2019年度アドバンス・トップエスイー
最先端ソフトウェア工学ゼミ　個別ゼミⅠ成果発表
Microservices グループ

Microservices
～その定義と効用～

酒井　響平
瀬戸　司
城戸　英之

調査のモチベーション

- **Microservices化する動機に合ったサービス分割を行い、分割を評価する開発時・運用時の指標を調査したい（酒井）**
- **Microservicesを構築するうえでのアプリケーションアーキテクチャの設計法について具体的な技術を調査したい（瀬戸）**
- **家庭用コンシューマ製品の組込みソフトには省リソースの観点から向いてないが、リソースをふんだんに使える高機能な商用コマース製品には適用可能か調査したい（城戸）**

Outline

- マイクロサービスとは
- マイクロサービス: アーキテクチャと開発・運用体制
- マイクロサービスのメリット/デメリット
- マイクロサービス化の手順
- マイクロサービスパターン
- マイクロサービスの実行基盤
- マイクロサービス開発・運用のメトリクス
- 事例研究
- まとめ

マイクロサービスとは

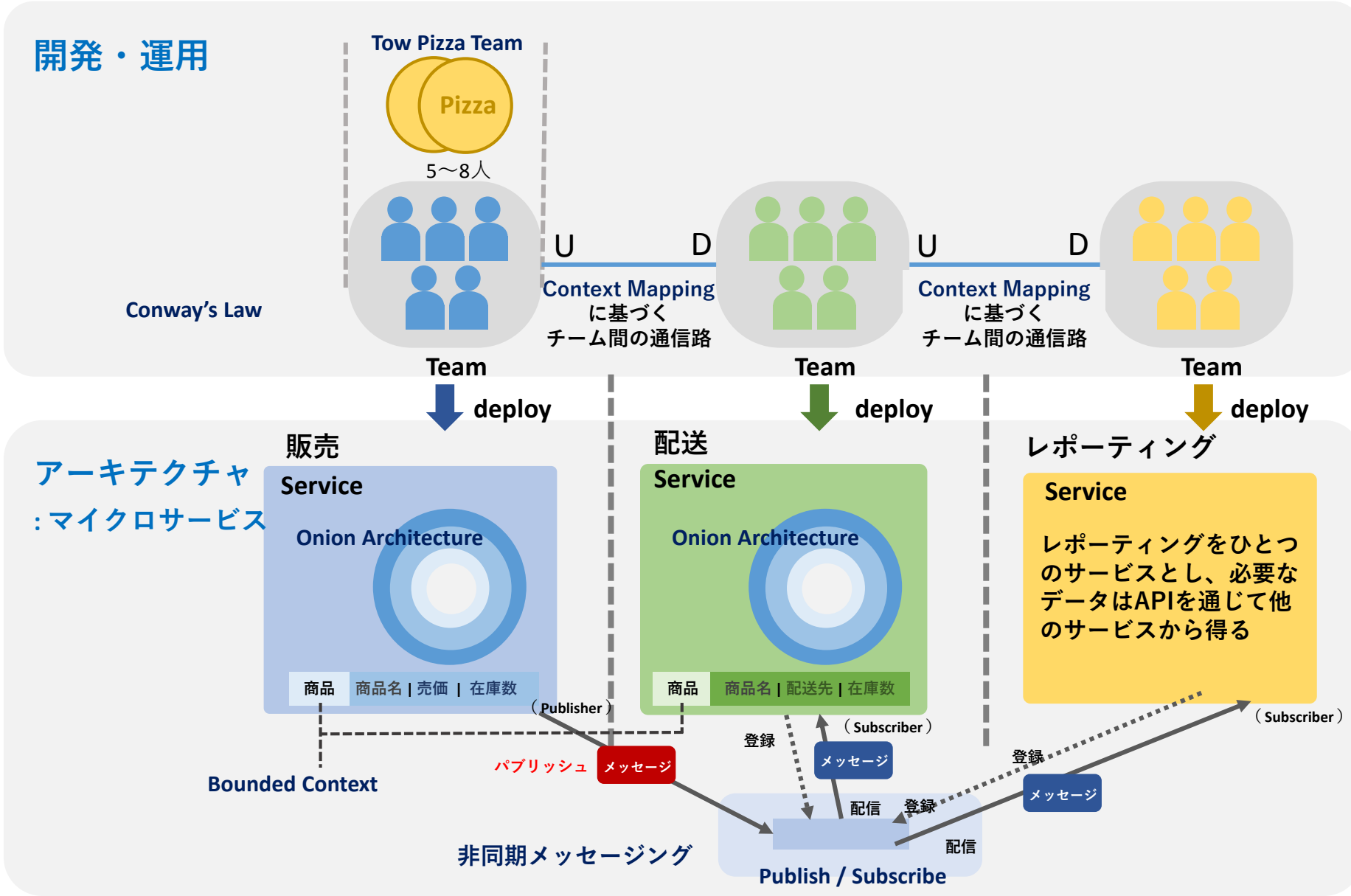
マイクロサービスとは

　それぞれが自律的に進化する独自のライフサイクルを持ち、共に協調する細粒度のサービス群の利用を促進する、 分散システムに対する一つのアプローチである。

サービスとは

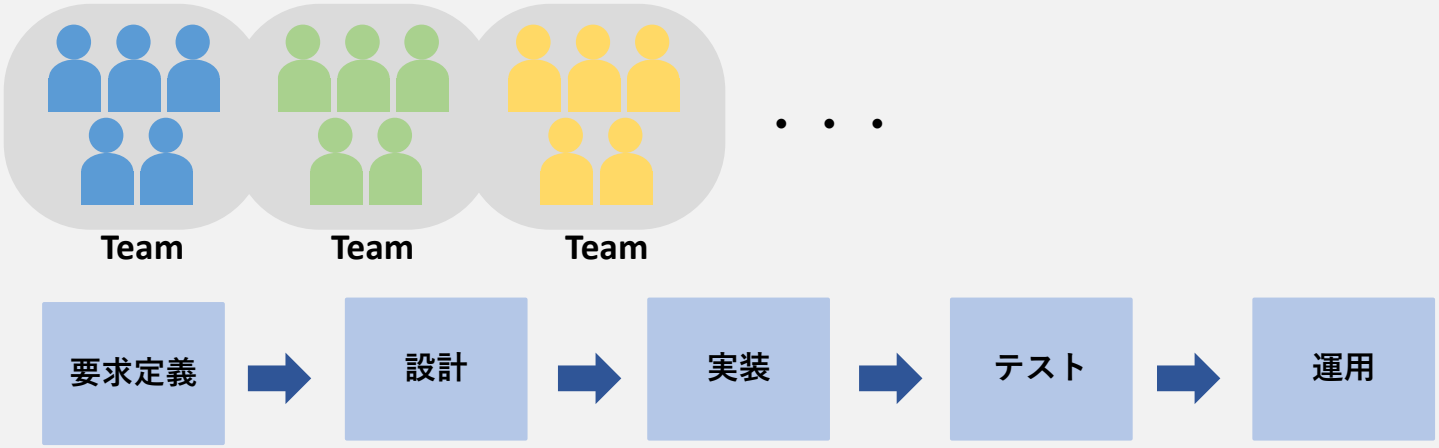
　Domain-Driven Designにおけるコンセプトであり、例えば、ECサイトのシステムにおいて、「商品」に関しては販売コンテキストと配送コンテキストに分ける． ドメイン駆動設計では、同じ意味で言葉を使う範囲をコンテキスト（Bounded Context）として定義する。

マイクロサービス: アーキテクチャと開発・運用体制

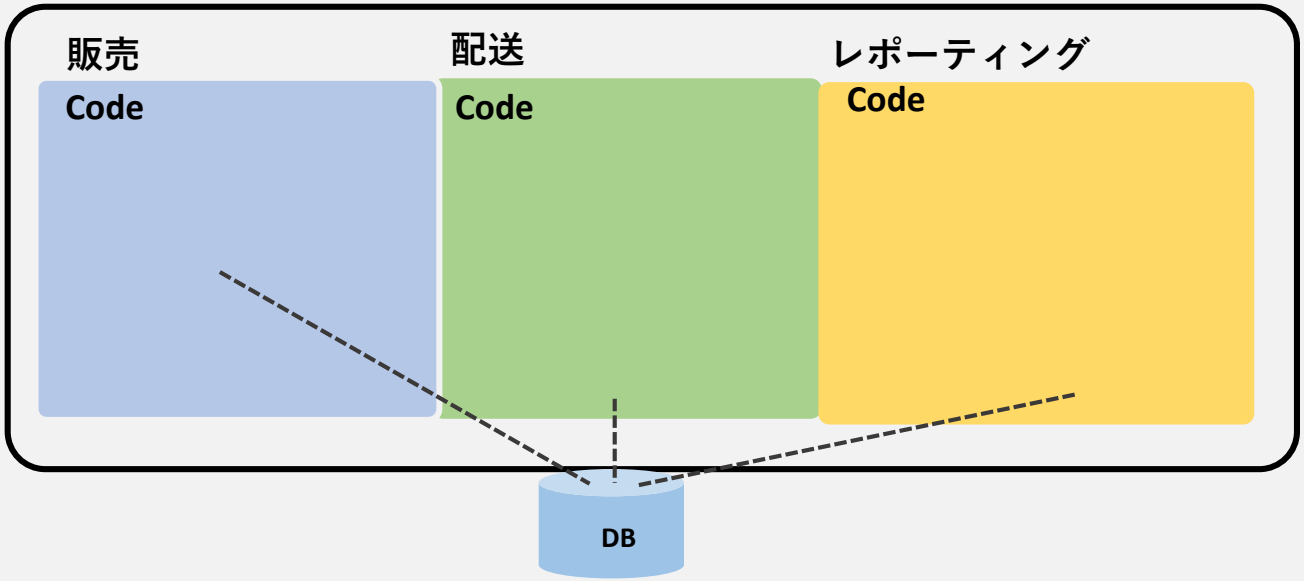


モノリシックとマイクロサービス

開発・運用



アーキテクチャ
: モノリシック



マイクロサービスのメリット/デメリット

マイクロサービスのメリット

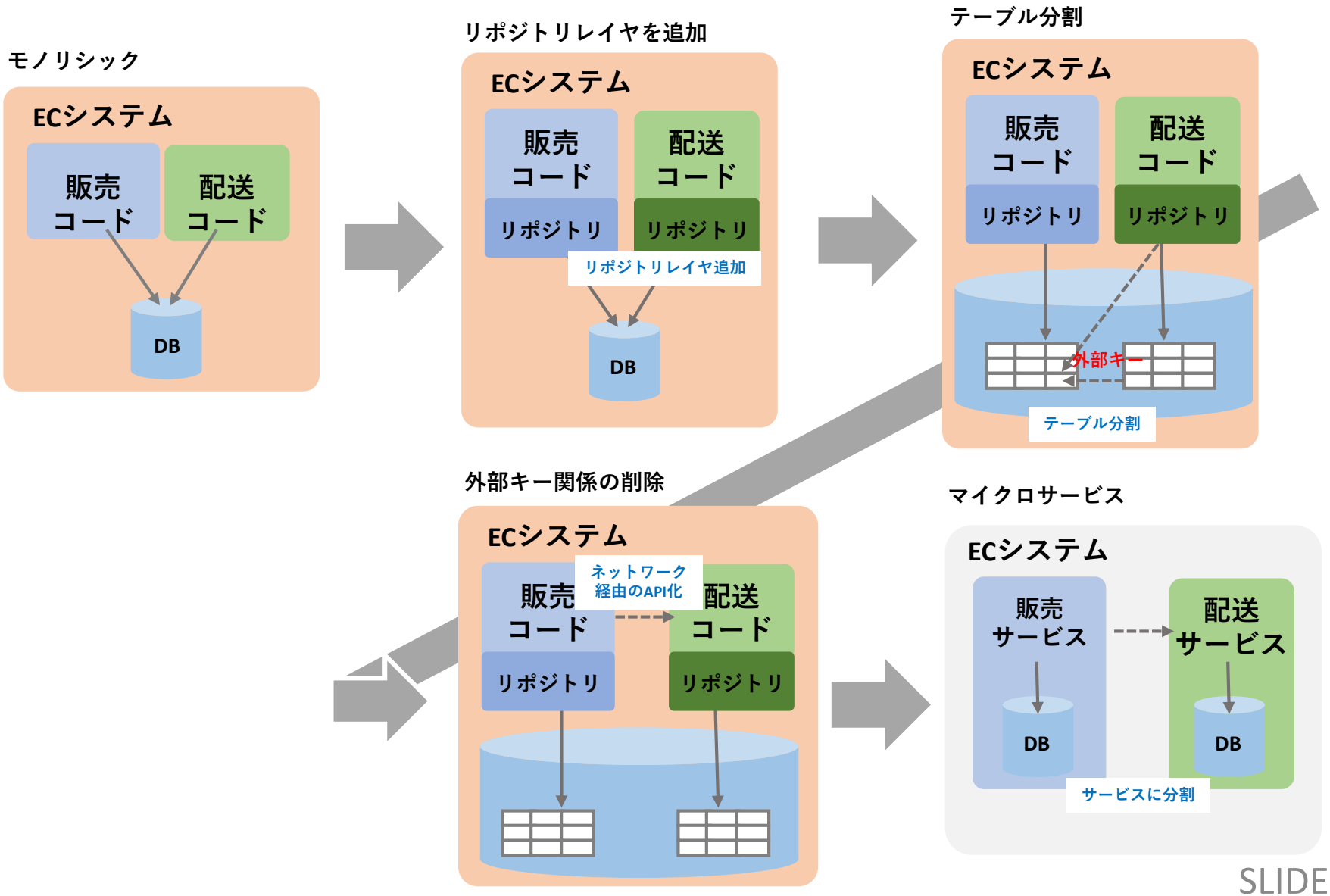
- サービス同士が疎結合であるため、一部サービスの変更が全体に影響を及ぼすが小さい
  → 変更影響・障害影響を局所化できる
- サービス単位でのビルド・デプロイが必須
  → サービスごとに独立したサイクルで開発可能
- プロビジョニングによる負荷分散が容易
  → スケーリングか必要なサービスのみスケール可能

マイクロサービスのデメリット

- サービスに分割することで、システム全体の複雑さが増加する
  → モニタリング・リポートの情報収集が難しくなる
- サービス間通信のオーバーヘッドが大きい

マイクロサービス化の手順

例： DB（外部キー参照）の分割ステップ



マイクロサービスパターン

- マイクロサービスの設計に利用できるパターンを企業・機関が公開している

  Microservices Patterns:
  https://microservices.io/

  Microsoft Azure:
  https://docs.microsoft.com/en-us/azure/architecture/microservices/design/patterns

- 設計パターンには、アプリケーション層からインフラ層までの問題領域に関する様々なパターンが紹介されている

メッセージングスタイルの問題領域
- Remote Procedure Invocation
- Messaging
- Domain-specific protocol

外部APIの問題領域
- API gateway
- Backend for front-end

サービスディスカバリーの問題領域
- Client-side discovery
- Server-side discovery
- Service registry
- Self registration
- 3rd party registration

信頼性の問題領域
- Circuit Breaker

セキュリティの問題領域
- Access Token

監視の問題領域
- Log aggregation
- Application metrics
- Audit logging
- Distributed tracing
- Exception tracking
- Health check API
- Log deployments and changes

UIパターン
- Server-side page fragment composition
- Client-side UI composition

サービス分割の問題領域
- Decompose by business capability
- Decompose by subdomain

データ整合性の問題領域
- Database per Service
- Shared database
- Saga
- API Composition
- CQRS
- Domain event
- Event sourcing

トランザクショナル・メッセージングの問題領域
- Transactional outbox
- Transaction log tailing
- Polling publisher

テスト自動化の問題領域
- Service Component Test
- Consumer-driven contract test
- Consumer-side contract test

デプロイの問題領域
- Multiple service instances per host
- Service instance per host
- Service instance per VM
- Service instance per Container
- Serverless deployment
- Service deployment platform

横断的関心事の問題領域
- Microservice chassis
- Externalized configuration

参考：A pattern language for microservices https://microservices.io/patterns/index.html

マイクロサービスの実行基盤

Kubernetes

- **Docker**によりコンテナ化されたアプリケーションのデプロイ、スケーリングなどの管理を自動化するためのコンテナオーケストレーションエンジン
- デファクトスタンダード
  パブリッククラウドで利用できるマネージドKubernetesサービス
  - Google Kubernetes Engine (GKE)
  - Azure Container Service (AKS)
  - Elastic Container Service for Kubernetes (EKS)
- マイクロサービスアーキテクチャで構成されたシステムをデプロイする場合、1つのマイクロサービスを1つのコンテナイメージとして開発し、デプロイを行う

参考：
北山 晋吾, 早川 博 (2019) 「Kubernetes実践ガイド」 株式会社インプレス
青山 真也 (2018) 「Kubernetes完全ガイド」 株式会社インプレス

マイクロサービスの実行基盤

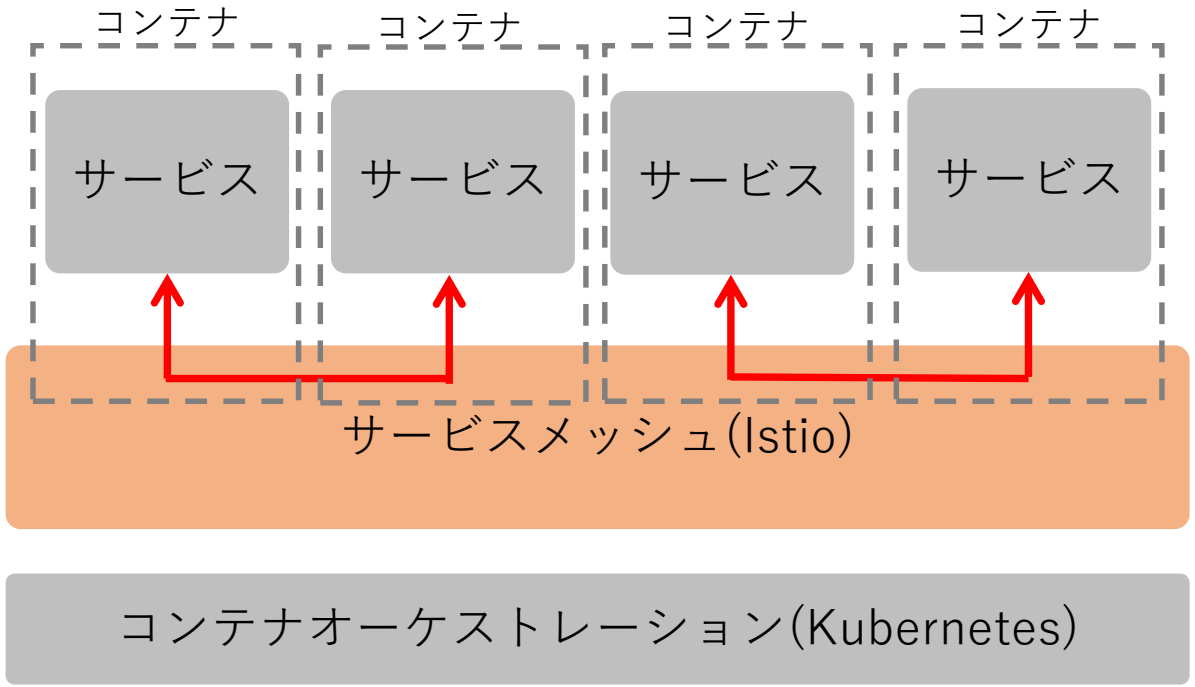
Istio(サービスメッシュ)

- Google/IBM/Lyftによって開発され、2017年にOSSとして公開されたサービスメッシュの実装
- サービスメッシュは、多数のマイクロサービスの分割・連結によって生じる課題の解決策となる考え方

  - ネットワーク通信に起因する障害
  - パフォーマンス劣化
  - 運用オペレーションの負荷の増加
  - 障害解析の高難度化

参考：
北山 晋吾, 早川 博 (2019) 「Kubernetes実践ガイド」 株式会社インプレス
青山 真也 (2018) 「Kubernetes完全ガイド」 株式会社インプレス

- 各マイクロサービス(Pod)間の通信経路にプロキシを挟むことでトラフィックのモニタリングやトラフィックをコントロール
  ※ 分散トレーシングにはJaeger/zipkinの導入が必要
- 既存のアプリケーションのコードに手を加える必要なく、サービスメッシュを構成することが可能



参考：
北山 晋吾, 早川 博 (2019) 「Kubernetes実践ガイド」 株式会社インプレス
青山 真也 (2018) 「Kubernetes完全ガイド」 株式会社インプレス

マイクロサービス開発・運用のメトリクス

マイクロサービスの開発・運用時のメトリクスの例として以下のものが挙げられる

設計時メトリクス

- 各サービスのステップ数(サービスのサイズ)
- 各チーム人数
- 同期呼び出しトレースの最大長(パフォーマンス)
- 同期/非同期インターフェース数(サービスのサイズ, 1 task/service)

運用時メトリクス

- 各API呼び出し回数(結合度、凝集度、メンテナンス性、ネットワーク複雑性)
- 非同期メッセージの平均長(パフォーマンス)
- レスポンスタイム
- 可用性
- CPU・メモリ使用率

※ 橙色はKubernetes, Istioによる収集が可能

参考：Thomas Engel. “Evaluation of Microservice Architectures: A Metric and Tool-Based Approach”. *Information Systems in the Big Data Era (CAiSE Forum 2018)* pp74-99, 2018.

事例研究

研究対象アプリケーション:

Acme Air

- モノリシック版
  - https://github.com/blueperf/acmeair-monolithic-java
- マイクロサービス版
  - https://github.com/blueperf/acmeair-authservice-java
  - https://github.com/blueperf/acmeair-mainservice-java
  - https://github.com/blueperf/acmeair-flightservice-java
  - https://github.com/blueperf/acmeair-customerservice-java
  - https://github.com/blueperf/acmeair-bookingservice-java

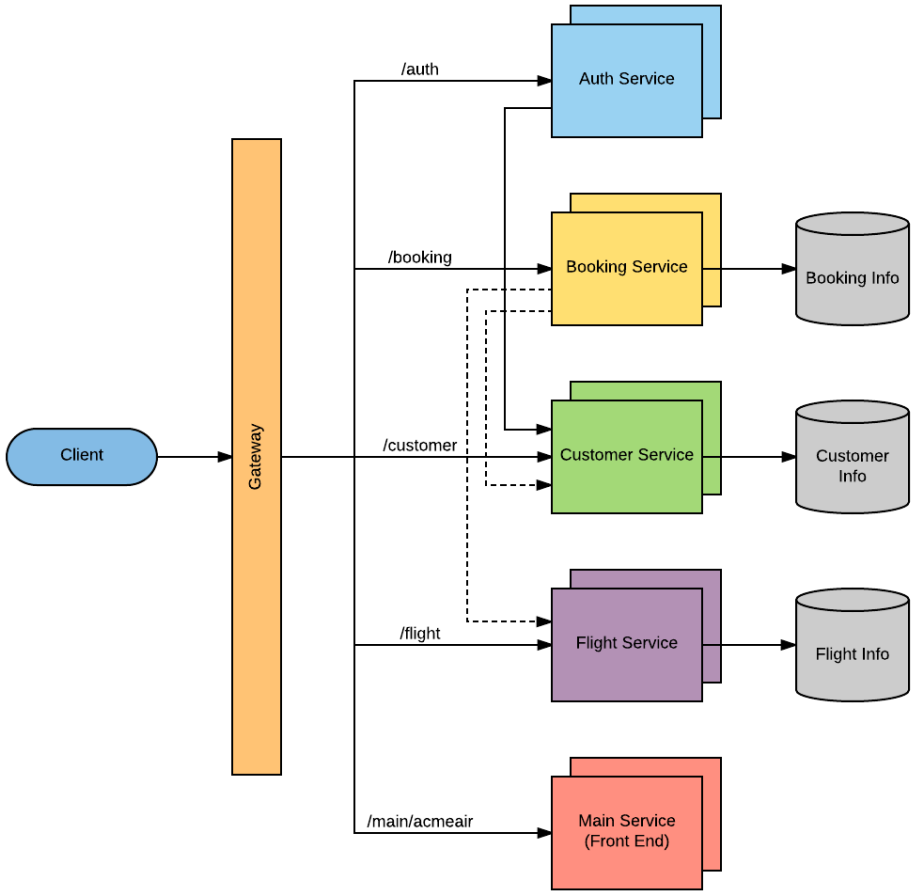
特徴

- IBMにより開発されたオープンソースのベンチマークアプリケーション
- Java/Node.jsで実装された架空の航空チケットシステム
- モノリシック・マイクロサービス双方による実装が含まれている

構成図

マイクロサービスによる実装では次の5サービス・3DBから構成される

- 認証サービス
- 予約サービス
- 顧客サービス
- フライトサービス
- メインサービス(フロントエンド)
- 予約DB
- 顧客DB
- フライトDB



実験環境の構築

以下のツール類を利用して環境を構築した

- 共通
  - OS：**Ubuntu 18.04**
  - 負荷生成：**Apache JMeter**
- モノリシック版
  - アプリケーションサーバ：**Apache Tomcat**
  - データベース：**MongoDB**
- マイクロサービス版
  - サービス：**Docker**
  - アプリケーションサーバ：**WebSphere Application Server**
  - データベース：**MongoDB**

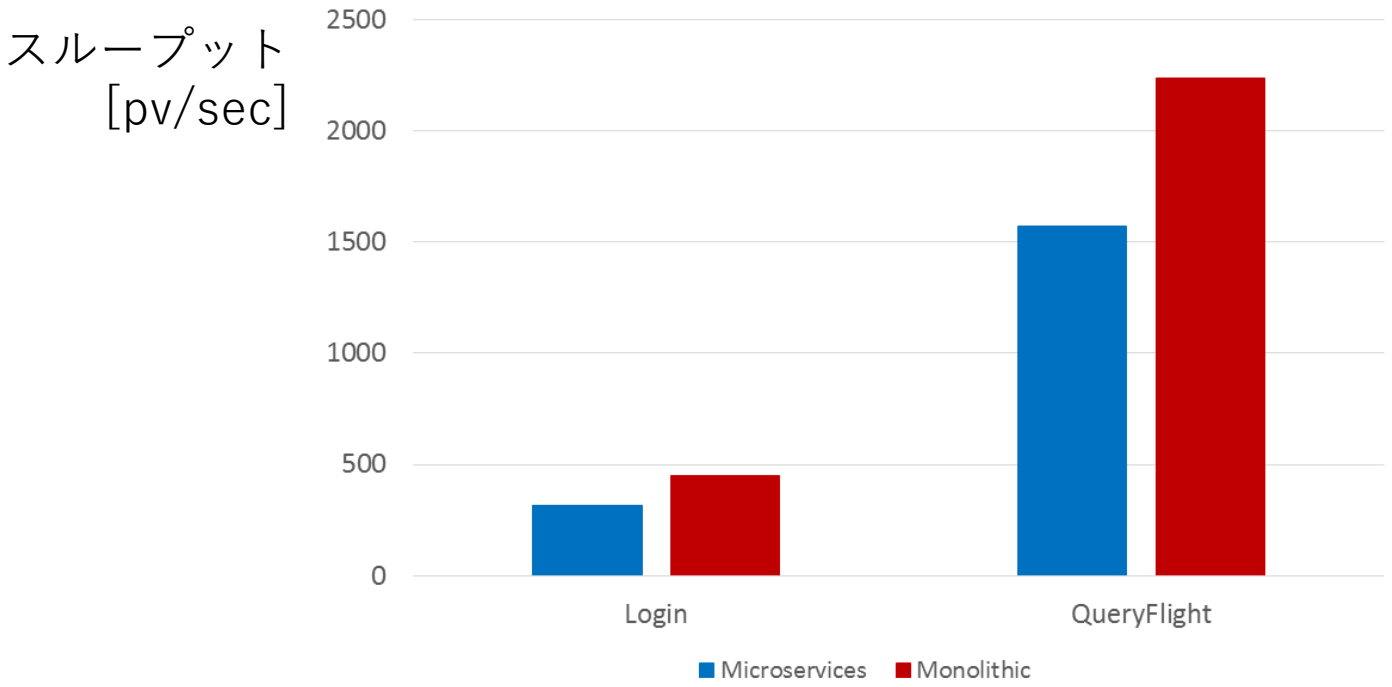
実験結果 (1/2)

	モノリシック	マイクロサービス （下段は5サービスの平均）
総ステップ数 [step]	1661	2222
		444.4
クラス数 [個]	37	70
		14
インターフェース数 [個]	5	11
デプロイ時間*1 [sec]	0.63	11.65

- マイクロサービス化により実装量が増加することから、マイクロサービス化するには相応の工数が必要だということが改めて確認できた
- モノリシック版に比べ、1サービスあたりの行数・クラス数は減少するため、メンテナンス性には優れると言える
- デプロイ時間はモノリシック版の方が優れる一方、一部機能のみの更新の場合でもシステム全体がダウンする時間は必ず発生する

*1: 10試行の平均値(実行環境：Ubuntu 18.04 / Intel Atom x5-Z8300@1.44GHz / RAM 1.8GiB)
モノリシック：アプリケーションサーバへのwarファイル配備に要する時間
マイクロサービス：`docker-compose`コマンドによる対象サービスの再配備に要する時間

実験結果 (2/2)



2つのエンドポイントに一定時間負荷をかけた結果、
モノリシック版の方がスループットが優れると判明

→　マイクロサービス版ではサービス間がHTTP通信を行うため、
その通信オーバーヘッドが影響したと考えられる

実行環境：Ubuntu 18.04 (Windows 10 + VirtualBox 6.0) / Intel Core i9 @3.6GHz / RAM 64GB

まとめ

- マイクロサービスへの分割を評価するのに適したパターンと指標を洗い出せた一方、サービス分割を行うためのアプローチが難解で、知見を展開することへの課題感が残る（酒井）

- ビジネスロジックを明確化し、UI、フレームワーク、DBなどから独立する移植性や保守性を高めるDDDの設計思想は、業務においても参考になるところが多い（瀬戸）

- 高機能な商用コマース製品のプリントワークフローにおける、受注システムのUI・生産システムの改善に適用可能だと考えられる（城戸）

参考文献

ドメイン駆動・マイクロサービス

- Sam Newman(著), 佐藤 直生(監訳), 木下 哲也(訳) (2016) 「マイクロサービスアーキテクチャ」 株式会社オライリー・ジャパン
- エリックエヴァンス(著), 今関 剛(監訳), 和智 右桂, 牧野 祐子(訳) (2011) 「エリック・エヴァンスのドメイン駆動設計」株式会社翔泳社

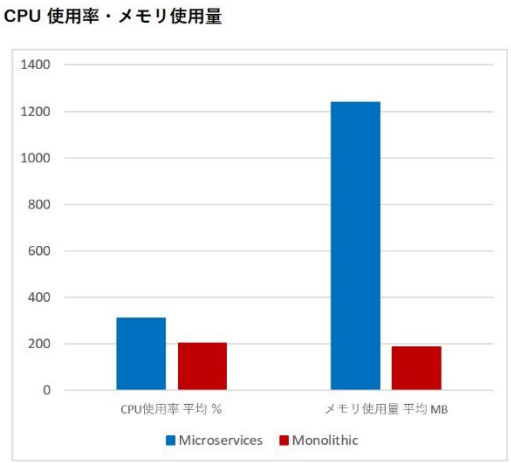
マイクロサービス 開発・運用のためのツール

- 北山 晋吾, 早川 博 (2019) 「Kubernetes実践ガイド」 株式会社インプレス
- 青山 真也 (2018) 「Kubernetes完全ガイド」 株式会社インプレス

マイクロサービス開発・運用のメトリクス

- Thomas Engel. “Evaluation of Microservice Architectures: A Metric and Tool-Based Approach”. *Information Systems in the Big Data Era (CAiSE Forum 2018)* pp74-99, 2018.

CPU・メモリ使用率の変化



	Microser	Monolithi
CPU使用率 平均 %	309.7066	202.505
メモリ使用量 平均 M	1239.759	188.1917