New Relic

# サーバーレス化トレンドレポート

## 開発者、DevOpsチームのための AWS Lambdaベンチマークレポート

# 目次

# 要約（エクゼクティブサマリー）

## 2020年、サーバーレス化の始動、調整、オーバープロビジョニングの停止

サーバーレスにいま注目が集まっています。しかし、開発者やDevOpsチーム、意思決定者がサーバーレスの導入を検討するための重要な指標やベンチマークの定量的なデータは、まだ多くはありません。

**New Relic One**は、世界中で毎月数兆件にのぼるサーバーレスイベントを処理しています。このレポートでは、同じサンプルセットを時間の経過に合わせて集計・分析し、サーバーレスに取り組むNew Relicのユーザーが優れたデジタルサービス構築のためにアーキテクチャとパフォーマンスでどのような目標を置いているのかを学ぶことができます。

さらに、デベロッパーコミュニティーのサーバーレスの専門家にも、業界の現状と方向性についてインタビューを実施し、本レポート内に収めました。

2019年12月に開催された「AWS re:Invent」に参加した人やイベントの講演内容をフォローしている人なら、数百件ものサポートセッションとともに、Amazon Virtual Private Cloud（Amazon VPC）のコールドスタート対策、Provisioned Concurrency、AWS Lambdaなどサーバーレスに関する情報が次々と発表されているのを目にしたでしょう。

New Relic Oneでも、クラウドネイティブな企業だけでなく、従来型のエンタープライズ企業内でも、サーバーレスの導入が急増していることは確認できています。

Forrester Researchの報告書「Serverless Development Best Practices」で取り上げられた「2019年度 Global Business Technographics®デベロッパー調査」によると、49%の企業が1年以内にサーバーレスアーキテクチャを利用する、または導入する予定があるという結果が出ています。

「開発チームは、サーバーレスプラットフォームを使えば、新しいデジタルサービスや機能を数時間ですぐにテストできます。これまでのように数日もかける必要はありません。インフラストラクチャのプロビジョニングにかかる時間が最小限に短縮できるからです」とForresterは説明しています。

デジタルサービスを運用されている企業では、ピーク時のトラフィックを想定した可用性を持ったインフラストラクチャ環境を構築していることがほとんどです。こういった環境からサーバーレス環境で実行されるような柔軟なプログラミングモデルに移行することで、ピーク時のトラフィックを24時間365日サポートするのではなく、繁忙期に合わせて環境を自動スケールするように調整することができるようになります。

「サーバーレス」という単語には、GoogleやMicrosoftなどのさまざまなクラウドプロバイダーが提供するサービスが含まれていますが、今回のレポートではAWS Lambdaのデータを調査しました。

# 調査結果

- 企業におけるサーバーレスの導入は、過去12ヶ月間の週平均呼び出し回数が206%増加するなど、増加が続いています。本番環境でサーバーレスを採用している企業は、1アカウントあたりの関数が178%増加しています。
- 関数のボリュームに関しては、デベロッパーはLambda上にサーバーレスアプリケーションを構築する場合は、主にNode.jsとPythonを使用することが多く、3番目に使用頻度が高いランタイムはJavaです。しかし、AWSがコールドスタートの影響を緩和する**Provisioned Concurrency**や**VPCの改良**を行ったことで、隔離環境が必要な企業にとってLambdaに対する注目が高まり、2020年にはJavaが導入される傾向が高くなると予想されています。
- AWSのデプロイパッケージのサイズに制限があるため、関数のコードサイズが縮小化する傾向が続いていますが、これによって、全体的なコードサイズが小さく、明確に定義されたシングルタスクを実行する関数作成のベストプラクティスに対応できるようになっています。
- AWSからの**廃止発表**がされた後は、デベロッパーは、最新の言語バージョンへのアップデートを先延ばしにする傾向があります。かなりの数の関数がいまだにNode.js.6.10、Python 2.7、さらに古いバージョンでも動作しています。これらの関数はメンテナンスされていない可能性が高く、エラー率やコスト増加の原因になっています。

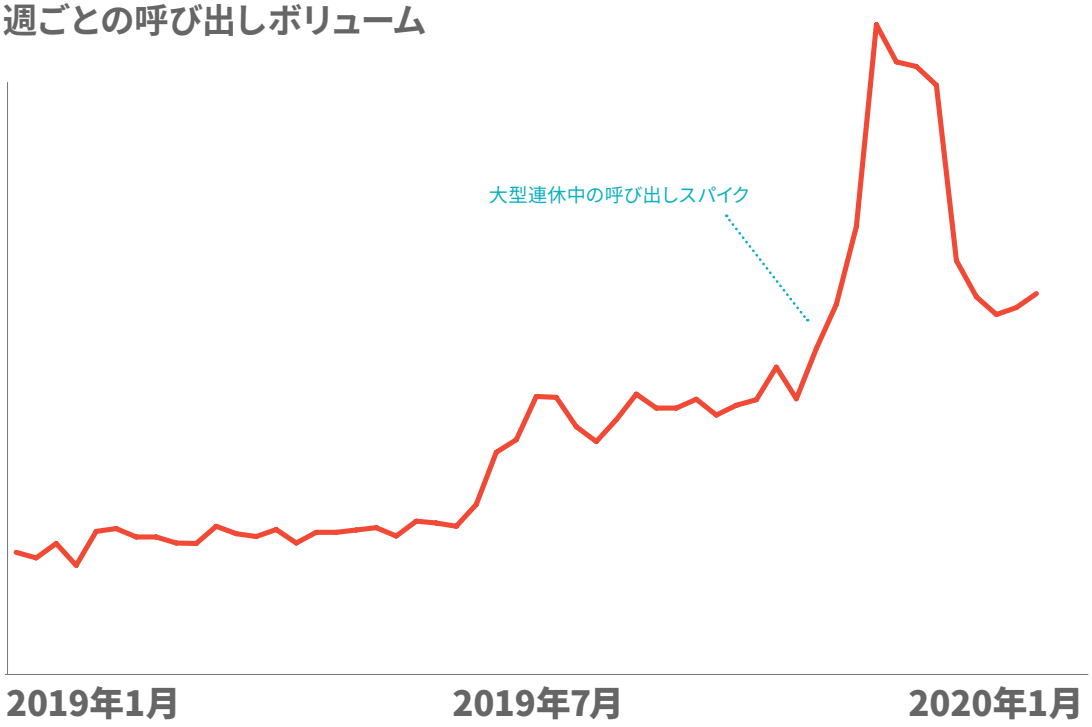




New Relic.

# サーバーレスの台頭

### 週ごとの呼び出しボリューム




## 考察

- サーバーレスの導入は、過去12ヵ月間の週平均呼び出し回数が209%増加するなど、増加が続いています。
- 大型連休中の呼び出しが急増することで、小売、マスコミ、物流などの業界における休日のピーク時のワークロードをサポートするため、自動スケーリングできるサーバーレスの使用が強化されています。

### 呼び出しの増加

# + 209%

**過去12ヵ月間のサンプルセット内の週平均の呼び出し量**





New Relic.

# FORREST BRAZEAL

A CLOUD GURUシニアマネージャー

### サーバーレスの導入において組織が直面する最大の課題とは？

クラウドやベンダーに依存しないアーキテクチャへの関心が高まっていますが、これは現実的な選択というよりも、イデオロギー的な選択であり、サーバーレスの導入を阻む最大の障壁になっています。

この課題の解決には、チームが新しい枠組みの価値に確信を持って構築するための適切なトレーニングを提供することが必要です。簡単な調査だけで止まってしまい、実務に取り掛かれないということはよくあることです。

### サーバーレスの導入における最大の技術的な課題とは？

2020年、サーバーレスが直面する最大の技術的課題は、分散型アーキテクチャやマイクロサービスベースのアーキテクチャ全般に共通します。レガシーシステムやコードベースを書き換えなければならないバックログ（未処理ログ）が膨大にあり、その解決のためのツールも、まだ十分に機能していません。

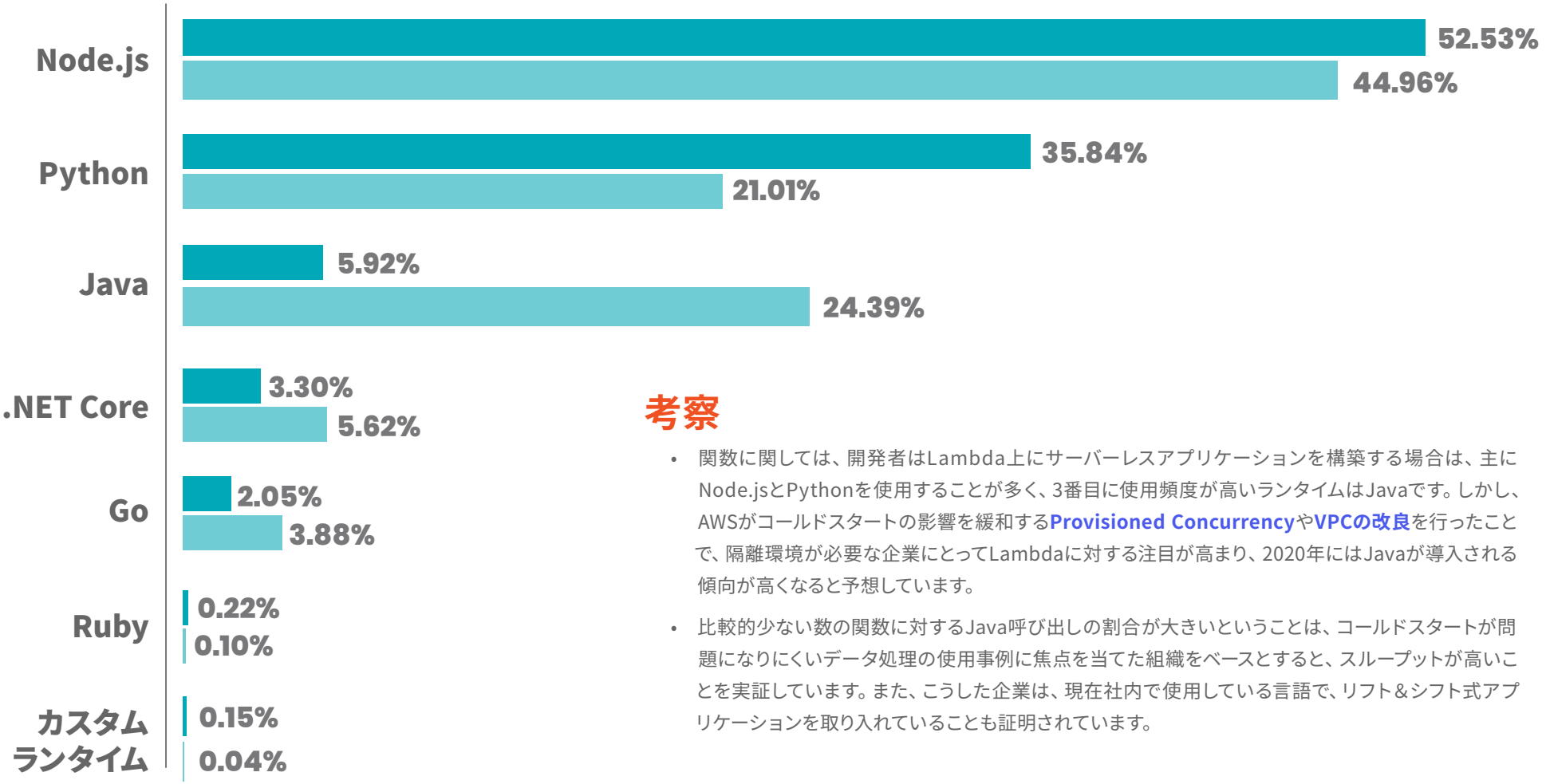
### サーバーレスはどこに向かうと思いますか？

FaaS（Function-as-a-Service）プラットフォームには「ステートフル」なオプションが追加され、AWS Fargateなどのマネージドコンテナによって、境界線がさらに曖昧になると考えます。また、AWS Amplifyなどの開発者にとって使いやすいサービスのベストプラクティスが抽象化されるのではないかと思います。さらに、サーバーレス導入に対する反対意見は、厳密には技術的なギャップではなく、新しいことへの挑戦を避ける惰性や特定ベンダーへの依存など、ビジネスや組織の問題が多く存在しています。

### 根強く残っているサーバーレスへの「誤ったイメージ」は何だと思いますか？

「サーバーレス化」は、主に技術的な判断と思われがちですが、実際には組織的な変革です。つまり、開発チームと運用チームが新たな役割を担い、経営陣が新たな価値を支持し、受け入れる必要があります。

# ランタイム別Lambda導入




## 考察

- 関数に関しては、開発者はLambda上にサーバーレスアプリケーションを構築する場合は、主にNode.jsとPythonを使用することが多く、3番目に使用頻度が高いランタイムはJavaです。しかし、AWSがコールドスタートの影響を緩和するProvisioned ConcurrencyやVPCの改良を行ったことで、隔離環境が必要な企業にとってLambdaに対する注目が高まり、2020年にはJavaが導入される傾向が高くなると予想しています。
- 比較的少ない数の関数に対するJava呼び出しの割合が大きいということは、コールドスタートが問題になりにくいデータ処理の使用事例に焦点を当てた組織をベースとすると、スループットが高いことを実証しています。また、こうした企業は、現在社内で使用している言語で、リフト＆シフト式アプリケーションを取り入れていることも証明されています。

モニタリングされている全**関数**の割合
モニタリングされている全**呼び出し**の割合

2019年7月-12月





New Relic.

# DAVE TOWNSEND

MATSON ソフトウェアエンジニア主任

### サーバーレスの導入において組織が直面する最大の課題とは？

ワークロードのサーバーレス化が進む2020年、組織が直面する課題の一つは、既存のスタッフへの研修とこの新しい枠組みへの適用でしょう。サーバーレス化は現在のアプリケーションの構築方法を根本的に変えてしまうことになります。着手は早ければ早いほど良いでしょう。アドバイスとしては、小規模なものから始め、何度も繰り返し、全員に関与してもらうことです。そして1つ言えることは「絶対に今、始めるべき」だということです。また、特に大手の組織では、経営陣を参加させることも重要なステップです。サーバーレスのメリットが分かるよう、魅力的なストーリーと裏付けとなるデータを用意しておいたほうが良いでしょう。

### サーバーレスの導入における最大の技術的な課題とは？

チームや組織がinfrastructure-as-code (IaC)を採用することは、確かに課題となるでしょう。インフラにあまり触れない従来型の開発を行ってきたチームは、いくつかの新しいスキルを身につけ、使ってもらう必要があります。サーバーレス環境では運営上のガバナンスが重要です。サーバーレスアプリを作成しているチームは、これまで作成を許可されていなかったかもしれないリソースを作成するために、より多くのアカウントへのアクセスが必要になります。また、サーバーレスアプリケーションの展開方法を理解することも課題です。もちろん、CI/CDの概念は今でも適用されますが、IaCですべてを行うわけではありません。いまはコードとインフラの分離がほとんどないため、慣れるのには時間がかかるかもしれません。サーバーレスはまだ初期の段階であり、コミュニティとともに進化を続けていると思います。まだまだ理想とのギャップはありますが、「re:Invent 2019」でのサーバーレスに関するリリースのように、AWSはさまざまな声に耳を傾け、ギャップは埋まりつつあります。特に、VPCでのLambdaの起動時間を向上したことで、企業の発展を妨げてきた課題の一つを解決することができるはずです。実はこの課題に言及していた人の多くは、サーバーレス化に取り掛かってもいない人々でした。この分野で成功を収めたいクラウドベンダーにとっては、こうした導入の課題を解消する機能向上が続くのは、間違いないでしょう。

### サーバーレスはどこに向かうと思いますか？

確実なことは言えませんが、実験を開始し、成功事例を共有する企業が増えているため、状況を見る限り、着実に伸びていると言えます。サーバーレス化は、既存の設備に少し手を加えるだけで切り替えられるものではないので、サーバーレスが爆発的に増えることはないかもしれません。サーバーレスというものは、これまでの常識とは全く異なるものなのです。童話の「うさぎとかめ」のように、ゆっくり、コツコツ進むことこそ、成功への近道だと考えています。

### サーバーレスの状況について、最も期待していることは何ですか？

現在はクラウドベンダー側から見ても、ツール側から見ても、素晴らしい状況です。サーバーレスのフレームワークやSAMにおける急激な進化、サーバーレス周辺の監視や可観測性の分野における進化を見ることが楽しみです。

AWSに限って言えば、AppSyncやAmplifyの取り組みは良いと思います。Amplifyとともに、**Architect**などのフレームワークは、既存の多くのフロントエンドの担当者の中からフルスタックデベロッパーという全く新しいカテゴリーを生み出せる大きな可能性を秘めています。これらのフレームワークによって、クラウドの経験がほとんどないフロントエンド開発者でも、サーバーレスのバックエンドを簡単かつ素早く立ち上げることができるようになりました。また、私は**EventBridge**にも大きな期待を寄せています。このサービスによって、イベントベースのサーバーレスアーキテクチャパターンへの扉が開くことでしょう。

# ランタイムバージョン別のLambda導入

### 呼び出しの割合別のランタイムバージョントップ10

2019年7月-12月

| | |
|---|---|
| Node.js 8.10 | 30.58% |
| Java 8 | 24.39% |
| Node.js 10.x | 8.99% |
| Python 3.6 | 8.38% |
| Python 3.7 | 6.67% |
| Python 2.7 | 5.96% |
| .NET Core 2.1 | 4.27% |
| Go 1.x | 3.88% |
| Node.js 6.10 | 3.07% |
| Node.js 4.3 | 2.19% |

### 関数の割合に対するトップ10のランタイムバージョン

2019年7月-12月

| | |
|---|---|
| Node.js 8.10 | 32.42% |
| Python 3.6 | 13.45% |
| Python 2.7 | 12.32% |
| Node.js 10.x | 11.27% |
| Python 3.7 | 9.85% |
| Java 8 | 5.89% |
| Node.js 6.10 | 3.69% |
| .NET Core 2.1 | 2.77% |
| Go 1.x | 2.05% |
| Node.js 4.3 | 1.40% |

## 考察

- Lambdaでは、Node.js 8.10やPython 2.7のバージョンがまだ主流です。しかし、Node.js 8.10の廃止（推奨からの除外）が予定されているため、Node.js 10.x.への移行が進むと予想できます。一方、Python2.7はサポートが終了しましたが、AWSは2020年中はサポートを継続する予定です。
- AWSからの**廃止発表**がされた後は、開発者は最新の言語バージョンへのアップデートを先延ばしにする傾向があります。かなりの数の関数がいまだにNode.js.6.10、Python 2.7、さらに古いバージョンでも動作しています。これらの関数はメンテナンスされていない可能性が高く、エラー率やコスト増加の原因になっています。

# FARRAH CAMPBELL

STACKERY社 エコシステムディレクター

### サーバーレスの導入において組織が直面する最大の課題とは？

プロジェクトの主要なアーキテクチャソリューションとして、どのようにサーバーレスを導入するかが第一の課題です。1〜2組のチームがサーバーレスで大成功をおさめた組織は多いものの、その成功を広い範囲で再現できた例はまだほとんどありません。こうした組織では、知識や経験を共有する優秀な人材が数多く抱えていますが、他のアーキテクチャパターンのアプリケーションを構築するチームをサポートするのと同じように、中心となるチームのサポートをするためのツールが必要になります。

### サーバーレスの導入における最大の技術的な課題とは？

この数年間で、「ウォーム」関数のプロビジョニングや仮想ネットワークに接続した場合のスケーラビリティの改善など、サーバーレスプラットフォームの課題の多くは解消されています。サーバーレスがほぼすべてのユースケースに対応できるようになった今、最大の技術的課題は、幅広い開発者がサーバーレスを導入できるように、適切なワークフローとツールの開発を続けることでしょう。

### サーバーレスはどこに向かうと思いますか？

大規模な組織による導入が増えることで、サーバーレスの限界に対する固定観念が覆され、普及が進むと思います。コールドスタートの問題を懸念して、これまではサーバーレスを避けていたJavaや.NETを使う環境でも、サーバーレスを運用するためのツールの導入・構築を開始するようになるでしょう。そして従来のサーバー運用への投資に疑問を抱く組織も増えると思います。

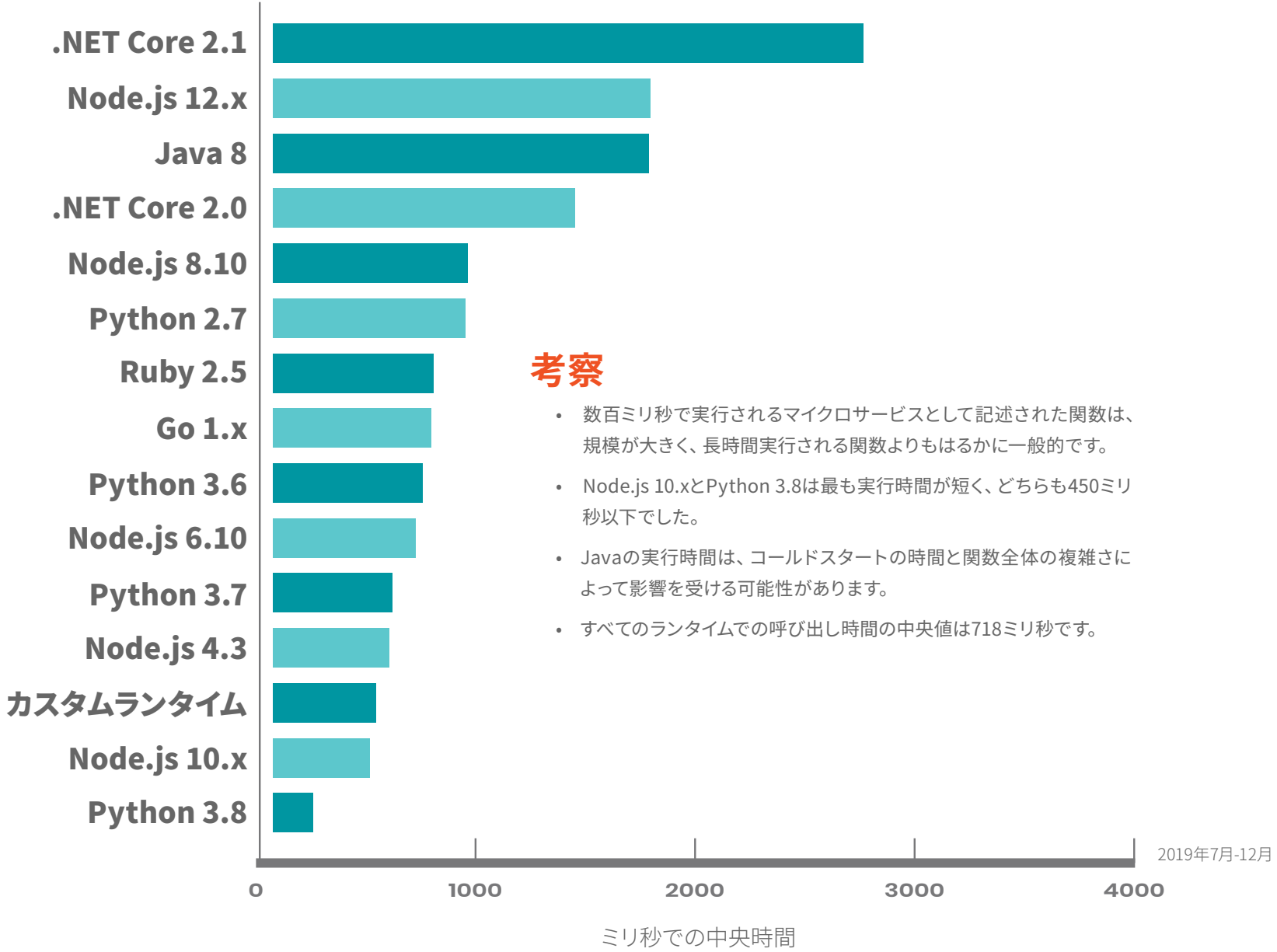
### サーバーレスの状況について、最も期待していることは何ですか？

従来、スケーラブルなウェブアプリケーションの構築は非常に難易度の高い作業でした。非常に狭く、深い専門知識が求められるため、この種のソフトウェア開発ができる人の数にも限りがあります。一方、サーバーレスは参入障壁が低く、さまざまな開発者がスケーラブルで柔軟なアプリケーションの構築を望んでいます。この影響が今後も続くと面白いと思います。

### 根強く残っているサーバーレスへの「誤ったイメージ」は何だと思いますか？

最大の誤った認識は、「サーバーレス=関数」であるということです。関数という固定概念が捨てきれず、限界やコストばかり分析している人が多いようです。こうした人々は、インフラストラクチャと運用管理が簡素化されるというサーバーレスのメリットを見落としています。少なくとも、ほとんどのプロジェクトではサーバーレスをソリューションとして優先するべきで、代替案は必要な場合にのみ検討するべきです。そう言い切ってしまえるほど、サーバーレスのROI（投資対効果）が非常に高いと言えるでしょう。

# ランタイムバージョン別の呼び出し時間




## 考察

- 数百ミリ秒で実行されるマイクロサービスとして記述された関数は、規模が大きく、長時間実行される関数よりもはるかに一般的です。
- Node.js 10.xとPython 3.8は最も実行時間が短く、どちらも450ミリ秒以下でした。
- Javaの実行時間は、コールドスタートの時間と関数全体の複雑さによって影響を受ける可能性があります。
- すべてのランタイムでの呼び出し時間の中央値は718ミリ秒です。





New Relic.

# BRIAN LEROUX

BEGIN社 CTOおよび共同創業者

### サーバーレスの導入において組織が直面する最大の課題とは？

「サーバーレス」とは、大規模であることが前提のムーブメントで、クラウドを運用するために必要なものです。クラウドベンダーが成長するにつれて、インフラストラクチャの購入や構築が増え、その過程で総所有コストが削減されます。クラウドベンダーが提供するサービスは、コモディティ化しています。最初はケージやラックのレンタルから始まり、仮想化に移行し、最終的には100ミリ秒単位で実行する関数をレンタルできるようになりました。サーバーレスの定義が拡大し、IaC（Infrastructure as Code）やオブザーバビリティ（可観測性）などの概念が含まれるようになりましたが、核となる理念は変わりません。つまり、差別化されていないコモディティ化されたワークロードをアウトソーシングし、コアビジネスの価値に集中するという理念です。

当然、多くの組織が、さまざまなクラウドベンダーのサーバーレス提供を評価しています。ただし、「サーバーレス」というラベルが貼られた機能は多いものの、ベンダーによってその内容は大きく異なることは意外と知られていません。「サーバーレス」という言葉自体に、期待を膨らませるのは良いことですが、次のステップには幻滅という落とし穴が待ち受けています。私が考える2020年の課題は、今、私たちがその「落とし穴の中」にいるということです。

あらゆるクラウドベンダーには、それぞれに優位性があります（それが他社を大きく凌ぐ企業もある）。酷いことを言うつもりはありませんが、トップ2社のクラウドベンダーだけを見ても、その違いはかなり大きいのです。未来がサーバーレス化するのは間違いありませんが、いまはまだまだ初期の段階です。

注意深く、計画的かつ意図的に、従来のモノリシックなアプリケーションを移植することは可能ですが、最初から完全なサーバーレス化としてスタートする方がずっと簡単です。それでも、そのプロセスは険しく、簡単とは言えません。**Begin.com**の目標は、ストレスフリーなAWSサーバーレスを実現することです。

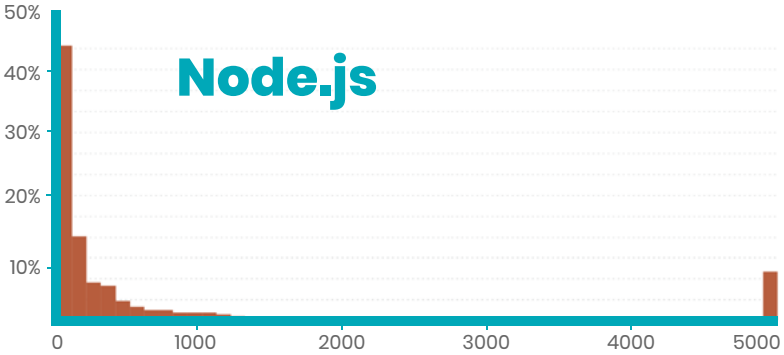
### サーバーレスの状況について、最も期待していることは何ですか？

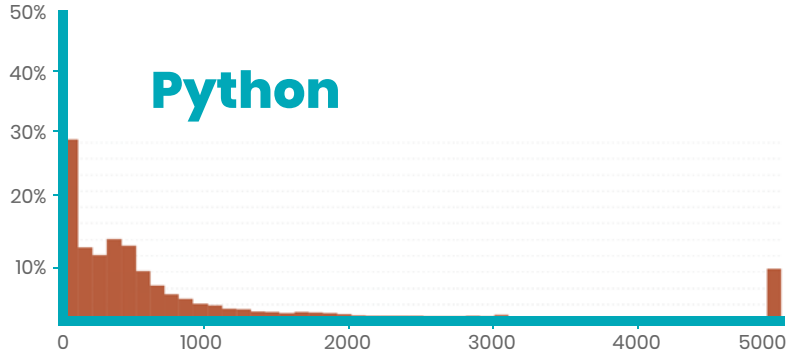
「**Deno**」という新しい実験的なJavaScriptランタイムには非常に期待しています。サーバーレスのウェブ開発に適した特徴を備えているからです。これはNodeのクリエーターが手を加えて作ったもので、Nodeと同様、V8をベースにしており、優れたコールドスタートを実現します。Denoは、一般的なTypeScriptやJSXの方言に加えて、ESモジュールをネイティブでサポートしています。これは、サードパーティ製のビルドツールを使わずに、React（またはPreact）とTypeScriptのいずれかのサーバーレスサイドレンダリング（SSR）をLambdaで実行できることを意味しています。非常にスマートで、楽しくウェブアプリを「サーバーレス」で構築する方法になると思います。

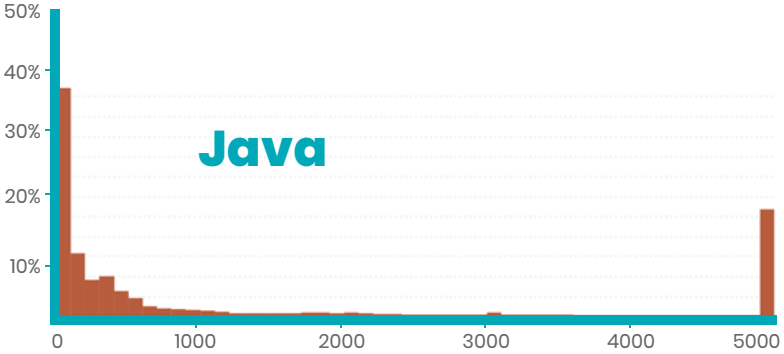
### 根強く残っているサーバーレスへの「誤ったイメージ」は何だと思いますか？

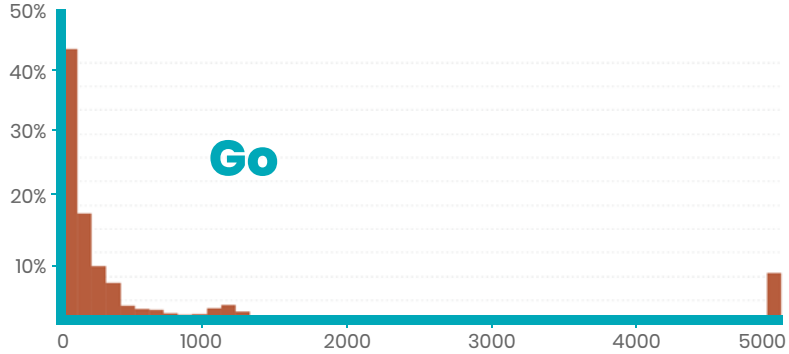
ベンダーロックインに関する考え方です。Architect、または未加工のSAM/CloudFormation、あるいはサーバーレスサービス（API、Lambda、DynamoDBのようなもの）だけを使ってAWSでアプリを構築できれば、実際には競合他社よりも有利になる可能性が非常に高いです。

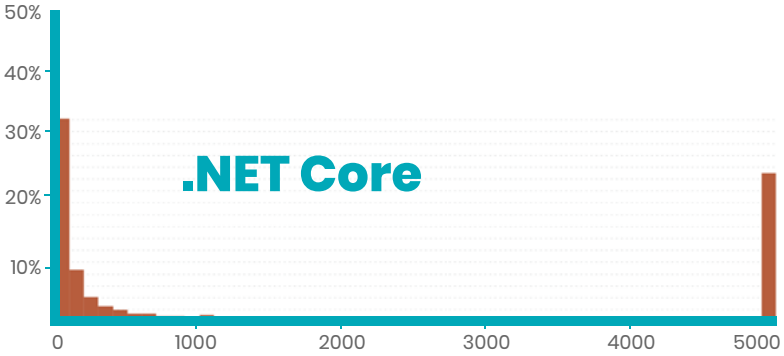
# ランタイム別の持続時間ヒストグラム




x-axis – 平均関数時間 (ms)
y-axis – 関数の割合

## 考察

- Javaでも、呼び出しの大半は500ミリ秒以下です。コールドスタートの影響を軽減するAWSの新しいProvisioned Concurrency機能が利用可能になったことで、LambdaでのJavaはクリティカルなワークロードに適した選択肢になりました。
- ランタイム全体で3秒間という小さなスパイクは、デベロッパーがデフォルトのLambdaタイムアウト制限を調整していないことを示しています。一方、右端の5秒間のスパイクは、タイムアウトのベストプラクティスであると同時に、5000ミリ秒以上実行する関数のロングテールです。

# ALEX CASALBONI

SERVERLESSDAYS MILAN
テクニカルエバンジェリスト、AWS共同創業者

### サーバーレスの導入において組織が直面する最大の課題とは？

組織の課題は、主に社風（企業文化）やテクノロジーの変化に関連するものになると思います。5年以上経った今、テクノロジーは成熟しており、もはやツールやUXのせいにすることはできません。もし、小規模な組織と大規模な組織の両方の組織的な課題の克服に役立つアプローチを2つ提案するとしたら、私はトレーニングに投資し、少人数のチーム（小規模なサービスの構築を担当）に重点を置くことを選びます。トレーニングは、ちょっとしたMeet upに参加したり、定期的にアップされるブログを読むだけでも十分です。小規模なチーム（Amazonでは「**two-pizza team**」と呼ばれる）は、従来の組織では考えにくい形ですが、すべてのチームメンバーの意識と生産性の両方に影響を与える可能性が非常に高いのです。

### サーバーレスの導入における最大の技術的な課題とは？

ほとんどの技術的な課題は、この2〜3年で解決されたと思います。2016年にサーバーレスを導入しましたが、当時の障壁であった要因の多くは解消されています。エッジケース（極端なユースケース）を可能にし、一般的な状況をシンプルにするためにも、技術的な改良は今後のロードマップに含まれることになるでしょう。しかし、この12ヵ月間を振り返ってみると、この分野では多くのイノベーション（**Amazon EventBridge**、**Amazon RDS Proxy**、**Provisioned Concurrency for AWS Lambda**、**Custom Runtimes**、**Data API for Amazon Aurora Serverless**、**AWS CDK**など）が生まれています。2020年は、これらのアーキテクチャの新しい設計方法をすべて組み合わせることで、デベロッパーの作業をシンプルにできると思います。

### サーバーレスはどこに向かうと思いますか？

一つパターンを挙げるなら「コードを書く量を減らす」傾向にあることです。多くのデベロッパーが自分の主な仕事はコードを書くことだと思い込んでいます。40年前は「開発者」が考える自分の仕事は、カードに穴を開けることでした。開発者の仕事は、問題を解決し、お客様に満足していただくことです。今では、マネージドサービスの統合やサードパーティのAPIを介して、コードを書く量を減らしても、同じことをはるかに速く実行できるようになりました。このパターンは数年前から始まっており、「サーバーレス」の代わりに「サービスフル」という言葉も生まれています。2020年は、これが新しい常識になるかもしれません。

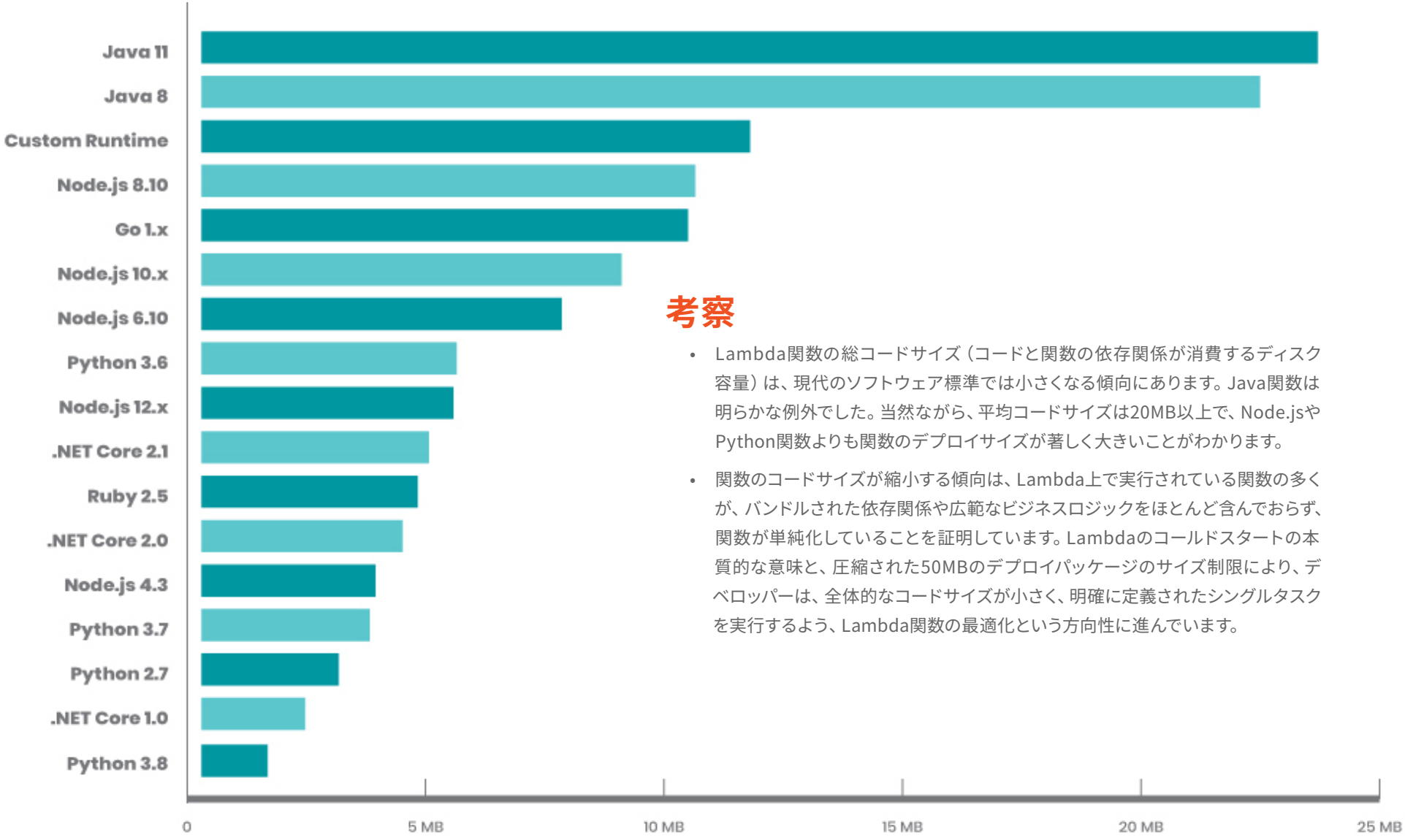
### サーバーレスの状況について、最も期待していることは何ですか？

少なくとも半年間は「Lambda上で動作しない」というクレームは、聞いていません。いち早く着手し、メリットを体験するための第一歩として、「リフト＆シフト」方式でサーバーレス化へ移行する機会も増えています。

### 根強く残っているサーバーレスへの「誤ったイメージ」は何だと思いますか？

パフォーマンスが悪いという思い込みでしょう。組み込み型の統合（カスタムポーリングや回避策ではなく）が増えており、内部で継続的にパフォーマンスが向上していることから、「サーバーレスは遅い」という誤ったイメージは再考されるべきだと思います。特に、2019年の2つの大きな改善項目（**VPC**のコールドスタートの解消、Provisioned Concurrency（プロビジョニングされた同時実行性）の導入）は考慮するべきでしょう。この二つを組み合わせることで、遅延の影響を受けやすいアプリケーションに関連するほとんどの問題を解決することができます。

# 遅延の影響を受けやすいアプリケーションは




2019年7月-12月

## 考察

- Lambda関数の総コードサイズ（コードと関数の依存関係が消費するディスク容量）は、現代のソフトウェア標準では小さくなる傾向にあります。Java関数は明らかな例外でした。当然ながら、平均コードサイズは20MB以上で、Node.jsやPython関数よりも関数のデプロイサイズが著しく大きいことがわかります。
- 関数のコードサイズが縮小する傾向は、Lambda上で実行されている関数の多くが、バンドルされた依存関係や広範なビジネスロジックをほとんど含んでおらず、関数が単純化していることを証明しています。Lambdaのコールドスタートの本質的な意味と、圧縮された50MBのデプロイパッケージのサイズ制限により、デベロッパーは、全体的なコードサイズが小さく、明確に定義されたシングルタスクを実行するよう、Lambda関数の最適化という方向性に進んでいます。

# ERICA WINDISCH

NEW RELIC社 エンジニア主任

**サーバーレスの導入において組織が直面する最大の課題とは？**

サーバーレスに着手したばかりであれば、サーバーレスアプリケーションのインベントリや支出の把握を含め、CI/CDやリリースプロセスと格闘している人も多いかもしれません。経験豊富なサーバーレス組織のデベロッパーは、複雑で強力なアプリケーションを構築し、規模を拡大してワークロードを運用するという課題に直面することになると思います。

**サーバーレスはどこに向かうと思いますか？**

数年前まで、サーバーレスコンピューティングといえば、AWSがほとんどでしたが、CloudFlare WorkersやTwilio Functionsが台頭してきているようです。AWSではなく、Lambdaで優位性を出そうとするサービスよりも、IoT、モバイル、電気通信をターゲットとする、差別化された製品で、サーバーレス市場を拡大する企業の方が増えるのではないかと思います。

**根強く残っているサーバーレスへの「誤ったイメージ」は何だと思いますか？**

サーバーレスプロジェクトの運用コストの予測は今も難しく、コストがかかりすぎるというイメージがまだ残っていると思います。「servers.lol」などのツールは、こうした不安を払拭し、プロジェクトを企画する上での正確な予測を立てるのに役立ちます。

# JEREMY DALY 「SERVERLESS CHATS」ホスト

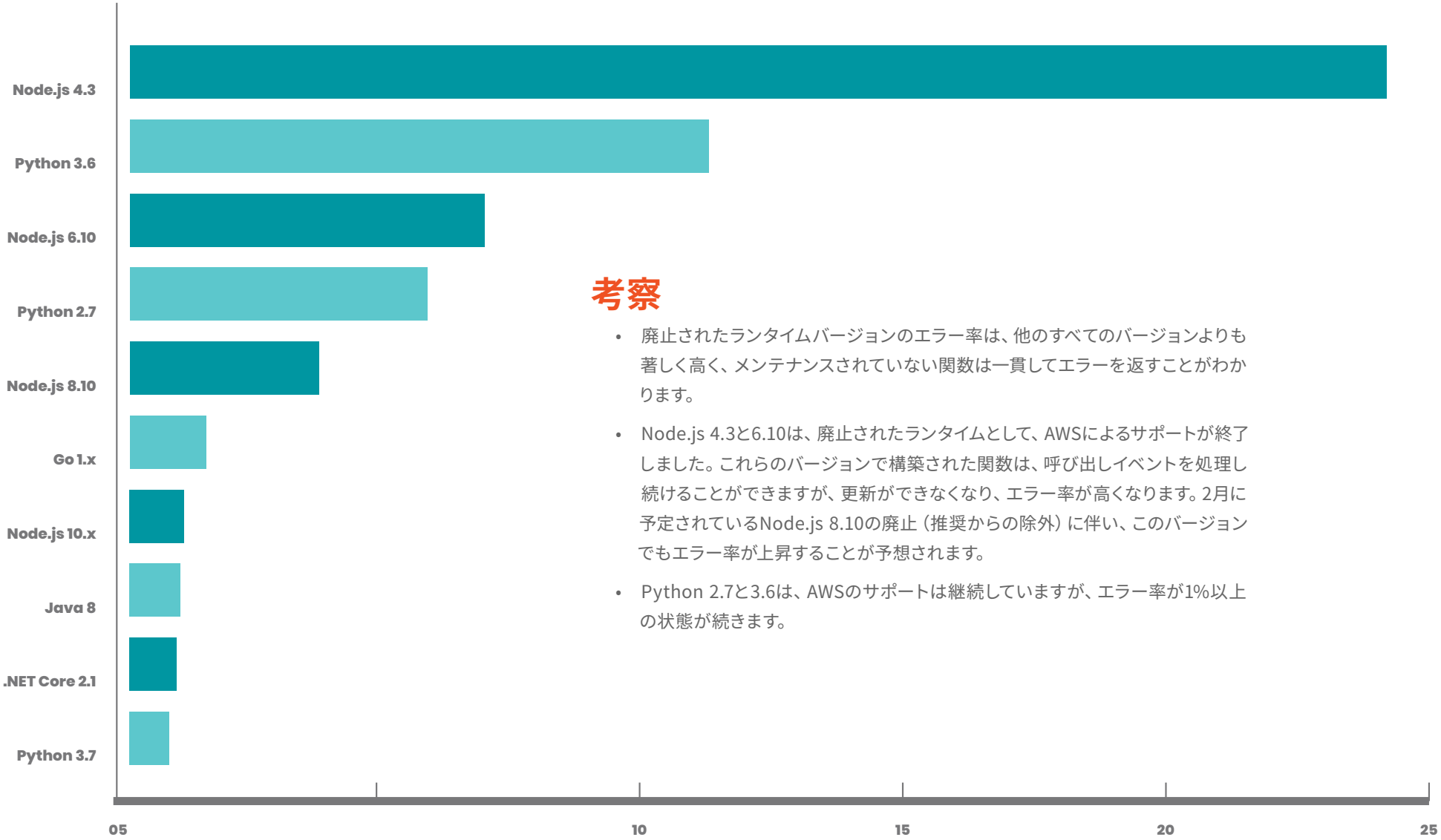
### サーバーレスはどこに向かうと思いますか？

サーバーレスは、簡素化するよりも複雑になっていくと思います。シンプルな機能を立ち上げ、実行するという基礎は非常に簡単ですが、複数のマネージドサービスで通信する複雑なアプリケーションを構築するのはまったく別の話です。インフラストラクチャの管理を抽象化しても、デベロッパーが適切なユースケースや制限を理解する必要がなくなるわけではありません。そのためには、マネージドサービスがどのように機能するか、どのようにパフォーマンスを最適化するか、そして最も重要なこととして、分散システムで避けられない障害に対処する方法について、理解を深める必要があります。このような複雑さは必要なものかもしれませんが、私が「Abstraction as a Service（サービスとしての抽象化）」と呼ぶ取り組みを行う企業が増えると思います。これは、ベストプラクティスやコンプライアンス要件を高レベルなコンポーネントにカプセル化し、コミュニティやチームメンバーが再利用できるようにするものです。

### サーバーレスの状況について、最も期待していることは何ですか？

サーバーレスは、膨大な数のユースケースを網羅できるポイントまで進化していると思います。AWS Data Proxy、Provisioned Concurrency、Lambda Destinationsなどのツールが最近追加されたことで、サーバーレスに対して抵抗がある人からの反対意見は減りました。多くの企業が新しい機能の構築や既存の機能の移行のために、「サーバーレスファースト」ソリューションを検討し始めています。

# ランタイム別エラー率




2019年7月-12月

## 考察

- 廃止されたランタイムバージョンのエラー率は、他のすべてのバージョンよりも著しく高く、メンテナンスされていない関数は一貫してエラーを返すことがわかります。
- Node.js 4.3と6.10は、廃止されたランタイムとして、AWSによるサポートが終了しました。これらのバージョンで構築された関数は、呼び出しイベントを処理し続けることができますが、更新ができなくなり、エラー率が高くなります。2月に予定されているNode.js 8.10の廃止（推奨からの除外）に伴い、このバージョンでもエラー率が上昇することが予想されます。
- Python 2.7と3.6は、AWSのサポートは継続していますが、エラー率が1%以上の状態が続きます。





New Relic.

# ランタイム別タイムアウト

| | |
|---|---|
| Python 3.8 | 5.40% |
| Ruby 2.5 | 3.36% |
| Node.js 12.x | 3.09% |
| Python 2.7 | 2.96% |
| カスタムランタイム | 2.95% |
| Node.js 6.10 | 2.32% |
| .NET Core 1.0 | 2.00% |
| Python 3.6 | 1.93% |
| Node.js 4.3 | 1.44% |
| Python 3.7 | 1.41% |
| Node.js 8.10 | 0.94% |
| .NET Core 2.1 | 0.85% |
| Go 1.x | 0.78% |
| Node.js 10.x | 0.77% |
| .NET Core 2.0 | 0.42% |
| Java 8 | 0.36% |

タイムフレーム 2019年7月-12月

## 考察

- Javaは、コールドスタートが発生しない大容量データプロセスのワークロードに頻繁に使用されています。つまり、高度に調整されており、タイムアウトになることはほとんどありません。
- Node言語やPython言語は、APIなどのCRUD操作になる可能性が高く、サードパーティのサービスやデータベースへの接続が増えるとタイムアウトが発生しやすくなります。
- タイムアウトエラーは、デフォルトのタイムアウトリミットを調整していないなど、さまざまな原因で発生する可能性があります。デフォルトの実行時間のリミットである3秒は顧客に対応するイベント駆動型の機能には最適ですが、機能が複雑になると、さらに実行時間が長くなる場合があります。タイムアウトとランタイムバージョンの間に意味のある相関関係を作るには、ユーザー変数や要件が多すぎます。

# SHEEN BRISALS

LEGO GROUP、シニアエンジニアマネージャー

### サーバーレスの導入において組織が直面する最大の課題とは？

サーバーレスを単なるLambda関数の集合体と見なすのではなく、マネージドサービスのエコシステムと見なすようにするため、組織的な意識改革を進めることです。

オンプレミス環境やホスト環境とサーバーレス環境とのコスト比較を説得力のある方法で上層部に提示するのは、難題（あるいは不可能）である場合がほとんどです。このように直接的な比較を行わなければ、サーバーレスへの移行や管理者によるサーバーレスの承認が遅れることは珍しいことではありません。

ほとんどの場合、サーバーレスサービスは間接的な支出などコストを削減します。ただし、監視やオブザーバビリティ（可観測性）などのサポートソリューションを追加すると、コストがさらに複雑化する可能性があります。たとえば、可観測性ツールの価格モデル（イベントやデータポイントに基づいた）は、サーバーレスのエコシステムが拡大すると、価格が高騰するおそれがあります。サーバーレスへの移行によってコスト削減しても、こうしたソリューションを購入することで、結局差し引きゼロになってしまいます。これがサーバーレス反対派になるきっかけになります。

最後に「ベンダーロックイン」という言葉も反対される原因になるでしょう。

### サーバーレスの導入における最大の技術的な課題とは？

ツール作り。少しずつ改善されてはいますが、まだ多くの混乱や不整合が残っています。

オブザーバビリティ（可観測性）。小規模な新興企業の観点から見ると、可観測性を実現するためのソリューションは（私の意見では）高価だと感じることがあるかもしれません。

同様の機能を持ち、似たようなサービスが数多く混在することで、エンジニアやいち早く導入したユーザーの間に混乱が起こります。こうした状況によって、勢いが鈍ります。

また、通常、従来の開発環境からクラウドとサーバーレスに移行するチームは、サーバーレスの導入に必要な完璧なソリューション、プラットフォーム、運用環境を求めます。これにより、想定外の遅延と摩擦が生じます。

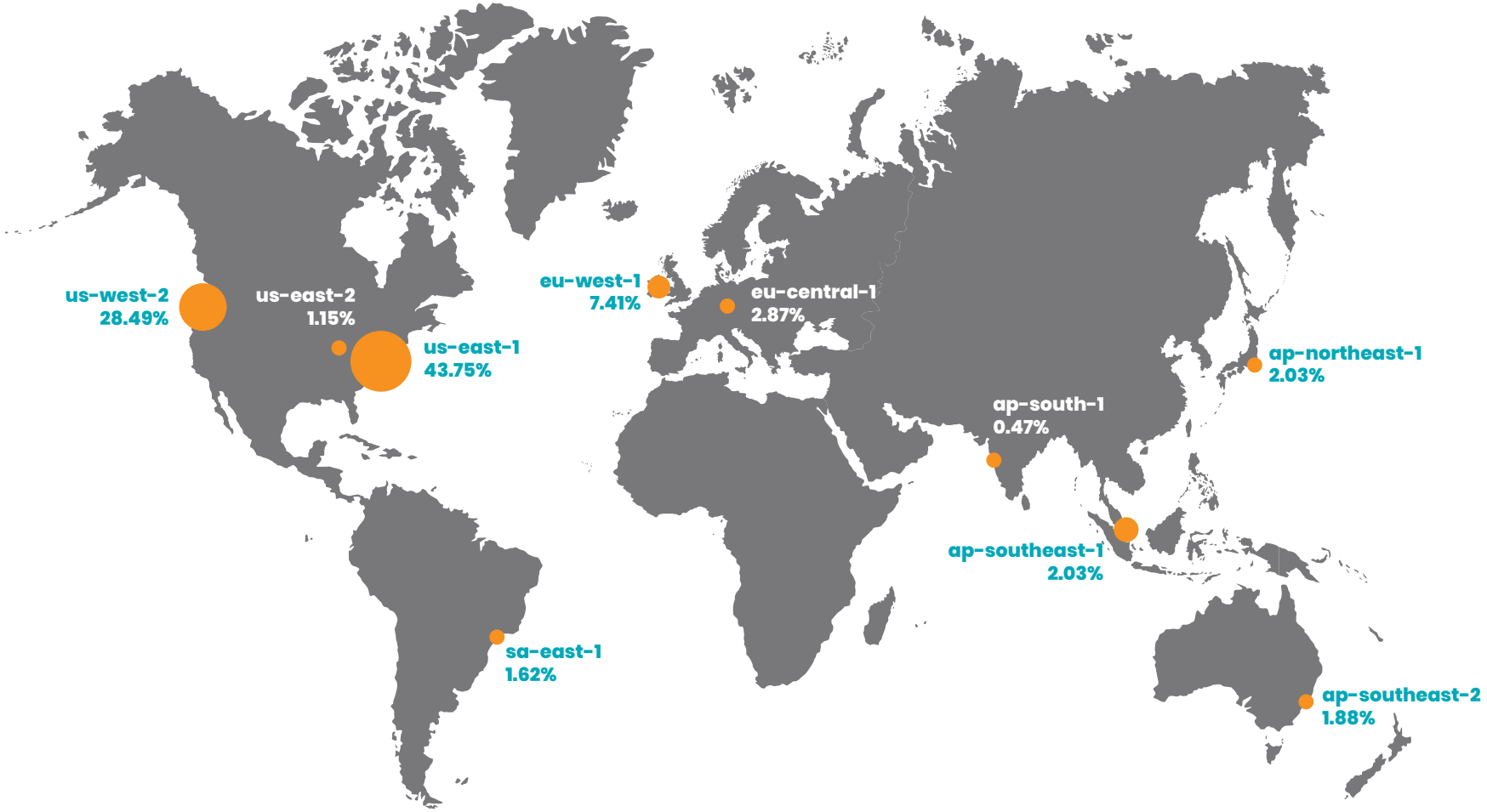
### サーバーレスはどこに向かうと思いますか？

初心者にとっては、関数が少ない、あるいはコードが少ないなど、関数が少ない形態が主流になっていくと思います。イベント駆動型になり、何にでも関数を追加する必要性がなくなるでしょう。

また、正当な理由で業界での導入が増えるのではないかと思います。AWSは、従来のサーバーリソースのアクセス性とフェンスのサーバーレス側からサービスの間の障壁を徐々に取り払いつつあります。つまり、サーバーレス以外のサービスにアクセスするためだけに、コンテナベースのソリューションを選択する必要はありません。まだ完全には到達していませんが、順調に進んでいます。

そして、AWS EventBridgeが中心となって、カスタムサービスとAWSサービス、そして認定パートナーソリューションの間で、関数を使わないイベント駆動型構築を調整できるのではないかと期待しています。EventBridgeにさらに機能が追加されると期待しています。

# 地域別の呼び出し割合




## 考察

- Lambdaは、世界中のほとんどのAWS対応地域で公開されているため、ヨーロッパ、北米、アジアで稼働しているのを見ても驚きはありません。
- 当社のサンプルデータでは、米国の地域が呼び出しの大部分を占めています。米国西部（オレゴン州）と米国東部（バージニア州）が監視対象の呼び出しの70%以上を占めています。

2019年7月-12月

# 地域別エラー率


## 考察

- ヨーロッパ（アイルランド）の対応地域では、エラー率が6.81%と最も高いのは、ステージングにおけるLamba関数の割合が高く、特に呼び出しの割合が他の地域に比べて少ないことに起因していると考えられます。
- 次にエラー率が高かったのは、アジア太平洋地域（シドニー）とアジア太平洋地域（シンガポール）で、5%を超えていました。

2019年7月-12月

# AWS Lambdaの関数高速処理



サーバーレスの導入率と使用事例の複雑さが高まるにつれて、サーバーレスアーキテクチャ上で大規模なアプリケーションを構築するデベロッパーにとって、オブザーバビリティ（可観測性）は非常に重要な要素となっています。

**New Relic Serverless**を使って、すべてのLambda関数を一括で監視、可視化、トラブルシューティング、アラートを発信する方法を紹介します。ぜひNew Relic Serverless for AWS Lambdaを無料でスタートしてみましょう。





New Relic