ビッグデータ分析・活用基盤における Kubernetesの活用とEKS

2020/03/20

株式会社NTTドコモ
サービスイノベーション部
ビッグデータ担当
依田玲央奈

本日のトピックス

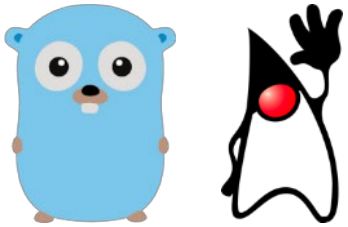
- ドコモにおけるビッグデータ分析/活用について
- NW制約下でのdocker/Kubernetes利用のメリット
- 前処理システムのEKS移行の経緯
- 利用しているツールや、前処理での並列化手法など

自己紹介

よだ　れおな
依田　玲央奈

株式会社NTTドコモ
サービスイノベーション部
ビッグデータ担当

業務: データエンジニア＋α

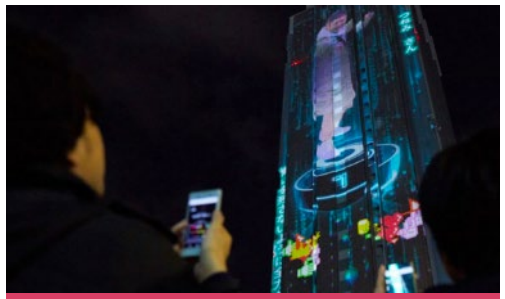


NTTドコモの事業領域

通信事業

通信エリア構築・運用

ショップ販売



5G通信

スマートライフ事業

コンテンツ・コマース

金融・決済

ライフスタイル

法人ソリューション

→大量・多様なデータが生成

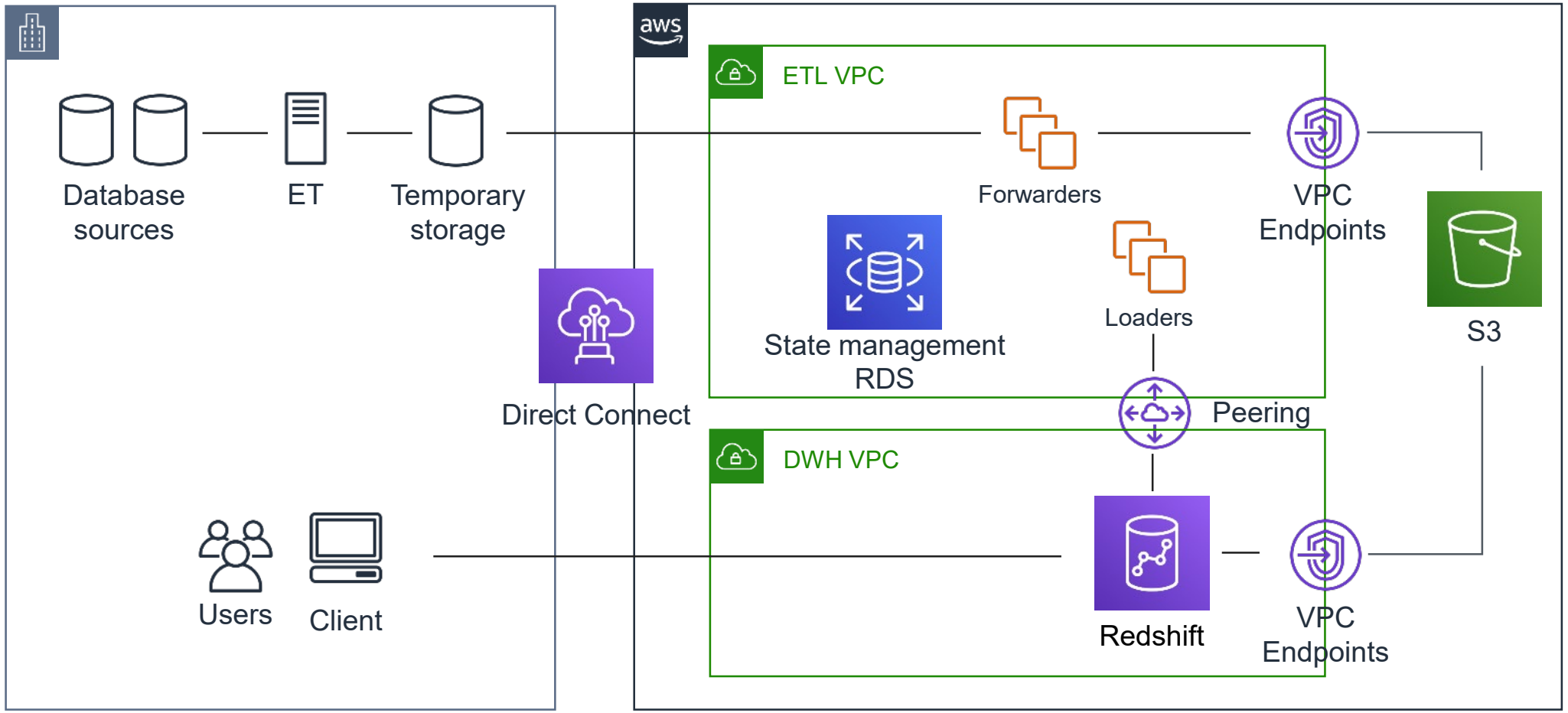
ドコモのデータ分析/活用基盤

ドコモの膨大なデータを、効率的/横断的に分析/活用できる基盤

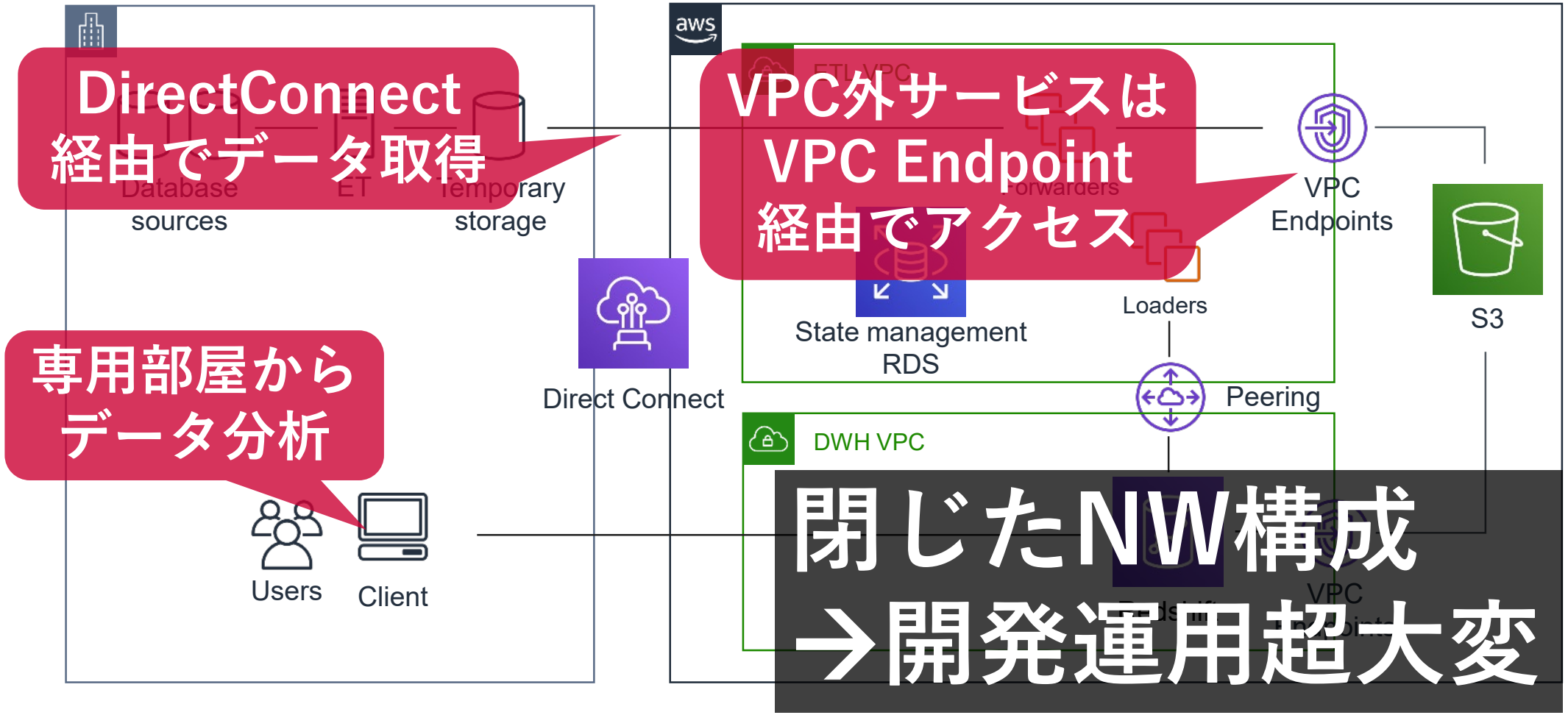
IDAP
INTEGRATED DATA ANALYSIS PLATFORM

- **データサイズ　数PB**
  数十TB/day
- **分析者　数百名**　データサイエンティストから、分析を専門としない人まで
  開発運用　2pizza x 2チーム
- **オンプレ+クラウド**
  専用線を利用し**閉域網**で構成

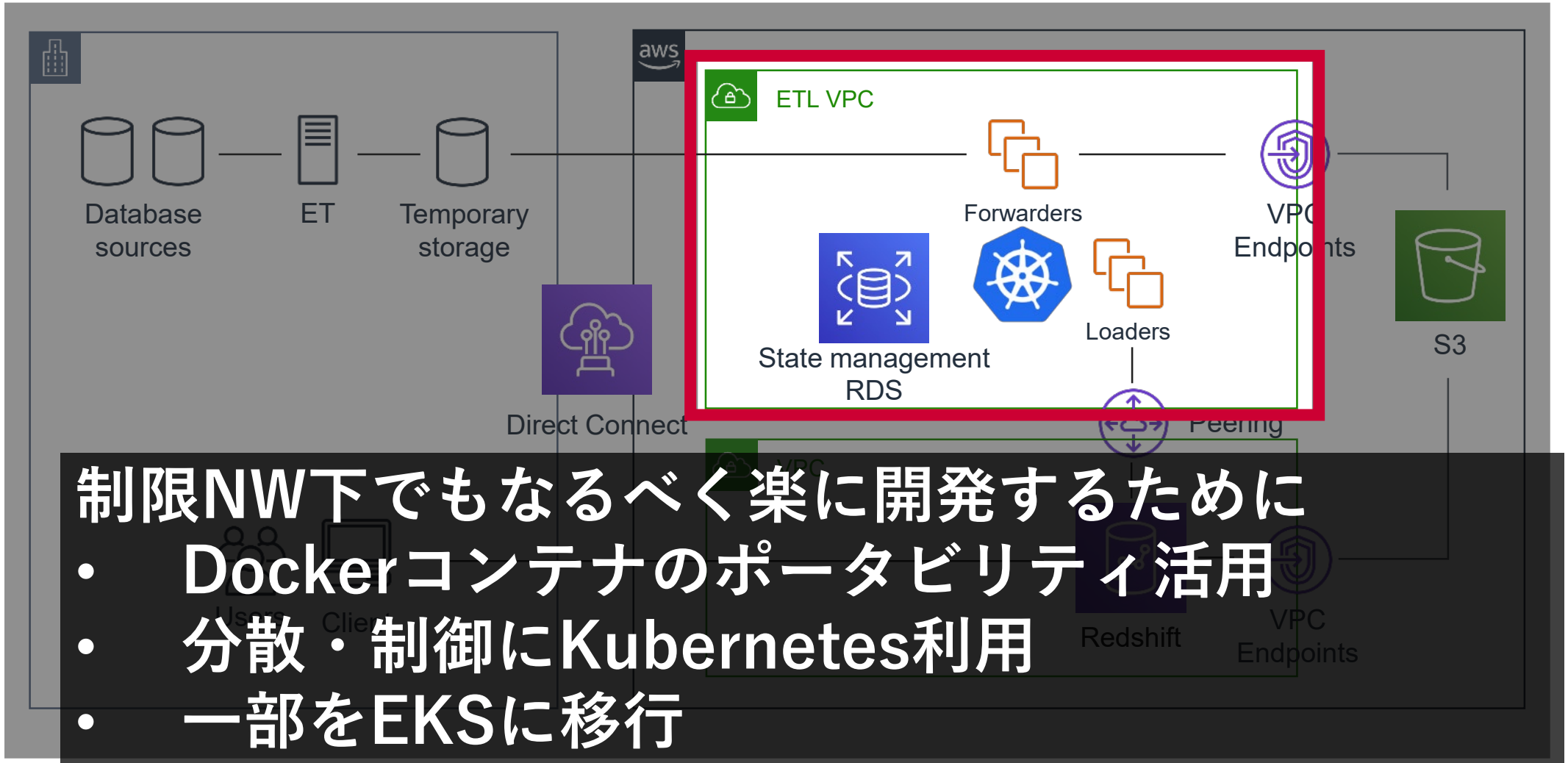
アーキテクチャ概要



外部接続不可！



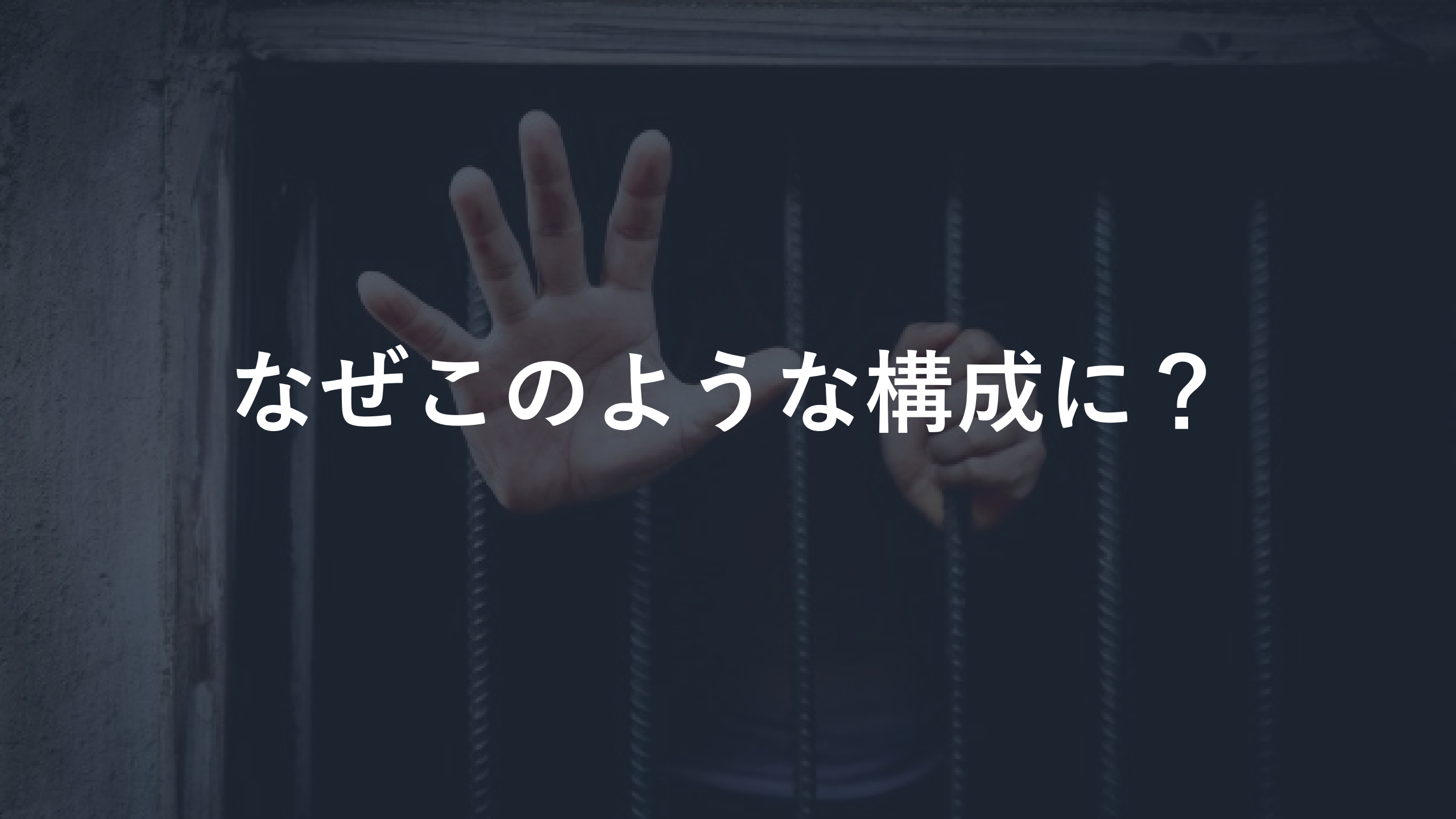
本日のお話：前処理部分の開発



制限NW下でもなるべく楽に開発するために

- Dockerコンテナのポータビリティ活用
- 分散・制御にKubernetes利用
- 一部をEKSに移行

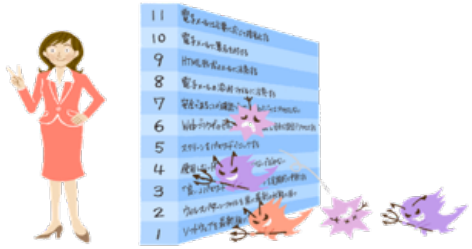
をしてるというお話です


なぜこのような構成に？

ドコモのセキュリティ基準

社内規定類
- 情報管理規定
- 情報管理細則
  不正行為禁止
  セキュリティ対策基準
  サイバー攻撃対応
- 情報管理マニュアル
- 顧客情報管理マニュアル

etc...



具体的対策項目
- 内部・外部ログ
- 権限付与ルール
- アカウント管理
- 入退室管理
- データ暗号化
- 情報セキュリティ監査

etc…

200項目以上

オンプレ・クラウドともに各種規定に
準拠した上で開発する必要あり

クラウド利用時、特に重要なポイント

- **国内のデータセンターを利用できるか**
  - 東京/大阪リージョンでサービスが提供されているか
- **クラウドプロバイダが、適切な情報セキュリティ管理策が実施しているか**
  - SOCやISOなど第三者認証が取得されているか
- **適切な経路で通信可能か**
  - 特に重要なデータの場合、Internetを経由せずに送受信できるか

NWが制限されていると

メリット

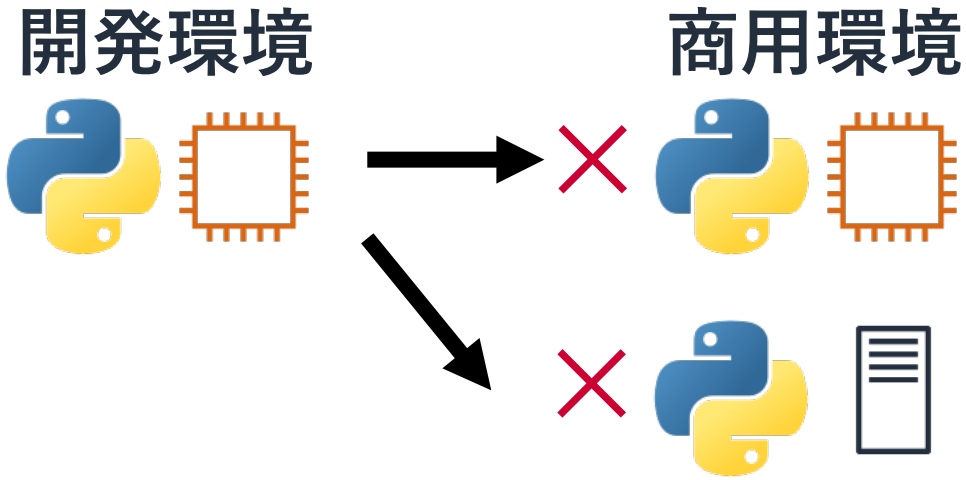
- 外部からの攻撃リスク↘
- 内部からの情報流出リスク↘
  初心者が“やらかす”確率の低減

初心者でも安心してチャレンジできる、と言う点でかなり重要なポイント

デメリット

- 専用設備のコスト↗
- **開発の煩雑さ↗**

例: 開発環境では動いたのに



- NW
- IAM, SG, … (AWSの場合)
- **Python自体**
- **ライブラリやミドルウェア**

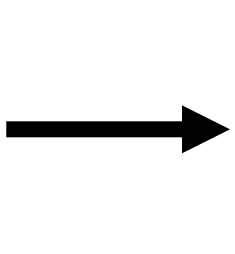
依存関係の解消にInternetを利用不可

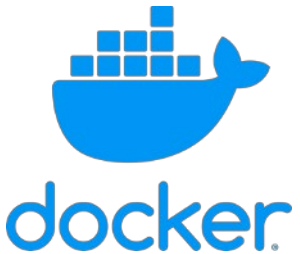
→ 問題の特定・解決までのサイクルが鈍化

これをシンプルに解決したい…!

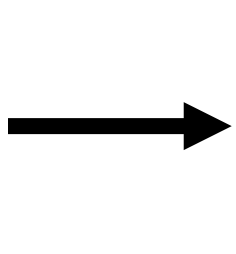
dockerコンテナの活用

外の環境で
開発・試験





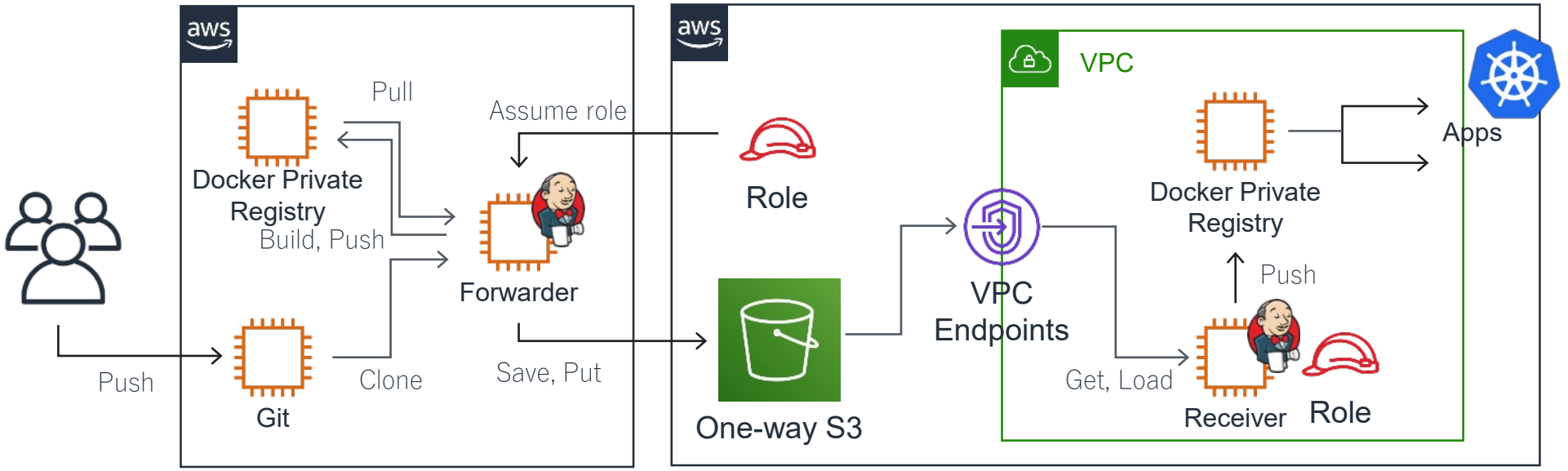
docker save/exportで
コンテナごとtarballに



商用環境に
デプロイ

開発環境で完成したアプリケーションを
そのまま商用環境で走行

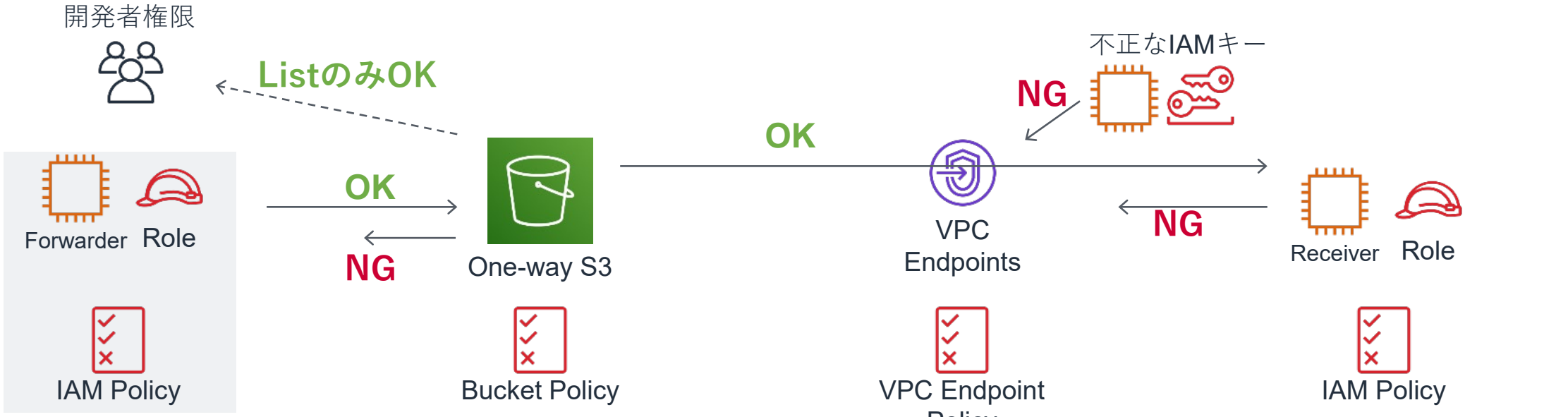
ポリシーを工夫した専用バケットによるコンテナ持込



IAM Policy　BucketPolicy　VPC Endpoint Policy　IAM Policy

4つのポリシーをうまく制御（次スライド）

VPC内への持込のみ可能なS3

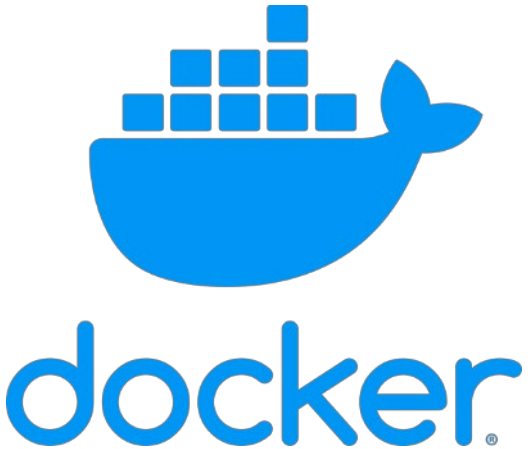


- S3に対するPutを許可
- List系以外明示拒否
- Forwarderロールと Endpoint経由のアクセスを例外
- One-way S3に対するGet/List系を許可
- One-way S3に対するGet/List系を許可

持ち出しの可能性を排除しつつS3利用が可能

EKSへの移行のてんまつと運用形態

ドコモでのコンテナ利用の経緯



2015　docker

- **環境の共通化**
  - 複数台サーバでも
  - オンプレでも、クラウドでも

  →管理コスト減

- **使ってみたかった**

当時の担当者談

ドコモでのコンテナ利用の経緯

2016　Kubernetes

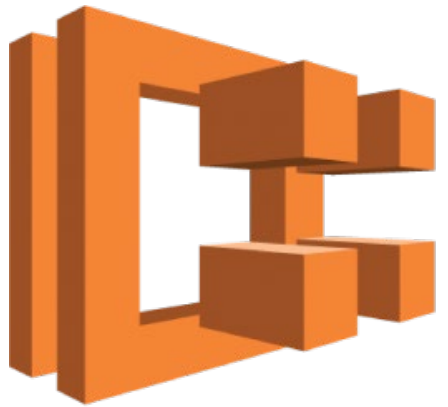
- **複数ノードの管理を効率的に**
  - Swarm/Mesosなど比較検証の結果k8sを採用
  - 「良いにおいがした」

当時の担当者談

比較的早い時期からdocker/k8sを利用

なぜECSを選ばなかったか？

- 東京リージョンなし（当時）
- 第三者認証なし（当時）
- VPC Endpointなし（当時）



単純に要件に合わなかった

K8sクラスタ構築ツールの利用

独自の構築方法
→ メンテナンスに難(k8sのバージョンアップ等)

大規模ETLシステム開発を機に、
クラスタ構築ツール利用

- **Kubernetes直下プロジェクト**
- **GitHubスター数**
  を鑑みKopsを選択

したは良いものの

セキュリティ要件との相性の悪さ

クラスタの構築/アップデート時に

- **Internet接続が必須**
  - 構築時に一時的にInternet gatewayを付与する必要
- **S3/VPC/IAMなどのFullAccessを要求**
  - ソースコードを読んで権限を絞る必要が発生
- **EBS暗号化オプション未対応**
  - カスタムAMIを作成する必要

結局構築/運用作業が複雑化・属人化…

EKSを利用できなかった理由（当時）

- 東京リージョン未対応
- 第三者認証なし
- 構築時Internet接続必要
  （ECRのVPC Endpoint未対応）

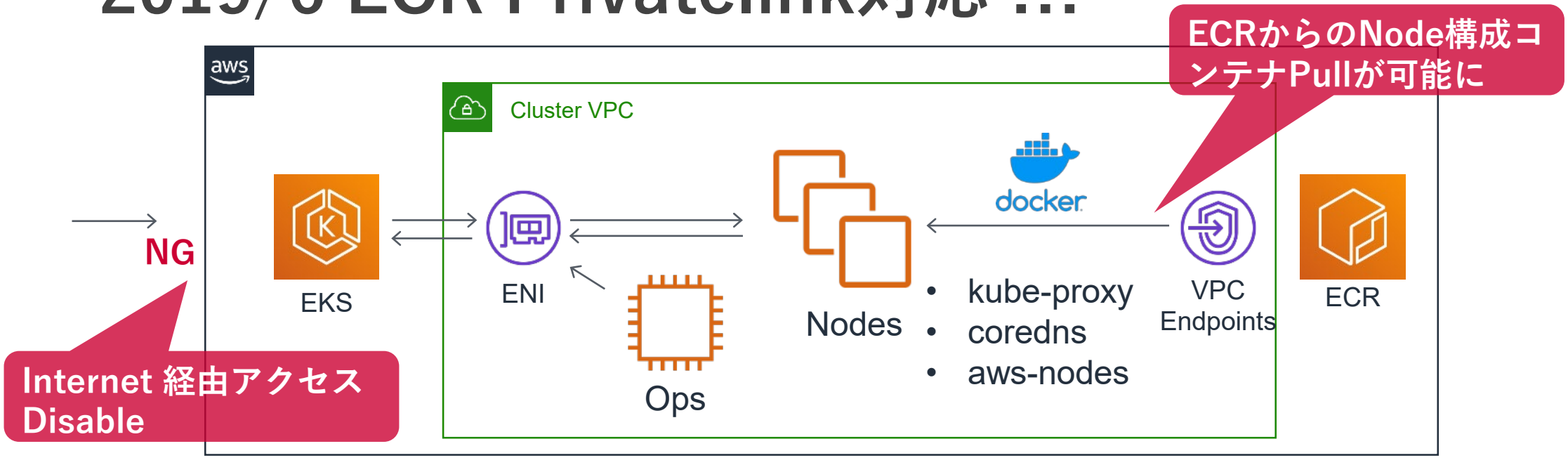
※無理やり使えなくはなかった

EKSの進化

- **2018/12 東京リージョン!**
- **2019/1 ISOの認定取得!**
  - 2019/5　SOC認定祭りの際にSOC123も
- **2019/3 エンドポイントアクセスコントロール!**
  - kubectlの実行をVPC内のみに制限可能に

EKSの進化

- 2019/6 ECR Privatelink対応 !!!



発表されたその日から全力で移行検証開始

全雑務上司押付男

移行の際注意したこと①

- 配布されているCloudFormationをカスタマイズ
  今で言うUnmanaged nodesのテンプレート

Nodes

- Public IP設定除去
- PlacementGroup 設定

Volume

- 暗号化オプション追加

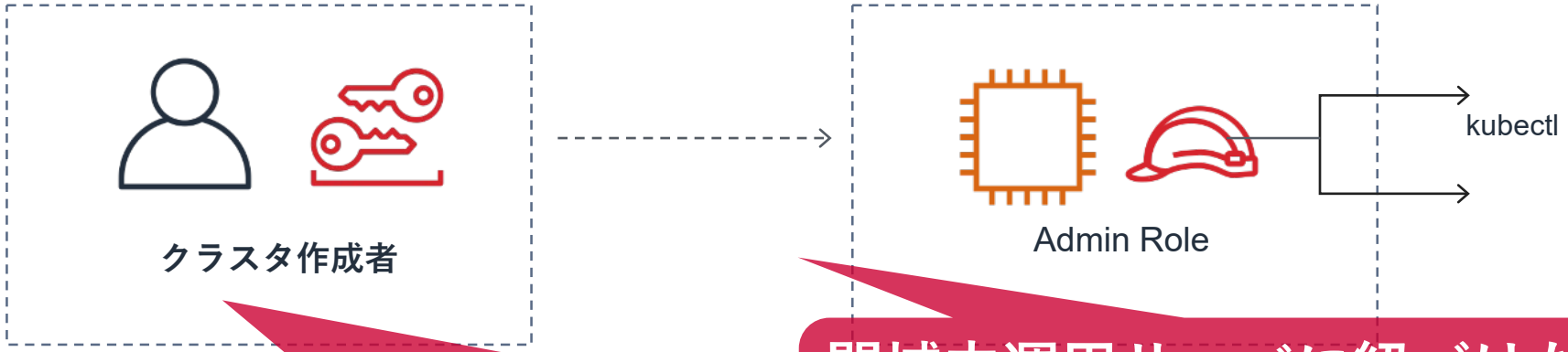
Role

- 分離し明確化

必要に応じ細かな設定可能な点が◎

移行の際注意したこと②

- 登場人物とIAMの整理



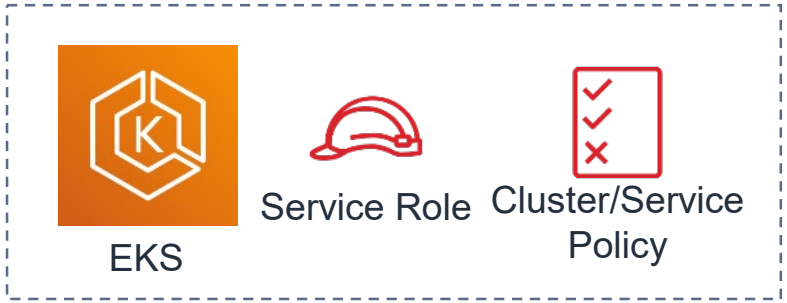
デフォルトで
クラスタ全操作権限保持
→管理用IAM（厳重管理対象）で作成後、基本利用せず

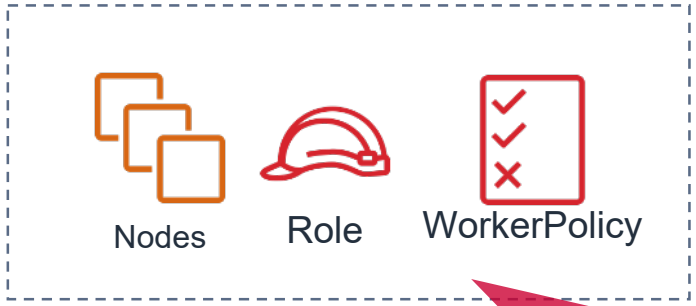
**閉域内運用サーバに紐づけた
ロールにクラスタ管理権限委譲**

この際IAM Policyは空でも良いのが
面白い？ポイント

移行の際注意したこと②

- 登場人物とIAMの整理





2つの管理ポリシー必須でカスタム不可
→ ServiceやIngressにより自動で
IGW付与されないかチェック → OK

ECRからのPull権限が必要

VPC内に操作限定できることを確認

結果

移行自体はスムーズに完了

- RBACがEnableになるため一部対処したくらい

CFn(+ CLI)走行させるだけで商用と同じ環境が作成可能に

- Private AccessやLoggingの設定はCFn対応せず
- 検証環境はVPC PeeringやSSMにより擬似的に閉域再現

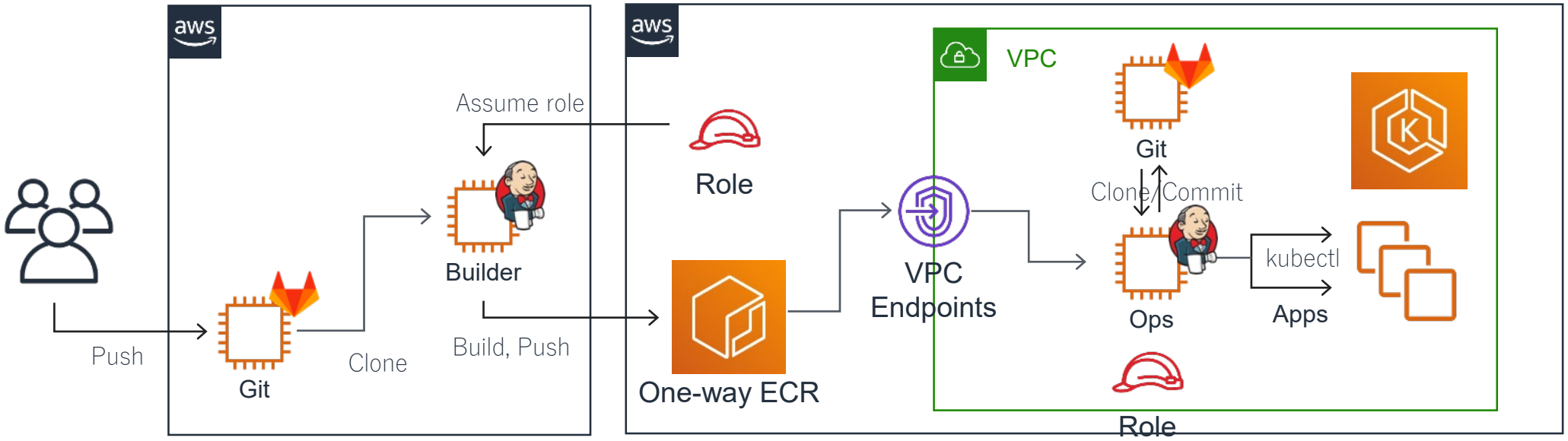
実際の作成手順について

（簡略化版を）Qiitaに置いてあります。

「ドコモ　サービスイノベーション」
でググっていただければ
ML～インフラまでごった煮ですが

https://qiita.com/organizations/nttdocomo_service_innovation

ECRによるデプロイチェーン簡略化

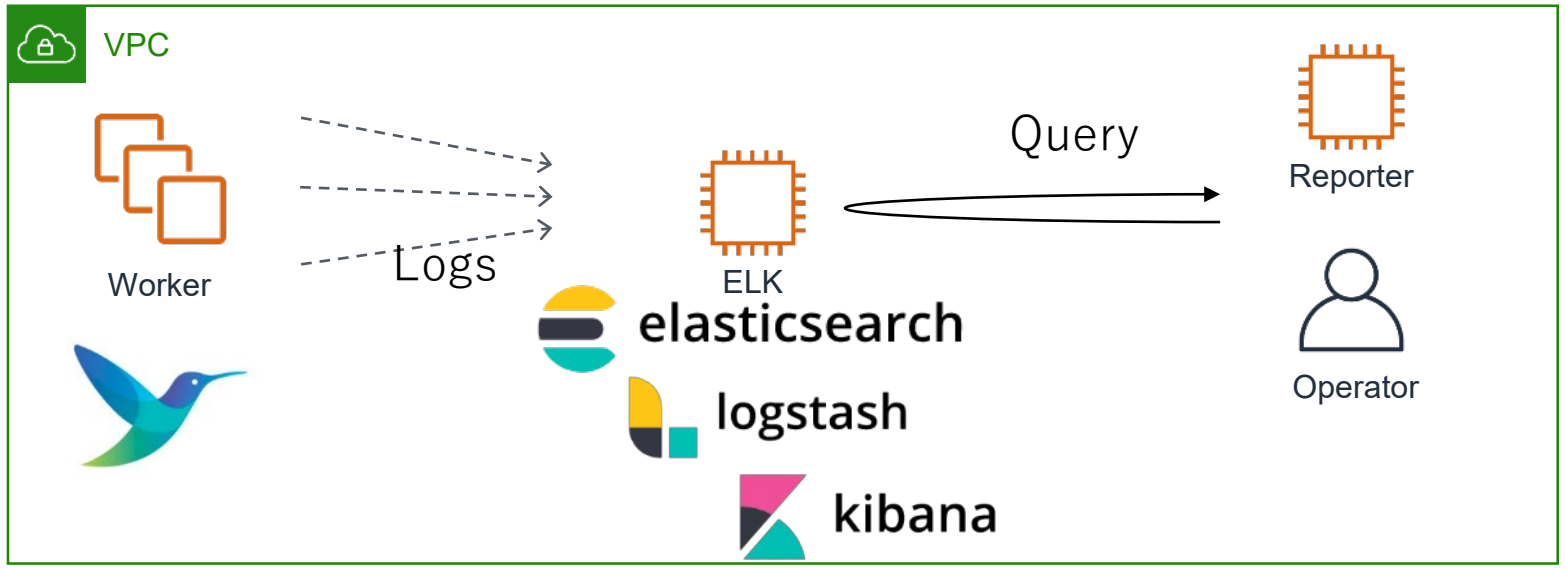


S3と同様の手法をECRに適用

2019年はポリシー大拡充の年でした

アプリケーションログ

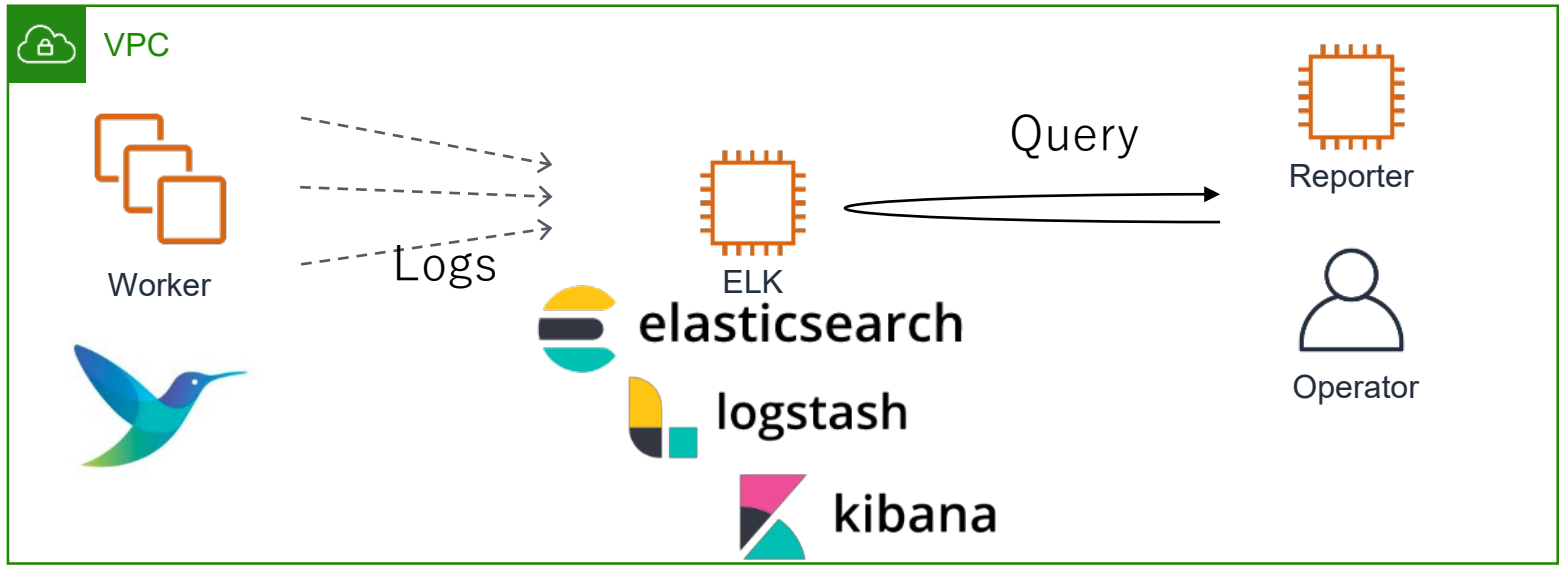
Fluentbit + ELK



- ドキュメントの通りDeamonsetを適用するだけで簡単に動作
- データをVPC内に留めたいのでELKをコレクタに

アプリケーションログ

Fluentbit + ELK



- ドキュメントの通りDaemonsetを適用するだけで簡単に動作
- データをVPC内に留めたいのでELKをコレクタに

Fluentbitの利点

Fluentdに比べメモリ使用量減

CloudWatch Plugins: Fluentd vs Fluent Bit

Log Lines Per second	Data Out	Fluentd CPU	Fluent Bit CPU	Fluentd Memory	Fluent Bit Memory
100	25 KB/s	0.013 vCPU	0.003 vCPU	146 MB	27 MB
1000	250 KB/s	0.103 vCPU	0.03 vCPU	303 MB	44 MB
10000	2.5 MB/s	1.03 vCPU	0.19 vCPU	376 MB	65 MB

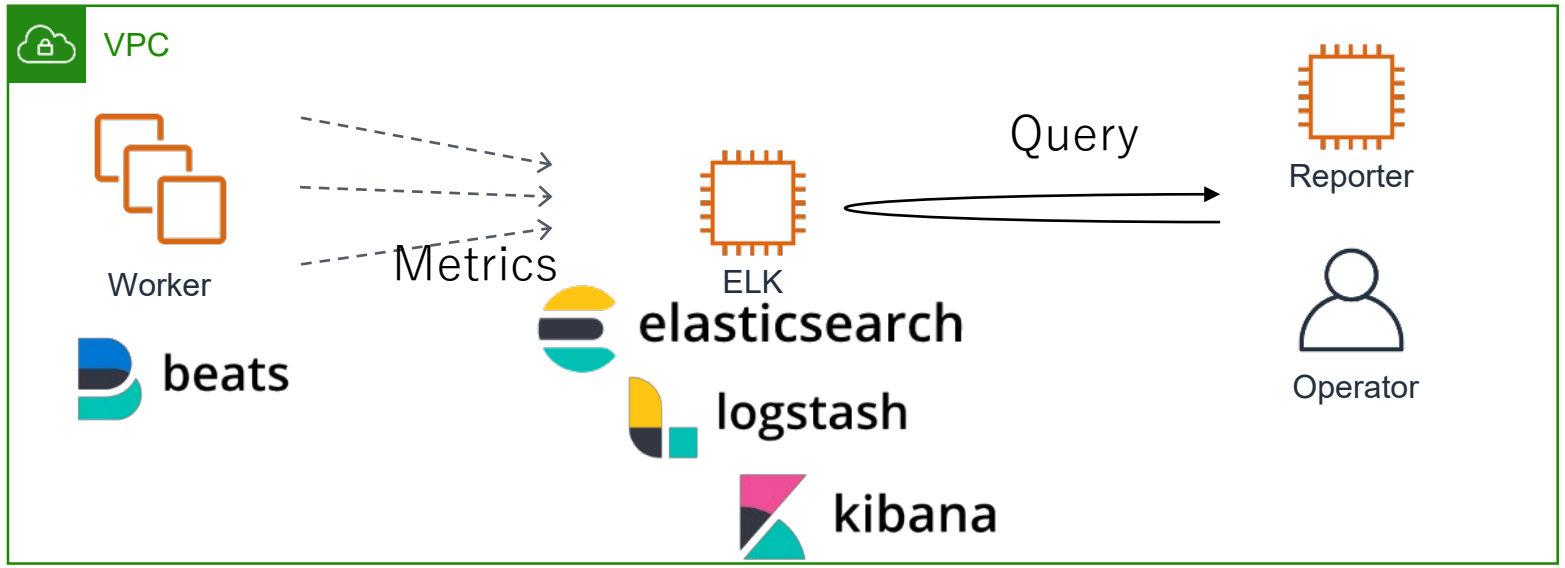
https://aws.amazon.com/jp/blogs/opensource/centralized-container-logging-fluent-bit/ より引用、
私の体感もおなじくらい

なるべくアプリケーションコンテナに
メモリ割り当てたい状況でもいい感じに動作

オススメです

使用リソースモニタリング

- Metricbeat（Elasticの軽量シッパー）



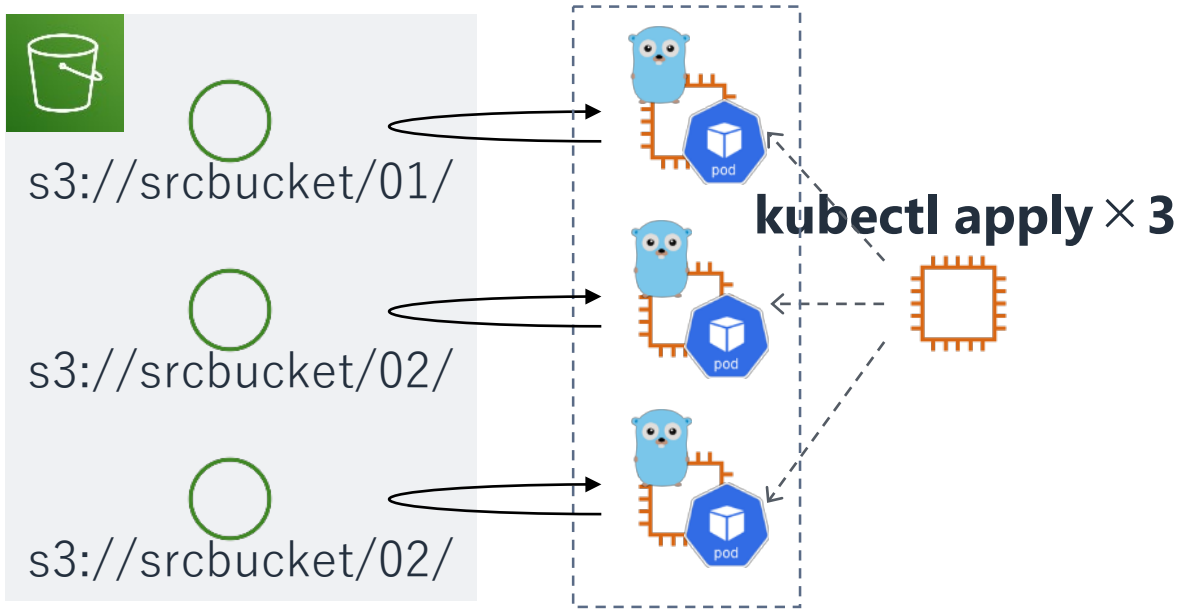
ログ収集に建てたEKLを再利用

- jolokia module でのJVM情報の集約も

Kubernetesによる処理の並列化について

K8sを活用した並列処理手法①

処理対象を分割しそれぞれにPod割当



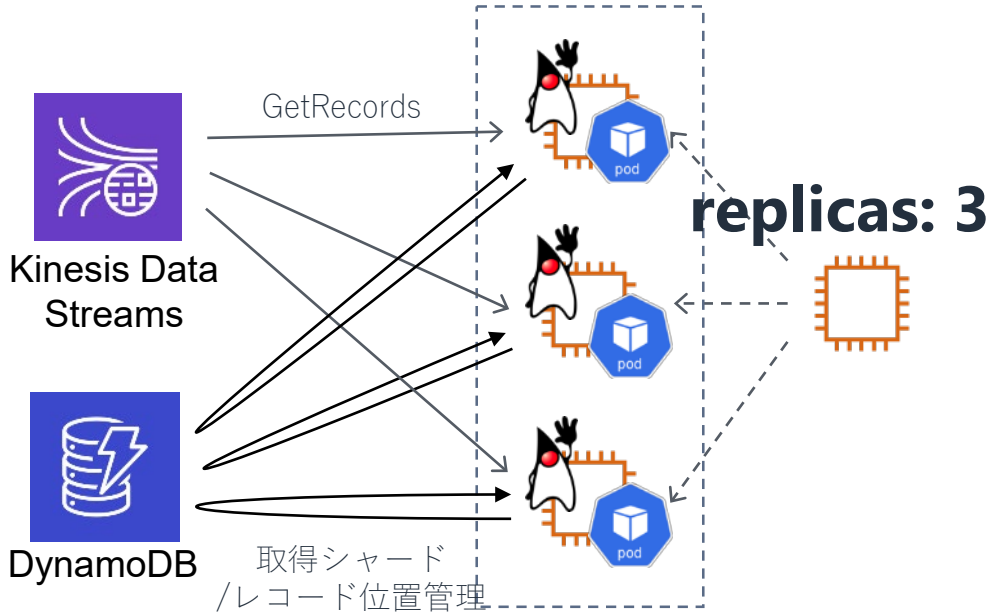
担当範囲をenvなどで渡しPod起動

【長所】
- **シンプル**

【短所】
- データ分割の必要
- リトライなど複雑な処理に難

K8sを活用した並列処理手法②

状態管理DBの利用



RDSやDynamoで状態管理

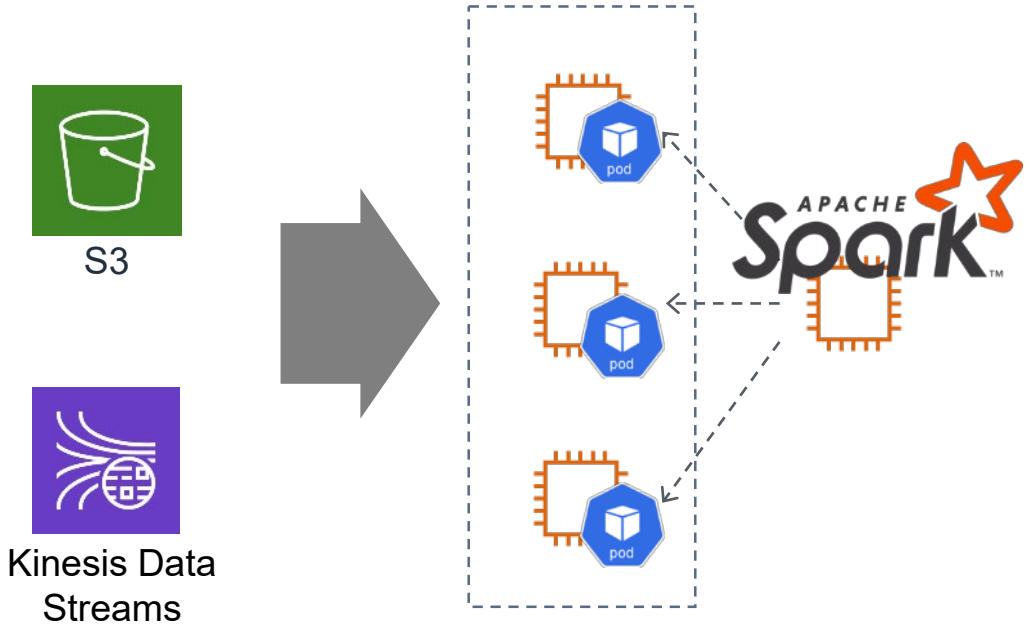
※ストリーム処理の場合左図のように Kinesis Client Libraryが便利

【長所】
- **Pod数増減での並列度調整**
- **リトライなどの複雑な制御**

【短所】
- アプリケーション作り込み
- 管理対象増

K8sを活用した並列処理手法③

分散処理エンジンの利用



Apach Spark, Apach Flinkなどの
K8sをサポートするエンジンを利用

【長所】
- **複雑な処理も簡潔に記述**
- 高速

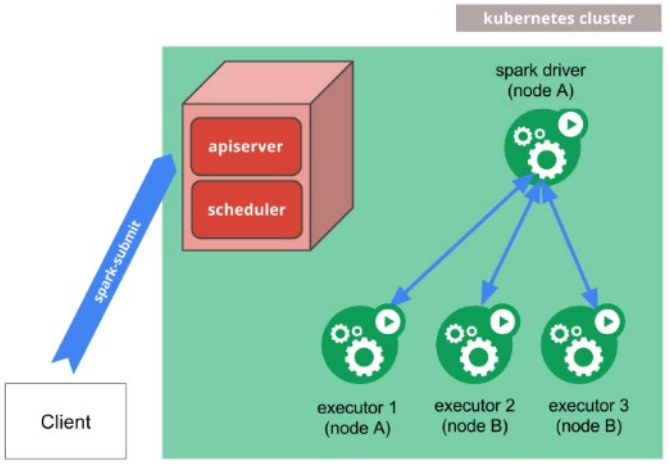
【短所】
- 学習コスト
- K8s実行環境の習熟度

どのパターンも、共通基盤上で動かせるのがK8sの強み

Spark on k8s

K8sをリソースマネージャに利用可能

- K8sのインフラを活用しつつ
  Sparkの強力な分散処理を利用



spark-submitコマンドを実行すると
driver podが起動、次にdriver がexecutor podを
起動しアプリケーションを実行、と言う流れ

YAMLを適用するとは少し違うお作法
Googleは（GCPは）これのOperatorを推進中

Spark on k8sの注意点

- Currently experimental
  - 未実装機能有（Dynamic Resource Allocationなど）
  - 細かな挙動にバグ（log4j.propertiesの上書きなど）
  - 性能面で不利な場合も（YARNに比してのDISK I/Oなど）

開発自体や事例報告は活発なので
- **なるべく最新版を追う**
- **ApacheのJIRAを調べる**
- **KubeconやSpark summitの発表を確認する**

などで大体は解決する、くらいの感覚

本当にあった怖い話

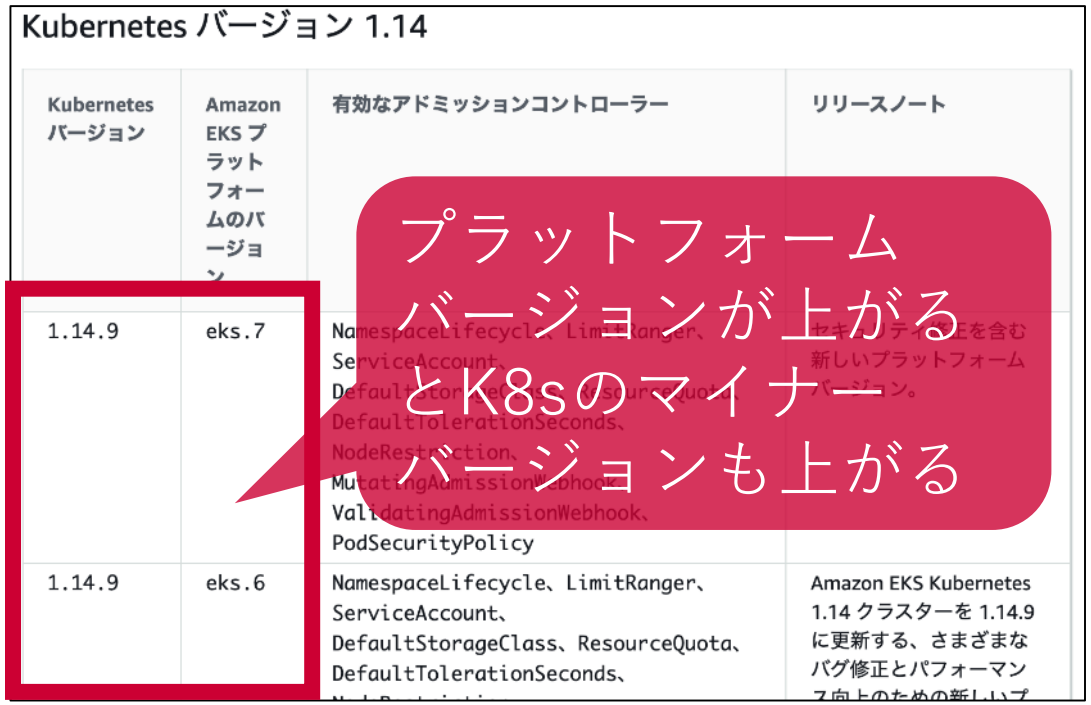
ある日突然Sparkアプケーションが停止

- アプリケーションは全く変更せず
- となりのQA用クラスタでは通常動作

→ EKSのプラットフォームバージョン
自動アップデートが原因

本当にあった怖い話

- Amazon EKS は、既存のすべてのクラスターを、対応する Kubernetes マイナーバージョン用の最新の Amazon EKS プラットフォームバージョンに自動的にアップグレードします。



Kubernetes バージョン 1.14

Kubernetes バージョン	Amazon EKS プラットフォームのバージョン	有効なアドミッションコントローラー	リリースノート
1.14.9	eks.7	NamespaceLifecycle、LimitRanger、ServiceAccount、DefaultStorageClass、ResourceQuota、DefaultTolerationSeconds、NodeRestriction、MutatingAdmissionWebhook、ValidatingAdmissionWebhook、PodSecurityPolicy	セキュリティ修正を含む新しいプラットフォームバージョン。
1.14.9	eks.6	NamespaceLifecycle、LimitRanger、ServiceAccount、DefaultStorageClass、ResourceQuota、DefaultTolerationSeconds、	Amazon EKS Kubernetes 1.14 クラスターを 1.14.9 に更新する、さまざまなバグ修正とパフォーマンス向上のための新しいプ

1. 脆弱性対処のため、Golangのnet/url.Parse()の挙動が修正
2. 新しいGolangでビルドされたK8sのURLリクエストに対する挙動も変化
3. 特殊なリクエストをするSparkが疎通エラーと言うレアケース

- **アップデートタイミングの告知**
- **猶予期間**

くらいは欲しいな…とうお話しでした。

まとめ

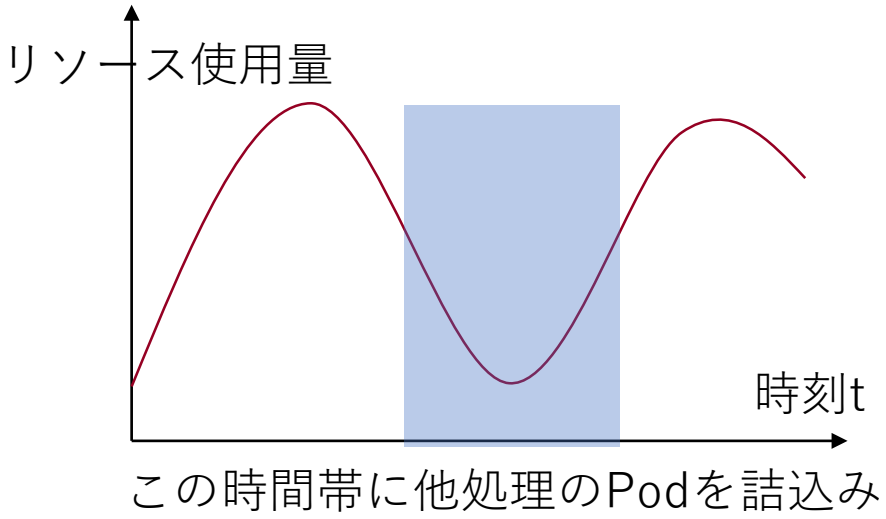
- **制限されたNW下でも、dockerとKubernetesを活用して効率良く前処理システム開発**
  - dockerを活用したアプリケーション持ち込み
  - K8sを土台とした並列処理機構

- **2019年のアップデートの結果、ポリシーに沿う形でEKS移行が可能に**
  - 構築運用まわりの面倒から解放
  - ECRなど周辺サービス活用

今後のとりくみ

リソースの有効活用をしたい

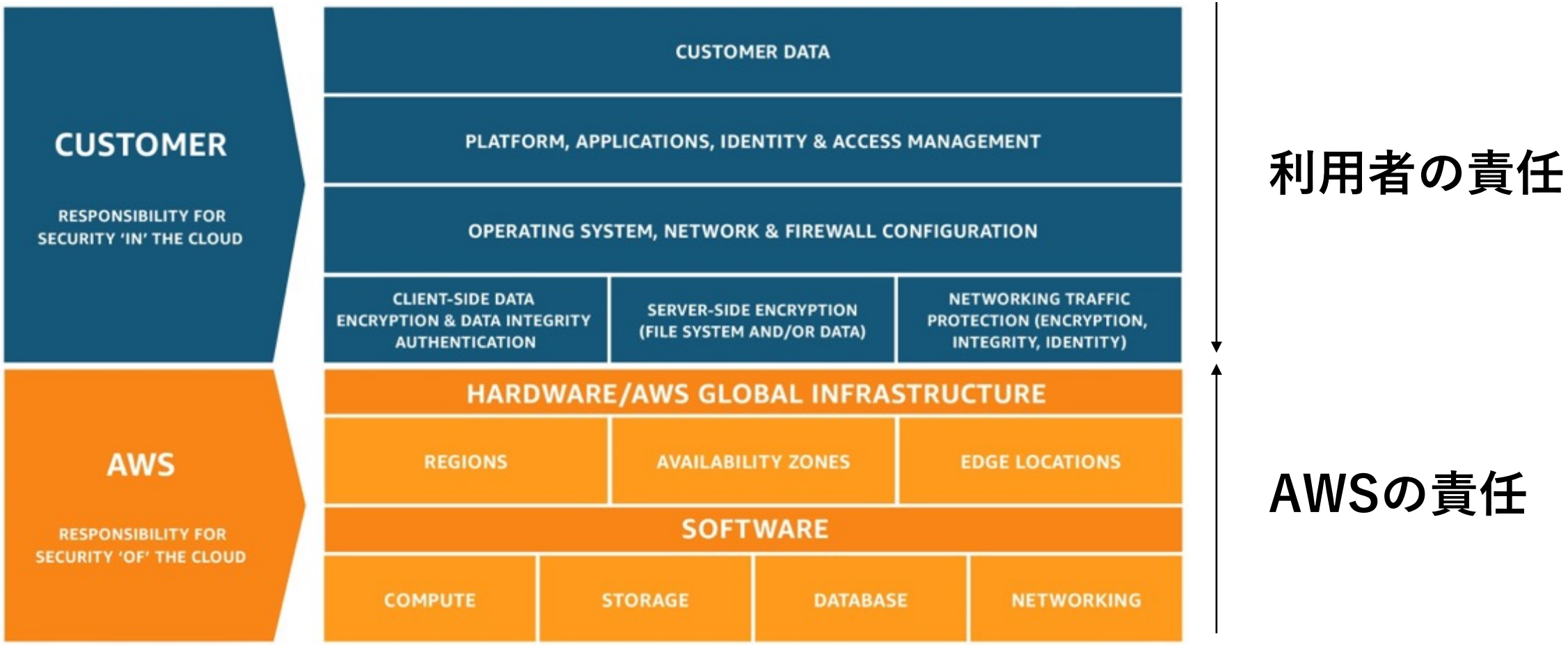
- バッチ処理のFargate利用
- ストリーム処理の変動対処



DOCOMO Innovations, Inc.と
Drupalプラグインで
試行錯誤しています

予備スライド

AWSの責任共有モデル



https://aws.amazon.com/jp/compliance/shared-responsibility-model/

評価ポイント

AWSに実装されていないと
どうしようもないものたち

AWSの責任 ⟵ ⟶ **ドコモの責任**

AWSの責任範囲

- 第三者認証レポート
- ホワイトペーパー
- AWS Artifact
- AWSセキュリティチームへのヒアリング

etc…

利用可能なセキュリティオプション

- IAM
- SG/NACL
- 必要なログの取得可否
- 暗号化オプション

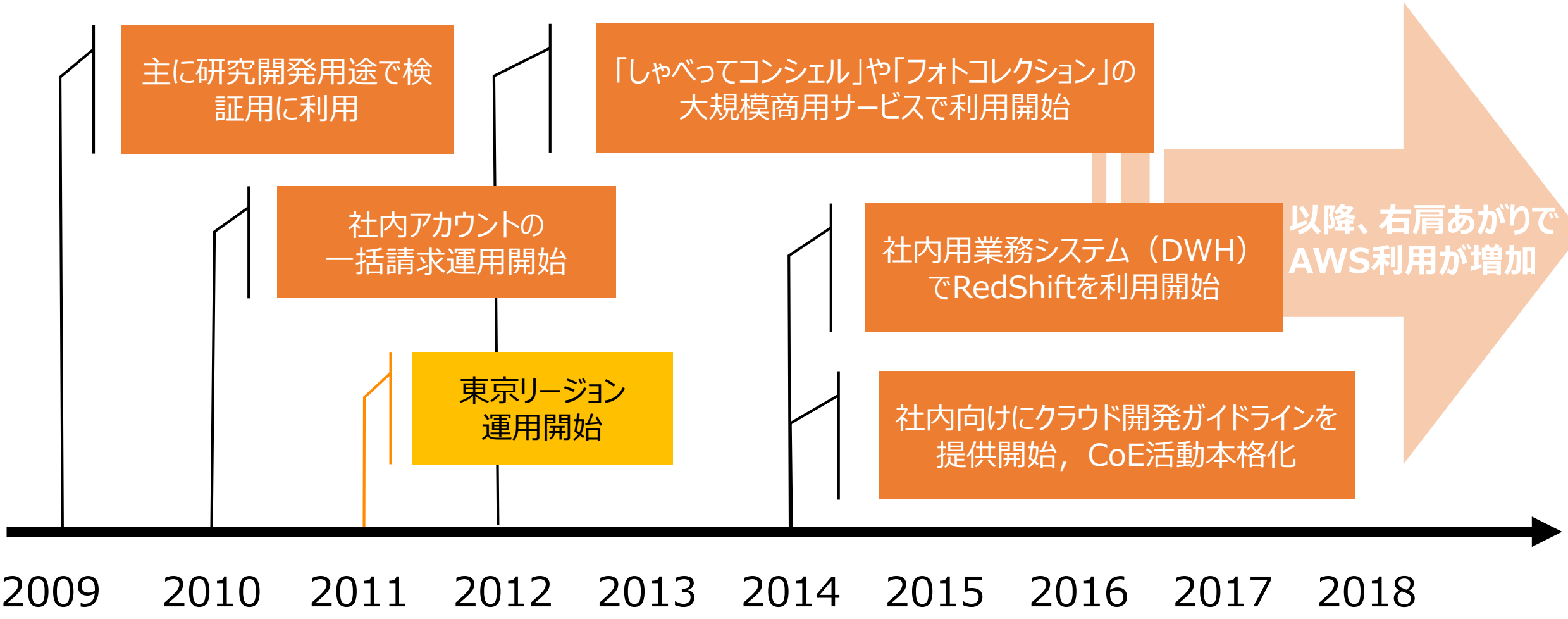
etc…

ドコモが対策すべきこと

- 暗号危殆化
- ユーザデータの管理
- アプリケーションのセキュア設計

etc..

ドコモにおけるAWS利用動向



アイコン

AWS Cloud

Amazon RDS

Amazon Simple Storage Service

/aws.amazon.com/jp/solutions/ s/docomo/

AWS Direct Connect

Amazon Redshift

VPC

Instances

Server contents

Corporate data center

Client

Generic database

Amazon Elastic Kubernetes Service

Generic group

Generic group

Peering connection

Office building

Traditional server

Users