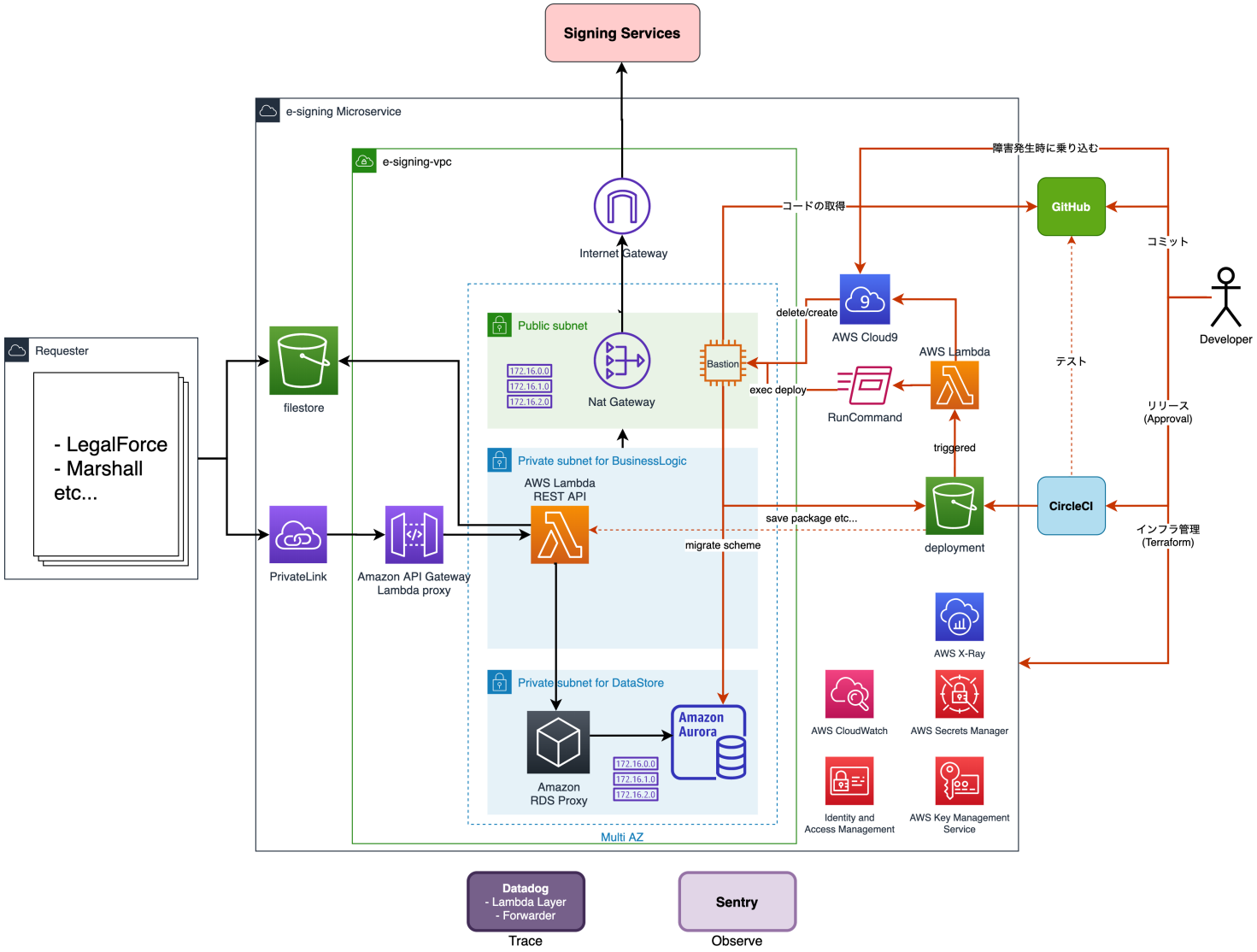
サーバレスアーキテクチャにおけるDBマイグレーション

- RDSを利用しているため、機能のリリースに伴いDBマイグレーションを行いたいケースがある
- CI/CDフローの中で行いたいが、本番DBへの接続が必要なためなるべく閉じた環境で実行したい

- AWS Systems Manager経由でコマンド発行し、AWS Cloud9から開発者が確認できる構成に



最終的なアーキテクチャ



Thank you!

時武 佑太

yuta.tokitake@legalforce.co.jp

aws SUMMIT ONLINE