Amazon EC2は使わない！？クラウドネイティブな開発スタイルがもたらすビジネスの加速化

Toshiaki IWAMOTO
Section Manager
Gurunavi,Inc. / Infrastructure Services Section

Agenda

なぜ「ぐるなび」はアマゾン ウェブ サービス（AWS）を導入したのか？

クラウドネイティブがもたらすビジネスの加速化

「ぐるなび」って何？

About ぐるなび

総載店舗数	約50万店
有料加盟店舗数	59,660店
	2019年3月末現在
月間ユニークユーザー数	6,100万人
	2018年12月現在
会員数	1,718万人
	2019年4月1日現在

ぐるなびの事業展開

外部との
連携・協業

支付宝 ALIPAY / Ctrip 携程 / JR東日本 / 樂吃購!日本 TRAVEL GUIDE / kkday / KLOOK / 雄獅旅遊 www.liontravel.com / 美团 / 大众点评 / MS&C 株式会社 MS & Consulting / NTT docomo / PERSOL / Rakuten / SmartNews / 微信支付 / TORETA / tripadvisor / JR西日本 / YOLO JAPAN

（アルファベット順）

外部とのコラボレーションによりサポート領域の拡大を加速

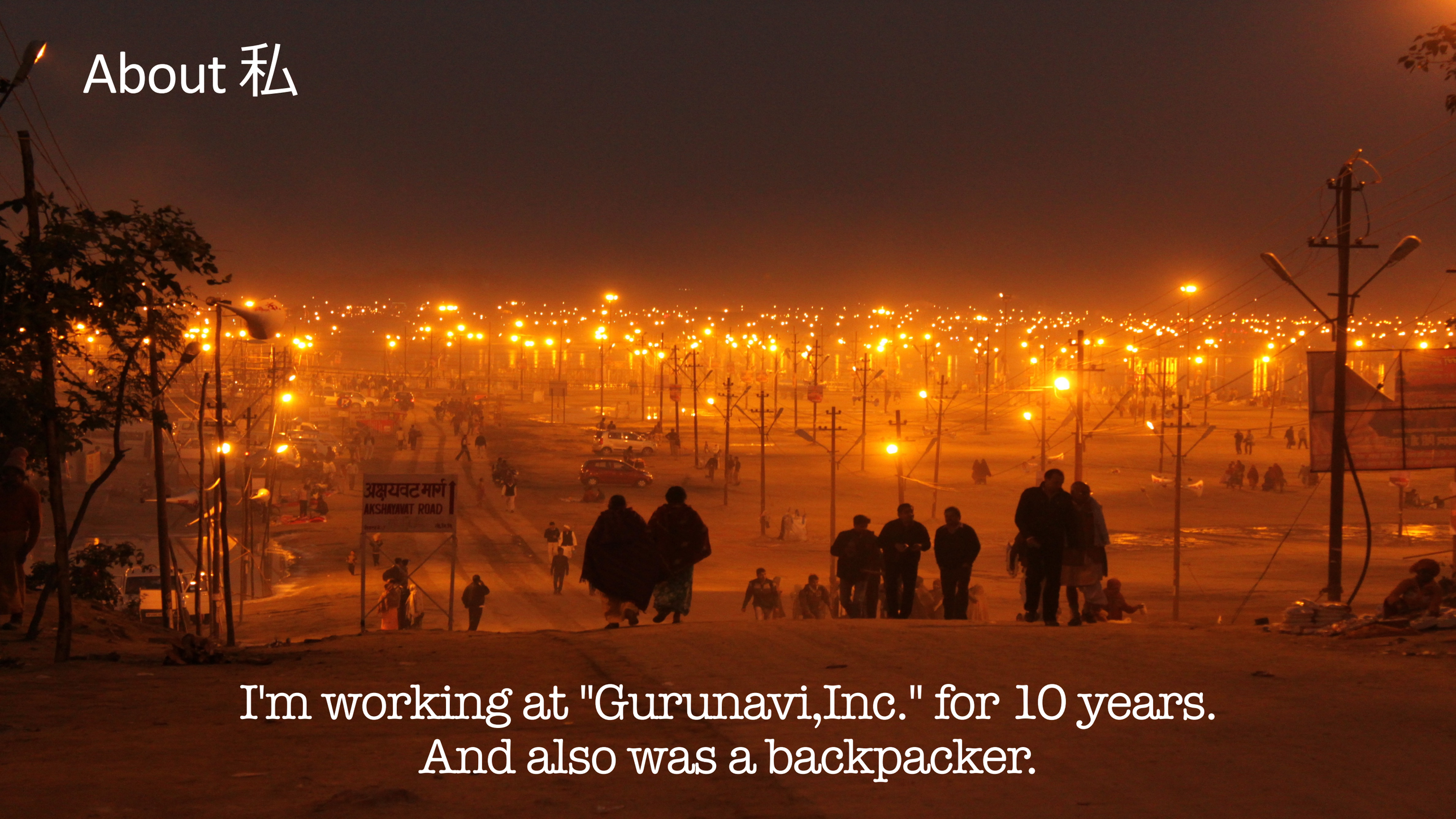
支援領域

ぐるなびサイト集客支援 → 他社連携集客支援 → インバウンド集客支援 → 業務代行支援 → 店内ICT化支援 → 採用育成支援 → ….

飲食店
サポート

※2019年3月期 決算説明会資料より抜粋

About 私



I'm working at "Gurunavi,Inc." for 10 years.
And also was a backpacker.

On-Premises

サービス開始 1996年6月～

- 200+ 提供サービス
- 5000+ Git repositories
- X0000+ Server accounts
- インフラチーム領域
  - 仮想基盤(VM)・コンテナ基盤（K8s/Docker registry）・CI/CD(Jenkins/GitLab)・サーバOS・ミドルウェア・ネットワーク、RDB(MySQL/Oracle)の設計・構築・運用

AWS

利用開始 2018年11月～(半年程度)

- 70+ AWS accounts
- 10+ 公開サービス
- 100+ AWS利用者アカウント
- 3 クラウド管理者

なぜ「ぐるなび」はAWSを導入したのか？

Deploy to Production

300+/month

AWS導入の目的

開発効率の向上

開発効率の向上は何を実現すべきか？

開発/リリースに伴う間接工数を減らす/なくす
リリースサイクルの高速化
予測できないサイジングに時間を使わない
SaaSサービス・フルマネージドサービスを活用する
高いメンテナンス性を実現する

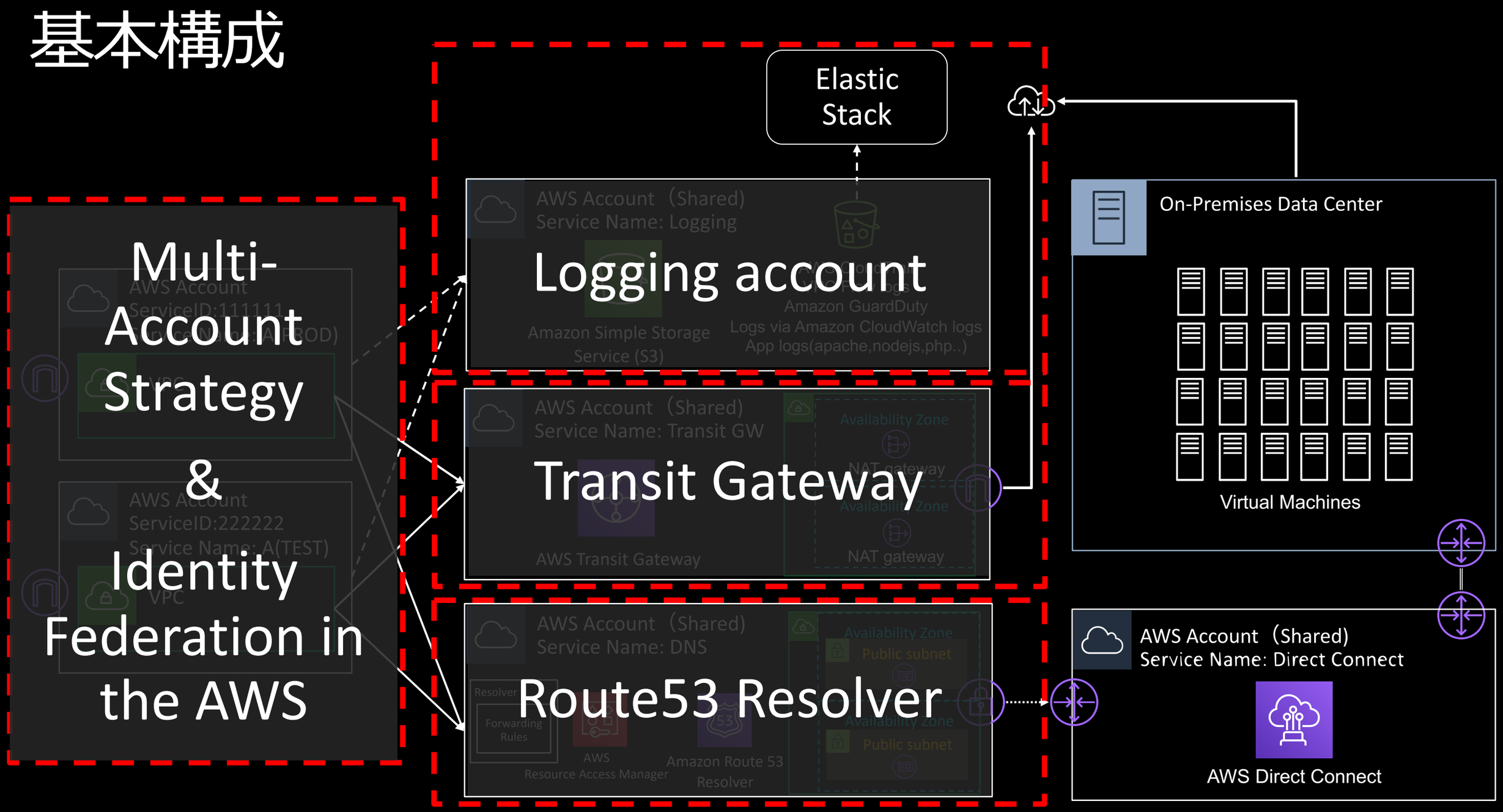
コスト削減ではなくコストを意識する

アプリケーションを維持するためにかかっている費用が適切か知る必要がある。AWSを利用する事で利用しているリソース全ての費用を把握できるようになる。

フルマネージドサービスを選択したり性能を向上する言語の選択を行うことでもっと低コスト・もっと高品質を実現できる

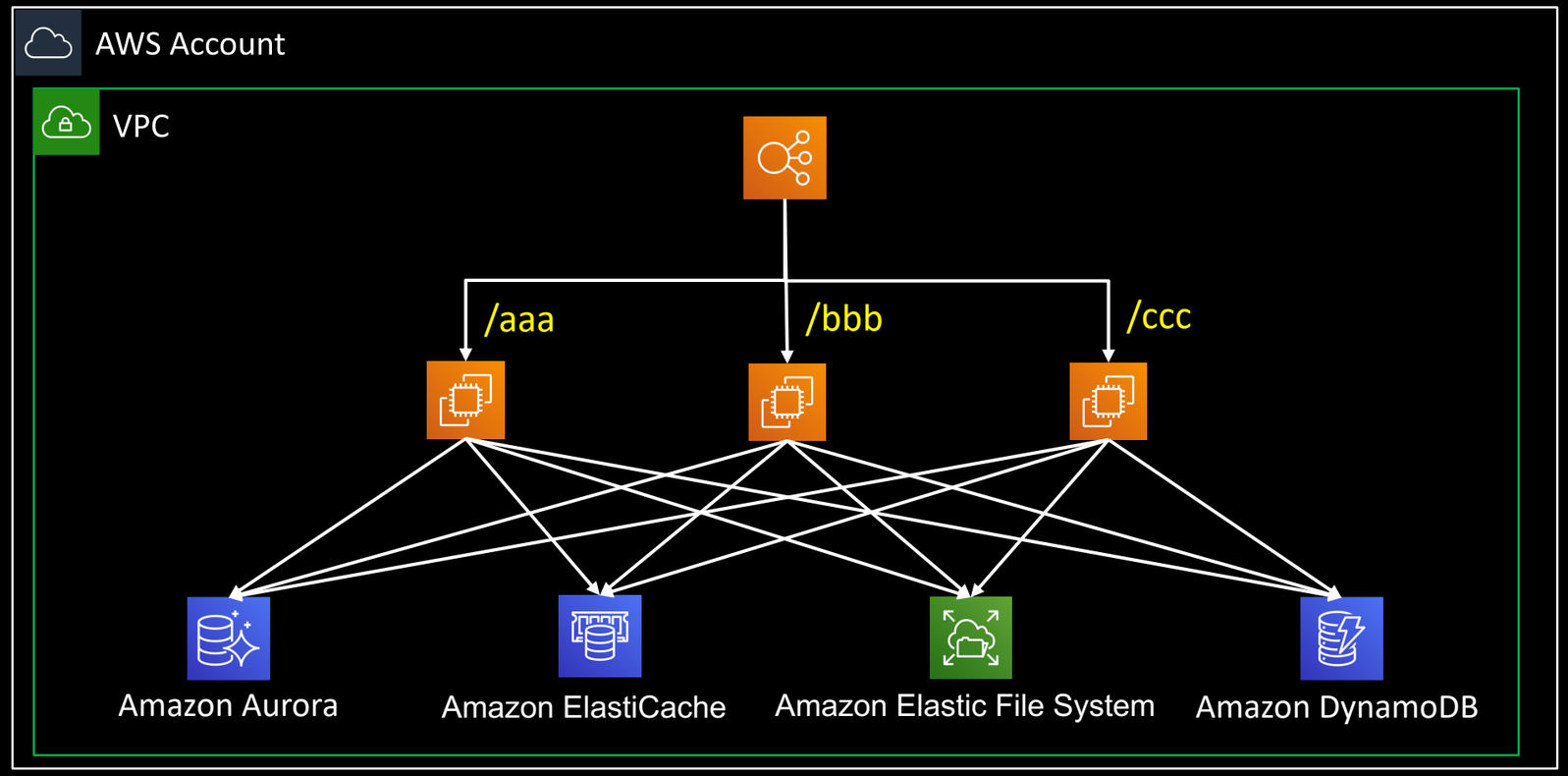
導入し始める前に準備したこと

基本構成


Elastic Stack
AWS Account （Shared）
Service Name: Logging
Logging account
Amazon Simple Storage Service (S3)
Amazon GuardDuty
Logs via Amazon CloudWatch logs
App logs(apache,nodejs,php..)
AWS Account （Shared）
Service Name: Transit GW
Transit Gateway
AWS Transit Gateway
Availability Zone
NAT gateway
AWS Account （Shared）
Service Name: DNS
Route53 Resolver
Resolver
Forwarding Rules
AWS Resource Access Manager
Amazon Route 53 Resolver
Availability Zone
Public subnet
Multi-Account Strategy
&
Identity Federation in the AWS
On-Premises Data Center
Virtual Machines
AWS Account （Shared）
Service Name: Direct Connect
AWS Direct Connect

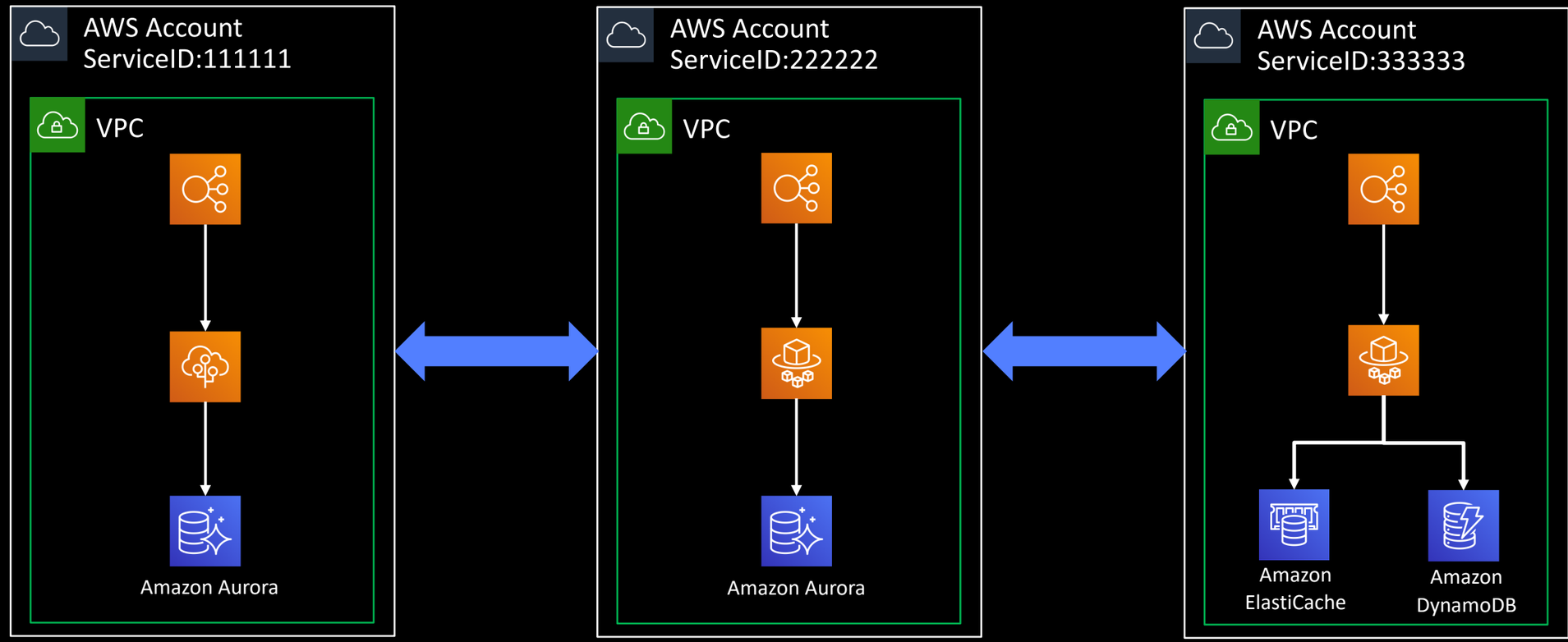
Multi-Account Strategy

モノリシック。データベースやNAS等を共有して利用するパターン。アプリケーションが他のサービスと依存している為、メンテナンス性が悪くサービス単位でのコストが可視化しづらい。



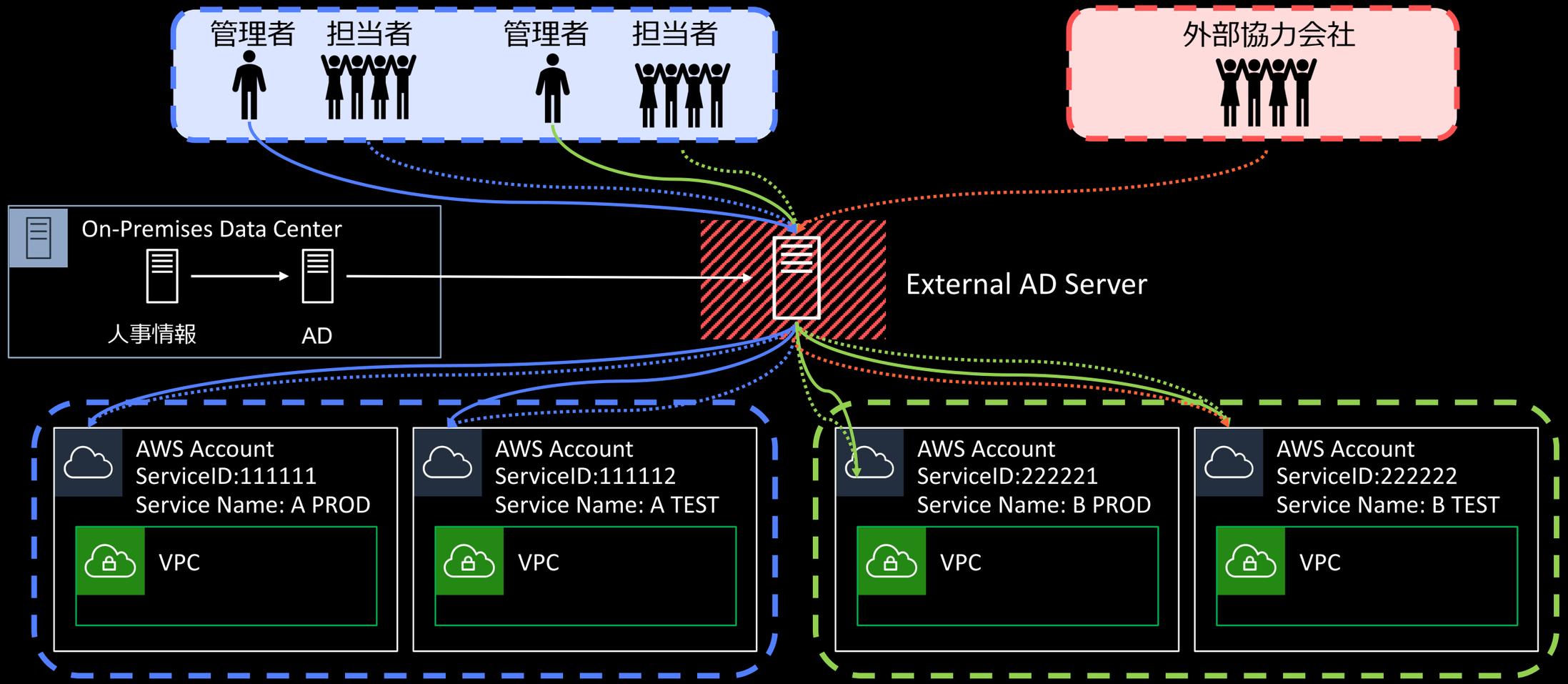
Multi-Account Strategy

サービス単位・アプリケーション開発チーム単位でのアカウントを払い出すことで、サービス単位でコストを可視化できるようになる。メンテナンス性が高くなり運用しやすい。



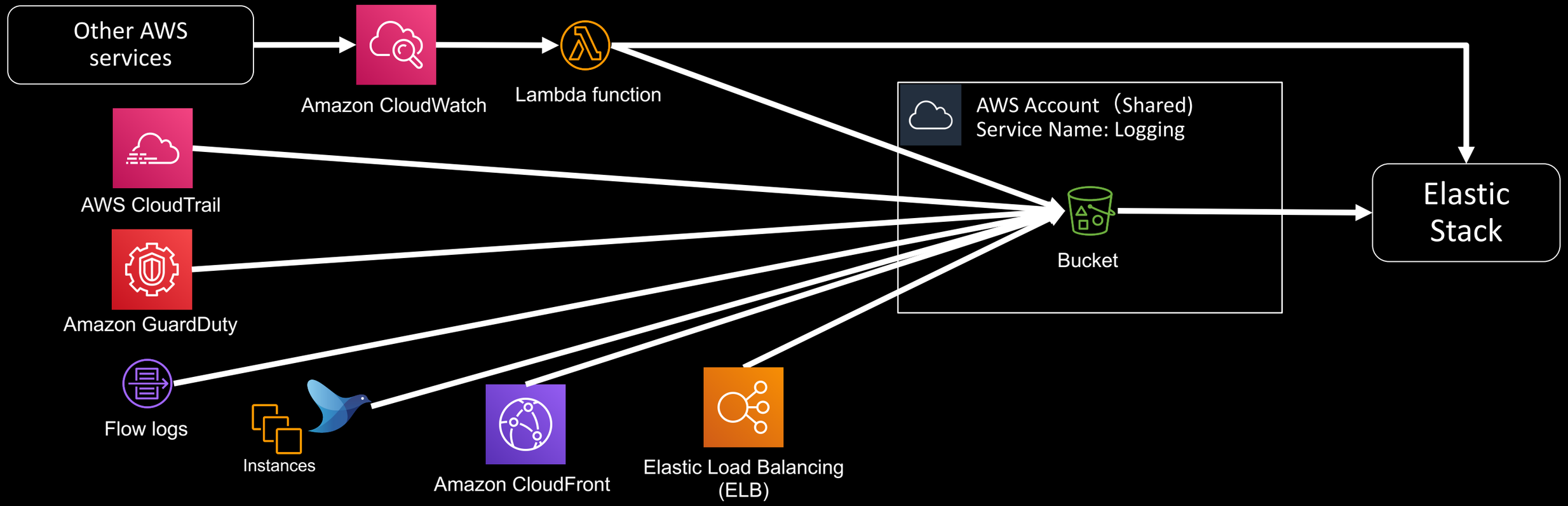
Identity Federation in the AWS

AWS マネジメントコンソールへのログインにIAMユーザ作成不要。
ログインアカウントと人事情報を紐付ける事で管理コストを削減。



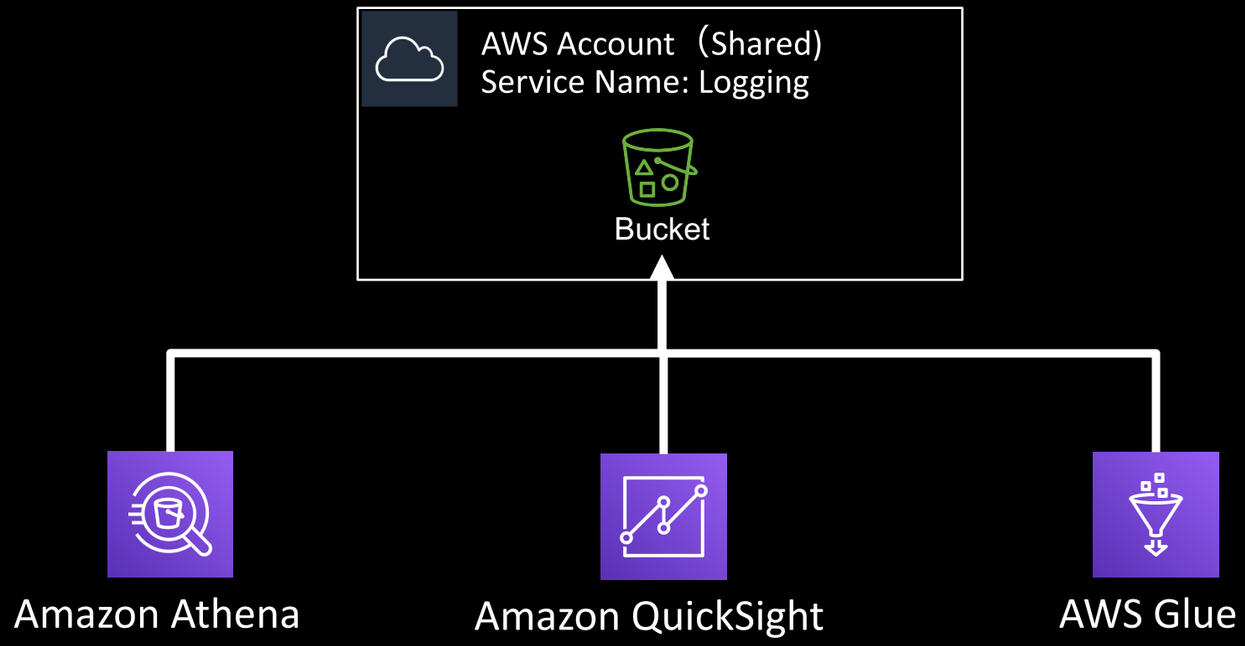
Logging account

各アカウントから出力されるログを一つのアカウントに集約する。
AWS内で取れるありとあらゆるログ（セキュリティ、アプリケーション、アクティビティ等）をリアルタイムに可視化できる。



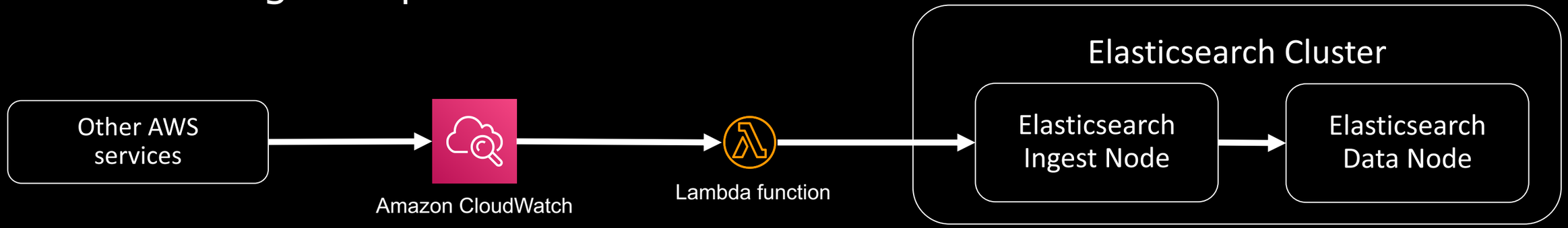
Logging account

Amazon S3に集められたログはAWS Glueでデータ抽出をしたり、Amazon AthenaでSQLを使った分析、Amazon QuickSightで可視化ができる。リアリタイムログは、Elasticsearchに取り込むことでログを可視化する。導入当初最も要望が多かったのはこのパターン。



Logging account

Elasticsearch Ingest Pipeline

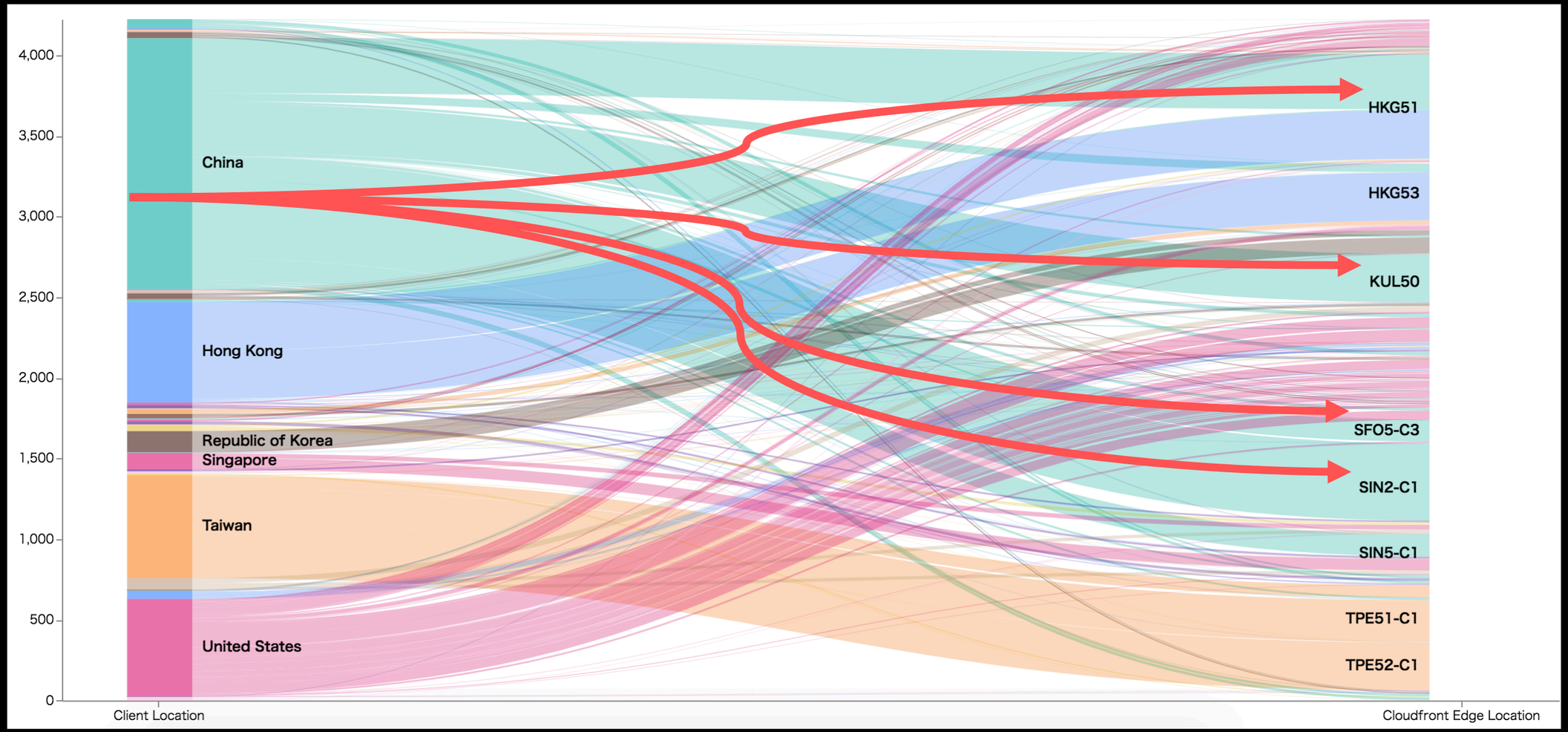


PUT _ingest/pipeline/ecs-apache

```
{
   "description" : "Apache Combinedlogs ",
   "processors" : [
    {
      "grok" : {
       "field" : "message",
       "patterns" : [
         """%{IPORHOST:clientip} %{IPORHOST:x-forwarded-for} %{USER:auth} ¥[%{HTTPDATE:timestamp}¥]
"%{WORD:method} %{DATA:request} HTTP/%{NUMBER:proto}" %{NUMBER:status:int} (?:-
|%{NUMBER:size:int}) %{QS:referrer} %{QS:agent} %{NUMBER:responsetime:int}"""
       ]
     # 処理を追加
}
```

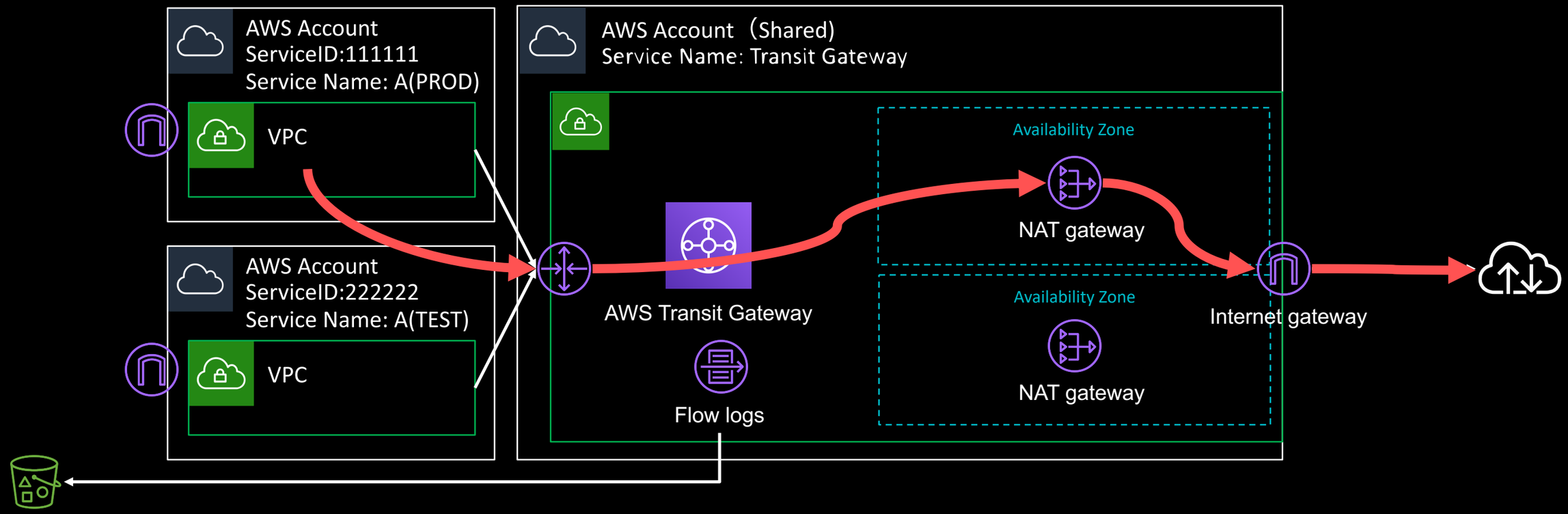
Logging account

Visualize CloudFront logs Client Location to Edge Location



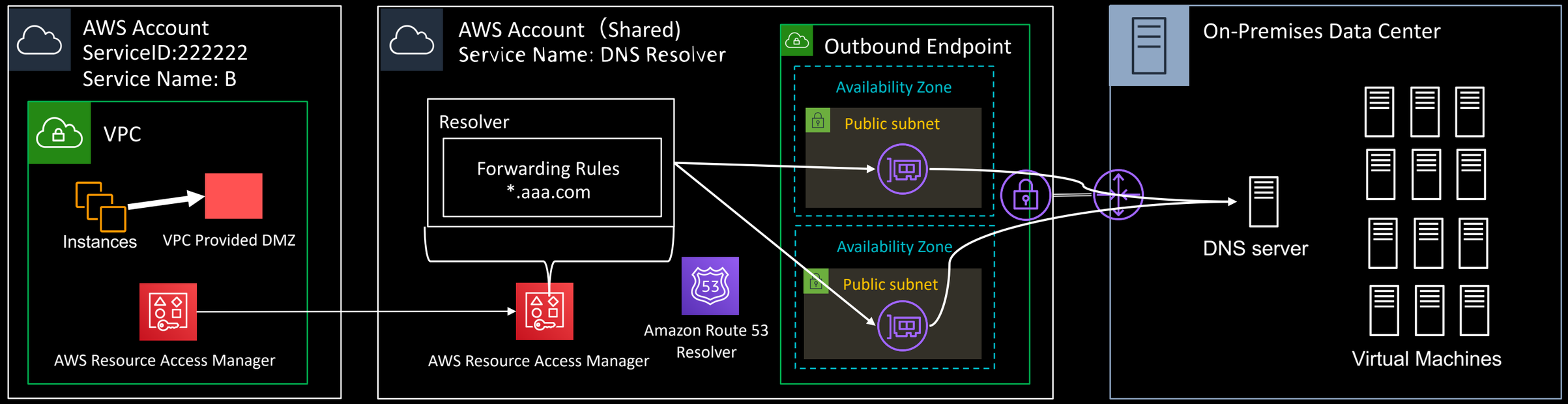
Transit Gateway

NAT Gatewayの共有化。各アカウントのNAT Gatewayを一つのアカウントに集約することでOutbound Trafficのログを集中管理することができる。DX接続にも応用が効く（最近）。



Route53 Resolver

オンプレミスとAWS内の名前解決を簡単に実現することができる。
専用のDNSサーバを立てる必要がない。



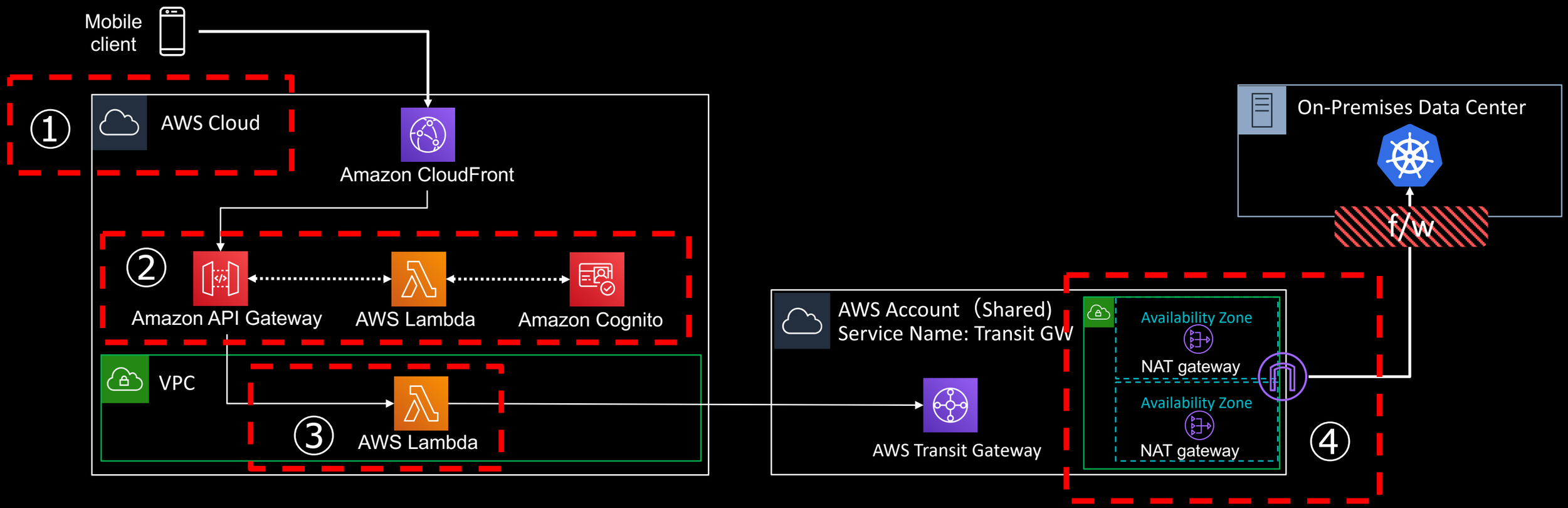
導入して半年。何ができたか？

導入例: Global API Gateway

開発期間は１ヶ月

海外アライアンス先へのAPI提供基盤

① グローバル配信に最適なリージョン選択
② API Gateway + Cognitoで認証認可を簡単に作れる
③ 簡単なAPIはLambdaで十分
④ NAT Gateway経由のみIP制限,Transit Gatewayを使うことで運用軽減



意識した事

(1)IaC

(2)Training

(3)Security

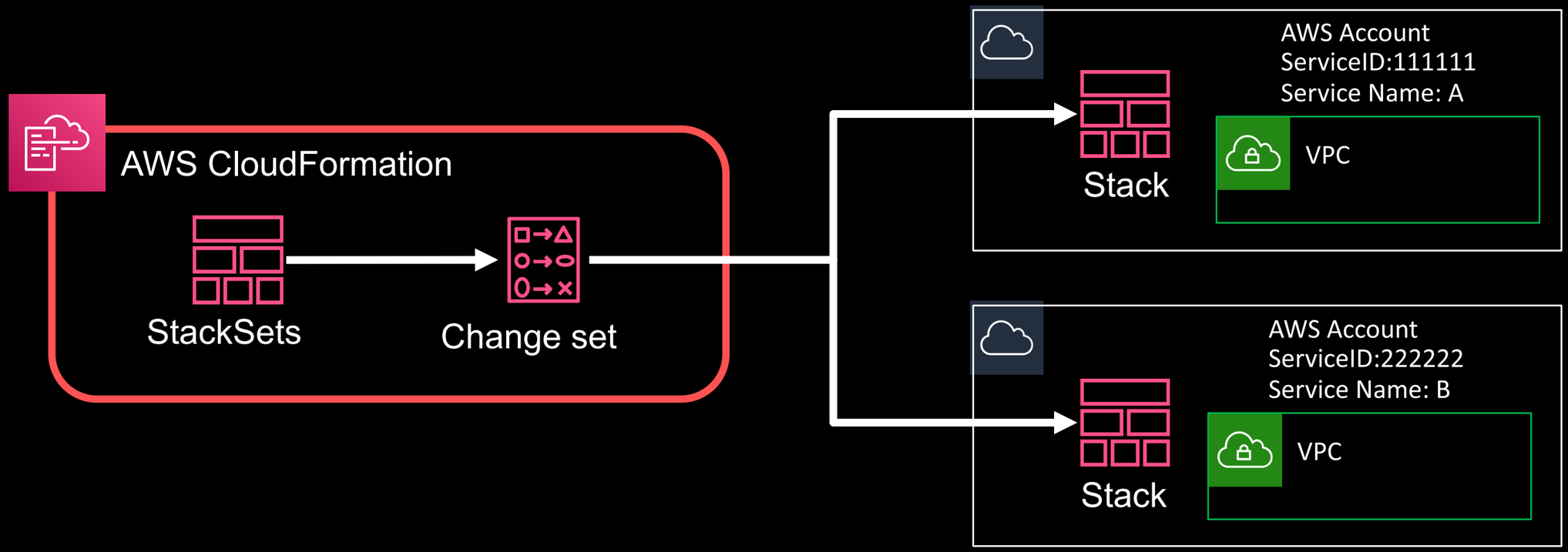
(4)Observability

(5)Lift & Shift or Re-architecting

(6)SLO

意識した事(1) **IaC**

とにかくアップデートが多い。
AWS CloudFormation/Stacksetsで管理対象アカウントをまとめて変更。アップデートを頻繁に行えるようにする。



意識した事(2) **Training**

アップデートを追いかける。

AWSは情報が溢れているのでGoogle検索とAWSのSAさんがいれば使いこなせる。

これが欲しかった！というアップデートがくると嬉しくなる

意識した事(3) **Security**

ガバナンスに沿ったセキュリティ確保。AWS Organization/IAM管理、ログの可視化、VPC設計、AWS WAFの導入など全てのアカウントが利用開始時に設定

Security, Identity and Compliance


AWS Identity and Access Management (IAM)


Amazon GuardDuty


AWS Organizations


AWS WAF


Amazon Inspector


AWS Key Management Service

Management & Governance


AWS Config


AWS CloudTrail


AWS Systems Manager

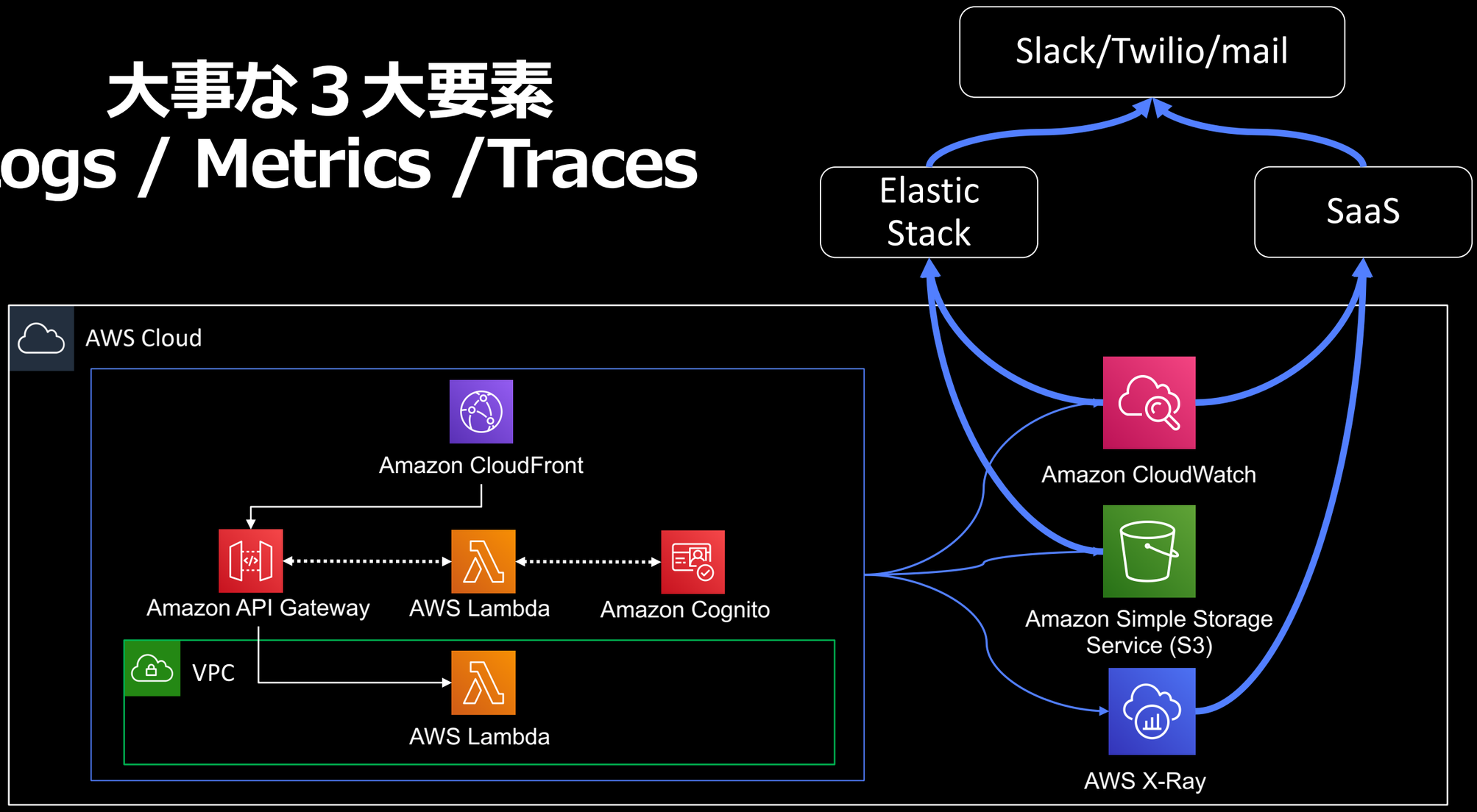

AWS Trusted Advisor


AWS Control Tower


Amazon CloudWatch

意識した事(4) **Observability**

大事な３大要素
Logs / Metrics /Traces



意識した事(5)
Lift & Shift Or Re-architecting

But Don’t Just Lift.

意識した事(6) SLO

The Network is Not Reliable.

Amazon EC2は使わない！？

Amazon EC2は使わない！？

使わない ≒ 使っていることを意識しない

EC2の運用開発者の負担になる

Deployment method

Connect to on-premises services

Direct Connect
OR
HTTPS request

AWS Direct Connect

Only use fully managed serverless services

Amazon Simple Storage Service (S3)

Amazon API Gateway

Amazon DynamoDB

Amazon Cognito

Amazon Simple Notification Service

AWS WAF

Function OR Container OR Other
+
Managed services and
Fully managed services

AWS Lambda

AWS Fargate

AWS Elastic Beanstalk

Amazon Aurora

Amazon ElastiCache

Deployment method

Connect to on-premises services

Direct Connect
OR
HTTPS request

AWS Direct Connect

Only use fully managed serverless services

Amazon Simple Storage Service (S3)

Amazon API Gateway

Amazon DynamoDB

Amazon Cognito

Amazon Simple Notification Service

AWS WAF

Function OR Container OR Other
+
Managed services and
Fully managed services

AWS Lambda

AWS Fargate

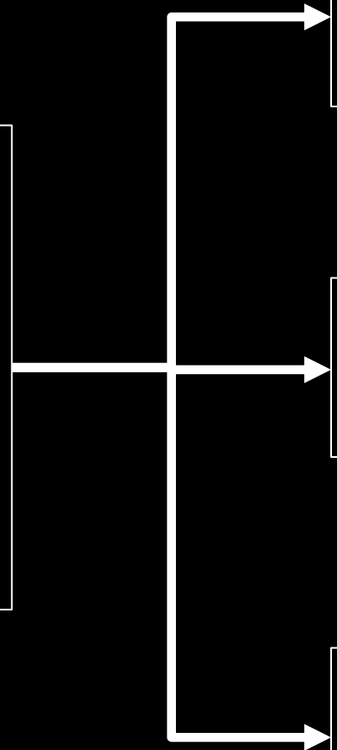
AWS Elastic Beanstalk

Amazon Aurora

Amazon ElastiCache

Deployment method

Function OR Container OR Other
+
Managed services and
Fully managed services



Functionに適している
言語?サイズ?実行時間?

AWS Lambda

Dockerで開発している？

AWS ECS / AWS Fargate

まだDocker化されていない

AWS Elastic Beanstalk

Deployment method

Function OR Container OR Other
+
Managed services and
Fully managed services

Functionに適している
言語?サイズ?実行時間?

AWS Lambda

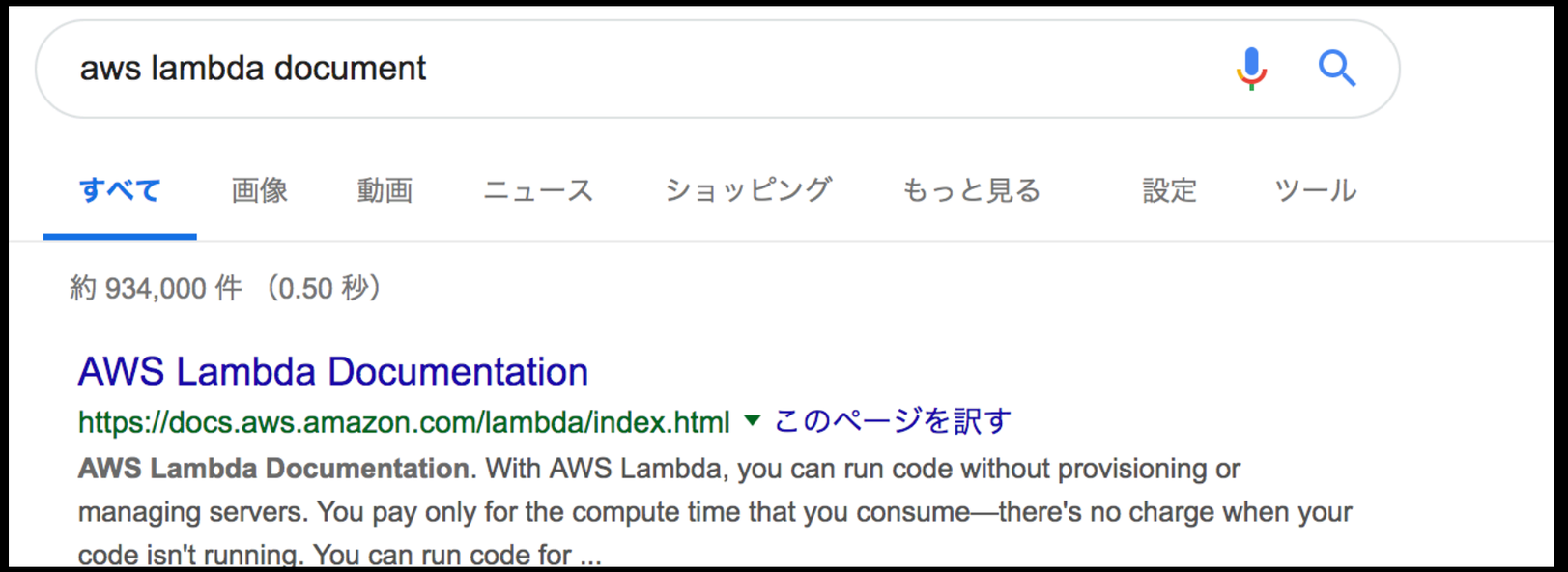
Dockerで開発している？

AWS ECS / AWS Fargate

まだDocker化されていない

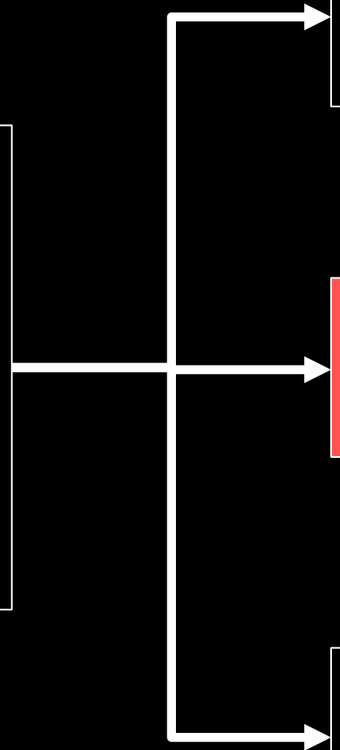
AWS Elastic Beanstalk

AWS Lambda…


aws lambda document
すべて
画像
動画
ニュース
ショッピング
もっと見る
設定
ツール
約 934,000 件 （0.50 秒）
AWS Lambda Documentation
https://docs.aws.amazon.com/lambda/index.html ▼ このページを訳す
AWS Lambda Documentation. With AWS Lambda, you can run code without provisioning or managing servers. You pay only for the compute time that you consume—there's no charge when your code isn't running. You can run code for ...

Deployment method

Function OR Container OR Other
+
Managed services and
Fully managed services



Functionに適している
言語?サイズ?実行時間?

AWS Lambda

Dockerで開発している？

AWS ECS / AWS Fargate

まだDocker化されていない

AWS Elastic Beanstalk

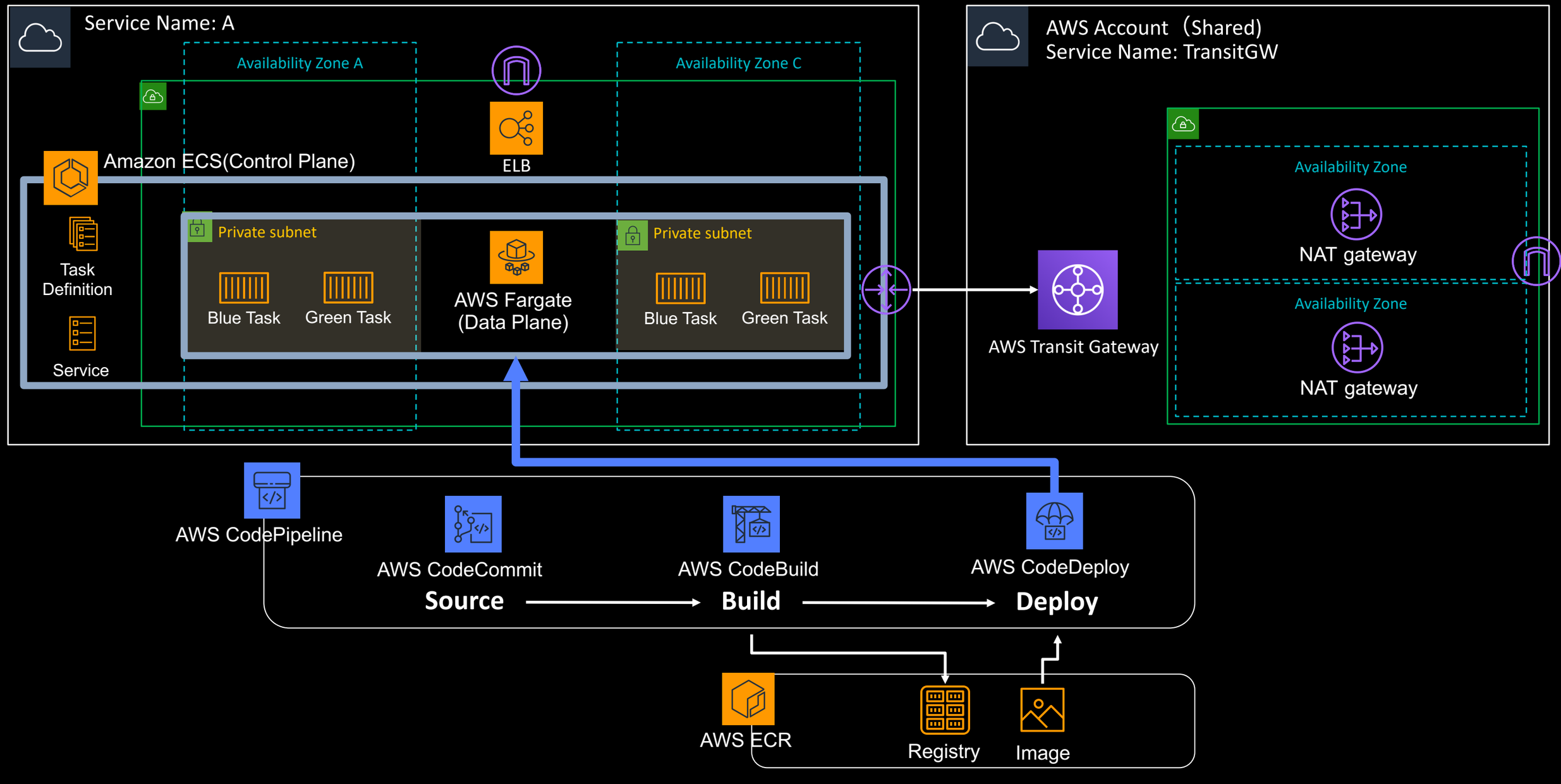
AWS ECS / AWS Fargate

Amazon EC2 Container Registry
Image Registry

Amazon Elastic Container Service
Management

AWS Fargate
Hosting

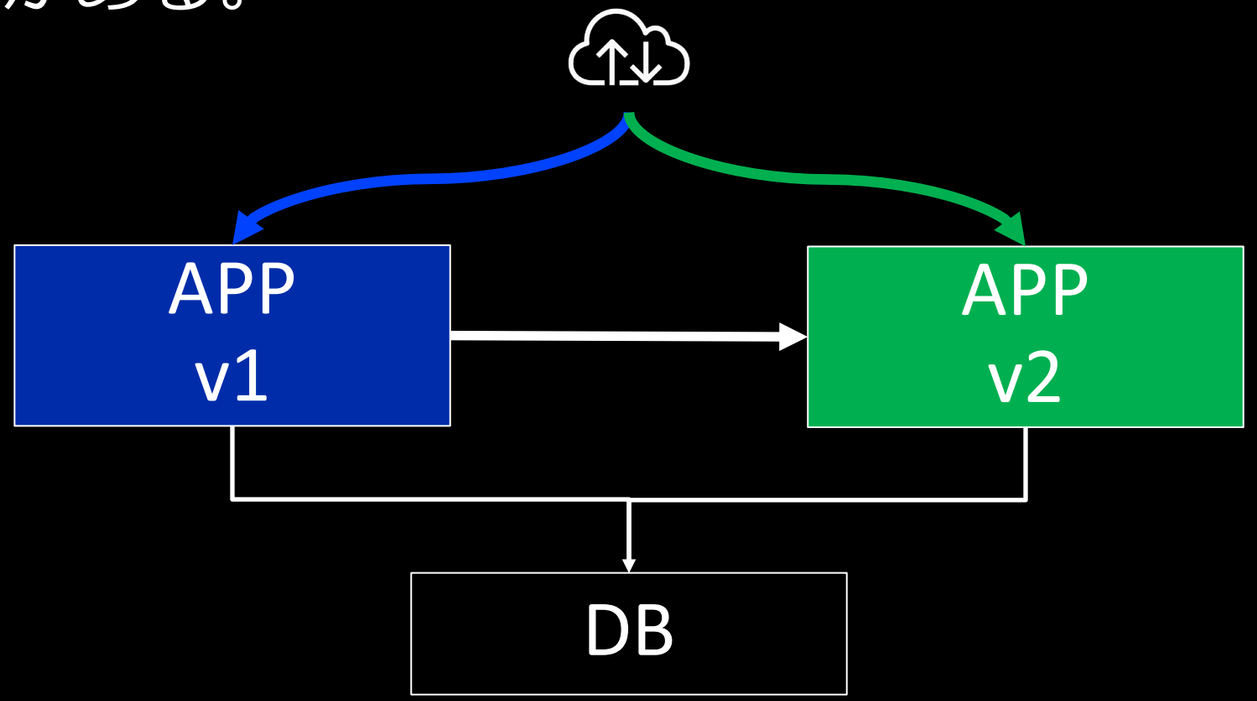
AWS Fargate + Blue/Green Deployment



AWS Fargate + Blue/Green Deployment

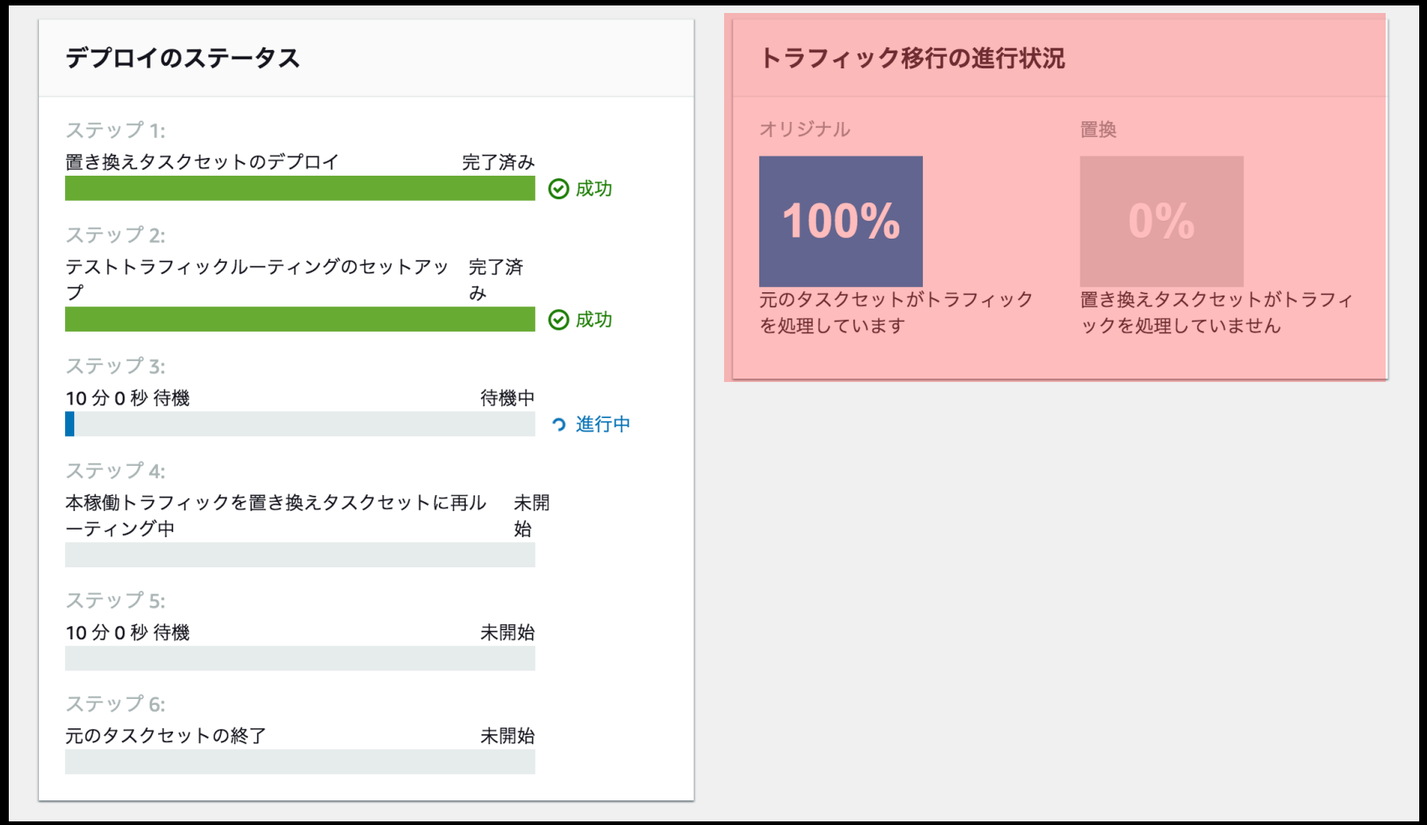
そもそもBlue/Greenデプロイメントとは

アプリケーションの実行環境を２面（Blue/Green)用意しLBやDNSを切り替えることでリリースするデプロイ手法。２面同じスケールで用意する為、オンプレミスだとコンテナでしかやっていなかった。VMだとリソースを大量に確保する必要がある。



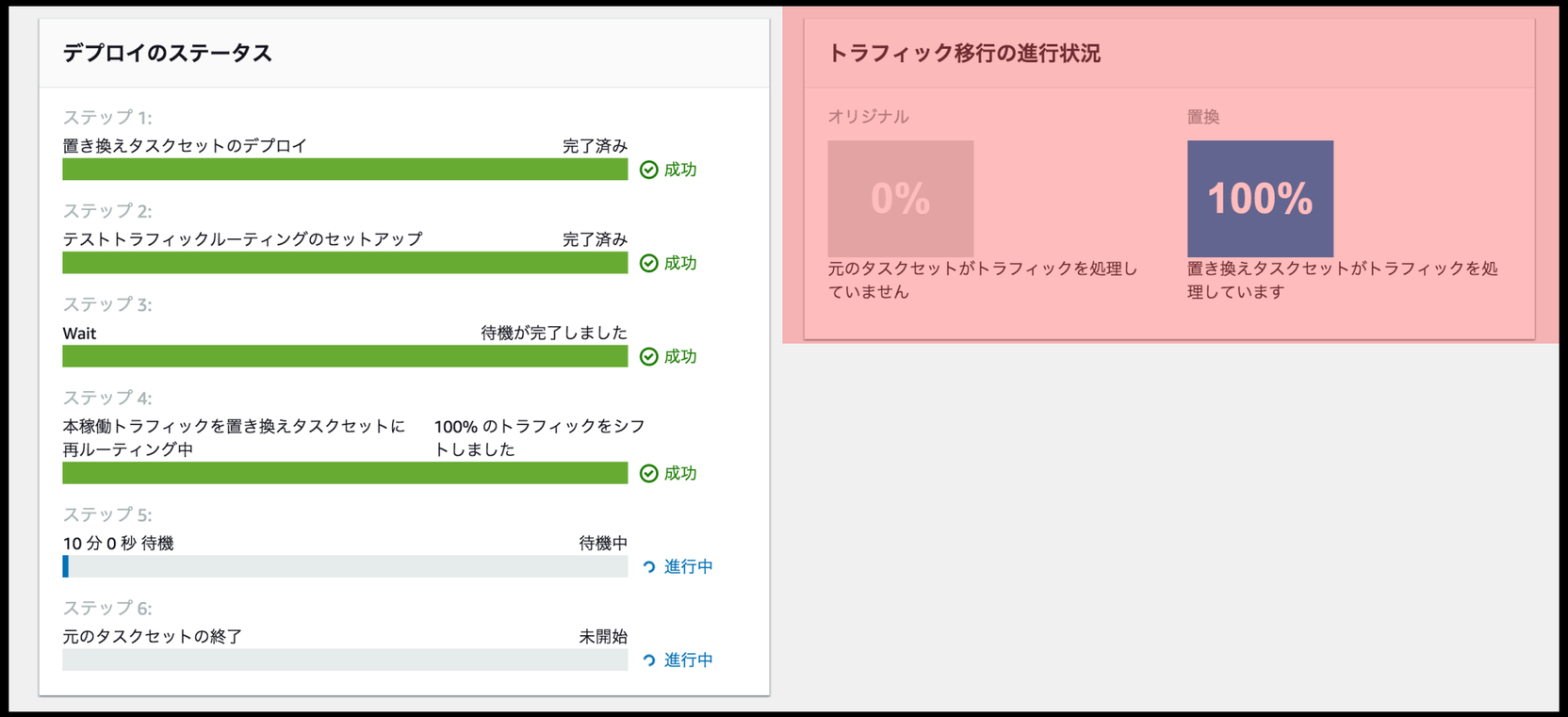
AWS Fargate + Blue/Green Deployment

現環境と同じスケールのリリース環境を作成する。待ち時間を設定し、リリース環境のアプリケーションの動作確認を行う。



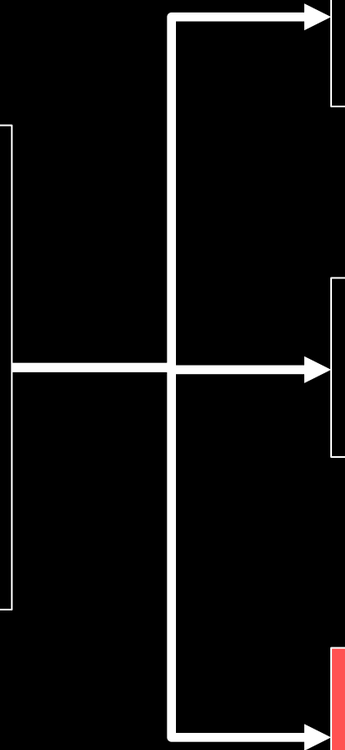
AWS Fargate + Blue/Green Deployment

問題無かったら切り替え。切り替え後も待ち時間を設定できる為、切り戻しも容易に行える。切り戻し判断がなくなればオリジナル環境を削除する。



Deployment method

Function OR Container OR Other
+
Managed services and
Fully managed services



Functionに適している
言語?サイズ?実行時間?

AWS Lambda

Dockerで開発している？

AWS ECS / AWS Fargate

まだDocker化されていない

AWS Elastic Beanstalk

AWS Elastic Beanstalk / GitLab CI

AWS Elastic Beanstalk

Deployment

Application

開発者はアプリケーションのデプロイを実施

Amazon Linux(AMI)は自動でアップデート

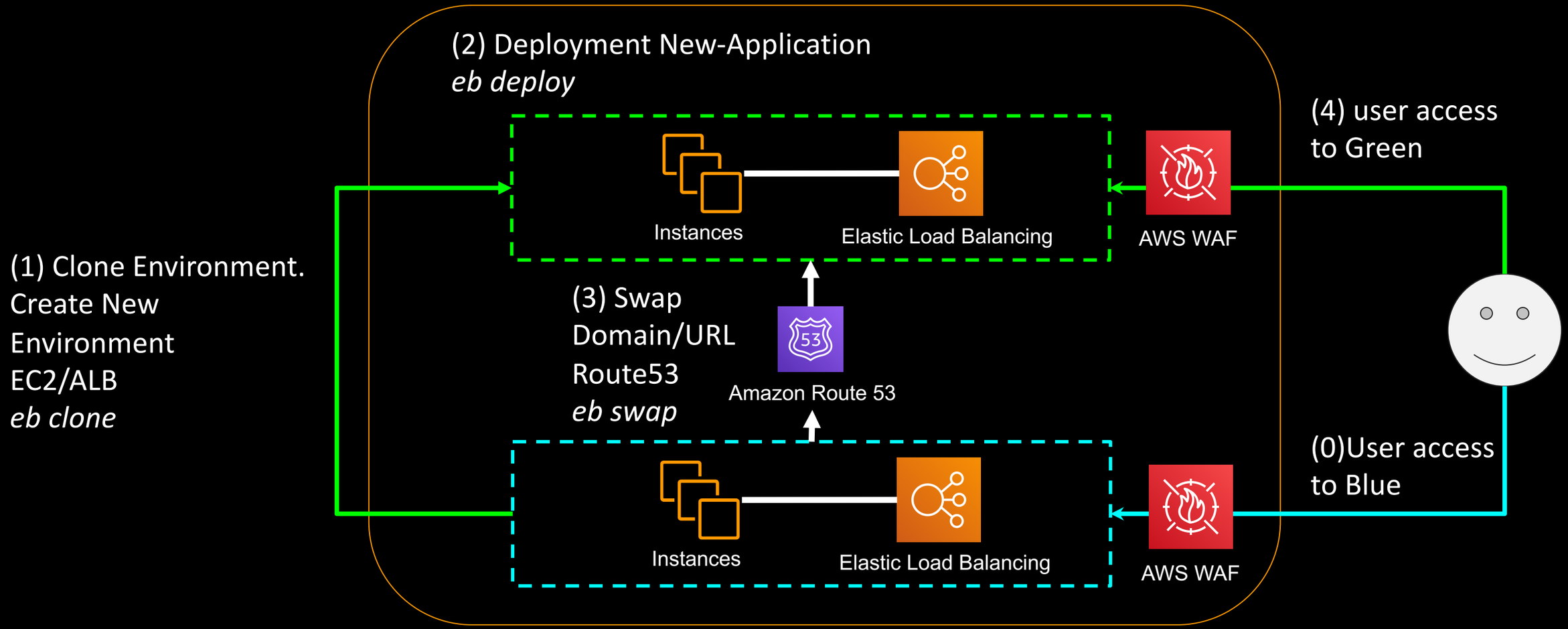
Blue/Green Deploymentでリリース

Immutable. ELB/EC2/Auto Scaling Groupがデプロイ単位で都度作成する

Docker化されていない開発チームも利用できる

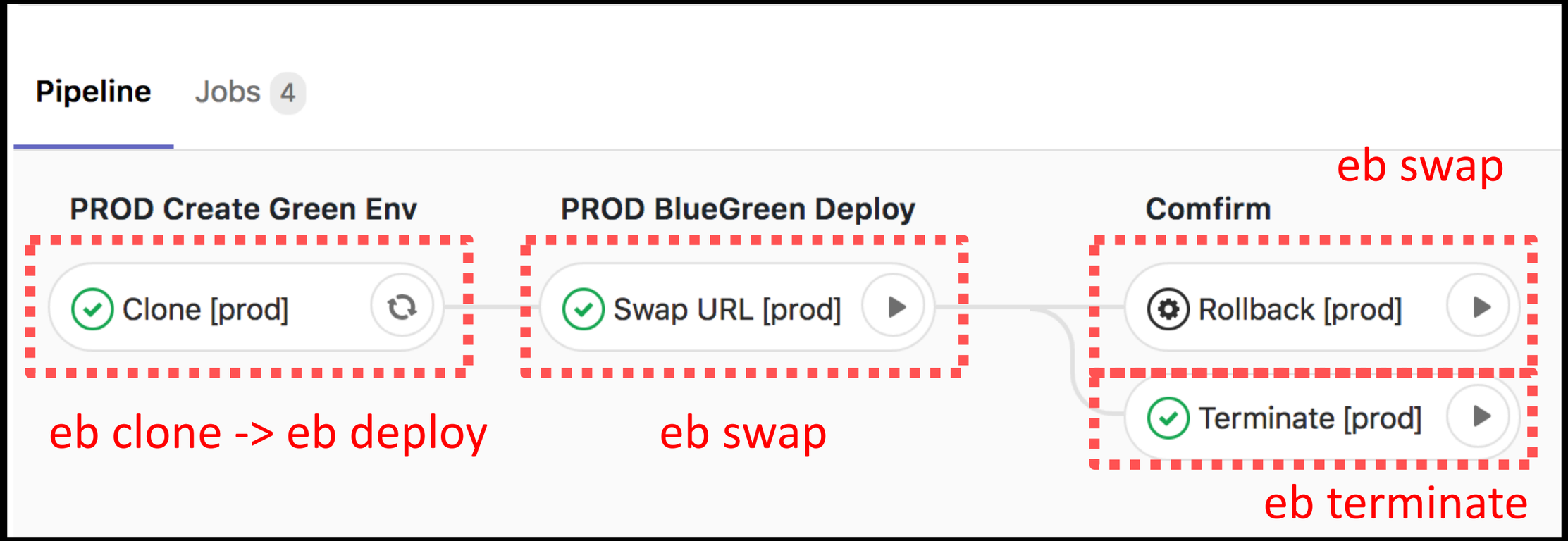
AWS Elastic Beanstalk / GitLab CI

ElasticBeanstalk Blue/Green Deployment



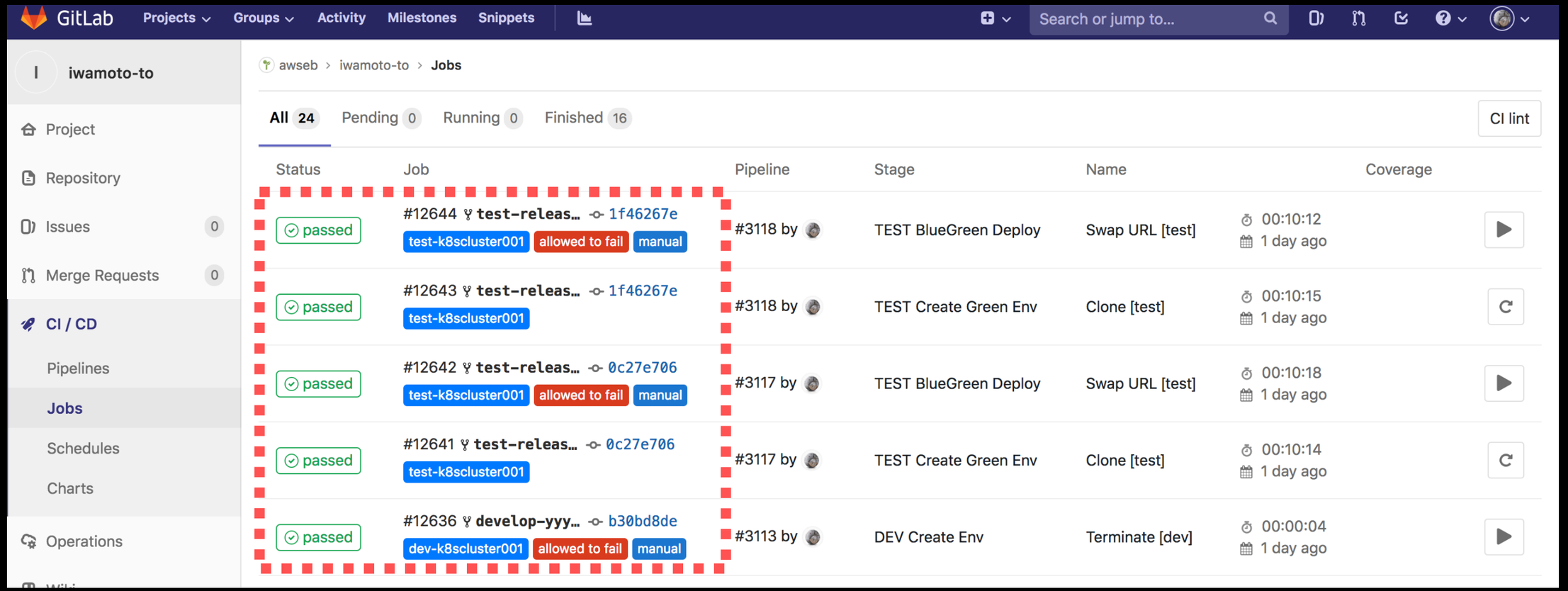
AWS Elastic Beanstalk / GitLab CI

GitLab CI / GitLab Runner (Pipeline Clone -> deploy -> Swap -> Terminate or Rollback)



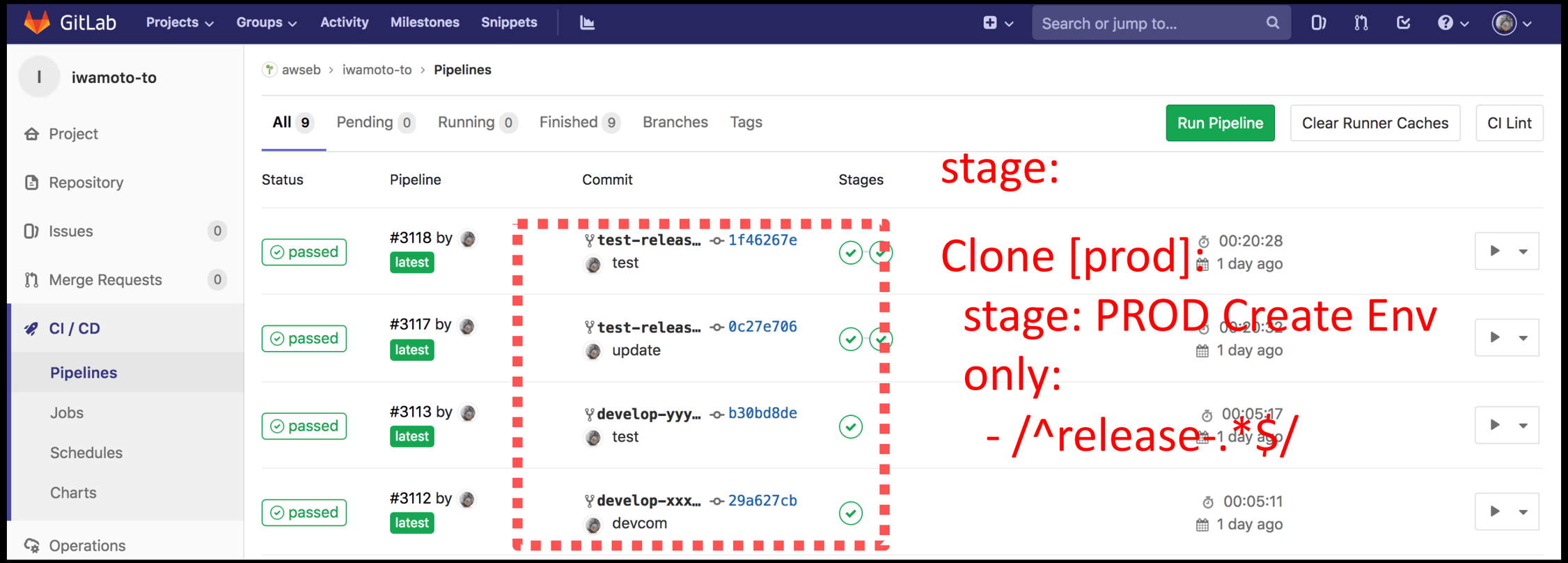
AWS Elastic Beanstalk / GitLab CI

GitLab CI / GitLab Runner (Job Status)



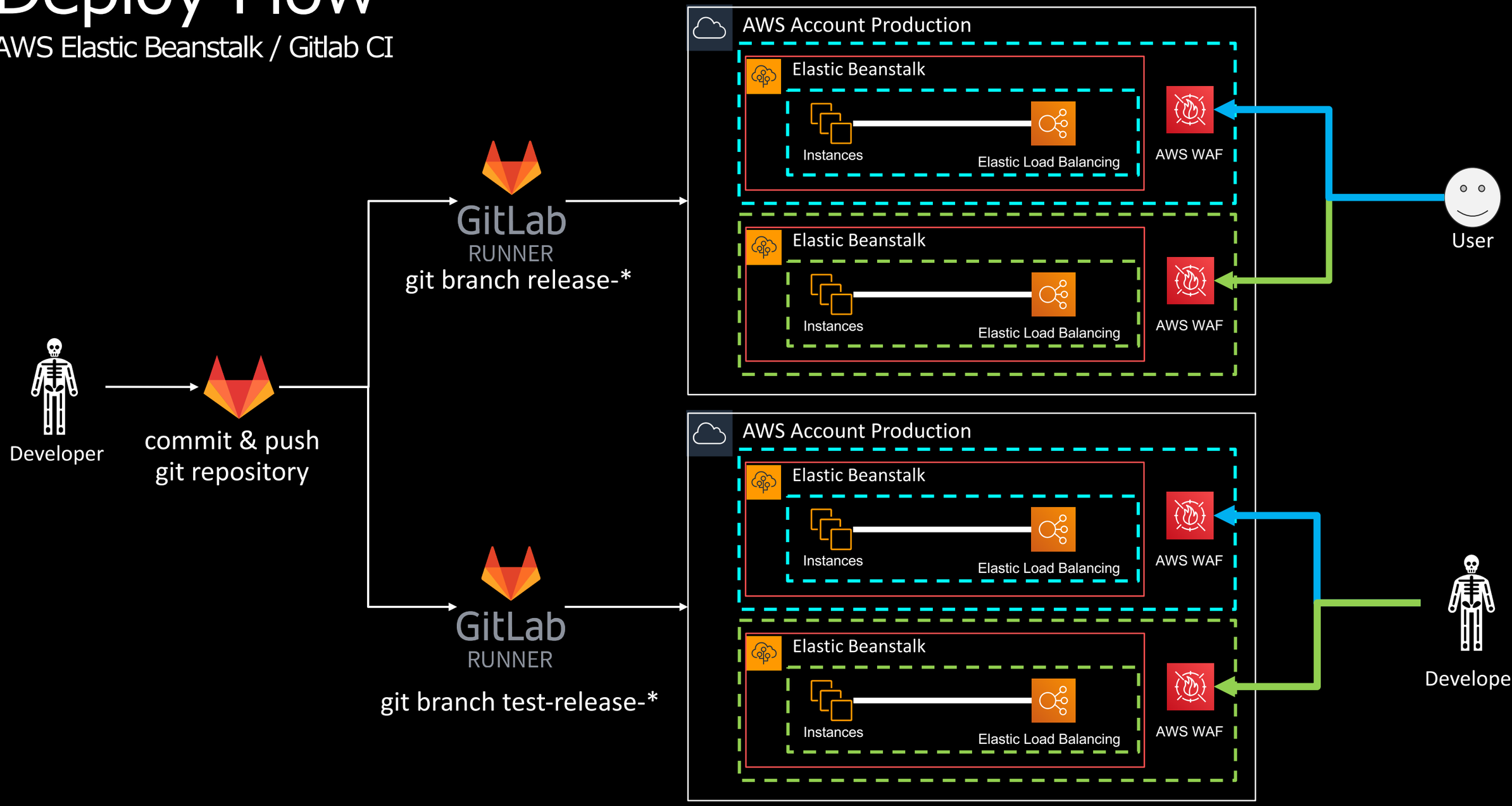
AWS Elastic Beanstalk / GitLab CI

GitLab CI / GitLab Runner (Gitlab-ci.yml setting)



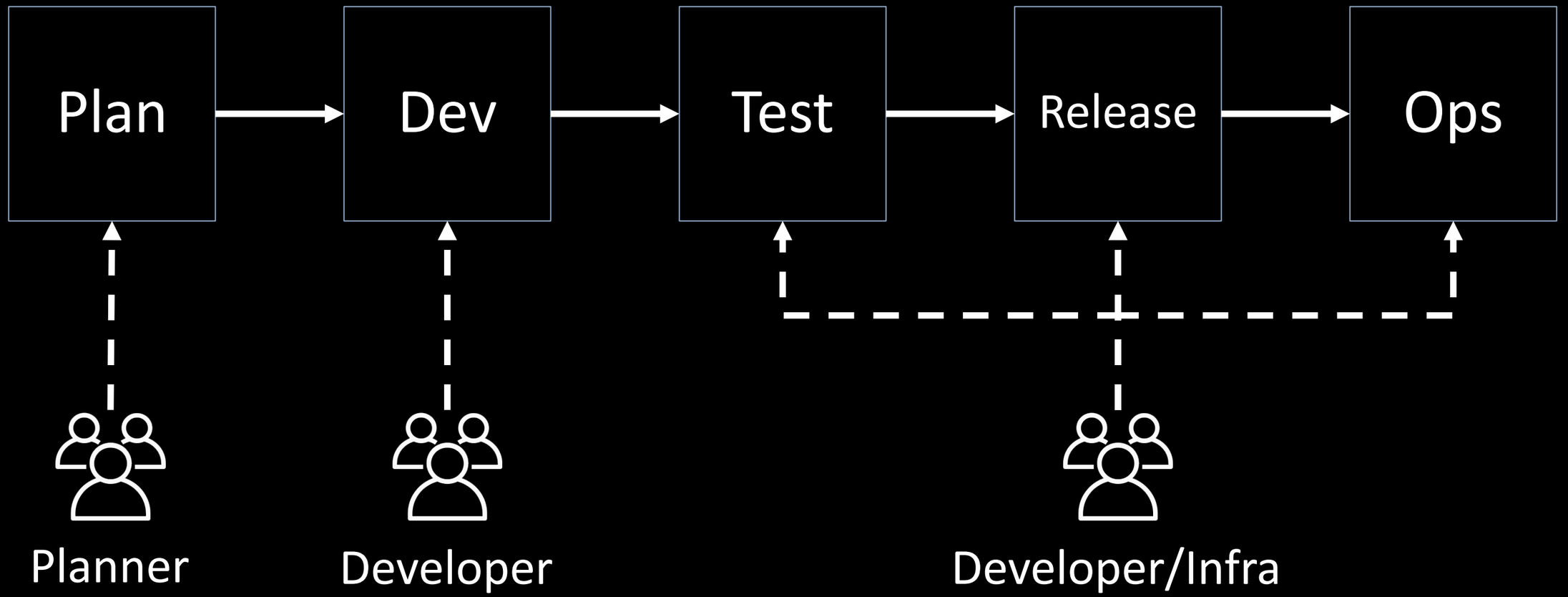
Deploy Flow

AWS Elastic Beanstalk / Gitlab CI

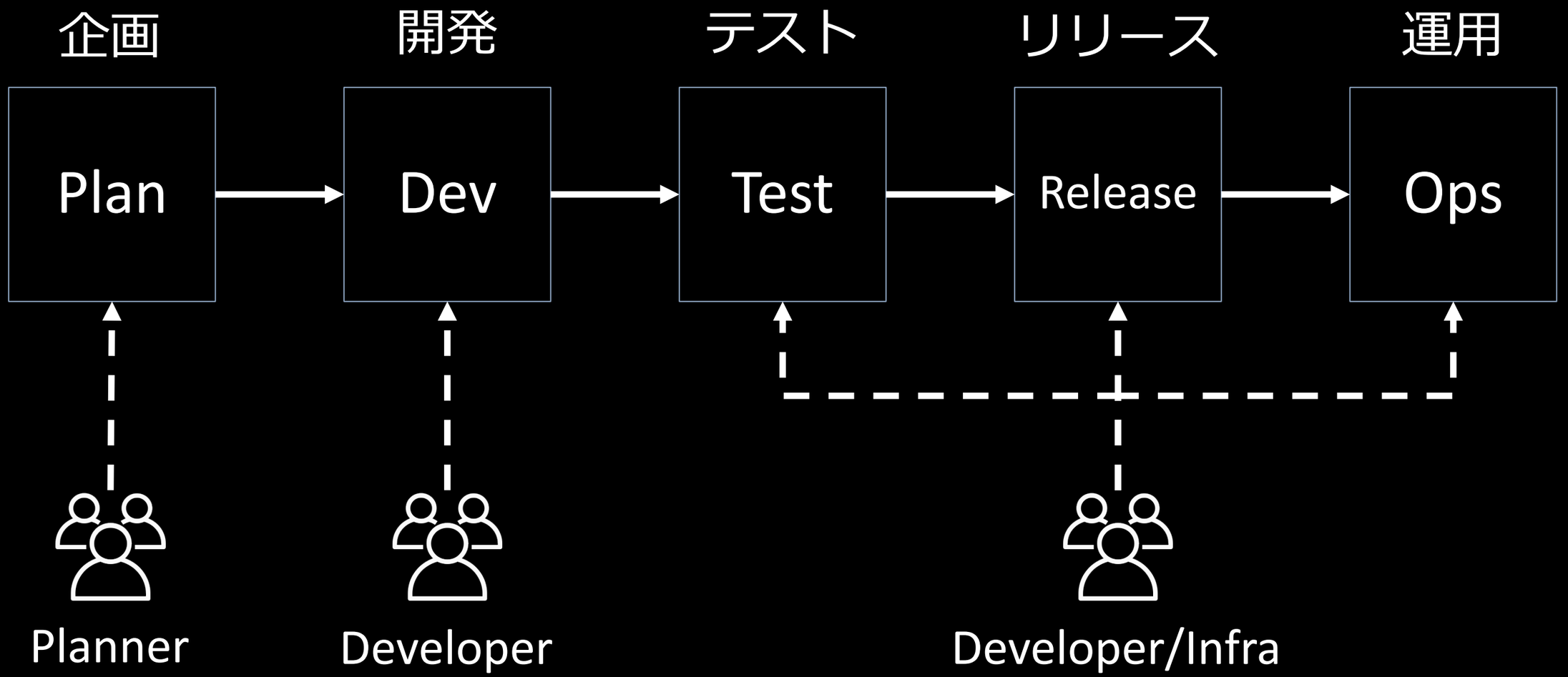


クラウドネイティブがもたらすビジネスの加速化

Webサイトができるまで

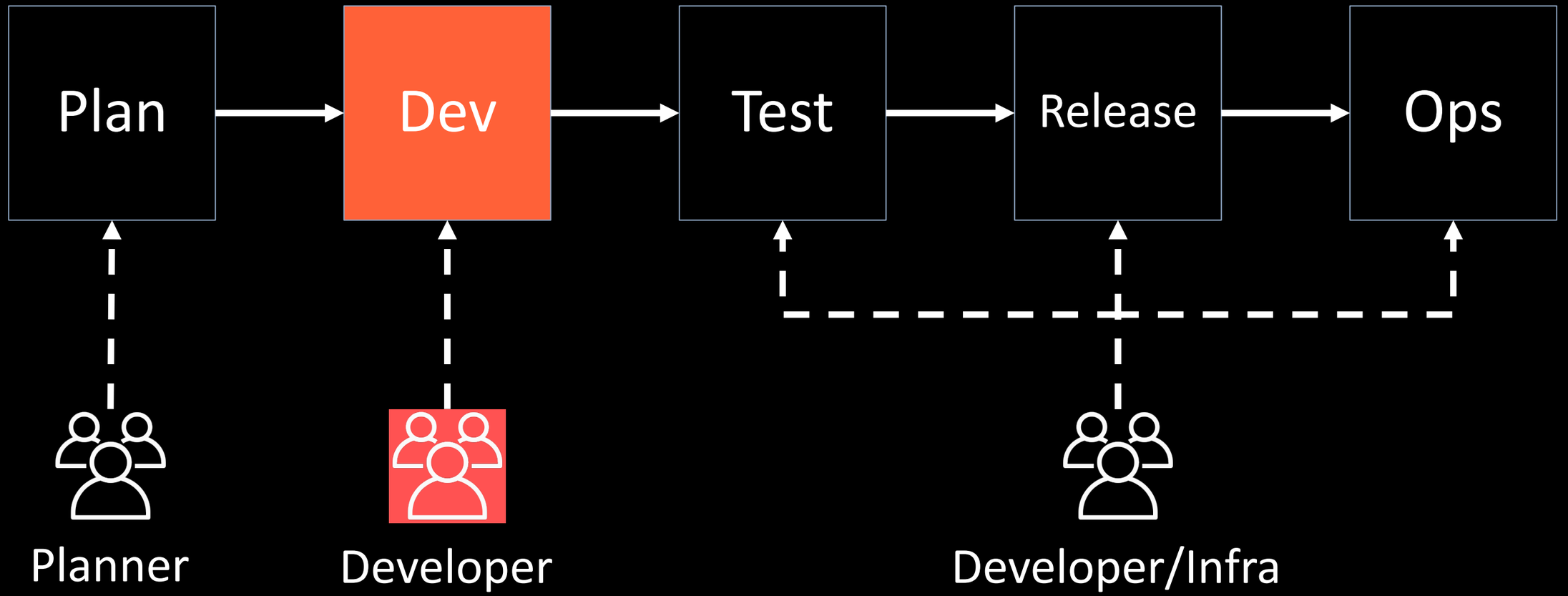


Webサイトができるまで



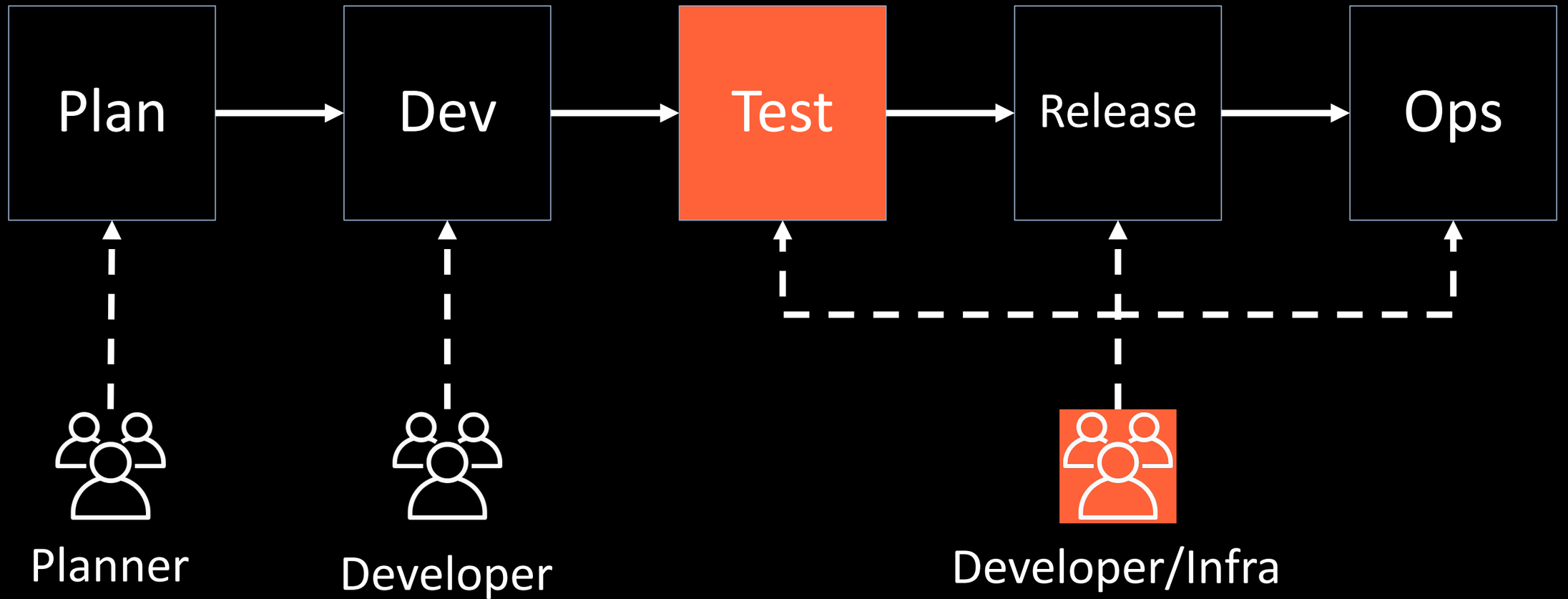
Webサイトができるまで

激しい密結合による開発コスト激増



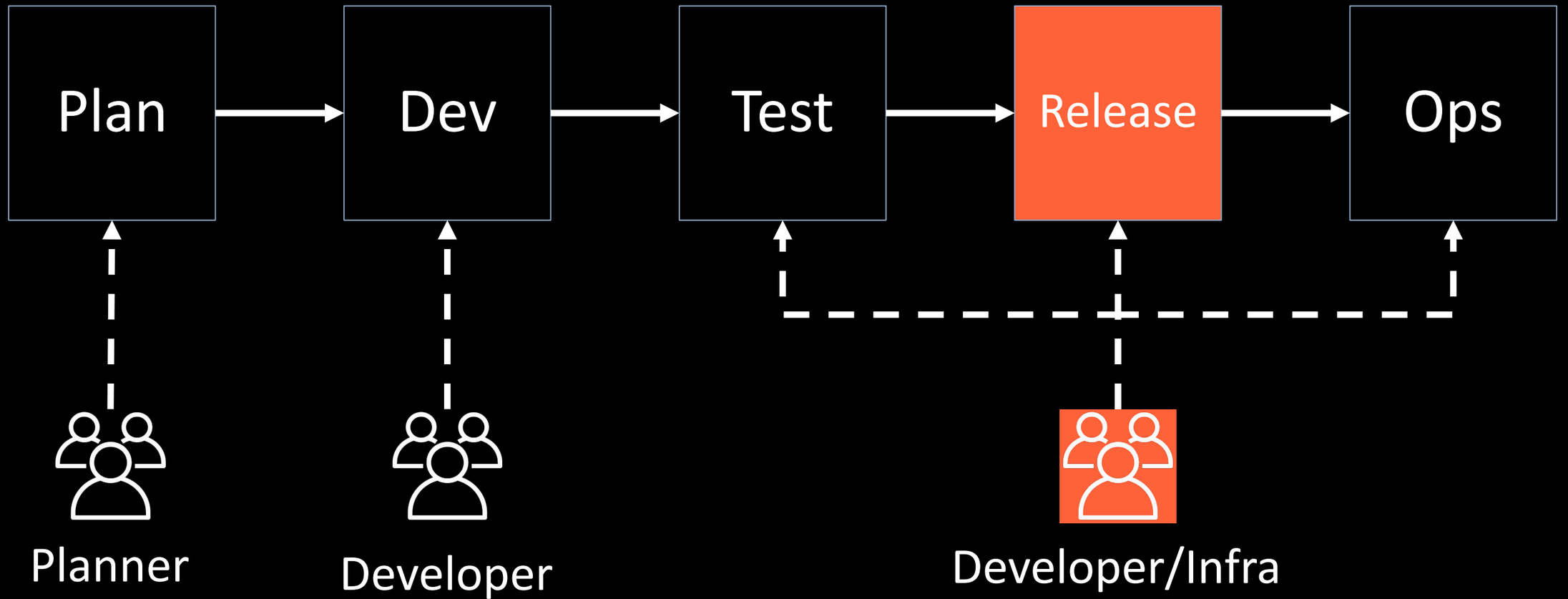
Webサイトができるまで

複雑なシステムへのテストで漏れが多発



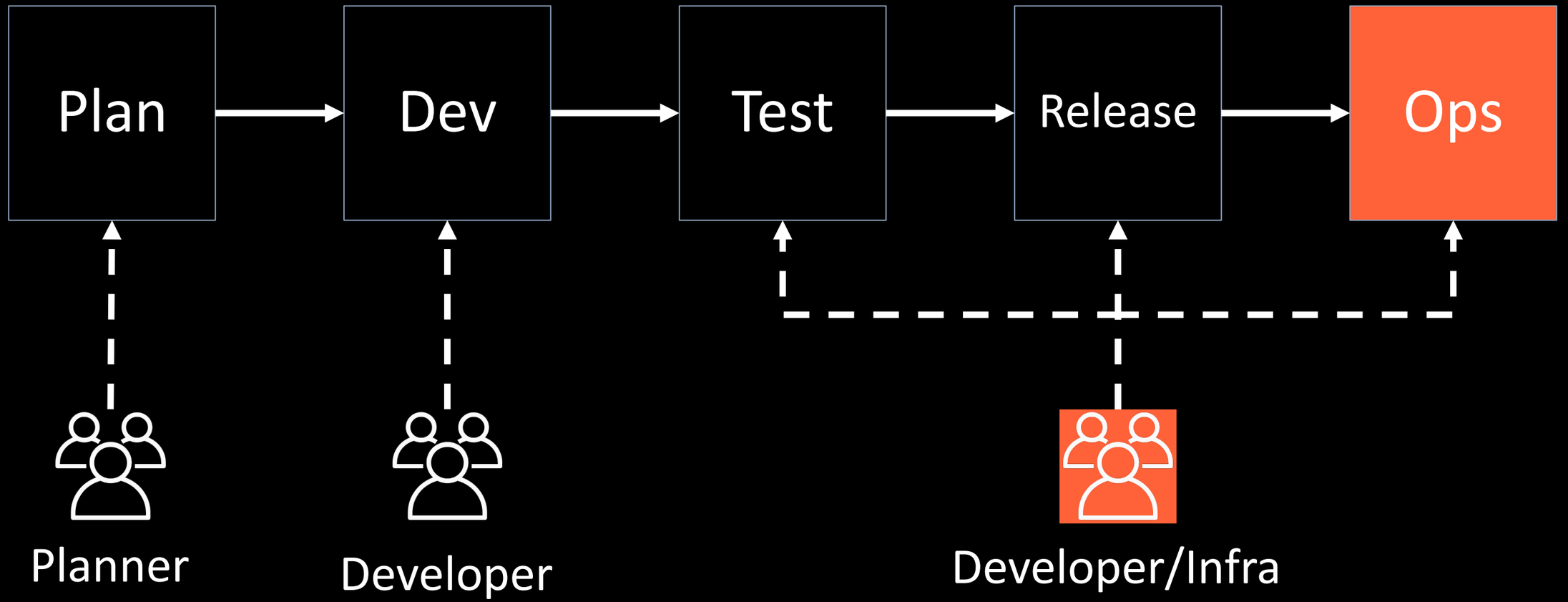
Webサイトができるまで

テスト漏れによる事故発生!! 一発勝負の**男気**リリース



Webサイトができるまで

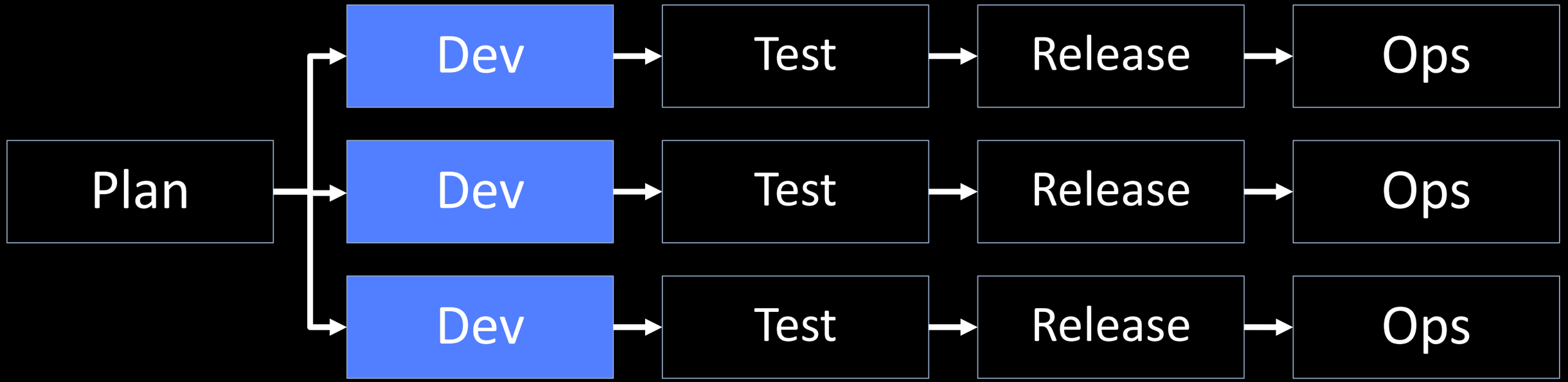
運用メンテナンス性の低下/人的リソースのの無駄使い



クラウドネイティブに開発する

コンテナ開発によりシステムの分散化

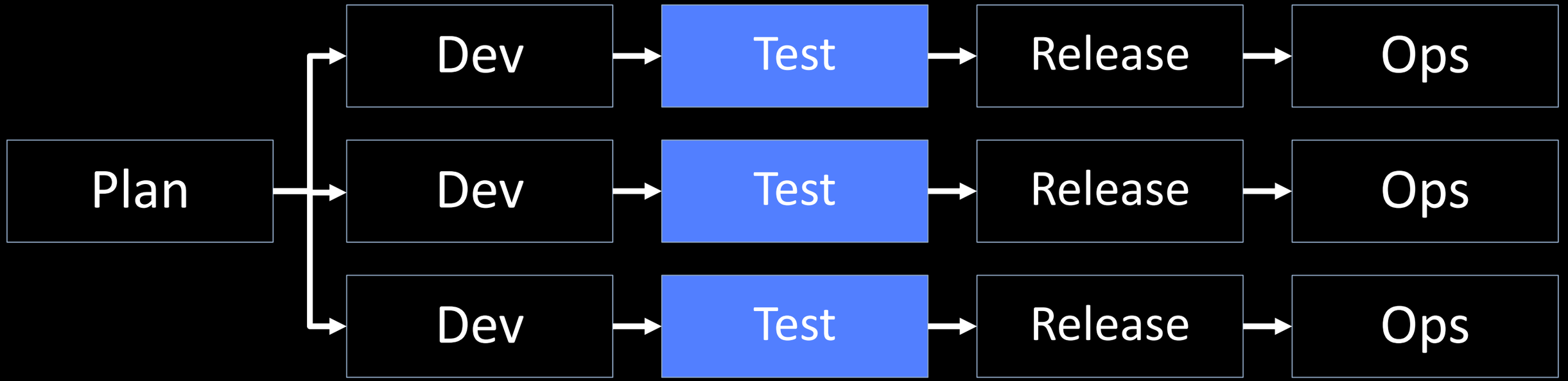
機能別・システム別にシステムを分割する



クラウドネイティブに開発する

見える範囲でテストが実施できる

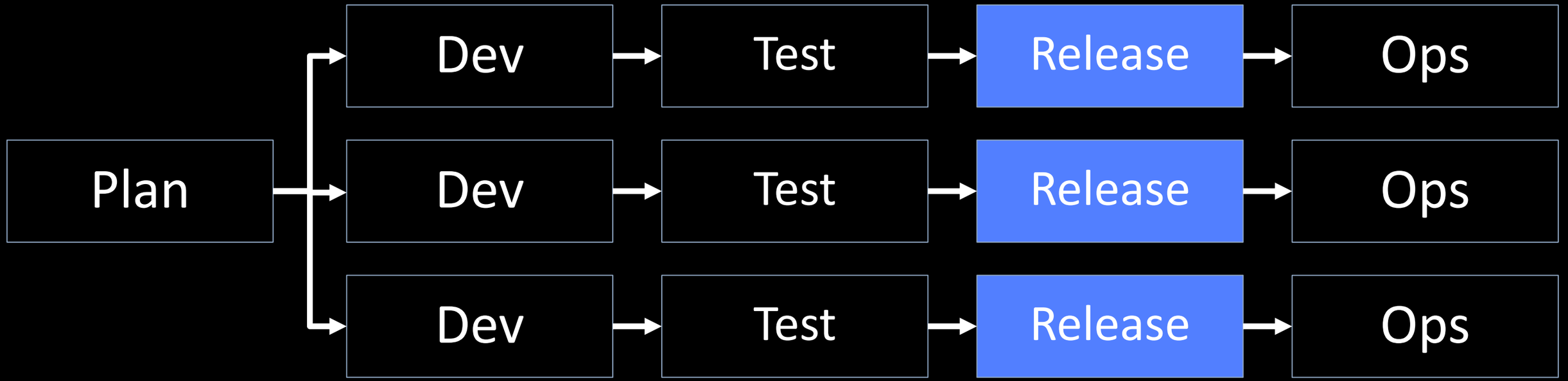
漏れが発生しにくくなる



クラウドネイティブに開発する

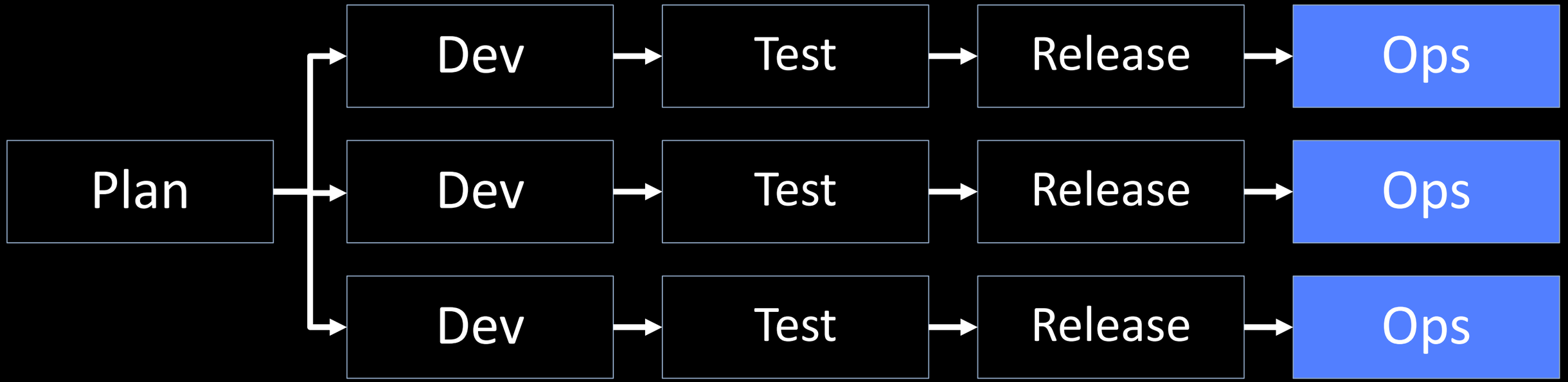
Blue/Greenデプロイメントで安全にリリース

問題が発生すれば瞬時にロールバック



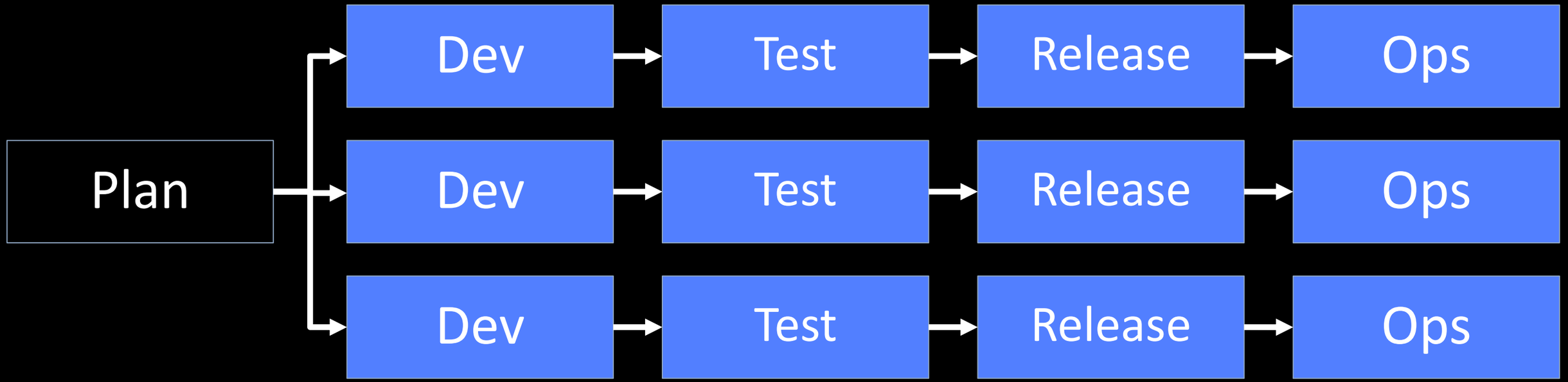
クラウドネイティブに開発する

高いメンテナンス性
アップデートが容易になる

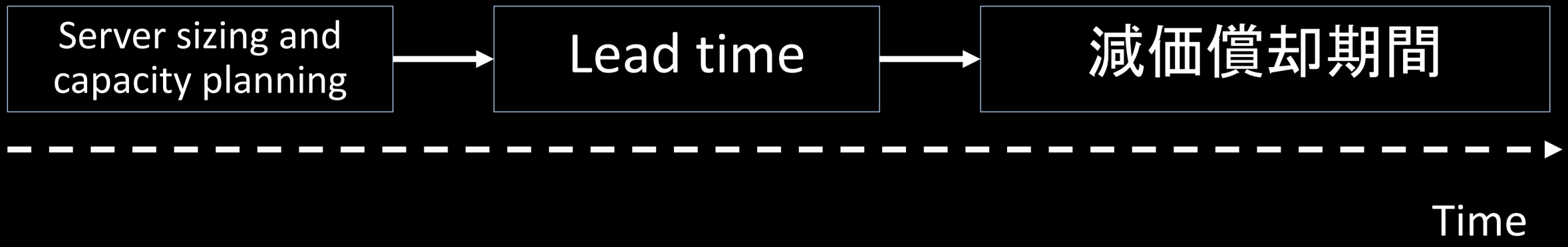


クラウドネイティブに開発する

オンプレミスでも
ここまでは実現できる

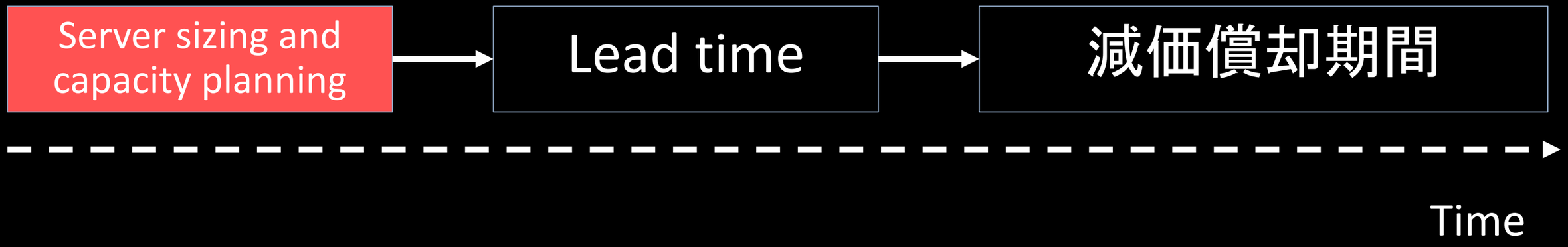


オンプレミスでの課題



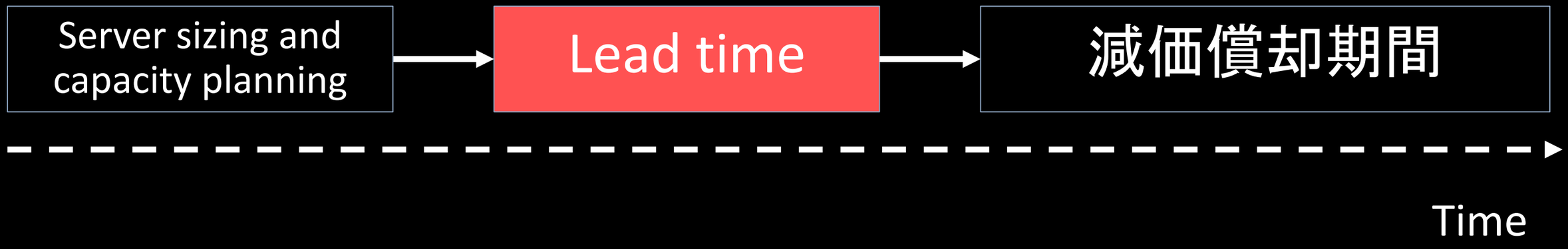
オンプレミスでの課題

機器を購入するために必要なサイジング
予測できない予測をする



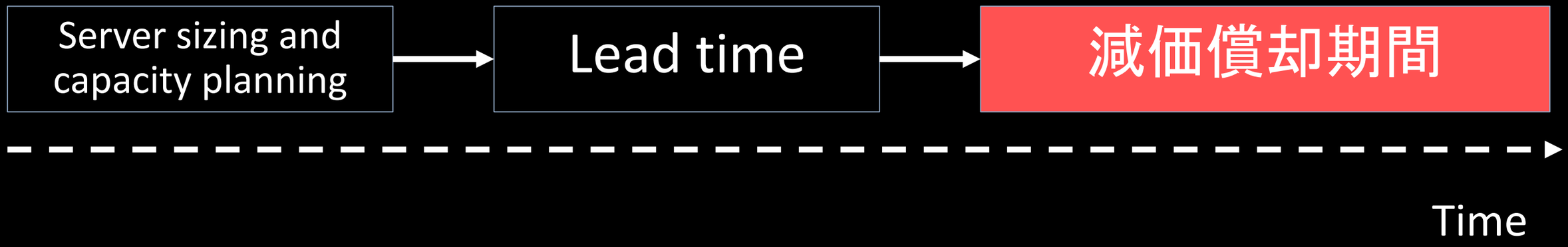
オンプレミスでの課題

調達リードタイム
結果的に余剰にリソースを抱える可能性もある



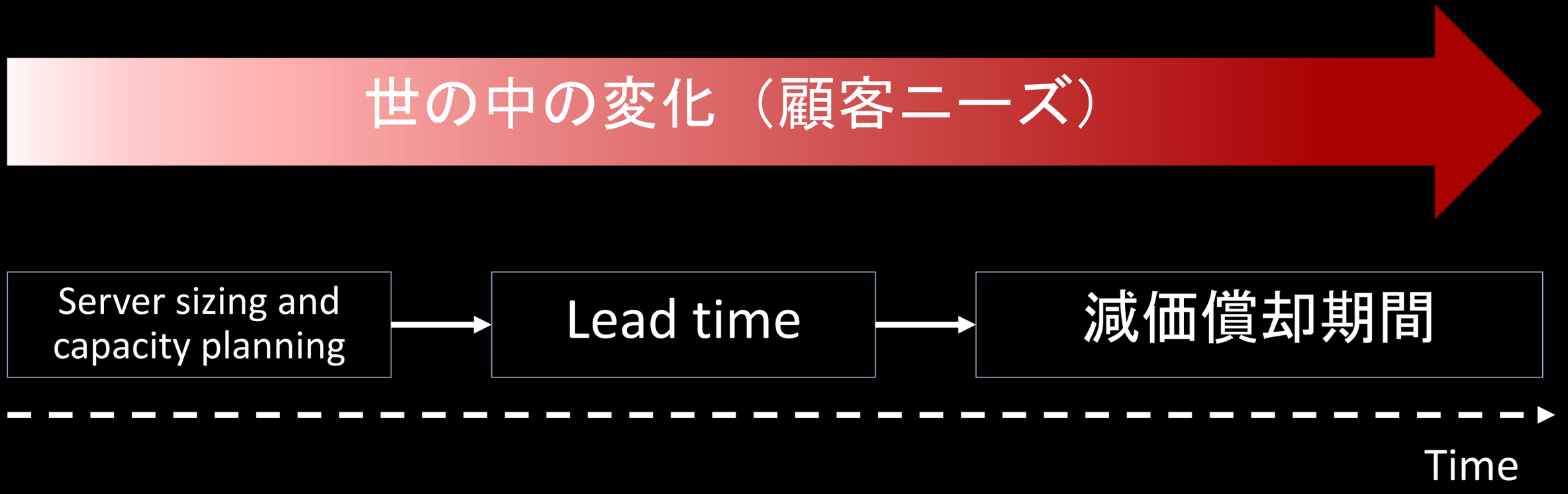
オンプレミスでの課題

システムのリプレース
償却が終わるまで利用を続ける



オンプレミスでの課題

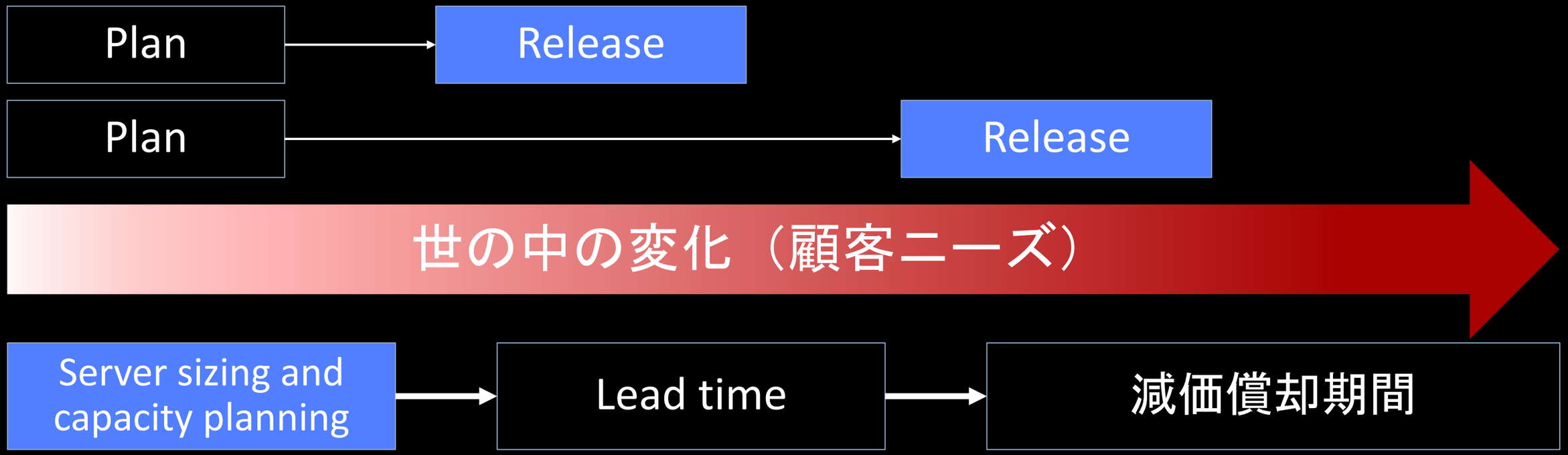
「顧客ニーズ（ビジネスサイクル）」と「システムサイクル」の間に激しいギャップが発生する



クラウドネイティブ on AWS がビジネスを加速化させる

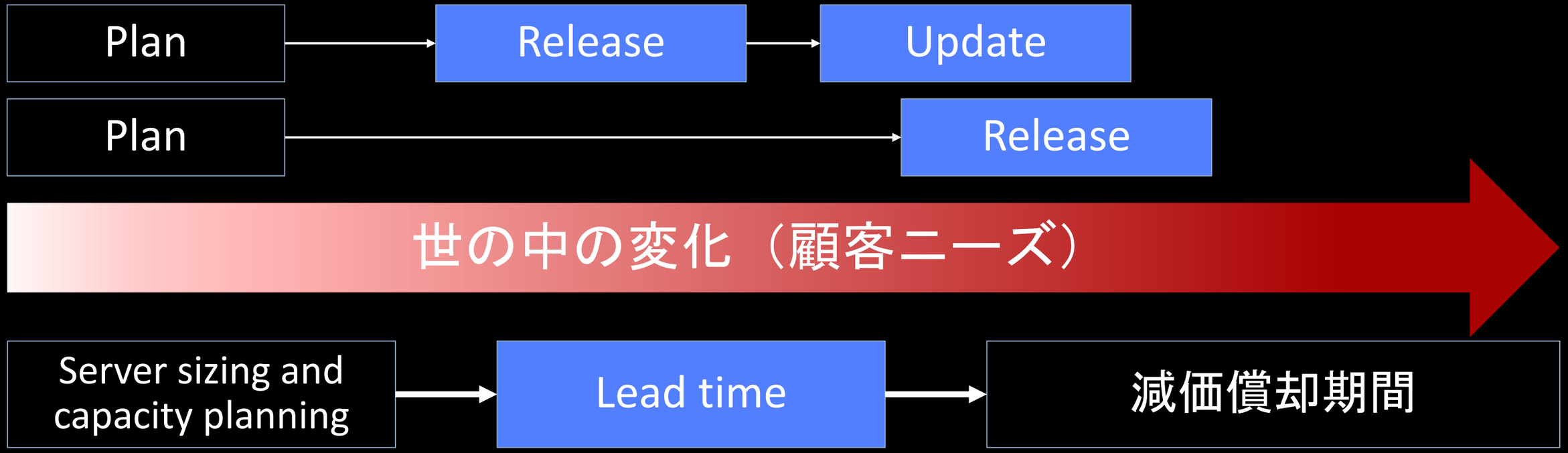
予測できないサイジングは必要ない

必要なタイミングで必要なリソースをスケールさせる



クラウドネイティブ on AWS がビジネスを加速化させる

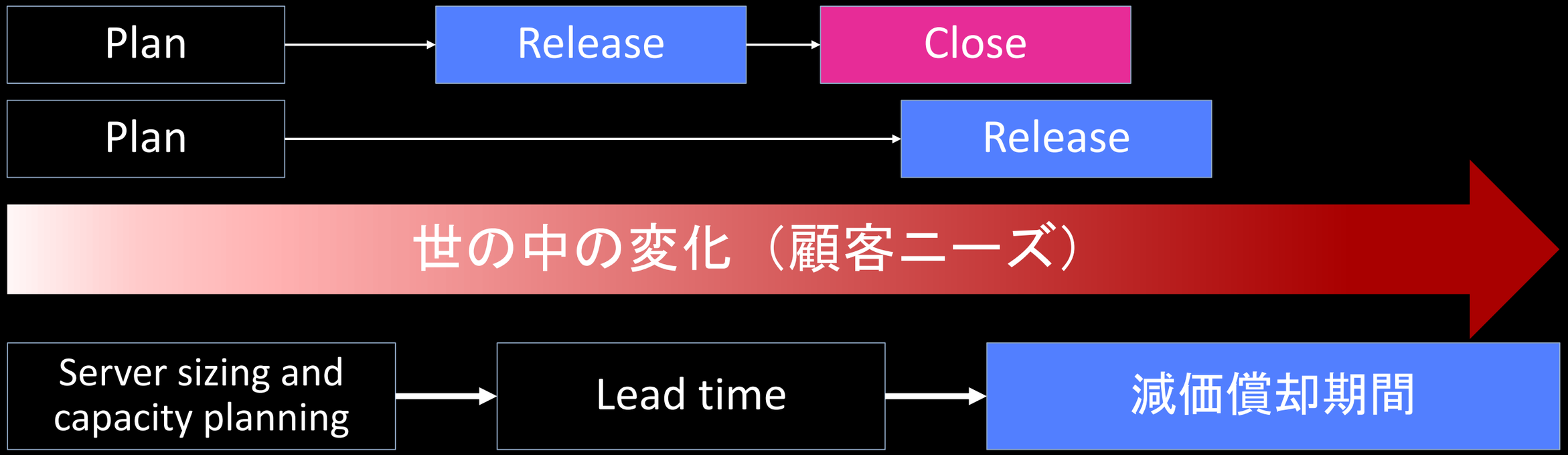
必要な時に必要なリソースを利用できる
ビジネスサイクルに合わせたリリースが可能



クラウドネイティブ on AWS がビジネスを加速化させる

ビジネスサイクルに合わせたクローズも重要

顧客ニーズに合わせてリリースするサイクルを作り出す



まとめ

クラウドネイティブなシステム開発を行う上で、ビジネス側とシステム開発との間にギャップがないことが重要になる。

開発効率の向上はシステム面の変化だけでは実現できない。

ギャップを埋めることでビジネスは加速する。

Thank you!

Toshiaki IWAMOTO
Gurunavi,Inc. / Infrastructure Services Section