

GKEを用いたレガシーシステムからの リプレース事例

2019/3/26

自己紹介・作った物

1. プロジェクト概要
2. GKE利用までの経緯
3. 取り組む上での課題(組織面)
4. 取り組む上での課題(開発面)
5. 効果
6. 取り組んでみて

富士フイルムソフトウェア 所属

主にプリントビジネス関連開発を入社以来担当。
フロントエンド・バックエンド・生産現場と、
入口から出口まで携わる。

最近ではプロジェクトリーダー業務の中でも
主に開発プロセス改善に取り組む。

FUJIFILM Prints & Gifts

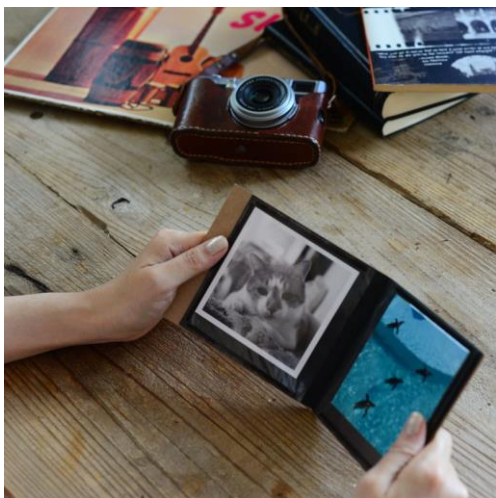
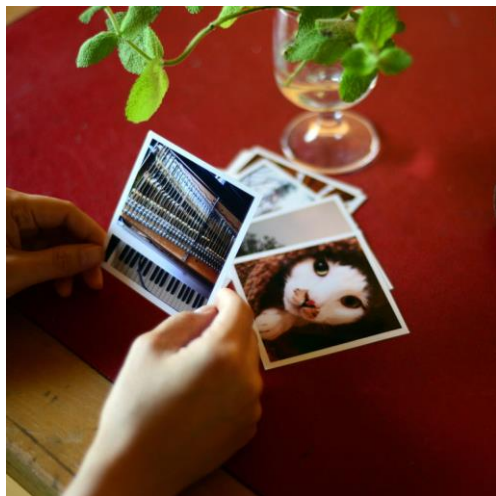
富士フィルムのプリントや写真雑貨を注文出来るサービス

<https://pg-ja.fujifilm.com/home>

このバックエンドの注文管理システム(以降OMS)を開発

The screenshot shows the homepage of the FUJIFILM Prints & Gifts website. The header includes the site name, navigation icons for '再検索' (Search), 'ログイン' (Login), '会員登録' (Member Registration), and 'カート' (Cart). Below the header is a search bar with the text 'キーワードを検索' and a 'ヘルプ' (Help) button. The main content area features three promotional banners: 1. A 20th anniversary banner for 'FUJIFILM Prints & Gifts' with the text 'Renewal Open since 1998 20 anniversary' and 'thank you!!'. 2. A 'WALL DECOR' banner with the text 'FUJIFILM WALL DECOR -時を飾ろう-' and an image of a woman in a room. 3. A banner for 'For GIFT' showing a hand writing on a photo card, with the text '写真プリントと写真雑貨が一緒に購入できるようになりました。' (You can now purchase photo prints and photo goods together). At the bottom, there are two news items: '2018年10月23日新サービス「プレミアムプリント」スタートしました。' and '2018年10月23日富士フィルムの「フォトカレンダー」2019年1月始まりスタートいたしました。'

FUJIFILM Prints & Gifts商品一例



ましかくプリント

スタンダードな写真プリント

- かわいくおさまる正方形
- フチのパターンが選べる
- 雑貨との組み合わせ◎

FUJIFILM Prints & Gifts商品一例



WALL DECOR

- ・ 壁掛けタイプの大判印刷
- ・ インテリアに最適

FUJIFILM Prints & Gifts商品一例

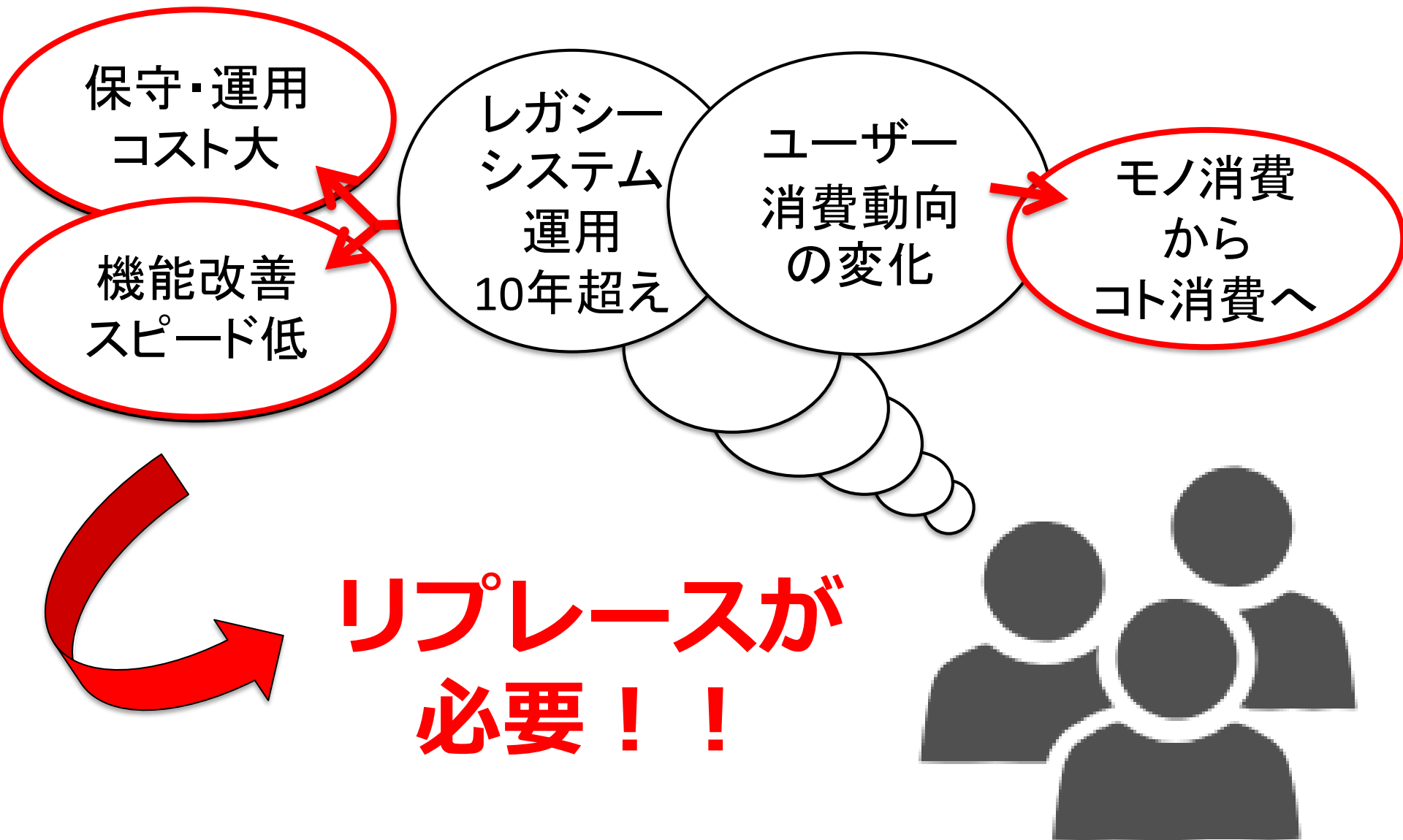


写真雑貨

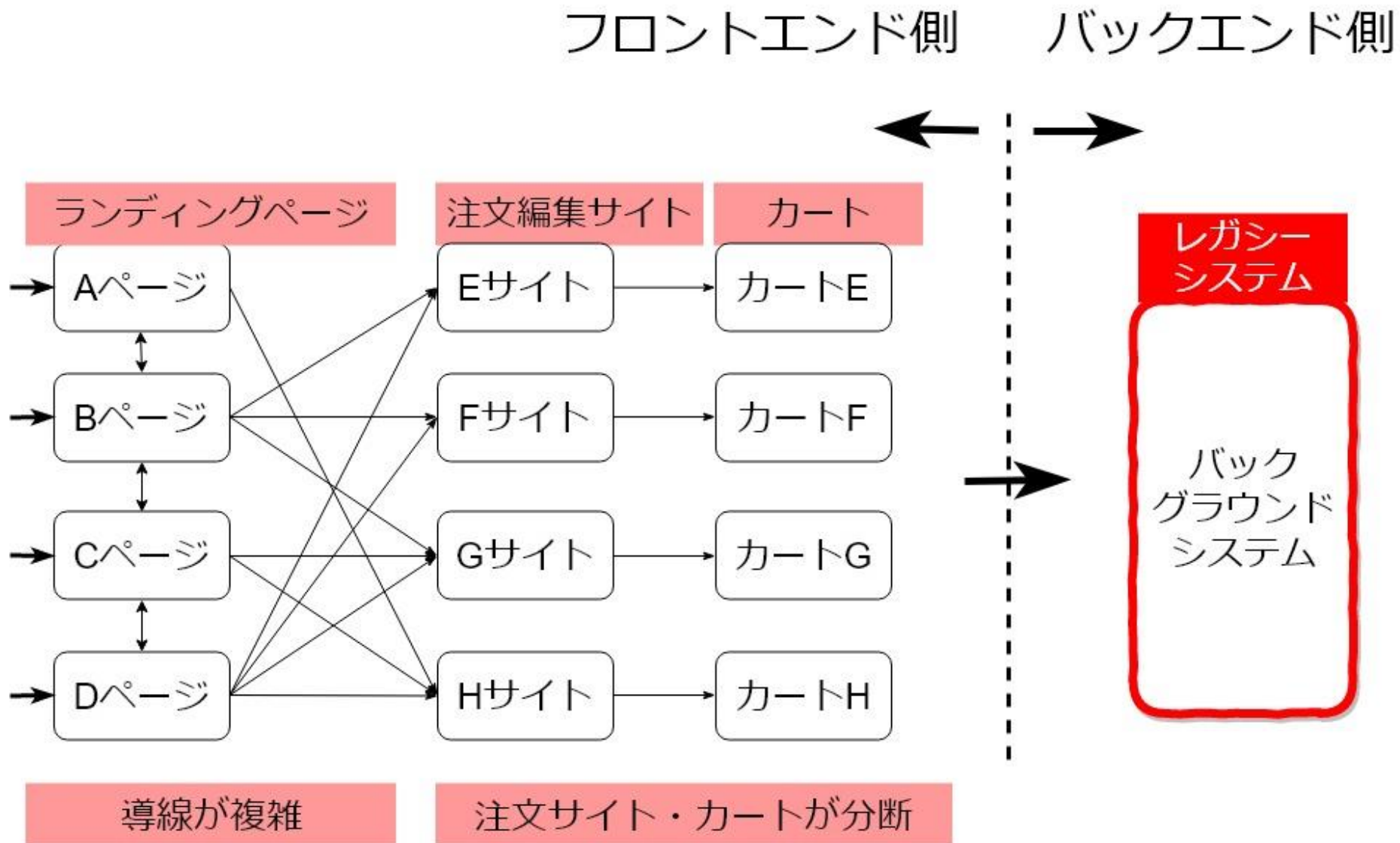
- ポケットアルバム
- フォトフレーム
- ギフトタグ
- etc...



1.プロジェクト概要

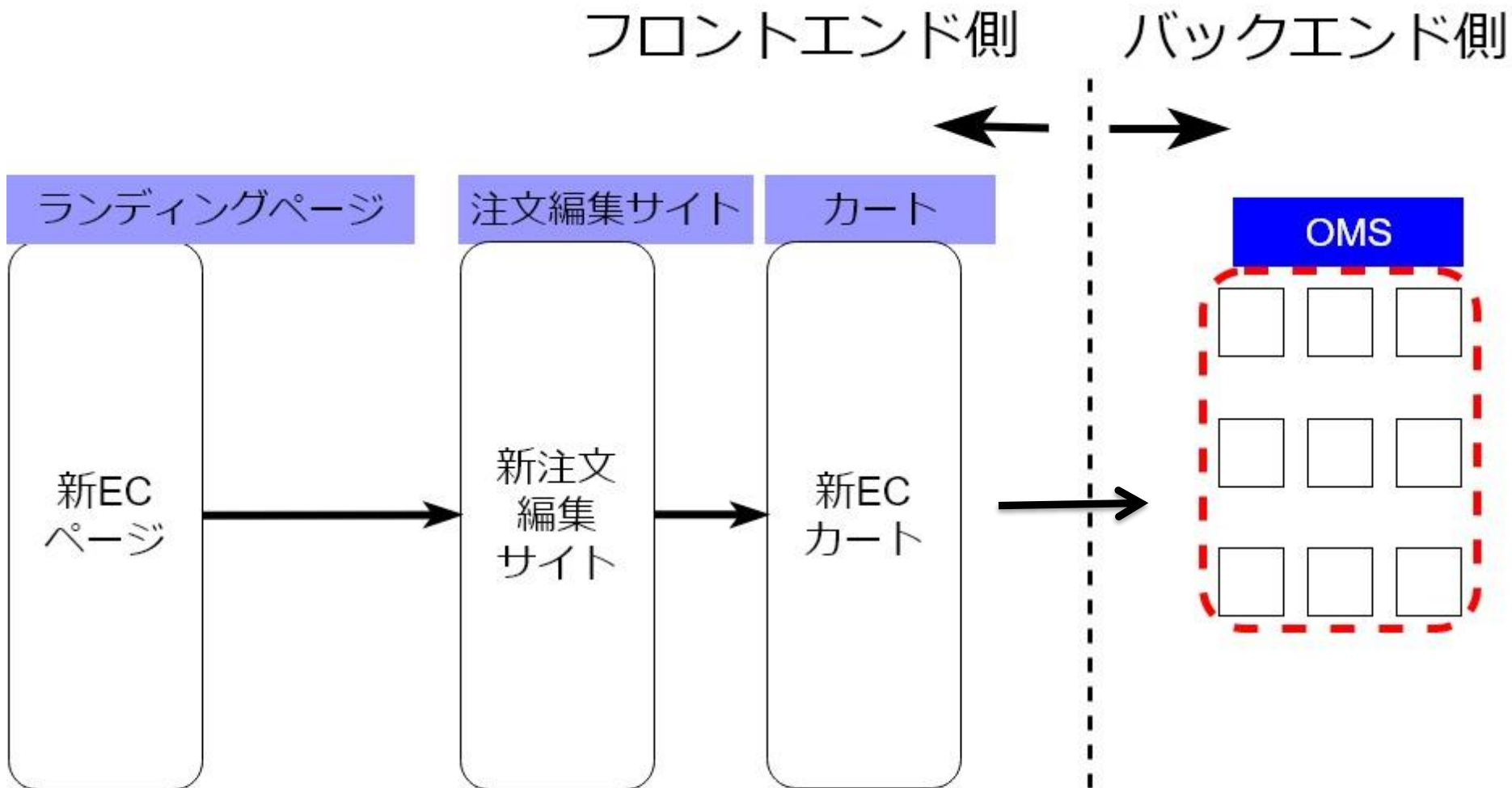


1.プロジェクト概要



新旧システム比較(旧)

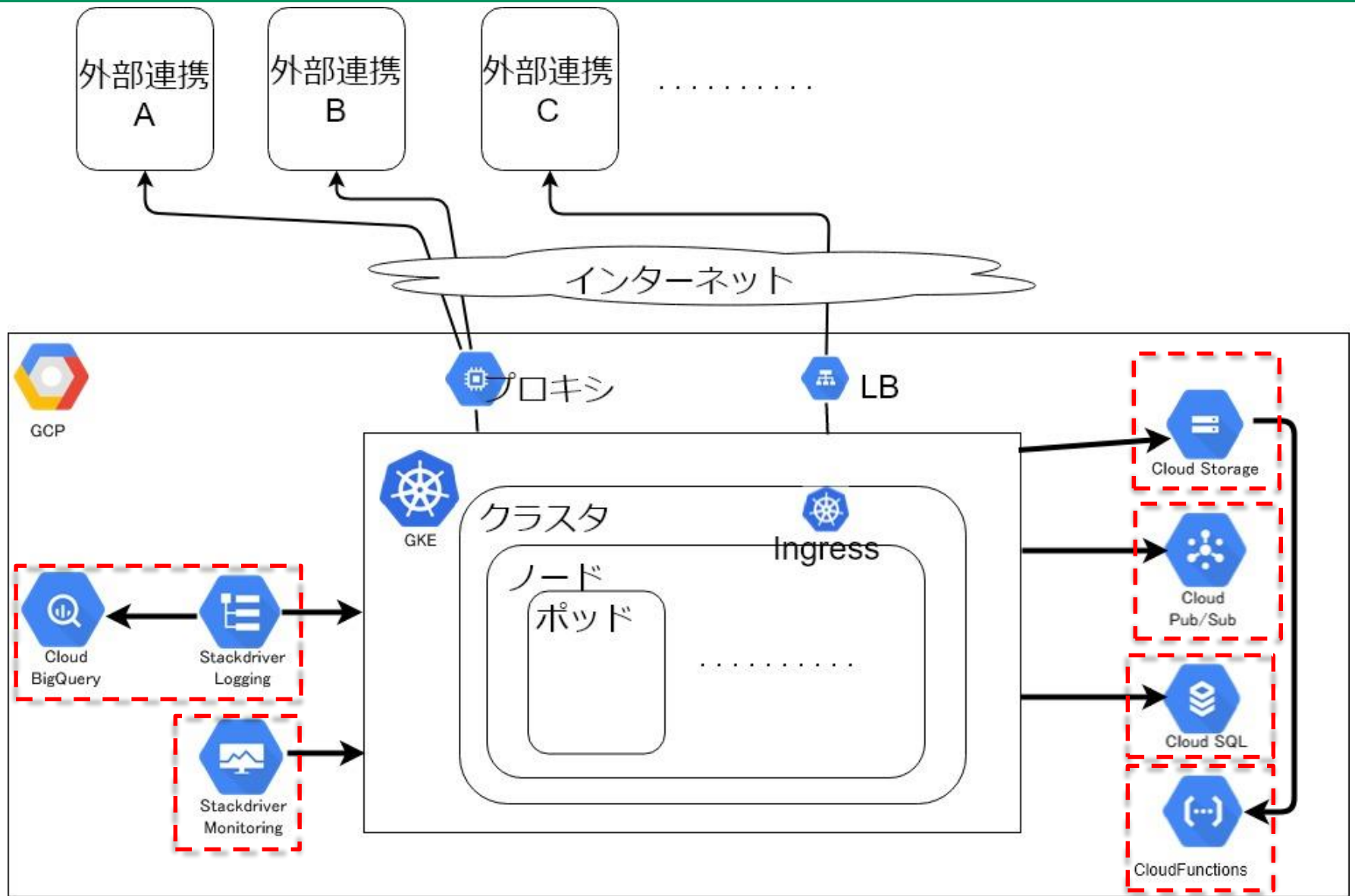
1.プロジェクト概要



導線シンプル化、統合マーケティング

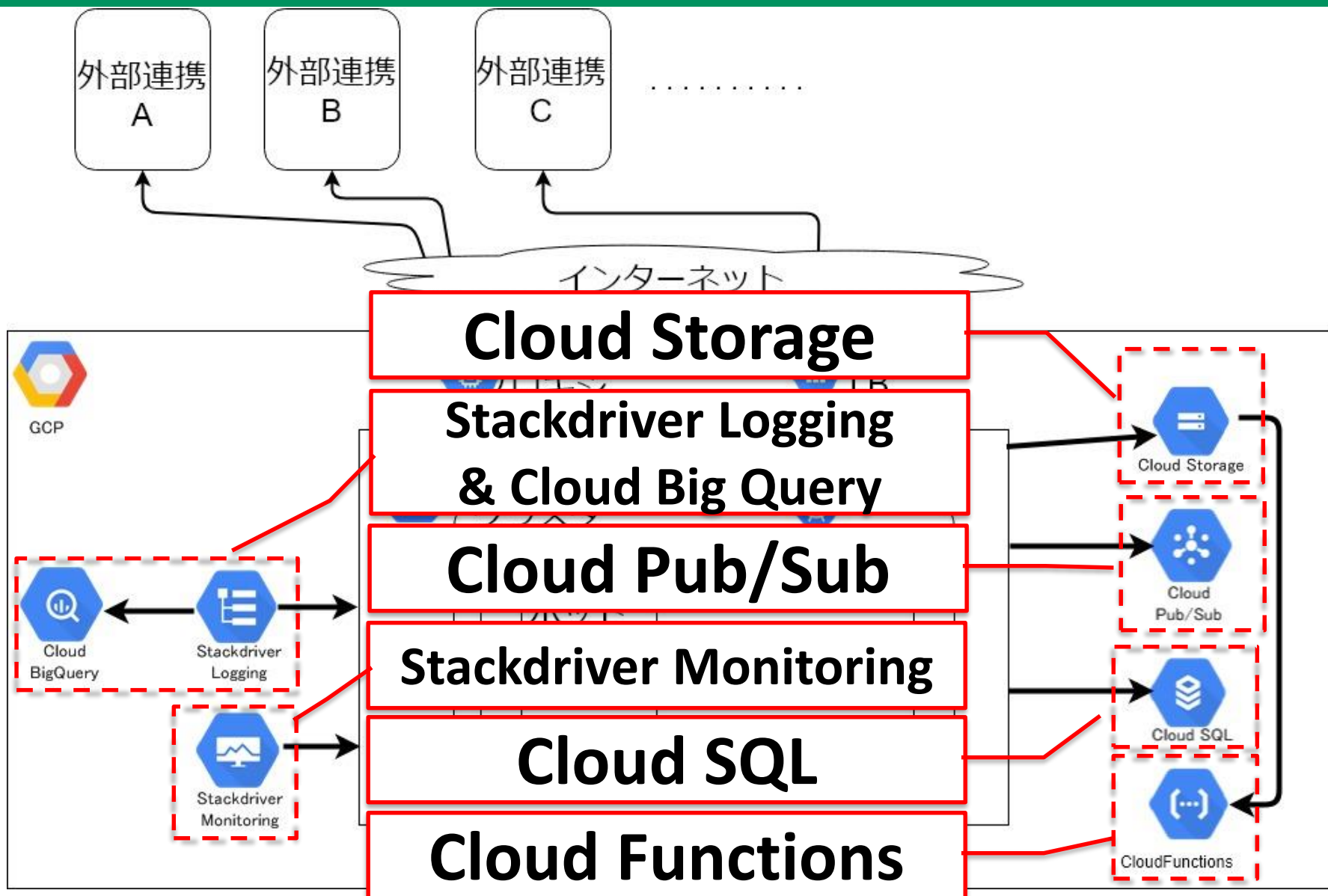
新旧システム比較(新)

1.プロジェクト概要



OMS概略

1.プロジェクト概要



目次

自己紹介・作った物

1. プロジェクト概要

2. GKE利用までの経緯

3. 取り組む上での課題(組織面)

4. 取り組む上での課題(開発面)

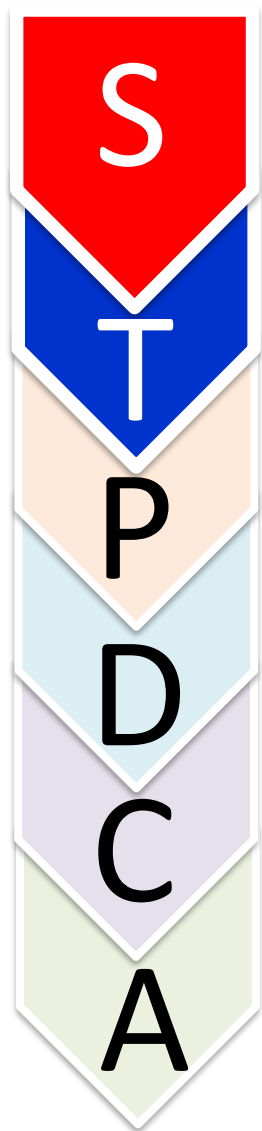
5. 効果

6. 取り組んでみて

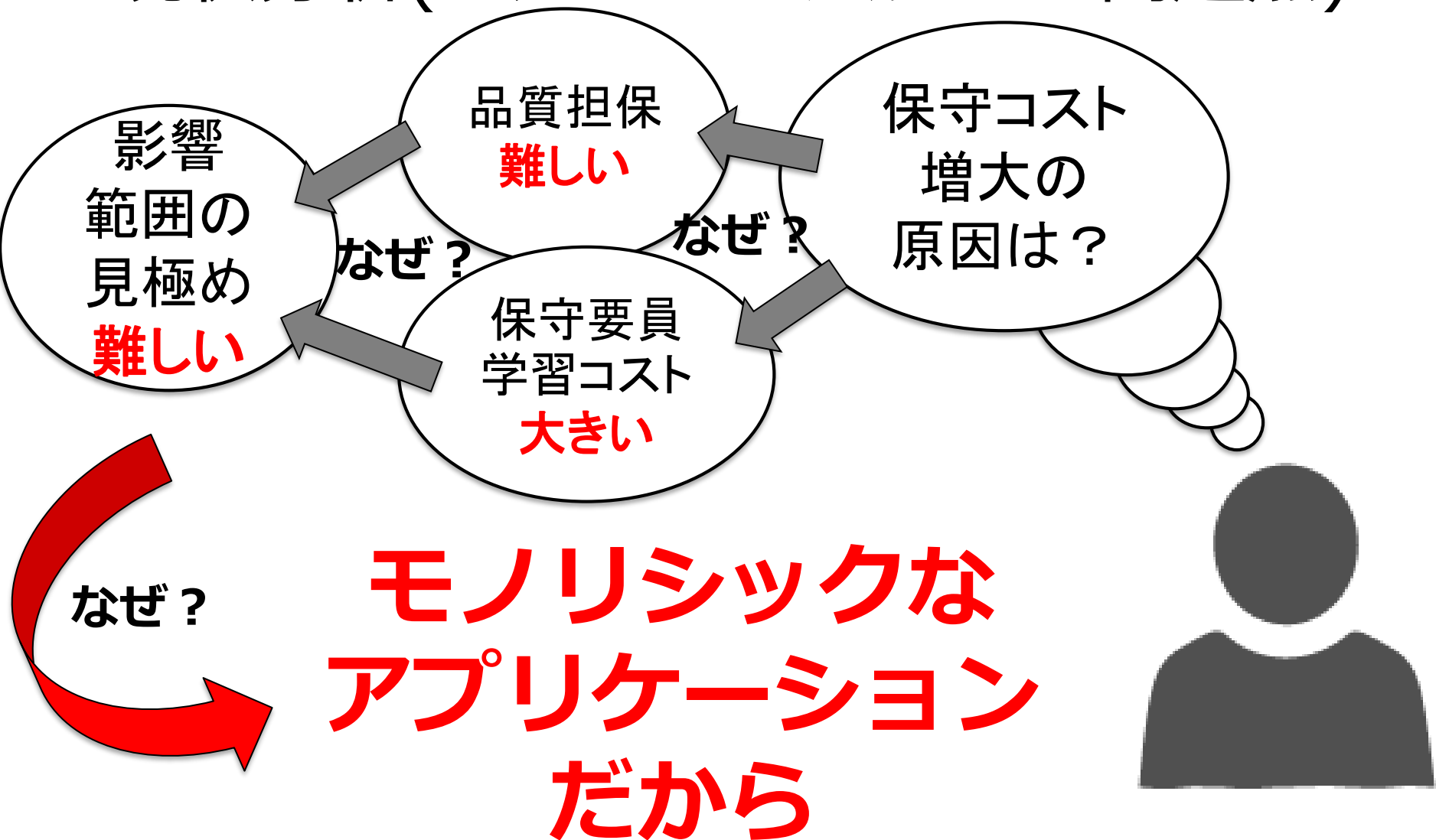
富士フイルムの文化

特に[S]eeと[T]hinkを重要視

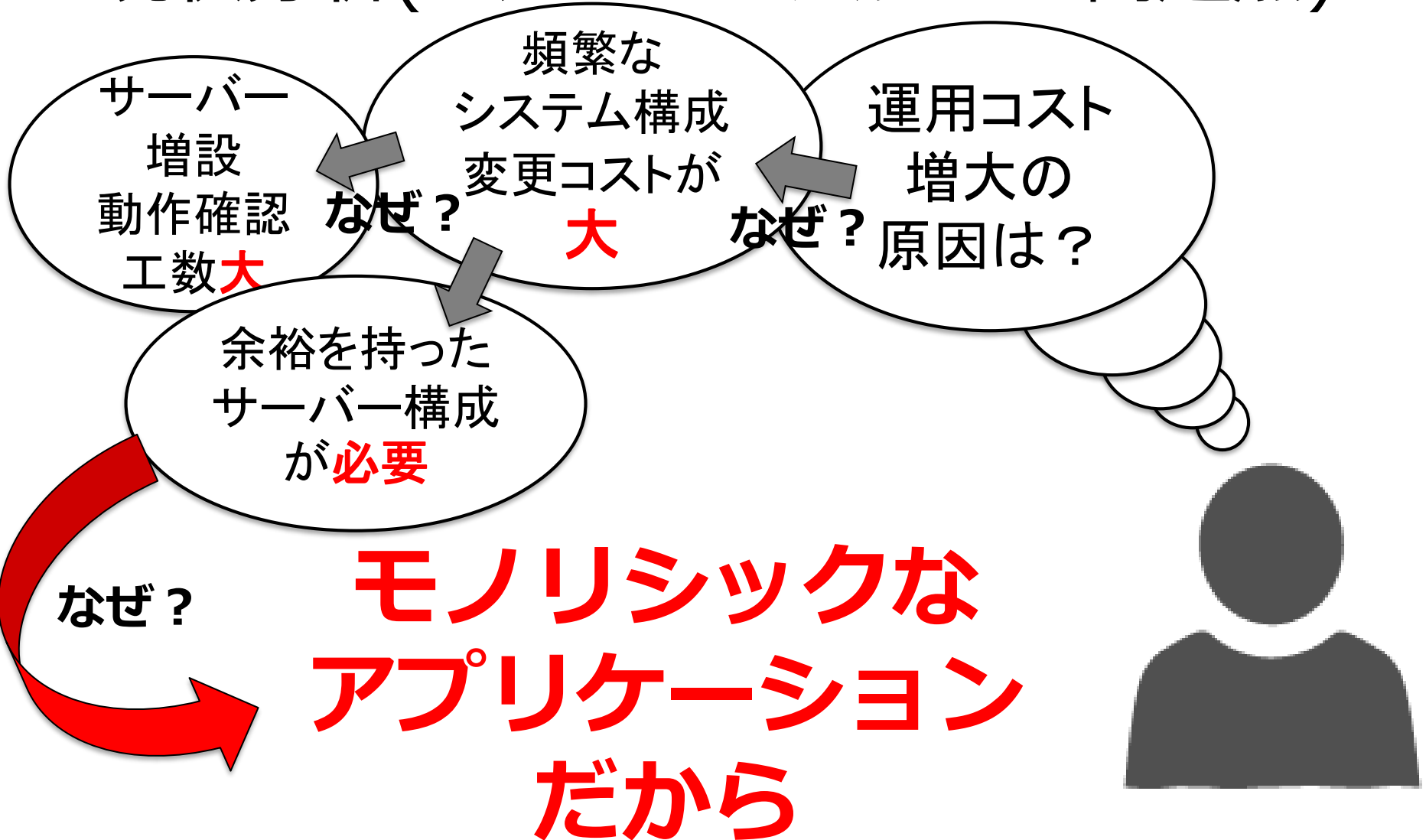
現状分析をしっかりと行い
目的を明確にする。



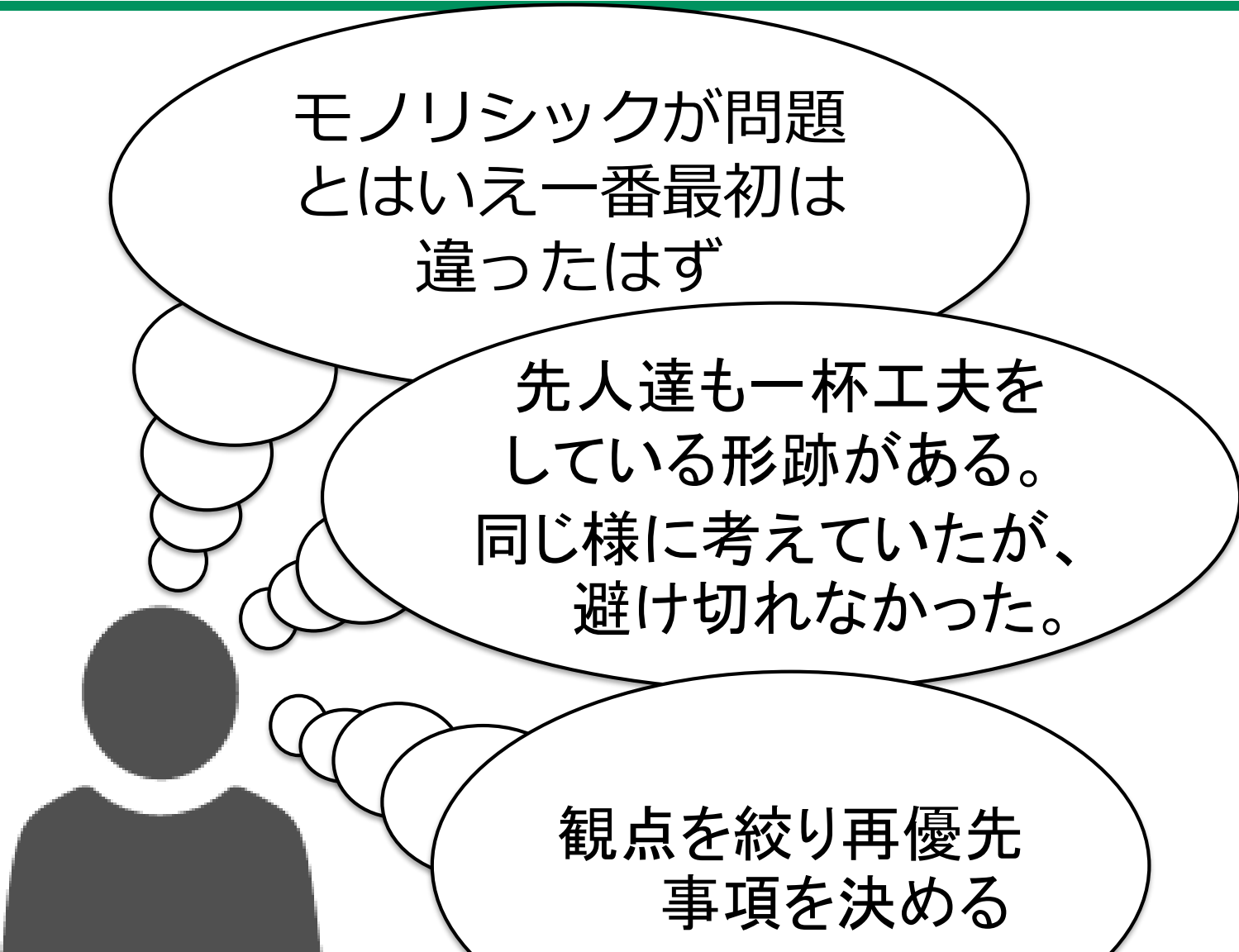
■ 現状分析(レガシーシステムの問題点)



■ 現状分析(レガシーシステムの問題点)



2.GKE利用までの経緯



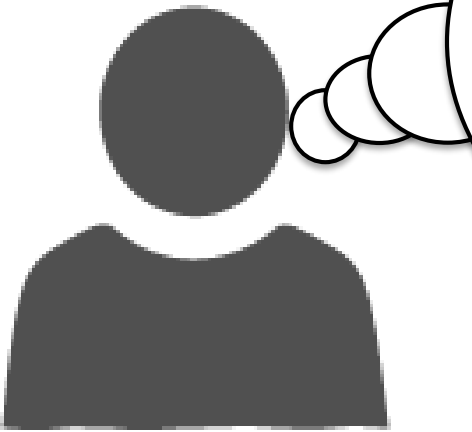
モノリシックが問題
とはいえ一番最初は
違ったはず

先人達も一杯工夫を
している形跡がある。
同じ様に考えていたが、
避け切れなかった。

観点を絞り再優先
事項を決める

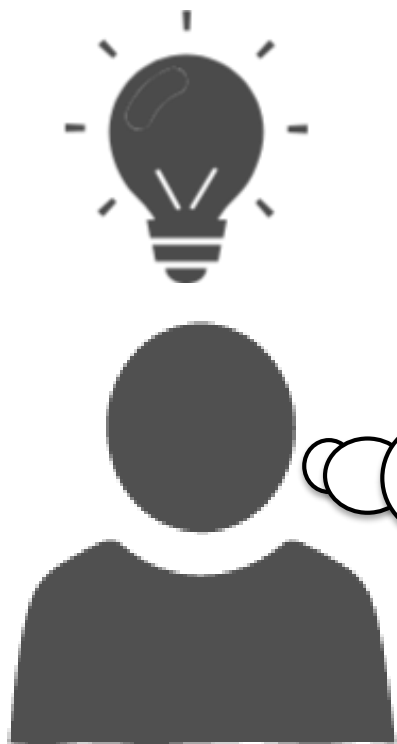
特に苦勞しているのは…

- ・ 季節イベント
- ・ キャンペーン



負荷量の変動に対して
システムが追従しにくい。
増やした分だけ管理対象は
増える。多機能な大きい
固まりを増やしていくので
確認の手間も増える。

負荷量変動への対応を
最優先にする！



スケラビリティ
確保しやすい
仕組みを最優
先にする！

コンテナを使う！

他の問題点もある程度解消出来るそう

保守面を意識すれば、オーケストレーションツールは使いたい。
でもまだ課題がある
大きい物は以下2つ



何が
標準か

自分達で
運用
出来るか

何が
標準か

- 流行度
- 覇権争い
- 仕様策定中
etc ...



自分達で
運用
出来るか

- 使いこなせない
と意味がない
- 運用環境に耐え
うるレベルか



調査



Docker、Kubernetesを取り巻く環境の変化 (様々なニュースリリースからピックアップ)

2013年

3月：Docker社がdockerをオープンソースソフトウェアとしてリリース。

2014年

5月：Googleは既に全てのソフトウェアをコンテナに乗せており、毎週20億個ものコンテナを起動している事を発表。

Googleでは既に当たり前の技術として利用されていた。

6月：docker v1.0をリリース。

GoogleはKubernetesをオープンソースとして公開。

12月：Dockerが独自のオーケストレーションツールのDocker Swarmを発表

2015年

6月：コンテナ実装のあり方について覇権争いしていたDockerとCoreOSがMS・Google・RedHat・Vmware・Amazonと共同でコンテナの標準団体「Open Container Initiative(以下OCI)」の発足を発表。それまでベンダが独自に実装していたコンテナが標準化に舵を切る。

7月：Googleが主導して実装を進めてきたKubernetesがv1.0に。同時にCloud Native Computing Foundation(以下CNCF)を設立し、開発主体がこの団体へ移動。

2016年

6月：dockerにコンテナオーケストレーション機能のDocker Swarmを内蔵。SwarmとKubernetesの主導権争いが表面化。

2017年

7月：OCIがコンテナランタイムとコンテナイメージの標準化作業完了、OCI1.0として発表。

10月：dockerがKubertnetesの統合とサポートを発表。これによりコンテナオーケストレーションの事実上の標準がKubernetesになった。

10月：MSがAzureのKubernetesのマネージドサービス「Azrue Container Service(AKS)」を発表。

Google Cloudはこれまで提供してきたKubernetesベースのGoogle Container Engineを「Google Kubernetes Engine(GKE)」にサービス名変更。

11月：Amazonも「Amazon Elastic Kubernetes Service(Amazon EKS)」を発表。

コンテナランタイムとKubernetesの間の通信インターフェースCRIを定義。

これによりコンテナランタイムとコンテナイメージの標準は「OCI」、コンテナランタイムとKubernetesとの間のAPIは「CRI」で標準化された。

Docker、Kubernetesを取り巻く環境の変化 (様々なニュースリリースからピックアップ)

2013年

3月：Docker社がdockerをオープンソースソフトウェアとしてリリース。

2014年

5月：Googleは既に全てのソフトウェアをコンテナに乗せており、毎週20億個ものコンテナを起動している事を発表。
Googleでは既に当たり前の技術として利用されていた。

6月：docker v1.0をリリース。

GoogleはKubernetesをオープンソースとして公開。

12月：Dockerが独自のオーケストレーションツールのDocker Swarmを発表

2015年

6月：コ

7月：C

2016年

6月：C

2017年

7月：C

10月：

10月：

11月：

これに

2013年3月：Docker社がdockerをオープンソースソフトウェアとしてリリース。

を切る。

た。

Docker、Kubernetesを取り巻く環境の変化 (様々なニュースリリースからピックアップ)

2013年

3月：Docker社がdockerをオープンソースソフトウェアとしてリリース。

2014年

5月：Googleは既に全てのソフトウェアをコンテナに乗せており、毎週20億個ものコンテナを起動している事を発表。

Googleでは既に当たり前の技術として利用されていた。

6月：docker v1.0をリリース。

GoogleはKubernetesをオープンソースとして公開。

12月：Dockerが独自のオーケストレーションツールのDocker Swarmを発表

2015年

6月：コンテナ実装のあり方について覇権争いしていたDockerとCoreOSがMS・Google・RedHat・Vmware・Amazonと共同でコンテナの標準団体「Open Container Initiative(以下OCI)」の発足を発表。それまでベンダが独自に実装していたコンテナが標準化に舵を切る。

7月：Googleが主導して実装を標準化する「Container Engine」を同時にGoogle Cloud Platform (以下GCP)で提供し、

公開

2016年

6月：C

2017年

7月：C

10月：

10月：

11月：

これに

**2014年5月：Googleは既に全てのソフトウェアを
コンテナに乗せており、毎週20億個ものコンテナ
を起動している事を発表。**

**→Googleでは既に当たり前の技術として
利用されていた。**

Docker、Kubernetesを取り巻く環境の変化 (様々なニュースリリースからピックアップ)

2013年

3月：Docker社がdockerをオープンソースソフトウェアとしてリリース。

2014年

5月：Googleは既に全てのソフトウェアをコンテナに乗せており、毎週20億個ものコンテナを起動している事を発表。

Googleでは既に当たり前の技術として利用されていた。

6月：docker v1.0をリリース。

GoogleはKubernetesをオープンソースとして公開。

12月：Dockerが独自のオーケストレーションツールDocker Swarmを発表

2015年

6月：コンテナ実装のあり方について覇権争いしていたDockerとCoreOSがMS・Google・RedHat・Vmware・Amazonと共同でコンテナの標準団体「Open Container Initiative(以下OCI)」の発足を発表。それまでベンダが独自に実装していたコンテナが標準化に舵を切る。

7月：Googleが主導して実装を標準化するが、CoreOSは同時にCloud Native Computing Foundation(以下CNCF)を設立し

開

2016年

6月：C

2017年

7月：C

10月：

10月：

11月：

これに

2014年6月：

▪ **docker v1.0をリリース**

▪ **GoogleはKubernetesをオープンソースとして公開。**

Docker、Kubernetesを取り巻く環境の変化 (様々なニュースリリースからピックアップ)

2013年

3月：Docker社がdockerをオープンソースソフトウェアとしてリリース。

2014年

5月：Googleは既に全てのソフトウェアをコンテナに乗せており、毎週20億個ものコンテナを起動している事を発表。
Googleでは既に当たり前の技術として利用されていた。

6月：docker v1.0をリリース。

GoogleはKubernetesをオープンソースとして公開。

12月：Dockerが独自のオーケストレーションツールのDocker Swarmを発表

2015年

6月：コンテナ実装のあり方について覇権争いしていたDockerとCoreOSがMS・Google・RedHat・Vmware・Amazonと共同でコンテナの標準団体「Open Container Initiative(以下OCI)」の発足を発表。それまでベンダが独自に実装していたコンテナが標準化に舵を切る。

7月：Googleが主導して実装を標準化するOCIに、同時にCloud Native Computing Foundation (以下CNCF)を設立し、

開

2016年

6月：C

2017年

7月：C

10月：

10月：

11月：

これに

2014年12月：

- **Dockerが独自のオーケストレーションツールの
Docker Swarmを発表**

→オーケストレーションツールの対抗馬に

Docker、Kubernetesを取り巻く環境の変化 (様々なニュースリリースからピックアップ)

2013年

3月：Docker社がdockerをオープンソースソフトウェアとしてリリース。

2014年

5月：Googleは既に全てのソフトウェアをコンテナに乗せており、毎週20億個ものコンテナを起動している事を発表。
Googleでは既に当たり前の技術として利用されていた。

6月：docker v1.0をリリース。

GoogleはKubernetesをオープンソースとして公開。

12月：Dockerが独自のオーケストレーションツールのDocker Swarmを発表

2015年

6月：コンテナ実装のあり方について覇権争いしていたDockerとCoreOSがMS・Google・RedHat・Vmware・Amazonと共同で
コンテナの標準団体「Open Container Initiative(以下OCI)」の発足を発表。それまでベンダが独自に実装していたコンテナが標準化に舵を切る。

7月：Googleが主導して実装を進めてきたKubernetesがv1.0に。同時にCloud Native Computing Foundation(以下CNCF)を設立し、
開発主体がこの団体へ移動。

2016年

6月：dock

2017年

7月：C

10月：

10月：

11月：

これに

2015年6月：

- ・コンテナ実装のあり方について覇権争いしていたDockerとCoreOSがMS・Google・RedHat・Vmware・Amazonと共同でコンテナの標準団体「Open Container Initiative(以下OCI)」の発足を発表
→標準化に舵を切る

Docker、Kubernetesを取り巻く環境の変化 (様々なニュースリリースからピックアップ)

2013年

3月：Docker社がdockerをオープンソースソフトウェアとしてリリース。

2014年

5月：Googleは既に全てのソフトウェアをコンテナに乗せており、毎週20億個ものコンテナを起動している事を発表。

Googleでは既に当たり前の技術として利用されていた。

6月：docker v1.0をリリース。

GoogleはKubernetesをオープンソースとして公開。

12月：Dockerが独自のオーケストレーションツールのDocker Swarmを発表

2015年

6月：コンテナ実装のあり方について覇権争いしていたDockerとCoreOSがMS・Google・RedHat・Vmware・Amazonと共同でコンテナの標準団体「Open Container Initiative(以下OCI)」の発足を発表。それまでベンダが独自に実装していたコンテナが標準化に舵を切る。

7月：Googleが主導して実装を進めてきたKubernetesがv1.0に。同時にCloud Native Computing Foundation(以下CNCF)を設立し、開発主体がこの団体へ移動。

2016年

6月：dockerにコンテナオーケストレーション機能のDocker Swarmを内蔵。SwarmとKubernetesの主導権争いが表面化。

2017年

7月：OCI

10月：

10月：

11月：

これに

2015年7月：

- Kubernetesがv1.0に。
- Cloud Native Computing Foundation(以下CNCF)を設立し、開発主体がこの団体へ移動。

**→オープンテクノロジー
ベンダーロックイン無し**

Docker、Kubernetesを取り巻く環境の変化 (様々なニュースリリースからピックアップ)

2013年

3月：Docker社がdockerをオープンソースソフトウェアとしてリリース。

2014年

5月：Googleは既に全てのソフトウェアをコンテナに乗せており、毎週20億個ものコンテナを起動している事を発表。

Googleでは既に当たり前の技術として利用されていた。

6月：docker v1.0をリリース。

GoogleはKubernetesをオープンソースとして公開。

12月：Dockerが独自のオーケストレーションツールのDocker Swarmを発表

2015年

6月：コンテナ実装のあり方について覇権争いしていたDockerとCoreOSがMS・Google・RedHat・Vmware・Amazonと共同で

コンテナの標準団体「Open Container Initiative(以下OCI)」の発足を発表。それまでベンダが独自に実装していたコンテナが標準化に舵を切る。

7月：Googleが主導して実装を進めてきたKubernetesがv1.0に。同時にCloud Native Computing Foundation(以下CNCF)を設立し、

開発主体がこの団体へ移動。

2016年

6月：dockerにコンテナオーケストレーション機能のDocker Swarmを内蔵。SwarmとKubernetesの主導権争いが表面化。

2017年

7月：OCI

10月：

10月

11月

これに

2016年6月：

- **dockerにコンテナオーケストレーション機能の
Docker Swarmを内蔵。
SwarmとKubernetesの主導権争いが表面化。**

→争い中

Docker、Kubernetesを取り巻く環境の変化 (様々なニュースリリースからピックアップ)

2017年7月：

- OCIがコンテナランタイムとコンテナイメージの標準化作業完了、OCI1.0として発表。

→標準化！

舵を切る。

2013年
3月：

2014年
5月

6月：

12月

2015年

6月：

7月：

2016年

6月：

2017年

7月：OCIがコンテナランタイムとコンテナイメージの標準化作業完了、OCI1.0として発表。

10月：dockerがKubernetesの統合とサポートを発表。これによりコンテナオーケストレーションの事実上の標準がKubernetesになった。

10月：MSがAzureのKubernetesのマネージドサービス「Azrue Container Service(AKS)」を発表。

Google Cloudはこれまで提供してきたKubernetesベースのGoogle Container Engineを「Google Kubernetes Engine(GKE)」にサービス名変更。

11月：Amazonも「Amazon Elastic Kubernetes Service(Amazon EKS)」を発表。

コンテナランタイムとKubernetesの間の通信インターフェースCRIを定義。

これによりコンテナランタイムとコンテナイメージの標準は「OCI」、コンテナランタイムとKubernetesとの間のAPIは「CRI」で標準化された。

Docker、Kubernetesを取り巻く環境の変化 (様々なニュースリリースからピックアップ)

2017年10月:

▪ dockerがKubertnetesの統合とサポートを発表。

→ Kubernetesがコンテナオーケストレーション
ツールの事実上の標準に

を切る。

10月: MSがAzureのKubernetesのマネージドサービス「Azrue Container Service(AKS)」を発表。

Google Cloudはこれまで提供してきたKubernetesベースのGoogle Container Engineを「Google Kubernetes Engine(GKE)」にサービス名変更。

11月: Amazonも「Amazon Elastic Kubernetes Service(Amazon EKS)」を発表。

コンテナランタイムとKubernetesの間の通信インターフェースCRIを定義。

これによりコンテナランタイムとコンテナイメージの標準は「OCI」、コンテナランタイムとKubernetesとの間のAPIは「CRI」で標準化された。

Docker、Kubernetesを取り巻く環境の変化 (様々なニュースリリースからピックアップ)

2017年10月+11月:

- MSがAzureのKubernetesのマネージドサービス「Azrue Container Service(AKS)」を発表。
- Googleはこれまで提供してきたKubernetesベースのGoogle Container Engineを「Google Kubernetes Engine(GKE)」にサービス名変更
- Amazonも「Amazon Elastic Kubernetes Service(Amazon EKS)」を発表

→ 大手ベンダのサポートが手厚くなってきた。

を切る。

Docker、Kubernetesを取り巻く環境の変化 (様々なニュースリリースからピックアップ)

2013年
3月 : Do
2014年
5月 : G
G
6月 : dc
G
12月 : D
2015年
6月 : コ
7月 : G
開
2016年
6月 : dc
2017年
7月 : O
10月 : d
10月 : M
G
セ
11月 : A
これによ

2017年11月 :

- ・コンテナランタイムとKubernetesの間の通信
インターフェースCRIを定義

→標準化！

を切る。

Kubernetesがコンテナオーケストレーションツールのデファクトスタンダードになった。

→**規格争いによる技術の陳腐化懸念が後退。**

主要クラウドベンダはKubernetesを用いたマネージドサービスを開始。

→**コンテナやオーケストレーションツールが動作する環境を自前で準備する必要がなくなり、安定動作する環境が整いつつ有る。**

大きい課題はクリア。
であれば残る課題は後1つ

どのクラウドベンダ
を選定するか



2.GKE利用までの経緯(まとめ)

■ サービス運用状況について

2018年1月時点で日本国内GAしているのは
Googleのみ(他はβ版)

正式サービス運用は大きな判断基準となる。

2.GKE利用までの経緯(まとめ)

■ 動作安定性について

Kubernetesの開発元である事、コンテナを先取りして取り組み、4年以上の安定動作運用実績が有ることから信頼性が高い。完全にコンテナ化出来ない部分でもライブマイグレーション機能の強みを活かせる。

マネージドサービスではユーザーが対応出来ない領域も増える。どの程度安定して運用出来るのかを見極めるのは難しい。だからこそ安定動作している実績を買いたい

2.GKE利用までの経緯(まとめ)



GKEを
選定

The illustration features a central white sign with a brown border, supported by a grey building with a grid of windows. Above the sign are six streetlights with yellow glows. The background is decorated with several colorful fireworks in shades of pink, purple, blue, and yellow.

目次

自己紹介・作った物

1. プロジェクト概要

2. GKE利用までの経緯

3. 取り組む上での課題(組織面)

4. 取り組む上での課題(開発面)

5. 効果

6. 取り組んでみて

3.取り組む上での課題(組織面)

周りの理解を得る



3. 取り組む上での課題(組織面)

- ・ コンテナやオーケストレーションツールに対する周囲の理解を深めないといけない
- ・ 技術的優位性を説明出来ないといけない
- ・ 総論は賛成、だけど各論は”？”
- ・ リスクを背負えるか？
(納期遵守のPJだった)

所属部門・関係部門の合意を取り付けられない

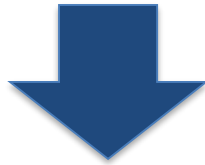
近道



なかった！

3. 取り組む上での課題(組織面)

- コンテナやオーケストレーションツールに対する周囲の理解を深めないといけない
- 技術的優位性を説明出来ないといけない
- 総論は賛成、だけど各論は”?”



技術学習

解説資料作成

説明行脚

*

∞

あとは熱意

3.取り組む上での課題(組織面)

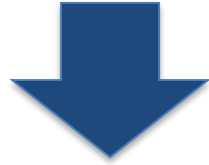
現場レベルでは味方は多かった。

- ・プロジェクトメンバー
- ・インフラエンジニア
- ・部門横断のエンジニア

技術学習の協力や相談が効率化。
求める所が同じがゆえの連帯感、等

3.取り組む上での課題(組織面)

- ・ リスクを背負えるか？
(納期遵守のPJだった)



- ・ バックアッププランの準備、提案
- ・ 技術習得状況の説明
- ・ Googleエンジニアのバックアップ

あとは熱意

目次

自己紹介・作った物

1. プロジェクト概要
2. GKE利用までの経緯
3. 取り組む上での課題(組織面)
4. 取り組む上での課題(開発面)
5. 効果
6. 取り組んでみて

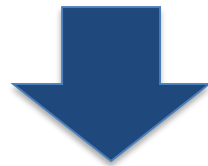
4.取り組む上での課題(開発面)

■ 技術習得

コンテナ/オーケストレーションツール/GCP/GKE/その他GoogleのPaaSサービス使用経験無し。

独学+ハンズオンで基礎固め。

+ Googleエンジニアのサポートを受けつつ、
トライアンドエラーで学習実施。



別クラウドや類似サービス実績は持っているので
なんとなく出来てくる。

4.取り組む上での課題(開発面) 苦労した点、詳細版1

■ 事前調査

- レガシーシステムのルール把握。どの程度引きずられるか、実施してから発覚した問題あり(詳細後述)。
- 効率的なシステム間連携に知識が必要、pub/sub連携のイベント発火条件も当初は無駄が多かった。プログラムか、サービス活用か、その見極め。
当初案：プログラム制御でイベント発火。
対応案：ファイル配置→Cloud Functionsで配置イベント検知→Pub/Sub投げ込み、の様に分離。

4. 取り組む上での課題(開発面) 苦労した点、詳細版2

■ 運用

- ログ出力やアラート関連は最終的に上手く出来たがメトリクス書き方に大分迷った。
GCPドキュメントも、この部分は手薄だった印象。
Googleエンジニアにサポート頂き解消。

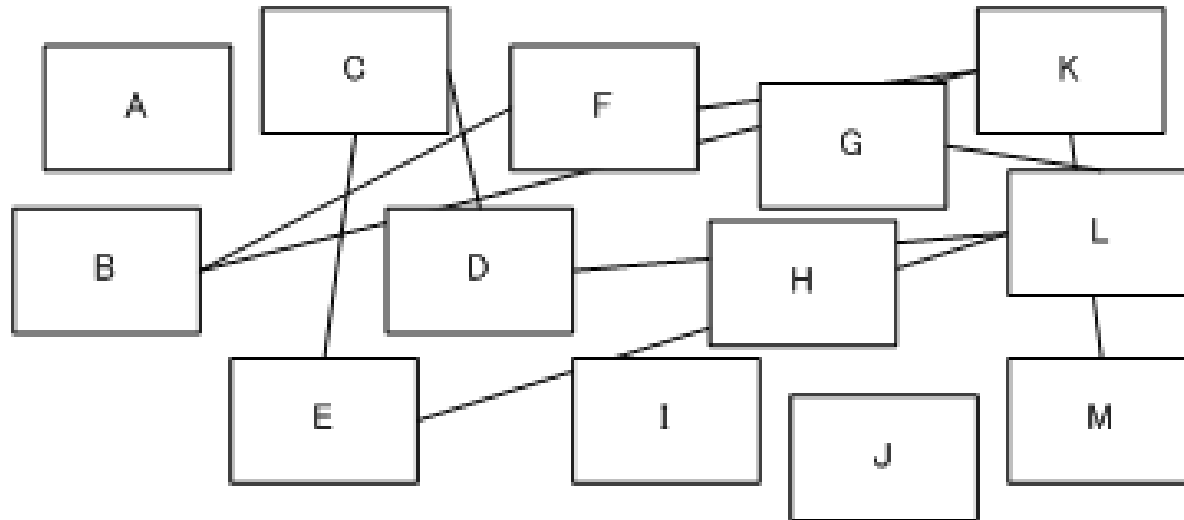
■ 設計・設定

- Kubernetes設定記述方法。
yamlの書き方の勘所に慣れるまで多少時間掛かった。
- 永続データの取扱に適切なサービス選定(今回はCloud Storage)。設計上のキーになる部分で、事前に綿密な調査が出来ていればなお良かった。

4. 取り組む上での課題(開発面)

■ 基本設計(サービス分割)

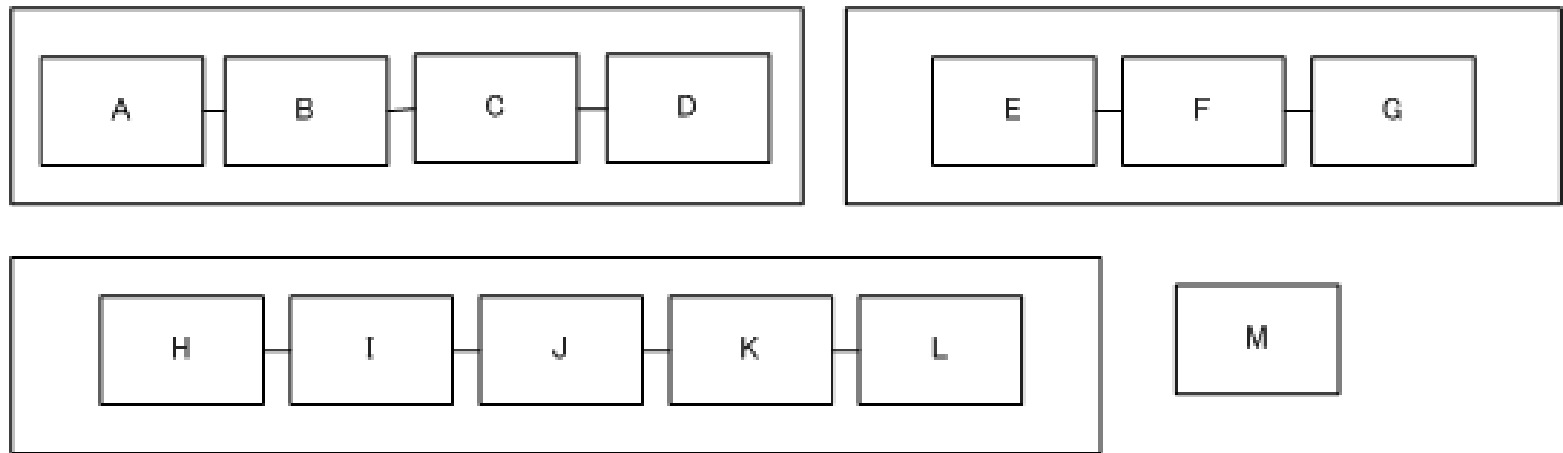
せっかくの小さく出来る機会、MAX分割。



レビューで鬼のような指摘、制御不能。
エラー発生時のリカバリパターンが非常に複雑化。
やりすぎた。

4. 取り組む上での課題(開発面)

■ 基本設計(サービス分割) 完成系 (概略)



すっきり。
小さくしやすい事に過信は禁物。

4. 取り組む上での課題(開発面)

■ 商用利用に向けての課題

アラートやアプリ安定稼働等、運用面の保証が必要。
試験PJでは無いので動けば良いという物では無い。

マネージドサービスでカバー出来る範囲の見極め。
不足する部分は自前で準備。
特に監視面は社内の別PJとそこそこに近くしたい。



PaaSサービスの特徴や仕様についてGoogle
エンジニアのサポートを受けながら選定、
確認を進める。

4. 取り組む上での課題(開発面)

■ レガシーシステムとの連携

レガシーシステム側を変える事は難しく、ルールに従わないといけない。

IP制限や、連携方法の指定。

元々予定していなかったインフラ構築やネットワーク設定が追加。

基本的には全面降伏で対応。

目次

自己紹介・作った物

1. OMSプロジェクト概要
2. GKE利用までの経緯
3. 取り組む上での課題(組織面)
4. 取り組む上での課題(開発面)
- 5. 効果**
6. 取り組んでみて

スケーラビリティは確保出来たか？

→出来た

- ・システム単位を現時点で十分な小ささに出来た。
- ・負荷の高い機能が移植予定でも分離しやすい構造。
※注文受注量が瞬間的に、恒常的に数倍に増加しても処理速度劣化無し。
- ・他国展開等、横展開しやすい構造。

**ただし今後悪化しないように
管理していく必要がある。**

保守・運用コスト大は改善されたか？

→された

- ・モノリシックな構造を改善し影響範囲は小さくなった。
保守コストが削減(機能搭載費用約1/2)
- ・サーバーリソースの有効活用でランニングコスト減。
(約3/5)
- ・負荷向上時にもピンポイントなスケールで不要な
サーバー構築が不要な構造に。
- ・ログ調査やアラート連携が改善、分析基盤のベースに。
- ・導入8ヶ月でサービスダウンタイム無し、安定動作中。

機能改善スピードは向上したか？

→向上した

- ・モノリシックな構造を改善できた結果、影響範囲が見極めやすくなり、対応速度が向上した(約2倍)
- ・イベント開催にも柔軟に対応可能な構造に。

目次

自己紹介・作った物

1. OMSプロジェクト概要
2. GKE利用までの経緯
3. 取り組む上での課題(組織面)
4. 取り組む上での課題(開発面)
5. 効果
6. 取り組んでみて

6. 取り組んでみて

- GKE関連の学習コストは(得られるリターンに比べれば)小さい。すぐに動かせる環境が手に入る
ので、トライアンドエラーもやりやすい。
GCPドキュメントの充実も大きい。
ただ、使いこなすという面ではまだまだ勉強が必要。
- アプリ開発に集中出来る環境を整える事が出来た。
従来比でインフラ懸念を大きく取り除く事が出来た。
- 組織の壁は高い、乗り越えるには熱意
仲間がいれば突破しやすい。
- クラウドベンダエンジニアの協力は偉大。

ご清聴
ありがとうございました

