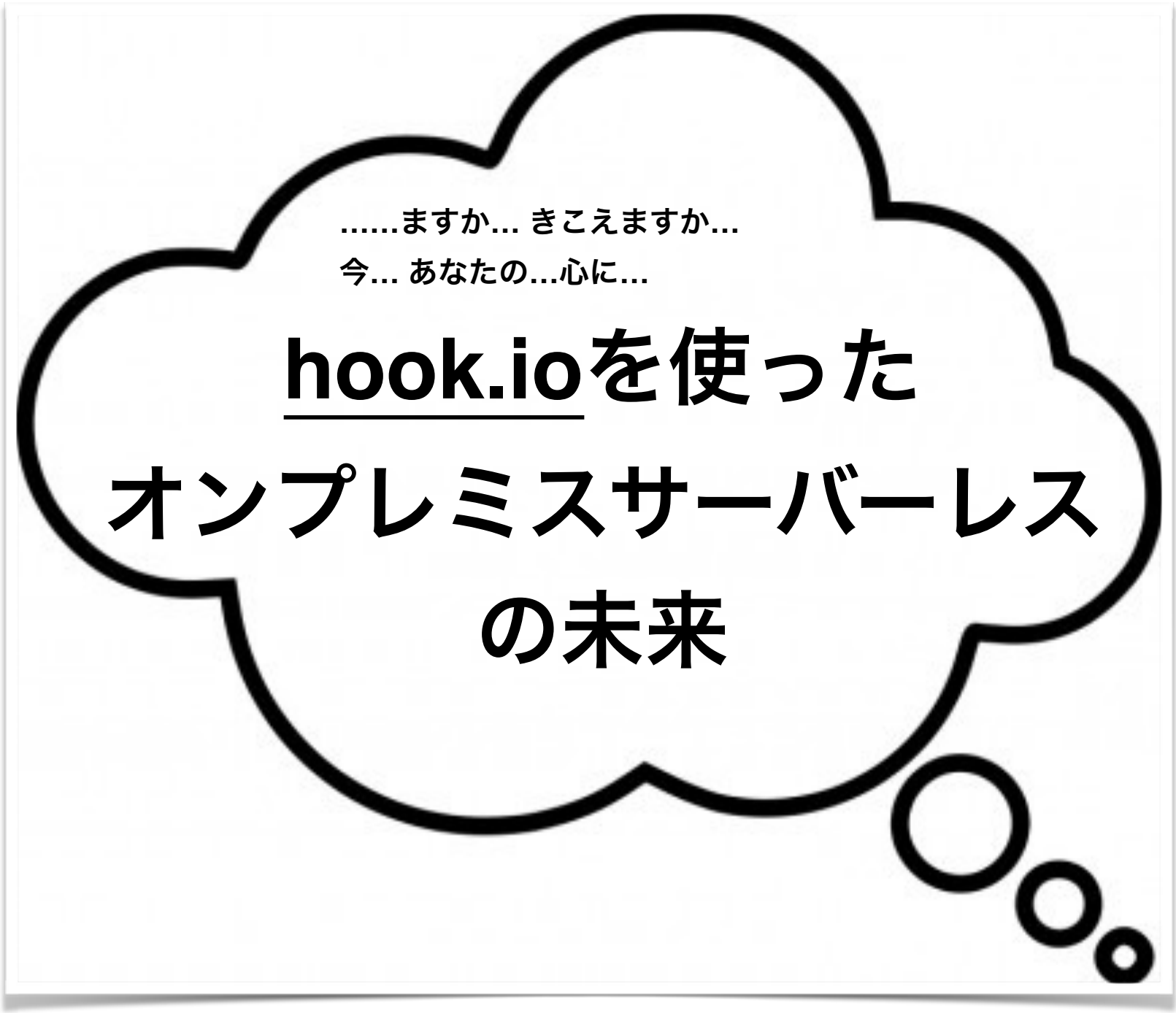


hook.io contributor

今日の登壇者

川上礼次

LEAP MOTION Developers JP

エンジニア歴 2〜5年

CBR250RRN/攻殻機動隊/ライ麦畑/桶川スポーツランド/筑波サーキット/Perception Neuron/LeapMotion/RealSense/Oculus/UE/Unity

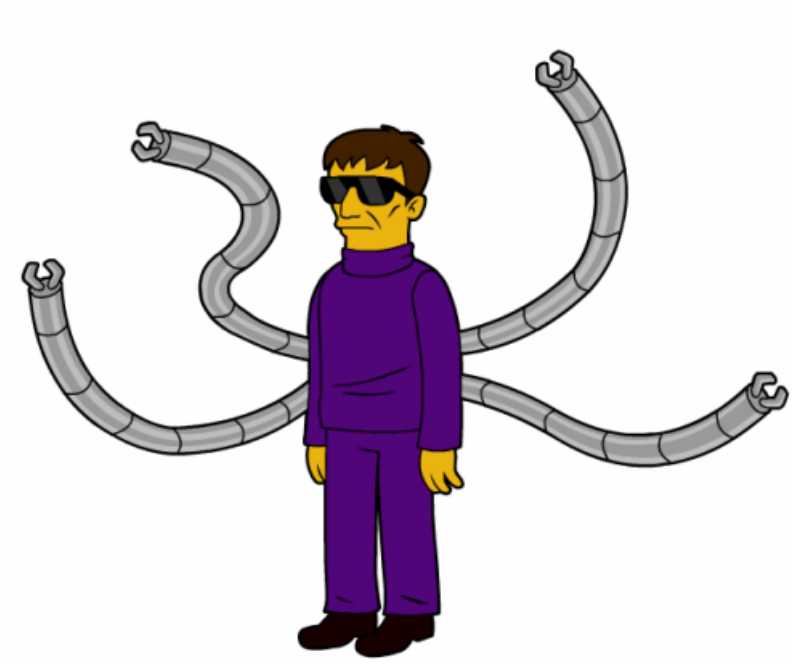
スマピラ

https://github.com/smapira

雑多なエンジニア歴 1x年

完全無欠コーヒーはじめました

1. サーバーレスとはなにか？



サーバレスといえば

FaaS （Function as a Service）

Q: どのようなサービス?

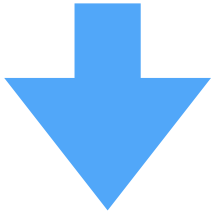
サーバーのリソースの用意や管理なしでコードを実行できる。コードさえアップロードすれば、高可用性を実現しながらコードを実行およびスケーリングが自動で行われる。

Q: サーバーレスコンピューティングとは何ですか?

アプリケーションとサービスを構築して実行する際に、サーバーについて検討する必要がなくなる。サーバーでアプリケーションを実行しながらも、サーバーの管理はすべてサービスによって行われます。

~~サーバー管理やシステム管理を不要なものにする、~~

プログラム実行環境



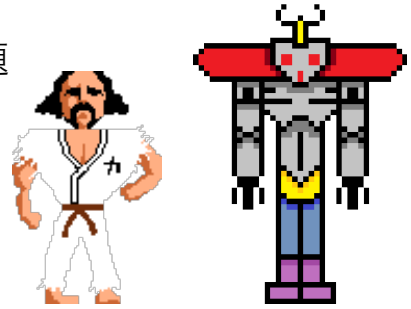
今日の講演では、サーバーレスアーキテクチャーを、REPL(Read-eval-print loop) 風な、マイクロサービスを発展させた形のアーキテクチャーとして捉える。

目新しいとか革新的というつもりはなくて、少なくともUnixの設計思想を根本に考えたいです。

2. ソフトウェアアーキテクトのパラダイム

2003年	Unixのコミュニティで「モノリス」という単語が使われ始める*1
2011年	ベネチア近郊で行われたソフトウェアアーキテクトのワークショップから「マイクロサービス」という単語が使われ始める*2
2014年	AWSの年次イベント「re:Invent 2014」で発表された、サーバーを必要としないイベント駆動型のプログラム実行環境、Lambdaの説明として「サーバーレス」という単語が使われる*3

*1 The Art of Unix Programming
*2 James Lewis/Martin Fowlerの"Microservices"日本語訳 - 自由課題
*3 原典不明



比較をしていきます

モノリス

VS

マイクロサービス

VS

サーバレス

アーキテクチャ比較

	伝統的なアプリケーション モノリシック(一枚岩)	マイクロサービス	サーバレス
依存関係	密結合	疎結合	マイクロサービスと同じ
技術スタック	アプリケーション単位 ライセンスに関連した商用モデルにより単一のデータベース	サービス単位 単一・または複数のデータベース （Polyglot Persistence）	プログラム単位 単一・または複数のデータベース （Polyglot Persistence）
コードベース	単一	複数	マイクロサービスと同じ
スケール	アプリケーション全体	サービス単位	サービス単位*1
ドメイン境界	曖昧になりやすい	凝集性のあるドメイン境界を通じる 変更を最小化	マイクロサービスと同じ
コンポーネント	単一	独立して交換・アップグレード可能な複数のサービス	マイクロサービスと同じ
リクエスト	全て単一のプロセスで処理。中央集権的なツールによるオーケストレーションのような複雑なプロトコル	プロセス は 廃棄容易で即座に起動と終了。 REST風プロトコル。協調を重要視し協調方法は一貫性の保証は結果整合性のみ	マイクロサービスと同じ
コンポーネント間通信	メソッド呼び出しもしくは関数コール	リソースAPIを用いたHTTPの リクエスト-レスポンス 軽量なメッセージ通信	マイクロサービスと同じ
アジリティ (デプロイ間隔)	数週間 モノリス全てのリビルドとデプロイを必要 伝統的な中央集権的な統治モデル	数時間 サービスが独立してデプロイ可能 分散統治	数時間 プログラムが独立してデプロイ可能 分散統治
デプロイする人	異なる人	同じ人 (you build, you run it)	マイクロサービスと同じ
開発/本番環境一致	異なる	できるだけ一致	マイクロサービスと同じ
チーム境界	完成後は運用組織に引き渡されチームは解散。運用組織はソフトウェアの断片をデリバリ	UX、ミドルウェア、データベース、そしてプロジェクト管理など必要な全てのスキルを含む機能横断型	UX、ミドルウェア、データベース、そしてプロジェクト管理など必要な全てのスキルを含む機能横断型、または専門型

*1 hook.ioの場合。AWS LambdaなどFaaSの場合はプログラム単位

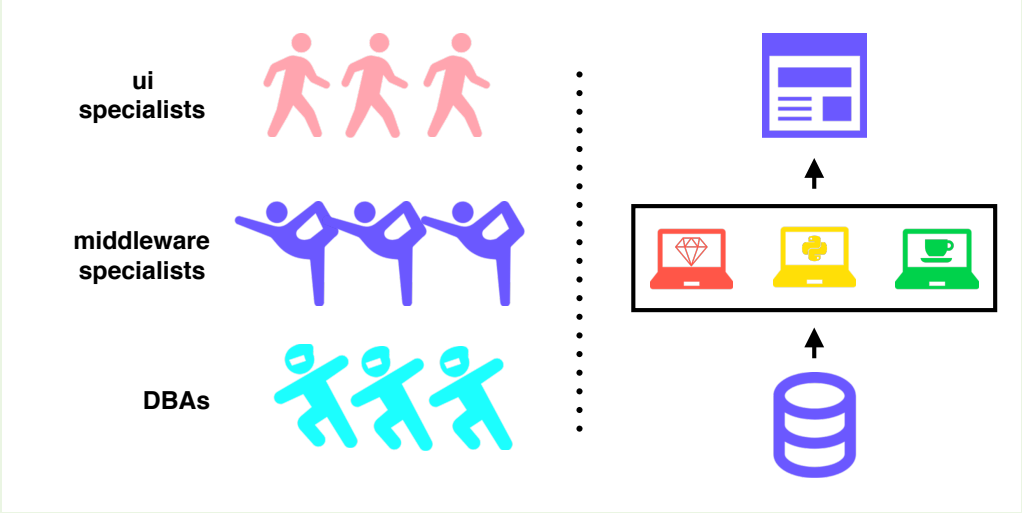
チーム構成に着目してマイクロサービス
とサーバーレスについて考えてみましょう

3. チーム編成

モノリシック

プロジェクトモデル

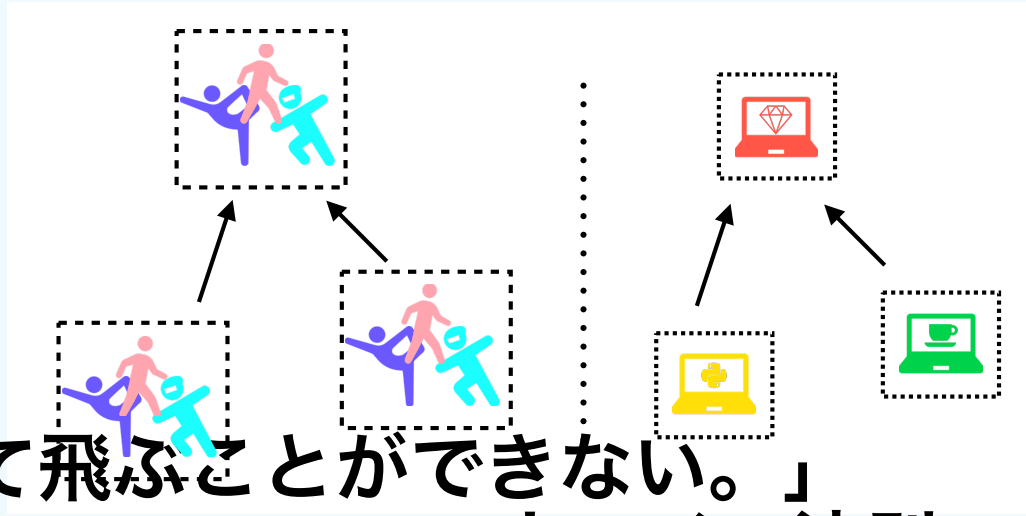
- UXデザイナー
- ミドルウェアスペシャリスト
- データベースアーキテクト



マイクロサービス

機能横断型(cross-functional)

- UXデザイナー
- ミドルウェアスペシャリスト
- データベースアーキテクト



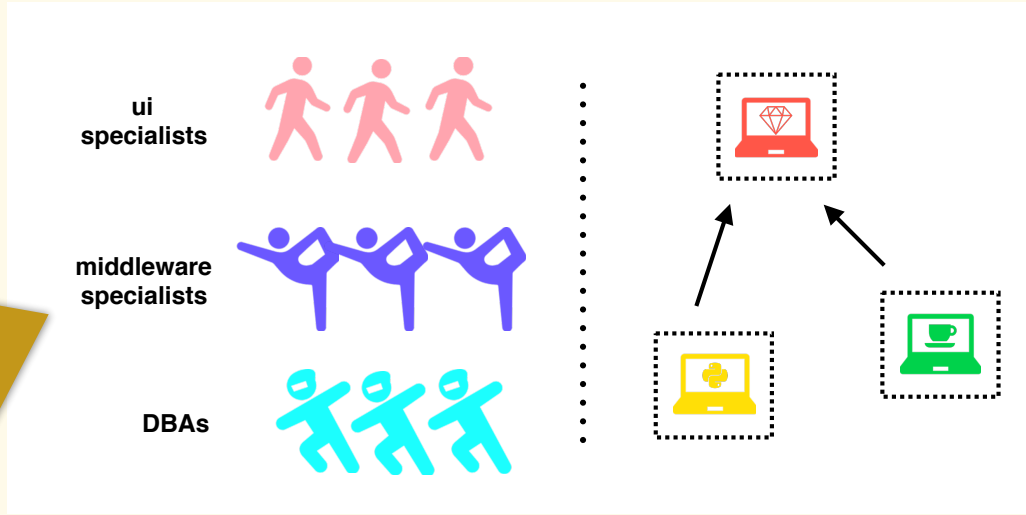
「魂を重力に引かれて飛ぶことができない。」
コンウェイの法則

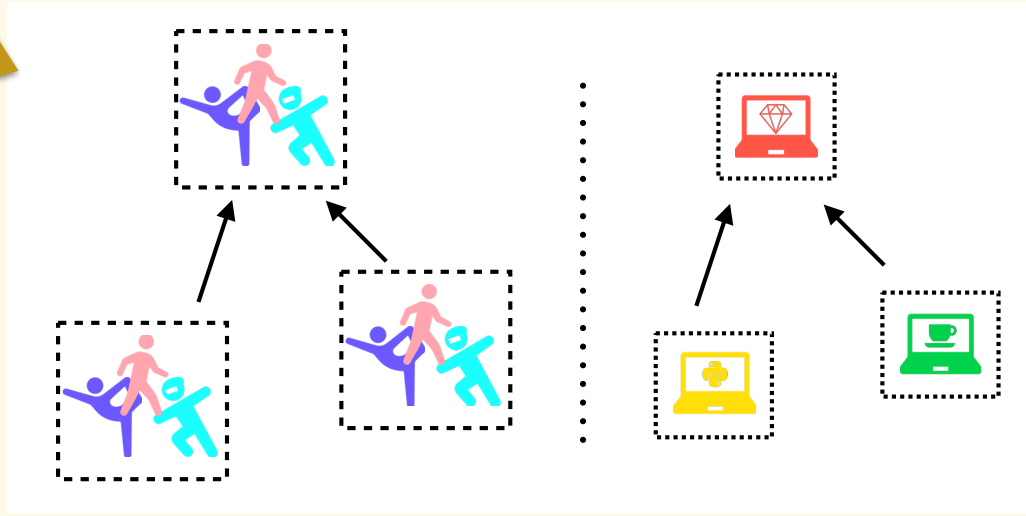
チーム編成

サーバーレス

機能横断型、または機能専任型

- UXデザイナー
- ミドルウェアスペシャリスト
- データベースアーキテクト





4. hook.ioでなにがうれしいの？

hook.ioとは「オープンソースマイクロサービスホスティングプラットフォーム」

- モダンな**ウェブプラットフォーム**
- 高い**アジリティ**でマーケットの機会をとらえた数時間単位の変更サイクル
- 豊富な対応言語、制限がないライブラリによる企業の**既存資産の活用**。{JavaScript, Lua, Perl, PHP, Python, Ruby, Scheme, SmallTalk, TCL, C, Java*, Coffee-script*, Common lisp*, Bash*, Golang*, OCaml*, Rust*, R*} *microculeの場合
- 巨大なアプリケーションのプロジェクト全体理解が必要なくなり、ヤックシェービングを防いだ**漸進的な開発**
- 機能横断型のチームが必要なマイクロサービススタイルより、プロジェクト理解要求が低く、**ベテラン10xプログラマと新人プログラマの差が大きくならない**。コードや標準出力を読まない、大規模開発の経験、デザインパターンの知識がない新人プログラマもビジネス遂行の戦力にする。
- **ドメイン駆動設計**の感覚であるエンティティに関連付けられていない重要な処理を行うオブジェクトを、サービス契約を満たしながら進化

ステートレス

5. デモンストレーション

1. hook.ioの操作説明

複数言語によるサンプルAPIを作成してみる

http://hook.io

2. 実際の開発環境を想定した操作説明

サンプルコードの説明と実行

Githubへ変更をプッシュ

変更の確認

複数言語のサンプルの実行例

tensorflowライブラリを利用した例

デモ詳細

1 gui hook.io web
Api sample 作成
Api sample の更新
2 cui
A)どっかーの立ち上げ
B)Curl でハローワールド
Github Python のコード説明
Curl コマンドを叩く
Pythonコードの変更 (ライブ)
Curlコマンド叩く
以降、ルビー、ocaml、rustを繰り返し
C) Apiの連携、ライブラリの追加応用例(画像解析)
ゴリラの画像をぐぐる
URLをルビーに渡す
Pythonで解析
D) 参加者からすきな動物を聞く
c)を繰り返す(ライブ)

参考資料など

文献

The Twelve-Factor App
Serverless Architectures / martinfowler.com
James Lewis/Martin Fowlerの"Microservices"日本語訳 - 自由課題
マイクロサービスアーキテクチャ Sam Newman
エリック・エヴァンスのドメイン駆動設計

イメージ画像

©2017 ClipartPanda.com
©clipart-library.com
©www.1001freedownloads.com
©photo.elsoar.com
©pixabay.com
©TopeconHeroes
©Freepik
©flaticon.com
©Bogdan Rosu Creative
©Iconfinder ApS
©AC部コミュニティサイト

インスパイア

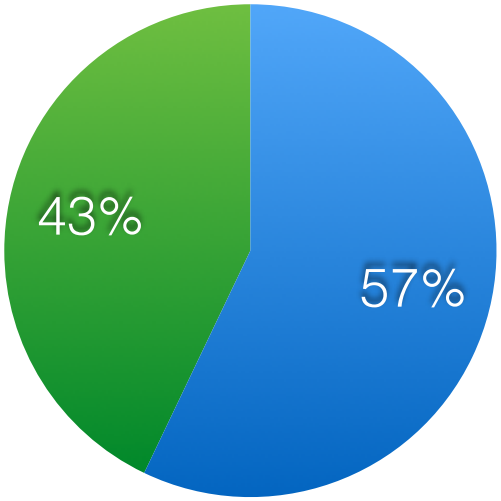
©glitch.com

hook.io
https://github.com/bigcompany/hook.io

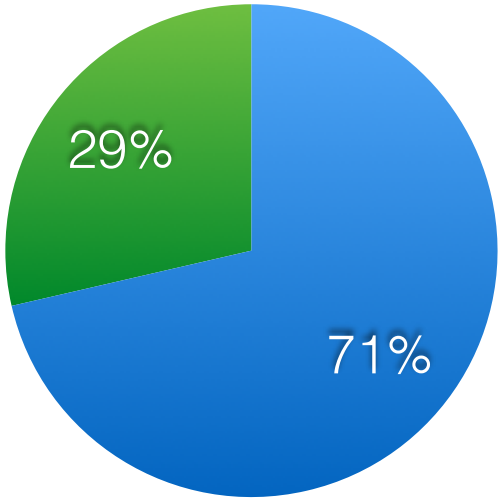
おまけ　アンケート結果

アンケートは複数回答

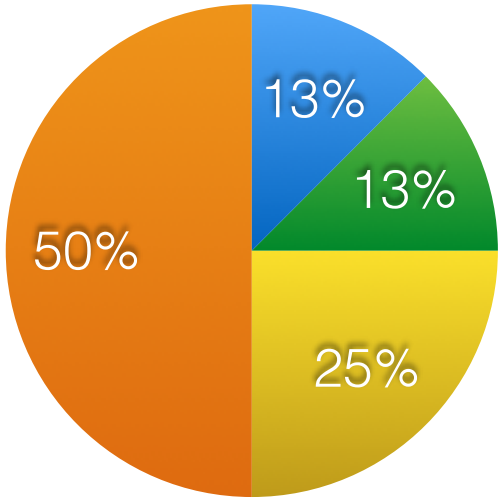
アンケート回収率



主に携わるシステム形態

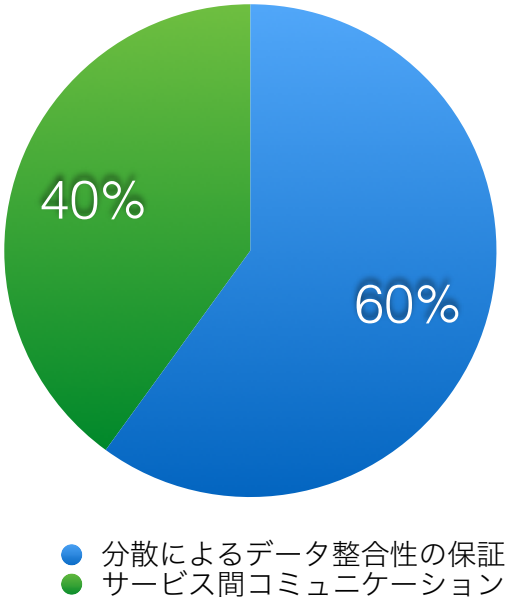


利点に感じた箇所

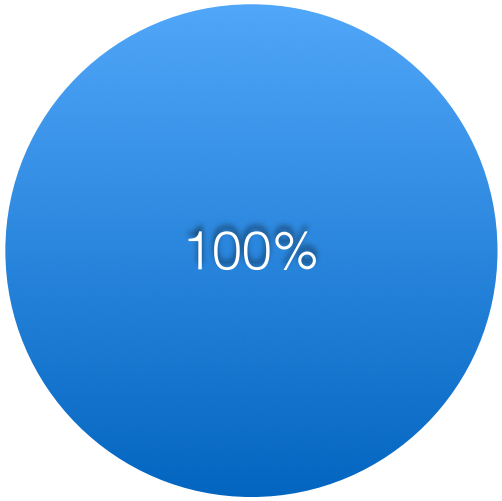


おまけ　アンケート結果2

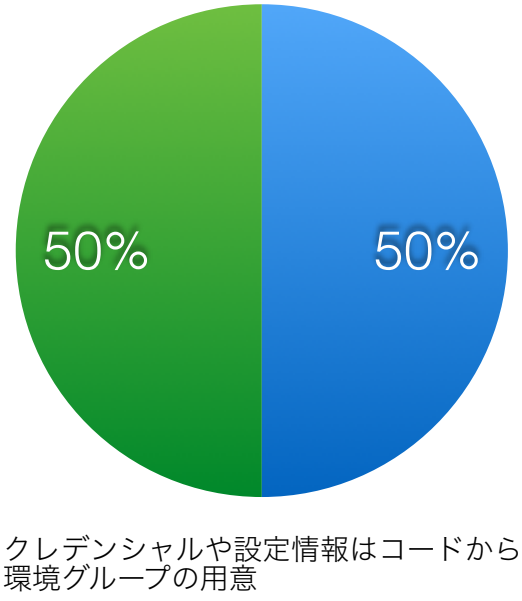
サーバーレス導入に関する懸念



導入済みDevOps



導入済み継続的インテグレーション



おまけ　アンケート結果3