

Microservices Discussion

この発表の趣旨

- マイクロサービス・アーキテクチャに関連するいくつかの発表にインスパイアされて、いろいろモヤモヤ考えたことを共有します
- あとディスカッションがなにかできたらと

目次

- イン트로ダクション
- モノリシックらしさ、マイクロサービスらしさの指標
- ケーススタディ - とあるモジュラモノリスなアプリケーションの話
- 残された議論

目次

- イン트로ダクション
- モノリシックらしさ、マイクロサービスらしさの指標
- ケーススタディ - とあるモジュラモノリスなアプリケーションの話
- 残された議論

モノリシック・アーキテクチャ

単一のアプリケーションとしてシステムを実装

- 単一の技術スタック
- アプリケーションの構成要素はプロセス内で呼び合う（コード上のメソッド呼び出し）
- アプリ全体がデプロイメントの一単位

（以下、「モノリシック」と書きます）

マイクロサービス・アーキテクチャ

小規模なサービスの組み合わせでシステムを実装

- サービス毎に異なる技術スタック
- サービス同士がネットワーク経由で呼び合う（疎結合）
 - REST over HTTP, gRPC, 非同期メッセージング ...etc
- 個々のサービスがデプロイメントの一単位

（以下、「マイクロサービス」と書きます）

モノリシックの課題

モノリシックはビジネス環境の変化に対応しにくい⇒マイクロサービス

- モノリシック・アーキテクチャは、大規模化すると機能要素同士の依存関係が複雑化しがち
 - 一部の変更が全体にどう影響するか把握しにくい
 - 大量の回帰テストなどのために、システムの更新に長い工期を要する
- マイクロサービス
 - 疎結合な連携方式のため、変更の影響範囲を個々のサービスに留められる
 - システムを（サービス単位で）素早く更新できる

モノリシックとマイクロサービスの二元論

現実のシステムは必ずしもどちらかに決まるわけではない

- モノリシックとマイクロサービス、2つに1つの二元論で考えがち。でもそんなに単純な話だろうか？

Monolith

(or)

MSA

- モノリシックであっても、内部がきれいにモジュール化されていると「よく設計されたモノリス」とか言ったりする
- モノリシックとマイクロサービスの **間の状態** というのがありそう

モノリック vs. マイクロサービス 二元論の弊害

マイクロサービスへの移行を阻む要因になっていないか？

- ふたつにひとつと捉えていると、マイクロサービスへの移行が大きなジャンプアップのように感じられる
 - 何をどうしたら良いのか、具体的に考えにくい
 - 実態以上に難易度が高いように感じられる
 - 作業が見積もれないので投資対効果が判断できない

Monolith - - - - - (大ジャンプ!) - - - - - -> MSA

モノリシックとマイクロサービスの間 - [Spotifyのモジュラモノリス](#)

内部実装をモジュール化して依存関係を整理

- コードベースは一本化し、ひとつのサーバーにすべてデプロイ（モノリシックと同じ）
- 機能ドメインごとにモジュラー化して境界を設ける。
 - Reorganize Code: ビジネス機能を基準にコードを分けて管理
 - Isolate Dependencies: 公開APIを通じてコンポーネントを利用
 - Enforce Boundaries: 依存コンポーネントを明示的に宣言

マイクロサービスの弊害を避けた結果、妥当な落とし所がこれだった

モノリシックとマイクロサービスの間 - [Grafana Lokiのモノマイクロリス](#)

モノリスとしてもマイクロサービスとしても利用可能

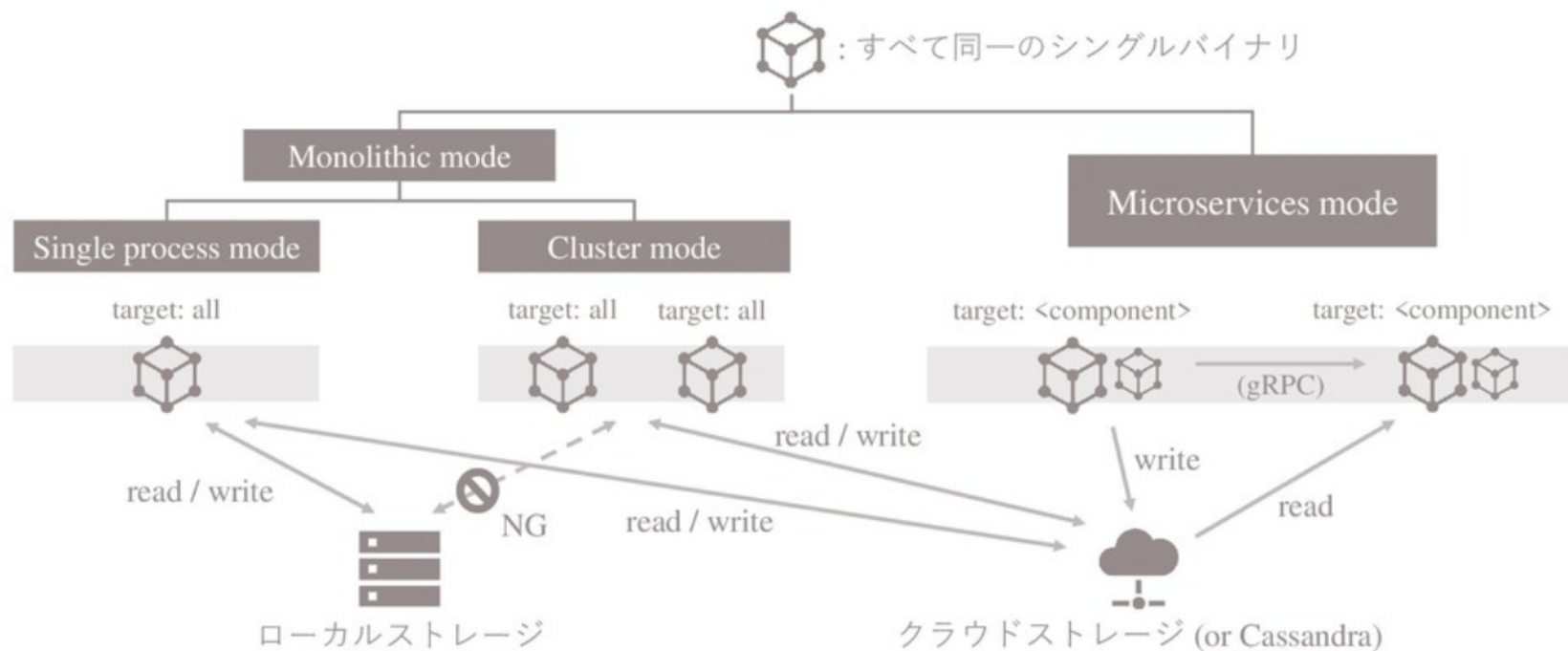
- 単一バイナリ、単一リポジトリで、モードを切り替えることで役割の異なるサービスとして動作ように実装
- 単一デプロイ、マルチモード・分散配置のいずれでも利用可能

大規模運用に適応しつつ、開発体験を高めるための工夫の結果

モノリシックとマイクロサービスの間 - [Grafana Lokiのモノマイクロリス](#)

Loki のデプロイ形態

Loki はユニークなシングルバイナリのまま、複数の実行モードを持つ



果たしてこれらが全てを解決するか…？

他にもたくさんの考慮点が残っていそう

- モジュールに分けるくらいなすでにやっているのだが…？
- DBは分けるの分けないの？
- コードベース一本化はビルド時間がつらいんじゃないか…？
- 組織はどうしたらいいの？
- ...etc

...考えることがまだありそう。

脱二元論！

モノリシックとマイクロサービスの間は連続的

- もっといろいろなアーキテクチャの状態があって、モノリシック／マイクロサービスらしさの程度は連続的なはず
- 「らしさ」を決める基準があれば、今どの辺なのかわかる

(このへんとか？)

↓

Monolith <- - - - - (連続的な状態のつながり) - - - - -> MSA

モノリシック／マイクロサービス間のものさし

らしさを測ることで移行作業を細分化できるのでは

- 「モノリシック／マイクロサービスらしさ」を測ることができれば
 - 今どの辺りなのかがわかる（現状）
 - 段階を分けて少しずつマイクロサービスに近づけられる（移行パス）



目次

- イン트로ダクション
- モノリシックらしさ、マイクロサービスらしさの指標
- ケーススタディ - とあるモジュラモノリスなアプリケーションの話
- 残された議論

モノリスらしさ、マイクロサービスらしさ

こういうものを定義したいのです。



Monolithらしさ、MSAらしさの指標

- 認証/認可とフロントエンド
- アプリケーションランタイム
- コンポーネント間のアクセス方式
- DB
- コードベース

注) 上記の項目には運用の観点（監視、ロギング）や組織論が入っていません

認証/認可とフロントエンド

バックエンドの分割が可能なように、認証/認可とフロントの構成を決めることがマイクロサービス化の前提。指すUXと実装難度を勘案して決める

- コンポーネント毎のサーバーサイドレンダリング + SSO
- ビュー統合サービス
- APIゲートウェイ型（SPA + APIのみのバックエンド）

メルカリはAPIゲートウェイ型を選択した

<https://logmi.jp/tech/articles/320198>

アプリケーションラインタイムの分離

(下に行くほどマイクロサービスらしい構成)

1. よくないモノリス
2. プロセスの分離
3. バイナリセットの分離
4. サーバー(OS)の分離

考えて見たいこと

- 実現の難度が高いのはどの分離点か
- コンポーネント毎のライフサイクルの独立性が得られる境界はどこか

DBの分離

1. データベース共有
2. スキーマとアクセス権の分離
3. コンテナDBによる分離
4. DBインスタンス分離
5. Polyglot Data Source

考えて見たいこと

- 実現の難度が高いのはどの分離点か
- トランザクションが効かなくなる境界は
- ライフサイクルの独立性が得られる境界は


コンポーネント間のアクセス方式の分離

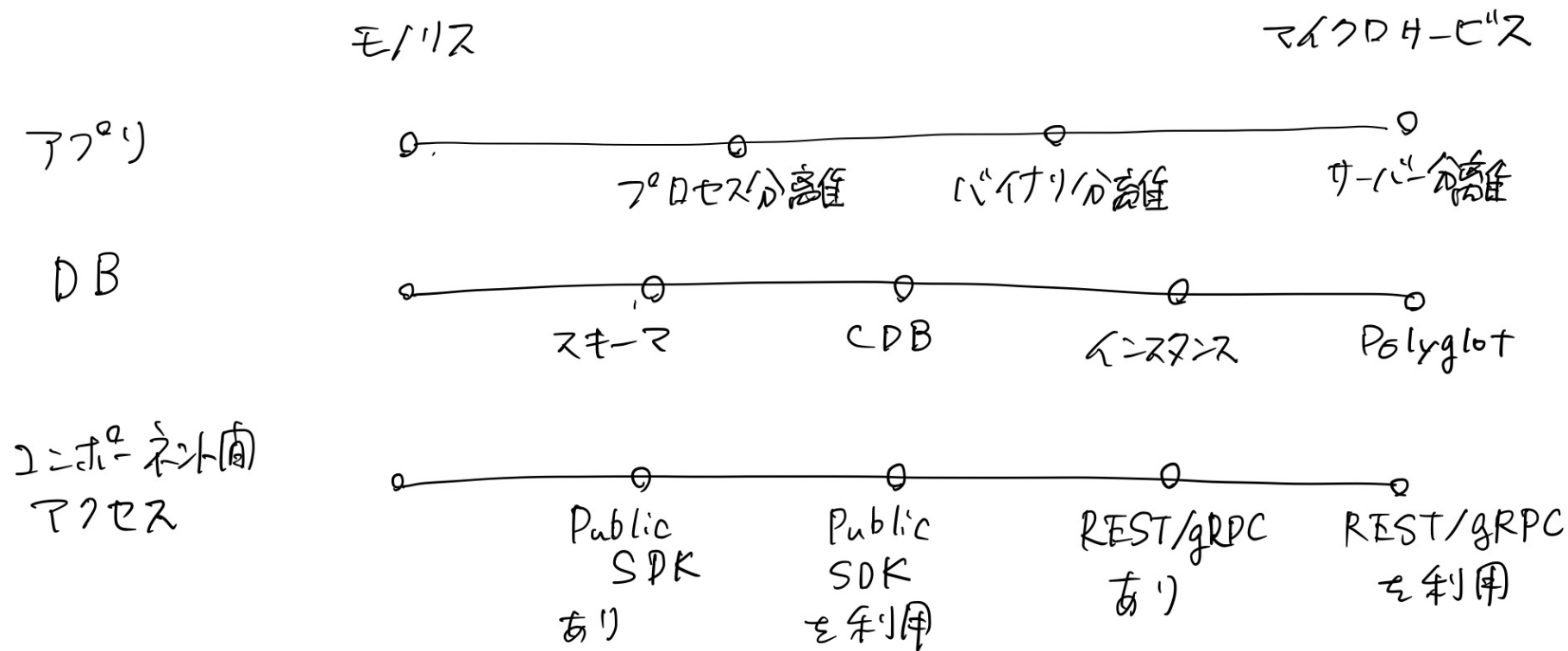
1. よくないモノリス
2. Public SDKが整理され、実装されている（サーバー側の実装）
3. Public SDKのみでアクセスが行われている（クライアント側の実装）
4. REST / gRPCが実装されている（サーバー側の実装）
5. REST / gRPCのみでアクセスが行われている（クライアント側の実装）

考えて見たいこと

- 非同期メッセージングはどのレベルに入るか

モノリスらしさ、マイクロサービスらしさ - 出来上がったもの

認証・認可 / フロントエンド: 

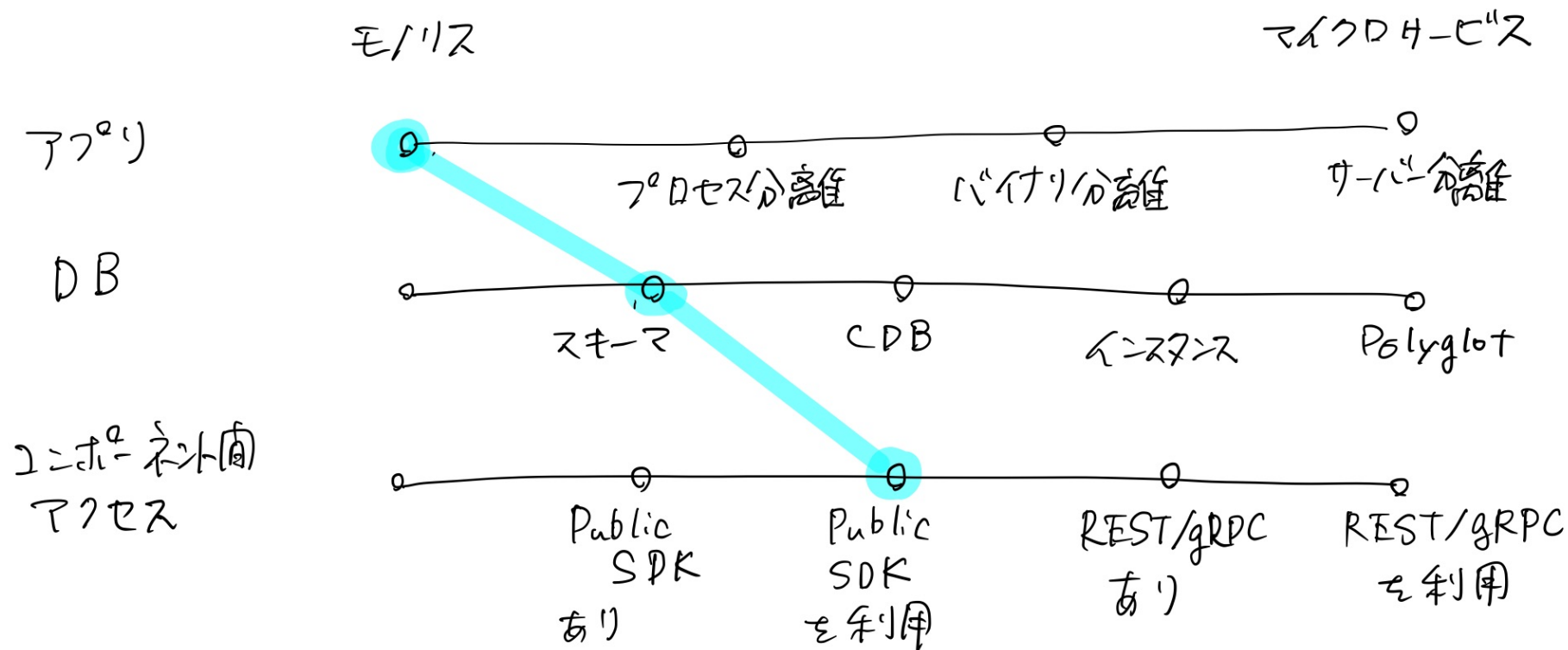


目次

- イン트로ダクション
- モノリシックらしさ、マイクロサービスらしさの指標__
- ケーススタディ - とあるモジュラモノリスなアプリケーションの話
- 残された議論

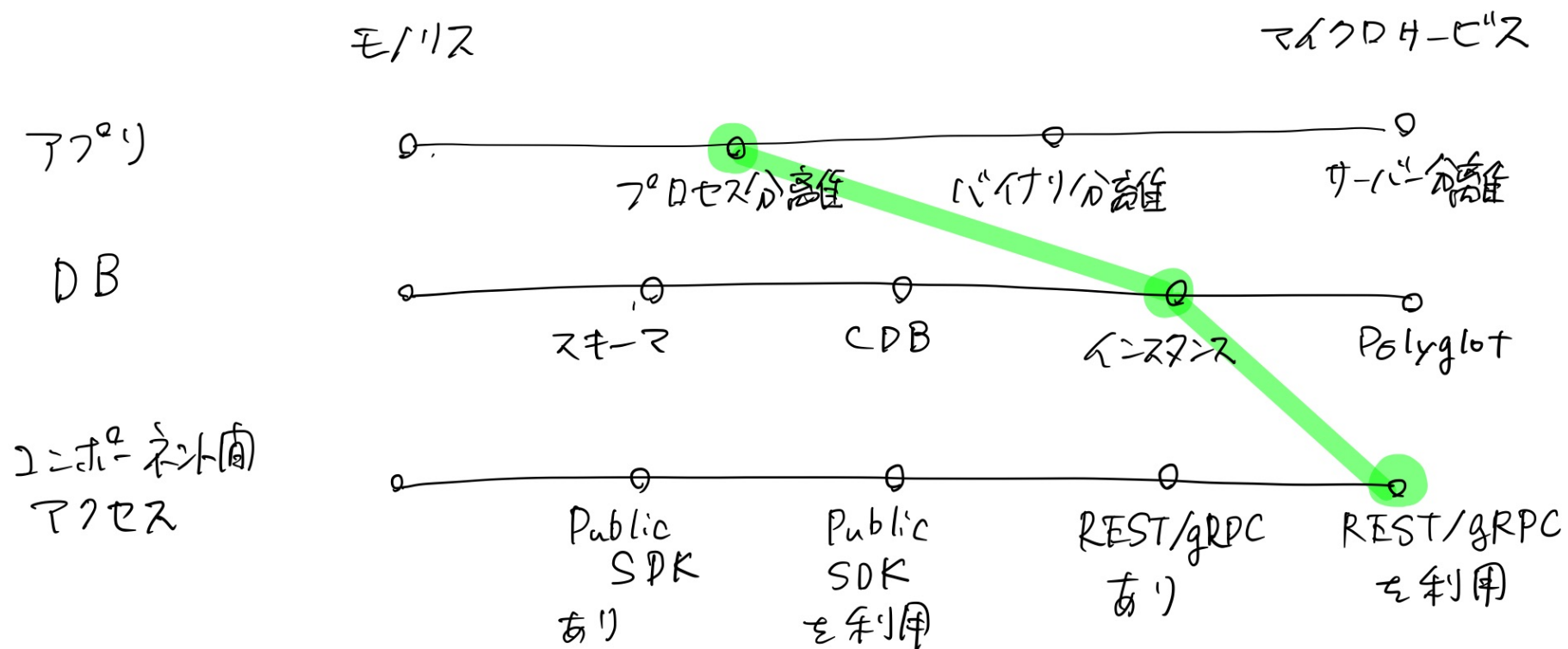
モジュラモノリス

認証・認可 / フロントエンド: (不明)



モノマイクロリス

認証・認可 / フロントエンド: (不明)



ケーススタディ

他の例を題材にモノリシック／マイクロサービスらしさを考えていきます

- Fuji Xerox ArcSuite Engineering
 - 「『ドキュメント』と『コミュニケーション』を一元管理。製造業の情報活用を促進し、品質および生産性の向上に寄与します。」（[公式サイト](#)より）
 - 企業内システムとして利用されるWebアプリケーション
 - お客様のDCやサーバールームに設置
 - Webアプリケーションとして機能を提供
 - 基本的にお客様の社員がイントラネットからアクセス

ArcSuiteを構成する機能コンポーネント 1/2

文書管理システムの機能を中核に、複数のグループウェア機能で構成

- Document Repository
 - 文書管理のコアとなるエンジン
 - 他コンポーネントに文書管理のためのパブリックAPIを提供
- ドキュメントスペース
 - いわゆる文書管理アプリ。 c.f.,) WebCenter Contentのあの画面
- コラボスペース
 - いわゆる掲示板。Slackのチャンネル、スレッド的な要素をもつ
 - 添付ファイルの処理などでDocument Repositoryと連携

ArcSuiteを構成する機能コンポーネント 2/2

文書管理システムの機能を中核に、複数のグループウェア機能で構成

- ワークフロー
 - ワークフローエンジン
 - 承認系の文書の管理などでDocument Repositoryと連携
- 共通ログイン／ユーザー管理
 - 上記複数の機能に対するSSOを提供

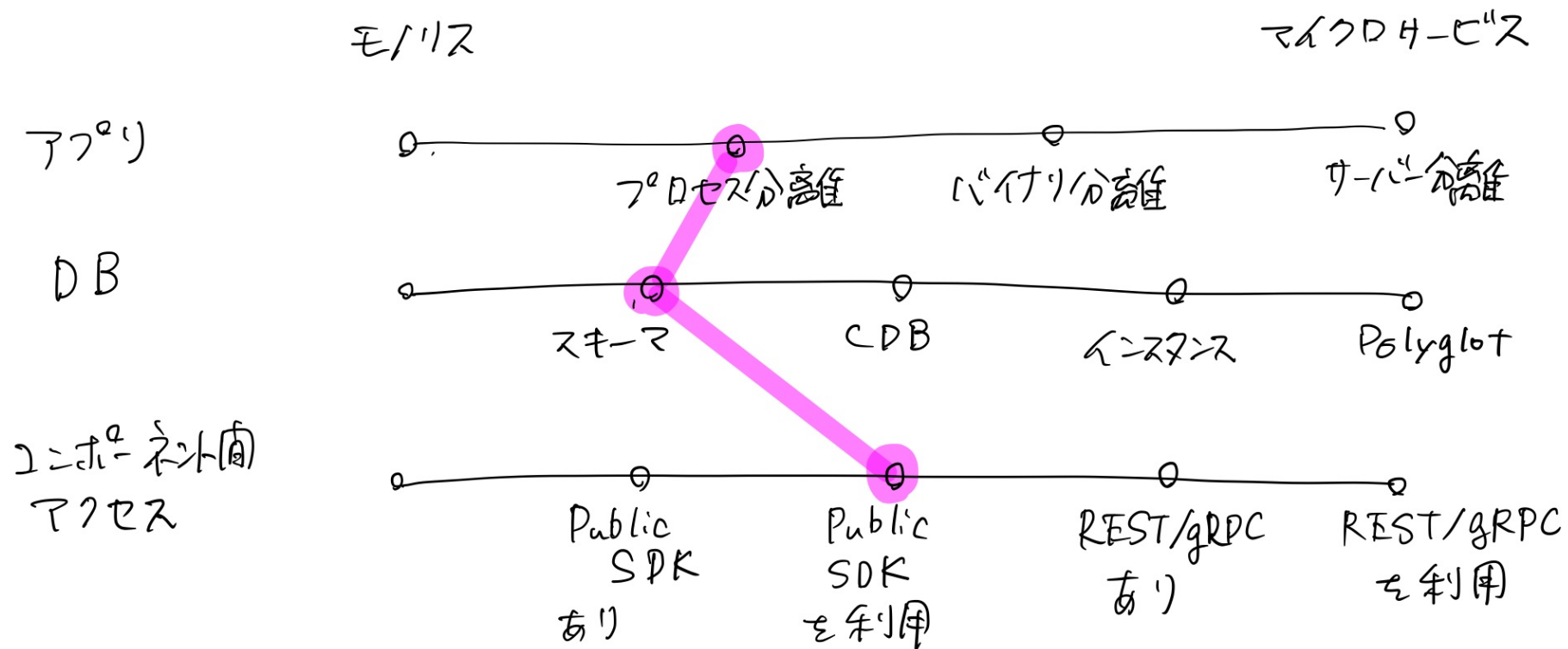
(その他いろいろな機能がありますが割愛)

ArcSuiteのアーキテクチャ

(Confidentialのためホワイトボードに書きます)

ArcSuite

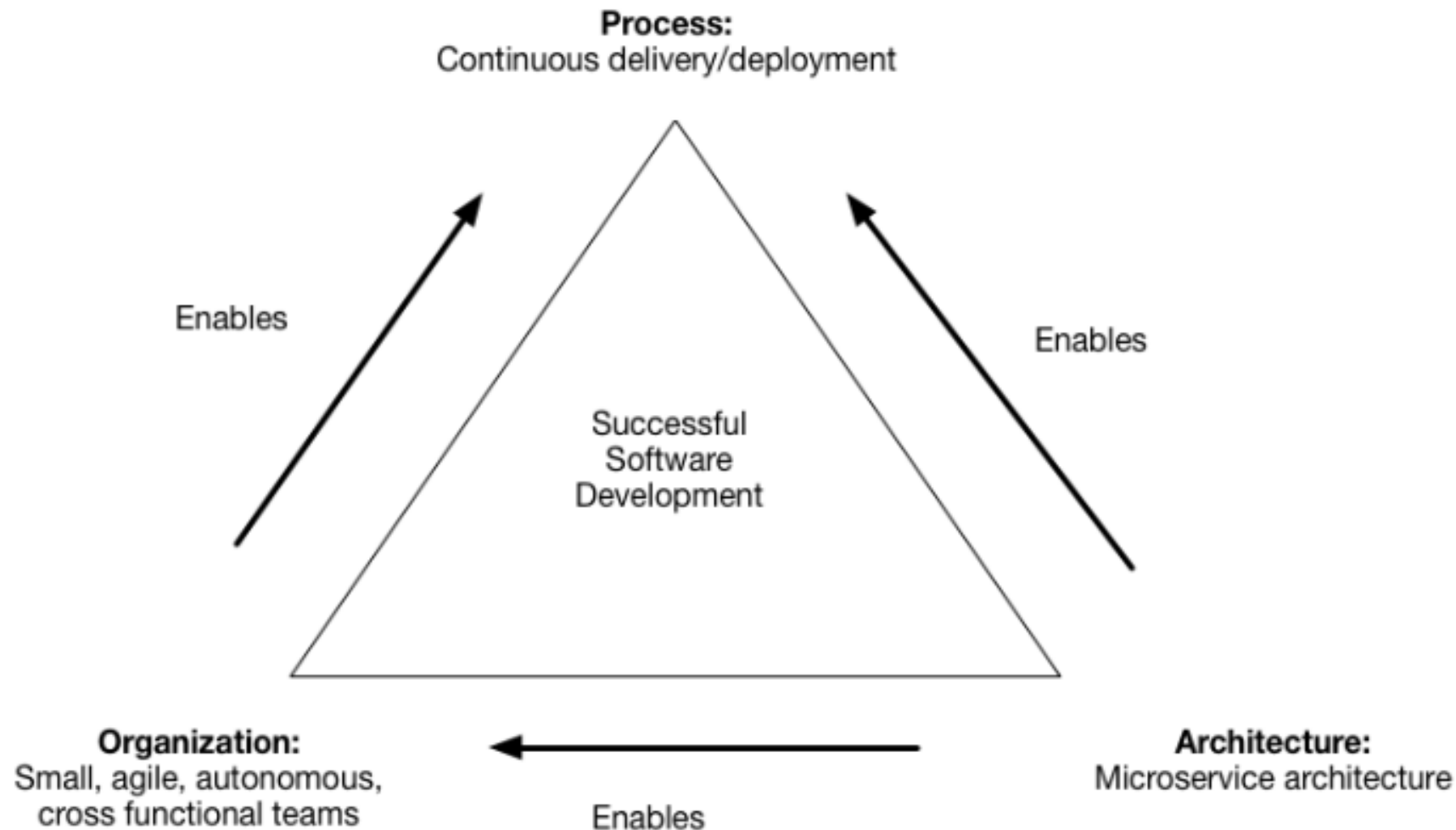
認証・認可 / フロントエンド: SSO



目次

- イン트로ダクション
- モノリシックらしさ、マイクロサービスらしさの指標__
- ケーススタディ - とあるモジュラモノリスなアプリケーションの話
- 残された議論

組織論、CD、コードベース



参考文献

- <https://www.youtube.com/watch?v=ISYKx8sa53g>
- <https://engineering.shopify.com/blogs/engineering/deconstructing-monolith-designing-software-maximizes-developer-productivity>
- <https://blog.kymmt.com/entry/the-modular-monolith-rails-architecture>
- <https://blog.eventuate.io/2017/01/04/the-microservice-architecture-is-a-means-to-an-end-enabling-continuous-deliverydeployment/>
- <https://speakerdeck.com/polar3130/rethink-modern-architectures-with-monolith>
- https://speakerdeck.com/nari_ex/design-and-operation-of-operational-engineer-organization?slide=19