

Google Cloud

Next

Tokyo

マルチテナントな Internal Developer Platform における FinOps の設計と導入

金田 拓 (KANEDA Taku)


株式会社リクルート データ推進室
エンジニア

2019~2021 インターネット・メディア系企業
ETL・データパイプライン開発

2021~ 株式会社リクルート
データプラットフォーム開発・SRE



アジェンダ

- 
- 01 リクルートの横断データ組織
 - 02 Knile とは？
 - 03 見えてきた Knile の課題
 - 04 Knile における
FinOps の導入と成果
 - 05 現在・今後の取り組み

リクルートの横断データ組織

リクルートにおけるデータ活用の実例と
それを実現する組織体制

株式会社リクルート

Simplify Business Processes



image from <https://www.recruit.co.jp/company/business/>

データ推進室とは？

- マトリクス型組織
 - タテ: 事業領域特化
 - ヨコ: 横断の専門職
- 300名以上在籍
- データサイエンスからデータマネジメントまで幅広い職種
- 横断組織データプロダクトユニットには様々な Internal Developer Platform (IDP) が存在

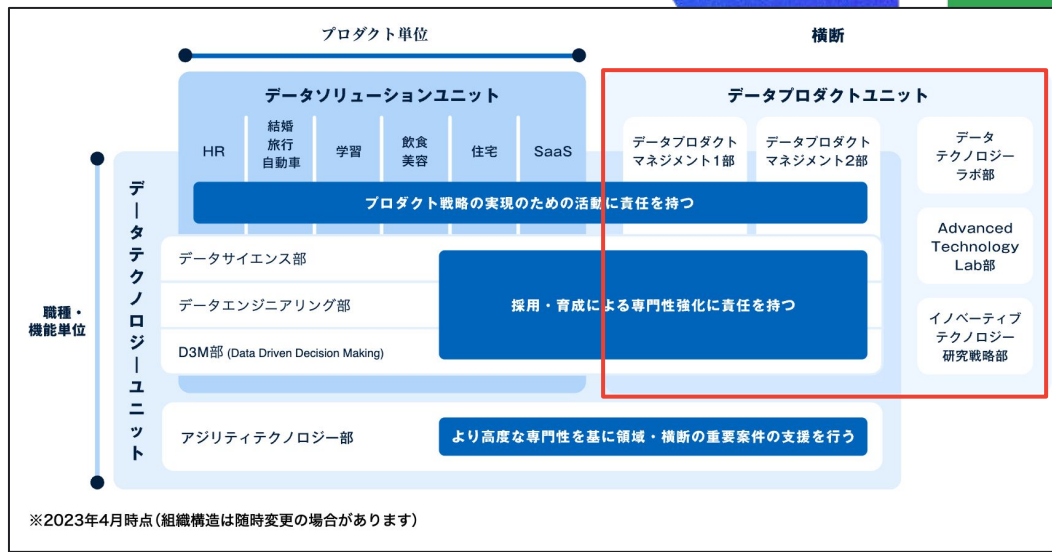


image from https://www.recruit.co.jp/employment/mid-career/data_lp/

Knile とは？

リクルートにはさまざまな Internal Developer Platform (IDP) が存在

その一つである Knile とはどんなプロダクトなのか

Knile (/naɪl/)

『データサイエンスをアジャイルに実行できる基盤』

社内の DS/DEng 向けの Internal Developer Platform (IDP)



“Knile Project”

- データ施策ごとに払い出し
- プリセットリソースの作成や権限分離
- ガバナンス・セキュリティ強化

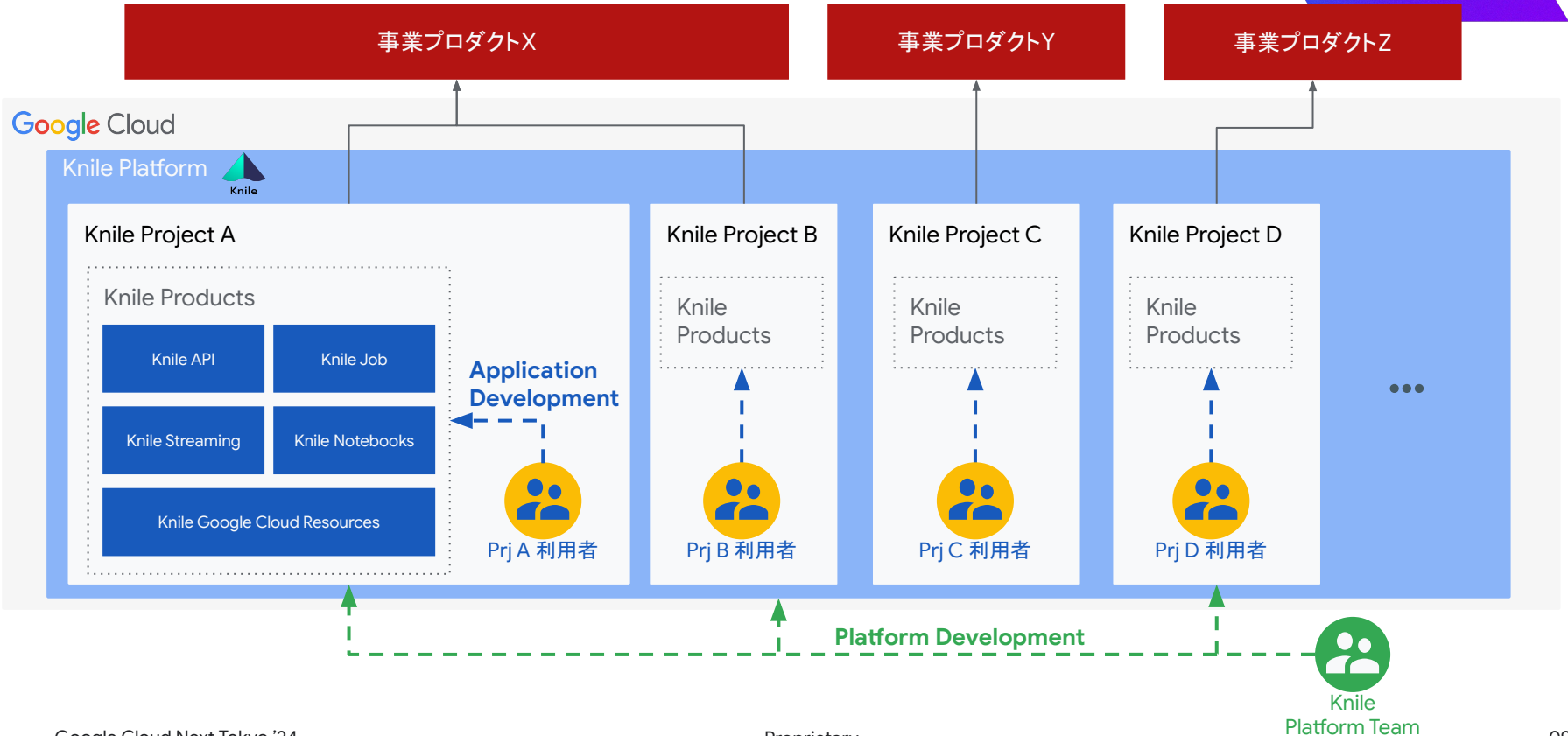
組み合わせ可能な サブプロダクト群

- Knile API: API 基盤
- Knile Job: バッチ基盤
- Knile Notebooks: Ad-hoc 分析基盤
- Knile Streaming: リアルタイムデータ処理基盤
- Project Resource: Google Cloud リソース (e.g. BigQuery dataset)

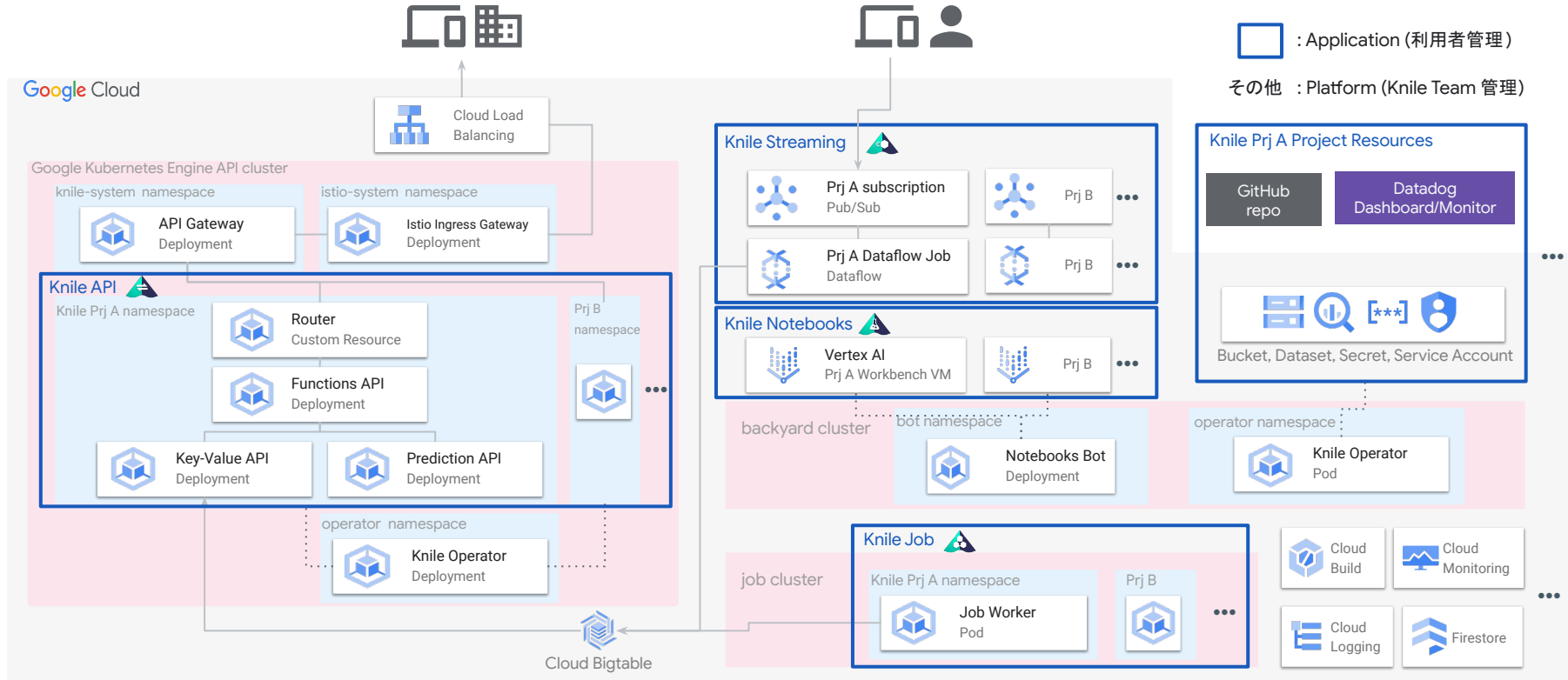
一気通貫した開発体験

- 本番環境への反映も GitHub 上で完了
- レビューもコードも一元管理
- CI/CD 環境の提供

Knile Overview



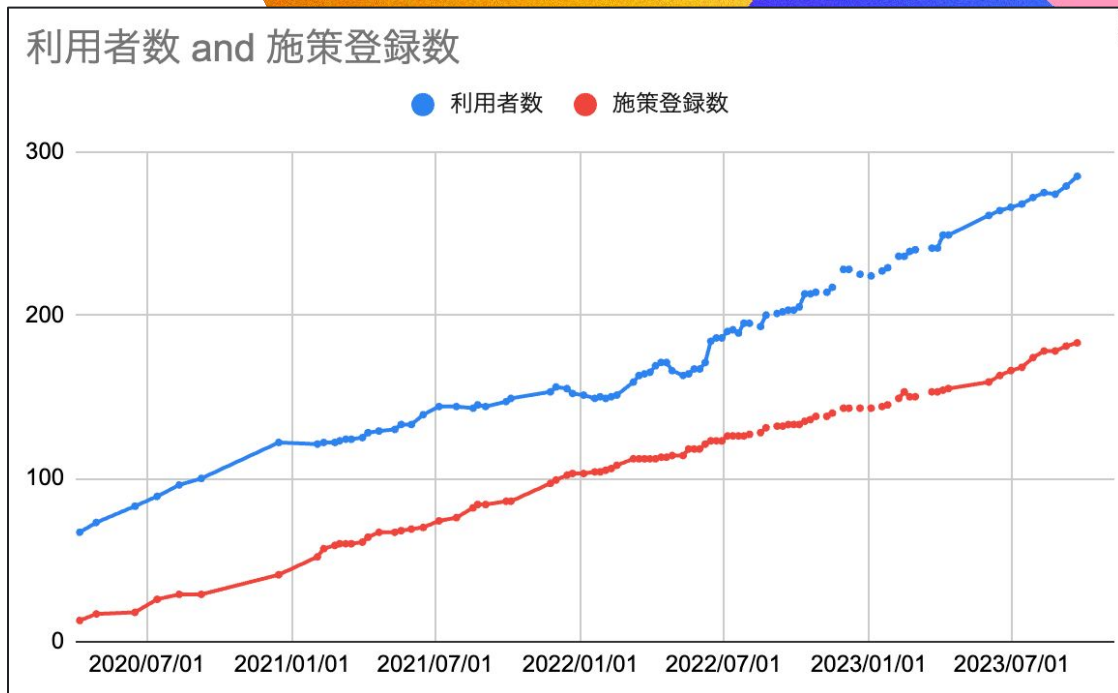
Knile Project Overview



Knile Usage Overview

- 施策数 200+
- 利用者数 300+
- 事業領域数 10+

様々な事業領域に導入され
順調に利用が増えていった



見えてきた課題

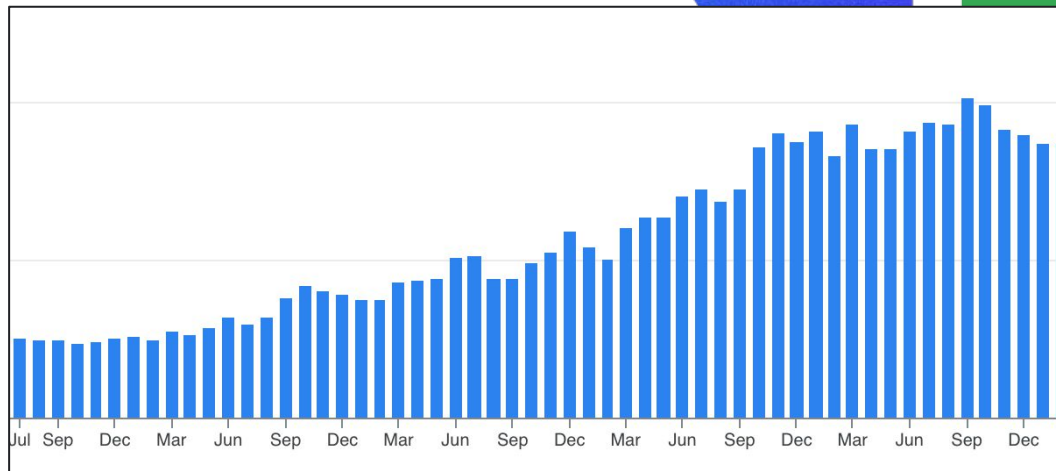
順調に利用が増えていった Knile
しかし徐々に問題も発生し ...

コストの内訳が不透明

- リソース単位のコスト不明
- 施策単位でも不明

どこに・どれだけのコストがかかっているのか

利用者・Knile Team の誰もわからない状態... 🙄

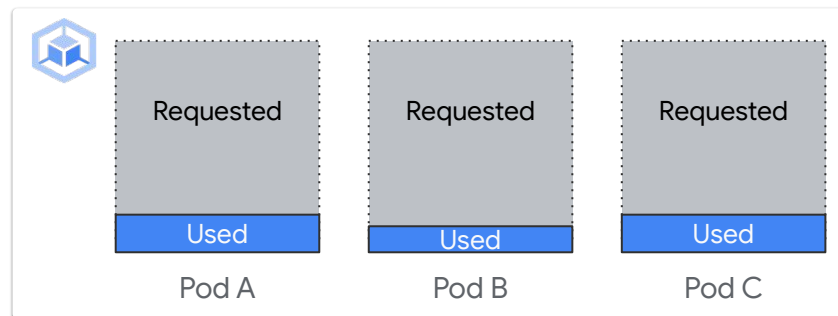


Knile prd 環境 Google Cloud Project の利用額推移

不意のコスト増・リソース余剰

- Knile にはリソースのオートスケーリング機能が存在
 - 意図しないオーバーサイジングによるコスト増大リスク
- 利用者側でリソースの設定も可能
 - 必要以上に過剰なリソースを設定してしまい、余剰が多すぎる状態も

→ 🤔 「なんか Knile って高くないですか？」
と言われてしまう事態に



API で余剰が多い例

いびつな費用負担構造

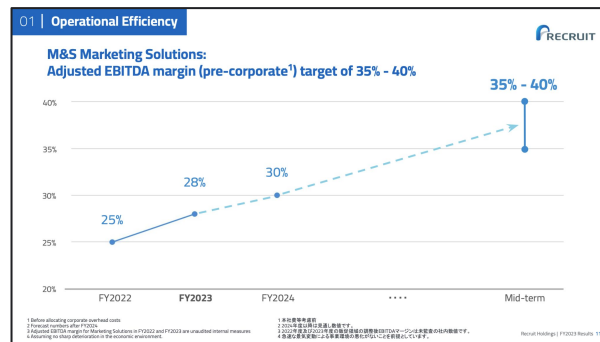
歴史的経緯により Knile の費用負担の構造はいびつなものに...

- 🤔 全社コスト時代
 - Knile のコストは共通基盤として全社コストで賄われていた
 - いわば“公費”であり利用者に支払い義務はなく、コスト意識も低い状態
 - 😞 固定費請求時代
 - 上記課題を踏まえ、利用者に請求する形をとった
 - ただ、コストの内訳が不明だったため固定費請求となってしまった
 - 利用増減で実態と請求が異なり、利用者としては納得感がない状態
- 😊 利用者が利用した分だけ支払う「受益者負担の原則」の実装が求められた

組織内のコスト意識高まり

- Knile が所属するデータ推進室ではコストに対する意識もより高まった
 - 会社の経営方針として“事業の効率化”を図る動き

→ 😊「コスト周りをどうにかしないといけない！
FinOps の導入だ！」



株式会社リクルート [2024年3月期通期決算決算説明動画プレゼンテーション資料](#)より↑

Knile での FinOps の導入

Knile では先の課題を踏まえ、FinOps を必須の機能と捉えることに

そもそも FinOps とは？



FinOps is an operational framework and cultural practice which maximizes the business value of cloud, enables timely data-driven decision making, and creates financial accountability through collaboration between engineering, finance, and business teams”

「FinOps は、クラウドのビジネス価値を最大化し、タイムリーなデータ主導の意思決定を可能にし、エンジニアリング、財務、ビジネス チーム間の連携を通じて財務責任を生み出す運用フレームワークおよび文化的実践です。」

<https://www.finops.org/framework/>

FinOps で実現したいこと



利用者のコスト意識向上

Knile 利用者一人ひとりが自身の施策のコストを意識することが必要



アクセス可能なデータ

ビジネスに関わる人は誰でも、施策のコストへのアクセスを可能にすることが必要



コスト最適化

ビジネス利益を最大化するため、コストを適切に最適化することが必要

「最適化 ≠ 削減」
コストが安ければよいというわけではない

FinOps Phase

3つの iterative な phase

- Inform (可視化と割り当て)
- Optimize (最適化の計画)
- Operate (実行と運用)

Knile では最初の Inform すら十分に実現できていなかった...

→ プロダクトに必要な「いち機能」として捉え、まずは Inform できるように



image from <https://www.finops.org/framework/phases/>

FinOps Inform Phase 導入プロセス



①設計

プロダクトの課金設計を行う

プロダクト (Google Cloud にて構築) だけでなく、その他の要素も関連する場合があるため下準備が必要。

②実装

設計をベースに実装をする

設計をもとに必要な実装を行う。Google Cloud の場合はラベルをできるだけ利用する。

③運用

課金・コスト集計に関わる業務遂行

コストの可視化や課金請求の運用業務。利用者からの問い合わせ対応なども含む。

FinOps Inform Phase 導入プロセス



①設計

プロダクトの課金設計を行う

プロダクト (Google Cloud にて構築) だけでなく、その他の要素も関連する場合があるため下準備が必要。

②実装

設計をベースに実装をする

設計をもとに必要な実装を行う。Google Cloud の場合はラベルをできるだけ利用する。

③運用

課金・コスト集計に関わる業務遂行

コストの可視化や課金請求の運用業務。利用者からの問い合わせ対応なども含む。

制約条件の確認

プロダクトの課金設計は様々な要因によって決まる

それぞれの会社・事業・プロジェクトなどで求められる要件が異なるため、その都度適切な設計が必要！

- 会社 or 部署の経営・会計方針(予算管理方法や確定時期 など)
- 費用負担構造 (IDP 側 or 利用者側?)
- 利用しているインフラ基盤(各種クラウドベンダー or オンプレなど)
- コスト請求の粒度
- 人件費や減価償却の扱い...

→ これから可視化する数字が「正しく」なるよう、自身のプロダクトにおける制約条件を確認しておくこと！

制約の例: Knile の課金設計の確定時期

Knile を含むデータ推進室の IDP の課金設計は、事業領域の予算策定のベースになる
→ 確定時期が早い

7月

課金設計の開始

FP&A グループから課金方針が共有される

Knile を含む各横断データプロダクト (IDP) の課金設計を進める

8月

課金設計の確定

決裁会議にて Knile の課金設計が確定される

9月~

事業領域の予算策定

IDP の課金設計をもとに事業側の予算策定を行う
予算の確定時期は領域ごとに異なる場合も

...

翌年4月~

課金請求の開始

課金設計をもとに領域への請求を実施

Knile の特徴

1

マルチテナント

一つの Google Cloud 上に複数事業の施策 が乗っているため、適切に分離をしなければならない

2

IDP

IDP ゆえ 利用者管理の Application Layer と Knile 管理の Platform Layer にも分離されている
Platform Layer は全 Knile Project で共通で利用しており、そのコストを適切に按分する必要がある

3

支出=収入 (利益 = 0) とする必要

会社の方針として、IDP はかかる支出をすべて利用する事業側へ請求 する必要がある

Knile における課金設計

設計方針:『Knile 利用者に利用した分だけを請求し、
Knile Platform としては収支 ± 0 とする』

- Knile Project で論理分離できる単位でコストを集計
 - Application Layer は基本分離可能
- そのために必要なリソースの粒度の確認
 - e.g. Knile API であれば Pod 単位・時間は day 単位
- それでは分離しきれないリソースの按分方法を決める
 - ラベルでは分離できない Application Layer リソース
 - 全 Knile Project で共通利用するリソース (Platform Layer)

FinOps Inform Phase 導入プロセス



①設計

プロダクトの課金設計を行う

プロダクト (Google Cloud にて構築) だけでなく、その他の要素も関連する場合があるため下準備が必要。

②実装

設計をベースに実装をする

設計をもとに必要な実装を行う。Google Cloud の場合はラベルをできるだけ利用する。

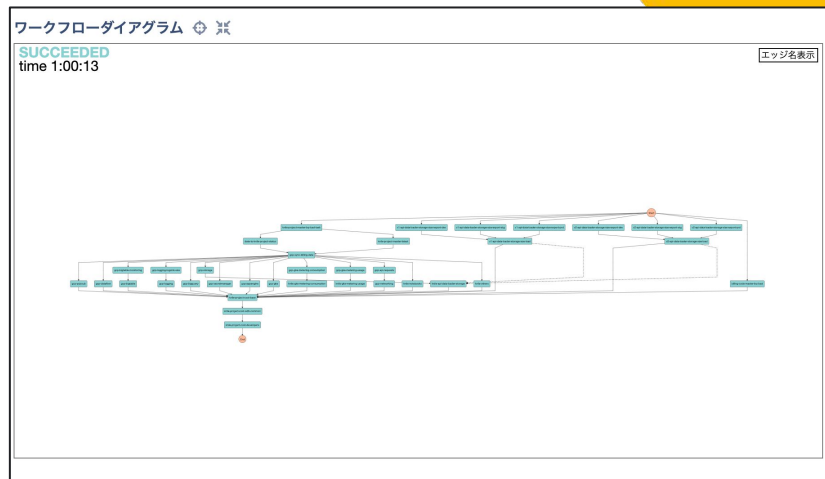
③運用

課金・コスト集計に関わる業務遂行

コストの可視化や課金請求の運用業務。利用者からの問い合わせ対応なども含む。

実装の全体像

- Google Cloud で事前に準備された機能を利用
 - リソースラベル
 - GKE usage metering
- Cloud Billing detailed usage cost data を BigQuery に連携
- SQL にてコストを集計するバッチ処理の作成
 - 各種 Google Cloud Product & Knile Product ごとにコストを集計 → 最後にすべてを合わせる



コスト集計ジョブの DAG 全体像
バッチ処理は Knile Job 上で実行

実装詳細

各種 Knile Project ごとに按分されたデータをまとめて一つの BigQuery table とする

Application Layer

- Google Cloud の機能でリソース分類
- 上記で分離できない場合は各リソースごとに按分

Platform Layer

- 共通費の按分



実装詳細

各種 Knile Project ごとに按分されたデータをまとめて一つの BigQuery table とする

Application Layer

- Google Cloud の機能でリソース分類
- 上記で分離できない場合は各リソースごとに按分

Platform Layer

- 共通費の按分



Google Cloud のラベル

Google Cloud の各種プロダクトはラベルによってコストの分類が可能

- 関連するリソースをグループ化するために Google Cloud で使用できる Key-Value ペア
- 「ラベルで分離可能な粒度」 < 「Knile Project の論理分離粒度」であればこれを適用
- ラベルの設計・設定もできるだけコード管理する

Knile Project Resource

`knile-<ENV>-<KNILE_PROJECT_ID>` bucket に以下の label を Knile Operator で付与する。

```
knile-project: $KNILE_PROJECT_ID
knile-product: storage
```

ラベル設計の例 (Cloud Storage bucket)

ラベルの実装例

	labels.key	labels.value
1	knile-product	storage
	knile-project	test-maintainers1
	managed-by	knile-operator
	managed-by-cnrm	true
	storage-bucket	[REDACTED]
2	knile-product	storage
	knile-project	test-maintainers1
	managed-by	knile-operator
	managed-by-cnrm	true
	storage-bucket	[REDACTED]

```
336     bucket.Namespace = common.GetNamespaceFromProject(instance.Name)
337     if _, err := ctrl.CreateOrUpdate(ctx, r.Client, bucket, func() error {
338         bucket.Annotations = map[string]string{
339             "cnrm.cloud.google.com/project-id": r.GcpProjectID,
340             // ref: https://cloud.google.com/config-connector/docs/how-to/managing-delet
341             "cnrm.cloud.google.com/deletion-policy": "abandon",
342             "cnrm.cloud.google.com/state-into-spec": "merge",
343         }
344         bucket.Labels = map[string]string{
345             "managed-by": "knile-operator",
346             "knile-project": instance.Name,
347             "knile-product": "storage",
348             "storage-bucket": bucketName,
349         }
350         bucket.Spec.BucketPolicyOnly = func(b bool) *bool { return &b }(true)
351         bucket.Spec.Location = func(s string) *string { return &s }("ASIA")
352         if bucketName == systemEphemeralBucketName {
353             bucket.Spec.LifecycleRule = []cnrmstoragev1beta1.BucketLifecycleRule{
354                 {
355                     Action: cnrmstoragev1beta1.BucketAction{Type: "Delete"}
```

GKE usage metering

- GKE ではクラスターソースの request/consumption のデータ (metering) を BigQuery に連携可能
 - **gke_cluster_resource_usage**
 - **gke_cluster_resource_consumption**
- Billing データと組み合わせることで、Namespace や Kubernetes label の単位でコスト算出もできる
- Knile では主に Knile API のコスト集計で利用

GKE metering 実装例

labels.key	labels.value
knile.jp/project	knile-examples
service.istio.io/canonical-name	knile-function-api
app.kubernetes.io/name	knile-function-api
knile.jp/name	call-kv-bulk-sample-fn
pod-template-hash	866fd4b7c
security.istio.io/tlsMode	istio
service.istio.io/canonical-revisi...	TODO
version	TODO

```
24 func (r *KeyValueReconciler) createOrUpdateDeployment(kv *apiv1alpha1.KeyValue,
25     appName := kv.AppName())
26     labels := map[string]string{
27         "app":                appName,
28         "version":            "TODO",
29         "app.kubernetes.io/instance": appName,
30         "app.kubernetes.io/name":    kv.KeyValueAPIName,
31         "app.kubernetes.io/managed-by": managedBy,
32         "knile.jp/name":            appName,
33         "knile.jp/project":         kv.GetKnileProject(),
34     }
35     if deploy.ObjectMeta.Labels == nil {
36         deploy.ObjectMeta.Labels = labels
37     }
38
```

ラベルで分類できない場合は？

いくつかの Google Cloud のリソースは、ラベルでは Knile Project の論理単位で分類が不可能な場合がある

- 例えば、Cloud Logging では「ログエントリごとにラベルによって分類する」といったことは現状できない
- しかし、Knile Project ごとに適切にコストを按分する必要がある

→ リソースごとに按分ロジックを取り決める

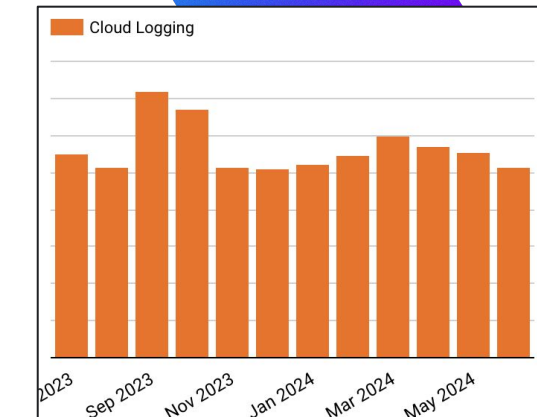
按分の例: Cloud Logging

- Knile ではデバッグ・分析用に Cloud Logging を利用者に公開
 - Cloud Logging のコストは logging storage で決定 (retention 30d の場合)

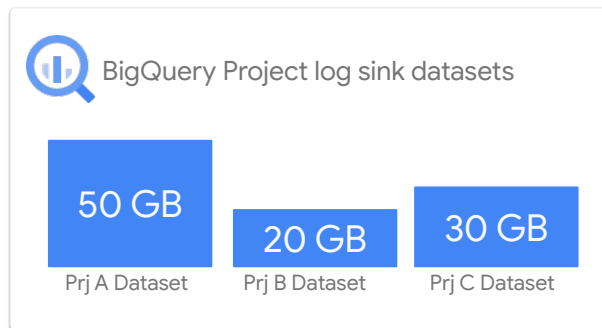
→ Knile Project 別にフィルタリングしたログを BigQuery
へ log sink

- Cloud Logging 全体のコストを Knile Project 毎の log sink size の比率で按分

按分比率例:



Cloud Logging は Service 単位での課金



50%
(=50G/100G)

20%

30%

実装詳細

各種 Knile Project ごとに按分されたデータをまとめて一つの BigQuery table とする

Application Layer

- Google Cloud の機能でリソース分類
- 上記で分離できない場合は各リソースごとに按分

Platform Layer

- 共通費の按分



「共通費」の按分は？

- マルチテナントの特性上、Knile には全 Knile Project で共通利用のコンポーネントが複数存在 (= Platform Layer のリソース)
 - e.g. Istio Ingress Gateway, Knile Operator, Knile Notebooks Bot...
- そもそも Knile Project ごとに分離不能なリソース
 - e.g. Knile 自体の開発環境・サンプルの Knile Project...
 - 減価償却費・人件費など Google Cloud 以外のコスト

こういったリソースの按分方法も考慮事項しておく

→ Knile では「**Application Layer の利用金額比率**」で按分している

実装まとめ

各種 Knile Project ごとに按分されたデータをまとめて一つの BigQuery table とする

Application Layer

- Google Cloud の機能でリソース分類
 - ✓ 可能な限り Google Cloud ラベルを利用
 - ✓ GKE の場合は metering データを利用
- 上記で分離できない場合は各リソースごとに按分
 - ✓ できるだけ適切かつシンプルなロジック

Platform Layer

- 共通費の按分
 - ✓ Application の金額割合で按分

FinOps Inform Phase 導入プロセス



①設計

プロダクトの課金設計を行う

プロダクト (Google Cloud にて構築) だけでなく、その他の要素も関連する場合があるため下準備が必要。

②実装

設計をベースに実装をする

設計をもとに必要な実装を行う。Google Cloud の場合はラベルをできるだけ利用する

③運用

課金・コスト集計に関わる業務遂行

コストの可視化や課金請求の運用業務。利用者からの問い合わせ対応なども含む。

Documentation

FinOps の実現のためには、Knile 利用者にもコストの構造を理解して貰う必要がある

そのためにも documentation をきちんと書くことは非常に重要

The screenshot shows the Knile v2 Tutorials website. The left sidebar contains a navigation menu with categories like 'Knile 全体', 'コスト・課金', and 'Knile API Platform'. The main content area is titled 'リファレンス' and 'FYE 2025/03 課金設計・オペレーション'. It lists various cost-related items such as '課金設計', '課金ロジック', 'Knile Project v2', 'Knile API v1', 'Knile Job v1', '旧 CET 系インフラ費', 'Knile 施策別インフラ費 詳細', 'Knile 施策別インフラ費 コストテーブル', 'Knile Project GCP Resources', 'Cloud Storage Bucket', 'Secret Manager Secret', 'Networking (FYE 2025/03 追加)', 'Cloud Logging (FYE 2025/03 追加)', 'Knile API', 'Kubernetes Resources', 'Data Loader Storage', and 'Bigtable (FYE 2025/03 追加)'.

Cloud Logging (FYE 2025/03 追加)

Cloud Logging 全体のコストを Knile API ログシンク用の BigQuery dataset 内全 table の、日毎パーティションのデータサイズで按分します。

$$C_{\text{Cloud Logging},p} = C_{\text{All Cloud Logging Usage}} \times \frac{\text{Size of Logging Sink BigQuery dataset}_p}{\sum_p \text{Size of Logging Sink BigQuery dataset}_p}$$

コストテーブルでは下記の情報が利用できます。Cloud Logging は Knile Project ごとの単位で集計されます。

Knile API

Knile API は Knile API の実際のリソースである Kubernetes Resource と、Key-Value API で利用される Data loader storage と Bigtable の額で計算されます。

$$C_{\text{Knile API},p} = C_{\text{Kubernetes Resources},p} + C_{\text{Data Loader Storage},p} + C_{\text{Bigtable},p}$$

Kubernetes Resources

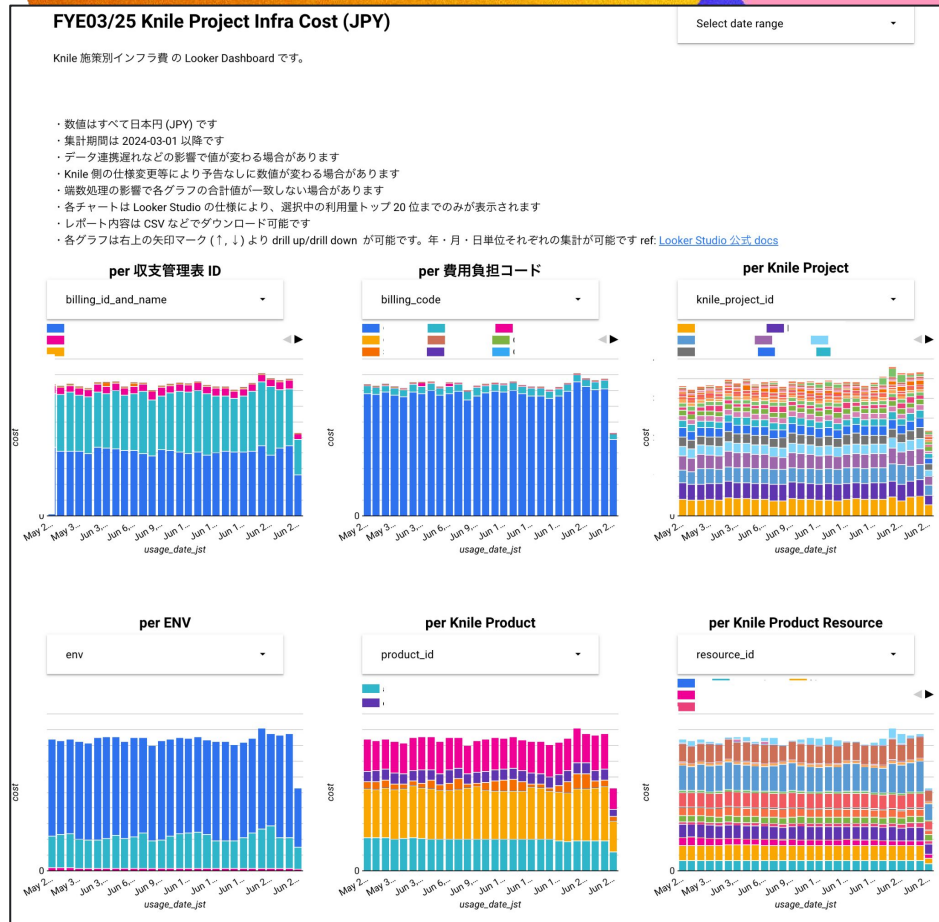
Knile API がホストされている Frontage Cluster のコストを、同 Cluster 上で起動する各種 Pod (Functions/Key-Value/Prediction) の Resource の利用率で按分します。

$$C_{\text{Kubernetes Resources},p} = C_{\text{Frontage Cluster}} \times \frac{C_{\text{API Pod Resource Request},p}}{\sum_p C_{\text{API Pod Resource Request},p}}$$

① 注釈

Looker Studio による 可視化

- 領域・Knile Project・環境・Knile Product などの dimension
- 閲覧者により観点が異なる
 - 事業領域の部長: 領域全体のコスト推移
 - 事業領域の開発 mgr: 担当施策全体のコスト推移
 - 施策担当の DEng: 担当施策の Product ごとのコスト詳細



スプレッドシートによるデータ公開も

- Knile のコストに関わるのはエンジニアに限らない
 - データプランナー・経理・営業推進 etc... など様々な職種
- 特に経理チームに馴染み深いスプレッドシートでも情報公開
 - コネクテッドシートにより自動で連携
 - 権限管理もスプレッドシート側で対応可能



knile_pricing_fye2503_SAMPLE | 4 Rows | Refresh options

Chart | Pivot table | Function | Extract | + Calculated column | Column stats

PREVIEW

Tr	Tr	Tr	Tr	Tr	Tr	123	123	123
billing_id	billing_nar	billing_cox	month	knile_proj1	knile_proj2	knile_pric1	crois_pric1	total_pric1
A0510000	サンプル領域	AA0000	4/1/2024	knile-awesome1	active	5000	1000	6000
A0510000	サンプル領域	AA0000	4/1/2024	knile-awesome2	active	1000	10000	11000
A0510000	サンプル領域	AA0000	5/1/2024	knile-awesome2	active	5000	15000	20000
A0510000	サンプル領域	AA0000	5/1/2024	knile-awesome1	active	6000	2000	8000

End of data

Charts, pivot tables, and functions will use the entire data set. Create an subset of the data on a different sheet. [Learn more](#)

FinOps の導入成果

「FinOps」と銘打って取り組みを始めて 1 年ほど経過
その成果とは？

可視化の効果例

- ある施策で不要な dev 環境を利用していることが判明
- 可視化するまでこのリソースの存在に利用者は気づけていなかった

→ dev リソースの削除により
コスト削減につながった！



Knile の現在地

- Inform は整備された 
 - 各コストがどこにかかるのかがわかる
 - コスト責務の分離
 - Platform vs Application
- 次は **Optimize & Operate** に取り組む！ 



再掲 image from <https://www.finops.org/framework/phases/>

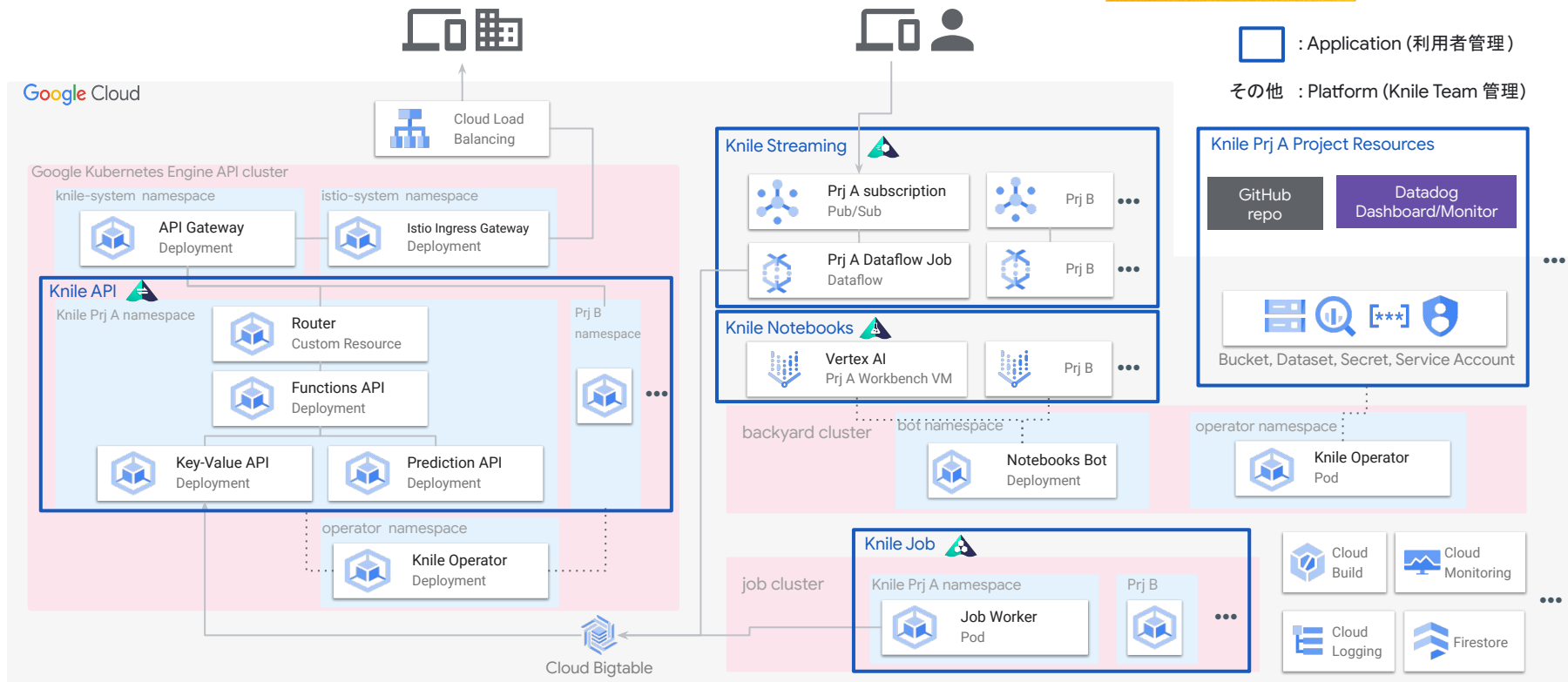
現在・今後の取り組み

Knile における FinOps Optimize/Operate Phase の
最近の取り組み内容を紹介

Cost の分類

- コスト可視化 (Inform) により Knile のコストを Platform と Application に分類可能
 - Platform: 共通コンポーネントなど Knile Platform 内に閉じる
 - Application: 利用者が開発・運用
- コスト管理の責務もそれぞれ分離
- 両者に対してアプローチしていくことが可能

[再掲] Knile Project Overview



Platform Cost Optimization

- Knile Platform 内部のみで完結 する内容
- 利用者とのネゴシエーションなどの必要がない

現在の取り組み一例

- System Logging の最適化
 - Application logging は利用者管理のため責務外
- GKE API cluster への Spot VM 導入検討
- システム利用 BigQuery Dataset/Google Cloud Storage Bucket への lifecycle 導入

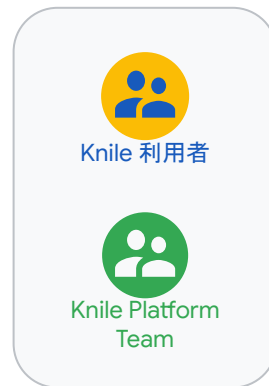
Application Cost Optimization

- 利用者が管理する Application に手を入れる
- 現状の Knile では利用者だけで完結できる取り組みに限度がある
 - **Knile が Enabling Team として振る舞う**

現在の取り組み一例

- Knile API コスト最適化
 - Default の CPU/Memory request 調整
 - HPA minReplicas 数調整
 - HPA custom metrics & VPA 検討

Collaboration



Cost 最適化はどこまで進める？

- FinOps phase にある通りコスト最適化は終わりのないサイクル活動
- Knile では **SLO (service level objective) のエラーバジェット** を消費する形で進めている
- 上記の内容を利用者へ啓蒙しつつ、SRE 的な文化を醸成

Knile コスト最適化

FYE03/25 上期の取り組み内容

2024/05/23

DPM1部 DP2E2G

金田 拓

Application コスト最適化:
Phase1 API default min replica 数削減

- 現状 PRD API の default の最小レプリカ数は 3 つ
- 過剰に可用性を担保している可能性があり、2 つへ減らせないか検証

API:x → optimize → API:x

Input (やること)	Output	Risk
default min replica 数を 2 にする対象 API は Application を参照	default min replica が 2 になった API	● 可用性 (success rate)

taku_kaneda (8/19-9/15 STEP) @here

【▲ Knile API コスト最適化の取り組み共有 ▲】

今期 Knile では API コスト最適化に取り組んでおります。その内容について共有いたします。

- 取り組み内容
- Knile API リソースの各種デフォルトリソース・パラメーターを Knile 側で調整します。実行は 4 つのフェーズに分け
- phase1: prd API default min replica 数削減
- phase2: API default resource チューニング
- phase3: HPA チューニング
- phase4: API リソースレコメンド・自動調整システム

詳細は右記資料を参照してください: <https://docs.google.com/presentation/>

■ どのように進めるのか?

Knile T の通常の開発フローに則り、SLO エラーバジェットの範囲で進めます。SLO については [サービスレベル](#) を

- 問題発生時の対応方法
- 導入により Knile API 全体で SLO を毀損する問題が発生した場合は一律で切り戻しを実施します
- Knile 全体の SLO に問題はないものの一部 API で劣化が見られた場合、当該 API 開発者の方へ API Config の調

↑利用者とのコミュニケーションを取りながら進める
(社内向け共有資料より抜粋)

まとめ

要件・制約の確認

- FinOps もいち Feature
- 要件は会社・部署・プロジェクトなどにより様々
- IDP & マルチテナントの場合は複数の部署との交渉も

まずは計測から

- 設計をもとにコスト集計の実装を進める
- BI ツールを使い可視化
- データは誰でもアクセスできるように

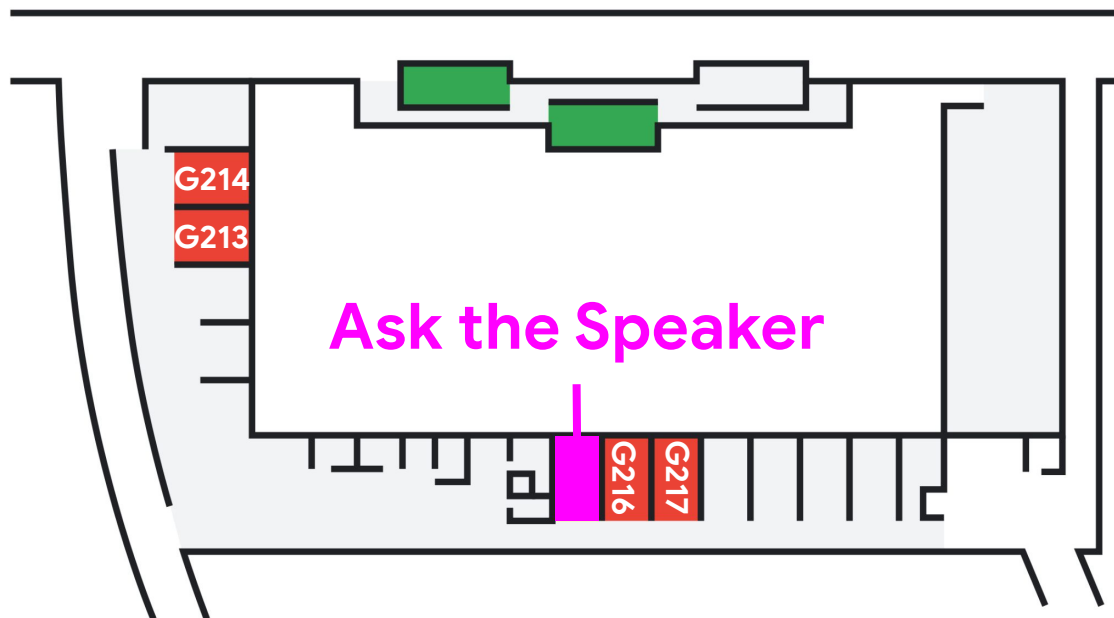
最適化を始めよう

- 計測により最適化ができる部分が見えてくる
- IDP では Platform と Application に責務分類
- それぞれ Platform/Enabling Team としてアプローチ

Ask the Speaker にぜひお越しく下さい

セッションに関する質問にスピーカーが直接お答えします！

2F





Thank you