組織と事業の急拡大に立ち向かうための
マルチテナント Amazon EKS クラスタ/
マルチアカウントアーキテクチャ

AWS Summit Online 2020

自己紹介

小笠原 純也

@0gajun

- 2018年 4月 入社
- サービスインフラグループ
- 全社でのアマゾン ウェブ サービス (AWS) 移行のテクニカルリード

今日話すこと

- マネーフォワードについて
- 組織と事業の拡大に伴うインフラにおける課題
- マルチテナントAmazon EKS /マルチアカウントへの挑戦
- マルチテナントAmazon EKS /マルチアカウントアーキテクチャ
- 開発フローの変化
- カルチャーを変えていく為に
- まとめ

今日話すこと

マネーフォワードがどのようにして

マルチアカウント + マルチテナント Amazon EKS クラスタを用いて

組織及び事業の拡大に取り組もうとしているか

取り組み始めたばかりですが、

我々の現在進行中の挑戦を紹介します

マネーフォワードについて

Money Forward

お金を前へ。人生をもっと前へ。

Money Forward Business

ビジネスの成長を加速させる。

- Money Forward クラウド
  - バックオフィス向け業務効率化ソリューション
  - Money Forward クラウド会計
  - Money Forward クラウド確定申告
  - Money Forward クラウド請求書
  - Money Forward クラウド給与
  - Money Forward クラウド経費
  - Money Forward クラウドマイナンバー
  - Money Forward クラウド勤怠
  - Money Forward クラウド社会保険
  - Money Forward クラウド会計Plus
- Money Forward 会社設立
- STREAMED
- Manageboard
- BOXIL
- ワクフリ
- BALES
- BALES CLOUD

Money Forward Home

すべての人生を、
便利で豊かにする。

- Money Forward ME
  - お金の見える化サービス
- Money Forward Mall
  - 金融商品の比較・申し込みサイト
- MONEY PLUS
  - くらしの経済メディア
- SiraTama
  - 自動貯金アプリ
- Money Forward お金の相談
  - マネーフォワード MEユーザーのための FP相談窓口
- Money Forward おかねせんせい Beta
  - マネーフォワード MEのデータを分析し最適な行動をアドバイス
- Money Forward Career
  - DX人材特化のキャリア支援サービス

Money Forward X

パートナーと共に、
新たな金融サービスを創出する。

- Money Forward for ○○
  - 金融機関お客様向け自動家計簿・資産管理サービス
- 通帳アプリ
  - 金融機関お客様向け通帳アプリ
- MF Unit
  - 金融機関のアプリへの一部機能提供
- BFM
  - 法人向け資金管理サービス

Money Forward Finance

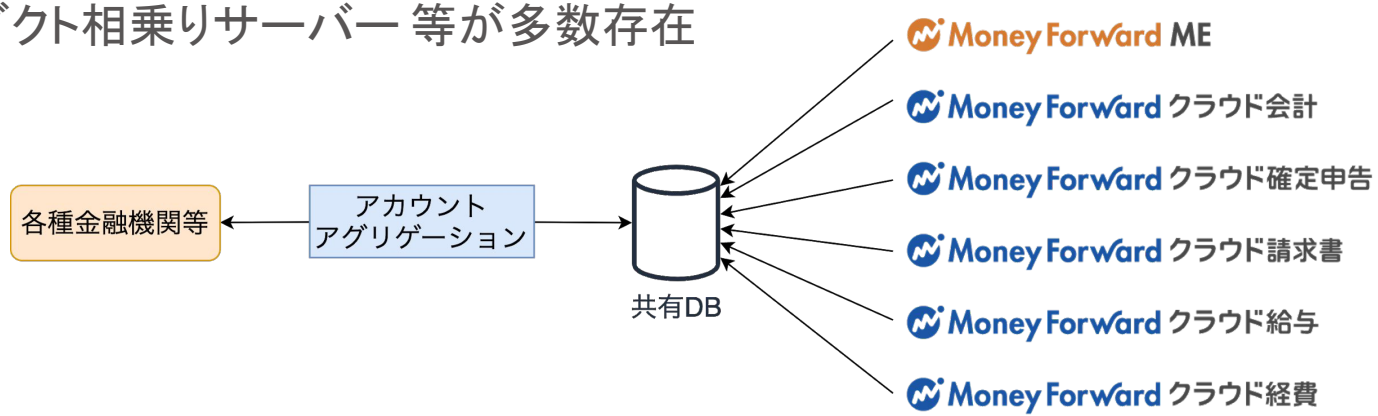
お金をいい方向へと動かす。

- MF ESSAI
  - 企業間後払い決済サービス
- MF ESSAI
  - アーリーペイメント
  - 売掛金早期資金化サービス
- Money Forward Synca
  - 成長企業向けフィナンシャル・アドバイザリーサービス

組織と事業の拡大に伴う
インフラにおける課題

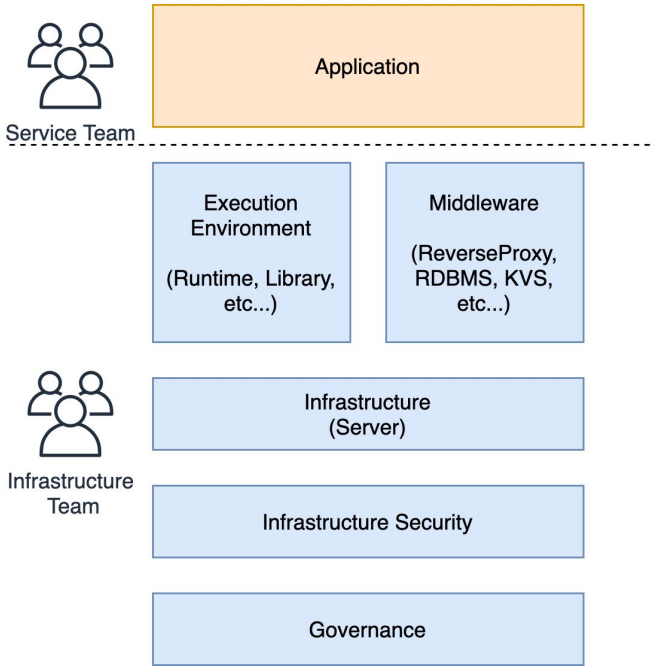
メインサービスのインフラストラクチャ

- 2012 年の創業当初からオンプレミス @北海道
  - 物理サーバーを購入し、インフラがセットアップし運用
- 数 TB の共有メイン DB
  - 主要サービスの多くがメイン DB を直参照し密結合
- プロダクト共通の実行環境
  - 複数プロダクト相乗りサーバー 等が多数存在



オンプレミス環境におけるインフラとサービスチームの役割分担

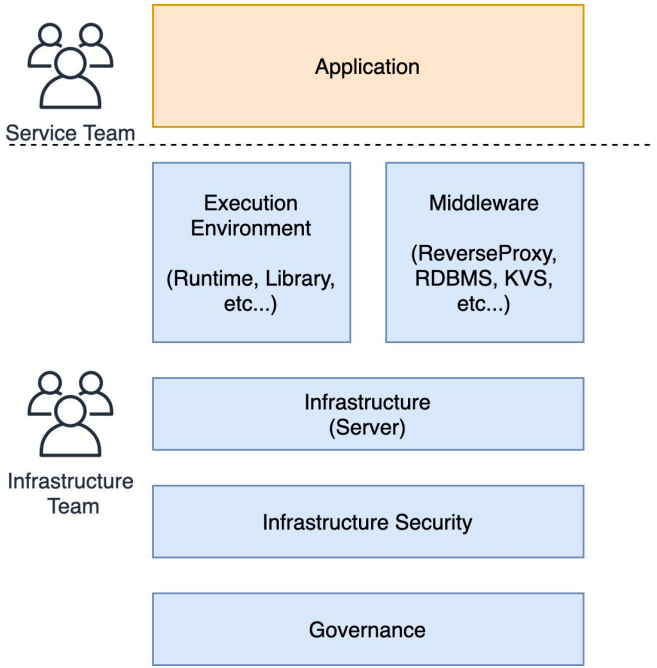
- サービスチーム
  - アプリケーションコードを管理
- インフラチーム
  - アプリケーションの実行に必要な環境を全て管理
    - Ruby ver. up はインフラ
    - リバースプロキシの設定もインフラ
    - サーバー増やすのもインフラ
    - etc...



オンプレミス環境におけるインフラとサービスチームの役割分担

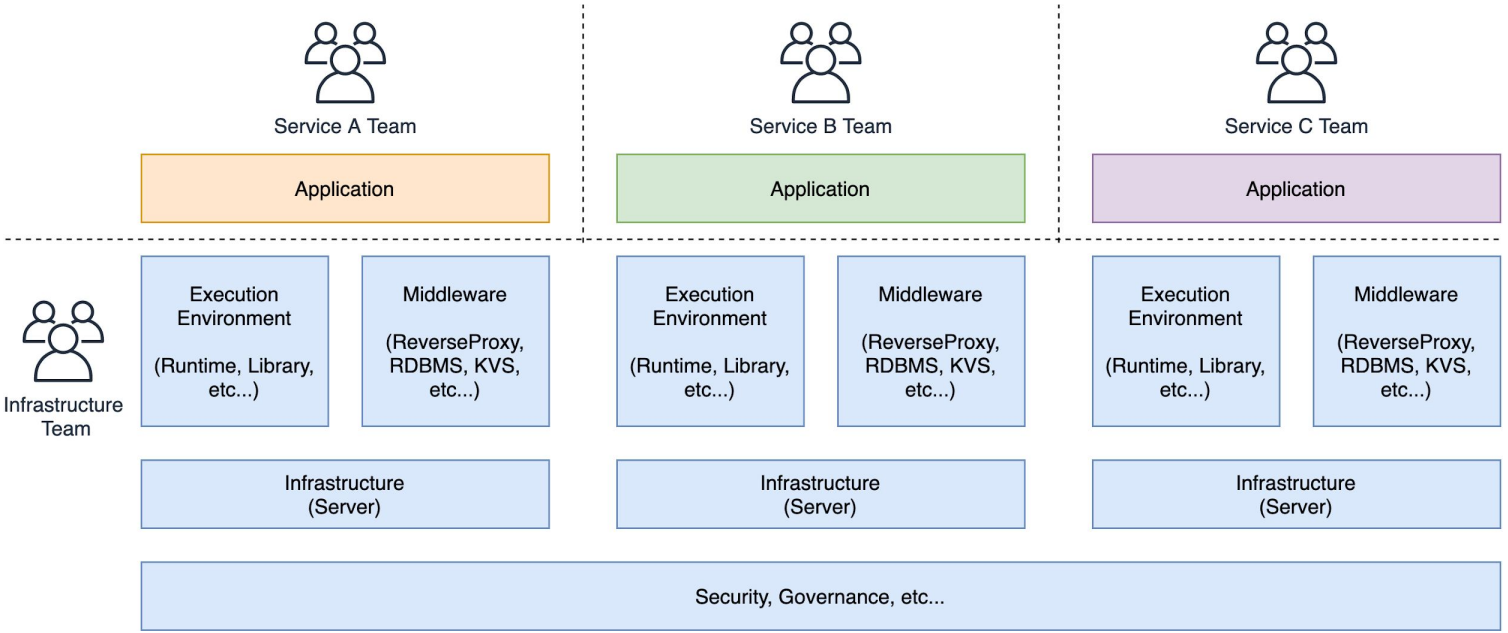
- 実際、各々得意な領域を責任持って行う
  モデルでうまく行っていた

- ただし、サービス数が少ない内は…



提供サービスの増加と開発組織の拡大

- □□をリリース 🎉 △△をリリース🎉
  - 数サービスなら単一インフラチームだけでもなんとか…

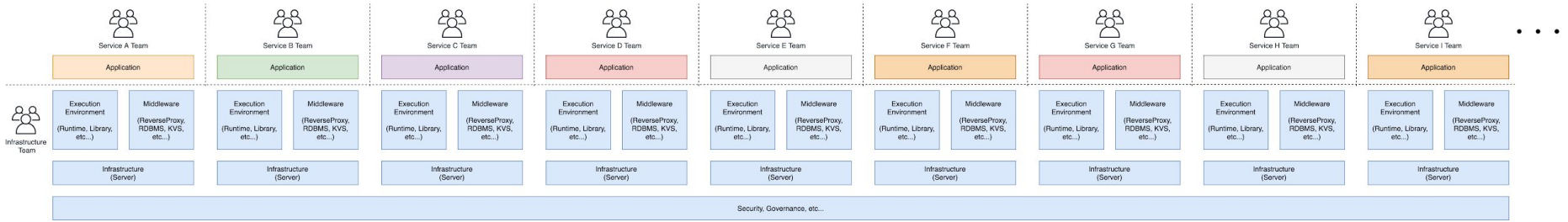


提供サービスの増加と開発組織の拡大

- 更に新規サービスや開発チームは増えていき…
  - 数十サービスの運用を一手に引き受ける状態に
- またスピードを最優先に取り組んできたため、サービスが増える毎に運用負荷が増えていく仕組みしか無かった

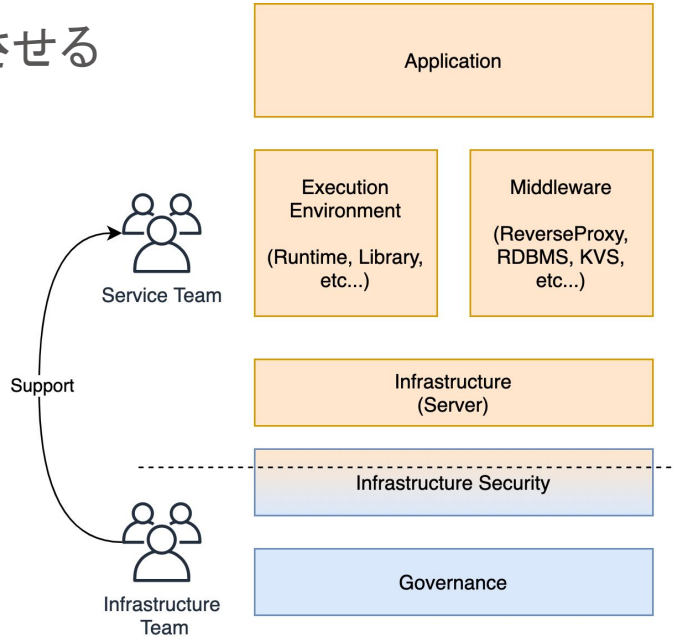
「今すぐ」な仕事にしか取り組めない状況

& インフラチームが開発のボトルネックになりがちに…



組織と事業の拡大に立ち向かうために

- サービスチームに比べてインフラチームのメンバーは増えにくい現状
  - インフラチームが全て見続けるのは現実的ではない
- サービスチームが各サービスのインフラを自分たちで管理できる形を目指す
  - 開発ライフサイクルをサービスチーム内で完結させる
  - インフラチームは必要に応じてサポート
- そのためにサービスチームがより広いレイヤを触ることができるような仕組みを作る
  - インフラレイヤの権限移譲が必要

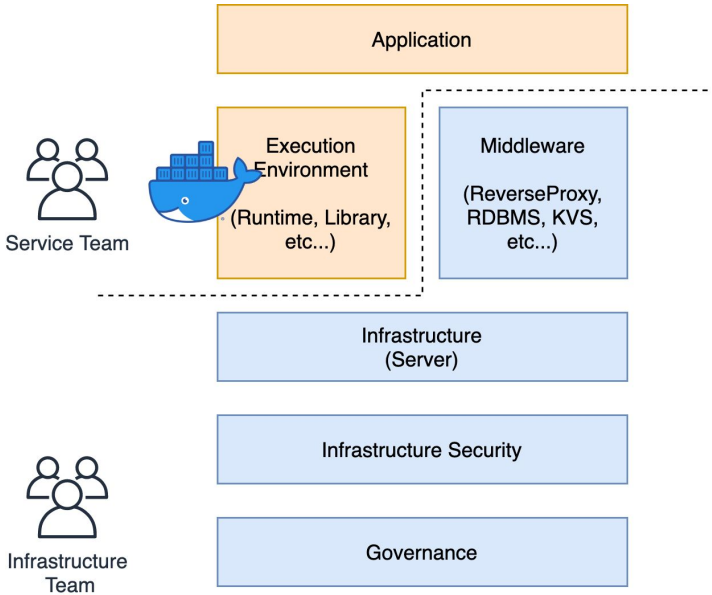


サービスチームに何を移譲していけばよいか

- アプリケーションの実行環境の管理
  - 使う言語のバージョンアップや必要なライブラリのインストール 等
- サーバーリソースの管理
  - 新規のプロセス追加やスケールアウト/イン 等
- ミドルウェアの管理
  - DB や KVS の作成や設定変更、スケールアップ/ダウン 等

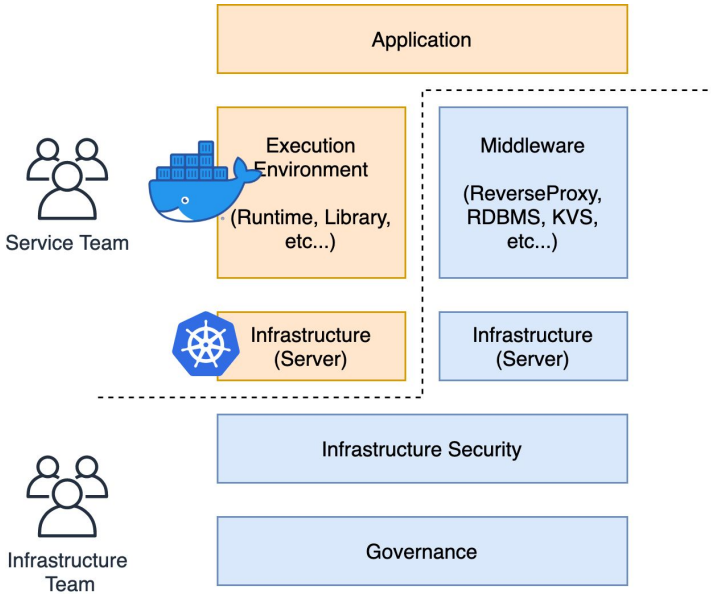
どのようにして移譲を進めるか

- アプリケーションの実行環境の管理 → **Docker**
  - 使う言語のバージョンアップや必要なライブラリのインストール 等



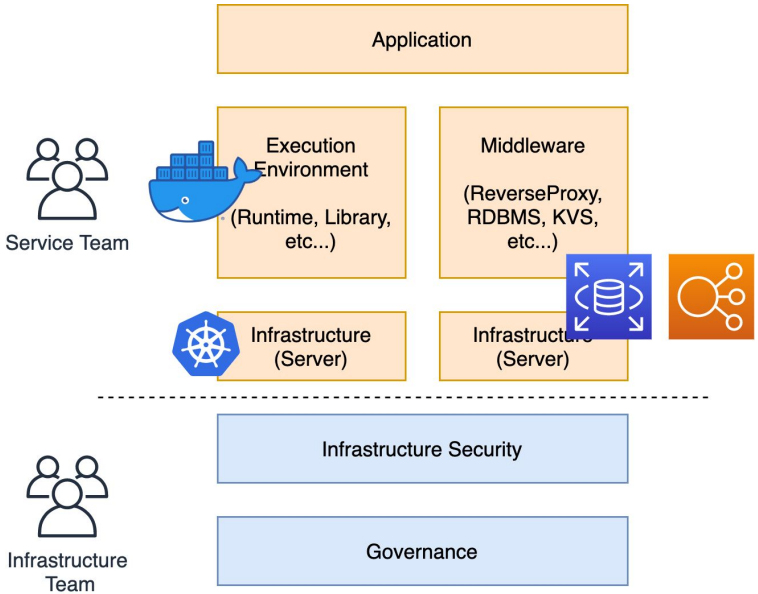
どのようにして移譲を進めるか

- サーバーリソースの管理 → **Kubernetes**
  - 新規のプロセス追加やスケールアウト/イン 等



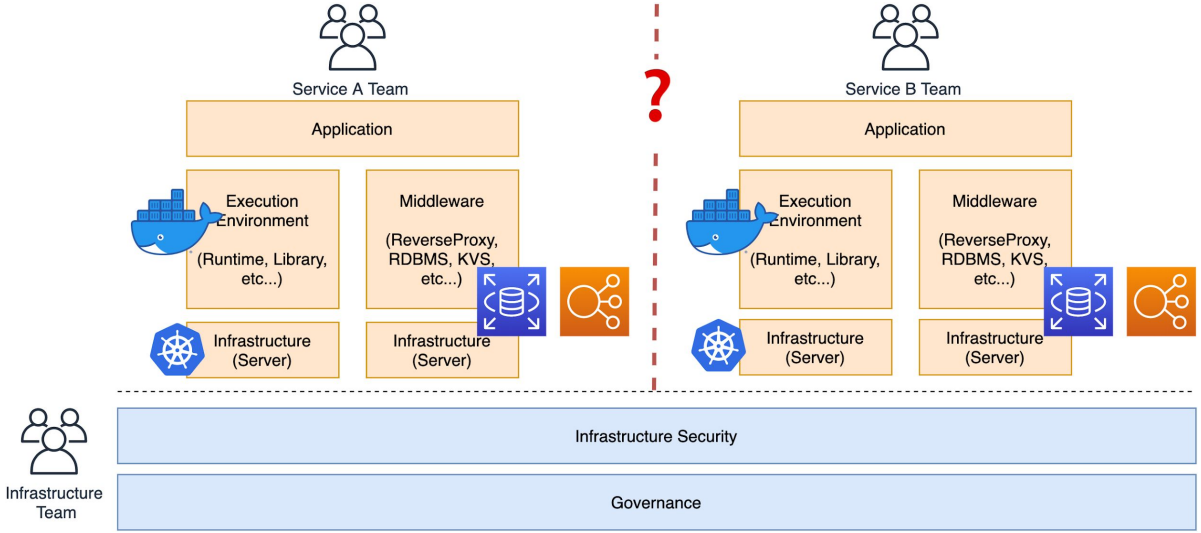
どのようにして移譲を進めるか

- ミドルウェアの管理 → **AWS**
  - DB や KVS の作成及び設定変更、スケールアップ/ダウン 等



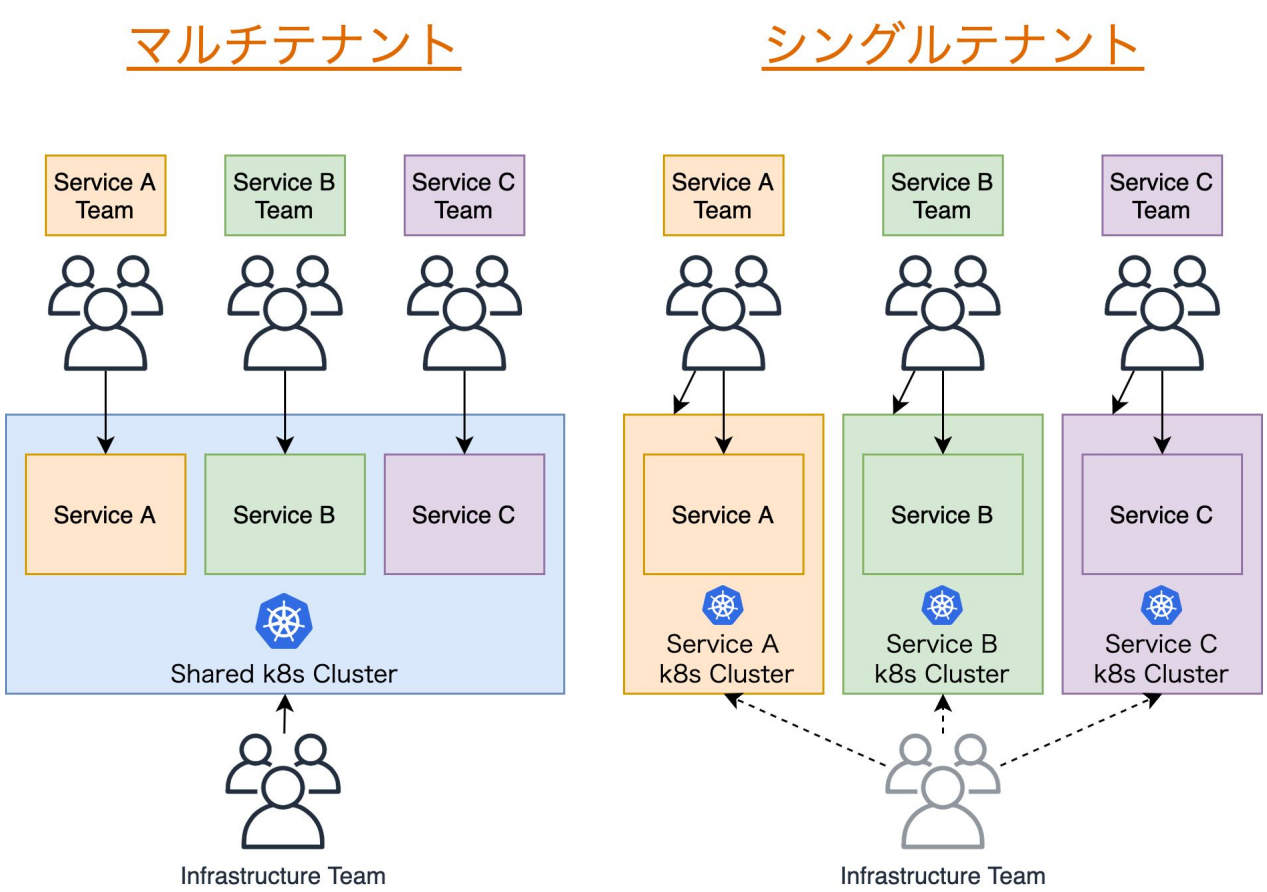
Docker + Kubernetes + AWS による移譲

- Docker + Kubernetes + AWS によるインフラの移譲
  - インフラチームの管理レイヤを徐々にサービスチームへ移していく
- 一方、サービスチーム間の独立性はどのようにして保つか？
  - 各サービスチームは他チームのことを考えずに開発できるようにしたい



マルチテナント Amazon EKS / マルチアカウントへの挑戦

マルチテナント Amazon EKS vs. シングルテナント Amazon EKS



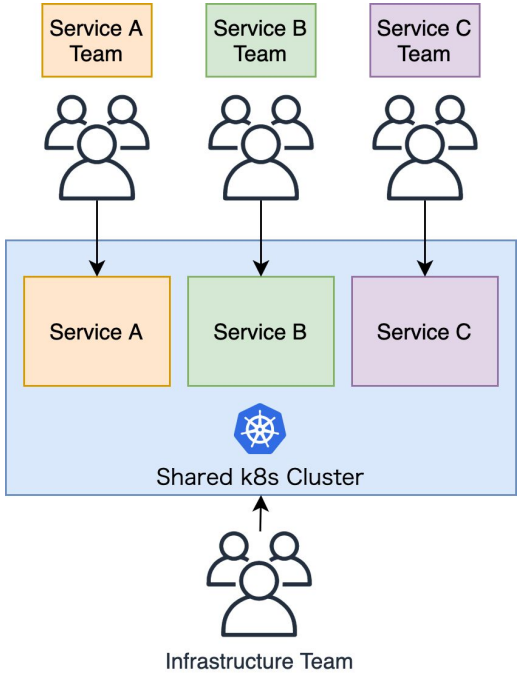
マルチテナント Amazon EKS

Good 👍

- サービスチームはクラスタ運用が不要
- インフラチームは単一クラスタだけをメンテナンス
- Namespace 毎にチーム間の分離が可能

Bad 👎

- Blast Radius (障害時の影響範囲) が大きい
- サービスチームはクラスタレベルの権限を持つことはできない
    - 自由さが制限される



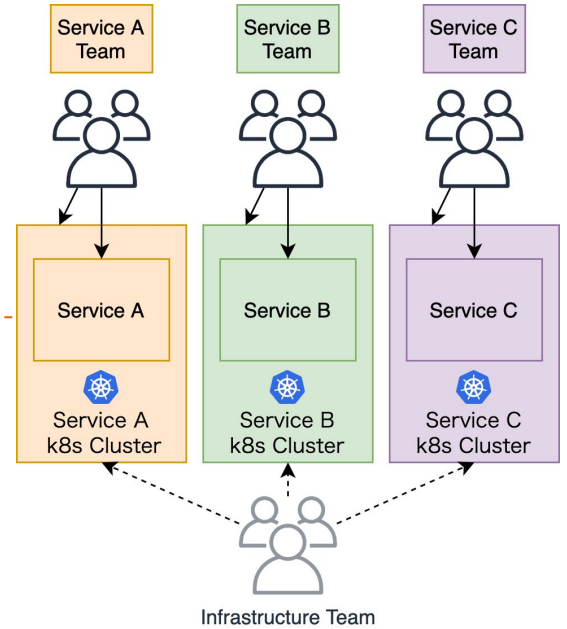
シングルテナント Amazon EKS

Good 👍

- Blast Radius が最小限
- 各サービスチームがクラスタレベルの権限を持つことができる
    - 自由度は極めて高い

Bad 👎

- 管理するクラスタの台数が増えてしまう
- 各サービスチームが自分たちでクラスタを管理しないとスケールしない
    - その文化や仕組み作りが必要



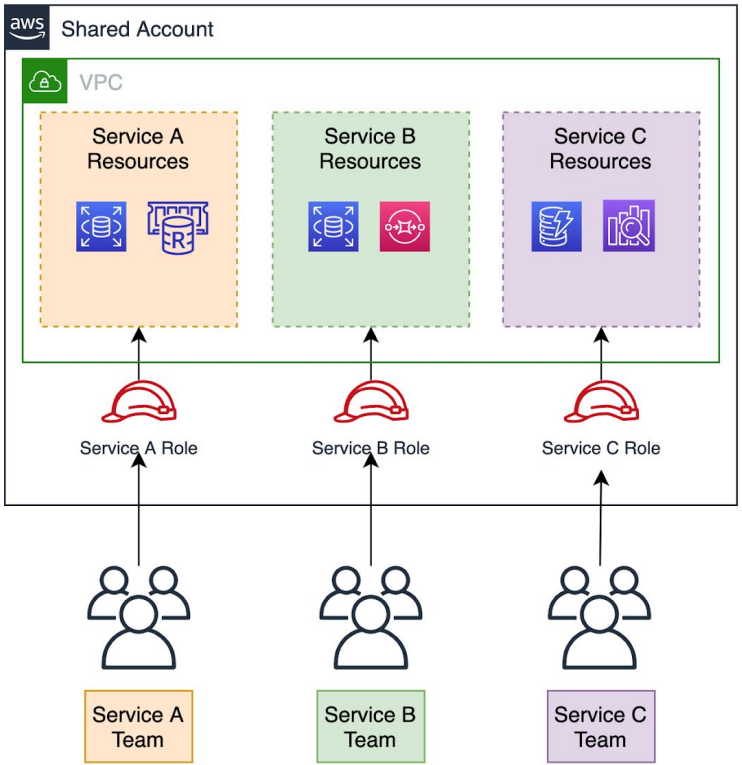
マルチテナントAmazon EKS vs. シングルテナント Amazon EKS

- シングルテナントの Blast Radius の小ささ及びチーム毎の自由は非常に魅力
- インフラチームがインフラ全てを見ていた環境から、AWS へ移行した瞬間にクラスタ管理から全部お願い！は Technical Gap が大きすぎる
    - 多数のシングルテナントクラスタをインフラチームが見るのも非現実的で運用負荷が高すぎる & 見えない

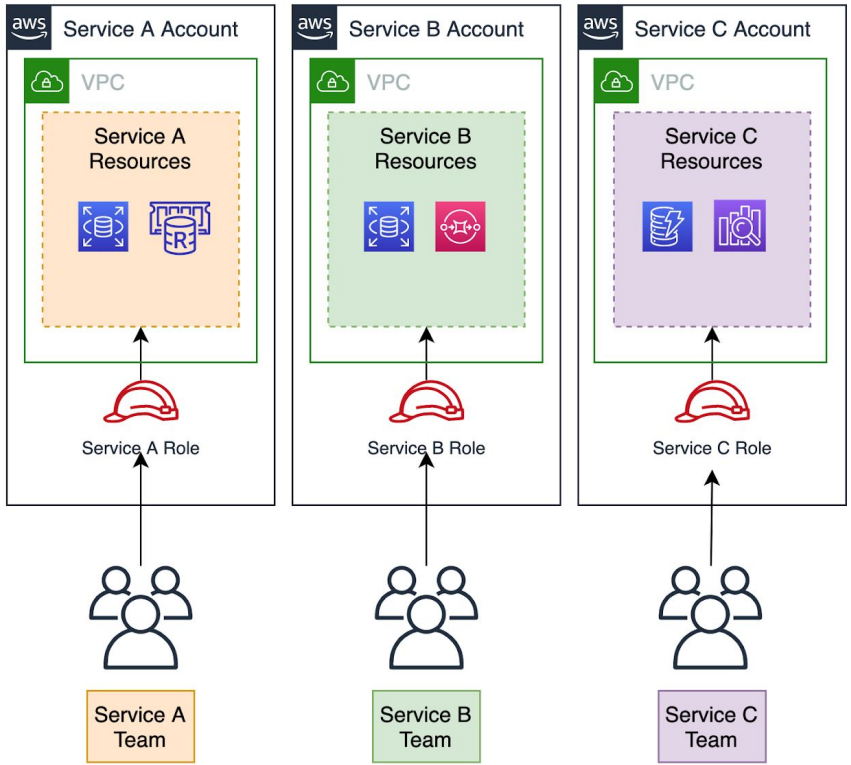
まずはマルチテナントで Namespace 単位での権限移譲を行い、
サービスチームの触ることができる範囲を広げることを最優先
将来的にシングルテナントへの分割を目指すことに

マルチアカウント vs. シングルアカウント

シングルアカウント



マルチアカウント



※簡単のため、非VPCリソースも
VPC内に記述しています

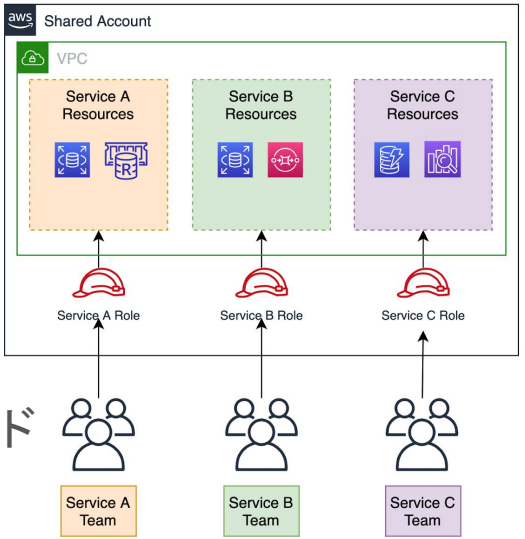
シングルアカウント

Good 👍

- 単一アカウントだけなので管理が容易
- 構成がシンプル
- 統制も単一アカウントを対象にすれば良いので導入及び運用が (初期は) 容易

Bad 👎

- IAM Policy の管理が煩雑 & 難しくなりがち
  - サービスが増えたときの権限移譲のハードルが上がる
- コスト管理では正しく Billing Tag を付与する必要がある



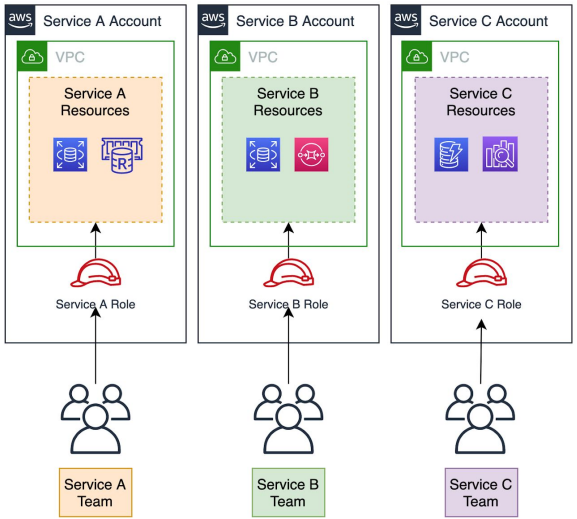
マルチアカウント

Good 👍

- アカウント間の Isolation が保たれるので IAM Policy がシンプルに
  - サービスチームによる Role の管理
- BillingTag を付与せずともコスト配賦可

Bad 👎

- アカウント管理の手間
- マルチアカウントの統制の仕組みを作る必要がある
- 構成がシンプルではなくなる
  - VPC 間通信、クロスアカウントアクセス



シングルアカウント vs. マルチアカウント

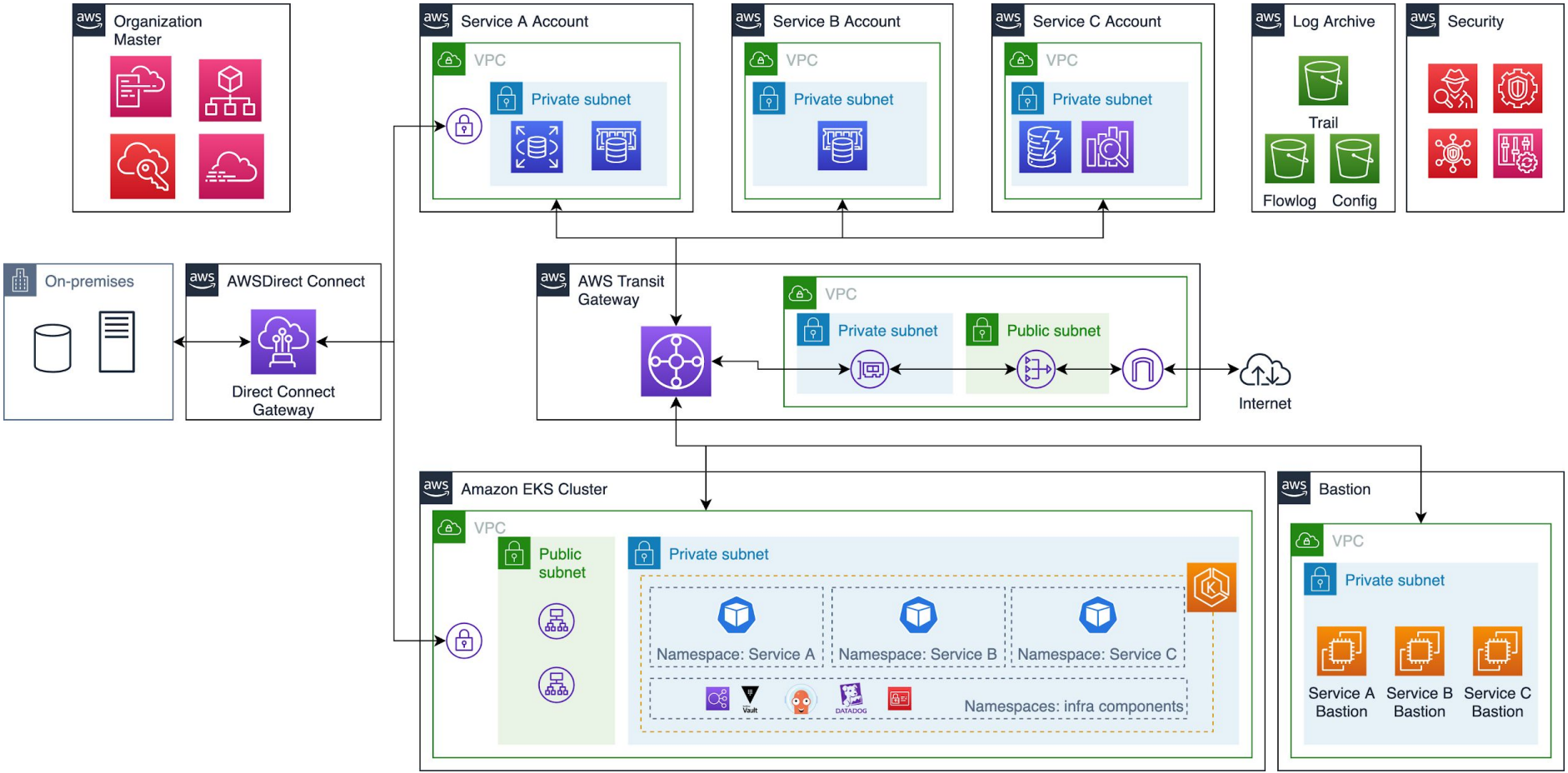
- サービスチームが他のサービスに影響なくAWSを利用できるようにしたい
- シングルアカウントを多数のサービスチームが利用する形だと、リソース間の Isolation を保つのがかなり大変
- マルチアカウントならアカウントをリソースコンテナとして扱い、Isolation を簡単に保つことができる
    - サービス毎のコスト配賦も簡単に

より自由に動ける環境を目指しマルチアカウントを選択

統制やネットワークの課題周りはインフラチームが解決

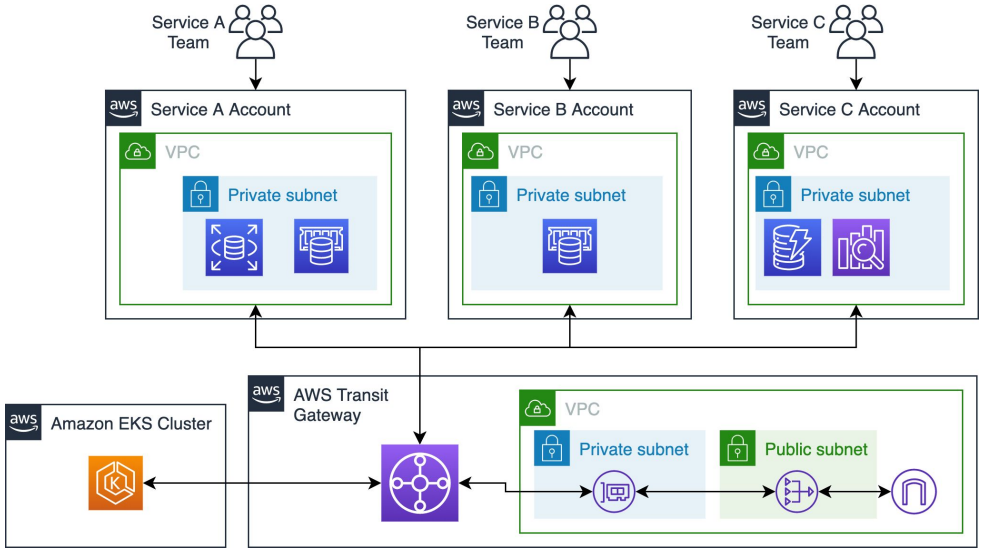
マルチテナント Amazon EKS / マルチアカウント アーキテクチャ

アーキテクチャ概要


Organization Master
Service A Account
VPC
Private subnet
Service B Account
VPC
Private subnet
Service C Account
VPC
Private subnet
Log Archive
Trail
Flowlog
Config
Security
On-premises
AWSDirect Connect
Direct Connect Gateway
AWS Transit Gateway
VPC
Private subnet
Public subnet
Internet
Amazon EKS Cluster
VPC
Public subnet
Private subnet
Namespace: Service A
Namespace: Service B
Namespace: Service C
Vault
DATADOG
Namespaces: infra components
Bastion
VPC
Private subnet
Service A Bastion
Service B Bastion
Service C Bastion

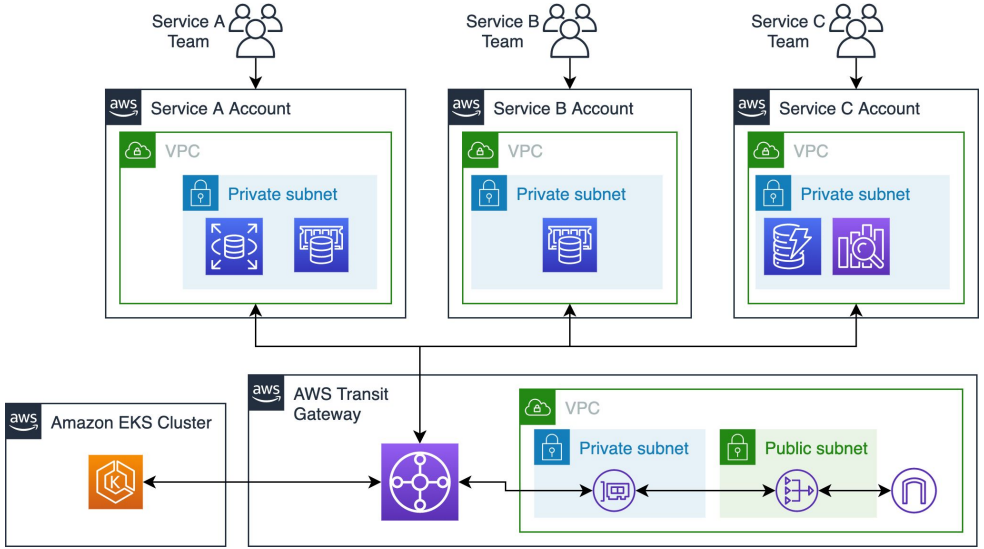
マルチアカウント 1 / 2

- サービス毎に 別々の AWS アカウントを作成
  - 各サービスチームは他プロダクトへの影響を考えずに AWS を利用可能
  - アカウント単位で Isolation が担保されるので、権限付与も容易



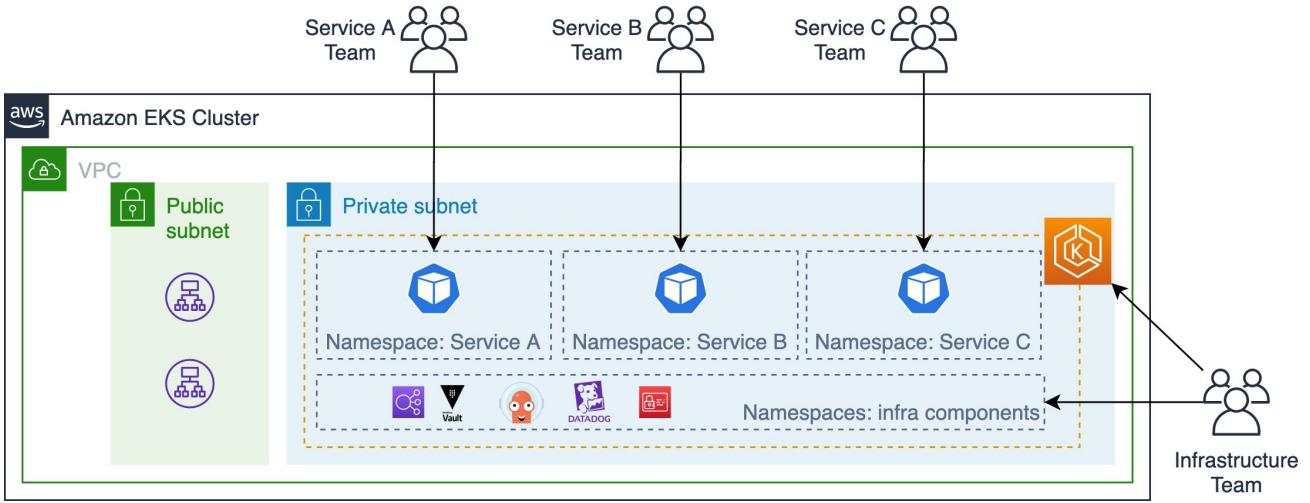
マルチアカウント 2 / 2

- マルチテナント Amazon EKS や AWS Transit Gateway 等も独立アカウント
  - インフラ関連のアカウントは Role 毎に細かくアカウントを分割
  - 各アカウントの責務最小化及び、将来的なアカウントオーナーの変更にも耐えうる設計に



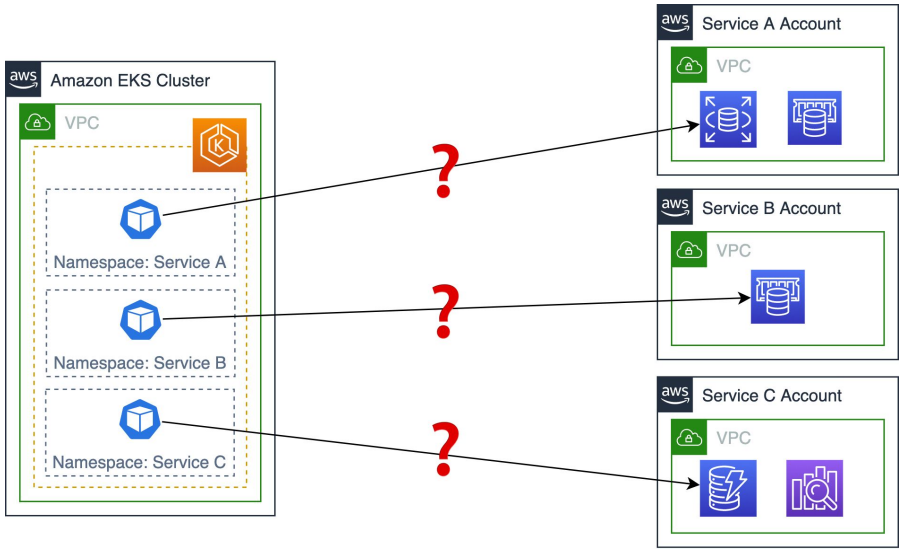
マルチテナント Amazon EKS

- サービスチームはサービス毎の Namespace 内のリソースを管理
  - 新規 Deployment / CronJob 等の追加や変更がサービスチーム内で完結
- インフラチームがクラスタを整備 & 運用
  - クラスタや各種コンポーネントのアップグレード 等を担当



マルチアカウント / マルチテナント Amazon EKS におけるネットワーク問題

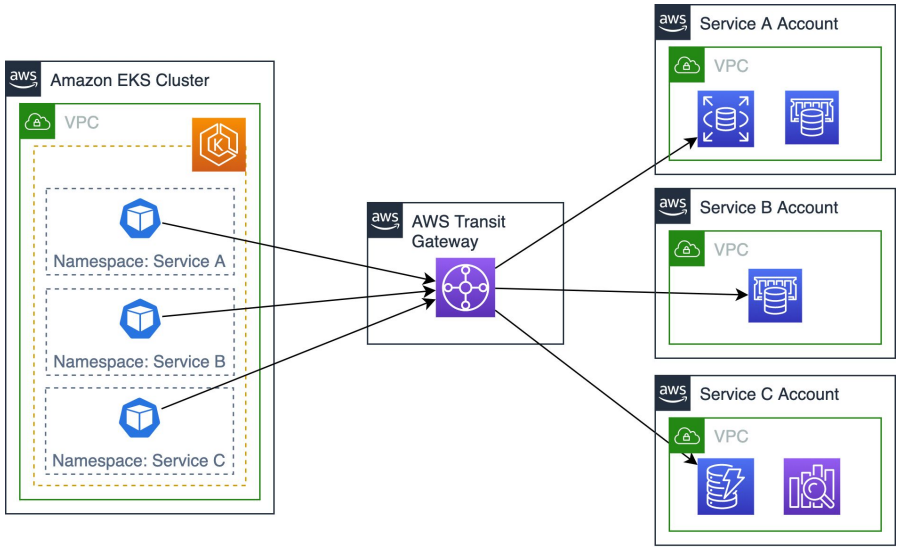
- マルチテナント Amazon EKS アカウントのアプリケーションはサービス別アカウントの VPC リソース (e.g. RDS) にアクセスできる必要がある
- しかし、シングルアカウント内に Amazon EKS も VPC リソースもある状況のように簡単にはアクセスができない



マルチアカウント + マルチテナントAmazon EKS + AWS Transit Gateway

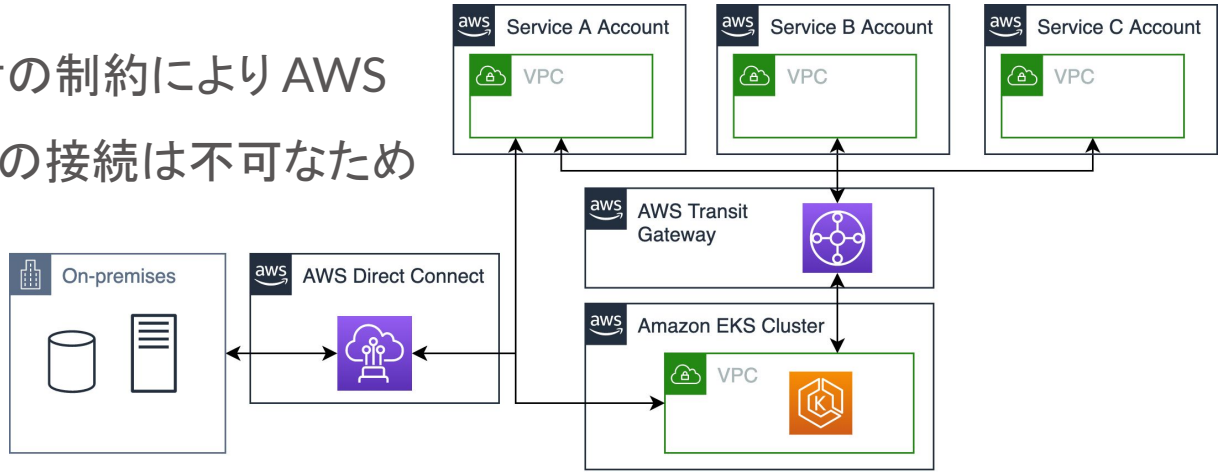
- 全てのアカウントの VPC を AWS Transit Gateway を用いて接続
  - Amazon EKS クラスタとサービス別アカウントのリソースが通信可能に
- VPC Peering に比べてアカウント数が増えても複雑化しない
- VPC Sharing は将来的なシングルテナント化を見据えると分割の壁に

今後の組織及び事業の拡大と変化に備えて AWS Transit Gateway を選択



AWS Direct Connect

- 密結合故に、あるサービスを AWS に移行したとしてもオンプレミス環境への接続性を提供する必要がある
  - 共有 DB への接続が必要なアプリケーションも多々残っている
- Direct Connect Gateway を利用して複数の AWS アカウントをオンプレミスと接続
  - オンプレ側の事業者の制約により AWS Transit Gateway への接続は不可なため

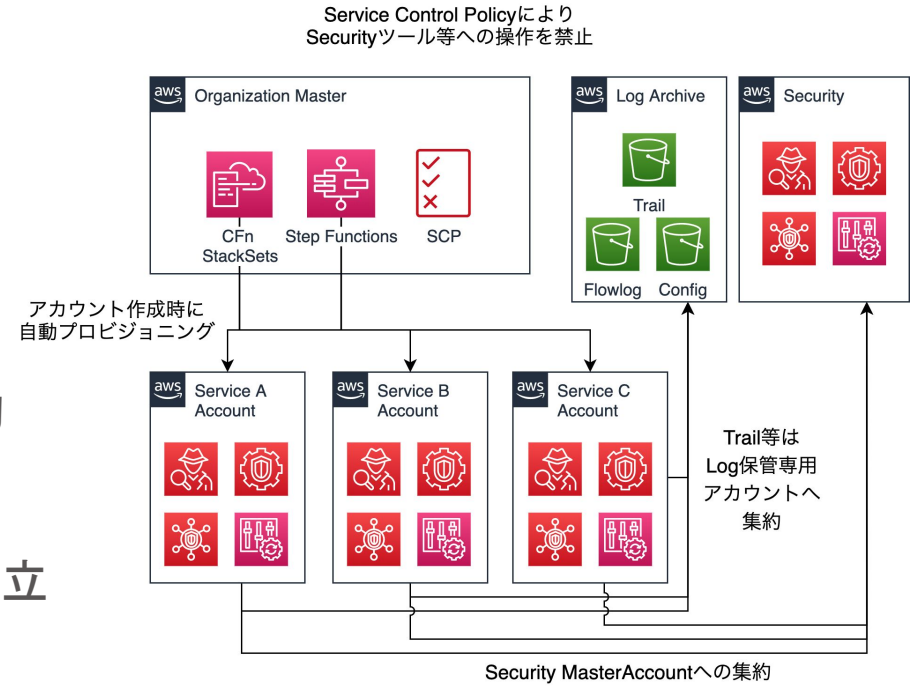


ガバナンスをどうするか

- サービスチームにアカウント単位の自由を与えることは達成できそう
- 一方で、最低限のセキュリティにおけるベースラインは担保したい
  - 大切なユーザーデータを守ることは弊社の最重要事項
- 全てのアカウントに以下のようなベースラインを強制していく必要がある
  - 本番データへのアクセス経路の制限
  - 外部との通信ログの保全
  - AWS における監査ログの保全
  - 設定ミスによる意図しないデータ漏洩防止 etc...

AWS ガバナンス

- アカウント作成時に各種 Security Tools を完全自動でプロビジョニング
  - CloudFormation Stack Sets + Step Functions
- Service Control Policy によりメンバーアカウントでの操作を一部 Deny
  - Security Tools に対する操作
  - リージョン制限 etc…
- Cloud Trail はログ専用アカウントへ集約
  - 人間がログインできないように
- Security Tools のマスターアカウントも独立

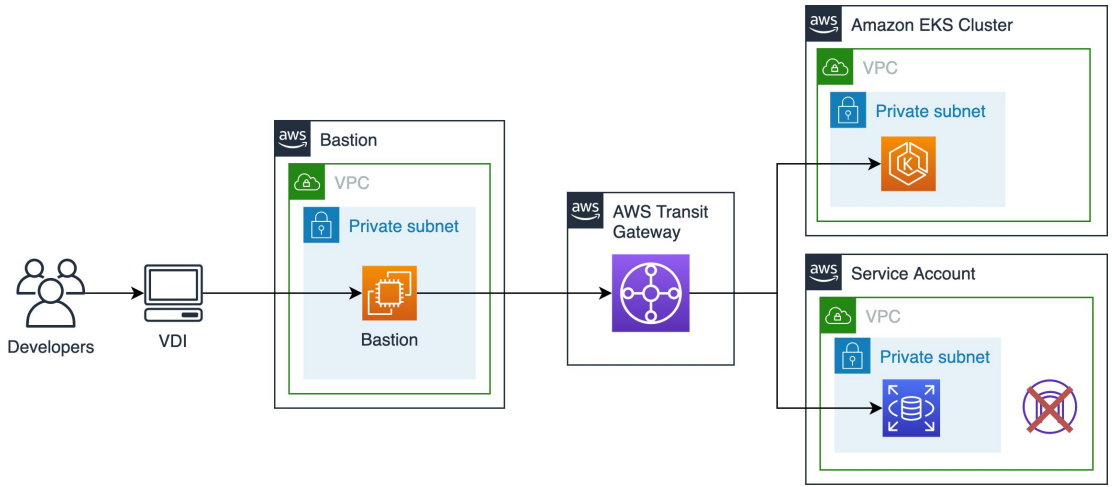


Network ガバナンス 1 / 2

- マネーフォワードには本番環境を触るための VDI 環境が存在
  - うっかり本番環境への他の経路が開いてしまうことは避けたい

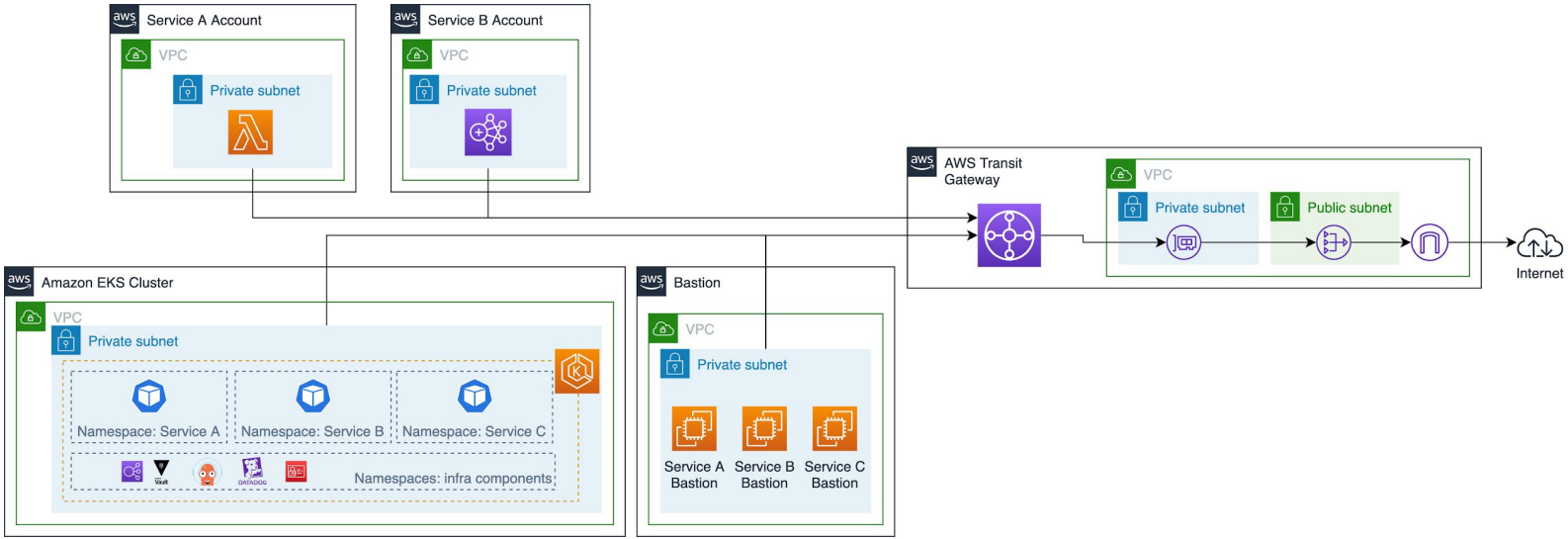
各サービスアカウントに対して Internet Gateway の作成を (基本的に) Deny

本番環境へのアクセスは全て Bastion アカウント経由で行うことに



Network ガバナンス 2 / 2

- 外部との通信ログも全て記録したい
- 外向き通信の出口を Egress Gateway VPC に集約
  - アプリケーションや Bastion から外に行く通信は全て記録
  - セキュリティアプライアンスへの移行も可能な設計に



自由さとガバナンスの折衷案のアーキテクチャ

- EKS クラスタは共通でも、サービスチームは Kubernetes 及び AWS を自由に触ることができるアーキテクチャを実現
- 但し、完全な自由ではなく制限された自由になっている
    - ガバナンスのためにやってほしくないことはアーキテクチャで制限

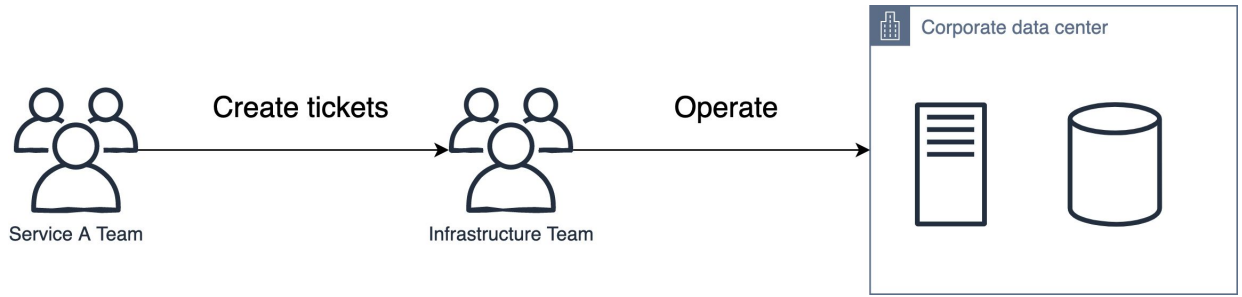
セキュリティレベルを保つために制限を入れつつも最大限の自由を

自由さとガバナンスの折衷案をとったアーキテクチャを採用

開発フローの変化

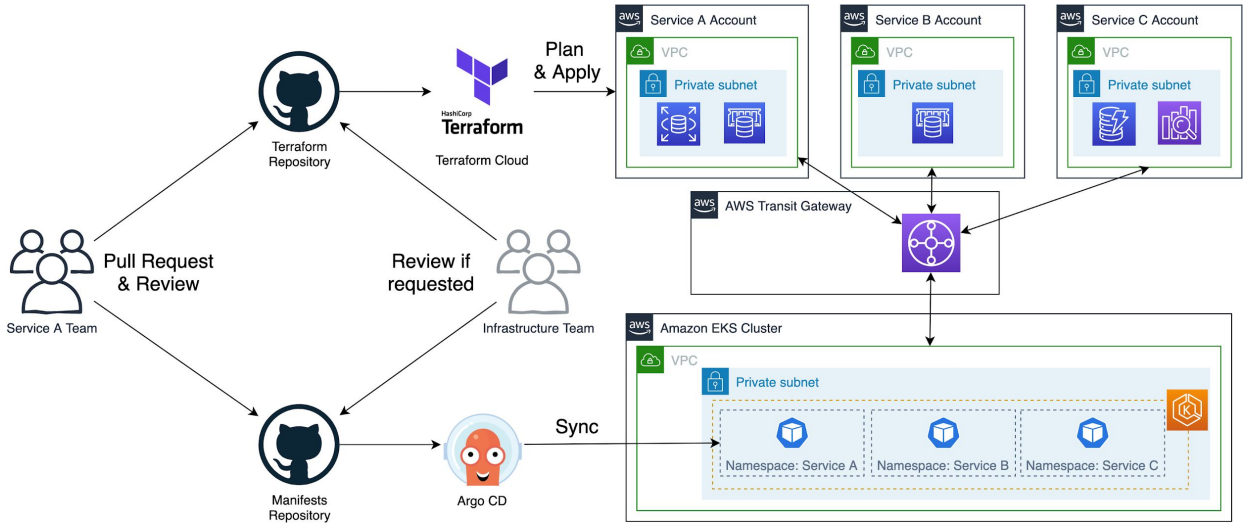
開発フロー : 移行前

- インフラに関する変更は全てインフラチームを経由
  - 依頼フォームからチケットを切ってもらい、それを元に作業
  - 新規サービスの追加、新規プロセスの追加、Ruby バージョンアップ、サーバーの増強、etc ...



開発フロー : 移行後

- 各サービスチームが PR を作成し、チーム内で Review & Merge
  - インフラチームはリクエストされればレビューを実施
- AWS に対する変更は Terraform Cloud を用いて Terraform の Plan & Apply
- Kubernetes に対する変更は ArgoCD を利用



※記載されている会社名および商品・製品・サービス名（ロゴマーク等を含む）は、各社の商標または各権利者の登録商標です。


カルチャーを変えていく為に

カルチャーを変えていく必要性

- 今までの「触ることができない」インフラから「触ろうと思えば触ることができる」インフラには徐々に変わりつつある
- しかし、サービスチームが「触ろうと思えば触ることができる」と「実際に触っている」の間には大きなギャップが存在する
  - 今までインフラチームが全てを見ていた環境からであれば尚更
  - よりスケールする組織にするためには「実際に触っている」ことが必要

インフラのアーキテクチャだけでなく

組織のカルチャーも変えていかなければならない

カルチャーを変えるためにどうするか

- 徐々に慣れていってもらうことを目標に以下のようなことに取り組んでいる
  - Amazon EKS へ移行したチームの島に常駐して質問しやすい環境作り
  - 積極的なナレッジシェア。Zoom 等での社内 webinar を実施
  - 新規コンポーネント 等の追加やオペレーション時にはペア/モブプロ
    - コストを払ってでもサービスチームを積極的に巻き込んでいく
  - サービスチームからインフラチームへの留学プログラム
  - 継続的なヒアリング
    - 移行後に感じたペインを把握して、Platform の改善に活かす
  - etc.…
- 漸進的な Technical & Cultural Change を目指して色々と試行錯誤中


まとめ

まとめ

- 組織と事業の拡大に立ち向かうため、マルチテナント Amazon EKS / マルチアカウントアーキテクチャを採用
  - サービスチームにインフラを移譲し、スケールする組織を目指す
- マルチテナント Amazon EKS で組織としての Technical Gap を最小限にしながら、サービスチーム毎の権限移譲は Namespace により確保
- マルチアカウントでサービスチームが AWS を自由に利用できるように
- 開発プロセスの間にインフラチームが挟まらなくても良い仕組みに
  - 実際に今後組織のカルチャーを変える努力をしながら、インフラチームが挟まらない状態を目指す

We’re hiring!